Report on Research on Back-Up and Recovery Mechanisms

Document Identification	
Date	20/10/2014
Status	Final
Version	1.0

Related SP / WP	SP3 / WP35	**Document Reference**	D35.4
Related Deliverable(s)	D35.1	**Dissemination Level**	PU
Lead Participant	IBM	**Lead Author**	Stephan Krenn
Contributors	IBM	**Reviewers**	G&D, TUG

Abstract: In this document, we present efficient constructions that allow a user to re-obtain a credential from an issuer after losing access to some of the attributes embedded in the first credential. For instance, this may be the case if the credential is bound to a specific device which breaks or gets lost. Our constructions can be added to most existing credential schemes without requiring major revisions of the underlying system.

This document is issued within the frame and for the purpose of the *FutureID* project. This project has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under Grant Agreement no. 318424.

1 Executive Summary

In many real-world scenarios, authentication credentials are bound to specific devices. This may be to increase security by requiring physical access to, e.g., a smart card to authenticate towards a service, or for economic reasons by, e.g., requiring that a user possesses a device of a certain manufacturer in order to access an online service. In both settings, the credential can no longer be used by the user in case of loss or disfunction of the device. In particular in the former case one further has to assume that an adversary gains access to the device and tries to impersonate the user. Revoking the lost credentials and re-obtaining new ones may be a tedious process in practice, as it may, e.g., involve physical appearance at some public authority.

In this document, we provide multiple constructions that circumvent this problem. By letting the legitimate owner of a credential compute some backup data and store some secret cryptographic keys, it can be guaranteed that only the legitimate owner may re-obtain a credential from an issuer, in a fully online protocol.

The given construction are generic, in the sense that they can be integrated into most currently existing anonymous credential schemes. The first construction protects a user against disfunction of the device, the second one against theft, and the third one further restricts the harm an adversary can do if the user does not immediately realize that his credentials were stolen. Each construction is computationally more expensive than the previous one, but all three constructions can be realized at practically acceptable costs.

2 Document Information

2.1 Contributors

Name	Partner
Jan Camenisch	IBM
Lucjan Hanzlik	IBM
Stephan Krenn	IBM
Anja Lehmann	IBM
Gregory Neven	IBM

2.2 History

Version	Date	Author	Changes
0.1	Oct. 05, 2014	Stephan Krenn	Preliminary first version
0.2	Oct. 08, 2014	Stephan Krenn	Added intro and preliminaries
0.3	Oct. 09, 2014	Stephan Krenn	Added abstract, conclusion, and explanatory texts
0.3.1	Oct. 09, 2014	Lucjan Hanzlik	Fixed typos and minor bugs
0.4	Oct. 09, 2014	Stephan Krenn	Added security against malicious issuers
0.4.1	Oct. 13, 2014	Stephan Krenn	Incorporated reviewer's comments from G&D
0.4.2	Oct. 15, 2014	Stephan Krenn	Incorporated reviewer's comments from TUG
1.0	Oct. 20, 2014	Stephan Krenn	Added parameter-overview and final editing

Table of Contents

4 Project Description

The *FutureID* project builds a comprehensive, flexible, privacy-aware and ubiquitously usable identity management infrastructure for Europe, which integrates existing eID technology and trust infrastructures, emerging federated identity management services and modern credential technologies to provide a user-centric system for the trustworthy and accountable management of identity claims.

The *FutureID* infrastructure will provide great benefits to all stakeholders involved in the eID value chain. Users will benefit from the availability of a ubiquitously usable open source eID client that is capable of running on arbitrary desktop PCs, tablets and modern smart phones. *FutureID* will allow application and service providers to easily integrate their existing services with the *FutureID* infrastructure, providing them with the benefits from the strong security offered by eIDs without requiring them to make substantial investments.

This will enable service providers to offer this technology to users as an alternative to username/password based systems, providing them with a choice for a more trustworthy, usable and innovative technology. For existing and emerging trust service providers and card issuers *FutureID* will provide an integrative framework, which eases using their authentication and signature related products across Europe and beyond.

To demonstrate the applicability of the developed technologies and the feasibility of the overall approach *FutureID* will develop two pilot applications and is open for additional application services who want to use the innovative *FutureID* technology.

FutureID is a three-year duration project funded by the European Commission Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 318424.

5 Introduction

Anonymous credentials allow for cryptographically strong user authentication while preserving the user's privacy by giving users full control over the information they reveal. In such a system, three types of entities exist: *Issuers* certify lists of attributes to *users*, who can then prove possession of their credentials to *verifiers*, thereby revealing an arbitrary subset of the attributes of (potentially multiple) credentials. While malicious users must not be able to pool their credentials or to prove possession of credentials they never received from an issuer, unrevealed attributes must not be leaked to the verifier and different presentations of the same (set of) credentials cannot be linked to each other.

This basic functionality can be enriched in a number of ways such that, e.g., users can carry over attributes from one credential to another one without the issuer learning the concrete attribute values, users can prove that some attributes are equal to each other, or that credentials can be revoked. Furthermore, in practical scenarios, the usage of credentials is often bound to the possession of a specific device to further increase the security level: For instance, a public authority may link credentials that allow a user to cast votes in an election to the possession of a physical smart card to disable attackers from phishing the voting credentials from the user and to cast a vote on his behalf. Another use case may be that a user may only access the online store of a company if it is using a device of this company. Technically this may be realized by adding an unextractable device secret key as an attribute to the credential.

However, binding credentials to a specific device also means that the user cannot use his credential any longer when the device breaks or gets stolen, as he does not have access to some of the required attributes (i.e., the device key) any longer. In theory, in this case a user could get a new credential by going through the entire issuance process again, but this might be undesirable in practice, e.g., because of off-line authentication steps including appearance in person at a public authority. Thus, efficient means of re-obtaining a credential in case of loss of a device need to be developed, potentially requiring the user to store some backup data to convince the issuer that he already owned a credential before.

In addition, users also often store pure software credentials on their smart phones or tablets to increase their flexibility and convenience. Now, when the device gets stolen, the adversary learns all the credentials – device-bound or not – and all the data needed to use them, and thus obtains the possibility to impersonate the user. It is thus necessary that the re-issuance process ensures that only the user has a possibility to convince the issuer that he was the legitimate owner of the credential. Solving this issue clearly requires the user to additionally store some secret data related to the credential and the backup. This secret data must not be necessary to present a credential and must not be stored on any device the user is carrying with him, but, e.g., on an offline machine or in a secure vault. In particular, upon loss a device, this secret information will not be leaked to the adversary.

