

Delegatable and Updatable Anonymous Credentials

Jan Bobolz

October 6, 2026

A dissertation submitted to the
Department of Computer Science,
Paderborn University,
for the degree of
Doktor der Naturwissenschaften
(doctor rerum naturalium)

Contents

Abstract	iv
Acknowledgements	vi
1 Introduction	1
1.1 Contribution	4
2 Basics and notation	7
2.1 Basic mathematical notation	7
2.2 Algorithms and pseudocode notation	7
2.2.1 Basic ppt algorithms	7
2.2.2 Error handling	9
2.2.3 Oracles	9
2.2.4 Interactive algorithms	11
2.2.5 Asynchronous execution of protocols	12
2.3 Setting generator and security definitions	13
2.4 Assumptions	17
2.5 Digital signatures	18
2.6 Hash functions	19
2.7 Commitment schemes	20
2.8 Encryption schemes	24
2.9 The random oracle model	26
2.10 The generic group model	29
3 Zero-knowledge proof systems	40
3.1 Sigma protocols	43
3.1.1 Syntax of Sigma protocols	43
3.1.2 Special honest-verifier zero-knowledge	44
3.1.3 (Strong) special soundness and rewinding	45
3.2 Schnorr proof ecosystem	47
3.2.1 The basic generalized Schnorr protocol	48
3.2.2 “Send-first” protocols	50
3.2.3 Strongly-sound generalized Schnorr proofs with quasi-unique re- sponses	53
3.2.4 Standard tricks of the Schnorr proof ecosystem	55

3.2.5	AND and OR composition of Sigma protocols	57
3.3	Definitions	60
3.3.1	Syntax of proof systems	61
3.3.2	Relay attacks	62
3.3.3	Overview of desirable security properties	63
3.3.4	Concurrently adaptively simulation-extractable zero-knowledge proofs of knowledge	65
3.4	From Sigma protocols to secure proofs	69
3.5	The issues with rewinding	75
3.6	Overview — bringing it all together	79
4	Traditional anonymous credentials	81
4.1	Pseudonyms	84
4.2	Formal definition	88
4.2.1	Syntax and usage	88
4.2.2	Anonymity	90
4.2.3	Unforgeability and impersonation	95
4.3	Standard template	98
4.4	Non-transferability	100
4.4.1	The degenerate standard template	101
4.4.2	Modeling non-transferability	102
4.4.3	Non-transferability via pseudonym inspection	103
4.4.4	Non-transferability via artificial objective measure	103
4.4.5	Non-transferability via outside-of-system meaningful user keys . . .	104
4.4.6	All-or-nothing non-transferability	105
4.4.7	Non-transferability via unclonable hardware	105
4.4.8	Conclusion	106
4.5	History of anonymous credentials	108
5	Updatable credentials	110
5.1	Related work	115
5.2	Definition	115
5.2.1	Syntax	116
5.2.2	Anonymity	117
5.2.3	Explainability	120
5.3	Construction	126
5.3.1	Signing a committed value	127
5.3.2	The standard template for UACS	130
5.3.3	Anonymity of standard template UACS	134
5.3.4	Explainability of standard template UACS	137
5.3.5	Instantiating the standard template	144
5.4	Traditional credentials from UACS	149

6	Delegatable credentials	162
6.1	Related work	168
6.2	Definition	171
6.2.1	Syntax	171
6.2.2	Anonymity	174
6.2.3	Explainability	177
6.3	Dynamically malleable signatures	181
6.3.1	Syntax	182
6.3.2	Unforgeability	184
6.3.3	Anonymity	186
6.3.4	Construction	187
6.4	Construction	205
6.4.1	The standard template with delegation	205
6.4.2	Anonymity of standard template DACS	209
6.4.3	Explainability of standard template DACS	213
6.4.4	Instantiating the standard template	225
7	Conclusion	231
	References	237

Abstract

English

Anonymous credential systems enable authentication without identification. In such systems, issuers issue credentials to users. These credentials contain attributes encoding information about the user or credential. Users can present these credentials to verifiers, for example to establish that the user has good reputation or to prove possession of certain access rights. At the core of these systems is privacy: Both receiving and presenting credentials exposes the minimum information needed to issuers and verifiers. For example, this means that users can establish that they have the **access-to-building-X** attribute without revealing other attributes such as their name.

In this thesis, we design extensions to anonymous credentials in the form of additional privacy-preserving operations: Credentials can not only be issued and presented, they can also be updated and delegated.

We define *updatable* anonymous credential systems, which additionally allow issuers to update a user's attributes without learning unnecessary information about the old or new attributes. Updatable anonymous credentials provide a new abstracted view on anonymous credentials and enable privacy-preserving systems such as point collection systems. We show how to construct updatable anonymous credentials generically, based on standard building blocks: Signature schemes with efficient protocols and zero-knowledge proof systems.

We also define (attribute-based) *delegatable* anonymous credential systems, where users can additionally delegate a weaker version of their credentials to other users. We show how to construct delegatable anonymous credentials generically, based on zero-knowledge proof systems and a new building block we call *dynamically malleable signatures*. These special signatures feature a novel notion of controlled malleability, which represents the core insight for deriving weaker versions of credentials suitable for delegation.

Finally, we examine the fundamentals of anonymous credentials and our extensions: We revisit the relevant zero-knowledge proof systems, providing definitions, formalizations of folklore techniques, and a coherent framework for designing the zero-knowledge proof systems needed for anonymous credential systems. Additionally, we discuss what we call "traditional anonymous credentials", which follow Chaum's identity-centric vision. We describe a modernized definitions of such systems with the goal of understanding their properties and establishing a baseline functionality that our new updatable/delegatable anonymous credential definitions must be able to implement.

Deutsch

Anonyme Zugangsberechtigungssysteme ermöglichen Authentifikation ohne Identifikation. In solchen Systemen stellen Organisationen sogenannte *Credentials* an Nutzer aus. Ein Credential enthält Attribute, die Daten über den Nutzer oder über das Credential darstellen. Nutzer können diese Credentials gegenüber einem Nachweisprüfer präsentieren, zum Beispiel um eine positive Reputation zu etablieren oder um Zugriffsrechte nachzuweisen. Im Zentrum dieser Systeme steht der Schutz der Privatsphäre: Beim Empfangen und Präsentieren eines Credentials wird nur ein Minimum an Information an die ausstellende Organisation bzw. den Nachweisprüfer übertragen. Zum Beispiel bedeutet das, dass Nutzer ihr **Zugang-zu-Gebäude-X** Attribut nachweisen können ohne dabei andere Attribute, wie ihren Namen, preiszugeben.

In dieser Arbeit erweitern wir anonyme Zugangsberechtigungssysteme um zusätzliche privatheitswahrende Operationen: Neben dem Ausstellen und Präsentieren von Credentials erlauben wir auch das Aktualisieren und Delegieren.

Wir definieren *aktualisierbare* anonyme Zugangsberechtigungssysteme, mit denen eine Organisation die Attribute eines Nutzers aktualisieren kann, ohne dabei unnötige Informationen über die alten oder neuen Attribute zu erfahren. Aktualisierbare anonyme Zugangsberechtigungssysteme bieten eine neue abstrahierte Sicht auf anonyme Zugangsberechtigungssysteme und ermöglichen privatheitswahrende Systeme wie Punktsammelsysteme. Wir zeigen, wie aktualisierbare anonyme Zugangsberechtigungssysteme generisch aus bekannten Schemata wie Signaturen mit effizienten Protokollen und Zero-Knowledge Beweisen konstruiert werden können.

Wir definieren auch (attributbasierte) *delegierbare* anonyme Zugangsberechtigungssysteme, welche Nutzern erlauben, eine schwächere Version ihrer Credentials an andere Nutzer zu delegieren. Wir zeigen, wie delegierbare anonyme Zugangsberechtigungssysteme generisch aus Zero-Knowledge Beweisen und einem neuen Schema, welches wir *dynamisch formbare Signaturen* nennen, konstruiert werden können. Diese speziellen Signaturen stellen eine neue Form von kontrollierter Formbarkeit zur Schau, welche die Kerneinsicht darstellt, mit der schwächere Versionen eines Credentials zum Delegieren abgeleitet werden können.

Schließlich untersuchen wir die Grundlagen für anonyme Zugangsberechtigungssysteme und unsere Erweiterungen: Wir werfen einen zweiten Blick auf die relevanten Zero-Knowledge Beweissysteme, für die wir Definitionen, eine Formalisierung von oft als "Folklore" bezeichneten Techniken und ein kohärentes Konzept für die Konstruktion der Beweissysteme, die wir für anonyme Berechtigungssysteme benötigen, bereitstellen. Zusätzlich diskutieren wir Systeme, die wir "traditionelle anonyme Berechtigungssysteme" nennen und welche der identitätsorientierten Vision von Chaum folgen. Wir beschreiben eine modernisierte Definition solcher Systeme mit dem Ziel, ihre Eigenschaften zu verstehen und eine Grundfunktionalität zu etablieren, die unsere neuen aktualisierbaren und delegierbaren anonymen Zugangsberechtigungssysteme implementieren können müssen.

Acknowledgements

I want to thank Johannes Blömer, Tibor Jager, and Andy Rupp for reviewing this thesis, as well as my PhD committee consisting of Johannes Blömer, Tibor Jager, Juraj Somorovsky, Christian Scheideler, and Yasemin Acar.

I am immensely grateful to my advisor Johannes Blömer for getting me into cryptography, for all the feedback and advice over the years, and for being willing to experiment with so many teaching, research, and organization ideas. I very much appreciate the academic freedom I was given, as well as the opportunity to learn, travel, and hire student assistants.

Our Codes & Cryptography group at Paderborn University was a place where ideas could be discussed freely, teaching was taken seriously and got the attention it deserves, and research was treated as a genuine intellectual pursuit rather than the mere production of papers. In particular, I want to thank Johannes for creating this environment, Claudia for having our backs administratively and for maintaining social cohesion, and, of course, Fabian, Gennadij, Henrik, Jakob, Kathrin, Laurens, Nils, Peter, and Sascha for many coffee and lunch meetings, seminars, random whiteboard and non-whiteboard discussions, and generally for making work fun. For similar reasons, I am thankful to Tibor and his group (particularly Denis, Peter C., and Saqib), Bahar, my side gig coworkers at Studienfonds OWL, and to many more people.

Special thanks to all my coauthors, other collaborators, and students working with me. There are legitimately too many to enumerate, but I appreciate every one of you. I do want to particularly mention Fabian Eidens for great collaborations and conversations all around. Thanks to everyone in an admin/tech/scientific support role for their help, advice, and for keeping the lights on (figuratively and literally).

I want to thank my (non-work) friends, especially Rebekka for countless discussions about academia and life; and Markus, Nelli, Robin, and Tessa for housing, food, and good times.

Finally, I owe gratitude to my family, particularly my parents Annegret and Thomas for . . . ultimately everything, but especially for getting into computers early, letting me write naive BASIC programs at an early age, and for enabling my higher education. Danke!

Chapter 1

Introduction

We have to imagine a world in which
any data you create is under your
control.

Sir Tim Berners-Lee

A warning come true. In the year 1984, around the time when personal computing emerged, David Chaum warns about a possible future [Cha84; Cha85]. In this future, users routinely provide their name, address, payment account number, and other uniquely identifying information to organizations they interact with. Organizations identify the user for every transaction and keep records of their conduct. These records are routinely exchanged between organizations (for example to check credit rating, or history of relationships with landlords or employers). Chaum warns about the “dossier society”, where large amounts of automatically captured data across multiple organizations contribute to the creation of detailed user dossiers. He notes that even if such dossiers are *not* created in practice, the existence of linkable data across many organizations is still a threat: If policies change in the future, dossiers can be created easily using existing data.

Today, the scenario Chaum warned about has largely materialized. We routinely identify ourselves to organizations, providing full names and contact information, for example when buying a train or plane ticket, checking into a hotel, or paying phone bills. There is little incentive for organizations not to collect such data: It helps combat illegitimate accounts, it enables holding users liable in case of misbehavior, and it makes account management and recovery easy. Data collection may even be mandated by law in certain places (such as hotels).

Organizations also routinely share data. For example, credit checks rely on organizations sharing experiences about some specific user’s payment behavior. Landlords rely on references to establish a future tenant’s reputation.

Chaum’s solution. To combat the dossier society, Chaum proposes a paradigm shift around the interaction between users and organizations [Cha84; Cha85]: Instead of identifying themselves to organizations, users would establish *pseudonyms* with organizations.

A pseudonym hides the user’s real identity but serves as an identifier of a specific user to the organization. Different organizations would know the same user under different pseudonyms. As a consequence, two organizations cannot easily exchange data pertaining to a specific user (users’ pseudonyms for different organizations cannot be *linked*).

Recognizing the legitimate interest of organizations to share data about users, Chaum proposes that data sharing involve users: Organizations issue *credentials* to their users, encoding shareable information about the user (e.g., the user’s credit rating with this organization). The user can present a credential to another organization (under another pseudonym), effectively sharing the data without establishing a link between the different pseudonyms. This preserves the user’s privacy while still enabling organizations to carry out crucial checks about the user and enacting a form of access control.

Such systems that allow sharing authenticated information between organizations are nowadays known as *anonymous credential systems*. They are the focus of this thesis.

Anonymous credential systems today. Over time, the field of cryptography developed tools that enable construction of efficient and provably secure anonymous credential systems. Today, one style of construction has largely prevailed [CL02], which we call the “standard template” for anonymous credential systems. It involves commitments as pseudonyms, digital signatures to authenticate credential content (“attributes”), and zero-knowledge proofs to present a credential. We discuss details in Section 4.3 and alternative construction styles in Section 4.5.

While the tools are mature and well-understood, and there are implementations of anonymous credential systems (e.g., [BBBBD+18; CH02; AJ13; CDLP22; PZ23]), anonymous credential systems have seen little use in practice so far. Ultimately, users identifying themselves to organizations is seen as socially acceptable and perfectly normal (perhaps even seen as inevitable). Hence there is little incentive for organizations to forgo real user identities and deploy an anonymous credential system. Additionally, we are faced with a chicken-and-egg problem: Users do not hold credentials because no organization accepts them. Organizations do not accept credentials because users do not yet hold any.

Recently, the situation changed due to the European Digital Identity (EUDI) Regulation [Law24], which requires EU member states to establish a digital identity infrastructure not unlike anonymous credentials. While early versions of this infrastructure are unlikely to use the cryptographic tools mentioned above [BBCHL+24], the EUDI regulation does bring anonymous credential ideas into practice. Additionally, anonymous credentials are a candidate solution for online age verification, which has been legislatively pushed recently in many countries, too.

This thesis does not advocate for or against any such practical endeavors. Instead, we aim to provide a deep understanding of anonymous credential systems (mainly as a mathematical object) and explore advanced concepts not currently planned for practical implementation.

Broadening our view of anonymous credentials. In Chaum’s view, the main privacy threat is data collected by different organizations coming together into a joint user dossier. This view informs the idea of organization-specific pseudonyms, which ensure that each organization knows a user under a different name, making it difficult to compile data about the same user from different organizations. In this setting, anonymous credentials enable the flow of user information between organizations.

Nowadays, several organizations have grown into powerful platforms that users spend a significant amount of time on. In the natural course of using the platforms, users reveal significant amounts of personal data, especially about their preferences. Even without linking the data they collect with data from other organizations, these organizations can create detailed dossiers about their users. Indeed, the creation of such dossiers contributes to personalized recommendations and targeted advertisement, which is often the central business model of these platforms. Still, the question arises whether we can, in principle, design platforms with strong privacy guarantees.

Clearly, if user interactions are not decentralized across many small organizations but rather concentrated around few large organizations, the original pseudonym ideas only offer weak protection against dossier creation. Instead, we strive for the following view of anonymous credentials: Whenever possible, instead of organizations storing information about a user (or pseudonym) in a database, organizations should *issue a credential containing this information* to the user. For example, instead of storing “The user with pseudonym N has bought a bus ticket for zones A and B” in the transport organization’s database, and checking the user’s account when boarding the bus, the organization would instead issue a credential to the user, which the user presents on the bus. In the first case detailed movement data can be collected and associated with the pseudonym N . In the second case, credential presentation can be made independent of pseudonyms, providing fully anonymous bus usage, with data not attributable to any specific pseudonym.

In this updated view, the user’s data with an organization is stored by the user (in authenticated credential form), and the user reveals the appropriate data in future interactions with the same organization. This transforms anonymous credentials from a tool transferring user data between different pseudonyms at different organizations to a tool that provides unlinkability between different interactions even within the same organization (or between organizations). Note that implementation of this new view does not necessitate significant changes to the construction of anonymous credentials — existing techniques already provide what we need. Instead, this is a conceptual change.

Updatable anonymous credentials. In the “users store their own database entries” view of anonymous credentials, writing a new database entry corresponds to issuing a credential. Reading from a database entry corresponds to the user presenting a credential. An important missing operation is *updating* an existing database entry, which corresponds to issuing a credential containing modified data relative to some prior credential held by the user. Crucially, we want to implement such a credential update without revealing unnecessary information.

For example, suppose the user participates in a loyalty system, earning rewards for

repeated use of the bus system. Using our new view of anonymous credentials, such a system can be implemented by having the user store a credential with their current ride count n . When entering a bus, we expect the bus organization to update n to $n + 1$. Naively, we may have the user present their credential with content n and then have the organization issue a new credential with content $n + 1$. However, this reveals the user's current n to the organization, which is unnecessary, potentially even uniquely identifying information. The only information we want the organization to learn is that they are executing a $+1$ update. The organization should not learn n , not learn $n + 1$, and not learn the user's identity or pseudonym.

In this thesis, we introduce *updatable anonymous credential systems*, which bring this strong notion of privacy-preserving updates to anonymous credentials. Such systems enable advanced use cases of privacy-preserving and unlinkable interaction of a user with an organization.

Delegatable credentials. In the settings described above, credentials are exclusively created by organizations. In contrast, *delegatable* credentials allow users holding a credential to delegate (a weaker version of) their credential to another user. Similarly to how updatable anonymous credentials allow privacy-preserving updates to the data held by users, delegatable credentials allow users to create new data records (for other users) in some controlled fashion.

As a simple example, we can imagine a user holding a credential that entitles him to access all buildings on a university campus. Using his credential, that user can delegate a credential to another user, which grants them temporary access to some subset of buildings. Using the delegated credential, the second user can gain anonymous access to the specified buildings, but not to other buildings and not after a specified expiration date.

In this thesis, we introduce *delegatable updatable anonymous credential systems*, which support delegation capabilities that can be suitably restricted by credential issuers and delegators. Similar to updatable anonymous credentials, this enables new advanced applications for anonymous credentials.

1.1 Contribution

In this thesis, we discuss (plain) anonymous credentials, updatable anonymous credentials, and delegatable credentials. We discuss these types of schemes through the lens of formal definitions, provable security, and practical concerns. Our goal is to properly understand plain anonymous credentials and their guarantees, to formally model and construct updatable anonymous credentials, and to formally model and construct delegatable anonymous credentials.

Zero-knowledge proof systems. Most anonymous credential constructions rely on zero-knowledge proof systems, which serve a crucial function both in the issuance phase

(e.g., ensuring users behave correctly) and when presenting credentials (e.g., hiding information about the user’s attributes from the verifier). While the types of proof systems needed (Schnorr-style proofs) are relatively simple in principle (compared to, say, modern circuit-based proof systems), the details turn out to be quite intricate and nontrivial. These details are considered “folklore”. In the anonymous credential literature, they are generally omitted. As a consequence, newcomers and other communities generally do not have access to critical nuance around proof systems for anonymous credentials, and they may not realize how to securely instantiate proof systems and how to implement certain credential presentation policies in zero-knowledge. To combat these issues, Chapter 3 provides an overview of relevant techniques for Schnorr-style proofs in a formal framework. We show how to construct the proof systems necessary for anonymous credentials, including how to compose different statements into one Sigma protocol. Finally, we discuss important context, such as issues surrounding rewinding techniques.

Chapter 3 organizes and formalizes existing concepts (“folklore”) into one coherent formal framework. Apart from the formalization and some small insights into existing constructions, the chapter does not claim substantial conceptual novelty. It should be considered a *systemization of knowledge* effort.

Traditional anonymous credentials. The anonymous credential literature is large and there is no unified definition of *anonymous credentials*. This is largely because most publications deal with a specific set of features (e.g., pseudonyms, revocation, . . .), and they customize their definitions to fit this feature set. Still, there is a core anonymous credential system implicit in the early literature that we consider “traditional” anonymous credentials, following early influential works [CL01; CL02; CL04]. We synthesize these ideas into a modern formal definition in Chapter 4. This definition is used in two ways: First, we discuss aspects of traditional anonymous credentials in depth, to develop a good understanding what these systems focus on and what properties they provide or do not provide. Second, the definition provides a baseline of functionality that must be realizable with our advanced constructions in Chapters 5 and 6.

Again, apart from the modernized formalization, Chapter 4 does not claim substantial conceptual novelty. Rather, it provides discussion of existing concepts.

Updatable anonymous credentials. As explained above, updatable anonymous credentials (UACS) both introduce a new notable feature with blind updates, as well as represent a definitional paradigm shift for anonymous credentials.

Updatable anonymous credentials were first formalized and constructed in [BBDE19]. We revisit this formalization in Chapter 5 and improve upon it. The concept, formalization, and construction of updatable anonymous credentials are some of the main contributions of this thesis. We prove that our formalization can be used to instantiate traditional anonymous credentials.

Delegatable anonymous credentials. As explained above, delegatable anonymous credentials (DACS) introduce the ability for users to delegate weaker versions of their

credentials to other users.

While other kinds of delegatable credentials existed before (e.g., [CL06; BCKLS09; CL19; CDD17]), *attribute-based* delegatable credentials with strong privacy and notions of restricted attribute issuance were first formalized and constructed in [BB18]. We revisit this formalization in Chapter 6, improve aspects of it, and adapt it to our Chapter 5 definitional framework. This means that we our delegatable credentials profit from the Chapter 5 definitional advancements, such as generalized blind issuance, and that we combine updatable and delegatable credentials: Users can use their own credentials to update another user’s credential. We additionally improve upon the original construction [BB18], allowing for expression of more powerful delegation and presentation policies. The formalization and construction of attribute-based delegatable anonymous credentials (and its main building block, dynamically malleable signatures) are some of the main contributions of this thesis.

The structure of this thesis. We explain our notation and revisit the basic concepts used in this thesis in Chapter 2. In Chapter 3, we revisit definitions and constructions of zero-knowledge proof systems, laying the foundation for the proof systems used for our constructions later. In Chapter 4, we revisit traditional anonymous credentials. We introduce updatable anonymous credentials in Chapter 5, and delegatable anonymous credentials in Chapter 6. Finally, Chapter 7 concludes the thesis with some reflection and discussion.

Chapter 2

Basics and notation

You can't build a great building on a weak foundation.

Gordon B. Hinckley

In this chapter, we explain notation and basic concepts used throughout the thesis. Readers familiar with standard cryptography concepts may skip this chapter for their first read and return to it if questions arise.

2.1 Basic mathematical notation

You can find an overview of our notation regarding sets and probabilities in Table 2.1. Our probability spaces will be finite or countably infinite. Random variables can take arbitrary objects (such as group elements) as values. They behave as one would expect, generalizing real-valued random variables.

2.2 Algorithms and pseudocode notation

In this thesis, we can think of algorithms as probabilistic interactive Turing machines with oracles, i.e., Turing machines with additional tapes and instructions for sending and receiving messages to another Turing machine and for calling and receiving a result from an oracle. In the following, we explain our notation regarding such algorithms.

2.2.1 Basic ppt algorithms

We say that an algorithm $\mathcal{A}$ is “probabilistic polynomial time” (ppt) if it runs in polynomial time (i.e., there is a polynomial $poly$ such that on any input x , $\mathcal{A}$ makes at most $poly(|x|)$ many computation steps) and it is probabilistic (i.e., it has access to an additional tape that contains a uniformly random bit string $r \leftarrow \{0, 1\}^{poly(|x|)}$).

By default, we consider algorithms to be stateless, i.e., whenever we write $y \leftarrow \mathcal{A}(x)$, a “fresh instance” of $\mathcal{A}$ is used for the computation. When we say that an algorithm is “stateful”, we allow $\mathcal{A}$ to keep state $state_{\mathcal{A}}$ between invocations. So for a stateful

Notation	Meaning
Sets	
$\mathbb{N}$	Set of natural numbers $\mathbb{N} = \{1, 2, \dots\}$
$\mathbb{N}_0$	Set of natural numbers $\mathbb{N}_0 = \{0, 1, 2, \dots\}$ including 0
$\mathbb{Z}$	Set of integers $\mathbb{Z} = \mathbb{N}_0 \cup \{-n \mid n \in \mathbb{N}\}$
$\mathbb{R}$	Set of real numbers
$\mathbb{Z}_p$	$\mathbb{Z}_p = \mathbb{Z}/p\mathbb{Z} = \{0, \dots, p-1\}$ with addition/multiplication mod p
$\mathbb{Z}_p^*$	$\mathbb{Z}_p^* = \{x \in \mathbb{Z}_p \mid \exists y : xy = 1\}$ (for prime p , we get $\mathbb{Z}_p^* = \mathbb{Z}_p \setminus \{0\}$)
$[n]$	$[n] = [1, n] = \{1, \dots, n\} \subset \mathbb{Z}$
$\text{im}(f)$	Range of (partial) function $f : \mathcal{X} \rightarrow \mathcal{Y}$, i.e., $\text{im}(f) = \{y \in \mathcal{Y} \mid \exists x \in \mathcal{X} : f(x) = y\}$
$\text{preim}(f)$	Domain of (partial) function $f : \mathcal{X} \rightarrow \mathcal{Y}$, i.e., $\text{preim}(f) = \{x \in \mathcal{X} \mid \exists y \in \mathcal{Y} : f(x) = y\}$ (for total functions f , $\text{preim}(f) = \mathcal{X}$)
$\text{span}_R(S)$	Span of a set S w.r.t. ring R , i.e., $\text{span}_R(S) = \{\sum_{i=1}^q a_i \cdot s_i \mid q \in \mathbb{N} \wedge a_1, \dots, a_q \in R \wedge s_1, \dots, s_q \in S\}$
S^n	Set of tuples of length $n \in \mathbb{N}_0$ with elements from set S . $S^n = \{(s_1, \dots, s_n) \mid \forall i \in [n] : s_i \in S\}$
$1^n, \star^n$	The symbol 1 (or $\star$) repeated $n \in \mathbb{N}_0$ times. Note: $1^n \in \{1\}^n$.
Randomness	
$Y \leftarrow S$	Define Y as a random variable whose value is chosen uniformly at random from finite set S
$Y \leftarrow \mathcal{A}(x)$	Define Y as a random variable whose value is (potentially randomly) chosen by algorithm $\mathcal{A}$ on input x
$\Pr[\phi(Y)]$	The probability for the random variable Y to take on a value satisfying predicate ϕ
$\Pr[\phi(Y) \mid Y \leftarrow S]$	Same as $\Pr[\phi(Y)]$, but explicitly defining Y as a random variable picked uniformly from finite set S
$\Pr[\phi(Y) \mid \psi(Y)]$	The probability of $\phi(Y)$ conditioned on $\psi(Y)$. $\Pr[\phi(Y) \mid \psi(Y)] = \Pr[\phi(Y) \wedge \psi(Y)] / \Pr[\psi(Y)]$.
$[Y]$	The support $[Y] = \{x \mid \Pr[Y = x] > 0\}$ of random variable Y
Miscellaneous	
$\log(t)$	The base 2 logarithm of t , i.e., $2^{\log(t)} = t$ for all $t \in \mathbb{R}, t > 0$
$ x $	The length of the bit string $x \in \{0, 1\}^*$ or the number of elements in the set x
$\deg(f)$	Denotes the total degree of a (multivariate) polynomial $f \in \mathbb{F}[X_1, \dots, X_n]$. For example, $\deg(X_1^5 X_2^3 + X_3^2) = 8$. Fulfills $\deg(f \cdot f') = \deg(f) + \deg(f')$ if $\mathbb{F}$ is a field.

Table 2.1: Mathematical notation

algorithm $\mathcal{A}$, the expression “ $y_1 \leftarrow \mathcal{A}(x_1); y_2 \leftarrow \mathcal{A}(x_2)$ ” means “ $(y_1, \text{state}_{\mathcal{A}}) \leftarrow \mathcal{A}(x_1); y_2 \leftarrow \mathcal{A}(x_2, \text{state}_{\mathcal{A}})$ ”, i.e., we omit the state $\text{state}_{\mathcal{A}}$ in our notation. The length of the state $\text{state}_{\mathcal{A}}$ does not count towards the input size $|x|$ when bounding an algorithm’s runtime (to avoid an algorithm being able to, say, double the size of its state with every invocation).

We list basic pseudocode notation in Table 2.2.

2.2.2 Error handling

Pseudocode needs to be able to handle errors and unexpected output formats. We generally allow algorithms we design to output the error symbol (e.g., to indicate that something went wrong) or to explicitly raise an error that stops execution (as explained in Table 2.2). When it comes to algorithms not explicitly designed in this thesis, such as the adversary, these are not allowed to raise error events to stop the execution.

Other than that, the input and output format of the algorithms we design in this thesis follow consistent typing. We do not explicitly list types in pseudocode, but all types are well-defined and clear from the context. However, we need to deal with the adversary outputting values that do not conform to the types we expect. For this, we use a parser that is guaranteed to output the expected output format. For example, when we write $(Y_1, Y_2) \leftarrow \mathcal{A}(x)$ and the context makes clear that $Y_1, Y_2 \in \mathbb{N}$ are supposed to be natural numbers, then the output of $\mathcal{A}$ is suitably parsed into a pair of natural numbers. If $\mathcal{A}$ outputs something that does not resemble a pair of natural numbers (e.g., it just outputs a single bit 0), then the parser simply sets Y_1, Y_2 to some arbitrary default numbers, e.g., $Y_1 = Y_2 = 1$. With this convention, we do not need to handle invalid adversarial output: Any output will be interpreted to be (at least syntactically) valid. In particular, if we do not expect the adversary to output an error symbol $\perp$, then we do not need to handle the adversary outputting one: Its output will be forcibly parsed into a non-error symbol.

2.2.3 Oracles

Algorithms $\mathcal{A}$ may get oracle access to some functionality, which we model as *oracle* calls. This can be used to model certain actions that $\mathcal{A}$ cannot do itself, but, for example, can be executed by other actors in a security model.

- For a (stateful or stateless) randomized algorithm $\mathcal{O}$ (the “oracle”), writing $\mathcal{A}^{\mathcal{O}}$ means that $\mathcal{A}$ can, at any time during its execution, *call* the oracle: $\mathcal{A}$ specifies any x , then $y \leftarrow \mathcal{O}(x)$ is executed, then $\mathcal{A}$ gets y as a result. The time to execute $\mathcal{O}(x)$ does not count towards $\mathcal{A}$ ’s runtime.
- $\mathcal{A}$ may get access to multiple oracles $\mathcal{O}_1, \dots, \mathcal{O}_n$, in which case $\mathcal{A}^{\mathcal{O}_1, \dots, \mathcal{O}_n}$ gets to specify the index $i \in [1, n]$ for the oracle it wants to call.
- We often write expressions like $\mathcal{A}^{\mathcal{O}.f_1, \mathcal{O}.f_2}$, which means that $\mathcal{A}$ is executed having access to two oracles. Querying either oracle will trigger the same (stateful or stateless) machine $\mathcal{O}$, just calling different methods (f_1 or f_2) on it (as described in Table 2.2). In particular, these different methods share state.

Notation	Meaning
Basics	
$Y \leftarrow S$	Denotes picking the value of variable Y uniformly at random from the finite set S (this is a polynomial time operation)
$Y \leftarrow \mathcal{A}(x)$	Denotes execution of (randomized) algorithm $\mathcal{A}$ on input x and storing the result as variable Y . If $\mathcal{A}$ is stateful, the current state $state_{\mathcal{A}}$ is implicitly passed and updated. We also write “Hand x to $\mathcal{A}$, it returns Y ”.
$(Y_1, Y_2) \leftarrow \mathcal{A}(x)$	Denotes that the result of $\mathcal{A}(x)$ is parsed into a tuple (Y_1, Y_2)
$\mathcal{A}(x_1, x_2)$	Denotes execution of $\mathcal{A}$ with input (x_1, x_2)
$\mathcal{A}(x_1; x_2)$	Same as $\mathcal{A}(x_1, x_2)$. Semicolons in parameter lists do not have special meaning, they may be used to group inputs for readability.
$\mathcal{A}.f(x)$	Denotes execution of method f of algorithm $\mathcal{A}$. Can be seen as a shorthand for $\mathcal{A}(\text{"f"}, x)$. Also used to address function f of scheme Π (denoted $\Pi.f$).
$\mathcal{A}(x \mid r)$	Denotes execution of randomized algorithm $\mathcal{A}$ with random tape set to r
Special keywords	
return x	Outputs x to the caller and skips execution of the rest of the method
assert ϕ	Equivalent to “if not ϕ , then return $\perp$.” (we regard $\perp$ as an error symbol)
raise fail_x	Raises the event/exception fail_x . Stops all currently running machines immediately (except for algorithms that define error handling like “if fail_x happens, then ...”).
note fail_x	Notes the event/exception fail_x . Does not have any consequences regarding program flow, but is used to denote that an event occurs, which we may reference in other algorithms or in formal proofs.
Special variables	
$state_{\mathcal{A}}$	Denotes the current <i>state</i> of algorithm $\mathcal{A}$ (see beginning of Section 2.2.1)
$view_{\mathcal{A}}$	Denotes the (current) <i>view</i> of algorithm $\mathcal{A}$: Input, randomness, and (interactive/oracle) messages received (Definition 2.1)

Table 2.2: Basic pseudocode notation

Note that in any case, $\mathcal{A}$ does not learn what exact algorithm (or method) is executed, just its output.

2.2.4 Interactive algorithms

Two algorithms $\mathcal{A}, \mathcal{B}$ can interact, i.e., exchange messages. This allows us to model interactive protocols.

- The expression $x \leftarrow \mathcal{A}(a) \leftrightarrow \mathcal{B}(b) \rightarrow y$ means that $\mathcal{A}$ on input a and $\mathcal{B}$ on input b interact, at the end of which $\mathcal{A}$ outputs x and $\mathcal{B}$ outputs y .
 - $\mathcal{A}$ is started with input a and $\mathcal{B}$ is given input b but is not started yet.
 - At some point, $\mathcal{A}$ outputs its first message to $\mathcal{B}$. Then execution of $\mathcal{A}$ is suspended, its message is given to $\mathcal{B}$, and execution of $\mathcal{B}$ is resumed (or started).
 - Analogously, when $\mathcal{B}$ outputs a message, it is given to $\mathcal{A}$ and then $\mathcal{A}$ is resumed, while $\mathcal{B}$ is suspended.
 - $\mathcal{A}$ or $\mathcal{B}$ may also choose to halt instead of sending a message. In that case, the other machine, if not halted yet, is resumed (with a special “HALTED” message) and run to completion, ignoring any messages it may still want to send.
- Similar to above, *stateful* algorithms keep state between multiple runs of a protocol.

In order to analyze and to interact with interactive algorithms, we need the definition of the *view* of a machine. Intuitively, the view consists of everything one would need to retrace all the computation steps an algorithm has done.

Definition 2.1. Let $\mathcal{A}$ be a potentially interactive algorithm. The *view* of $\mathcal{A}$, denoted $view_{\mathcal{A}}$, consists of

- The input that $\mathcal{A}$ has been (initially) started with.
- The random string R that $\mathcal{A}$ uses as randomness (cf. Section 2.2.1).
- The list of all messages that $\mathcal{A}$ may have received running protocols with other algorithms.
- The list of all replies from oracle queries that $\mathcal{A}$ has made.

If $view_{\mathcal{A}}$ is used inside of pseudocode where $\mathcal{A}$ is run as a subroutine, it denotes its *current* values of the data listed above (e.g., all messages received *so far*). Otherwise, it denotes the values for after $\mathcal{A}$ halts.

Finally, we define a notion modeling that an algorithm behaves exactly like another, but may exhibit *additional* behavior that does not impact the original expectations.

Definition 2.2 (Perfect emulation). Let $\mathcal{A}_{\text{orig}}, \mathcal{A}_{\text{emul}}$ be interactive algorithms and let $\mathcal{O}_1, \dots, \mathcal{O}_k, \dots, \mathcal{O}_n$ be (stateful) algorithms. We say that $\mathcal{A}_{\text{emul}}^{\mathcal{O}_1, \dots, \mathcal{O}_n}$ *perfectly emulates*

$\mathcal{A}_{\text{orig}}^{\mathcal{O}_1, \dots, \mathcal{O}_k}$ if for all a, x, Z and all interactive algorithms $\mathcal{B}$,

$$\begin{aligned} & \Pr[(x, \text{view}_{\mathcal{B}}, (\text{view}_{\mathcal{O}_i})_{i=1}^k) = Z \mid x \leftarrow \mathcal{A}_{\text{orig}}^{\mathcal{O}_1, \dots, \mathcal{O}_k}(a) \leftrightarrow \mathcal{B}] \\ &= \Pr[(x, \text{view}_{\mathcal{B}}, (\text{view}_{\mathcal{O}_i})_{i=1}^k) = Z \mid x \leftarrow \mathcal{A}_{\text{emul}}^{\mathcal{O}_1, \dots, \mathcal{O}_n}(a) \leftrightarrow \mathcal{B}] \end{aligned}$$

i.e., $\mathcal{A}_{\text{orig}}$ and $\mathcal{A}_{\text{emul}}$ on the same input produce the same output (distribution), the same (distribution of) messages, and the same (distribution of) queries to shared oracles.

$\mathcal{A}_{\text{emul}}$ has access to additional oracles $\mathcal{O}_{k+1}, \dots, \mathcal{O}_n$, for which there is no constraint regarding how $\mathcal{A}_{\text{emul}}$ queries them. This notion will be used later (Definition 5.3) in order to define a drop-in replacement $\mathcal{A}_{\text{emul}}$ for $\mathcal{A}_{\text{orig}}$ that behaves exactly like $\mathcal{A}_{\text{orig}}$ outwardly, but may exhibit additional functionality through oracle queries. Concretely, we will replace some “original” protocol algorithm with one that essentially runs the same protocol (and hence can be easily run in the original protocol’s place), but additionally outputs (through oracle queries) metadata explaining what is happening during the protocol.

2.2.5 Asynchronous execution of protocols

When we model security requirements of our constructions in this thesis, we need to be able to describe realistic (attack) scenarios. One aspect of realistic attacks is that of concurrency: In reality, cryptographic protocols are not usually executed one after the other, but they happen in parallel. Their messages can be arbitrarily interleaved. This will play a central role for zero-knowledge proof systems later. We introduce useful syntax to treat interleaving protocol executions within the usual oracle call notation.

Parallel execution. Let $\mathcal{A}^{\mathcal{O}}$ be an algorithm. Let $\mathcal{P}$ be an interactive algorithm. Consider the oracle $\mathcal{O}$ defined as follows.

```

 $\mathcal{O}(x)$ 
 $y \leftarrow \mathcal{P}(x) \leftrightarrow \mathcal{A}$ 
return  $y$ 

```

Intuitively, this pseudocode notation means that when $\mathcal{A}$ calls $\mathcal{O}(x)$, then the oracle starts running (a fresh copy of) the interactive algorithm $\mathcal{P}$, exchanging messages with $\mathcal{A}$ until the protocol concludes. Whenever $\mathcal{P}$ sends a message to $\mathcal{A}$, execution of $\mathcal{O}$ is suspended and $\mathcal{A}$ ’s code is executed. To model concurrency, $\mathcal{A}$ can then decide to either start a new instance of $\mathcal{O}$ or resume one that already exists. As a result, multiple of such oracle executions can be running at the same time.

Similarly, consider the following oracle.

```

 $\mathcal{O}(x)$ 
 $y \leftarrow \mathcal{A}(x)$  //We also write “Hand  $x$  to  $\mathcal{A}$ , receive  $y$ ”.
return  $y$ 

```

We understand a *call* to $\mathcal{A}$ similarly to a protocol execution. This means that the execution of $\mathcal{O}$ is paused in the first line of code until $\mathcal{A}$ provides some value y . In the meantime, $\mathcal{A}$ is able to start new executions of this oracle (or others) before resuming this particular execution.

More formally, if an oracle $\mathcal{O}$ contains an instruction to run a protocol $\mathcal{P}$ interacting with $\mathcal{A}$ or to call $\mathcal{A}$, we understand $\mathcal{O}$ as being implicitly replaced with the following oracle $\mathcal{O}^{\text{actual}}$.

$\mathcal{O}^{\text{actual}}(h, x)$

If $\mathcal{O}^{\text{actual}}(h, \cdot)$ has not been called before, set $state_h$ to the initial program state for $\mathcal{O}$ with input x .

Otherwise, retrieve the stored program state $state_h$. Interpret x as the next message (or return value) of $\mathcal{A}$, and add it to $state_h$.

Continue executing $\mathcal{O}$ with $state_h$ until:

If $\mathcal{O}$ encounters a return y statement, then return (return, y) .

If $\mathcal{O}$ encounters instruction to send message m to $\mathcal{A}$ or to call $\mathcal{A}(m)$, then save current state of $\mathcal{O}$ in $state_h$ and return (msg, m)

The $\mathcal{O}^{\text{actual}}$ oracle takes as input a handle $h \in \{0, 1\}^*$, which $\mathcal{A}$ can use to specify *which execution* of the oracle $\mathcal{O}$ it wants to start or resume. The value h itself has no meaning other than as an identifier for an execution. It can be chosen by $\mathcal{A}$ arbitrarily. The input x is either interpreted as the input to $\mathcal{O}$ (for a new execution) or as a message or output by $\mathcal{A}$ (when resuming an execution that was suspended due to waiting for a message/output from $\mathcal{A}$). The oracle $\mathcal{O}^{\text{actual}}$ remembers the current state of $\mathcal{O}$ for each handle h so that its execution can be resumed.

With this mechanism, we model concurrency. Messages of interactive protocols can be arbitrarily interleaved by $\mathcal{A}$. This should not be mistaken for a fully concurrent model. At any point in time, only one machine is active (in the process of executing code) and every line of code is executed sequentially (execution can only be interrupted by returning a value or by sending a message to another machine or by calling an oracle). However, our model of concurrency is sufficient to model security for real-world situations.

2.3 Setting generator and security definitions

In this thesis, we are concerned with *provable security*, i.e., we are going to design formal security models and then prove that our constructions are secure (under the assumption that certain computations are hard to do). For formal security definitions, we follow the standard asymptotic framework, in which we say that for any ppt adversary, the chances of breaking security diminishes rapidly as we increase the *security parameter* λ of the system. In the following, we will detail the basic mechanisms that deal with this.

Setting generator For our constructions in this thesis, we are going to compose multiple smaller building blocks together. These building blocks operate over bilinear groups.

In order to express that those building blocks work over *the same* bilinear group, we define the notion of a *setting generator*. All constructions and building blocks in this thesis will use the same setting generator, which generates the parameters of a bilinear group. Because all building blocks share the same setting generator, they can share bilinear group parameters, which gives us some sense of compatibility.

Definition 2.3 (Setting generator). A setting generator is a ppt algorithm **GenSetting** that takes 1^λ (the string consisting of $\lambda \in \mathbb{N}$ many ones) as input.

For this thesis, the setting generator outputs a bilinear group $\mathcal{G} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, e, p)$ with the following properties:

- $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t$ are (multiplicative) groups of prime order $p > 2^\lambda$.
- $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_t$ is an efficiently computable non-degenerate bilinear map (often called “pairing”), i.e.,
 - Bilinearity: $e(g^a, h^b) = e(g, h)^{ab}$ for all $g \in \mathbb{G}_1, h \in \mathbb{G}_2, a, b \in \mathbb{Z}$.
 - Non-degeneracy: If $g \neq 1$ and $h \neq 1$, we have that $e(g, h) \neq 1$.
 - Efficiency: e can be computed in polynomial time.

As is customary in cryptography, we pass 1^λ (the string containing λ many 1s) as input instead of the more natural λ . This is to ensure that **GenSetting** gets runtime polynomial in the security parameter λ . If the input were simply λ , the length of the input is only $\log(\lambda)$ because numbers are encoded base 2. Note that $\mathcal{G} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, e, p)$ is encoded with at least λ bits, so any algorithm taking $\mathcal{G}$ as input is afforded polynomial runtime in λ .

In theory, we imagine the setting generator **GenSetting** is executed by a trusted party and the setting/parameters $\mathcal{G}$ are published for everyone to know and use. Every invocation of **GenSetting** would likely yield a new, random bilinear group whose size (and hence assumed security level) scales with the security parameter λ . In practice, instead of generating a random new bilinear group, one would generally use one of the standard parameter sets, such as BLS12-381, which have been crafted to be efficient and have the concrete security properties we require against realistic attackers. In a sense, these standard parameters have been chosen by a trusted party (though not necessarily purely at random). If one were not to trust those standard parameters, one can generate a random bilinear group even in practice (at the cost of interoperability and, likely, performance).

Note that all the groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t$ are of prime order p , which makes them very convenient to work with. For $i \in \{1, 2, t\}$, we get:

- Every element $g \in \mathbb{G}_i \setminus \{1\}$ is a generator of $\mathbb{G}_i$.
- We can meaningfully compute mod p in the exponent, i.e., $\varphi_i : \mathbb{Z}_p \rightarrow \mathbb{G}_i$, $\varphi(x) = g^x$ is a group homomorphism (an isomorphism if $g \neq 1$).
- For any element $g \in \mathbb{G}_i$ and $h \neq 1$ (even if they are chosen randomly according to any distribution), and for $r \leftarrow \mathbb{Z}_p$, we get that $g \cdot h^r$ is distributed uniformly at random,

statistically independent of g and h . We will use this to hide information about group elements (or their discrete logarithms).

For our purposes, the pairing is mostly used to efficiently decide whether or not two non-neutral elements $g^x \in \mathbb{G}_1, h^y \in \mathbb{G}_2$ have the same discrete logarithm. This is the case if and only if $e(g^x, h) = e(g, h^y)$. Because by bilinearity, we get $e(g^x, h) = e(g, h)^x$ and $e(g, h^y) = e(g, h)^y$, which are equal if and only if $x = y \pmod p$.

We focus on type 3 pairings because those are the most efficient. In addition to this, type 3 pairings have additional security properties. For a type 3 curve, we assume that there is no efficiently computable isomorphism between $\mathbb{G}_1$ and $\mathbb{G}_2$ (or vice versa), which differentiates it from type 1 and type 2 pairings.

Negligible functions. Above, we have used a “security parameter” $\lambda \in \mathbb{N}$. Intuitively, the larger λ is chosen, the harder it should get for an adversary to break security properties. Generally, we will say a system is secure if any ppt adversary has *negligible* probability of breaking it. Negligible functions are those that tend towards zero faster than the inverse of any positive polynomial.

Definition 2.4 (Negligible function). A function $negl : \mathbb{N} \rightarrow \mathbb{R}$ is *negligible* if $\forall c \in \mathbb{N} \exists n_0 \in \mathbb{N} \forall \lambda > n_0 : negl(\lambda) < 1/\lambda^c$.

While there are certainly other forms of negligible functions, exponentially decreasing functions (such as $\lambda \mapsto 2^{-\lambda}$) are a useful and typical example for negligible functions.

For an expression $expr(\lambda)$ directly or indirectly involving the security parameter λ , we will simply say “ $expr$ is negligible”, meaning that $\lambda \mapsto expr(\lambda)$ is a negligible function. For example, we can say “ $\Pr[x = 0 \mid x \leftarrow \{0, \dots, 2^\lambda\}]$ is negligible”.

Negligible functions have useful composition properties. For a polynomial p and negligible functions $negl, negl'$, we have that $negl + negl'$ is negligible and $p \cdot negl$ is negligible. Furthermore, if $negl(\lambda) \leq |f(\lambda) - negl'(\lambda)|$, then also f is negligible. We will use these properties throughout the thesis without further comment.

Notions of “large”. Not only will we have to say “something is small”, but also “something is large”.

Definition 2.5 (Notions of “large”). A function $f : \mathbb{N} \rightarrow \mathbb{R}$ is

- *overwhelming* if $1 - f$ is negligible, i.e., $\forall c \in \mathbb{N} \exists n_0 \in \mathbb{N} \forall \lambda > n_0 : 1 - 1/\lambda^c < f(\lambda)$.
 An overwhelming function tends towards 1 faster than the inverse of any polynomial.
- *noticeable* if $\exists c \in \mathbb{N} \exists n_0 \in \mathbb{N} \forall \lambda > n_0 : f(\lambda) \geq 1/\lambda^c$.
 A noticeable function can be lower-bounded by the inverse of some polynomial (for sufficiently large λ).
- *super-poly* if $1/f$ is negligible, i.e., if $\forall c \in \mathbb{N} \exists n_0 \in \mathbb{N} \forall \lambda > n_0 : \lambda^c < f(\lambda)$.
 A super-poly function grows faster than any polynomial.

We will sometimes say that an event E_λ happens with *overwhelming* probability, meaning that $\neg E_\lambda$ happens with negligible probability.

There are functions that are neither negligible nor noticeable (e.g., $f(\lambda) = \text{negl}(\lambda) + 2^{(\lambda \bmod 2)}$), but of all the nonnegligible functions, noticeable functions are the easiest to work with (e.g., for counterexamples).

The groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, \mathbb{Z}_p$ (as in Definition 2.3) are of super-poly size. This means that the probability $\Pr[r = g \mid r \leftarrow \mathbb{G}_1] = 1/p$ of choosing any fixed element $g \in \mathbb{G}_1$ uniformly at random is negligible.

On security definitions. Assume we can prove that for all ppt adversaries $\mathcal{A}$, the probability for $\mathcal{A}$ to break the security of a cryptosystem is negligible (this will be our goal). In that setting, whenever a new real-world attack $\mathcal{A}$ comes along (with a new strategy or with more resources to carry out more complex computations), then all we need to do is increase the security parameter until the success probability of that new attack $\mathcal{A}$ becomes small enough to comfortably ignore. For example, attacks that succeed with probability 2^{-128} are, realistically, never going to succeed in practice.

Increasing the security parameter comes with a cost: The computations of honest parties become more expensive. All algorithms of the cryptosystem can be carried out in polynomial time. Increasing the security parameter will polynomially increase the required resources to execute it honestly (concretely, when increasing the size of a bilinear group, all group operations become more expensive to compute). If we prove that $\mathcal{A}$'s success probability is *negligible*, then we know that this trade-off generally favors the honest parties: The time to execute the cryptosystem increases polynomially with λ , but the winning probability of the adversary $\mathcal{A}$ decreases faster than the inverse of any polynomial.

While this models security quite nicely in theory, this notion is not necessarily as appealing in practice. The abstract guarantee that increasing the security parameter will *eventually* decrease adversaries' winning probabilities is not enough in practice, where we need to *concretely* set parameters. This poses additional challenges, which are treated with notions like *concrete security* and *tightness*. For this thesis, we keep to the asymptotic security framework explained above, and we leave setting concrete security parameters to practitioners. We note that oftentimes, setting these parameters simply boils down to choosing one of the standard elliptic curves (such as BLS12-381), as discussed around Definition 2.3.

We formally model security with *games*: We design an experiment ("game") where an adversary interacts with some system that tries to approximate reality. For example, the adversary is presented with certain keys and is allowed to interact with oracles representing the behavior of honest parties. The goal is to design these experiments such that any real-world adversarial behavior (and honest user behavior) can be mapped to actions in the experiment.

We then define winning conditions, which govern when we consider an adversary wins the experiment (breaks security). We typically demand that the probability for these conditions to occur be negligible. For example, an adversary wins the unforgeability game of a signature scheme if it can output a valid signature on a message that was not signed by an honest user (see Definition 2.9).

Alternatively, the definition may take on the form of an *indistinguishability* definition, where we demand that an adversary cannot distinguish whether it interacts with one of two experiments. For example, we model indistinguishability of encryptions by demanding that an adversary cannot distinguish an experiment where an honest user encrypts message m_0 from an experiment where he encrypts message m_1 (see Definition 2.22).

2.4 Assumptions

In order to conduct (interesting) cryptography, we need to make assumptions about limits of what efficient (adversarial) algorithms are capable of achieving. Whether these assumptions hold is unknown. Indeed, an important unsolved question in complexity theory is whether $\mathbf{P} = \mathbf{NP}$. If this were the case, then there is essentially no hope for any cryptography with security against arbitrary polynomial-time adversaries.

However, $\mathbf{P}$ vs $\mathbf{NP}$ is still unsolved. For practical purposes, certain hardness assumptions (such as the ones below) are perfectly reasonable: Breaking these assumptions would constitute a significant and surprising theoretical breakthrough. At the very least, these assumptions give us a way to establish practical security against adversaries that cannot produce theoretical math breakthroughs. We can gain some confidence in assumptions by proving they hold for restricted adversary classes, see Section 2.10 for example.

The most fundamental assumption we need for this thesis is the assumption that discrete logarithms are difficult to compute. This is fundamental because if this assumption is broken, then most security guarantees of our constructions will be broken, too.

Definition 2.6 (Discrete Logarithm assumption). The *discrete logarithm assumption* states that for all ppt $\mathcal{A}$, there is a negligible function negl such that for all λ ,

$$\begin{aligned} \Pr[\mathcal{A}(\mathcal{G}, g_1, g_1^x) = x \mid g_1 \leftarrow \mathbb{G}_1 \setminus \{1\}, x \leftarrow \mathbb{Z}_p] &\leq \text{negl}(\lambda) \\ \Pr[\mathcal{A}(\mathcal{G}, g_2, g_2^x) = x \mid g_2 \leftarrow \mathbb{G}_2 \setminus \{1\}, x \leftarrow \mathbb{Z}_p] &\leq \text{negl}(\lambda) \\ \Pr[\mathcal{A}(\mathcal{G}, g_1, g_2, e(g_1, g_2)^x) = x \mid g_1 \leftarrow \mathbb{G}_1 \setminus \{1\}, g_2 \leftarrow \mathbb{G}_2 \setminus \{1\}, x \leftarrow \mathbb{Z}_p] &\leq \text{negl}(\lambda) \end{aligned}$$

where $\mathcal{G} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, e, p) \leftarrow \text{GenSetting}(1^\lambda)$.

Note that hardness of discrete logarithms for $\mathbb{G}_t$ is implied by hardness in $\mathbb{G}_1$ (and by hardness in $\mathbb{G}_2$) because one can always compute the discrete logarithm of $e(g_1^x, g_2) = e(g_1, g_2)^x$ to retrieve the discrete logarithm of g_1^x .

Additionally, we use the decisional Diffie-Hellman assumption, which enables encryption in groups.

Definition 2.7 (Decisional Diffie-Hellman assumption). The *decisional Diffie-Hellman assumption* states that for all ppt $\mathcal{A}$, there is a negligible function negl such that for all λ ,

$$\left| \begin{array}{l} \Pr[\mathcal{A}(\mathcal{G}, g_1, g_1^x, g_1^y, g_1^{x \cdot y}) = 1] \\ - \Pr[\mathcal{A}(\mathcal{G}, g_1, g_1^x, g_1^y, \underline{g_1^z}) = 1] \end{array} \right| \leq \text{negl}(\lambda)$$

$$\left| \begin{array}{l} \Pr[\mathcal{A}(\mathcal{G}, g_2, g_2^x, g_2^y, g_2^{x \cdot y}) = 1] \\ - \Pr[\mathcal{A}(\mathcal{G}, g_2, g_2^x, g_2^y, g_2^z) = 1] \end{array} \right| \leq \text{negl}(\lambda)$$

where $\mathcal{G} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, e, p) \leftarrow \text{GenSetting}(1^\lambda)$, $g_1 \leftarrow \mathbb{G}_1 \setminus \{1\}$, $g_2 \leftarrow \mathbb{G}_2 \setminus \{1\}$, $x, y, z \leftarrow \mathbb{Z}_p$.

Because we require that Decisional Diffie-Hellman is hard in both $\mathbb{G}_1$ and $\mathbb{G}_2$, this assumption is often called the “symmetric external Diffie-Hellman assumption” (SXDH). Hardness of Decisional Diffie-Hellman for $\mathbb{G}_t$ is not explicitly required. It is implied by hardness in $\mathbb{G}_1$ (and by hardness in $\mathbb{G}_2$).

2.5 Digital signatures

Perhaps the most important building block for anonymous credentials are digital signatures. Looking ahead, a credential is a signature on the user’s identity and attributes.

A digital signature is meant to *authenticate* data. For this, we have a *signer* who publishes a public key pk and holds on to a secret key sk . With the secret key, the signer can create signatures σ on arbitrary messages m . Using pk , anyone can check whether a given signature σ is valid for message m , which effectively guarantees that the signer agreed to sign m . For consistency with our *dynamically malleable signatures* later, we define a signature scheme to explicitly authenticate a *vector* of messages.

Definition 2.8 (Digital signature scheme). A digital signature scheme Π_{sig} with message space $\mathbb{M}_{\mathcal{G}}$ consists of the following ppt algorithms.

$\text{Setup}(\mathcal{G}) \rightarrow pp$

Generates public parameters pp .

$\text{Keygen}(pp, 1^n) \rightarrow (pk, sk)$

Generates a public key pk and a secret key sk . We assume n can be efficiently derived from pk . $\heartsuit$ The key is good for signing messages $m \in \mathbb{M}^n$

$\text{Sign}(pp, sk, \vec{m}) \rightarrow \sigma$

Creates a signature σ on message $\vec{m} \in \mathbb{M}^n$.

$\text{Verify}(pp, pk, \vec{m}, \sigma) = b$

A deterministic algorithm that outputs a bit b . $\heartsuit$ $b = 1$ indicates that σ is a valid signature on message $\vec{m}$.

A digital signature scheme is *correct*, i.e., for all $\lambda, n \in \mathbb{N}$, $\mathcal{G} \in [\text{GenSetting}(1^\lambda)]$, $pp \in [\text{Setup}(\mathcal{G})]$, all keys $(pk, sk) \in [\text{Keygen}(pp, 1^n)]$, all messages $\vec{m} \in \mathbb{M}^n$, and all signatures $\sigma \in [\text{Sign}(pp, sk, \vec{m})]$, it holds that

$$\text{Verify}(pp, pk, \vec{m}, \sigma) = 1$$

Furthermore, we require that there is a deterministic polynomial-time algorithm that, on input $\mathcal{G} \in [\text{GenSetting}]$, outputs some message $m \in \mathbb{M}_{\mathcal{G}}$. We simply call this fixed message 0. $\P$ This is a very mild requirement (essentially just requiring it is easy to find *any* message in the message space), but having some fixed message is a useful property we will use later.

Secure signature schemes shall be unforgeable, i.e., an adversary cannot create a valid signature on some message not signed by the honest signer.

Definition 2.9 (Unforgeability). A digital signature scheme Π_{sig} is *existentially unforgeable under chosen-message attacks* (EUF-CMA) if for all $n \in \mathbb{N}$ and all ppt adversaries $\mathcal{A}$,

$$\Pr \left[\begin{array}{l} \text{Verify}(pp, pk, \vec{m}^*, \sigma^*) = 1 \\ \wedge \quad \vec{m}^* \notin \mathcal{T}_{\text{signed}} \end{array} \middle| \begin{array}{l} \mathcal{G} \leftarrow \text{GenSetting}(1^\lambda) \\ pp \leftarrow \text{Setup}(\mathcal{G}) \\ (pk, sk) \leftarrow \text{Keygen}(pp, 1^n) \\ (\vec{m}^*, \sigma^*) \leftarrow \mathcal{A}^{\mathcal{O}.\text{Sign}}(\mathcal{G}, pp, pk) \end{array} \right]$$

is negligible. Here, $\mathcal{O}.\text{Sign}(\vec{m})$ is an oracle that adds $\vec{m}$ to the initially empty set $\mathcal{T}_{\text{signed}}$ and then returns $\sigma \leftarrow \text{Sign}(pp, sk, \vec{m})$.

For technical reasons, we are interested in signature schemes with large message spaces.

Definition 2.10 (Super-poly message space). A digital signature scheme Π_{sig} has *super-poly message space* if there is a super-poly function f such that $|\mathbb{M}_{\mathcal{G}}| \geq f(\lambda)$ for all $\lambda \in \mathbb{N}$ and $\mathcal{G} \in [\text{GenSetting}(1^\lambda)]$.

Looking ahead, we are not going to use arbitrary signature schemes, but rather require that the signature scheme come with efficient protocols, see Definition 5.4.

2.6 Hash functions

An important object in cryptography are hash functions, which compress values from a large domain into a short digest (“hash value”).

Definition 2.11 (Hash function). A hash function Π_{hash} with domain $\mathcal{X}_{\mathcal{G}}$ and codomain $\mathcal{Y}_{\mathcal{G}}$ consists of the following ppt algorithms:

$\text{Setup}(\mathcal{G}) \rightarrow pp$
Generates public parameters pp .

$\mathcal{H}(pp, x) = y$
A deterministic function that, given $x \in \mathcal{X}_{\mathcal{G}}$, outputs a hash value $y \in \mathcal{Y}_{\mathcal{G}}$.

In this thesis, we are mostly interested in random oracle hash functions (Section 2.9). However, basic hash functions with the collision resistance property are instructive for several discussions later, hence we list the property here: A hash function is collision-resistant if no ppt adversary can find two preimages with the same hash value.

Definition 2.12 (Collision resistance). A hash function Π_{hash} is *collision-resistant* if for all ppt $\mathcal{A}$,

$$\Pr \left[\begin{array}{l} x, x' \in \mathcal{X}_{\mathcal{G}} \\ \wedge x \neq x' \wedge \mathcal{H}(pp, x) = \mathcal{H}(pp, x') \end{array} \middle| \begin{array}{l} \mathcal{G} \leftarrow \text{GenSetting}(1^\lambda), \\ pp \leftarrow \text{Setup}(\mathcal{G}), (x, x') \leftarrow \mathcal{A}(\mathcal{G}, pp) \end{array} \right]$$

is negligible.

Examples for a collision-resistant hash function are the SHA-2, SHA-3 functions (they have even stronger properties, as discussed in Section 2.9 and Lemma 2.25).

Another important hash function is the Pedersen hash function. It has a useful algebraic structure, which we will see used implicitly in many places. It is provably secure under the discrete logarithm assumption.

Construction 2.13 (Pedersen hash). Let $n \in \mathbb{N}$. The Pedersen hash function has domain $\mathcal{X}_{(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, e, p)} = \mathbb{Z}_p^n$ and codomain $\mathcal{Y}_{(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, e, p)} = \mathbb{G}_1$. It works as follows:

$\text{Setup}(\mathcal{G}) \rightarrow pp$

Generates $pp = (h_i)_{i=1}^n \leftarrow \mathbb{G}_1^n$.

$\mathcal{H}(pp, x) = y$

For $x = (x_1, \dots, x_n) \in \mathbb{Z}_p^n$, outputs $y = \prod_{i=1}^n h_i^{x_i}$.

Theorem 2.14. *If the discrete logarithm assumption holds (Definition 2.6), then the Pedersen hash function is collision-resistant.*

Proof. Let $\mathcal{A}$ be a ppt against collision-resistance. We construct $\mathcal{B}$ against the discrete logarithm assumption (for $\mathbb{G}_1$). $\mathcal{B}$ gets input $\mathcal{G} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, e, p)$ and g, h with $h = g^z$ and $g \in \mathbb{G}_1 \setminus \{1\}$. It chooses $\alpha_i, \beta_i \leftarrow \mathbb{Z}_p$ and sets $pp = (h_i)_{i=1}^n := (h^{\alpha_i} g^{\beta_i})_{i=1}^n = (g^{z\alpha_i + \beta_i})_{i=1}^n$. It then runs $\mathcal{A}$ on input $\mathcal{G}, pp$. $\mathcal{A}$ returns x, x' . If $\mathcal{A}$ wins the collision-resistance game, then $x \neq x'$ and $\mathcal{H}(pp, x) = \mathcal{H}(pp, x')$. Let $\Delta_i = x_i - x'_i$. This means that

$$\prod_{i=1}^n h_i^{x_i} = \prod_{i=1}^n h_i^{x'_i} \Rightarrow \prod_{i=1}^n h_i^{\Delta_i} = 1 \Rightarrow \sum_{i=1}^n \Delta_i (z\alpha_i + \beta_i) = 0 \Rightarrow z \cdot \sum_{i=1}^n \alpha_i \Delta_i + \sum_{i=1}^n \beta_i \Delta_i = 0.$$

If $\sum_{i=1}^n \alpha_i \Delta_i \neq 0$, $\mathcal{B}$ outputs $z = \frac{-\sum_{i=1}^n \beta_i \Delta_i}{\sum_{i=1}^n \alpha_i \Delta_i}$. Because $\mathcal{A}$'s view is independent of the α_i and we have $\Delta_j \neq 0$ for at least one $j \in [n]$, Schwartz-Zippel (Lemma 2.27) implies $\Pr[\sum_{i=1}^n \alpha_i \Delta_i = 0] \leq 1/p$. So $\Pr[\mathcal{B}(\mathcal{G}, g, g^z) = z] \geq \Pr[\mathcal{A} \text{ outputs collision}] \cdot (1 - 1/p)$. By assumption, $\Pr[\mathcal{B}(\mathcal{G}, g, g^z) = z]$ is negligible, hence $\Pr[\mathcal{A} \text{ outputs collision}]$ must be negligible, too. $\square$

2.7 Commitment schemes

Commitment schemes play a crucial role in this thesis. With a commitment scheme, a party can compute a *commitment* C to some message m using randomness o . The

commitment C hides m and can be safely revealed to other parties. However, C is binding, i.e., every commitment C hides a unique message m . The randomness o serves as proof that the value hidden in C is indeed m (and not some other message). One can imagine that the committer first reveals C to other parties, which does not reveal m to them, but guarantees that the committer cannot change their mind about m anymore. Later, she reveals m, o to show that m is indeed the unique message she committed to earlier. Oftentimes, instead of revealing m, o in the clear, the committer may keep them hidden and instead only prove some statement about them in zero-knowledge.

Commitment schemes are a crucial component of traditional anonymous credential systems: A pseudonym is generally a commitment to the user's secret key.

Beyond that, we will frequently use commitments as glue between multiple protocols. For example, for our updatable anonymous credentials construction (Construction 5.7), we have one component (a zero-knowledge proof system) that proves that the user's requested attributes are valid, and another component that blindly creates a signature on those attributes. To ensure that both components talk about the same attributes (while keeping them hidden), we commit to the attributes and have both components work on the same commitment, which, effectively guarantees they work on the same attributes.

Definition 2.15 (Commitment scheme). A commitment scheme Π_{commit} with message space $\mathbb{M}_{\mathcal{G}}$ and randomness space $\mathbb{O}_{\mathcal{G}}$ consists of the following ppt algorithms.

$\text{Setup}(\mathcal{G}) \rightarrow pp$

Generates public parameters pp .

$\text{Commit}(pp, m, o) = C$

A deterministic algorithm that takes as input a message $m \in \mathbb{M}_{\mathcal{G}}$ and commitment randomness $o \in \mathbb{O}$, and outputs a commitment C .

The idea is that to commit to a value $m \in \mathbb{M}_{\mathcal{G}}$, the committer first picks a uniformly random $o \leftarrow \mathbb{O}$ and computes the commitment as $C = \text{Commit}(pp, m, o)$. The committer keeps m, o secret (she may reveal them later or use them in other advanced protocols).

The following definition formalizes the property that every commitment C encodes a unique message m .

Definition 2.16 (Computational binding). A commitment scheme Π_{commit} is *computationally binding* if for all ppt $\mathcal{A}$,

$$\Pr \left[\begin{array}{l} (m, o), (m', o') \in \mathbb{M}_{\mathcal{G}} \times \mathbb{O}_{\mathcal{G}} \wedge m \neq m' \\ \wedge \text{Commit}(pp, m, o) = \text{Commit}(pp, m', o') \end{array} \middle| \begin{array}{l} \mathcal{G} \leftarrow \text{GenSetting}(1^\lambda), \\ pp \leftarrow \text{Setup}(\mathcal{G}), \\ (m, o, m', o') \leftarrow \mathcal{A}(\mathcal{G}, pp) \end{array} \right]$$

is negligible.

We say that Π_{commit} is *strongly* computationally binding if for all ppt $\mathcal{A}$,

$$\Pr \left[\frac{(m, o), (m', o') \in \mathbb{M}_{\mathcal{G}} \times \mathbb{O}_{\mathcal{G}} \wedge (m, o) \neq (m', o')}{\wedge \text{Commit}(pp, m, o) = \text{Commit}(pp, m', o')} \parallel \begin{array}{l} \mathcal{G} \leftarrow \text{GenSetting}(1^\lambda), \\ pp \leftarrow \text{Setup}(\mathcal{G}), \\ (m, o, m', o') \leftarrow \mathcal{A}(\mathcal{G}, pp) \end{array} \right]$$

is negligible.

Phrased differently, let $\Pi_{\text{commit}} = (\text{Setup}, \text{Commit})$ be a commitment scheme. If there is some value $0 \in \bigcap_{\mathcal{G} \in [\text{GenSetting}]} \mathbb{O}_{\mathcal{G}}$, then Π_{commit} is computationally binding iff $(\text{Setup}, \mathcal{H})$ with $\mathcal{H}(pp, m) = \text{Commit}(pp, m, 0)$ is a collision-resistant hash function (Definition 2.12) with message space $\mathcal{X}_{\mathcal{G}} = \mathbb{M}_{\mathcal{G}}$. It is *strongly* computationally binding iff $\mathcal{H}(pp, m, o) = \text{Commit}(pp, m, o)$ is a collision-resistant hash function with message space $\mathcal{X}_{\mathcal{G}} = \mathbb{M}_{\mathcal{G}} \times \mathbb{O}_{\mathcal{G}}$. In particular, a commitment scheme with fixed randomness is a collision-resistant hash function.

The following definition formalizes that C does not reveal anything about the committed message m .

Definition 2.17 (Hiding). A commitment scheme Π_{commit} is *computationally hiding* if for all ppt $\mathcal{A}$,

$$|\Pr[\mathcal{A}^{\mathcal{O}_0}(\mathcal{G}, pp) = 1] - \Pr[\mathcal{A}^{\mathcal{O}_1}(\mathcal{G}, pp) = 1]|$$

is negligible, where $\mathcal{G} \leftarrow \text{GenSetting}(1^\lambda)$, $pp \leftarrow \text{Setup}(\mathcal{G})$. The oracle $\mathcal{O}_b(m_0, m_1)$ checks that $m_0, m_1 \in \mathbb{M}_{\mathcal{G}}$ and if so, chooses $o \leftarrow \mathbb{O}$, and returns $\text{Commit}(pp, m_b, o)$.

We say that Π_{commit} is *perfectly* hiding if the difference above is 0 for all algorithms $\mathcal{A}$.

An extension to commitment schemes are *trapdoor* commitments, where there is a trapdoor td that essentially allows cheating binding: With the trapdoor, one can equivocate a commitment to message m to instead open to some other message m' . In any practical construction, td will be kept secret from all parties. It is an artifact usually only used in security proofs.

Definition 2.18 (Trapdoor commitments). A trapdoor commitment scheme Π_{commit} is a commitment scheme with two additional ppt algorithms:

$\text{SetupTd}(\mathcal{G}) \rightarrow (pp, td)$

Generates parameters pp and a trapdoor td .

$\text{Equivocate}(pp, td, m, o, m') \rightarrow o'$

Takes as input an opening (m, o) and another message m' and outputs an opening value $o' \in \mathbb{O}$. $\mathfrak{?}$ o' is supposed to be a valid opening for m' .

We require that

- pp in $(pp, td) \leftarrow \text{SetupTd}(\mathcal{G})$ is distributed exactly as if generated by $\text{Setup}(\mathcal{G})$.

- $\text{Commit}(pp, m', \text{Equivocate}(pp, td, m, o, m')) = \text{Commit}(pp, m, o)$

The commitment scheme in our setting (prime order groups) are Pedersen commitments. One of their main features is the algebraic structure of commitments, which makes them friendly to zero-knowledge proofs (discussed later). They also have a simple and elegant equivocation trapdoor. We will see this commitment scheme both explicitly and implicitly in several constructions later. The Pedersen commitment scheme is essentially the Pedersen hash function (Construction 2.13), with the observation that including a uniformly random message in the hash makes the hash value uniformly random (and hence hiding).

Construction 2.19 (Pedersen commitment). The Pedersen (trapdoor) commitment scheme for n values has message space $\mathbb{M}_{\mathcal{G}} = \mathbb{Z}_p^n$ and randomness space $\mathbb{O} = \mathbb{Z}_p$. It works as follows:

$\text{Setup}(\mathcal{G}) \rightarrow pp$

Generates $pp = (g, h_1, \dots, h_n) \leftarrow (\mathbb{G}_1 \setminus \{1\}) \times \mathbb{G}_1^n$

$\text{SetupTd}(\mathcal{G}) \rightarrow (pp, td)$

Generates $g \leftarrow \mathbb{G}_1 \setminus \{1\}$ and $td = (td_1, \dots, td_n) \leftarrow \mathbb{Z}_p^n$. It sets $pp = (g, (h_i)_{i=1}^n := (g^{td_i})_{i=1}^n)$. It outputs (pp, td) .

$\text{Commit}(pp, m, o) = C$

For $\vec{m} \in \mathbb{Z}_p^n$, outputs $C = g^o \cdot \prod_{i=1}^n h_i^{m_i}$.

$\text{Equivocate}(pp, td, m, o, m') \rightarrow o'$

Computes $o' = o + \sum_{i=1}^n td_i(m_i - m'_i)$.

We get that

$$\begin{aligned}
 & \text{Commit}(pp, m', \text{Equivocate}(pp, td, m, o, m')) \\
 &= g^{o + \sum_{i=1}^n td_i(m_i - m'_i)} \cdot \prod_{i=1}^n h_i^{m'_i} \\
 &= g^{o + \sum_{i=1}^n td_i(m_i - m'_i)} \cdot \prod_{i=1}^n g^{td_i m'_i} \\
 &= g^o \cdot \prod_{i=1}^n g^{td_i(m_i - m'_i)} \cdot \prod_{i=1}^n g^{td_i m'_i} \\
 &= g^o \cdot \prod_{i=1}^n g^{td_i m_i} \\
 &= \text{Commit}(pp, m, o)
 \end{aligned}$$

as required.

Theorem 2.20. *The Pedersen trapdoor commitment scheme is perfectly hiding (Defi-*

inition 2.17) and, if the discrete logarithm assumption (Definition 2.6) holds, then it is strongly computationally binding.

Proof. The commitment scheme is perfectly hiding because the distribution of a commitment $C = g^o \cdot \prod_{i=1}^n h_i^{m_i}$ is uniformly random, independent of the messages, because $o \leftarrow \mathbb{Z}_p$ is uniformly random. The strong computational binding property follows from Theorem 2.14, given that a Pedersen commitment is essentially a Pedersen hash $\mathcal{H}(pp, o, m_1, \dots, m_n)$. More concretely, let $\mathcal{A}$ be a ppt adversary against strong computational binding. We construct $\mathcal{B}(\mathcal{G}, pp = (g, h_1, \dots, h_n))$ against collision-resistance of the Pedersen hash function with $n + 1$ messages. If $g = 1$, then $\mathcal{B}$ outputs the hash collision $((1, 0, 0, \dots), (0, 0, 0, \dots))$. Otherwise, $\mathcal{B}$ invokes $(m, o, m', o') \leftarrow \mathcal{A}(\mathcal{G}, pp)$ and outputs (x, x') with $x = (o, m)$ and $x' = (o', m')$. Whenever $\mathcal{A}$ outputs two different valid openings, $\mathcal{B}$ outputs a hash collision. Because $\mathcal{B}$ is a ppt, and the Pedersen hash function is secure under the discrete logarithm assumption, the probability that $\mathcal{A}$ outputs two different valid openings must be negligible. $\square$

2.8 Encryption schemes

Perhaps the most famous type of construction in cryptography are encryption schemes. Given some public key pk , anyone can encrypt a message m , which yields a ciphertext C . Without the secret decryption key sk , the ciphertext does not reveal anything about the message m (similar to the hiding property of commitment schemes discussed above). However, given sk , one can efficiently decrypt C to retrieve the message m .

Looking ahead, encryption schemes make for good pseudonyms for traditional anonymous credentials (see Construction 4.5): We may give sk to a trusted party who can then use it to decrypt the pseudonym, revealing the user's hidden identity.

Definition 2.21 (Encryption scheme). An encryption scheme Π_{enc} with message space $\mathbb{M}_{\mathcal{G}}$ consists of the following ppt algorithms.

Setup($\mathcal{G}$) $\rightarrow$ pp

Generates public parameters pp .

Keygen(pp) $\rightarrow$ (pk, sk)

Generates a public key pk and a secret key sk .

Encrypt(pp, pk, m) $\rightarrow$ C

Takes as input a public key pk and a message $m \in \mathbb{M}$. It outputs a ciphertext C .

Decrypt(pp, sk, C) $= m$

A deterministic algorithm that on input secret key sk and a ciphertext C outputs a message m .

An encryption scheme is *correct*, i.e., for all $\lambda \in \mathbb{N}$, $\mathcal{G} \in [\text{GenSetting}(1^\lambda)]$, $pp \in$

$[\text{Setup}(\mathcal{G})]$, all keys $(pk, sk) \in [\text{Keygen}(pp)]$, all messages $m \in \mathbb{M}$, it holds that

$$\text{Decrypt}(pp, sk, \text{Encrypt}(pp, pk, m)) = m$$

Definition 2.22 (CPA security). An encryption scheme Π_{enc} is *secure against chosen-plaintext attacks* (CPA-secure) if for all ppt adversaries $\mathcal{A}$,

$$|\Pr[\mathcal{A}^{\mathcal{O}_0}(\mathcal{G}, pp, pk) = 1] - \Pr[\mathcal{A}^{\mathcal{O}_1}(\mathcal{G}, pp, pk) = 1]|$$

is negligible. Here, the randomness is over $\mathcal{G} \leftarrow \text{GenSetting}(1^\lambda)$, $pp \leftarrow \text{Setup}(\mathcal{G})$, $(pk, sk) \leftarrow \text{Keygen}(pp)$. For $b \in \{0, 1\}$, the oracle $\mathcal{O}_b(m_0, m_1)$ for $m_0, m_1 \in \mathbb{G}_1$ returns $C = \text{Encrypt}(pk, m_b)$.

Traditionally, the CPA security definition allows the adversary only a single query to the oracle. Our version is equivalent to the traditional one, which can be shown using a standard hybrid argument.

The most important encryption scheme for our purposes is the ElGamal encryption scheme. Similarly to the Pedersen commitment scheme, one big advantage of ElGamal encryption is the algebraic structure of ciphertexts, which makes it friendly for zero-knowledge proofs (discussed later).

Construction 2.23 (ElGamal encryption). The ElGamal encryption scheme for $\mathcal{G} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, e, p)$ has message space $\mathbb{G}_1$ and works as follows:

$\text{Setup}(\mathcal{G}) \rightarrow pp$

Picks random $g \leftarrow \mathbb{G}_1$ and outputs $pp = (\mathcal{G}, g)$.

$\text{Keygen}(pp) \rightarrow (pk, sk)$

Picks random $sk \leftarrow \mathbb{Z}_p$ and sets $pk = g^{sk}$. It outputs (pk, sk) .

$\text{Encrypt}(pp, pk, m) \rightarrow C$

Picks random $r \leftarrow \mathbb{Z}_p$ and outputs ciphertext $C = (C_0, C_1) = (m \cdot pk^r, g^r)$.

$\text{Decrypt}(pp, sk, (C_0, C_1))$

Outputs $m = C_0 \cdot C_1^{-sk}$.

The scheme is *correct* because $C_0 \cdot C_1^{-sk} = m \cdot pk^r \cdot (g^r)^{-sk} = m \cdot g^{sk \cdot r} \cdot g^{-sk \cdot r} = m$.

Lemma 2.24. *If the DDH assumption (Definition 2.7) holds (for $\mathbb{G}_1$) then ElGamal (Construction 2.23) is CPA-secure (Definition 2.22).*

The proof is an easy exercise.

2.9 The random oracle model

We are going to use the random oracle model (mostly to instantiate zero-knowledge proofs) in this thesis. Recent practical secure hash functions H (e.g., SHA-3) are designed such that their outputs “look random”. In particular, if you were to run standard statistical tests on, say, the sequence $H(1), \dots, H(n)$, the statistical test would attest that this sequence looks like a uniformly random sequence.

Of course, hash functions are deterministic functions, so formally, they are actually *far* from random. First formally introduced in [BR93], the *random oracle model* is an idealized model of a “random-looking hash function”.

The black box $\mathcal{RO}$. The random oracle model replaces the deterministic function $H : \{0, 1\}^* \rightarrow S$ with a black box that models a random function $\mathcal{RO}$ as in Figure 2.1.

$\mathcal{RO}(x)$

If y_x is not defined:

Choose $y \leftarrow S$, then store $y_x := y$.

Retrieve y_x from memory and return it.

Figure 2.1: The code for how the random oracle $\mathcal{RO}$ behaves.

For this, S needs to be a finite set. Typical values for S are $S = \{0, 1\}^{2^\lambda}$ ($S = \{0, 1\}^{256}$ is a typical example in practice) or $S = \mathbb{G}_1$ for a group $\mathbb{G}_1$. We can imagine the state of $\mathcal{RO}$, consisting of the y_x that have been generated, as a table $x \mapsto y_x$, which we refer to as the *random oracle table*.

Properties of $\mathcal{RO}$. By design of the black box $\mathcal{RO}$, two queries $\mathcal{RO}(x), \mathcal{RO}(x)$ for the same x will always return the same value y_x (hence $\mathcal{RO}$ properly models a *function*). If $\mathcal{RO}$ is queried on a *new* preimage x , its image $\mathcal{RO}(x) = y_x$ is chosen uniformly at random from the codomain S . Hence the only way to get any information about $\mathcal{RO}(x)$ is to ask the black box for its value. If S is large enough, then $\mathcal{RO}$ is collision-resistant, i.e., it is hard for a ppt algorithm to find $x \neq x'$ with $\mathcal{RO}(x) = \mathcal{RO}(x')$.

Lemma 2.25 (Random oracles are collision-resistant). *Let S be a finite non-empty set and $q \in \mathbb{N}$. Then*

$$\Pr[\exists i \neq j : y_i = y_j \mid y_1, \dots, y_q \leftarrow S] \leq \frac{q(q-1)}{2|S|}.$$

In particular, if a random oracle $\mathcal{RO} : \{0, 1\}^* \rightarrow S_\lambda$ is queried at most $q(\lambda)$ times, then the probability for a hash collision among these queries is smaller than $\frac{q(\lambda)^2}{2|S_\lambda|}$. If $|S_\lambda| \geq 2^\lambda$ and q is a polynomial, the collision probability is negligible.

Proof.

$$\begin{aligned} & \Pr[\exists i \neq j : y_i = y_j] \\ &= \Pr\left[\bigvee_{i < j} y_i = y_j\right] \stackrel{\text{union bound}}{\leq} \sum_{i < j} \Pr[y_i = y_j] = \sum_{i < j} \frac{1}{|S|} = \binom{q}{2} \cdot \frac{1}{|S|} = \frac{q(q-1)}{2|S|} \quad \square \end{aligned}$$

Instantiation. In practice, the hash function H would be instantiated using a state-of-the-art hash function (such as SHA-3). Especially SHA-3 has been designed to approximate the properties we expect from a random oracle. Its predecessor SHA-2 suffered from length-extension attacks, i.e., it is trivial, given $\text{SHA2}(x)$ (but not x) to compute $\text{SHA2}(x||y)$ for certain y , which is not possible in the random oracle model.

A random oracle hash function that hashes into a group (e.g., $H : \{0, 1\}^* \rightarrow \mathbb{G}_1$) is usually instantiated as follows: First, $H(x)$ computes $y' = H'(x)$ using a hash function $H' : \{0, 1\}^* \rightarrow \{0, 1\}^{\text{poly}(\lambda)}$, which we model as a random oracle. Then, y' is deterministically mapped into the group $\mathbb{G}_1$, yielding $H(x) = h(y') = h(H'(x))$. The map h must be carefully chosen. For example, for $h : \{0, 1\}^{\text{poly}(\lambda)} \rightarrow \mathbb{G}_1, y' \mapsto g^{y'}$ for some fixed generator $g \in \mathbb{G}_1$, there is a huge difference between the modeled $\mathcal{RO} : \{0, 1\}^* \rightarrow \mathbb{G}_1$ and the instantiated hash function H . It is easy to compute the discrete logarithm of $H(x)$, which is $H'(x)$. In contrast, we assume the discrete logarithm of values returned by $\mathcal{RO}$ is hard to compute, since they are random group elements. A proper instantiation for h (called “admissible encoding” in [BF03]) depends on the concrete group $\mathbb{G}_1$. For elliptic curve groups, a good general strategy is to view $H'(x)$ as the x-coordinate of an element of the elliptic curve group. However, even then, care must be taken not to introduce (too much) bias when converting bit strings into coordinates, which live in a finite field $\mathbb{Z}_p$ for a prime number p . Apart from these cautionary remarks, in this thesis we will simply assume that random oracle model hash functions H that hash into groups $\mathbb{G}_1$ are designed suitably.

Security in the random oracle model. Let Π be some cryptographic construction (a list of algorithms). We say that we model H as a random oracle if any invocations of $H(x)$ in the construction are replaced with calls to a random oracle $\mathcal{RO}(x)$. In the security definition, the adversary $\mathcal{A}^{\mathcal{RO}}$ and the security game $\mathfrak{E}^{\mathcal{RO}}$ get oracle access to the random oracle. Oftentimes, this oracle access is not explicitly stated in standard security experiments, but when a theorem says “... is secure in the random oracle model”, then it is understood that oracle access is given and calls of the hash function in the construction are replaced with calls to $\mathcal{RO}$.

There are two ways to interpret a security proof in the random oracle model. First, as sketched above, the random oracle can be seen as an idealization of a good hash function. Intuitively, good hash functions should resist analysis so well that in the end, its images might as well be random, and the only way to gain any useful information about an image is to just compute the hash function. Second, the random oracle model can be seen as a restriction on the adversary. Security in the random oracle model guarantees security against adversaries that *do not exploit the hash function’s structure* and only

engage with the hash function by computing hash values.

Note that there are examples of constructions that can be proven secure in the random oracle model, but are completely *insecure* in practice when instantiated with *any* concrete hash function [CGH04]. These examples use the hash function H in a whitebox manner, constructing a scheme that becomes insecure if and only if the code of H is known (which is trivially the case for concrete hash functions, but in the random oracle model, there *is no* deterministic code that can compute $\mathcal{RO}$ reliably). Additionally, counterexamples against non-contrived, deployed constructions have been developed recently [KRS25]. As a consequence, a proof in the random oracle model does not guarantee real-world security. We do not have any reason to believe that the constructions in this thesis are vulnerable when the random oracle is instantiated with a good practical hash function. Counterexamples rely on the construction to evaluate arbitrary code supplied as input (either a contrived construction to allow for this [CGH04] or a construction that has this ability naturally [KRS25]). This is not the case for our schemes.

Standard tricks for security proofs. In random oracle model security proofs, we are presented with an adversary $\mathcal{A}^{\mathcal{RO}}$ that attempts to break some property. In the usual structure of security proofs, our goal is to construct an adversary $\mathcal{B}$, trying to break an assumption, which runs $\mathcal{A}^{\mathcal{RO}}$ as a subroutine. Because $\mathcal{A}^{\mathcal{RO}}$ is run as a subroutine by $\mathcal{B}$, it is $\mathcal{B}$'s task to answer $\mathcal{A}$'s queries to $\mathcal{RO}$. The easiest way for $\mathcal{B}$ to do so is by simply running the $\mathcal{RO}$ code in Figure 2.1 itself to answer $\mathcal{A}$'s queries. However, there are some standard tricks that $\mathcal{B}$ can employ to make full use of the random oracle model:

Observing random oracle queries.

Because $\mathcal{B}$ plays the role of the random oracle, it in particular gets to observe the queries that $\mathcal{A}$ makes to it. This is a powerful capability: It gives $\mathcal{B}$ some insight into the computations made by $\mathcal{A}$, which is typically hidden from real-world parties (as hashes can be computed privately).

Programming the random oracle.

$\mathcal{B}$ can change the way random oracle queries are answered. For any preimage x that has not been queried yet, $\mathcal{B}$ can set its hash $\mathcal{RO}(x)$ to any value y . We are going to say “ $\mathcal{B}$ programs/sets $\mathcal{RO}(x) := y$ ”, which means that $\mathcal{B}$ sets $y_x := x$ in Figure 2.1. Apart from sometimes setting y_x in a special way, $\mathcal{B}$ simply defaults to running the code specified by $\mathcal{RO}$ in Figure 2.1 when $\mathcal{A}$ queries the random oracle.

The main restriction is that $\mathcal{A}$ must not be able to distinguish proper random oracle responses from programmed random oracle responses chosen by $\mathcal{B}$. This means two things: First, when $\mathcal{B}$ programs $\mathcal{RO}(x) := y$, then y_x must not be defined yet (or $y_x = y$). Second, the value y must be distributed properly, i.e., it must be distributed uniformly random in S (or at least be indistinguishable from a uniformly random choice). If one of these conditions is not met, then we change the distribution of random oracle replies expected by $\mathcal{A}$ and hence lose the ability to argue about the adversary $\mathcal{A}$'s winning probability (which is always with respect to a proper random oracle, not with respect to $\mathcal{B}$).

However, even with these restrictions, programming the random oracle is still a powerful tool. For example, $\mathcal{B}$ can choose all hash values y in advance, which grants $\mathcal{B}$ foresight about which hash values are going to be chosen. Additionally, $\mathcal{B}$ can choose a random y and then cleverly choose a corresponding x , whereas the adversary $\mathcal{A}$ is restricted to the ordinary order of choosing x and then getting told a random y .

Of course, these strategies can be combined.

2.10 The generic group model

Similarly to how random oracles idealize the properties of a good hash function, the *generic group model* (GGM) [Nec94; Sho97; Mau05] idealizes the properties of a good *group* in the sense that it does not expose any “unwanted” structure. Essentially, in the generic group model, the adversary treats the bilinear group as a black box and only executes group and pairing operations.

We are going to prove security of our dynamically malleable signature scheme construction (Construction 6.11) in the generic group model.

Formal model of algorithms in generic groups. To model this black box treatment formally, an algorithm in the generic group model gets oracle access to oracles $\mathcal{O.op}_1, \mathcal{O.op}_2, \mathcal{O.op}_t$, allowing it to execute group operations in $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t$, respectively, and access to $\mathcal{O.pair}$ to execute the pairing operation. We call such an algorithm with access to these oracles “generic algorithm”. Following [Sho97], to ensure that the algorithm does not make any use of the group structure apart from querying the oracles, the algorithm only gets to see *random bit string encodings* $\mu_z(g) \in \mathcal{Y}_z$ of group elements $g \in \mathbb{G}_z$. Because the encodings are random, they do not have any algebraic structure that the adversary may use. They only serve as an interface for the oracles, which take encoded input and output the encoded result (for example, querying $\mathcal{O.op}_1(\mu_1(g), \mu_1(g'))$ yields the encoded group operation result $\mu_1(g \cdot g')$). In addition, there is a one-to-one relation between group elements and their encodings, i.e., the encoding functions $\mu_z : \mathbb{G}_z \rightarrow \mathcal{Y}_z$ are bijective. As a result of μ_z being injective, the algorithm can check equality of group elements by checking equality of their encodings ($\mu_z(g) = \mu_z(g')$ if and only if $g = g'$), and use the encodings in efficient lookup data structures. Furthermore, as a result of μ_z mapping surjectively into a well-defined set $\mathcal{Y}_z$, the algorithm can (obviously) sample a random group element by sampling a random encoding, as discussed below.

We define an execution of an algorithm $\mathcal{A}$ in the generic group model for bilinear group $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, e, p)$, encoding sets $\mathcal{Y}_1, \mathcal{Y}_2, \mathcal{Y}_t$, and generators $g_1 \in \mathbb{G}_1, g_2 \in \mathbb{G}_2$ as an experiment $\text{GGM}_{\mathcal{A}}((\mathbb{G}_z, \mathcal{Y}_z)_{z \in \{1,2,t\}}, e, p, g_1, g_2)$ shown in Figure 2.2.

Discussion and design choices regarding $\text{GGM}_{\mathcal{A}}((\mathbb{G}_z, \mathcal{Y}_z)_{z \in \{1,2,t\}}, e, p, g_1, g_2)$. First, the choice of making the encoding functions $\mu_z : \mathbb{G}_z \rightarrow \mathcal{Y}_z$ *bijective* is not a universally agreed standard for the generic group model. In other models, μ is often a random

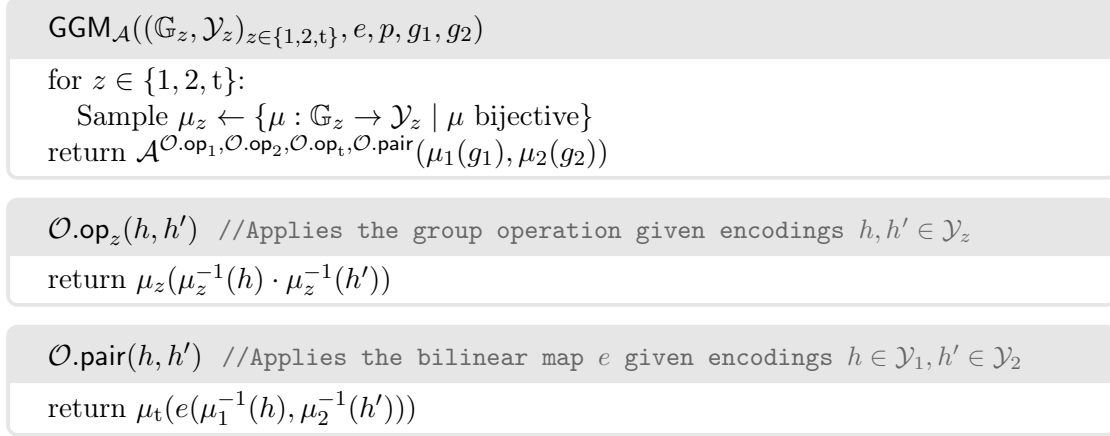


Figure 2.2: The code of an execution of $\mathcal{A}$ in the generic group model for a prime order bilinear group $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, e, p)$ and encoding sets $\mathcal{Y}_1, \mathcal{Y}_2, \mathcal{Y}_t \subset \{0, 1\}^*$ ($|\mathbb{G}_z| = |\mathcal{Y}_z| = p$), and generators g_1, g_2 .

injective function (which is not surjective). If the encoding space $\mathcal{Y}_z$ is much larger than the group $\mathbb{G}_z$, then $\mathcal{A}$, given limited runtime, may not be able to find any valid group element encodings $h \in \text{im}(\mu_z) \subset \mathcal{Y}_z$ that were not previously output by an oracle. This means that $\mathcal{A}$ is effectively restricted to doing group operations on the set of group elements it has already seen as input or as the result of prior group operations. However, for non-generic groups, one can usually easily write down any (random) group element $g \in \mathbb{G}_z$ without doing any group operations. For example, one can easily address a (random) element of $\mathbb{Z}_p^*$ by pointing at a random integer $g \in \{1, \dots, p-1\}$. For elliptic curve groups, with details omitted, one can often take an x coordinate $x \in \mathbb{F}_q$ and simply compute a fitting y coordinate according to the elliptic curve equation to obtain an element $(x, y) \in \mathbb{G}_1$. The issue that the GGM does not model this capability was identified in the context of the algebraic group model [LPS23]. They demonstrate that absence of this feature (which they call *oblivious sampling*) undermines the meaningfulness of algebraic/generic group proofs. In our version (and in [BFKT24]), the algorithm $\mathcal{A}$ *can* simply sample any random $h \leftarrow \mathcal{Y}_z$, which, because μ_z is bijective, certainly corresponds to *some* (unknown to $\mathcal{A}$) group element $g = \mu_z^{-1}(h)$. $\mathcal{A}$ can use h as input to oracle calls. As a result, a generic algorithm $\mathcal{A}$ (in our formulation) is able to run the BLS signature scheme, where a signature is $\sigma = \mathcal{H}(m)^{sk}$, using a hash function $\mathcal{H} : \{0, 1\}^* \rightarrow \mathcal{Y}_1$. This is in contrast to the version with large $\mathcal{Y}_z$, where hashing into the group cannot be faithfully executed by $\mathcal{A}$, given that the hashes $\mathcal{H}(m) \in \mathcal{Y}_1$ likely do not correspond to group elements under μ_1 .

As a minor design choice, one may note the absence of an inversion oracle to model computation of g^{-1} from g . Because the model of Figure 2.2 is defined for finite groups (of known order p) only, $\mathcal{A}$ can efficiently compute (the encoding of) g^{-1} as g^{p-1} using the $\mathcal{O.op}_z$ oracle and a square & multiply algorithm. As a simple illustration of the square & multiply algorithm idea, one can compute each element in the series $(g^{2^i})_{i=0}^{\lceil \log(p) \rceil}$ simply

by squaring its predecessor, and then use those values to compute g^{p-1} as $\prod_{i=0}^{\lfloor \log(p) \rfloor} (g^{2^i})^{b_i}$, where $b_i \in \{0, 1\}$ represent the bit decomposition of $p - 1$, i.e., $p - 1 = \sum_{i=0}^{\lfloor \log(p) \rfloor} b_i \cdot 2^i$ (this generalizes to exponents other than $p - 1$). Hence $\mathcal{A}$ can efficiently invert group elements without a dedicated inversion oracle, with no material impact on its running time (meaning that a ppt algorithm using a hypothetical inversion oracle can be rephrased as a ppt algorithm using the workaround above).

Finally, we note that our GGM definitions consider fixed groups and generators, and algorithms with no runtime restriction. This is in contrast to the rest of this thesis, where groups are chosen depending on a security parameter (see Definition 2.3) and the adversary is restricted to polynomial time. This GGM definition is not a restriction but rather a strengthening: The GGM results hold statistically for all groups and adversaries (though the guarantees will get worse and eventually trivial for adversaries that make many GGM queries). In particular, the results can be applied to a setting where the group is chosen at the start of an experiment depending on a security parameter.

Observation: Isomorphic groups are perfectly indistinguishable. It is easy to see that the GGM game behaves exactly the same for two *isomorphic* groups. Concretely, consider bilinear groups $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, e, p), (\mathbb{G}'_1, \mathbb{G}'_2, \mathbb{G}'_t, e', p)$ with isomorphisms $\phi_z : \mathbb{G}_z \rightarrow \mathbb{G}'_z$ that are compatible with the pairing, i.e., $\phi_t(e(g_1, g_2)) = e'(\phi_1(g_1), \phi_2(g_2))$. Then for their corresponding GGM games, $\Pr[\text{GGM}_{\mathcal{A}}((\mathbb{G}_z, \mathcal{Y}_z)_{z \in \{1,2,t\}}, e, p, g_1, g_2) = 1] = \Pr[\text{GGM}_{\mathcal{A}}((\mathbb{G}'_z, \mathcal{Y}_z)_{z \in \{1,2,t\}}, e', p, \phi_1(g_1), \phi_2(g_2)) = 1]$. This is easy to see: The game $\text{GGM}_{\mathcal{A}}((\mathbb{G}'_z, \mathcal{Y}_z)_{z \in \{1,2,t\}}, e', p, \phi_1(g_1), \phi_2(g_2))$ using encoding functions μ'_z behaves exactly like the game $\text{GGM}_{\mathcal{A}}((\mathbb{G}_z, \mathcal{Y}_z)_{z \in \{1,2,t\}}, e, p, g_1, g_2)$ using $\mu_z = \mu'_z \circ \phi_z$.

As a result, we can canonically use the bilinear group $(\mathbb{Z}_p, \mathbb{Z}_p, \mathbb{Z}_p, e, p)$ with $e(a, b) = a \cdot b$ and generators $g_1 = 1, g_2 = 1$ as the backing for the generic group model. Note that the group operation for $\mathbb{Z}_p$ is $+$, so when talking about $\mathbb{Z}_p$, we use additive notation, which is in contrast to the multiplicative notation in Figure 2.2 (in particular, 1 is indeed a generator and not the neutral element of $\mathbb{Z}_p$). This bilinear group is isomorphic to all bilinear groups $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, e, p)$ of the same prime order p , with isomorphisms $\phi_z : \mathbb{Z}_p \rightarrow \mathbb{G}_z$, $\phi_z(x) = g_z^x$. We then get $\Pr[\text{GGM}_{\mathcal{A}}((\mathbb{Z}_p, \mathcal{Y}_z)_{z \in \{1,2,t\}}, \cdot, p, 1, 1) = 1] = \Pr[\text{GGM}_{\mathcal{A}}((\mathbb{G}_z, \mathcal{Y}_z)_{z \in \{1,2,t\}}, e, p, g_1, g_2) = 1]$ using the observations above.

On the one hand, this perhaps illustrates that the generic group model is dangerous in the sense that it can be used to abstract even groups that are completely unsuitable for cryptographic purposes, such as $\mathbb{Z}_p$. Even the most basic assumption, the discrete logarithm assumption (Definition 2.6), is trivial for $\mathbb{Z}_p$: Using generator 1, the “discrete logarithm” of any element $x \in \mathbb{Z}_p$ to base 1 is simply $x \in \mathbb{Z}$ itself, since $x \cdot 1 = x \pmod p$ (where scalar multiplication with x in additive notation corresponds to exponentiation with x in multiplicative notation). This is one reason why, similar to the random oracle model, security in the GGM does not formally imply security in a real-world scenario with a concrete group. We discuss the interpretation of GGM proofs later in this section.

However, for modeling purposes, the backing group $\mathbb{Z}_p$ is convenient when doing GGM proofs, precisely because any group element is its own discrete logarithm. This allows us, intuitively speaking, to directly look at all operations that $\mathcal{A}$ makes through the lens

of discrete logarithms, which often makes reasoning easier. For this reason, we are going to use $\mathbb{Z}_p$ as a backing group in the following.

Security proofs in the generic group model. So far, we have only described what it means to execute a generic algorithm $\mathcal{A}$. Our goal is to prove statements of the form “no generic algorithm $\mathcal{A}$ can solve [problem]”, for example “no generic algorithm can compute discrete logarithms except with negligible probability” (cf. Definition 2.6).

This analysis is almost universally done using a strategy one may call “symbolic analysis”. There is no agreed-upon formalization of this strategy (we explain our view on it below). Intuitively, when doing symbolic analysis, we essentially replace secret values $x_i \in \mathbb{Z}_p$ with symbolic (polynomial) variables $X_i \in \mathbb{Z}_p[X_1, \dots, X_n]$. This enables very convenient analyses that are independent of concrete random values x_i . Typically, the analysis for assumptions is very simple in the symbolic setting (while a direct analysis of events in the GGM is difficult). For example, in the setting with symbolic variables, it will be easy to argue that the probability to solve the discrete logarithm problem is simply 0 for any adversary. We will see this later.

Formalizing symbolic analysis. First, we formalize symbolic analysis. This formalization is depicted in Figure 2.3. The figure shows an experiment $\text{GGM}_{\mathcal{B}}^{\text{hiddenVars}, X}$, where an algorithm $\mathcal{B}$ interacts with several oracles. Later, $\mathcal{B}$ will essentially be a game modeling security of some assumption. The experiment comes with two settings, a REAL and an IDEAL setting. REAL models an execution of $\mathcal{B}$ in the generic group model, while IDEAL models an execution of $\mathcal{B}$ in a symbolic setting where analysis of $\mathcal{B}$ is easy (as we explain later).

We discuss the main differences of the two settings. The $X = \text{REAL}$ setting models a “normal” execution of the generic group model as follows. The experiment chooses random values x_i and keeps them hidden from algorithm $\mathcal{B}$. The $x_i \in \mathbb{Z}_p$ should be thought of as secret exponents. However, while the x_i values are kept hidden, $\mathcal{B}$ can ask for group operations involving $x_1, \dots, x_n$ by specifying the computation as a polynomial $f \in \mathbb{Z}_p[X_1, \dots, X_n]$ and querying the $\mathcal{O}.\text{compute}_z(f)$, which returns the encoding $\mu_z(f(x_1, \dots, x_n))$. In the $X = \text{IDEAL}$ setting, the experiment never uses the random values x_i , but uses polynomial variables X_i in their stead. As in the REAL setting, $\mathcal{B}$ can ask for group operations involving the X_i through the $\mathcal{O}.\text{compute}_z(f)$ oracle. In this IDEAL setting, however, the oracle uses the polynomial variables X_i instead of the x_i to answer these queries, i.e., it returns the encoding $\mu_z(f(X_1, \dots, X_n)) = \mu_z(f)$.

Contrasted with Figure 2.2, the group in Figure 2.3 is fixed to $\mathbb{Z}_p$ with generator 1, which does not make a difference compared to any other group/generators, as discussed on Page 31. In particular, this explains the use of additive notation in Figure 2.2. Furthermore, the encoding functions μ_z in Figure 2.3 are not chosen fully at the start of the experiment. Instead, their images and preimages are lazily chosen randomly when they are first accessed (via $\mathcal{O}.\text{encode}_z, \mathcal{O}.\text{decode}_z$). Lazy sampling is necessary because the encoding functions now encode polynomials ($\mu_z : \mathbb{Z}_p[X_1, \dots, X_n] \rightarrow \mathcal{Y}_z$), hence no injective encoding function exists. However, in the $X = \text{REAL}$ setting, the only elements

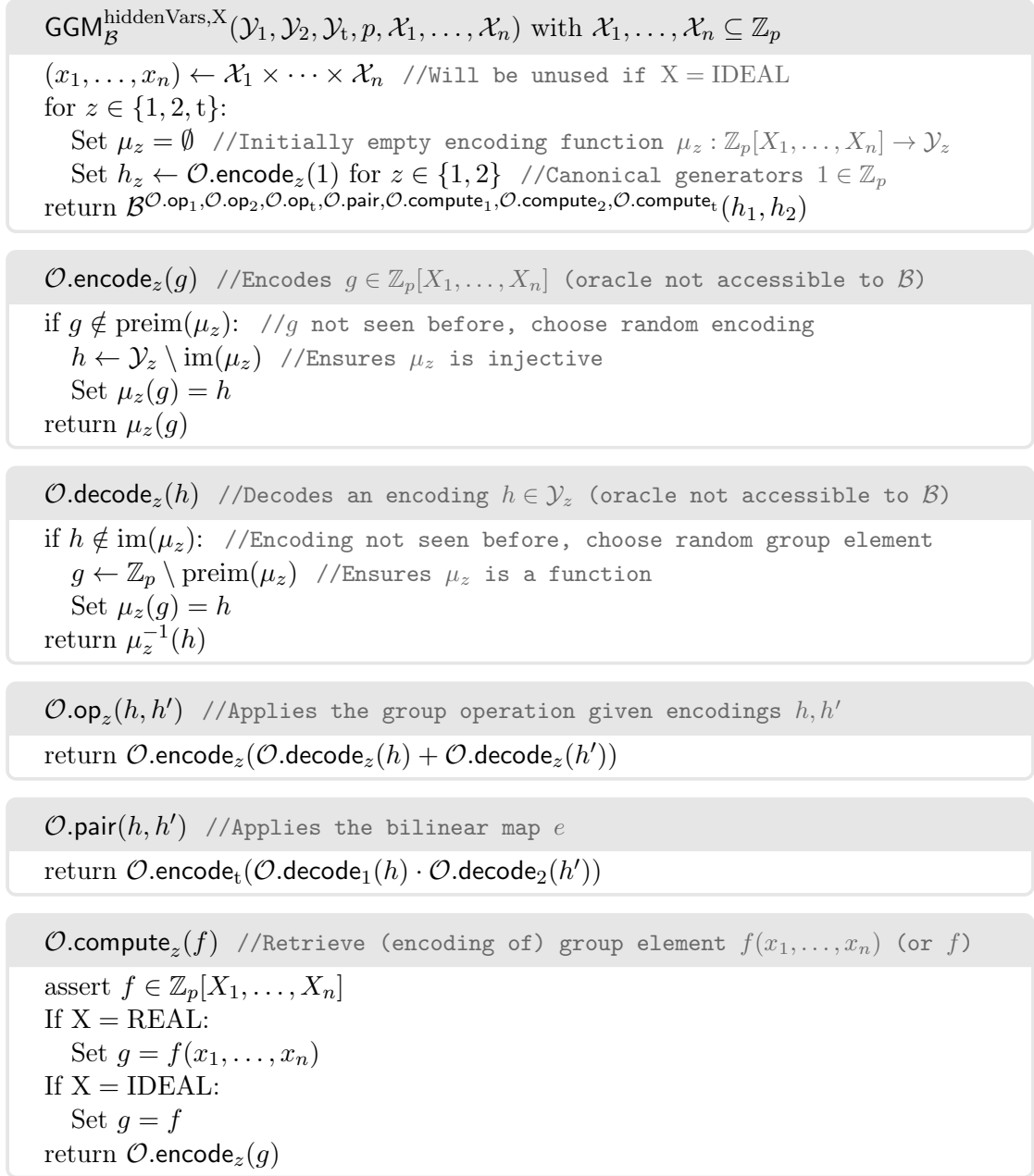


Figure 2.3: Execution of $\mathcal{B}$ in the generic group model with explicitly modeled secret variables $x_1, \dots, x_n$. X = IDEAL models a setting enabling symbolic analysis of $\mathcal{B}$, while X = REAL models a setting comparable to Figure 2.2.

being encoded are in $\mathbb{Z}_p$ (which is a subset of $\mathbb{Z}_p[X_1, \dots, X_n]$), and here, lazy sampling is equivalent to choosing all images/preimages of the permutation (restricted to $\mathbb{Z}_p$) at the start as in Figure 2.2.

We note that for $X = \text{IDEAL}$, it is possible for the experiment to run out of encodings during $\mathcal{O}.\text{encode}_z$, i.e., to have $\mathcal{Y}_z \setminus \text{im}(\mu_z) = \emptyset$, at which point the experiment is, strictly speaking, not well-defined. However, exhausting the encoding space $\mathcal{Y}_z$ requires at least $\lfloor p/3 \rfloor$ oracle queries made by $\mathcal{B}$, at which point our main theorem (Theorem 2.26) around Figure 2.3 does not offer any meaningful guarantee anymore, anyway. For this reason, one may handle the event of running out of encodings in any arbitrary way.

Symbolic analysis is sufficient to establish security in the GGM. The following theorem establishes that $\mathcal{B}$ cannot distinguish between the normal generic group setting $X = \text{REAL}$ and the symbolic analysis setting $X = \text{IDEAL}$. This means that the simple analysis of the symbolic setting carries over to the normal setting. For example, the discrete logarithm game $\mathcal{B}$ *never* outputs 1 for $X = \text{IDEAL}$ (the adversary cannot win the game $\mathcal{B}$). The upcoming theorem then implies that $\mathcal{B}$ (almost) never outputs 1 for $X = \text{REAL}$, either, which then shows that the discrete logarithm problem is hard in the generic group model. We discuss details below.

Theorem 2.26 (Symbolic analysis in the GGM). *Let $p \in \mathbb{N}$ be prime, $\mathcal{X}_1, \dots, \mathcal{X}_n \subseteq \mathbb{Z}_p$, and $d_{\max} \in \mathbb{N}$. Let $\mathcal{B}$ be a (generic) algorithm. Let $q \in \mathbb{N}$ be an upper bound such that $\mathcal{B}$ makes at most $q - 2$ queries to its oracles. Suppose that for all queries $\mathcal{O}.\text{compute}_z(f)$, we have $\deg(f) \leq d_{\max}$. Then*

$$\left| \begin{array}{l} \Pr[\text{GGM}_{\mathcal{B}}^{\text{hiddenVars}, \text{REAL}}(\mathcal{Y}_1, \mathcal{Y}_2, \mathcal{Y}_t, p, \mathcal{X}_1, \dots, \mathcal{X}_n) = 1] \\ - \Pr[\text{GGM}_{\mathcal{B}}^{\text{hiddenVars}, \text{IDEAL}}(\mathcal{Y}_1, \mathcal{Y}_2, \mathcal{Y}_t, p, \mathcal{X}_1, \dots, \mathcal{X}_n) = 1] \end{array} \right| \leq \frac{12q^2}{p} + \frac{\binom{3q}{2} \cdot 2d_{\max}}{\min_i(|\mathcal{X}_i|)},$$

which can be upper-bounded by $\frac{q^2 \cdot (9 \cdot d_{\max} + 12)}{\min_i(|\mathcal{X}_i|)}$.

In particular, if p and all $|\mathcal{X}_i|$ are superpoly in λ while $d_{\max}, q$ are polynomial, and we can show that $\Pr[\text{GGM}_{\mathcal{B}}^{\text{hiddenVars}, \text{IDEAL}}(\mathcal{Y}_1, \mathcal{Y}_2, \mathcal{Y}_t, p, \mathcal{X}_1, \dots, \mathcal{X}_n) = 1] \leq \text{negl}(\lambda)$, then there is also a negligible function upper-bounding REAL executions. We will prove Theorem 2.26 on Page 37.

