Delegatable Anonymous Credentials

Mira Belenkiy[1], Jan Camenisch[2], Melissa Chase[3], Markulf Kohlweiss[4],
Anna Lysyanskaya[5], and Hovav Shacham[6]

[1] Microsoft, mibelenk@microsoft.com
[2] IBM Zurich Research Laboratory, jca@zurich.ibm.com
[3] Microsoft Research, melissac@microsoft.com
[4] K.U. Leuven, mkohlwei@esat.kuleuven.be
[5] Brown University, anna@cs.brown.edu
[6] UC San Diego, hovav@cs.ucsd.edu

Abstract. We construct an efficient delegatable anonymous credential system. Users can anonymously and unlinkably obtain credentials from any authority, delegate their credentials to other users, and prove possession of a credential L levels away from the given authority. The size of the proof (and time to compute it) is $O(Lk)$, where k is the security parameter. The only other construction of delegatable anonymous credentials (Chase and Lysyanskaya, Crypto 2006) relies on general non-interactive proofs for NP-complete languages of size $k\Omega(2^L)$.

We revise the entire approach to constructing anonymous credentials and identify *randomizable* zero-knowledge proof of knowledge systems as the key building block. We formally define the notion of randomizable non-interactive zero-knowledge proofs, and give the first construction by showing how to appropriately rerandomize Groth and Sahai (Eurocrypt 2008) proofs. We show that such proof systems, in combination with an appropriate authentication scheme and a few other protocols, allow us to construct delegatable anonymous credentials. Finally, we instantiate these building blocks under appropriate assumptions about groups with bilinear maps.

1 Introduction

Access control is one of the most common problems in security. We frequently need to answer the question: does the person requesting access to a resource possess the required credentials? A credential typically consists of a certification chain rooted at some authority responsible for managing access to the resource and ending at the public key of a user in question. The user presents the credential and demonstrates that he knows the corresponding secret key. Sometimes, the trusted authority issues certificates *directly* to each user (so the length of each certification chain is 1). More often, the authority *delegates* responsibility. A system administrator allows several webmasters to use his server. A webmaster can create several forums, with different moderators for each forum. Moderators approve some messages, reject others, and even give favored users unlimited posting privileges. Imagine a world where a single system administrator has to approve every single message posted on every single forum!

We want cryptographic credentials to follow the same delegation model as access control follows in the real world. Our system administrator issues credentials of length 1 to his trusted webmasters. Each webmaster can *extend* the credential chain by delegating a credential of length 2 to a moderator. In general, a user with a credential of length L can issue a credential of length $L + 1$.

A conventional signature scheme immediately allows for (non-anonymous) delegatable credentials: Alice, who has a public signing key pk_A and a certification chain of length L, can sign Bob's public key pk_B, giving Bob a certification chain of length $L + 1$.

The design of an *anonymous* delegatable credential scheme in which participants can obtain, delegate and demonstrate possession of credential chains without revealing any additional information about themselves, is a natural and desirable goal. The solution has, until now, proven elusive. Our main contribution is the first fully delegatable anonymous credential scheme. The only known construction of delegatable anonymous credentials, due to Chase and Lysyanskaya [CL06], needs $k^{\Omega(L)}$ space to store a

certification chain of length L (for security parameter k), and therefore could not tolerate non-constant L. Our solution is *practical*: all operations on chains of length L need $\Theta(kL)$ time and space.[7]

Why this is a challenging problem. There is no straightforward transformation of anonymous credential schemes without delegation [Cha85,Dam90,Bra99,LRSW99,CL01,CL04,BCKL08] into delegatable schemes. Prior work on anonymous credentials [Lys02,CL02,CL04,BCKL08] used signature schemes that lend themselves to the design of efficient protocols for (1) obtaining a signature on a committed value; and (2) proving that a committed value has been signed. Suppose Oliver wants to issue a credential to Alice. They would run protocol (1) so that Alice obtained a signature on her secret key. Whenever Alice wants to show her credential, she would give the verifier a fresh commitment to her secret key and use protocol (2) to prove that the committed key has been signed.