In this document we describe three efficient constructions for backing up and re-issuing credentials to users. Each construction offers a higher level of security to the user, but also comes at higher computational costs. The first construction, presented in § 8.1, only protects users against broken devices, and has no impact on the every-day usage of credentials. The next construction, presented

in § 8.2, also protects users against theft of their devices and credentials. The additional costs are essentially given by one additional attribute that has to be embedded into every credential. The last construction, presented in § 8.3, not only disables adversaries from getting the stolen credentials re-issued from an issuer, but from obtaining *any* new usable credential that is linked to the user's identity, even if the issuer is corrupted. For this construction every presentation of a credential has to be extended by proving knowledge of a secret signature. While this is a considerable overhead, it can still be realized efficiently in practice.
We do not give concrete instantiations but keep the description at an abstract level, such that the construction can directly be applied to most existing anonymous credential schemes.

5.1 Related Work

Anonymous credential systems were originally envisioned by Chaum [Cha81, Cha85], and subsequently a large number of schemes have been proposed. The currently most prevalent solutions are IBM's Identity Mixer based on CL-signatures [CH02, CL01, CL02, CL04] and Microsoft's U-Prove based on Brands' signatures [Bra99, PZ13]. The most holistic definitional framework for anonymous credentials was recently presented by Camenisch et al. [CKL+14]. The syntax we introduce in the following is inspired by theirs, but kept at the minimum that is required to present our constructions.
Only little work has been done on backup of anonymous credentials. The construction presented in § 8.1 can be seen as an abstraction of those of Camenisch et al. [CLN12] and Baldimtsi et al. [BCLN12]. Our second construction, presented in § 8.2 is a variant of that proposed by Camenisch et al. [CKLN14].

6 Preliminaries

In the following we introduce some notation and the cryptographic background that is needed for our constructions. Readers familiar with the topic may safely skip this chapter upon first reading.

6.1 Notation

In the following we introduce the notation that we use in the rest of this paper.
We denote algorithms by type-writer fonts, e.g., $\mathtt{A}$, $\mathtt{B}$. For deterministic algorithms we write $a \leftarrow \mathtt{A}(in)$, if a is the output of $\mathtt{A}$ on input in, and we write $a \leftarrow_r \mathtt{A}(in)$ if $\mathtt{A}$ is probabilistic. For an interactive protocol $\langle \mathtt{A}; \mathtt{B} \rangle$ let $(out_{\mathtt{A}}, out_{\mathtt{B}}) \leftarrow_r \langle \mathtt{A}(in_{\mathtt{A}}); \mathtt{B}(in_{\mathtt{B}}) \rangle$ denote that, on private inputs $in_{\mathtt{A}}$ to $\mathtt{A}$ and $in_{\mathtt{B}}$ to $\mathtt{B}$, $\mathtt{A}$ and $\mathtt{B}$ obtained outputs $out_{\mathtt{A}}$ and $out_{\mathtt{B}}$, respectively. We write $(x_i)_{i=1}^k$ to denote the vector $(x_1, \ldots, x_k)$.
By λ we denote the main security parameter, and by ε the empty string or empty list, depending on the context.

6.2 Basic Cryptographic Primitives

In the following we informally recapitulate the cryptographic primitives that we use in our constructions.

Commitments. A commitment scheme is a triple of algorithms $(\mathtt{CKGen}, \mathtt{Commit}, \mathtt{Open})$. $\mathtt{CKGen}$ generates a public commitment key *pck*. Taking this key as input, a user can commit to a message m as $(c, o) \leftarrow_r \mathtt{Commit}(m, \mathit{pck})$, obtaining a commitment c and opening information o. The validity of (c, o) with respect to m can be verified using the algorithm $\mathtt{Open}(c, o, m, \mathit{pck})$. For notational convenience, we will no longer make the commitment key explicit in the following.
Intuitively, a commitment scheme should satisfy the following security properties. First, it should be *complete*, meaning that an honestly computed commitment/opening pair always passes the algorithm $\mathtt{Open}$. Second, it should be *binding*, meaning that no adversary can come up with a commitment c, openings o, o', and two distinct messages m, m', such that $\mathtt{Open}(c, o, m) = \mathtt{Open}(c, o', m') = 1$. That is, every commitment can only be opened to a single value m. Finally, the scheme should be *hiding*, meaning that only knowing c, but not m or o, does not leak any information about the message m contained in c.
For detailed constructions and formal definitions we refer to the original literature, e.g., Pedersen [Ped91] or Damgård and Fujisaki [DF02].

Digital signatures. A digital signature is a triple of algorithms $(\mathtt{SKGen}, \mathtt{Sign}, \mathtt{SVerify})$. $\mathtt{SKGen}$ takes as input the security parameter λ and outputs a signature/verification key pair $(\mathit{ssk}, \mathit{svk})$. A signature on a message m can now be computed as $\sigma \leftarrow_r \mathtt{Sign}(m, \mathit{ssk})$. The validity of the signature can be checked using the signature verification key as $0/1 \leftarrow \mathtt{SVerify}(\sigma, m, \mathit{svk})$.

Informally, a signature scheme has to satisfy the following security properties. First, it must be *complete*, meaning that an honestly computed signature always passes $\mathtt{SVerify}$. Second, the scheme must be *existentially unforgeable under chosen message attacks*, meaning that no adversary can come up with a valid signature on a message that he had not obtained a signature for, even if the adversary may obtain an arbitrary number of signatures on messages of his choice.
For constructions and formal definitions we refer to the original literature, e.g., Camenisch and Lysyanskaya [CL02] and Brands [Bra99].

Zero-knowledge proofs of knowledge. A zero-knowledge proof of knowledge for some binary relation $\mathcal{R}$ is a two-party protocol $\langle \mathtt{Prover}; \mathtt{Verifier} \rangle$. $\mathtt{Prover}$ takes as inputs some $(x, w) \in \mathcal{R}$, while $\mathtt{Verifier}$ only takes x as an input. At the end of the interaction, $\mathtt{Verifier}$ outputs a single bit, indicating whether it accepts or rejects the proof.
Informally, a zero-knowledge proof of knowledge has to satisfy the following security properties. First, it must be *complete*, meaning that $\mathtt{Verifier}$ always accepts if $(x, w) \in \mathcal{R}$ for an honest prover. Second, it must be *sound*, meaning that with high probability $\mathtt{Verifier}$ will reject the proof if $(x, w) \notin \mathcal{R}$ for every adversarial prover. Finally, the protocol must be *zero-knowledge*, meaning that no information (in a computational sense) about the secret w is leaked from the interaction. We further call the zero-knowledge proof *non-interactive*, if the protocol only consists of a single message being exchanged between the two parties.
For formal definitions we again refer to the original literature, e.g., Bellare and Goldreich [BG92]. It is a well-known result that zero-knowledge proofs of knowledge exist for all binary relations in $\mathcal{NP}$ [GMW91]. Furthermore, highly efficient protocols for proving many practically relevant relation exist, cf., e.g., Schnorr [Sch91], Camenisch et al. [CKY09], Fujisaki et al. [DF02, FO97], and Lipmaa [Lip03].