Symbolic analysis framework application examples. Before going into the proof, we give some examples of how to use Theorem 2.26 to prove hardness of problems in the generic group model. As the simplest example, take the discrete logarithm problem (Definition 2.6) in $\mathbb{G}_1$. Let $\mathcal{A}$ be a generic adversary against the discrete logarithm problem, and let $\mathcal{B}$ be the game algorithm that models the discrete logarithm problem in the generic group model as an interactive game. We can model the game $\mathcal{B}$ in the context of $\text{GGM}_{\mathcal{B}}^{\text{hiddenVars}, X}(\mathcal{Y}_1, \mathcal{Y}_2, \mathcal{Y}_t, p, \mathcal{X}_1)$ as follows. $\mathcal{X}_1 = \mathbb{Z}_p$ (i.e., we choose the discrete logarithm from all of $\mathbb{Z}_p$), and $\mathcal{B}$ works as follows:

- $\mathcal{B}^{\mathcal{O}.op_1, \mathcal{O}.op_2, \mathcal{O}.op_t, \mathcal{O}.pair, \mathcal{O}.compute_1, \mathcal{O}.compute_2, \mathcal{O}.compute_t}(h_1, h_2)$ queries for the challenge element $h_1^* = \mathcal{O}.compute_1(X_1)$ $\P$ which is the encoding of $x_1 \cdot 1$ for a random hidden discrete $\log x_1 \leftarrow \mathbb{Z}_p$ w.r.t. generator 1 of $\mathbb{Z}_p$.
- It then hands h_1^* to $\mathcal{A}^{\mathcal{O}.op_1, \mathcal{O}.op_2, \mathcal{O}.op_t, \mathcal{O}.pair}(h_1, h_2)$. $\P$ The generic adversary $\mathcal{A}$ gets the same oracles as in Figure 2.2.
- Whenever $\mathcal{A}$ queries its oracles, $\mathcal{B}$ just relays the query to its own corresponding oracles and gives the result back to $\mathcal{A}$.
- Eventually, $\mathcal{A}$ outputs a discrete logarithm guess $x' \in \mathbb{Z}_p$.
- $\mathcal{B}$ outputs 1 iff $\mathcal{O}.compute_1(x') = h_1^*$. $\P$ This check is equivalent to $\mu_1(x') = \mu_1(x_1)$ in the REAL setting, and $\mu_1(x') = \mu_1(X_1)$ in the IDEAL setting, which in turn are equivalent to $x' = x_1$ and $x' = X_1$, respectively.

From the point of view of $\mathcal{A}$, using $X = \text{REAL}$ faithfully models an execution against the discrete logarithm problem, as in our initial GGM modeling Figure 2.2. Analyzing the $X = \text{IDEAL}$ case, we can easily see that $\mathcal{B}$ will *never* output 1: No possible guess $x' \in \mathbb{Z}_p$ will equal the symbol X_1 (in the polynomial ring, $X_1 \neq x'$ for all $x' \in \mathbb{Z}_p$). Using Theorem 2.26, we hence can bound the probability that $\mathcal{A}$ making at most $q_{\mathcal{A}}$ GGM oracle queries, breaks the discrete logarithm problem in the generic group model by $\frac{21(q_{\mathcal{A}}+2)^2}{p}$ (using that $\mathcal{B}$ only makes two oracle queries in addition to the $q_{\mathcal{A}}$ queries triggered by $\mathcal{A}$, and asks at most for degree $d_{\max} = 1$ polynomials).

We will see a more involved GGM proof with Theorem 6.12 later. To summarize the big picture, we have started with the original group $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, e, p)$ as the backing group for the generic group model, replaced it by the isomorphic bilinear group $\mathbb{Z}_p$ with generator 1 (which is unconditionally undetectable by $\mathcal{B}$ as argued on Page 31), and finally replaced $\mathbb{Z}_p$ with the polynomial ring $\mathbb{Z}_p[X_1, \dots, X_n]$, which is undetectable by $\mathcal{B}$ according to Theorem 2.26. With those arguments, it suffices to do analysis over $\mathbb{Z}_p[X_1, \dots, X_n]$, which is usually much easier, as demonstrated with the discrete logarithm example above.

Proving Theorem 2.26 For the proof of Theorem 2.26, we will crucially rely on the following famous lemma.

Lemma 2.27 (Schwartz-Zippel). *Let $n \in \mathbb{N}$. Let $\mathbb{F}$ be a field, let $\mathcal{X}_1, \dots, \mathcal{X}_n \subseteq \mathbb{F}$ be finite non-empty subsets, $f \in \mathbb{F}[X_1, \dots, X_n]$, $f \neq 0$. Then*

$$\Pr[f(x_1, \dots, x_n) = 0 \mid (x_1, \dots, x_n) \leftarrow \mathcal{X}_1 \times \dots \times \mathcal{X}_n] \leq \frac{\deg(f)}{\min_i(|\mathcal{X}_i|)}$$

We are using a slightly generalized version of the lemma, allowing the x_i to be chosen randomly from different domains $\mathcal{X}_i$. The original lemma can be recovered by setting $\mathcal{X}_1 = \mathcal{X}_2 = \dots = \mathcal{X}_n$.

The lemma implies that for two polynomials $g \neq g'$, we can straightforwardly bound the collision probability $\Pr[g(\vec{x}) = g'(\vec{x})] = \Pr[(g - g')(\vec{x}) = 0]$ using this lemma.

Proof of Lemma 2.27. Let $\mathbb{F}$ be a field. The proof is via induction over the number of variables n .

For $n = 1$, any (univariate) polynomial $f \in \mathbb{F}[X_1] \setminus \{0\}$ has at most $\deg(f)$ roots in $\mathcal{X}_1 \subseteq \mathbb{F}$. Hence the probability of hitting one of those roots randomly is $\Pr[f(x_1) = 0 \mid x_1 \leftarrow \mathcal{X}_1] \leq \frac{\deg(f)}{\min_i |\mathcal{X}_i|}$ as required.

Let $n > 1$ and assume the statement is true for $n - 1$ variables. Let $\mathcal{X}_1, \dots, \mathcal{X}_n \subseteq \mathbb{F}$ be non-empty finite subsets, let $f \in \mathbb{F}[X_1, \dots, X_n]$, $f \neq 0$. We factor out the X_1 terms in f , writing

$$f = \sum_{i=0}^{\deg(f)} X_1^i \cdot f_i(X_2, \dots, X_n)$$

for some polynomials $f_0, \dots, f_{\deg(f)} \in \mathbb{F}[X_2, \dots, X_n]$.

Let $k \in \{0, \dots, \deg(f)\}$ be the largest index such that $f_k \neq 0$. Note that $\deg(f) \geq \deg(X^k \cdot f_k) = k + \deg(f_k)$, hence $\deg(f_k) \leq \deg(f) - k$. Since f_k is over $n - 1$ variables, we get (from induction)

$$\Pr[f_k(x_2, \dots, x_n) = 0] \leq \frac{\deg(f) - k}{\min_{i \in \{2, \dots, n\}} (|\mathcal{X}_i|)}.$$

Furthermore, let $y_2, \dots, y_n \in \mathbb{F}$ be arbitrary scalars. Consider the univariate polynomial $f(X_1, y_2, \dots, y_n) \in \mathbb{F}[X_1]$. If $f_k(y_2, \dots, y_n) \neq 0$, then $f(X_1, y_2, \dots, y_n) \neq 0$, indeed we even get $\deg(f(X_1, y_2, \dots, y_n)) = k$. Given that $f(X_1, y_2, \dots, y_n)$ has only one variable, the $n = 1$ induction base case gives us

$$\Pr[f(x_1, y_2, \dots, y_n) = 0] \leq \frac{k}{|\mathcal{X}_1|}$$

for all $y_2, \dots, y_n \in \mathbb{Z}_p$ with $f_k(y_2, \dots, y_n) \neq 0$.

Overall, we get

$$\begin{aligned} & \Pr[f(x_1, \dots, x_n) = 0] \\ &= \Pr[f(x_1, \dots, x_n) = 0 \wedge f_k(x_2, \dots, x_n) = 0] \\ & \quad + \Pr[f(x_1, \dots, x_n) = 0 \wedge f_k(x_2, \dots, x_n) \neq 0] \\ &\leq \Pr[f_k(x_2, \dots, x_n) = 0] + \Pr[f(x_1, \dots, x_n) = 0 \mid f_k(x_2, \dots, x_n) \neq 0] \\ &\leq \frac{\deg(f) - k}{\min_{i \in \{2, \dots, n\}} (|\mathcal{X}_i|)} + \frac{k}{|\mathcal{X}_1|} \\ &\leq \frac{\deg(f)}{\min_{i \in \{1, \dots, n\}} (|\mathcal{X}_i|)} \end{aligned}$$

where the probability is over $(x_1, \dots, x_n) \leftarrow \mathcal{X}_1 \times \dots \times \mathcal{X}_n$. $\square$

The proof is largely taken from [Wik23] and only repeated here to deal with the minor generalization of multiple sets $\mathcal{X}_i$ (the original lemma has $\mathcal{X}_1 = \dots = \mathcal{X}_n$).

We can now prove Theorem 2.26, using the Schwartz-Zippel lemma to show that there

is a 1:1 relation between polynomials and their evaluations, bounding the probability of the bad event $\text{fail}_{\text{coll}}$ that two different polynomials in the IDEAL setting evaluate to the same concrete value in the REAL setting.

Proof of Theorem 2.26. Let $G_{\text{ideal}} = \text{GGM}_{\mathcal{B}}^{\text{hiddenVars, IDEAL}}(\mathcal{Y}_1, \mathcal{Y}_2, \mathcal{Y}_t, p, \mathcal{X}_1, \dots, \mathcal{X}_n)$ and $G_{\text{real}} = \text{GGM}_{\mathcal{B}}^{\text{hiddenVars, REAL}}(\mathcal{Y}_1, \mathcal{Y}_2, \mathcal{Y}_t, p, \mathcal{X}_1, \dots, \mathcal{X}_n)$.

For $X \in \{\text{real}, \text{ideal}\}$, consider the altered game G'_X that works like G_X except that in $\mathcal{O}.\text{decode}_z$, instead of choosing $g \leftarrow \mathbb{Z}_p \setminus \text{preim}(\mu_z)$, we choose $g \leftarrow \mathbb{Z}_p$. In principle, this may lead to $\mathcal{O}.\text{decode}_z$ overwriting some $\mu_z(g)$, which we tolerate in G'_X . Let $\text{fail}_{\text{diff}}$ be the event that at some point during the execution of G'_X , the random choices during $g \leftarrow \mathbb{Z}_p$ in $\mathcal{O}.\text{decode}_z$ result in a previously assigned value, i.e., some $g \in \text{preim}(\mu_z)$. Because $|\text{preim}(\mu_z)| \leq 3q$ (where $q - 2$ is an upper-bound of the number of oracle queries made by $\mathcal{B}$ and the -2 accounts for the two initial $\mathcal{O}.\text{encode}_z(1)$ calls), and the random choice under consideration happens at most $2q$ times, we have that $\Pr[\text{fail}_{\text{diff}}] \leq 2q \cdot 3q/p = 6q^2/p$. Note that as long as $\neg \text{fail}_{\text{diff}}$, there is no difference between G_X and G'_X , i.e., $(G'_X = 1 \wedge \neg \text{fail}_{\text{diff}}) \Leftrightarrow (G_X = 1 \wedge \neg \text{fail}_{\text{diff}})$. Applying the difference lemma [Sho04], we get that

$$|\Pr[G'_X = 1] - \Pr[G_X = 1]| \leq \Pr[\text{fail}_{\text{diff}}] \leq 6q^2/p.$$

Let $\phi : \mathbb{Z}_p[X_1, \dots, X_n] \rightarrow \mathbb{Z}_p$ be the evaluation homomorphism w.r.t. the x_i chosen by the game, i.e., $\phi(f) = f(x_1, \dots, x_n)$ and for sets $S \subseteq \mathbb{Z}_p[X_1, \dots, X_n]$, we set $\phi(S) = \{f(x_1, \dots, x_n) \mid f \in S\}$. Let $\text{fail}_{\text{coll}}$ be the event that during a run of G'_{ideal} , we encounter two different polynomials $g, g' \in \text{preim}(\mu_z)$ for some $z \in \{1, 2, t\}$ such that $g \neq g'$ but $\phi(g) = \phi(g')$. Because during G'_X , at most $3q$ values are added to $\text{preim}(\mu_z)$ (over all $z \in \{1, 2, t\}$ and including values that are later overwritten by $\mathcal{O}.\text{decode}_z$), we have at most $\binom{3q}{2}$ pairs of polynomials to consider. The degree of those polynomials is at most $2d_{\text{max}}$ (where the factor 2 comes from the $\mathcal{O}.\text{pair}$ oracle). Putting this together, we get

$$\Pr[\text{fail}_{\text{coll}}] \leq \binom{3q}{2} \cdot \frac{2d_{\text{max}}}{\min_i(|\mathcal{X}_i|)}$$

from the Schwartz-Zippel lemma (Lemma 2.27) and the union bound.

Let μ_z^{IDEAL} and μ_z^{REAL} denote the encoding functions μ_z in G'_{ideal} and G'_{real} , respectively. Comparing a run of G'_{ideal} with a run of G'_{real} with the same randomness, we claim that the oracle outputs are exactly the same as long as $\text{fail}_{\text{coll}}$ does not happen.

This claim can be checked for each individual $\mathcal{O}.\text{compute}_z$, $\mathcal{O}.\text{op}_z$, and $\mathcal{O}.\text{pair}_z$ query using the invariant that before and after every query, either $\text{fail}_{\text{coll}}$ occurs, or a 1:1 relation between μ_z^{IDEAL} and μ_z^{REAL} through ϕ is maintained. Formally, this 1:1 relation can be expressed as $\text{im}(\mu_z^{\text{IDEAL}}) = \text{im}(\mu_z^{\text{REAL}})$ and $\phi((\mu_z^{\text{IDEAL}})^{-1}(h)) = (\mu_z^{\text{REAL}})^{-1}(h)$ for all $h \in \text{im}(\mu_z^{\text{IDEAL}})$, $z \in \{1, 2, t\}$ (which in particular implies that $g \in \text{preim}(\mu_z^{\text{IDEAL}}) \Leftrightarrow \phi(g) \in \text{preim}(\mu_z^{\text{REAL}})$). We highlight that (1) during $\mathcal{O}.\text{encode}_z(g)$, assuming $\neg \text{fail}_{\text{coll}}$ gives us that $g \in \text{preim}(\mu_z^{\text{IDEAL}}) \Leftrightarrow \phi(g) \in \text{preim}(\mu_z^{\text{REAL}})$, keeping both experiments synchronized. Furthermore, we highlight that (2) during $\mathcal{O}.\text{decode}_z(h)$, by definition of

G'_X , we choose preimages from $\mathbb{Z}_p$, which may overwrite values, but does so in the same manner in both G'_{ideal} and G'_{real} , maintaining the invariant.

Applying the difference lemma again, we get that

$$|\Pr[G'_{\text{ideal}} = 1] - \Pr[G'_{\text{real}} = 1]| \leq \Pr[\text{fail}_{\text{coll}}] \leq \frac{\binom{3q}{2} \cdot 2d_{\max}}{\min_i(|\mathcal{X}_i|)}.$$

Overall, with triangle inequality, we get for the original games G_{ideal} and G_{real} that

$$|\Pr[G_{\text{ideal}} = 1] - \Pr[G_{\text{real}} = 1]| \leq 2 \cdot 6q^2/p + \frac{\binom{3q}{2} \cdot 2d_{\max}}{\min_i(|\mathcal{X}_i|)}$$

as required. $\square$

Finally, we make the following simple but useful observation, which essentially states that in a symbolic GGM execution, the adversary (and game) are limited to affine linear operations in the secret variables X_i .

Lemma 2.28 (The structure of elements in ideal GGM executions). *Consider an execution of $\text{GGM}_{\text{IDEAL}}^{\text{hiddenVars}, \mathcal{B}}(\mathcal{Y}_1, \mathcal{Y}_2, \mathcal{Y}_t, p, \mathcal{X}_1, \dots, \mathcal{X}_n)$. Let $Q_z \subset \mathbb{Z}_p[X_1, \dots, X_n]$ be the set of polynomials queried to $\mathcal{O}.\text{compute}_z$. Invariantly, it holds that*

- $\text{preim}(\mu_1) \subset \text{span}_{\mathbb{Z}_p}(Q_1 \cup \{1\})$
- $\text{preim}(\mu_2) \subset \text{span}_{\mathbb{Z}_p}(Q_2 \cup \{1\})$
- $\text{preim}(\mu_t) \subset \text{span}_{\mathbb{Z}_p}(Q_1 \cup Q_2 \cup Q_t \cup (Q_1 \cdot Q_2) \cup \{1\})$

where $Q_1 \cdot Q_2 = \{f_1 \cdot f_2 \mid f_1 \in Q_1, f_2 \in Q_2\}$.

The presence of the 1 (constant terms) in the spans, as well as the presence of Q_1 and Q_2 (in addition to $Q_1 \cdot Q_2$), is due to $\mathcal{B}$ receiving generators $h_1 = \mu_1(1), h_2 = \mu_2(1)$ as input and being allowed to obviously sample.

Instantiation in practice. In practice, we would use a bilinear group such as BLS12-381 for a scheme we prove secure in the generic group model, cf. Section 2.3.

Security in the generic group model. The generic group model is commonly used to give an indication on whether a certain algebraic assumption holds. For example, it would be surprising if we were able to unconditionally prove that the discrete logarithm problem is hard for some group, as that would imply $\mathbf{P} \neq \mathbf{NP}$. In lieu of a proper proof, we can at least prove that the discrete logarithm problem is hard *in the generic group model*.

As with the random oracle model, there are two ways to interpret a proof in the generic group model. First, we can interpret the generic group model as an idealization of a “good” group, which exposes no extra structure that can be exploited to break an

assumption. In this view, intuitively, we may say that the discrete logarithm is hard in groups that do not expose any extra structure.

Second, the generic group model can be seen as a restriction on the adversary. A proof in the generic group model *guarantees* security against *generic* algorithms, i.e., algorithms that treat the group as a black box and do not depend on the concrete group’s structure.

The issue in both interpretation is that the best algorithms to compute discrete logarithms (which implies the ability to break any typical bilinear group assumption) in bilinear groups are not generic. Bilinear groups do expose some usable structure in the form of $\mathbb{G}_t$, which is a subgroup of $\mathbb{F}_{q^k}^*$ for small k (e.g., $\mathbb{G}_t \subset \mathbb{F}_{q^{12}}^*$ for BLS12-381). Making use of the concrete structure of $\mathbb{F}_{q^k}^*$, index calculus can compute discrete logarithms significantly more efficiently than through generic group operations. This is not an issue in practice: This method of solving discrete logarithms is still not polynomial time and far from practical. However, it does showcase a limitation of the applicability of the generic group model.

In addition to those examples for practical non-generic attack vectors and operations, and similarly to the random oracle model, there are (contrived) examples of assumptions that can be proven to be true in the generic group model, which are false for any concrete bilinear group [Den02]. Further criticism can be found in [Fis00].

Still, the generic group model can be a useful tool to give *some* credence to cryptographic assumptions, which is usually the best we can get without proving $\mathbf{P} \neq \mathbf{NP}$. Despite criticism and somewhat better alternatives (as discussed next), we use the generic group model for this work. We note that the generic group model is still widely used. We also note that the main GGM proof (Theorem 6.12) in this thesis translates to other models, with its core arguments untouched.

Alternative models. Our flavor of the generic group model (Figure 2.2) is inspired by Shoup’s model [Sho97]. Maurer’s model [Mau05] captures a similar spirit, but does so without assigning group elements any encodings. Instead, all group elements are stored in a list and the oracles take indices of that list as input. This results in a model in which the adversary is even more restricted than in Shoup’s model, meaning that security proofs in Maurer’s model do not necessarily carry over to Shoup’s model [Zha22].

To tackle the criticism that index calculus shows that the generic group model does not capture the best known attacks, the *semi-generic group model* [JR10] treats $\mathbb{G}_1$ and $\mathbb{G}_2$ as generic, but gives the *actual* (not randomly encoded) representations of group elements in $\mathbb{G}_t$ to the adversary (allowing it to, for example, apply index calculus in $\mathbb{G}_t$).

An important alternative to the generic group model is the *algebraic group model* (AGM) [FKL18], where the adversary gets to see the original encodings of group elements, but whenever it outputs a group element to some security experiment, it must additionally supply the element’s discrete logarithms. This model imposes fewer restrictions on the adversary. For example, because it can see standard encoded group elements, an AGM adversary can also apply index calculus. Oblivious sampling can also be added to the AGM [LPS23].

Chapter 3

Zero-knowledge proof systems

Remember... All I'm offering is the truth. Nothing more

Morpheus

The privacy-preserving constructions in this thesis heavily rely on zero-knowledge proofs of knowledge (ZKPoK) [GMR19] for their privacy properties. Let $\mathcal{R}$ be an **NP**-relation, let $stmt$ be some “statement”, and let w be a “witness”, i.e., $(stmt, w) \in \mathcal{R}$. A ZKPoK is a protocol between a prover, who gets input $(stmt, w)$, and a verifier, who only gets $stmt$ as input. Informally speaking, the prover tries to convince the verifier that he knows a witness w for $stmt$, without revealing w to the verifier. In more detail, from a ZKPoK, we want the following properties:

- Proof of knowledge: The prover should only be able to convince the verifier if the prover *knows* a witness w — it is not enough for that witness to just *exist*; it must be *known* by the prover.
- Zero-knowledge: The verifier should not be able to learn *anything* about w other than its existence.

These properties lie at the heart of privacy-preserving constructions: Without zero-knowledge protocols, the prover would be forced to reveal their secret w in plain so that the verifier can check $(stmt, w) \in \mathcal{R}$. For example, take an access control scenario where the verifier wants to check that the prover holds a signature on “person x has access to y ”. Revealing the signature in plain convinces the verifier, but leaks the prover’s identity x . With ZKPoK, the verifier can instead accept a *proof* of w for the relation $\mathcal{R} = \{(stmt, w) \mid stmt = (pk, y), w = (\sigma, x), \sigma \text{ is a valid signature on “person } x \text{ has access to } y” \text{ under } pk\}$. Because the proof is zero-knowledge, the verifier will not learn anything about w , maintaining privacy. But because of the proof of knowledge property, the verifier can be sure that the prover is in possession of a valid signature σ for resource y , ensuring access is only granted to people who have been given a signature. The signature example also shows why the proof of knowledge aspect is important: For the relation above, a valid signature clearly *exists* (in a mathematical “ $\exists$ ” sense), so just convincing the

verifier of the *existence* of a witness is not meaningful. In contrast, *knowing* a signature *is* meaningful, since that implies that a signature has been given to the prover by the signer.

It is not immediately clear how to model that a machine *knows something* or *learns nothing*. In Section 3.3, we will see concrete definitions for this, but those are very technical. For now, let us discuss the basic underlying ideas, even though our concrete definitions later will deviate somewhat from them.

We could naively say that a machine *knows* a value if it is stored in its memory. However, we want to be able to give a meaningful definition even for black-box algorithms where we do not know exactly how they work. In particular, we want to be able to say “an (arbitrary) adversary $\mathcal{A}$ knows w ”. With the memory definition, we quickly run into encoding issues: For example, what if $\mathcal{A}$ never stores w in plain, but encodes it with inverted bits? Arguably, such an $\mathcal{A}$ would still “know” w , but it cleverly evades our definition.

For this reason (and others), we instead define knowledge of a machine as *whatever can be efficiently computed given the machine’s view*, i.e., its input, randomness, and received messages/oracle answers (see Definition 2.1). More concretely, we say that a machine $\mathcal{A}$ knows a value (with some desired properties) if there exists another algorithm (the *extractor*) that can, given $\mathcal{A}$ ’s view, efficiently compute such a value. This clearly fixes the encoding issue: The extractor could just repeat $\mathcal{A}$ ’s computation, then decode the value stored in $\mathcal{A}$ ’s state. It also more generally turns out to be the more useful definition, as we will later see proofs of knowledge for which extractors work completely independently of internal encodings used by $\mathcal{A}$.

With a similar approach, we can say that a machine *does not learn* a value w from a protocol execution if the set of efficiently computable data (its “knowledge”) remains unchanged before and after the protocol. More specifically, we want that any computation that relies on the protocol transcript can instead be run without that transcript, yielding essentially the same results. We can model this by requiring that there exists an algorithm (the *simulator*) that can generate convincing protocol transcripts *without* needing w . So any computation that needs the protocol transcript from a prover (who knows w) can instead run the simulator to generate (without w) a transcript and continue the computation as usual.

Zero-knowledge proofs of knowledge in the literature. Because ZKPoK play a *crucial* role for the constructions and even for the security definitions in this thesis, it is important to understand them correctly. Unfortunately, ZKPoK are complicated and information about them is scattered throughout many papers and “folklore”, accessible only through extensive study, if at all. There are, generally, two ways that research papers proposing new privacy-preserving constructions handle this. They either (1) use a specific ZKPoK system and its specific security guarantees (e.g., Fiat-Shamir [FS86] and a forking lemma [PS96; BN06; EK18; HKL19]), or (2) use informal phrases like “use the extractor to compute a witness” or “use the simulator to simulate the protocol” without further detail in their security proofs.

The issue with (1) is that details of a concrete ZKPoK are integrated into the construction and security proof. This makes it difficult to understand the construction and its proof, and difficult to exchange the concrete ZKPoK for another one. Ideally, instead of encoding a concrete ZKPoK into the construction, one would use a suitable formal abstraction of ZKPoK. The challenge here is to find a suitable abstraction that still facilitates security proofs.

The issue with (2) is that it leaves make-or-break details to the reader that are exceedingly easy to get wrong. For example, folklore suggests that ZKPoK are *usually* constructed by applying the Fiat-Shamir transform [FS86] to some interactive ZKPoK with certain properties. However, as we will see in this chapter (particularly Section 3.5), this approach actually offers only a surprisingly weak notion of security, insufficient for most anonymous credential constructions.

Contribution. To avoid the issues mentioned above, we give a formal security definition for ZKPoK in this chapter, and explain how to construct ZKPoK that conform to this definition.

Our security definition (Definition 3.22) for ZKPoK is aimed at privacy-preserving construction designers. Essentially, our definition gives us a relatively straightforward way to formally express the phrases “use the extractor to compute a witness” and “use the simulator to simulate the protocol”. This enables us to design and prove our constructions secure without involving too many implementation details of the used ZKPoK, while still giving precise definitions what we require of them.

Instantiating a ZKPoK is done in two steps: One first designs a *Sigma protocol* for the statement one wants to prove. Sigma protocols have very *simple* zero-knowledge and proof of knowledge properties, making it easy to design such protocols. We show how to construct Sigma protocols and provide a convenient formalization that helps implement several “standard tricks” (e.g., Sections 3.2.2, 3.2.4 and 3.2.5).

Unfortunately, the properties of Sigma protocols are either too weak or too far removed from what we need in our security proofs in our larger constructions. In the second step, the Sigma protocol is then *upgraded* to one that *does* fulfill our strong definitions. We show how to achieve this with the randomized Fischlin transform [Fis05; KS22] in Section 3.4.

This chapter should be considered a *systemization of knowledge* effort for Schnorr-based [Sch89] ZKPoK. By its nature, most of this chapter’s content is not fully original research, but rather an effort to organize and adapt known facts and constructions from the research area into a coherent unified framework. Additionally, we discover and solve some subtle issues with existing security proofs for Fischlin’s transform (Theorem 3.24).

Structure of this chapter. We start by defining Sigma protocols and their properties in Section 3.1. In Section 3.2, we discuss how to instantiate Sigma protocols. Then in Section 3.3, we define the properties we require of ZKPoK in this thesis. We then show in Section 3.4 how to upgrade a Sigma protocol to a ZKPoK. Finally, in Section 3.6, we summarize how to instantiate ZKPoK using this chapter, and establish notation.

3.1 Sigma protocols

Sigma protocols [Cra97] are easy to describe proof systems, and they encode some of the *basic* ideas for simulation and extraction. They can be viewed as an easy-to-construct basic form of a proof system, which we enhance later in Section 3.4 to make them secure in practice. In this section, we describe Sigma protocols and their properties.

3.1.1 Syntax of Sigma protocols

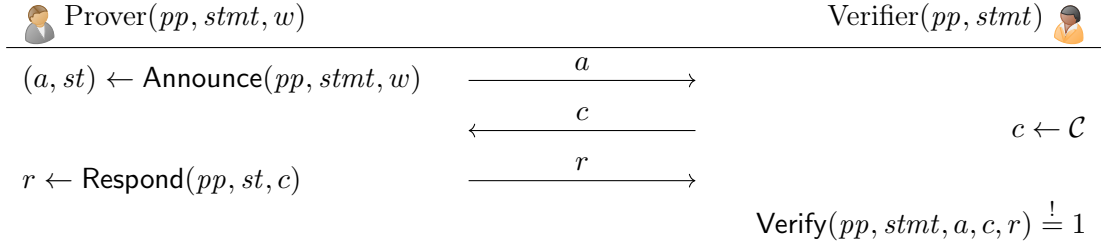


Figure 3.1: Schematic view of a Sigma protocol

A Sigma protocol is a three-message protocol where first, the prover sends an *announcement* a , then the verifier sends a random *challenge* c (independent of a), and the prover responds to that challenge with some *response* r . Then the verifier checks the transcript (a, c, r) . See Figure 3.1 for a schematic view. Formally, we define Sigma protocols as follows.

Definition 3.1 (Sigma protocol). A *Sigma protocol* Π_{Sigma} for relation $\mathcal{R}_{\mathcal{G}}$ consists of the following algorithms/sets:

Setup($\mathcal{G}$) $\rightarrow pp$

A ppt algorithm, generates public parameters pp .

Announce($pp, stmt, w$) $\rightarrow (a, st)$

A ppt algorithm that takes a statement $stmt$ and a witness w as input and computes an announcement a and a state st . ! The announcement a is the first message sent by the prover to the verifier.

$\mathcal{C} = \mathcal{C}_{\mathcal{G}}$

A finite set of *challenges* $c \in \mathcal{C}_{\mathcal{G}}$ that depends on $\mathcal{G}$. ! The verifier sends a uniformly random challenge $c \leftarrow \mathcal{C}_{\mathcal{G}}$ to the prover after receiving the announcement a .

Respond(pp, st, c) $\rightarrow r$

A ppt algorithm that generates a *response* r . ! The prover responds to challenge c by sending r to the verifier.

Verify($pp, stmt, a, c, r$) $= b$

A deterministic polynomial-time algorithm that outputs a bit b . ! Run by the

verifier, where $b = 1$ means that the verifier is convinced.

A Sigma protocol is *complete*, i.e., for all $\lambda \in \mathbb{N}, \mathcal{G} \in [\text{GenSetting}(1^\lambda)], pp \in [\text{Setup}(\mathcal{G})]$ and all $(stmt, w) \in \mathcal{R}_{\mathcal{G}}$,

$$\Pr \left[\text{Verify}(pp, stmt, a, c, r) = 1 \mid \begin{array}{l} (a, st) \leftarrow \text{Announce}(pp, stmt, w), \\ c \leftarrow \mathcal{C}_{\mathcal{G}}, \\ r \leftarrow \text{Respond}(pp, st, c) \end{array} \right] = 1$$

We say that a transcript is *accepting* if `Verify` outputs 1 for it (assuming that pp and $stmt$ are clear from the context). We will often denote concrete Sigma protocols in the style of Figure 3.1 instead of giving the description of each of the algorithms explicitly.

Because Sigma protocols are a stepping stone towards zero-knowledge proofs of knowledge, they have properties that approximate those notions, namely special honest-verifier zero-knowledge (a weak version of zero-knowledge) and special soundness (a strong property that is related to proofs of knowledge).

3.1.2 Special honest-verifier zero-knowledge

We start with special honest-verifier zero-knowledge. As the name suggests, this models some form of zero-knowledge for the case where the verifier is honest. For Sigma protocols specifically, honesty means that we assume the verifier chooses her challenge c uniformly at random from $\mathcal{C}$ (instead of, say, trying to choose c cleverly depending on $stmt, a$ to maybe learn “more”). This notion by itself is not directly useful in practice because at its core, zero-knowledge is supposed to protect the prover from *adversarial* verifiers trying to learn something about his witness. There is no reason to assume that an adversarial verifier will adhere to the protocol. However, we will later (Section 3.4) upgrade this property to construct proper zero-knowledge proofs.

As discussed at the beginning of Chapter 3, zero-knowledge is modeled using a simulator that can generate convincing transcripts without a witness. In the case of special honest-verifier zero-knowledge Sigma protocols, this means that we need to generate transcripts that look like the transcripts between an honest prover and an *honest* verifier. The *special* part means that this simulator can even generate convincing transcripts *given* any challenge c (as opposed to letting the simulator choose c) and requiring it to output transcripts with the right distribution conditioned on c . To generate a completely random transcript, one can of course first choose $c \leftarrow \mathcal{C}$ and then run the special simulator on input c .

Definition 3.2 (Special honest-verifier zero-knowledge). A Sigma protocol Π_{Sigma} for relation $\mathcal{R}_{\mathcal{G}}$ is *special honest-verifier zero-knowledge* (HVZK) if there exists a ppt algorithm $\mathcal{S}$ (the *simulator*) such that for all $\lambda \in \mathbb{N}, \mathcal{G} \in [\text{GenSetting}(1^\lambda)], pp \in [\text{Setup}(\mathcal{G})]$, all $(stmt, w) \in \mathcal{R}_{\mathcal{G}}$ and all challenges $c \in \mathcal{C}_{\mathcal{G}}$, the following distributions are the same:

- (a, c, r) where $(a, st) \leftarrow \text{Announce}(pp, stmt, w), r \leftarrow \text{Respond}(pp, st, c)$

- $(a, c, r) \leftarrow \mathcal{S}(pp, stmt, c)$

Crucially, $\mathcal{S}$ does not get the witness w as input. As a consequence, $\mathcal{S}$ obviously cannot run **Announce** and **Respond** to generate its transcript. So how can $\mathcal{S}$ possibly work? The trick here will be in the order of computation of a , c , and r : While the regular prover is required to first commit to an a , then gets a random challenge c , and finally he has to compute a fitting response r , the simulator $\mathcal{S}$ *first* gets to see the challenge c and *then* can prepare fitting a, r for it. As it turns out, Sigma protocols will be set up in a way that the order of operations that the prover has to go through is hard to do without a witness, but the order of operations for the simulator is always easy, even without a witness. We will see this trick in action in Construction 3.7. When constructing proper zero-knowledge proofs, we will exploit special honest-verifier zero-knowledge by also bringing the proper simulator into a position where it gets to see/specify c before having to commit to an announcement a . In that position, the proper simulator can simply run the honest-verifier zero-knowledge simulator for the Sigma protocol.

3.1.3 (Strong) special soundness and rewinding

Next, we have the special soundness property, which will enable proofs of knowledge from Sigma protocols. In short, the special soundness property states that given two accepting transcripts $(a, c, r), (a, c', r')$ with the same announcement a but different challenges $c \neq c'$, we can efficiently compute a valid witness w . For the formal definition, we slightly weaken this notion to *computational* special soundness, meaning that a witness does not have to be efficiently computable from *any* such pair of transcripts, but it should be computationally infeasible to find a pair of transcripts where witness computation is not possible.

Definition 3.3 ((Computational) special soundness). A Sigma protocol Π_{Sigma} for relation $\mathcal{R}_{\mathcal{G}}$ has *special soundness* if there exists a deterministic polynomial-time algorithm $\mathcal{E}$ (the *extractor*) such that for all ppt $\mathcal{A}$,

$$\Pr \left[\begin{array}{l} \text{Verify}(pp, stmt, a, c, r) = 1 \\ \wedge \text{Verify}(pp, stmt, a, c', r') = 1 \\ \wedge c, c' \in \mathcal{C} \wedge c \neq c' \\ \wedge (stmt, w) \notin \mathcal{R}_{\mathcal{G}} \end{array} \right] \parallel \left[\begin{array}{l} \mathcal{G} \leftarrow \text{GenSetting}(1^\lambda), pp \leftarrow \text{Setup}(\mathcal{G}), \\ (stmt, a, (c, r), (c', r')) \leftarrow \mathcal{A}(pp), \\ w = \mathcal{E}(pp, stmt, a, (c, r), (c', r')) \end{array} \right]$$

is negligible.

In Section 3.4, we will see how to get a proper proof of knowledge property from special soundness. The basic idea is, of course, to somehow extract two accepting transcripts with the same announcement but different challenges from a successful prover. Then, we can apply special soundness to compute the witness. There are several ways to get those two transcripts. We are going to use the Fischlin transform way of obtaining them (see Section 3.4), but the most illustrative and historically influential way of obtaining the two transcripts is the idea of *rewinding*.

The intuition of extraction by rewinding is as follows. Suppose we have a prover machine $\mathcal{P}$ that has just successfully run a Sigma protocol proving that it knows w such that $(stmt, w) \in \mathcal{R}$. We save the transcript (a, c, r) . To get a second transcript, we (in our head), run a copy of $\mathcal{P}$ with the same input and same randomness. This will cause that copy of $\mathcal{P}$ to send the exact same announcement a again. However, we now feed a different challenge $c' \neq c$ to our copy of $\mathcal{P}$, which then answers with some response r' . This is the second transcript (a, c', r') . Intuitively, a good prover $\mathcal{P}$ needs to be able to answer many challenges correctly (otherwise, our scenario, where $\mathcal{P}$ has already won the protocol once, becomes unlikely), so the second transcript should also have a decent chance to be accepting, allowing us to apply our special soundness extractor to compute a witness. This intuition is formalized in the famous forking lemma [PS96; BN06]. The terminology “rewinding” and “forking” comes from the view that the process described above is the same as taking a snapshot of $\mathcal{P}$ just before we send it its challenge, and then *rewinding* it back to that point, effectively creating a *fork* of its execution on which we send it a different challenge. We discuss issues with this approach in Section 3.5.

In practice, the usefulness of the special soundness property depends crucially on the size of the challenge space $\mathcal{C}$. Note that a trivial way for a prover to attack a Sigma protocol *without* a witness w is by generating an accepting transcript (a, c, r) using the honest-verifier zero-knowledge property (Definition 3.2), sending a , and hoping that the verifier will randomly choose the same challenge c as in the simulated transcript. If the verifier does choose this challenge, the attacker can send the valid response r , convincing the verifier without ever touching a witness. If the verifier chooses a different challenge $c' \neq c$, the attacker, intuitively, needs to *know* the witness to compute a response r' , or break the special soundness property (Definition 3.3). So essentially, an adversary can always “win” the protocol without any knowledge of a witness with probability $1/|\mathcal{C}|$ (often called the “knowledge error”) by correctly guessing what challenge the verifier will send. As a consequence, large challenge spaces are quite useful. To model “large” challenge spaces, we define *super-poly challenge spaces*.

Definition 3.4 (Super-poly challenge space). A Sigma protocol Π_{Sigma} has *super-poly challenge space* if there is a super-poly function f such that $|\mathcal{C}_{\mathcal{G}}| \geq f(\lambda)$ for all λ and $\mathcal{G} \in [\text{GenSetting}(1^\lambda)]$.

A Sigma protocol without special honest-verifier zero-knowledge or without special soundness is trivial to construct: Without HVZK, the prover could just send the witness in plain; without (computational) special soundness, the verifier could just accept *any* transcript without checking anything. For this reason, we will never consider Sigma protocols without those two properties.

A less crucial, but useful property, which not all relevant Sigma protocols have, is the *unique responses* property, which says that for any announcement-challenge pair (a, c) , there is only a single response r that would make the verifier accept. For us, it will be sufficient to consider *quasi-unique responses*, i.e., while a second response may exist, it is computationally hard to find it.

Definition 3.5. A Sigma protocol Π_{Sigma} has *quasi-unique responses* if for all ppt $\mathcal{A}$,

$$\Pr \left[\begin{array}{l} \text{Verify}(pp, stmt, a, c, r) = 1 \\ \wedge \text{Verify}(pp, stmt, a, c, r') = 1 \\ \wedge r \neq r' \end{array} \middle| \begin{array}{l} \mathcal{G} \leftarrow \text{GenSetting}(1^\lambda), \\ pp \leftarrow \text{Setup}(\mathcal{G}), \\ (stmt, a, c, r, r') \leftarrow \mathcal{A}(\mathcal{G}, pp) \end{array} \right]$$

is negligible. If the probability is 0 for all (arbitrary runtime) $\mathcal{A}$, then we say that Π_{Sigma} has *unique responses*.

Generalizing quasi-unique responses, some protocols have a *strong* special soundness property, which is a stronger version of special soundness (Definition 3.3). In Definition 3.3, we required that it is hard to find two transcripts $(a, c, r) \neq (a, c', r')$ with $c \neq c'$, such that the extractor is not able to compute a witness from those transcripts. *Strong* special soundness will additionally require that it is also hard to find transcripts $(a, c, r) \neq (a, c, r')$ (same challenge c) for which the extractor cannot compute a witness. This property, similar to the one proposed in [KS22], will be useful in Section 3.4 because randomized Fischlin's transform requires it.

Definition 3.6 (Strong special soundness). A Sigma protocol Π_{Sigma} for relation $\mathcal{R}_{\mathcal{G}}$ has *strong special soundness* if there exists a deterministic polynomial-time algorithm $\mathcal{E}$ (the *extractor*) such that for all ppt $\mathcal{A}$,

$$\Pr \left[\begin{array}{l} \text{Verify}(pp, stmt, a, c, r) = 1 \\ \wedge \text{Verify}(pp, stmt, a, c', r') = 1 \\ \wedge c, c' \in \mathcal{C} \wedge (c, r) \neq (c', r') \\ \wedge (stmt, w) \notin \mathcal{R}_{\mathcal{G}} \end{array} \middle| \begin{array}{l} \mathcal{G} \leftarrow \text{GenSetting}(1^\lambda), pp \leftarrow \text{Setup}(\mathcal{G}), \\ (stmt, a, (c, r), (c', r')) \leftarrow \mathcal{A}(\mathcal{G}, pp), \\ w = \mathcal{E}(pp, stmt, a, (c, r), (c', r')) \end{array} \right]$$

is negligible.

Strong special soundness generalizes quasi-unique responses (Definition 3.5): Any protocol that has quasi-unique responses (Definition 3.5) and regular special soundness (Definition 3.3) also has *strong* special soundness simply using the regular special soundness extractor. This is because the quasi-unique responses property makes the case $c = c' \wedge r \neq r'$ unlikely, which leaves $c \neq c'$ as the only realistic winning condition for $\mathcal{A}$. But if the two transcripts have different challenges, we can apply the standard special-soundness extractor. There *are* protocols that do not have quasi-unique responses but do have strong special soundness, i.e., quasi-unique responses are not the only way to fulfill this definition. An example for this are OR proofs (Construction 3.17) over protocols with unique responses.

3.2 Sigma protocol constructions: The Schnorr proof ecosystem

In this section, we show how to construct the Sigma protocols relevant for this thesis. We start with the basic generalized Schnorr protocol (Construction 3.7), which will be

the basis for all future protocols. Then we build upon that in several steps, establishing a framework (Construction 3.12 and Section 3.2.5) that will make it easy to implement future proof statements, and demonstrating how to add the quasi-unique responses property to Schnorr protocols (Construction 3.15).

3.2.1 The basic generalized Schnorr protocol

The generalized Schnorr protocol is a Sigma protocol for proving knowledge of a homomorphism preimage. We will simply call this protocol the (generalized) *Schnorr protocol*, given that Schnorr [Sch89] devised the general idea of these protocols, even though the generalization we are interested in can be more accurately credited to Maurer [Mau09; Mau15].

Construction 3.7 (Generalized Schnorr protocol). Let $(\text{Hom}_{\mathcal{G}})_{\mathcal{G}}$ be a family of sets of group homomorphisms $\varphi : G \rightarrow H$ such that for all $\varphi \in \text{Hom}_{(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, e, p)}$, the groups G, H are finite and every element in H has order p or 1. Membership in Hom must be computationally easy to verify.

Let $\mathcal{R}_{\mathcal{G}} = \{(\text{stmt} = (\varphi, y), w) \mid \varphi \in \text{Hom}_{\mathcal{G}} \wedge \varphi(w) = y\}$. The generalized Schnorr proof Π_{Schnorr} is a Sigma protocol for $\mathcal{R}_{\mathcal{G}}$ and it is defined as in Figure 3.2, with $\text{Setup}(\mathcal{G}) := \mathcal{G}$.

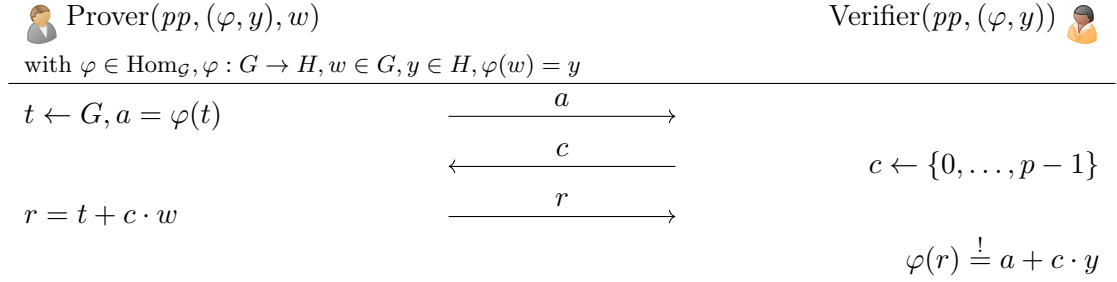


Figure 3.2: The generalized Schnorr protocol

It is easy to see that an honest prover will convince an honest verifier with probability 1 because $\varphi(r) = \varphi(t + c \cdot w) = \varphi(t) + c \cdot \varphi(w) = a + c \cdot y$.

An important (and typical) example for the class $\text{Hom}_{(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, e, p)}$ is the set of homomorphisms $\varphi : \mathbb{Z}_p^n \times \mathbb{G}_1^{m_1} \times \mathbb{G}_2^{m_2} \times \mathbb{G}_t^{m_3} \rightarrow \mathbb{Z}_p^{n'} \times \mathbb{G}_1^{m'_1} \times \mathbb{G}_2^{m'_2} \times \mathbb{G}_t^{m'_3}$ that are “easily recognized as homomorphisms”. We do not intend to formally define this property, but it is easy (if tedious) to come up with a criterion that covers all the homomorphisms φ we use in this thesis.

In the literature, the homomorphism φ is usually a hardcoded part of the relation $\mathcal{R}$ instead of part of the statement stmt . In that formalization, the Schnorr protocol is actually a family of protocols, one for every φ . This has the disadvantage that statements cannot be dynamically set, but rather need to be determined at design time (i.e., the protocol designer has to choose the relation $\mathcal{R}$ for φ and then implement the right Schnorr protocol for that relation). In our formalization, there is one implementation for *the*

Schnorr protocol (and for all other protocols in this thesis that build on it), and that implementation is able to handle different statements (different φ) dynamically, i.e., it takes φ as input at runtime. This will allow users, for example, to dynamically decide what information to disclose in an anonymous credential instead of having the protocol designer fix that at design time.

The Schnorr protocol covers many interesting statements. We get the well-known original Schnorr protocol [Sch89] for proving knowledge of the discrete logarithm of $y \in \mathbb{G}_1$ to base g using $\varphi : \mathbb{Z}_p \rightarrow \mathbb{G}_1$ with $\varphi(w) = g^w$. We can prove that we know the message $m \in \mathbb{G}_1$ of an ElGamal ciphertext $y = (m \cdot pk^r, g^r)$ with $\varphi : \mathbb{Z}_p \times \mathbb{G}_1 \rightarrow \mathbb{G}_1^2$, $\varphi(r, m) = (m \cdot pk^r, g^r)$. We can prove that we know a valid weakly-secure Boneh-Boyen signature [BB08] $\sigma = g^{1/(sk+m)}$ on a public message m using $\varphi : \mathbb{G}_1 \rightarrow \mathbb{G}_t$ with $\varphi(\sigma) = e(\sigma, pk \cdot h^m)$ and $y = e(g, h)$.

Because Schnorr protocols are so pervasive, we establish a convenient notation for them.

Definition 3.8 (Camenisch-Stadler notation [CS97] for Schnorr proofs).

$$\text{Schnorr} [(w); \varphi(w) = y]$$

denotes (the execution of) a Schnorr protocol with common input $stmt = (\varphi, y)$ and secret input w . Public parameters are assumed to be available from the context.

We use the notation in the following form. For example, for an ElGamal ciphertext (C_0, C_1) , we may write something like

$$\text{Schnorr} [(r, m); m \cdot pk^r = C_0 \wedge g^r = C_1],$$

which corresponds to an execution of the Schnorr protocol with $\varphi(r, m) = (m \cdot pk^r, g^r)$ and $y = (C_0, C_1)$. Generally, the expression before the semicolon is the list of witnesses. The expression after the semicolon is a statement that can be expressed (potentially with a little bit of rewriting) as a homomorphism-image-pair statement $stmt = (\varphi, y)$.

In the following, we establish that the generalized Schnorr protocol has all the required Sigma protocol properties.

Theorem 3.9. *The (generalized) Schnorr protocol is honest-verifier zero-knowledge (Definition 3.2).*

Proof. We define $\mathcal{S}(pp, (\varphi, y), c)$ as follows:

- $r \leftarrow G$
- $a := \varphi(r) - c \cdot y$
- Output (a, c, r) .

Let $pp = \mathcal{G} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, e, p)$, $((\varphi, y), w) \in \mathcal{R}_{\mathcal{G}}$, and $c \in \{0, \dots, p-1\}$. Looking at the “honest” transcript distribution (a, c, r) , where $t \leftarrow G, a = \varphi(t), r = t + c \cdot w$, we

can see that r is distributed uniformly at random in G (because t is uniformly random). Furthermore, we know that honest transcripts are accepting, hence $a = \varphi(r) - c \cdot y$. Overall, this means that the honest distribution is the same as the one output by $\mathcal{S}$. $\square$

Theorem 3.10. *The (generalized) Schnorr protocol has special soundness (see Definition 3.3).*

Proof. We define the extractor $\mathcal{E}(pp, (\varphi, y), a, (c, r), (c', r'))$ to output $w = c^* \cdot (r - r')$ where $c^* = (c - c')^{-1} \bmod p$ (the inverse exists because $0 < |c - c'| < p$ by definition of the challenge space $\mathcal{C} = \{0, \dots, p-1\}$ and $c \neq c'$). If the input transcripts are accepting, then w is a valid witness: $\varphi(w) = \varphi(c^* \cdot (r - r')) = c^* \cdot (\varphi(r) - \varphi(r')) = c^* \cdot (c \cdot y - c' \cdot y) = c^* \cdot (c - c') \cdot y = y$, where the third equality follows from the Verify check $\varphi(r) \stackrel{!}{=} a + c \cdot y$ and the last equality uses that every element y in H has order p or 1 as defined in Construction 3.7. $\square$

Theorem 3.11. *The (generalized) Schnorr protocol has super-poly challenge space.*

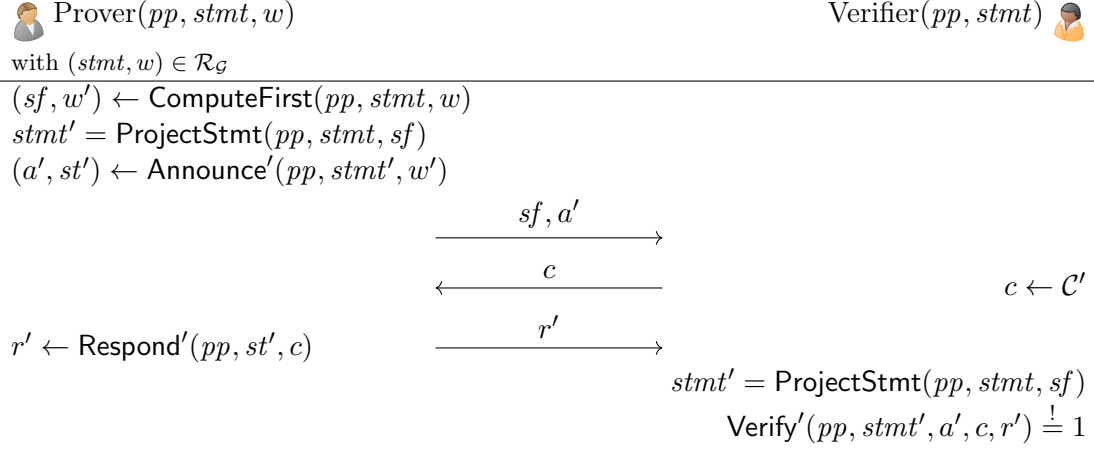
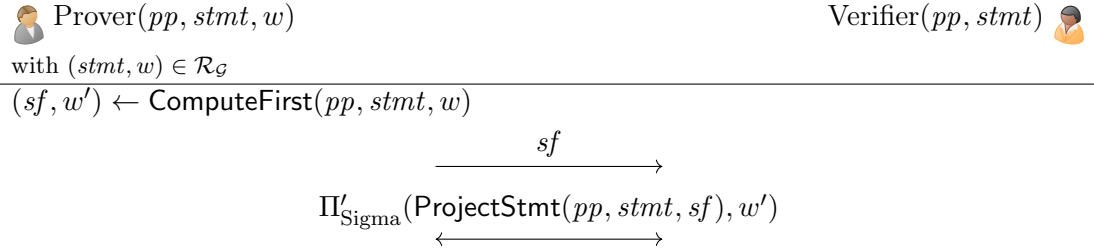
Proof. The size of the challenge space is $p > 2^\lambda$ (cf. Definition 2.3). $\square$

3.2.2 “Send-first” protocols

A number of important protocols send an auxiliary value sf and then run another Sigma protocol on some statement $stmt'$ that depends on sf . We call such protocols “send-first” protocols and sf the “send-first value”. For example, proving knowledge of a signature often works as follows (e.g., Construction 5.13): (1) The prover randomizes the signature and sends the randomized signature to the verifier, then (2) the prover proves with a Schnorr protocol that he knows how to derandomize the randomized signature such that it passes the verification check. The big advantage is that the auxiliary value (e.g., the randomized signature) can be regarded as *a constant* for the inner protocol, which makes it easier (or even feasible in the first place) to phrase the inner protocol as a homomorphism preimage problem (as required for Schnorr protocols). The same trick is also used for set membership and range proofs [CCS08] and many other applications, see Section 3.2.4.

If done correctly, such protocols are still (secure) Sigma protocols, which makes them very convenient to use (e.g., compile them to a secure proof with Section 3.4). Note that in the description above, the prover first sends sf and then sends the announcement a' of the inner protocol. These two messages can be sent simultaneously, so the result still has only three rounds (cf. Figure 3.3).

In the following, we model such protocols formally, then we prove desired properties.

**Figure 3.3:** Schematic view of a send-first protocol Π_{sf} **Figure 3.4:** Succinct syntax for a send-first protocol Π_{sf} , where $\Pi'_{\text{Sigma}}(\text{ProjectStmt}(pp, stmt, sf), w')$ denotes an execution of Π'_{Sigma} with the statement $stmt' = \text{ProjectStmt}(pp, stmt, sf)$ and witness w' . Equivalent to Figure 3.3

Construction 3.12 (Send-first protocol). Let $\mathcal{R}$ be a relation. Let $\Pi'_{\text{Sigma}} = (\text{Setup}', \text{Announce}', \text{Respond}', \text{Verify}')$ be a Sigma protocol for relation $\mathcal{R}'$ with challenge space $\mathcal{C}'$. The *send-first* protocol $\Pi_{\text{sf}}[\Pi'_{\text{Sigma}}, \text{SetupFirst}, \text{ComputeFirst}, \text{ProjectStmt}]$ is a Sigma protocol for $\mathcal{R}$ that works as in Figures 3.3 and 3.4, with $\Pi_{\text{sf}}.\text{Setup} = \text{Setup}' \times \text{SetupFirst}$.

We require that **SetupFirst** and **ComputeFirst** are ppt algorithms and **ProjectStmt** is a deterministic polynomial-time algorithm. Also, for all $\lambda \in \mathbb{N}$, $\mathcal{G} \in [\text{GenSetting}(1^\lambda)]$, $pp \in [\Pi_{\text{sf}}.\text{Setup}(\mathcal{G})]$, and all $(sf, w') \in [\text{ComputeFirst}(stmt, w)]$,

$$(stmt, w) \in \mathcal{R} \Rightarrow (\text{ProjectStmt}(pp, stmt, sf), w') \in \mathcal{R}'.$$

$\Pi_{\text{sf}}[\Pi'_{\text{Sigma}}, \text{SetupFirst}, \text{ComputeFirst}, \text{ProjectStmt}]$ is a Sigma protocol for relation $\mathcal{R}$.

We use the notation $\text{Setup}' \times \text{SetupFirst}$ to denote the algorithm that on input $\mathcal{G}$ runs both $pp_0 \leftarrow \text{Setup}'(\mathcal{G})$ and $pp_1 \leftarrow \text{SetupFirst}(\mathcal{G})$, and outputs (pp_0, pp_1) .

To make sure that the resulting send-first protocol can be simulated, we need to be able to simulate the send-first value sf without a witness using some method **SimFirst**. Given that, the resulting protocol is special honest-verifier zero-knowledge:

Theorem 3.13 (Send-first is special HVZK). *If the inner protocol Π'_{Sigma} is special honest-verifier zero-knowledge and there is a ppt algorithm SimFirst such that for all $\lambda \in \mathbb{N}, \mathcal{G} \in \text{GenSetting}(1^\lambda), pp \in [\Pi_{\text{sf}}.\text{Setup}(\mathcal{G})]$ and all $(\text{stmt}, w) \in \mathcal{R}_{\mathcal{G}}$, the value sf is distributed the same in the following two distributions:*

- $(\text{sf}, w') \leftarrow \text{ComputeFirst}(pp, \text{stmt}, w)$
- $\text{sf} \leftarrow \text{SimFirst}(pp, \text{stmt})$

Then Π_{sf} is special honest-verifier zero-knowledge (Definition 3.2).

Proof. Let $\mathcal{S}'$ be a simulator for Π'_{Sigma} . The simulator $\mathcal{S}(pp, \text{stmt}, c)$ for Π_{sf} works as follows:

- It simulates a send-first value $\text{sf} \leftarrow \text{SimFirst}(pp, \text{stmt})$.
- It computes the inner protocol statement $\text{stmt}' = \text{ProjectStmt}(pp, \text{stmt}, \text{sf})$.
- It simulates the inner protocol transcript $(a', c, r') \leftarrow \mathcal{S}'(pp, \text{stmt}', c)$.

It sets $a = (\text{sf}, a')$ and $r = r'$ and outputs (a, c, r) . It is easy to see that this simulator fulfills the requirements of Definition 3.2. $\square$

To extract from a send-first proof, we need to be able to convert or interpret a witness for the inner protocol as a witness for the outer send-first protocol using some method InterpretWitness . For example, sf is a randomized signature and the inner protocol proves knowledge of derandomization values that make the randomized signature valid. In this case, the outer witness can be computed by applying the derandomization values (i.e., the inner protocol's witness) to the randomized signature. Given a reliable way to reinterpret the inner witness to get an outer witness, the send-first protocol inherits its soundness guarantees from the inner Sigma protocol:

Theorem 3.14 (Send-first soundness). *If the inner protocol Π'_{Sigma} has (computational) special soundness and there is a deterministic polynomial-time algorithm InterpretWitness such that for all ppt $\mathcal{A}$,*

$$\Pr \left[\begin{array}{l} (\text{stmt}', w') \in \mathcal{R}' \\ \wedge (\text{stmt}, w) \notin \mathcal{R} \end{array} \middle| \begin{array}{l} \mathcal{G} \leftarrow \text{GenSetting}(1^\lambda), pp \leftarrow \Pi_{\text{sf}}.\text{Setup}(\mathcal{G}) \\ (\text{stmt}, \text{sf}, w') \leftarrow \mathcal{A}(\mathcal{G}, pp) \\ \text{stmt}' = \text{ProjectStmt}(pp, \text{stmt}, \text{sf}) \\ w = \text{InterpretWitness}(pp, \text{stmt}, \text{sf}, w') \end{array} \right]$$

is negligible, then the send-first protocol Π_{sf} has computational special soundness (Definition 3.3). If Π'_{Sigma} additionally has strong special-soundness, then Π_{sf} also has strong special-soundness.

Proof. Let $\mathcal{E}'$ be a (strong) special soundness extractor for the inner protocol Π'_{Sigma} . We construct $\mathcal{E}(pp, stmt, (sf, a'), (c_0, r'_0), (c_1, r'_1))$ for Π_{sf} as follows.

- $\mathcal{E}$ first computes the inner protocol statement $stmt' = \text{ProjectStmt}(pp, stmt, sf)$.
- It then extracts the inner witness as $w' = \mathcal{E}'(pp, stmt', a', (c_0, r'_0), (c_1, r'_1))$.
- It then computes and outputs the witness $w = \text{InterpretWitness}(pp, stmt, sf, w')$ for the outer protocol.

It is easy to see that if $\mathcal{E}'$ fulfills the (strong) special-soundness property for Π'_{Sigma} and $\mathcal{R}'$, then the constructed $\mathcal{E}$ fulfills (strong) special-soundness for Π_{Sigma} and $\mathcal{R}$: If w' is not a valid witness for the inner protocol (i.e., $(stmt', w') \notin \mathcal{R}'$), then $(stmt', a', (c_0, r'_0), (c_1, r'_1))$ serves as a counterexample to the (strong) special-soundness property for Π'_{Sigma} . If w' is valid but w is not a valid witness for the outer protocol (i.e., $(stmt, w) \notin \mathcal{R}$), then $(stmt, sf, w')$ serves as a counterexample to the property required from Π_{sf} in the theorem statement. $\square$

Overall, send-first protocols are easy to prove secure: One just needs to show how to simulate the send-first value and how to interpret the inner witness to get the outer witness. Because they are Sigma protocols, they are easy to work with and easy to convert to a secure protocol (with Section 3.4).

3.2.3 Strongly-sound generalized Schnorr proofs with quasi-unique responses

The Schnorr protocol does not generally have quasi-unique responses. This is easy to see for φ where it is easy to find two witnesses $w \neq w'$ for the same image y , e.g., $\varphi : \mathbb{Z}_p^2 \rightarrow \mathbb{Z}_p, \varphi(w) = \varphi(w_0, w_1) = w_0 + w_1$ with $w = (y, 0)$ and $w' = (0, y)$. Even in an honest execution, both $r = t + c \cdot w$ and $r' = t + c \cdot w'$ are valid responses. So $((\varphi, y), \varphi(t), c, t + c \cdot w, t + c \cdot w')$ for $w \neq w'$ breaks the quasi-unique responses property (Definition 3.5). This is unfortunate since some of the more important ways of transforming Sigma protocols to fully-secure proof systems require the Sigma protocol to have strong soundness.

If φ has a collision-resistance-like property (Definition 2.12), then the overall protocol has quasi-unique responses (and hence, strong soundness). This is because if an adversary finds two accepting responses $r \neq r'$ for the same announcement a and challenge c , then the acceptance equations $\varphi(r) \stackrel{!}{=} a + c \cdot y$ and $\varphi(r') \stackrel{!}{=} a + c \cdot y$ guarantee that $\varphi(r) = \varphi(r')$, so a collision for φ has been found.

We exploit this property to present a general adapted Schnorr protocol with quasi-unique responses. For this, the prover first Pedersen-commits (Construction 2.19) to his witnesses and then runs a standard Schnorr proof with a homomorphism φ' , ensuring that the witnesses are a valid φ preimage *and* they have been correctly committed to (see Figure 3.5). As established in Section 3.2.2, such a protocol can be conveniently phrased as a send-first protocol. In this protocol, the Pedersen commitment is part of the inner (standard) Schnorr proof's homomorphism image $y' = \varphi'(w')$. Because of the

commitment's strong binding property (Definition 2.16 and Theorem 2.20), this makes it computationally infeasible to find two different preimages for φ' , giving us quasi-unique responses.

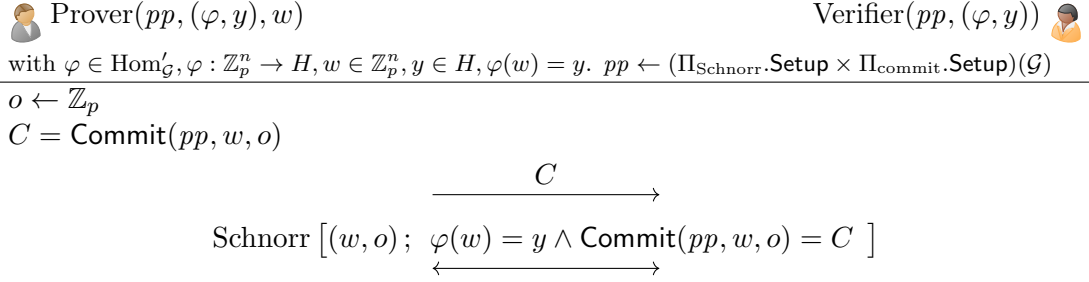


Figure 3.5: The generalized Schnorr protocol (for $\mathbb{Z}_p^n$ preimages) augmented to provide quasi-unique responses.

Construction 3.15 (Quasi-unique responses Schnorr proofs). Let $(\text{Hom}_{\mathcal{G}})_{\mathcal{G}}$ be a family of sets of group homomorphisms with the requirements as in Construction 3.7. Let $n \in \mathbb{N}$. Let $\Pi_{\text{commit}} = (\text{Setup}, \text{Commit})$ be the Pedersen commitment scheme for $n + 1$ values as in Construction 2.19.

The quasi-unique responses Schnorr protocol is the send-first protocol (Construction 3.12) as in Figure 3.5, i.e., with $\Pi'_{\text{Sigma}} = \Pi_{\text{Schnorr}}$ and

SetupFirst = $\Pi_{\text{commit}}.\text{Setup}$
Simply executes the Π_{commit} setup.

ComputeFirst($\text{pp}, \text{stmt}, w$)
Chooses a random $o \leftarrow \mathbb{Z}_p$ and outputs $\text{sf} = \text{Commit}(\text{pp}, w, o)$ and $w' = (w, o)$.

ProjectStmt($\text{pp}, \text{stmt}, \text{sf}$)
For $\text{stmt} = (\varphi, y)$ returns $\text{stmt}' = (\varphi', (y, \text{sf}))$, where $\varphi' : \mathbb{Z}_p^{n+1} \rightarrow H \times \mathbb{G}_1$ with $\varphi'(w, o) = (\varphi(w), \text{Commit}(\text{pp}, w, o))$.

It is a Sigma protocol for relation $\mathcal{R}_{\mathcal{G}} = \{(\text{stmt} = (\varphi, y), w) \mid \varphi \in \text{Hom}'_{\mathcal{G}} \wedge \varphi(w) = y\}$, where $\text{Hom}'_{\mathcal{G}} = \{\varphi : \mathbb{Z}_p^n \rightarrow H \mid \varphi' \in \text{Hom}_{\mathcal{G}}\}$.

The overhead of Construction 3.15 over the standard Schnorr protocol for φ (Construction 3.7) is modest: two $\mathbb{G}_1$ elements in the announcement (C and one from the $\text{Commit}(\dots) = C$ statement in the inner Schnorr protocol) and one $\mathbb{Z}_p$ element in the response (the blinded o). The prover has to compute those overhead values (two multi-exponentiations in $\mathbb{G}_1$ and one computation in $\mathbb{Z}_p$), the verifier only has to do one additional multi-exponentiation. The setup of the protocol has to be done honestly, but since the commitment scheme's pp only consists of uniformly random group elements (cf. Construction 2.19), the honest setup can be replaced by computing the group elements as hashes of some well-known constants in the random oracle model. We only define

this protocol for witnesses in $\mathbb{Z}_p^n$, but the general approach of committing to witnesses is likely possible for group element witnesses, too.

The following theorem establishes that the quasi-unique responses Schnorr protocol indeed has quasi-unique responses (under the discrete logarithm assumption, which ensures that the Pedersen commitments are computationally binding).

Theorem 3.16 (Quasi-unique responses Schnorr proofs security). *Construction 3.15 is special honest-verifier zero-knowledge (Definition 3.2), and has special soundness (Definition 3.3). Under the discrete logarithm assumption (Definition 2.6), it has quasi-unique responses (Definition 3.5).*

In particular, Construction 3.15 has *strong* special soundness under the discrete logarithm assumption.

In the following, we sketch the proof of security for Construction 3.15. For this, we can make use of Theorems 3.13 and 3.14 for send-first protocols. To prove the quasi-unique responses property, we can follow the intuition mentioned above: If an attacker can find two accepting responses $r'_1 \neq r'_2$ for the inner Schnorr protocol, then $\varphi'(r'_1) = \varphi'(r'_2)$ and in particular, we will have found two different openings for the commitment C .

Proof. The inner Schnorr protocol (Construction 3.7) is special honest-verifier zero-knowledge (Theorem 3.9). So with $\text{SimFirst}(pp, stmt)$ computing $sf \leftarrow \mathbb{G}_1$, we get from the HVZK property of the send-first approach (Theorem 3.13) that the overall protocol is special honest-verifier zero-knowledge.

The inner Schnorr protocol also has special soundness (Theorem 3.10). So with $\text{InterpretWitness}(pp, stmt, sf, w' = (w, o)) = w$, it follows easily with Theorem 3.14 that this send-first construction has the special soundness property, too.

It remains to prove that the resulting protocol has quasi-unique responses. We prove this by reduction to the binding property of Pedersen commitments. Let $\mathcal{A}$ be an attacker against quasi-unique responses of Construction 3.15. We construct $\mathcal{B}$ against the strong computational binding property of Pedersen commitments. $\mathcal{B}(\mathcal{G}, pp)$ runs $(stmt, (sf, a'), c, r'_1, r'_2) \leftarrow \mathcal{A}(\mathcal{G}, pp)$. $\mathcal{B}$ parses $sf = C$, $r'_1 = (m_1, o_1) \in \mathbb{Z}_p^n \times \mathbb{Z}_p$, and $r'_2 = (m_2, o_2) \in \mathbb{Z}_p^n \times \mathbb{Z}_p$. It outputs (m_1, o_1, m_2, o_2) .

If the transcripts are accepting and $r'_1 \neq r'_2$, then $\mathcal{B}$ wins the strong computational binding game: From the inner Schnorr protocol's acceptance condition, we get $\varphi'(r'_1) = a' + c \cdot (y, C) = \varphi'(r'_2)$. By definition, $\varphi'(x) = (\varphi(x), \text{Commit}(pp, x))$, so in particular, we get $\text{Commit}(pp, r'_1) = \text{Commit}(pp, r'_2)$.

So $\Pr[\mathcal{A} \text{ breaks quasi-unique responses}] \leq \Pr[\mathcal{B} \text{ breaks strong binding}]$. The latter probability is negligible under the discrete logarithm assumption, so $\mathcal{A}$'s win probability is also negligible. $\square$

3.2.4 Standard tricks of the Schnorr proof ecosystem

We briefly discuss some advanced statements that can be proven with Sigma protocols and the send-first idea (Construction 3.12). These are useful in the context of anonymous credential systems because they allow users to prove additional statements about

their attributes. We explain them in a standalone setting, but composition with larger statements is discussed in Section 3.2.5.

Products of witnesses. To prove the product relation $w_1 \cdot w_2 = w_3$ about witnesses $w_1, w_2, w_3 \in \mathbb{Z}_p$, one Pedersen-commits to w_1 as $C = g^o \cdot h^{w_1}$ and proves that one can open C^{w_2} to w_3 . This was observed by [Mau09]. In more detail, we sketch the following send-first protocol: The prover Pedersen-commits to w_1 as $C = g^o \cdot h^{w_1}$ for fresh $o \leftarrow \mathbb{Z}_p$, computes the additional witness $o' = o \cdot w_2$, sends $sf = C$ to the verifier, and proves

$$\text{Schnorr} \left[(w_1, w_2, w_3, o, o') ; C = g^o \cdot h^{w_1} \wedge C^{w_2} = g^{o'} \cdot h^{w_3} \right].$$

It is easy to see that this fulfills the required properties of a send-first protocol (Theorems 3.13 and 3.14): The send-first message C can be simulated by sending a random $C \leftarrow \mathbb{G}_1$. If an adversary manages to produce a witness w_1, w_2, w_3, o, o' where $w_3 \neq w_1 \cdot w_2$, then the commitment C^{w_2} can be opened to both $w_1 \cdot w_2$ (with opening $o \cdot w_2$, as per the first statement in the Schnorr proof), and to w_3 (with opening o' , as per the second statement in the Schnorr proof). Because Pedersen commitments are binding (Theorem 2.20) assuming discrete logarithms are hard, this case happens only with negligible probability.

Of course, this protocol by itself is not too interesting (clearly, it is trivial to come up with three numbers where one is the product of the other two), but the witnesses w_1, w_2, w_3 are available as Schnorr witnesses and additional statements about them can be proven, see Section 3.2.5.

As a sidenote, with this protocol, one can in principle prove the correct evaluation of any algebraic circuit (and hence *any* polynomial-time computation). However, this is exceedingly costly (unless the circuit is very small) and there are dedicated protocols that prove correct circuit evaluation much more efficiently.

Witness not zero. To prove that some witness $w \neq 0$ is not zero, the prover can simply prove knowledge of its inverse w^{-1} using the product trick above. In more detail, we sketch the following send-first protocol: The prover Pedersen-commits to w as $C = g^o \cdot h^w$ for fresh $o \leftarrow \mathbb{Z}_p$, computes the additional witnesses $w' = w^{-1}, o' = o \cdot w'$, sends $sf = C$ to the verifier, and proves

$$\text{Schnorr} \left[(w, w', o, o') ; C = g^o \cdot h^w \wedge C^{w'} = g^{o'} \cdot h^1 \right].$$

Two witnesses are different. To prove that two witnesses $w_1, w_2 \in \mathbb{Z}_p$ are not equal, one simply proves that $w_1 - w_2 \neq 0$ using the protocol above. In more detail, we sketch the following send-first protocol: The prover Pedersen-commits to $w_1 - w_2$ as $C = g^o \cdot h^{w_1 - w_2}$ for fresh $o \leftarrow \mathbb{Z}_p$, computes the additional witnesses $w' = (w_1 - w_2)^{-1}, o' = ow'$, sends $sf = C$ to the verifier, and proves

$$\text{Schnorr} \left[(w_1, w_2, w', o, o') ; C = g^o \cdot h^{w_1 - w_2} \wedge C^{w'} = g^{o'} \cdot h^1 \right].$$

Witness not equal to constant. To prove that some witness $w \in \mathbb{Z}_p$ is not equal to some public value t , one can use the approach above, replacing w_2 with a constant t .

Set membership. Sometimes, it is useful to efficiently prove statements of the form $w \in S$ for some witness $w \in \mathbb{Z}_p$ and public set $S \subset \mathbb{Z}_p$. Following the approach of [CCS08], assume that some party that the verifier trusts has computed and published a signature verification key pk and a signature σ_v on each value $v \in S$ (and *only* for those values). Concretely, these signatures are weakly secure Boneh-Boyen signatures [BB08]. Hence the public key is $pk = g_2^{sk}$ and the signatures are of the form $\sigma_v = g^{1/(sk+v)}$.

To prove that a witness w is part of the set S , the prover picks the signature σ_w from the public list of signatures and computes a blinded version $\sigma' = \sigma_w^r$ using $r \leftarrow \mathbb{Z}_p^*$. The prover sends σ' to the verifier (as the send-first message) and then proves that he can derandomize σ' to a valid signature on w :

$$\text{Schnorr} \left[(w, r); e(\sigma', pk \cdot g_2^w) = e(g_1, g_2)^r \wedge \sigma' \neq 1 \right].$$

Note that the $\sigma' \neq 1$ check can be done in plain by the verifier.

It is easy to see that this fulfills the required properties of a send-first protocol (Theorems 3.13 and 3.14): The send-first message σ' can be simulated by sending a random $\sigma' \leftarrow \mathbb{G}_1 \setminus \{1\}$. If an adversary manages to produce a witness w, r where $w \notin S$, then $(\sigma')^{1/r}$ is a valid signature forgery on w (note that $\sigma' \neq 1$ guarantees $r \neq 0$). Hence assuming unforgeability of weakly-secure Boneh-Boyen signatures, this case happens only with negligible probability.

An alternative is using a set accumulator (e.g., [Ngu05; CKS09]), which features a constant-size digest (at the cost of public parameters linear in the size of S).

Range proofs. Using the set membership proofs above, one can also efficiently instantiate range proofs [CCS08]. For small ranges $[A, B]$, one may simply use the set membership proof directly ($S = [A, B]$). For larger ranges $[0, b^k - 1]$, one may decompose the witness $w < b^k$ to base b , i.e., $w = \sum_{i=0}^{k-1} w_i b^i$ and prove that all w_i are valid digits (i.e., $w_i \in S = [0, b]$, using the set membership proof) and that they properly represent w (i.e., $w = \sum_{i=0}^{k-1} w_i b^i$). This range proof works because the only numbers that can be represented with k digits are in the interval $[0, b^k - 1]$ as required. Other large ranges $[A, B]$ can intuitively be implemented by proving $w - A \geq 0$ and $B - w \geq 0$, by which we mean $w - A \in [0, b^k - 1]$ and $B - w \in [0, b^k - 1]$ for sufficiently large b^k (i.e., $b^k > B - A$).

We omit details but note that such proofs have been implemented [BEHF21]. We also note that there are other approaches for range proofs in the literature [CBCMRW24].

3.2.5 AND and OR composition of Sigma protocols

We have seen how to prove homomorphism-preimage statements with Schnorr protocols. In this section, we discuss how to prove statements like $\varphi_1(w_1) = y_1 \wedge \varphi_2(w_2) = y_2$ (AND composition) or $\varphi_1(w_1) = y_1 \vee \varphi_2(w_2) = y_2$ (OR composition). The AND composition is

something that is inherent to many of the constructions in this thesis: We often need to combine several statements within the same proof (e.g., “this signature is valid” *and* “the signed message has certain properties”). The OR composition is not strictly required to instantiate the systems we build in this thesis, but it is a strong tool for privacy. For example, it enables users to prove, “I am allowed access because I am an engineer *or* because I am a manager” without revealing which one of the options is true. This makes the anonymity set for authenticating users larger, i.e., they reveal less about their identity.

3.2.5.1 AND composition

Sigma protocol statements can be generically AND-composed: Given a Sigma protocol Π_{Sigma} for relation $\mathcal{R}$, the AND composition is a protocol Π'_{Sigma} that runs Π_{Sigma} twice in parallel with the same challenge $c \leftarrow \mathcal{C}$. This is a protocol for the relation $\mathcal{R}' = \{((stmt_1, stmt_2); (w_1, w_2)) \mid (stmt_1, w_1) \in \mathcal{R} \wedge (stmt_2, w_2) \in \mathcal{R}\}$. However, this comes with the disadvantage that the witnesses w_1, w_2 are completely independent, so this kind of composition does not easily lend itself to proving statements such as “my anonymous credential’s attributes simultaneously have property 1 *and* property 2”, as the prover could use different attributes for property 1 and property 2.

In this thesis, we will generally think of AND composition as happening on the level of Schnorr statements. So for statements $stmt_1 = (\varphi_1, y_1)$ and $stmt_2 = (\varphi_2, y_2)$, where φ_1, φ_2 are over the same variables $(w_1, \dots, w_n)$, we can instead prove the composite statement $stmt_1 \wedge stmt_2 := ((\varphi_1, \varphi_2), (y_1, y_2))$, where $(\varphi_1, \varphi_2)(w_1, \dots, w_n) := (\varphi_1(w_1, \dots, w_n), \varphi_2(w_1, \dots, w_n))$. This has the advantage that, in contrast to the generic composition, φ_1, φ_2 operate over the same variables. We have already seen this in action in Definition 3.8 and Figure 3.5.

For example, to AND-compose statements $h_1 = g^{w_1}$ and $h_2 = g^{w_1} \cdot g_2^{w_2}$, one would artificially extend the first statement’s homomorphism φ_1 with an unused variable w_2 , i.e., $\varphi_1(w_1, w_2) = g^{w_1}$. Together with $\varphi_2(w_1, w_2) = g^{w_1} \cdot g_2^{w_2}$, the overall composite statement would be $(w_1, w_2) \mapsto ((g^{w_1}, g^{w_1} \cdot g_2^{w_2}), (h_1, h_2))$.

In [BEHF21], we have established the notion of *Schnorr fragments*, which enables an even richer way of composing Schnorr-style protocols. In that framework, a set membership or range proof, which are complex send-first Sigma protocols, can be seen as Schnorr fragments and can be easily AND-composed with one another and arbitrary homomorphism statements. We recommend using this notion for implementing Schnorr-style proofs. However, for the purposes of following this thesis, it is not strictly required.

3.2.5.2 OR composition

Sigma protocol statements can also be generically OR-composed. In contrast to the AND composition, there does not seem to be a straightforward way to do OR compositions by manipulating Schnorr statements, but rather we can only OR-compose on the Sigma protocol level. The composition [CDS94] works by giving the prover a degree of freedom so that he can choose the challenge himself for the protocol for which he does not have

a witness. The result is again a Sigma protocol.

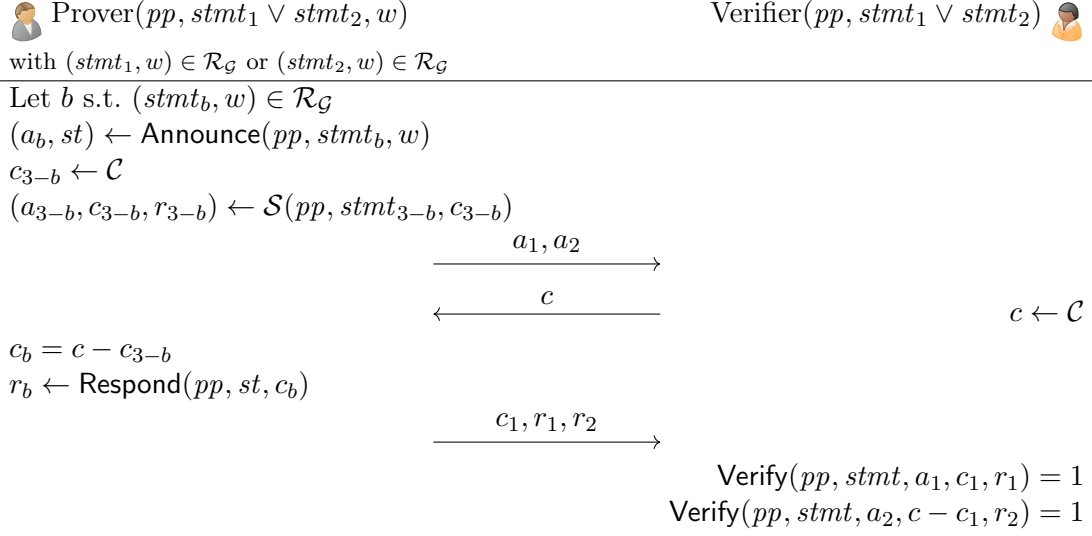


Figure 3.6: Schematic view of an OR proof.

Construction 3.17 (OR proof [CDS94]). Let Π_{Sigma} be a special honest-verifier zero-knowledge Sigma protocol for relation $\mathcal{R}_G$ with challenge space $\mathcal{C}_G$ such that $\mathcal{C}_G$ is a group (e.g., $\mathcal{C}_G = \mathbb{Z}_p$, or $\mathcal{C}_G = \{0, 1\}^\lambda$ with the xor group operation). Let $\mathcal{S}$ be a simulator from the special honest-verifier zero-knowledge property (Definition 3.2) of Π_{Sigma} . Let Π'_{Sigma} be the Sigma protocol with $\Pi'_{\text{Sigma}}.\text{Setup} = \Pi_{\text{Sigma}}.\text{Setup}$ that for statements of the form $stmt_1 \vee stmt_2$ works as in Figure 3.6 and otherwise runs the original protocol Π_{Sigma} . Then Π'_{Sigma} is a Sigma protocol for $\mathcal{R}'_G = \mathcal{R}_G \cup \{(stmt_1 \vee stmt_2; w) \mid (stmt_1, w) \in \mathcal{R}_G \vee (stmt_2, w) \in \mathcal{R}_G\}$.

As one can see in Figure 3.6, the prover essentially runs two copies of the original protocol in parallel. For one of the protocols, the prover uses a random challenge c_{3-b} chosen by himself, which allows him to simulate a transcript. The other protocol is run honestly, using the prover's witness to compute the protocol messages. For the two challenges c_1, c_2 , it must hold that $c_1 + c_2 = c$ (where c is the actual challenge sent by the verifier). This means that after the prover chooses c_{3-b} for one of the protocol copies, the challenge c uniquely determines the challenge c_b for the other protocol copy, making the challenge c_b unpredictable, ensuring that this OR composition is sound.

Essentially the same construction also works for OR-composing two different Sigma protocols for different relations. We now discuss how this construction preserves the zero-knowledge and soundness properties of the original protocol.

Theorem 3.18. *If Π_{Sigma} is special honest-verifier zero-knowledge, then the OR composition (Construction 3.17) is special honest-verifier zero-knowledge.*

Simulation is easy: Given challenge c , choose random c_1, c_2 with $c_1 + c_2 = c$ and compute the rest of the transcript using the simulators of the underlying protocol for $stmt_1, c_1$ and $stmt_2, c_2$. We omit a formal proof.

Theorem 3.19. *If the underlying protocol has special soundness, the OR composition also has special soundness. If the underlying protocol has strong special soundness, the OR composition also has strong special soundness.*

The proof is a simple exercise.

Because the OR proof is a Sigma protocol again, we can recursively OR-compose it with itself to prove more complex OR statements. For example, let Π''_{Sigma} be the OR proof construction for Π'_{Sigma} (which in turn is the OR proof construction for Π_{Sigma}). This allows us, for $stmt_1, stmt_2, stmt_3$ from the original relation $\mathcal{R}$, to prove the statement $stmt_1 \vee stmt_2 \vee stmt_3$ (written as $(stmt_1 \vee stmt_2) \vee stmt_3$) using Π''_{Sigma} . Together with the Schnorr-level AND composition explained above, one can essentially prove any disjunctive normal form composition of provable statements. For example, say we want to prove possession of a signature on a message m such that $\varphi_1(m) = 1$ or $\varphi_2(m) = 1$, i.e., we want to prove that we know m, σ such that $\text{Verify}(pp, pk, m, \sigma) = 1 \wedge (\varphi_1(m) = 1 \vee \varphi_2(m) = 1)$. To achieve this, we bring this statement into disjunctive normal form: $(\text{Verify}(pp, pk, m, \sigma) = 1 \wedge \varphi_1(m) = 1) \vee (\text{Verify}(pp, pk, m, \sigma) = 1 \wedge \varphi_2(m) = 1)$. We can design a Schnorr-style proof for each $(\text{Verify}(pp, pk, m, \sigma) = 1 \wedge \varphi_i(m) = 1)$ and then OR-compose them.

More generally, using generalizations of the protocols shown above, one can compose *arbitrary trees* of AND/OR statements [CDS94]. However, the disjunctive normal form has the advantage that it inherently does not cause issues with variables that are required to be the same in different sub-statements (since only one of the OR operands has to be fulfilled and the other operand is irrelevant, it does not matter whether the variables are in any way “consistent” in the other operand). This makes the disjunctive normal form convenient to use (in particular, the transformation sketched above yields the results one would expect). In some cases, one may want to deviate from this approach in order to improve performance.

3.3 Defining zero-knowledge proofs of knowledge

As mentioned at the beginning of Chapter 3, the first step to designing a proof system is to design a Sigma protocol for the statement we want. We have seen how to do this in Sections 3.1 and 3.2. Designing a Sigma protocol is relatively easy because the properties we require of Sigma protocols are simple. Sigma protocols do not have a zero-knowledge property against dishonest verifiers and their soundness notion only describes the very last step of the extraction process, leaving it open how to find two accepting transcripts from a successful prover.

The next step is then to compile this Sigma protocol into a real-world secure protocol. But first, we need to define “real-world” security. This section deals with the formal definition. This means we now leave the perspective of a protocol designer and take the

perspective of someone wanting to *use* the protocol in their construction (which is the perspective we will need for most of this thesis). Our focus will be on *strong* security definitions (which are useful for security proofs of constructions relying on proof systems) instead of *simple* ones (which were more useful for designing new protocols). Then later, in Section 3.4, we show how to convert a Sigma protocol to a protocol that fulfills the stronger definition.

3.3.1 Syntax of proof systems

We first define the syntax of proof systems as follows.

Definition 3.20 (Proof system). An (interactive) *proof system* Π_{zk} for relation $\mathcal{R}_{\mathcal{G}}$ (which may depend on $\mathcal{G}$) consists of the following ppt algorithms/protocols

$\text{Setup}(\mathcal{G}) \rightarrow pp$

Generates public parameters pp .

$\text{Prove}(pp, stmt, ctx; w) \leftrightarrow \text{Verify}(pp, stmt, ctx) \rightarrow b$

An interactive protocol with common input pp , a statement $stmt$, and a context string ctx , and with a witness w as the prover's private input. The verifier outputs a bit b . $\forall b = 1$ indicates that the verifier is convinced that $(stmt, w) \in \mathcal{R}_{\mathcal{G}}$.

Π_{zk} is *complete*, i.e., for all $\lambda \in \mathbb{N}$, all $\mathcal{G} \in [\text{GenSetting}(1^\lambda)]$, $pp \in [\text{Setup}(\mathcal{G})]$, all $(stmt, w) \in \mathcal{R}_{\mathcal{G}}$ and all $ctx \in \{0, 1\}^*$,

$$\Pr[b = 1 \mid \text{Prove}(pp, stmt, ctx, w) \leftrightarrow \text{Verify}(pp, stmt, ctx) \rightarrow b] = 1$$

Completeness guarantees that if the prover uses a valid witness for the statement $stmt$, the verifier always accepts. The context string ctx allows us to bind the execution of this protocol to some specific context. This is further discussed in Section 3.3.2.

Definition 3.20 is written such that the proving process is an interactive protocol. A special case of this are non-interactive proofs.

Definition 3.21 (Non-interactive proof system). A proof system Π_{zk} is *non-interactive* if its $\text{Prove} \leftrightarrow \text{Verify}$ protocol consists of a single message from the prover to the verifier.

Because a proof in a non-interactive Π_{zk} consists only of a single message, this message itself can be called a “proof”. Non-interactive proofs are particularly useful for two reasons. First, they do not require multiple rounds of communication, which represent a performance issue (as round trips take time), but also an API issue (interactive proofs are not as easy to integrate into, say, a REST API). Second, because the non-interactive verifier is completely passive, anyone can take the role of the verifier and check the proof. As a consequence, non-interactive proofs can be treated like digital signatures in the sense that they are transferable and publicly verifiable. Indeed, non-interactive proofs are sometimes called *signatures of knowledge* (where ctx can be considered the signed message). This is in contrast to interactive proofs, which are not necessarily convincing to third parties.

3.3.2 Relay attacks

In our Definition 3.20, the prover and verifier take an additional context string ctx as input that you would not find in most proof system definitions, especially interactive ones. ctx will serve a role around a security guarantee for the prover, making sure that the proof is bound to the context the prover expects.

To illustrate the need for this, say we use a proof system for authentication, i.e., the prover proves knowledge of some secret in order to gain access to a resource (indeed, this will essentially be our use case for anonymous credentials). Now what could happen is that we have a party $\mathcal{A}$ and services S, S' . $\mathcal{A}$ wants to authenticate with service S , so he runs the proof with them. However, while this is happening, S may try to authenticate with S' as $\mathcal{A}$. The service S does so by simply relaying all the protocol messages between $\mathcal{A}$ and S' as in Figure 3.7. At the end, this means that S is able to impersonate $\mathcal{A}$ w.r.t. S' by using $\mathcal{A}$'s ability to answer the challenge sent by S' .

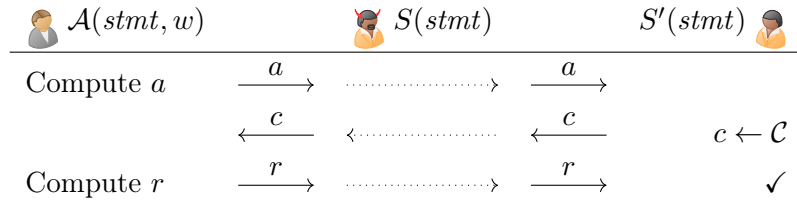


Figure 3.7: An attack where S takes the role of prover w.r.t. S' and the role of verifier w.r.t. $\mathcal{A}$. In the end, S will have convinced S' that S knows $\mathcal{A}$'s secret, allowing S to impersonate $\mathcal{A}$. Example protocol with three messages.

This sort of relay attack [BBDGQ91; Dam10] is guaranteed to work by the completeness property (cf. Definition 3.20) of the proof system. Note that it cannot be prevented by external means; for example, the attack still works if the channels between S and S' and between S and $\mathcal{A}$ are secured by TLS: Everyone is already sending messages exactly to whom they want to be sending messages.

To thwart relay attacks, in practice, a suitable ctx should be chosen and sanity-checked by $\mathcal{A}$. For example, in the scenario above, ctx may be a string like “I want to authenticate with S for session 0x8a52”. This string clearly describes what $\mathcal{A}$ is allowing this protocol execution to be used for. S' will not accept ctx as a context string for authentication and $\mathcal{A}$ will not run the protocol for some ctx' that indicates authentication with S' (instead of the expected S). Our security definitions later will guarantee that $\mathcal{A}$'s answers with respect to ctx are useless to S for $ctx' \neq ctx$, thwarting the attack.

For non-interactive proofs, this sort of context string is standard in the literature (often called the *message* when looking at the proof as a *signature* of knowledge). Indeed, the problem of being able to relay proofs is much more obvious in the non-interactive case because *as a feature*, non-interactive proofs can be relayed like digital signatures.

Relay attacks apply to other (authentication) scenarios and constructions in this thesis. To deal with this, many of the syntax definitions of larger constructions also contain references to ctx , serving the same function as explained above. In implementations, applications must carefully choose ctx so that the proof's scope (or more generally, the

protocol's scope) is limited to what the prover intends. This can be compared to a password manager first checking what domain (*ctx*) a password is going to be sent to before sending it, to prevent phishing attacks.

3.3.3 Overview of desirable security properties

For the security of proof systems, we are interested in two properties: (1) zero-knowledge and (2) proof of knowledge, as intuitively described in at the beginning of Chapter 3. In this section, we differentiate these properties further based on established terminology. Afterward, in Section 3.3.4, we introduce the formal definition we use for this thesis.

The following (intuitively described) security properties refine the general ideas of zero-knowledge and proof of knowledge.

Concurrent zero-knowledge [Dam00]

Means that concurrently run instances of the proof system can be simulated. In contrast, more traditional zero-knowledge properties [Gol01] only require simulation to work for a single, isolated run of the protocol. Concurrent zero-knowledge is a crucial property for most real-world scenarios, where almost always, multiple (correlated) protocols instances can be run concurrently.

Witness-hiding [FS90]

Means that an adversary cannot distinguish a proof using witness w_0 from a proof for the same statement using witness w_1 . This property is implied by zero-knowledge, as both the proof with w_0 and the proof with w_1 are indistinguishable from a simulated proof, and hence by transitivity, they are indistinguishable.

Soundness

Means that it is hard for a dishonest prover to convince an honest verifier of a *false* statement, i.e., one for which there *exists* no witness. Soundness is the most basic of security requirements for the verifier and is always expected. It is implied by the *proof of knowledge* property (because if a witness can be *extracted*, it must also *exist*, guaranteeing validity of the statement).

Straight-line extractability [Fis05]

Means that the proof of knowledge extractor can efficiently compute a witness given only a copy of the verifier's view in the protocol (and the prover's random oracle queries). The name comes from the fact that straight-line extractors do not use rewinding for extraction, i.e., they keep the execution graph of the prover a straight line, instead of forking it like rewinding extractors would (cf. Figure 3.10). Straight-line extractability is a property that greatly simplifies security proofs.

Adaptive extractability [BFW16]

Means that the proof of knowledge extractor also works for multiple proofs, where each proof may depend (in a computation-flow sense) on the extracted witnesses of the previous proofs (which motivates the name “adaptive”). This is in contrast

to the standard notions of extractions, which only consider extraction of a single witness from a single isolated run of the prover [Gol01].

Usually, adaptive extractability is implemented via straight-line extractability. Indeed, one can show that in some models (with non-interactive proofs), all adaptive proofs are straight-line extractable proofs [BNW17]. However, in one of our models, (interactive) protocol execution happens sequentially, which enables us to set up an adaptively extractable proof with rewinding (i.e., without straight-line extractability).

Simulation-soundness

Means that soundness still holds in the presence of simulated proofs. So even if a malicious prover can ask to see simulated proofs, they still cannot prove false statements. This property is required for all security proofs in which the reduction algorithm both simulates some proofs but still requires the adversary's proofs to be sound. While there are sometimes ways around requiring this property, this is the exception rather than the norm for privacy-preserving constructions.

For non-interactive proofs, simulation-soundness is often understood to imply non-malleability of the proof, i.e., given a simulated proof π for a false statement, it is infeasible to create a new proof $\pi' \neq \pi$ for the same statement. However, we do not use the terminology that way.

Simulation-extractability

Means that the proof-of-knowledge/extractability property still holds in the presence of simulated proofs. This property is completely analogous to simulation-soundness, but with extractability. This also means that simulation-extractability implies simulation-soundness.

For our anonymous credential applications later, our proofs have to be concurrent zero-knowledge and adaptively simulation-extractable proof of knowledge.

The properties mentioned above are useful descriptors when designing new proof systems and also to describe the different possible requirements for a proof system. We now take the perspective of the *user* of a proof system, i.e., the designer of a larger construction. Putting the long list of possible properties aside, how do we want to be able to think about a proof system? In a security proof for our constructions, we intuitively want that:

- (1) we can essentially ignore any occasions where the construction forces *us* to prove something *to the adversary* (because we can just simulate that proof),
- (2) we can read any occasions where *the adversary* proves something *to us* as “the adversary sends us a witness” (because we can extract witnesses from proofs).

These intuitive properties will be formalized in Section 3.3.4. We will require that there are (efficient) machines $\mathcal{S}, \mathcal{E}$, such that (1) whenever we have to prove a statement to the adversary using a witness, we can instead make $\mathcal{S}$ prove the statement to $\mathcal{A}$ without any witness, and (2) whenever the adversary convinces the honest verifier algorithm, we can ask $\mathcal{E}$ to compute a witness for the statement.

3.3.4 Concurrently adaptively simulation-extractable zero-knowledge proofs of knowledge

To model our informal security expectation explained above, we require the existence of machines $\mathcal{S}, \mathcal{E}$ that play the role of the simulator and extractor, respectively. We then consider some adversary $\mathcal{A}$. Zero-knowledge means that $\mathcal{A}$ cannot distinguish interactions with honest provers (with witnesses determined by $\mathcal{A}$) from interactions with the simulator $\mathcal{S}$ (which does not get a witness as input). Proof of knowledge means that whenever $\mathcal{A}$ successfully convinces an honest verifier, $\mathcal{E}$ must be able to output a witness (to $\mathcal{A}$, modeling adaptive extractability). This should hold even if $\mathcal{A}$ gets to see some simulated proofs (cf. simulation extractability).

The adversary $\mathcal{A}$ must not be able to simulate or extract proofs. To understand this, note that one generally cannot extract witnesses from simulated proofs. If this were possible, we would be able to compute witnesses for arbitrary (even computationally hard) statements out of thin air by first simulating a proof for the statement and then extracting the witness from it. So if $\mathcal{A}$ were able to simulate proofs, the extractor would not be able to extract proofs. If $\mathcal{A}$ were able to extract proofs, $\mathcal{A}$ would be able to distinguish simulated proofs from real proofs by checking whether extraction succeeds. As a consequence, the simulator $\mathcal{S}$ and extractor $\mathcal{E}$ need some sort of advantage over $\mathcal{A}$ so that simulation/extraction becomes possible for them but not for $\mathcal{A}$.

In our definition, we give $\mathcal{S}, \mathcal{E}$ two advantages: First, the simulator $\mathcal{S}$ will be allowed to set up the proof system's public parameters, allowing it to secretly embed a trapdoor that can be used for simulation. Second, the simulator $\mathcal{S}$ gets to control the random oracle. Because there will be information flow from the simulator $\mathcal{S}$ to the extractor $\mathcal{E}$, the extractor can also benefit from a possible trapdoor or random oracle information. If some advantage is not required by $\mathcal{S}, \mathcal{E}$, they can simply ignore it (e.g., simulate the random oracle honestly if they do not need to program it).

Definition 3.22 (Proof system security). We say that Π_{zk} is a *concurrently adaptively simulation-extractable zero-knowledge proof of knowledge* (in the ROM) if there exists an interactive stateful algorithm $\mathcal{S}$ and a stateless algorithm $\mathcal{E}$ such that

- Zero-knowledge: For all ppt $\mathcal{A}$,

$$\left| \begin{array}{l} \Pr[\mathcal{A}^{\mathcal{RO}, \mathcal{O}. \text{Prove}^{\text{REAL}}}(\mathcal{G}, pp) = 1 \mid pp \leftarrow \text{Setup}(\mathcal{G})] \\ - \Pr[\mathcal{A}^{\mathcal{S}. \mathcal{RO}, \mathcal{O}. \text{Prove}^{\text{SIM}}}(\mathcal{G}, pp) = 1 \mid pp \leftarrow \mathcal{S}. \text{Setup}(\mathcal{G})] \end{array} \right| \text{ is negligible}$$

- Proof of knowledge / simulation extractability: For all ppt $\mathcal{A}$,

$$\Pr \left[\mathcal{O}. \text{Extract} \text{ raises } \text{ExtractionError} \mid \begin{array}{l} pp \leftarrow \mathcal{S}. \text{Setup}(\mathcal{G}) \\ \mathcal{A}^{\mathcal{S}. \mathcal{RO}, \mathcal{O}. \text{Simul}, \mathcal{O}. \text{Extract}}(\mathcal{G}, pp) \end{array} \right]$$

is negligible.

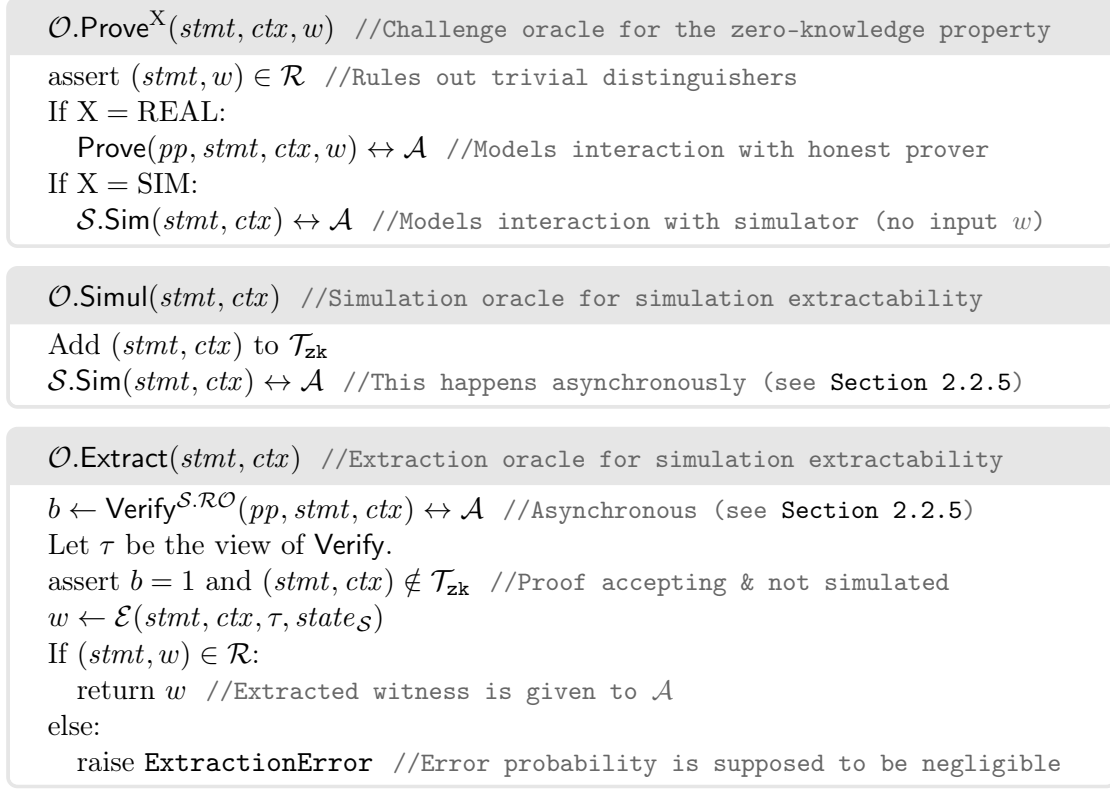


Figure 3.8: Oracles for proof system security.

- Runtime: For all ppt $\mathcal{A}$, there exists a polynomial rt such that the runtime of a complete run of $pp \leftarrow \mathcal{S}.$ Setup($\mathcal{G}$), $\mathcal{A}^{\mathcal{S}, \mathcal{RO}, \mathcal{O}.\text{Simul}, \mathcal{O}.\text{Extract}}(\mathcal{G}, pp)$ (including runtime of the oracles and $\mathcal{S}, \mathcal{E}$) is at most $rt(\lambda)$ for all λ .

where $\mathcal{G} \leftarrow \text{GenSetting}(1^\lambda)$ in each of the probabilities and the oracles are defined as in Figures 2.1 and 3.8.

We say that Π_{zk} is (only) *concurrent zero-knowledge* (in the ROM) if there is an interactive stateful algorithm $\mathcal{S}$ such that the first bullet point holds and the runtime of $\mathcal{A}^{\mathcal{S}, \mathcal{RO}, \mathcal{O}.\text{Prove}^{\text{SIM}}}$ is upper-bounded by some polynomial (which may depend on $\mathcal{A}$) similar to the last bullet point.

In the following, we discuss this definition.

Concurrency. The “concurrent” part of the Definition 3.22 naming refers to the fact that protocol executions (both proving and verifying) can be interleaved, i.e., for each oracle call, $\mathcal{A}$ can choose whether start a new protocol or whether to continue one of the currently running protocols (see Section 2.2.5 for an explanation of the $\leftrightarrow$ notation in the oracles). This models arbitrarily interleaved concurrent execution of protocols in practice, which is what we usually want.

Zero-knowledge. The zero-knowledge property (more specifically: concurrent zero-knowledge) is easily modeled: $\mathcal{A}$ must not be able to distinguish between interacting with an honest prover (and an honest random oracle and honestly generated pp) and interacting with the simulator $\mathcal{S}$ (and the random oracle implemented by $\mathcal{S}$ and pp generated by $\mathcal{S}$).

We only require $\mathcal{S}$ to simulate proofs for which $\mathcal{A}$ supplies a valid witness (even though the witness is withheld from $\mathcal{S}$ by the $\mathcal{O}.\text{Prove}^{\text{SIM}}$ oracle). If the $\mathcal{O}.\text{Prove}^X(stmt, ctx, w)$ oracle did not check validity of w , $\mathcal{A}$ could trivially distinguish honest from simulated proofs by supplying an invalid witness w' . In that case, we would expect the honest prover to fail, but $\mathcal{S}$ still produces a convincing proof (because $\mathcal{S}$ does not get to see $w' \neq w$ and hence $\mathcal{S}$ is oblivious regarding its validity). We do however note that this does not mean that $\mathcal{S}$ can generally only simulate proofs for true statements (i.e., statements for which a witness exists). Intuitively, if it is hard to decide whether any witness for a statement exists, $\mathcal{S}$ cannot decide this efficiently either, so it will *have to* provide a proof that “looks convincing”, anyway.

Proof of knowledge. The proof of knowledge property (more specifically: concurrent adaptive simulation-extractability) is modeled by allowing an adversary $\mathcal{A}$ to ask for simulated proofs (via $\mathcal{O}.\text{Simul}$) and to have a witness extracted from an interaction between $\mathcal{A}$ and an honest prover (via $\mathcal{O}.\text{Extract}$). Note that in contrast to the simulation oracle $\mathcal{O}.\text{Prove}^{\text{SIM}}$ in the zero-knowledge part of the definition, $\mathcal{O}.\text{Simul}(stmt, ctx)$ does not take a witness as input and does not execute any checks about the validity of $stmt$. This means that for the sake of extraction, the simulator may even be presented with statements for which no witness is known or no witness exists.

When $\mathcal{A}$ asks for witness extraction with the $\mathcal{O}.\text{Extract}$ oracle, the oracle first has $\mathcal{A}$ prove the statement to an honest verifier and then, if the honest verifier is convinced, the oracle runs the extraction process. The property states that the probability the extractor fails to output a valid witness (over any of the oracle queries), is negligible. $\mathcal{A}$ can query the $\mathcal{O}.\text{Extract}$ oracle multiple times, both sequentially and interleaving its execution with other oracle queries. This also means that $\mathcal{A}$ can, in principle, attempt a relay attack by relaying the messages of the simulator to the honest verifier in the extraction oracle. Trivially, this procedure must (almost always) succeed if the simulator simulates for the same $(stmt, ctx)$ that the extraction oracle verifier expects. However, in the relay case, clearly $\mathcal{A}$ does not require any *knowledge* of the witness to convince the honest verifier, meaning that we cannot expect the extractor to be able to output a witness. To prevent this trivial attack, the extractor is only invoked if $(stmt, ctx) \notin \mathcal{T}_{zk}$, i.e., $\mathcal{A}$ has not requested simulation for $(stmt, ctx)$.

For the extraction process, the extractor $\mathcal{E}$ gets as input the statement $stmt$ and context ctx of the proof, as well as the verifier’s view τ and the current state $state_{\mathcal{S}}$ of the simulator $\mathcal{S}$. Depending on the concrete construction of $\mathcal{S}$, this state can contain, for example, (1) a trapdoor for the public parameters pp , or (2) data about random oracle entries programmed by $\mathcal{S}$ or queried by $\mathcal{A}$.

Some works distinguish *proof of knowledge* and *argument of knowledge*. The former

is sometimes understood to carry an information-theoretical notion: even adversaries with unbounded runtime cannot break the proof of knowledge property. In contrast, arguments of knowledge only claim security against ppt provers. We do not make this distinction and simply call the property “proof of knowledge”, even though Definition 3.22 only guarantees security against ppt adversaries.

Runtime. The runtime constraint required in the definition ensures that if $\mathcal{A}$ is a polynomial-time algorithm, then also the overall runtime of the extraction scenario runs in polynomial-time, which implies that simulation/extraction is not too inefficient to do. Note that the runtime of $\mathcal{S}$ depends on $\mathcal{A}$: If $\mathcal{A}$ makes k oracle queries, then also $\mathcal{S}$ will have to run for at least time k . For this reason, we cannot simply say “ $\mathcal{S}, \mathcal{E}$ are ppt”, as that would mean that $\mathcal{S}$ *always* run in polynomial time, independent of how many queries $\mathcal{A}$ makes. Instead, the polynomial rt , which (indirectly) constrains the runtime $\mathcal{S}, \mathcal{E}$, depends on $\mathcal{A}$.

Usage remarks This definition is designed to be easy to use for security proofs that require the proof system’s security properties. It is intended to be used as follows. The starting point is some security game G_0 that creates and verifies proofs. First, we modify G_0 to G_1 so that it uses the simulator $\mathcal{S}$ to set up the pp for the proof system and to create proofs. By the zero-knowledge property, there is a negligible difference in the result of those two games (to prove this, one has to argue that the witnesses used by the game are all valid).

If desired, one can now make further changes to the game, using the fact that witnesses w are now unused (not input to the simulator). In particular, those changes may include modifying proof statements $stmt$ to $stmt'$ in a way that is undetectable to ppt algorithms. For example, assume $stmt$ contains an encryption of x . We can imagine replacing the encryption of x with an encryption of 0, even if that means that there does not exist a valid witness for $stmt'$ anymore. When proving that the encryption modification only negligibly changes the game output, one can regard the simulator $\mathcal{S}$ to just operate on the ciphertexts (without any access to anything else such as encryption randomness that may have been part of the witness in G_0). So if $\mathcal{S}$ *were* to suddenly output invalid proofs, that could be used to distinguish encryptions.

Finally, in the setting using the simulator, extraction now amounts to just using the extractor $\mathcal{E}$ as a subroutine (with verifier view τ , public proof data $stmt, ctx$, and the current simulator state $state_{\mathcal{S}}$ as input). This can be done, essentially, in any setting where $state_{\mathcal{S}}$ is available, so both for (1) game-hopping to a game where extracted witnesses are to be used within the game and for (2) reduction algorithms, where the extracted witness is to be used to break some assumption. The guarantee that the extractor reliably outputs witnesses for valid (non-simulated) proofs follows easily from the simulation extractability property.

Note that information only ever flows from the simulator to the extractor (via $state_{\mathcal{S}}$), not the other way around. This design is deliberate (and motivates differentiating between $\mathcal{S}$ and $\mathcal{E}$ in the first place) because if extraction had side effects regarding sim-

ulation, we would not be able to argue that extraction does not change the game’s outcome significantly. Indeed, the only guarantee that the simulator “works” is from the zero-knowledge property, which is in a setting where no extraction happens. So if the simulator were to *notice* extraction, we have no guarantee that it does not simply stop working afterward. With the current design, there are no side effects to extraction other than that the game now has access to the witness. It is on the writer of the security proof to use the witness in a manner that does not impact the game result noticeably.

3.4 Converting Sigma protocols to secure proof systems: Fischlin’s transform

In this section, we take a Sigma protocol Π_{Sigma} , which only fulfills the weak properties from Section 3.1 and convert it to a proof system Π_{zk} that is a zero-knowledge proof of knowledge according to Definition 3.22.

For this, we consider Fischlin’s transform [Fis05], which gives us *straight-line extractable* proofs in the random oracle model, i.e., they will be secure in a concurrent execution setting (Definition 3.22). Because we want to work in the concurrent setting, we cannot rely on rewinding to find two transcripts $(a, c, r), (a, c', r')$ for extraction (cf. Section 3.5). The idea of Fischlin is to force the prover to query a random oracle with transcripts $(a, c, r), (a, c', r')$ in order to create a valid proof. This way, when we want to extract a witness, we only need to look through the list of random oracle queries to find the two transcripts we need for extraction, which can be done without rewinding. This idea has also been applied in Pass’s transform [Pas03], which is less efficient than Fischlin’s.

The intuition for Fischlin’s transform is as follows. Forcing the prover to query the random oracle works in a *proof of work* fashion: Take some random oracle $\mathcal{RO} : \{0, 1\}^* \rightarrow \{0, \dots, W-1\}$ with relatively small codomain (e.g., by truncating the output of a random oracle with large codomain). The idea is to assign a value $v = \mathcal{RO}(a, c, r)$ to a transcript (a, c, r) , and we only accept proofs with $v = 0$. The prover will try several challenges c and query the random oracle with the corresponding responses r_c until he finds one for which (a, c, r_c) has a zero hash $v = 0$. During this process, the prover likely queries $\mathcal{RO}(a, c, r_c)$ and $\mathcal{RO}(a, c', r_{c'})$ with $c \neq c'$, hence the extractor controlling the random oracle can compute a witness from the queried transcripts. Of course, we need to ensure that a malicious prover cannot simply choose fresh announcements a every time he tries for zero hash values (as we need them to query two transcripts with the *same* announcement a but different challenges c). To achieve this, the protocol is essentially run n times in parallel and the prover commits to all announcements $(a_i)_{i=1}^n$ in the beginning. More concretely, we can imagine the value v_i for the i th protocol instance to be $v_i = \mathcal{RO}((a_i)_{i=1}^n, i, c_i, r_i)$. It is very unlikely that *all* n instances require only a single try to find $\mathcal{RO}(\dots) = 0$, so for at least one i , the adversary will have to try multiple c_i for at least one i (or throw away *all* a_i and try again, but the probability for this to succeed is, again, negligible).

We use the *randomized* variation of Fischlin’s transform [KS22], where the honest prover tries random challenges instead of going through a list of fixed challenges. This

avoids subtle issues present in proof of the original (non-randomized) construction's zero-knowledge property [Fis05], which only works for Sigma protocols with properly unique responses (i.e., for every a, c , there *exists* exactly one accepting response r). We will prove the randomized construction secure for the much larger class of Sigma protocols with strong soundness (Definition 3.6). The construction and the strong soundness insight are taken (in spirit) from [KS22], our security proof is new work, of course building on the ideas of [Fis05; KS22].

Construction 3.23 (Randomized Fischlin transform). Let Π_{Sigma} be a Sigma protocol for relation $\mathcal{R}_{\mathcal{G}}$. Let $W \in \mathbb{N}$. Let $\mathcal{RO} : \{0, 1\}^* \rightarrow \{0, \dots, W-1\}$ be a hash function (modeled as a random oracle). Let $n := n(\lambda) = \lambda / \log(W)$ and $m := m(\lambda) = \lambda \cdot W$. Fischlin's transform is a proof system for $\mathcal{R}$, defined as follows:

Setup($\mathcal{G}$) $\rightarrow pp$

Generates $pp \leftarrow \Pi_{\text{Sigma}}.\text{Setup}(\mathcal{G})$.

Prove($pp, stmt, ctx; w$) $\leftrightarrow$ Verify($pp, stmt, ctx$) $\rightarrow b$

The prover computes a proof π as follows:

- Compute announcements $(a_i, st_i) \leftarrow \text{Announce}(pp, stmt, w)$ for $i \in \{1, \dots, n\}$.
- For each $i \in \{1, \dots, n\}$:
 - Let $U = \emptyset$.
 - Pick random challenge $c \leftarrow \mathcal{C} \setminus U$. Compute a response $r \leftarrow \text{Respond}(pp, st_i, c)$ and check whether $\mathcal{RO}(stmt, ctx, (a_\ell)_{\ell=1}^n, i, c, r) = 0$. If so, set $c_i := c$ and $r_i := r$ and go on to the next i in the loop. Otherwise, add c to U and repeat this step.
 - When the step above has been tried unsuccessfully m times (i.e., $|U| = m$), the prover sends the witness $w =: \pi$ as the proof to the verifier and stops.
- The proof is $\pi = (a_i, c_i, r_i)_{i=1}^n$.

The prover sends π to the verifier. The verifier outputs $b = 1$ for $\pi = (a_i, c_i, r_i)_{i=1}^n$ iff for all $i \in \{1, \dots, n\}$,

- $\text{Verify}(pp, stmt, a_i, c_i, r_i) = 1$ and
- $\mathcal{RO}(stmt, ctx, (a_\ell)_{\ell=1}^n, i, c_i, r_i) = 0$.

otherwise, the verifier outputs $b = 0$ (unless $(stmt, \pi) \in \mathcal{R}$, in which case it outputs $b = 1$).

It is clear that completeness holds, i.e., if $(stmt, w) \in \mathcal{R}$, then the verifier accepts an honest prover's proof.

For pure formal convenience reasons, in order to ensure that the prover is always able to output a convincing proof in (strict) polynomial time, as required in Definition 3.20, we make the prover output their witness as proof if the process of looking for accepting transcripts with zero hash takes too long. The probability that this happens is negligible

(as we will discuss in the proof of the zero-knowledge property). In reality, one would just implement the prover to retry random challenges until a zero hash value is found.

The *expected* runtime for computing a proof is only $n(\lambda) \cdot W = \lambda \cdot W / \log(W)$ invocations of $\mathcal{RO}$ (the worst-case runtime is at most $n(\lambda) \cdot m(\lambda) = \lambda^2 \cdot W / \log(W)$ invocations of $\mathcal{RO}$). By increasing the parameter W , one can decrease proof size $\lambda / \log(W)$ (and hence verifier runtime) as a tradeoff for increasing the expected prover computation time $\lambda \cdot W / \log(W)$. Note, however, that the prover computation time increase is almost linear in W , while the proof size savings only factor in logarithmically. See Table 3.1 for typical numbers.

W	Expected prover runtime $\lfloor \lambda \cdot W / \log(W) \rfloor$	Proof size $\lceil \lambda / \log(W) \rceil$
2	256	128
4	256	64
8	341	43
16	512	32
32	819	26
64	1365	22
128	2341	19
256	4096	16
512	7281	15
1024	13107	13
2048	23831	12

Table 3.1: Values for the parameter W in Construction 3.23 and their corresponding performance characteristics. All numbers are with $\lambda = 128$, which means that the probability for creating a proof by pure hashing luck is upper-bounded by $q \cdot 2^{-128}$, where q is the number of random oracle queries the malicious prover makes. Expected prover runtime is measured in the number of hash function invocations (and Sigma protocol response computations) and the proof size is measured in the number of underlying Sigma protocol transcripts $n(\lambda)$ that make up the proof π .

There are variations to this scheme: In [Fis05], Fischlin only requires that the prover find transcripts whose hashes *sum up to a small number* instead of requiring them all to be zero. This makes it easier for provers to compute an accepting proof, but complicates the analysis. In [KS22], they have the prover grind for *hash collisions* instead of hash zeros, which decreases the number of transcripts the prover has to try slightly. [CL24] describes optimizations for implementing Fischlin’s transform.

Note that Sigma protocol transcripts can often be compressed [BEHF21]. For example, a Schnorr protocol (Construction 3.7) transcript (a, c, r) can be compressed to (c, r) as the accepting announcement a is uniquely determined by and efficiently computable from (c, r) as $\varphi(r) - c \cdot y$ (for $stmt = (\varphi, y)$). Additionally, for a send-first protocol (Construction 3.12), one can re-use the same send-first value sf for all n announcements. These insights can be used to compress a Fischlin proof in practice.

Theorem 3.24. *If Π_{Sigma} is honest-verifier zero-knowledge (Definition 3.2), has strong special soundness (Definition 3.6), and super-poly challenge space (Definition 3.4), then the randomized Fischlin transform (Construction 3.23) is a concurrently adaptively simulation-extractable zero-knowledge proof of knowledge in the random oracle model (Definition 3.22).*

Intuitively, to simulate a proof, the simulator $\mathcal{S}$ can circumvent the proof of work by programming the random oracle. $\mathcal{S}$ generates n simulated Sigma protocol transcripts (a_i, c_i, r_i) . $\mathcal{S}$ then programs the random oracle such that $\mathcal{RO}(stmt, ctx, (a_\ell)_{\ell=1}^n, i, c_i, r_i) = 0$ and outputs the simulated proof $(a_i, c_i, r_i)_{i=1}^n$. An issue in existing proofs for Construction 3.23 is that the process described above biases the random oracle. The random oracle is skewed towards 0 in the sense that the programmed preimages are programmed to have a zero image while all other preimages have a uniformly random image. For this reason, zero is (slightly) overrepresented as a random oracle image in the presence of the simulator.

One can see that this is indeed an issue by considering an extreme thought experiment. Suppose an adversary against the zero-knowledge property asks for an absurdly high number of proofs, with the same statement $stmt$ and context ctx for all proofs. Suppose the number of simulation queries is so high that it is likely the simulator will have randomly generated *all* possible combinations of n transcripts. If that happens, then *all* random oracle values $\mathcal{RO}(stmt, ctx, (a_\ell)_{\ell=1}^n, i, c_i, r_i)$ have been programmed to zero. Hence a trivial way for the adversary to distinguish real from simulated proofs is to ask for this absurd number of proofs and then generate n random transcripts $(a_i, c_i, r_i)_{i=1}^n$ using the Sigma protocol simulator and query the random oracle for $\mathcal{RO}(stmt, ctx, (a_\ell)_{\ell=1}^n, i, c_i, r_i)$ for all $i \in [n]$. If the Fischlin proofs were simulated, these random oracle queries will likely all yield 0. If the Fischlin proofs were generated by an honest prover, it is very unlikely that n random transcripts all hash to zero (as we will see in the proof of knowledge part of the proof for Theorem 3.24). Of course, a ppt adversary cannot actually query for so many proofs. However, even a single simulated proof already (very slightly) increases the chance of the adversary's $\mathcal{RO}(stmt, ctx, (a_\ell)_{\ell=1}^n, i, c_i, r_i)$ queries to all yield zero. This needs to be handled in the proof.

To fix this issue, our simulator $\mathcal{S}$ will virtually grind for random oracle images $v = 0$, trying different random challenges c similar to how the honest prover would. However, when emulating an honest prover's grind for zero images, $\mathcal{S}$ cannot know the exact preimages $(stmt, ctx, (a_\ell)_{\ell=1}^n, i, c'_i, r'_i)$ (which contains Sigma protocol responses r'_i for different challenges c'_i) that the honest prover would query, as that would imply that $\mathcal{S}$ can compute a witness. To work around this, $\mathcal{S}$ simply “burns” the challenges c'_i for which $\mathcal{S}$ *should* have programmed the random oracle $\mathcal{RO}(stmt, ctx, (a_\ell)_{\ell=1}^n, i, c'_i, ?)$ during its simulated grind. We then hope that the adversary $\mathcal{A}$ simply never queries the random oracle for a preimage containing any such burnt challenge. Because $\mathcal{A}$ does not know what random challenges the honest prover (or the simulator) tries unsuccessfully, it is unlikely that $\mathcal{A}$ would ever query the random oracle for one of those challenges.

To extract, as explained above, the extractor simply looks for a second accepting transcript $(a, c', r') \neq (a, c, r)$ in the list of random oracle queries the adversary makes.

Given the second transcript, the strong soundness property of the underlying Sigma protocol allows extraction of a witness.

Proof. Zero-knowledge Let $\Pi_{\text{Sigma}} \cdot \mathcal{S}$ be the Sigma protocol's simulator from Definition 3.2. We construct the simulator $\mathcal{S}$ as follows.

$\mathcal{S}.\text{Setup}(\mathcal{G}) \rightarrow pp$

Computed honestly as in Construction 3.23. $\mathcal{S}$ remembers pp and initializes a set $B = \emptyset$.

$\mathcal{S}.\mathcal{RO}(x) = y$

Behaves like a standard programmable random oracle (cf. Section 2.9).

$\mathcal{S}.\text{Sim}(stmt, ctx) \leftrightarrow \mathcal{A}$

For each $i \in \{1, \dots, n\}$,

- $\mathcal{S}$ picks a challenge $c \leftarrow \mathcal{C}$ and a random oracle image $v \leftarrow \{0, \dots, W-1\}$. If $v = 0$, it sets $c_i := c$. Otherwise, it adds c to B (“burns” the challenge) and repeats this step. $\P$ We are going to hope that $\mathcal{A}$ never queries $\mathcal{RO}(\cdot, \cdot, \cdot, \cdot, c, \cdot)$ for $c \in B$ because $\mathcal{S}$ does not know what preimage $(stmt, ctx, (a_\ell)_{\ell=1}^n, i, c, ?)$ should have been programmed to v . The set B itself is introduced here for explanatory purposes only.
- If the step above has been tried unsuccessfully m times, the simulator aborts and outputs $\pi = \perp$ as proof. $\P$ This mimics the unlikely failure case in which the honest prover just outputs the witness.
- Otherwise, $\mathcal{S}$ generates a transcript $(a_i, c_i, r_i) \leftarrow \Pi_{\text{Sigma}} \cdot \mathcal{S}(pp, stmt, c_i)$.

Finally, $\mathcal{S}$ programs the random oracle, setting $\mathcal{RO}(stmt, ctx, (a_\ell)_{\ell=1}^n, i, c_i, r_i) := 0$ for all i . It then sends $\pi = (a_i, c_i, r_i)_{i=1}^n$ to $\mathcal{A}$.

Let $\mathcal{A}$ be a ppt making at most $q(\lambda)$ many random oracle or proof/simulation queries (for some polynomial q). We now consider $|\Pr[\mathcal{A}^{\mathcal{RO}, \mathcal{O}.\text{Prove}^{\text{REAL}}}(\mathcal{G}, \text{Setup}(\mathcal{G})) = 1] - \Pr[\mathcal{A}^{\mathcal{S}.\mathcal{RO}, \mathcal{O}.\text{Prove}^{\text{SIM}}}(\mathcal{G}, \mathcal{S}.\text{Setup}(\mathcal{G})) = 1]|$.

As a first step, we modify the $\Pi_{\text{zk}}.\text{Prove}$ algorithm such that it outputs $\pi = \perp$ if it runs out of tries for some i instead of outputting the witness w . Let $\mathcal{O}.\text{Prove}'$ be like $\mathcal{O}.\text{Prove}^{\text{REAL}}$ except that it runs the modified version of Prove . The probability to need more than m tries to find a 0 hash image is $\left(\frac{W-1}{W}\right)^m = (1-1/W)^m = ((1-1/W)^W)^{m/W} \leq e^{-m/W} = e^{-\lambda}$. For every $\mathcal{O}.\text{Prove}^{\text{REAL}}$ query, this process of trying for small hash values is run at most n times. There are at most $q(\lambda)$ queries to $\mathcal{O}.\text{Prove}^{\text{REAL}}$. Hence the probability that during the experiment, Prove ever needs more than m tries is at most $q(\lambda) \cdot n(\lambda) \cdot e^{-\lambda}$, which is negligible. By the difference lemma [Sho04], we get $|\Pr[\mathcal{A}^{\mathcal{RO}, \mathcal{O}.\text{Prove}^{\text{REAL}}}(\mathcal{G}, \text{Setup}(\mathcal{G})) = 1] - \Pr[\mathcal{A}^{\mathcal{RO}, \mathcal{O}.\text{Prove}'}(\mathcal{G}, \text{Setup}(\mathcal{G})) = 1]| \leq q(\lambda) \cdot n(\lambda) \cdot e^{-\lambda}$.

We now compare observable effects of an honest prover with observable effects of the simulator. From the point of view of $\mathcal{A}$, the only difference between the honest

prover and the simulator is that the honest prover queries the random oracle for all the challenges he tries, whereas the simulator does not. The simulator *does* choose random hash values v , similar to how they are chosen by the random oracle for the honest prover, but the simulator does not actually program the random oracle with them. Intuitively, the simulator may sample v , but does not know what random oracle preimage the honest prover would request to receive v .

Let $c_1^*, \dots, c_k^*$ (where $k \leq q(\lambda) \cdot nm$) denote the sequence of challenges that the prover/simulator try (we can imagine they choose all of those independently at the beginning of the experiment). Let $v_1^*, \dots, v_k^*$ be the corresponding hash values (chosen by either the random oracle for the honest prover, or by the simulator when it chooses $v \leftarrow \{1, \dots, W\}$). Let $\text{fail}_{\text{query}}$ be the event that at some point, $\mathcal{A}$ queries $\mathcal{RO}(\cdot, \cdot, \cdot, \cdot, c_j^*, \cdot)$ for some j with $v_j^* \neq 0$. $\heartsuit \text{fail}_{\text{query}}$ means that $\mathcal{A}$ queries the random oracle with one of the unsuccessfully tried challenges, for which the simulator does not know what preimage to program. For the simulator, this corresponds to $\mathcal{A}$ querying a “burnt” challenge. Let $\text{fail}_{\text{program}}$ be the event that $\mathcal{A}$ queries $\mathcal{RO}(\cdot, \cdot, \cdot, \cdot, c_j^*, \cdot)$ for some j with $v_j^* = 0$ *before* the simulation oracle outputs a proof π containing c_j^* . $\heartsuit \text{fail}_{\text{program}}$ means that $\mathcal{A}$ queries the random oracle for a successfully tried challenge before the prover/simulator tries it. This may cause issues when the simulator attempts to program the random oracle for that challenge. Both events boil down to the event that $\mathcal{A}$ makes a hash query for a preimage containing a random challenge c from a set of random challenges that are independent of $\mathcal{A}$ ’s view. Since $\mathcal{A}$ makes at most $q(\lambda)$ many $\mathcal{RO}$ queries and the honest prover/simulator choose at most k challenges c_j^* , we have that $\Pr[\text{fail}_{\text{query}} \vee \text{fail}_{\text{program}}] \leq q(\lambda) \cdot k/|\mathcal{C}| \leq q(\lambda) \cdot q(\lambda)mn/|\mathcal{C}|$.

If $\text{fail}_{\text{program}}$ does not occur, then there is no issue when $\mathcal{S}$ programs the random oracle (i.e., the preimage has not been queried before programming). If $\text{fail}_{\text{query}}$ also does not occur, then there is *no* difference in the distribution of random oracle responses that $\mathcal{A}$ observes for the honest prover or the simulator (for this, we use that $\Pi_{\text{Sigma}}.\mathcal{S}$ outputs transcripts with the exact same distribution as the honest prover).

With the difference lemma, we upper-bound $|\Pr[\mathcal{A}^{\mathcal{RO}, \mathcal{O}. \text{Prove}'}(\mathcal{G}, \text{Setup}(\mathcal{G})) = 1] - \Pr[\mathcal{A}^{\mathcal{S}, \mathcal{RO}, \mathcal{O}. \text{Prove}^{\text{SIM}}}(\mathcal{G}, \mathcal{S}.\text{Setup}(\mathcal{G})) = 1]| \leq \Pr[\text{fail}_{\text{program}} \vee \text{fail}_{\text{query}}] \leq q^2(\lambda)nm/|\mathcal{C}|$. In summary, using triangle inequality, we get that $|\Pr[\mathcal{A}^{\mathcal{RO}, \mathcal{O}. \text{Prove}^{\text{REAL}}}(\mathcal{G}, \text{Setup}(\mathcal{G})) = 1] - \Pr[\mathcal{A}^{\mathcal{S}, \mathcal{RO}, \mathcal{O}. \text{Prove}^{\text{SIM}}}(\mathcal{G}, \mathcal{S}.\text{Setup}(\mathcal{G})) = 1]| \leq q(\lambda) \cdot n(\lambda) \cdot e^{-\lambda} + q^2(\lambda)n(\lambda)m(\lambda)/|\mathcal{C}|$, which is negligible (using that $|\mathcal{C}|$ is super-poly).

Simulation extractability: Let $\Pi_{\text{Sigma}}.\mathcal{E}$ be Sigma protocol’s extractor from Definition 3.6. We construct the extractor $\mathcal{E}$ as follows (we simply denote the verifier’s view τ as the proof π).

$\mathcal{E}(stmt, ctx, \pi, state_{\mathcal{S}}) \rightarrow w$

Checks if $(stmt, \pi) \in \mathcal{R}_{\mathcal{G}}$ and outputs π if that is the case. Otherwise, for $\pi = (a_i, c_i, r_i)_{i=1}^n$, $\mathcal{E}$ checks the random oracle table (in $state_{\mathcal{S}}$) for a query of the form $(stmt, ctx, (a_{\ell})_{\ell=1}^n, i, c'_i, r'_i)$ with $i \in \{1, \dots, n\}$, $(c'_i, r'_i) \neq (c_i, r_i)$, and $\Pi_{\text{Sigma}}.\text{Verify}(pp, stmt, a_i, c'_i, r'_i) = 1$. If no such query is found, $\mathcal{E}$ aborts and outputs fail_{404} . Otherwise, it computes $w = \Pi_{\text{Sigma}}.\mathcal{E}(pp, stmt, a_i, (c_i, r_i), (c'_i, r'_i))$. If $(stmt, w) \in \mathcal{R}_{\mathcal{G}}$, then $\mathcal{E}$ outputs w . Otherwise, it outputs $\text{fail}_{\text{extr}}$.

If neither fail_{404} nor $\text{fail}_{\text{extr}}$ are triggered, then $\mathcal{E}$ outputs a valid witness. We now analyze the probabilities for those events.

First, note that Definition 3.22 guarantees that $\mathcal{E}(\text{stmt}, \text{ctx}, \cdot, \cdot)$ is only ever called for $(\text{stmt}, \text{ctx})$ if there has not been a $\mathcal{S}.\text{Sim}(\text{stmt}, \text{ctx})$ call, hence the random oracle has *not* been programmed for preimages of the form $(\text{stmt}, \text{ctx}, \dots)$ (and hence all $\mathcal{RO}$ queries of that form come from $\mathcal{A}$ or Verify). As a consequence, the only way to trigger fail_{404} is for $\mathcal{A}$ to set up a prefix $\text{stmt}, \text{ctx}, (a_\ell)_{\ell=1}^n$ and query (or have Verify query by outputting a proof) the random oracle $\mathcal{RO}(\text{stmt}, \text{ctx}, (a_\ell)_{\ell=1}^n, i, c_i, r_i)$ *exactly once* for each $i \in \{1, \dots, n\}$ (where we only count queries with accepting transcripts, i.e., $\text{Verify}(pp, \text{stmt}, a_i, c_i, r_i) = 1$). Because random oracle hashes are uniformly random, the probability for all n hash values to be 0 is $1/W^n$. Because $\mathcal{A}$ makes at most $q(\lambda)$ random oracle queries, there are at most $q(\lambda)$ relevant prefixes for which $\mathcal{A}$ can try this, hence $\Pr[\text{fail}_{404}] \leq q(\lambda)/W^n = q(\lambda) \cdot W^{-\lambda/\log(W)} = q(\lambda) \cdot 2^{-\lambda}$.

$\Pr[\text{fail}_{\text{extr}}]$ can be analyzed by reduction to the strong special soundness property (Definition 3.6). For this, let $\mathcal{B}(\mathcal{G}, pp)$ run $\mathcal{A}^{\mathcal{S}.\mathcal{RO}, \mathcal{O}.\text{Simul}, \mathcal{O}.\text{Extract}}(\mathcal{G}, \mathcal{S}.\text{Setup}(\mathcal{G}))$ with the change that (1) $\mathcal{S}.\text{Setup}$ outputs pp from $\mathcal{B}$'s input and (2) if $\text{fail}_{\text{extr}}$ occurs, $\mathcal{B}$ stops and outputs $(\text{stmt}, a_i, (c_i, r_i), (c'_i, r'_i))$ (those variables are as in the algorithm description of $\mathcal{E}$). It is easy to see that whenever $\text{fail}_{\text{extr}}$ happens, $\mathcal{B}$ outputs two valid transcripts with $(c_i, r_i) \neq (c'_i, r'_i)$ but $\Pi_{\text{Sigma}}.\mathcal{E}$ cannot compute a witness. $\mathcal{B}$ runs in polynomial time so the probability for this to happen is negligible. Hence $\Pr[\text{fail}_{\text{extr}}] \leq \text{negl}(\lambda)$.

Overall, $\Pr[\text{ExtractionError}] \leq \Pr[\text{fail}_{404}] + \Pr[\text{fail}_{\text{extr}}] \leq q(\lambda) \cdot 2^{-\lambda} + \text{negl}(\lambda)$ is negligible.

Runtime: Clearly, since $\mathcal{A}$ and all other algorithms involved run in polynomial time, also $\mathcal{S}$ and $\mathcal{E}$ run in polynomial time. $\square$

3.5 The issues with rewinding

Fischlin's transform (Construction 3.23) provides one way for the extractor to retrieve the two correlated transcripts $(a, c, r) \neq (a, c', r')$ needed by the Sigma protocol extractor. It forces the prover to query the random oracle with such transcripts. In Section 3.1, we have discussed another way: rewinding. Consider just running the Sigma protocol as-is as a proof of knowledge. This does not provide any reasonable notion of zero-knowledge, but for the discussion below, we are only interested in witness extraction. The discussion also applies to protocols that do have reasonable zero-knowledge guarantees (e.g., [Dam00]), as well as the influential Fiat-Shamir heuristic [FS86].

To get a second transcript, the rewinding extractor runs a copy of the prover and replays its view (input, randomness, prior messages from other protocols, and oracle replies), causing the prover to send the same statement stmt and Sigma protocol announcement a . The extractor then feeds a different challenge c' to the prover, and runs the prover until it sends the response r' . If the response is accepting, the extractor has found its second transcript and can run the Sigma protocol extractor to retrieve the witness. This strategy is elegant and has enabled many constructions of proofs of knowledge, both early ones and recent ones. Hence a reasonable question is: Why can we not use

this strategy for our proof systems in anonymous credentials? There are several issues that keep rewinding from being applicable to Definition 3.22.

One issue with this is that the probability for the second transcript to be accepting is not generally overwhelming (Definition 2.5) [PS96; BN06]. As a result, this extractor fails with non-negligible probability. However, there are ways to make the extractor work with negligible failure probability (running in expected polynomial time), by essentially having the extractor try multiple challenges until it finds one that fits.

The more severe problem with this approach is the running time of this extractor when the Sigma protocol is run in a setting with concurrency and adaptive extraction, which we discuss in the remainder of this section.

The sequential setting. First, let us illustrate the way extraction is set up in the sequential setting. Let $\mathcal{A}$ be an adversary against the sequential adaptive proof of knowledge property (which we did not define formally, but effectively, $\mathcal{A}$ wins if it convinces the honest verifier but the extractor $\mathcal{E}$ fails to provide $\mathcal{A}$ with a witness). Consider Figure 3.9. In the sequential setting, protocol execution by $\mathcal{A}$ necessarily obeys a strict order: one Sigma protocol execution finishes before the next one is started.

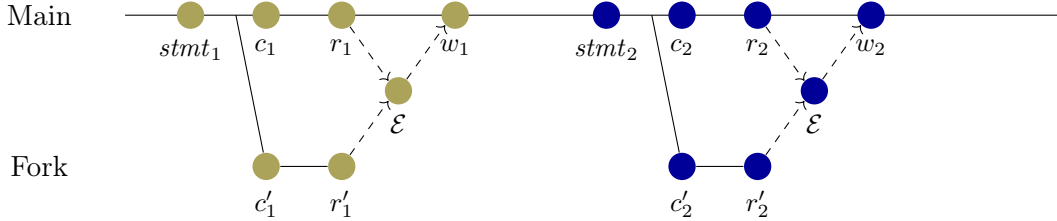


Figure 3.9: An illustration of (adaptive) extraction in the sequential setting.

First, $\mathcal{A}$ announces the statement $stmt_1$ and sends the first message a_1 (which we omit from Figure 3.9 but which happens at the same time as $stmt_1$ is announced). The (honest) verifier then sends a challenge c_1 . $\mathcal{A}$ responds with r_1 . Then $\mathcal{A}$ is supposed to receive a witness w_1 . For this, the extractor essentially starts a new execution fork of $\mathcal{A}$ in which it sends some other challenge c'_1 to $\mathcal{A}$ instead of c_1 . On that fork, $\mathcal{A}$ responds with some r'_1 , which the extractor $\mathcal{E}$ combines with r_1 to compute and return a witness to the main execution of $\mathcal{A}$. Because of sequentiality, only after this process is fully completed, the next protocol for $stmt_2$ may start, repeating this process exactly the same way.

The concurrent setting. Now consider using such a rewinding strategy in a concurrent setting, as depicted in Figure 3.10. In the concurrent setting, the adversary can interleave protocol executions. In particular, $\mathcal{A}$ may decide to send the response r_1 to the first protocol's execution only after seeing the witness w_2 of the second protocol (this is a common counterexample [SG02] for rewinding issues). This is depicted in the main execution line in Figure 3.10.

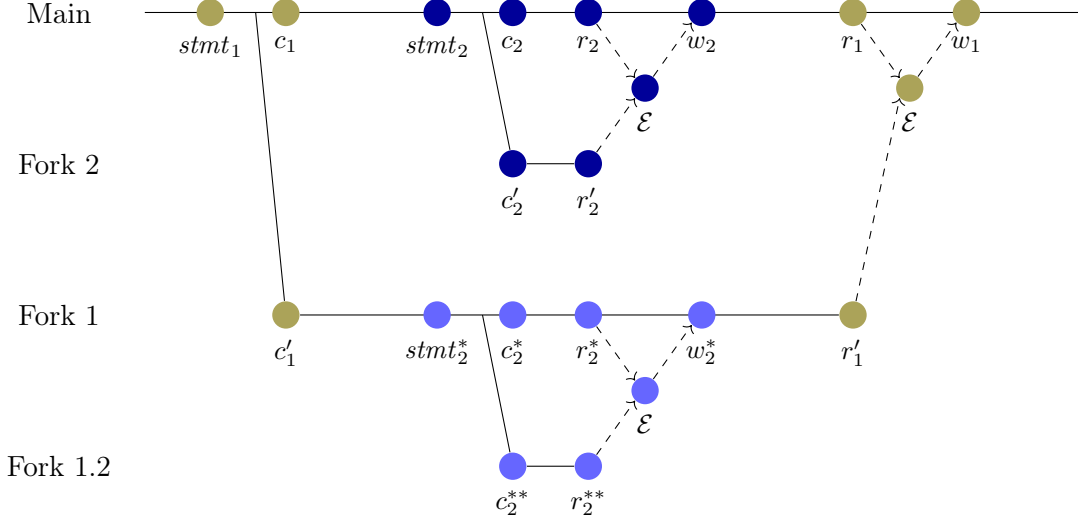


Figure 3.10: An illustration of forking issues in a concurrent setting with adaptive extraction.

For the second protocol ($stmt_2$), extraction works as usual. To extract for the first protocol ($stmt_1$), however, something interesting happens. We again fork the execution of $\mathcal{A}$ right before sending the challenge c_1 in order to send a different challenge c'_1 . Our goal is again to observe $\mathcal{A}$'s response r'_1 to c'_1 . However, before getting to the point in the fork's execution where $\mathcal{A}$ outputs r'_1 , the adversary $\mathcal{A}$ starts a second protocol execution (for $stmt_2^*$, which can be different from $stmt_2$ on the main execution line because $\mathcal{A}$ can choose it depending on the challenge of the first protocol). Then $\mathcal{A}$ expects to be provided with the witness w_2^* for $stmt_2^*$ before releasing the response r'_1 that we are interested in. In order to compute w_2^* , we need to again run the extraction process. This means we need to fork execution of Fork 1 again, right before sending the challenge c_2^* to send a different challenge c_2^{**} . From the corresponding responses, we can finally compute the witness w_2^* and then receive r'_1 to compute w_1 .

This kind of recursive rewinding causes a blow-up of extraction runtime. Note that in this setting, there are now twice as many “last statements” ($stmt_i$ with largest i) in the execution than before (namely $stmt_2, stmt_2^*$). This generalizes: If the adversary needs to see w_i before responding with r_{i-1} , they make $stmt_i$ dependent on challenge c_{i-1} , and they query for n extractions (on the main execution line), then the extraction process involves 2^{n-1} different last statements $stmt_n$ across the forks. Figure 3.11 illustrates this by adding an additional extraction for $stmt_0$ to the situation in Figure 3.10. It hence showcases the blow-up with $n = 3$, causing 4 extractions for the last statements ($stmt_2, stmt_2^*, stmt_2^{**}, stmt_2^{***}$) across the forks, as opposed to 2 extractions for $n = 2$.

The concurrent but non-adaptive setting. The example causing runtime blow-up in the concurrent setting above crucially relies on the fact that $\mathcal{A}$ expects to see witness w_i before sending response r_{i-1} . In Section 3.3.3, we call the corresponding property of proofs of knowledge *adaptive extractability*, which is the idea that the adversary gets

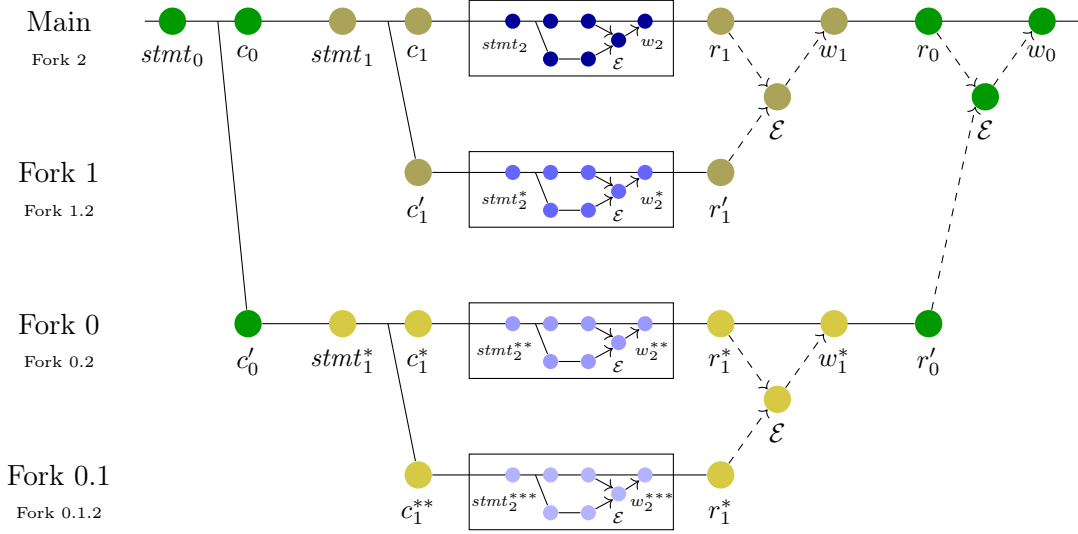


Figure 3.11: An illustration of forking issues in the adaptive concurrent setting, larger example to illustrate runtime explosion.

to see witnesses during its execution. This is what we have defined in Definition 3.22: The extraction oracles return the extracted witness to $\mathcal{A}$ immediately. We could imagine a concurrent *non-adaptive* definition with rewinding, where witnesses are computed by the oracles, but not given to $\mathcal{A}$ until the end of its execution (when no further protocol execution may happen). This notion can be appropriate in security proofs for constructions in which witnesses are not required by the reduction algorithm to execute the underlying adversary. Not giving $\mathcal{A}$ the witness directly after extraction, as depicted in Figure 3.12, solves the runtime issues of the concurrent setting. Recursive forking is not necessary anymore because the execution of forks does not require witness extraction in the non-adaptive setting.

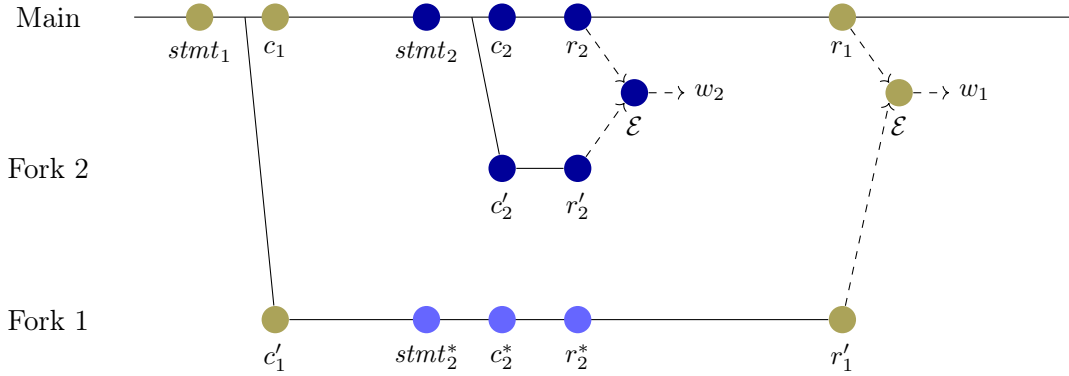


Figure 3.12: An illustration of extraction in a concurrent but non-adaptive setting.

The fact that the runtime issue in the concurrent setting vanishes when removing adap-

tive extraction suggests that the main issue with extraction-by-rewinding is not necessarily the nature of *concurrent proof execution*, but the nature of *adaptive extractability*.

Overall, the runtime issue only occurs when combining concurrent proof execution *and* adaptive extractability. If adaptive extractability is not needed, Damgård’s technique [Dam00] or even Fiat-Shamir [FS86] may be applicable.

An exact characterization of applicability of extraction-by-rewinding is outside the scope of this discussion. In this thesis, we generally require adaptive extractability, which is because it is required by our signature schemes for signing a committed value (Definition 5.5). We discuss this requirement in Chapter 7. The adaptive extractability requirement rules out using rewinding in a concurrent setting, as discussed above. To work around this, we opt for Fischlin’s transform (Construction 3.23) instead. An alternative may be to restrict anonymous credential definitions to sequential execution, but this is a significant restriction that is unlikely to reflect real world deployments.

3.6 Overview — bringing it all together

The goal of this chapter is to thoroughly define the requirements for the proof systems we are going to use in this thesis, and to instruct the reader how to instantiate the proof systems securely. This serves as the basis upon which the constructions in this thesis crucially rely, and we want to avoid leaving make-or-break details up to folklore.

In this section, we establish a convenient notation to mark execution of a proof. This notation hides the complexity behind the proof system within larger constructions. It is often referred to as *Camenisch-Stadler notation* [CS97].

Definition 3.25. Let Π_{zk} be a proof system with relation $\mathcal{R}_{\mathcal{G}}$ and public parameters pp . Let $\mathcal{A}$ and $\mathcal{B}$ be interactive algorithms whose code we are defining. Let $vars$ be a list of variable values available to both $\mathcal{A}$ and $\mathcal{B}$ in the current programming context. Let φ_{vars} be a predicate. The pseudocode notation

$$\mathcal{A} \text{ proves ZKPoK } [(w); \varphi_{vars}(w)](ctx)$$

is equivalent to:

- $\mathcal{A}$ runs $\text{Prove}(pp, stmt, ctx; w)$, interacting with $\mathcal{B}$ running $\text{Verify}(pp, stmt, ctx) \rightarrow b$, where $stmt = \varphi_{vars}$. φ $stmt$ is *supposed* to be the same for both $\mathcal{A}$ and $\mathcal{B}$, but formally, of course, $\mathcal{A}$ and $\mathcal{B}$ use their respective local variable values $vars$ to derive it.
- $\mathcal{B}$ asserts $b = 1$. φ This means that if the proof fails, $\mathcal{B}$ aborts the current method that has called Verify and returns $\perp$.

Public parameters pp are assumed to be available from the context. Consider several such expressions $\text{ZKPoK } [(w_i); \varphi_{i, vars_i}](\cdot)$. We say that $\mathcal{R}_{\mathcal{G}}$ is *compatible* with those

expressions if for all i and all (programmatically reachable) values of $vars_i$,

$$(\varphi_{i,vars_i}, w_i) \in \mathcal{R}_G \Leftrightarrow \varphi_{i,vars_i}(w_i)$$

For example, consider the following coin-flipping protocol between $\mathcal{A}(\mathcal{G}, g, h, m)$ and $\mathcal{B}(\mathcal{G}, g, h, m)$:

- $\mathcal{A}$ chooses random $z, r \leftarrow \mathbb{Z}_p$ and computes the commitment $C = g^r \cdot h^z$. It sends C to $\mathcal{B}$.
- $\mathcal{A}$ proves

$$\text{ZKPoK} \left[(z, r); C = g^r \cdot h^z \right] (ctx = C).$$

- $\mathcal{B}$ chooses $z' \leftarrow \mathbb{Z}_p$ and sends it to $\mathcal{A}$.
- $\mathcal{A}$ sends z, r to $\mathcal{B}$ and returns $z + z'$.
- $\mathcal{B}$ returns $\perp$ if $g^r \cdot h^z \neq C$. Otherwise, $\mathcal{B}$ returns $z + z'$.

In this example, $vars = (g, h, C)$ (where g, h is common input to both $\mathcal{A}$ and $\mathcal{B}$; and the commitment C is computed by $\mathcal{A}$ and sent to $\mathcal{B}$, which makes it available to both). The proven statement is $\varphi_{vars}(z, r) \Leftrightarrow C = g^r \cdot h^z$.

To implement this example, one needs to come up with a proof system with a compatible relation. This should be done in the following steps.

Design a Sigma protocol for the relation. First, design a Sigma protocol Π_{Sigma} for a relation $\mathcal{R}_G$ compatible with the $\text{ZKPoK} \left[(\cdot); \cdot \right] (\cdot)$ expression(s). We usually do not explicitly detail this step and expect the reader to be able to execute it by themselves. However, to facilitate this, we explain how to design Sigma protocols for certain important statements (such as proving knowledge of a signature).

For the example above, designing Π_{Sigma} is very simple: Prover and verifier simply run Schnorr $\left[(z, r); C = g^z \cdot h^r \right]$ (as discussed in Definition 3.8). In general, one may have to design a slightly more complicated Sigma protocol, using the send-first mechanism (Section 3.2.2), standard tricks (Section 3.2.4), or the AND/OR composition (Section 3.2.5). With the explanations given in this chapter, designing Π_{Sigma} generally will not be a difficult task. For an implementation in code, we refer to the notion of Schnorr fragments [BEHF21], which enables dynamic composition of complex Schnorr-style protocols.

Upgrade the Sigma protocol to a secure proof system. In the second step, one needs to upgrade Π_{Sigma} to a proper proof system satisfying our security definition (Definition 3.22). To achieve this, we use the randomized Fischlin transform (Construction 3.23). In order to be able to apply randomized Fischlin, the Sigma protocol designed in the first step needs to have strong special soundness. Construction 3.15 describes how the basic Schnorr protocol can be upgraded to provide strong special soundness. If the protocol is more complicated than that, note that the send-first mechanism (Section 3.2.2) and the AND/OR composition (Section 3.2.5) preserve strong special soundness.

Chapter 4

Traditional anonymous credentials

On the Internet, nobody knows you're
a dog

Peter Steiner

A traditional anonymous credential system, as first envisioned by Chaum [Cha84; Cha85], is a tool meant for user authentication. The grand vision is for users to establish *pseudonyms* as their identity placeholder, and interact with organizations without revealing their real identity. Anonymous credentials can be used to establish trustworthiness or access rights for a user. In an anonymous credential system, there are three roles: *issuers*, *users*, and *verifiers* (cf. Figure 4.1). Issuers are entities that manage user attributes or access rights by issuing *credentials* to users. Verifiers generally guard some service or resource, enforcing access control. In order to access the service/resource, a user proves, to a verifier, that he is in possession of a credential. If the verifier is satisfied with the proof, she grants access. Users want *anonymity*, i.e., while they receive a credential or authenticate with it, they do not want to reveal more about their identity than strictly necessary. Verifiers and issuers (and, to some extent, honest users, cf. Section 4.2.3) want to enforce proper access control, i.e., only eligible users should be able to access services/resources.

While the term is now used more broadly (as we will discuss in Chapter 5), traditionally, anonymous credential systems have been very much centered around user identities. The idea is that user identities are always *present*, but *hidden*. Following this line of thinking, the following concepts have been made an integral part of traditional anonymous credential systems: (1) Every user has a unique secret identity (which we can think of as a random bit string). (2) Whenever the user interacts with another party, he does so under a *pseudonym*. We discuss pseudonyms in detail in Section 4.1. For now, pseudonyms are cryptographically *related* the user's identity, but do not reveal it. (3) Credentials are specific to the user's identity (even though that identity is hidden from the issuer and verifier). In this sense, traditional credentials are more like passports or flight tickets (which are bound to your specific identity) than a carnival ride ticket (which is also used for access control, but typically is not bound to an identity).

With the identity theme, there is often another role in traditional anonymous credential

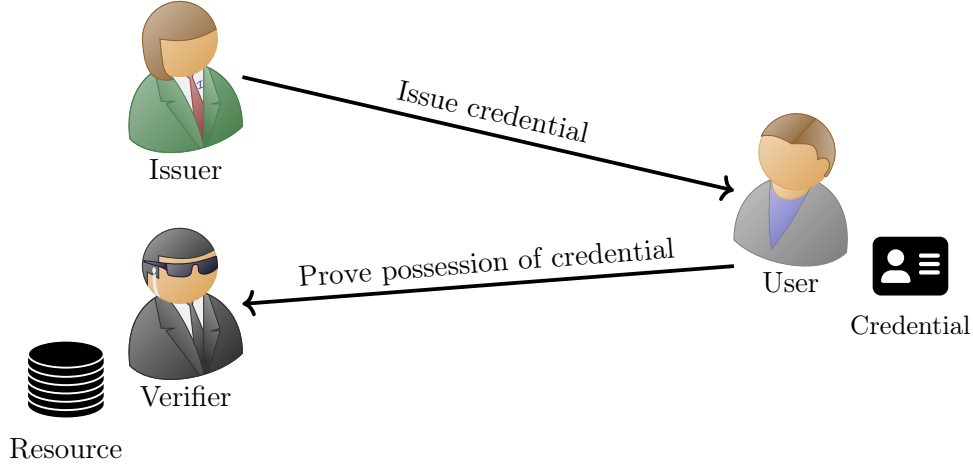


Figure 4.1: Overview of the most basic roles in anonymous credential systems.

systems: the *inspector* (cf. Figure 4.2). Pseudonyms generally hide the user’s real identity. In case of misuse or, say, for legal prosecution, the inspector can use a special trapdoor to reveal the user’s identity behind a pseudonym.

Finally, early on, the notion of *attributes* has been incorporated into traditional anonymous credentials [CE86; CL01]. Every credential encodes an ordered list of *attributes*, which we typically imagine encoding information about a person (e.g., name, age, citizenship, position within an organization), or about the credential (e.g., expiration date). When authenticating with a verifier, some statement about the attributes can be made an access requirement (e.g., “age ≥ 18 ”, or “position $\in \{\text{IT-support, Manager}\}$ ”). In that case, the verifier only learns that the statement about the user’s attributes is true, but nothing more about the attributes (protecting the user’s privacy).

Take, for example, the following scenario, showcasing the power of traditional anonymous credentials. User U holds a passport credential with attributes $\vec{attr} = (\text{name}, \text{address}, \text{birth date}, \text{photo}) = (\text{Max Sampleson}, 24 \text{ Main Street}, 1985\text{-}01\text{-}02, \text{[photo]})$. This credential has been issued to U by the government. Say U gets tested for an airborne viral disease. For this, U generates a new pseudonym nym_{Dr} and approaches a doctor. The doctor needs to know (1) the user’s **photo** and (2) that the user is at least 18 years old. U , under pseudonym nym_{Dr} , reveals his photo [photo] to the doctor and proves possession of a credential with attributes $\vec{attr} = (\text{name}, \text{address}, \text{birth date}, \text{photo})$ with the predicate $\phi(\vec{attr}) \equiv (\text{photo} = \text{[photo]}) \wedge (\text{birth date} \leq \text{today} - 18y)$. In this process, the doctor (playing the credential verifier role) only learns about the attributes $\vec{attr}$ what the predicate ϕ reveals about them (namely, the validity of the revealed photo [photo] and an upper bound on the birthdate). She does not learn anything about the values of **name** and **address**, and she does not learn the exact birthdate.

After testing the user U negative for the disease, the doctor (now playing the role of credential issuer) issues a credential with attributes $\vec{attr} = (\text{type}, \text{date}) = (\text{negative test}, 2021\text{-}06\text{-}07)$ to the pseudonym nym_{Dr} .

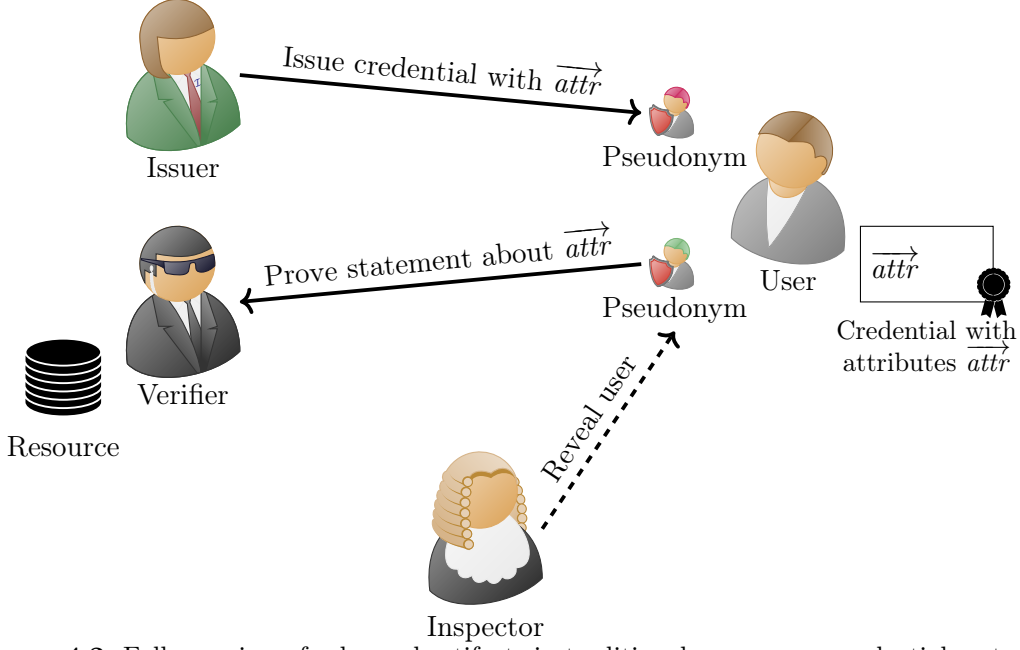


Figure 4.2: Full overview of roles and artifacts in traditional anonymous credential systems.

Afterward, U generates a new pseudonym nym_{cinema} and approaches a cinema. For health concerns, the cinema implements a policy where customers must have tested negative or have been vaccinated. So U , under the pseudonym nym_{cinema} , proves possession of a credential with attributes $\vec{attr} = (\text{type}, \text{date})$ that fulfills the predicate $\phi(\vec{attr}) \equiv (\text{type} = \text{negative test} \wedge \text{date} = \text{today}) \vee (\text{type} = \text{vaccinated} \wedge \text{date} \geq \text{today} - 1y)$. Even if the cinema and the doctor cooperate, they will not be able to find out that nym_{Dr} and nym_{cinema} belong to the same user and the cinema will not be able to learn whether the user is getting access because he has been tested or because he has been vaccinated.

This example showcases that it can make sense to issue a credential to someone without knowing their full identity: The doctor does not even know the name of U . This motivates the requirement that credentials can be issued without revealing the user's full identity. The example also showcases the identity-binding aspect of pseudonyms: The doctor deliberately issues the health credential to the same pseudonym nym_{Dr} from which she has seen the passport credential. Because of this, the health credential is bound to *the same* (hidden) identity as the passport credential is. We discuss this aspect, called non-transferability, further in Section 4.4.

In this chapter, we take a closer look at traditional anonymous credential systems. How can we define security? How are anonymous credential systems generally constructed? We also discuss some literature on anonymous credentials.

Contribution By the nature of this chapter, which is essentially a discussion of a certain understanding of anonymous credentials, we lean heavily on prior work. Because there are no widely established standard definitions or security notions for anonymous

credentials (in contrast, to, say, public-key encryption), Section 4.2 contains our own careful formalization of traditional anonymous credentials and their security. Before this, we discuss pseudonyms Section 4.1. We discuss the standard template for constructing anonymous credentials in Section 4.3, as well as definitional aspects, especially regarding non-transferability of credentials in Section 4.4. We briefly summarize the history of traditional anonymous credentials in Section 4.5.

4.1 Pseudonyms

We start our exploration of anonymous credentials by identifying a building block: pseudonym systems. As discussed above, traditional anonymous credential systems are all about identity: Users have a secret identity, which we will call the user public key upk . The user's identity stays hidden, the user instead can generate any number of pseudonyms nym . Pseudonyms hide the user's upk , but are still bound to it. Often, pseudonyms are equated with commitments (Definition 2.15), which intuitively have the properties we need: They hide a value (like upk or the corresponding secret usk), and they are binding, i.e., after committing to an identity, there is no way to open the same commitment to a different identity. We note that in the literature, a *pseudonym system* is sometimes understood to be a full anonymous credential system [LRSW99]. We do not use the term this way here.

In this section, we discuss pseudonym systems more broadly. We will distill the properties we need from pseudonym systems. As mentioned above, we introduce a designated trusted entity, the *inspector*. The inspector can use their trapdoor key ntd to inspect a pseudonym, revealing the hidden identity. This is not the only choice. There are pseudonym systems without inspection. However, the inspector will make it possible to define non-transferability properties for anonymous credentials, as we will discuss in Section 4.4.

Definition 4.1 (Pseudonym system). A pseudonym system consists of the following ppt algorithms:

$\text{Setup}(\mathcal{G}) \rightarrow (pp, ntd)$

Generates public parameters pp and a trapdoor ntd .

$\text{KeyGen}(pp) \rightarrow (upk, usk)$

Generates a user's public key upk and secret key usk .

$\text{GenNym}(pp, usk) \rightarrow (nym, nsk)$

Generates a pseudonym nym and a corresponding pseudonym secret nsk . ♡ Can be run arbitrarily often by a user to generate multiple pseudonyms.

$\text{VerifyNym}(pp, usk, nym, nsk) = b$

A deterministic algorithm that outputs a bit b . ♡ It can be run to check that nym is a valid pseudonym for usk .

$\text{InspectNym}(pp, nym, ntd) = upk$

A deterministic algorithm that outputs a user's public key upk . ? Run by the inspector to reveal the user's identity behind a pseudonym.

A pseudonym system is *correct*, i.e., for all $\lambda \in \mathbb{N}$, $\mathcal{G} \in [\text{GenSetting}(1^\lambda)]$, $(pp, ntd) \in [\text{Setup}(\mathcal{G})]$, $(upk, usk) \in [\text{KeyGen}(pp)]$, $(nym, nsk) \in [\text{GenNym}(pp, usk)]$, we have that

$$\begin{aligned} \text{VerifyNym}(pp, usk, nym, nsk) &= 1 \\ \text{and } \text{InspectNym}(pp, nym, ntd) &= upk \end{aligned}$$

The function of VerifyNym is essentially to establish a link between usk and nym . Because only the user knows the values usk, nsk it expects as input, VerifyNym might seem somewhat useless. VerifyNym will later be used by the user within a zero-knowledge proof of knowledge, which allows him to convince others that nym is his pseudonym without revealing the secret usk, nsk .

The choice to have KeyGen output a public key upk for the user is somewhat arbitrary: upk is not input to any further algorithms, so strictly speaking, the user could just throw it away. The only place upk reappears is as output for InspectNym , which, at least for honestly generated pseudonyms, outputs the user's upk . We made this choice because it is convenient, in future definitions, to be able to point to the value that an honest user's pseudonyms will open to.

In contrast to our definition, pseudonyms in the literature (and in Chaum's original vision) can often be *scoped* [Cha84; Cha85], i.e., a pseudonym can be bound to some public string ctx such that every user has exactly one (valid) pseudonym per ctx . This allows applications to enforce uniqueness of users. Because this can be achieved through other means as well, we do not consider scoped pseudonyms here (but all constructions in this thesis that build on pseudonym systems, such as Construction 5.14, would also work with scoped pseudonyms, if their syntax were appropriately adapted).

Regarding security of pseudonym systems, we want that (1) pseudonyms are unlinkable (for privacy), (2) pseudonyms are properly bound to a (hidden) identity, and (3) one cannot impersonate pseudonymous users.

We start with unlinkability. Intuitively, for a new pseudonym, it should be hard to tell whether it belongs to user 0 or user 1, even if the adversary has already seen pseudonyms from both users. The adversary's task is to distinguish an oracle generating pseudonyms for user 0 from one generating pseudonyms of user 1 (without the inspector trapdoor ntd).

Definition 4.2 (Pseudonym unlinkability). For $b \in \{0, 1\}$, let $\mathcal{O}.\text{GenNym}_b()$ be an oracle that runs $(nym, nsk) \leftarrow \text{GenNym}(pp, usk_b)$ and outputs nym . A pseudonym system Π_{nym} has *unlinkable pseudonyms* if for all ppt $\mathcal{A}$, there exists a negligible function negl such

that for all λ ,

$$\left| \Pr \left[\mathcal{A}^{\mathcal{O}.\text{GenNym}_0, \mathcal{O}.\text{GenNym}_1, \mathcal{O}.\text{GenNym}_0}(\mathcal{G}, pp, upk_0, upk_1) = 1 \right] - \Pr \left[\mathcal{A}^{\mathcal{O}.\text{GenNym}_0, \mathcal{O}.\text{GenNym}_1, \mathcal{O}.\text{GenNym}_1}(\mathcal{G}, pp, upk_0, upk_1) = 1 \right] \right| \leq \text{negl}(\lambda)$$

where $\mathcal{G} \leftarrow \text{GenSetting}(1^\lambda)$, $(pp, ntd) \leftarrow \text{Setup}(\mathcal{G})$, and $(upk_0, usk_0), (upk_1, usk_1) \leftarrow \text{KeyGen}(pp)$.

Next, we want pseudonyms to contain a *unique* identity. This is simple in our case: We can say that the identity contained in nym is uniquely defined by the output of $\text{InspectNym}(pp, nym, ntd)$, which is deterministic and hence enforces that every pseudonym has a unique identity behind it. In addition to that, we require some sort of consistency regarding the traced identity. We want that if two pseudonyms belong to the same usk according to VerifyNym , then the upk output by InspectNym must be the same for both pseudonyms.

Definition 4.3 (Pseudonym binding). A pseudonym system Π_{nym} has *binding pseudonyms* if for all $\lambda \in \mathbb{N}$, $\mathcal{G} \in [\text{GenSetting}(1^\lambda)]$, $(pp, ntd) \in [\text{Setup}(\mathcal{G})]$, and all usk, nym_b, nsk_b with $\forall b \in \{0, 1\} : \text{VerifyNym}(pp, usk, nym_b, nsk_b) = 1$,

$$\text{InspectNym}(pp, nym_0, ntd) = \text{InspectNym}(pp, nym_1, ntd)$$

This definition can be understood similarly to collision-resistance for hash functions (Definition 2.12): There are no “collisions” for pseudonyms, in the sense that whenever inspection yields two different $upk_0 \neq upk_1$, then the underlying usk_0, usk_1 hidden within nym_0, nym_1 according to InspectNym must also have $usk_0 \neq usk_1$.

Consider an honest user with keys (upk, usk) and one of his pseudonyms nym . The correctness property requires that that user’s pseudonyms trace to upk . Hence binding pseudonyms imply that nobody can produce values (usk', nym', nsk') such that $\text{VerifyNym}(pp, usk', nym', nsk')$ and nym' traces to the honest user’s upk , but $usk' \neq usk$.

Finally, users do not want to be impersonated, even while they act under pseudonyms. So intuitively, it should be difficult for anyone to use a pseudonym that the inspector traces to an honest user’s upk . With “using” a pseudonym nym , we mean producing usk, nsk that are accepted by VerifyNym . This is motivated by the way the pseudonym system building block will be used in larger constructions. The way users will establish ownership of a pseudonym nym will be by proving (in zero-knowledge) that they know usk, nsk that are accepted by VerifyNym .

Definition 4.4 (Pseudonym impersonation resistance). Let $\mathcal{O}.\text{GenNym}$ be an oracle that runs $(nym, nsk) \leftarrow \text{GenNym}(pp, usk)$ and outputs nym . A pseudonym system is *resistant*

to impersonation if for all ppt $\mathcal{A}$,

$$\Pr \left[\begin{array}{l} \text{VerifyNym}(pp, usk', nym', nsk') = 1 \\ \wedge \text{InspectNym}(pp, nym', ntd) = upk \end{array} \middle| \begin{array}{l} \mathcal{G} \leftarrow \text{GenSetting}(1^\lambda) \\ (pp, ntd) \leftarrow \text{Setup}(\mathcal{G}) \\ (upk, usk) \leftarrow \text{KeyGen}(pp) \\ \swarrow \mathcal{A}^{\mathcal{O}.\text{GenNym}}(\mathcal{G}, pp, ntd, upk) \\ (usk', nym', nsk') \end{array} \right]$$

is negligible.

Because honest users' pseudonyms nym trace to their upk , Definition 4.4 also implies that it is hard to find usk, nsk that are accepted by $\text{VerifyNym}(pp, usk, nym, nsk)$ for an honest user's pseudonym nym . This means that malicious users cannot “capture” an honest user's pseudonym, in the sense that they cannot produce usk, nsk attesting ownership over the honest user's pseudonym.

Pseudonym systems can be constructed quite easily. As mentioned above, they are usually [LRSW99] constructed from commitments. Because we want the inspection functionality, we construct them from encryption schemes (which can be seen as commitments), such as ElGamal encryption (Construction 2.23). The idea is that $usk \leftarrow \mathbb{Z}_p$ is a random exponent, $upk = g^{usk}$, and a pseudonym is an encryption of upk .

Construction 4.5 (ElGamal pseudonyms). The ElGamal pseudonym system works as follows.

$\text{Setup}(\mathcal{G}) \rightarrow (pp, ntd)$

Chooses $g \leftarrow \mathbb{G}_1 \setminus \{1\}$ and a trapdoor $ntd \leftarrow \mathbb{Z}_p$. It sets $h = g^{ntd}$ and $pp = (g, h)$. It outputs (pp, ntd) .

$\text{KeyGen}(pp) \rightarrow (upk, usk)$

Chooses $usk \leftarrow \mathbb{Z}_p$ and sets $upk = g^{usk}$. It outputs (upk, usk) .

$\text{GenNym}(pp, usk) \rightarrow (nym, nsk)$

Chooses $nsk \leftarrow \mathbb{Z}_p$ and sets $nym = (g^{usk} \cdot h^{nsk}, g^{nsk})$. It outputs (nym, nsk) .

$\text{VerifyNym}(pp, usk, nym, nsk) = b$

Outputs $b = 1$ iff $nym = (g^{usk} \cdot h^{nsk}, g^{nsk})$.

$\text{InspectNym}(pp, nym, ntd) = upk$

For $nym = (c_1, c_2)$, it outputs $upk = c_1 \cdot c_2^{-ntd}$.

Theorem 4.6 (ElGamal pseudonyms are secure). *If the DDH assumption (Definition 2.7) (for $\mathbb{G}_1$) holds, then Construction 4.5 has unlinkable (Definition 4.2) and binding (Definition 4.3) pseudonyms, and it is resistant to impersonation (Definition 4.4).*

The proof is an easy exercise. Impersonation resistance is reduced to the discrete logarithm assumption (Definition 2.6), which is implied by the DDH assumption. A successful

impersonator $\mathcal{A}$ must be able to compute the discrete logarithm usk of upk .

4.2 Formally defining traditional anonymous credential systems and their requirements

In this section, we formally define traditional anonymous credential systems. We first define its syntax and discuss how to use it, then we define security properties.

4.2.1 Syntax and usage

Armed with our understanding of pseudonym systems, we are now ready to define traditional anonymous credential systems, where the first four algorithms are essentially (syntactically) the same as for pseudonym systems. Additionally, an anonymous credential system has other protocols to deal with issuing and presenting credentials. For an overview of anonymous credential systems and its roles, please refer to the beginning of Chapter 4.

Definition 4.7 (Traditional ACS). A traditional anonymous credential system Π_{acs} with attribute space $\mathbb{A} = \mathbb{A}_{\mathcal{G}}$ and predicate space $\Phi = \Phi_{\mathcal{G}}$ consists of the following ppt algorithms and protocols.

$\text{Setup}(\mathcal{G}) \rightarrow (pp, ntd)$

Generates public parameters pp and a pseudonym trapdoor ntd . ! The pseudonym trapdoor is meant for the inspector.

$\text{UserKeyGen}(pp) \rightarrow (upk, usk)$

Generates a user public key upk and a corresponding secret key usk .

$\text{GenNym}(pp, usk) \rightarrow (nym, nsk)$

Generates a user pseudonym nym and corresponding secret nsk . ! Can be run arbitrarily often by a user to generate a pseudonym.

$\text{InspectNym}(pp, nym, ntd) = upk$

A deterministic algorithm that outputs a user public key upk . ! Run by the inspector to reveal the user public key upk hidden in nym .

$\text{IssuerKeyGen}(pp, 1^n) \rightarrow (ipk, isk)$

Generates an issuer public key ipk and a corresponding secret key isk . ! The issuer will be able to issue credentials with n attributes.

$cred \leftarrow \text{ReceiveCred}(pp, ipk, nym, \overrightarrow{attr}; nsk, usk) \leftrightarrow \text{IssueCred}(pp, ipk, nym, \overrightarrow{attr}; isk)$

An interactive protocol. ! The user receives a credential $cred$ with attributes $\overrightarrow{attr} \in \mathbb{A}^n$, bound to user's identity consistent with pseudonym nym .

$\text{PresentCred}(pp, ipk, nym, \phi, ctx; \overrightarrow{attr}, nsk, usk, cred)$
 $\leftrightarrow \text{VerifyCred}(pp, ipk, nym, \phi, ctx) \rightarrow b$

An interactive protocol. ! The user, under pseudonym nym , presents credential $cred$ to the verifier. The verifier outputs a bit indicating that $\phi(\overrightarrow{attr}, \alpha) = 1$ and that the credential was bound to the same user identity as nym .

$\text{CheckCred}(pp, ipk, \overrightarrow{attr}, usk, cred) = b$

A deterministic algorithm that outputs a bit b . ! Can be used to check that $cred$ is a valid credential from issuer ipk for attributes $\overrightarrow{attr}$ and bound to user secret usk .

A traditional anonymous credential system is *correct*, which means that for all $n, \lambda \in \mathbb{N}$, $\mathcal{G} \in [\text{GenSetting}(1^\lambda)]$, $(pp, ntd) \in [\text{Setup}(\mathcal{G})]$, $(upk, usk) \in [\text{UserKeyGen}(pp)]$, $(nym, nsk) \in [\text{GenNym}(pp, usk)]$, $(ipk, isk) \in [\text{IssuerKeyGen}(pp, 1^n)]$, and all $\overrightarrow{attr} \in \mathbb{A}^n, \phi \in \Phi, ctx \in \{0, 1\}^*$,

- Pseudonyms trace to user's public key: $\text{InspectNym}(pp, nym, ntd) = upk$.
- Issued credentials are valid: If $cred \in [\text{ReceiveCred}(pp, ipk, nym, \overrightarrow{attr}; nsk, usk) \leftrightarrow \text{IssueCred}(pp, ipk, nym, \overrightarrow{attr}; isk)]$, then $\text{CheckCred}(pp, ipk, \overrightarrow{attr}, usk, cred) = 1$.
- Valid credentials can be presented: If $\text{CheckCred}(pp, ipk, \overrightarrow{attr}, usk, cred) = 1$ and $\phi(\overrightarrow{attr}) = 1$, then we get $\Pr[\text{PresentCred}(pp, ipk, nym, \phi, ctx; \overrightarrow{attr}, nsk, usk, cred) \leftrightarrow \text{VerifyCred}(pp, ipk, nym, \phi, ctx) = 1] = 1$.

The inclusion of context strings ctx is to guard against relay attacks, where a malicious verifier could otherwise impersonate a user by relaying the user's messages to an honest verifier (cf. Section 3.3.2). As a context string, in practice you would, for example, use a name unique to the verifier (say, a DNS domain name) and a session ID or a description of what the user is trying to accomplish by authenticating.

We model attributes as a vector $\overrightarrow{attr} \in \mathbb{A}^n$. It is up to each issuer to define the format of this vector. The format could be something like “the first vector entry contains the SHA-3 hash of the user's name; the second vector entry contains the integer timestamp of the user's birthdate”. Concrete constructions usually have $\mathbb{A} = \mathbb{Z}_p$, where p , in practice, is around 256 bit long. Hence standard integer attributes can be encoded easily in $\mathbb{Z}_p$ (some care should be taken regarding modulo p). Strings or file attributes can usually be encoded as hash values using a collision-resistant hash function (Section 2.6).

The $\text{PresentCred} \leftrightarrow \text{VerifyCred}$ protocol takes ipk, ϕ as part of the common input. We can call (ipk, ϕ) the access policy, as the values determine what attributes are accepted (via ϕ) and the credential issuer (via ipk) who is trusted by the verifier. Where does this input come from in practice? Naively, we can imagine that before the protocol starts, the verifier simply sends their desired ipk, ϕ to the user. After all, one could argue, the verifier should be the one to freely determine their access policy. However, this approach has the disadvantage that, philosophically, the user should check whether he is okay with ipk, ϕ , as those values have privacy implications. For example, ϕ could be maliciously designed so narrowly that it reveals a lot of information about the user's attributes. Furthermore, the user usually has some input in determining ipk, ϕ . For

example, a verifier may accept credentials from either of two issuers ipk, ipk' , and the user can choose which one he actually has a credential from.

In practice, instead of having the verifier choose them, the values of ipk, ϕ are usually derived deterministically by both the user and the verifier from shared data. This means that during design time of an application, the designers set up a deterministic method that generates the proper ipk, ϕ for a given situation. This approach has the advantage that the user can, philosophically, inspect this deterministic method ahead of time and check that he is okay with the choices it makes.

As an example, take the following process, where the user wants to access a resource Z that is age-restricted.

- The user has a credential on attributes $\overrightarrow{attr} = (\text{name}, \text{address}, \text{birth date}, \text{photo})$ issued by ipk_{Berlin}
- The user sends “I want to authenticate to access resource Z . My pseudonym is nym , I’m from Berlin, and here is my photo:  ” to the verifier.
- Both user and verifier separately run a deterministic method. It outputs
 - $ipk = ipk_{\text{Berlin}}$ (chosen from a list of known public keys)
 - $\phi(\overrightarrow{attr}) \equiv (\text{photo} = \text{)} \wedge (\text{birth date} \leq \text{today} - 18\text{y})$
 (note that ϕ depends on the user’s request with its concrete photo , as well as on the fact that the requested resource Z is age-restricted, and on the current date; all of this is information both the user and the verifier have.)
 - $ctx = \text{”access.example.net/resources/Z”}$
- They then run the **PresentCred** $\leftrightarrow$ **VerifyCred** protocol using the ipk, ϕ, ctx they have derived from the request and other application data separately.

If the **PresentCred** $\leftrightarrow$ **VerifyCred** protocol is non-interactive, i.e., it only consists of a single message from the user to the verifier, then in the example, the user can send that single message directly alongside the request for the resource.

Finally, note that according to the correctness criteria, credentials can be “transferred” between different pseudonyms of the same user: A user can receive a credential under pseudonym nym and then present it under a different pseudonym nym' .

4.2.2 Anonymity

For security of traditional anonymous credential systems, we want (1) anonymity for users, (2) that users cannot present credentials they have not been issued, and (3) that users cannot be impersonated. We present the formal anonymity requirements here and will discuss the other security notions in Section 4.2.3

Anonymity protects the privacy of honest users against malicious issuers and verifiers. Intuitively, we want that even if issuers/verifiers have gathered lots of information about a user, when the user generates a new pseudonym, that pseudonym should essentially start with a clean slate. This means that for verifiers/issuers, it should look like the

new pseudonym might as well belong to a completely new/different user. By presenting credentials to verifiers under the new pseudonym, the user of course reveals more and more information about his identity — if access is granted to the new pseudonym and only few users have those access rights, then clearly, the pseudonym must belong to one of those users. Still, malicious verifiers should not be able to distinguish between two possible users *unless* the pseudonym accesses a resource that only one of the users could have accessed. We start by giving intuition and justification for our security model, then present the model itself (Definition 4.8), and afterward discuss some finer details.

The security model involves a game with an adversary $\mathcal{A}$, playing the role of malicious issuers and verifiers. There are two honest users in the game, $U = 0$ and $U = 1$. We allow the adversary $\mathcal{A}$ to interact freely with each user, asking for pseudonyms and issuing/verifying the users' credentials. This can be used by $\mathcal{A}$ to set up any situation regarding pseudonyms and credentials of the two users. Because $\mathcal{A}$ directs the actions of each user, $\mathcal{A}$ has full information about which pseudonyms/credentials belong to which user.

Now we introduce some uncertainty for $\mathcal{A}$ into the game. We imagine *one* of the users, namely user b , establishes a new set of pseudonyms. As discussed above, a new pseudonym should essentially be a clean slate and look like it belongs to a completely new user. We model this by assigning those pseudonyms, for which $\mathcal{A}$ does not know which user they belong to, to a virtual user $U = \star$. The virtual user $U = \star$ uses the user secret usk_b of user b . The task of $\mathcal{A}$ is to distinguish $b = 0$ from $b = 1$, i.e., find out which user is behind the pseudonyms of $U = \star$. To cover all conceivable real-world scenarios of how $\mathcal{A}$ may interact with unknown-user pseudonyms, we allow $\mathcal{A}$ to issue credentials to those pseudonyms and to verify credentials from those pseudonyms. Furthermore, $\mathcal{A}$ can ask for an arbitrary number of those unknown-user pseudonyms.

Overall, pseudonyms and credentials in this game are partitioned into pseudonyms and credentials that $\mathcal{A}$ knows belong to $U = 0$, those that $\mathcal{A}$ knows belong to $U = 1$, and those where $\mathcal{A}$ knows they belong to one of the users, but does not know which one ($U = \star$). This setup can be seen in Figure 4.3. As mentioned above, the trivial way for $\mathcal{A}$ to gain distinguishing information about b is to play verifier for a policy that can only be fulfilled by one of the users. This can happen in two ways: (1) $\mathcal{A}$ asks a pseudonym associated with $U = \star$ to present a credential that has only been issued to $U = 0$ or only been issued to $U = 1$, or (2) $\mathcal{A}$ asks a pseudonym associated with $U \in \{0, 1\}$ to present a credential that has been issued to $U = \star$. In both cases, the distinguishing factor is that depending on b , the credential showing (even with an honest verifier) will either succeed or fail (because depending on b , either the user behind the pseudonym has a fitting credential or does not have one).

To avoid trivially leaking b when presenting a credential, we have $\mathcal{A}$ specify *two* credentials restricted so that the first is guaranteed to work if $b = 0$ and the second is guaranteed to work if $b = 1$. For example, as in Figure 4.3, $\mathcal{A}$ can ask a pseudonym N of user $U = \star$ to present a credential if $\mathcal{A}$ can point to a credential issued to $U = 0$ and one issued to $U = 1$ that both fulfill the access policy. If $b = 0$, presentation is done with the first credential, if $b = 1$, it is done with the second. $\mathcal{A}$ should not be able to distinguish these cases. As another example, $\mathcal{A}$ can ask a pseudonym N of user $U = 0$ to present

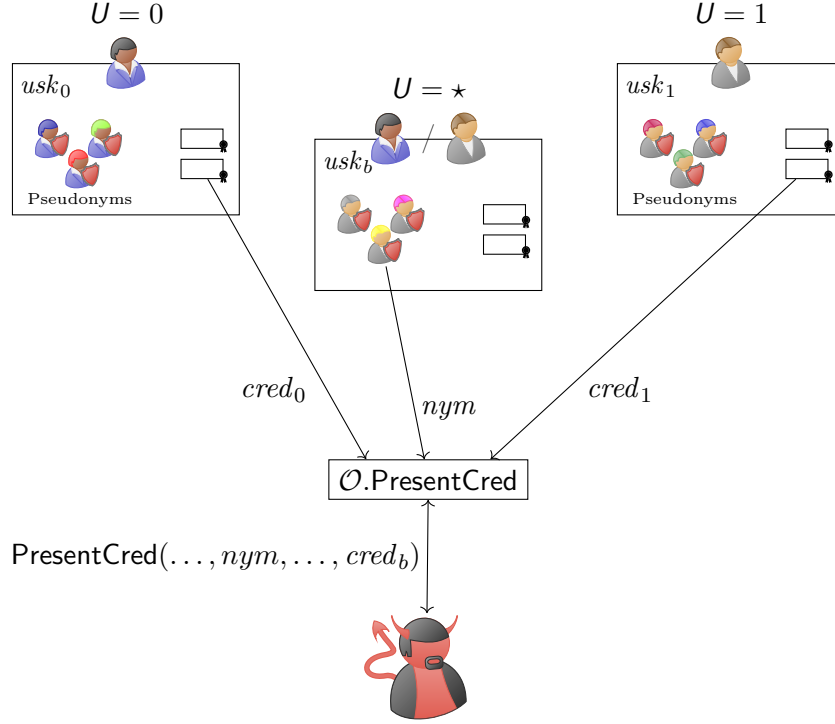


Figure 4.3: Overview of users $U \in \{0, 1, *\}$ and their associated pseudonyms and credentials. Example of an $\mathcal{O}.\text{PresentCred}$ query with the pseudonym of the virtual user $U = *$ and a credential $cred_0$ issued to $U = 0$ or a credential $cred_1$ issued to $U = 1$.

a credential issued to $U = *$ if it can also point to a credential issued to $U = 0$ (which would be used in case $b = 1$, i.e., the first credential *cannot* be used by $U = 0$ because it has not been issued to that user).

In reality, we imagine that an attacker asks a pseudonym to present a credential with certain requirements. Then the user scans his list of credentials in some way to find a fitting one to use. $\mathcal{A}$ can mimic this behavior: $\mathcal{A}$ can compute the exact list of credentials for each pseudonym for the case $b = 0$ and for the case $b = 1$. For example, if $b = 0$ and the pseudonym belongs to $U = 0$, then the credential list available to that pseudonym is the list of $U = 0$ concatenated with the list of $U = *$. $\mathcal{A}$ can then run whatever procedure would be used by an honest user to find a fitting credential for both cases, $b \in \{0, 1\}$. Then $\mathcal{A}$, in the experiment, can point to exactly those two credentials that would be picked by honest users in the two cases, and try to use the resulting protocol run to distinguish.

Formally, we define anonymity as follows.

Definition 4.8 (Traditional credential anonymity). Let $\mathcal{A}$ be an algorithm. Consider the following security games $\text{Exp}_{\Pi_{\text{acs}}}^{\text{anon}}(\Pi_{\text{acs}}, \mathcal{A}, \lambda, b)$ for $b \in \{0, 1\}$.

$\text{Exp}_{\Pi_{\text{acs}}}^{\text{anon}}(\Pi_{\text{acs}}, \mathcal{A}, \lambda, b)$

$\mathcal{G} \leftarrow \text{GenSetting}(1^\lambda), (pp, ntd) \leftarrow \text{Setup}(\mathcal{G})$
 $(upk_0, usk_0), (upk_1, usk_1) \leftarrow \text{UserKeyGen}(pp)$.
 Set $usk_\star := usk_b$
 Run $b' \leftarrow \mathcal{A}(\mathcal{G}, pp, upk_0, upk_1)$ with access to the oracles listed below
 return b'

$\mathcal{O}.\text{GenNym}(U)$ //Have honest user generate pseudonym

assert $U \in \{0, 1, \star\}$
 Choose new pseudonym handle N
 Set $U_N := U$ //Remember pseudonym owner
 $(nym_N, nsk_N) \leftarrow \text{GenNym}(pp, usk_U)$
 return (nym_N, N)

$\mathcal{O}.\text{ReceiveCred}(ipk, N, \overrightarrow{attr})$ //Have honest user receive credential

Choose new credential handle C //Disjoint from pseudonym handles
 Set $ipk_C := ipk, U_C := U_N, \overrightarrow{attr}_C := \overrightarrow{attr}$ //Remember issuer, user, attributes
 Set $cred_C = \perp$
 $cred_C \leftarrow \text{ReceiveCred}(pp, ipk, nym_N, \overrightarrow{attr}; nsk_N, usk_{U_N}) \leftrightarrow \mathcal{A}$
 return C if $cred_C \neq \perp$

$\mathcal{O}.\text{PresentCred}(N, C_0, C_1, \phi, ctx)$ //Have honest user present credential C_b

// $C_0 = C_1$ is allowed
 assert $cred_{C_0} \neq \perp, cred_{C_1} \neq \perp$ //Both credentials are valid
 assert $ipk_{C_0} = ipk_{C_1}$ //Both credentials were issued by same issuer
 assert $\phi(\overrightarrow{attr}_{C_0}) = \phi(\overrightarrow{attr}_{C_1}) = 1$ //Both credentials' attributes fulfill ϕ
 assert $\{U_{C_0}, U_N\} \neq \{0, 1\}$ and $\{U_{C_0}, U_N\} \neq \{\star, 1\}$ // C_0 is viable for $b = 0$
 assert $\{U_{C_1}, U_N\} \neq \{0, 1\}$ and $\{U_{C_1}, U_N\} \neq \{\star, 0\}$ // C_1 is viable for $b = 1$
 $\text{PresentCred}(pp, ipk_{C_b}, nym_N, \phi, ctx; \overrightarrow{attr}_{C_b}, nsk_N, usk_{U_N}, cred_{C_b}) \leftrightarrow \mathcal{A}$

A traditional ACS Π_{acs} is *anonymous* if for all ppt $\mathcal{A}$,

$$|\Pr[\text{Exp}_{\Pi_{\text{acs}}}^{\text{anon}}(\Pi_{\text{acs}}, \mathcal{A}, \lambda, \underline{0}) = 1] - \Pr[\text{Exp}_{\Pi_{\text{acs}}}^{\text{anon}}(\Pi_{\text{acs}}, \mathcal{A}, \lambda, \underline{1}) = 1]|$$

is negligible

We make some notes about this definition. The *handles* C, N work as you would expect: They are basically pointers to a credential or pseudonym, respectively. Note that all assert statements in the oracles can be evaluated by $\mathcal{A}$ itself, as $\mathcal{A}$ knows for every credential handle C its associated attributes $\overrightarrow{attr}_C$, issuer key ipk_C , as well as the pseudonym N_C it was issued to. Furthermore, it knows for every pseudonym (handle) N which (virtual) user $U_N \in \{0, 1, \star\}$ it is associated with.

The $\mathcal{O}.\text{PresentCred}$ oracle is set up such that $\mathcal{A}$ is allowed to point to any credentials

C_0, C_1 as long as C_0 is viable in case $b = 0$, and C_1 is viable in case $b = 1$. With “viable”, we mean that the credential is associated with the same user secret usk_0 or usk_1 as the pseudonym. Then C_b is presented to $\mathcal{A}$. $\mathcal{A}$ can, in particular, use $\mathcal{O}.\text{PresentCred}$ as follows:

- Have one of two credentials by user $U \in \{0, 1, \star\}$ presented (using $U_N = U_{C_0} = U_{C_1}$).
- Have a specific credential presented (as above, but $C_0 = C_1$).
- Have user $U = \star$ present a credential issued to $U = 0$ or to $U = 1$, as in Figure 4.3 (using $U_N = \star, U_{C_0} = 0, U_{C_1} = 1$).
- Have user $U = 0$ present a credential issued to $U = \star$ or to $U = 0$ (using $U_N = 0, U_{C_0} = \star, U_{C_1} = 0$).

In particular, along with the other privacy properties discussed at the beginning of this section, the definition implies that it is hard to distinguish presentation of a credential with attributes $\overrightarrow{attr}$ from presentation of a credential with attributes $\overrightarrow{attr}'$ (as long as both fulfill ϕ and have been issued by the same issuer), even if they both belong to the same user.

The only forbidden combinations for $\mathcal{O}.\text{PresentCred}$ queries are those where pseudonym and credential do not belong to the same user:

- $\{U_{C_i}, U_N\} = \{0, 1\}$: The credential and pseudonym belong to two different known users (independent of b). This means that presenting C_i would be expected to fail as the credential has not been issued to the owner of the pseudonym.
- $\{U_{C_0}, U_N\} = \{\star, 1\}$: The credential C_0 belongs to $\star$ but the pseudonym N belongs to 1 (or vice versa, the pseudonym belongs to $\star$ and the credential belongs to 1). Because C_0 is presented in case $b = 0$, this means that the owner of the credential corresponds to user 0 but the owner of the pseudonym is user 1 (or vice versa). Because pseudonym and credential belong to different users, we would expect the presentation to fail.
- $\{U_{C_1}, U_N\} = \{\star, 0\}$: The credential C_1 belongs to $\star$ but the pseudonym N belongs to 0 (or vice versa, the pseudonym belongs to $\star$ and the credential belongs to 0). Because C_1 is presented in case $b = 1$, this means that the owner of the credential corresponds to user 1 but the owner of the pseudonym is user 0 (or vice versa). Because pseudonym and credential belong to different users, we would expect the presentation to fail.

All other combinations are allowed, giving the adversary enough power to model any real-world situation that does not allow the verifier to distinguish trivially.

Furthermore, we point out that the protocol execution in our anonymity definition is *asynchronous* in nature. The $\leftrightarrow$ in the oracles denotes asynchronous protocol execution, so several instances of the relevant protocols can be interleaved (see Section 2.2.5).

Obviously, we cannot give $\mathcal{A}$ the pseudonym trapdoor ntd , as that would allow it to immediately distinguish users. One can imagine a “CCA”-like attack, where a verifier would blame some pseudonym nym for an action, making the inspector reveal its identity.

To manage complexity of the construction, we did not model this here. One could easily amend the security model, adding an inspection oracle. This would certainly improve the security definition on a theoretical level. However, in practice, a much more involved mechanism to handle inspection requests is needed. For example, a trivial way to de-anonymize a user is to produce fake evidence of wrongdoing by a pseudonym, and then asking the inspector to reveal the user's identity. This is a plausible attack that is not prevented even by the modified “CCA” security definition. So in order to really make privacy in the presence of an inspector work, there needs to be a more involved mechanism, making sure that users' identities are not revealed without any wrongdoing. One way to achieve this, for example, could be to have the user pseudonymously sign off on all his actions. Then in order to reveal the user behind a pseudonym, the inspector would expect unforgeable proof of wrongdoing in the form of the pseudonym's/user's signatures. As a consequence, users cannot be blamed anymore for actions they did not commit. However, for the sake of simplicity of definition and construction, and to avoid a false sense of security, we choose not to build such a mechanism explicitly into the model. Anonymity is only guaranteed in scenarios where the inspector is not involved at all.

4.2.3 Unforgeability and impersonation

Unforgeability protects honest issuers, users, and verifiers against malicious users (and malicious issuers/verifiers) trying to gain access to a resource they should not have access to. Intuitively, the only way to convince a verifier should be to have previously been issued a fitting credential.

As a first step, it is easiest to think about an honest issuer and an honest verifier (meaning that for now, *all* users are malicious). Our adversary $\mathcal{A}$ takes the role of malicious users. We allow $\mathcal{A}$ to ask to be issued credentials from the honest issuer. The goal of $\mathcal{A}$ is to convince the honest verifier that it has a credential from the honest issuer that it has not asked for. More specifically, this means that $\mathcal{A}$, under a pseudonym nym_{show} , tries to convince the honest verifier that it has a credential with attributes satisfying a predicate ϕ . $\mathcal{A}$ can convince the verifier “legally” if it has been issued a credential with attributes $\vec{attr}$ such that $\phi(\vec{attr}) = 1$. However, if $\mathcal{A}$ has not asked for such a credential before, we consider this a forgery. We also consider it a forgery if $\mathcal{A}$ has been issued a credential with such attributes but did so under a different user identity (according to the pseudonym inspector) than it used presenting the credential. This process is modeled through the oracles $\mathcal{O}.\text{IssueCred}$ and $\mathcal{O}.\text{VerifyCred}$ in Definition 4.9.

In particular, this defines a notion of non-transferability of credentials: If a credential is issued to a pseudonym nym , it can only be shown on pseudonyms nym' that trace to *the same user* as nym . In this sense, credentials cannot be transferred between users: If a user hands their credential to someone else, the original credential holder will still be held responsible for the actions of that other user (in case the pseudonym is inspected). This discourages handing credentials to someone else. We discuss this more in Section 4.4.

So far, we have only talked about malicious users. However, *honest* users also have a role to play in access control. A malicious user may try to illegally access a resource by

interacting with honest users and trying to extract their access rights. A trivial example for such an attack is a relay attack (Section 3.3.2), where the malicious user plays the role of a verifier and passes an honest user's messages as his own.

In order to model attacks involving honest users, we allow $\mathcal{A}$ to interact with any number of honest users, directing them (1) to generate and show pseudonyms, (2) to receive credentials from $\mathcal{A}$ and from the honest issuer, and (3) to present credentials to $\mathcal{A}$. This interaction with honest users is modeled through the oracles $\mathcal{O}.\text{GenNym}$, $\mathcal{O}.\text{ReceiveCred}$, $\mathcal{O}.\text{HonestIssue}$, and $\mathcal{O}.\text{PresentCred}$ in Definition 4.9.

Even in the presence of honest users, $\mathcal{A}$ should not be able to illegally gain access anywhere or to place blame on an honest user by acting under one of his pseudonyms. Concretely, we say that $\mathcal{A}$ wins in the following cases:

- **Forgery:** $\mathcal{A}$ is able to present a credential that was not issued to $\mathcal{A}$ (it may have been issued to an honest user).
- **Impersonation:** $\mathcal{A}$ impersonates an honest user by acting under a pseudonym that traces to an honest user via the inspector.

Formally, the security definition is as follows.

Definition 4.9 (Traditional credential unforgeability and impersonation resistance). Consider the following game $\text{Exp}_{\Pi_{\text{acs}}}^{\text{unforge}}(\Pi_{\text{acs}}, \mathcal{A}, n, \lambda)$.

$\text{Exp}_{\Pi_{\text{acs}}}^{\text{unforge}}(\Pi_{\text{acs}}, \mathcal{A}, n, \lambda)$

$\mathcal{G} \leftarrow \text{GenSetting}(1^\lambda)$, $(pp, ntd) \leftarrow \text{Setup}(\mathcal{G})$
 $(ipk^*, isk^*) \leftarrow \text{IssuerKeyGen}(pp, 1^n)$ //Honest issuer key
 $\mathcal{A}(\mathcal{G}, pp, ipk^*, ntd)$, with access to the oracles listed below
 When $\mathcal{O}.\text{VerifyCred}$ raises $\text{win}_{\text{forgery}}$ or $\text{win}_{\text{impersonate}}$, output 1
 If no such event is raised, output 0

$\mathcal{O}.\text{GenNym}(U)$ //Have honest user generate pseudonym

If user handle U has not been queried to this oracle before:
 Generate $(upk_U, usk_U) \leftarrow \text{UserKeyGen}(pp)$
 Choose new pseudonym handle N and set $U_N := U$
 $(nym_N, nsk_N) \leftarrow \text{GenNym}(pp, usk_U)$
 return (upk_U, nym_N, N)

$\mathcal{O}.\text{ReceiveCred}(ipk, N, \overrightarrow{attr})$ //Have honest user receive credential from $\mathcal{A}$

Choose new credential handle C
 Set $ipk_C := ipk$, $N_C := N$, $\overrightarrow{attr}_C := \overrightarrow{attr}$, $cred_C := \perp$
 $cred_C \leftarrow \text{ReceiveCred}(pp, ipk, nym_N, \overrightarrow{attr}; nsk_N, usk_{U_N}) \leftrightarrow \mathcal{A}$
 return C if $cred_C \neq \perp$

$\mathcal{O}.\text{HonestIssue}(N, \overrightarrow{attr})$ //Have honest issuer issue to honest user

```

assert  $\overrightarrow{attr} \in \mathbb{A}^n$ 
Choose new credential handle  $C$ 
Set  $ipk_C := ipk^*, N_C := N, \overrightarrow{attr}_C := \overrightarrow{attr}, cred_C := \perp$ 
 $cred_C \leftarrow \text{ReceiveCred}(pp, ipk^*, nym_N, \overrightarrow{attr}; nsk_N, usk_{U_N})$ 
 $\leftrightarrow \text{IssueCred}(pp, ipk^*, nym_N, \overrightarrow{attr}; isk^*)$ 
return  $C$ 

```

$\mathcal{O}.\text{PresentCred}(N, C, \phi, ctx)$ //Have honest user present credential to $\mathcal{A}$

```

assert  $U_N = U_{N_C}$  //Pseudonym belongs to same user as credential
assert  $\phi(\overrightarrow{attr}_C) = 1$  //Attributes fulfill  $\phi$ 
Add  $(ipk_C, nym_N, \phi, ctx)$  to  $\mathcal{T}_{\text{presented}}$ 
 $\text{PresentCred}(pp, ipk_C, nym_N, \phi, ctx; \overrightarrow{attr}_C, nsk_N, usk_{U_N}, cred_C) \leftrightarrow \mathcal{A}$ 

```

$\mathcal{O}.\text{IssueCred}(nym, \overrightarrow{attr})$ //Have honest issuer issue to $\mathcal{A}$

```

Add  $(nym, \overrightarrow{attr})$  to  $\mathcal{T}_{\text{issued}}$ 
 $\text{IssueCred}(pp, ipk^*, nym, \overrightarrow{attr}; isk^*) \leftrightarrow \mathcal{A}$ 

```

$\mathcal{O}.\text{VerifyCred}(ipk, nym, \phi, ctx)$ // $\mathcal{A}$ shows credential to honest verifier

```

 $b \leftarrow \text{VerifyCred}(pp, ipk, nym, \phi, ctx) \leftrightarrow \mathcal{A}$ 
If  $b = 0$  or  $(ipk, nym, \phi, ctx) \in \mathcal{T}_{\text{presented}}$ :
    return  $b$  //Verification unsuccessful or authenticated by honest user
 $upk = \text{InspectNym}(pp, nym, ntd)$  //Identity behind  $nym$  (for win conditions)
If  $\exists N : (nym_N = nym \vee \text{InspectNym}(pp, nym_N, ntd) = upk)$ :
    raise  $\text{win}_{\text{impersonate}}$  //Pseudonym belongs to or traces to honest user
If  $ipk = ipk^*$ :
    If  $\forall (nym', \overrightarrow{attr}') \in \mathcal{T}_{\text{issued}} : \phi(\overrightarrow{attr}') = 0 \vee \text{InspectNym}(pp, nym', ntd) \neq upk$ :
        raise  $\text{win}_{\text{forgery}}$  //  $\mathcal{A}$  presented credential it never requested
return  $b$ 

```

A traditional ACS Π_{acs} is *unforgeable* and has *impersonation resistance* if for all $n \in \mathbb{N}$ and all ppt $\mathcal{A}$, $\Pr[\text{Exp}_{\Pi_{\text{acs}}}^{\text{unforge}}(\Pi_{\text{acs}}, \mathcal{A}, n, \lambda) = 1]$ is negligible.

We make a note of the following facts about this definition. First, we give the adversary the inspector's pseudonym trapdoor ntd . This implies the even a malicious inspector cannot break unforgeability. However, in this definition, pp, ntd must be generated honestly. This is not inherent in traditional anonymous credential systems, but rather done here to limit definitional complexity.

In order to prevent trivial relay attacks, we use that the presentation of a pseudonym or credential is bound to a context ctx . The role of ctx is to ensure that in practice, honest users have some agency regarding the context in which their presentation can be used. We discuss this in Section 3.3.2. For the unforgeability definition, this means

that, roughly, we consider it trivial for $\mathcal{A}$ to convince an honest verifier for a context ctx that an honest user has already used. We note that having ctx in the anonymous credential definitions is not simply an artifact of our reliance on zero-knowledge proofs in concrete constructions. Instead, relay attacks *generically* apply to credential presentation protocols, even if they do not rely on zero-knowledge proofs.

Credentials are bound to their owner in the following sense. When a credential is issued, it is issued for a specific pseudonym nym_{iss} . If the adversary presents a credential with pseudonym nym_{pres} , the $\text{win}_{\text{forgery}}$ winning condition only considers credentials issued to nym_{iss} that open to the same user as nym_{pres} . This means that if an adversary can transfer the credential to another user, it wins. In particular, this implies that credentials requested for honest users must be useless to the adversary. Indeed, if the adversary queries $\mathcal{O}.\text{PresentCred}$ with an honest user's pseudonym nym_{hon} and attributes $\overrightarrow{attr}$, this does not enable it to trivially present $\overrightarrow{attr}$ via $\mathcal{O}.\text{VerifyCred}$. If the adversary calls $\mathcal{O}.\text{VerifyCred}$ with a pseudonym nym_{pres} that traces to an honest user, then $\text{win}_{\text{impersonate}}$ is raised. If the adversary calls $\mathcal{O}.\text{VerifyCred}$ with a pseudonym nym_{pres} that does *not* trace to an honest user, then the entry $(nym_{hon}, \overrightarrow{attr})$ in $\mathcal{T}_{\text{issued}}$ does not help avoid triggering $\text{win}_{\text{forgery}}$. This is because nym_{hon} and nym_{pres} trace to different upk (the former traces to an honest user's upk_{hon} due to correctness, the latter does not trace to an honest user as assumed for this case distinction).

4.3 The standard template of traditional anonymous credential systems

Traditional anonymous credentials are usually constructed from a pseudonym system, digital signatures with efficient protocols, and zero-knowledge proofs of knowledge, following the template first explicitly articulated in [CL02]. This “standard” template works as follows.

- As discussed in Section 4.1, pseudonyms are generally commitments to the user's secret key usk or to his public key upk . The pseudonym secret nsk is the opening randomness of the commitment. Because we want inspectable pseudonyms, pseudonyms are encryptions of upk under the inspector's public encryption key. In our case, there is a one-to-one correspondence of public keys and secret keys, so this encryption can be seen as a commitment to usk .
- Credentials are signatures under the issuer's public key on the message $(usk, \overrightarrow{attr})$, i.e., on the credential owner's user secret and on the credential's attributes. This binds the credential and its attributes to a specific user.
- Issuing a credential is difficult because the issuer must not learn usk , which is part of the message to be signed. For this, issuer and user engage in a protocol to blindly sign usk . This protocol can be seen as a secure two-party computation with common input $(nym, \overrightarrow{attr})$. The user secretly inputs his secret key usk and pseudonym secret

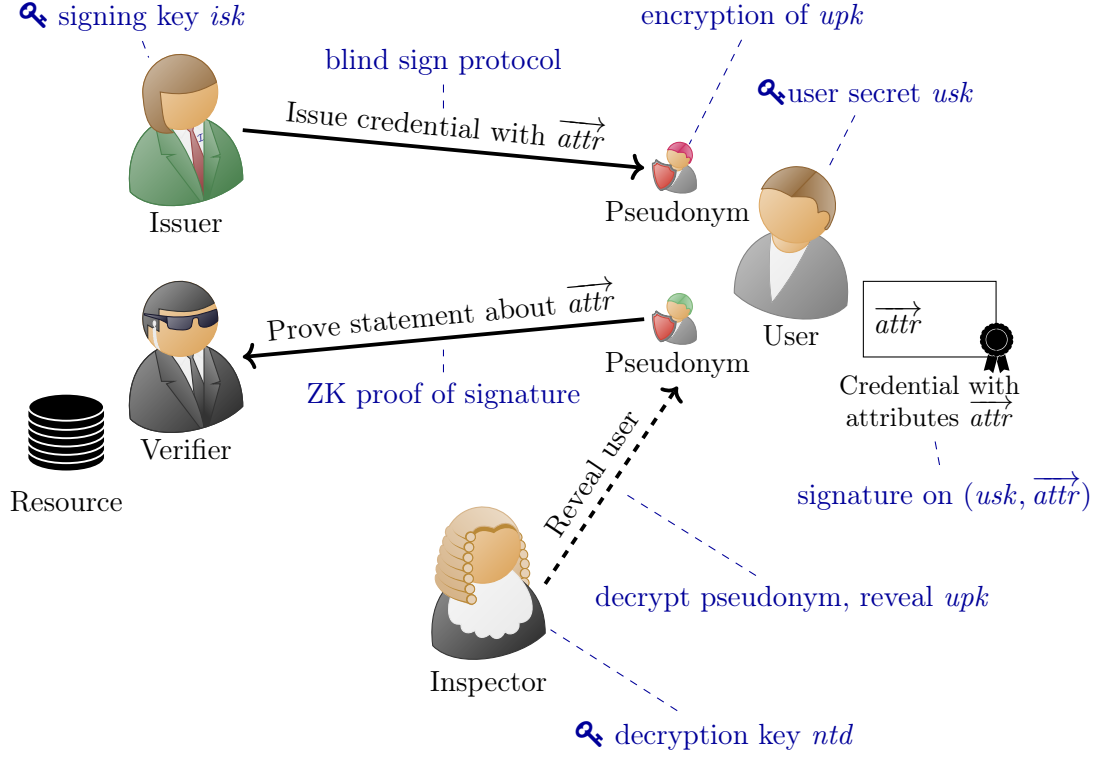


Figure 4.4: The standard template for constructing traditional anonymous credential systems.

nsk , and the issuer secretly inputs her signing key isk . The output for the user is a signature σ on $(usk, \vec{attr})$ if usk is indeed the value committed to in nym .

- This protocol ensures that the user only learns the credential (nothing about the issuer’s key isk) and that the issuer does not learn anything about usk (other than that it is the value hidden in nym).
- As we will see later, this usually will not be implemented using generic MPC tools, but rather we use special signature schemes that come with efficient protocols for blindly signing the value usk .

We discuss this in more detail in Section 5.3.1.

- To present a credential, the user proves, with a zero-knowledge proof of knowledge, that he knows $usk, nsk, \sigma, \vec{attr}$ such that
 - nym can be opened to usk via nsk .
 - σ is a valid signature on $(usk, \vec{attr})$ under the issuer’s public key.
 - the attributes fulfill the verifier’s predicate, i.e., $\phi(\vec{attr}) = 1$.

Clearly, this proof can only succeed if the user has a fitting credential and owns the pseudonym nym .

You can find an overview in Figure 4.4. *Most* traditional anonymous credential system constructions conform to this template. Some notable exceptions are anonymous credential systems based on structure-preserving signatures on equivalence classes [FHS19; BF20], where presentation does not (significantly) rely on zero-knowledge proofs.

On a high level, anonymity for constructions following the template can be argued as follows: (1) The zero-knowledge proof used in the presentation protocol can be simulated without the user’s secret values, and similarly (2) the interaction in the issuing protocol can be replaced with a simulated version, too. At this point the only part that still depends on user identities is the pseudonym system. However, the unlinkability property (Definition 4.2) of the pseudonym system guarantees that $\mathcal{A}$ cannot exploit this.

For unforgeability and impersonation-resistance, we replace all honest user actions with simulated versions, just like in steps 1-2 for anonymity. So roughly speaking, the honest users do not help the adversary. If the adversary wins by raising $\text{win}_{\text{impersonate}}$, we can extract, from his proof of knowledge, a solution to the pseudonym impersonation resistance experiment (Definition 4.4). If the adversary wins by raising $\text{win}_{\text{forgery}}$, we can extract a signature scheme forgery.

The details for the security proofs are much more involved. However, we will not formally prove security for constructions following the template. Instead, we generalize the template in Chapter 5, formalizing the more powerful notion of *updatable* anonymous credentials. We then show how to construct traditional anonymous credentials from updatable anonymous credentials in Section 5.4.

The choice to include usk in the credential instead of upk is in some sense arbitrary. Its function is to cryptographically bind the signature to the user’s identity, which upk would also be able to achieve. Including usk in the signature is convenient in the bilinear group setting, where secret keys usk and signature scheme messages usually live in $\mathbb{Z}_p$ (making it easy to sign usk), whereas public keys upk usually live in the group (making it difficult to sign upk).

4.4 Discussion about binding credentials to identities and non-transferability

Traditional anonymous credential systems put a lot of effort into binding credentials to identities. In the standard template, this is done by including the user’s secret usk alongside the attributes $\overrightarrow{attr}$ in the issuer’s signature. Because the issuer must not learn usk (or even *anything* about the identity of the user), this necessitates a complex protocol to allow the issuer to *blindly* sign usk . All this is done, intuitively, to enforce *non-transferability*. The idea is that a credential issued to user A is *bound* to him and cannot be used by user B . In this section, we discuss how non-transferability is at the core of traditional anonymous credentials. Without it, the standard template for constructing anonymous credentials degenerates into a *trivial* composition of a pseudonym system and a signature scheme (plus zero-knowledge proofs for each) that function separately and independently. We will also discuss how, even though it is very important to traditional anonymous credentials, the concept of non-transferability is somewhat brittle, which is

an inherent issue with combining anonymity and non-transferability. As a result, non-transferability is often handwaved in definitions. There are solutions in the literature, which we discuss here, too.

4.4.1 The degenerate standard template

Let us explore what a traditional anonymous credential system looks like if we do not bind credentials to identities. For this, we start with the standard template (Section 4.3), but we change it such that a credential is now a signature just on $\overrightarrow{attr}$ (instead of on $(usk, \overrightarrow{attr})$). This change radically simplifies the issuing process because we can now get rid of the blind signing protocol and simply have the issuer send a signature on $\overrightarrow{attr}$ to the user. More specifically, if we change the protocols to naturally account for usk not being part of the signature anymore, we get the following:

- To issue a credential, the issuer simply sends a signature σ on $\overrightarrow{attr}$ to the user.
- When presenting a credential under pseudonym nym with predicate ϕ , the user first proves, in zero-knowledge, that nym belongs to him. Then the user proves, in zero-knowledge, that he knows a signature σ on $\overrightarrow{attr}$ such that $\phi(\overrightarrow{attr}) = 1$.

As we can see, the simplified standard template cleanly splits into a part for the pseudonym system, which ensures impersonation does not take place, and a part for the signature scheme, which authenticates attributes. Those two building blocks are now run sequentially with nonintersecting input, so this composition is rather trivial. Our goal is to understand why this much simpler construction is flawed, and how this is reflected in the security definitions. In essence, why are anonymous credentials not simply constructed like this?

It is easy to see intuitively that this degenerate standard template does not have any transferability prevention anymore: A credential is simply a signature on $\overrightarrow{attr}$, independent of any user identities, so it can be handed to and used by anyone. So how does this manifest in our security definitions? Perhaps surprisingly, all of the non-transferability guarantees in the security definitions can be traced to a single short expression in Definition 4.9, underlined in the following:

```

 $\mathcal{O}.$ VerifyCred( $ipk, nym, \phi, ctx$ ) //  $\mathcal{A}$  shows credential to honest verifier
 $b \leftarrow \text{VerifyCred}(pp, ipk, nym, \phi, ctx) \leftrightarrow \mathcal{A}$ 
If  $b = 0$  or  $(ipk, nym, \phi, ctx) \in \mathcal{T}_{\text{presented}}$ :
    return  $b$  // Verification unsuccessful or authenticated by honest user
 $upk = \text{InspectNym}(pp, nym, ntd)$  // Identity behind  $nym$  (for win conditions)
If  $\exists N : (nym_N = nym \vee \text{InspectNym}(pp, nym_N, ntd) = upk)$ :
    raise  $\text{win}_{\text{impersonate}}$  // Pseudonym belongs to or traces to honest user
If  $ipk = ipk^*$ :
    If  $\forall (nym', \overrightarrow{attr}') \in \mathcal{T}_{\text{issued}} : \phi(\overrightarrow{attr}') = 0 \vee \text{InspectNym}(pp, nym', ntd) \neq upk$ :
        raise  $\text{win}_{\text{forgery}}$  //  $\mathcal{A}$  presented credential it never requested
return  $b$ 

```

If we remove this expression from the unforgeability definition, then this degenerate standard template would be considered secure. It would still guarantee meaningful unforgeability (the adversary cannot present a credential it has not asked for) and impersonation resistance (the adversary cannot present a credential under an honest user's pseudonym).

4.4.2 Modeling non-transferability

We have now seen that the way we model non-transferability in the unforgeability game hinges *crucially* on a single expression involving pseudonym inspection. If this expression were not there, a degenerate construction without *any* non-transferability guarantees would be considered secure. This also means that if we had not defined traditional anonymous credentials *with pseudonym inspection*, we would not have been able to define non-transferability like this, at all (which is why we chose to include pseudonym inspection, even though it is often just considered an optional feature).

There is an inherent issue with modeling non-transferability for traditional anonymous credentials: It is difficult to formulate a winning conditions like “ $\mathcal{A}$ wins if the credential was issued to a different user than the one currently presenting the credential”. This is because of anonymity guarantees: Issuers/verifiers (whom we are trying to protect) *cannot tell* to whom they issue credentials or to whom a pseudonym belongs. To phrase this even stronger: From the point of view of (honest) issuers/verifiers, it might just as well be the case that $\mathcal{A}$ always uses the same *usk* for all their pseudonyms. Because of anonymity, issuers/verifiers will not be able to tell this case from any other. But if $\mathcal{A}$ always uses the same *usk*, then transferring credentials becomes a feature: Credentials bound to the same *usk* are *supposed to be* transferable between pseudonyms of the same user.

This raises the following question:

! If issuer/verifiers *cannot distinguish* between the transferable case and the non-transferable case, why would they even care about allowing people to transfer their credentials?

We will come back to this question in Section 4.4.8.

First, in the following, we discuss ways to enforce non-transferability. For each candidate solution, we have to answer:

- How do we model the non-transferability win condition in the theoretical definition?
- How does the mechanism deter users from transferring/sharing credentials in practice?

For the second question, note that by default, any person can enable another person to use their credentials by giving him *usk* together with the credential. So we need to be able to say why users are discouraged from doing this or how we can make it such that they would not be able to do it at all.

4.4.3 Non-transferability via pseudonym inspection

The way we chose in this thesis around the issues explained above is by introducing pseudonym inspection [CL01] into the system. A trusted third party can always find out who is behind a pseudonym. There is now an objective measure whom a pseudonym belongs to: If the inspector says two pseudonyms belong to the same user, we consider this to be the case. We can use this objective measure to define the winning condition for transferring a credential in the theoretical definition, as we do in Definition 4.9.

While in reality, issuers/verifiers by themselves *still* cannot distinguish between different users, the inspector can. Hence there is now a mechanism in place to deter users from sharing credentials/pseudonyms: If a user (somehow) hands his credential to someone else, the unforgeability guarantees imply that the inspector will trace any malicious behavior associated with the credential to its original owner.

Of course, for this threat to carry weight in practice, when the inspector outputs some upk , he should be able to map this value to a real person. But anyone can locally generate any number of user keys (upk, usk) that are not inherently linked to any person. To fix this, we need some more infrastructure: We can imagine that when the user first joins the system, he generates (upk, usk) and then non-anonymously, with plain name and address, registers upk with an authority. In return, the user receives a system credential, whose only role is to certify that the user's upk has been registered. Then, whenever the user establishes a new pseudonym, the verifier will also ask to present the system credential. This ensures that whatever upk is hidden within the pseudonym has been registered, i.e., can be traced to a real person by the authorities. This process can be implemented generically with traditional anonymous credentials as defined in Section 4.2.1: To receive the system credential, the user just generates any pseudonym nym_{join} and sends that to the authority. The authority then has the inspector inspect nym_{join} , receiving some upk . The authority remembers upk , associates this to the user's real identity, and issues a credential to nym_{join} . Non-transferability, in the sense of Definition 4.9, then guarantees that the credential can only be shown with pseudonyms that trace to the same upk , making sure the user is traceable. In practice, one would probably not involve the inspector in this joining process, but rather have the user send upk himself and make him prove that the value is consistent with the one in nym_{join} . In the standard template construction (Section 4.3), this proof can be achieved using a simple zero-knowledge proof system.

4.4.4 Non-transferability via artificial objective measure

Another way to define non-transferability is to say, essentially, “there is a (not realistically computable) function f that outputs the hidden user identity behind each pseudonym”. For example, [LRSW99] extracts the identity behind a pseudonym via rewinding. This solution is very similar to our previous pseudonym inspection solution (where f is the inspection function). f gives us an objective measure whether two pseudonyms belong to the same user, allowing us to define the win condition in the theoretical game similarly to Definition 4.9.

Although theoretically appealing, there is an issue with this approach: There is *no* real-world mechanism to deter sharing credentials by just revealing *usk*. This is in contrast to the pseudonym inspection solution, where we can argue that liability is a deterrent.

Overall, this shows that artificially defining credential non-transferability without any real-world consequences is not a good strategy. It is, however, a good intermediate step towards constructions with meaningful non-transferability, which certainly makes it a useful abstraction.