Hash function. A cryptographic hash function $\mathtt{H}$ is a function taking elements of $\{0, 1\}^*$ as inputs, and outputting elements within a finite co-domain.
Informally, a cryptographic hash function has to be *collision resistant*, meaning that it is computationally infeasible to find $x \neq x' \in \{0, 1\}^*$ such that $\mathtt{H}(x) = \mathtt{H}(x')$.

6.3 Anonymous Credentials

After informally introducing anonymous credentials in § 5, we next formally describe the syntax of such systems. Furthermore, in § 6.3.2, we informally recapitulate the security properties that they need to satisfy.

6.3.1 Syntax of Anonymous Credential Systems

We next introduce the syntax we use for describing anonymous credential systems (without backup). We here only recapitulate those algorithms that are needed for our constructions and do not focus on enhanced features of anonymous credential systems such as pseudonyms or equality predicates. More general interfaces and descriptions may, e.g., be found in Camenisch et al. [CKL+14].
At the beginning of the lifetime of the systems, some system parameters are generated using $\mathtt{SPGen}$. Furthermore, every issuer computes a secret/public issuer key pair using $\mathtt{IKGen}$. Users can obtain credentials from issuers using the protocol $\langle \mathcal{U}.\mathtt{Issue}; \mathcal{I}.\mathtt{Issue} \rangle$. On the one hand, the user may decide which attributes he wants to reveal to the issuer, and on the other hand, the issuer has

the possibility to require some attributes to be carried-over from previous trusted credentials owned by the user. Users can then derive a presentation token from a (single) credential using `Present`, which can then be publicly verified using `Verify`. Again, users can specify which attributes they want to reveal and which should be kept secret. They can further prove that certain attributes are also equal to the messages contained in given commitments. This is useful for advanced issuance with carry-over attributes, and furthermore allows users to link certain presentation tokens to each other: For instance, a verifier may require that the user possesses two credentials from different issuers. The user may then commit to a uniquely identifying attribute (e.g., a secret user key, his social insurance number, etc.), and in each presentation token prove that the name certified in the credential is identical to the one contained in the commitment. By the binding property of the commitment, the verifier will thus be ensured that the two presentation tokens were indeed derived from credentials issued to the same user.
We now formally define the interfaces:

System parameter generation. This algorithm takes as input the security parameter in unary, i.e., 1^λ, and outputs system-wide parameters *spar*:

$$\mathit{spar} \leftarrow_r \mathtt{SPGen}(1^\lambda) .$$

Issuer key generation. This algorithm takes as input the system parameters, and outputs a secret key/public key pair for an issuer:

$$(\mathit{isk}, \mathit{ipk}) \leftarrow_r \mathtt{IKGen}(\mathit{spar}) .$$

Issuance. This interactive protocol is executed between a user and an issuer whenever the user wishes to obtain a credential from the issuer. The user takes as input the issuer's public key *ipk*, all attributes $(a_i)_{i=1}^n$, a list of commitments/openings $(c_i, o_i)_{i\in C}$, and a set of integers R. The issuer takes as inputs its secret key *isk*, the revealed attributes $(a_i)_{i\in R}$, the commitments $(c_i)_{i\in C}$. At the end of the interaction, the user either outputs $\perp$ or a new credential *cred* on $(a_i)_{i=1}^n$, and the issuer a bit indicating whether the interaction was successful or not:

$$(\perp/\mathit{cred}, 0/1) \leftarrow_r \langle(\mathcal{U}.\mathtt{Issue}(\mathit{ipk}, (a_i)_{i=1}^n, (c_i, o_i)_{i\in C}, R); \mathcal{I}.\mathtt{Issue}(\mathit{isk}, (a_i)_{i\in R}, (c_i)_{i\in C}\rangle .$$

The semantics of $(c_i, o_i)_{i\in C}$ are the following: In many practical scenarios, a user has to present a credential (e.g., issued by a public authority) to an issuer before the issuer is willing to issue a credential himself. Furthermore, the issuer might require that some attributes (e.g., the birth date of the user) are carried-over into the new credential, but the user is not willing to reveal the attribute value to the issuer. In this case, the user can compute a commitment/opening pair to the specific attribute for the presentation, thereby proving that the commitment contains the same value that was also certified by the credential. Then, at issuance, the protocol ensures that the value contained in the commitment will also be included in the new credential. That is, the commitments are used to bind a run of the issuance protocol to a preceding presentation.

The set R specifies the indices of those attributes whose values are revealed to the issuer at issuance time and might heavily depend on the concrete application.

Presentation token generation. This algorithm takes as input the issuer's public key *ipk*, a credential *cred* and the contained attributes $(a_i)_{i=1}^n$, a list of commitments/openings $(c_i, o_i)_{i\in C}$, and a set of integers R. It outputs a presentation token *pt*:

$$\mathit{pt} \leftarrow_r \mathtt{Present}(\mathit{ipk}, \mathit{cred}, (a_i)_{i=1}^n, (c_i, o_i)_{i\in C}, R) .$$

Verification. This algorithm takes as input the issuer's public key *ipk*, a presentation token *pt*, a set of revealed attributes $(a_i)_{i \in R}$, and commitments $(c_i)_{i \in C}$ and outputs a bit indicating whether to accept or to reject the presentation token:

$$0/1 \leftarrow \mathtt{Verify}(\mathit{ipk}, \mathit{pt}, (a_i)_{i \in R}, (c_i)_{i \in C}) \, .$$

6.3.2 Security Requirements

In the following we briefly recap the security requirements a basic anonymous credential system has to satisfy. For a more formal treatment, we refer to Camenisch et al. [CKL+14].

Correctness. If all parties behave honestly, a user should always be able to prove possession of a credential that he previously got issued from an issuer.

Unforgeability. No adversary must be able to prove possession of a credential that he did not previously get issued from an issuer. This must hold true even if the adversary has obtained an arbitrary number of credentials, and has seen an arbitrary number of presentation tokens of other users.

Privacy. This property guarantees the user's privacy. On the one hand, the issuer should not learn any information about the attributes that were not explicitly revealed during issuance. On the other hand, presentation tokens derived from the same credential should neither be linkable to a specific issuance session. In particular, attributes that were not explicitly revealed must not get leaked by a presentation token.