4.4.5 Non-transferability via outside-of-system meaningful user keys

A popular way to discourage sharing credentials is by making the user secret *usk* an important secret outside-of-system such that users are discouraged sharing their *usk* with anyone [LRSW99; Lys02]. A common example is making *usk* the key to a bank account or to a cryptocurrency wallet, so sharing *usk* involves potentially losing a lot of money. Alternatively, *usk* could be made sufficient for authentication with an important entity, like the government, so sharing *usk* would enable impersonation regarding the government. In any case, giving *usk* to someone else should be associated with major real-world risks.

To make this idea work, we need to formally model that the only way to share a credential is by also sharing *usk*. Note that this is indeed the case for the standard template (Section 4.3): Presenting a credential involves proving knowledge (in the sense of proofs of knowledge) of the user secret *usk* that is embedded in the signature. So anyone able to present the signature is also, in some sense, necessarily able to compute *usk*.

However, an unforgeability definition in the style of Definition 4.9 does not imply this property at all. One seemingly plausible attempt could be to take Definition 4.9 (without the inspector, as discussed above) but we additionally give to the adversary the credentials *cred* that honest users receive from the honest issuer. The rationale would be to model “without *usk*, a credential is worthless”.

This does, however, do nothing to ensure that *malicious* users cannot share some *version* of their credential, that *can* be used without *usk*. For example, with minor details omitted, the degenerate standard template (Section 4.4.1) can be altered such that before the honest user outputs a freshly issued *cred*, he first encrypts it with *usk*. For future procedures involving *cred*, the user first decrypts *cred* with *usk*. This construction would still be considered secure even if we give honest users’ (encrypted) credentials *cred* to the adversary. However, clearly, a malicious user could just hand the decrypted *cred* to another user to be used without the original user’s *usk*.

So there does not seem to be a shortcut to properly modeling the “credential presentation requires knowledge of *usk*”. In order to model this property, we would have to design a definition along the lines of proofs of knowledge, i.e., there is an extractor that can efficiently extract *usk* from a credential presentation process. We do provide such a definition in the context of updatable credentials (Definition 5.3).

Overall, this notion seems fit to discourage sharing *usk*, but requires much heavier definitional machinery to make work in theory. It also requires significant pieces of

infrastructure in the real world (e.g., a bank account that can be accessed through *usk*). Getting the details for this infrastructure right is not easy. Key management is a big challenge in practice, and associating *usk* with even more crucial assets significantly increases the stakes.

4.4.6 All-or-nothing non-transferability

To alleviate some of the implementation challenges with the meaningful outside-of-system user key approach, one may instead implement that user keys are meaningful *inside* the system. Similarly to the previous approach, using shared credentials or pseudonyms requires knowledge of *usk*. Unlike the previous approach, *usk* is not a meaningful secret outside-of-system, but rather enables the holder to use *all* credentials associated with *usk*. This motivates the naming “all-or-nothing non-transferability” [CL01]. To implement this, issuers publish all credentials they issue and pseudonyms they observe, in a way that can be decrypted with *usk* (we omit details [CL01]).

Here, the deterrent against sharing a credential is that this requires sharing *usk*, which enables the recipient to fully impersonate the user within the system. Formally modeling this requires an extraction-based definition to model that using shared credentials requires knowledge of *usk*, as discussed above. Additionally, we require existence of some public algorithm to show that with *usk* and public information, all credentials of the user become usable.

4.4.7 Non-transferability via unclonable hardware

Another way to ensure that *usk* cannot be easily handed to other people is by using secure hardware. The key *usk* would be stored on a special device with specially hardened memory such that *usk* cannot be physically extracted from the device. The device must not release *usk* as part of its operation. Instead, it only releases values from which *usk* cannot be efficiently computed, e.g., g^{usk} or part of a zero-knowledge proof. This means that transferring *usk* can *only* be done by handing the device to someone else. Users are discouraged from doing this because while the device is with someone else, they cannot use it themselves. Even if the user is willing to give up his own use, at the very least there is a significant hurdle in allowing others access, as the device has to be passed around physically. Ideally, the device uses biometric sensors (e.g., a fingerprint reader) to make sure the owner of *usk* is actually present when it is used. In that case, even handing the device to someone else does not enable them to use someone else’s credentials.

This is a great solution for making it difficult to transfer credentials. To model the solution formally, the syntax of anonymous credentials and the unforgeability definition would have to be amended with a model of the secure hardware. We would additionally give the adversary oracle access to the secure hardware’s functions in the unforgeability experiment. The adversary’s goal then is to authenticate without querying the hardware for that process, which should be hard to do.

The main challenge is constructing a traditional anonymous credential system with secure hardware in mind. Naively, we could have the secure hardware play the full

role of the user, storing usk , pseudonyms, as well as credentials, and running all the required protocols itself. However, the design goal for secure hardware is to keep it as simple as possible in order to keep the cost reasonable and in order to avoid introducing security issues creeping into large systems. So the role of the hardware should ideally be limited to storing usk and executing only those exact parts of the protocols that require usk , without revealing usk . For the sake of illustration, consider a bilinear group setting with standard template credentials. We can imagine the hardware would expose an interface like “raise input g to the power of usk ” and “raise inputs $g_1, \dots, g_n$ to a random (hidden) power r ; then on the next invocation with input c , output $r + c \cdot usk$ ”. The former interface enables creating pseudonyms and checking signature (credential) validity. The latter interface is to enable Schnorr proofs involving usk without revealing usk to the user. Please note that there are some more details that need to be taken care of. For example, we must ensure that users are forced to actually use the hardware and cannot join the system without it. Furthermore, some additional care has to be taken to regarding defining zero-knowledge proofs of knowledge in the trusted hardware scenario. Our description here omits those details. Other approaches for implementing hardware-based non-transferability exist, e.g., [BCHKLN15; HS21; FHLL26].

4.4.8 Conclusion

We have seen that if we do not care about having anonymous credentials be transferable, then the standard template for traditional anonymous credentials degenerates into a trivial composition of a pseudonym part and a signature part. That seems to indicate that traditional anonymous credentials either (1) have to make non-transferability a front-and-center feature, or (2) they become very simple, to the point where we might as well forget about “anonymous credentials” and think about pseudonym systems and signatures separately, instead.

We have seen several ways to implement non-transferability. In all of these, non-transferability is achieved by binding usk to a specific user, discouraging him from sharing it (through very different means), and binding credentials to usk . We note, however, that there is a trivial attack where a user A enables another user B to use his access rights. This attack is essentially a relay attack (Section 3.3.2): In order to authenticate, B simply relays all the messages between the A and the verifier. If A is willing to help B , then none of the countermeasures mentioned above can prevent or discourage this. From the point of view of the verifier, it is as if they are talking to A , so authentication works. From the point of view of B , they only see the messages that the verifier sees, so by the anonymity guarantees, B will not learn anything damaging to A during this process.

There may be ways to implement hardware countermeasures to this (like isolating the authenticating user from outside communication, or requiring replies in time faster than an internet round trip), but nevertheless, enforcing real non-transferability is highly nontrivial.

This brings us to the question from Section 4.4.2: Why do we want this at all? If verifiers/issuers are okay with anonymity, meaning that they cannot tell the difference between interacting with the same user twice or with two different users, why would they

not be okay with two different users presenting the same credentials?

The answer to this question will heavily depend on the concrete application. For critical access control applications (e.g., controlling access to a lab with expensive or dangerous equipment), arguably, transferring access rights is actually okay. Indeed, in Chapter 6 we will call this transference “delegation” and consider it a *feature*. What is really important in critical access control scenarios is *accountability*. If lab equipment is destroyed, the person who gave away the access rights should be held responsible. So for this scenario, the pseudonym inspection solution is the right one.

For ticketing applications (e.g., selling bus tickets in the form of anonymous credentials), transferring access rights could be considered an issue. In the degenerate standard template, credentials are not bound to any user. So what could happen is that one person buys a ticket and then publishes it for free on the internet for anyone to use. Non-transferability (with any of the solutions discussed above) of course fixes this issue. But if we look closely, the *actual* property that ticketing applications require is not non-transferability, but rather *rarity* or *uniqueness*: It must not be possible for multiple people to gain entry on the same ticket. It is not an issue if a user gives/sells his ticket to someone else, as long as only one person can use the ticket. Making single-use tickets from anonymous credentials is something we will see in Chapter 5.

Note that none of the credential sharing countermeasures proposed above are *unconditionally* effective — a dedicated user can, potentially at cost, circumvent them and pass their credentials to someone else. For applications where verifiers need to make 100% sure that they have the right person (say, for immigration at an airport), this is an issue. For those applications, the *goal* is exact identification of the user. They are set up so that the credential’s attributes uniquely identify a person. So those applications should just have the user disclose sufficiently many attributes to identify them uniquely (e.g., name, birthdate, photo, fingerprint). In this sense, they enforce non-transferability with attributes: If I hand my passport to someone else, they will not be able to use it since they do not share my appearance or fingerprint. This way, those applications implement non-transferability themselves, with no need for pseudonyms or any of the countermeasures listed above. They can still profit from anonymous credentials, e.g., to hide certain attributes not required for identification. However, they should consider using the framework we lay out in Chapter 5, where we do not consider pseudonyms an integral part of anonymous credentials anymore. Some works have also explored settings where credentials are tied to users via biometrics (similar to the fingerprint attribute above), but where additional infrastructure keeps the identifying biometrics from the verifier [Ada11; HSS23; RKS24]. These are interesting and powerful suggestions, but out of scope for our *usk*-centric discussion here.

To conclude, when deploying traditional anonymous credentials, we suggest to critically evaluate whether non-transferability is required at all. If not, the application goals can probably be achieved simpler than with traditional anonymous credentials (e.g., with (non-traditional) anonymous credentials as we define them Chapter 5). If non-transferability is indeed what is needed, then you need to be prepared to set up the additional infrastructure for one of the non-transferability solutions above.

4.5 A short history of traditional anonymous credentials

When Chaum first suggested the idea of pseudonyms and anonymous credentials [Cha84; Cha85], no concrete construction was known. We discuss different constructions for anonymous credentials in this section.

The early days. Some early results took a theoretical route, proving feasibility of constructing provably secure anonymous credentials. In a construction surprisingly close to today’s standard template, Damgård [Dam88] uses (bit) commitments on a user identity to generate pseudonyms, a generic MPC protocol to blindly issue a signature on the committed user identity, and another generic MPC protocol to present the signature for another pseudonym. Later on, this construction was adapted [LRSW99] to replace generic MPC with merely generic zero-knowledge proofs (by signing the pseudonym instead of the secret user identity). Due to their reliance on generic MPC or generic zero-knowledge proofs, these theoretical constructions were not considered practical.

Another branch of research attempted to build practically efficient anonymous credential systems. Early attempts for practically efficient constructions followed the intuition laid out by Chaum [Cha84; Cha85], where a credential is a signature on a pseudonym with some mechanism to transfer the credential to another pseudonym of the same user. Early efficiency-oriented works [CE86; Bra96; Che95; LRSW99; Ver01] suffered from several restrictions. Typical restrictions include: strong assumptions, a need for additional trusted parties, lack of formal security proofs, lack of multi-show unlinkability (privacy when presenting the same credential multiple times), or lack of support for attributes.

The emerging standard template. The breakthrough for practically efficient anonymous credential systems was arguably [CL01], which builds a provably secure anonymous credential system under the RSA and DDH assumptions, with none of the restrictions mentioned above. Shortly thereafter, [CL02] effectively rephrased [CL01] in terms of what we now consider the *standard template* (Section 4.3). Using signatures with algebraic structure (often called “CL signatures”), the required protocols for blindly signing a committed value and proving knowledge of a signature were easy to instantiate without resorting to generic MPC techniques.

This template has been highly influential, with many later constructions following it. Notably, [CL04] replaces the RSA-based signature scheme with one based on prime order groups with bilinear maps, which significantly improves concrete performance compared to the RSA variant. Technically, they use the signature scheme implicitly introduced in [LRSW99], but cast it into the pairing setting, which makes signatures publicly verifiable. This brings multi-show unlinkability to [LRSW99]. Additionally, [CL04] sketches how to instantiate the standard template using BBS signatures [BBS04], which is a popular way to implement anonymous credentials to this day. [BCKL08] instantiates the standard template with Groth-Sahai proofs [GS08] and a corresponding signature scheme.

Over the years, many works have followed the standard template in spirit, using with custom protocols for signing committed values and proving knowledge of signatures. Our generalizations (Chapters 5 and 6) broadly follow the standard template, too, with some

modifications. Several signature schemes compatible with the standard template have been developed [CL02; CL04; BBS04; BCKL08; PS16; BLNS23; AGJLRS24; Gro15].

Other ways of constructing anonymous credentials. While most constructions tend to follow the standard template, thanks to its excellent extensibility, we want to briefly highlight alternative approaches.

One approach is using structure-preserving signatures on equivalence classes (SPSEQ; [FHS19; BF20]). In this approach, the issuer signs a set commitment to the user's attributes. The user can jointly randomize the commitment and the signature on the commitment, creating an unlinkable copy of the credential that can be sent in the clear. One advantage of this approach is concrete and asymptotic efficiency: The signature and a subset of the user's attributes can be verified without resorting to a zero-knowledge proof. Pseudonyms can be implemented in such constructions as described in [CL19].

In settings where it is acceptable that the verifier holds a copy of the issuer's key (e.g., the issuer and verifier are the same entity), keyed-verification anonymous credentials (KVAC; [CMZ14; CPZ20; MBSBA+25]) can be used. The advantage of KVACs is that they can be constructed in plain prime order groups without pairings. To mitigate the downsides of KVACs and make them more practical, there are approaches to derive publicly verifiable one-time presentation tokens from KVACs using a helper [CHLT25] and to issue a designated-verifier KVAC while hiding the designated verifier from the issuer [BBLMÖS25].

Other approaches include the following. Instead of issuing individual signatures to certify attributes, user attributes are encoded within a publicly accessible and regularly updated Merkle tree [RWGM23]. A similar approach is followed by [GGM14], where valid committed user data is simply appended to some public data structure. One can also construct anonymous credentials from sanitizable signatures, implementing partial disclosure of attributes by sanitizing parts of the signature the verifier shall not see [CL13].

Chapter 5

Updatable anonymous credentials

It's no use going back to yesterday,
because I was a different person then.

Alice

In this chapter, we extend traditional anonymous credential systems (Chapter 4) to *updatable* anonymous credential systems (UACS). This solves several issues with traditional anonymous credentials and even unlocks new applications, as we will detail in the following paragraphs.

Traditional anonymous credentials systems are very good at hiding user attributes from *verifiers*: Users can disclose that some statement ϕ is true about their attributes, not revealing anything else. In contrast, traditional anonymous credential systems do not hide user attributes from *issuers* at all: When being issued a credential, the user's attributes are fully known to the issuer. This is a natural requirement in most typical authentication scenarios: The issuer's task is to attest to the user's attributes (such as name, age, position in organizational hierarchy, etc.), which the issuer cannot do effectively if she were not aware of what attributes are being issued.

There are, however, scenarios, where the issuer does not necessarily *have to* have full information about the user's attributes. Take, for example, a scenario in which anonymous credentials are used to control access to a subscription service. The user's attributes are $\vec{attr} = (\mathbf{name}, \mathbf{subscriptionend})$. In order to access the service, the user proves $\phi(\vec{attr}) \equiv (\mathbf{subscriptionend} > \mathbf{now})$. Say the user buys an extension to his subscription, adding another month of service on top of the currently running subscription. For this, the user would currently have to approach the issuer, reveal attributes $(\mathbf{name}, \mathbf{subscriptionend})$ to her, and prove possession of a credential with those attributes. Then the issuer would issue a new credential with attributes $\vec{attr}' = (\mathbf{name}, \mathbf{subscriptionend} + 1 \text{ month})$. The security definitions for traditional anonymous credential guarantee that the issuer does not learn the exact identity of the user, only the value of his attributes. However, in this case, the attributes are clearly identifying: They contain the user's name $\mathbf{name}$ and even $\mathbf{subscriptionend}$ may be unique by itself, e.g., if it is a timestamp with millisecond resolution.

Note that in the subscription example, there is no functional requirement for the

issuer to be aware of the user's **subscriptionend** (there is no need to check it against real-world data). Functionally, the only thing that is happening is that the user with attributes $(\text{name}, \text{subscriptionend})$ acquires a new credential with updated attributes $(\text{name}, \text{subscriptionend} + 1 \text{ month})$. Updatable anonymous credentials allow the user and issuer to execute such an update in a privacy-preserving way, i.e., the issuer does not learn anything about the user's old or new attributes. The issuer only knows what update operation is being executed, in this case a 1-month extension.

Modeling updates and hidden attributes. A credential in a UACS authenticates attributes $\vec{attr} \in \mathbb{A}^n$. To update the credential, the user approaches the issuer. Both parties then agree on an update function $\psi : (\mathbb{A}^n \times \{0, 1\}^*) \rightarrow (\mathbb{A}^n \cup \{\perp\})$ and run the update protocol. The updated credential will have attributes $\vec{attr}' = \psi(\vec{attr}, \alpha)$, where $\vec{attr}$ are the user's old attributes and α is some secret parameter chosen freely by the user (subject to $\psi(\vec{attr}, \alpha) \neq \perp$, meaning that ψ can encode requirements on $\vec{attr}, \alpha$). During the update process, the issuer only learns the update function ψ , but not $\vec{attr}$, not α , and not $\vec{attr}'$ (except that $\psi(\vec{attr}, \alpha) \neq \perp$). Our subscription update example can be described with $\psi((\text{name}, \text{subscriptionend}), \alpha) = (\text{name}, \text{subscriptionend} + 1 \text{ month})$. In that particular example, the user's secret parameter α is ignored by ψ .

We view issuing a fresh credential similar to updating an “empty” credential, i.e., issuer and user agree on an update function $\psi : \{\perp\} \times \{0, 1\}^* \rightarrow (\mathbb{A}^n \cup \{\perp\})$, the user chooses α freely, and the issued credential will have attributes $\psi(\perp, \alpha)$ (except if $\psi(\perp, \alpha) = \perp$, in which case issuing should fail).

An overview of the processes regarding UACS is depicted in Figure 5.1.

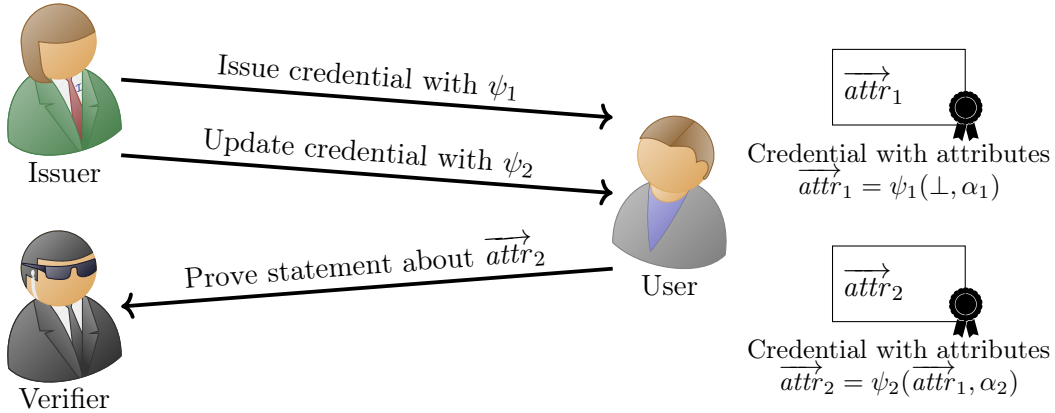


Figure 5.1: Overview of the protocols in updatable anonymous credential systems.

The challenges and power of hidden attributes. With UACS, we now have the ability to describe *attributes* that are hidden, even during the issuance process. While hidden attributes may sound like a simple extension to traditional anonymous credentials, it comes with significant implications.

The first implication is that we can now simply treat the user’s identity as one such hidden UACS attribute. To issue a credential containing the user’s secret key usk and attribute A , under the condition that usk is consistent with a pseudonym nym (Section 4.1), one may use the update function

$$\psi(\perp, \alpha = (usk, nsk)) = \begin{cases} (usk, A) & \text{if } \text{VerifyNym}(pp, usk, nym, nsk) = 1 \\ \perp & \text{otherwise} \end{cases}$$

where the secret input α contains the user’s secret key usk and the pseudonym’s key nsk . The update function ψ in this case depends on pp, nym, A , which can be considered hardcoded into ψ .

As a consequence, binding credentials to identities in a privacy-preserving way can be seen as a simple application of the natural abilities of UACS instead of the tightly integrated feature it represents for traditional anonymous credentials. For this reason, and because, as discussed in Section 4.4.8, binding credentials to identities is not universally useful in all settings, we describe UACS *without* any built-in mechanism to bind credentials to identities. Instead, we simply allow system developers to optionally bind credentials to identities using UACS’s hidden attribute mechanism, if desired, as showcased above. This simplifies the description of UACS tremendously. While for traditional anonymous credentials, a lot of definitional complexity comes from managing user identities and pseudonyms, those aspects naturally become an optional add-on for UACS.

The second implication of hidden attributes is that we need to fundamentally rethink the unforgeability property of UACS compared to traditional anonymous credentials. For the latter, the unforgeability property essentially boils down to “the adversary wins if he can prove possession of attributes that the honest issuer has never issued to him”. But for UACS, in contrast to traditional anonymous credentials, the issuer *does not necessarily know* what exact (hidden) attributes she issues to users, or what attributes result from an update operation. This means that the previous strategy for checking the unforgeability winning condition, namely iterating over all issued attributes, is inherently not possible for UACS. For this reason, we need to rethink the unforgeability property for UACS from the ground up. Our solution to this problem borrows definitional techniques from Chapter 3, making issued attributes visible in the unforgeability experiment through a process similar to proof of knowledge extraction.

Invalidating old credentials. One very useful example of hidden attributes are *nullifiers*. By default, updating a credential just creates a new credential, it does not invalidate the old one. A simple mechanism to invalidate a credential is by embedding a random nullifier attribute **nfy** into it and then having the user reveal its value to invalidate it.

As an illustrative example, consider a simple voucher application: Users buy (with some non-anonymous means like a credit card) *points* and then get to anonymously spend these points on products or services. In this example, a credential’s attributes are (**nfy**, **pts**), where **nfy** uniquely identifies this particular credential, and **pts** are the number of points that the credential is worth. After the user pays for, say, 50 points, the

user and issuer run the issuance protocol with update function

$$\psi(\perp, \alpha = \mathbf{nfy}) = (\mathbf{nfy}, 50),$$

where $\mathbf{nfy} \leftarrow \mathbb{A}$ is picked randomly by the user and kept hidden from the issuer. As a result, the user now holds a valid credential $cred$ with attributes $(\mathbf{nfy}, 50)$. Using $cred$, the user can then spend, say, 20 of his points as follows.

- The user sends his nullifier $\mathbf{nfy}_{\text{reveal}} = \mathbf{nfy}$ to the issuer.
- The issuer checks that $\mathbf{nfy}_{\text{reveal}}$ has not been revealed before (by consulting a database of all nullifiers revealed to her so far).
- Then user and issuer run the update protocol to blindly update the credential with the updated point count $\mathbf{pts}' = \mathbf{pts} - 20$ and a fresh nullifier $\mathbf{nfy}'$. Concretely, they use the update function

$$\psi((\mathbf{nfy}, \mathbf{pts}), \alpha = \mathbf{nfy}') = \begin{cases} (\mathbf{nfy}', \mathbf{pts} - 20) & \text{if } \mathbf{nfy} = \mathbf{nfy}_{\text{reveal}} \wedge \mathbf{pts} \geq 20 \\ \perp & \text{otherwise} \end{cases}$$

where $\mathbf{nfy}' \leftarrow \mathbb{A}$ is picked randomly by the user and kept hidden from the issuer. In other words, if the revealed $\mathbf{nfy}_{\text{reveal}}$ matches the $\mathbf{nfy}$ in the user's credential and the user has enough points, then the protocol succeeds and the user receives a new credential with attributes $(\mathbf{nfy}', \mathbf{pts} - 20)$ for the remainder amount of his points with a fresh nullifier $\mathbf{nfy}'$.

- If the protocol succeeds, the issuer considers the 20 points spent and provides the corresponding goods or services to the user.

In this process, the old credential (with attributes $(\mathbf{nfy}, 50)$) is effectively invalidated: If the user reveals $\mathbf{nfy}_{\text{reveal}} = \mathbf{nfy}$ to the issuer *again*, she will refuse to run the update protocol. If the user reveals some fake unused nullifier $\mathbf{nfy}_{\text{reveal}} \neq \mathbf{nfy}$ for which he does not own a credential, then the update protocol will fail ($\psi(\dots) = \perp$). This is crucial because the user must not be able to keep using his initial 50 points credential, given that he is actually left with only 30 points. Assuming $\mathbb{A}$ is large enough (super-poly), it is exceedingly unlikely that an honest user's choice of $\mathbf{nfy}' \leftarrow \mathbb{A}$ collides with any nullifier already in the issuer's database, meaning that honest users will be able to redeem their points with overwhelming probability.

There are alternative approaches to invalidating credentials, useful in different scenarios. One example are n -times unlinkable presentations [CHKLM06], where a credential can be presented n times (where $n = 1$ yields the semantics of single-time use as above). For this, the user embeds a random key $\mathbf{k}$ for a pseudorandom function F as an attribute into his credential. When using the credential, he reveals a nullifier of the form $\mathbf{nfy} = F(\mathbf{k}, i)$ to the verifier, where $i \in [n]$, and has the presentation predicate ϕ verify that the nullifier was correctly computed. This effectively allows the user to derive n different pseudorandom nullifiers for his credential, making the credential usable n times in an unlinkable manner.

Other approaches enable *offline* double-spending protection [CFN88; CL01; BBDE19], allowing verifiers to speculatively accept a credential’s nullifier without immediately checking it against the database of known nullifiers. If it later turns out that the user has exceeded the usage limit (used the same nullifier twice), the user’s identity is revealed and verifiers can take steps to remedy the situation (e.g., demand additional payment). Technically, this is achieved by having users additionally embed a long-term identifying secret usk (similar to our pseudonym mechanism above) and a random secret value $r \leftarrow \mathbb{A}$ into his credential. When presenting his credential, the user reveals $c \cdot usk + r$ for a fresh verifier-chosen challenge $c \leftarrow \mathbb{A}$ (where $\mathbb{A} = \mathbb{Z}_p$ in this example). A user presenting the same credential twice, with different random challenges c, c' will reveal $s = c \cdot usk + r$ and $s' = c' \cdot usk + r$, which allows the verifier to compute $usk = (s - s') / (c - c')$, identifying the offending user.

Applications beyond access control. An exciting aspect of UACS is that they enable *new* privacy-preserving applications, not covered by traditional anonymous credentials. Essentially, traditional anonymous credential systems allow issuers to store static user data (attributes) with the user instead of storing it in a centralized database, with the guarantee that users cannot change their data themselves. Users can then choose to reveal some predicate about their data to verifiers in a privacy-preserving way. *Updatable* anonymous credentials go one step further: In addition to outsourcing and “reading” user data, user data can now also be *modified* in a privacy-preserving manner. This enables a much wider range of privacy-preserving applications beyond access control, for which user data modification is a requirement.

One major example for such an application can be found with privacy-preserving point collection, such as loyalty/incentive systems (e.g., frequent flyer miles). These can be implemented [BBDE19] like the voucher system sketched above, with an additional update operation to *add* points (rather than start with a set number and just subtract). Similarly, one can implement simple anonymous payment (e.g., for parking fees) using updatable credentials [YXZHLD24].

Other examples can be found in other applications that store dynamic user data. We could imagine a public forum with reputation/karma points in the style of Stack Overflow or Reddit. Instead of storing reputation points in a central database for all users, users can keep their reputation score stored in an updatable credential. This means that users do not have to identify themselves to the forum, but can still participate by proving possession of sufficient reputation (e.g., enough to post a question). Using the update function, users can also have the forum update their reputation score without revealing their identity to the forum. Other mechanisms, e.g., to curb spam, can be built into the system using the extensible nature of UACS.

Contribution. In this chapter, we introduce updatable anonymous credentials. We define their syntax and security formally in Section 5.2. We show how to construct UACS from (zero-knowledge) proof systems and signature schemes with efficient protocols in Section 5.3. As a simple application example, we then construct a traditional anonymous

credential system (with pseudonyms) from UACS in Section 5.4.

This chapter is based on the paper [BBDE19], for which Fabian Eidens and Jan Bobolz contributed the idea, as well as the definitions and standard template construction. Denis Diemert contributed the benchmark implementation of the incentive system built in that paper. Fabian Eidens focused on the original anonymity proof, whereas Jan Bobolz focused on the original soundness proof (whose counterpart we call “simulation-explainability” in this thesis). Since the focus of this thesis is more broadly anonymous authentication (whereas [BBDE19] mostly focused on the specific UACS use case of an incentive system), we present completely reworked and heavily improved security notions (and, consequently, their corresponding security proofs) in this thesis. We discuss some notable differences in Sections 5.2.2 and 5.2.3 (including concurrent protocol execution and the presence of honest users when attacking issuers). Additionally, the instantiation of traditional anonymous credentials from UACS (Section 5.4) has been specifically developed for this thesis and is independent of [BBDE19].

5.1 Related work

As far as we are aware, there were no formalizations of updatable credentials comparable to ours in the literature prior to our work [BBDE19]. Prior to our paper, [NDD06; DD05] developed a similar idea, but without providing formal security definitions or proofs. There are schemes that allow for updates to anonymous credentials for attributes revealed to the issuer [CKS10]. This mechanism is used for revocation. There is no support for privacy-preserving issuance or update of hidden attributes, which we provide in this chapter.

Other works have suggested carrying over attributes from one credential to another [CKLMNP15], which one may view as an update that simply copies attributes. Stateful anonymous credentials [CHL05; CGH11] enable privacy-preserving update of a state according to some public state machine. This is a special case of our update function.

We want to particularly highlight recent work [SRKM25], which shares our vision of using anonymous credentials to store (and update) generic user data. Their idea is to allow user A to asynchronously submit an update to the data of user B, say, A downvotes a forum post by B. They devise a mechanism that forces user B to accept the update (even if it is negative).

Finally, we note that many extensions to anonymous credentials are compatible and can be easily implemented using our UACS formalism. One example for this is the inclusion of pseudonyms (and non-transferability), which we discuss in Section 5.4.

5.2 Defining updatable anonymous credentials

In this section, we define syntax and security for updatable anonymous credential systems.

5.2.1 Syntax

As always, we start with the syntax. Compared to traditional anonymous credentials (Definition 4.7), we add a new protocol for updating a credential. We also remove pseudonyms from the system. As a (more general) replacement, the protocol for issuing a credential now allows issuance of hidden attributes (fulfilling some basic requirements) using an update function ψ , as explained above. This can be used to implement a traditional anonymous credential system by using the user's secret key as a hidden attribute, as we demonstrate formally in Section 5.4.

Definition 5.1 (Updatable ACS). An updatable anonymous credential system Π_{uacs} with attribute space $\mathbb{A} = \mathbb{A}_{\mathcal{G}}$, update function space $\Psi = \Psi_{\mathcal{G}} \subset \{\psi : (\mathbb{A}^n \cup \{\perp\}) \times \{0, 1\}^* \rightarrow (\mathbb{A}^n \cup \{\perp\}) \mid n \in \mathbb{N}\}$, and predicate space $\Phi = \Phi_{\mathcal{G}} \subset \{\phi : \mathbb{A}^n \times \{0, 1\}^* \rightarrow \{0, 1\} \mid n \in \mathbb{N}\}$, consists of the following ppt algorithms and protocols.

$\text{Setup}(\mathcal{G}) \rightarrow pp$

Generates public parameters pp .

$\text{IssuerKeyGen}(pp, 1^n) \rightarrow (ipk, isk)$

Generates issuer public key ipk and corresponding secret key isk . $\mathfrak{V}$ ipk supports attribute vectors of length n .

$cred' \leftarrow \text{ReceiveCred}(pp, ipk, \psi, ctx; \alpha) \leftrightarrow \text{IssueCred}(pp, ipk, \psi, ctx; isk) \rightarrow b$

An interactive protocol. $\mathfrak{V}$ Run by the user to receive a credential $cred'$ with attributes $\overrightarrow{attr'} = \psi(\perp, \alpha)$. The issuer outputs a bit b indicating $\psi(\perp, \alpha) \neq \perp$.

$cred' \leftarrow \text{ReceiveUpdate}(pp, ipk, \psi, ctx; \overrightarrow{attr}, cred, \alpha) \leftrightarrow \text{UpdateCred}(pp, ipk, \psi, ctx; isk) \rightarrow b$

An interactive protocol. $\mathfrak{V}$ Run by the user to receive an updated credential $cred'$ with attributes $\overrightarrow{attr'} = \psi(\overrightarrow{attr}, \alpha)$ based on his existing credential $cred$ with attributes $\overrightarrow{attr}$. The issuer outputs a bit b indicating $\psi(\overrightarrow{attr}, \alpha) \neq \perp$.

$\text{PresentCred}(pp, ipk, \phi, ctx; \overrightarrow{attr}, cred, \alpha) \leftrightarrow \text{VerifyCred}(pp, ipk, \phi, ctx) \rightarrow b$

An interactive protocol. $\mathfrak{V}$ Run by the user to convince the verifier of possession of a credential. The verifier outputs a bit b indicating $\phi(\overrightarrow{attr}, \alpha) = 1$.

$\text{CheckCred}(pp, ipk, \overrightarrow{attr}, cred) = b$

A deterministic algorithm that outputs a bit b . $\mathfrak{V}$ It checks that $cred$ is a valid credential from issuer ipk for attributes $\overrightarrow{attr}$.

An updatable anonymous credential system is *correct*, which means that for all $n, \lambda \in \mathbb{N}$, $\mathcal{G} \in [\text{GenSetting}(1^\lambda)]$, $pp \in [\text{Setup}(\mathcal{G})]$, $(ipk, isk) \in [\text{IssuerKeyGen}(pp, 1^n)]$, and all $\overrightarrow{attr} \in \mathbb{A}^n$, $\phi \in \Phi_{\mathcal{G}}$, $\psi \in \Psi_{\mathcal{G}}$, $\alpha \in \{0, 1\}^*$, $ctx \in \{0, 1\}^*$ and all $cred$,

- “Issuing results in a credential on $\overrightarrow{attr}' = \psi(\perp, \alpha)$ ”: If $\psi(\perp, \alpha) \neq \perp$, then

$$\Pr \left[\begin{array}{l} b = 1 \wedge \\ \text{CheckCred}(pp, ipk, \\ \psi(\perp, \alpha), cred') = 1 \end{array} \parallel \begin{array}{l} cred' \leftarrow \text{ReceiveCred}(pp, ipk, \psi, ctx; \alpha) \\ \leftrightarrow \text{IssueCred}(pp, ipk, \psi, ctx; isk) \rightarrow b \end{array} \right] = 1$$

- “Updating results in a credential on $\overrightarrow{attr}' = \psi(\overrightarrow{attr}, \alpha)$ ”: If $\psi(\overrightarrow{attr}, \alpha) \neq \perp$ and $\text{CheckCred}(pp, ipk, \overrightarrow{attr}, cred) = 1$, then

$$\Pr \left[\begin{array}{l} b = 1 \wedge \\ \text{CheckCred}(pp, ipk, \\ \psi(\overrightarrow{attr}, \alpha), cred') = 1 \end{array} \parallel \begin{array}{l} \vdash cred' \\ \text{ReceiveUpdate}(pp, ipk, \psi, ctx; \overrightarrow{attr}, cred, \alpha) \\ \leftrightarrow \text{UpdateCred}(pp, ipk, \psi, ctx; isk) \rightarrow b \end{array} \right] = 1$$

- “Presenting a credential works”: If $\phi(\overrightarrow{attr}, \alpha) = 1$ and $\text{CheckCred}(pp, ipk, \overrightarrow{attr}, cred) = 1$, then

$$\Pr \left[b = 1 \parallel \begin{array}{l} \text{PresentCred}(pp, ipk, \phi, ctx; \overrightarrow{attr}, cred, \alpha) \\ \leftrightarrow \text{VerifyCred}(pp, ipk, \phi, ctx) \rightarrow b \end{array} \right] = 1$$

In traditional anonymous credentials, there is no hidden parameter α chosen by the user. For UACS, α can be used by the user for receiving/updating a credential as additional hidden input to the update function ψ . This allows the user to embed hidden attributes, such as their identity. It also enables users to use secret information for validating a claim necessary for the update (modeled as $\psi(\overrightarrow{attr}, \alpha) \neq \perp$), such as the opening value for a commitment. For this reason, UACS also have α as input to presentation predicates $\phi(\overrightarrow{attr}, \alpha)$.

One of our security properties (Definition 5.3) will ensure that the update protocol $\text{ReceiveUpdate} \leftrightarrow \text{UpdateCred} \rightarrow b$ doubles as a presentation of the user’s old credential. By this, we mean that *presenting* a credential for policy ϕ can be emulated as an update

$$\psi(\overrightarrow{attr}, \alpha) = \begin{cases} \overrightarrow{attr} & \text{if } \phi(\overrightarrow{attr}, \alpha) = 1 \\ \perp & \text{otherwise} \end{cases}$$

If UpdateCred outputs 1, the issuer can be sure that the user is in possession of a credential with attributes $\overrightarrow{attr}$ for which $\psi(\overrightarrow{attr}, \alpha) \neq \perp$, i.e., $\phi(\overrightarrow{attr}, \alpha) = 1$.

5.2.2 Anonymity

Our anonymity definition of traditional anonymous credential (Definition 4.8) very much revolves around identities: We set up two user identities and a virtual “mystery” user, who is one of the two set up users. We then let the adversary interact with the three of them in any way that does not trivially leak the identity of the mystery user. This is very appropriate for traditional anonymous credentials: It models that an adversary can only gain information about an unknown user if that unknown user presents access rights that

only one of the users has. However, UACS do not revolve around identity at all, there is not even a notion of a user identity or key. Instead, our UACS anonymity definition revolves around the user's secret input to the issue/update/presentation protocols.

Naively, we could imagine a definition of the form “given $(\vec{attr}_0, cred_0, \alpha_0)$ and $(\vec{attr}_1, cred_1, \alpha_1)$ such that both are valid user inputs, the adversary cannot tell which of the two is used for a presentation protocol run”. At first glance, this kind of indistinguishability definition works perfectly fine. However, in some important cases, like when the predicate ϕ is such that there is only a single possible value for $\vec{attr}$ with $\phi(\vec{attr}, \cdot) = 1$, this indistinguishability notion does not provide meaningful security guarantees. Indistinguishability only guarantees that one cannot distinguish a protocol run with valid input a from one run with valid input b , but if the only valid input is $a = b = x$, then the statement is trivial.

This issue even occurs for the seemingly simple extension of implementing pseudonyms with UACS (see Section 5.4): Because for every pseudonym nym , there may be at most one fitting user secret usk (e.g., Construction 4.5), we run into that issue and the sketched indistinguishability definition is not meaningful or usable.

Because we want UACS to be a useful and generically extensible building block (something that is typically not an ambition of definitions for traditional anonymous credentials), we need a stronger anonymity definition. For this reason, we define anonymity via simulation, similar to the zero-knowledge property in Chapter 3. The definition will be, roughly, of the form “there exists a simulator that can execute the user side of the issue/update/presentation protocol convincingly *without* being given $\vec{attr}$, $cred$, or α ”.

In practice, this means that anything an issuer/verifier observes during an interaction with the user can just as well be simulated without the user's data. So intuitively, what the issuer/verifier observes does not reveal anything about the user's data $\vec{attr}, \alpha$ beyond what can be inferred from $\psi(\vec{attr}, \alpha) \neq \perp$ (or from $\phi(\vec{attr}, \alpha) = 1$ for the credential presentation case).

The simulation-based definition is meaningful even if there is only a single possible valid $\vec{attr}$ (or α) for issuance/presentation. It enables pseudonyms as a generic extension, as we demonstrate in Theorem 5.15. In general, simulation-based definitions tend to be more powerful than their indistinguishability counterparts (in the sense that any scheme that is secure in a simulation-based sense is also secure in an indistinguishability sense, but not necessarily vice versa).

More formally, Definition 5.2 is reminiscent of the zero-knowledge property (e.g., Definition 3.22): We require existence of a simulator such that no ppt adversary can distinguish interactions with the simulator from interactions with honestly executed protocols. The adversary takes the role of corrupted issuers and verifiers, whereas the experiment (and simulator) takes the role of honest users.

Definition 5.2 (Simulation anonymity). Let $\mathcal{S}$ be a stateful interactive algorithm and let $\mathcal{A}$ be a ppt algorithm. Consider the following game $\text{Exp}_{\Pi_{\text{uacs}}}^{\text{sim-anon}}(\Pi_{\text{uacs}}, \mathcal{S}, \mathcal{A}, \lambda, X)$ for $X \in \{\text{REAL}, \text{SIM}\}$.

$\text{Exp}_{\Pi_{\text{uacs}}}^{\text{sim-anon}}(\Pi_{\text{uacs}}, \mathcal{S}, \mathcal{A}, \lambda, X)$

If $X = \text{REAL}$:

$pp \leftarrow \text{Setup}(\mathcal{G})$

If $X = \text{SIM}$:

$pp \leftarrow \mathcal{S}.\text{Setup}(\mathcal{G})$ //Simulator gets to set up pp

Run $b \leftarrow \mathcal{A}^{\mathcal{O}.\mathcal{RO}^X, \mathcal{O}.\text{ReceiveCred}^X, \mathcal{O}.\text{ReceiveUpdate}^X, \mathcal{O}.\text{PresentCred}^X}(\mathcal{G}, pp)$

return b

$\mathcal{O}.\mathcal{RO}^X(x)$ //Random oracle access

If $X = \text{REAL}$:

return $\mathcal{RO}(x)$

If $X = \text{SIM}$:

return $\mathcal{S}.\mathcal{RO}(x)$ //Simulator gets to answer random oracle queries

$\mathcal{O}.\text{ReceiveCred}^X(ipk, \psi, ctx, \alpha)$ //Have honest user receive credential

assert $\psi(\perp, \alpha) \neq \perp$

Choose new credential handle C'

Set $ipk_{C'} := ipk$ and $\overrightarrow{attr}_{C'} := \psi(\perp, \alpha)$ and $cred_{C'} := \perp$

If $X = \text{REAL}$:

$cred_{C'} \leftarrow \text{ReceiveCred}^{\mathcal{RO}}(pp, ipk, \psi, ctx; \alpha) \leftrightarrow \mathcal{A}$

If $X = \text{SIM}$:

$b \leftarrow \mathcal{S}.\text{ReceiveCred}(pp, ipk, \psi, ctx) \leftrightarrow \mathcal{A}$

If $b = 1$, set $cred_{C'} = 1$ //Mark as a valid credential $cred_{C'} \neq \perp$

return C' if $cred_{C'} \neq \perp$.

$\mathcal{O}.\text{ReceiveUpdate}^X(C, \psi, ctx, \alpha)$ //Have honest user receive credential update

assert $cred_C \neq \perp$ and $\psi(\overrightarrow{attr}_C, \alpha) \neq \perp$

Choose new credential handle C'

Set $ipk_{C'} := ipk_C$ and $\overrightarrow{attr}_{C'} := \psi(\overrightarrow{attr}_C, \alpha)$ and $cred_{C'} := \perp$

If $X = \text{REAL}$:

$cred_{C'} \leftarrow \text{ReceiveUpdate}^{\mathcal{RO}}(pp, ipk_C, \psi, ctx; \overrightarrow{attr}_C, cred_C, \alpha) \leftrightarrow \mathcal{A}$

If $X = \text{SIM}$:

$b \leftarrow \mathcal{S}.\text{ReceiveUpdate}(pp, ipk_C, \psi, ctx) \leftrightarrow \mathcal{A}$

If $b = 1$, set $cred_{C'} = 1$ //Mark as a valid credential $cred_{C'} \neq \perp$

return C' if $cred_{C'} \neq \perp$.

$\mathcal{O}.\text{PresentCred}^X(C, \phi, ctx, \alpha)$ //Have honest user present credential

assert $cred_C \neq \perp$ and $\phi(\overrightarrow{attr}_C, \alpha) = 1$

If $X = \text{REAL}$:

$\text{PresentCred}^{\mathcal{RO}}(pp, ipk_C, \phi, ctx; \overrightarrow{attr}_C, cred_C, \alpha) \leftrightarrow \mathcal{A}$

If $X = \text{SIM}$:

$\mathcal{S}.\text{PresentCred}(pp, ipk_C, \phi, ctx) \leftrightarrow \mathcal{A}$

An updatable anonymous credential system Π_{uacs} has *simulation anonymity* if there exists a stateful interactive algorithm $\mathcal{S}$ such that for all ppt $\mathcal{A}$,

$$\left| \begin{array}{l} \Pr[\text{Exp}_{\Pi_{\text{uacs}}}^{\text{sim-anon}}(\Pi_{\text{uacs}}, \mathcal{S}, \mathcal{A}, \lambda, \text{REAL}) = 1] \\ - \Pr[\text{Exp}_{\Pi_{\text{uacs}}}^{\text{sim-anon}}(\Pi_{\text{uacs}}, \mathcal{S}, \mathcal{A}, \lambda, \text{SIM}) = 1] \end{array} \right|$$

is negligible and the runtime of $\text{Exp}_{\Pi_{\text{uacs}}}^{\text{sim-anon}}(\Pi_{\text{uacs}}, \mathcal{S}, \mathcal{A}, \lambda, \text{SIM})$ is polynomially bounded (where the polynomial depends on $\mathcal{A}$).

Notably, for $\mathcal{O}.\text{ReceiveCred}^{\text{IDEAL}}$ and $\mathcal{O}.\text{ReceiveUpdate}^{\text{IDEAL}}$, the simulator does not need to output the issued/updated credential after the interaction with the adversarial issuer. We cannot possibly expect the simulator to do this because we do not provide the attributes the credential is supposed to be valid for. However, the simulator is expected to simulate the validity bit b of the received credential. In reality, an issuer can often easily infer whether the user accepts the credential ($\perp \neq cred \leftarrow \text{ReceiveCred}$) because, for example, the user may retry the issuance/update, or fail future presentation checks. Forcing the simulator to output the validity bit models that the issuer does not gain any information about the user's hidden values from this validity bit.

The anonymity definition in our original paper [BBDE19] only considers a single run of a protocol and requires information-theoretically perfect simulation, instead of giving an adversary oracle access to real/simulated protocol executions and only require computationally indistinguishable simulation like we do in Definition 5.2. Any scheme that is anonymous w.r.t. the original paper's definition is also anonymous w.r.t. Definition 5.2. However, Definition 5.2 is much easier to fulfill, meaning that it allows for a wider range of constructions to be considered secure (without compromising “real world” security implications). Essentially, the definition in [BBDE19] calls for *perfectly* simulatable zero-knowledge proofs, which can be achieved with Damgård's technique [Dam00], as well as some more inefficient simulation techniques utilizing small challenge space. However, even random oracle based zero-knowledge proofs (such as Fischlin's transform, see Construction 3.23) do not fit the definition of [BBDE19] because those generally have a small probability of simulation failure.

5.2.3 Explainability

Recall the unforgeability definition (Definition 4.9) of traditional anonymous credentials. That definition is, again, centered around identity. It states that adversarial users (1) cannot impersonate an honest user and (2) cannot show attributes that have not been

issued, even while (3) the adversary gets to play the issuer/verifier role interacting with honest users.

The impersonation aspect (1) simply vanishes from our concerns, as UACS are not centered around identity anymore. Impersonation guarantees shall be naturally implied by our new definition when using UACS to implement a system with identities (as in Construction 5.14).

The aspect of interaction with honest users (3) simply becomes simulation of honest user protocols in our new definition. In Section 5.2.2, we have already discussed a simulation notion where a simulator convincingly imitates an honest user, *without needing their attributes or credentials* (or secret input α). This fits our purpose here: It models that the adversary cannot learn anything valuable from interacting with honest users. In particular, the adversary cannot learn an honest user's credentials, nor his attributes that may help impersonate the user in certain cases (such as *usk* in Construction 5.14).

Finally, our new definition also needs some mechanism comparable to (2), modeling that the adversary cannot present a credential with attributes that have not been issued. This was easy to define for traditional credentials: We could simply keep track of all the attributes that were issued by honest issuers and then check whether any of them match the presentation predicate ϕ . For UACS, however, this is not easy at all. This is because inherently, in a UACS, the issuers are *supposed* to be unaware of what exact attributes are being issued (or result from an update), as guaranteed by Definition 5.2. As a result, we cannot model honest issuer/verifier behavior in the style of Definition 4.9 and expect to get any insights into what attributes are being issued to the adversary.

This is not to say that issuers and verifiers do not have any expectations about users' attributes. As a trivial example, say an issuer issues two credentials with attribute $attr = 0$, and then runs two updates that increment $attr$ by one ($\psi(attr, \alpha) = attr + 1$). Then of course, the issuer will not know whether the second update updates $attr = 1$ to $attr' = 2$ or $attr = 0$ to $attr' = 1$. But clearly, after this sequence of updates, nobody should hold a credential with $attr = 3$. More generally, issuers expect *consistency*: Even though they do not *know* what exactly happens in any particular issue/update protocol, for any credential attributes that end up in a user's hands, there should be a reasonable *explanation* in the form of a sequence of updates and the hidden arguments they are applied to. In the example above, an intuitive explanation for a user holding an $attr = 2$ credential is that the user first updated $attr = 0$ to $attr' = 1$ and then updated the $attr = 1$ credential to $attr' = 2$.

We follow this idea of requiring consistent explanations for all operations. In our unforgeability definition for UACS (which we call “simulation explainability”), we require the existence of an efficient “explainer” algorithm $\mathcal{E}$ that produces such explanations.

Whenever the adversary (in the role of the user) asks to be *issued* a credential with update function ψ , the explainer should produce an explanation in the form of the value α , which would mean that the explainer suggests that the attributes issued were $\vec{attr}' = \psi(\perp, \alpha)$. Whenever the adversary asks to *update* a credential with update function ψ , the explainer should produce $\vec{attr}, \alpha$, meaning that the explainer suggests the updated attributes are $\vec{attr}' = \psi(\vec{attr}, \alpha) \neq \perp$. Figure 5.2 illustrates this process. For this explanation to be considered consistent, we require the old attributes $\vec{attr}$ to be

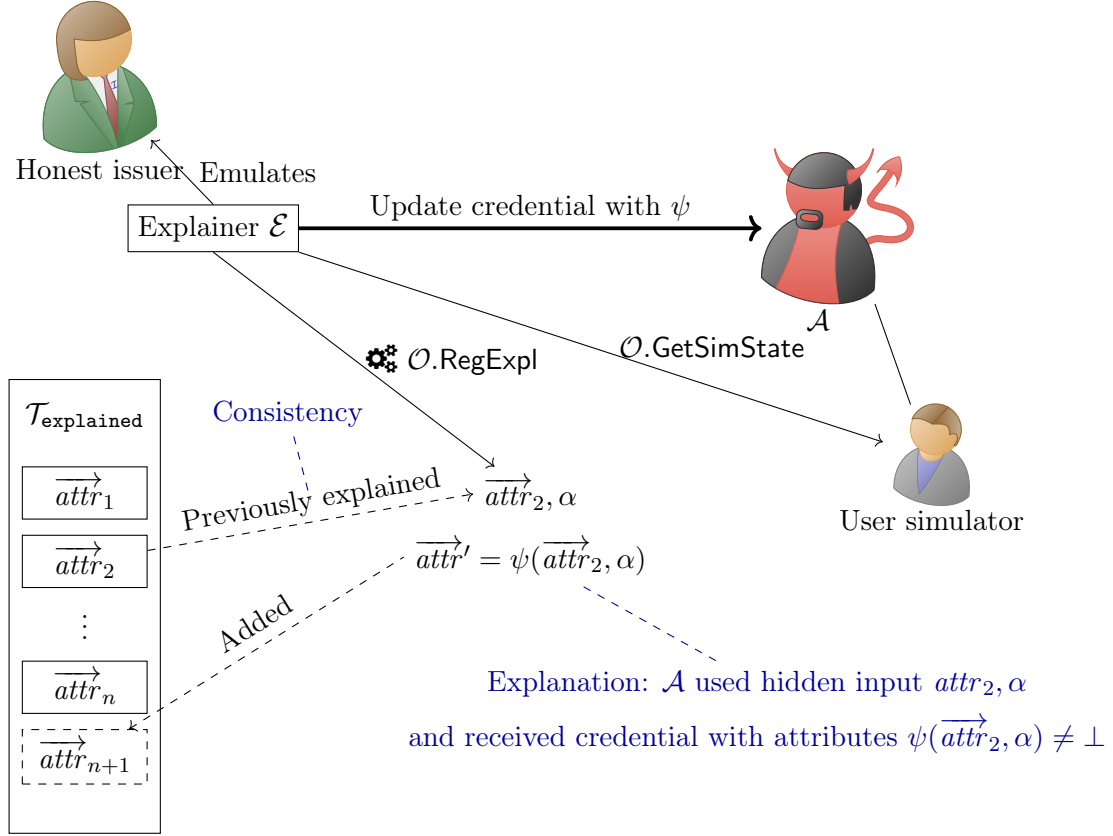


Figure 5.2: Overview of the simulation-explainability definition (Definition 5.3), illustrating the explanation procedure for a credential update.

explainable with the explainer’s previous explanations, i.e., there must have been some earlier issue or update operation that the explainer suggested resulted in a credential with attributes $\vec{attr}$. Finally, whenever the adversary *presents* a credential with predicate ϕ , the explainer should produce $\vec{attr}, \alpha$ such that $\phi(\vec{attr}, \alpha) = 1$. Again, we consider this explanation consistent if $\vec{attr}$ are the result of an earlier issue/update operation according to the explanations given so far.

This means that in practice, while the details of concrete attributes are hidden from issuers/verifiers, they can be sure that an explanation *exists* for every interaction that is consistent with the issues/updates run so far. Not only does an explanation exist, we require it to be efficiently computable, which is particularly useful when using UACS as a building block such that the update functions ψ involve cryptographic operations (like pseudonym verification in Construction 5.14).

Technically, the explainer $\mathcal{E}$ will take the role of honest issuers and verifiers. We require $\mathcal{E}$ to follow the honest issuer’s/verifier’s protocols exactly (outwardly), while running additional computation steps to extract explanations from the protocol execution. Once an explanation is found, $\mathcal{E}$ can register it by calling its $\mathcal{O}.RegExpl$ oracle. To help extract

explanations, the explainer $\mathcal{E}$ additionally gets access to the simulator's state via an oracle $\mathcal{O}.\text{GetSimState}$. This state may contain helpful information such as the adversary's random oracle queries, or a trapdoor that the simulator embedded into pp .

Definition 5.3 (Simulation explainability). Let $\mathcal{S}$ be a stateful interactive algorithm, let $\mathcal{E}$ be an interactive algorithm, and let $\mathcal{A}$ be a ppt algorithm. Consider the following game $\text{Exp}_{\Pi_{\text{uacs}}}^{\text{sim-expl}}(\Pi_{\text{uacs}}, \mathcal{A}, \mathcal{S}, \mathcal{E}, n, \lambda)$.

$\text{Exp}_{\Pi_{\text{uacs}}}^{\text{sim-expl}}(\Pi_{\text{uacs}}, \mathcal{A}, \mathcal{S}, \mathcal{E}, n, \lambda)$

$\mathcal{G} \leftarrow \text{GenSetting}(1^\lambda), pp \leftarrow \mathcal{S}.\text{Setup}(\mathcal{G})$ //Simulated setup
 $(ipk^*, isk^*) \leftarrow \text{IssuerKeyGen}(pp, 1^n)$ //Honest issuer key
 Run $\mathcal{A}(\mathcal{G}, pp, ipk^*)$, granting it access to oracles $\mathcal{S}.\mathcal{RO}, \mathcal{O}.\text{IssueCred}, \mathcal{O}.\text{UpdateCred}, \mathcal{O}.\text{VerifyCred}, \mathcal{O}.\text{SimReceiveCred}, \mathcal{O}.\text{SimReceiveUpdate}, \mathcal{O}.\text{SimPresent}$
 When an oracle raises $\text{fail}_{\text{explain}}$, output 1
 If no such event has been raised when $\mathcal{A}$ halts, output 0

$\mathcal{O}.\text{IssueCred}(\psi, ctx)$ //Adversarial user gets issued a credential

Set $explained = 0$ and $\alpha = \perp$
 $b \leftarrow \mathcal{E}.\text{IssueCred}^{\mathcal{S}.\mathcal{RO}, \mathcal{O}.\text{GetSimState}, \mathcal{O}.\text{RegExpl}}(pp, ipk^*, \psi, ctx; isk^*) \leftrightarrow \mathcal{A}$

$\mathcal{O}.\text{RegExpl}(\alpha)$ //Callback for explainer to register explanation

assert $explained = 0$ //Only accept one explanation
 assert $\psi(\perp, \alpha) \neq \perp$ //Explanation consistent with ψ
 Add $\psi(\perp, \alpha)$ to $\mathcal{T}_{\text{explained}}$ //Recognize $\psi(\perp, \alpha)$ for future explanations
 Set $explained = 1$

If $explained = 0$ and $b = 1$ and $(\text{ReceiveCred}, ipk^*, \psi, ctx) \notin \mathcal{T}_{\text{sim}}$:

raise $\text{fail}_{\text{explain}}$ // $\mathcal{E}$ failed to produce an explanation

If $explained = 1$ and $b = 0$:

raise $\text{fail}_{\text{explain}}$ // $b = 0$: No credential must be issued

return α //Give explanation to $\mathcal{A}$

$\mathcal{O}.\text{UpdateCred}(\psi, ctx)$ //Adversarial user gets a credential updated

Set $explained = 0$ and $(\overrightarrow{attr}, \alpha) = (\perp, \perp)$
 $b \leftarrow \mathcal{E}.\text{UpdateCred}^{S.\mathcal{RO}, \mathcal{O}.\text{GetSimState}, \mathcal{O}.\text{RegExpl}}(pp, ipk^*, \psi, ctx; isk^*) \leftrightarrow \mathcal{A}$

$\mathcal{O}.\text{RegExpl}(\overrightarrow{attr}, \alpha)$ //Callback for explainer to register explanation

assert $explained = 0$ //Only accept one explanation
 assert $\psi(\overrightarrow{attr}, \alpha) \neq \perp$ //Explanation consistent with ψ
 assert $\overrightarrow{attr} \in \mathcal{T}_{\text{explained}}$ //Old attributes must have been explained
 Add $\psi(\overrightarrow{attr}, \alpha)$ to $\mathcal{T}_{\text{explained}}$ //Consider updated attributes explained
 Set $explained = 1$

If $explained = 0$ and $b = 1$ and $(\text{ReceiveUpdate}, ipk^*, \psi, ctx) \notin \mathcal{T}_{\text{sim}}$:

raise $\text{fail}_{\text{explain}}$ // $\mathcal{E}$ failed to produce an explanation

If $explained = 1$ and $b = 0$:

raise $\text{fail}_{\text{explain}}$ // $b = 0$: No credential must be issued

return $(\overrightarrow{attr}, \alpha)$ //Give explanation to $\mathcal{A}$

$\mathcal{O}.\text{VerifyCred}(ipk, \phi, ctx)$ //Adversarial user presents a credential

Set $explained = 0$ and $(\overrightarrow{attr}, \alpha) = (\perp, \perp)$
 $b \leftarrow \mathcal{E}.\text{VerifyCred}^{S.\mathcal{RO}, \mathcal{O}.\text{GetSimState}, \mathcal{O}.\text{RegExpl}}(pp, ipk, \phi, ctx) \leftrightarrow \mathcal{A}$

$\mathcal{O}.\text{RegExpl}(\overrightarrow{attr}, \alpha)$ //Callback for explainer to register explanation

assert $explained = 0$ //Only accept one explanation
 assert $\phi(\overrightarrow{attr}, \alpha) = 1$ //Explanation consistent with ϕ
 assert $ipk \neq ipk^* \vee \overrightarrow{attr} \in \mathcal{T}_{\text{explained}}$ //Honest issuer $\Rightarrow \overrightarrow{attr}$ explained
 Set $explained = 1$

If $explained = 0$ and $b = 1$ and $(\text{PresentCred}, ipk, \phi, ctx) \notin \mathcal{T}_{\text{sim}}$:

raise $\text{fail}_{\text{explain}}$ // $\mathcal{E}$ failed to produce an explanation

return $(\overrightarrow{attr}, \alpha)$ //Give explanation to $\mathcal{A}$

$\mathcal{O}.\text{GetSimState}()$ //Used by explainer $\mathcal{E}$. Not accessible to $\mathcal{A}$

return $state_S$

$\mathcal{O}.\text{SimReceiveCred}(ipk, \psi, ctx)$ //Model honest user receiving credential

Add $(\text{ReceiveCred}, ipk, \psi, ctx)$ to $\mathcal{T}_{\text{sim}}$
 $b \leftarrow \mathcal{S}.\text{ReceiveCred}(pp, ipk, \psi, ctx) \leftrightarrow \mathcal{A}$
 return b

$\mathcal{O}.\text{SimReceiveUpdate}(ipk, \psi, ctx)$ //Model honest user updating credential

Add $(\text{ReceiveUpdate}, ipk, \psi, ctx)$ to $\mathcal{T}_{\text{sim}}$
 $b \leftarrow \mathcal{S}.\text{ReceiveUpdate}(pp, ipk, \psi, ctx) \leftrightarrow \mathcal{A}$
 return b

$\mathcal{O}.\text{SimPresent}(ipk, \phi, ctx)$ //Model honest user presenting credential

Add $(\text{PresentCred}, ipk, \phi, ctx)$ to $\mathcal{T}_{\text{sim}}$
 $\mathcal{S}.\text{PresentCred}(pp, ipk, \phi, ctx) \leftrightarrow \mathcal{A}$

An updatable anonymous credential system Π_{uacs} has *simulation explainability*, if there exists a stateful interactive algorithm $\mathcal{S}$ and an interactive algorithm $\mathcal{E}$ such that:

- $\mathcal{S}$ fulfills the requirements of Definition 5.2.
- $\mathcal{E}.X^{\mathcal{S}, \mathcal{RO}, \mathcal{O}.\text{GetSimState}, \mathcal{O}.\text{RegExpl}}$ perfectly emulates $\Pi_{\text{uacs}}.X^{\mathcal{S}, \mathcal{RO}}$ (cf. Definition 2.2) for $X \in \{\text{IssueCred}, \text{UpdateCred}, \text{VerifyCred}\}$.
- For all $n \in \mathbb{N}$ and all ppt $\mathcal{A}$, $\Pr[\text{Exp}_{\Pi_{\text{uacs}}}^{\text{sim-expl}}(\Pi_{\text{uacs}}, \mathcal{A}, \mathcal{S}, \mathcal{E}, n, \lambda) = 1]$ is negligible.
- For all $n \in \mathbb{N}$ and all ppt $\mathcal{A}$, there exists a polynomial poly such that the runtime of $\text{Exp}_{\Pi_{\text{uacs}}}^{\text{sim-expl}}(\Pi_{\text{uacs}}, \mathcal{A}, \mathcal{S}, \mathcal{E}, n, \lambda)$ is at most $\text{poly}(\lambda)$.

This definition is reminiscent of our Definition 3.22 of simulation-extractable proofs. This is not by accident: UACS are quite powerful and versatile. One can easily build a secure proof system from a UACS (by, essentially, using presentation predicates $\phi(attr, \alpha)$ to prove certain statements about the witness α and simply disregard the values of $attr$).

Trivially, an adversary can mount a relay attack (cf. Section 3.3.2) by relaying the messages between $\mathcal{O}.\text{SimReceiveCred} \leftrightarrow \mathcal{O}.\text{IssueCred}$, $\mathcal{O}.\text{SimReceiveUpdate} \leftrightarrow \mathcal{O}.\text{UpdateCred}$, or $\mathcal{O}.\text{SimPresent} \leftrightarrow \mathcal{O}.\text{VerifyCred}$. Since the user's messages are simulated, we cannot expect the explainer to output a convincing explanation. We keep track of simulation requests via $\mathcal{T}_{\text{sim}}$ and relieve the explainer from its obligation to provide an explanation if the simulator is involved. However, note that for $\mathcal{O}.\text{IssueCred}$ and $\mathcal{O}.\text{UpdateCred}$, not providing an explanation effectively implies that the adversary must not receive a new credential (otherwise, the adversary is now in possession of an unexplained credential and can easily win). This models that an adversary cannot interact with an honest user and honest issuer in a relay-attack fashion to receive credentials that circumvent the explanation mechanism.

A design decision that may look strange at first glance, especially compared to our proof of knowledge definitions, is that the explainer $\mathcal{E}$ registers the explanation via a callback oracle $\mathcal{O}.\text{RegExpl}$, instead of simply returning the explanation after the protocol has concluded. To understand this decision, consider a protocol where the user learns the issued credential from the last message of the issuer (both for issuance and updates). In this setting, the explainer $\mathcal{E}$ runs into an ordering problem: $\mathcal{E}$ needs to register the explanation *before* sending the last message to the adversary. If $\mathcal{E}$ cannot do this, then the adversary receives a credential and can use it before its explanation is registered, leading

to trivial wins. The $\mathcal{O}.\text{RegExpl}$ oracle provides a way for $\mathcal{E}$ to register the explanation at any time during its execution of the honest issuer’s protocol, resolving the ordering problem.

As another detail, note that the $\mathcal{O}.\text{VerifyCred}(ipk, \phi, ctx)$ oracle takes an arbitrary issuer key ipk as input. The explainer is still forced to produce convincing explanations even for $ipk \neq ipk^*$, i.e., for credentials issued by dishonest issuers. Of course, in that case, there is no requirement that the dishonest issuer must have previously issued certain attributes (as we have no information about attributes that $\mathcal{A}$ essentially issues to itself).

Alternative versions of Definition 5.3 can also be considered: (1) a version where the explainer outputs all explanations at the end of the game (as in the original paper [BBDE19]) instead of returning the explanations to $\mathcal{A}$ immediately, or even (2) a version in which explanations only need to *exist*, but are not necessarily efficiently computable. Version (1) would relax the requirements of $\mathcal{E}$. In a sense, that would be the equivalent to going to non-adaptive extraction for proofs of knowledge (Section 3.3.3). This notion still seems to be widely applicable, if somewhat less convenient to think about. Version (2), requiring only *existence* of an explanation, is adequate for applications in which update functions ψ are sufficiently simple. For example, if updates are just incrementing a value. For more advanced applications, such as our construction of traditional anonymous credentials from UACS (Construction 5.14), mere existence of an explanation is not sufficient. In this case, this is because an explanation for a corrupt user impersonating an honest user certainly *exists*, but due to security of the pseudonym system, it is not efficiently computable. We will see details in the proof of Theorem 5.16.

In our original paper [BBDE19], the unforgeability definition (called “soundness” in [BBDE19]) does not consider the presence of honest users, which we handle in Definition 5.3 by giving $\mathcal{A}$ access to simulated protocol executions. This was not necessarily an issue in [BBDE19], because our incentive system application for UACS does not require it, but our Definition 5.3 is more powerful than what was considered in [BBDE19]. Modeling honest users for explainability enables, for example, Construction 5.14.

Finally, note that Definition 5.3 considers concurrent execution of protocols (which is also in contrast to [BBDE19]), as described in Section 2.2.5.

5.3 Constructing UACS

In order to construct a UACS, we need two ingredients: (1) zero-knowledge proofs of knowledge, which we discussed in Chapter 3, and (2) a blind signature scheme, which will allow the issuer to sign values without learning what was signed. We will need a special kind of blind signature scheme, namely one that allows us to sign *committed values*.

We discuss and model the blind signature schemes we need in Section 5.3.1, show how to generically combine a proof system and a blind signature scheme into an updatable anonymous credential system in Section 5.3.2, prove security in Sections 5.3.3 and 5.3.4, and show how to instantiate the generic construction in Section 5.3.5.

5.3.1 Signing a committed value

A credential in a UACS is essentially a digital signature on attributes $\overrightarrow{attr}$. In order to hide attributes from the issuer, we require a blind signature scheme, which allows signers to sign messages (attributes) without learning what message is being signed. We require a *certain type* of blind signature scheme, which must have the following form.

- The user commits to the messages $\vec{m}$ he wants to have signed and sends the commitment C to the signer.
- Then the user and the signer run some protocol $\text{BlindReceive} \leftrightarrow \text{BlindSign}$, after which the user outputs a signature σ on $\vec{m}$ and the signer does not learn anything about $\vec{m}$.

We require this special form because we need some way to *constrain* the values the issuer is willing to blindly sign. For example, when updating a credential, the user cannot just choose any arbitrary message to be signed, the message must be specifically $\overrightarrow{attr}' = \psi(\overrightarrow{attr}, \alpha)$ for some signed $\overrightarrow{attr}$ and fitting secret α — in other words, a *valid* update. The explicit commitment C allows us to interface with the blind signature scheme: If the user can prove in zero-knowledge that the commitment contains the message $\psi(\overrightarrow{attr}, \alpha)$, we can continue and allow the user to receive a signature on that message.

Definition 5.4 (Blind signature scheme for signing a committed value). A *blind signature scheme* Π_{blind} with message space $\mathbb{M} = \mathbb{M}_{\mathcal{G}}$ and commitment randomness space $\mathbb{O} = \mathbb{O}_{\mathcal{G}}$ consists of the following ppt algorithms.

$\text{Setup}(\mathcal{G}) \rightarrow pp$

Generates public parameters pp .

$\text{Keygen}(pp, 1^n) \rightarrow (pk, sk)$

Generates a public key pk and a secret key sk . We assume n can be efficiently derived from pk . $\heartsuit$ The key can be used to sign messages $m \in \mathbb{M}^n$.

$\text{Sign}(pp, sk, \vec{m}) \rightarrow \sigma$

Computes a signature σ on message $\vec{m}$.

$\text{Verify}(pp, pk, \vec{m}, \sigma) = b$

A deterministic algorithm. $\heartsuit$ $b = 1$ indicates that σ is valid signature on message $\vec{m}$.

$\text{Commit}(pp, pk, \vec{m}, o) = C$

A deterministic algorithm. It outputs a commitment C to messages $\vec{m} \in \mathbb{M}_{\mathcal{G}}^n$ with randomness $o \in \mathbb{O}_{\mathcal{G}}$.

$\text{BlindSign}(pp, pk, C; sk) \leftrightarrow \text{BlindReceive}(pp, pk, C; \vec{m}, o) \rightarrow \sigma$

An interactive protocol after which the receiver outputs a signature σ or the error symbol $\perp$. $\heartsuit$ The receiver gets a signature σ on $\vec{m}$. The receiver is supposed to have previously computed $C = \text{Commit}(pp, pk, \vec{m}, o)$ and sent C to the signer.

The scheme is *correct*, i.e., $(\text{Setup}, \text{Keygen}, \text{Sign}, \text{Verify})$ is a (correct) signature scheme and for all $\lambda, n \in \mathbb{N}$, all $\mathcal{G} \in [\text{GenSetting}(1^\lambda)]$, $pp \in [\text{Setup}(\mathcal{G})]$, $(pk, sk) \in [\text{Keygen}(pp, 1^n)]$, and all $\vec{m} \in \mathbb{M}_{\mathcal{G}}^n, o \in \mathbb{O}_{\mathcal{G}}$, it holds that

$$\Pr \left[\text{Verify}(pp, pk, \vec{m}, \sigma) = 1 \mid \begin{array}{l} C = \text{Commit}(pp, pk; \vec{m}, o) \\ \text{BlindSign}(pp, pk, C, sk) \\ \leftrightarrow \text{BlindReceive}(pp, pk, C; \vec{m}, o) \rightarrow \sigma \end{array} \right] = 1$$

We require three security properties for blind signature schemes: (1) unforgeability, i.e., users cannot produce a signature on a message they have not requested to be signed, (2) commitment binding, i.e., users cannot open any commitment to two different messages, and (3) signing blindness, i.e., signers do not learn anything about the signed message. We detail those properties in the following.

Typically, unforgeability for blind signatures is defined in a one-more fashion: The adversary wins if it can output more signatures than the number of times it has asked the signer to run the blind signing protocol. Such a notion is not useful for us because we need guarantees that the message being signed blindly is indeed the committed message (about which we can externally prove some additional statements in zero-knowledge). For example, a blind signature scheme in which the signer accepts a special “ignore the commitment C and instead give me a signature on $\vec{m}'$ ” message and signs $\vec{m}'$ instead, can still be considered one-more secure, but would essentially allow the user to get credentials for arbitrarily chosen attributes, instead of being constrained to valid updates.

In lieu of a one-more definition, we need some other way to get around the fact that in blind signatures, the typical unforgeability winning condition “output a signature on a message *not* signed by the signer” is difficult to model, given that the signer does not learn the message she signs blindly. Our way around this is to consider *chosen-known-message* attacks: Whenever the adversary asks for a blind signature, it must supply the message $\vec{m}$ and the opening value o for the commitment. This way, we can easily log what messages are being signed, and phrase our winning condition accordingly. When using a blind signature scheme (secure against chosen-known-message-attacks) in a larger construction, one must ensure that the user indeed knows the hidden values $\vec{m}, o$ in C whenever he queries for a blind signature. This can be achieved using a proof of knowledge.

Indeed, having the blind signature protocol execute a proof of knowledge is how many works describe their protocols for signing committed values [CL02; CL04; PS16]. We believe the chosen-known-message-attack route is the better one: It separates the signature scheme from whatever way the commitment opening is extracted from corrupt users. This still supports relevant constructions [CL04; BBS04; BCKL08; PS16] but makes for cleaner proofs. Additionally, it is useful for composability of multiple building blocks: In anonymous credential systems, proving knowledge of the commitment opening is part of a larger zero-knowledge proof (proving statements about the committed message) and not something that happens in isolation within the blind signing protocol.

Definition 5.5 (Unforgeability). A blind signature scheme Π_{blind} is *existentially unforgeable under chosen-known-message attacks* if for all $n \in \mathbb{N}$ and all ppt adversaries $\mathcal{A}$,

$$\Pr \left[\begin{array}{l} \text{Verify}(pp, pk, \vec{m}^*, \sigma^*) = 1 \\ \wedge \quad \vec{m}^* \notin \mathcal{T}_{\text{signed}} \end{array} \middle| \begin{array}{l} \mathcal{G} \leftarrow \text{GenSetting}(1^\lambda) \\ pp \leftarrow \text{Setup}(\mathcal{G}) \\ (pk, sk) \leftarrow \text{Keygen}(pp, 1^n) \\ (\vec{m}^*, \sigma^*) \leftarrow \mathcal{A}^{\mathcal{O}.\text{BlindSign}}(\mathcal{G}, pp, pk) \end{array} \right]$$

is negligible, where

$\mathcal{O}.\text{BlindSign}(\vec{m}, o)$ //Models signature queries for unforgeability

$C = \text{Commit}(pp, pk, \vec{m}, o)$

Add $\vec{m}$ to $\mathcal{T}_{\text{signed}}$

$\text{BlindSign}(pp, pk, C; sk) \leftrightarrow \mathcal{A}$

Note that this definition does not guarantee security for any scenarios that involve the (local) $\Pi_{\text{blind}}.\text{Sign}$ algorithm. If this is required, one must give $\mathcal{A}$ an additional oracle that outputs a signature according to $\Pi_{\text{blind}}.\text{Sign}$ instead of running the blind protocol. However, for our constructions, $\Pi_{\text{blind}}.\text{Sign}$ is unused, so this definition suffices. In general, one can always use a “canonical” $\Pi_{\text{blind}}.\text{Sign}$ implementation that simply has the signer run the blind signature protocol with herself. In that case, Definition 5.5 also covers $\Pi_{\text{blind}}.\text{Sign}$.

On a more positive note, Definition 5.5 implies that the commitment scheme induced by Π_{blind} is computationally binding (Definition 2.16). This is because if some commitment C can be opened to both $\vec{m}$ and $\vec{m}' \neq \vec{m}$ with randomness o and o' respectively, then asking for a signature for $(\vec{m}, o)$ and following the protocol honestly for $(\vec{m}', o')$ will result in a signature that is valid on $\vec{m}'$ (by the correctness guarantees), which would be considered a forgery by Definition 5.5.

However, it is not necessarily clear that the commitment scheme is still binding against an adversary that can query the $\mathcal{O}.\text{BlindSign}$ oracle. If an adversary were able to, say, produce a collision for messages $\vec{m} \neq \vec{m}'$ *only* by using information gained from querying the $\mathcal{O}.\text{BlindSign}$ oracle with those two messages, then the collision would not constitute a forgery. This is not an issue for our purposes, but if a full binding property is required, it needs to be defined separately.

Finally, we define signing blindness, which aims to hide an honest user’s message from the signer in the blind signing process. Our definition here is a simple left-or-right game, in which the adversary, in the role of the signer, can choose two messages $\vec{m}_0, \vec{m}_1$ for the user and must guess which one the user commits to and runs the blind signing protocol with.

Definition 5.6 (Blindness). A blind signature scheme Π_{blind} has *signing blindness* if for

all ppt adversaries $\mathcal{A}$,

$$\left| \Pr[\mathcal{A}^{\mathcal{O}.\text{BlindReceive}^0}(\mathcal{G}, pp) = 1] - \Pr[\mathcal{A}^{\mathcal{O}.\text{BlindReceive}^1}(\mathcal{G}, pp) = 1] \right|$$

is negligible, where $\mathcal{G} \leftarrow \text{GenSetting}(1^\lambda)$, $pp \leftarrow \text{Setup}(\mathcal{G})$ and

```

 $\mathcal{O}.\text{BlindReceive}^b(pk, \vec{m}^{(0)}, \vec{m}^{(1)})$  //Left-or-right oracle to model blindness
assert  $\vec{m}_0, \vec{m}_1 \in \mathbb{M}^n$ 
 $C = \text{Commit}(pp, pk, \vec{m}^{(b)}, o)$  with  $o \leftarrow \mathbb{O}$ 
Hand  $C$  to  $\mathcal{A}$ 
 $\sigma \leftarrow \text{BlindReceive}(pp, pk, C; \vec{m}^{(b)}, o) \leftrightarrow \mathcal{A}$ 
return  $\text{Verify}(pp, pk, \vec{m}^{(b)}, \sigma)$ 

```

Definition 5.6 implies that the commitment scheme used in Π_{blind} is computationally hiding (Definition 2.17). This is because $\mathcal{A}$ gets the commitment C as a distinguishing factor.

In Definition 5.6, the adversary $\mathcal{A}$ never gets to see any of the signatures σ that the honest user receives. This is unusual for blind signature schemes (and one can argue that calling a Definition 5.4 scheme a “blind signature scheme” is confusing for that reason), which generally require that the adversary cannot recognize a signature she blindly created earlier. However, in our setting, there is a good reason not to model this property: The signature, of course, leaks which message has been signed. Additionally, there is no need to reveal the signature to the adversary: In our construction of UACS, users will never reveal their received signatures (credentials) in the clear, but always hide them within zero-knowledge proofs. We do, however, let $\mathcal{A}$ know whether the signature output by the user is valid or not. This is information that will be leaked by almost all applications, given that a failure of the blind signing protocol typically cannot be easily recovered from without explicitly asking the signer to retry. If $\mathcal{A}$ could behave cleverly such that protocol failures happen conditionally depending on the user’s hidden message, then this would constitute a serious privacy issue.

Note that while most definitions around UACS are simulation-based, we still define blindness via indistinguishability between two messages. One can always “simulate” the user side of the blind signing process without knowledge of $\vec{m}$ by simply committing to and running the protocol with some fixed message $\vec{m}_{\text{dummy}} \in \mathbb{M}^n$, instead.

5.3.2 The standard template for updatable anonymous credentials

The construction of updatable anonymous credentials requires two building blocks: a blind signature scheme for signing committed values Π_{blind} and a proof system Π_{zk} . The resulting UACS is illustrated in Figure 5.3. We call it the “standard template” because of its similarity to the traditional anonymous credential standard template (Section 4.3, [CL02]) and because we consider it a natural way to construct such systems.

Credentials are signatures σ on their certified attributes $\overrightarrow{\text{attr}}$ (similar to traditional anonymous credentials, see Section 4.3). To issue a credential, the issuer and the user

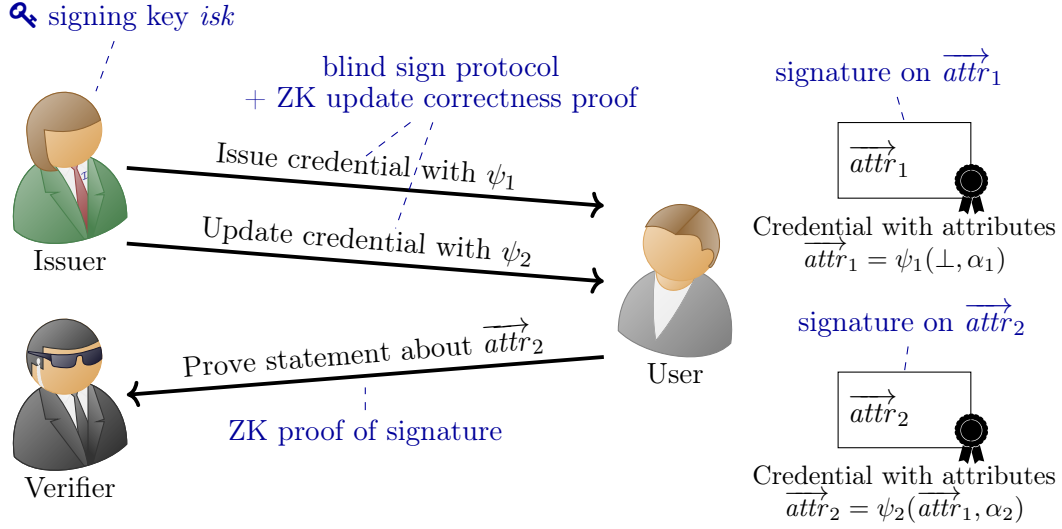


Figure 5.3: The standard template for constructing updatable anonymous credential systems.

run the blind signing protocol. In order to ensure that the issued attributes are of the form $\vec{attr}' = \psi(\perp, \alpha)$ for some secret α known to the user, the user proves, with Π_{zk} , that the committed messages to be signed are $\psi(\perp, \alpha)$. Updating a credential works similarly: The user proves that the signed attributes are of the form $\vec{attr}' = \psi(\vec{attr}, \alpha)$ such that the user holds a signature on $\vec{attr}$. If the proof is accepting, issuer and user run the blind signing protocol so that the user receives a signature on $\vec{attr}'$. Presenting a credential just involves proving knowledge of a signature on $\vec{attr}$ and a secret value α such that $\phi(\vec{attr}, \alpha) = 1$.

Construction 5.7 (Standard template updatable anonymous credentials). Let Π_{blind} be a blind signature scheme (Definition 5.4) with message space $\mathbb{M}_{\mathcal{G}}$ and randomness space $\mathbb{O}_{\mathcal{G}}$. Let Π_{zk} be a proof system for relation $\mathcal{R}_{\mathcal{G}}$. Let $\Psi_{\mathcal{G}}$ and $\Phi_{\mathcal{G}}$ be sets of update functions and predicates, respectively. We construct an updatable anonymous credential system Π_{uacs} with attribute space $\mathbb{A}_{\mathcal{G}} = \mathbb{M}_{\mathcal{G}}$ as follows:

Setup($\mathcal{G}$)

Generates public parameters $pp = (\mathcal{G}, pp_0, pp_1)$, where $pp_0 \leftarrow \Pi_{\text{blind}}.\text{Setup}(\mathcal{G})$ and $pp_1 \leftarrow \Pi_{zk}.\text{Setup}(\mathcal{G})$.

IssuerKeyGen($pp, 1^n$)

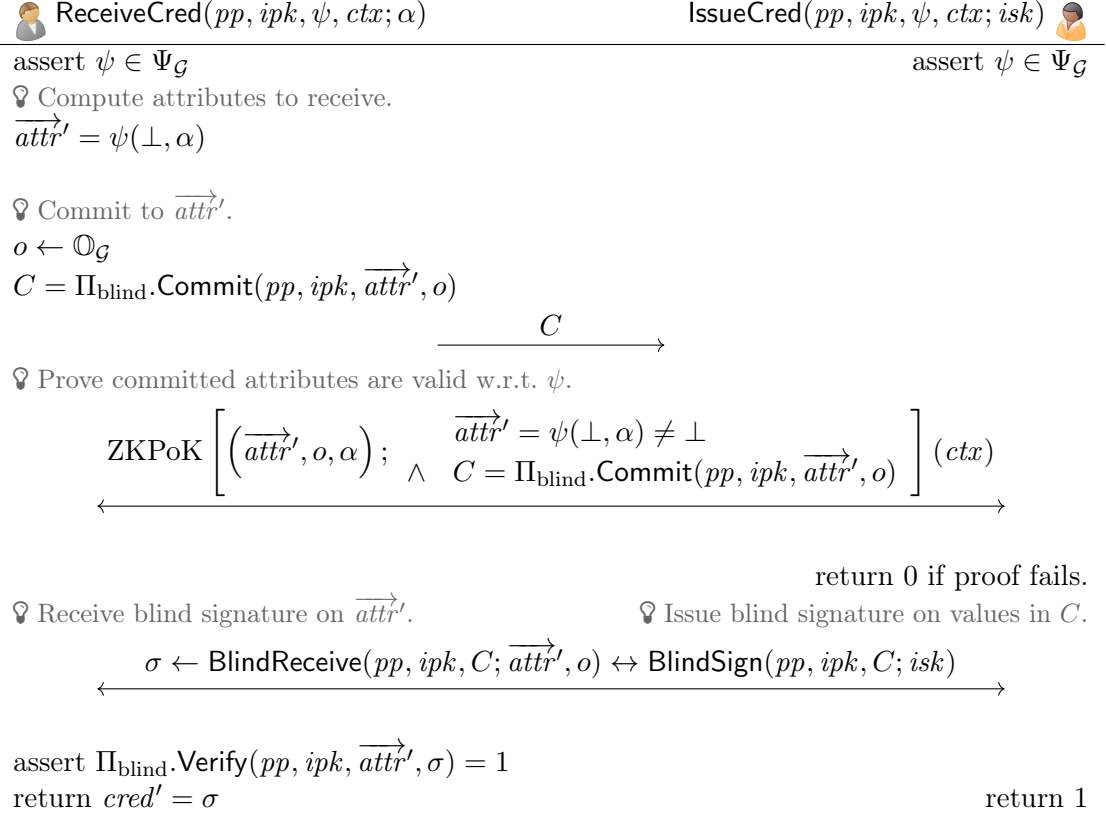
Generates issuer keys $(ipk, isk) \leftarrow \Pi_{\text{blind}}.\text{Keygen}(pp, 1^n)$.

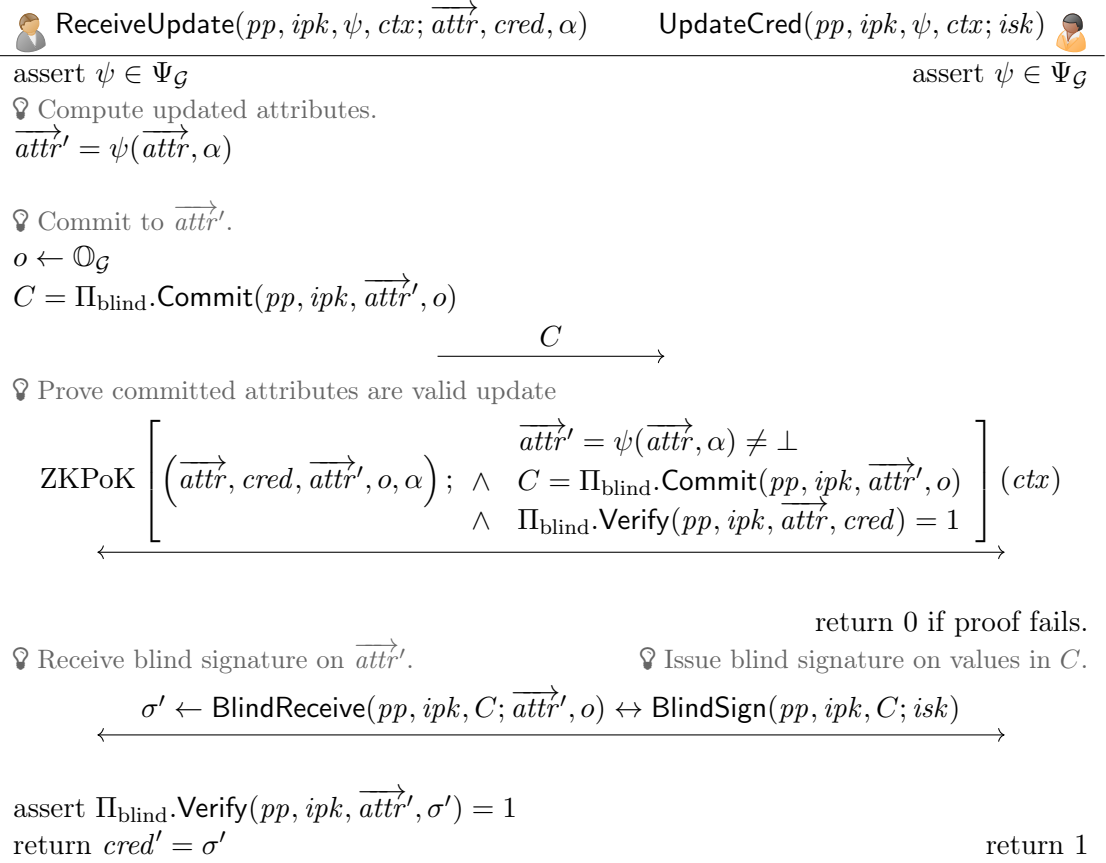
ReceiveCred($pp, ipk, \psi, ctx; \alpha$) $\leftrightarrow$ IssueCred($pp, ipk, \psi, ctx; isk$)

Works as shown in Figure 5.4.

$cred' \leftarrow \text{ReceiveUpdate}(pp, ipk, \psi, ctx; \vec{attr}, cred, \alpha) \leftrightarrow \text{UpdateCred}(pp, ipk, \psi, ctx; isk)$

Works as show in Figure 5.5.

**Figure 5.4:** The protocol for issuing a credential.

**Figure 5.5:** The protocol for updating a credential.

$\text{PresentCred}(pp, ipk, \phi, ctx; \overrightarrow{attr}, cred, \alpha) \leftrightarrow \text{VerifyCred}(pp, ipk, \phi, ctx) \rightarrow b$
 Asserts $\phi \in \Phi_G$, then has PresentCred prove

$$\text{ZKPoK} \left[\left(\overrightarrow{attr}, cred, \alpha \right); \quad \phi(\overrightarrow{attr}, \alpha) = 1 \quad \wedge \quad \Pi_{\text{blind}}.\text{Verify}(pp, ipk, \overrightarrow{attr}, cred) = 1 \right] (ctx)$$

If successful, VerifyCred outputs 1.

$\text{CheckCred}(pp, ipk, \overrightarrow{attr}, cred)$
 Outputs $\Pi_{\text{blind}}.\text{Verify}(pp, ipk, \overrightarrow{attr}, cred)$.

We require that $\mathcal{R}_G$ is compatible (Definition 3.25) with the ZKPoK $[(\cdot); \cdot](\cdot)$ expressions above and in Figures 5.4 and 5.5.

It is easy to check that this construction is a *correct* UACS, given that Π_{blind} is correct and Π_{zk} is complete.

The compatibility requirement for the zero-knowledge provable relation $\mathcal{R}_G$ may represent a restriction on the update function space Ψ_G and predicate space Φ_G . They need to be “sufficiently simple” so that the statements $\psi(\cdot, \cdot) \neq \perp$ and $\phi(\cdot, \cdot) = 1$ are provable using the proof system. Roughly speaking, in practice, this means that update functions are easy to implement efficiently using Chapter 3 if they only involve things like replacing a value, checking some boolean formula over attribute equality or inequality, or linear relations between attributes, α , and public values. With modern proof systems, one can, in principle, implement any polynomial-time computable update function (but depending on the concrete update function and proof system, the result may not be considered practically efficient).

On a more general note, because the update protocol only requires the issuer’s secret key isk to sign the updated credential but not to verify the old credential, one could easily change the UACS model and this construction to allow issuers to “update” *another* issuer’s credential (instead of just being able to update credentials she has issued herself). For the sake of conceptual simplicity, we do not consider this generalization.

5.3.3 Anonymity of standard template UACS

We now prove security of the standard template, starting with simulation anonymity.

Theorem 5.8 (Standard template UACS have simulation anonymity). *If Π_{blind} has signing blindness (Definition 5.6) and Π_{zk} is concurrent zero-knowledge (Definition 3.22), then Construction 5.7 has simulation anonymity (Definition 5.2).*

Essentially, there are three components to the issue or update protocol: (1) the commitment to the new attributes, (2) the proof, and (3) the blind signature protocol. Our simulator can simulate (1) the commitment and (3) the blind signature protocol simply by using some random fixed message $\vec{m}_{\text{dummy}}$ instead of the user’s desired attributes (which it does not know), and (2) the proof using the zero-knowledge simulator.

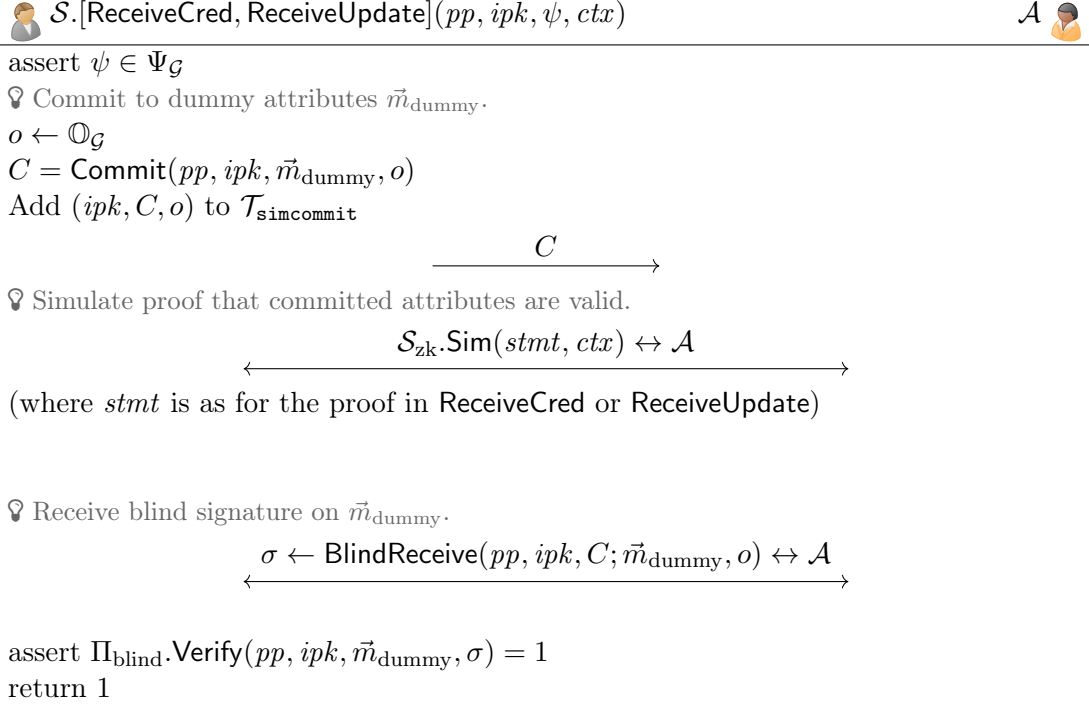


Figure 5.6: The simulator $\mathcal{S}$ for simulating receiving a credential or a credential update.

The proof proceeds in two steps: We first show that the REAL game cannot be distinguished from a HYBRID game in which the zero-knowledge proofs are being simulated. We then show that the HYBRID game is indistinguishable from the SIM game.

Proof of Theorem 5.8. Let $\mathcal{S}_{\text{zk}}$ be the simulator from the zero-knowledge property of Π_{zk} . We construct the simulator $\mathcal{S}$ for UACS as follows:

- $\mathcal{S}.\text{Setup}(\mathcal{G})$ outputs $pp = (\mathcal{G}, pp_0, pp_1)$, where $pp_0 \leftarrow \Pi_{\text{blind}}.\text{Setup}(\mathcal{G})$ and $pp_1 \leftarrow \mathcal{S}_{\text{zk}}.\text{Setup}(\mathcal{G})$. ⚡ Note that Π_{zk} 's setup is executed by the ZK simulator. Furthermore, $\mathcal{S}$ chooses a random dummy message $m_{\text{dummy}} \leftarrow \mathbb{M}$. We denote by $\vec{m}_{\text{dummy}}$ the vector $\vec{m}_{\text{dummy}} = (m_{\text{dummy}}, m_{\text{dummy}}, \dots) \in \mathbb{M}^n$ of the adequate length (the length will be apparent from the context, as it can be derived efficiently from signature public keys ipk). ⚡ $\mathcal{S}$ will simulate commitment and protocol using message $\vec{m}_{\text{dummy}}$ instead of the real message. $\mathcal{S}$ initializes a set $\mathcal{T}_{\text{simcommit}} = \emptyset$. ⚡ In $\mathcal{T}_{\text{simcommit}}$, we will remember the openings to commitments created by $\mathcal{S}$, which will be relevant for Theorem 5.9
- $\mathcal{S}.\text{ReceiveCred}$ and $\mathcal{S}.\text{ReceiveUpdate}$ work as described in Figure 5.6. ⚡ Essentially, the simulator simulates the ZK proof and runs the blind signature procedure (including the commitment) with $\vec{m}_{\text{dummy}}$ as a message.
- $\mathcal{S}.\text{PresentCred}(pp, ipk, \phi, ctx)$ asserts that $\phi \in \Phi$ and then runs $\mathcal{S}_{\text{zk}}.\text{Sim}(stmt, ctx)$, where $stmt$ is as in PresentCred.

- $\mathcal{S}.\mathcal{RO}(x)$ returns $\mathcal{S}_{zk}.\mathcal{RO}(x)$.

Let $\mathcal{A}$ be a ppt. We need to show that

$$\left| \Pr[\text{Exp}_{\Pi_{uacs}}^{\text{sim-anon}}(\Pi_{uacs}, \mathcal{S}, \mathcal{A}, \lambda, \text{REAL}) = 1] - \Pr[\text{Exp}_{\Pi_{uacs}}^{\text{sim-anon}}(\Pi_{uacs}, \mathcal{S}, \mathcal{A}, \lambda, \text{SIM}) = 1] \right|$$

is negligible and that the runtime of $\text{Exp}_{\Pi_{uacs}}^{\text{sim-anon}}(\Pi_{uacs}, \mathcal{S}, \mathcal{A}, \lambda, \text{SIM})$ is polynomially bounded. The latter follows easily from the fact that the runtime of $\mathcal{S}_{zk}$ is polynomially bounded (Definition 3.22) and all the other building block functions are also polynomial-time computable.

Towards analyzing the difference between the REAL game and the SIM game, we introduce a hybrid game $\text{Exp}_{\Pi_{uacs}}^{\text{sim-anon}}(\Pi_{uacs}, \mathcal{S}, \mathcal{A}, \lambda, \text{HYBRID})$, which we define to work just like $\text{Exp}_{\Pi_{uacs}}^{\text{sim-anon}}(\Pi_{uacs}, \mathcal{S}, \mathcal{A}, \lambda, \text{REAL})$ except that $pp_1 \leftarrow \mathcal{S}_{zk}.\text{Setup}(\mathcal{G})$ in the setup phase, the oracle $\mathcal{O}.\mathcal{RO}$ is replaced with $\mathcal{S}.\mathcal{RO}$, and that the Π_{zk} proofs in $\mathcal{O}.\text{ReceiveCred}^{\text{REAL}}$, $\mathcal{O}.\text{ReceiveUpdate}^{\text{REAL}}$, $\mathcal{O}.\text{PresentCred}^{\text{REAL}}$ are replaced with $\mathcal{S}_{zk}.\text{Sim}(stmt, ctx)$ (where $stmt, ctx$ are as in the original protocols). Through a trivial reduction to the zero-knowledge property of Π_{zk} , one can show that

$$\left| \Pr[\text{Exp}_{\Pi_{uacs}}^{\text{sim-anon}}(\Pi_{uacs}, \mathcal{S}, \mathcal{A}, \lambda, \text{REAL}) = 1] - \Pr[\text{Exp}_{\Pi_{uacs}}^{\text{sim-anon}}(\Pi_{uacs}, \mathcal{S}, \mathcal{A}, \lambda, \text{HYBRID}) = 1] \right|$$

is negligible.

The only difference between the HYBRID game and the SIM game, regarding what can be possibly observed by $\mathcal{A}$, is that in $\mathcal{O}.\text{ReceiveCred}$ and $\mathcal{O}.\text{ReceiveUpdate}$, the SIM game runs the commitment and blind signature protocol with message $\vec{m}_{\text{dummy}}$, while HYBRID uses the actual attributes $\vec{attr}'$. Through a trivial reduction to the signing blindness property of Π_{blind} , in which the reduction algorithm $\mathcal{B}$ uses oracle queries $\mathcal{O}.\text{BlindReceive}^b(ipk, \psi(X, \alpha), \vec{m}_{\text{dummy}})$ to answer $\mathcal{O}.\text{ReceiveCred}$ queries ($X = \perp$) and $\mathcal{O}.\text{ReceiveUpdate}$ queries ($X = \vec{attr}$), we get that

$$\left| \Pr[\text{Exp}_{\Pi_{uacs}}^{\text{sim-anon}}(\Pi_{uacs}, \mathcal{S}, \mathcal{A}, \lambda, \text{HYBRID}) = 1] - \Pr[\text{Exp}_{\Pi_{uacs}}^{\text{sim-anon}}(\Pi_{uacs}, \mathcal{S}, \mathcal{A}, \lambda, \text{SIM}) = 1] \right|$$

is negligible. Through triangle inequality, we overall get that

$$\begin{aligned} & \left| \Pr[\text{Exp}_{\Pi_{uacs}}^{\text{sim-anon}}(\Pi_{uacs}, \mathcal{S}, \mathcal{A}, \lambda, \text{REAL}) = 1] - \Pr[\text{Exp}_{\Pi_{uacs}}^{\text{sim-anon}}(\Pi_{uacs}, \mathcal{S}, \mathcal{A}, \lambda, \text{SIM}) = 1] \right| \\ & \leq \left| \Pr[\text{Exp}_{\Pi_{uacs}}^{\text{sim-anon}}(\Pi_{uacs}, \mathcal{S}, \mathcal{A}, \lambda, \text{REAL}) = 1] - \Pr[\text{Exp}_{\Pi_{uacs}}^{\text{sim-anon}}(\Pi_{uacs}, \mathcal{S}, \mathcal{A}, \lambda, \text{HYBRID}) = 1] \right| \\ & \quad + \left| \Pr[\text{Exp}_{\Pi_{uacs}}^{\text{sim-anon}}(\Pi_{uacs}, \mathcal{S}, \mathcal{A}, \lambda, \text{HYBRID}) = 1] - \Pr[\text{Exp}_{\Pi_{uacs}}^{\text{sim-anon}}(\Pi_{uacs}, \mathcal{S}, \mathcal{A}, \lambda, \text{SIM}) = 1] \right| \end{aligned}$$

is negligible, as required. $\square$

5.3.4 Explainability of standard template UACS

We now consider unforgeability, i.e., simulation-explainability.

Theorem 5.9 (Standard template UACS have simulation-explainability). *If Π_{blind} has signing blindness (Definition 5.6) and super-poly message space (Definition 2.10), and is existentially unforgeable under chosen-known-message attacks (Definition 5.5), and Π_{zk} is a concurrently adaptively simulation-extractable zero-knowledge proof of knowledge (Definition 3.22), then Construction 5.7 has simulation explainability (Definition 5.2).*

With some (important) details omitted, intuitively, the proof works as follows. Because users prove knowledge of exactly what kind of update they are running or what credential they are presenting, we can use the proof system extractor $\mathcal{E}_{\text{zk}}$ in order to produce explanations. Consistency of these explanations then follows from simulation-extractability of Π_{zk} (ensuring, for example, $\psi(\overrightarrow{\text{attr}}, \alpha) \neq \perp$) and unforgeability of Π_{blind} (ensuring that a credential with $\overrightarrow{\text{attr}}$ must have been issued before).

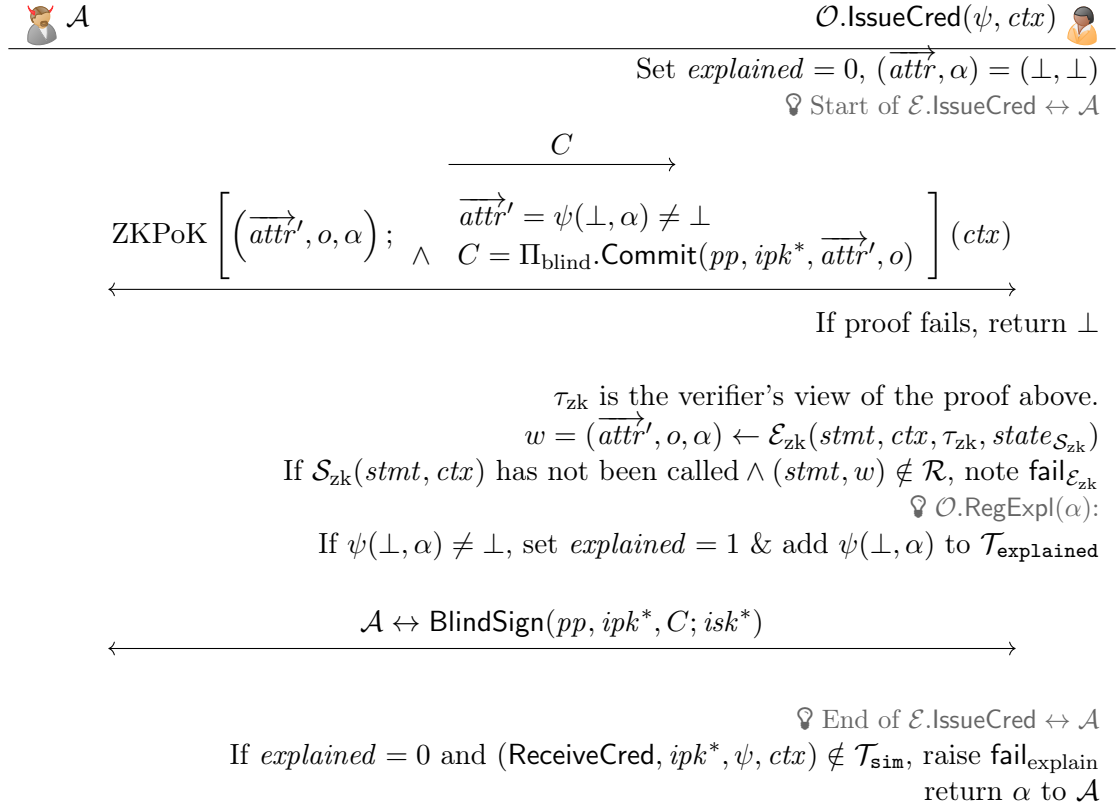


Figure 5.7: The $\mathcal{O}.\text{IssueCred}$ oracle execution in $\text{Exp}_{\Pi_{\text{uacs}}}^{\text{sim-expl}}(\Pi_{\text{uacs}}, \mathcal{A}, \mathcal{S}, \mathcal{E}, n, \lambda)$ for the proof of Theorem 5.9.

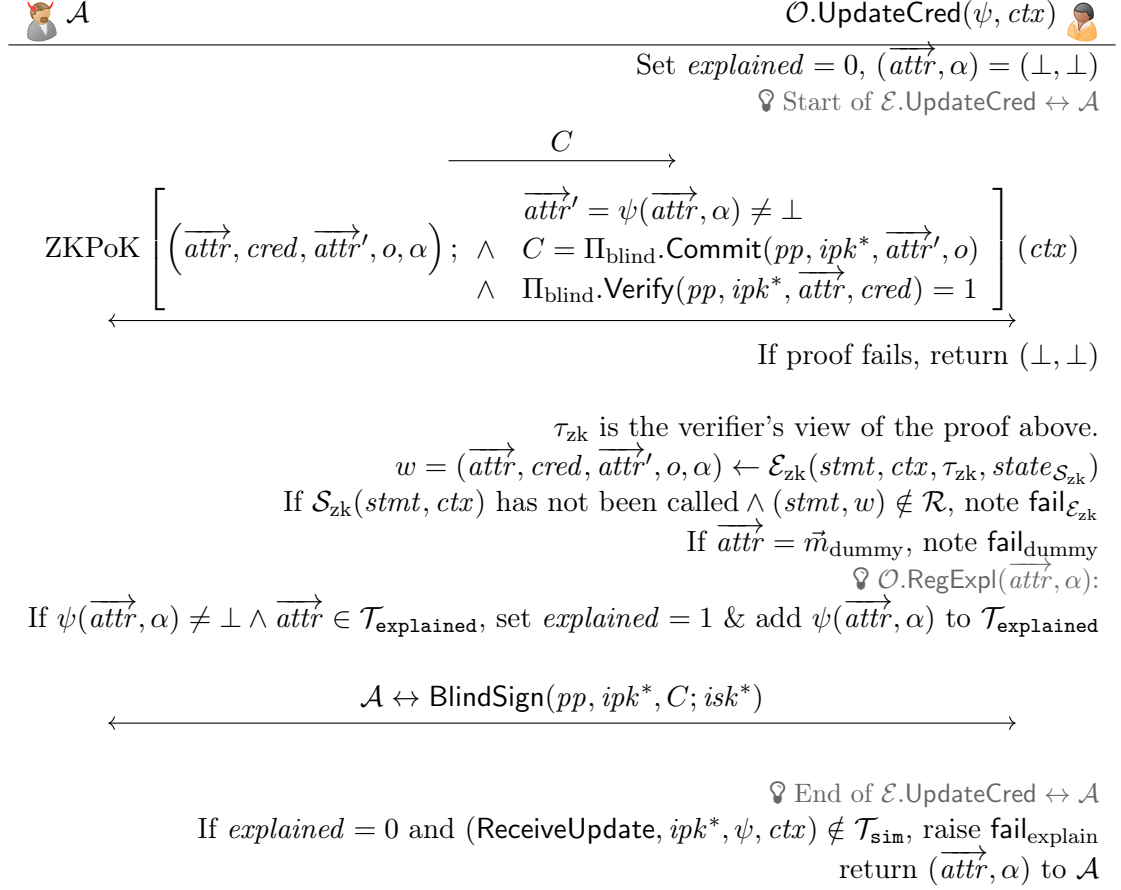


Figure 5.8: The $\mathcal{O}.\text{UpdateCred}$ oracle execution in $\text{Exp}_{\Pi_{\text{uacs}}}^{\text{sim-expl}}(\Pi_{\text{uacs}}, \mathcal{A}, \mathcal{S}, \mathcal{E}, n, \lambda)$ for the proof of Theorem 5.9.

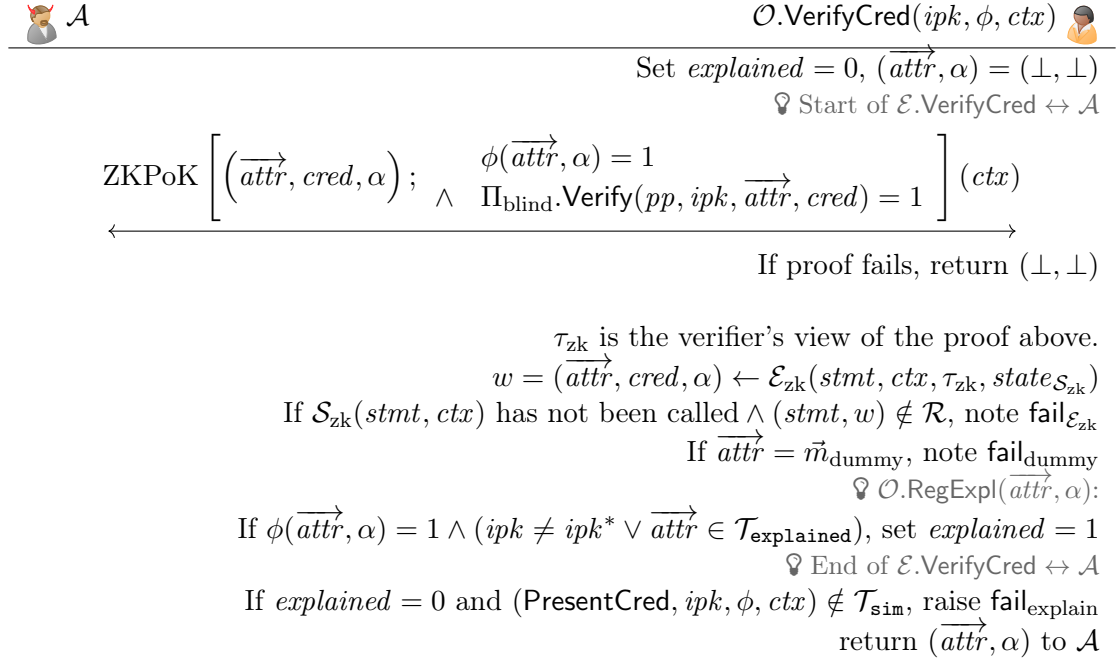


Figure 5.9: The $\mathcal{O}.\text{VerifyCred}$ oracle execution in $\text{Exp}_{\Pi_{\text{uacs}}}^{\text{sim-expl}}(\Pi_{\text{uacs}}, \mathcal{A}, \mathcal{S}, \mathcal{E}, n, \lambda)$ for the proof of Theorem 5.9.