We say that an anonymous credential system is *weakly private*, if presentation tokens derived from the same credential can be linked to each other, as is the case for UProve [PZ13]. The system is said to satisfy *strong privacy*, if this is not the case, such as, e.g., Idemix [CL02, CH02].

7 Anonymous Credentials with Backup

In the following we introduce the syntax and security requirements for anonymous credential systems with backup.

7.1 Syntax of Anonymous Credentials with Backup

In addition to the algorithms defined in § 6.3.1, an anonymous credential system with backup consists of a number of additional algorithms. Each user computes a vault user secret/public key pair using the algorithm VKGen. The vault user secret key is a long-term secret that can be stored in a secure place by the user, and must never be used to present credentials, but may be required to revoke credentials and to get lost credentials re-issued. Before getting a new credential issued, the user further computes a backup secret/public key pair for this specific credential using BKGen. As before, the backup secret key is not required for presenting a credential, but may be required for revocation and re-issuance. Now, to backup a credential, the user runs Backup, obtaining a secret/public backup token. To get a credential re-issued, the user sends the public backup token to the issuer, and they then execute the protocol $\langle \mathcal{U}.\mathtt{Reissue}; \mathcal{I}.\mathtt{Reissue}\rangle$. As for issuance, the user can decide which attributes to reveal to the issuer at re-issuance.
In the following we now formally specify the additional algorithms that are required to build an anonymous credential system with backup.

Vault key generation. This algorithm takes as input the system parameters and outputs a vault user secret key/public key pair:

$$(\mathit{vusk}, \mathit{vupk}) \leftarrow_r \mathtt{VKGen}(\mathit{spar}) .$$

This algorithm is executed only once by each user.

Backup key generation. This algorithm takes as inputs the system parameters and the user's vault key pair, and outputs a backup key pair:

$$(\mathit{bsk}, \mathit{bpk}) \leftarrow_r \mathtt{BKGen}(\mathit{spar}, \mathit{vusk}, \mathit{vupk}) .$$

This algorithm is executed every time before a credential gets (re-)issued, i.e., in contrast to $(\mathit{vusk}, \mathit{vupk})$, $(\mathit{bsk}, \mathit{bpk})$ may be different for each credential a user owns.

Backup. This algorithm takes as input the issuer's public key *ipk*, a credential *cred*, the certified attributes $(a_i)_{i=1}^n$, a backup key pair $(\mathit{bsk}, \mathit{bpk})$, and the vault key pair $(\mathit{vusk}, \mathit{vupk})$, and outputs a secret/public backup token:

$$(\mathit{sbt}, \mathit{pbt}) \leftarrow_r \mathtt{Backup}(\mathit{ipk}, \mathit{cred}, (a_i)_{i=1}^n, \mathit{bsk}, \mathit{bpk}, \mathit{vusk}, \mathit{vupk}) .$$

Reissue. This interactive protocol is executed between a user and an issuer whenever the user wishes to obtain a credential from the issuer. The user takes as inputs the issuer's public key *ipk*, the secret part *sbt* of a backup token, the attributes to be certified by the new credential $(a_i)_{i=1}^n$, a set of integers R specifying the indices of the revealed attributes, the secret backup key *bsk*, and the vault user secret key *vusk*. The issuer takes as input its secret key *isk*, the public part *pbt* of a backup token, and the revealed attributes $(a_i)_{i \in R}$. At the end of the interaction, the user either outputs $\perp$ or a new credential *cred* on $(a_i)_{i=1}^n$, and the issuer a bit indicating whether the interaction was successful or not:

$$(\perp/\mathit{cred}, 0/1) \leftarrow_r \langle \mathcal{U}.\mathtt{Reissue}(\mathit{ipk}, \mathit{sbt}, (a_i)_{i=1}^n, R, \mathit{bsk}, \mathit{vusk}); \mathcal{I}.\mathtt{Reissue}(\mathit{isk}, \mathit{pbt}, (a_i)_{i \in R}) \rangle .$$

7.1.1 Overview of Parameters

Parameter	Stored	Description
$(\mathit{vusk}, \mathit{vupk})$	offline / vault	vault user key pair; one per user
$(\mathit{bsk}, \mathit{bpk})$	offline / vault	backup key pair; one per credential
$(\mathit{sbt}, \mathit{pbt})$	online / cloud	backup tokens; one per credential

Table 7.1: Overview of the additional parameters for anonymous credentials with backup.

Table 7.1 gives an overview of the additional parameters introduced by the backup algorithms. Parameters marked as being stored offline have to be kept secret by the user such that they are never leaked to the adversary, and must thus not be stored on any device the adversary could get access to, but, e.g., on an offline data storage medium. Consequently, these values are not required for any presentations, but may be used for (re-)obtaining or revoking credentials.
On the other hand, parameters marked as being stored online may be stored, e.g., in the cloud. In particular, for the backup mechanisms to work, we do not make any assumptions on the confidentiality of these tokens, i.e., the adversary will not learn any more information from these tokens than what he can already learn from a stolen device. Also, an adversary not having the device will still not be able to impersonate the user, even if he has access to these values. However, as the backup tokens will typically contain personal information about the user, it is still recommended to not make the backup tokens publicly accessible for privacy reasons, but to store them in an encrypted way.

7.2 Security Requirements

In the following we informally describe the security requirements that are requested from an anonymous credential system with backup. They can be seen as a direct generalization of the requirements listed in § 6.3.2.

Correctness. If all parties behave honestly, a user should always be able to prove possession of a credential that he previously got issued from an issuer. Furthermore, an honest user should always be able to successfully get a credential re-issued for which he has correctly computed all the specified backup algorithms.

Unforgeability. No adversary must be able to prove possession of a credential that he did not previously get issued from an issuer. This must hold true even if the adversary has obtained an arbitrary number of credentials, has seen an arbitrary number of presentation tokens and re-issuance protocols of other users, and even if he gained access to the backup tokens $(\mathit{sbt}, \mathit{pbt})$ of an honest user. The latter in particular means that knowing the vault and backup secret keys is mandatory for successfully using backup tokens in re-issuance protocols.

Privacy. This property guarantees the user's privacy. On the one hand, the issuer should not learn any information about the attributes that were not explicitly revealed during (re-)issuance. On the other hand, presentation tokens derived from the same credential should neither be linkable to their (re-)issuance session, nor should they reveal whether they were derived from original or re-issued credentials. Also, re-issuance sessions should be be linkable to each other, and not to specific issuance sessions.

Again, we call the system *weakly* or *strongly private*, if presentation tokens derived from the same credential can or cannot be linked to each other. Furthermore, in the case of weak privacy, presentation tokens may be linkable to subsequent re-issuance sessions. However, presentations of the new credential should then again be unlinkable to those of the old credential.

8 Constructions

In the following we now describe three constructions for backing up anonymous credentials. As described in § 5, each construction comes at higher computational costs than the previous one, but also offers higher security guarantees to all parties in the system.

8.1 Recovering Credentials for Disfunctional Devices

In the following we propose a very simple construction that can be used to backup anonymous credentials. On a high level, to backup, the user stores a presentation token of his credential that also links all the attributes that are to be carried-over into a new credential to commitments. Then, to restore, this presentation token and the commitments are used as if they were coming from a normal presentation preceding an advanced issuance protocol.

System parameter generation. The algorithm `SPGen` is left unchanged.

Issuer key generation. The algorithm `IKGen` is left unchanged.

Vault key generation. The algorithm `VKGen` always outputs $(\mathit{vusk}, \mathit{vupk}) := (\varepsilon, \varepsilon)$.

Backup key generation. The algorithm `BKGen` always outputs $(\mathit{vusk}, \mathit{vupk}) := (\varepsilon, \varepsilon)$.

Presentation. The algorithm `Present` is left unchanged.

Verification. The algorithm `Verify` is left unchanged.

Backup. The algorithm Backup proceeds as follows:

- First, the user computes commitments and openings $(c_i, o_i) \leftarrow_r \mathtt{Commit}(a_i)$ for all attributes that are to be carried over into a new credential at re-issuance. These attributes might, e.g., contain the name or birth date of a user; other attributes such as the issuance date of the credential or device secret keys that cannot be extracted from the device do not need to be committed to. We denote the set of indices of the attributes that are to be backed up by B.
- Next, the algorithm computes the following presentation token:

$$\mathit{pt} \leftarrow_r \mathtt{Present}(\mathit{ipk}, \mathit{cred}, (a_i)_{i=1}^n, (c_i, o_i)_{i\in B}, \emptyset)\,.$$

- The public and secret backup tokens are given by:

$$\begin{aligned} \mathit{sbt} &= (\mathit{pt}, (c_i, o_i, a_i)_{i\in B}), \text{ and} \\ \mathit{pbt} &= (\mathit{pt}, (c_i)_{i\in B})\,. \end{aligned}$$

Reissue. The re-issuance protocol consists of the following steps:

- The issuer first checks whether *pt* verifies correctly, i.e., it runs:

  $$\texttt{Verify}(\mathit{ipk}, \mathit{pt}, \emptyset, (c_i)_{i \in B}) ,$$

  and aborts if this is not the case.
- Finally, the user and the issuer perform the following protocol:

  $$(\perp/\mathit{cred}, 0/1) \leftarrow_r \langle \mathcal{U}.\texttt{Issue}(\mathit{ipk}, ((a'_i)_{i=1}^n), (c_i, o_i)_{i \in B}, R); \\ \mathcal{I}.\texttt{Issue}(\mathit{isk}, (a'_i)_{i \in R}, (c_i)_{i \in B}) \rangle .$$

  Here, the a'_i are the new attributes that are certified by the new credential. In particular, because of the commitments c_i, the issuer is ensured that $a'_i = a_i$ for all $i \in B$. Furthermore, the set R of revealed attributes may be empty, or include indices of, e.g., the expiration date of the new credential.

On the positive side this construction allows a user to efficiently re-obtain a credential if he loses access to some of the required attributes (e.g., the device secret keys upon disfunction of the device). Furthermore, it does not require any changes to the underlying credential system. It is therefore easy to deploy in real-world systems, and does not increase the costs for the every-day usage of the credentials.
However, it is easy to see that the given solution does not give any security guarantees against theft. This is because upon loss of the device carrying the credential, the adversary will learn all the data the user knows as well, and thus there is no way for the user to prove legitimate ownership of the credential.

8.2 Adding Protection against Theft

In the following we propose a generic extension for any anonymous credential scheme that tackles the problem of stolen credential by letting the user secretly store some data that later will allow him to prove legitimate ownership.
On a high level, the idea is that a user embeds the special attribute *bpk* into his credential, and privately stores a trapdoor *bsk* to *bpk*. This trapdoor will never be used for using, i.e., presenting, the credential, and thus the user does not need to, and must not, store *bsk* on his devices; furthermore, *bpk* will always be treated as an unrevealed attribute at issuance and presentation. As a backup token, the user stores a presentation token for the credential, revealing *bpk*. To get a credential re-issued, the user now sends the presentation token and the secret trapdoor *bsk* to the issuer, who will only engage in a re-issuance protocol if the presentation token is correct and if *bsk* is indeed a trapdoor for *bpk*. The latter ensures that only the legitimate owner of the credential is able to initiate a re-issuance, as an adversary gaining access to the user's device would be able to compute the required valid presentation token, but would not be able to present the secret trapdoor *bsk* to the issuer.

System parameter generation. The algorithm $\texttt{SPGen}$ is left unchanged.

Issuer key generation. The algorithm $\texttt{IKGen}$ is left unchanged.

Vault key generation. $\texttt{VKGen}(\mathit{spar})$ always outputs $(\mathit{vusk}, \mathit{vupk}) := (\varepsilon, \varepsilon)$.

Backup key generation. $\mathtt{BKGen}(\mathit{spar}, \mathit{vusk}, \mathit{vupk})$ computes $(\mathit{bsk}, \mathit{bpk})$ as a fresh authentication key pair. The probably most efficient solution is to choose bsk uniformly at random and set $\mathit{bpk} := \mathtt{H}(\mathit{bsk})$, where $\mathtt{H}$ is a cryptographic hash function mapping into the attribute space of the credential system.

Issuance. Before obtaining a credential for some attributes $(a_i)_{i=1}^k$, the list of attributes is extended by bpk, which is treated as an unrevealed attribute at issuance. That is, the issuance protocol is slightly adapted to:

$$(\perp/\mathit{cred}, 0/1) \leftarrow_r \langle(\mathcal{U}.\mathtt{Issue}(\mathit{ipk}, ((a_i)_{i=1}^n, \mathit{bpk}), (c_i, o_i)_{i\in C}, R); \\ \mathcal{I}.\mathtt{Issue}(\mathit{isk}, (a_i)_{i\in R}, (c_i)_{i\in C})\rangle .$$

Presentation. When presenting a credential, the additional attribute bpk is always treated as an unrevealed attribute:

$$\mathit{pt} \leftarrow_r \mathtt{Present}(\mathit{ipk}, \mathit{cred}, ((a_i)_{i=1}^n, \mathit{bpk}), (c_i, o_i)_{i\in C}, R) .$$

Verification. The algorithm $\mathtt{Verify}$ is left unchanged.