Proof of Theorem 5.9. Let $\mathcal{S}$ be as in the proof of Theorem 5.8. Let $state_{\mathcal{S}_{zk}}$ denote the state of $\mathcal{S}_{zk}$ within $\mathcal{S}$. Let m_{dummy} be the randomly generated message stored in $state_{\mathcal{S}}$ after $\mathcal{S}.\text{Setup}$. Let $\mathcal{E}_{zk}$ be an extractor for Π_{zk} as in Definition 3.22. Without loss of generality, we assume that $\mathcal{E}_{zk}(stmt, ctx, \tau_{zk}, state_{\mathcal{S}_{zk}})$ aborts immediately and outputs $w = \perp$ if $\mathcal{S}_{zk}(stmt, ctx)$ has been called before (according to $state_{\mathcal{S}_{zk}}$). $\text{This is a small detail. In Definition 3.22, if } \mathcal{S}_{zk}(stmt, ctx) \text{ has been called before, } \mathcal{O}.\text{Extract} \text{ simply outputs } \perp \text{ and does not run } \mathcal{E}_{zk}(stmt, ctx, \tau_{zk}, state_{\mathcal{S}_{zk}}) \text{ at all. For syntactic simplicity, we want to be able to run } \mathcal{E}_{zk} \text{ even for simulated proofs (with no expectation of receiving a valid witness), so we make } \mathcal{E}_{zk} \text{ immediately output } \perp \text{ for those cases, which then is consistent to the oracle behavior in Definition 3.22.}$

We define the explainer $\mathcal{E}.X^{\mathcal{S}, \mathcal{RO}, \mathcal{O}.\text{GetSimState}, \mathcal{O}.\text{RegExpl}}(input)$ (for $X \in \{\text{IssueCred}, \text{UpdateCred}, \text{VerifyCred}\}$), to run $\Pi_{\text{uacs}}.X^{\mathcal{S}, \mathcal{RO}}(input)$ with the following additional behavior right after $\Pi_{zk}.\text{Verify}(stmt, ctx)$ within X outputs 1:

- $\mathcal{E}$ takes the simulator state $state_{\mathcal{S}_{zk}}$ from $state_{\mathcal{S}} \leftarrow \mathcal{O}.\text{GetSimState}()$ and the verifier view τ_{zk} , and runs the proof system extractor $w \leftarrow \mathcal{E}_{zk}(stmt, ctx, \tau_{zk}, state_{\mathcal{S}_{zk}})$.
- If $\mathcal{S}_{zk}(stmt, ctx)$ has not been called and $(stmt, w) \notin \mathcal{R}$, $\mathcal{E}$ notes event $\text{fail}_{\mathcal{E}_{zk}}$.
- Then, $\mathcal{E}$ retrieves the dummy message m_{dummy} from $state_{\mathcal{S}}$ and
 - for $X = \text{IssueCred}$, parse $w = (\overrightarrow{attr}', o, \alpha)$. $\mathcal{E}$ calls $\mathcal{O}.\text{RegExpl}(\alpha)$.

- for $X = \text{UpdateCred}$, parse $w = (\overrightarrow{attr}, cred, \overrightarrow{attr'}, o, \alpha)$. If $\overrightarrow{attr} = \vec{m}_{\text{dummy}}$, $\mathcal{E}$ notes event $\text{fail}_{\text{dummy}}$. $\mathcal{E}$ calls $\mathcal{O}.\text{RegExpl}(\overrightarrow{attr}, \alpha)$.
- for $X = \text{VerifyCred}$, parse $w = (\overrightarrow{attr}, cred, \alpha)$. If $\overrightarrow{attr} = \vec{m}_{\text{dummy}}$, $\mathcal{E}$ notes event $\text{fail}_{\text{dummy}}$. $\mathcal{E}$ calls $\mathcal{O}.\text{RegExpl}(\overrightarrow{attr}, \alpha)$.
- $\mathcal{E}$ then continues execution of X and outputs whatever X outputs.

Clearly, $\mathcal{E}.X^{\mathcal{S}, \mathcal{RO}, \mathcal{O}.\text{GetSimState}, \mathcal{O}.\text{RegExpl}}$ perfectly emulates $\Pi_{\text{uacs}}.X^{\mathcal{S}, \mathcal{RO}}$ (given that the additional behavior does not have any side effects other than the additional oracle calls to $\mathcal{O}.\text{GetSimState}$ and $\mathcal{O}.\text{RegExpl}$).

For a better overview, we detail the effective behavior of $\mathcal{O}.X$ with explainer $\mathcal{E}$ in Figures 5.7 to 5.9. Note that because by definition, $\mathcal{E}$ only calls $\mathcal{O}.\text{RegExpl}$ once per execution, and because protocol execution in case the initial proof fails ($b = 0$) is aborted immediately, those figures leave out trivially fulfilled oracle checks regarding $b = 0$ and $b = 1$, while still remaining perfectly faithful to the definition of $\text{Exp}_{\Pi_{\text{uacs}}}^{\text{sim-expl}}(\Pi_{\text{uacs}}, \mathcal{A}, \mathcal{S}, \mathcal{E}, n, \lambda)$.

Let $n \in \mathbb{N}$ and let $\mathcal{A}$ be a ppt. The running time of $\text{Exp}_{\Pi_{\text{uacs}}}^{\text{sim-expl}}(\Pi_{\text{uacs}}, \mathcal{A}, \mathcal{S}, \mathcal{E}, n, \lambda)$ can easily be upper-bounded by a polynomial depending on $\mathcal{A}$ because all building blocks are polynomial-time and the overall running time of $\mathcal{S}_{\text{zk}}, \mathcal{E}_{\text{zk}}$ can be upper-bounded using Definition 3.22. ! Here, we use our additional “abort immediately for previously simulated *stmt*, *ctx*” assumption on $\mathcal{E}_{\text{zk}}$ from the beginning of this proof.

It remains to show that $\Pr[\text{fail}_{\text{explain}}]$ is negligible. For this, we consider the events $\text{fail}_{\text{dummy}}$ and $\text{fail}_{\mathcal{E}_{\text{zk}}}$ as defined by $\mathcal{E}$ (see also Figures 5.7 to 5.9). We will prove that $\Pr[\text{fail}_{\text{dummy}}]$ and $\Pr[\text{fail}_{\mathcal{E}_{\text{zk}}}]$ are negligible, and then finally that $\Pr[\neg \text{fail}_{\mathcal{E}_{\text{zk}}} \wedge \neg \text{fail}_{\text{dummy}} \wedge \text{fail}_{\text{explain}}]$ is negligible.

Showing that $\Pr[\text{fail}_{\text{dummy}}]$ is negligible. ! $\text{fail}_{\text{dummy}}$ is the event that the explainer outputs an explanation of the form “ $\mathcal{A}$ used a credential with $\overrightarrow{attr} = \vec{m}_{\text{dummy}}$ ”, where $\vec{m}_{\text{dummy}}$ is the message used by the simulator $\mathcal{S}$ as a stand-in for the actual unknown message in the blind signing procedure (see Theorem 5.8). We do not want this to happen, because in the final step of this proof, which is a reduction to Π_{blind} ’s unforgeability, we will have to query signatures on $\vec{m}_{\text{dummy}}$ in order to make the simulator $\mathcal{S}$ work, so we cannot claim $\vec{m}_{\text{dummy}}$ as a forgery. Thankfully, intuitively, $\mathcal{A}$ cannot guess $\vec{m}_{\text{dummy}}$, so it is unlikely that $\text{fail}_{\text{dummy}}$ happens. Consider a modified game G_{zero} that works like $\text{Exp}_{\Pi_{\text{uacs}}}^{\text{sim-expl}}(\Pi_{\text{uacs}}, \mathcal{A}, \mathcal{S}, \mathcal{E}, n, \lambda)$ except that whenever $\mathcal{S}$ would use $\vec{m}_{\text{dummy}}$ as input to $\Pi_{\text{blind}}.\text{Commit}$ or $\Pi_{\text{blind}}.\text{BlindReceive}$, it instead uses $\vec{0}$ (of the appropriate length). As a consequence, the dummy message $m_{\text{dummy}} \leftarrow \mathbb{M}$ chosen randomly by $\mathcal{S}$ is never used in G_{zero} ($\mathcal{E}$ has access to it through *state_S*, but only reads the value to check whether $\text{fail}_{\text{dummy}}$ happens). Because of that and because the message space has super-poly size, $\Pr[\text{fail}_{\text{dummy}} \mid G_{\text{zero}}] \leq q(\lambda)/|\mathbb{M}_{\mathcal{G}}|$, where q is a polynomial upper-bounding the number of calls to the explainer $\mathcal{E}$, is negligible. ! As a consequence, as we show next, $\Pr[\text{fail}_{\text{dummy}} \mid \text{Exp}_{\Pi_{\text{uacs}}}^{\text{sim-expl}}]$ should also be negligible, because the Π_{blind} blindness property ensures that $\mathcal{A}$ cannot distinguish between $\mathcal{S}$ using $\vec{m}_{\text{dummy}}$ or $\vec{0}$.

We construct distinguisher $\mathcal{B}$ against the signing blindness property (Definition 5.6)

of Π_{blind} . $\mathcal{B}^{\mathcal{O}.\text{BlindReceive}^b}(\mathcal{G}, pp_0)$ internally runs $\text{Exp}_{\Pi_{\text{uacs}}}^{\text{sim-expl}}(\Pi_{\text{uacs}}, \mathcal{A}, \mathcal{S}, \mathcal{E}, n, \lambda)$ with the following differences:

- $\mathcal{B}$ uses $\mathcal{G}$ and pp_0 from its input instead of generating them at the beginning of $\text{Exp}_{\Pi_{\text{uacs}}}^{\text{sim-expl}}$.
- Whenever the simulator $\mathcal{S}$ would compute the commitment C and run BlindReceive , both are delegated to $\mathcal{B}$'s oracle $\mathcal{O}.\text{BlindReceive}^b(ipk, \vec{m}_{\text{dummy}}, \vec{0})$. ? So for $b = 0$, $\mathcal{B}$ behaves like $\text{Exp}_{\Pi_{\text{uacs}}}^{\text{sim-expl}}$, using $\vec{m}_{\text{dummy}}$ for simulation, and for $b = 1$, it behaves like $\mathcal{G}_{\text{zero}}$, using $\vec{0}$. ($\mathcal{S}$ then cannot keep the list of (C, o) in its state because it does not get to see o , but that list is not accessed in $\text{Exp}_{\Pi_{\text{uacs}}}^{\text{sim-expl}}(\Pi_{\text{uacs}}, \mathcal{A}, \mathcal{S}, \mathcal{E}, n, \lambda)$, anyway.)

When $\text{fail}_{\text{dummy}}$ happens, $\mathcal{B}$ outputs 1. Otherwise, if that event does not happen during its run, then $\mathcal{B}$ outputs 0. It is easy to check that

$$\Pr[\mathcal{B}^{\mathcal{O}.\text{BlindReceive}^0}(\mathcal{G}, pp_0) = 1] = \Pr[\text{fail}_{\text{dummy}} \mid \mid \text{Exp}_{\Pi_{\text{uacs}}}^{\text{sim-expl}}(\Pi_{\text{uacs}}, \mathcal{A}, \mathcal{S}, \mathcal{E}, n, \lambda)]$$

and

$$\Pr[\mathcal{B}^{\mathcal{O}.\text{BlindReceive}^1}(\mathcal{G}, pp_0) = 1] = \Pr[\text{fail}_{\text{dummy}} \mid \mid \mathcal{G}_{\text{zero}}]$$

Because $\mathcal{B}$ is ppt, we get from Definition 5.6 that

$$\begin{aligned} & \left| \Pr[\mathcal{B}^{\mathcal{O}.\text{BlindReceive}^0}(\mathcal{G}, pp_0) = 1] - \Pr[\mathcal{B}^{\mathcal{O}.\text{BlindReceive}^1}(\mathcal{G}, pp_0) = 1] \right| \\ &= \left| \Pr[\text{fail}_{\text{dummy}} \mid \mid \text{Exp}_{\Pi_{\text{uacs}}}^{\text{sim-expl}}(\Pi_{\text{uacs}}, \mathcal{A}, \mathcal{S}, \mathcal{E}, n, \lambda)] - \Pr[\text{fail}_{\text{dummy}} \mid \mid \mathcal{G}_{\text{zero}}] \right| \end{aligned}$$

is negligible. Because $\Pr[\text{fail}_{\text{dummy}} \mid \mid \mathcal{G}_{\text{zero}}]$ is negligible (as explained above), we get that $\Pr[\text{fail}_{\text{dummy}} \mid \mid \text{Exp}_{\Pi_{\text{uacs}}}^{\text{sim-expl}}(\Pi_{\text{uacs}}, \mathcal{A}, \mathcal{S}, \mathcal{E}, n, \lambda)]$ must be negligible, too.

Showing that $\Pr[\text{fail}_{\mathcal{E}_{\text{zk}}}]$ is negligible. ? $\text{fail}_{\mathcal{E}_{\text{zk}}}$ is the event that the proof of knowledge extractor $\mathcal{E}_{\text{zk}}$ within the UACS explainer $\mathcal{E}$ fails to provide a valid witness at some point. We can show that $\Pr[\text{fail}_{\mathcal{E}_{\text{zk}}}]$ is negligible via reduction to the simulation-extractability property of Π_{zk} . For this, we construct the ppt $\mathcal{B}^{\mathcal{S}_{\text{zk}}, \mathcal{RO}, \mathcal{O}.\text{Simul}, \mathcal{O}.\text{Extract}}(\mathcal{G}, pp_1)$, which internally runs $\text{Exp}_{\Pi_{\text{uacs}}}^{\text{sim-expl}}(\Pi_{\text{uacs}}, \mathcal{A}, \mathcal{S}, \mathcal{E}, n, \lambda)$ with the following differences: ? Essentially, $\mathcal{B}$ never runs $\mathcal{S}_{\text{zk}}$ or $\mathcal{E}_{\text{zk}}$ itself, but delegates this to its oracles.

- $\mathcal{B}$ uses $\mathcal{G}$ and pp_1 from its input instead of generating them at the beginning of $\text{Exp}_{\Pi_{\text{uacs}}}^{\text{sim-expl}}(\Pi_{\text{uacs}}, \mathcal{A}, \mathcal{S}, \mathcal{E}, n, \lambda)$. ? As a consequence, $\mathcal{B}$ does not have access to $\text{state}_{\mathcal{S}_{\text{zk}}}$, so we replace all occurrences of $\mathcal{S}_{\text{zk}}$ and $\mathcal{E}_{\text{zk}}$ with oracle queries below.
- Whenever $\mathcal{S}$ would execute $\mathcal{S}_{\text{zk}}(stmt, ctx)$ (for some $stmt, ctx$), $\mathcal{B}$ instead calls its oracle $\mathcal{O}.\text{Simul}(stmt, ctx)$. ? This happens during oracle $\mathcal{O}.\text{SimReceiveCred}$, $\mathcal{O}.\text{SimReceiveUpdate}$, and $\mathcal{O}.\text{SimPresent}$ queries, since the Π_{uacs} simulator $\mathcal{S}$ calls the Π_{zk} simulator $\mathcal{S}_{\text{zk}}$ as a sub-routine.

- Whenever $\mathcal{E}$ would execute $\Pi_{\text{zk}}.\text{Verify}(stmt, ctx) \leftrightarrow \mathcal{A}$ (for some $stmt, ctx$) followed by $w \leftarrow \mathcal{E}_{\text{zk}}(stmt, ctx, \tau_{\text{zk}}, state_{\mathcal{S}_{\text{zk}}})$, $\mathcal{B}$ instead calls its oracle $w \leftarrow \mathcal{O}.\text{Extract}(stmt, ctx)$ and relays the Π_{zk} messages between the oracle and $\mathcal{A}$. ? This happens whenever the UACS explainer $\mathcal{E}$ is run within $\mathcal{O}.\text{IssueCred}$, $\mathcal{O}.\text{UpdateCred}$, and $\mathcal{O}.\text{VerifyCred}$. See Figures 5.7 to 5.9.
- Random oracle calls $\mathcal{S}.\mathcal{RO}(\cdot)$ are answered using $\mathcal{B}$'s $\mathcal{S}_{\text{zk}}.\mathcal{RO}(\cdot)$ oracle.

It is easy to check that $\Pr[\text{ExtractionError} \parallel \mathcal{B}^{\mathcal{S}_{\text{zk}}.\mathcal{RO}, \mathcal{O}.\text{Simul}, \mathcal{O}.\text{Extract}}(\mathcal{G}, pp_1)] = \Pr[\text{fail}_{\mathcal{E}_{\text{zk}}} \parallel \text{Exp}_{\Pi_{\text{uacs}}}^{\text{sim-expl}}(\Pi_{\text{uacs}}, \mathcal{A}, \mathcal{S}, \mathcal{E}, n, \lambda)]$, where **ExtractionError** is the error triggered in Definition 3.22 when $\mathcal{E}_{\text{zk}}$ outputs an invalid witness for $(stmt, ctx) \notin \mathcal{T}_{\text{zk}}$. Because $\mathcal{B}$ is a ppt, the simulation-extractability property (Definition 3.22) implies that $\Pr[\text{ExtractionError} \parallel \mathcal{B}^{\mathcal{S}_{\text{zk}}.\mathcal{RO}, \mathcal{O}.\text{Simul}, \mathcal{O}.\text{Extract}}(\mathcal{G}, pp_1)]$ is negligible, which also means that $\Pr[\text{fail}_{\mathcal{E}_{\text{zk}}} \parallel \text{Exp}_{\Pi_{\text{uacs}}}^{\text{sim-expl}}(\Pi_{\text{uacs}}, \mathcal{A}, \mathcal{S}, \mathcal{E}, n, \lambda)]$ is negligible.

Showing that $\Pr[\neg \text{fail}_{\mathcal{E}_{\text{zk}}} \wedge \neg \text{fail}_{\text{dummy}} \wedge \text{fail}_{\text{explain}}]$ is negligible. ? As one can see in Figures 5.7 to 5.9, there are two ways to trigger $\text{fail}_{\text{explain}}$: (1) If the explanation does not fit the update function ψ or the predicate ϕ , i.e., $\psi(\cdot, \alpha) = \perp$ or $\phi(\overrightarrow{attr}, \alpha) = 0$ for the values output by $\mathcal{E}_{\text{zk}}$, or (2) if $\overrightarrow{attr} \notin \mathcal{T}_{\text{explained}}$ in $\mathcal{O}.\text{UpdateCred}$ or $\mathcal{O}.\text{VerifyCred}$. Given $\neg \text{fail}_{\mathcal{E}_{\text{zk}}}$, the event $\text{fail}_{\text{explain}}$ cannot be triggered by (1) because $\overrightarrow{attr}, \alpha$ were extracted from a proof guaranteeing exactly those properties. Hence the only way to trigger $\text{fail}_{\text{explain}}$ is through the $\overrightarrow{attr} \notin \mathcal{T}_{\text{explained}}$ criterion (2) in $\mathcal{O}.\text{UpdateCred}$ or $\mathcal{O}.\text{VerifyCred}$. But if that happens, then intuitively, the honest issuer has never issued a (blind) signature on $\overrightarrow{attr}$, so the extracted signature $cred$ on $\overrightarrow{attr}$ must be a forgery.

We prove that $\Pr[\neg \text{fail}_{\mathcal{E}_{\text{zk}}} \wedge \neg \text{fail}_{\text{dummy}} \wedge \text{fail}_{\text{explain}}]$ is negligible via reduction to the unforgeability property (Definition 5.5) of Π_{blind} . For this, we construct $\mathcal{B}$ against existential unforgeability under chosen-known-message attacks of Π_{blind} . $\mathcal{B}^{\mathcal{O}.\text{BlindSign}}(\mathcal{G}, pp_0, ipk^*)$ internally runs $\text{Exp}_{\Pi_{\text{uacs}}}^{\text{sim-expl}}(\Pi_{\text{uacs}}, \mathcal{A}, \mathcal{S}, \mathcal{E}, n, \lambda)$ with the following differences.

- $\mathcal{B}$ uses $\mathcal{G}, pp_0, ipk^*$ from its input instead of generating them at the beginning of $\text{Exp}_{\Pi_{\text{uacs}}}^{\text{sim-expl}}$. ? As a consequence, $\mathcal{B}$ does not have access to isk^* , so it cannot execute $\Pi_{\text{blind}}.\text{BlindSign}$ itself.
- In $\mathcal{O}.\text{IssueCred}$, $\mathcal{O}.\text{UpdateCred}$, instead of running $\Pi_{\text{blind}}.\text{BlindSign}(pp, ipk^*, C; isk^*)$ (as in Figures 5.7 and 5.8), $\mathcal{B}$ instead calls its oracle $\mathcal{O}.\text{BlindSign}$, with input depending on the following case distinction:
 - If there is some o' such that $(ipk^*, C, o') \in \mathcal{T}_{\text{simcommit}}$, then $\mathcal{B}$ calls the oracle $\mathcal{O}.\text{BlindSign}(\vec{m}_{\text{dummy}}, o')$ (where $\mathcal{T}_{\text{simcommit}}, \vec{m}_{\text{dummy}}$ are read from $state_{\mathcal{S}}$). ? We know how to open the commitment C because our simulator $\mathcal{S}$ has created C . See Figure 5.6.
 - Otherwise, $\mathcal{B}$ calls $\mathcal{O}.\text{BlindSign}(\overrightarrow{attr}', o)$ using variables $\overrightarrow{attr}', o$ from the witness w output by $\mathcal{E}_{\text{zk}}$. ? Use the opening for C from the proof of knowledge extractor.

$\mathcal{B}$ relays messages between $\mathcal{A}$ and $\mathcal{O}.\text{BlindSign}$.

- If $\text{fail}_{\text{explain}}$ occurs during $\mathcal{O}.\text{UpdateCred}$, $\mathcal{O}.\text{VerifyCred}$, $\mathcal{B}$ outputs $(\overrightarrow{\text{attr}}, \text{cred})$ as a signature forgery (variables $\overrightarrow{\text{attr}}, \text{cred}$ as in Figures 5.8 and 5.9).
- If $\text{fail}_{\mathcal{E}_{\text{zk}}}$ or $\text{fail}_{\text{dummy}}$ are noted by $\mathcal{E}$ at some point, $\mathcal{B}$ immediately aborts execution.

First, we argue that whenever $\mathcal{B}$ calls $\mathcal{O}.\text{BlindSign}$, then the protocol run by the oracle uses the same C as $\mathcal{A}$ sent (i.e., $\mathcal{B}$ supplies a valid opening for C to $\mathcal{O}.\text{BlindSign}$).
 ♡ Remember that $\mathcal{O}.\text{BlindSign}(\overrightarrow{\text{attr}}, o)$ recomputes $C = \text{Commit}(pp, ipk^*, \overrightarrow{\text{attr}}, o)$, then runs $\text{BlindSign}(pp, ipk^*, C; isk^*) \leftrightarrow A$ with that C (see Definition 5.5), so if a different C were recomputed than what $\mathcal{A}$ sends, we are not convincingly simulating $\text{Exp}_{\Pi_{\text{uacs}}}^{\text{sim-expl}}(\Pi_{\text{uacs}}, \mathcal{A}, \mathcal{S}, \mathcal{E}, n, \lambda)$ anymore. For this, consider some $\mathcal{O}.\text{IssueCred}$ or $\mathcal{O}.\text{UpdateCred}$ query. There are two cases: (1) If $\mathcal{S}_{\text{zk}}(stmt, ctx)$ has been called before, then the UACS simulator $\mathcal{S}$ has created the commitment C for ipk^* (as $\mathcal{S}_{\text{zk}}$ is only called within $\mathcal{S}$ and the statement $stmt$ fixes C and ipk^*), so there is some o' such that $(ipk^*, C, o') \in \mathcal{T}_{\text{simcommit}}$. Hence $C = \text{Commit}(pp, ipk^*, \vec{m}_{\text{dummy}}, o')$ as required. (2) If $\mathcal{S}_{\text{zk}}(stmt, ctx)$ has *not* been called at the point where $\mathcal{E}_{\text{zk}}$ outputs w , then necessarily, $(stmt, w) \in \mathcal{R}$ (because $(stmt, w) \notin \mathcal{R}$ would have triggered $\text{fail}_{\mathcal{E}_{\text{zk}}}$ and $\mathcal{B}$ would have aborted). In particular, this guarantees that $C = \Pi_{\text{blind}}.\text{Commit}(pp, ipk^*, \overrightarrow{\text{attr}}', o)$. Hence $C = \text{Commit}(pp, ipk^*, \overrightarrow{\text{attr}}', o)$ as required. ♡ Zooming out a bit, we note that case (1) can happen in the following example scenario, which is essentially a relay attack. $\mathcal{A}$ queries $\mathcal{O}.\text{SimReceiveCred}$ and $\mathcal{O}.\text{IssueCred}$, and relays the messages between the simulated user and honest issuer (the same discussion applies for $\mathcal{O}.\text{SimReceiveUpdate}$ and $\mathcal{O}.\text{UpdateCred}$). In this case, the ZKPoK proof, which we would usually extract a commitment opening from (case (2)), is simulated, and consequently cannot be extracted from. However, in this case, the simulator has created the commitment and has left opening information in $\mathcal{T}_{\text{simcommit}}$ that $\mathcal{B}$ can use.

As a consequence, unless $\text{fail}_{\mathcal{E}_{\text{zk}}}$ or $\text{fail}_{\text{dummy}}$ happen (in which case $\mathcal{B}$ would prematurely abort), $\mathcal{B}$ perfectly simulates $\text{Exp}_{\Pi_{\text{uacs}}}^{\text{sim-expl}}(\Pi_{\text{uacs}}, \mathcal{A}, \mathcal{S}, \mathcal{E}, n, \lambda)$, i.e., exactly the same algorithms are run with the same input in $\text{Exp}_{\Pi_{\text{uacs}}}^{\text{sim-expl}}(\Pi_{\text{uacs}}, \mathcal{A}, \mathcal{S}, \mathcal{E}, n, \lambda)$ as in the unforgeability game (Definition 5.5) with $\mathcal{B}$, just some of those computations are now happening within the $\mathcal{O}.\text{BlindSign}$ oracle of $\mathcal{B}$. As a consequence, $\Pr[\neg \text{fail}_{\mathcal{E}_{\text{zk}}} \wedge \neg \text{fail}_{\text{dummy}} \wedge \text{fail}_{\text{explain}} \mid \text{Exp}_{\Pi_{\text{uacs}}}^{\text{sim-expl}}(\Pi_{\text{uacs}}, \mathcal{A}, \mathcal{S}, \mathcal{E}, n, \lambda)] = \Pr[\neg \text{fail}_{\mathcal{E}_{\text{zk}}} \wedge \neg \text{fail}_{\text{dummy}} \wedge \text{fail}_{\text{explain}} \mid \mathcal{B}]$.

We now argue that when $\text{fail}_{\text{explain}}$ occurs in $\mathcal{B}$, then $\mathcal{B}$ indeed outputs a forgery in the Π_{blind} unforgeability game (Definition 5.5). For this, first note the following facts about $\mathcal{B}$ and its variable values in $\mathcal{O}.\text{IssueCred}$, $\mathcal{O}.\text{UpdateCred}$, or $\mathcal{O}.\text{VerifyCred}$ that can be easily checked by inspection of Figures 5.7 to 5.9.

1. If the event $\text{fail}_{\text{explain}}$ is raised during $\mathcal{O}.\text{IssueCred}$, $\mathcal{O}.\text{UpdateCred}$, or $\mathcal{O}.\text{VerifyCred}$, then $(stmt, w) \in \mathcal{R}$. ♡ When $\text{fail}_{\text{explain}}$ is raised, $\text{fail}_{\mathcal{E}_{\text{zk}}}$ has not occurred (otherwise, $\mathcal{B}$ would have aborted earlier) and $\mathcal{S}_{\text{zk}}(stmt, ctx)$ has not been called (because of the $\mathcal{T}_{\text{sim}}$ check). Hence necessarily, $(stmt, w) \in \mathcal{R}$.
2. $\text{fail}_{\text{explain}}$ cannot occur during $\mathcal{O}.\text{IssueCred}$ queries. ♡ $\text{fail}_{\text{explain}}$ would imply $(stmt, w) \in \mathcal{R}$ (see point 1), which implies $\psi(\perp, \alpha) \neq \perp$, which would mean that *explained* is set to 1,

contradicting $\text{fail}_{\text{explain}}$.

3. If $\text{fail}_{\text{explain}}$ happens during $\mathcal{O}.\text{UpdateCred}$ or $\mathcal{O}.\text{VerifyCred}$, then cred is a valid signature on $\overrightarrow{\text{attr}}$, i.e., $\text{Verify}(pp, \text{ipk}^*, \overrightarrow{\text{attr}}, \text{cred}) = 1$. ! This follows from point 1. Note for $\mathcal{O}.\text{VerifyCred}$ that, due to point 1, we necessarily have that $\text{ipk} = \text{ipk}^*$ whenever $\text{fail}_{\text{explain}}$ happens.
4. If $\text{fail}_{\text{explain}}$ happens during $\mathcal{O}.\text{UpdateCred}$ or $\mathcal{O}.\text{VerifyCred}$, then $\overrightarrow{\text{attr}} \notin \mathcal{T}_{\text{explained}}$. ! Because of point 1, the conditions $\psi(\overrightarrow{\text{attr}}, \alpha) \neq \perp$ or $\phi(\overrightarrow{\text{attr}}, \alpha) = 1$ are necessarily fulfilled. So the only way for $\text{fail}_{\text{explain}}$ to occur is by falsifying the $\text{attr} \in \mathcal{T}_{\text{explained}}$ condition.
5. $\mathcal{T}_{\text{signed}} \subseteq \mathcal{T}_{\text{explained}} \cup \{\vec{m}_{\text{dummy}}\}$. ! By definition, $\mathcal{B}$ only queries $\mathcal{O}.\text{BlindSign}$ with $\vec{m}_{\text{dummy}}$ and with $\overrightarrow{\text{attr}'}$ output by the extractor $\mathcal{E}_{\text{zk}}$ during $\mathcal{O}.\text{IssueCred}$ or $\mathcal{O}.\text{UpdateCred}$. In the latter case, right before querying $\mathcal{O}.\text{BlindSign}(\overrightarrow{\text{attr}'}, o)$, $\mathcal{B}$ adds $\overrightarrow{\text{attr}'}$ to $\mathcal{T}_{\text{explained}}$.

So whenever $\text{fail}_{\text{explain}}$ happens, $\mathcal{B}$ outputs a candidate forgery $(\overrightarrow{\text{attr}}, \text{cred})$, of which we know that $\text{Verify}(pp, \text{ipk}^*, \overrightarrow{\text{attr}}, \text{cred}) = 1$ (point 3). Furthermore, from point 4, we get that $\overrightarrow{\text{attr}} \notin \mathcal{T}_{\text{explained}}$. Because of $\neg \text{fail}_{\text{dummy}}$ ($\mathcal{B}$ would have aborted earlier, otherwise), we know that $\overrightarrow{\text{attr}} \neq \vec{m}_{\text{dummy}}$. Together with point 5, this implies that $\overrightarrow{\text{attr}} \notin \mathcal{T}_{\text{signed}}$.

Overall, whenever $\text{fail}_{\text{explain}} \wedge \neg \text{fail}_{\mathcal{E}_{\text{zk}}} \wedge \neg \text{fail}_{\text{dummy}}$, then $\mathcal{B}$ outputs a valid forgery on an unqueried message in the unforgeability game. Because $\mathcal{B}$ is a ppt, this means that $\Pr[\text{fail}_{\text{explain}} \wedge \neg \text{fail}_{\mathcal{E}_{\text{zk}}} \wedge \neg \text{fail}_{\text{dummy}}]$ must be negligible.

Conclusion. Overall, applying basic probability theory (law of total probability, basic inequalities) to the analysis of $\text{Exp}_{\Pi_{\text{uacs}}}^{\text{sim-expl}}(\Pi_{\text{uacs}}, \mathcal{A}, \mathcal{S}, \mathcal{E}, n, \lambda)$, we get $\Pr[\text{fail}_{\text{explain}}] \leq \Pr[\text{fail}_{\text{explain}} \wedge \neg \text{fail}_{\mathcal{E}_{\text{zk}}} \wedge \neg \text{fail}_{\text{dummy}}] + \Pr[\text{fail}_{\mathcal{E}_{\text{zk}}}] + \Pr[\text{fail}_{\text{dummy}}]$. We have shown above that all three summands are negligible, hence $\Pr[\text{fail}_{\text{explain}}]$ is negligible. $\square$

5.3.5 Instantiating the standard template

In this section, we show how to instantiate the standard template. We discuss a blind signature scheme that works well with Construction 5.7, prove that it fits our security formalization, and sketch how to instantiate the zero-knowledge proof system (the part involving the signature scheme; For concrete construction, composition with other statements, and additional capabilities, we refer to Chapter 3).

5.3.5.1 Pointcheval-Sanders signatures

A good instantiation for the (blind) signature scheme in Construction 5.7 are Pointcheval-Sanders signatures [PS16], which transfer the main ideas of CL signatures [CL04], used for early anonymous credential constructions, to the type 3 pairing setting. We present the Pointcheval-Sanders (blind) signature scheme here using our framework for blind signatures (Section 5.3.1).

Construction 5.10 (Pointcheval-Sanders blind signatures [PS16]). The Pointcheval-Sanders blind signature scheme Π_{ps} has message space $\mathbb{M}_{(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, e, p)} = \mathbb{Z}_p$ and commitment randomness space $\mathbb{O}_{(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, e, p)} = \mathbb{Z}_p$. It consists of the following algorithms.

Setup($\mathcal{G}$) $\rightarrow pp$

Outputs $pp = (\mathcal{G}, g, \tilde{g})$, where $g \leftarrow \mathbb{G}_1 \setminus \{1\}, \tilde{g} \leftarrow \mathbb{G}_2 \setminus \{1\}$.

Keygen($pp, 1^n$) $\rightarrow (pk, sk)$

Chooses secret key $sk = (x, y_1, \dots, y_n) \leftarrow \mathbb{Z}_p^{n+1}$ and derives the public key $pk = (\tilde{X}, (Y_i)_{i=1}^n, (\tilde{Y}_i)_{i=1}^n) = (\tilde{g}^x, (g^{y_i})_{i=1}^n, (\tilde{g}^{y_i})_{i=1}^n) \in \mathbb{G}_2 \times \mathbb{G}_1^n \times \mathbb{G}_2^n$.

Sign($pp, sk, \vec{m}$) $\rightarrow \sigma$

Chooses a random $h \leftarrow \mathbb{G}_1$ and outputs $\sigma = (h, h^{x + \sum_{i=1}^n y_i m_i})$.

Verify($pp, pk, \vec{m}, (\sigma_0, \sigma_1)$) $= b$

Outputs $b = 1$ iff $\sigma_0 \neq 1$ and $e(\sigma_1, \tilde{g}) = e(\sigma_0, \tilde{X} \cdot \prod_{i=1}^n \tilde{Y}_i^{m_i})$. $\heartsuit$ This check is equivalent to (σ_0, σ_1) being of the form $(h, h^{x + \sum_{i=1}^n y_i m_i})$ for some $h \neq 1$.

Commit($pp, pk, \vec{m}, o$) $= C$

Computes the commitment $C = g^o \cdot \prod_{i=1}^n Y_i^{m_i}$. $\heartsuit$ This is a Pedersen commitment to $\vec{m}$.

BlindSign($pp, pk, C; sk$) $\leftrightarrow$ BlindReceive($pp, pk, C; \vec{m}, o$) $\rightarrow \sigma$

Works as follows.

- **BlindSign** chooses a random $r \leftarrow \mathbb{Z}_p^*$ and sends $(\sigma'_0, \sigma'_1) = (g^r, (g^x \cdot C)^r)$ to the receiver. $\heartsuit$ If C has been computed honestly, this value is of the form $(h, h^{o+x+\sum y_i m_i})$, i.e., almost a valid signature except for the o dirt term.
- Then **BlindReceive** checks that $e(Y_i, \tilde{g}) = e(g, \tilde{Y}_i)$ for all $i \in [n]$. If the check fails, it outputs $\perp$. $\heartsuit$ This mitigates privacy attacks involving malformed Y_i , see Theorem 5.11. Otherwise, it computes and outputs $\sigma = (\sigma'_0, (\sigma'_0)^{-o} \cdot \sigma'_1)$. $\heartsuit$ This removes the o dirt term.

Π_{ps} is a *correct* blind signature scheme for signing committed values, as is easy to check.

Of course, [PS16] does not list the signature scheme in this form with the explicit commitment phase, nor with the same security notions, but in spirit, we did not make any significant changes to the scheme compared to [PS16].

A notable feature of Construction 5.10 is that any valid signature (σ_0, σ_1) can be easily randomized to yield a new valid signature with the same distribution as **Sign** would produce. This can be done by raising $(\sigma_0, \sigma_1) = (h, h^{x + \sum_{i=1}^n y_i m_i})$ to a random power $r \leftarrow \mathbb{Z}_p^*$, yielding a fresh signature $(h', (h')^{x + \sum_{i=1}^n y_i m_i})$, with $h' = h^r$ being distributed as in **Sign**. This feature is used in the **BlindSign** $\leftrightarrow$ **BlindReceive** protocol in order to produce a fresh-looking signature (which will be what enables the unforgeability proof, Theorem 5.12) and it simplifies building zero-knowledge proofs to prove knowledge of a signature (see Construction 5.13).

We now prove that Construction 5.10 fulfills our security requirements (Section 5.3.1).

Theorem 5.11. *Construction 5.10 has signing blindness (Definition 5.6).*

The proof of this theorem is purely information-theoretical. Because the Pedersen commitment scheme is perfectly hiding, the commitment given to the signer does not leak any information about the message. The main challenge is to argue that the verification bit computed by the receiver also does not leak any information about the hidden message. This is less clear because verification of the received signature seems to depend on the hidden message. However, we show that one can equivalently express the verification check independently of the message, meaning that the verification bit does not leak anything about the message, either.

Proof. Let $\mathcal{A}$ be an adversary. Consider a run of $\mathcal{O}.\text{BlindReceive}^b(pk, \vec{m}^{(0)}, \vec{m}^{(1)})$. The oracle generates a commitment $C = g^o \cdot \prod_{i=1}^n Y_i^{m_i^{(b)}}$ and hands C to $\mathcal{A}$, which replies with a blinded signature (σ'_0, σ'_1) . Finally, the oracle (within **BlindReceive**) unblinds the signature to $\sigma = (\sigma_0, \sigma_1) = (\sigma'_0, (\sigma'_0)^{-o} \cdot \sigma'_1)$. It outputs 1 iff the following conditions hold

- (1) $e(Y_i, \tilde{g}) = e(g, \tilde{Y}_i)$ (cf. **BlindReceive**) for the public key $pk = (\tilde{X}, (Y_i)_{i=1}^n, (\tilde{Y}_i)_{i=1}^n)$ supplied by the adversary
- (2) $\sigma_0 \neq 1$ (cf. **Verify**)
- (3) $e(\sigma_1, \tilde{g}) = e(\sigma_0, \tilde{X} \cdot \prod_{i=1}^n \tilde{Y}_i^{m_i^{(b)}})$ (cf. **Verify**)

We first make the observation that (3) can be rewritten in a way that only depends on the commitment C and not on the message $\vec{m}^{(b)}$. For this, let $r, x, y_1, \dots, y_n \in \mathbb{Z}_p$ be such that $\sigma_0 = \sigma'_0 = g^r$, $\tilde{X} = \tilde{g}^x$, and $\tilde{Y}_i = \tilde{g}^{y_i}$ for $i \in [n]$. Then

$$\begin{aligned}
 (1) \wedge (3) &\Leftrightarrow (1) \wedge e(\sigma_1, \tilde{g}) = e(\sigma_0, \tilde{X} \cdot \prod_{i=1}^n \tilde{Y}_i^{m_i^{(b)}}) \\
 &\Leftrightarrow (1) \wedge e(\sigma_1, \tilde{g}) = e(g^r, \tilde{g}^{x + \sum_{i=1}^n y_i m_i^{(b)}}) \\
 &\Leftrightarrow (1) \wedge e(\sigma_1, \tilde{g}) = e(g, \tilde{g})^{r \cdot (x + \sum_{i=1}^n y_i m_i^{(b)})} \\
 &\Leftrightarrow (1) \wedge \sigma_1 = g^{r \cdot (x + \sum_{i=1}^n y_i m_i^{(b)})} \\
 &\Leftrightarrow (1) \wedge \sigma_1 = (g^{x + \sum_{i=1}^n y_i m_i^{(b)}})^r
 \end{aligned}$$

💡 Using $Y_i = g^{y_i}$, as implied by (1)

$$\begin{aligned}
 &\Leftrightarrow (1) \wedge \sigma_1 = (g^x \cdot \prod_{i=1}^n Y_i^{m_i^{(b)}})^r \\
 &\Leftrightarrow (1) \wedge \sigma_1 = (g^x \cdot C \cdot g^{-o})^r
 \end{aligned}$$

⚡ Switching to $\mathcal{A}$'s sent values (σ'_0, σ'_1) :

$$\begin{aligned} &\Leftrightarrow (1) \wedge \sigma'_1 \cdot (\sigma'_0)^{-o} = (g^x \cdot C \cdot g^{-o})^r \\ &\Leftrightarrow (1) \wedge \sigma'_1 \cdot g^{-ro} = (g^x \cdot C \cdot g^{-o})^r \\ &\Leftrightarrow (1) \wedge \sigma'_1 = (g^x \cdot C)^r \end{aligned}$$

⚡ In other words, the final signature's σ_1 is valid iff $\mathcal{A}$ has correctly processed C using its secret key x and any randomness r , no matter what message is hidden in C .

Given this rewrite of the receiver's verification equations, consider the following oracle $\mathcal{O}.\text{ReceiveSim}$ that behaves exactly like $\mathcal{O}.\text{BlindReceive}^b$ (notably for both $b \in \{0, 1\}$).

$\mathcal{O}.\text{ReceiveSim}(pk, \vec{m}^{(0)}, \vec{m}^{(1)})$

```

assert  $\vec{m}_0, \vec{m}_1 \in \mathbb{M}^n$ 
Parse  $pk = (\tilde{X}, (Y_i)_{i=1}^n, (\tilde{Y}_i)_{i=1}^n)$ 
 $C \leftarrow \mathbb{G}_1$  //Random commitment
Hand  $C$  to  $\mathcal{A}$ , get  $(\sigma'_0, \sigma'_1)$ 
Let  $r, x \in \mathbb{Z}_p$  such that  $\sigma'_0 = g^r$  and  $\tilde{X} = \tilde{g}^x$ 
return 1 iff these conditions all hold:

•  $e(Y_i, \tilde{g}) = e(g, \tilde{Y}_i)$  ⚡ Check (1)

•  $\sigma'_0 \neq 1$  ⚡ Check (2)

•  $\sigma'_1 = (g^x \cdot C)^r$  ⚡ Corresponds to (3), see above.
```

⚡ This oracle cannot be efficiently implemented because of its use of discrete logarithms r, x . However, this is not an issue for this information-theoretical proof.

It is easy to see that from the point of view of $\mathcal{A}$, for all possible inputs, the oracle $\mathcal{O}.\text{ReceiveSim}(pk, \vec{m}^{(0)}, \vec{m}^{(1)})$ behaves exactly like $\mathcal{O}.\text{BlindReceive}^b(pk, \vec{m}^{(0)}, \vec{m}^{(1)})$ for both $b \in \{0, 1\}$: In either case, $\mathcal{A}$ gets a commitment C distributed uniformly in $\mathbb{G}_1$. The conditions for the oracles to return 1 are also equivalent, as argued above.

Overall, we get that $\Pr[\mathcal{A}^{\mathcal{O}.\text{BlindReceive}^b}(\mathcal{G}, pp) = 1] = \Pr[\mathcal{A}^{\mathcal{O}.\text{ReceiveSim}}(\mathcal{G}, pp) = 1]$ for $b \in \{0, 1\}$. Hence

$$\left| \Pr[\mathcal{A}^{\mathcal{O}.\text{BlindReceive}^0}(\mathcal{G}, pp) = 1] - \Pr[\mathcal{A}^{\mathcal{O}.\text{BlindReceive}^1}(\mathcal{G}, pp) = 1] \right| = 0 \quad \square$$

Theorem 5.12. *Let $\Pi_{\text{ps}} = (\text{Setup}, \text{Keygen}, \text{Sign}, \text{Verify}, \text{Commit}, \text{BlindSign}, \text{BlindReceive})$ be the scheme of Construction 5.10. If $(\text{Setup}, \text{Keygen}, \text{Sign}, \text{Verify})$ is EUF-CMA-secure (Definition 2.9), then Π_{ps} is existentially unforgeable under chosen-known-message attacks (Definition 5.5).*

A proof for the signature scheme (itself, without the blind signing) being EUF-CMA-secure can be found in [PS16]. It relies on the assumption (Assumption 1 in [PS16]) that the scheme is EUF-CMA-secure for a single message ($n = 1$), which can be shown

[PS16] to hold in the generic group model. In [PS18], they show that the scheme is also secure under a milder (non-interactive) assumption if, essentially, the signer includes an additional random message in each signature. We do not consider this altered scheme here for simplicity, but we note that the altered scheme can also be used in place of Construction 5.10.

We now prove that if the signature scheme (without blind signing) is EUF-CMA-secure, then also the full construction with blind signing is secure.

Proof of Theorem 5.12. Let $\mathcal{A}$ be a ppt adversary (against Π_{ps}). Construct an adversary against EUF-CMA-security of $(\text{Setup}, \text{Keygen}, \text{Sign}, \text{Verify})$ (i.e., without the blind protocol). Construct $\mathcal{B}^{\mathcal{O}.\text{Sign}}(\mathcal{G}, pp, pk)$ as follows:

- $\mathcal{B}$ starts running $\mathcal{A}(\mathcal{G}, pp, pk)$.
- Whenever $\mathcal{A}$ queries its $\mathcal{O}.\text{BlindSign}(\vec{m}, o)$ oracle, $\mathcal{B}$ queries its $\mathcal{O}.\text{Sign}(\vec{m})$ oracle to receive a signature (σ_0, σ_1) . $\mathcal{B}$ then sends $\sigma' = (\sigma_0, \sigma_0^o \cdot \sigma_1)$ to $\mathcal{A}$. $\mathcal{B}$ adds the dirt term o to the signature to be consistent with what $\mathcal{A}$ expects from BlindSign .
- Eventually, $\mathcal{A}$ outputs a candidate forgery $(\vec{m}^*, \sigma^*)$, which $\mathcal{B}$ also outputs.

Note that $\mathcal{A}$'s oracle queries are answered exactly as in the original protocol: In both cases, $\mathcal{A}$ is sent σ' of the form $(h, h^{o+x+\sum y_i m_i})$ for a uniformly random $h \leftarrow \mathbb{G}_1 \setminus \{1\}$.

Let $\text{win}_{\mathcal{A}}$ be the event that $\mathcal{A}$ succeeds in producing a forgery, i.e., $\text{Verify}(pp, pk, \vec{m}^*, \sigma^*) = 1$ and $\mathcal{A}$ has not queried for $\mathcal{O}.\text{BlindSign}(\vec{m}^*, \cdot)$. If $\text{win}_{\mathcal{A}}$, then $\mathcal{B}$ also outputs a valid forgery without querying $\mathcal{O}.\text{Sign}(\vec{m}^*)$. Because $\mathcal{B}$ is a ppt and $(\text{Setup}, \text{Keygen}, \text{Sign}, \text{Verify})$ is EUF-CMA-secure, this happens with negligible probability. Hence $\Pr[\text{win}_{\mathcal{A}}]$ must be negligible as well. $\square$

For the UACS template (Construction 5.7), we also need a protocol to prove, in zero-knowledge, knowledge of a valid opening for $\Pi_{\text{ps}}.\text{Commit}$, and of a valid Pointcheval-Sanders signature. Because $\Pi_{\text{ps}}.\text{Commit}$ is the Pedersen commitment scheme, it is easy to set up a Schnorr-based Sigma protocol for proving the required statement. A Sigma protocol for proving knowledge of a valid signature can be easily constructed with the framework presented in Chapter 3 as a Schnorr-based send-first protocol (Construction 3.12). The following construction is meant as a building block, subject to the composition described in Section 3.6 in order to get the full protocols required in Construction 5.7.

Construction 5.13 (Proving knowledge of a valid Pointcheval-Sanders signatures on hidden messages). Figure 5.10 depicts a send-first Sigma protocol (Construction 3.12) for the relation $\mathcal{R}_{\mathcal{G}} = \{(stmt = (\Pi_{\text{ps}}.pp, pk, \phi), w = (\sigma_0, \sigma_1, \vec{m})) \mid \Pi_{\text{ps}}.pp = (\mathcal{G}, g, \tilde{g}) \wedge g \in \mathbb{G}_1 \wedge \tilde{g} \in \mathbb{G}_2 \setminus \{1\} \wedge pk \in \mathbb{G}_2 \times \mathbb{G}_1^n \times \mathbb{G}_2^n \wedge \phi(\vec{m}) = 1 \wedge \text{Verify}(\Pi_{\text{ps}}.pp, pk, \vec{m}, (\sigma_0, \sigma_1)) = 1\}$. It does not require additional setup, i.e., $\text{SetupFirst}(\mathcal{G}) = \perp$.

To clarify the Schnorr homomorphism for the inner protocol, note that the expression $e(\sigma'_1, \tilde{g}) = e(\sigma'_0, \tilde{X} \cdot \prod_{i=1}^n \tilde{Y}_i^{m_i})$ can be equivalently written as $e(\sigma'_1, \tilde{g}) \cdot e(\sigma'_0, \prod_{i=1}^n \tilde{Y}_i^{-m_i}) = e(\sigma'_0, \tilde{X})$, which makes the left-hand side of the equation a homomorphism over $\mathbb{G}_1 \times \mathbb{Z}_p^n \rightarrow$

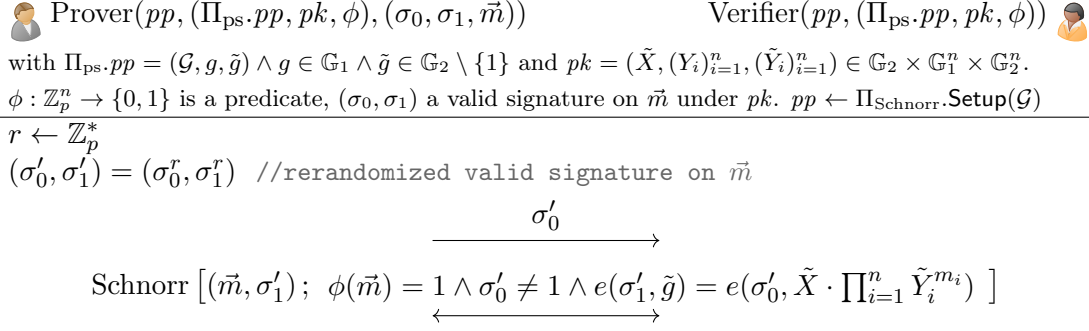


Figure 5.10: A send-first protocol (Construction 3.12) to prove knowledge of a signature on hidden messages fulfilling some public predicate ϕ .

$\mathbb{G}_t$ and the right-hand side a constant (constant in the context of the Schnorr protocol/homomorphism). We also note that the “ $\sigma'_0 \neq 1$ ” part of the Schnorr statement can be checked in plain by the verifier and hence can be handled outside the homomorphism.

The original paper [PS16] also provides a similar protocol. Instead of using the Schnorr protocol’s ability to handle $\mathbb{G}_1$ elements as witnesses (see Construction 3.7) in order to use σ'_1 as a witness directly, they have the prover send $\sigma'_1 = (\sigma_0^t \sigma_1)^r$ as an additional send-first value and prove he knows the derandomization factor t . Our Sigma protocol is simpler (fewer randomization factors) and more efficient with no downside, sending one less $\mathbb{Z}_p$ element than [PS16] and computing fewer group operations.

It is easy to see that Construction 5.13 is special honest-verifier zero-knowledge (via Theorem 3.13, $\text{SimFirst}(pp, stmt)$ outputting a random send-first message $\sigma'_0 \leftarrow \mathbb{G}_1 \setminus \{1\}$). It has (computational) special soundness (via Theorem 3.14, using $\text{InterpretWitness}(pp, stmt, \sigma'_0, (\vec{m}, \sigma'_1)) = (\sigma'_0, \sigma'_1, \vec{m})$ to compute a valid signature from the inner protocol’s witness).

If one requires the protocol to have quasi-unique responses (e.g., to be able to apply Construction 3.23), one can apply a transformation similar to Construction 3.15, having the prover additionally commit to the messages $\vec{m}$ using a Pedersen commitment and adding a statement that $\vec{m}$ are indeed committed to the Schnorr proof. In contrast to the spirit of Construction 3.15, there is no need to explicitly commit to σ'_1 (even though it is part of the witness) because the inner Schnorr protocol’s homomorphism $\Phi(\sigma'_1, \vec{m}) = e(\sigma'_1, \tilde{g}) \cdot e(\sigma'_0, \prod_{i=1}^n \tilde{Y}_i^{-m_i})$ is already injective when fixing $\vec{m}$ (i.e., $\sigma'_1 \mapsto \Phi(\sigma'_1, \vec{m})$ is injective for all $\vec{m}$). This means that if $\vec{m}$ is committed to, this already effectively prevents collisions for Φ . With this insight, the quasi-unique response property can be proven similar to Theorem 3.16. We leave the details to the reader.

5.4 Traditional credentials from updatable credentials

We now construct a traditional anonymous credential system from an updatable anonymous credential system. This serves as a simple example to demonstrate generically implementing pseudonyms using the UACS capabilities, and as a showcase for how to

use the UACS security definitions. The basic idea for this construction is quite simple: A traditional credential with attributes $\overrightarrow{attr}$ will be a UACS credential with attributes $\overrightarrow{attr}^{\text{uacs}} = (usk, \overrightarrow{attr})$, where usk is the user's secret. We will use an update function $\psi_{nym, \overrightarrow{attr}}^{\text{uacs}}$ and the hidden parameter α to hide usk from the issuer. When presenting the credential, the predicate $\phi_{nym, \phi}^{\text{uacs}}$ ensures that the attributes fulfill the desired access policy ϕ , as well as that the user knows usk consistent with nym .

When instantiating Construction 5.14 with the UACS standard template (Construction 5.7), then the resulting traditional anonymous credential system also simply follows its standard template (Section 4.3). This goes to show that our very powerful updatable credential formalism does not add any performance penalty when applied to simple use cases. Rather, UACS offer a closer description for what is possible to achieve with the standard techniques (whereas the traditional credential formalism loses some functionality, such as hidden attributes, in its abstraction).

Construction 5.14 (Traditional ACS from UACS). Let Π_{nym} be a pseudonym system (Definition 4.1). Let Π_{uacs} be an updatable anonymous credential system (Definition 5.1) with attribute space $\mathbb{A}_{\mathcal{G}}^{\text{uacs}}$, update function space $\Psi_{\mathcal{G}}^{\text{uacs}}$, and predicate space $\Phi_{\mathcal{G}}^{\text{uacs}}$.

We construct a traditional anonymous credential system Π_{acs} (Definition 4.7) with attribute space $\mathbb{A}_{\mathcal{G}}$ and predicate space $\Phi_{\mathcal{G}}$ as follows. Let

$$\psi_{nym, \overrightarrow{attr}}(\perp, (usk, nsk)) = \begin{cases} (usk, \overrightarrow{attr}) & \text{if } \Pi_{nym}.\text{VerifyNym}(pp, usk, nym, nsk) = 1 \\ \perp & \text{otherwise} \end{cases}$$

and let

$$\phi_{nym, \phi}^{\text{uacs}}((usk, \overrightarrow{attr}), nsk) = [\phi(\overrightarrow{attr}) = 1 \wedge \Pi_{nym}.\text{VerifyNym}(pp, usk, nym, nsk) = 1].$$

We require that $\mathbb{A}_{\mathcal{G}}^{\text{uacs}} = \mathbb{A}_{\mathcal{G}}$ and $usk \in \mathbb{A}_{\mathcal{G}}$ for all usk output by GenNym . Furthermore, we require that if $\overrightarrow{attr} \in \mathbb{A}_{\mathcal{G}}$, $\phi \in \Phi$, then $\phi_{nym, \phi}^{\text{uacs}} \in \Phi_{\mathcal{G}}^{\text{uacs}}$ and $\psi_{nym, \overrightarrow{attr}} \in \Psi_{\mathcal{G}}^{\text{uacs}}$ for all nym . Π_{acs} works as follows.

$\text{Setup}(\mathcal{G}) \rightarrow (pp, ntd)$

Computes $pp_0 \leftarrow \Pi_{\text{uacs}}.\text{Setup}(\mathcal{G})$ and $(pp_1, ntd) \leftarrow \Pi_{nym}.\text{Setup}(\mathcal{G})$. It sets $pp = (pp_0, pp_1)$ and outputs (pp, ntd) .

$\text{UserKeyGen}(pp) \rightarrow (upk, usk)$

Outputs $(upk, usk) \leftarrow \Pi_{nym}.\text{KeyGen}(pp)$.

$\text{GenNym}(pp, usk) \rightarrow (nym, nsk)$

Outputs $(nym, nsk) \leftarrow \Pi_{nym}.\text{GenNym}(pp, usk)$.

$\text{InspectNym}(pp, nym, ntd) = upk$

Outputs $upk = \Pi_{nym}.\text{InspectNym}(pp, nym, ntd)$.

$$\begin{array}{l}
\text{IssuerKeyGen}(pp, 1^n) \rightarrow (ipk, isk) \\
\hline
\text{Outputs } (ipk, isk) \leftarrow \Pi_{\text{uacs}}.\text{IssuerKeyGen}(pp, 1^{n+1}). \\
\\
cred \leftarrow \text{ReceiveCred}(pp, ipk, nym, \overrightarrow{attr}; nsk, usk) \leftrightarrow \text{IssueCred}(pp, ipk, nym, \overrightarrow{attr}; isk) \\
\hline
\text{The parties set } ctx^{\text{uacs}} = \text{"Issue"} \text{ and } \psi^{\text{uacs}} = \psi_{nym, \overrightarrow{attr}}. \text{ ReceiveCred runs } cred \leftarrow \Pi_{\text{uacs}}.\text{ReceiveCred}(pp, ipk, \psi^{\text{uacs}}, ctx^{\text{uacs}}, \alpha) \text{ with } \alpha = (usk, nsk), \text{ while IssueCred runs } \Pi_{\text{uacs}}.\text{IssueCred}(pp, ipk, \psi^{\text{uacs}}, ctx^{\text{uacs}}, isk). \text{ ReceiveCred outputs } cred. \\
\\
\text{PresentCred}(pp, ipk, nym, \phi, ctx; \overrightarrow{attr}, nsk, usk, cred) \\
\leftrightarrow \text{VerifyCred}(pp, ipk, nym, \phi, ctx) \rightarrow b \\
\hline
\text{Both parties set } ctx^{\text{uacs}} = (\text{"verify"}, ctx) \text{ and } \phi^{\text{uacs}} = \phi_{nym, \phi}^{\text{uacs}}. \text{ PresentCred sets } \overrightarrow{attr}^{\text{uacs}} = (usk, \overrightarrow{attr}), \alpha = nsk \text{ and runs } \Pi_{\text{uacs}}.\text{PresentCred}(pp, ipk, \phi^{\text{uacs}}, ctx^{\text{uacs}}, \overrightarrow{attr}^{\text{uacs}}, cred, \alpha), \text{ while VerifyCred runs } \Pi_{\text{uacs}}.\text{VerifyCred}(pp, ipk, \phi^{\text{uacs}}, ctx^{\text{uacs}}) \rightarrow b. \text{ VerifyCred outputs } b. \\
\\
\text{CheckCred}(pp, ipk, \overrightarrow{attr}, usk, cred) = b \\
\hline
\text{Outputs } b = \text{CheckCred}(pp, ipk, \overrightarrow{attr}^{\text{uacs}}, cred) \text{ for } \overrightarrow{attr}^{\text{uacs}} = (usk, \overrightarrow{attr}).
\end{array}$$

Correctness follows immediately from correctness of Π_{nym} and Π_{uacs} .

Because Construction 5.14 is a traditional anonymous credential system, it does not offer any update functionality. We could also easily imagine doing credential updates in the presence of pseudonyms, using update functions such as

$$\psi_{nym, \psi}((usk, \overrightarrow{attr}), (\alpha, nsk)) = \begin{cases} \psi(\overrightarrow{attr}, \alpha) & \text{if } \Pi_{nym}.\text{VerifyNym}(pp, usk, nym, nsk) = 1 \\ \perp & \text{otherwise} \end{cases}$$

in order to execute an update ψ subject to pseudonym ownership. However, such an update functionality is inherently not supported by the unforgeability definition of traditional anonymous credentials (Definition 4.9), which by its nature needs to know exactly what attributes the honest issuer issues in order to decide the winning condition. For this reason, we keep to the standard traditional credential definition without updates.

In the following, we prove security of Construction 5.14.

Theorem 5.15 (Traditional ACS from UACS are anonymous). *If Π_{nym} has pseudonym unlinkability (Definition 4.2) and Π_{uacs} has simulation anonymity (Definition 5.2), then Construction 5.14 is anonymous (Definition 4.8).*

On an intuitive level, the adversary can learn something about a user in two ways: (1) by looking at the pseudonyms and (2) by looking at the messages of `ReceiveCred` or `PresentCred`. For the proof, we will first game-hop from the games $\text{Exp}_{\Pi_{\text{acs}}}^{\text{anon}}(\Pi_{\text{acs}}, \mathcal{A}, \lambda, b)$ to $\mathcal{G}_{\text{sim}}^b$, in which the UACS protocols are simulated. In $\mathcal{G}_{\text{sim}}^b$, because the UACS protocols are now run without requiring private input, the only visible information left about users are the pseudonyms. In the second step, we show that $\mathcal{G}_{\text{sim}}^0$ is indistinguishable from $\mathcal{G}_{\text{sim}}^1$ by reduction to the pseudonym unlinkability property.

Proof. Let $\mathcal{A}$ be a ppt adversary (against Π_{acs}). Let $\mathcal{S}$ be the (stateful, interactive) UACS simulator from Definition 5.2. As a first step, we define new games $G_{\text{sim}}^b(\lambda)$ as follows (read the code including the dashed boxes, excluding the solid boxes), with the behavior of $\text{Exp}_{\Pi_{\text{acs}}}^{\text{anon}}(\Pi_{\text{acs}}, \mathcal{A}, \lambda, b)$ listed for overview (read the code including the solid boxes, excluding the dashed boxes). ♡ Idea: G_{sim}^b replaces all execution of Π_{uacs} protocols in the anonymity game $\text{Exp}_{\Pi_{\text{acs}}}^{\text{anon}}$ by simulated versions.

$\text{Exp}_{\Pi_{\text{acs}}}^{\text{anon}}$ and G_{sim}^b

$\mathcal{G} \leftarrow \text{GenSetting}(1^\lambda)$

$pp_0 \leftarrow \Pi_{\text{uacs}}.\text{Setup}(\mathcal{G}), (pp_1, ntd) \leftarrow \Pi_{\text{nym}}.\text{Setup}(\mathcal{G}), pp = (pp_0, pp_1) \quad // \text{Exp}_{\Pi_{\text{acs}}}^{\text{anon}}$

$pp_0 \leftarrow \mathcal{S}.\text{Setup}(\mathcal{G}), (pp_1, ntd) \leftarrow \Pi_{\text{nym}}.\text{Setup}(\mathcal{G}), pp = (pp_0, pp_1) \quad // G_{\text{sim}}^b$

$(upk_0, usk_0), (upk_1, usk_1) \leftarrow \Pi_{\text{nym}}.\text{KeyGen}(pp)$, set $usk_\star = usk_b$

Run $\mathcal{A}(\mathcal{G}, pp, upk_0, upk_1)$ with oracle access to the oracles listed below.

Output whatever $\mathcal{A}$ outputs.

$\mathcal{O}.\text{GenNym}(U)$ in $\text{Exp}_{\Pi_{\text{acs}}}^{\text{anon}}$ and G_{sim}^b

assert $U \in \{0, 1, \star\}$

Choose new pseudonym handle N

Set $U_N := U$

$(nym_N, nsk_N) \leftarrow \text{GenNym}(pp, usk_U)$

return (nym_N, N)

$\mathcal{O}.\text{ReceiveCred}(ipk, N, \overrightarrow{attr})$ in $\text{Exp}_{\Pi_{\text{acs}}}^{\text{anon}}$ and G_{sim}^b

Choose new credential handle C

Set $ipk_C := ipk, U_C := U_N, \overrightarrow{attr}_C := \overrightarrow{attr}$

Set $cred_C = \perp$

//Start of $\Pi_{\text{acs}}.\text{ReceiveCred}$

Set $ctx^{\text{uacs}} = \text{"Issue"}$ and $\psi^{\text{uacs}} = \psi_{nym_N, \overrightarrow{attr}}$

// $\text{Exp}_{\Pi_{\text{acs}}}^{\text{anon}}$

Set $\alpha = (usk_{U_N}, nsk_N)$

$cred_C \leftarrow \Pi_{\text{uacs}}.\text{ReceiveCred}(pp, ipk, \psi^{\text{uacs}}, ctx^{\text{uacs}}; \alpha) \leftrightarrow \mathcal{A}$

// G_{sim}^b

$b \leftarrow \mathcal{S}.\text{ReceiveCred}(pp, ipk, \psi^{\text{uacs}}, ctx^{\text{uacs}}) \leftrightarrow \mathcal{A}$

Set $cred_C = 1$ if $b = 1$, otherwise set $cred_C = \perp$

return C if $cred_C \neq \perp$

$\mathcal{O}.\text{PresentCred}(N, C_0, C_1, \phi, ctx)$ in $\text{Exp}_{\Pi_{\text{acs}}}^{\text{anon}}$ and $\mathcal{G}_{\text{sim}}^b$

```

assert  $cred_{C_0} \neq \perp, cred_{C_1} \neq \perp$ 
assert  $ipk_{C_0} = ipk_{C_1}$ 
assert  $\phi(\overrightarrow{attr_{C_0}}) = \phi(\overrightarrow{attr_{C_1}}) = 1$ 
assert  $\{U_{C_0}, U_N\} \neq \{0, 1\}$  and  $\{U_{C_0}, U_N\} \neq \{\star, 1\}$ 
assert  $\{U_{C_1}, U_N\} \neq \{0, 1\}$  and  $\{U_{C_1}, U_N\} \neq \{\star, 0\}$ 
//Start of  $\Pi_{\text{acs}}.\text{PresentCred}$ 
Set  $ctx^{\text{uacs}} = ("verify", ctx)$  and  $\phi^{\text{uacs}} = \phi_{nym_N, \phi}^{\text{uacs}}$ 

// $\text{Exp}_{\Pi_{\text{acs}}}^{\text{anon}}$ 
Set  $\overrightarrow{attr}^{\text{uacs}} = (usk_{U_N}, \overrightarrow{attr})$  and  $\alpha = nsk_N$ 
 $\Pi_{\text{uacs}}.\text{PresentCred}(pp, ipk_{C_b}, \phi^{\text{uacs}}, ctx^{\text{uacs}}, \overrightarrow{attr}^{\text{uacs}}, cred_{C_b}, \alpha) \leftrightarrow \mathcal{A}$ 

// $\mathcal{G}_{\text{sim}}^b$ 
 $\mathcal{S}.\text{PresentCred}(pp, ipk_{C_0}, \phi^{\text{uacs}}, ctx^{\text{uacs}}) \leftrightarrow \mathcal{A}$ 

```

$\mathcal{O}.\mathcal{RO}(x)$ in $\text{Exp}_{\Pi_{\text{acs}}}^{\text{anon}}$ and $\mathcal{G}_{\text{sim}}^b$

```

return  $\mathcal{RO}(x)$  // $\text{Exp}_{\Pi_{\text{acs}}}^{\text{anon}}$ 

return  $\mathcal{S}.\mathcal{RO}(x)$  // $\mathcal{G}_{\text{sim}}^b$ 

```

Note that whenever $\text{Exp}_{\Pi_{\text{acs}}}^{\text{anon}}$ runs $\Pi_{\text{uacs}}.\text{ReceiveCred}(pp, ipk, \psi^{\text{uacs}}, ctx^{\text{uacs}}; \alpha)$, we have $\psi^{\text{uacs}}(\perp, \alpha) = 1$. Similarly, whenever $\text{Exp}_{\Pi_{\text{acs}}}^{\text{anon}}$ runs $\Pi_{\text{uacs}}.\text{PresentCred}(pp, ipk_{C_b}, \phi^{\text{uacs}}, ctx^{\text{uacs}}; \overrightarrow{attr}^{\text{uacs}}, cred_{C_b}, \alpha)$, we have $cred_{C_b} \neq \perp$ and $\phi^{\text{uacs}}(\overrightarrow{attr}, \alpha) = 1$.

We can show that $|\Pr[\text{Exp}_{\Pi_{\text{acs}}}^{\text{anon}}(\Pi_{\text{acs}}, \mathcal{A}, \lambda, b) = 1] - \Pr[\mathcal{G}_{\text{sim}}^b = 1]|$ (for both $b \in \{0, 1\}$) is negligible via reduction to the simulation anonymity property of Π_{uacs} . The reduction algorithm $\mathcal{B}^{\mathcal{O}.\mathcal{RO}^X, \mathcal{O}.\text{ReceiveCred}^X, \mathcal{O}.\text{ReceiveUpdate}^X, \mathcal{O}.\text{PresentCred}^X}(\mathcal{G}, pp_0)$ against the simulation anonymity property (Definition 5.2) runs $\text{Exp}_{\Pi_{\text{acs}}}^{\text{anon}}(\Pi_{\text{acs}}, \mathcal{A}, \lambda, b)$, but uses $\mathcal{G}, pp_0$ from its input. Furthermore, it replaces all calls to $\Pi_{\text{uacs}}.\text{ReceiveCred}(pp, ipk, \psi^{\text{uacs}}, ctx^{\text{uacs}}; \alpha)$ with the appropriate calls to $\mathcal{O}.\text{ReceiveCred}^X(ipk, \psi^{\text{uacs}}, ctx^{\text{uacs}}, \alpha)$. Finally, it replaces calls to $\Pi_{\text{uacs}}.\text{PresentCred}(pp, ipk_{C_b}, \phi^{\text{uacs}}, ctx^{\text{uacs}}; \overrightarrow{attr}^{\text{uacs}}, cred_{C_b}, \alpha)$ within the oracle $\mathcal{O}.\text{PresentCred}(N, C_0, C_1, \phi, ctx)$ with appropriate calls to $\mathcal{O}.\text{PresentCred}^X(C_b, \phi, ctx, \alpha)$. Random oracle $\mathcal{O}.\mathcal{RO}$ calls by $\mathcal{A}$ are relayed to $\mathcal{B}$'s corresponding oracle. Given that the assert statements in $\text{Exp}_{\Pi_{\text{acs}}}^{\text{anon}}$ ensure that protocols are always fed with valid input that survives the assertions in $\text{Exp}_{\Pi_{\text{uacs}}}^{\text{sim-anon}}$ (as remarked above), it is easy to check that for $X = \text{REAL}$, the view of $\mathcal{A}$ is distributed exactly as in $\text{Exp}_{\Pi_{\text{acs}}}^{\text{anon}}(\Pi_{\text{acs}}, \mathcal{A}, \lambda, b)$ and for $X = \text{SIM}$, the view of $\mathcal{A}$ is distributed exactly as in $\mathcal{G}_{\text{sim}}^b$. Because $\mathcal{B}$ is a ppt (given that $\mathcal{A}$ is a ppt), we get that

$$= |\Pr[\text{Exp}_{\Pi_{\text{uacs}}}^{\text{sim-anon}}(\Pi_{\text{uacs}}, \mathcal{S}, \mathcal{B}, \lambda, \text{REAL}) = 1] - \Pr[\text{Exp}_{\Pi_{\text{uacs}}}^{\text{sim-anon}}(\Pi_{\text{uacs}}, \mathcal{S}, \mathcal{B}, \lambda, \text{SIM}) = 1]|$$

$$= \left| \Pr[\text{Exp}_{\Pi_{\text{acs}}}^{\text{anon}}(\Pi_{\text{acs}}, \mathcal{A}, \lambda, b) = 1] - \Pr[G_{\text{sim}}^b(\lambda) = 1] \right|$$

is negligible for $b \in \{0, 1\}$.

For the second step, we show that $|\Pr[G_{\text{sim}}^0(\lambda) = 1] - \Pr[G_{\text{sim}}^1(\lambda) = 1]|$ is negligible. For this, we construct $\mathcal{B}$ against the pseudonym unlinkability property of Π_{nym} . $\mathcal{B}^{\mathcal{O}.\text{GenNym}_0, \mathcal{O}.\text{GenNym}_1, \mathcal{O}.\text{GenNym}_b}(\mathcal{G}, pp_1, upk_0, upk_1)$ runs G_{sim}^0 with the following differences:

- It uses pp_1, upk_0, upk_1 from its input instead of generating it in the setup phase. $\heartsuit$ As a consequence, $\mathcal{B}$ does not have access to ntd and usk_0, usk_1 . Note that ntd is unused in G_{sim}^0 and the usk_i are only used in $\mathcal{O}.\text{GenNym}$.
- In $\mathcal{O}.\text{GenNym}(U)$, instead of running $(nym_N, nsk_N) \leftarrow \text{GenNym}(pp, usk_U)$, $\mathcal{B}$ calls its $\mathcal{O}.\text{GenNym}_0$, $\mathcal{O}.\text{GenNym}_1$, or $\mathcal{O}.\text{GenNym}_b$ oracle for $U = 0$, $U = 1$, or $U = \star$, respectively, to receive the pseudonym nym_N . $\heartsuit$ $\mathcal{B}$ does not get the corresponding nsk_N , but because proofs in G_{sim}^0 are simulated, that value is never read by G_{sim}^0 , anyway.

One can easily check that if $\mathcal{B}$'s last oracle is $\mathcal{O}.\text{GenNym}_b$, then the view of $\mathcal{A}$ is distributed exactly as in G_{sim}^b (same b). $\heartsuit$ Meaning that for user $U = \star$, pseudonyms are generated using usk_b . Because $\mathcal{B}$ is a ppt (given that $\mathcal{A}$ is a ppt and the runtime of $\mathcal{S}$ can be polynomially bounded by guarantees of simulation anonymity), we get that

$$\begin{aligned} & \left| \Pr \left[\mathcal{A}^{\mathcal{O}.\text{GenNym}_0, \mathcal{O}.\text{GenNym}_1, \mathcal{O}.\text{GenNym}_0}(\mathcal{G}, pp, upk_0, upk_1) = 1 \right] \right. \\ & \quad \left. - \Pr \left[\mathcal{A}^{\mathcal{O}.\text{GenNym}_0, \mathcal{O}.\text{GenNym}_1, \mathcal{O}.\text{GenNym}_1}(\mathcal{G}, pp, upk_0, upk_1) = 1 \right] \right| \\ &= |\Pr[G_{\text{sim}}^0(\lambda) = 1] - \Pr[G_{\text{sim}}^1(\lambda) = 1]| \end{aligned}$$

is negligible. Overall, we have

$$|\Pr[\text{Exp}_{\Pi_{\text{acs}}}^{\text{anon}}(\Pi_{\text{acs}}, \mathcal{A}, \lambda, 0) = 1] - \Pr[\text{Exp}_{\Pi_{\text{acs}}}^{\text{anon}}(\Pi_{\text{acs}}, \mathcal{A}, \lambda, 1) = 1]|$$

$\heartsuit$ Triangle inequality

$$\begin{aligned} & \leq |\Pr[\text{Exp}_{\Pi_{\text{acs}}}^{\text{anon}}(\Pi_{\text{acs}}, \mathcal{A}, \lambda, 0) = 1] - \Pr[G_{\text{sim}}^0(\lambda) = 1]| \\ & \quad + |\Pr[G_{\text{sim}}^0(\lambda) = 1] - \Pr[G_{\text{sim}}^1(\lambda) = 1]| \\ & \quad + |\Pr[G_{\text{sim}}^1(\lambda) = 1] - \Pr[\text{Exp}_{\Pi_{\text{acs}}}^{\text{anon}}(\Pi_{\text{acs}}, \mathcal{A}, \lambda, 1) = 1]| \end{aligned}$$

We have shown above that all three summands are negligible, so as a consequence, $|\Pr[\text{Exp}_{\Pi_{\text{acs}}}^{\text{anon}}(\Pi_{\text{acs}}, \mathcal{A}, \lambda, 0) = 1] - \Pr[\text{Exp}_{\Pi_{\text{acs}}}^{\text{anon}}(\Pi_{\text{acs}}, \mathcal{A}, \lambda, 1) = 1]|$ is negligible as required. $\square$

Next, we prove unforgeability of our traditional anonymous credential construction (Construction 5.14) from UACS. This serves as a showcase for the UACS simulation-explainability definition.

Theorem 5.16 (Traditional ACS from UACS are unforgeable). *If Π_{uacs} has simulation-explainability (Definition 5.3) and Π_{nym} has binding pseudonyms (Definition 4.3) and is resistant to impersonation (Definition 4.4), then Π_{acs} (Construction 5.14) is unforgeable and has impersonation resistance (Definition 4.9).*

Intuitively, traditional credentials are a simple application: There is only a single hidden attribute, namely the user's secret usk . The traditional ACS unforgeability game $\text{Exp}_{\Pi_{\text{acs}}}^{\text{unforge}}$ can still efficiently check whether a forgery/impersonation occurs using the InspectNym algorithm. In our UACS instantiation, the explainer algorithm $\mathcal{E}$ can tell us exactly what usk is hidden behind a pseudonym nym .

Like any proof involving UACS simulation-explainability, our unforgeability proof will start out as follows. First, we will game-hop to a modified game G_{extr} that uses the UACS simulator $\mathcal{S}$ to simulate all honest users' protocols in $\text{Exp}_{\Pi_{\text{acs}}}^{\text{unforge}}$. This models that the $\text{Exp}_{\Pi_{\text{acs}}}^{\text{unforge}}$ adversary cannot learn anything from honest users and their credentials. In addition to that, all honest issuer/verifier UACS protocols X can be replaced with their explainer counterpart $\mathcal{E}.X$, which perfectly emulates the original protocol (so this change is not noticeable), but $\mathcal{E}$ additionally outputs explanations for each of the protocols, which in our case relates to a user's usk and his pseudonym secret nsk . In our case, these explanations are mostly ignored in G_{extr} (except for checking win conditions).

With this setup, we can analyze G_{extr} . We inspect the explanations and show that $\mathcal{A}$ cannot win by impersonation. If $\mathcal{A}$ were to impersonate an honest user, then, intuitively, the explainer will also output usk, nsk for the honest user's pseudonym. That would break impersonation resistance of the pseudonym system (note that the honest user UACS protocols are simulated, revealing no information about usk, nsk to $\mathcal{A}$).

Finally, we will have to argue that $\mathcal{A}$ cannot win by forging a credential. This is intuitively easy: If $\mathcal{A}$ is able to present a credential that was not issued, then also the explainer cannot output a consistent explanation for that presentation. This step involves arguing that the winning condition of $\text{Exp}_{\Pi_{\text{acs}}}^{\text{unforge}}$, which is defined over InspectNym for binding credentials to identities, aligns with the UACS explainer guarantees, which in this construction is essentially defined over the hidden attribute usk for binding credentials to identities. The pseudonym system's binding property provides this consistency.

Due to $\text{Exp}_{\Pi_{\text{acs}}}^{\text{unforge}}$ complexities, the actual proof is somewhat more involved.

Proof. Let $n \in \mathbb{N}$ and $\mathcal{A}$ be a ppt adversary (against unforgeability and impersonation resistance of Construction 5.14). The overall goal of the proof is to show that $\Pr[\text{Exp}_{\Pi_{\text{acs}}}^{\text{unforge}}(\Pi_{\text{acs}}, \mathcal{A}, n, \lambda) = 1]$ is negligible. Let $\mathcal{S}, \mathcal{E}$ be the simulator and explainer for Π_{uacs} from Definition 5.3.

Simulation. We first define a new game G_{expl} as follows. ? The new game G_{expl} will simulate honest user UACS protocols and use the UACS explainer to run honest issuer/verifier protocols, producing an explanation for those transactions that is ignored for now (except to check for events we will analyze later). Because this game behaves indistinguishably from the original $\text{Exp}_{\Pi_{\text{acs}}}^{\text{unforge}}(\Pi_{\text{acs}}, \mathcal{A}, n, \lambda)$, the rest of the proof will instead analyze G_{expl}

G_{expl} //Changes from $\text{Exp}_{\Pi_{\text{acs}}}^{\text{unforge}}(\Pi_{\text{acs}}, \mathcal{A}, n, \lambda)$: simulated UACS setup

$\mathcal{G} \leftarrow \text{GenSetting}(1^\lambda)$
 $pp_0 \leftarrow \mathcal{S}.\text{Setup}(\mathcal{G}), (pp_1, ntd) \leftarrow \Pi_{\text{nym}}.\text{Setup}(\mathcal{G}), pp = (pp_0, pp_1)$
 $(ipk^*, isk^*) \leftarrow \Pi_{\text{uacs}}.\text{IssuerKeyGen}(pp, 1^n)$
 Run $\mathcal{A}(\mathcal{G}, pp, ipk^*)$, granting it access to oracles $\mathcal{S}.\mathcal{RO}, \mathcal{O}.\text{IssueCred}, \mathcal{O}.\text{UpdateCred},$
 $\mathcal{O}.\text{VerifyCred}, \mathcal{O}.\text{SimReceiveCred}, \mathcal{O}.\text{SimReceiveUpdate}, \mathcal{O}.\text{SimPresent}$
 When an oracle raises $\text{fail}_{\text{explain}}$, output 1
 If no such event has been raised when $\mathcal{A}$ halts, output 0

$G_{\text{expl}}.\mathcal{O}.\text{IssueCred}(nym, \overrightarrow{attr})$ //Changes: Runs explainer, checks fail

Add $(nym, \overrightarrow{attr})$ to $\mathcal{T}_{\text{issued}}$
 //Start of IssueCred (with explainer)
 Set $ctx^{\text{uacs}} = \text{"Issue"}$ and $\psi^{\text{uacs}} = \psi_{nym, \overrightarrow{attr}}$. Set $explained = 0$.
 $b \leftarrow \mathcal{E}.\text{IssueCred}^{\mathcal{S}.\mathcal{RO}, \mathcal{O}.\text{GetSimState}, \mathcal{O}.\text{RegExpl}}(pp, ipk^*, \psi^{\text{uacs}}, ctx^{\text{uacs}}; isk^*) \leftrightarrow \mathcal{A}$

$\mathcal{O}.\text{RegExpl}(\alpha)$

Parse $\alpha = (usk, nsk)$
 assert $explained = 0 \wedge \psi^{\text{uacs}}(\perp, \alpha) \neq \perp$
 //Implies $\text{VerifyNym}(pp, usk, nym, nsk) = 1$ and $\psi^{\text{uacs}}(\perp, \alpha) = (usk, \overrightarrow{attr})$
 If $\text{InspectNym}(pp, nym, ntd) = upk_U$ for some U , note $\text{fail}_{\text{impersonate}}$
 Add $(usk, \overrightarrow{attr})$ to $\mathcal{T}_{\text{explained}}$
 Set $explained = 1$

$G_{\text{expl}}.\mathcal{O}.\text{VerifyCred}(ipk, nym, \phi, ctx)$ //Changes: Runs explainer, checks fail

//Start of VerifyCred

Set $ctx^{\text{uacs}} = ("verify", ctx)$ and $\phi^{\text{uacs}} = \phi_{nym, \phi}^{\text{uacs}}$. Set $explained = 0$.

$b \leftarrow \mathcal{E}.\text{VerifyCred}^{\mathcal{S}, \mathcal{RO}, \mathcal{O}.\text{GetSimState}, \mathcal{O}.\text{RegExpl}}(pp, ipk, \phi^{\text{uacs}}, ctx^{\text{uacs}})$

$\mathcal{O}.\text{RegExpl}(\overrightarrow{attr}^{\text{uacs}}, \alpha)$

Parse $\overrightarrow{attr}^{\text{uacs}} = (usk, \overrightarrow{attr})$, $\alpha = nsk$

assert $explained = 0 \wedge \phi^{\text{uacs}}(\overrightarrow{attr}^{\text{uacs}}, \alpha) = 1$

//Implies $\text{VerifyNym}(pp, usk, nym, nsk) = 1$ and $\phi(\overrightarrow{attr}) = 1$

assert $ipk \neq ipk^* \vee \overrightarrow{attr}^{\text{uacs}} \in \mathcal{T}_{\text{explained}}$

If $\text{InspectNym}(pp, nym, ntd) = upk_U$ for some U , note $\text{fail}_{\text{impersonate}}$

Set $explained = 1$

If $explained = 0$ and $b = 1$ and $(\text{PresentCred}, ipk, \phi^{\text{uacs}}, ctx^{\text{uacs}}) \notin \mathcal{T}_{\text{sim}}$:

note $\text{fail}_{\text{explain}}$

//Checking winning condition as in $\text{Exp}_{\Pi_{\text{acs}}}^{\text{unforge}}$

If $b = 0$ or $(ipk, nym, \phi, ctx) \in \mathcal{T}_{\text{presented}}$:

return b

$upk = \text{InspectNym}(pp, nym, ntd)$

If $\exists N : (nym_N = nym \vee \text{InspectNym}(pp, nym_N, ntd) = upk)$:

raise $\text{win}_{\text{impersonate}}$

If $ipk = ipk^*$:

If $\forall (nym', \overrightarrow{attr}') \in \mathcal{T}_{\text{issued}} : \phi(\overrightarrow{attr}') = 0 \vee \text{InspectNym}(pp, nym', ntd) \neq upk$:

raise $\text{win}_{\text{forgery}}$

return b

$G_{\text{expl}}.\mathcal{O}.\text{GenNym}(U)$ //Unchanged from $\text{Exp}_{\Pi_{\text{acs}}}^{\text{unforge}}$

If user handle U has not been queried to this oracle before:

Generate $(upk_U, usk_U) \leftarrow \text{UserKeyGen}(pp)$

Choose new pseudonym handle N and set $U_N := U$

$(nym_N, nsk_N) \leftarrow \text{GenNym}(pp, usk_U)$

return (upk_U, nym_N, N)

$G_{\text{expl}}.\mathcal{O}.\text{ReceiveCred}(ipk, N, \overrightarrow{attr})$ //Changes: simulates honest user

Choose new credential handle C

Set $ipk_C := ipk$, $N_C := N$, $\overrightarrow{attr}_C := \overrightarrow{attr}$, $cred_C := \perp$

Set $ctx^{\text{uacs}} = "Issue"$ and $\psi^{\text{uacs}} = \psi_{nym_N, \overrightarrow{attr}}$

Add $(\text{ReceiveCred}, ipk, \psi^{\text{uacs}}, ctx^{\text{uacs}})$ to $\mathcal{T}_{\text{sim}}$

$b \leftarrow \mathcal{S}.\text{ReceiveCred}(pp, ipk, \psi^{\text{uacs}}, ctx^{\text{uacs}}) \leftrightarrow \mathcal{A}$ //Simulate receiver

Set $cred_C = 1$ if $b = 1$, otherwise set $cred_C = \perp$

return C if $cred_C \neq \perp$

$G_{\text{expl}}.\mathcal{O}.\text{HonestIssue}(N, \overrightarrow{\text{attr}})$ //Changes: skips protocol

assert $\overrightarrow{\text{attr}} \in \mathbb{A}^n$
 Choose new credential handle C
 Set $\text{ipk}_C := \text{ipk}^*, N_C := N, \overrightarrow{\text{attr}}_C := \overrightarrow{\text{attr}}, \text{cred}_C := \perp$
 //Skip execution of $\text{cred}_C \leftarrow \text{ReceiveCred} \leftrightarrow \text{IssueCred}$
 Set $\text{cred}_C = 1$
 return C

$G_{\text{expl}}.\mathcal{O}.\text{PresentCred}(N, C, \phi, \text{ctx})$ //Changes: simulates honest user

assert $U_N = U_{N_C}$
 assert $\phi(\overrightarrow{\text{attr}}_C) = 1$
 Add $(\text{ipk}_C, \text{nym}_N, \phi, \text{ctx})$ to $\mathcal{T}_{\text{presented}}$
 Set $\text{ctx}^{\text{uacs}} = (\text{"verify"}, \text{ctx})$ and $\phi^{\text{uacs}} = \phi_{\text{nym}_N, \phi}^{\text{uacs}}$
 Add $(\text{PresentCred}, \text{ipk}_C, \phi^{\text{uacs}}, \text{ctx}^{\text{uacs}})$ to $\mathcal{T}_{\text{sim}}$
 $S.\text{PresentCred}(pp, \text{ipk}_C, \phi^{\text{uacs}}, \text{ctx}^{\text{uacs}}) \leftrightarrow \mathcal{A}$ //Simulate presentation

$G_{\text{expl}}.\mathcal{O}.\text{GetSimState}()$ //Used by explainer $\mathcal{E}$. Not accessible to $\mathcal{A}$

return state_S

⚡ G_{expl} works like $\text{Exp}_{\Pi_{\text{acs}}}^{\text{unforge}}(\Pi_{\text{acs}}, \mathcal{A}, n, \lambda)$ except that (1) the UACS simulator is used instead of honest user protocol executions with $\mathcal{A}$, (2) the UACS explainer is used instead of honest issuer/verifier protocol executions, and (3) G_{expl} adds some additional behavior, which is transparent for $\mathcal{A}$ and does not affect the occurrence/non-occurrence of $\text{win}_{\text{forgery}}$ or $\text{win}_{\text{impersonate}}$. Note the addition of noted events $\text{fail}_{\text{explain}}$, which occurs when the UACS explainer fails, and $\text{fail}_{\text{impersonate}}$, which occurs when we get an explanation for the adversary using a pseudonym belonging to/tracing to an honest user (not to be confused with $\text{win}_{\text{impersonate}}$ as in $\text{Exp}_{\Pi_{\text{acs}}}^{\text{unforge}}$).

As a first step, we will show that $|\Pr[\text{Exp}_{\Pi_{\text{acs}}}^{\text{unforge}}(\Pi_{\text{acs}}, \mathcal{A}, n, \lambda) = 1] - \Pr[G_{\text{expl}} = 1]|$ is negligible by reduction to simulation anonymity of Π_{uacs} (Definition 5.2), which is implied by simulation-explainability, see Definition 5.3. For this, we construct $\mathcal{B}$ against the simulation anonymity property. $\mathcal{B}^{\mathcal{O}.\mathcal{RO}^X, \mathcal{O}.\text{ReceiveCred}^X, \mathcal{O}.\text{ReceiveUpdate}^X, \mathcal{O}.\text{PresentCred}^X}(\mathcal{G}, pp_0)$ runs $\text{Exp}_{\Pi_{\text{acs}}}^{\text{unforge}}(\Pi_{\text{acs}}, \mathcal{A}, n, \lambda)$, but uses $\mathcal{G}, pp_0$ from its input. Furthermore, it replaces all calls to $\Pi_{\text{uacs}}.\text{ReceiveCred}(pp, \text{ipk}, \psi^{\text{uacs}}, \text{ctx}^{\text{uacs}}; \alpha)$ with the appropriate call to $\mathcal{B}$'s oracle $\mathcal{O}.\text{ReceiveCred}^X(\text{ipk}, \psi^{\text{uacs}}, \text{ctx}^{\text{uacs}}, \alpha)$. Calls to $\Pi_{\text{uacs}}.\text{PresentCred}(pp, \text{ipk}_{C_b}, \phi^{\text{uacs}}, \text{ctx}^{\text{uacs}}; \overrightarrow{\text{attr}}^{\text{uacs}}, \text{cred}_{C_b}, \alpha)$ are replaced with the appropriate calls to $\mathcal{O}.\text{PresentCred}^X(C, \phi, \text{ctx}, \alpha)$. Finally, random oracle $\mathcal{O}.\mathcal{RO}$ calls by $\mathcal{A}$ are relayed to $\mathcal{B}$'s corresponding oracle.

It is easy to check that for $X = \text{REAL}$, the view of $\mathcal{A}$ and the events $\text{win}_{\text{forgery}}$, $\text{win}_{\text{impersonate}}$ are distributed exactly as in $\text{Exp}_{\Pi_{\text{acs}}}^{\text{unforge}}(\Pi_{\text{acs}}, \mathcal{A}, n, \lambda)$. For $X = \text{SIM}$, the view of $\mathcal{A}$ and the events $\text{win}_{\text{forgery}}$, $\text{win}_{\text{impersonate}}$ are distributed exactly as in G_{expl} . This can be checked by inspection of G_{expl} . For this, observe that all differences between

$\text{Exp}_{\Pi_{\text{acs}}}^{\text{unforge}}(\Pi_{\text{acs}}, \mathcal{A}, n, \lambda)$ and G_{expl} , apart from usage of the simulator $\mathcal{S}$, have no influence over the view of $\mathcal{A}$ or the occurrences of $\text{win}_{\text{forgery}}$, or $\text{win}_{\text{impersonate}}$. This is because, in particular, the explainer perfectly emulates the `IssueCred`, `VerifyCred` protocols, the protocol in `O.HonestIssue` can be skipped without impacting the view of $\mathcal{A}$, and all other changes are just additional checks/bookkeeping that do not impact the rest of the execution.

Because $\mathcal{B}$ is a ppt algorithm (given that $\mathcal{A}$ is a ppt algorithm), we get that

$$\begin{aligned} &= \left| \Pr[\text{Exp}_{\Pi_{\text{uacs}}}^{\text{sim-anon}}(\Pi_{\text{uacs}}, \mathcal{S}, \mathcal{B}, \lambda, \text{REAL}) = 1] - \Pr[\text{Exp}_{\Pi_{\text{uacs}}}^{\text{sim-anon}}(\Pi_{\text{uacs}}, \mathcal{S}, \mathcal{B}, \lambda, \text{SIM}) = 1] \right| \\ &= \left| \Pr[\text{Exp}_{\Pi_{\text{acs}}}^{\text{unforge}}(\Pi_{\text{acs}}, \mathcal{A}, n, \lambda) = 1] - \Pr[G_{\text{expl}}(\lambda) = 1] \right| \end{aligned}$$

is negligible.

In the remainder of this proof, we analyze $\Pr[G_{\text{expl}} = 1]$. Specifically, we show that the fail events $\text{fail}_{\text{explain}}$, $\text{fail}_{\text{impersonate}}$ introduced in G_{expl} happen only with negligible probability. We then argue that if those failure events do not happen, then also the winning conditions for $\text{win}_{\text{forgery}}$, $\text{win}_{\text{impersonate}}$ do not occur.

Explainability. We start by showing that $\text{fail}_{\text{explain}}$ (as defined in G_{expl} above) happens only with negligible probability. $\text{fail}_{\text{explain}}$ corresponds to the UACS explainer failing to produce a correct explanation for some protocol run. For this, we construct a reduction algorithm $\mathcal{B}^{\mathcal{S}, \mathcal{RO}, \mathcal{O}.\text{SimReceiveCred}, \dots}(\mathcal{G}, pp_0, ipk^*)$ against $\text{Exp}_{\Pi_{\text{uacs}}}^{\text{sim-expl}}$ (Definition 5.3). $\mathcal{B}$ runs G_{expl} , but uses $\mathcal{G}, pp_0, ipk^*$ from its input instead of generating them, and relays random oracle queries to its $\mathcal{S}, \mathcal{RO}$ oracle. $\text{fail}_{\text{explain}}$ Hence $\mathcal{B}$ does not have access to isk^* or to the simulator state $state_{\mathcal{S}}$ and will use its oracles to execute the portions of the code that require those values. Furthermore, $\mathcal{B}$ uses its oracle $\mathcal{O}.\text{SimReceiveCred}$ to execute $\mathcal{S}.\text{ReceiveCred}(\dots) \leftrightarrow \mathcal{A}$ in $\mathcal{O}.\text{ReceiveCred}$, its oracle $\mathcal{O}.\text{SimPresent}$ to execute $\mathcal{S}.\text{PresentCred}$ in $\mathcal{O}.\text{PresentCred}$, its oracle $\mathcal{O}.\text{IssueCred}$ to execute $\mathcal{E}.\text{IssueCred}$ in $\mathcal{O}.\text{IssueCred}$, and its oracle $\mathcal{O}.\text{VerifyCred}$ to execute $\mathcal{E}.\text{VerifyCred}$ in $\mathcal{O}.\text{VerifyCred}$ in the natural way. By design of G_{expl} (which essentially contains a copy of $\mathcal{T}_{\text{explained}}$ as in $\text{Exp}_{\Pi_{\text{uacs}}}^{\text{sim-expl}}$), whenever $\text{fail}_{\text{explain}}$ occurs in G_{expl} , then also $\text{Exp}_{\Pi_{\text{uacs}}}^{\text{sim-expl}}(\Pi_{\text{uacs}}, \mathcal{B}, \mathcal{S}, \mathcal{E}, n, \lambda)$ raises $\text{fail}_{\text{explain}}$. However, since $\mathcal{B}$ is a ppt, this happens only with negligible probability given simulation explainability of Π_{uacs} . Hence $\Pr[\text{fail}_{\text{explain}}]$ is negligible.

Impersonation. We now show that $\text{fail}_{\text{impersonate}}$ (as defined in G_{expl} above) happens only with negligible probability. $\text{fail}_{\text{impersonate}}$ corresponds to $\mathcal{A}$ using an honest user's pseudonym. Of course, if that happens, we can take usk, nsk from the UACS explainer, giving rise to a reduction to the pseudonym system's impersonation resistance. For this, let $k(\lambda)$ be an upper-bound for the number of users created by $\mathcal{A}$ (i.e., the maximal number of unique U such that $\mathcal{A}$ calls $\mathcal{O}.\text{GenNym}(U)$ in G_{expl}). Because $\mathcal{A}$ is a ppt, k is a polynomial.

We construct a reduction algorithm $\mathcal{B}^{\mathcal{O}.\text{GenNym}}(\mathcal{G}, pp_1, ntd, upk^*)$ against pseudonym impersonation resistance of Π_{nym} (Definition 4.4). $\mathcal{B}$ runs G_{expl} except that it uses

$\mathcal{G}, pp_1, ntd$ from its input instead of generating those values at the start of G_{expl} . $\mathcal{B}$ chooses a random index $i^* \leftarrow \{1, \dots, k\}$ and for the i^* th execution of $(upk_U, usk_U) \leftarrow \text{UserKeyGen}(pp)$, $\mathcal{B}$ instead sets $U^* = U$ and $upk_{U^*} = upk^*$ (where upk^* is from $\mathcal{B}$'s input). $\heartsuit$ $\mathcal{B}$ is hoping that $\mathcal{A}$ will attack upk^* (or, more correctly, $\mathcal{B}$ hopes that when $\text{fail}_{\text{impersonate}}$ happens, it happens for a pseudonym of U^*). In $\text{GenNym}(U^*)$ queries, instead of running $(nym_N, nsk_N) \leftarrow \text{GenNym}(pp, usk^*)$, $\mathcal{B}$ instead queries its oracle $nym_N \leftarrow \mathcal{O}.\text{GenNym}()$. $\heartsuit$ Note that thanks to the UACS simulator $\mathcal{S}$, usk^* is not needed anywhere else in G_{expl} , and neither is nsk_N .

Let usk, nym, nsk be the values in $G_{\text{expl}}.\mathcal{O}.\text{IssueCred}$ or $G_{\text{expl}}.\mathcal{O}.\text{VerifyCred}$ at the point where $\text{fail}_{\text{impersonate}}$ happens for the first time. When $\text{fail}_{\text{impersonate}}$ occurs, $\mathcal{B}$ checks that $\text{InspectNym}(pp, nym, ntd) = upk^*$ (i.e., the pseudonym belongs to our guessed user). If so, we say that event **guess** happens, and $\mathcal{B}$ outputs (usk, nym, nsk) as a solution for pseudonym impersonation against Π_{nym} . Note that in case of **guess**, we have that $\text{VerifyNym}(pp, usk, nym, nsk) = 1$ (from G_{expl} 's assertion about ψ^{uacs} or ϕ^{uacs}). If instead, $\text{InspectNym}(pp, nym, ntd) \neq upk^*$, then $\mathcal{B}$ aborts.

Whenever $\text{fail}_{\text{impersonate}} \wedge \text{guess}$ happens, $\mathcal{B}$ outputs valid usk, nym, nsk such that $\text{InspectNym}(pp, nym, ntd) = upk^*$ and $\text{VerifyNym}(pp, usk, nym, nsk) = 1$. Because $\mathcal{B}$ is a ppt (using Definition 5.3 to bound the runtime of $\mathcal{S}, \mathcal{E}$) and Π_{nym} is resistant to impersonation, we get that $\Pr[\text{fail}_{\text{impersonate}} \wedge \text{guess}]$ must be negligible. Furthermore, until $\mathcal{B}$ aborts, $\mathcal{A}$'s view is distributed exactly as for G_{expl} , and it is independent of $\mathcal{B}$'s random choice of i^* . Hence $\Pr[\text{fail}_{\text{impersonate}} \wedge \text{guess}] = \Pr[\text{fail}_{\text{impersonate}}] \cdot \Pr[\text{guess}] = \Pr[\text{fail}_{\text{impersonate}}]/k(\lambda)$, which means that $\Pr[\text{fail}_{\text{impersonate}}]$ must be negligible (as k is a polynomial and $\Pr[\text{fail}_{\text{impersonate}} \wedge \text{guess}]$ is negligible).

Impersonation win condition. We now show that $\neg \text{fail}_{\text{impersonate}} \wedge \neg \text{fail}_{\text{explain}} \Rightarrow \neg \text{win}_{\text{impersonate}}$. $\heartsuit$ In words: If no error occurs, then $\mathcal{A}$ also does not win via impersonation. Assume for contradiction that $\text{win}_{\text{impersonate}}$ happens during some $G_{\text{expl}}.\mathcal{O}.\text{VerifyCred}$ query, but $\text{fail}_{\text{explain}}$ and $\text{fail}_{\text{impersonate}}$ do not happen. Let $ipk, nym, \phi, ctx, b, usk, nsk, explained$, etc. be the values from the $G_{\text{expl}}.\mathcal{O}.\text{VerifyCred}$ query at the point where $\text{win}_{\text{impersonate}}$ happens. Because of the conditional return statement, the only way for $\text{win}_{\text{impersonate}}$ to happen is if $b = 1 \wedge (ipk, nym, \phi, ctx) \notin \mathcal{T}_{\text{presented}}$. Together with $\neg \text{fail}_{\text{explain}}$, this implies that $explained = 1$ (using that by design of G_{expl} , we get $(ipk, nym, \phi, ctx) \in \mathcal{T}_{\text{presented}} \Leftrightarrow (\text{PresentCred}, ipk, \phi_{nym, \phi}, ctx^{\text{uacs}}) \in \mathcal{T}_{\text{sim}}$). $explained = 1$ means that the $\mathcal{O}.\text{RegExpl}$ code has been executed. Since by assumption, $\neg \text{fail}_{\text{impersonate}}$, we know that $\text{InspectNym}(pp, nym, ntd) \neq upk_U$ for all honest user handles U . In particular, for all honest user pseudonym handles N , we get that $\text{InspectNym}(pp, nym, ntd) \neq \text{InspectNym}(pp, nym_N, ntd) = upk_{U_N}$ (using that honest user pseudonyms trace correctly, cf. Definition 4.1) and, as a consequence, $nym \neq nym_N$. This negates the necessary condition for $\text{win}_{\text{impersonate}}$, completing the proof by contradiction.

Forgery win condition. We now show that $\neg \text{fail}_{\text{impersonate}} \wedge \neg \text{fail}_{\text{explain}} \Rightarrow \neg \text{win}_{\text{forgery}}$. $\heartsuit$ In words: If no error occurs, then $\mathcal{A}$ also does not win via forgery. For this, assume for

contradiction that $\neg \text{fail}_{\text{impersonate}} \wedge \neg \text{fail}_{\text{explain}} \wedge \text{win}_{\text{forgery}}$. First, note that as an invariant, if $(usk, \overrightarrow{attr}) \in \mathcal{T}_{\text{explained}}$ and $\neg \text{fail}_{\text{impersonate}}$, then $\exists nym, nsk : (nym, \overrightarrow{attr}) \in \mathcal{T}_{\text{issued}} \wedge \text{VerifyNym}(pp, usk, nym, nsk) = 1$ (this can be seen by inspection of $G_{\text{expl}}.\mathcal{O}.\text{IssueCred}$, using that honest users' pseudonyms trace to their upk by correctness of Π_{nym}).

Now consider a $G_{\text{expl}}.\mathcal{O}.\text{VerifyCred}(ipk^*, nym, \phi, ctx)$ query in which $\text{win}_{\text{forgery}}$ happens. In the following, we use the variable values from the moment where $\text{win}_{\text{forgery}}$ occurs. Because of the conditional return statement in $\mathcal{O}.\text{VerifyCred}$, the only way for $\text{win}_{\text{forgery}}$ to happen is if $b = 1 \wedge (ipk^*, nym, \phi, ctx) \notin \mathcal{T}_{\text{presented}}$. Together with $\neg \text{fail}_{\text{explain}}$, this implies that $\text{explained} = 1$. In turn, this means that the explainer found $usk, \overrightarrow{attr}, nsk$ with $\text{VerifyNym}(pp, usk, nym, nsk) = 1$ and $\phi(\overrightarrow{attr}) = 1$, and added $(usk, \overrightarrow{attr})$ to $\mathcal{T}_{\text{explained}}$. By our invariant above, we know that $(usk, \overrightarrow{attr}) \in \mathcal{T}_{\text{explained}}$ implies that there are nym', nsk' with $(nym', \overrightarrow{attr}) \in \mathcal{T}_{\text{issued}}$ and $\text{VerifyNym}(pp, usk, nym', nsk') = 1$. Because Π_{nym} has binding pseudonyms (Definition 4.3) and $\text{VerifyNym}(pp, usk, nym, nsk') = 1$ and $\text{VerifyNym}(pp, usk, nym', nsk') = 1$, we immediately get $\text{InspectNym}(pp, nym, ntd) = \text{InspectNym}(pp, nym', ntd)$. Overall, we have found $(nym', \overrightarrow{attr}) \in \mathcal{T}_{\text{issued}}$ with $\phi(\overrightarrow{attr}) = 1$ and $\text{InspectNym}(pp, nym, ntd) = \text{InspectNym}(pp, nym', ntd)$, negating the condition for $\text{win}_{\text{forgery}}$ to occur, concluding the proof by contradiction.

Summary. Overall, we have shown that $\neg \text{fail}_{\text{impersonate}} \wedge \neg \text{fail}_{\text{explain}} \Rightarrow \neg \text{win}_{\text{impersonate}} \wedge \neg \text{win}_{\text{forgery}}$. As a consequence,

$$\Pr[\text{win}_{\text{impersonation}} \vee \text{win}_{\text{forgery}}]$$

💡 Using implication $\text{win}_{\text{impersonate}} \vee \text{win}_{\text{forgery}} \Rightarrow \text{fail}_{\text{impersonate}} \vee \text{fail}_{\text{explain}}$ from above.

$$\leq \Pr[\text{fail}_{\text{impersonate}} \vee \text{fail}_{\text{explain}}]$$

💡 Union bound

$$\leq \Pr[\text{fail}_{\text{impersonate}}] + \Pr[\text{fail}_{\text{explain}}]$$

We have shown above that the two summands $\Pr[\text{fail}_{\text{impersonate}}]$ and $\Pr[\text{fail}_{\text{explain}}]$ are negligible, hence $\Pr[\text{win}_{\text{impersonation}} \vee \text{win}_{\text{forgery}}] = \Pr[G_{\text{expl}} = 1]$ is negligible. Finally, because $\left| \Pr[\text{Exp}_{\Pi_{\text{acs}}}^{\text{unforge}}(\Pi_{\text{acs}}, \mathcal{A}, n, \lambda) = 1] - \Pr[G_{\text{expl}}(\lambda) = 1] \right|$ is negligible, this means that $\Pr[\text{Exp}_{\Pi_{\text{acs}}}^{\text{unforge}}(\Pi_{\text{acs}}, \mathcal{A}, n, \lambda) = 1]$ must be negligible, too. $\square$

Overall, note that the proof in large parts deals with the logic for the traditional unforgeability game. The actual UACS proof parts are quite simple to use and make it convenient to argue about, in this case, the hidden usk attribute and its properties. In particular, this construction is much easier to prove secure given the UACS route. If we were to construct a traditional anonymous credential system (using the standard template from Section 4.3), then this would involve using the zero-knowledge property and the blindness of the blind signature scheme, the proof of knowledge property, and the blind signature unforgeability (on top of the pseudonym system properties). In the UACS instantiation, the roles of the proof system and blind signature scheme are unified into the convenient simulation-explainability property.

Chapter 6

Delegatable updatable anonymous credentials

You can delegate authority, but you cannot delegate responsibility.

Byron L. Dorgan

In this chapter, we consider *delegation* of anonymous credentials. Broadly speaking, delegation refers to the process where one person holds a right and grants this right to another person. For example, someone with the right to access a building might want to *delegate* that right to someone else, meaning to give them access to the building, too.

In a delegatable credential system, a user holding a credential is able to issue a weaker version of his credential to another user. For example, a user holding a credential that grants him access to buildings *A* and *B* should be able to issue a credential granting access to building *A* to another user (without also granting access to *B*).

In our previous UACS system, we can approximate delegation by involving the issuer. Alice would inform the issuer what attributes she wants to delegate to Bob. Alice would then prove to the issuer that she has permission to delegate these attributes (presumably, this involves showing her own credential, whose attributes encode her delegation rights). Finally, the issuer runs the issuance process with Bob for the specified attributes.

One can reasonably extend this sketch such that Alice's and Bob's attributes remain hidden from the issuer: Alice would send to the issuer *a commitment* to Bob's desired attributes, and show that her credential's attributes allow her to delegate the committed attributes. Alice would give Bob the commitment and opening, and Bob would request a credential on the committed attributes. Thanks to the powerful UACS modeling, this extension is easily implemented.

The downside of this naive approach is that the issuer needs to be involved in every issuance. This is undesirable because it represents additional constraints on the infrastructure (the issuer must be available for potentially frequent delegation tasks, not just for new issuances). Furthermore, because the issuer's key is the most valuable artifact in an anonymous credential system and needs to be guarded carefully (e.g., through expensive hardware), we ideally want to involve the issuer in as few actions as possible.

The goal of this chapter is to design *delegatable anonymous credential systems* (DACS), where Alice is able to delegate credentials to Bob *without the involvement of the issuer*.

Modeling hierarchies of attributes. Delegating *all* of Alice’s rights to Bob is trivial: Alice can just give her credential, as-is, to Bob. Of course, delegating all of your rights is almost always undesirable and not particularly useful in real-world scenarios. Instead, Alice should be able to delegate “less than” her rights, and perhaps impose further usage restrictions on the delegated credential.

To rigorously model this intuition in terms of attribute-based anonymous credentials, we need to answer the following question for any given $\overrightarrow{attr}_{\text{Alice}}, \overrightarrow{attr}_{\text{Bob}}$:

Alice has attributes $\overrightarrow{attr}_{\text{Alice}}$ and wants to delegate attributes $\overrightarrow{attr}_{\text{Bob}}$ to Bob.
Is she allowed to?

There are many useful ways this question can be answered. For the purposes of this thesis, we answer the question by means of a partial order $\succeq$ over attributes. Alice with attributes $\overrightarrow{attr}_{\text{Alice}}$ is allowed to delegate $\overrightarrow{attr}_{\text{Bob}}$ if and only if $\overrightarrow{attr}_{\text{Alice}} \succeq \overrightarrow{attr}_{\text{Bob}}$. Towards defining $\succeq$, we first introduce a new modeling artifact for DACS. In our prior UACS model, a credential authenticates a vector $\overrightarrow{attr} \in \mathbb{A}^n$ of attributes. For DACS, we introduce a new special attribute “ $\star$ ”. A DACS credential authenticates a vector $\overrightarrow{attr} \in (\mathbb{A} \cup \{\star\})^n$. Intuitively, $\star$ is a placeholder attribute that can be delegated as *any* $attr \in \mathbb{A} \cup \{\star\}$. On the flip-side, non-placeholder attributes $attr \in \mathbb{A}$ are fixed and cannot be changed during delegation.

Definition 6.1 (Delegation relation). Let $n \in \mathbb{N}$ and let $\mathbb{A}$ be some set with $\star \notin \mathbb{A}$. We say that $\overrightarrow{attr} = (attr_1, \dots, attr_n) \in (\mathbb{A} \cup \{\star\})^n$ *covers* $\overrightarrow{attr}' = (attr'_1, \dots, attr'_n) \in (\mathbb{A} \cup \{\star\})^n$, if

$$\forall i \in \{1, \dots, n\} : (attr_i \neq \star \Rightarrow attr'_i = attr_i)$$

In that case, we write

$$\overrightarrow{attr} \succeq \overrightarrow{attr}'$$

(or $\overrightarrow{attr}' \preceq \overrightarrow{attr}$). For a set $S \subseteq (\mathbb{A} \cup \{\star\})^n$, we write $\overrightarrow{attr} \preceq S$ if $\overrightarrow{attr}$ is covered by some attribute vector in S , i.e., if $\exists \overrightarrow{attr}' \in S : \overrightarrow{attr} \preceq \overrightarrow{attr}'$.