Backup. The $\mathtt{Backup}$ algorithm proceeds as follows:

- It first computes commitments and openings $(c_i, o_i) \leftarrow_r \mathtt{Commit}(a_i)$ $(i \in B)$ to all attributes a_i that are supposed to be carried into a new credential at re-issuance.
- The algorithm next computes the following presentation token pt':

  $$\mathit{pt}' \leftarrow_r \mathtt{Present}(\mathit{ipk}, \mathit{cred}, ((a_i)_{i=1}^n, \mathit{bpk}), (c_i, o_i)_{i\in B}, \{n+1\}) .$$

  Note that the set of revealed attributes $\{n+1\}$ means that only the value of bpk is revealed by pt'.
- The public and secret backup tokens are given by:

  $$\mathit{sbt} = (\mathit{pt}', \mathit{bpk}, (c_i, o_i, a_i)_{i\in B}), \text{ and} \\ \mathit{pbt} = (\mathit{pt}', \mathit{bpk}, (c_i)_{i\in B}) .$$

Reissue. The re-issuance protocol consists of the following steps:

- The issuer first checks that the presentation token pt' is correct, i.e., it runs:

  $$\mathtt{Verify}(\mathit{ipk}, \mathit{pt}', \mathit{bpk}, (c_i)_{i\in B}) ,$$

  and aborts if this is not the case.
- In the next step, the user sends to the issuer the backup secret key bsk, and the issuer checks that $(\mathit{bsk}, \mathit{bpk})$ indeed form a valid authentication pair. The issuer aborts if this is not the case. This check guarantees that only the legitimate owner of the credential is able to initiate a re-issuance protocol, as bsk is not known to any other party, even after potentially gaining access to all credentials of a user.

- Finally, the user and the issuer perform the following protocol:

$$(\perp/\mathit{cred}, 0/1) \leftarrow_r \langle \mathcal{U}.\texttt{Issue}(\mathit{ipk}, ((a'_i)_{i=1}^n, \mathit{bpk}'), (c_i, o_i)_{i\in B}, R); \\ \mathcal{I}.\texttt{Issue}(\mathit{isk}, (a'_i)_{i\in R}, (c_i)_{i\in B})\rangle \,.$$

  Here, $(\mathit{bsk}', \mathit{bpk}')$ is a new backup key pair computed as described above, and the remaining inputs are as in § 8.1.

Regarding issuance and presentation of credentials, the computational costs of the above construction are essentially given by embedding one additional attribute into the credential, and always treating it as an unrevealed attribute.

8.2.1 Disabling Users from Duplicating Credentials

In many practical settings, the issuer may want to be ensured that the backup procedure cannot be used to duplicate credentials. That is, the issuer may only be willing to re-issue a credential if the user can convince him that the old credential was revoked before re-issuance. If the issuer and the revocation authority are the same entity, one possibility is to directly use *bpk* as the revocation handle, and letting the issuer (i) check that *bpk* is unrevoked when checking the validity of π', and (ii) revoke *bpk* before starting the re-issuance protocol. In the following we present an extension of the above constructions which can also be used if the revocation authority and the issuer are not the same identity.

- When generating backup keys, the user additionally generates a second authentication pair, the revocation key pair, $(\mathit{rsk}, \mathit{rpk})$. Similarly to *bpk*, *rpk* is added as an attribute to the credential, and always treated as an unrevealed attribute. The value of *rsk* is kept secret and is not needed to perform any presentation proofs.
- When generating the backup tokens, the user additionally computes a commitment/opening pair $(c_{\mathsf{rev}}, o_{\mathsf{rev}})$ to *rpk*. The presentation proof π' is modified such that it additionally proves that c_{rev} is consistent with the attribute *rpk*. The user additionally computes π'' as a presentation token that proves possession of a credential *cred*, revealing the value of *rpk*.
- To revoke a credential, the user sends π'' and $(\mathit{rsk}, \mathit{rpk})$ to the revocation authority, who checks that π'' verifies correctly and that $(\mathit{rsk}, \mathit{rpk})$ is a valid authentication key pair. The latter check guarantees that only the legitimate owner of the credential *cred* is able to revoke the credential, similarly to the case of re-issuance. After revoking *rpk*, the revocation authority signs *rpk* and sends the signature σ to the user.
- Before issuing a new credential, the issuer now first checks that the value *bpk* has not yet been used, i.e., that no new credential has been obtained so far for a fixed first credential. The user then proves to the issuer in zero-knowledge that he possesses a signature σ under the revocation authority's key on the message contained in c_{rev}. As π' showed that c_{rev} contains the same value that serves a the revocation handle in the credential, this will convince the issuer that the old credential was indeed revoked.

8.3 Disabling the Adversary from Obtaining New Credentials

In the previous section we presented a construction that provides high security guarantees and only comes with a small computational overhead compared to the underlying plain credential system.

However, it does not tackle the following problem of an adversary obtaining a new credential on behalf of the user after stealing his device.
As a motivating example, consider the following scenario. An adversary gains access to the user's electronic identity card containing a user credential issued by a public agency, and assume that the user does not immediately take notice of this fact and thus does not immediately revoke his credential. Then, of course, the adversary can use the credential and still derive presentation tokens from it. While this is unavoidable (as we assumed that the device contains all the information required to use the credential), the adversary may even request new credentials from new issuers, carrying over parts of the user's attributes into this new credentials. For instance, the adversary may now obtain a credential for some online casino and link this credential to the user's identity. The adversary could then impersonate the user also using this new credential and, e.g., run into debt on behalf of the user, even though the user had never requested a credential for this casino.
In the following we propose a construction which resolves this problem by disabling the adversary from obtaining new presentable credentials on behalf of the legitimate owner of a first credential. In contrast to the previous construction, the user no longer uses his long-term secret keys to get a credential re-issued, but to sign a unique attribute in each of his credentials under his own secret signing key. Then, at presentation, the user not only proves that he possesses the required credentials, but also that he knows signatures on the unique credential identifiers under the correct key. By embedding the vault user public key *vupk* into each credential, blindly carrying it over from one credential into the next one in case of advanced issuance and using it as an identifying attribute to link presentations to each other (cf. the discussion in § 6.3.1), the adversary will now no longer be able to obtain a new credential which he can later successfully present on behalf of the user, as this would require him to generate a signature on the credential identifier of the new credential under the vault user secret key *vusk*. However, by construction, *vusk* is not needed to use a credential and therefore not stored on the device, either. Thus, the security of the construction follows from the unforgeability of the used signatures scheme.
While adding another layer of security to the credential system, the following construction is computationally more expensive than the one presented in the previous section. Every presentation also requires an additional proof of knowledge of a secret signature on a secret message under a secret key. Furthermore, the signature scheme and the credential system have to be compatible, i.e., the public verification key of the signature scheme has to be in the attribute space of the credential system.
In the following we formally specify our construction.