To help understand this definition, consider the following examples.

- $\star^n = (\star, \star, \dots, \star)$ is the most powerful attribute vector: It covers *all* attribute vectors, i.e., $\star^n \succeq \overrightarrow{attr}$ for all $\overrightarrow{attr} \in (\mathbb{A} \cup \{\star\})^n$.
- Concrete vectors $\overrightarrow{attr} \in \mathbb{A}^n$ (with no $\star$) are the weakest attribute vectors: They only cover themselves, i.e., $\overrightarrow{attr} \succeq \overrightarrow{attr}' \Leftrightarrow \overrightarrow{attr} = \overrightarrow{attr}'$.
- $(A, \star, B)$ covers any vector of the form (A, X, B) for $X \in \mathbb{A} \cup \{\star\}$. In contrast, $(A, \star, B)$ does *not* cover (B, B, B) since the first attribute is fixed to A and cannot be changed.

The $\succeq$ relation is a reflexive partial order over attribute vectors. Another instructive example for such a relation is the superset relation $\supseteq$ over sets.

- The relation is reflexive, i.e., any attribute vector $\overrightarrow{attr} \in (\mathbb{A} \cup \{\star\})^n$ covers itself. This means that Alice is allowed to delegate her attributes $\overrightarrow{attr}_{\text{Alice}}$ unchanged to Bob. As discussed above, this is an inherent ability because Alice can always give her whole credential, as-is, to Bob.
- The relation is transitive, i.e., if $\overrightarrow{attr}_1 \succeq \overrightarrow{attr}_2$ and $\overrightarrow{attr}_2 \succeq \overrightarrow{attr}_3$, then $\overrightarrow{attr}_1 \succeq \overrightarrow{attr}_3$. This means that if Alice delegates to Bob, then whatever Bob can delegate to Charlie could also have been delegated by Alice (i.e., Bob's credential cannot be stronger than Alice's credential).
- The relation is antisymmetric, i.e., for $\overrightarrow{attr}_1 \neq \overrightarrow{attr}_2$, it's impossible for $\overrightarrow{attr}_1$ to cover $\overrightarrow{attr}_2$ and $\overrightarrow{attr}_2$ to cover $\overrightarrow{attr}_1$ at the same time. This means that if Alice delegates attributes $\overrightarrow{attr}_{\text{Bob}} \neq \overrightarrow{attr}_{\text{Alice}}$ to Bob, then these are *strictly* weaker: Bob's attributes do not cover Alice's attributes. Due to transitivity, this property extends to any attributes delegated by Bob.

Application examples. To convince ourselves that Definition 6.1 is a useful choice for our delegation relation, we explore some application examples enabled by this modeling.

First, we note that our model covers the notion of users holding a *set* $S \subseteq [n]$ of permissions and being able to choose to delegate a *subset* of these permissions. As a concrete example, consider a building access control system managing access to n buildings. For this scenario, we imagine the user receives a credential on attributes $\overrightarrow{attr} = (B_1, \dots, B_n)$ where we set the attribute

$$B_i = \begin{cases} \star & \text{if the user has access to building } i \\ 0 & \text{otherwise} \end{cases}$$

For example, Alice may hold a credential on $\overrightarrow{attr}_{\text{Alice}} = (0, \star, \star)$, giving her access to buildings 2 and 3, but not to building 1. This enables her to delegate a credential with attributes $\overrightarrow{attr}_{\text{Bob}} = (0, \star, 0)$ to Bob, giving him access only to building 2. Note that Alice cannot give Bob access to building 1.

However, Definition 6.1 goes beyond delegation of subsets. For example, we may allow Alice to embed an expiration date into the delegated credential. Keeping to the example above, we may set $\overrightarrow{attr} = (B_1, \dots, B_n, \text{expire})$. Alice would hold a credential for $(0, \star, \star, \star)$ and can give Bob access to (only) building 2 until a certain *date* by delegating $\overrightarrow{attr} = (0, \star, 0, \text{date})$ to Bob. In this setting, the verifier checking access for building 2 would check that $\phi(\overrightarrow{attr}) \equiv (B_2 = \star \wedge \text{expire} > \text{now})$, where now is the current timestamp hardcoded as a constant into ϕ . Note that Alice's attributes $(0, \star, \star, \star)$ do not, strictly speaking, fulfill the policy because $\star \not> \text{now}$ (we do not consider $\star$ a number). However, to access the building, Alice can simply delegate a credential with some arbitrary future **expire** date to herself.

While the example above works well, a potential downside is that Alice can encode completely arbitrary expiration dates into Bob's credential. If Alice's credential holds an expiration date itself, a natural requirement may be that Alice can only delegate access

up to the point where her own credential expires. Essentially, we want someone with attribute **expire** to only be able to delegate credentials with $\mathbf{expire}' \leq \mathbf{expire}$. Perhaps surprisingly, such a requirement can be modeled with Definition 6.1 as well. We encode the expiration date attribute bit-wise, i.e., we consider attributes $\mathbf{expire}_\ell, \dots, \mathbf{expire}_0$ where $\mathbf{expire}_i$ represents the i th bit of the expiration date. The verifier would check $\phi(\dots, \mathbf{expire}_\ell, \dots, \mathbf{expire}_0) \equiv (\dots \wedge \sum_{i=0}^{\ell} \mathbf{expire}_i \cdot 2^i > \text{now} \wedge \bigwedge_{i=0}^{\ell} \mathbf{expire}_i \in \{0, 1\})$. This makes it easy to imagine how we can restrict Alice: Given a credential with, say, expiration date attributes $(\mathbf{expire}_4, \dots, \mathbf{expire}_0) = (0, 0, \star, \star, \star)$, Alice can only delegate expiration dates up to the numeric value 7, i.e., $(0, 0, 1, 1, 1)$. In general, given any binary number, one can produce any *smaller* binary number by flipping some 1 bit $\mathbf{expire}_k = 1$ to 0 and setting the less significant bits arbitrarily. We can enable Alice to execute this procedure by issuing her multiple credentials, one for every 1 bit in her expiration date, i.e., for every k with $\mathbf{expire}_k = 1$, we would give her credentials with attributes

$$\mathbf{expire}_i^{(k)} = \begin{cases} \star & i < k \\ 0 & i = k \\ \mathbf{expire}_i & i > k \end{cases}.$$

For example, let $\ell = 4$ (i.e., the maximum timestamp is $2^4 - 1 = 31$). To encode the expiration timestamp 13, we give Alice credentials with the following attribute vectors:

- $\mathbf{expire} = (0, 1, 1, 0, 1)$ //The timestamp 13 itself
- $\mathbf{expire}^{(0)} = (0, 1, 1, 0, 0)$ // $k = 0$ bit flipped. Enables deriving 12
- $\mathbf{expire}^{(2)} = (0, 1, 0, \star, \star)$ // $k = 2$ bit flipped. Enables deriving 8 to 11
- $\mathbf{expire}^{(3)} = (0, 0, \star, \star, \star)$ // $k = 3$ bit flipped. Enables deriving 0 to 7

This enables her to derive any expiration timestamp up to 13, and prevents her from deriving any timestamps larger than 13. Notably, she can also issue multiple credentials to Bob with a similar encoding to give him the ability to further delegate with even smaller timestamps (e.g., enable Bob to delegate credentials with timestamps of at most 8).

Taking a slightly different perspective, we highlight an application example that showcases how delegatable credential can serve as an enhancement for privacy. Consider the issuance of driver's licenses. We can imagine that every city is able to issue driver's licenses to their citizens. Since anonymous credentials do not hide the issuer of a credential from verifiers, this poses a privacy issue: Any verifier would inevitably need to learn the credential's issuer, which is the city the license's holder lives in. To solve this issue, we can use delegatable credentials in a hierarchical fashion as follows. The state sets up an issuer, meaning that credentials will verify under the state's key. The attribute format is $\overrightarrow{attr} = (\mathbf{county}, \mathbf{city}, \mathbf{name})$. The state delegates a credential with $(\text{Los Angeles}, \star, \star)$ to Los Angeles, which then delegates a credential to Santa Monica with attributes $(\text{Los Angeles}, \text{Santa Monica}, \star)$. This enables Santa Monica to issue credentials to its citizens (with arbitrary **name**). Because credentials verify under the state's key, users can now hide information about their county or city if desired.

Modeling delegatable credential systems. The main insight driving our modeling of DACS is to view delegation as *issuance where the issuer's key is a credential*. As a result, we will essentially stick to the UACS model (Chapter 5), but use credentials in place of the issuer key. This has the advantage that DACS inherits all the modeling power and simplicity of UACS, such as the ability to model hidden attributes and pseudonyms as explained in Chapter 5. In the following, we use the terminology “issue” and “delegate” as synonyms, given that in our formalization, they refer to the same process.

More concretely, a DACS credential authenticates attributes $\overrightarrow{attr} \in (\mathbb{A} \cup \{\star\})^n$. The issuer generates an issuer public key ipk together with a root credential with attributes $\overrightarrow{attr} = \star^n = (\star, \dots, \star)$, which replaces the issuer's secret key. As observed when discussing our delegation relation (Definition 6.1), such a credential is able to delegate arbitrary attribute vectors $\overrightarrow{attr} \in (\mathbb{A} \cup \{\star\})^n$ and hence serves as a useful issuer key.

An overview of the processes in DACS is depicted in Figure 6.1. Issuing credentials works as in UACS, with an issuer/delegator and receiver agreeing on an update function $\psi : \{\perp\} \times \{0, 1\}^* \rightarrow (\mathbb{A} \cup \{\star\})^n \cup \{\perp\}$. The receiver will get a credential with attributes $\psi(\perp, \alpha)$ for a value $\alpha \in \{0, 1\}^*$ of his choice. Like in UACS, this enables blind issuance: The issuer/delegator does not necessarily need to know the exact values of the attributes she is issuing. The main difference between UACS and DACS is that in DACS, the issuer/delegator inputs her credential instead of an issuer's secret key. This enables anyone with a credential to issue credentials, subject to attribute restrictions imposed by the delegation relation (Definition 6.1). So the issuer with her root credential with attributes $(\star, \dots, \star)$ can issue any attributes $\overrightarrow{attr}_1 = \psi_1(\perp, \alpha_1)$ to a user. This enables the user to then issue a credential with attributes $\overrightarrow{attr}_2 = \psi_2(\perp, \alpha_2)$ to another user as long as his credential is powerful enough, i.e., $\overrightarrow{attr}_1 \succeq \overrightarrow{attr}_2$.

In addition to this, users can also make their delegation contingent on the receiver having some prior credential, i.e., *update* another user's existing credentials. A user with attributes $\overrightarrow{attr}_3$ can update his attributes to $\overrightarrow{attr}_4 = \psi(\overrightarrow{attr}_3, \alpha_4)$ with the help of another user with attributes $\overrightarrow{attr}_2 \succeq \overrightarrow{attr}_4$ (see Figure 6.1).

Relation to non-transferability. At first glance, the prospect of delegation seems antithetical to previously stated goals. In Section 4.4, we identified the goal of non-transferability: Users should be *disincentivized* from sharing their access rights with someone else. With delegatable credentials, users being able to share their access rights is essentially the goal. Does this mean that delegation and non-transferability are contradictory goals?

In many ways, they *are* indeed contradictory goals. However, non-transferability does not need to apply universally: For *some* parties, it may be perfectly fine to transfer credentials. For *others*, we may want to discourage transfer. With DACS, one can implement fine-grained mechanisms governing what transfer is allowed, and who bears responsibility for transferred/delegated credentials.

For example, we can implement the non-transferability/accountability mechanism from Section 5.4, where we embed the credential holder's secret key usk as an attribute $\mathbf{usk}$ into every credential. Verifiers would make authenticating users reveal a (decryptable)

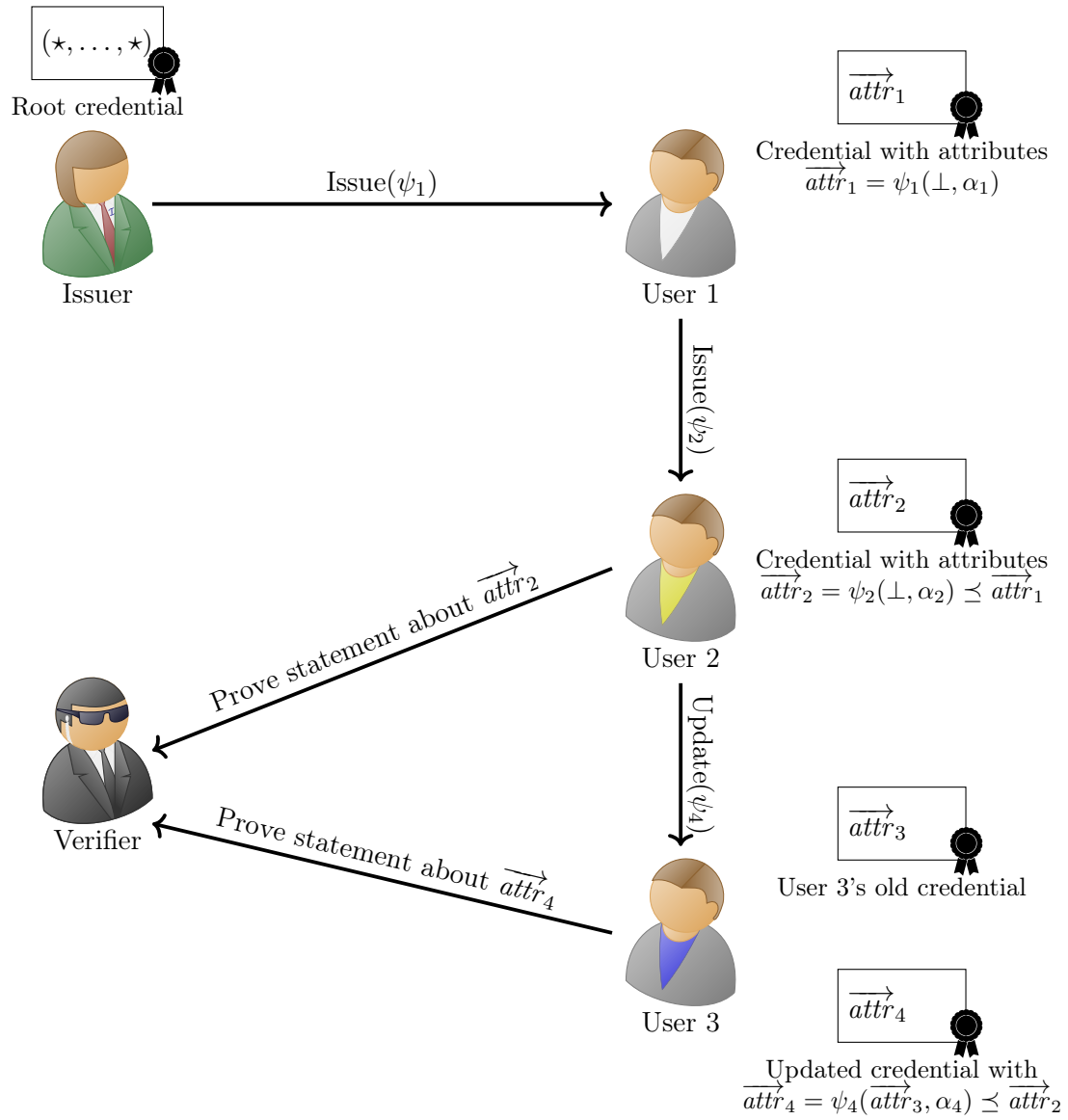


Figure 6.1: Overview of the protocols in a delegatable credential system.

pseudonym (Section 4.1) bound to **usk**. In DACS, this leaves us with essentially two choices for setting the **usk** attribute.

- **usk** = $usk \in \mathbb{A}$ binds the credential (and all derived/delegated credentials) to the given user’s secret usk . As a consequence, whenever this credential is presented, we can hold the user behind usk accountable.
- **usk** = $\star$ makes the credential delegatable to arbitrary users. Accountability is on the receiver. The downside is that holders of this credential can easily avoid accountability by delegating the credential to themselves, setting **usk** to some random value not associated with any real-world person.

In a setting where a city issues driver’s licenses to users, we would trust the city with **usk** = $\star$. The city avoiding accountability is not a concern. The city would make sure to only issue credentials to valid **usk** as described in Section 4.4.

We can further increase accountability on the city. Consider attributes in the format $(\text{county}, \text{city}, \text{name}, \text{usk})$. As before, when presenting the credential, the user would reveal a pseudonym bound to **usk**. However, in addition, we can make the user reveal an encryption of **city** as well (e.g., an encryption of **city** = Santa Monica). When we later want to hold the user accountable, but notice that the user’s pseudonym does not decrypt/trace to any valid user, then we can instead keep the city accountable. This works as long as we trust anyone with a credential $(\cdot, \text{city} = \star, \cdot, \cdot)$ (such as the county) not to falsely blame the city.

Contribution. In this chapter, we introduce our notion of delegatable anonymous credentials. We define its syntax and security formally in Section 6.2. In Section 6.3, we define and construct the main building block for our DACS construction: dynamically malleable signatures (with efficient protocols). Finally, we construct DACS from DMS and (zero-knowledge) proof systems in Section 6.4.

This chapter is based on the paper [BB18], for which Jan Bobolz contributed the idea, definitions, construction, and proofs. This chapter revisits the formalization from the original paper [BB18], building on our definitions in Chapter 5. In particular, this means that the DACS in this thesis supports general blind issuance (where the only blind attribute in the original paper [BB18] is the user secret usk). Furthermore, we treat $\star$ as a proper attribute: Presentation policies in this thesis can check whether an attribute is $\star$ and users can blindly receive $\star$ as an attribute (if the issuance/update policy allows). This results in a more natural and powerful model, supporting additional use cases. Additionally, we put significant effort into keeping the construction very efficient despite its more powerful model, as described in Section 6.4.4. We discuss details in the following sections.

6.1 Related work

In this section, we discuss other types of constructions of delegatable credentials.

The first construction: recursive proofs. Early work on delegatable credentials follows a chain-of-certificates paradigm: A level L credential is created by extending a level $L - 1$ credential with another signature (or other means of authentication). Attributes were not generally considered. Instead, the main relevant property of a credential is the *length* of the delegation chain. A level L credential is considered more powerful/trustworthy than a level $L + 1$ credential.

The first construction of delegatable credentials [CL06] uses a recursive proof strategy: A level L credential for the user with public key pk_L is a non-interactive zero-knowledge proof (“signature”) of knowledge authenticating $ctx = pk_L$ while proving knowledge of (1) a level $L - 1$ credential for pk_{L-1} and (2) the secret key of pk_{L-1} . Here, (2) ensures that only the holder of the level $L - 1$ credential can create the level L credential, and (1) ensures that level $L - 1$ credential holder is in possession of a valid credential.

The construction was not considered practically efficient due to the inefficiency of non-interactive proof systems. Because at the time, a proof was generally larger than its witness by some factor > 1 , a [CL06] credential has exponential size in L . Nowadays, we have many practically efficient *succinct* proof systems, making credentials linear size or even constant size and providing practical feasibility (though concretely, the resulting scheme is significantly less computationally efficient than other solutions).

Notably, using succinct general-purpose proof systems, one is afforded significant flexibility. For example, it is easy to embed (committed) attributes at every level, and have the holder of the level L credential prove that the attributes at level $L - 1$ are sufficiently powerful to issue the attributes at level L . Here, “sufficiently powerful” can be according to Definition 6.1, but also any other polynomial-time computable relation.

We conjecture that this represents an alternative way of implementing our DACS. However, the flexibility comes with significantly increased concrete computational cost compared to our solution (Construction 6.16) based on dynamically malleable signatures, as well as requiring stronger complexity assumptions.

More efficient constructions: malleable proofs. Improving over the low performance of the recursive proof strategy, follow-up work focused on *malleable* proofs instead [BCKLS09; Fuc11; CKLM13; CKLM14]. A level L credential for the user with public key/pseudonym pk_L is a non-interactive zero-knowledge proof of knowledge of a list of signatures σ_i under public key pk_i on the message pk_{i+1} (for $i < L$). So while a [CL06] credential is a (recursive) chain of proofs, a [BCKLS09] credential is a proof of a chain of signatures. As a consequence, the credential size is linear in the chain length L (assuming the proof size is linear in the witness).

This approach comes with two challenges: presenting the credential in an unlinkable manner and extending the level L credential to include the additional witnesses and statements needed for the level $L + 1$ credential. Both these challenges are solved using malleable proofs, namely Groth-Sahai proofs, which can be rerandomized to look like any fresh proof and for which witnesses and statements can be extended suitably.

This approach is technically elegant and significantly improves performance over the recursive proof approach. However, Groth-Sahai proofs are inherently concretely large

and somewhat difficult to implement, diminishing the practicality of the approach. We are not aware of other instantiations for the malleable proof system. Public attributes are supported, but all attributes for all levels must be revealed upon presentation.

Practically efficient constructions: mercurial signatures. Improving over the low concrete performance of the malleable proof strategy, follow-up work devised a scheme based on *mercurial signatures* [CL19; CL21]. A level L credential for a user with pseudonym pk_L is a list of signatures σ_i under public key pk_i on the message pk_{i+1} (for $i < L$). So while a [CL06] credential is a (recursive) chain of proofs and a [BC-CKLS09] credential is a proof of a chain of signatures, a [CL19] credential is a chain of signatures.

Mercurial signatures have the property that they are malleable in both the verification key and the signed message. A signature can be adapted to any verification key (within an equivalence class of keys) and any message (within an equivalence class of messages). This allows a credential to be fully re-randomized into an unlinkable copy, enabling privacy-preserving delegation and presentation.

The resulting construction is very practically efficient. One privacy downside of the construction is that issuers can recognize their own pseudonyms in delegation chains. This was later fixed [GLMPS24].

Sacrificing delegator privacy. The constructions above put great effort into keeping the identities of delegators secret from credential holders. In many settings, we can tolerate credential holders knowing the identities on the delegation chain. In that case, the construction of delegatable credentials becomes much easier [CDD17]. A credential is simply a list of signatures σ_i under public key pk_i on the message pk_{i+1} (for $i < L$) and attributes. In contrast to [CL19] above, the signature scheme is not malleable and the credential holder learns all pk_i (in unrandomized form) and attributes. To present a credential, the user simply proves knowledge of the list of credentials. This construction is concretely efficient and because it generally follows the standard template for anonymous credentials (unlike the constructions discussed above), it is easy to use and extend.

Constructions with advanced treatment of attributes. With the exception of [CDD17], the constructions mentioned above only feature limited support for attributes: All attributes in the chain must be revealed during credential presentation. Following our paper [BB18] on attribute-based delegatable credentials, which this chapter is based on, additional works have incorporated attributes more meaningfully.

The authors of [PM23] extend constructions based on mercurial signatures [CL19; CL21] with attributes that can be partially delegated and partially disclosed. Crucially, they enforce restrictions on delegation: A user with attribute set S can only delegate subsets $S' \subset S$ of their existing attributes. To achieve this, [PM23] includes a commitment to the set of attributes (similar to [FHS19]) in the message authenticated by the mercurial signature. Additionally, they include carefully designed witnesses to prove that the committed set of attributes at level i is a subset of the committed attributes at level

$i - 1$. Commitments and witnesses can be suitably randomized (similar to pseudonyms in [CL19]) to provide strong privacy.

Around the same time, [MSBM23] designs delegatable credentials using similar techniques to our DMS-based construction, but applied to the SPS-EQ scheme [FHS19] instead of Pointcheval-Sanders signatures [PS16]. In more detail, [MSBM23] defines and constructs SPS-UC, which is an SPS-EQ made message-malleable by including additional group elements in the signature. Like our construction (Construction 6.11), throwing away the elements that allow for malleability and rerandomizing the remaining signature results in the desired weaker (delegated) version of the signature. Unlike our construction, the vector that is authenticated by SPS-UC is a vector of *sets* of messages (whereas we authenticate just a vector of individual messages). This enables fast selective disclosure of attributes without zero-knowledge proofs, as in [FHS19]. The framing of [MSBM23] delegatable credentials is quite different from ours: They imagine that at every delegation step, a new set of attributes (pertaining to the receiver of the credential) is attached to the credential. In this framing, there is no relation between attributes of different users in the delegation chain and no restriction of what attributes can be issued. A subset relation can be enforced by applying [PM23]. However, by changing the framing and using the capabilities of SPS-UC in a different way (e.g., closer to how we use our DMS in our construction), we expect other delegation restrictions (such as Definition 6.1 but also others) can be implemented. For this, a particularly interesting ability of their construction is the ability to remove (sets of) attributes from the delegated credential by simply refusing to pass the set commitment opening to the credential receiver.

Overall, these approaches are exciting and different use cases call for different constructions. A strength of our solution is that it is close to the standard template and hence easily extendable (for example, different types of pseudonyms can be implemented easily as in Construction 5.14). Approaches based on equivalence class signatures tend to be more efficient for pure partial disclosure of attributes. Approaches encoding the delegation chain (which we do not do) can have advantages, such as the ability to revoke credentials delegated by a certain user.

6.2 Defining delegatable updatable anonymous credentials

In this section, we define delegatable (updatable) anonymous credentials schemes. We define their syntax in Section 6.2.1 and their security in Sections 6.4.2 and 6.4.3.

6.2.1 Syntax

As discussed above, we model delegatable anonymous credential systems (DACs) as an extension to updatable credential systems (UACS), where anyone holding a credential cred_{iss} on attributes $\overrightarrow{\text{attr}}_{\text{iss}}$ can play the role of an issuer for attributes $\overrightarrow{\text{attr}}' \preceq \overrightarrow{\text{attr}}_{\text{iss}}$, using cred_{iss} instead of the issuer's key. As a consequence, the definition below is very much reminiscent of the syntax definition for UACS (Definition 5.1) except that (1) credential attributes $\overrightarrow{\text{attr}}$ may now contain $\star$ in addition to values from $\mathbb{A}$, (2) the issuer's credential

$\overrightarrow{cred}_{iss}$ takes the place of the issuance key isk , and (3) the issuer's (credential's) attributes $\overrightarrow{attr}_{iss}$ are input to both issuer and receiver during issuance and updates. $\overrightarrow{attr}_{iss}$ helps issuer and receiver understand the limits of what attributes $\overrightarrow{attr}'$ an issuance/update can possibly result in (namely $\overrightarrow{attr}' \preceq \overrightarrow{attr}_{iss}$, anything else would be considered a forgery). Correctness is restricted to cases where those restrictions are honored.

Definition 6.2 (Delegatable UACS). A delegatable updatable anonymous credential system Π_{dacs} with attribute space $\mathbb{A} = \mathbb{A}_G$, update function spaces $\Psi_G[\overrightarrow{attr}_{iss}] \subset \{\psi : ((\mathbb{A} \cup \{\star\})^n \cup \{\perp\}) \times \{0, 1\}^* \rightarrow \{\overrightarrow{attr}' \mid \overrightarrow{attr}' \preceq \overrightarrow{attr}_{iss}\} \cup \{\perp\}\}$ (for every $\overrightarrow{attr}_{iss} \in (\mathbb{A} \cup \{\star\})^n$, $n \in \mathbb{N}$), and predicate spaces $\Phi = \Phi_G \subset \{\phi : (\mathbb{A} \cup \{\star\})^n \times \{0, 1\}^* \rightarrow \{0, 1\} \mid n \in \mathbb{N}\}$, consists of the following ppt algorithms and protocols.

Setup($\mathcal{G}$) $\rightarrow pp$

Generates public parameters pp .

IssuerKeyGen($pp, 1^n$) $\rightarrow (ipk, cred^*)$

Generates issuer public key ipk and a corresponding root credential $cred^*$. $\mathcal{V} cred^*$ is a credential on $\star^n$ and can hence be used to derive credentials on arbitrary attribute vectors $\overrightarrow{attr} \in (\mathbb{A} \cup \{\star\})^n$. It replaces the issuer's secret key isk (we can imagine $isk = cred^*$).

$cred' \leftarrow \text{ReceiveCred}(pp, ipk, \overrightarrow{attr}_{iss}, \psi, ctx; \alpha)$
 $\leftrightarrow \text{IssueCred}(pp, ipk, \overrightarrow{attr}_{iss}, \psi, ctx; cred_{iss}) \rightarrow b$

An interactive protocol. $\mathcal{V}$ Run by the user to receive a credential $cred'$ with attributes $\overrightarrow{attr}' = \psi(\perp, \alpha) \preceq \overrightarrow{attr}_{iss}$. The issuer uses a credential $cred_{iss}$ on $\overrightarrow{attr}_{iss}$ in order to issue $cred'$. The issuer outputs a bit b indicating $\psi(\perp, \alpha) \neq \perp$.

$cred' \leftarrow \text{ReceiveUpdate}(pp, ipk, \overrightarrow{attr}_{iss}, \psi, ctx; \overrightarrow{attr}, cred, \alpha)$
 $\leftrightarrow \text{UpdateCred}(pp, ipk, \overrightarrow{attr}_{iss}, \psi, ctx; cred_{iss}) \rightarrow b$

An interactive protocol. $\mathcal{V}$ Run by the user to receive an updated credential $cred'$ with attributes $\overrightarrow{attr}' = \psi(\overrightarrow{attr}, \alpha) \preceq \overrightarrow{attr}_{iss}$ based on his existing credential $cred$ with attributes $\overrightarrow{attr}$. The issuer uses credential $cred_{iss}$ with attributes $\overrightarrow{attr}_{iss}$ to issue the updated credential $cred'$. The issuer outputs a bit b indicating $\psi(\overrightarrow{attr}, \alpha) \neq \perp$.

PresentCred($pp, ipk, \phi, ctx; \overrightarrow{attr}, cred, \alpha$) $\leftrightarrow$ VerifyCred(pp, ipk, ϕ, ctx) $\rightarrow b$

An interactive protocol. $\mathcal{V}$ Run by the user to convince the verifier of possession of a credential. The verifier outputs a bit b indicating $\phi(\overrightarrow{attr}, \alpha) = 1$.

CheckCred($pp, ipk, \overrightarrow{attr}, cred$) $= b$

A deterministic algorithm that outputs a bit b . $\mathcal{V}$ It checks that $cred$ is a valid credential with respect to issuer ipk with attributes $\overrightarrow{attr}$.

A delegatable updatable anonymous credential system is *correct*, which means that for all $n, \lambda \in \mathbb{N}$, $\mathcal{G} \in [\text{GenSetting}(1^\lambda)]$, $pp \in [\text{Setup}(\mathcal{G})]$, $(ipk, cred^*) \in [\text{IssuerKeyGen}(pp, 1^n)]$, and all $\overrightarrow{attr} \in (\mathbb{A} \cup \{\star\})^n$, $\overrightarrow{attr}_{iss} \in (\mathbb{A} \cup \{\star\})^n \setminus \mathbb{A}^n$, $\phi \in \Phi_G$, $\psi \in \Psi_G[\overrightarrow{attr}_{iss}]$, $\alpha \in \{0, 1\}^*$, $ctx \in \{0, 1\}^*$ and all $cred, cred_{iss}$,

- “The root credential is valid”:

$$\text{CheckCred}(pp, ipk, \star^n, cred^\star) = 1$$

- “Issuance/delegation results in a credential on $\overrightarrow{attr'} = \psi(\perp, \alpha)$ ”: If $\psi(\perp, \alpha) \neq \perp$ and $\text{CheckCred}(pp, ipk, \overrightarrow{attr}_{\text{iss}}, cred_{\text{iss}}) = 1$, then

$$\Pr \left[\begin{array}{l} b = 1 \wedge \\ \text{CheckCred}(pp, ipk, \\ \psi(\perp, \alpha), cred') = 1 \end{array} \parallel \begin{array}{l} cred' \leftarrow \text{ReceiveCred}(pp, ipk, \overrightarrow{attr}_{\text{iss}}, \psi, ctx; \alpha) \\ \leftrightarrow \text{IssueCred}(pp, ipk, \overrightarrow{attr}_{\text{iss}}, \psi, ctx; cred_{\text{iss}}) \rightarrow b \end{array} \right] = 1$$

- “Updating results in a credential on $\overrightarrow{attr'} = \psi(\overrightarrow{attr}, \alpha)$ ”: If $\psi(\overrightarrow{attr}, \alpha) \neq \perp$ and $\text{CheckCred}(pp, ipk, \overrightarrow{attr}, cred) = \text{CheckCred}(pp, ipk, \overrightarrow{attr}_{\text{iss}}, cred_{\text{iss}}) = 1$, then

$$\Pr \left[\begin{array}{l} b = 1 \wedge \\ \text{CheckCred}(pp, ipk, \\ \psi(\overrightarrow{attr}, \alpha), cred') = 1 \end{array} \parallel \begin{array}{l} \vdash cred' \\ \text{ReceiveUpdate}(pp, ipk, \overrightarrow{attr}_{\text{iss}}, \psi, ctx; \overrightarrow{attr}, cred, \alpha) \\ \leftrightarrow \text{UpdateCred}(pp, ipk, \overrightarrow{attr}_{\text{iss}}, \psi, ctx; cred_{\text{iss}}) \rightarrow b \end{array} \right] = 1$$

- “Presenting a credential works”: If $\phi(\overrightarrow{attr}, \alpha) = 1$ and $\text{CheckCred}(pp, ipk, \overrightarrow{attr}, cred) = 1$, then

$$\Pr \left[\begin{array}{l} b = 1 \end{array} \parallel \begin{array}{l} \text{PresentCred}(pp, ipk, \phi, ctx; \overrightarrow{attr}, cred, \alpha) \\ \leftrightarrow \text{VerifyCred}(pp, ipk, \phi, ctx) \rightarrow b \end{array} \right] = 1$$

Note that correctness has an inductive character: The root credential is valid (according to CheckCred) and any credential issued using a valid credential is valid (and hence can be used for further issuance). Because issuance is naturally restricted to attributes covered by the issuer’s attributes $\overrightarrow{attr}_{\text{iss}}$, we restrict the update function $\psi \in \Psi_{\mathcal{G}}[\overrightarrow{attr}_{\text{iss}}]$ to only output $\overrightarrow{attr'} \preceq \overrightarrow{attr}_{\text{iss}}$. Formally, if $\psi \in \Psi_{\mathcal{G}}[\overrightarrow{attr}_{\text{iss}}]$, then it has the signature $\psi : ((\mathbb{A} \cup \{\star\})^n \cup \{\perp\}) \times \{0, 1\}^* \rightarrow \{\overrightarrow{attr'} \mid \overrightarrow{attr'} \preceq \overrightarrow{attr}_{\text{iss}}\} \cup \{\perp\}$.

Notably, correctness of issuance and updating is restricted to cases where the issuer’s attributes $\overrightarrow{attr}_{\text{iss}}$ contain at least one $\star$, i.e., we require $\overrightarrow{attr}_{\text{iss}} \in (\mathbb{A} \cup \{\star\})^n \setminus \mathbb{A}^n$. This is not a significant restriction: If $\overrightarrow{attr}_{\text{iss}}$ does not contain $\star$, then delegation is largely trivial: The only attributes that can be delegated are $\overrightarrow{attr'} = \overrightarrow{attr}_{\text{iss}}$, which, intuitively speaking, can be achieved by simply handing the issuer’s credential $cred_{\text{iss}}$ to the user. We discuss reasons behind this restriction and ways to circumvent it in Section 6.4.3.1.

In contrast to our original paper [BB18], Definition 6.2 treats $\star$ as a proper attribute. Predicates ϕ can encode statements such as “my i th attribute is $\star$ ”. Similarly, update functions ψ can output attributes that contain $\star$ values. For example, this enables applications where a user can prove they have certain delegation rights (encoded by $\star$ attributes). It also enables applications where a user receives a credential update where the $\star$ attributes are copied from the old to the new credential without revealing their

positions to the issuer/delegator. Such basic applications are not supported by our original work [BB18], where verification only works for credentials without $\star$ and the issuer/delegator always learns all attributes (except for the special hidden usk attribute) in the issued/delegated credential. One can think of the original [BB18] as the delegatable version of *traditional* anonymous credentials (Chapter 4), while our updated presentation in this thesis represents a delegatable version of *updatable* anonymous credentials (Chapter 5), with all the advantages that come with that formalism.

6.2.2 Anonymity

We consider two notions of anonymity: (1) anonymity for users receiving, updating, or presenting credentials, and (2) anonymity for the issuer. The latter is a new requirement brought on by the fact that in DACS, issuers (delegators) are also users and may expect some notion of privacy during delegation.

We start with anonymity for users receiving, updating, or presenting credentials. We want to ensure that issuers and verifiers do not learn anything about user's credentials, attributes, or hidden value α , except what is revealed by the update function $\psi(\cdot, \cdot) \neq \perp$ or the presentation predicate $\phi(\cdot, \cdot) = 1$. This means that we require existence of a simulator $\mathcal{S}$ that does not get $cred, \overrightarrow{attr}, \alpha$ as input but can still produce protocol messages indistinguishable from an honest user's. The notion is essentially the same as simulation anonymity notion for UACS (Definition 5.2), with very minor syntax adaptations due to the new $\overrightarrow{attr}_{iss}$ input to the user's protocols.

Definition 6.3 (Simulation anonymity). Let $\mathcal{S}$ be a stateful interactive algorithm. Let $\mathcal{A}$ be a ppt algorithm. Consider the following game $\text{Exp}_{\Pi_{dacs}}^{\text{sim-anon}}(\Pi_{dacs}, \mathcal{S}, \mathcal{A}, \lambda, X)$ for $X \in \{\text{REAL}, \text{SIM}\}$.

$\text{Exp}_{\Pi_{dacs}}^{\text{sim-anon}}(\Pi_{dacs}, \mathcal{S}, \mathcal{A}, \lambda, X)$

If $X = \text{REAL}$:

$pp \leftarrow \text{Setup}(\mathcal{G})$

If $X = \text{SIM}$:

$pp \leftarrow \mathcal{S}.\text{Setup}(\mathcal{G})$ // Simulator gets to set up pp

Run $b \leftarrow \mathcal{A}^{\mathcal{O}.\mathcal{RO}^X, \mathcal{O}.\text{ReceiveCred}^X, \mathcal{O}.\text{ReceiveUpdate}^X, \mathcal{O}.\text{PresentCred}^X}(\mathcal{G}, pp)$

return b

$\mathcal{O}.\mathcal{RO}^X(x)$ //Random oracle access

If $X = \text{REAL}$:

 return $\mathcal{RO}(x)$

If $X = \text{SIM}$:

 return $\mathcal{S}.\mathcal{RO}(x)$ //Simulator gets to answer random oracle queries

$\mathcal{O}.\text{ReceiveCred}^X(ipk, \overrightarrow{attr}_{iss}, \psi, ctx, \alpha)$ //Have user receive credential

assert $\psi(\perp, \alpha) \neq \perp$ and $\overrightarrow{attr}_{iss} \succeq \psi(\perp, \alpha)$
 Choose new credential handle C'
 Set $ipk_{C'} := ipk$ and $\overrightarrow{attr}_{C'} := \psi(\perp, \alpha)$ and $cred_{C'} := \perp$
 If $X = \text{REAL}$:
 $cred_{C'} \leftarrow \text{ReceiveCred}^{\mathcal{RO}}(pp, ipk, \overrightarrow{attr}_{iss}, \psi, ctx; \alpha) \leftrightarrow \mathcal{A}$
 If $X = \text{SIM}$:
 $b \leftarrow \mathcal{S}.\text{ReceiveCred}(pp, ipk, \overrightarrow{attr}_{iss}, \psi, ctx) \leftrightarrow \mathcal{A}$
 If $b = 1$, set $cred_{C'} = 1$ //Mark as a valid credential $cred_{C'} \neq \perp$
 return C' if $cred_{C'} \neq \perp$.

$\mathcal{O}.\text{ReceiveUpdate}^X(C_{\text{upd}}, \overrightarrow{attr}_{iss}, \psi, ctx, \alpha)$ //Have user receive update

assert $cred_{C_{\text{upd}}} \neq \perp$ and $\psi(\overrightarrow{attr}_{C_{\text{upd}}}, \alpha) \neq \perp$ and $\overrightarrow{attr}_{iss} \succeq \psi(\overrightarrow{attr}_{C_{\text{upd}}}, \alpha)$
 Choose new credential handle C'
 Set $ipk_{C'} := ipk_C$ and $\overrightarrow{attr}_{C'} := \psi(\overrightarrow{attr}_{C_{\text{upd}}}, \alpha)$ and $cred_{C'} := \perp$
 If $X = \text{REAL}$:
 $cred_{C'} \leftarrow \text{ReceiveUpdate}^{\mathcal{RO}}(pp, ipk_C, \overrightarrow{attr}_{iss}, \psi, ctx; \overrightarrow{attr}_{C_{\text{upd}}}, cred_{C_{\text{upd}}}, \alpha) \leftrightarrow \mathcal{A}$
 If $X = \text{SIM}$:
 $b \leftarrow \mathcal{S}.\text{ReceiveUpdate}(pp, ipk_C, \overrightarrow{attr}_{iss}, \psi, ctx) \leftrightarrow \mathcal{A}$
 If $b = 1$, set $cred_{C'} = 1$ //Mark as a valid credential $cred_{C'} \neq \perp$
 return C' if $cred_{C'} \neq \perp$.

$\mathcal{O}.\text{PresentCred}^X(C, \phi, ctx, \alpha)$ //Have user present credential

assert $cred_C \neq \perp$ and $\phi(\overrightarrow{attr}_C, \alpha) = 1$
 If $X = \text{REAL}$:
 $\text{PresentCred}^{\mathcal{RO}}(pp, ipk_C, \phi, ctx; \overrightarrow{attr}_C, cred_C, \alpha) \leftrightarrow \mathcal{A}$
 If $X = \text{SIM}$:
 $\mathcal{S}.\text{PresentCred}(pp, ipk_C, \phi, ctx) \leftrightarrow \mathcal{A}$

A delegatable updatable anonymous credential system Π_{dacs} has *simulation anonymity* if there exists a stateful interactive algorithm $\mathcal{S}$ such that for all ppt $\mathcal{A}$,

$$\left| \begin{array}{l} \Pr[\text{Exp}_{\Pi_{\text{dacs}}}^{\text{sim-anon}}(\Pi_{\text{dacs}}, \mathcal{S}, \mathcal{A}, \lambda, \text{REAL}) = 1] \\ - \Pr[\text{Exp}_{\Pi_{\text{dacs}}}^{\text{sim-anon}}(\Pi_{\text{dacs}}, \mathcal{S}, \mathcal{A}, \lambda, \text{SIM}) = 1] \end{array} \right|$$

is negligible and the runtime of $\text{Exp}_{\Pi_{\text{dacs}}}^{\text{sim-anon}}(\Pi_{\text{dacs}}, \mathcal{S}, \mathcal{A}, \lambda, \text{SIM})$ is polynomially bounded (where the polynomial depends on $\mathcal{A}$).

The above definition covers anonymity for credential receivers. One may wonder whether credential issuers (delegators) also deserve some notion of privacy. Indeed, we can imagine scenarios where the delegator wants to preserve her anonymity. For example, consider an anonymous forum where moderation rights are managed via anonymous credentials. An anonymous moderator may want to delegate (partial) moderation rights

to another anonymous moderator.

We have made the modeling choice of disclosing the issuer's attributes $\overrightarrow{attr}_{iss}$ to the receiver. This means that issuers can only be anonymous among the set of issuers who hold attributes $\overrightarrow{attr}_{iss}$ (or can delegate $\overrightarrow{attr}_{iss}$ to themselves). Within these constraints, we get a strong notion of anonymity: The receiver cannot distinguish between any two issuers using different credentials $cred_{iss}$ with the same attributes $\overrightarrow{attr}_{iss}$.

Definition 6.4 (Perfect anonymity for the delegating user). Let $\mathcal{A}$ be an algorithm. A delegatable updatable anonymous credential system has *perfect delegator anonymity* if for all $n, \lambda \in \mathbb{N}$, $\mathcal{G} \in [\text{GenSetting}(1^\lambda)]$, $pp \in [\text{Setup}(\mathcal{G})]$, all $ipk, \overrightarrow{attr}_{iss}, cred_{iss}, cred'_{iss}$ with $\text{CheckCred}(pp, ipk, \overrightarrow{attr}_{iss}, cred_{iss}) = \text{CheckCred}(pp, ipk, \overrightarrow{attr}_{iss}, cred'_{iss}) = 1$, all algorithms $\mathcal{A}$ and all ψ, ctx, v

$$\begin{aligned} & \Pr[\text{view}_{\mathcal{A}} = v \mid \mathcal{A} \leftrightarrow \text{IssueCred}(pp, ipk, \overrightarrow{attr}_{iss}, \psi, ctx; cred_{iss})] \\ &= \Pr[\text{view}_{\mathcal{A}} = v \mid \mathcal{A} \leftrightarrow \text{IssueCred}(pp, ipk, \overrightarrow{attr}_{iss}, \psi, ctx; cred'_{iss})] \end{aligned}$$

and

$$\begin{aligned} & \Pr[(\text{view}_{\mathcal{A}}, b) = v \mid \mathcal{A} \leftrightarrow \text{UpdateCred}(pp, ipk, \overrightarrow{attr}_{iss}, \psi, ctx; cred_{iss}) \rightarrow b] \\ &= \Pr[(\text{view}_{\mathcal{A}}, b) = v \mid \mathcal{A} \leftrightarrow \text{UpdateCred}(pp, ipk, \overrightarrow{attr}_{iss}, \psi, ctx; cred'_{iss}) \rightarrow b] \end{aligned}$$

This definition is information-theoretical, i.e., the distribution of the (adversarial) receiver's views is independent of which (valid) $cred_{iss}$ is used. In particular, this means that the receiver cannot distinguish whether it interacts with the same issuer twice (with the issuer using the same credential both times) or with two different issuers (who use their respective credentials).

In practice, privacy-conscious delegators should make sure that $\overrightarrow{attr}_{iss}$ reveals as little as possible about them. In particular, if an issuer has a more powerful credential than needed for the issuance/update, the issuer should first delegate a weaker credential, with more restricted attributes, to herself, and use that self-delegated credential for issuance. In our anonymous forum example, consider a moderator who has moderation rights for Board 1 and Board 2, but not Board 3, encoded as attributes $(\text{Board1}, \text{Board2}, \text{Board3}) = (\star, \star, 0)$ (cf. discussion around Definition 6.1). Note that this combination may reveal significant information about the moderator's identity. If the moderator wants to delegate rights for only Board 2, she should first delegate attributes $\overrightarrow{attr}_{iss} = (0, \star, 0)$ to herself and use *that* in the issuance protocol. This act makes the moderator indistinguishable from any other moderator who is allowed to delegate moderation rights for Board 2.

One may ask whether the design choice of revealing the issuer's attributes $\overrightarrow{attr}_{iss}$, which seems to restrict issuer anonymity significantly, is reasonable. First, we note that this choice is convenient for DACS constructions: $\overrightarrow{attr}_{iss}$ clearly communicates the effective limits for the update function ψ : For issuance to succeed, the function ψ must output the non- $\star$ attributes from $\overrightarrow{attr}_{iss}$ unchanged, and it is allowed to choose values for the $\star$ attributes. Second, these limits on ψ also make it difficult in general to hide $\overrightarrow{attr}_{iss}$ in any strong sense. The receiver learns about the $\star$ attribute values because those are the ones

ψ is allowed to change (and even if we tried to hide this, issuance would *have to* visibly fail if the receiver tries to receive a credential where it tries to modify a non- $\star$ attribute). The receiver learns about the values of non- $\star$ attribute because they will inevitably end up as attributes, unchanged, in the received credential. So in that sense, we argue that $\overrightarrow{attr}_{iss}$ is natural leakage to the receiver: The receiver can generally gain information about $\overrightarrow{attr}_{iss}$ by testing whether $\overrightarrow{attr}_{iss}$ is able to issue some given test attribute vector. Hence we have decided not to hide $\overrightarrow{attr}_{iss}$ at all, leaving it up to the issuer to delegate to herself if she wants to hide unique aspects of $\overrightarrow{attr}_{iss}$. Other choices are also reasonable, such as building the delegate-to-self step into the issuance protocol.

6.2.3 Explainability

We now turn to soundness guarantees for issuers and verifiers, i.e., we want to ensure that users cannot forge credentials, or prove false claims during presentation (or issuance). The approach for DACS is the same as for UACS (Definition 5.3): We require existence of an explainer algorithm $\mathcal{E}$ that outputs the hidden input of ψ or ϕ . We then require that the explanations of $\mathcal{E}$ are *consistent*. For example, if the explanation for a presentation is that the adversary used a credential with attributes $\overrightarrow{attr}$, then we expect that some earlier issuance or update operation must have issued attributes $\overrightarrow{attr}$ (according to $\mathcal{E}$). Like in Definition 5.3, we model honest users by giving the adversary access to the simulator from Definition 6.3.

Overall, the following definition is very similar to its UACS counterpart (Definition 5.3), with the following notable differences.

- The addition of $\star$ attributes means we need to modify the requirements on the explainer. If an explanation points towards the adversary using attributes $\overrightarrow{attr}$, then we previously required that exactly $\overrightarrow{attr}$ must have been issued before. For DACS, of course it suffices that any $\overrightarrow{attr}' \succeq \overrightarrow{attr}$ has been issued before (since that enables the adversary to delegate $\overrightarrow{attr}$ to himself). For example, if the adversary asks for a credential for attributes $(\star, 2)$, then it is legal for the adversary to later present $(1, 2)$.
- We need to enable the adversary to model the delegation scenarios it wants to attack. In the UACS definition, the honest issuer using *isk* is the only way to issue credentials to the adversary. In DACS, credentials can be issued using other honest users' credentials. Hence we allow the adversary to choose which credential an honest user should use to issue/update the adversary's credential. To this end, we model honest users' credentials: We introduce new oracles $\mathcal{O}.\text{IssueHonest}$, $\mathcal{O}.\text{UpdateHonest}$, to have honest users issue or update credentials among themselves, and oracles $\mathcal{O}.\text{ReceiveCred}$, $\mathcal{O}.\text{ReceiveUpdate}$, to have honest users receive credentials from the adversary. The adversary can then demand any credential created during these processes be delegated to the adversary (i.e., used as $cred_{iss}$ during issuance or update).

The presence of oracles $\mathcal{O}.\text{ReceiveCred}$, $\mathcal{O}.\text{ReceiveUpdate}$ may seem unnecessary at first glance: If the adversary delegates attributes $\overrightarrow{attr}$ to some honest user and then has the honest user issue $\overrightarrow{attr}' \preceq \overrightarrow{attr}$ back to the adversary, then intuitively, nothing is gained.

The adversary already had a credential for $\overrightarrow{attr}$ and could have delegated $\overrightarrow{attr}'$ to itself. However, these oracles are crucial to ensure that maliciously issued credentials cannot be used to circumvent other guarantees of the explainability game, such as the guarantee that all adversarial operations can be explained. For example, consider a scheme where the corrupt issuer can embed some bit b into the issued credential $cred$ that makes the honest user skip important checks when delegating $cred$. This can cause the explainer to fail to produce an explanation.

Definition 6.5 (Simulation explainability). Let $\mathcal{S}$ be a stateful interactive algorithm, let $\mathcal{E}$ be an interactive algorithm, and let $\mathcal{A}$ be a ppt. Consider the following experiment $\text{Exp}_{\Pi_{\text{dacs}}}^{\text{sim-expl}}(\Pi_{\text{dacs}}, \mathcal{A}, \mathcal{S}, \mathcal{E}, n, \lambda)$.

$\text{Exp}_{\Pi_{\text{dacs}}}^{\text{sim-expl}}(\Pi_{\text{dacs}}, \mathcal{A}, \mathcal{S}, \mathcal{E}, n, \lambda)$

$\mathcal{G} \leftarrow \text{GenSetting}(1^\lambda), pp \leftarrow \mathcal{S}.\text{Setup}(\mathcal{G})$ //Simulated setup
 $(ipk^*, cred_{\text{root}}) \leftarrow \text{IssuerKeyGen}(pp, 1^n)$ //Honest issuer root key
 Set $\overrightarrow{attr}_{\text{root}} = \star^n$
 //root is a credential handle: $C_{\text{iss}} = \text{root}$ is valid input to oracles below
 Run $\mathcal{A}(\mathcal{G}, pp, ipk^*)$, granting it access to oracles $\mathcal{S}.\mathcal{RO}, \mathcal{O}.\text{IssueCred}, \mathcal{O}.\text{UpdateCred},$
 $\mathcal{O}.\text{VerifyCred}, \mathcal{O}.\text{SimReceiveCred}, \mathcal{O}.\text{SimReceiveUpdate}, \mathcal{O}.\text{SimPresent}.$
 When an oracle raises $\text{fail}_{\text{explain}}$, output 1
 If no such event has been raised when $\mathcal{A}$ halts, output 0

$\mathcal{O}.\text{IssueCred}(C_{\text{iss}}, \psi, ctx)$ //Adversarial user gets issued/delegated credential

Set $explained = 0$ and $\alpha = \perp$
 $b \leftarrow \mathcal{E}.\text{IssueCred}^{\mathcal{S}.\mathcal{RO}, \mathcal{O}.\text{GetSimState}, \mathcal{O}.\text{RegExpl}}(pp, ipk^*, \overrightarrow{attr}_{C_{\text{iss}}}, \psi, ctx; cred_{C_{\text{iss}}}) \leftrightarrow \mathcal{A}$

$\mathcal{O}.\text{RegExpl}(\alpha)$ //Callback for explainer to register explanation

assert $explained = 0$ //Only accept one explanation
 assert $\psi(\perp, \alpha) \neq \perp$ //Explanation consistent with ψ
 Add $\psi(\perp, \alpha)$ to $\mathcal{T}_{\text{explained}}$ //Recognize $\psi(\perp, \alpha)$ for future explanations
 Set $explained = 1$

If $explained = 0$ and $b = 1$ and $(\text{ReceiveCred}, ipk^*, \overrightarrow{attr}_{C_{\text{iss}}}, \psi, ctx) \notin \mathcal{T}_{\text{sim}}$:

raise $\text{fail}_{\text{explain}}$ // $\mathcal{E}$ failed to produce an explanation

If $explained = 1$ and $b = 0$:

raise $\text{fail}_{\text{explain}}$ // $b = 0$: No credential must be issued

return α //Give explanation to $\mathcal{A}$

$\mathcal{O}.\text{UpdateCred}(C_{\text{iss}}, \psi, ctx)$ //Adversarial user gets a credential updated

Set $explained = 0$ and $(\vec{attr}, \alpha) = (\perp, \perp)$
 $b \leftarrow \mathcal{E}.\text{UpdateCred}^{S.\mathcal{RO}, \mathcal{O}.\text{GetSimState}, \mathcal{O}.\text{RegExpl}}(pp, ipk^*, \vec{attr}_{C_{\text{iss}}}, \psi, ctx; cred_{C_{\text{iss}}}) \leftrightarrow \mathcal{A}$

$\mathcal{O}.\text{RegExpl}(\vec{attr}, \alpha)$ //Callback for explainer to register explanation

assert $explained = 0$ //Only accept one explanation
 assert $\psi(\vec{attr}, \alpha) \neq \perp$ //Explanation consistent with ψ
 assert $\vec{attr} \preceq \mathcal{T}_{\text{explained}}$ //Old attributes explained
 Add $\psi(\vec{attr}, \alpha)$ to $\mathcal{T}_{\text{explained}}$ //Consider updated attributes explained
 Set $explained = 1$

If $explained = 0$ and $b = 1$ and $(\text{ReceiveUpdate}, ipk^*, \vec{attr}_{C_{\text{iss}}}, \psi, ctx) \notin \mathcal{T}_{\text{sim}}$:
 raise $\text{fail}_{\text{explain}}$ // $\mathcal{E}$ failed to produce an explanation
 If $explained = 1$ and $b = 0$:
 raise $\text{fail}_{\text{explain}}$ // $b = 0$: No credential must be issued
 return $(\vec{attr}, \alpha)$ //Give explanation to $\mathcal{A}$

$\mathcal{O}.\text{VerifyCred}(ipk, \phi, ctx)$ //Adversarial user presents a credential

Set $explained = 0$ and $(\vec{attr}, \alpha) = (\perp, \perp)$
 $b \leftarrow \mathcal{E}.\text{VerifyCred}^{S.\mathcal{RO}, \mathcal{O}.\text{GetSimState}, \mathcal{O}.\text{RegExpl}}(pp, ipk, \phi, ctx) \leftrightarrow \mathcal{A}$

$\mathcal{O}.\text{RegExpl}(\vec{attr}, \alpha)$ //Callback for explainer to register explanation

assert $explained = 0$ //Only accept one explanation
 assert $\phi(\vec{attr}, \alpha) = 1$ //Explanation consistent with ϕ
 assert $ipk \neq ipk^* \vee \vec{attr} \preceq \mathcal{T}_{\text{explained}}$ //Honest issuer $\Rightarrow \vec{attr}$ explained
 Set $explained = 1$

If $explained = 0$ and $b = 1$ and $(\text{PresentCred}, ipk, \phi, ctx) \notin \mathcal{T}_{\text{sim}}$:
 raise $\text{fail}_{\text{explain}}$ // $\mathcal{E}$ failed to produce an explanation
 return $(\vec{attr}, \alpha)$ //Give explanation to $\mathcal{A}$

$\mathcal{O}.\text{IssueHonest}(C_{\text{iss}}, \psi, \alpha, ctx)$ //Honest user delegates to another honest user

assert $\vec{attr}_{C_{\text{iss}}} \in (\mathbb{A} \cup \{\star\})^n \setminus \mathbb{A}^n$ and $\psi \in \Psi_{\mathcal{G}}[\vec{attr}_{C_{\text{iss}}}]$ and $\perp \neq \psi(\perp, \alpha)$
 Choose new credential handle C'
 Set $\vec{attr}_{C'} = \psi(\perp, \alpha)$
 $cred_{C'} \leftarrow \text{ReceiveCred}^{S.\mathcal{RO}}(pp, ipk, \vec{attr}_{C_{\text{iss}}}, \psi, ctx; \alpha)$
 $\quad \leftrightarrow \text{IssueCred}^{S.\mathcal{RO}}(pp, ipk, \vec{attr}_{C_{\text{iss}}}, \psi, ctx; cred_{C_{\text{iss}}})$
 return C'

$\mathcal{O}.\text{UpdateHonest}(C_{\text{iss}}, C_{\text{upd}}, \psi, \alpha, ctx)$ //Honest user updates honest user's cred

assert $\vec{attr}_{C_{\text{iss}}} \in (\mathbb{A} \cup \{\star\})^n \setminus \mathbb{A}^n$ and $\psi \in \Psi_{\mathcal{G}}[\vec{attr}_{C_{\text{iss}}}]$ and $\perp \neq \psi(\vec{attr}_{C_{\text{upd}}}, \alpha)$
 Choose new credential handle C'
 Set $\vec{attr}_{C'} = \psi(\vec{attr}_{C_{\text{upd}}}, \alpha)$
 $cred_{C'} \leftarrow \text{ReceiveUpdate}^{S.\mathcal{RO}}(pp, ipk, \vec{attr}_{C_{\text{iss}}}, \psi, ctx; \vec{attr}_{C_{\text{upd}}}, cred_{C_{\text{upd}}}, \alpha)$
 $\leftrightarrow \text{UpdateCred}^{S.\mathcal{RO}}(pp, ipk, \vec{attr}_{C_{\text{iss}}}, \psi, ctx; cred_{C_{\text{iss}}})$
 return C'

$\mathcal{O}.\text{ReceiveCred}(\vec{attr}_{\text{iss}}, \psi, \alpha, ctx)$ //Honest user receiving credential from $\mathcal{A}$

assert $\perp \neq \psi(\perp, \alpha)$
 $cred' \leftarrow \text{ReceiveCred}^{S.\mathcal{RO}}(pp, ipk^*, \vec{attr}_{\text{iss}}, \psi, ctx; \alpha) \leftrightarrow \mathcal{A}$
 assert $cred' \neq \perp$
 Choose new credential handle C'
 Set $cred_{C'} = cred'$ and $\vec{attr}_{C'} = \psi(\perp, \alpha)$
 return C'

$\mathcal{O}.\text{ReceiveUpdate}(\vec{attr}_{\text{iss}}, C_{\text{upd}}, \psi, \alpha, ctx)$ //Honest user gets update from $\mathcal{A}$

assert $\perp \neq \psi(\vec{attr}_{C_{\text{upd}}}, \alpha)$
 $cred' \leftarrow \text{ReceiveUpdate}^{S.\mathcal{RO}}(pp, ipk^*, \vec{attr}_{\text{iss}}, \psi, ctx; \vec{attr}_{C_{\text{upd}}}, cred_{C_{\text{upd}}}, \alpha) \leftrightarrow \mathcal{A}$
 assert $cred' \neq \perp$
 Choose new credential handle C'
 Set $cred_{C'} = cred'$ and $\vec{attr}_{C'} = \psi(\vec{attr}_{C_{\text{upd}}}, \alpha)$
 return C'

$\mathcal{O}.\text{GetSimState}()$ //Used by explainer $\mathcal{E}$. Not accessible to $\mathcal{A}$

return $states$

$\mathcal{O}.\text{SimReceiveCred}(ipk, \vec{attr}_{\text{iss}}, \psi, ctx)$ //Anonymity of honest user receiving

Add $(\text{ReceiveCred}, ipk, \vec{attr}_{\text{iss}}, \psi, ctx)$ to $\mathcal{T}_{\text{sim}}$
 $b \leftarrow S.\text{ReceiveCred}(pp, ipk, \vec{attr}_{\text{iss}}, \psi, ctx) \leftrightarrow \mathcal{A}$
 return b

$\mathcal{O}.\text{SimReceiveUpdate}(ipk, \vec{attr}_{\text{iss}}, \psi, ctx)$ //Anonymity of honest user updating

Add $(\text{ReceiveUpdate}, ipk, \vec{attr}_{\text{iss}}, \psi, ctx)$ to $\mathcal{T}_{\text{sim}}$
 $b \leftarrow S.\text{ReceiveUpdate}(pp, ipk, \vec{attr}_{\text{iss}}, \psi, ctx) \leftrightarrow \mathcal{A}$
 return b

$\mathcal{O}.\text{SimPresent}(ipk, \phi, ctx)$ //Anonymity of honest user presenting

Add $(\text{PresentCred}, ipk, \phi, ctx)$ to $\mathcal{T}_{\text{sim}}$

$\mathcal{S}.\text{PresentCred}(pp, ipk, \phi, ctx) \leftrightarrow \mathcal{A}$

A delegatable updatable anonymous credential system Π_{dacs} has *simulation explainability*, if there exists a stateful interactive algorithm $\mathcal{S}$ and an interactive algorithm $\mathcal{E}$ such that:

- $\mathcal{S}$ fulfills the requirements of Definition 6.3.
- $\mathcal{E}.X^{\mathcal{S}, \mathcal{RO}, \mathcal{O}.\text{GetSimState}, \mathcal{O}.\text{RegExpl}}$ perfectly emulates $\Pi_{\text{dacs}}.X^{\mathcal{S}, \mathcal{RO}}$ (cf. Definition 2.2) for $X \in \{\text{IssueCred}, \text{UpdateCred}, \text{VerifyCred}\}$.
- For all $n \in \mathbb{N}$ and all ppt $\mathcal{A}$, $\Pr[\text{Exp}_{\Pi_{\text{dacs}}}^{\text{sim-expl}}(\Pi_{\text{uacs}}, \mathcal{A}, \mathcal{S}, \mathcal{E}, n, \lambda) = 1]$ is negligible.
- For all $n \in \mathbb{N}$ and all ppt $\mathcal{A}$, there exists a polynomial $poly$ such that the runtime of $\text{Exp}_{\Pi_{\text{dacs}}}^{\text{sim-expl}}(\Pi_{\text{uacs}}, \mathcal{A}, \mathcal{S}, \mathcal{E}, n, \lambda)$ is at most $poly(\lambda)$.

We note that perfect delegator anonymity (Definition 6.4) implies that it is irrelevant which credential cred_{iss} the issuer uses for issuance/updates, as long as it is valid under attributes $\overrightarrow{\text{attr}}_{\text{iss}}$. So assuming perfect delegator anonymity, we can simplify the simulation explainability definition above. Instead of pointing to credential handles $\mathbf{C}_{\text{iss}}$, the adversary would just give us the desired attributes $\overrightarrow{\text{attr}}_{\text{iss}}$, the experiment would generate a fitting cred_{iss} on $\overrightarrow{\text{attr}}_{\text{iss}}$ by delegating the $\star^n$ credential to itself, and then use that for the protocol. This change does not affect the outcome of the experiment, assuming (1) perfect delegator anonymity, and (2) all credentials output by honest receivers (even after interacting with corrupted issuers) are valid according to CheckCred . The latter requirement can be assumed without loss of generality, as the receiver can always check CheckCred itself before outputting a credential. Eliminating the need for $\mathcal{A}$ to specify issuance credential handles $\mathbf{C}_{\text{iss}}$ would allow us to remove the oracles $\mathcal{O}.\text{IssueHonest}$, $\mathcal{O}.\text{UpdateHonest}$ and $\mathcal{O}.\text{ReceiveCred}, \mathcal{O}.\text{ReceiveUpdate}$, whose sole purpose is to model honest users receiving credentials in order to delegate them to the adversary. However, while a simple definition is desirable, the perfect delegator anonymity property is not *essential* to delegatable anonymous credentials. We can easily imagine scenarios, such as the driver's license example from the Chapter 6 introduction, that are indifferent to delegator anonymity. For this reason, we want to be able to model systems without delegator anonymity, which is why the simulation explainability property is defined in a way that does not presuppose it.

6.3 Dynamically malleable signatures

The central building block for our delegatable anonymous credential system is going to be a *dynamically malleable signature scheme* (DMS scheme). Signatures of DMS schemes authenticate vectors of messages $\vec{m} \in (\mathbb{M} \cup \{\star\})^n$. Given a signature σ_{iss} on $\vec{m}^{(\text{iss})}$, one

can efficiently compute a signature σ' on any $\vec{m}' \preceq \vec{m}^{(\text{iss})}$ (where $\preceq$ is as in Definition 6.1). One can view this process as adapting σ_{iss} to the new message $\vec{m}'$. In this sense, σ_{iss} is *malleable*. In contrast to regular malleable signature schemes (e.g., [CKLM14]), the degree to which a signature is malleable is modified during that process. For example, a signature on $\vec{m} \in \mathbb{M}^n$ (no $\star$ entries) is not malleable at all: One cannot derive a valid signature on *any* other message from it. In contrast, a signature on $\star^n$ (all $\star$ entries) is maximally malleable: One can derive a valid signature on *any* message from it. In general, whenever one derives a signature on $\vec{m}' \neq \vec{m}^{(\text{iss})}$, the resulting signature is strictly less malleable than the original one (as it contains fewer $\star$ entries). In this sense, the scheme is *dynamically* malleable: The degree of malleability is not statically fixed but can be changed by the holder of a signature.

The DMS scheme fills the same role for DACS as the blind signature scheme for signing a committed value (Definition 5.4) fills in UACS: A (delegatable) credential is a (dynamically malleable) signature. A DMS scheme comes with an efficient protocol to sign committed values, which will allow us to (blindly) issue and update credentials.

As usual, we start by defining the syntax of DMS (Section 6.3.1), then proceed to define security (Sections 6.3.2 and 6.3.3), then present our construction and prove it secure (Section 6.3.4).

6.3.1 Syntax

Definition 6.6 (Dynamically malleable signature scheme with a protocol for blindly signing committed values). A *dynamically malleable signature scheme (DMS)* Π_{dms} with message space $\mathbb{M}_{\mathcal{G}}$ consists of the following ppt algorithms.

Setup($\mathcal{G}$) $\rightarrow$ pp

Outputs public parameters pp .

Keygen($pp, 1^n$) $\rightarrow$ $(pk, \sigma_{\text{root}})$

Generates a public key pk and a signature σ_{root} on $\star^n$. $\heartsuit$ From σ_{root} , one can derive signatures on all possible messages $\vec{m} \in (\mathbb{M} \cup \{\star\})^n$.

Sign($pp, pk, \sigma_{\text{iss}}, \vec{m}^{(\text{iss})}, \vec{m}'$) $\rightarrow$ σ'

Given a signature σ_{iss} on $\vec{m}^{(\text{iss})}$, generates a signature σ' on $\vec{m}' \preceq \vec{m}^{(\text{iss})}$.

Verify($pp, pk, \vec{m}, \sigma$) $= b$

A deterministic algorithm, checking whether σ is a valid signature on $\vec{m} \in (\mathbb{M} \cup \{\star\})^n$.

Precommit($pp, pk, \vec{m}^{(\text{iss})}, \sigma_{\text{iss}}$) $\rightarrow$ (K, k)

Given a signature σ_{iss} on $\vec{m}^{(\text{iss})}$, generates a public commitment key K and a private key k . $\heartsuit$ This key K is used by the receiver to commit to the messages he desires to receive.

Commit($pp, pk, K, \vec{m}^{(\text{iss})}; \vec{m}'$) $\rightarrow$ (C, o)

Outputs a commitment C to message $\vec{m}' \in (\mathbb{M}_{\mathcal{G}} \cup \{\star\})^n$ with opening o .

$\text{VerifyCommit}(pp, pk, K, \vec{m}^{(\text{iss})}, C; \vec{m}', o) = b$
 A deterministic algorithm checking commitment C .

$\text{BlindSign}(pp, pk, K, \vec{m}^{(\text{iss})}, C; \sigma_{\text{iss}}, k) \leftrightarrow \text{BlindReceive}(pp, pk, K, \vec{m}^{(\text{iss})}, C; \vec{m}', o) \rightarrow \sigma'$
 An interactive protocol after which the receiver outputs a signature σ' on $\vec{m}'$.

The scheme is *correct*, i.e., if for all $\lambda, n \in \mathbb{N}$, all $\mathcal{G} \in [\text{GenSetting}(1^\lambda)]$, $pp \in [\text{Setup}(\mathcal{G})]$, all $(pk, \sigma_{\text{root}}) \in [\text{Keygen}(pp, 1^n)]$, all $\sigma_{\text{iss}} \in \{0, 1\}^*$, and all $\vec{m}^{(\text{iss})}, \vec{m}' \in (\mathbb{M}_{\mathcal{G}} \cup \{\star\})^n$:

- It holds that

$$\text{Verify}(pp, pk, \star^n, \sigma_{\text{root}}) = 1$$

💡 σ_{root} output by **Keygen** is a valid signature on $\star^n$.

- If $\text{Verify}(pp, pk, \vec{m}^{(\text{iss})}, \sigma_{\text{iss}}) = 1$ and $\vec{m}^{(\text{iss})} \succeq \vec{m}'$, then

$$\Pr[\text{Verify}(pp, pk, \vec{m}', \sigma') = 1 \mid \sigma' \leftarrow \text{Sign}(pp, pk, \sigma_{\text{iss}}, \vec{m}^{(\text{iss})}, \vec{m}')] = 1$$

💡 **Sign** outputs valid signatures.

- If $\vec{m}^{(\text{iss})} \succeq \vec{m}'$, then for all pk', K ,

$$\Pr \left[\begin{array}{c} \text{VerifyCommit}(pp, pk', K, \\ \vec{m}^{(\text{iss})}, C; \vec{m}', o) = 1 \end{array} \mid (C, o) \leftarrow \text{Commit}(pp, pk', K, \vec{m}^{(\text{iss})}; \vec{m}') \right] = 1$$

💡 The commitment procedure outputs valid commitments (even for malicious keys pk', K).

- If $\text{Verify}(pp, pk, \vec{m}^{(\text{iss})}, \sigma_{\text{iss}}) = 1$ and $\vec{m}^{(\text{iss})} \succeq \vec{m}'$, then

$$\Pr \left[\begin{array}{c} \text{Verify}(pp, pk, \\ \vec{m}', \sigma') = 1 \end{array} \mid \begin{array}{l} (K, k) \leftarrow \text{Precommit}(pp, pk, \vec{m}^{(\text{iss})}, \sigma_{\text{iss}}) \\ (C, o) \leftarrow \text{Commit}(pp, pk, K, \vec{m}^{(\text{iss})}; \vec{m}') \\ \text{BlindSign}(pp, pk, K, \vec{m}^{(\text{iss})}, C; \sigma_{\text{iss}}, k) \\ \leftrightarrow \text{BlindReceive}(pp, pk, K, \vec{m}^{(\text{iss})}, C; \vec{m}', o) \rightarrow \sigma' \end{array} \right] = 1$$

💡 The blind signing protocol outputs valid signatures.

The idea for such schemes is as follows. **Keygen**, **Sign**, **Verify** are as for ordinary signature schemes, with the difference that instead of a secret key sk , **Keygen** outputs a signature σ_{root} on the message $\star^n$. Similarly, **Sign** takes as input some valid signature σ_{iss} (and its corresponding message $\vec{m}^{(\text{iss})}$) instead of a secret key sk . This models malleability: Creation of additional signatures does not require sk , but rather some previous (sufficiently powerful) signature.

The protocol for signing a committed value works as follows. First, the signer computes a commitment key K via **Precommit** and sends K to the receiver. She keeps the corresponding private key k secret. Then using **Commit**, the receiver commits to the message $\vec{m}'$ he wants to have signed and sends the commitment C to the signer. Then signer and receiver run the **BlindSign** $\leftrightarrow$ **BlindReceive** protocol. After the protocol, the receiver gets a signature σ' on $\vec{m}'$.

Similar to Definition 6.2, correctness of DMS has a recursive flavor: If one runs the blind signing protocol with a valid signature σ_{iss} , then the resulting signature is valid (and hence reasonable input to future signing protocols).

The signing algorithm is somewhat redundant: To create a signature, the signer can always run the blind signing process with herself. We include it here as a convenient way to describe what signatures look like in our construction later. The **Precommit** algorithm and commitment key K allows commitments to suitably depend on the signer's signature σ_{iss} . Intuitively, this was not necessary in our previous protocol for signing committed values (Definition 5.4) because there, the signer always uses the same secret key sk for signing (whereas for DMS, any signature enables signing).

6.3.2 Unforgeability

We first define unforgeability of the signature scheme itself (**Keygen**, **Sign**, **Verify**), without the protocol for signing committed values. Unforgeability is similar to the standard EUF-CMA notion, except that it is trivial to generate signatures on new messages via malleability. For this reason, we restrict the adversary's winning condition to cases where it has not queried for a signature on any message $\vec{m}$ from which the forgery message $\vec{m}^*$ can be legally derived.

Definition 6.7 (Unforgeability). A DMS signature scheme Π_{dms} is *existentially unforgeable under chosen-message attacks* if for all $n \in \mathbb{N}$ and all ppt adversaries $\mathcal{A}$,

$$\Pr \left[\begin{array}{l} \text{Verify}(pp, pk, \vec{m}^*, \sigma^*) = 1 \\ \wedge \neg \exists \vec{m} \in \mathcal{T}_{\text{signed}} : \vec{m} \succeq \vec{m}^* \end{array} \middle| \begin{array}{l} \mathcal{G} \leftarrow \text{GenSetting}(1^\lambda) \\ pp \leftarrow \text{Setup}(\mathcal{G}) \\ (pk, \sigma_{\text{root}}) \leftarrow \text{Keygen}(pp, 1^n) \\ (\vec{m}^*, \sigma^*) \leftarrow \mathcal{A}^{\mathcal{O}.\text{PrivateSign}, \mathcal{O}.\text{Sign}}(\mathcal{G}, pp, pk) \end{array} \right]$$

is negligible, where

$\mathcal{O}.\text{PrivateSign}(S, \vec{m}') \quad // \text{Models deriving signature for honest party}$

```
assert  $\sigma_S \neq \perp$  and  $\vec{m}_S \succeq \vec{m}'$ 
Choose new signature handle  $S'$ 
 $\sigma_{S'} \leftarrow \text{Sign}(pp, pk, \sigma_S, \vec{m}_S, \vec{m}')$ 
Set  $\vec{m}_{S'} = \vec{m}'$ 
return  $S'$ 
```

$\mathcal{O}.\text{Sign}(S, \vec{m}') \quad // \text{Models signature queries for unforgeability}$

```
assert  $\sigma_S \neq \perp$  and  $\vec{m}_S \succeq \vec{m}'$ 
Add  $\vec{m}'$  to  $\mathcal{T}_{\text{signed}}$ 
 $\sigma' \leftarrow \text{Sign}(pp, pk, \sigma_S, \vec{m}_S, \vec{m}')$ 
return  $\sigma'$ 
```

Using the handle $S = \text{root}$ enables $\mathcal{A}$ to reference the root signature σ_{root} .

In this definition, $\mathcal{O}.\text{PrivateSign}$ allows the adversary to have honest users create signatures (without revealing them to $\mathcal{A}$). These can be referenced via the handle S for $\mathcal{O}.\text{Sign}$ queries, giving the adversary control over what signatures are used to derive the signatures it receives.

We now proceed to define unforgeability with respect to the blind signing a committed value protocol. Like Definition 5.5, this definition requires the adversary to provide both the message $\vec{m}'$ it committed to and the commitment opening o to the signing oracle.

Definition 6.8 (Blind unforgeability). A DMS signature scheme Π_{dms} is *blindly unforgeable under chosen-known-message attacks* if for all $n \in \mathbb{N}$ and all ppt adversaries $\mathcal{A}$,

$$\Pr \left[\begin{array}{l} \text{Verify}(pp, pk, \vec{m}^*, \sigma^*) = 1 \\ \wedge \neg \exists \vec{m} \in \mathcal{T}_{\text{signed}} : \vec{m} \succeq \vec{m}^* \end{array} \middle| \begin{array}{l} \mathcal{G} \leftarrow \text{GenSetting}(1^\lambda) \\ pp \leftarrow \text{Setup}(\mathcal{G}) \\ (pk, \sigma_{\text{root}}) \leftarrow \text{Keygen}(pp, 1^n) \\ \downarrow \mathcal{A}^{\mathcal{O}.\text{PrivateBlindSign}, \mathcal{O}.\text{BlindSign}}(\mathcal{G}, pp, pk) \\ (\vec{m}^*, \sigma^*) \end{array} \right]$$

is negligible, where

$\mathcal{O}.\text{PrivateBlindSign}(S, \vec{m}') // \text{Models deriving signature for honest party}$

```

assert  $\sigma_S \neq \perp$  and  $\vec{m}_S \succeq \vec{m}'$ 
Choose new signature handle  $S'$ 
 $(K, k) \leftarrow \text{Precommit}(pp, pk, \vec{m}_S, \sigma_S)$ 
 $(C, o) \leftarrow \text{Commit}(pp, pk, K, \vec{m}_S; \vec{m}')$ 
 $\text{BlindSign}(pp, pk, K, \vec{m}_S, C; \sigma_S, k) \leftrightarrow \text{BlindReceive}(pp, pk, K, \vec{m}_S, C; \vec{m}', o) \rightarrow \sigma_{S'}$ 
Set  $\vec{m}_{S'} = \vec{m}'$ 
return  $S'$ 

```

$\mathcal{O}.\text{BlindSign}(S) // \text{Models blind signature queries for unforgeability}$

```

assert  $\sigma_S \neq \perp$ 
 $(K, k) \leftarrow \text{Precommit}(pp, pk, \vec{m}_S, \sigma_S)$ 
 $(C, \vec{m}', o) \leftarrow \mathcal{A}(K) // \text{Models that } \mathcal{A} \text{ knows opening (cf. Definition 5.5)}$ 
assert  $\text{VerifyCommit}(pp, pk, K, \vec{m}_S, C; \vec{m}', o) = 1$ 
assert  $\vec{m}_S \succeq \vec{m}'$ 
Add  $\vec{m}'$  to  $\mathcal{T}_{\text{signed}}$ 
 $\text{BlindSign}(pp, pk, K, \vec{m}_S, C; \sigma_S, k) \leftrightarrow \mathcal{A}$ 

```

Using the handle $S = \text{root}$ enables $\mathcal{A}$ to reference the root signature σ_{root} .

Again, we have two oracles. $\mathcal{O}.\text{PrivateBlindSign}$ models deriving signatures that are not revealed to $\mathcal{A}$ and $\mathcal{O}.\text{BlindSign}$ models $\mathcal{A}$ requesting a signature. For the latter, note that the oracle essentially plays through the blind signing protocol, including **Precommit** and **commitment** phase. These blind signing protocol executions can be interleaved (cf.

Section 2.2.5). Unlike a real world execution, $\mathcal{A}$ is required to reveal the opening $\vec{m}', o$. When we later use this definition, we will extract $\vec{m}', o$ from a proof of knowledge and then reveal them to the oracle accordingly.

6.3.3 Anonymity

We now define privacy properties of the blind signing protocol in two parts: Privacy for the receiver and privacy for the signer.

For privacy for the receiver, we require that an adversarial signer cannot distinguish which of two messages is being signed.

Definition 6.9 (Blindness). A DMS signature scheme Π_{dms} has *signing blindness* if for all ppt adversaries $\mathcal{A}$,

$$\left| \Pr[\mathcal{A}^{\mathcal{O}.\text{BlindReceive}^0}(\mathcal{G}, pp) = 1] - \Pr[\mathcal{A}^{\mathcal{O}.\text{BlindReceive}^1}(\mathcal{G}, pp) = 1] \right|$$

is negligible, where $\mathcal{G} \leftarrow \text{GenSetting}(1^\lambda)$, $pp \leftarrow \text{Setup}(\mathcal{G})$ and

```

 $\mathcal{O}.\text{BlindReceive}^b(pk, K, \vec{m}^{(\text{iss})}, \vec{m}^{(0)}, \vec{m}^{(1)})$ 
assert  $\vec{m}^{(\text{iss})} \succeq \vec{m}^{(0)}$  and  $\vec{m}^{(\text{iss})} \succeq \vec{m}^{(1)}$ 
 $(C, o) \leftarrow \text{Commit}(pp, pk, K, \vec{m}^{(\text{iss})}; \vec{m}^{(b)})$ 
Hand  $C$  to  $\mathcal{A}$ 
 $\sigma' \leftarrow \text{BlindReceive}(pp, pk, K, \vec{m}^{(\text{iss})}, C; \vec{m}^{(b)}, o) \leftrightarrow \mathcal{A}$ 
return  $\text{Verify}(pp, pk, \vec{m}^{(b)}, \sigma')$ 

```

Note that this definition allows for multiple concurrent copies of the protocol. The oracle is a typical left-or-right oracle, where either message $\vec{m}^{(0)}$ or message $\vec{m}^{(1)}$ is signed.

Regarding anonymity for the signer, we demand that no adversarial receiver can distinguish which of two signatures σ_{iss} is used to run the blind signing protocol.

Definition 6.10 (Perfect signer anonymity). Let $\mathcal{A}$ be an algorithm. A dynamically malleable signature scheme Π_{dms} has *perfect signer anonymity* if for all $n, \lambda \in \mathbb{N}$, $\mathcal{G} \in [\text{GenSetting}(1^\lambda)]$, $pp \in [\text{Setup}(\mathcal{G})]$, all $pk, \vec{m}^{(\text{iss})}, \sigma_{\text{iss}}, \sigma'_{\text{iss}}$ with $\text{Verify}(pp, pk, \vec{m}^{(\text{iss})}, \sigma_{\text{iss}}) = \text{Verify}(pp, pk, \vec{m}^{(\text{iss})}, \sigma'_{\text{iss}}) = 1$, all algorithms $\mathcal{A}, \mathcal{B}$ and all v

$$\Pr \left[(K, \text{view}_{\mathcal{A}}, \text{view}_{\mathcal{B}}) = v \mid \begin{array}{l} (K, k) \leftarrow \text{Precommit}(pp, pk, \vec{m}^{(\text{iss})}, \sigma_{\text{iss}}) \\ (C, st) \leftarrow \mathcal{A}(K) \\ \text{BlindSign}(pp, pk, K, \vec{m}^{(\text{iss})}, C; \sigma_{\text{iss}}, k) \leftrightarrow \mathcal{B}(st) \end{array} \right] \\ = \Pr \left[(K, \text{view}_{\mathcal{A}}, \text{view}_{\mathcal{B}}) = v \mid \begin{array}{l} (K, k) \leftarrow \text{Precommit}(pp, pk, \vec{m}^{(\text{iss})}, \sigma'_{\text{iss}}) \\ (C, st) \leftarrow \mathcal{A}(K) \\ \text{BlindSign}(pp, pk, K, \vec{m}^{(\text{iss})}, C; \sigma'_{\text{iss}}, k) \leftrightarrow \mathcal{B}(st) \end{array} \right]$$

A discussion around the level of privacy this affords can be found around Definition 6.4.

6.3.4 Construction

We discuss the intuition for our construction of DMS, which is an improved variant of previous work [BB18]. Consider a Pointcheval-Sanders signature (Construction 5.10)

$$(h, h^{x+\sum_i y_i m_i}).$$

In this signature scheme, there is no malleability of messages. Our main insight for making the index i message m_i malleable is to add the value h^{y_i} to the signature. Intuitively, using h^{y_i} , one can additively modify the $h^{y_i m_i}$ part of the signature: Multiplying $h^{x+\sum_i y_i m_i}$ in the signature with $(h^{y_i})^{m'_i}$ results in a valid signature on the message $(m_1, \dots, (m_i + m'_i), \dots, m_n)$. Furthermore, this ability does not impact the integrity of messages m_j for $j \neq i$.

An important aspect of this idea is that the added malleability-enabler values h^{y_i} can be randomized alongside the rest of the signature to a new random base $h' = h^r$ (i.e., the malleability-enabler for the randomized signature is simply $(h^{y_i})^r = (h')^{y_i}$). To remove the ability to modify m_i , one would simply throw away h^{y_i} and randomize the signature without it. The randomization of derived signatures is crucial: It ensures that the h^{y_i} terms for one signature are useless for modifying the messages within other signatures (for which one might need $(h')^{y_i}$ to another random basis h').

Armed with this insight, we can construct our DMS scheme: A signature on $\vec{m}$ is of the form

$$(h, h^{x+\sum_{i: m_i \neq \star} y_i m_i}, (h^{y_i})_{i: m_i = \star})$$

for a random $h \leftarrow \mathbb{G}_1 \setminus \{1\}$. Note that for positions where $m_i = \star$, we provide the malleability-enabler h^{y_i} (and we simply remove m_i part from the sum in the original Pointcheval-Sanders part of the signature). To derive a signature on a message where previously $m_i = \star$ and the new $m'_i \neq \star$, we simply use the h^{y_i} value to modify the sum original Pointcheval-Sanders part of the signature to include $y_i m'_i$ in the sum. We then conceptually remove h^{y_i} from the signature and re-randomize it by raising the whole signature to the power of a random $r \in \mathbb{Z}_p^*$.

For an intuition for the protocol for signing committed values, assume that only one message $m_i = \star$ is malleable and the recipient wants a signature on some committed value $m'_i \in \mathbb{Z}_p$. Intuitively, the signer first randomizes her signature and then sends her h, h^{y_i} values of the randomized signature to the receiver (this is the commitment key). Then the receiver computes the (Pedersen) commitment to m'_i as $C_{\text{fix}} = h^{o_{\text{fix}}} \cdot (h^{y_i})^{m'_i}$ for random $o_{\text{fix}} \in \mathbb{Z}_p$. This allows the signer to homomorphically modify her signature by multiplying the $h^{x+\sum_j y_j m_j}$ part with C_{fix} , adding the term $y_i m'_i$ to the sum in the exponent (it also introduces an o_{fix} error term to the signature, but that can be removed by the receiver later). The signer then randomizes the homomorphically modified signature again (to a new base h') and sends it to the receiver.

The full protocol is more involved: Improving over our previous work [BB18], we want to allow the receiver to commit to $m'_i = \star$ as well (instead of just $m'_i \in \mathbb{Z}_p$), in which case we want him to learn the $(h')^{y_i}$ term of the final signature. However, due to blindness, the signer must not learn whether or not she reveals $(h')^{y_i}$ to the receiver. Implementing

this turns out to be relatively simple: The receiver either sends $h^{y_i} \cdot h^{o_i}$ or just h^{o_i} to the signer as part of his commitment. The signer randomizes the received value together with the signature, i.e., she sends $(h')^{y_i} \cdot (h')^{o_i}$ or just $(h')^{o_i}$ to the receiver. In the first case, the receiver can unblind the received value and learn $(h')^{y_i}$, in the second case he does not learn anything about $(h')^{y_i}$.

Armed with these intuitions, the construction of DMS based on Pointcheval-Sanders signatures works as follows.

Construction 6.11 (PS-based DMS). The Pointcheval-Sanders-based dynamically malleable signature scheme $\Pi_{\text{ps-dms}}$ has message space $\mathbb{M}_{(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, e, p)} = \mathbb{Z}_p$. It consists of the following algorithms.

Setup($\mathcal{G}$) $\rightarrow pp$

Outputs $pp = (\mathcal{G}, g, \tilde{g})$, where $g \leftarrow \mathbb{G}_1 \setminus \{1\}, \tilde{g} \leftarrow \mathbb{G}_2 \setminus \{1\}$.

Keygen($pp, 1^n$) $\rightarrow (pk, \sigma_{\text{root}})$

Chooses the (implicit) secret key $sk = (x, y_1, \dots, y_n) \leftarrow \mathbb{Z}_p^{n+1}$ and derives the public key $pk = (\tilde{X}, (Y_i)_{i=1}^n, (\tilde{Y}_i)_{i=1}^n) = (\tilde{g}^x, (g^{y_i})_{i=1}^n, (\tilde{g}^{y_i})_{i=1}^n) \in \mathbb{G}_2 \times \mathbb{G}_1^n \times \mathbb{G}_2^n$, and the root signature $\sigma_{\text{root}} = (g, g^x, (g^{y_i})_{i \in [n]})$.

Sign($pp, pk, \sigma_{\text{iss}}, \vec{m}^{(\text{iss})}, \vec{m}'$) $\rightarrow \sigma'$

For $\sigma_{\text{iss}} = (h_{\text{iss}}, \mu_{\text{iss}}, (\gamma_{\text{iss}, i})_{i: m_i^{(\text{iss})} = \star})$, chooses a random $r \leftarrow \mathbb{Z}_p^*$ and outputs

$$\sigma' = \left(h_{\text{iss}}, \mu_{\text{iss}} \cdot \prod_{\substack{i: m_i^{(\text{iss})} = \star \\ \wedge m'_i \neq \star}} \gamma_{\text{iss}, i}^{m'_i}, (\gamma_{\text{iss}, i})_{i: m'_i = \star} \right)^r$$

(Note that all vector components are raised to the power of r .)

💡 The resulting signature is of the form

$$\sigma' = (h', \mu', \vec{\gamma}') = \left(h, h^{x + \sum_{i: m'_i \neq \star} m'_i y_i}, (h^{y_i})_{i: m'_i = \star} \right),$$

where h is distributed uniformly in $\mathbb{G}_1 \setminus \{1\}$. μ' encodes the “fixed” messages $m'_i \neq \star$, and for each $m'_i = \star$ message, we get the corresponding γ'_i .

Verify($pp, pk, \vec{m}, (h, \mu, \vec{\gamma})$) $= b$

Outputs $b = 1$ iff $h \neq 1$ and $e(\mu, \tilde{g}) = e(h, \tilde{X} \cdot \prod_{i: m_i \neq \star} \tilde{Y}_i^{m_i})$ and $e(\gamma_i, \tilde{g}) = e(h, \tilde{Y}_i)$ for all i with $m_i = \star$. 💡 This check is equivalent to $(h, \mu, \vec{\gamma})$ being of the form $(h, h^{x + \sum_{i: m_i \neq \star} y_i m_i}, (h^{y_i})_{i: m_i = \star})$ for some $h \neq 1$.

Precommit($pp, pk, \vec{m}^{(\text{iss})}, \sigma_{\text{iss}}$) $\rightarrow (K, k)$

For signature $\sigma_{\text{iss}} = (h_{\text{iss}}, \mu_{\text{iss}}, \vec{\gamma}_{\text{iss}})$, chooses $k \leftarrow \mathbb{Z}_p^*$ and sets commitment key $K =$

$(K_0, (K_i)_{i: m_i^{(\text{iss})} = \star}) = (h_{\text{iss}}^k, (\gamma_{\text{iss}, i}^k)_{i: m_i^{(\text{iss})} = \star})$. It outputs (K, k) . ⚡ The commitment key K is of the form $K = (h, (h^{y_i})_{i: m_i^{(\text{iss})} = \star})$ for a random $h \in \mathbb{G}_1 \setminus \{1\}$.

Commit($pp, pk, K, \vec{m}^{(\text{iss})}; \vec{m}'$) $\rightarrow (C, o)$

For commitment key $K = (K_0, (K_i)_{i: m_i^{(\text{iss})} = \star}) \in (\mathbb{G}_1 \setminus \{1\})^{1+|\{i: m_i^{(\text{iss})} = \star\}|}$ chooses randomness $(o_{\text{fix}}, (o_i)_{i: m_i^{(\text{iss})} = \star}) \leftarrow \mathbb{Z}_p^{1+|\{i: m_i^{(\text{iss})} = \star\}|}$, computes a commitment to the newly fixed messages

$$C_{\text{fix}} = K_0^{o_{\text{fix}}} \cdot \prod_{i: m_i^{(\text{iss})} = \star \wedge m'_i \neq \star} K_i^{m'_i}$$

⚡ C_{fix} encodes the changes to be made to μ_{iss} , namely the messages that go from $m_i^{(\text{iss})} = \star$ to some fixed message $m'_i \neq \star$.

and terms for the new “ $\star$ ” positions, for i with $m_i^{(\text{iss})} = \star$,

$$C_i = \begin{cases} K_i \cdot K_0^{o_i} & \text{if } m'_i = \star \\ K_0^{o_i} & \text{if } m'_i \neq \star \end{cases}$$

⚡ Looking ahead, having the K_i term in C_i will allow the user to learn γ_i' .

It outputs $C = (C_{\text{fix}}, (C_i)_{i: m_i^{(\text{iss})} = \star})$ and $o = (o_{\text{fix}}, (o_i, \omega_i, s_i)_{i: m_i^{(\text{iss})} = \star})$, where

$$(\omega_i, s_i) = \begin{cases} (0, 1) & \text{if } m'_i = \star \\ (m'_i, 0) & \text{if } m'_i \neq \star \end{cases}$$

⚡ (ω_i, s_i) are chosen such that $C_{\text{fix}} = K_0^{o_{\text{fix}}} \cdot \prod_{i: m_i^{(\text{iss})} = \star} K_i^{\omega_i}$ and $C_i = K_i^{s_i} \cdot K_0^{o_i}$, as required by VerifyCommit below.

VerifyCommit($pp, pk, K, \vec{m}^{(\text{iss})}, C; \vec{m}', o$) $= b$

For $K = (K_0, (K_i)_{i: m_i^{(\text{iss})} = \star}) \in (\mathbb{G}_1 \setminus \{1\})^{1+|\{i: m_i^{(\text{iss})} = \star\}|}$, $C = (C_{\text{fix}}, (C_i)_{i: m_i^{(\text{iss})} = \star})$, and $o = (o_{\text{fix}}, (o_i, \omega_i, s_i)_{i: m_i^{(\text{iss})} = \star})$ outputs 1 iff the following checks succeed:

$$C_{\text{fix}} = K_0^{o_{\text{fix}}} \cdot \prod_{i: m_i^{(\text{iss})} = \star} K_i^{\omega_i}$$

⚡ The equation above allows for K_i terms for i where $m'_i = \star$, whereas Commit will never include such terms.

and for all i with $m_i^{(\text{iss})} = \star$,

$$C_i = K_i^{s_i} \cdot K_0^{o_i}$$

💡 The equation above allows for $s_i \notin \{0, 1\}$, whereas Commit will always set $s_i \in \{0, 1\}$.

and

$$s_i = 0 \Leftrightarrow m'_i \neq \star \quad \text{and} \quad m'_i \neq \star \Rightarrow \omega_i = m'_i$$

💡 We view $(\omega_i, s_i) \in \mathbb{Z}_p^2$ as a redundant encoding of m'_i : The value s_i encodes whether $m'_i = \star$ (any non-zero s_i value means $m'_i = \star$) and if $m'_i \neq \star$, then ω_i encodes m'_i (otherwise, ω_i is arbitrary).

BlindSign $(pp, pk, K, \vec{m}^{(\text{iss})}, C; \sigma_{\text{iss}}, k) \leftrightarrow \text{BlindReceive}(pp, pk, K, \vec{m}^{(\text{iss})}, C; \vec{m}', o) \rightarrow \sigma'$
 For $\sigma_{\text{iss}} = (h_{\text{iss}}, \mu_{\text{iss}}, \vec{\gamma}_{\text{iss}})$, $C = (C_{\text{fix}}, (C_i)_{i: m_i^{(\text{iss})} = \star})$, $K = (K_0, (K_i)_{i: m_i^{(\text{iss})} = \star})$, and $o = (o_{\text{fix}}, (o_i, \omega_i, s_i)_{i: m_i^{(\text{iss})} = \star})$ works as follows.

- **BlindSign** chooses a random $r \leftarrow \mathbb{Z}_p^*$ and sends

$$\sigma_{\text{blind}} = (h_{\text{blind}}, \mu_{\text{blind}}, \vec{\gamma}_{\text{blind}}) = \left(h_{\text{iss}}^{rk}, (C_{\text{fix}} \cdot \mu_{\text{iss}}^k)^r, (C_i^r)_{i: m_i^{(\text{iss})} = \star} \right)$$

to the receiver. 💡 If keys, σ_{iss} , and C have been computed correctly, this value is of the form $(h, h^{o+x+\sum y_i m'_i}, (h^{y_i \delta_i \cdot o_i})_{i: m_i^{(\text{iss})} = \star})$, where $\delta_i = 1$ if $m'_i = \star$, otherwise $\delta_i = 0$. This means that the first two signature parts are consistent with a valid signature on m'_i (except for the o_{fix}, o_i dirt term). The last part, for i such that $m'_i = \star$, is consistent with a valid signature's $(\gamma_i)_{i: m'_i = \star} = (h^{y_i})_{i: m'_i = \star}$ (except for the o_i randomization). For those indices where the issuer's message is $m_i^{(\text{iss})} = \star$ but the receiver's message is $m'_i \neq \star$, this part contains useless dummy values h^{o_i} , which shall be discarded by the receiver.

- **BlindReceive** first checks that
 - $(h_{\text{blind}})^{o_i} = \gamma_{\text{blind}, i}$ for all i with $m'_i \neq \star$
 - $e(K_i, \tilde{g}) = e(K_0, \tilde{Y}_i)$ for all i with $m_i^{(\text{iss})} = \star$

and outputs $\perp$ if any check fails. 💡 This mitigates privacy issues that could otherwise arise from malformed $\gamma_{\text{blind}, i}$ or K_i (cf. Theorem 6.14). If the checks succeed, it computes and outputs

$$\sigma' = (h_{\text{blind}}, (h_{\text{blind}})^{-o_{\text{fix}}} \cdot \mu_{\text{blind}}, ((h_{\text{blind}})^{-o_i} \cdot \gamma_{\text{blind}, i})_{i: m'_i = \star})$$

💡 This removes the dirt terms.

Note that **VerifyCommit** accepts several openings that will never be output by **Commit**. Later, in Section 6.4.4, this leniency will play a pivotal role in efficiently proving knowledge of a commitment opening.

Perhaps surprisingly, the values Y_i in the public key are never used in the scheme. They are nevertheless included so that our security proofs cover that their presence does not harm security. The values Y_i can be used to compute a convincing commitment key K without having seen any signature, which is important when arguing about the unforgeability of the blind signing protocol (see Theorem 6.13).

$\Pi_{\text{ps-dms}}$ is *correct*: Let $\lambda, n \in \mathbb{N}$, $\mathcal{G} \in [\text{GenSetting}(1^\lambda)]$, $pp \in [\text{Setup}(\mathcal{G})]$, $(pk, \sigma_{\text{root}}) \in [\text{Keygen}(pp, 1^n)]$, $\sigma_{\text{iss}} \in \{0, 1\}^*$, $\vec{m}^{(\text{iss})}, \vec{m}' \in (\mathbb{M}_{\mathcal{G}} \cup \{\star\})^n$. Let $x, y_1, \dots, y_n$ denote the implicit secret key for pk , i.e., $pk = (\tilde{g}^x, (g^{y_i})_{i=1}^n, (\tilde{g}^{y_i})_{i=1}^n)$. Note that for all $\vec{m}, \sigma$, we get $\text{Verify}(pp, pk, \vec{m}, \sigma = (h, \mu, \vec{\gamma}))$ if and only if $\sigma = (h, h^{x + \sum_{i: m_i \neq \star} y_i m_i}, (h^{y_i})_{i: m_i = \star})$.

- σ_{root} output by Keygen is a valid signature on $\star^n$. $\sigma_{\text{root}} = (h, \mu, \vec{\gamma}) = (g, g^x, (g^{y_i})_{i \in [n]})$. Hence $e(\mu, \tilde{g}) = e(g, \tilde{g})^x = e(h, \tilde{X})$ and $e(\gamma_i, \tilde{g}) = e(g, \tilde{g})^{y_i} = e(h, \tilde{Y}_i)$ for all $i \in [n]$. So overall, $\text{Verify}(pp, pk, \star^n, \sigma_{\text{root}}) = 1$.
- Sign outputs valid signatures. Assume $\text{Verify}(pp, pk, \vec{m}^{(\text{iss})}, \sigma_{\text{iss}}) = 1$ and $\vec{m}^{(\text{iss})} \succeq \vec{m}'$. Consider signature σ' output by $\text{Sign}(pp, pk, \sigma_{\text{iss}}, \vec{m}^{(\text{iss})}, \vec{m}')$ and let $r \in \mathbb{Z}_p$ be the randomness used. Let $h' = h_{\text{iss}}^r$. It holds that

$$\sigma' = \left(h_{\text{iss}}, \quad \mu_{\text{iss}} \cdot \prod_{i: m_i^{(\text{iss})} = \star \wedge m'_i \neq \star} \gamma_{\text{iss}, i}^{m'_i}, \quad (\gamma_{\text{iss}, i})_{i: m'_i = \star} \right)^r \quad \text{By definition}$$

Using $\text{Verify}(pp, pk, \vec{m}^{(\text{iss})}, \sigma_{\text{iss}}) = 1$:

$$= \left(h_{\text{iss}}, \quad \frac{h_{\text{iss}}^{x + \sum_{i: m_i^{(\text{iss})} \neq \star} y_i m_i^{(\text{iss})}}}{h_{\text{iss}}^{x + \sum_{i: m_i^{(\text{iss})} \neq \star} y_i m'_i}} \cdot \prod_{i: m_i^{(\text{iss})} = \star \wedge m'_i \neq \star} h_{\text{iss}}^{y_i m'_i}, \quad (h_{\text{iss}}^{y_i})_{i: m'_i = \star} \right)^r$$

Using $m_i^{(\text{iss})} \neq \star \Rightarrow m'_i \neq \star$ and $m_i^{(\text{iss})} \neq \star \Rightarrow m_i^{(\text{iss})} = m'_i$ from $\vec{m}^{(\text{iss})} \succeq \vec{m}'$:

$$\begin{aligned} &= \left(h_{\text{iss}}, \quad h_{\text{iss}}^{x + \sum_{i: m_i^{(\text{iss})} \neq \star \wedge m'_i \neq \star} y_i m'_i} \cdot \prod_{i: m_i^{(\text{iss})} = \star \wedge m'_i \neq \star} h_{\text{iss}}^{y_i m'_i}, \quad (h_{\text{iss}}^{y_i})_{i: m'_i = \star} \right)^r \\ &= \left(h_{\text{iss}}, \quad h_{\text{iss}}^{x + \sum_{i: m'_i \neq \star} y_i m'_i}, \quad (h_{\text{iss}}^{y_i})_{i: m'_i = \star} \right)^r \end{aligned}$$

Using $h' = h_{\text{iss}}^r$ (as defined above):

$$= \left(h', \quad (h')^{x + \sum_{i: m'_i \neq \star} y_i m'_i}, \quad (h'^{y_i})_{i: m'_i = \star} \right)$$

As a consequence, $\text{Verify}(pp, pk, \vec{m}', \sigma') = 1$.

- The commitment procedure outputs valid commitments. Assume $\vec{m}^{(\text{iss})} \succeq \vec{m}'$. Let pk' be arbitrary and let $K = (K_0, (K_i)_{i: m_i^{(\text{iss})} = \star}) \in (\mathbb{G}_1 \setminus \{1\})^{1 + |\{i | m_i^{(\text{iss})} = \star\}|}$ (as parsed by Commit and VerifyCommit). Let $C = (C_{\text{fix}}, (C_i)_{i: m_i^{(\text{iss})} = \star})$ be output by Commit. Using the variables defined in Commit, we get

$$C_{\text{fix}} = K_0^{O_{\text{fix}}} \cdot \prod_{i: m_i^{(\text{iss})} = \star \wedge m'_i \neq \star} K_i^{m'_i}$$

Using $\omega_i = m'_i$ for i with $m'_i \neq \star$:

$$= K_0^{O_{\text{fix}}} \cdot \prod_{i: m_i^{(\text{iss})} = \star \wedge m'_i \neq \star} K_i^{\omega_i}$$

💡 Using that $\omega_i = 0$ for i with $m'_i = \star$:

$$= K_0^{o_{\text{fix}}} \cdot \prod_{i: m_i^{(\text{iss})} = \star} K_i^{\omega_i}$$

Hence C_{fix} is accepted by **VerifyCommit**.

Furthermore, we get for i with $m_i^{(\text{iss})} = \star$,

$$C_i = \begin{cases} K_i \cdot K_0^{o_i} & \text{if } m'_i = \star \\ K_0^{o_i} & \text{if } m'_i \neq \star \end{cases}$$

💡 Using $s_i = 1$ for $m'_i = \star$ and $s_i = 0$ for $m'_i \neq \star$:

$$= K_i^{s_i} \cdot K_0^{o_i}$$

Hence the C_i are accepted by **VerifyCommit**.

Finally, the “ $s_i = 0 \Leftrightarrow m'_i \neq \star$ and $m'_i \neq \star \Rightarrow \omega_i = m'_i$ ” check in **VerifyCommit** succeeds by definition of (ω_i, s_i) :

$$(\omega_i, s_i) = \begin{cases} (0, 1) & \text{if } m'_i = \star \\ (m'_i, 0) & \text{if } m'_i \neq \star \end{cases}$$

- 💡 The blind signing protocol outputs valid signatures. Assume $\text{Verify}(pp, pk, \vec{m}^{(\text{iss})}, \sigma_{\text{iss}}) = 1$ and $\vec{m}^{(\text{iss})} \succeq \vec{m}'$. Consider a signature σ' output by **BlindReceive**. Let $h' = h_{\text{iss}}^{rk}$. Using the variables defined in **Precommit**, **Commit**, **BlindSign**, **BlindReceive**, it holds that

$$\sigma' = (h_{\text{blind}}, \quad h_{\text{blind}}^{-o_{\text{fix}}} \cdot \mu_{\text{blind}}, \quad (h_{\text{blind}}^{-o_i} \cdot \gamma_{\text{blind}, i})_{i: m'_i = \star}) \quad \text{💡 By definition}$$

💡 Plugging in values sent by **BlindSign**:

$$= \left(h_{\text{iss}}^{rk}, \quad h_{\text{iss}}^{-rk o_{\text{fix}}} \cdot (C_{\text{fix}} \cdot \mu_{\text{iss}}^k)^r, \quad (h_{\text{iss}}^{-rk o_i} \cdot (C_i)^r)_{i: m'_i = \star} \right)$$

💡 Plugging in values for commitment (**Commit**):

$$= \left(h_{\text{iss}}^{rk}, \quad h_{\text{iss}}^{-rk o_{\text{fix}}} \cdot \left(K_0^{o_{\text{fix}}} \cdot \prod_{i: m_i^{(\text{iss})} = \star \wedge m'_i \neq \star} K_i^{m'_i} \cdot \mu_{\text{iss}}^k \right)^r, \quad (h_{\text{iss}}^{-rk o_i} \cdot (K_i \cdot K_0^{o_i})^r)_{i: m'_i = \star} \right)$$

💡 Plugging in values for commitment key (**Precommit**):

$$\begin{aligned} &= \left(h_{\text{iss}}^{rk}, \quad h_{\text{iss}}^{-rk o_{\text{fix}}} \cdot \left(h_{\text{iss}}^{k o_{\text{fix}}} \cdot \prod_{i: m_i^{(\text{iss})} = \star \wedge m'_i \neq \star} \gamma_{\text{iss}, i}^{k m'_i} \cdot \mu_{\text{iss}}^k \right)^r, \quad (h_{\text{iss}}^{-rk o_i} \cdot (\gamma_{\text{iss}, i}^k \cdot h_{\text{iss}}^{k o_i})^r)_{i: m'_i = \star} \right) \\ &= \left(h_{\text{iss}}^{rk}, \quad \left(\prod_{i: m_i^{(\text{iss})} = \star \wedge m'_i \neq \star} \gamma_{\text{iss}, i}^{k m'_i} \cdot \mu_{\text{iss}}^k \right)^r, \quad (\gamma_{\text{iss}, i}^{rk})_{i: m'_i = \star} \right) \\ &= \left(h_{\text{iss}}, \quad \left(\prod_{i: m_i^{(\text{iss})} = \star \wedge m'_i \neq \star} \gamma_{\text{iss}, i}^{m'_i} \cdot \mu_{\text{iss}} \right)^{rk}, \quad (\gamma_{\text{iss}, i})_{i: m'_i = \star} \right) \end{aligned}$$

💡 Using that σ_{iss} is valid:

$$= \left(h_{\text{iss}} , \left(\prod_{i: m_i^{(\text{iss})} = \star \wedge m'_i \neq \star} \frac{h_{\text{iss}}^{y_i m'_i}}{h_{\text{iss}}} \cdot \frac{x + \sum_{i: m_i^{(\text{iss})} \neq \star} y_i m_i^{(\text{iss})}}{h_{\text{iss}}} \right) , \quad (h_{\text{iss}}^{y_i})_{i: m'_i = \star} \right)^{rk}$$

💡 Using $m_i^{(\text{iss})} \neq \star \Rightarrow m'_i \neq \star$ and $m_i^{(\text{iss})} \neq \star \Rightarrow m_i^{(\text{iss})} = m'_i$ from $\vec{m}^{(\text{iss})} \succeq \vec{m}'$:

$$= \left(h_{\text{iss}} , \left(\prod_{i: m_i^{(\text{iss})} = \star \wedge m'_i \neq \star} h_{\text{iss}}^{y_i m'_i} \cdot h_{\text{iss}}^{x + \sum_{i: m_i^{(\text{iss})} \neq \star \wedge m'_i \neq \star} y_i m'_i} \right) , \quad (h_{\text{iss}}^{y_i})_{i: m'_i = \star} \right)^{rk}$$

$$= \left(h_{\text{iss}} , \left(h_{\text{iss}}^{x + \sum_{i: m'_i \neq \star} y_i m'_i} \right) , \quad (h_{\text{iss}}^{y_i})_{i: m'_i = \star} \right)^{rk}$$

💡 Using $h' = h_{\text{iss}}^{rk}$ (as defined above)

$$= \left(h' , \left((h')^{x + \sum_{i: m'_i \neq \star} y_i m'_i} \right) , \quad (h'^{y_i})_{i: m'_i = \star} \right)$$

As a consequence, $\text{Verify}(pp, pk, \vec{m}', \sigma') = 1$.

We now prove unforgeability of the basic signature scheme (without the blind signing protocol). The proof is in the generic group model, as described in Section 2.10. The generic group model was originally conceived to justify the hardness of assumptions, rather than the security of a concrete construction. However, it is not unusual for signature schemes to be proven secure directly in the generic group model (e.g., [FHS19]). We skip the step of rephrasing the unforgeability experiment as an assumption (this artificial step is present, for example, in the single-message construction of the original Pointcheval-Sanders signature scheme [PS16]) and instead directly prove that generic adversaries cannot break unforgeability.

Theorem 6.12 ($\Pi_{\text{ps-dms}}$ unforgeability). $\Pi_{\text{ps-dms}}$ (Construction 6.11) is existentially unforgeable under chosen-message attacks (Definition 6.7) in the generic group model (Section 2.10).

The proof boils down to symbolic analysis in the GGM (as described around Theorem 2.26), essentially arguing that because the adversary is restricted to linear operations (Lemma 2.28), any valid signature it outputs must be one that can be derived trivially from previously queried signatures.

A similar proof is sketched in our original paper [BB18]. As a minor difference, the original construction does not contain the Y_i values in pk and its notion of malleability is slightly different (malleability of index i was modeled via set membership $i \in I$ in a set of malleable indices rather than modeled as $m_i = \star$). The major difference is the added detail around the generic group model.

Proof of Theorem 6.12. The proof starts by simplifying the signing oracles of Definition 6.7, continues by modeling the unforgeability game (with simplified oracles) in the GGM, and finally conducts symbolic analysis in the GGM (Section 2.10) to bound the

adversary's forging advantage.