System parameter generation. The algorithm `SPGen` is left unchanged.

Issuer key generation. The algorithm `IKGen` is left unchanged.

Vault key generation. $\texttt{VKGen}(\mathit{spar})$ generates a fresh signing/verification key pair $(\mathit{ssk}, \mathit{svk})$ for an existentially unforgeable signature scheme where *svk* lies in the attribute space of the credential system. It outputs $(\mathit{vusk}, \mathit{vupk}) := (\mathit{ssk}, \mathit{svk})$.

Backup key generation. The algorithm `BKGen` always outputs $(\varepsilon, \varepsilon)$, i.e., there is no credential-specific backup key.

Issuance. To issue a credential, the following steps are performed:

- First, the user and the issuer perform a joint computation at the end of which the user obtains a commitment/opening pair $(c_{\mathit{cid}}, o_{\mathit{cid}})$ to a fresh credential identifier *cid*. It is required that *cid* is uniformly random in the attribute space if at least one of the parties behaves honestly, and that the issuer does not learn any information about its value.

- The list $(a_i)_{i=1}^n$ is extended by $(\mathit{vupk}, \mathit{cid})$. The user and the issuer than engage in the following protocol:

$$(\perp/\mathit{cred}', 0/1) \leftarrow_r \langle \mathcal{U}.\mathtt{Issue}(\mathit{ipk}, ((a_i)_{i=1}^n, \mathit{vupk}, \mathit{cid}), ((c_i, o_i)_{i\in C}, (c_{\mathit{cid}}, o_{\mathit{cid}})), R); \\ \mathcal{I}.\mathtt{Issue}(\mathit{isk}, (a_i)_{i\in R}, ((c_i)_{i\in C}, c_{\mathit{cid}})) \rangle\,.$$

- Next, the user signs *cid* using his vault user secret key *vusk*, obtaining a signature σ.
- The user finally outputs $\mathit{cred} := (\mathit{cred}', \sigma)$.

Presentation. To compute a presentation token *pt*, a user performs the following computations:

- He first computes a fresh commitment $(c_{\mathit{cid}}, o_{\mathit{cid}})$ to *cid*.
- He then computes the following presentation token:

$$\mathit{pt}' \leftarrow_r \mathtt{Present}(\mathit{ipk}, \mathit{cred}', ((a_i)_{i=1}^n, \mathit{vupk}, \mathit{cid}), ((c_i, o_i)_{i\in C}, (c_{\mathit{cid}}, o_{\mathit{cid}})), R)\,.$$

- Next, the user computes a non-interactive zero-knowledge proof of knowledge π, proving that he knows a (secret) signature σ on the (secret) value *cid* contained in c_{cid}, that verifies under the (secret) value *vupk*. Remember that *vupk* was the verification key of a signature scheme, corresponding to the signing key *vusk*.
- The user finally outputs the presentation token $\mathit{pt} := (\mathit{pt}', c_{\mathit{cid}}, \pi)$.

Verification. The verifier first checks whether `Verify` accepts the presentation token pt', and then checks if π verifies correctly. It outputs 1 if and only if both these tests accept.

Backup. The algorithm `Backup` proceeds as follows:

- It first computes commitments and openings $(c_i, o_i) \leftarrow_r \mathtt{Commit}(a_i)$ to all attributes a_i ($i \in B$) that are supposed to be carried into a new credential at re-issuance. Furthermore, it computes a commitment/opening pair $(c_{\mathit{vupk}}, o_{\mathit{vupk}}) \leftarrow_r \mathtt{Commit}(\mathit{vupk})$ to the vault user public key.
- The algorithm next computes the following presentation token pt':

$$\mathit{pt}' \leftarrow_r \mathtt{Present}(\mathit{ipk}, \mathit{cred}', ((a_i)_{i=1}^n, \mathit{vupk}, \mathit{cid}), ((c_i, o_i)_{i\in B}, (c_{\mathit{vupk}}, o_{\mathit{vupk}})), \emptyset)\,.$$

- The public and secret backup tokens are given by:

$$\mathit{sbt} = (\mathit{pt}', (c_i, o_i, a_i)_{i\in B}, (c_{\mathit{vupk}}, o_{\mathit{vupk}})), \text{ and} \\ \mathit{pbt} = (\mathit{pt}', (c_i)_{i\in B}, c_{\mathit{vupk}})\,.$$

Reissue. The re-issuance protocol consists of the following steps:

- The issuer first checks that the presentation token pt' is correct, i.e., it runs:

$$\mathtt{Verify}(\mathit{ipk}, \mathit{pt}', \emptyset, ((c_i)_{i\in B}, c_{\mathit{vupk}}))\,.$$

- The user then proves to the issuer that he knows the secret value *vusk* corresponding to the message contained in the commitment c_{vupk}, and the issuer aborts if this proof fails. This check guarantees the issuer that the legitimate owner of the credential is trying to initiate the re-issuance protocol.
- Finally, the user and the issuer run the extended issuance protocol explained above, using the $(c_i)_{i \in B}$ and c_{vupk} as commitments for carry-over attributes. The set R of revealed attributes depends on the issuer's policy.

Note that as for the basic construction in § 8.2, the above construction does not disable the user from duplicating his credentials using the backup procedure. However, this problem can again be tackled by the extension described in § 8.2.1.

8.3.1 Protecting against Malicious Issuers

The construction given above disables an adversary from obtaining new credentials from an honest issuer on behalf of the honest user. However, it does not offer protection if the issuer colludes with the adversary: in this case, *cid* can no longer be guaranteed to be uniformly random as required in the construction. Thus, the adversary could re-use the *cid* of the stolen credential also for a new credential, and then present the resulting credential successfully as he can simply re-use the signature σ on *cid*. This problem can partially be solved by using the credential identifiers as additional revocation handles: if the user now revokes the *cid* of the stolen credential, also all credentials re-using this *cid* will be revoked.
However, it is even possible to fully exclude this attack by modifying the above construction as follows:

- The credential identifier *cid* and all related commitments and signatures are removed from the construction.
- Instead of signing *cid*, the user computes a signature σ on $((a_i)_{i=1}^n, \mathit{vupk}, \mathit{ipk})$, i.e., he signs all the attributes certified by the credential including the vault user public key, as well as the issuer's public key.
- Upon presentation, the user modifies the proof π to show that σ verifies correctly under *vupk*, where all unrevealed attributes of the signed tuple are kept secret for this proof.

By signing all the users attributes in one signature, the adversary would now have to carry over *all* of them into a new credential. By furthermore signing the issuer public key, the only possibility for an adversary would be to request a new credential with the identical attributes from the same issuer as the original attack; this, however, does not give him any more power than directly using the stolen credential.

8.3.2 Efficient Instantiation for Weak Privacy

In contrast to the constructions presented in § 8.1 and § 8.2, it is not straightforward to efficiently instantiate the above construction, as one needs to prove knowledge of a signature σ on a message m with respect to a verification key *svk*, in general without revealing any of these values.
However, if the underlying credential system is only weakly private, i.e., presentation tokens can be linked to each other anyways, the above construction can easily be modified and instantiated, as now $\mathit{svk} = \mathit{vupk}$ may be given to the verifier without giving an adversary any further power to link

presentation tokens. If the used signature scheme allows the user to efficiently prove knowledge of a secret signature on a secret message for a *public* verification key, as is the case for CL-signatures [CL02], the proof π required for the successful presentation of the credential can be efficiently computed. Furthermore, the requirement of the signature verification key lying in the attribute space of the credential system can be dropped, and one may just embed $\mathtt{H}(\mathit{vupk})$ into the credential instead of *vupk* for a cryptographic hash function $\mathtt{H}$. Then, at presentation, also the value of $\mathtt{H}(\mathit{vupk})$ can be opened, and the verifier checks that this hash value is indeed consistent with *vupk*.

9 Conclusion

We presented three constructions that allow users to backup their anonymous credentials and re-obtain new credentials, even if they lost access to some of the attributes certified by the credential. The first two constructions come at (virtually) no computational overhead and can easily be deployed in the real world. The last construction is computationally more expensive, but additionally protects a user that does not realize that his credentials were leaked to the adversary to the maximum extent possible. However, if only aiming for weak privacy, it can still be realized at reasonable costs.
Future work will include finding formal security proofs for all three constructions in appropriate security models, and finding practically efficient instantiations for the last construction for strong privacy.

10 References

[BCLN12] F. Baldimtsi, J. Camenisch, A. Lehmann, and G. Neven. Anonymous Credential Backup. Technical report, 2012.

[BG92] M. Bellare and O. Goldreich. On Defining Proofs of Knowledge. In E. F. Brickell, editor, *CRYPTO 92*, volume 740 of *LNCS*, pages 390–420. Springer, 1992.

[Bra99] S. Brands. *Rethinking Public Key Infrastructure and Digital Certificates – Building in Privacy*. PhD thesis, Eindhoven Institute of Technology, 1999.

[CH02] J. Camenisch and E. Van Herreweghen. Design and Implementation of the *idemix* Anonymous Credential System. In V. Atluri, editor, *ACM CCS 02*, pages 21–30. ACM, 2002.

[Cha81] D. Chaum. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. *Commun. ACM*, 24(2):84–88, 1981.

[Cha85] D. Chaum. Security Without Identification: Transaction Systems to Make Big Brother Obsolete. *Commun. ACM*, 28(10):1030–1044, 1985.

[CKL$^+$14] J. Camenisch, S. Krenn, A. Lehmann, G. L. Mikkelsen, G. Neven, and M. Ø. Pedersen. Formal Treatment of Privacy-Enhancing Credential Systems. Cryptology ePrint Archive, Report 2014/708, 2014. http://eprint.iacr.org/.

[CKLN14] J. Camenisch, S. Krenn, A. Lehmann, and G. Neven. A Method to Backup and Revoke Authentication Credentials. Technical report, 2014.

[CKY09] J. Camenisch, A. Kiayias, and M. Yung. On the Portability of Generalized Schnorr Proofs. In A. Joux, editor, *EUROCRYPT 09*, volume 5479 of *LNCS*, pages 425–442. Springer, 2009.

[CL01] J. Camenisch and A. Lysyanskaya. An Efficient System for Non-transferable Anonymous Credentials with Optional Anonymity Revocation. In B. Pfitzmann, editor, *EUROCRYPT 01*, volume 2045 of *LNCS*, pages 93–118. Springer, 2001.

[CL02] J. Camenisch and A. Lysyanskaya. A Signature Scheme with Efficient Protocols. In S. Cimato, C. Galdi, and G. Persiano, editors, *SCN 02*, volume 2576 of *LNCS*, pages 268–289. Springer, 2002.

[CL04] J. Camenisch and A. Lysyanskaya. Signature Schemes and Anonymous Credentials from Bilinear Maps. In M. K. Franklin, editor, *CRYPTO 04*, volume 3152 of *LNCS*, pages 56–72. Springer, 2004.

[CLN12] J. Camenisch, A. Lehmann, and G. Neven. Method to Backup Authentication Credentials Embedded in Tamperproof Devices. Technical report, 2012.

[DF02] I. Damgård and E. Fujisaki. A Statistically-Hiding Integer Commitment Scheme Based on Groups with Hidden Order. In Y. Zheng, editor, *ASIACRYPT 02*, volume 2501 of *LNCS*, pages 125–142. Springer, 2002.

[FO97] E. Fujisaki and T. Okamoto. Statistical Zero Knowledge Protocols to Prove Modular Polynomial Relations. In B. S. Kaliski Jr., editor, *CRYPTO 97*, volume 1294 of *LNCS*, pages 16–30. Springer, 1997.

[GMW91] O. Goldreich, S. Micali, and A. Wigderson. Proofs that Yield Nothing But Their Validity or: All Languages in NP Have Zero-Knowledge Proof Systems. *J. ACM*, 38(3):691–729, 1991.

[Lip03] H. Lipmaa. On Diophantine Complexity and Statistical Zero Knowledge Arguments. In C.-S. Laih, editor, *ASIACRYPT 03*, volume 2894 of *LNCS*, pages 398–415. Springer, 2003.

[Ped91] T. P. Pedersen. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In J. Feigenbaum, editor, *CRYPTO 91*, volume 576 of *LNCS*, pages 129–140. Springer, 1991.

[PZ13] C. Paquin and G. Zaverucha. U-prove Cryptographic Specification v1.1 (Revision 2). Technical report, Microsoft Corporation, April 2013.

[Sch91] C. Schnorr. Efficient Signature Generation by Smart Cards. *Journal of Cryptology*, 4(3):161–174, 1991.