Simplifying the oracles. Let $n \in \mathbb{N}$. First, we observe that for Construction 6.11 specifically, we can simplify the adversary's oracles because the distribution of signatures output by **Sign** does not depend on which signature is used to compute the new signature. Consider the following simplified signing oracle $\mathcal{O}.\text{Sign}(\vec{m}^{(k)})$:

```

 $\mathcal{O}.\text{Sign}(\vec{m}^{(k)})$ 
Add  $\vec{m}^{(k)}$  to  $\mathcal{T}_{\text{signed}}$ 
Increment query counter  $k$ 
//Create signature using trapdoors  $x, y_i$ :
Set  $h^{(k)} = g^{r^{(k)}}$  for  $r^{(k)} \leftarrow \mathbb{Z}_p^*$ 
Set  $\mu^{(k)} = g^{r^{(k)} \cdot (x + \sum_{i: m_i^{(k)} \neq \star} m_i^{(k)} y_i)}$ 
Set  $\gamma_i^{(k)} = g^{r^{(k)} \cdot y_i}$  for  $i$  with  $m_i^{(k)} = \star$ 
return  $\sigma' = (h^{(k)}, \mu^{(k)}, \vec{\gamma}^{(k)})$ 

```

where $x, y_1, \dots, y_n \in \mathbb{Z}_p$ are as chosen during **Keygen** and k is a counter for $\mathcal{O}.\text{Sign}$ queries (initially, $k = 0$).

By inspection, it is easy to see that signatures generated by $\mathcal{O}.\text{Sign}(\vec{m}^{(k)})$ follow exactly the same distribution as in the original $\mathcal{O}.\text{Sign}(\mathcal{S}, \vec{m}^{(k)})$ oracle (Definition 6.7) where $\mathcal{S}$ points to *any* honestly generated signature $\sigma_{\mathcal{S}}$ on some $\vec{m}_{\mathcal{S}} \succeq \vec{m}^{(k)}$.

More concretely, we can take any adversary $\mathcal{A}'$ against the original unforgeability definition for $\Pi_{\text{ps-dms}}$ and construct an adversary $\mathcal{A}^{\mathcal{O}.\text{Sign}(\cdot)}$ that only queries the oracle specified above. $\mathcal{A}$ behaves exactly like $\mathcal{A}'$ except that: When $\mathcal{A}'$ queries $\mathcal{O}.\text{PrivateSign}(\mathcal{S}, \vec{m}')$, $\mathcal{A}$ simply generates a new signature handle S' , remembers $\vec{m}_{S'} = \vec{m}'$, and returns S' to $\mathcal{A}'$ (note that no secrets are involved in this process, $\mathcal{A}$ does not ask for a signature on $\vec{m}'$). When $\mathcal{A}'$ queries $\mathcal{O}.\text{Sign}(\mathcal{S}, \vec{m}')$, $\mathcal{A}$ checks that $\vec{m}_{\mathcal{S}} \neq \perp$ and $\vec{m}_{\mathcal{S}} \succeq \vec{m}'$ and returns $\perp$ if the check fails. Otherwise, $\mathcal{A}$ queries its oracle $\sigma \leftarrow \mathcal{O}.\text{Sign}(\vec{m}')$ and returns σ to $\mathcal{A}'$.

It is easy to see that $\mathcal{A}$ and $\mathcal{A}'$ have the exact same probability of winning their respective flavors of the unforgeability game. In the remainder of this proof, we will only consider an adversary $\mathcal{A}$ that uses the simplified signing oracle.

Phrasing the problem as a game in the GGM. Let $p \in \mathbb{N}$ be prime. Let $\mathcal{Y}_1, \mathcal{Y}_2, \mathcal{Y}_t \subset \{0, 1\}^*$ be arbitrary sets of size p . Let $\mathcal{A}$ be a generic algorithm (against the unforgeability experiment with simplified signing oracle) that makes at most q_{sign} queries to $\mathcal{O}.\text{Sign}$ and at most $q_{\mathcal{A}}$ queries to the GGM oracles $\mathcal{O}.\text{op}_z$ or $\mathcal{O}.\text{pair}$.

For this proof, the hidden variables X_i in Figure 2.3 are $(x, y_1, \dots, y_n, r^{(1)}, \dots, r^{(q_{\text{sign}})})$.  The variables represent the secret key $x, y_1, \dots, y_n$ and randomness $r^{(k)}$ used for the k th signature query. Hence we work over the polynomial ring $\mathbb{Z}_p[x, y_1, \dots, y_n, r^{(1)}, \dots, r^{(q_{\text{sign}})}]$. Let $(\mathcal{X})_{i=1}^{n+1+q_{\text{sign}}} = (\mathbb{Z}_p^{n+1}, (\mathbb{Z}_p^*)^{q_{\text{sign}}})$.  The secret key values are chosen from $\mathbb{Z}_p$, the $r^{(k)}$ values from $\mathbb{Z}_p^*$. We construct the generic algorithm $\mathcal{B}$ designed to run in the context

of the Figure 2.3 GGM experiment $\text{GGM}_{\mathcal{B}}^{\text{hiddenVars}, X}(\mathcal{Y}_1, \mathcal{Y}_2, \mathcal{Y}_t, p, \mathcal{X}_1, \dots, \mathcal{X}_{n+q_{\text{sign}}+1})$. $\mathcal{B}$ models the generic unforgeability experiment (with the simplified oracle described above) as follows.

- $\mathcal{B}^{\mathcal{O}.\text{op}_1, \mathcal{O}.\text{op}_2, \mathcal{O}.\text{op}_t, \mathcal{O}.\text{pair}, \mathcal{O}.\text{compute}_1, \mathcal{O}.\text{compute}_2, \mathcal{O}.\text{compute}_t}(g, \tilde{g})$ computes the public key pk by querying $\tilde{X} \leftarrow \mathcal{O}.\text{compute}_2(x)$, $Y_i \leftarrow \mathcal{O}.\text{compute}_1(y_i)$, and $\tilde{Y}_i \leftarrow \mathcal{O}.\text{compute}_2(y_i)$ for $i \in [n]$ and setting $pk = (\tilde{X}, (Y_i)_{i=1}^n, (\tilde{Y}_i)_{i=1}^n)$.
- It then hands pk to $\mathcal{A}^{\mathcal{O}.\text{op}_1, \mathcal{O}.\text{op}_2, \mathcal{O}.\text{op}_t, \mathcal{O}.\text{pair}}(g, \tilde{g})$.
- Whenever $\mathcal{A}$ queries its GGM oracles $\mathcal{O}.\text{op}_z$ or $\mathcal{O}.\text{pair}$, $\mathcal{B}$ just relays the query to its own corresponding oracles and gives the result back to $\mathcal{A}$.
- When $\mathcal{A}$ queries its $\mathcal{O}.\text{Sign}(\vec{m}^{(k)})$ oracle for the k th time, $\mathcal{B}$ responds with the signature
 - $h^{(k)} = \mathcal{O}.\text{compute}_1(r^{(k)})$
 - $\mu^{(k)} = \mathcal{O}.\text{compute}_1(r^{(k)}) \cdot (x + \sum_{i: m_i^{(k)} \neq \star} m_i^{(k)} y_i)$
 - $\gamma_i^{(k)} = \mathcal{O}.\text{compute}_1(r^{(k)}) \cdot y_i$ for i with $m_i^{(k)} = \star$
- Eventually, $\mathcal{A}$ outputs a candidate forgery $(\vec{m}^*, (h^*, \mu^*, \vec{\gamma}^*))$.
- $\mathcal{B}$ checks signature validity (as in Construction 6.11) and non-triviality conditions (as in Definition 6.7):
 - $h^* \neq \mathcal{O}.\text{compute}_1(0)$
 $\heartsuit$ Equivalent to: $\mu_1^{-1}(h^*) \neq 0$, where μ_1 is the GGM encoding function.
 - $\mathcal{O}.\text{pair}(\mu^*, \tilde{g}) = \mathcal{O}.\text{pair}(h, \mathcal{O}.\text{compute}_2(x + \sum_{i: m_i^* \neq \star} m_i^* y_i))$
 $\heartsuit$ Equivalent to: $\mu_1^{-1}(\mu^*) = \mu_1^{-1}(h^*) \cdot (x + \sum_{i: m_i^* \neq \star} m_i^* y_i)$.
 - $\mathcal{O}.\text{pair}(\gamma_i^*, \tilde{g}) = \mathcal{O}.\text{pair}(h^*, \tilde{Y}_i)$ for all i with $m_i^* = \star$
 $\heartsuit$ Equivalent to: $\mu_1^{-1}(\gamma_i^*) = \mu_1^{-1}(h^*) \cdot y_i$ for $m_i^* = \star$.
 - There is no $\vec{m}^{(k)} \in \mathcal{T}_{\text{signed}}$ with $\vec{m}^{(k)} \succeq \vec{m}^*$ $\heartsuit$ This is the nontriviality condition, requiring that a signature on $\vec{m}^*$ shall not be legally derivable from queried signatures.

$\mathcal{B}$ outputs 1 iff all conditions are met.

Note that $\text{GGM}_{\mathcal{B}}^{\text{hiddenVars}, \text{REAL}}(\mathcal{Y}_1, \mathcal{Y}_2, \mathcal{Y}_t, p, \mathcal{X}_1, \dots, \mathcal{X}_{n+q_{\text{sign}}+1})$ faithfully models the unforgeability experiment with simplified signing oracle in the generic group model. Applying Theorem 2.26, we get

$$\left| \begin{array}{l} \Pr[\text{GGM}_{\mathcal{B}}^{\text{hiddenVars}, \text{REAL}}(\mathcal{Y}_1, \mathcal{Y}_2, \mathcal{Y}_t, p, \mathcal{X}_1, \dots, \mathcal{X}_{n+q_{\text{sign}}+1}) = 1] \\ - \Pr[\text{GGM}_{\mathcal{B}}^{\text{hiddenVars}, \text{IDEAL}}(\mathcal{Y}_1, \mathcal{Y}_2, \mathcal{Y}_t, p, \mathcal{X}_1, \dots, \mathcal{X}_{n+q_{\text{sign}}+1}) = 1] \end{array} \right| \leq \frac{q^2 \cdot (9d_{\max} + 12)}{p - 1},$$

where the maximum number of GGM queries of $\mathcal{B}$ is $q - 2 := q_{\mathcal{A}} + 3q_{\text{sign}} + 4n + 5$ and the degree of $\mathcal{O}.\text{compute}$ queries is at most $d_{\max} = 2$.

Symbolic analysis in the GGM. We now analyze $\Pr[\text{GGM}_{\mathcal{B}}^{\text{hiddenVars, IDEAL}}(\mathcal{Y}_1, \mathcal{Y}_2, \mathcal{Y}_t, p, \mathcal{X}_1, \dots, \mathcal{X}_{n+q_{\text{sign}}+1}) = 1]$. As described in Section 2.10, in this setting, the variables $x, y_1, \dots, y_n, r^{(1)}, \dots, r^{(q_{\text{sign}})}$ are *polynomial variables*, i.e., they are treated purely symbolically. This greatly simplifies analysis of $\mathcal{B}$. The encoding functions μ_z map polynomials $f \in \mathbb{Z}_p[x, y_1, \dots, y_n, r^{(1)}, \dots, r^{(q_{\text{sign}})}]$ to random encodings $\mu_z(f) \in \mathcal{Y}_z$. However, to ease notation, for any encoding $h \in \mathcal{Y}_z$ we simply write “ h ” to refer to the corresponding polynomial $\mu_z^{-1}(h)$. Consider the candidate forgery $(\vec{m}^*, (h^*, \mu^*, \vec{\gamma}^*))$ output by $\mathcal{A}$. Assuming the signature output is valid, we get the following facts from the verification equations checked by $\mathcal{B}$:

- (1) $h^* \neq 0$ $\heartsuit$ This is shorthand notation for $\mu_1^{-1}(h^*) \neq 0$, which is an inequality of polynomials, as noted above.
- (2) $\mu^* = h^* \cdot (x + \sum_{i: m_i^* \neq \star} m_i^* y_i)$
- (3) $\gamma_i^* = h^* \cdot y_i$ for i with $m_i^* = \star$

As observed in Lemma 2.28, the adversary $\mathcal{A}$ (and $\mathcal{B}$) can only produce linear combinations of values queried to $\mathcal{O}.\text{compute}$. Considering the queries that $\mathcal{B}$ makes to $\mathcal{O}.\text{compute}_1$, this means that there are functions a, b, c, d, e mapping to $\mathbb{Z}_p$ such that we can write any polynomial $G \in \text{preim}(\mu_1) \subset \mathbb{Z}_p[x, y_1, \dots, y_n, r^{(1)}, \dots, r^{(q_{\text{sign}})}]$ as

$$\begin{aligned}
 G &= \sum_{k=1}^{q_{\text{sign}}} a(G, k) \cdot r^{(k)} && \heartsuit \text{ Terms from } h^{(k)} \\
 &+ \sum_{k=1}^{q_{\text{sign}}} b(G, k) \cdot r^{(k)} \left(x + \sum_{i: m_i^{(k)} \neq \star} m_i^{(k)} \cdot y_i \right) && \heartsuit \text{ Terms from } \mu^{(k)} \\
 &+ \sum_{k=1}^{q_{\text{sign}}} \sum_{i: m_i^{(k)} = \star} c(G, k, i) \cdot r^{(k)} y_i && \heartsuit \text{ Terms from } \gamma_i^{(k)} \\
 &+ \sum_{i=1}^n d(G, i) \cdot y_i && \heartsuit \text{ Terms from } pk \\
 &+ e(G) && \heartsuit \text{ Constant term (cf. Theorem 2.26)}
 \end{aligned}$$

From the verification equation, we know that $\mu^* = h^* \cdot (x + \sum_{i: m_i^* \neq \star} m_i^* y_i)$ (Fact (2)). With this equation, we can make the following inferences. There are no $r^{(k)} x^2$ terms in μ^* , hence h^* cannot contain an $r^{(k)} x$ term, meaning that $b(h^*, k) = 0$ for all k . Similarly, there are no $r^{(k)} y_i \cdot r^{(k')} x$ or $y_i \cdot r^{(k')} x$ or x terms in μ^* , so $c(h^*, k, i) = 0$ and $d(h^*, i) = 0$ and $e(h^*) = 0$ for all k, i . Plugging in the zeros, we get the following fact.

- (4) $h^* = \sum_{k=1}^{q_{\text{sign}}} a(h^*, k) \cdot r^{(k)}$ $\heartsuit$ h^* is a linear combination of other $h^{(k)}$. We note that this expression cannot be simplified further: h^* may indeed contain multiple different $r^{(k)}$ terms

from different queries k . For example, the sum (group operation) of two signatures on the same message yields another valid signature on the same message.

Given Fact (4), we also know that there are no (nonzero) constant terms or $r^{(k)}$ terms or y_i terms in $h^* \cdot (x + \sum_{i: m_i^* \neq \star} m_i^* y_i)$. For μ^* on the left-hand side of Fact (2), this implies $e(\mu^*) = 0$ and $a(\mu^*, k) = 0$ and $d(\mu^*, i) = 0$ for all k, i . Again, plugging in those zeros, we get the following.

$$(5) \quad \mu^* = \sum_{k=1}^{q_{\text{sign}}} \left(b(\mu^*, k) \cdot r^{(k)} \left(x + \sum_{i: m_i^{(k)} \neq \star} m_i^{(k)} \cdot y_i \right) + \sum_{i: m_i^{(k)} = \star} c(\mu^*, k, i) \cdot r^{(k)} y_i \right)$$

💡 μ^* is a linear combination of $\mu^{(k)}$ and $\gamma_i^{(k)}$ (which is what one would expect with regard to controlled malleability).

Putting together Facts (4), (5), and (2), we get the following.

(6) 💡 Verification equation $\mu^* = h^* \cdot (x + \sum_{i: m_i^* \neq \star} m_i^* y_i)$ with Facts (4), (5) plugged in:

$$\begin{aligned} & \sum_{k=1}^{q_{\text{sign}}} \left(b(\mu^*, k) \cdot r^{(k)} \left(x + \sum_{i: m_i^{(k)} \neq \star} m_i^{(k)} \cdot y_i \right) + \sum_{i: m_i^{(k)} = \star} c(\mu^*, k, i) \cdot r^{(k)} y_i \right) \\ &= \sum_{k=1}^{q_{\text{sign}}} a(h^*, k) \cdot r^{(k)} \left(x + \sum_{i: m_i^* \neq \star} m_i^* \cdot y_i \right) \end{aligned}$$

Because the polynomials are equal, their coefficients for the $r^{(k)}x$ terms must be equal. Hence $b(\mu^*, k) = a(h^*, k)$ for all $k \in [q_{\text{sign}}]$. Additionally, their coefficients for the $r^{(k)}y_i$ terms must be equal. For i, k where $m_i^{(k)} \neq \star$, the coefficients of $r^{(k)}y_i$ are $b(\mu^*, k)m_i^{(k)} = a(h^*, k)m_i^{(k)}$ on the left-hand side and $a(h^*, k)m_i^*$ on the right-hand side. As a result, we get

(7) For $i \in [n], k \in [q_{\text{sign}}]$, if $a(h^*, k) \neq 0$ and $m_i^{(k)} \neq \star$, then $m_i^* = m_i^{(k)}$ 💡 If $h^{(k)}$ has been used to form h^* , and the k th queried message has a concrete $\mathbb{Z}_p$ value (rather than $\star$), then the forgery message must have the same $\mathbb{Z}_p$ value.

Considering now the last components γ_i^* , of $\mathcal{A}$'s signature, from Fact (3), we get $\gamma_i^* = h^* \cdot y_i$ for i with $m_i^* = \star$. Plugging in Fact (4), we get $\gamma_i^* = \sum_{k=1}^{q_{\text{sign}}} a(h^*, k) \cdot r^{(k)} y_i$. Furthermore, γ_i^* must have been computed as a linear combination of polynomials queried to $\mathcal{O}.\text{compute}_1$ (as argued above). For this linear combination, note that there are no $r^{(k)}$ terms or $r^{(k)}x$ terms or y_i terms or constant terms in $\sum_{k=1}^{q_{\text{sign}}} a(h^*, k) \cdot r^{(k)} y_i$, hence $a(\gamma_i^*, k) = 0$ and $b(\gamma_i^*, k) = 0$ and $d(\gamma_i^*, i) = 0$ and $e(\gamma_i^*) = 0$ for all k, i . As a result, $\gamma_i^* = \sum_{k=1}^{q_{\text{sign}}} \sum_{i': m_{i'}^{(k)} = \star} c(G, k, i') \cdot r^{(k)} y_{i'}$. Putting both ways of writing γ_i^* together, we

get

$$\forall i \in [n] : m_i^* = \star \Rightarrow \gamma_i^* = \sum_{k=1}^{q_{\text{sign}}} a(h^*, k) \cdot r^{(k)} y_i = \sum_{k=1}^{q_{\text{sign}}} \sum_{i' : m_{i'}^{(k)} = \star} c(G, k, i') \cdot r^{(k)} y_{i'}$$

Because on the right-hand side, there are only terms $r^{(k)} y_{i'}$ for i' with $m_{i'}^{(k)} = \star$, the left-hand side can also just contain such terms. As a result, we get the following fact.

- (8) For $i \in [n], k \in [q_{\text{sign}}]$, if $a(h^*, k) \neq 0$ and $m_i^* = \star$, then $m_i^{(k)} = \star$. $\spadesuit$ If $h^{(k)}$ has been used to form h^* , and the forgery message has $m_i^* = \star$, then the k th queried message also has $m_i^{(k)} = \star$.

Finishing up, we show that any valid signature σ^* on m^* that $\mathcal{A}$ outputs triggers the nontriviality condition, i.e., $\mathcal{A}$ must have queried for a signature on message $m^{(k)} \in \mathcal{T}_{\text{signed}}$ such that $m^{(k)} \succeq m^*$ (as a result, $\mathcal{B}$ never outputs 1). From the verification equation, we know that $h^* \neq 0$ (Fact (1)). Using Fact (4), this means that there must be some $k \in [q_{\text{sign}}]$ such that $a(h^*, k) \neq 0$. For that k , we get from Fact (8) that $m_i^{(k)} \neq \star \Rightarrow m_i^* \neq \star$ for all $i \in [n]$. From Fact (7), we get that $m_i^{(k)} \neq \star \Rightarrow m_i = m_i^{(k)}$ for all $i \in [n]$. Together, we get that $m^{(k)} \succeq m^*$ (cf. Definition 6.1), i.e., the forgery can be trivially derived from queried messages. As a result, $\mathcal{B}$ always outputs 0 in the IDEAL GGM setting, formally $\Pr[\text{GGM}_{\mathcal{B}}^{\text{hiddenVars, IDEAL}}(\mathcal{Y}_1, \mathcal{Y}_2, \mathcal{Y}_t, p, \mathcal{X}_1, \dots, \mathcal{X}_{n+q_{\text{sign}}+1}) = 1] = 0$.

Applying the result to the asymptotic case. From the analysis above, we get that $\Pr[\text{GGM}_{\mathcal{B}}^{\text{hiddenVars, REAL}}(\mathcal{Y}_1, \mathcal{Y}_2, \mathcal{Y}_t, p, \mathcal{X}_1, \dots, \mathcal{X}_{n+q_{\text{sign}}+1}) = 1] \leq \frac{24q^2}{p-1}$, where $q = q_{\mathcal{A}} + 3q_{\text{sign}} + 4n + 7$. The analysis above holds for *every* prime group order p and any $q_{\mathcal{A}}, q_{\text{sign}}$ individually. In an asymptotic setting, we have $p > 2^\lambda$ (cf. Definition 2.3) and $q_{\mathcal{A}}(\lambda)$ and $q_{\text{sign}}(\lambda)$ are polynomial in λ because $\mathcal{A}$ is a ppt adversary, and n is a constant. Hence $q(\lambda) = q_{\mathcal{A}}(\lambda) + 3q_{\text{sign}}(\lambda) + 4n + 7$ is polynomial in λ . In that setting, we get that $\frac{24(q(\lambda))^2}{2^\lambda}$ is negligible and hence a generic adversary $\mathcal{A}$'s winning probability against the $\Pi_{\text{ps-dms}}$ unforgeability game is negligible, as required. $\square$

We now show that if the signature part of $\Pi_{\text{ps-dms}}$ is unforgeable (which we have just proven in the generic group model), then the protocol for signing committed values also has unforgeability.

Theorem 6.13 ($\Pi_{\text{ps-dms}}$ blind unforgeability). *If $\Pi_{\text{ps-dms}}$ (Construction 6.11) is existentially unforgeable under chosen-message attacks (Definition 6.7 and Theorem 6.12), then it is blindly unforgeable under chosen-known-message attacks (Definition 5.5).*

The main challenge for the reduction is convincingly running the **BlindSign** protocol without knowing σ_{iss} . For this, the reduction computes a fake but convincing commitment key K from the Y_i values in pk . Then it takes the opening $\vec{m}', o$ to the commitment C provided by the adversary, queries $\vec{m}'$ to its own signature oracle, and then blinds the signature it receives to make it look like the correct blinded signature for C .

Proof of Theorem 6.13. Let $\mathcal{A}$ be a ppt adversary against blind unforgeability of $\Pi_{\text{ps-dms}}$. We construct an adversary $\mathcal{B}^{\mathcal{O}.\text{PrivateSign}, \mathcal{O}.\text{Sign}}(\mathcal{G}, pp, pk)$ against the EUF-CMA-security of $\Pi_{\text{ps-dms}}$.

- $\mathcal{B}$ starts running $\mathcal{A}(\mathcal{G}, pp, pk)$.
- Whenever $\mathcal{A}$ queries its $\mathcal{O}.\text{PrivateBlindSign}(S, \vec{m}')$ oracle, $\mathcal{B}$ queries its own oracle $S' \leftarrow \mathcal{O}.\text{PrivateSign}(S, \vec{m}')$ and returns the handle S' to $\mathcal{A}$. $\mathcal{B}$ remembers $\vec{m}_{S'} = \vec{m}'$.
- Whenever $\mathcal{A}$ queries its $\mathcal{O}.\text{BlindSign}(S)$ oracle, $\mathcal{B}$ asserts that S is a signature handle that $\mathcal{B}$ has previously received from its $\mathcal{O}.\text{PrivateSign}$ oracle (or $S = \text{root}$). Let $(m_i^{(S)})_{i=1}^n := \vec{m}^{(\text{iss})} := \vec{m}_S$ denote the message corresponding to handle S . $\mathcal{B}$ sets $K = (K_0, (K_i)_{i: m_i^{(S)} = \star}) = (g^k, (Y_i^k)_{i: m_i^{(S)} = \star})$ for a random $k \leftarrow \mathbb{Z}_p^*$ and hands K to $\mathcal{A}$.
 ⚡ Note that $\mathcal{B}$ does not know signature σ_S , so it derives a dummy commitment key using the Y_i in pk . $\mathcal{A}$ returns $\vec{m}'$ with $\vec{m}_S \succeq \vec{m}'$, commitment $C = (C_{\text{fix}}, (C_i)_{i: m_i^{(\text{iss})} = \star})$, and commitment opening $o = (o_{\text{fix}}, (o_i, \omega_i, s_i)_{i: m_i^{(\text{iss})} = \star})$ such that $\text{VerifyCommit}(pp, pk, K, \vec{m}_S, C; \vec{m}', o) = 1$ (otherwise, $\mathcal{B}$ aborts this oracle query). $\mathcal{B}$ queries its $\mathcal{O}.\text{Sign}(S, \vec{m}')$ oracle to receive a signature $(h, \mu, (\gamma_i)_{i: m'_i = \star})$. $\mathcal{B}$ then computes the blinded signature $\sigma_{\text{blind}} = (h_{\text{blind}}, \mu_{\text{blind}}, (\gamma_{\text{blind}, i})_{i: m_i^{(\text{iss})} = \star})$ as

$$h_{\text{blind}} = h$$

$$\mu_{\text{blind}} = h^{o_{\text{fix}}} \cdot \mu \cdot \prod_{i: m'_i = \star} (\gamma_i)^{\omega_i}$$

and

$$\gamma_{\text{blind}, i} = \begin{cases} h^{o_i} \cdot \gamma_i^{s_i} & \text{if } m'_i = \star \\ h^{o_i} & \text{if } m'_i \neq \star \end{cases}$$

⚡ $\mathcal{B}$ adds dirt terms to be consistent with what $\mathcal{A}$ expects from BlindSign .

Then $\mathcal{B}$ returns σ_{blind} to $\mathcal{A}$.

- Eventually, $\mathcal{A}$ outputs a candidate forgery $(\vec{m}^*, \sigma^*)$, which $\mathcal{B}$ also outputs.

Note that $\mathcal{A}$'s oracle queries are answered exactly with the same distribution as $\mathcal{A}$ expects if it were run inside Definition 6.8. Disregarding error cases, for $\mathcal{O}.\text{PrivateBlindSign}(S, \vec{m}')$ queries, $\mathcal{A}$ expects a fresh signature handle S' , which it gets from $\mathcal{B}$. For $\mathcal{O}.\text{BlindSign}(S)$ queries, the adversary $\mathcal{A}$ expects a commitment key K of the form $(K_0, (K_0^{y_i})_{i: m_i^{(S)} = \star})$ for a random $K_0 \leftarrow \mathbb{G}_1 \setminus \{1\}$, which it indeed gets from $\mathcal{B}$. After outputting $(\vec{m}', C, o)$, $\mathcal{A}$ expects a blinded signature σ_{blind} according to $\Pi_{\text{ps-dms}}.\text{BlindSign}$, i.e., of the form

$$\left(h_{\text{iss}}^{rk'}, (C_{\text{fix}} \cdot \mu_{\text{iss}}^{k'})^r, (C_i^r)_{i: m_i^{(\text{iss})} = \star} \right),$$

where $\sigma_S = (h_{\text{iss}}, \mu_{\text{iss}}, \vec{\gamma}_{\text{iss}})$ is a valid signature on $\vec{m}^{(\text{iss})}$, $k' \in \mathbb{Z}_p^*$ is such that $h_{\text{iss}}^{k'} = K_0$, and $r \leftarrow \mathbb{Z}_p^*$ is random. We now analyze this expected distribution. Plugging in C_{fix}, C_i (as in `VerifyCommit`), we get that the expected blinded signature is

$$\left(h_{\text{iss}}^{rk'}, \quad (K_0^{o_{\text{fix}}} \cdot \prod_{i: m_i^{(\text{iss})} = \star} K_i^{\omega_i} \cdot \mu_{\text{iss}}^{k'})^r, \quad ((K_i^{s_i} \cdot K_0^{o_i})^r)_{i: m_i^{(\text{iss})} = \star} \right).$$

Plugging in $K_0 = h_{\text{iss}}^{k'}$ (by definition of k') and $K_i = K_0^{y_i} = h_{\text{iss}}^{k'y_i}$ (given how $\mathcal{B}$ chooses K_0, K_i), we get

$$\left(h_{\text{iss}}^{rk'}, \quad (h_{\text{iss}}^{k'o_{\text{fix}}} \cdot \prod_{i: m_i^{(\text{iss})} = \star} h_{\text{iss}}^{k'y_i \omega_i} \cdot \mu_{\text{iss}}^{k'})^r, \quad ((h_{\text{iss}}^{k'(o_i + y_i s_i)})^r)_{i: m_i^{(\text{iss})} = \star} \right).$$

Because r is distributed uniformly randomly and $h_{\text{iss}} \neq 1, k' \neq 0$, we can equivalently write the expected signature distribution as choosing a random $h \leftarrow \mathbb{G}_1 \setminus \{1\}$ and returning

$$\left(h, \quad h^{o_{\text{fix}}} \cdot \prod_{i: m_i^{(\text{iss})} = \star} h^{y_i \omega_i} \cdot \mu, \quad (h^{o_i + y_i s_i})_{i: m_i^{(\text{iss})} = \star} \right),$$

where the correspondence to the distribution above is via $h = h_{\text{iss}}^{rk'}$ and $\mu = \mu_{\text{iss}}^{rk'} = h_{\text{iss}}^{rk' \cdot (x + \sum_{i: m_i^{(\text{iss})} \neq \star} y_i m_i^{(\text{iss})})} = h^{(x + \sum_{i: m_i^{(\text{iss})} \neq \star} y_i m_i^{(\text{iss})})}$.

This is exactly the distribution of σ_{blind} output by $\mathcal{B}$ to $\mathcal{A}$ during `BlindSign` (note that $\gamma_i = h^{y_i}$). This establishes that $\mathcal{B}$ correctly simulates the view that $\mathcal{A}$ expects in its unforgeability game.

Let $\text{win}_{\mathcal{A}}$ be the event that $\mathcal{A}$ succeeds in producing a forgery, i.e., `Verify(pp, pk, \vec{m}^*, \sigma^*) = 1` and there is no $\vec{m} \in \mathcal{T}_{\text{signed}}$ with $\vec{m} \succeq \vec{m}^*$ (i.e., $\mathcal{A}$ did not ask for a signature that allows deriving a signature on $\vec{m}^*$). If $\text{win}_{\mathcal{A}}$, then $\mathcal{B}$ also outputs a valid forgery. Because $\mathcal{B}$ is a ppt and $\Pi_{\text{ps-dms}}$ is existentially unforgeable under chosen-message attacks, this happens with negligible probability. Hence $\Pr[\text{win}_{\mathcal{A}}]$ must be negligible as well. $\square$

Finally, we prove privacy for the receiver in the blind signing protocol.

Theorem 6.14. $\Pi_{\text{ps-dms}}$ (Construction 6.11) has signing blindness (Definition 6.9).

Signing blindness for Construction 6.11 is particularly simple: The commitment C is perfectly hiding and `BlindReceive` does not send any messages. The only concern left is the verification bit indicating whether the signature output by `BlindReceive` is valid. Fortunately, similarly to Theorem 5.11, we can rewrite the verification check in a way that does not involve the message $\vec{m}^{(b)}$.

Proof. Let $\mathcal{A}$ be an algorithm. For the sake of readability, we list the $\mathcal{O}.\text{BlindReceive}^b$ oracle from Definition 6.9 instantiated with Construction 6.11 here.

```

 $\mathcal{O}.\text{BlindReceive}^b(pk, K, \vec{m}^{(\text{iss})}, \vec{m}^{(0)}, \vec{m}^{(1)})$ 

Parse  $pk = (\tilde{X}, (Y_i)_{i=1}^n, (\tilde{Y}_i)_{i=1}^n) \in \mathbb{G}_2 \times \mathbb{G}_1^n \times \mathbb{G}_2^n$ 
assert  $\vec{m}^{(\text{iss})} \succeq \vec{m}^{(0)}$  and  $\vec{m}^{(\text{iss})} \succeq \vec{m}^{(1)}$ 
//Commit( $pp, pk, K, \vec{m}^{(\text{iss})}; \vec{m}^{(b)}$ ):
Parse  $K = (K_0, (K_i)_{i: m_i^{(\text{iss})} = \star}) \in (\mathbb{G}_1 \setminus \{1\})^{1+|\{i | m_i^{(\text{iss})} = \star\}|}$ 
Choose  $(o_{\text{fix}}, (o_i)_{i: m_i^{(\text{iss})} = \star}) \leftarrow \mathbb{Z}_p^{1+|\{i | m_i^{(\text{iss})} = \star\}|}$ 
Set  $C_{\text{fix}} = K_0^{o_{\text{fix}}} \cdot \prod_{i: m_i^{(\text{iss})} = \star \wedge m_i^{(b)} \neq \star} K_i^{m_i^{(b)}}$ 
 $C_i = K_i \cdot K_0^{o_i}$  for  $i$  with  $m_i^{(\text{iss})} = \star \wedge m_i^{(b)} = \star$ 
 $C_i = K_0^{o_i}$  for  $i$  with  $m_i^{(\text{iss})} = \star \wedge m_i^{(b)} \neq \star$ 
Hand  $C = (C_{\text{fix}}, (C_i)_{i: m_i^{(\text{iss})} = \star})$  to  $\mathcal{A}$ 
//BlindReceive( $pp, pk, K, \vec{m}^{(\text{iss})}, C; \vec{m}^{(b)}, o$ )  $\leftrightarrow \mathcal{A}$ :
 $\mathcal{A}$  returns  $\sigma_{\text{blind}} = (h_{\text{blind}}, \mu_{\text{blind}}, \vec{\gamma}_{\text{blind}})$ 
Set  $\sigma' = (h', \mu', (\gamma'_i)_{i: m_i^{(b)} = \star}) = (h_{\text{blind}}, (h_{\text{blind}})^{-o_{\text{fix}}} \cdot \mu_{\text{blind}}, ((h_{\text{blind}})^{-o_i} \cdot \gamma_{\text{blind}, i})_{i: m_i^{(b)} = \star})$ 
Set  $h := h' = h_{\text{blind}}$  //For notational convenience
//Verification
return 1 iff all these conditions are met:

(1)  $\gamma_{\text{blind}, i} = h^{o_i}$  for all  $i$  with  $m_i^{(b)} \neq \star$  //Check in BlindReceive
(2)  $e(K_i, \tilde{g}) = e(K_0, \tilde{Y}_i)$  for all  $i$  with  $m_i^{(\text{iss})} = \star$  //Check in BlindReceive
(3)  $h \neq 1$  //Check in Verify( $pp, pk, \vec{m}^{(b)}, \sigma$ )
(4)  $e(\mu', \tilde{g}) = e(h, \tilde{X} \cdot \prod_{i: m_i^{(b)} \neq \star} \tilde{Y}_i^{m_i^{(b)}})$  //Check in Verify( $pp, pk, \vec{m}^{(b)}, \sigma$ )
(5)  $e(\gamma_i, \tilde{g}) = e(h, \tilde{Y}_i)$  for all  $i$  with  $m_i^{(b)} = \star$  //Check in Verify( $pp, pk, \vec{m}^{(b)}, \sigma$ )

```

This oracle simply inlines **Commit** and **BlindReceive** from Construction 6.11 into the oracle from Definition 6.9. We syntactically group the checks made in **BlindReceive** into the final verification check (note that this is semantically equivalent, as **BlindReceive** outputs $\perp$ should any of its checks fail, making subsequent signature verification fail).

We now proceed to analyze the conditions under which $\mathcal{O}.\text{BlindReceive}^b$ outputs 1. For this, let $r, x, y_1, \dots, y_n \in \mathbb{Z}_p$ such that $h = h_{\text{blind}} = K_0^r$, $\tilde{g}^x = \tilde{X}$, and $\tilde{g}^{y_i} = \tilde{Y}_i$. Let $I_\star^{(\text{iss})} = \{i \mid m_i^{(\text{iss})} = \star\}$. Consider the following equivalence:

$$(2) \wedge (1) \wedge (5)$$

$$\begin{aligned}
&\Leftrightarrow (2) \wedge \forall i \in I_{\star}^{(\text{iss})} : \left(\left(m_i^{(b)} \neq \star \Rightarrow \gamma_{\text{blind},i} = h^{o_i} \right) \wedge \left(m_i^{(b)} = \star \Rightarrow e(\gamma_i, \tilde{g}) = e(h, \tilde{Y}_i) \right) \right) \\
&\Leftrightarrow (2) \wedge \forall i \in I_{\star}^{(\text{iss})} : \left(\left(m_i^{(b)} \neq \star \Rightarrow \gamma_{\text{blind},i} = h^{o_i} \right) \wedge \left(m_i^{(b)} = \star \Rightarrow \gamma_i = h^{y_i} \right) \right) \\
&\Leftrightarrow (2) \wedge \forall i \in I_{\star}^{(\text{iss})} : \left(\left(m_i^{(b)} \neq \star \Rightarrow \gamma_{\text{blind},i} = h^{o_i} \right) \wedge \left(m_i^{(b)} = \star \Rightarrow \gamma_{\text{blind},i} = h^{y_i + o_i} \right) \right) \\
&\Leftrightarrow (2) \wedge \forall i \in I_{\star}^{(\text{iss})} : \left(\left(m_i^{(b)} \neq \star \Rightarrow \gamma_{\text{blind},i} = K_0^{ro_i} \right) \wedge \left(m_i^{(b)} = \star \Rightarrow \gamma_{\text{blind},i} = K_0^{ry_i + ro_i} \right) \right) \\
&\quad \text{💡 Using } K_i = K_0^{y_i}, \text{ which is implied by (2)} \\
&\Leftrightarrow (2) \wedge \forall i \in I_{\star}^{(\text{iss})} : \left(\left(m_i^{(b)} \neq \star \Rightarrow \gamma_{\text{blind},i} = K_0^{ro_i} \right) \wedge \left(m_i^{(b)} = \star \Rightarrow \gamma_{\text{blind},i} = K_i^r \cdot K_0^{ro_i} \right) \right) \\
&\Leftrightarrow (2) \wedge \forall i \in I_{\star}^{(\text{iss})} : \left(\left(m_i^{(b)} \neq \star \Rightarrow \gamma_{\text{blind},i} = C_i^r \right) \wedge \left(m_i^{(b)} = \star \Rightarrow \gamma_{\text{blind},i} = C_i^r \right) \right) \\
&\Leftrightarrow (2) \wedge \forall i \in I_{\star}^{(\text{iss})} : \gamma_{\text{blind},i} = C_i^r
\end{aligned}$$

💡 Interpretation: We can rewrite checks (1) and (5) for $\vec{\gamma}_{\text{blind}}$ into one joint check that does not involve the message $\vec{m}^{(b)}$. In other words, the validity of $\vec{\gamma}_{\text{blind}}$ that $\mathcal{A}$ supplies only depends on pk , the commitments C_i , and the public message $\vec{m}^{(\text{iss})}$ (but not on the hidden message $\vec{m}^{(b)}$).

Further consider the following:

$$\begin{aligned}
(2) \wedge (4) &\Leftrightarrow (2) \wedge e(\mu', \tilde{g}) = e(h, \tilde{X} \cdot \prod_{i: m_i^{(b)} \neq \star} \tilde{Y}_i^{m_i^{(b)}}) \\
&\Leftrightarrow (2) \wedge \mu' = h^{x + \sum_{i: m_i^{(b)} \neq \star} y_i m_i^{(b)}} \\
&\Leftrightarrow (2) \wedge \mu' = h^x \cdot \prod_{i: m_i^{(\text{iss})} \neq \star \wedge m_i^{(b)} \neq \star} h^{y_i m_i^{(b)}} \cdot \prod_{i: m_i^{(\text{iss})} = \star \wedge m_i^{(b)} \neq \star} h^{y_i m_i^{(b)}}
\end{aligned}$$

💡 Using $\vec{m}^{(\text{iss})} \succeq \vec{m}^{(b)}$

$$\begin{aligned}
&\Leftrightarrow (2) \wedge \mu' = h^x \cdot \prod_{i: m_i^{(\text{iss})} \neq \star} h^{y_i m_i^{(\text{iss})}} \cdot \prod_{i: m_i^{(\text{iss})} = \star \wedge m_i^{(b)} \neq \star} h^{y_i m_i^{(b)}} \\
&\Leftrightarrow (2) \wedge \mu' = h^x \cdot \prod_{i: m_i^{(\text{iss})} \neq \star} h^{y_i m_i^{(\text{iss})}} \cdot \left(\prod_{i: m_i^{(\text{iss})} = \star \wedge m_i^{(b)} \neq \star} \frac{K_0^{y_i m_i^{(b)}}}{K_0^{y_i m_i^{(\text{iss})}}} \right)^r
\end{aligned}$$

💡 Using $K_i = K_0^{y_i}$, which is implied by (2)

$$\begin{aligned}
&\Leftrightarrow (2) \wedge \mu' = h^x \cdot \prod_{i: m_i^{(\text{iss})} \neq \star} h^{y_i m_i^{(\text{iss})}} \cdot \left(\prod_{i: m_i^{(\text{iss})} = \star \wedge m_i^{(b)} \neq \star} \frac{K_i^{m_i^{(b)}}}{K_i^{m_i^{(\text{iss})}}} \right)^r \\
&\Leftrightarrow (2) \wedge \mu' = h^x \cdot \prod_{i: m_i^{(\text{iss})} \neq \star} h^{y_i m_i^{(\text{iss})}} \cdot \left(C_{\text{fix}} \cdot K_0^{-o_{\text{fix}}} \right)^r
\end{aligned}$$

$$\begin{aligned}
&\Leftrightarrow (2) \wedge \mu' = h^x \cdot \prod_{i: m_i^{(\text{iss})} \neq \star} h^{y_i m_i^{(\text{iss})}} \cdot \frac{C_{\text{fix}}^r \cdot K_0^{-r o_{\text{fix}}}}{C_{\text{fix}}^r \cdot h^{-o_{\text{fix}}}} \\
&\Leftrightarrow (2) \wedge \mu' = h^x \cdot \prod_{i: m_i^{(\text{iss})} \neq \star} h^{y_i m_i^{(\text{iss})}} \cdot C_{\text{fix}}^r \cdot h^{-o_{\text{fix}}} \\
&\Leftrightarrow (2) \wedge \mu' \cdot \underline{h^{o_{\text{fix}}}} = h^x \cdot \prod_{i: m_i^{(\text{iss})} \neq \star} h^{y_i m_i^{(\text{iss})}} \cdot C_{\text{fix}}^r \\
&\Leftrightarrow (2) \wedge \underline{\mu_{\text{blind}}} = h^x \cdot \prod_{i: m_i^{(\text{iss})} \neq \star} h^{y_i m_i^{(\text{iss})}} \cdot C_{\text{fix}}^r
\end{aligned}$$

💡 Interpretation: We can rewrite check (4) for the fixed message part μ' of the final signature σ' in a way that does not involve the message $\vec{m}^{(b)}$. In other words, the validity of the blinded μ_{blind} that $\mathcal{A}$ supplies only depends on pk , the commitment C_{fix} , and the public message $\vec{m}^{(\text{iss})}$ (but not on the hidden message $\vec{m}^{(b)}$).

Armed with these facts, consider the following oracle $\mathcal{O}.\text{ReceiveSim}$ that, from $\mathcal{A}$'s point of view, behaves exactly like $\mathcal{O}.\text{BlindReceive}^b$.

$\mathcal{O}.\text{ReceiveSim}(pk, K, \vec{m}^{(\text{iss})}, \vec{m}^{(0)}, \vec{m}^{(1)})$

Parse $pk = (\tilde{X}, (Y_i)_{i=1}^n, (\tilde{Y}_i)_{i=1}^n) \in \mathbb{G}_2 \times \mathbb{G}_1^n \times \mathbb{G}_2^n$

assert $\vec{m}^{(\text{iss})} \succeq \vec{m}^{(0)}$ and $\vec{m}^{(\text{iss})} \succeq \vec{m}^{(1)}$

//Commit by choosing independent random values:

Parse $K = (K_0, (K_i)_{i: m_i^{(\text{iss})} = \star}) \in (\mathbb{G}_1 \setminus \{1\})^{1+|\{i: m_i^{(\text{iss})} = \star\}|}$

Choose $C_{\text{fix}} \leftarrow \mathbb{G}_1$ and $C_i \leftarrow \mathbb{G}_1$ for i with $m_i^{(\text{iss})} = \star$

Hand $C = (C_{\text{fix}}, (C_i)_{i: m_i^{(\text{iss})} = \star})$ to $\mathcal{A}$

//BlindReceive $\leftrightarrow \mathcal{A}$:

$\mathcal{A}$ returns $(h_{\text{blind}}, h_{\text{blind}}, (\gamma_{\text{blind}, i})_{i: m_i^{(\text{iss})} = \star})$

Let $r, x, y_1, \dots, y_n \in \mathbb{Z}_p$ be s.t. $K_0^r = h_{\text{blind}}$, $\tilde{X} = \tilde{g}^x$, and $\tilde{Y}_i = \tilde{g}^{y_i}$ for all $i \in [n]$
return 1 iff

- $K_i = K_0^{y_i}$ for all i with $m_i^{(\text{iss})} = \star$ 💡 Corresponds to (2)
- $h_{\text{blind}} \neq 1$ 💡 Same as (3)
- $\mu_{\text{blind}} = h_{\text{blind}}^x \cdot \prod_{i: m_i^{(\text{iss})} \neq \star} h_{\text{blind}}^{y_i m_i^{(\text{iss})}} \cdot C_{\text{fix}}^r$ 💡 Corresponds to (4), see above
- $\gamma_{\text{blind}, i} = C_i^r$ for all i with $m_i^{(\text{iss})} = \star$ 💡 Corresponds to (1) and (5), see above

💡 We note that this oracle is not efficiently implementable (given its reliance on discrete logarithms r, x, y_i). This is not an issue for this proof, which deals with an information-theoretical claim. We claim that for all inputs, $\mathcal{O}.\text{ReceiveSim}(pk, K, \vec{m}^{(\text{iss})}, \vec{m}^{(0)}, \vec{m}^{(1)})$ behaves ex-

actly like $\mathcal{O}.\text{BlindReceive}^b(pk, K, \vec{m}^{(\text{iss})}, \vec{m}^{(0)}, \vec{m}^{(1)})$ for both $b \in \{0, 1\}$. For this, we first note that in a run of the oracle $\mathcal{O}.\text{BlindReceive}^b$, the distribution of the commitment C is exactly as in $\mathcal{O}.\text{ReceiveSim}$, i.e., uniform over $\mathbb{G}_1^{1+|\{i|m_i^{(\text{iss})}=\star\}|}$. The criterion for outputting 1 is equivalent in both oracles, as established above. So overall, $\Pr[\mathcal{A}^{\mathcal{O}.\text{BlindReceive}^b}(\mathcal{G}, pp) = 1] = \Pr[\mathcal{A}^{\mathcal{O}.\text{ReceiveSim}}(\mathcal{G}, pp) = 1]$ for $b \in \{0, 1\}$. Hence

$$\left| \Pr[\mathcal{A}^{\mathcal{O}.\text{BlindReceive}^0}(\mathcal{G}, pp) = 1] - \Pr[\mathcal{A}^{\mathcal{O}.\text{BlindReceive}^1}(\mathcal{G}, pp) = 1] \right| = 0 \quad \square$$

We now prove privacy for the signer.

Theorem 6.15. *The $\Pi_{\text{ps-dms}}$ scheme (Construction 6.11) has perfect signer anonymity (Definition 6.10).*

Intuitively, because the signer randomizes her signature σ_{iss} to σ_{iss}^k for a random $k \in \mathbb{Z}_p^*$, it does not matter what signature σ_{iss} she starts with. Effectively, she uses the signature σ_{iss}^k that is distributed uniformly randomly in the set of all valid signatures on $\vec{m}^{(\text{iss})}$.

Proof. Let $pk = (\tilde{X}, (Y_i)_{i=1}^n, (\tilde{Y}_i)_{i=1}^n)$ and let $x, y_1, \dots, y_n \in \mathbb{Z}_p$ be such that $\tilde{X} = \tilde{g}^x$ and $\tilde{Y}_i = \tilde{g}^{y_i}$. Any valid signature $\sigma_{\text{iss}} = (h_{\text{iss}}, \mu_{\text{iss}}, (\gamma_{\text{iss},i})_{i: m_i^{(\text{iss})}=\star})$ on message $\vec{m}^{(\text{iss})}$ for public key pk must have the following form, as guaranteed by the verification equations in **Verify**:

- $h_{\text{iss}} \in \mathbb{G}_1 \setminus \{1\}$
- $\mu_{\text{iss}} = h_{\text{iss}}^{x + \sum_{i: m_i^{(\text{iss})} \neq \star} y_i m_i^{(\text{iss})}}$
- $\gamma_{\text{iss},i} = h_{\text{iss}}^{y_i}$ for all i where $m_i^{(\text{iss})} = \star$

This means that h_{iss} (together with $\vec{m}^{(\text{iss})}, pk$) uniquely determines μ_{iss} and $\vec{\gamma}_{\text{iss}}$. Let

$$\Sigma(pk, \vec{m}) = \left\{ (h, h^{x + \sum_{i: m_i \neq \star} y_i m_i}, (h^{y_i})_{i: m_i = \star}) \mid h \in \mathbb{G}_1 \setminus \{1\} \right\}$$

denote the set of valid signatures on $\vec{m}$ for public key pk .

The first step of **Precommit**($pp, pk, \vec{m}^{(\text{iss})}, \sigma_{\text{iss}}$) is to choose a random factor $k \leftarrow \mathbb{Z}_p^*$. If $\sigma_{\text{iss}} \in \Sigma(pk, \vec{m}^{(\text{iss})})$, then the resulting signature $\sigma_{\text{iss}}^k = (h_{\text{iss}}^k, \mu_{\text{iss}}^k, (\gamma_{\text{iss},i}^k)_{i: m_i^{(\text{iss})}=\star})$ is distributed uniformly in $\Sigma(pk, \vec{m}^{(\text{iss})})$. Both **Precommit** and **BlindSign** only ever refer to σ_{iss}^k , never to σ_{iss} itself. As a result, the outward behavior of **Precommit** and **BlindSign** is the same, no matter what signature $\sigma_{\text{iss}} \in \Sigma(\vec{m}^{(\text{iss})}, pk)$ we supply. The property required by Definition 6.10 follows immediately. $\square$

6.4 Constructing delegatable credentials

We provide a generic construction of delegatable credentials based on dynamically malleable signatures. First, we describe this construction in Section 6.4.1, then prove its security in Sections 6.4.2 and 6.4.3, and finally, we discuss how to efficiently instantiate the construction in Section 6.4.4.

6.4.1 The standard template with delegation

Our construction of delegatable (updatable) anonymous credentials broadly follows what we call the “standard template”, i.e., the established general structure for anonymous credential systems as described in Sections 4.3 and 5.3.2: Credentials are signatures on

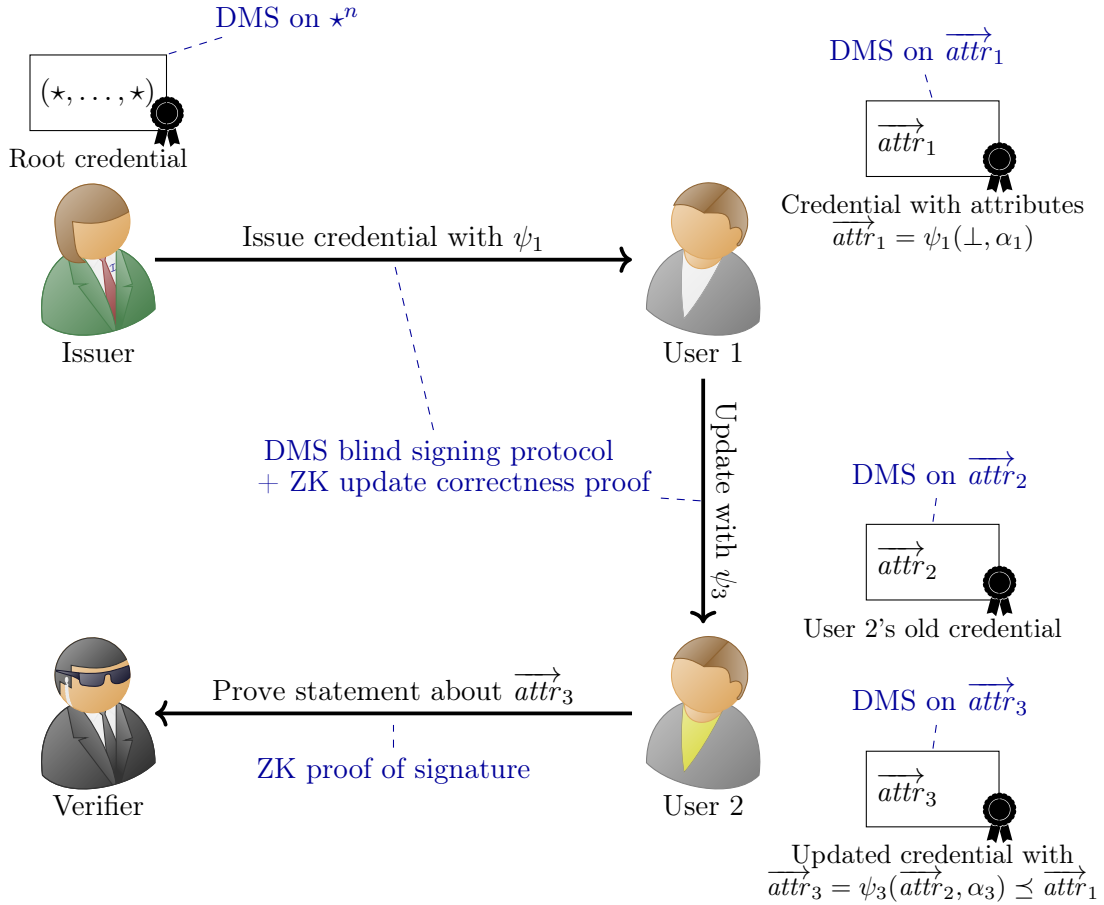


Figure 6.2: The standard template for constructing delegatable updatable anonymous credential systems.

the user’s attributes. Issuing a credential (on hidden attributes) consists of a commitment to the desired attributes, a zero-knowledge proof that the chosen attributes are valid, and a blind signing protocol to sign the committed attributes. Updates work similarly.

Crucially, for our notion of delegatable credentials, a credential is not simply *any* ordinary signature, but a dynamically malleable signature, which enables controlled malleability and producing weaker signatures for delegation. We depict the template in Figure 6.2.

Overall, the construction is quite similar to our construction of updatable anonymous credentials (Construction 5.7). Adherence to the ordinary way of constructing anonymous credentials is a considerable strength of our approach. In contrast, other constructions of delegatable credentials (with the exception of [CDD17]) often deviate significantly from the standard template, which makes it difficult to apply standard extensions such as revocation.

Construction 6.16 (Standard template delegatable updatable anonymous credentials). Let Π_{dms} be a dynamically malleable signature scheme (Definition 6.6) with message space $\mathbb{M}_{\mathcal{G}}$. Let Π_{zk} be a proof system for some relation $\mathcal{R}_{\mathcal{G}}$. Let $\Psi_{\mathcal{G}}[\overrightarrow{\text{attr}}_{\text{iss}}], \Phi_{\mathcal{G}}$ be some sets of update functions / predicates. We construct a delegatable updatable anonymous credential system Π_{dacs} with attribute space $\mathbb{A}_{\mathcal{G}} = \mathbb{M}_{\mathcal{G}}$ as follows:

Setup($\mathcal{G}$)

Generates public parameters $pp = (\mathcal{G}, pp_0, pp_1)$, where $pp_0 \leftarrow \Pi_{\text{dms}}.\text{Setup}(\mathcal{G})$ and $pp_1 \leftarrow \Pi_{\text{zk}}.\text{Setup}(\mathcal{G})$.

IssuerKeyGen($pp, 1^n$)

Generates issuer keys $(ipk, cred^*) \leftarrow \Pi_{\text{dms}}.\text{Keygen}(pp, 1^n)$.

ReceiveCred($pp, ipk, \overrightarrow{\text{attr}}_{\text{iss}}, \psi, ctx; \alpha$) $\leftrightarrow$ IssueCred($pp, ipk, \overrightarrow{\text{attr}}_{\text{iss}}, \psi, ctx; cred_{\text{iss}}$)

Works as shown in Figure 6.3.

ReceiveUpdate($pp, ipk, \overrightarrow{\text{attr}}_{\text{iss}}, \psi, ctx; \overrightarrow{\text{attr}}, cred, \alpha$)
 $\leftrightarrow$ UpdateCred($pp, ipk, \overrightarrow{\text{attr}}_{\text{iss}}, \psi, ctx; cred_{\text{iss}}$)

Works as shown in Figure 6.4.

PresentCred($pp, ipk, \phi, ctx; \overrightarrow{\text{attr}}, cred, \alpha$) $\leftrightarrow$ VerifyCred(pp, ipk, ϕ, ctx)

Asserts $\phi \in \Phi_{\mathcal{G}}$, then has PresentCred prove

$$\text{ZKPoK} \left[\left(\overrightarrow{\text{attr}}, cred, \alpha \right); \quad \phi(\overrightarrow{\text{attr}}, \alpha) = 1 \right. \\ \left. \wedge \quad \Pi_{\text{dms}}.\text{Verify}(pp, ipk, \overrightarrow{\text{attr}}, cred) = 1 \right] (ctx)$$

If successful, VerifyCred outputs 1.

CheckCred($pp, ipk, \overrightarrow{\text{attr}}, cred$)

Outputs $\Pi_{\text{dms}}.\text{Verify}(pp, ipk, \overrightarrow{\text{attr}}, cred)$.

We require that $\mathcal{R}_{\mathcal{G}}$ is compatible (Definition 3.25) with the ZKPoK $[(\cdot); \cdot] (\cdot)$ expressions above and in Figures 6.3 and 6.4.

It is easy to check that this construction is a *correct* delegatable UACS, given that Π_{dms} is correct and Π_{zk} is complete.

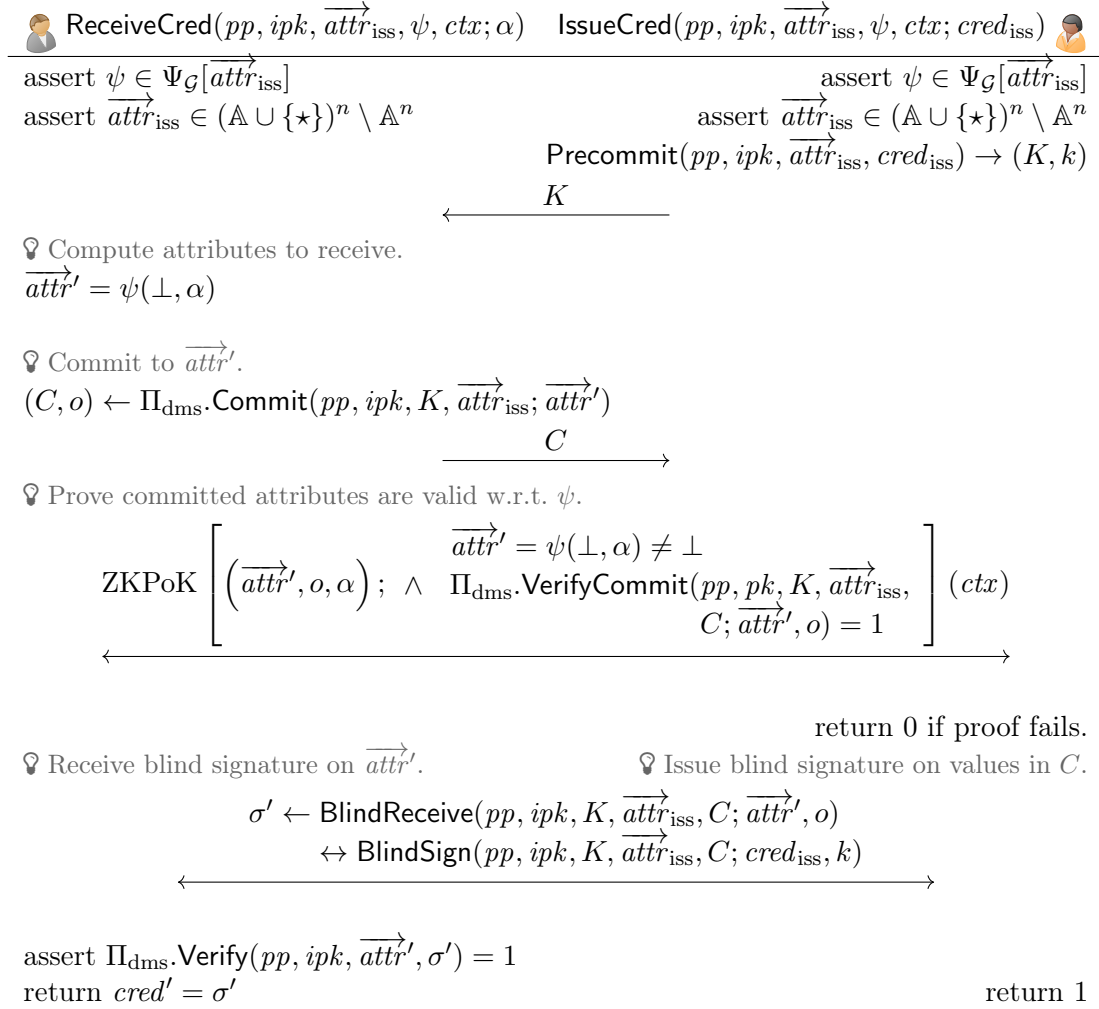


Figure 6.3: The protocol for issuing a credential by delegating $cred_{iss}$.

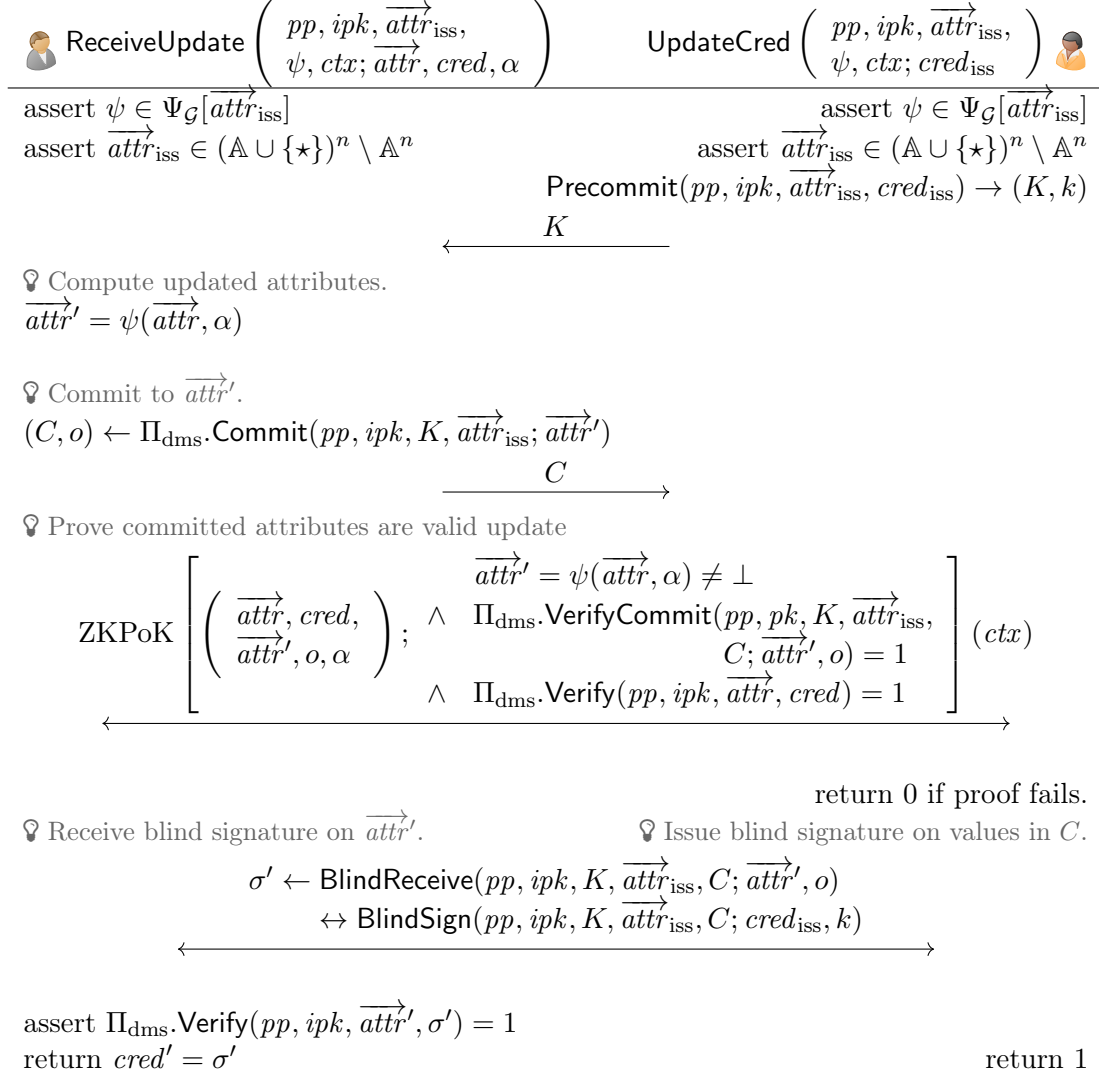


Figure 6.4: The protocol for updating a credential by delegating $cred_{iss}$.

6.4.2 Anonymity of standard template DACS

We now consider privacy for users.

Theorem 6.17 (Standard template DACS have simulation anonymity). *If Π_{dms} has signing blindness (Definition 6.9) and Π_{zk} is concurrent zero-knowledge (Definition 3.22), then Construction 6.16 has simulation anonymity (Definition 6.3).*

Similarly to its UACS equivalent (Theorem 5.8), there are three components in the issue or update protocol: (1) the commitment to the new attributes, (2) the proof, and (3) the blind signature protocol. As before, the proof (2) can be simulated using the zero-knowledge simulator. However, (1) and (3) require slight changes compared to Theorem 5.8. There, we were able to simulate (1) and (3) by committing to some random fixed dummy message $\vec{m}_{\text{dummy}} = (m_{\text{dummy}}, \dots, m_{\text{dummy}}) \in \mathbb{M}^n$ and running the blind signature protocol for that message. For DACS, in contrast, issuances/updates are restricted to the set of attributes that the issuer (delegator) can issue given her credential cred_{iss} on attributes $\vec{attr}_{\text{iss}} \in (\mathbb{M} \cup \{\star\})^n$. This prevents us from using the same $\vec{m}_{\text{dummy}}$ to simulate all issuances/updates. Instead, we are going to use dummy message vectors $\vec{m}_{\text{dummy}}(\vec{m}^{(\text{iss})}) \in \mathbb{A}^n$ depending on and compatible with the issuer's attributes $\vec{attr}_{\text{iss}} = \vec{m}^{(\text{iss})} \in (\mathbb{A} \cup \{\star\})^n$. Specifically, for a (random) dummy value $m_{\text{dummy}} \leftarrow \mathbb{M}$, we write

$$\vec{m}_{\text{dummy}}(\vec{m}^{(\text{iss})}) = (m_1, \dots, m_n),$$

where

$$m_i := \begin{cases} m_{\text{dummy}} & \text{if } m_i^{(\text{iss})} = \star \\ m_i^{(\text{iss})} & \text{otherwise} \end{cases}$$

Then for queries involving issuer message $\vec{m}^{(\text{iss})}$, the simulator uses $\vec{m}_{\text{dummy}}(\vec{m}^{(\text{iss})})$ to simulate the commitment and DMS blind issuance protocol. Note that $\vec{m}_{\text{dummy}}(\vec{m}^{(\text{iss})}) \preceq \vec{m}^{(\text{iss})}$ for all $\vec{m}^{(\text{iss})} \in (\mathbb{A} \cup \{\star\})^n$ by design.

Apart from this detail, the proof works almost exactly like its UACS counterpart (Theorem 5.8). We list the proof here for completeness and highlight significant changes with the symbol $\mathcal{P}$.

The proof proceeds in two steps: We first show that the REAL game cannot be distinguished from a HYBRID game in which the zero-knowledge proofs are being simulated. We then show that the HYBRID game is indistinguishable from the SIM game.

Proof of Theorem 6.17. Let $\mathcal{S}_{\text{zk}}$ be the simulator from the zero-knowledge property of Π_{zk} . We construct the simulator $\mathcal{S}$ for DACS as follows:

- $\mathcal{S}.\text{Setup}(\mathcal{G})$ outputs $pp = (\mathcal{G}, pp_0, pp_1)$, where $pp_0 \leftarrow \Pi_{\text{dms}}.\text{Setup}(\mathcal{G})$ and $pp_1 \leftarrow \mathcal{S}_{\text{zk}}.\text{Setup}(\mathcal{G})$. $\P$ Note that Π_{zk} 's setup is executed by the ZK simulator. Furthermore, $\mathcal{S}$ chooses a random dummy message $m_{\text{dummy}} \leftarrow \mathbb{M}$. For any $\vec{attr}_{\text{iss}} = \vec{m}^{(\text{iss})} \in (\mathbb{A} \cup \{\star\})^n$,

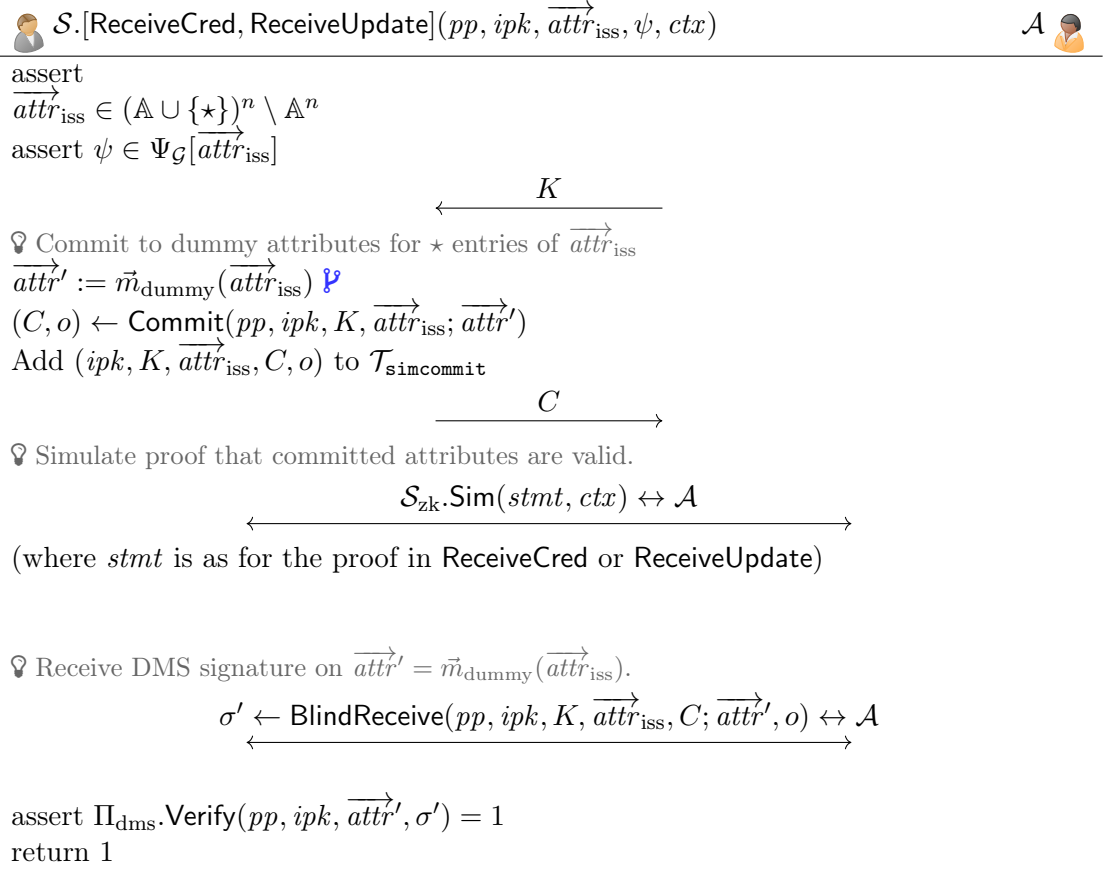


Figure 6.5: The simulator $\mathcal{S}$ for simulating receiving a credential or a credential update.

we denote by $\vec{m}_{\text{dummy}}(\vec{m}^{(\text{iss})})$ the vector $\vec{m}_{\text{dummy}}(\vec{m}^{(\text{iss})}) = (m_1, \dots, m_n) \in \mathbb{M}^n$ where

$$m_i := \begin{cases} m_{\text{dummy}} & \text{if } m_i^{(\text{iss})} = \star \\ m_i^{(\text{iss})} & \text{otherwise} \end{cases}$$

🔗 $\mathcal{S}$ will simulate commitment and protocol using message $\vec{m}_{\text{dummy}}(\vec{attr}_{\text{iss}})$, where $\vec{attr}_{\text{iss}}$ are the issuer's attributes. Note that $\vec{m}_{\text{dummy}}(\vec{attr}_{\text{iss}}) \preceq \vec{attr}_{\text{iss}}$, i.e., $\vec{m}_{\text{dummy}}(\vec{attr}_{\text{iss}})$ crucially represents attributes that the issuer *is able to* issue. $\mathcal{S}$ initializes a set $\mathcal{T}_{\text{simcommit}} = \emptyset$. 🔗 In $\mathcal{T}_{\text{simcommit}}$, we will remember the openings to commitments created by $\mathcal{S}$, which will be relevant for Theorem 6.19

- $\mathcal{S}.\text{ReceiveCred}$ and $\mathcal{S}.\text{ReceiveUpdate}$ work as described in Figure 6.5. 🔗 Essentially, the simulator simulates the ZK proof and runs the DMS blind signature procedure (including the commitment) with $\vec{m}_{\text{dummy}}(\vec{attr}_{\text{iss}})$ as a message.
- $\mathcal{S}.\text{PresentCred}(pp, ipk, \phi, ctx)$ asserts that $\phi \in \Phi$ and then runs $\mathcal{S}_{\text{zk}}.\text{Sim}(stmt, ctx)$, where $stmt$ is as in PresentCred .
- $\mathcal{S}.\mathcal{RO}(x)$ returns $\mathcal{S}_{\text{zk}}.\mathcal{RO}(x)$.

Let $\mathcal{A}$ be a ppt. We need to show that

$$\left| \begin{aligned} & \Pr[\text{Exp}_{\Pi_{\text{dacs}}}^{\text{sim-anon}}(\Pi_{\text{dacs}}, \mathcal{S}, \mathcal{A}, \lambda, \text{REAL}) = 1] \\ & - \Pr[\text{Exp}_{\Pi_{\text{dacs}}}^{\text{sim-anon}}(\Pi_{\text{dacs}}, \mathcal{S}, \mathcal{A}, \lambda, \text{SIM}) = 1] \end{aligned} \right|$$

is negligible and that the runtime of $\text{Exp}_{\Pi_{\text{dacs}}}^{\text{sim-anon}}(\Pi_{\text{dacs}}, \mathcal{S}, \mathcal{A}, \lambda, \text{SIM})$ is polynomially bounded. The latter follows easily from the fact that the runtime of $\mathcal{S}_{\text{zk}}$ is polynomially bounded (Definition 3.22) and all the other building block functions are also polynomial-time computable.

Towards analyzing the difference between the REAL game and the SIM game, we introduce a hybrid game $\text{Exp}_{\Pi_{\text{dacs}}}^{\text{sim-anon}}(\Pi_{\text{dacs}}, \mathcal{S}, \mathcal{A}, \lambda, \text{HYBRID})$, which we define to work just like $\text{Exp}_{\Pi_{\text{dacs}}}^{\text{sim-anon}}(\Pi_{\text{dacs}}, \mathcal{S}, \mathcal{A}, \lambda, \text{REAL})$ except that $pp_1 \leftarrow \mathcal{S}_{\text{zk}}.\text{Setup}(\mathcal{G})$ in the setup phase, the oracle $\mathcal{O}.\mathcal{RO}$ is replaced with $\mathcal{S}.\mathcal{RO}$, and that the Π_{zk} proofs in $\mathcal{O}.\text{ReceiveCred}^{\text{REAL}}$, $\mathcal{O}.\text{ReceiveUpdate}^{\text{REAL}}$, $\mathcal{O}.\text{PresentCred}^{\text{REAL}}$ are replaced with $\mathcal{S}_{\text{zk}}.\text{Sim}(stmt, ctx)$ (where the values $stmt, ctx$ are as in the original protocols). Through a trivial reduction to the zero-knowledge property of Π_{zk} , one can show that

$$\left| \begin{aligned} & \Pr[\text{Exp}_{\Pi_{\text{dacs}}}^{\text{sim-anon}}(\Pi_{\text{dacs}}, \mathcal{S}, \mathcal{A}, \lambda, \text{REAL}) = 1] \\ & - \Pr[\text{Exp}_{\Pi_{\text{dacs}}}^{\text{sim-anon}}(\Pi_{\text{dacs}}, \mathcal{S}, \mathcal{A}, \lambda, \text{HYBRID}) = 1] \end{aligned} \right|$$

is negligible.

The only difference between the HYBRID game and the SIM game, regarding what can be possibly observed by $\mathcal{A}$, is that in $\mathcal{O}.\text{ReceiveCred}$ and $\mathcal{O}.\text{ReceiveUpdate}$, the SIM game runs the commitment and blind signature protocol with message $\vec{m}_{\text{dummy}}(\vec{attr}_{\text{iss}})$,

while HYBRID uses the actual attributes $\overrightarrow{attr}'$. We set up a trivial reduction to the signing blindness property of Π_{dms} . The reduction algorithm $\mathcal{B}$ uses oracle queries $\mathcal{O}.\text{BlindReceive}^b(ipk, K, \overrightarrow{attr}_{\text{iss}}, \psi(X, \alpha), \vec{m}_{\text{dummy}}(\overrightarrow{attr}_{\text{iss}}))$ to answer queries to the oracles $\mathcal{O}.\text{ReceiveCred}$ (where $X = \perp$) and $\mathcal{O}.\text{ReceiveUpdate}$ (where $X = \overrightarrow{attr}_{\text{C}_{\text{upd}}}$). Note that $\vec{m}_{\text{dummy}}(\overrightarrow{attr}_{\text{iss}}) \preceq \overrightarrow{attr}_{\text{iss}}$ and both the HYBRID and SIM games check $\psi \in \Psi_{\mathcal{G}}[\overrightarrow{attr}_{\text{iss}}]$, implying that $\psi(X, \alpha) \preceq \overrightarrow{attr}_{\text{iss}}$. Hence the assertion in the $\mathcal{O}.\text{BlindReceive}^b$ oracle is fulfilled. With that reduction, we get that

$$\left| \begin{array}{l} \Pr[\text{Exp}_{\Pi_{\text{dacs}}}^{\text{sim-anon}}(\Pi_{\text{dacs}}, \mathcal{S}, \mathcal{A}, \lambda, \text{HYBRID}) = 1] \\ - \Pr[\text{Exp}_{\Pi_{\text{dacs}}}^{\text{sim-anon}}(\Pi_{\text{dacs}}, \mathcal{S}, \mathcal{A}, \lambda, \text{SIM}) = 1] \end{array} \right|$$

is negligible. Through triangle inequality, we overall get that

$$\begin{aligned} & \left| \begin{array}{l} \Pr[\text{Exp}_{\Pi_{\text{dacs}}}^{\text{sim-anon}}(\Pi_{\text{dacs}}, \mathcal{S}, \mathcal{A}, \lambda, \text{REAL}) = 1] \\ - \Pr[\text{Exp}_{\Pi_{\text{dacs}}}^{\text{sim-anon}}(\Pi_{\text{dacs}}, \mathcal{S}, \mathcal{A}, \lambda, \text{SIM}) = 1] \end{array} \right| \\ \leq & \left| \begin{array}{l} \Pr[\text{Exp}_{\Pi_{\text{dacs}}}^{\text{sim-anon}}(\Pi_{\text{dacs}}, \mathcal{S}, \mathcal{A}, \lambda, \text{REAL}) = 1] \\ - \Pr[\text{Exp}_{\Pi_{\text{dacs}}}^{\text{sim-anon}}(\Pi_{\text{dacs}}, \mathcal{S}, \mathcal{A}, \lambda, \text{HYBRID}) = 1] \end{array} \right| \\ & + \left| \begin{array}{l} \Pr[\text{Exp}_{\Pi_{\text{dacs}}}^{\text{sim-anon}}(\Pi_{\text{dacs}}, \mathcal{S}, \mathcal{A}, \lambda, \text{HYBRID}) = 1] \\ - \Pr[\text{Exp}_{\Pi_{\text{dacs}}}^{\text{sim-anon}}(\Pi_{\text{dacs}}, \mathcal{S}, \mathcal{A}, \lambda, \text{SIM}) = 1] \end{array} \right| \end{aligned}$$

is negligible, as required. $\square$

We now consider privacy for the delegator.

Theorem 6.18 (Standard template DACS have perfect delegator anonymity). *If Π_{dms} has perfect signer anonymity (Definition 6.10), then Construction 6.16 has perfect delegator anonymity (Definition 6.4).*

The property directly follows from Definition 6.10 due to its similarity to the property we need to prove (Definition 6.4). Intuitively speaking, the blind signing process for DMS does not reveal any information about the issuer's signature σ_{iss} , hence the DACS scheme does not reveal any information about $\text{cred}_{\text{iss}} = \sigma_{\text{iss}}$ during issuance/update, either: The credential/signature is only used for the commitment key K and for the blind signing protocol; the verification of the zero-knowledge proof does not meaningfully depend on the issuer's credential.

Proof. Let $n, \lambda \in \mathbb{N}$, $\mathcal{G} \in [\text{GenSetting}(1^\lambda)]$, $pp \in [\text{Setup}(\mathcal{G})]$, $ipk, \overrightarrow{attr}_{\text{iss}}, \text{cred}_{\text{iss}}, \text{cred}'_{\text{iss}}$ with $\text{CheckCred}(pp, ipk, \overrightarrow{attr}_{\text{iss}}, \text{cred}_{\text{iss}}) = \text{CheckCred}(pp, ipk, \overrightarrow{attr}_{\text{iss}}, \text{cred}'_{\text{iss}}) = 1$, $\mathcal{A}$, and ψ, ctx be as in Definition 6.4. For our protocol (assuming the issuer does not abort the protocol due to illegal input or invalid proof), the view of $\mathcal{A}$ when interacting with **IssueCred** with input cred_{iss} is $\text{view}_{\mathcal{A}} = (R, K, M)$, where R is the randomness string of $\mathcal{A}$, the commitment key K is computed by the honest issuer via **Precommit**, and M is the list of messages received from **BlindSign**. For $\mathcal{A}$ interacting with **IssueCred** with input $\text{cred}'_{\text{iss}}$, we denote the view $\text{view}'_{\mathcal{A}} = (R, K', M')$. We need to show that the distribution

of $view_{\mathcal{A}}$ is identical to $view'_{\mathcal{A}}$.

This is a straightforward application of Definition 6.10. For this, we use pp_0 , $pk = ipk$, $\vec{m}^{(iss)} = \vec{attr}_{iss}$, $\sigma_{iss} = cred_{iss}$, $\sigma'_{iss} = cred'_{iss}$. We define the algorithm $\mathcal{A}_{dms}(K)$ to run $\mathcal{A}$, send K to $\mathcal{A}$, and output the C that $\mathcal{A}$ replies with. $\mathcal{A}_{dms}$ also outputs $st = state_{\mathcal{A}}$. We define $\mathcal{B}_{dms}(st)$ to continue running $\mathcal{A}$ with state st , where $\mathcal{B}_{dms}$ plays the role of the proof system verifier for the proof within **IssueCred**. If the proof is accepted, $\mathcal{B}_{dms}$ interacts with **BlindSign** (Definition 6.10) by relaying messages between **BlindSign** and $\mathcal{A}$. Let $view_{\mathcal{B}_{dms}}$ be the view of $\mathcal{B}_{dms}$ when interacting with **BlindSign** with input σ_{iss} and let $view'_{\mathcal{B}_{dms}}$ be the view when interacting with **BlindSign** with input σ'_{iss} .

Applying Definition 6.10 to the choices above gives us that K is distributed identically to K' and because $view_{\mathcal{B}_{dms}}$ is distributed identically to $view'_{\mathcal{B}_{dms}}$ (which contain the messages from the **BlindSign** interaction), we also get that M is distributed identically to M' .

This covers **IssueCred**. The same reasoning applies to **UpdateCred**. $\square$

6.4.3 Explainability of standard template DACS

We now consider unforgeability, i.e., simulation-explainability.

Theorem 6.19 (Standard template DACS have simulation-explainability). *If Π_{dms} has signing blindness (Definition 6.9), is blindly unforgeable under chosen-known-message attacks (Definition 6.8), and has super-poly message space (Definition 2.10), and Π_{zk} is a concurrently adaptively simulation-extractable zero-knowledge proof of knowledge (Definition 3.22), then Construction 6.16 has simulation explainability (Definition 6.3).*

With some (important) details omitted, intuitively, the proof works as follows. Because users prove knowledge of exactly what kind of update they are running or what credential they are presenting, we can use the proof system extractor $\mathcal{E}_{zk}$ in order to produce explanations. Consistency of these explanations then follows from simulation-extractability of Π_{zk} (ensuring, for example, $\psi(\vec{attr}, \alpha) \neq \perp$) and unforgeability of Π_{dms} (ensuring that a credential with $\vec{attr}$ must have been issued before).

The proof is quite similar to its UACS counterpart (Theorem 5.9). We highlight significant changes in the proof below with the symbol $\mathcal{P}$. The main changes are syntactic (issuing now uses a credential rather than an issuer secret key, there is a precommitment step in the **IssueCred** and **UpdateCred** protocols, etc.), adaptations around the dummy message (as explained around Theorem 6.17), and accounting for the additional oracles $\mathcal{O}.\text{IssueHonest}$, $\mathcal{O}.\text{UpdateHonest}$ modeling honest users delegating credentials among themselves.

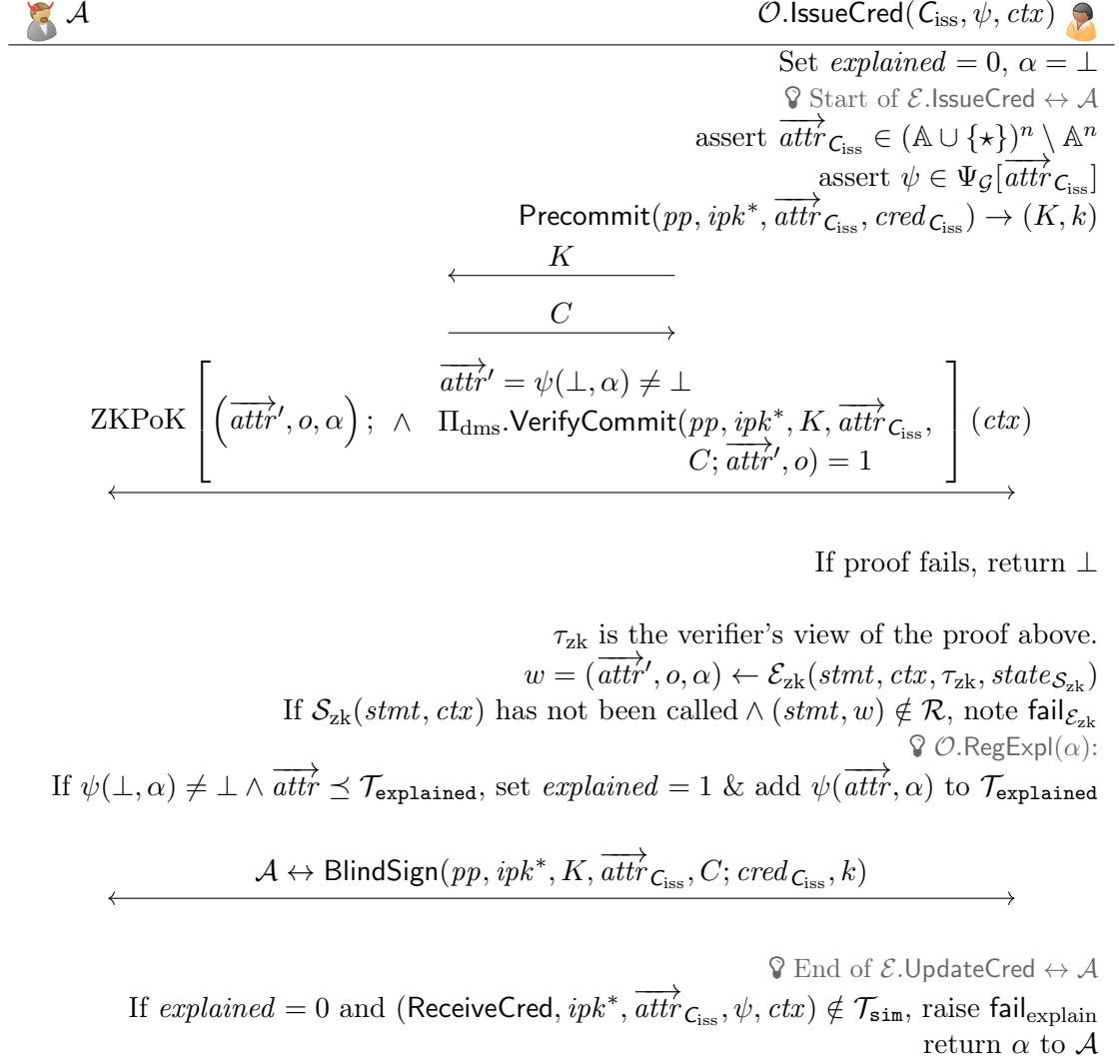


Figure 6.6: The $\mathcal{O}.\text{IssueCred}$ oracle execution in $\text{Exp}_{\Pi_{\text{dacs}}}^{\text{sim-expl}}(\Pi_{\text{dacs}}, \mathcal{A}, \mathcal{S}, \mathcal{E}, n, \lambda)$ for the proof of Theorem 6.19.

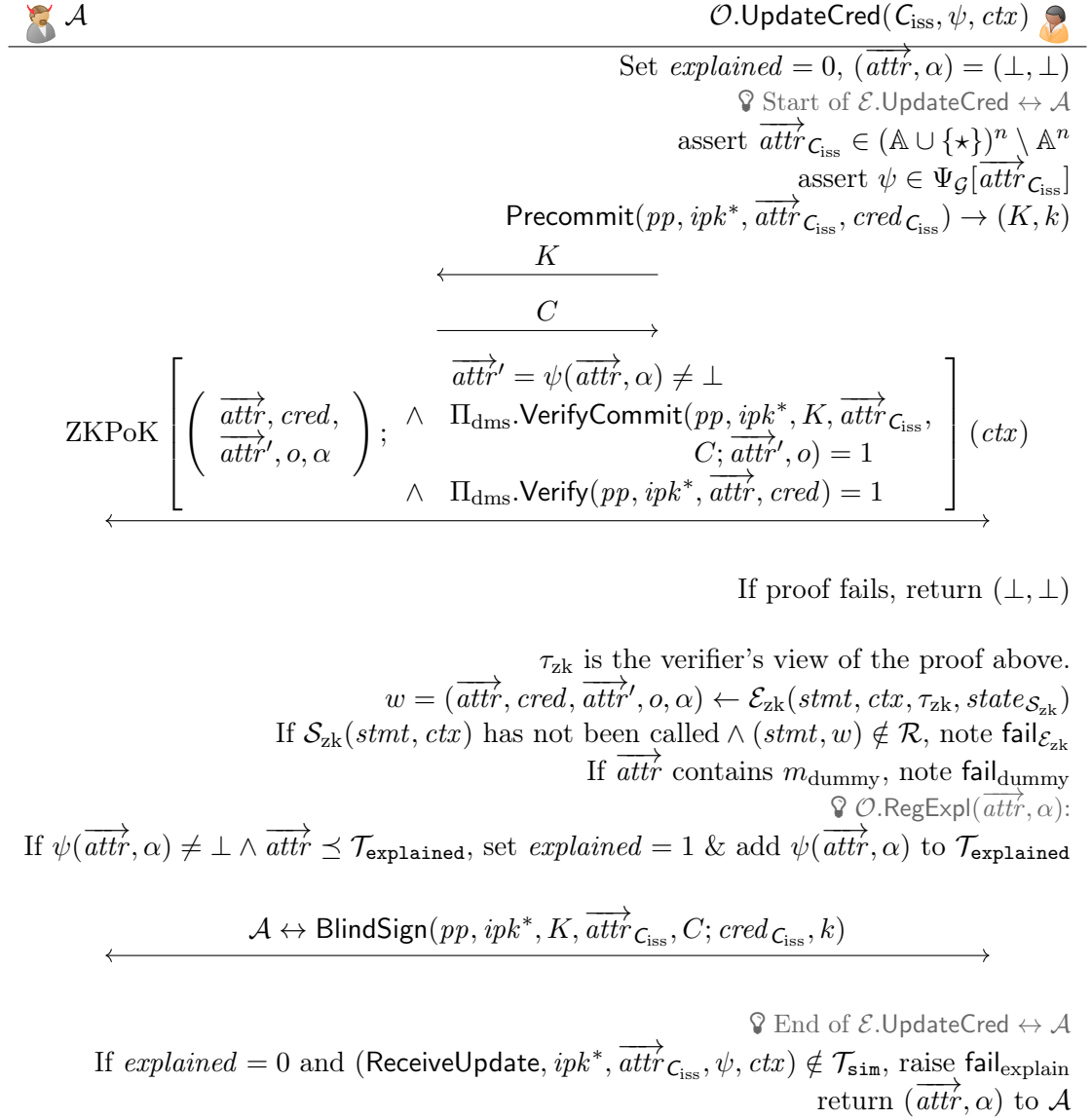


Figure 6.7: The $\mathcal{O}.\text{UpdateCred}$ oracle execution in $\text{Exp}_{\Pi_{\text{dacs}}}^{\text{sim-expl}}(\Pi_{\text{dacs}}, \mathcal{A}, \mathcal{S}, \mathcal{E}, n, \lambda)$ for the proof of Theorem 6.19.

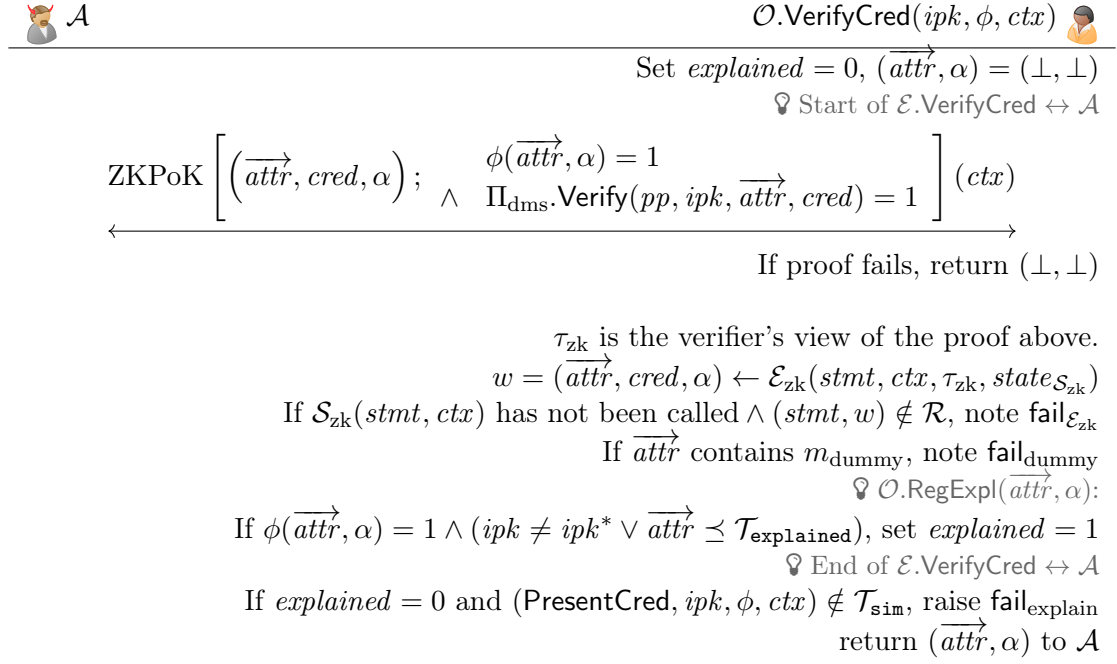


Figure 6.8: The $\mathcal{O}.VerifyCred$ oracle execution in $\text{Exp}_{\Pi_{dacs}}^{\text{sim-expl}}(\Pi_{dacs}, \mathcal{A}, \mathcal{S}, \mathcal{E}, n, \lambda)$ for the proof of Theorem 6.19.

Proof of Theorem 6.19. Let $\mathcal{S}$ be as in the proof of Theorem 6.17. Let $state_{\mathcal{S}_{zk}}$ denote the state of $\mathcal{S}_{zk}$ within $\mathcal{S}$. Let m_{dummy} be the randomly generated message stored in $state_{\mathcal{S}}$ after $\mathcal{S}.Setup$. Let $\mathcal{E}_{zk}$ be an extractor for Π_{zk} as in Definition 3.22. Without loss of generality, we assume that $\mathcal{E}_{zk}(stmt, ctx, \tau_{zk}, state_{\mathcal{S}_{zk}})$ aborts immediately and outputs $w = \perp$ if $\mathcal{S}_{zk}(stmt, ctx)$ has been called before (according to $state_{\mathcal{S}_{zk}}$). $\heartsuit$ This is a small detail. In Definition 3.22, if $\mathcal{S}_{zk}(stmt, ctx)$ has been called before, $\mathcal{O}.Extract$ simply outputs $\perp$ and does not run $\mathcal{E}_{zk}(stmt, ctx, \tau_{zk}, state_{\mathcal{S}_{zk}})$ at all. For syntactic simplicity, we want to be able to run $\mathcal{E}_{zk}$ even for simulated proofs (with no expectation of receiving a valid witness), so we make $\mathcal{E}_{zk}$ immediately output $\perp$ for those cases, which then is consistent to the oracle behavior in Definition 3.22.

We define the explainer $\mathcal{E}.X^{\mathcal{S}.RO, \mathcal{O}.GetSimState, \mathcal{O}.RegExpl(input)}$ (for $X \in \{\text{IssueCred}, \text{UpdateCred}, \text{VerifyCred}\}$), to run $\Pi_{dacs}.X^{\mathcal{S}.RO}(input)$ with the following additional behavior right after $\Pi_{zk}.Verify(stmt, ctx)$ within X outputs 1:

- $\mathcal{E}$ takes the simulator state $state_{\mathcal{S}_{zk}}$ from $state_{\mathcal{S}} \leftarrow \mathcal{O}.GetSimState()$ and the verifier view τ_{zk} , and runs the proof system extractor $w \leftarrow \mathcal{E}_{zk}(stmt, ctx, \tau_{zk}, state_{\mathcal{S}_{zk}})$.
- If $\mathcal{S}_{zk}(stmt, ctx)$ has not been called and $(stmt, w) \notin \mathcal{R}$, $\mathcal{E}$ notes event $fail_{\mathcal{E}_{zk}}$.
- Then, $\mathcal{E}$ retrieves the dummy message m_{dummy} from $state_{\mathcal{S}}$ and
 - for $X = \text{IssueCred}$, parse $w = (\vec{attr}', o, \alpha)$. $\mathcal{E}$ calls $\mathcal{O}.RegExpl(\alpha)$.

- for $X = \text{UpdateCred}$, parse $w = (\overrightarrow{attr}, cred, \overrightarrow{attr'}, o, \alpha)$. If $\overrightarrow{attr}$ contains m_{dummy} (P i.e., $\overrightarrow{attr} = (attr_1, \dots, attr_n)$ and $\exists i : attr_i = m_{\text{dummy}}$), then $\mathcal{E}$ notes event $\text{fail}_{\text{dummy}}$. $\mathcal{E}$ calls $\mathcal{O}.\text{RegExpl}(\overrightarrow{attr}, \alpha)$.
- for $X = \text{VerifyCred}$, parse $w = (\overrightarrow{attr}, cred, \alpha)$. If $\overrightarrow{attr}$ contains m_{dummy} (P i.e., $\overrightarrow{attr} = (attr_1, \dots, attr_n)$ and $\exists i : attr_i = m_{\text{dummy}}$), then $\mathcal{E}$ notes event $\text{fail}_{\text{dummy}}$. $\mathcal{E}$ calls $\mathcal{O}.\text{RegExpl}(\overrightarrow{attr}, \alpha)$.

- $\mathcal{E}$ then continues execution of X and outputs whatever X outputs.

Clearly, $\mathcal{E}.X^{\mathcal{S}, \mathcal{R}, \mathcal{O}, \mathcal{O}.\text{GetSimState}, \mathcal{O}.\text{RegExpl}}$ perfectly emulates $\Pi_{\text{dacs}}.X^{\mathcal{S}, \mathcal{R}, \mathcal{O}}$ (given that the additional behavior does not have any side effects other than the additional oracle calls to $\mathcal{O}.\text{GetSimState}$ and $\mathcal{O}.\text{RegExpl}$).

For a better overview, we detail the effective behavior of $\mathcal{O}.X$ with explainer $\mathcal{E}$ in Figures 6.6 to 6.8. Note that because by definition, $\mathcal{E}$ only calls $\mathcal{O}.\text{RegExpl}$ once per execution, and because protocol execution in case the initial proof fails ($b = 0$) is aborted immediately, those figures leave out trivially fulfilled oracle checks regarding $b = 0$ and $b = 1$, while still remaining perfectly faithful to the definition of $\text{Exp}_{\Pi_{\text{dacs}}}^{\text{sim-expl}}(\Pi_{\text{dacs}}, \mathcal{A}, \mathcal{S}, \mathcal{E}, n, \lambda)$.

P Additionally, we list the effective behavior of the oracles that delegate between two honest users, skipping some trivial checks. Note that by design of the oracles and correctness of our construction, any credential under handle C is necessarily a valid DMS signature on $\overrightarrow{attr}_C$.

$\mathcal{O}.\text{IssueHonest}(C_{\text{iss}}, \psi, \alpha, ctx)$ //Honest user delegates to another honest user

```

assert  $\overrightarrow{attr}_{C_{\text{iss}}} \in (\mathbb{A} \cup \{\star\})^n \setminus \mathbb{A}^n$  and  $\psi \in \Psi_{\mathcal{G}}[\overrightarrow{attr}_{C_{\text{iss}}}]$  and  $\perp \neq \psi(\perp, \alpha)$ 
Choose new credential handle  $C'$ 
Set  $\overrightarrow{attr}_{C'} = \psi(\perp, \alpha)$ 
P Start of ReceiveCred  $\leftrightarrow$  IssueCred
 $(K, k) \leftarrow \text{Precommit}(pp, ipk, \overrightarrow{attr}_{C_{\text{iss}}}, cred_{C_{\text{iss}}})$ 
 $(C, o) \leftarrow \Pi_{\text{dms}}.\text{Commit}(pp, ipk, K, \overrightarrow{attr}_{C_{\text{iss}}}; \overrightarrow{attr}_{C'})$ 
P We skip the ZKPoK, which will always succeed due to completeness of  $\Pi_{\text{zk}}$ .
 $\sigma' \leftarrow \text{BlindReceive}(pp, ipk, K, \overrightarrow{attr}_{C_{\text{iss}}}, C; \overrightarrow{attr}_{C'}, o)$ 
 $\leftrightarrow \text{BlindSign}(pp, ipk, K, \overrightarrow{attr}_{C_{\text{iss}}}, C; cred_{C_{\text{iss}}}, k)$ 
P We skip  $\Pi_{\text{dms}}.\text{Verify}$ , as it always succeeds due to  $\Pi_{\text{dms}}$  correctness.
Set  $cred_{C'} = \sigma'$ 
P End of ReceiveCred  $\leftrightarrow$  IssueCred
return  $C'$ 

```

```

 $\mathcal{O}.\text{UpdateHonest}(C_{\text{iss}}, C_{\text{upd}}, \psi, \alpha, ctx)$  //Honest user updates honest user's cred
assert  $\vec{attr}_{C_{\text{iss}}} \in (\mathbb{A} \cup \{\star\})^n \setminus \mathbb{A}^n$  and  $\psi \in \Psi_{\mathcal{G}}[\vec{attr}_{C_{\text{iss}}}]$  and  $\perp \neq \psi(\vec{attr}_{C_{\text{upd}}}, \alpha)$ 
Choose new credential handle  $C'$ 
Set  $\vec{attr}_{C'} = \psi(\vec{attr}_{C_{\text{upd}}}, \alpha)$ 
 $\text{Start of ReceiveUpdate} \leftrightarrow \text{ReceiveCred}$ 
 $(K, k) \leftarrow \text{Precommit}(pp, ipk, \vec{attr}_{C_{\text{iss}}}, cred_{C_{\text{iss}}})$ 
 $(C, o) \leftarrow \Pi_{\text{dms}}.\text{Commit}(pp, ipk, K, \vec{attr}_{C_{\text{iss}}}; \vec{attr}_{C'})$ 
 $\text{We skip the ZKPoK, which will always succeed due to completeness of } \Pi_{\text{zk}}.$ 
 $\sigma' \leftarrow \text{BlindReceive}(pp, ipk, K, \vec{attr}_{C_{\text{iss}}}, C; \vec{attr}_{C'}, o)$ 
 $\quad \leftrightarrow \text{BlindSign}(pp, ipk, K, \vec{attr}_{C_{\text{iss}}}, C; cred_{C_{\text{iss}}}, k)$ 
 $\text{We skip } \Pi_{\text{dms}}.\text{Verify}, \text{ as it always succeeds due to } \Pi_{\text{dms}} \text{ correctness.}$ 
Set  $cred_{C'} = \sigma'$ 
 $\text{End of ReceiveUpdate} \leftrightarrow \text{ReceiveCred}$ 
return  $C'$ 

```

Let $n \in \mathbb{N}$ and let $\mathcal{A}$ be a ppt. The running time of the simulation explainability experiment $\text{Exp}_{\Pi_{\text{dacs}}}^{\text{sim-expl}}(\Pi_{\text{dacs}}, \mathcal{A}, \mathcal{S}, \mathcal{E}, n, \lambda)$ can easily be upper-bounded by a polynomial depending on $\mathcal{A}$ because all building blocks are polynomial-time and the overall running time of $\mathcal{S}_{\text{zk}}, \mathcal{E}_{\text{zk}}$ can be upper-bounded using Definition 3.22. $\text{Here, we use our additional "abort immediately for previously simulated } stmt, ctx"$ assumption on $\mathcal{E}_{\text{zk}}$ from the beginning of this proof.

It remains to show that $\Pr[\text{fail}_{\text{explain}}]$ is negligible. For this, we consider the events $\text{fail}_{\text{dummy}}$ and $\text{fail}_{\mathcal{E}_{\text{zk}}}$ as defined by $\mathcal{E}$ (see also Figures 6.6 to 6.8). We will prove that $\Pr[\text{fail}_{\text{dummy}}]$ and $\Pr[\text{fail}_{\mathcal{E}_{\text{zk}}}]$ are negligible, and then finally that $\Pr[\neg \text{fail}_{\mathcal{E}_{\text{zk}}} \wedge \neg \text{fail}_{\text{dummy}} \wedge \text{fail}_{\text{explain}}]$ is negligible.

Showing that $\Pr[\text{fail}_{\text{dummy}}]$ is negligible. $\text{fail}_{\text{dummy}}$ is the event that the explainer outputs an explanation of the form “ $\mathcal{A}$ used a credential whose attributes contain m_{dummy} ”, where m_{dummy} is the message used by the simulator $\mathcal{S}$ as a stand-in for the actual unknown message in the blind signing procedure (see Theorem 6.17). We do not want this to happen, because in the final step of this proof, which is a reduction to Π_{dms} ’s unforgeability, we will have to query signatures on $\vec{m}_{\text{dummy}}(\cdot)$ in order to make the simulator $\mathcal{S}$ work, so we cannot claim $\vec{m}_{\text{dummy}}(\cdot)$ as a forgery. Thankfully, intuitively, $\mathcal{A}$ cannot guess m_{dummy} , so it is unlikely that $\text{fail}_{\text{dummy}}$ happens. $\text{In contrast to the UACS proof (Theorem 5.9), where we showed that the adversary will not use } (m_{\text{dummy}}, m_{\text{dummy}}, \dots) \text{ as attributes, here we show that the adversary will not use any vector that contains } m_{\text{dummy}} \text{ (among other messages). The general argument is the same: It is hard to guess } m_{\text{dummy}}.$ Consider a modified game $\mathbf{G}_{\text{zero}}$ that works like $\text{Exp}_{\Pi_{\text{dacs}}}^{\text{sim-expl}}(\Pi_{\text{dacs}}, \mathcal{A}, \mathcal{S}, \mathcal{E}, n, \lambda)$ except that whenever $\mathcal{S}$ would use $\vec{m}_{\text{dummy}}(\vec{attr}_{\text{iss}})$ as input to $\Pi_{\text{dms}}.\text{Commit}$ or $\Pi_{\text{dms}}.\text{BlindReceive}$, $\text{it instead uses } \vec{m}_{\text{zero}}(\vec{attr}_{\text{iss}}), \text{ where}$

$\vec{m}_{\text{zero}}(m_1^{(\text{iss})}, \dots, m_n^{(\text{iss})}) = (m_1, \dots, m_n)$ with

$$m_i := \begin{cases} 0 & \text{if } m_i^{(\text{iss})} = \star \\ m_i^{(\text{iss})} & \text{otherwise} \end{cases}.$$

⚡ So in G_{zero} , the simulator replaces all $\star$ in $\vec{attr}_{\text{iss}}$ with 0 instead of m_{dummy} . As a consequence, the dummy message $m_{\text{dummy}} \leftarrow \mathbb{M}$ chosen randomly by $\mathcal{S}$ is never used in G_{zero} ($\mathcal{E}$ has access to it through $state_{\mathcal{S}}$, but only reads the value to check whether $\text{fail}_{\text{dummy}}$ happens). Because of that and because the message space has super-poly size, $\Pr[\text{fail}_{\text{dummy}} \mid G_{\text{zero}}] \leq q(\lambda)/|\mathbb{M}_{\mathcal{G}}|$, where q is a polynomial upper-bounding the number of calls to the explainer $\mathcal{E}$, is negligible. ⚡ As a consequence, as we show next, $\Pr[\text{fail}_{\text{dummy}} \mid \text{Exp}_{\Pi_{\text{dacs}}}^{\text{sim-expl}}]$ should also be negligible because the Π_{dms} blindness property ensures that $\mathcal{A}$ cannot distinguish between $\mathcal{S}$ using $\vec{m}_{\text{dummy}}(\vec{attr}_{\text{iss}})$ or $\vec{m}_{\text{zero}}(\vec{attr}_{\text{iss}})$.

We construct distinguisher $\mathcal{B}$ against the signing blindness property (Definition 6.9) of Π_{dms} . $\mathcal{B}^{\mathcal{O}.\text{BlindReceive}^b}(\mathcal{G}, pp_0)$ internally runs $\text{Exp}_{\Pi_{\text{dacs}}}^{\text{sim-expl}}(\Pi_{\text{dacs}}, \mathcal{A}, \mathcal{S}, \mathcal{E}, n, \lambda)$ with the following differences:

- $\mathcal{B}$ uses $\mathcal{G}$ and pp_0 from its input instead of generating them at the beginning of $\text{Exp}_{\Pi_{\text{dacs}}}^{\text{sim-expl}}$.
- Whenever the simulator $\mathcal{S}$ would compute and check the commitment C and run BlindReceive during $\mathcal{O}.\text{SimReceiveCred}$ or $\mathcal{O}.\text{SimReceiveUpdate}$, both are delegated to $\mathcal{B}$'s oracle $\mathcal{O}.\text{BlindReceive}^b(ipk, K, \vec{attr}_{\text{iss}}, \vec{m}_{\text{dummy}}(\vec{attr}_{\text{iss}}), \vec{m}_{\text{zero}}(\vec{attr}_{\text{iss}}))$. ⚡ So for $b = 0$, $\mathcal{B}$ behaves like $\text{Exp}_{\Pi_{\text{dacs}}}^{\text{sim-expl}}$, using $\vec{m}_{\text{dummy}}(\vec{attr}_{\text{iss}})$ for simulation, and for $b = 1$, it behaves like G_{zero} , using $\vec{m}_{\text{zero}}(\vec{attr}_{\text{iss}})$. ($\mathcal{S}$ then cannot keep the set $\mathcal{T}_{\text{simcommit}}$ in its state because it does not get to see openings o , but $\mathcal{T}_{\text{simcommit}}$ is not accessed in $\text{Exp}_{\Pi_{\text{dacs}}}^{\text{sim-expl}}(\Pi_{\text{dacs}}, \mathcal{A}, \mathcal{S}, \mathcal{E}, n, \lambda)$, anyway.)

When $\text{fail}_{\text{dummy}}$ happens, $\mathcal{B}$ outputs 1. Otherwise, if that event does not happen during its run, then $\mathcal{B}$ outputs 0. It is easy to check that

$$\Pr[\mathcal{B}^{\mathcal{O}.\text{BlindReceive}^0}(\mathcal{G}, pp_0) = 1] = \Pr[\text{fail}_{\text{dummy}} \mid \text{Exp}_{\Pi_{\text{dacs}}}^{\text{sim-expl}}(\Pi_{\text{dacs}}, \mathcal{A}, \mathcal{S}, \mathcal{E}, n, \lambda)]$$

and

$$\Pr[\mathcal{B}^{\mathcal{O}.\text{BlindReceive}^1}(\mathcal{G}, pp_0) = 1] = \Pr[\text{fail}_{\text{dummy}} \mid G_{\text{zero}}]$$

Because $\mathcal{B}$ is ppt, we get from Definition 6.9 that

$$\left| \Pr[\mathcal{B}^{\mathcal{O}.\text{BlindReceive}^0}(\mathcal{G}, pp_0) = 1] - \Pr[\mathcal{B}^{\mathcal{O}.\text{BlindReceive}^1}(\mathcal{G}, pp_0) = 1] \right|$$

$$= \left| \Pr[\text{fail}_{\text{dummy}} \parallel \text{Exp}_{\Pi_{\text{dacs}}}^{\text{sim-expl}}(\Pi_{\text{dacs}}, \mathcal{A}, \mathcal{S}, \mathcal{E}, n, \lambda)] - \Pr[\text{fail}_{\text{dummy}} \parallel \mathbf{G}_{\text{zero}}] \right|$$

is negligible. Because $\Pr[\text{fail}_{\text{dummy}} \parallel \mathbf{G}_{\text{zero}}]$ is negligible (as explained above), we get that $\Pr[\text{fail}_{\text{dummy}} \parallel \text{Exp}_{\Pi_{\text{dacs}}}^{\text{sim-expl}}(\Pi_{\text{dacs}}, \mathcal{A}, \mathcal{S}, \mathcal{E}, n, \lambda)]$ must be negligible, too.

Showing that $\Pr[\text{fail}_{\mathcal{E}_{\text{zk}}}]$ is negligible. $\mathfrak{!}$ $\text{fail}_{\mathcal{E}_{\text{zk}}}$ is the event that the proof of knowledge extractor $\mathcal{E}_{\text{zk}}$ within the DACS explainer $\mathcal{E}$ fails to provide a valid witness at some point. We can show that $\Pr[\text{fail}_{\mathcal{E}_{\text{zk}}}]$ is negligible via reduction to the simulation-extractability property of Π_{zk} . For this, we construct the ppt $\mathcal{B}^{\mathcal{S}_{\text{zk}}.\mathcal{RO}, \mathcal{O}.\text{Simul}, \mathcal{O}.\text{Extract}}(\mathcal{G}, pp_1)$, which internally runs $\text{Exp}_{\Pi_{\text{dacs}}}^{\text{sim-expl}}(\Pi_{\text{dacs}}, \mathcal{A}, \mathcal{S}, \mathcal{E}, n, \lambda)$ with the following differences: $\mathfrak{!}$ Essentially, $\mathcal{B}$ never runs $\mathcal{S}_{\text{zk}}$ or $\mathcal{E}_{\text{zk}}$ itself, but delegates this to its oracles.

- $\mathcal{B}$ uses $\mathcal{G}$ and pp_1 from its input instead of generating them at the beginning of $\text{Exp}_{\Pi_{\text{dacs}}}^{\text{sim-expl}}(\Pi_{\text{dacs}}, \mathcal{A}, \mathcal{S}, \mathcal{E}, n, \lambda)$. $\mathfrak{!}$ As a consequence, $\mathcal{B}$ does not have access to $\text{state}_{\mathcal{S}_{\text{zk}}}$, so we replace all occurrences of $\mathcal{S}_{\text{zk}}$ and $\mathcal{E}_{\text{zk}}$ with oracle queries below.
- Whenever $\mathcal{S}$ would execute $\mathcal{S}_{\text{zk}}(\text{stmt}, \text{ctx})$ (for some stmt, ctx), $\mathcal{B}$ instead calls its oracle $\mathcal{O}.\text{Simul}(\text{stmt}, \text{ctx})$. $\mathfrak{!}$ This happens during oracle $\mathcal{O}.\text{SimReceiveCred}$, $\mathcal{O}.\text{SimReceiveUpdate}$, and $\mathcal{O}.\text{SimPresent}$ queries, since the Π_{dacs} simulator $\mathcal{S}$ calls the Π_{zk} simulator $\mathcal{S}_{\text{zk}}$ as a subroutine.
- Whenever $\mathcal{E}$ would execute $\Pi_{\text{zk}}.\text{Verify}(\text{stmt}, \text{ctx}) \leftrightarrow \mathcal{A}$ (for some stmt, ctx) followed by $w \leftarrow \mathcal{E}_{\text{zk}}(\text{stmt}, \text{ctx}, \tau_{\text{zk}}, \text{state}_{\mathcal{S}_{\text{zk}}})$, $\mathcal{B}$ instead calls its oracle $w \leftarrow \mathcal{O}.\text{Extract}(\text{stmt}, \text{ctx})$ and relays the Π_{zk} messages between the oracle and $\mathcal{A}$. $\mathfrak{!}$ This happens whenever the DACS explainer $\mathcal{E}$ is run within $\mathcal{O}.\text{IssueCred}$, $\mathcal{O}.\text{UpdateCred}$, and $\mathcal{O}.\text{VerifyCred}$. See Figures 6.6 to 6.8.
- Random oracle calls $\mathcal{S}.\mathcal{RO}(\cdot)$ are answered using $\mathcal{B}$'s $\mathcal{S}_{\text{zk}}.\mathcal{RO}(\cdot)$ oracle.

It is easy to check that $\Pr[\text{ExtractionError} \parallel \mathcal{B}^{\mathcal{S}_{\text{zk}}.\mathcal{RO}, \mathcal{O}.\text{Simul}, \mathcal{O}.\text{Extract}}(\mathcal{G}, pp_1)] = \Pr[\text{fail}_{\mathcal{E}_{\text{zk}}} \parallel \text{Exp}_{\Pi_{\text{dacs}}}^{\text{sim-expl}}(\Pi_{\text{dacs}}, \mathcal{A}, \mathcal{S}, \mathcal{E}, n, \lambda)]$, where **ExtractionError** is the error triggered in Definition 3.22 when $\mathcal{E}_{\text{zk}}$ outputs an invalid witness for $(\text{stmt}, \text{ctx}) \notin \mathcal{T}_{\text{zk}}$. Because $\mathcal{B}$ is a ppt, the simulation-extractability property (Definition 3.22) implies that $\Pr[\text{ExtractionError} \parallel \mathcal{B}^{\mathcal{S}_{\text{zk}}.\mathcal{RO}, \mathcal{O}.\text{Simul}, \mathcal{O}.\text{Extract}}(\mathcal{G}, pp_1)]$ is negligible, which also means that $\Pr[\text{fail}_{\mathcal{E}_{\text{zk}}} \parallel \text{Exp}_{\Pi_{\text{dacs}}}^{\text{sim-expl}}(\Pi_{\text{dacs}}, \mathcal{A}, \mathcal{S}, \mathcal{E}, n, \lambda)]$ is negligible.

Showing that $\Pr[\neg \text{fail}_{\mathcal{E}_{\text{zk}}} \wedge \neg \text{fail}_{\text{dummy}} \wedge \text{fail}_{\text{explain}}]$ is negligible. $\mathfrak{!}$ As one can see in Figures 6.6 to 6.8, there are two ways to trigger $\text{fail}_{\text{explain}}$: (1) If the explanation does not fit the update function ψ or the predicate ϕ , i.e., $\psi(\cdot, \alpha) = \perp$ or $\phi(\overrightarrow{\text{attr}}, \alpha) = 0$ for the values output by $\mathcal{E}_{\text{zk}}$, or (2) if $\overrightarrow{\text{attr}} \not\in \mathcal{T}_{\text{explained}}$ in $\mathcal{O}.\text{UpdateCred}$ or $\mathcal{O}.\text{VerifyCred}$. Given $\neg \text{fail}_{\mathcal{E}_{\text{zk}}}$, the event $\text{fail}_{\text{explain}}$ cannot be triggered by (1) because $\overrightarrow{\text{attr}}, \alpha$ were extracted from a proof guaranteeing exactly those properties. Hence the only way to trigger $\text{fail}_{\text{explain}}$ is through the $\overrightarrow{\text{attr}} \not\in \mathcal{T}_{\text{explained}}$ criterion (2) in $\mathcal{O}.\text{UpdateCred}$ or $\mathcal{O}.\text{VerifyCred}$. But if that happens, then intuitively, the honest

issuer has never issued a DMS signature on attributes covering $\overrightarrow{attr}$, so the extracted signature $cred$ on $\overrightarrow{attr}$ must be a forgery.

We prove that $\Pr[\neg \text{fail}_{\mathcal{E}_{zk}} \wedge \neg \text{fail}_{\text{dummy}} \wedge \text{fail}_{\text{explain}}]$ is negligible via reduction to the unforgeability property (Definition 6.8) of Π_{dms} . For this, we construct $\mathcal{B}$ against blind unforgeability under chosen-known-message attacks of Π_{dms} . Our reduction adversary $\mathcal{B}^{\mathcal{O}.\text{PrivateBlindSign}, \mathcal{O}.\text{BlindSign}}(\mathcal{G}, pp_0, ipk^*)$ runs $\text{Exp}_{\Pi_{\text{dacs}}}^{\text{sim-expl}}(\Pi_{\text{dacs}}, \mathcal{A}, \mathcal{S}, \mathcal{E}, n, \lambda)$, with the following differences.

- $\mathcal{B}$ uses $\mathcal{G}, pp_0, ipk^*$ from its input instead of generating them at the beginning of $\text{Exp}_{\Pi_{\text{dacs}}}^{\text{sim-expl}}$. ? As a consequence, $\mathcal{B}$ does not have access to ? the root credential $cred_{\text{root}}$ (and will not have access to most other credentials, either), so it cannot execute $\Pi_{\text{dms}}.\text{Precommit}$ and $\Pi_{\text{dms}}.\text{BlindSign}$ itself.
 - ? In $\mathcal{O}.\text{IssueHonest}(C_{\text{iss}}, \psi, \alpha, ctx)$, $\mathcal{B}$ asserts $\overrightarrow{attr}_{C_{\text{iss}}} \in (\mathbb{A} \cup \{\star\})^n \setminus \mathbb{A}^n$ and $\psi \in \Psi_{\mathcal{G}}[\overrightarrow{attr}_{C_{\text{iss}}}]$ and $\perp \neq \psi(\perp, \alpha)$ (as in $\mathcal{O}.\text{IssueHonest}$). Then, deviating from the usual $\mathcal{O}.\text{IssueHonest}$ behavior, $\mathcal{B}$ calls its oracle $C' \leftarrow \mathcal{O}.\text{PrivateBlindSign}(C_{\text{iss}}, \psi(\perp, \alpha))$, sets $\overrightarrow{attr}_{C'} = \psi(\perp, \alpha)$, and returns C' . ? For credentials issued by honest parties to honest parties, $\mathcal{B}$ simply queries its oracle $\mathcal{O}.\text{PrivateBlindSign}$, which uses the specified signature $cred_{C_{\text{iss}}}$ (not known to $\mathcal{B}$) and does not return the resulting signature $cred_{C'}$ but only the handle C' . Attributes issued this way do not count into $\mathcal{T}_{\text{explained}}$ (in the credential unforgeability game) or $\mathcal{T}_{\text{signed}}$ (in the DMS unforgeability game).
 - ? In $\mathcal{O}.\text{UpdateHonest}(C_{\text{iss}}, C_{\text{upd}}, \psi, \alpha, ctx)$, $\mathcal{B}$ asserts $\overrightarrow{attr}_{C_{\text{iss}}} \in (\mathbb{A} \cup \{\star\})^n \setminus \mathbb{A}^n$ and $\psi \in \Psi_{\mathcal{G}}[\overrightarrow{attr}_{C_{\text{iss}}}]$ and $\perp \neq \psi(\overrightarrow{attr}_{C_{\text{upd}}}, \alpha)$ (as in $\mathcal{O}.\text{UpdateHonest}$). Then, deviating from the usual $\mathcal{O}.\text{UpdateHonest}$ behavior, the reduction adversary $\mathcal{B}$ calls its oracle $C' \leftarrow \mathcal{O}.\text{PrivateBlindSign}(C_{\text{iss}}, \psi(\overrightarrow{attr}_{C_{\text{upd}}}, \alpha))$, sets $\overrightarrow{attr}_{C'} = \psi(\overrightarrow{attr}_{C_{\text{upd}}}, \alpha)$, and returns C' .
 - In $\mathcal{O}.\text{IssueCred}, \mathcal{O}.\text{UpdateCred}$, instead of running $\Pi_{\text{dms}}.\text{Precommit}(pp, ipk^*, \overrightarrow{attr}_{C_{\text{iss}}}, cred_{C_{\text{iss}}})$ (as in Figures 6.6 and 6.7), $\mathcal{B}$ calls its oracle $\mathcal{O}.\text{BlindSign}(C_{\text{iss}})$. ? The oracle outputs a precommit key K to $\mathcal{B}$, which $\mathcal{B}$ then sends to $\mathcal{A}$ as part of the issuance/update protocol. Then, instead of running $\Pi_{\text{dms}}.\text{BlindSign}(pp, ipk^*, K, \overrightarrow{attr}_{C_{\text{iss}}}, C; cred_{C_{\text{iss}}}, k)$, $\mathcal{B}$ outputs $(C, \vec{m}', o)$ to (the same instance of) $\mathcal{O}.\text{BlindSign}$, where C is as sent by $\mathcal{A}$ as part of the Π_{dacs} protocol, and $\vec{m}', o$ are chosen as follows:
 - If there is some o' such that $(ipk^*, K, \overrightarrow{attr}_{C_{\text{iss}}}, C, o') \in \mathcal{T}_{\text{simcommit}}$, then $\mathcal{B}$ uses $\vec{m}' = \vec{m}_{\text{dummy}}(\overrightarrow{attr}_{C_{\text{iss}}})$ and $o = o'$ (where $\mathcal{T}_{\text{simcommit}}, m_{\text{dummy}}$ are read from $state_{\mathcal{S}}$). ? We know how to open the commitment C because our simulator $\mathcal{S}$ has created C . See Figure 6.5.
 - Otherwise, $\mathcal{B}$ uses variables $\vec{m}' = \overrightarrow{attr}'$ and o from the witness w output by $\mathcal{E}_{zk}$. ? Use the opening for C from the proof of knowledge extractor.
- Then $\mathcal{B}$ relays the $\Pi_{\text{dms}}.\text{BlindSign}$ messages between $\mathcal{A}$ and $\mathcal{O}.\text{BlindSign}$.
- If $\text{fail}_{\text{explain}}$ occurs during $\mathcal{O}.\text{UpdateCred}, \mathcal{O}.\text{VerifyCred}$, $\mathcal{B}$ outputs $(\overrightarrow{attr}, cred)$ as a signature forgery (variables $\overrightarrow{attr}, cred$ as in Figures 6.7 and 6.8).

- If $\text{fail}_{\mathcal{E}_{\text{zk}}}$ or $\text{fail}_{\text{dummy}}$ are noted by $\mathcal{E}$ at some point, $\mathcal{B}$ immediately aborts execution.

First, we argue that whenever $\mathcal{B}$ hands $(C, \vec{m}', o)$ to $\mathcal{O}.\text{BlindSign}$, then the VerifyCommit check within $\mathcal{O}.\text{BlindSign}$ succeeds (i.e., $\mathcal{B}$ actually supplies a valid opening for C to $\mathcal{O}.\text{BlindSign}$). For this, consider some $\mathcal{O}.\text{IssueCred}$ or $\mathcal{O}.\text{UpdateCred}$ query. There are two cases: (1) If $\mathcal{S}_{\text{zk}}(\text{stmt}, \text{ctx})$ has been called before, then the DACS simulator $\mathcal{S}$ has created the commitment C for $\text{ipk}^*, K, \vec{\text{attr}}_{\text{C}_{\text{iss}}}$ (as $\mathcal{S}_{\text{zk}}$ is only called within $\mathcal{S}$ and the statement stmt fixes C and $\text{ipk}^*, K, \vec{\text{attr}}_{\text{C}_{\text{iss}}}$), so there is some o' such that $(\text{ipk}^*, K, \vec{\text{attr}}_{\text{C}_{\text{iss}}}, C, o') \in \mathcal{T}_{\text{simcommit}}$. In that setting, correctness of Π_{dms} guarantees that $\text{VerifyCommit}(pp, \text{ipk}^*, K, \vec{\text{attr}}_{\text{C}_{\text{iss}}}, C; \vec{m}', o') = 1$ with $\vec{m}' = \vec{m}_{\text{dummy}}(\vec{\text{attr}}_{\text{C}_{\text{iss}}})$ as required. (2) If $\mathcal{S}_{\text{zk}}(\text{stmt}, \text{ctx})$ has *not* been called at the point where $\mathcal{E}_{\text{zk}}$ outputs w , then necessarily, $(\text{stmt}, w) \in \mathcal{R}$ (because $(\text{stmt}, w) \notin \mathcal{R}$ would have triggered $\text{fail}_{\mathcal{E}_{\text{zk}}}$ and $\mathcal{B}$ would have aborted). In particular, this guarantees that $\Pi_{\text{dms}}.\text{VerifyCommit}(pp, \text{ipk}^*, K, \vec{\text{attr}}_{\text{C}_{\text{iss}}}, C; \vec{\text{attr}}', o) = 1$, where $\vec{\text{attr}}', o$ are taken from w , as required. $\text{Zooming out a bit, we note that case (1) can happen in the following example scenario, which is essentially a relay attack. } \mathcal{A}$ queries $\mathcal{O}.\text{SimReceiveCred}$ and $\mathcal{O}.\text{IssueCred}$, and relays the messages between the simulated user and honest issuer (the same discussion applies for $\mathcal{O}.\text{SimReceiveUpdate}$ and $\mathcal{O}.\text{UpdateCred}$). In this case, the ZKPoK proof, which we would usually extract a commitment opening from (case (2)), is simulated, and consequently cannot be extracted from. However, in this case, the simulator has created the commitment and has left opening information in $\mathcal{T}_{\text{simcommit}}$ that $\mathcal{B}$ can use.

As a consequence, unless $\text{fail}_{\mathcal{E}_{\text{zk}}}$ or $\text{fail}_{\text{dummy}}$ happen (in which case $\mathcal{B}$ would prematurely abort), $\mathcal{B}$ perfectly simulates $\text{Exp}_{\Pi_{\text{dacs}}}^{\text{sim-expl}}(\Pi_{\text{dacs}}, \mathcal{A}, \mathcal{S}, \mathcal{E}, n, \lambda)$, i.e., exactly the same algorithms are run with the same input in $\text{Exp}_{\Pi_{\text{dacs}}}^{\text{sim-expl}}(\Pi_{\text{dacs}}, \mathcal{A}, \mathcal{S}, \mathcal{E}, n, \lambda)$ as in the unforgeability game (Definition 6.8) with $\mathcal{B}$, just some of those computations are now happening within the $\mathcal{O}.\text{PrivateBlindSign}, \mathcal{O}.\text{BlindSign}$ oracles of $\mathcal{B}$. As a consequence, $\Pr[\neg \text{fail}_{\mathcal{E}_{\text{zk}}} \wedge \neg \text{fail}_{\text{dummy}} \wedge \text{fail}_{\text{explain}} \mid \mid \text{Exp}_{\Pi_{\text{dacs}}}^{\text{sim-expl}}(\Pi_{\text{dacs}}, \mathcal{A}, \mathcal{S}, \mathcal{E}, n, \lambda)] = \Pr[\neg \text{fail}_{\mathcal{E}_{\text{zk}}} \wedge \neg \text{fail}_{\text{dummy}} \wedge \text{fail}_{\text{explain}} \mid \mid \mathcal{B}]$.

We now argue that when $\text{fail}_{\text{explain}}$ occurs in $\mathcal{B}$, then $\mathcal{B}$ indeed outputs a forgery in the DMS unforgeability game (Definition 6.8). For this, first note the following facts about $\mathcal{B}$ and its variable values in $\mathcal{O}.\text{IssueCred}, \mathcal{O}.\text{UpdateCred}$, or $\mathcal{O}.\text{VerifyCred}$ that can be easily checked by inspection of Figures 6.6 to 6.8.

1. If the event $\text{fail}_{\text{explain}}$ is raised during $\mathcal{O}.\text{IssueCred}, \mathcal{O}.\text{UpdateCred}$, or $\mathcal{O}.\text{VerifyCred}$, then $(\text{stmt}, w) \in \mathcal{R}$. $\text{Zooming out a bit, we note that case (1) can happen in the following example scenario, which is essentially a relay attack. } \mathcal{A}$ queries $\mathcal{O}.\text{SimReceiveCred}$ and $\mathcal{O}.\text{IssueCred}$, and relays the messages between the simulated user and honest issuer (the same discussion applies for $\mathcal{O}.\text{SimReceiveUpdate}$ and $\mathcal{O}.\text{UpdateCred}$). In this case, the ZKPoK proof, which we would usually extract a commitment opening from (case (2)), is simulated, and consequently cannot be extracted from. However, in this case, the simulator has created the commitment and has left opening information in $\mathcal{T}_{\text{simcommit}}$ that $\mathcal{B}$ can use. When $\text{fail}_{\text{explain}}$ is raised, $\text{fail}_{\mathcal{E}_{\text{zk}}}$ has not occurred (otherwise, $\mathcal{B}$ would have aborted earlier) and $\mathcal{S}_{\text{zk}}(\text{stmt}, \text{ctx})$ has not been called (because of the $\mathcal{T}_{\text{sim}}$ check). Hence necessarily, $(\text{stmt}, w) \in \mathcal{R}$.
2. $\text{fail}_{\text{explain}}$ cannot occur during $\mathcal{O}.\text{IssueCred}$ queries. $\text{Zooming out a bit, we note that case (1) can happen in the following example scenario, which is essentially a relay attack. } \mathcal{A}$ queries $\mathcal{O}.\text{SimReceiveCred}$ and $\mathcal{O}.\text{IssueCred}$, and relays the messages between the simulated user and honest issuer (the same discussion applies for $\mathcal{O}.\text{SimReceiveUpdate}$ and $\mathcal{O}.\text{UpdateCred}$). In this case, the ZKPoK proof, which we would usually extract a commitment opening from (case (2)), is simulated, and consequently cannot be extracted from. However, in this case, the simulator has created the commitment and has left opening information in $\mathcal{T}_{\text{simcommit}}$ that $\mathcal{B}$ can use. $\text{fail}_{\text{explain}}$ would imply $(\text{stmt}, w) \in \mathcal{R}$ (see point 1), which implies $\psi(\perp, \alpha) \neq \perp$, which would mean that explained is set to 1, contradicting $\text{fail}_{\text{explain}}$.
3. If $\text{fail}_{\text{explain}}$ happens during $\mathcal{O}.\text{UpdateCred}$ or $\mathcal{O}.\text{VerifyCred}$, then cred is a valid signature on $\vec{\text{attr}}$, i.e., $\text{Verify}(pp, \text{ipk}^*, \vec{\text{attr}}, \text{cred}) = 1$. $\text{Zooming out a bit, we note that case (1) can happen in the following example scenario, which is essentially a relay attack. } \mathcal{A}$ queries $\mathcal{O}.\text{SimReceiveCred}$ and $\mathcal{O}.\text{IssueCred}$, and relays the messages between the simulated user and honest issuer (the same discussion applies for $\mathcal{O}.\text{SimReceiveUpdate}$ and $\mathcal{O}.\text{UpdateCred}$). In this case, the ZKPoK proof, which we would usually extract a commitment opening from (case (2)), is simulated, and consequently cannot be extracted from. However, in this case, the simulator has created the commitment and has left opening information in $\mathcal{T}_{\text{simcommit}}$ that $\mathcal{B}$ can use. This follows from point 1. Note for

$\mathcal{O}.\text{VerifyCred}$ that, due to point 1, we necessarily have that $ipk = ipk^*$ whenever $\text{fail}_{\text{explain}}$ happens.

4. If $\text{fail}_{\text{explain}}$ happens during $\mathcal{O}.\text{UpdateCred}$ or $\mathcal{O}.\text{VerifyCred}$, then $\vec{attr} \not\leq \mathcal{T}_{\text{explained}}$.
 ! Because of point 1, the conditions $\psi(\vec{attr}, \alpha) \neq \perp$ or $\phi(\vec{attr}, \alpha) = 1$ are necessarily fulfilled. So the only way for $\text{fail}_{\text{explain}}$ to occur is by falsifying the $\vec{attr} \leq \mathcal{T}_{\text{explained}}$ condition.
5. $\mathcal{T}_{\text{signed}} \subseteq \mathcal{T}_{\text{explained}} \cup \{\vec{attr} \in \mathbb{A}^n \mid \vec{attr} \text{ contains } m_{\text{dummy}}\}$. ! By definition, $\mathcal{B}$ only queries $\mathcal{O}.\text{BlindSign}$ with $\vec{m}_{\text{dummy}}(\cdot)$ (which contains m_{dummy}) and with $\vec{attr}'$ output by the extractor $\mathcal{E}_{\text{zk}}$ during $\mathcal{O}.\text{IssueCred}$ or $\mathcal{O}.\text{UpdateCred}$. In the latter case, right before outputting $(\vec{m}', o)$ to $\mathcal{O}.\text{BlindSign}$, $\mathcal{B}$ adds $\vec{m}' = \vec{attr}'$ to $\mathcal{T}_{\text{explained}}$. ! Here, we use our demand that the issuer's attributes $\vec{attr}_{\mathcal{C}_{\text{iss}}}$ contain at least one $\star$ in $\mathcal{O}.\text{IssueCred}$, and $\mathcal{O}.\text{UpdateCred}$. This ensures that $\{\vec{attr} \in \mathbb{A}^n \mid \vec{attr} \text{ contains } m_{\text{dummy}}\}$ indeed covers all of $\mathcal{B}$'s queries to $\mathcal{O}.\text{BlindSign}$ in case the adversary's commitment C was simulated.

So whenever $\text{fail}_{\text{explain}}$ happens, $\mathcal{B}$ outputs a candidate forgery $(\vec{attr}, cred)$, of which we know that $\text{Verify}(pp, ipk^*, \vec{attr}, cred) = 1$ (point 3). Furthermore, from point 4, we get that $\vec{attr} \not\leq \mathcal{T}_{\text{explained}}$. Because of $\neg \text{fail}_{\text{dummy}}$ ($\mathcal{B}$ would have aborted earlier, otherwise), we know that $\vec{attr}$ does not contain m_{dummy} . Together with point 5, this implies that $\vec{attr} \notin \mathcal{T}_{\text{signed}}$.

Overall, whenever $\text{fail}_{\text{explain}} \wedge \neg \text{fail}_{\mathcal{E}_{\text{zk}}} \wedge \neg \text{fail}_{\text{dummy}}$, then $\mathcal{B}$ outputs a valid forgery on an unqueried message in the DMS unforgeability game. Because $\mathcal{B}$ is a ppt, this means that $\Pr[\text{fail}_{\text{explain}} \wedge \neg \text{fail}_{\mathcal{E}_{\text{zk}}} \wedge \neg \text{fail}_{\text{dummy}}]$ must be negligible.

Conclusion. Overall, applying basic probability theory (law of total probability, basic inequalities) to the analysis of $\text{Exp}_{\Pi_{\text{dacs}}}^{\text{sim-expl}}(\Pi_{\text{dacs}}, \mathcal{A}, \mathcal{S}, \mathcal{E}, n, \lambda)$, we get $\Pr[\text{fail}_{\text{explain}}] \leq \Pr[\text{fail}_{\text{explain}} \wedge \neg \text{fail}_{\mathcal{E}_{\text{zk}}} \wedge \neg \text{fail}_{\text{dummy}}] + \Pr[\text{fail}_{\mathcal{E}_{\text{zk}}}] + \Pr[\text{fail}_{\text{dummy}}]$. We have shown above that all three summands are negligible, hence $\Pr[\text{fail}_{\text{explain}}]$ is negligible. $\square$

6.4.3.1 The case of delegation where the issuer does not have $\star$ attributes

In the syntax of DACS (Definition 6.2), we have restricted issuance and updating to cases where the issuer's attributes $\vec{attr}_{\text{iss}}$ contain at least one $\star$. This covers most use cases: Delegation without $\star$ is somewhat trivial, as the only attributes that issuer can issue are $\vec{attr}' = \vec{attr}_{\text{iss}}$. Intuitively (with some caveats discussed below), this can be achieved by just handing the issuer's credential $cred_{\text{iss}}$ with attributes $\vec{attr}_{\text{iss}}$ to the user.

Understanding the restriction. Let us first discuss why this restriction is necessary. For this, consider the following attack against simulation-explainability of our construction Π_{dacs} : The adversary does not hold any credentials. It calls $\mathcal{O}.\text{IssueCred}(\mathcal{C}_{\text{iss}}, \psi, ctx)$ to cause an honest issuer to issue (delegate) a credential, and $\mathcal{O}.\text{SimReceiveCred}(ipk^*, \vec{attr}_{\mathcal{C}_{\text{iss}}}, \psi, ctx)$ to simulate an honest receiver's messages. The adversary relays messages between the IssueCred protocol and the simulated ReceiveCred protocol run within the oracles. This models a relay attack, where the adversary uses an honest user's permission

to receive a credential that the adversary does not have permission to receive.

As a result of this behavior, the adversary learns the full transcript of the DMS blind issuance protocol. We need to ensure that the adversary does not learn the issued signature (as there would not be an explanation for it, given that $\mathcal{E}$ generally cannot extract from the simulated proof). This is easily ensured in our construction: Since the Π_{dms} blind issuance protocol has signing blindness, the adversary cannot distinguish the transcript for blind issuance of any real $\vec{attr}' \preceq \vec{attr}_{\text{iss}}$ from blind issuance of $\vec{m}_{\text{dummy}}(\vec{attr}_{\text{iss}})$. This is how we build the DACS simulator: It simply requests $\vec{m}_{\text{dummy}}(\vec{attr}_{\text{iss}})$. Then in the proof of Theorem 6.19, we essentially argue that the adversary cannot make use of those blind issuance transcripts because it cannot guess m_{dummy} (see the $\text{fail}_{\text{dummy}}$ event). Overall, with this argumentation, the attack of the adversary described above is harmless.

However, this argumentation crucially requires that m_{dummy} be part of all messages $\vec{m}_{\text{dummy}}(\vec{attr}_{\text{iss}})$ requested by the simulator. This is *not* the case if $\vec{attr}_{\text{iss}}$ does not contain any $\star$ values. If $\vec{attr}_{\text{iss}}$ does not contain $\star$, the only attributes that can be issued are $\vec{attr}' = \vec{attr}_{\text{iss}}$, and so the DMS blindness definition (Definition 6.9) becomes trivial. Indeed, the DMS blind issuance protocol may as well send the signature $cred_{\text{iss}}$ in plain in this case. As a consequence, the adversary (potentially) learns the signature resulting from the blind issuance protocol, as well as its signed message $\vec{attr}_{\text{iss}}$, meaning the adversary is now in possession of an unexplained credential on $\vec{attr}_{\text{iss}}$ (the explainer $\mathcal{E}$ cannot register $\vec{attr}_{\text{iss}}$ into $\mathcal{T}_{\text{explained}}$ as it cannot extract from the simulated zero-knowledge proof). This winning condition exists for good reason: We want to model that the adversary cannot benefit from such relay attacks.

Overall, the restriction that $\vec{attr}_{\text{iss}}$ must contain at least one $\star$ attribute is needed for our argumentation that the adversary cannot mount relay attacks.

Workarounds. If the application requires the ability to issue/update with $\vec{attr}_{\text{iss}}$ not containing $\star$, we suggest the following way to realize it. The user proves that he is allowed to receive $\vec{attr}_{\text{iss}}$ (for issue: via zero-knowledge proof; for update: via credential presentation). If so, then the issuer simply sends her credential $cred_{\text{iss}}$ to the user. This solution has the potential drawback of missing delegator anonymity (Definition 6.4), as the $cred_{\text{iss}}$ revealed is the same for every such interaction with the same issuer. Additionally, this solution is vulnerable to relay-style attacks as discussed above: The credential $cred_{\text{iss}}$ is usable by an adversary who relays the proof/presentation messages between the issuer and a user authorized to receive $\vec{attr}_{\text{iss}}$. In practice, this can be mitigated by using the relay attack mitigations discussed in Section 3.3.2, namely having ctx be structured appropriately, making it hard to convince an honest user to run the protocol with the adversary for a ctx that would also be accepted by the issuer.

A second, generic solution, is to change the format of credentials to always contain an artificial $\star$ attribute. Specifically, instead of attributes $\vec{attr} \in (\mathbb{A} \cup \{\star\})^n$, one would use $(\vec{attr}, \star) \in (\mathbb{A} \cup \{\star\})^{n+1}$. In that setting, all credentials have one $\star$ attribute, and hence can be securely delegated.

Finally, another solution is to change the construction to account for this special case.

Roughly sketched, user and issuer can establish a secure channel (e.g., by having the user include their share of a Diffie-Hellman key exchange in ctx). The issuer would send their blind issuance messages via the secure channel, which means that the adversary cannot gain information about the signature. If $\overrightarrow{attr}_{iss} = \star$, no such secure channel is needed, as the blind issuance protocol already sufficiently hides the signature, as argued above.

6.4.4 Instantiating the standard template

As far as we are aware, there are no other proposals for instantiating DMS signatures (Definition 6.6) other than our construction Π_{ps-dms} (Construction 6.11) based on Pointcheval-Sanders signatures. It seems plausible that one can construct DMS from the SPS-UC scheme in [MSBM23], but we leave this for future work and discuss the instantiation using our Π_{ps-dms} in the following. It remains to discuss how to instantiate the required zero-knowledge proofs of Construction 6.16.

We focus on the following proof system proving knowledge of a valid Π_{ps-dms} signature

$$\text{ZKPoK} \left[\left(\overrightarrow{attr}, cred \right); \Pi_{ps-dms}.Verify(pp, ipk, \overrightarrow{attr}, cred) = 1 \right] (ctx),$$

and the following proof system proving knowledge of a valid commitment opening

$$\text{ZKPoK} \left[\left(\overrightarrow{attr}', o \right); \Pi_{ps-dms}.VerifyCommit(pp, ipk, K, \overrightarrow{attr}_{iss}, C; \overrightarrow{attr}', o) \right] (ctx).$$

The proof systems in Construction 6.16 then follow easily using the tools in Chapter 3.

Encoding (star) attributes. First, for both proof systems, we need a way to encode attributes $attr_i \in \mathbb{Z}_p \cup \{\star\}$ in a way that is compatible with Schnorr-style proofs. Notably, $\mathbb{Z}_p \cup \{\star\}$ is not a prime order group, and it is unclear how $\star$ should be treated algebraically. The encoding must allow us to (1) efficiently prove typical policies ϕ and attribute updates ψ and (2) efficiently instantiate the two proof systems mentioned above. Our solution is to encode a single attribute $attr_i \in \mathbb{Z}_p \cup \{\star\}$ as a tuple

$$(\omega_i, s_i) \in \mathbb{Z}_p^2$$

that we interpret as

$$attr_i = \begin{cases} \omega_i & \text{if } s_i = 0 \\ \star & \text{if } s_i \neq 0 \end{cases}$$

Intuitively, we may interpret $s_i \in \mathbb{Z}_p$ as a C-style boolean value indicating whether $attr_i = \star$ (0 means false, any other value means true). For honest users, $s_i \in \{0, 1\}$, but we allow adversarially generated proofs to use any $s_i \in \mathbb{Z}_p$. This means that the proof system itself will not enforce $s_i \in \{0, 1\}$. Similarly, honest users will set $\omega_i = 0$ if $s_i \neq 0$ (note that in that case $attr_i = \star$ and the value of ω_i is effectively ignored anyway), but again, this is not enforced by the proof system. Enforcing these constraints within the proof system would represent a significant performance loss, as discussed below.

In our proof systems for Construction 6.16, we exclusively use encoded attributes $(\omega_i, s_i), (\omega'_i, s'_i)$ in place of attributes $attr_i, attr'_i$. This means statements like $\phi(\overrightarrow{attr}) = 1$ or $\psi(\overrightarrow{attr}, \alpha) = \overrightarrow{attr}'$, as in Construction 6.16 need to be rephrased as predicates over *encodings*, i.e., $\phi(\vec{\omega}, \vec{s}) = 1$ or $\psi(\vec{\omega}, \vec{s}, \alpha) = (\vec{\omega}', \vec{s}')$. The translation is straightforward. Some representative examples:

- $attr_i = 17$ translates to $\omega_i = 17 \wedge s_i = 0$.
- $attr_i = \star$ translates to $s_i = 1$.
- $attr_i \neq \star$ translates to $s_i = 0$.
- $attr_i = attr'_i$ translates to $s_i = s'_i \wedge \omega_i = \omega'_i$.

Note that some care needs to be taken when negating these statements as the encoding is not one-to-one. For example, $attr_i = \star$ translates to $s_i = 1$, but $attr_i \neq \star$ does not translate to $s_i \neq 1$ (as $s_i \neq 1$ does not rule out, say, $s_i = 2$, which would still be interpreted as $attr_i = \star$). Nevertheless, our encoding makes it easy to prove statements about $\overrightarrow{attr}, \overrightarrow{attr}'$ efficiently. In particular, the convention that honest users honor an even stricter encoding allows phrasing statements more efficiently (e.g., honest users for $attr_i = \star$ will always set $s_i = 1$, which is much easier to check in a Schnorr-style protocol than the more permissive $s_i \neq 0$).

Proving knowledge of a valid commitment opening. For the VerifyCommit algorithm of Π_{ps-dms} , the encoding $(\vec{\omega}', \vec{s}')$ for $\overrightarrow{attr}'$ coincides with the commitment opening $o = (o_{\text{fix}}, (o_i, \omega'_i, s'_i)_{i: m_i^{(\text{iss})} = \star})$. Hence the opening o already encodes the committed attributes $\overrightarrow{attr}'$. The proof system we need to instantiate is

$$\text{ZKPoK} \left[\left(o_{\text{fix}}, (o_i, \omega'_i, s'_i)_{i: m_i^{(\text{iss})} = \star} \right); \quad C_{\text{fix}} = K_0^{o_{\text{fix}}} \cdot \prod_{i: m_i^{(\text{iss})} = \star} K_i^{\omega'_i} \right. \\ \left. \wedge \quad \forall i \in \{j \mid m_j^{(\text{iss})} = \star\} : C_i = K_i^{s'_i} \cdot K_0^{o_i} \right] (ctx),$$

where $\vec{m}^{(\text{iss})}$ are the issuer's attributes, $K = (K_0, (K_i)_{i: m_i^{(\text{iss})} = \star}) \in (\mathbb{G}_1 \setminus \{1\})^{1+|\{i \mid m_i^{(\text{iss})} = \star\}|}$ is the commitment key and $C = (C_{\text{fix}}, (C_i)_{i: m_i^{(\text{iss})} = \star})$ is the commitment. Because all equations' right-hand sides are linear in the witness, we can trivially design a corresponding generalized Schnorr protocol (Construction 3.7). We leave the straightforward details to the reader.

Note that the protocol above is not particularly meaningful by itself as it does not enforce any limits on the committed message encoded by $\vec{\omega}', \vec{s}'$. Hence the protocol is a building block, meant to be extended with additional statements to instantiate the proof systems required by Construction 6.16.

Note that in contrast to updatable credentials (Construction 5.7), where the proof systems check that the commitment is exactly output of **Commit**, for delegatable credentials we have introduced the **VerifyCommit** procedure, which checks some more lenient statement about the commitment. Here is where this decision pays off: It makes

the proof system for proving knowledge of a valid commitment opening much simpler. If we were to instead implement a proof system for the stricter statement that $C = (C_{\text{fix}}, (C_i)_{i: m_i^{(\text{iss})} = \star})$ is plausible output of $\Pi_{\text{ps-dms}} \cdot \text{Commit}$, that would require to additionally prove $(s'_i \neq 0 \Rightarrow \omega'_i = 0)$ and $s'_i \in \{0, 1\}$ for all i with $m_i^{(\text{iss})} = \star$. Implementing these additional checks in a Schnorr-style proof is not impossible (e.g., using OR proofs, see Section 3.2.5) but comes with significant overhead. Luckily, as proven in Theorem 6.13, $\Pi_{\text{ps-dms}}$ is still unforgeable if we employ the more lenient `VerifyCommit` commitment verification, hence we enjoy much more efficient and straightforward proof systems.

Proving knowledge of a DMS signature. Using the encoding for $\overrightarrow{\text{attr}}$ above and the concrete `Verify` algorithm of $\Pi_{\text{ps-dms}}$, the proof system we need to instantiate to prove knowledge of a DMS signature $\text{cred} = (h, \mu, (\gamma_i)_{i: s_i \neq 0})$ over encoded attributes $\vec{\omega}, \vec{s}$ is

$$\text{ZKPoK} \left[\left(\begin{array}{c} \vec{\omega}, \vec{s}, \\ (h, \mu, (\gamma_i)_{i: s_i \neq 0}) \end{array} \right); \begin{array}{l} h \neq 1 \\ e(\mu, \tilde{g}) = e(h, \tilde{X} \cdot \prod_{i: s_i=0} \tilde{Y}_i^{\omega_i}) \\ \wedge \quad \forall i \in \{j \mid s_j \neq 0\} : e(\gamma_i, \tilde{g}) = e(h, \tilde{Y}_i) \end{array} \right] (\text{ctx}).$$

Again, such a proof system is not particularly meaningful by itself as it does not enforce any limits on the encoded signed message $\vec{\omega}, \vec{s}$. Hence the protocol is a building block, meant to be extended with additional statements to instantiate the proof systems required by Construction 6.16.

There are two challenges when instantiating this proof system. First, looking at the first pairing equation, the $\tilde{Y}_i$ terms in the product are filtered by $s = 0$, i.e., we expect a valid signature not to contain any $h^{y_i \omega_i}$ terms for indices i with $\text{attr}_i = \star$. Second, we must only check the γ_i pairing equation for those indices i where $s_i \neq 0$, i.e., $\text{attr}_i = \star$.

Thankfully, these challenges are easily solved. First, we simply include *all* $\tilde{Y}_i^{\omega_i}$ terms in the first pairing equation. When extracting a witness, this means we get μ with some extra $h^{y_i \omega_i}$ terms that should not be there. However, these extra terms occur for indices where $\text{attr}_i = \star$, and hence this part of the signature is malleable by design (as discussed in Section 6.3.4), so the extractor can easily remove these extra terms using the extracted malleability enablers γ_i . Second, instead of checking $e(\gamma_i, \tilde{g}) = e(h, \tilde{Y}_i)$ only for indices where $s_i \neq 0$, we check $e(\gamma_i, \tilde{g}) = e(h, \tilde{Y}_i^{s_i})$ for all indices $i \in [n]$. So for $s_i = 0$, the prover can trivially fulfill the check by setting $\gamma_i = 1$ (note that $s_i = 0$ implies $\text{attr}_i \neq \star$, so there is no need to extract a meaningful γ_i). For $s_i \neq 0$, we extract $\gamma_i^{s_i}$, from which the extractor can easily compute γ_i as desired.

Armed with these ideas, we can conveniently design the following send-first Sigma protocol to prove knowledge of a valid DMS signature on hidden messages.

Construction 6.20 (Proving knowledge of a valid $\Pi_{\text{ps-dms}}$ signatures on hidden messages). Figure 6.9 depicts a send-first Sigma protocol (Construction 3.12) for the relation $\mathcal{R}_{\mathcal{G}} = \{(stmt = (\Pi_{\text{dms}}.pp, pk), w = (\vec{\omega}, \vec{s}, cred = (h, \mu, (\gamma_i)_{i: s_i \neq 0}))) \mid \Pi_{\text{dms}}.pp = (\mathcal{G}, g, \tilde{g}) \wedge g \in \mathbb{G}_1 \wedge \tilde{g} \in \mathbb{G}_2 \setminus \{1\} \wedge pk = (\tilde{X}, (Y_i)_{i=1}^n, (\tilde{Y}_i)_{i=1}^n) \in \mathbb{G}_2 \times \mathbb{G}_1^n \times \mathbb{G}_2^n \wedge h \neq 1 \wedge e(\mu, \tilde{g}) = e(h, \tilde{X} \cdot \prod_{i: s_i=0} \tilde{Y}_i^{\text{attr}_i}) \wedge \forall i \in \{j \mid s_j \neq 0\} : e(\gamma_i, \tilde{g}) = e(h, \tilde{Y}_i)\}$. It does not

require additional setup, i.e., $\text{SetupFirst}(\mathcal{G}) = \perp$.

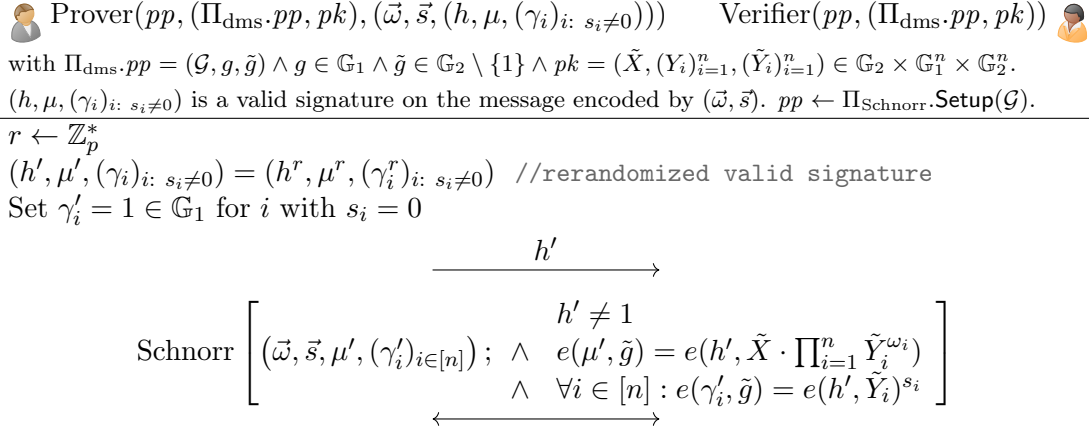


Figure 6.9: A send-first protocol (Construction 3.12) to prove knowledge of a signature on hidden messages $\vec{attr} \in (\mathbb{A} \cup \{\star\})^n$ encoded by $\vec{\omega}, \vec{s} \in \mathbb{Z}_p^n$.

To clarify the Schnorr homomorphism for the inner protocol, note that the expression $e(\mu', \tilde{g}) = e(h', \tilde{X} \cdot \prod_{i=1}^n \tilde{Y}_i^{\omega_i})$ can be equivalently written as $e(\mu', \tilde{g}) \cdot e(h', \prod_{i=1}^n \tilde{Y}_i^{-\omega_i}) = e(h', \tilde{X})$, which makes the left-hand side of the equation a homomorphism over $\mu', \vec{\omega}$ and the right-hand side a constant (constant in the context of the Schnorr protocol/homomorphism). Similarly, we can write the expression $e(\gamma'_i, \tilde{g}) = e(h', \tilde{Y}_i)^{s_i}$ equivalently as $e(\gamma'_i, \tilde{g}) \cdot e(h', \tilde{Y}_i)^{-s_i} = 1$, which makes the left-hand side a homomorphism over γ'_i, s_i . We also note that the “ $h' \neq 1$ ” part of the Schnorr statement can be checked in plain by the verifier and hence can be handled outside the homomorphism.

It is easy to see that Construction 6.20 is special honest-verifier zero-knowledge: We can apply Theorem 3.13, using a first message simulator $\text{SimFirst}(pp, stmt)$ outputs a random $h' \leftarrow \mathbb{G}_1 \setminus \{1\}$. For (computational) special soundness, we can apply Theorem 3.14 with the inner-protocol witness interpreter InterpretWitness following the intuition above. Concretely, $\text{InterpretWitness}(pp, stmt, h', w' = (\vec{\omega}, \vec{s}, \mu', (\gamma'_i)_{i \in [n]}))$ takes the inner protocol’s witness w' and outputs translated witness $w = (\vec{\omega}, \vec{s}, cred = (h, \mu, (\gamma_i)_{i: s_i \neq 0}))$ for the outer protocol, where

- $h = h'$
- $\gamma_i = (\gamma'_i)^{1/s_i}$ for i with $s_i \neq 0$
- $\mu = \mu' \cdot \prod_{i: s_i \neq 0} \gamma_i^{-\omega_i}$.

It is easy to check that InterpretWitness , given a valid witness for the inner protocol, always outputs a valid witness for the outer protocol. Concretely, we get $h = h' \neq 1$ directly from the inner protocol. We also get for all i where $s_i \neq 0$,

$$e(\gamma'_i, \tilde{g}) = e(h', \tilde{Y}_i)^{s_i}$$

💡 Plugging in InterpretWitness definitions:

$$\Rightarrow e(\underline{\gamma_i^{s_i}}, \tilde{g}) = e(\underline{h}, \tilde{Y_i})^{s_i}$$

💡 s_i cancels out (using bilinearity):

$$\Rightarrow e(\gamma_i, \tilde{g}) = e(h, \tilde{Y_i}),$$

meaning that the γ_i requirement on the interpreted witness is fulfilled. This implies that $\gamma_i = h^{y_i}$, where y_i is the discrete logarithm of $\tilde{Y_i} = \tilde{g}^{y_i}$. Furthermore,

$$e(\mu', \tilde{g}) = e(h', \tilde{X} \cdot \prod_{i=1}^n \tilde{Y_i}^{\omega_i})$$

💡 Plugging in InterpretWitness definitions:

$$\Rightarrow e(\mu \cdot \prod_{i: s_i \neq 0} \gamma_i^{\omega_i}, \tilde{g}) = e(h, \tilde{X} \cdot \prod_{i=1}^n \tilde{Y_i}^{\omega_i})$$

💡 Bilinearity:

$$\Rightarrow e(\mu, \tilde{g}) \cdot \prod_{i: s_i \neq 0} e(\gamma_i, \tilde{g})^{\omega_i} = e(h, \tilde{X} \cdot \prod_{i: s_i = 0} \tilde{Y_i}^{\omega_i}) \cdot \prod_{i: s_i \neq 0} e(h, \tilde{Y_i})^{\omega_i}$$

💡 Using $e(\gamma_i, \tilde{g}) = e(h, \tilde{Y_i})$ from above to cancel out these terms:

$$\Rightarrow e(\mu, \tilde{g}) = e(h, \tilde{X} \cdot \prod_{i: s_i = 0} \tilde{Y_i}^{\omega_i}),$$

meaning that the μ requirement on the interpreted witness is fulfilled.

Overall, this establishes that Construction 6.20 is a secure Sigma protocol. Using the techniques described in Chapter 3, one can easily extend it with other statements (to prove statements about the encoded attributes and to prove other statements needed to instantiate our delegatable credential scheme (Construction 6.16)), and finally upgrade the Sigma protocol to a proof system with stronger security, e.g., using Constructions 3.15 and 3.23.

Notes on optimization. We discuss several practical optimizations for our instantiation of Construction 6.16 explained above.

When it comes to issuing/updating credentials, recall that the commitment key K_i , the commitment C_i , and finally the issuer's response $\gamma_{\text{blind},i}$ are only relevant in case the user is asking to receive a credential where $\text{attr}'_i = \star$. If $\text{attr}'_i \neq \star$, then $\gamma_{\text{blind},i}$ will be useless to the user. In a general execution, we want to hide whether $\text{attr}'_i = \star$ from the issuer. However, if the scenario is such that the issuer a priori knows that $\text{attr}_i \neq \star$, e.g., ψ is such that it will never output $\text{attr}_i = \star$, then issuer and user can skip sending K_i , C_i , and $\gamma_{\text{blind},i}$, as well as skip the related statements within the proof systems above.

When it comes to proving knowledge of a signature with attributes attr_i as in Construction 6.20, be it for update or for presentation, we can apply some straightforward optimization. Namely, if the verifier knows a priori that $\text{attr}_i \neq \star$ (i.e., $s_i = 0$), then the prover can stop including γ'_i, s_i as a witness and skip the corresponding pairing equation

(improving both proof size and computation complexity). Notably, in some settings, this situation can be created by replacing any $attr_i = \star$ requirements in ϕ or ψ with the requirement that $attr_i = \mathbf{a}$ for some attribute value $\mathbf{a} \in \mathbb{Z}_p$ that by design of the application is never issued. In that setting, the only way to possess a credential with $attr_i = \mathbf{a}$ is by having one with $attr_i = \star$ and internally changing it to $attr_i = \mathbf{a}$.

Chapter 7

Conclusion

Difficult to see. Always in motion is the future.

Yoda

In this thesis, we have introduced, defined, and constructed updatable and delegatable anonymous credentials. Furthermore, we have provided background around zero-knowledge proof systems and traditional anonymous credentials. We conclude the thesis with some reflections on design decisions, thoughts on societal impact of real-world anonymous credentials, and possible avenues for future work.

Why would corporations choose to deploy privacy-preserving cryptography?

People working on privacy-preserving cryptography get asked one question again and again: What is the point of developing these systems when corporations *do not want* privacy for users? And indeed, it is generally against most corporations' interests to forgo collecting user data. Data is valuable, privacy-preserving systems are much harder to implement, and they make errors harder to fix.

First, we note that corporations in practice *do* sometimes opt for privacy, even sacrificing data collection opportunities. For example, there are many instant messenger services that offer end-to-end encryption. Signal even offers sealed sender functionality and a form of anonymous credentials for chat group membership [CPZ20], both of which hide metadata from Signal servers. Apple offers end-to-end encryption for their cloud services. CDNs like Cloudflare deploy Privacy Pass [DGSTV18] to offer a privacy-preserving way for users to skip CAPTCHAs. This casts doubt on the idea that corporations will behave purely in ways that maximize data collection. In practice, the incentives are more complicated than that.

Second, it is worth emphasizing that the role of academia is not restricted to producing systems that neatly fit into the current incentive structures of corporations. An important strength of academia is its freedom to explore concepts independent of perceived immediate usefulness. The freedom to think outside the box. Ultimately, the role of academia encompasses the development of new visions for how users and organizations interact, as Chaum did when devising the idea of anonymous credentials [Cha84; Cha85].

It is then the role of academia to deeply understand such ideas. In particular, it is important to understand and make accessible their technical and theoretical aspects, which this thesis contributes to, and to test their limits and further develop their capabilities, which this thesis also contributes to.

Finally, the body of academic results can inform policy and regulation: Once a technology is well-understood and mature enough for realistic deployment, it is either picked up by corporations (if it happens to fit their incentives and business models) or its deployment is demanded by lawmakers (if it furthers the common good). In the case of anonymous credentials, indeed, there is little incentive for (most) corporations to deploy them, but research can set the standard for what is considered state-of-the-art for security and privacy, which then enables laws to enforce such standards.

So what is the answer to our opening question? Why would corporations ever choose to deploy privacy-preserving cryptography when it is against their immediate interests? It is for the same reasons corporations may choose to use environmentally friendly or health-conscious processes and materials that result in higher cost, lower yield, or diminished customer satisfaction: They do so due to public (or internal) pressure and regulation.

Potential downsides of widely deployed anonymous credential systems. Let us imagine a future where anonymous credentials are widely deployed. Citizens are issued a digital ID in the form of an anonymous credential that is conveniently integrated into personal smartphones and can be used in many places with little friction. Strong privacy laws mandate that organizations accept anonymous credentials and identify users by their pseudonyms. At first glance, one may assume that this future improves the situation for users over the current state of the world in every conceivable way. This is not necessarily the case. We point out several threats.

First, if users have the ability to present their ID with little friction, then this enables *requiring* an ID in more places. For example, right now, one can generally sign up to online services without any form of government ID. An email address (which can be somewhat ephemeral) and a self-reported name are all one needs to sign up. If there is an effective mechanism to present government ID that is convenient enough so organizations do not lose customers by requiring it, then having an ID may become a requirement for signing up online. Of course, if the organization requests more information than is reasonable, this is a privacy risk. This is exacerbated by unforgeability of credentials: Contrary to today, one would not be able to register using fake data.

If privacy is strongly respected by the verifying organization, then this kind of signup is a big win for user privacy, and it may even have a positive effect for organizations, as they can cut down on fake accounts, and they get stronger authenticity guarantees than today.

However, even if privacy is perfectly respected, there is another potential downside: People without valid ID lose access to those services that now require ID. This puts a lot of power into the hands of ID issuers: If ID requirements are pervasive, then revoking an ID effectively bans the user from participating in society. Hence the set of issuers needs to be carefully considered. Nation states, in principle, already have the

ability to remove people from society. There are generally systems in place to restrict this ability. Still, widely deployed state-issued anonymous credentials create an infrastructure where banning (whole groups of) people from participating in society can conceivably be implemented very efficiently and quietly.

All this is not to argue that large-scale deployment of anonymous credentials is always a bad idea. Anonymous credentials are a wonderful tool that enables new trade-offs between anonymity and accountability that can be tremendously beneficial to society. They are, however, not a silver bullet. Certain design decisions may lead to unintended outcomes. One should advocate for careful, deliberate, and well-thought-out deployment.

Potential downsides of anonymity. If anonymous online forums are any indication, anonymity tends to provoke a rather unpleasant side of humanity. We seem to have a hard time empathizing with anonymous (or even just unfamiliar) people. Anonymity is also often a shield against consequences for antisocial behavior, and it provides opportunity for undetected illegal behavior.

When designing systems where users anonymously (or pseudonymously) engage with a service or other users, these aspects need to be taken into account. This certainly has a sociological or game-theoretic aspect to it, designing systems where good behavior is encouraged and rewarded.

On the more technical side, oftentimes, the cryptographer’s suggested trade-off between anonymity and accountability is that a trusted party with a special key can revoke anonymity and enforce bans (see, for example, pseudonyms for traditional anonymous credentials, Section 4.1). While this is a good start, enabling designs where privacy is preserved except in special cases, not all system providers have the infrastructure to handle the anonymity revocation key responsibly while properly moderating their platform. A good avenue for future work is finding sociological mechanisms that encourage good behavior and implement tools for privacy-preserving systems based on them. In the past, the cryptography community has developed various reputation systems [BEJ18; BBBBD+18; LM19; GG21; BBP23], point collection (incentive) systems [JR16; HHNR17; BBDE19; HKRR20], advanced notions of privacy revocation [KLN23] and rate limiting [CHKLM06], systems with offline-translatable pseudonyms [CL15; CL17], and many more.

Adaptive corruption. Our anonymous credential security definitions are all written with static corruption in mind: Corrupted parties start out corrupted and honest parties stay honest (and their secrets are never leaked). This is not unusual for the anonymous credential literature. Stronger definitions would allow the adversary to adaptively corrupt honest parties during the security game.

We do not get security against adaptive corruptions for free. In particular, our constructions of UACS and DACS cannot be proven secure in the adaptive corruption setting. Consider adaptive corruption for our anonymity definitions (Definitions 5.2 and 6.3). When a credential gets corrupted, the simulator learns the (previously hidden) attributes for the credential and needs to produce a signature on those attributes. However, since

the simulator does not know the (adversarial) issuers’ secret keys, this is not possible: It would amount to the simulator forging a signature. This establishes that our current constructions are not meaningfully secure against adaptive corruption. While there may be solutions to this particular problem (e.g., having issuers prove knowledge of their secret keys), we expect a full treatment of updatable/delegatable anonymous credentials with security against adaptive corruption to involve additional nontrivial challenges. We leave such treatment to future work.

The threat of quantum computers. Our concrete constructions rely on the hardness of computing discrete logarithms. As a consequence, there are no security guarantees against quantum adversaries. Indeed, given a sufficiently powerful quantum computer, one can easily compute the credential issuer’s secret key from her public key using Shor’s algorithm. This allows the adversary to forge arbitrary credentials.

Today’s quantum computers are far from breaking discrete logarithms for relevant security parameters. However, it is prudent to start designing post-quantum anonymous credential systems so that when (or if) cryptographically relevant quantum computers become available, we are ready. This is important future work. Approaches for post-quantum anonymous credentials exist (e.g., [BLNS23; LLLW23; AGJLRS24; DKLW25; NS26]), but many advanced anonymous credential features are not yet available in the post-quantum setting. Thankfully, generic constructions (such as our UACS construction) can, in principle, be instantiated using post-quantum secure building blocks.

Much more pressing than full post-quantum security, however, is resistance against harvest-now-decrypt-later-style attacks. The threat is issuers or verifiers collecting data (e.g., protocol transcripts) now, in the hopes that future quantum computers will be able to break the privacy property (e.g., identify a user from the transcript). Our instantiations of UACS and DACS (Sections 5.3.5 and 6.4.4) are secure against such attacks. The proof system can be implemented such that proofs are statistically independent of the witness (Section 3.6) and the protocols for signing committed values have unconditional privacy (Theorems 5.11 and 6.14). This means that these systems can be used today, in practice, with no threat to privacy. In principle, we only need to switch to a (fully) post-quantum secure construction the moment quantum computers exist.

Notably, our construction of traditional anonymous credentials is *not* secure against harvest-now-decrypt later: Pseudonyms are ElGamal encryptions of the user identity upk (see Construction 4.5). Hence an adversary may collect pseudonyms from interactions today and a future quantum computer would be able to map pseudonyms to user identities. In general, this is a significant downside of deploying mechanisms where some trusted party holds a trapdoor key circumventing some privacy guarantee. Any such mechanisms should be post-quantum secure in order to prevent harvest-now-decrypt-later attacks.

On the need of adaptive straight-line extraction. Our constructions rely on adaptively straight-line extractable proof systems. This means that we expect the extractor to reply with the extracted witness without (rewinding) access to the algorithm that generated the proof, and that security is maintained against a prover who has access to

an extraction oracle. Adaptive straight-line extractable proofs are much more expensive than rewinding-based extractability (using Fischlin’s transform Construction 3.23 instead of simply Fiat-Shamir). Hence our reliance on them, while not unusual, is a drawback.

The need for adaptive straight-line extractability comes from two places: First, in our UACS and DACS security definitions (Definitions 5.3 and 6.5), we return the extracted explanation for an issuance, update, or presentation to the adversary. This is a largely arbitrary choice. It makes our systems easier to use in larger constructions because the larger construction reduction can make use of explanations as they come in. However, this is not a strictly necessary feature for many applications.

Second, and more importantly, our protocols for blindly signing committed values (Constructions 5.10 and 6.11) require adaptive straight-line extractability. This is because the security proof for these protocols is a reduction to the unforgeability of the signature scheme. This reduction generally needs to extract the hidden committed message so that it can query the signing oracle for the committed message (in our formalization, extraction of the message happens in the credential system proofs (Theorems 5.9 and 6.19), but the principle is ultimately the same). Note that while this is seldom mentioned explicitly, the same requirement holds for many blind signature schemes with an efficient protocol for signing committed values, in particular [CL04; BBS04; PS16].

Given that we can expect most practical implementations to heuristically implement the proof system using Fiat-Shamir rather than the more expensive Fischlin transform, an important task is building confidence in the security of anonymous credential systems instantiated with a Fiat-Shamir-based proof system. A candidate way to remove the requirement for adaptive straight-line extraction for the blind (signing a committed value) signature scheme is to use a structure-preserving signature and sign the commitment instead of the committed value. We leave details, as well as a more elaborate treatment of anonymous credentials without adaptive straight-line extractability, for future work.

Composability and extendability. Given the breadth of extensions around anonymous credentials and similar techniques, an interesting challenge is developing a composability framework that allows generically combining these extensions. Towards this goal, our definitions of updatable and delegatable credentials is a first step: They model the “core” anonymous credential system without any extensions. In Construction 5.14, we show how to extend this core with pseudonyms, and in [BBDE19], we extend it with a point collection system. Apart from our definition style, where extensions essentially contribute to the update functions and presentation predicates, other approaches exist, e.g., using a commitment to attributes as the common basis for several extensions to work with [LSZ25]. An open problem is designing composable security definitions, where extensions can modify the security definitions in some reasonable ways (e.g., pseudonyms add non-impersonation guarantees and diminish privacy guarantees).

Conclusion. It is an exciting time for research on anonymous credentials. There are realistic paths to widespread deployment of basic anonymous credential systems. On the cryptography side of this effort, there is still work to do: We need to push for technically

sound solutions, develop post-quantum secure schemes and gain confidence in them, understand the security of Fiat-Shamir-based deployment, and handle aspects such as adaptive corruption and tight security proofs. To help more researchers get started on the topic, this thesis contributes to a better understanding of the basics of anonymous credentials, including development of relevant zero-knowledge proof systems, through the lens of formal definitions and provable security.

Besides imminent practical widespread deployment around basic digital identity, we need to keep developing tools for building future privacy-preserving systems for a wider range of applications. In this thesis, we formalize and construct updatable anonymous credentials and delegatable anonymous credentials. These advanced functionalities are not yet targeted for broad mainstream deployment, but they are powerful tools for building privacy-preserving systems (such as point collection / loyalty systems). Privacy-preserving systems tend to differ in their specific individual requirements. The highly customizable nature of our formalism points towards a path to realistic deployment despite individual requirements: A suitable off-the-shelf implementation of updatable anonymous credentials can conceivably be easily customized (by designing the specific presentation predicates and update functions) to any particular use case. Perhaps this implementation even supports a library of standard gadgets for presentation predicates and update functions (e.g., a gadget that implements pseudonyms, or one that handles revocation) that can be easily combined. Our definitions make it easy for cryptographers to reason about the security of the resulting system.

References

- [Ada11] Carlisle Adams. “Achieving non-transferability in credential systems using hidden biometrics”. In: *Secur. Commun. Networks* 4.2 (2011), pp. 195–206. DOI: 10.1002/SEC.136 (cit. on p. 107).
- [AGJLRS24] Sven Argo, Tim Güneysu, Corentin Jeudy, Georg Land, Adeline Roux-Langlois, and Olivier Sanders. “Practical Post-Quantum Signatures for Privacy”. In: *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, CCS 2024, Salt Lake City, UT, USA, October 14-18, 2024*. ACM, 2024, pp. 1523–1537. DOI: 10.1145/3658644.3670297 (cit. on pp. 109, 234).
- [AJ13] Gergely Alpár and Bart Jacobs. “Towards Practical Attribute-Based Identity Management: The IRMA Trajectory”. In: *Policies and Research in Identity Management - Third IFIP WG 11.6 Working Conference, IDMAN 2013, London, UK, April 8-9, 2013. Proceedings*. IFIP Advances in Information and Communication Technology. Springer, 2013, pp. 1–3. DOI: 10.1007/978-3-642-37282-7_1 (cit. on p. 2).
- [BB08] Dan Boneh and Xavier Boyen. “Short Signatures Without Random Oracles and the SDH Assumption in Bilinear Groups”. In: *J. Cryptol.* 21.2 (2008), pp. 149–177. DOI: 10.1007/s00145-007-9005-7 (cit. on pp. 49, 57).
- [BB18] Johannes Blömer and Jan Bobolz. “Delegatable Attribute-Based Anonymous Credentials from Dynamically Malleable Signatures”. In: *Applied Cryptography and Network Security - 16th International Conference, ACNS 2018, Leuven, Belgium, July 2-4, 2018, Proceedings*. Vol. 10892. Lecture Notes in Computer Science. Springer, 2018, pp. 221–239. DOI: 10.1007/978-3-319-93387-0_12 (cit. on pp. 6, 168, 170, 173, 174, 187, 193).
- [BBBBD+18] Kai Bemmman, Johannes Blömer, Jan Bobolz, Henrik Bröcher, Denis Diemert, Fabian Eidens, Lukas Eilers, Jan Haltermann, Jakob Juhnke, Burhan Otour, Laurens Porzenheim, Simon Pukrop, Erik Schilling, Michael Schlichtig, and Marcel Stienemeier. “Fully-Featured Anonymous Credentials with Reputation System”. In: *Proceedings of the 13th International Conference on Availability, Reliability and Security, ARES 2018, Hamburg, Germany, August 27-30, 2018*. ACM, 2018, 42:1–42:10. DOI: 10.1145/3230833.3234517 (cit. on pp. 2, 233).
- [BBCHL+24] Carsten Baum, Olivier Blazy, Jan Camenisch, Jaap-Henk Hoepman, Eysa Lee, Anja Lehmann, Anna Lysyanskaya, René Mayrhofer, Hart Montgomery, Ngoc Khanh Nguyen, Bart Preneel, abhi shelat abhi, Daniel Slamanig, Stefano Tessaro, Søren Eller Thomsen, and Carmela Troncoso. *Cryptographers’ Feedback on the EU Digital Identity’s ARF*. June 2024. URL: <https://github.com/eu-digital-identity-wallet/eudi-doc-architecture-and-reference-framework/discussions/211> (cit. on p. 2).

References

- [BBDE19] Johannes Blömer, Jan Bobolz, Denis Diemert, and Fabian Eidens. “Updatable Anonymous Credentials and Applications to Incentive Systems”. In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*. ACM, 2019, pp. 1671–1685. DOI: 10.1145/3319535.3354223 (cit. on pp. 5, 114, 115, 120, 126, 233, 235).
- [BBDGQ91] Samy Bengio, Gilles Brassard, Yvo Desmedt, Claude Goutier, and Jean-Jacques Quisquater. “Secure Implementations of Identification Systems”. In: *J. Cryptol.* 4.3 (1991), pp. 175–183. DOI: 10.1007/BF00196726 (cit. on p. 62).
- [BBLMÖS25] Jan Bobolz, Emad Heydari Beni, Anja Lehmann, Omid Mirzamohammadi, Cavit Özbay, and Mahdi Sedaghat. “Multi-Verifier Keyed-Verification Anonymous Credentials”. In: *IACR Cryptol. ePrint Arch.* 2025 (2025), p. 2156. URL: <https://eprint.iacr.org/2025/2156> (cit. on p. 109).
- [BBP23] Johannes Blömer, Jan Bobolz, and Laurens Porzenheim. “A Generic Construction of an Anonymous Reputation System and Instantiations from Lattices”. In: *Advances in Cryptology - ASIACRYPT 2023 - 29th International Conference on the Theory and Application of Cryptology and Information Security, Guangzhou, China, December 4-8, 2023, Proceedings, Part II*. Lecture Notes in Computer Science. Springer, 2023, pp. 418–452. DOI: 10.1007/978-981-99-8724-5_13 (cit. on p. 233).
- [BBS04] Dan Boneh, Xavier Boyen, and Hovav Shacham. “Short Group Signatures”. In: *Advances in Cryptology - CRYPTO 2004, 24th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings*. Vol. 3152. Lecture Notes in Computer Science. Springer, 2004, pp. 41–55. DOI: 10.1007/978-3-540-28628-8_3 (cit. on pp. 108, 109, 128, 235).
- [BCKLS09] Mira Belenkiy, Jan Camenisch, Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Hovav Shacham. “Randomizable Proofs and Delegatable Anonymous Credentials”. In: *Advances in Cryptology - CRYPTO 2009, 29th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2009. Proceedings*. Lecture Notes in Computer Science. Springer, 2009, pp. 108–125. DOI: 10.1007/978-3-642-03356-8_7 (cit. on pp. 6, 169, 170).
- [BCHKLN15] Foteini Baldimtsi, Jan Camenisch, Lucjan Hanzlik, Stephan Krenn, Anja Lehmann, and Gregory Neven. “Recovering Lost Device-Bound Credentials”. In: *Applied Cryptography and Network Security - 13th International Conference, ACNS 2015, New York, NY, USA, June 2-5, 2015, Revised Selected Papers*. Lecture Notes in Computer Science. Springer, 2015, pp. 307–327. DOI: 10.1007/978-3-319-28166-7_15 (cit. on p. 106).
- [BCKL08] Mira Belenkiy, Melissa Chase, Markulf Kohlweiss, and Anna Lysyanskaya. “P-signatures and Noninteractive Anonymous Credentials”. In: *Theory of Cryptography, Fifth Theory of Cryptography Conference, TCC 2008, New York, USA, March 19-21, 2008*. Lecture Notes in Computer Science. Springer, 2008, pp. 356–374. DOI: 10.1007/978-3-540-78524-8_20 (cit. on pp. 108, 109, 128).

References

- [BEHF21] Jan Bobolz, Fabian Eidens, Raphael Heitjohann, and Jeremy Fell. *Crypticleon: A Library for Fast Prototyping of Privacy-Preserving Cryptographic Schemes*. Cryptology ePrint Archive, Paper 2021/961. <https://eprint.iacr.org/2021/961>. 2021. URL: <https://eprint.iacr.org/2021/961> (cit. on pp. 57, 58, 71, 80).
- [BEJ18] Johannes Blömer, Fabian Eidens, and Jakob Juhnke. “Practical, Anonymous, and Publicly Linkable Universally-Composable Reputation Systems”. In: *Topics in Cryptology - CT-RSA 2018 - The Cryptographers’ Track at the RSA Conference 2018, San Francisco, CA, USA, April 16-20, 2018, Proceedings*. Lecture Notes in Computer Science. Springer, 2018, pp. 470–490. DOI: 10.1007/978-3-319-76953-0_25 (cit. on p. 233).
- [BF03] Dan Boneh and Matthew K. Franklin. “Identity-Based Encryption from the Weil Pairing”. In: *SIAM J. Comput.* 32.3 (2003), pp. 586–615. DOI: 10.1137/S0097539701398521 (cit. on p. 27).
- [BF20] Balthazar Bauer and Georg Fuchsbauer. “Efficient Signatures on Randomizable Ciphertexts”. In: *Security and Cryptography for Networks - 12th International Conference, SCN 2020, Amalfi, Italy, September 14-16, 2020, Proceedings*. Lecture Notes in Computer Science. Springer, 2020, pp. 359–381. DOI: 10.1007/978-3-030-57990-6_18 (cit. on pp. 100, 109).
- [BFKT24] Jan Bobolz, Pooya Farshim, Markulf Kohlweiss, and Akira Takahashi. “The Brave New World of Global Generic Groups and UC-Secure Zero-Overhead SNARKs”. In: *Theory of Cryptography - 22nd International Conference, TCC 2024, Milan, Italy, December 2-6, 2024, Proceedings, Part I*. Lecture Notes in Computer Science. Springer, 2024, pp. 90–124. DOI: 10.1007/978-3-031-78011-0_4 (cit. on p. 30).
- [BFW16] David Bernhard, Marc Fischlin, and Bogdan Warinschi. “Adaptive proofs of knowledge in the random oracle model”. In: *IET Inf. Secur.* 10.6 (2016), pp. 319–331. DOI: 10.1049/iet-ifs.2015.0506 (cit. on p. 63).
- [BLNS23] Jonathan Bootle, Vadim Lyubashevsky, Ngoc Khanh Nguyen, and Alessandro Sorniotti. “A Framework for Practical Anonymous Credentials from Lattices”. In: *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part II*. Lecture Notes in Computer Science. Springer, 2023, pp. 384–417. DOI: 10.1007/978-3-031-38545-2_13 (cit. on pp. 109, 234).
- [BN06] Mihir Bellare and Gregory Neven. “Multi-signatures in the plain public-Key model and a general forking lemma”. In: *Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006, Alexandria, VA, USA, October 30 - November 3, 2006*. ACM, 2006, pp. 390–399. DOI: 10.1145/1180405.1180453 (cit. on pp. 41, 46, 76).
- [BNW17] David Bernhard, Ngoc Khanh Nguyen, and Bogdan Warinschi. “Adaptive Proofs Have Straightline Extractors (in the Random Oracle Model)”. In: *Applied Cryptography and Network Security - 15th International Conference, ACNS 2017, Kanazawa, Japan, July 10-12, 2017, Proceedings*. Vol. 10355. Lecture Notes in Computer Science. Springer, 2017, pp. 336–353. DOI: 10.1007/978-3-319-61204-1_17 (cit. on p. 64).

References

- [BR93] Mihir Bellare and Phillip Rogaway. “Random Oracles are Practical: A Paradigm for Designing Efficient Protocols”. In: *CCS ’93, Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, Virginia, USA, November 3-5, 1993*. ACM, 1993, pp. 62–73. DOI: 10.1145/168588.168596 (cit. on p. 26).
- [Bra96] Stefanus A. Brands. “Privacy-protected transfer of electronic information”. US5604805A. 1996. URL: <https://patents.google.com/patent/US5604805A/en> (cit. on p. 108).
- [CBCMRW24] Miranda Christ, Foteini Baldimtsi, Konstantinos Kryptos Chalkias, Deepak Maram, Arnab Roy, and Joy Wang. “SoK: Zero-Knowledge Range Proofs”. In: *6th Conference on Advances in Financial Technologies, AFT 2024, Vienna, Austria, September 23-25, 2024*. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024, 14:1–14:25. DOI: 10.4230/LIPICS.AFT.2024.14 (cit. on p. 57).
- [CCS08] Jan Camenisch, Rafik Chaabouni, and Abhi Shelat. “Efficient Protocols for Set Membership and Range Proofs”. In: *Advances in Cryptology - ASIACRYPT 2008, 14th International Conference on the Theory and Application of Cryptology and Information Security, Melbourne, Australia, December 7-11, 2008. Proceedings*. Vol. 5350. Lecture Notes in Computer Science. Springer, 2008, pp. 234–252. DOI: 10.1007/978-3-540-89255-7_15 (cit. on pp. 50, 57).
- [CDD17] Jan Camenisch, Manu Drijvers, and Maria Dubovitskaya. “Practical UC-Secure Delegatable Credentials with Attributes and Their Application to Blockchain”. In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*. ACM, 2017, pp. 683–699. DOI: 10.1145/3133956.3134025 (cit. on pp. 6, 170, 206).
- [CDLP22] Aisling Connolly, Jérôme Deschamps, Pascal Lafourcade, and Octavio Perez-Kempner. “Protego: Efficient, Revocable and Auditable Anonymous Credentials with Applications to Hyperledger Fabric”. In: *Progress in Cryptology - INDOCRYPT 2022 - 23rd International Conference on Cryptology in India, Kolkata, India, December 11-14, 2022, Proceedings*. Lecture Notes in Computer Science. Springer, 2022, pp. 249–271. DOI: 10.1007/978-3-031-22912-1_11 (cit. on p. 2).
- [CDS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. “Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols”. In: *Advances in Cryptology - CRYPTO ’94, 14th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1994, Proceedings*. Vol. 839. Lecture Notes in Computer Science. Springer, 1994, pp. 174–187. DOI: 10.1007/3-540-48658-5_19 (cit. on pp. 58–60).
- [CE86] David Chaum and Jan-Hendrik Evertse. “A Secure and Privacy-protecting Protocol for Transmitting Personal Information Between Organizations”. In: *Advances in Cryptology - CRYPTO ’86, Santa Barbara, California, USA, 1986, Proceedings*. Lecture Notes in Computer Science. Springer, 1986, pp. 118–167. DOI: 10.1007/3-540-47721-7_10 (cit. on pp. 82, 108).

References

- [CFN88] David Chaum, Amos Fiat, and Moni Naor. “Untraceable Electronic Cash”. In: *Advances in Cryptology - CRYPTO '88, 8th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1988, Proceedings*. Lecture Notes in Computer Science. Springer, 1988, pp. 319–327. DOI: 10.1007/0-387-34799-2_25 (cit. on p. 114).
- [CGH04] Ran Canetti, Oded Goldreich, and Shai Halevi. “The random oracle methodology, revisited”. In: *J. ACM* 51.4 (2004), pp. 557–594. DOI: 10.1145/1008731.1008734 (cit. on p. 28).
- [CGH11] Scott E. Coull, Matthew Green, and Susan Hohenberger. “Access controls for oblivious and anonymous systems”. In: *ACM Trans. Inf. Syst. Secur.* 14.1 (2011), 10:1–10:28. DOI: 10.1145/1952982.1952992 (cit. on p. 115).
- [CH02] Jan Camenisch and Els Van Herreweghen. “Design and implementation of the *idemix* anonymous credential system”. In: *Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS 2002, Washington, DC, USA, November 18-22, 2002*. ACM, 2002, pp. 21–30. DOI: 10.1145/586110.586114 (cit. on p. 2).
- [Cha84] David Chaum. “A New Paradigm for Individuals in the Information Age”. In: *Proceedings of the 1984 IEEE Symposium on Security and Privacy, Oakland, California, USA, April 29 - May 2, 1984*. IEEE Computer Society, 1984, pp. 99–106. DOI: 10.1109/SP.1984.10025 (cit. on pp. 1, 81, 85, 108, 231).
- [Cha85] David Chaum. “Security Without Identification: Transaction Systems to Make Big Brother Obsolete”. In: *Commun. ACM* 28.10 (1985), pp. 1030–1044. DOI: 10.1145/4372.4373 (cit. on pp. 1, 81, 85, 108, 231).
- [Che95] Lidong Chen. “Access with Pseudonyms”. In: *Cryptography: Policy and Algorithms, International Conference, Brisbane, Queensland, Australia, July 3-5, 1995, Proceedings*. Lecture Notes in Computer Science. Springer, 1995, pp. 232–243. DOI: 10.1007/BFB0032362 (cit. on p. 108).
- [CHKLM06] Jan Camenisch, Susan Hohenberger, Markulf Kohlweiss, Anna Lysyanskaya, and Mira Meyerovich. “How to win the clonewars: efficient periodic n-times anonymous authentication”. In: *Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006, Alexandria, VA, USA, October 30 - November 3, 2006*. ACM, 2006, pp. 201–210. DOI: 10.1145/1180405.1180431 (cit. on pp. 113, 233).
- [CHL05] Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. “Compact E-Cash”. In: *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*. Lecture Notes in Computer Science. Springer, 2005, pp. 302–321. DOI: 10.1007/11426639_18 (cit. on p. 115).
- [CHLT25] Rutchathon Chairattana-Apirom, Franklin Harding, Anna Lysyanskaya, and Stefano Tessaro. “Server-Aided Anonymous Credentials”. In: *Advances in Cryptology - CRYPTO 2025 - 45th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2025, Proceedings, Part VI*. Lecture Notes in Computer Science. Springer, 2025, pp. 291–324. DOI: 10.1007/978-3-032-01887-8_10 (cit. on p. 109).

References

- [CKLM13] Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Sarah Meiklejohn. “Succinct Malleable NIZKs and an Application to Compact Shuffles”. In: *Theory of Cryptography - 10th Theory of Cryptography Conference, TCC 2013, Tokyo, Japan, March 3-6, 2013. Proceedings*. Lecture Notes in Computer Science. Springer, 2013, pp. 100–119. DOI: 10.1007/978-3-642-36594-2_6 (cit. on p. 169).
- [CKLM14] Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Sarah Meiklejohn. “Malleable Signatures: New Definitions and Delegatable Anonymous Credentials”. In: *IEEE 27th Computer Security Foundations Symposium, CSF 2014, Vienna, Austria, 19-22 July, 2014*. IEEE Computer Society, 2014, pp. 199–213. DOI: 10.1109/CSF.2014.22 (cit. on pp. 169, 182).
- [CKLMNP15] Jan Camenisch, Stephan Krenn, Anja Lehmann, Gert Læssøe Mikkelsen, Gregory Neven, and Michael Østergaard Pedersen. “Formal Treatment of Privacy-Enhancing Credential Systems”. In: *Selected Areas in Cryptography - SAC 2015 - 22nd International Conference, Sackville, NB, Canada, August 12-14, 2015, Revised Selected Papers*. Lecture Notes in Computer Science. Springer, 2015, pp. 3–24. DOI: 10.1007/978-3-319-31301-6_1 (cit. on p. 115).
- [CKS09] Jan Camenisch, Markulf Kohlweiss, and Claudio Soriente. “An Accumulator Based on Bilinear Maps and Efficient Revocation for Anonymous Credentials”. In: *Public Key Cryptography - PKC 2009, 12th International Conference on Practice and Theory in Public Key Cryptography, Irvine, CA, USA, March 18-20, 2009. Proceedings*. Lecture Notes in Computer Science. Springer, 2009, pp. 481–500. DOI: 10.1007/978-3-642-00468-1_27 (cit. on p. 57).
- [CKS10] Jan Camenisch, Markulf Kohlweiss, and Claudio Soriente. “Solving Revocation with Efficient Update of Anonymous Credentials”. In: *Security and Cryptography for Networks, 7th International Conference, SCN 2010, Amalfi, Italy, September 13-15, 2010. Proceedings*. Lecture Notes in Computer Science. Springer, 2010, pp. 454–471. DOI: 10.1007/978-3-642-15317-4_28 (cit. on p. 115).
- [CL01] Jan Camenisch and Anna Lysyanskaya. “An Efficient System for Non-transferable Anonymous Credentials with Optional Anonymity Revocation”. In: *Advances in Cryptology - EUROCRYPT 2001, International Conference on the Theory and Application of Cryptographic Techniques, Innsbruck, Austria, May 6-10, 2001, Proceeding*. Lecture Notes in Computer Science. Springer, 2001, pp. 93–118. DOI: 10.1007/3-540-44987-6_7 (cit. on pp. 5, 82, 103, 105, 108, 114).
- [CL02] Jan Camenisch and Anna Lysyanskaya. “A Signature Scheme with Efficient Protocols”. In: *Security in Communication Networks, Third International Conference, SCN 2002, Amalfi, Italy, September 11-13, 2002. Revised Papers*. Lecture Notes in Computer Science. Springer, 2002, pp. 268–289. DOI: 10.1007/3-540-36413-7_20 (cit. on pp. 2, 5, 98, 108, 109, 128, 130).
- [CL04] Jan Camenisch and Anna Lysyanskaya. “Signature Schemes and Anonymous Credentials from Bilinear Maps”. In: *Advances in Cryptology - CRYPTO 2004, 24th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings*. Vol. 3152. Lecture Notes in Computer Science. Springer, 2004, pp. 56–72. DOI: 10.1007/978-3-540-28628-8_4 (cit. on pp. 5, 108, 109, 128, 144, 235).

References

- [CL06] Melissa Chase and Anna Lysyanskaya. “On Signatures of Knowledge”. In: *Advances in Cryptology - CRYPTO 2006, 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2006, Proceedings*. Lecture Notes in Computer Science. Springer, 2006, pp. 78–96. DOI: 10.1007/11818175_5 (cit. on pp. 6, 169, 170).
- [CL13] Sébastien Canard and Roch Lescuyer. “Protecting privacy by sanitizing personal data: a new approach to anonymous credentials”. In: *8th ACM Symposium on Information, Computer and Communications Security, ASIA CCS ’13, Hangzhou, China - May 08 - 10, 2013*. ACM, 2013, pp. 381–392. DOI: 10.1145/2484313.2484363 (cit. on p. 109).
- [CL15] Jan Camenisch and Anja Lehmann. “(Un)linkable Pseudonyms for Governmental Databases”. In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*. ACM, 2015, pp. 1467–1479. DOI: 10.1145/2810103.2813658 (cit. on p. 233).
- [CL17] Jan Camenisch and Anja Lehmann. “Privacy-Preserving User-Auditable Pseudonym Systems”. In: *2017 IEEE European Symposium on Security and Privacy, EuroS&P 2017, Paris, France, April 26-28, 2017*. IEEE, 2017, pp. 269–284. DOI: 10.1109/EUROSP.2017.36 (cit. on p. 233).
- [CL19] Elizabeth C. Crites and Anna Lysyanskaya. “Delegatable Anonymous Credentials from Mercurial Signatures”. In: *Topics in Cryptology - CT-RSA 2019 - The Cryptographers’ Track at the RSA Conference 2019, San Francisco, CA, USA, March 4-8, 2019, Proceedings*. Lecture Notes in Computer Science. Springer, 2019, pp. 535–555. DOI: 10.1007/978-3-030-12612-4_27 (cit. on pp. 6, 109, 170, 171).
- [CL21] Elizabeth C. Crites and Anna Lysyanskaya. “Mercurial Signatures for Variable-Length Messages”. In: *Proc. Priv. Enhancing Technol.* 2021.4 (2021), pp. 441–463. DOI: 10.2478/POPETS-2021-0079 (cit. on p. 170).
- [CL24] Yi-Hsiu Chen and Yehuda Lindell. “Optimizing and Implementing Fischlin’s Transform for UC-Secure Zero Knowledge”. In: *IACR Commun. Cryptol.* 1.2 (2024), p. 11. DOI: 10.62056/A66CHEY6B (cit. on p. 71).
- [CMZ14] Melissa Chase, Sarah Meiklejohn, and Greg Zaverucha. “Algebraic MACs and Keyed-Verification Anonymous Credentials”. In: *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*. ACM, 2014, pp. 1205–1216. DOI: 10.1145/2660267.2660328 (cit. on p. 109).
- [CPZ20] Melissa Chase, Trevor Perrin, and Greg Zaverucha. “The Signal Private Group System and Anonymous Credentials Supporting Efficient Verifiable Encryption”. In: *CCS ’20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*. ACM, 2020, pp. 1445–1459. DOI: 10.1145/3372297.3417887 (cit. on pp. 109, 231).
- [Cra97] Ronald Cramer. “Modular Design of Secure yet Practical Cryptographic Protocols”. Research conducted at Universiteit van Amsterdam and CWI. PhD thesis. Amsterdam, The Netherlands: University of Amsterdam, 1997. ISBN: 978-90-74795-64-7 (cit. on p. 43).

References

- [CS97] Jan Camenisch and Markus Stadler. “Efficient Group Signature Schemes for Large Groups (Extended Abstract)”. In: *Advances in Cryptology - CRYPTO ’97, 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 1997, Proceedings*. Vol. 1294. Lecture Notes in Computer Science. Springer, 1997, pp. 410–424. DOI: 10.1007/BFb0052252 (cit. on pp. 49, 79).
- [Dam00] Ivan Damgård. “Efficient Concurrent Zero-Knowledge in the Auxiliary String Model”. In: *Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding*. Vol. 1807. Lecture Notes in Computer Science. Springer, 2000, pp. 418–430. DOI: 10.1007/3-540-45539-6_30 (cit. on pp. 63, 75, 79, 120).
- [Dam10] Ivan Damgård. “On Sigma Protocols”. In: *Aarhus University* (2010). URL: <https://cs.au.dk/~ivan/Sigma.pdf> (cit. on p. 62).
- [Dam88] Ivan Damgård. “Payment Systems and Credential Mechanisms with Provable Security Against Abuse by Individuals”. In: *Advances in Cryptology - CRYPTO ’88, 8th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1988, Proceedings*. Lecture Notes in Computer Science. Springer, 1988, pp. 328–335. DOI: 10.1007/0-387-34799-2_26 (cit. on p. 108).
- [DD05] Liesje Demuynck and Bart De Decker. *Anonymous Updating of Credentials*. Tech. rep. CW 430. Katholieke Universiteit Leuven, Dec. 2005. URL: <https://lirias.kuleuven.be/1653480> (cit. on p. 115).
- [Den02] Alexander W. Dent. “Adapting the Weaknesses of the Random Oracle Model to the Generic Group Model”. In: *Advances in Cryptology - ASIACRYPT 2002, 8th International Conference on the Theory and Application of Cryptology and Information Security, Queenstown, New Zealand, December 1-5, 2002, Proceedings*. Vol. 2501. Lecture Notes in Computer Science. Springer, 2002, pp. 100–109. DOI: 10.1007/3-540-36178-2_6 (cit. on p. 39).
- [DGSTV18] Alex Davidson, Ian Goldberg, Nick Sullivan, George Tankersley, and Filippo Valsorda. “Privacy Pass: Bypassing Internet Challenges Anonymously”. In: *Proc. Priv. Enhancing Technol.* 2018.3 (2018), pp. 164–180. DOI: 10.1515/POPETS-2018-0026 (cit. on p. 231).
- [DKLW25] Adrien Dubois, Michael Klooß, Russell W. F. Lai, and Ivy K. Y. Woo. “Lattice-Based Proof-Friendly Signatures from Vanishing Short Integer Solutions”. In: *Public-Key Cryptography - PKC 2025 - 28th IACR International Conference on Practice and Theory of Public-Key Cryptography, Røros, Norway, May 12-15, 2025, Proceedings, Part I*. Lecture Notes in Computer Science. Springer, 2025, pp. 452–486. DOI: 10.1007/978-3-031-91820-9_15 (cit. on p. 234).
- [EK18] Ali El Kaafarani and Shuichi Katsumata. “Attribute-Based Signatures for Unbounded Circuits in the ROM and Efficient Instantiations from Lattices”. In: *Public-Key Cryptography - PKC 2018 - 21st IACR International Conference on Practice and Theory of Public-Key Cryptography, Rio de Janeiro, Brazil, March 25-29, 2018, Proceedings, Part II*. Vol. 10770. Lecture Notes in Computer Science. Springer, 2018, pp. 89–119. DOI: 10.1007/978-3-319-76581-5_4 (cit. on p. 41).

References

- [FHLL26] Karla Friedrichs, Franklin Harding, Anja Lehmann, and Anna Lysyanskaya. “Device-Bound Anonymous Credentials With(out) Trusted Hardware”. In: *Advances in Cryptology - EUROCRYPT 2026 - 45th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Rome, Italy, May 10-14, 2026, Proceedings, Part II*. Lecture Notes in Computer Science. Springer, 2026, pp. 345–375. DOI: 10.1007/978-3-032-25317-0_12 (cit. on p. 106).
- [FHS19] Georg Fuchsbauer, Christian Hanser, and Daniel Slamanig. “Structure-Preserving Signatures on Equivalence Classes and Constant-Size Anonymous Credentials”. In: *J. Cryptol.* 32.2 (2019), pp. 498–546. DOI: 10.1007/s00145-018-9281-4 (cit. on pp. 100, 109, 170, 171, 193).
- [Fis00] Marc Fischlin. “A Note on Security Proofs in the Generic Model”. In: *Advances in Cryptology - ASIACRYPT 2000, 6th International Conference on the Theory and Application of Cryptology and Information Security, Kyoto, Japan, December 3-7, 2000, Proceedings*. Vol. 1976. Lecture Notes in Computer Science. Springer, 2000, pp. 458–469. DOI: 10.1007/3-540-44448-3_35 (cit. on p. 39).
- [Fis05] Marc Fischlin. “Communication-Efficient Non-interactive Proofs of Knowledge with Online Extractors”. In: *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*. Vol. 3621. Lecture Notes in Computer Science. Springer, 2005, pp. 152–168. DOI: 10.1007/11535218_10 (cit. on pp. 42, 63, 69–71).
- [FKL18] Georg Fuchsbauer, Eike Kiltz, and Julian Loss. “The Algebraic Group Model and its Applications”. In: *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part II*. Vol. 10992. Lecture Notes in Computer Science. Springer, 2018, pp. 33–62. DOI: 10.1007/978-3-319-96881-0_2 (cit. on p. 39).
- [FS86] Amos Fiat and Adi Shamir. “How to Prove Yourself: Practical Solutions to Identification and Signature Problems”. In: *Advances in Cryptology - CRYPTO ’86, Santa Barbara, California, USA, 1986, Proceedings*. Vol. 263. Lecture Notes in Computer Science. Springer, 1986, pp. 186–194. DOI: 10.1007/3-540-47721-7_12 (cit. on pp. 41, 42, 75, 79).
- [FS90] Uriel Feige and Adi Shamir. “Witness Indistinguishable and Witness Hiding Protocols”. In: *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*. ACM, 1990, pp. 416–426. DOI: 10.1145/100216.100272 (cit. on p. 63).
- [Fuc11] Georg Fuchsbauer. “Commuting Signatures and Verifiable Encryption”. In: *Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings*. Lecture Notes in Computer Science. Springer, 2011, pp. 224–245. DOI: 10.1007/978-3-642-20465-4_14 (cit. on p. 169).
- [GG21] Stan Gurtler and Ian Goldberg. “SoK: Privacy-Preserving Reputation Systems”. In: *Proc. Priv. Enhancing Technol.* 2021.1 (2021), pp. 107–127. DOI: 10.2478/POPETS-2021-0007 (cit. on p. 233).

References

- [GGM14] Christina Garman, Matthew Green, and Ian Miers. “Decentralized Anonymous Credentials”. In: *21st Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego, California, USA, February 23-26, 2014*. The Internet Society, 2014. URL: <https://www.ndss-symposium.org/ndss2014/decentralized-anonymous-credentials> (cit. on p. 109).
- [GLMPS24] Scott Griffy, Anna Lysyanskaya, Omid Mir, Octavio Perez-Kempner, and Daniel Slamanig. “Delegatable Anonymous Credentials from Mercurial Signatures with Stronger Privacy”. In: *Advances in Cryptology - ASIACRYPT 2024 - 30th International Conference on the Theory and Application of Cryptology and Information Security, Kolkata, India, December 9-13, 2024, Proceedings, Part II*. Lecture Notes in Computer Science. Springer, 2024, pp. 296–325. DOI: 10.1007/978-981-96-0888-1_10 (cit. on p. 170).
- [GMR19] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. “The knowledge complexity of interactive proof-systems”. In: *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*. ACM, 2019, pp. 203–225. DOI: 10.1145/3335741.3335750 (cit. on p. 40).
- [Gol01] Oded Goldreich. *The Foundations of Cryptography - Volume 1: Basic Techniques*. Cambridge University Press, 2001. ISBN: 0-521-79172-3. DOI: 10.1017/CB09780511546891 (cit. on pp. 63, 64).
- [Gro15] Jens Groth. “Efficient Fully Structure-Preserving Signatures for Large Messages”. In: *Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part I*. Lecture Notes in Computer Science. Springer, 2015, pp. 239–259. DOI: 10.1007/978-3-662-48797-6_11 (cit. on p. 109).
- [GS08] Jens Groth and Amit Sahai. “Efficient Non-interactive Proof Systems for Bilinear Groups”. In: *Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings*. Lecture Notes in Computer Science. Springer, 2008, pp. 415–432. DOI: 10.1007/978-3-540-78967-3_24 (cit. on p. 108).
- [HHNR17] Gunnar Hartung, Max Hoffmann, Matthias Nagel, and Andy Rupp. “BBA+: Improving the Security and Applicability of Privacy-Preserving Point Collection”. In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*. ACM, 2017, pp. 1925–1942. DOI: 10.1145/3133956.3134071 (cit. on p. 233).
- [HKL19] Eduard Hauck, Eike Kiltz, and Julian Loss. “A Modular Treatment of Blind Signatures from Identification Schemes”. In: *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part III*. Vol. 11478. Lecture Notes in Computer Science. Springer, 2019, pp. 345–375. DOI: 10.1007/978-3-030-17659-4_12 (cit. on p. 41).

References

- [HKRR20] Max Hoffmann, Michael Klooß, Markus Raiber, and Andy Rupp. “Black-Box Wallets: Fast Anonymous Two-Way Payments for Constrained Devices”. In: *Proc. Priv. Enhancing Technol.* 2020.1 (2020), pp. 165–194. DOI: 10.2478/POPETS-2020-0010 (cit. on p. 233).
- [HS21] Lucjan Hanzlik and Daniel Slamanig. “With a Little Help from My Friends: Constructing Practical Anonymous Credentials”. In: *CCS ’21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*. ACM, 2021, pp. 2004–2023. DOI: 10.1145/3460120.3484582 (cit. on p. 106).
- [HSS23] Julia Hesse, Nitin Singh, and Alessandro Sorniotti. “How to Bind Anonymous Credentials to Humans”. In: *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023*. USENIX Association, 2023, pp. 3047–3064. URL: <https://www.usenix.org/conference/usenixsecurity23/presentation/hesse> (cit. on p. 107).
- [JR10] Tibor Jager and Andy Rupp. “The Semi-Generic Group Model and Applications to Pairing-Based Cryptography”. In: *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings*. Vol. 6477. Lecture Notes in Computer Science. Springer, 2010, pp. 539–556. DOI: 10.1007/978-3-642-17373-8_31 (cit. on p. 39).
- [JR16] Tibor Jager and Andy Rupp. “Black-Box Accumulation: Collecting Incentives in a Privacy-Preserving Way”. In: *Proc. Priv. Enhancing Technol.* 2016.3 (2016), pp. 62–82. DOI: 10.1515/POPETS-2016-0016 (cit. on p. 233).
- [KLN23] Markulf Kohlweiss, Anna Lysyanskaya, and An Nguyen. “Privacy-Preserving Blueprints”. In: *Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part II*. Lecture Notes in Computer Science. Springer, 2023, pp. 594–625. DOI: 10.1007/978-3-031-30617-4_20 (cit. on p. 233).
- [KRS25] Dmitry Khovratovich, Ron D. Rothblum, and Lev Soukhanov. “How to Prove False Statements: Practical Attacks on Fiat-Shamir”. In: *Advances in Cryptology - CRYPTO 2025 - 45th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2025, Proceedings, Part VI*. Lecture Notes in Computer Science. Springer, 2025, pp. 3–26. DOI: 10.1007/978-3-032-01887-8_1 (cit. on p. 28).
- [KS22] Yashvanth Kondi and Abhi Shelat. “Improved Straight-Line Extraction in the Random Oracle Model with Applications to Signature Aggregation”. In: *Advances in Cryptology - ASIACRYPT 2022 - 28th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, December 5-9, 2022, Proceedings, Part II*. Lecture Notes in Computer Science. Springer, 2022, pp. 279–309. DOI: 10.1007/978-3-031-22966-4_10 (cit. on pp. 42, 47, 69–71).
- [Law24] European Union Lawmakers. *Regulation (EU) 2024/1183 of the European Parliament and of the Council of 11 April 2024 amending Regulation (EU) No 910/2014 as regards establishing the European Digital Identity Framework*. Apr. 11, 2024. URL: <https://data.europa.eu/eli/reg/2024/1183/oj> (visited on 05/28/2026) (cit. on p. 2).

References

- [LLLW23] Qiqi Lai, Feng-Hao Liu, Anna Lysyanskaya, and Zhedong Wang. “Lattice-based Commit-Transferrable Signatures and Applications to Anonymous Credentials”. In: *IACR Cryptol. ePrint Arch.* 2023 (2023), p. 766. URL: <https://eprint.iacr.org/2023/766> (cit. on p. 234).
- [LM19] Jia Liu and Mark Manulis. “pRate: Anonymous Star Rating with Rating Secrecy”. In: *Applied Cryptography and Network Security - 17th International Conference, ACNS 2019, Bogota, Colombia, June 5-7, 2019, Proceedings. Lecture Notes in Computer Science*. Springer, 2019, pp. 550–570. DOI: 10.1007/978-3-030-21568-2_27 (cit. on p. 233).
- [LPS23] Helger Lipmaa, Roberto Parisella, and Janno Siim. “Algebraic Group Model with Oblivious Sampling”. In: *Theory of Cryptography - 21st International Conference, TCC 2023, Taipei, Taiwan, November 29 - December 2, 2023, Proceedings, Part IV*. Vol. 14372. Lecture Notes in Computer Science. Springer, 2023, pp. 363–392. DOI: 10.1007/978-3-031-48624-1_14 (cit. on pp. 30, 39).
- [LRSW99] Anna Lysyanskaya, Ronald L. Rivest, Amit Sahai, and Stefan Wolf. “Pseudonym Systems”. In: *Selected Areas in Cryptography, 6th Annual International Workshop, SAC’99, Kingston, Ontario, Canada, August 9-10, 1999, Proceedings. Lecture Notes in Computer Science*. Springer, 1999, pp. 184–199. DOI: 10.1007/3-540-46513-8_14 (cit. on pp. 84, 87, 103, 104, 108).
- [LSZ25] Anja Lehmann, Andrey Sidorenko, and Alexandros Zacharakis. “Vision: A Modular Framework for Anonymous Credential Systems”. In: *IACR Cryptol. ePrint Arch.* 2025 (2025), p. 1981. URL: <https://eprint.iacr.org/2025/1981> (cit. on p. 235).
- [Lys02] Anna Lysyanskaya. “Signature schemes and applications to cryptographic protocol design”. PhD thesis. Massachusetts Institute of Technology, Cambridge, MA, USA, 2002. URL: <https://hdl.handle.net/1721.1/29271> (cit. on p. 104).
- [Mau05] Ueli M. Maurer. “Abstract Models of Computation in Cryptography”. In: *Cryptography and Coding, 10th IMA International Conference, Cirencester, UK, December 19-21, 2005, Proceedings*. Vol. 3796. Lecture Notes in Computer Science. Springer, 2005, pp. 1–12. DOI: 10.1007/11586821_1 (cit. on pp. 29, 39).
- [Mau09] Ueli M. Maurer. “Unifying Zero-Knowledge Proofs of Knowledge”. In: *Progress in Cryptology - AFRICACRYPT 2009, Second International Conference on Cryptology in Africa, Gammarth, Tunisia, June 21-25, 2009. Proceedings*. Vol. 5580. Lecture Notes in Computer Science. Springer, 2009, pp. 272–286. DOI: 10.1007/978-3-642-02384-2_17 (cit. on pp. 48, 56).
- [Mau15] Ueli Maurer. “Zero-knowledge proofs of knowledge for group homomorphisms”. In: *Des. Codes Cryptogr.* 77.2-3 (2015), pp. 663–676. DOI: 10.1007/S10623-015-0103-5 (cit. on p. 48).
- [MBSBA+25] Omid Mirzamohammadi, Jan Bobolz, Mahdi Sedaghat, Emad Heydari Beni, Aysajan Abidin, Dave Singelée, and Bart Preneel. “Keyed-Verification Anonymous Credentials with Highly Efficient Partial Disclosure”. In: *IACR Commun. Cryptol.* 2.3 (2025), p. 15. DOI: 10.62056/AEY7N59P1 (cit. on p. 109).

References

- [MSBM23] Omid Mir, Daniel Slamanig, Balthazar Bauer, and René Mayrhofer. “Practical Delegatable Anonymous Credentials From Equivalence Class Signatures”. In: *Proc. Priv. Enhancing Technol.* 2023.3 (2023), pp. 488–513. DOI: 10.56553/POPETS-2023-0093 (cit. on pp. 171, 225).
- [NDD06] Vincent Naessens, Liesje Demuyne, and Bart De Decker. “A Fair Anonymous Submission and Review System”. In: *Communications and Multimedia Security, 10th IFIP TC-6 TC-11 International Conference, CMS 2006, Heraklion, Crete, Greece, October 19-21, 2006, Proceedings*. Lecture Notes in Computer Science. Springer, 2006, pp. 43–53. DOI: 10.1007/11909033_5 (cit. on p. 115).
- [Nec94] Vassiliy Ilyich Nechaev. “Complexity of a determinate algorithm for the discrete logarithm”. In: *Mathematical Notes* 55.2 (1994), pp. 165–172 (cit. on p. 29).
- [Ngu05] Lan Nguyen. “Accumulators from Bilinear Pairings and Applications”. In: *Topics in Cryptology - CT-RSA 2005, The Cryptographers’ Track at the RSA Conference 2005, San Francisco, CA, USA, February 14-18, 2005, Proceedings*. Lecture Notes in Computer Science. Springer, 2005, pp. 275–292. DOI: 10.1007/978-3-540-30574-3_19 (cit. on p. 57).
- [NS26] Ngoc Khanh Nguyen and Jan Niklas Siemer. “Tight Reductions for SIS-with-Hints Assumptions with Applications to Anonymous Credentials”. In: *Public-Key Cryptography - PKC 2026 - 29th IACR International Conference on Practice and Theory of Public-Key Cryptography, West Palm Beach, FL, USA, May 25-28, 2026, Proceedings, Part I*. Lecture Notes in Computer Science. Springer, 2026, pp. 103–135. DOI: 10.1007/978-3-032-26731-3_4 (cit. on p. 234).
- [Pas03] Rafael Pass. “On Deniability in the Common Reference String and Random Oracle Model”. In: *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*. Vol. 2729. Lecture Notes in Computer Science. Springer, 2003, pp. 316–337. DOI: 10.1007/978-3-540-45146-4_19 (cit. on p. 69).
- [PM23] Colin Putman and Keith M. Martin. “Selective Delegation of Attributes in Mercurial Signature Credentials”. In: *Cryptography and Coding - 19th IMA International Conference, IMACC 2023, London, UK, December 12-14, 2023, Proceedings*. Lecture Notes in Computer Science. Springer, 2023, pp. 181–196. DOI: 10.1007/978-3-031-47818-5_10 (cit. on pp. 170, 171).
- [PS16] David Pointcheval and Olivier Sanders. “Short Randomizable Signatures”. In: *Topics in Cryptology - CT-RSA 2016 - The Cryptographers’ Track at the RSA Conference 2016, San Francisco, CA, USA, February 29 - March 4, 2016, Proceedings*. Vol. 9610. Lecture Notes in Computer Science. Springer, 2016, pp. 111–126. DOI: 10.1007/978-3-319-29485-8_7 (cit. on pp. 109, 128, 144, 145, 147–149, 171, 193, 235).
- [PS18] David Pointcheval and Olivier Sanders. “Reassessing Security of Randomizable Signatures”. In: *Topics in Cryptology - CT-RSA 2018 - The Cryptographers’ Track at the RSA Conference 2018, San Francisco, CA, USA, April 16-20, 2018, Proceedings*. Vol. 10808. Lecture Notes in Computer Science. Springer, 2018, pp. 319–338. DOI: 10.1007/978-3-319-76953-0_17 (cit. on p. 148).

References

- [PS96] David Pointcheval and Jacques Stern. “Security Proofs for Signature Schemes”. In: *Advances in Cryptology - EUROCRYPT '96, International Conference on the Theory and Application of Cryptographic Techniques, Saragossa, Spain, May 12-16, 1996, Proceeding*. Vol. 1070. Lecture Notes in Computer Science. Springer, 1996, pp. 387–398. DOI: 10.1007/3-540-68339-9_33 (cit. on pp. 41, 46, 76).
- [PZ23] Christian Paquin and Greg Zaverucha. *U-Prove Cryptographic Specification V1.1: Revision 5*. Tech. rep. Microsoft Corporation, 2023 (cit. on p. 2).
- [RKS24] Jesús García Rodríguez, Stephan Krenn, and Daniel Slamanig. “To pass or not to pass: Privacy-preserving physical access control”. In: *Comput. Secur.* 136 (2024), p. 103566. DOI: 10.1016/J.COSE.2023.103566 (cit. on p. 107).
- [RWGM23] Michael Rosenberg, Jacob D. White, Christina Garman, and Ian Miers. “zk-creds: Flexible Anonymous Credentials from zkSNARKs and Existing Identity Infrastructure”. In: *44th IEEE Symposium on Security and Privacy, SP 2023, San Francisco, CA, USA, May 21-25, 2023*. IEEE, 2023, pp. 790–808. DOI: 10.1109/SP46215.2023.10179430 (cit. on p. 109).
- [Sch89] Claus-Peter Schnorr. “Efficient Identification and Signatures for Smart Cards”. In: *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*. Vol. 435. Lecture Notes in Computer Science. Springer, 1989, pp. 239–252. DOI: 10.1007/0-387-34805-0_22 (cit. on pp. 42, 48, 49).
- [SG02] Victor Shoup and Rosario Gennaro. “Securing Threshold Cryptosystems against Chosen Ciphertext Attack”. In: *J. Cryptol.* 15.2 (2002), pp. 75–96. DOI: 10.1007/s00145-001-0020-9 (cit. on p. 76).
- [Sho04] Victor Shoup. “Sequences of games: a tool for taming complexity in security proofs”. In: *IACR Cryptol. ePrint Arch.* 2004 (2004), p. 332. URL: <http://eprint.iacr.org/2004/332> (cit. on pp. 37, 73).
- [Sho97] Victor Shoup. “Lower Bounds for Discrete Logarithms and Related Problems”. In: *Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997, Proceeding*. Vol. 1233. Lecture Notes in Computer Science. Springer, 1997, pp. 256–266. DOI: 10.1007/3-540-69053-0_18 (cit. on pp. 29, 39).
- [SRKM25] Maurice Shih, Michael Rosenberg, Hari Kailad, and Ian Miers. “zk-promises: Anonymous Moderation, Reputation, and Blocking from Anonymous Credentials with Callbacks”. In: *34th USENIX Security Symposium, USENIX Security 2025, Seattle, WA, USA, August 13-15, 2025*. USENIX Association, 2025, pp. 4995–5014. URL: <https://www.usenix.org/conference/usenixsecurity25/presentation/shih> (cit. on p. 115).
- [Ver01] Eric R. Verheul. “Self-Blindable Credential Certificates from the Weil Pairing”. In: *Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9-13, 2001, Proceedings*. Lecture Notes in Computer Science. Springer, 2001, pp. 533–551. DOI: 10.1007/3-540-45682-1_31 (cit. on p. 108).

References

- [Wik23] Contributors Wikipedia. *Schwartz–Zippel lemma* — *Wikipedia, The Free Encyclopedia*. [Online; accessed 23-August-2023]. 2023. URL: https://en.wikipedia.org/w/index.php?title=Schwartz%E2%80%93Zippel_lemma&oldid=1145153444 (cit. on p. 36).
- [YXZHLD24] Yang Yang, Wenyi Xue, Yonghua Zhan, Minming Huang, Yingjiu Li, and Robert H. Deng. “AnoPay: Anonymous Payment for Vehicle Parking With Updatable Credential”. In: *IEEE Trans. Dependable Secur. Comput.* 21.4 (2024), pp. 1621–1638. DOI: 10.1109/TDSC.2023.3287228 (cit. on p. 114).
- [Zha22] Mark Zhandry. “To Label, or Not To Label (in Generic Groups)”. In: *Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15-18, 2022, Proceedings, Part III*. Vol. 13509. Lecture Notes in Computer Science. Springer, 2022, pp. 66–96. DOI: 10.1007/978-3-031-15982-4_3 (cit. on p. 39).