The problem with this approach is that Alice gets Oliver's signature, and the signature might reveal his identity. Generalizing to credential chains, Alice would learn the identity of all intermediate signers. Thus, the old approach does not yield itself to delegation and we must try something very different.

Our approach. Our insight is that instead of giving Alice his signature, Oliver gives Alice a non-interactive proof-of-knowledge of the signature. The trick is to find a proof-system that would then let Alice (1) delegate the credential by extending the proof and (2) rerandomize the proof every time she shows (or extends it) to preserve her anonymity.

Let's say Oliver is a credential authority with public key pk_O and secret key sk_O; and let's say Alice is a user with secret key sk_A. Alice wants to obtain the credential directly from Oliver (so her certification chain will be of length 1). Under the old approach, they would run a secure two-party protocol as a result of which Alice obtains a signature $\sigma_{pk_O}(sk_A)$ on sk_A, while Oliver gets no output. Under the new approach, Alice's output is (C_A, π_A), where C_A is a commitment to her secret key sk_A, and π_A is a *proof of knowledge* of Oliver authenticating the contents of C_A. Note that a *symmetric* authentication scheme is sufficient because *no one ever sees the authenticator*; all verification is done on the proof of knowledge. The symmetric key sk_O remains secret to Oliver; we create a "public" key C_O that is simply a commitment to sk_O.

How can Alice use this credential anonymously? If the underlying proof system is *malleable* in just the right way, then given (C_A, π_A) and the opening to C_A, Alice can compute (C'_A, π'_A) such that C'_A is another commitment to her sk_A that she can successfully open, while π'_A is a proof of knowledge of Oliver authenticating the contents of C'_A. Malleability is usually considered a bug rather than a feature. But in combination with the correct extraction properties, we still manage to guarantee that these randomizable proofs give us a useful building block for the construction. The bottom line is that (C'_A, π'_A) should not be linkable to (C_A, π_A), and also it should not be possible to obtain such a tuple without Oliver's assistance.

How does Alice delegate her credential to Bob? Alice and Bob can run a secure protocol as a result of which Bob obtains (C_B, π_B) where C_B is a commitment to Bob's secret key sk_B and π_B is a proof of knowledge of an authenticator issued by the owner of C'_A on the contents of C_B. Now, essentially, the set of values $(C'_A, C_B, \pi'_A, \pi_B)$ together indicate that the owner of C'_A got a credential from Oliver and delegated to the owner of C_B, and so it constitutes a proof of possession of a certification chain. Moreover, it hides the identity of the delegator Alice! Now Bob can, in turn, use the randomization properties of the underlying proof system to randomize this set of values so that it becomes unlinkable to his original pseudonym C_B; he can also, in turn, delegate to Carol.

It may be somewhat counter-intuitive that adversaries cannot take advantage of the malleable proofs to forge proofs of possessing the certification chain. The explanation is that we use a perfectly *extractable* proof system. This lets us extract a certification chain from any proof, no matter how randomized. As

[7] Barak [Bar01] introduced a somewhat related notion of delegatable signatures which allows delegation without an exponential growth. However, his construction relies on general NIZK and CS proofs, and thus is purely theoretical, and it is not clear how to use such signatures to build a delegatable credential system that would satisfy our definitions.

a result, we can use an adversary that fakes a proof to attack the unforgeability properties of the underlying authentication scheme. Our solution also addresses some important details such as (1) how to make it impossible for adversarial users to mix and match pieces of different certification chains to create unauthorized certification chains; (2) how to define and construct an authentication scheme that remains secure even when an adversary can ask an honest user to authenticate chosen messages *and* obtain authentication tokens on the user's secret key. (i.e. the adversary gets a signature on the user's secret key, but does not learn the secret key itself).

Attributes. In many contexts, we want the ability to express *why* a credential has been delegated. For example, our system administrator may want to give each webmaster access only to his own website. Webmasters may want to give some moderators the right to add new users to the forum, while other moderators would only have the right to approve/reject messages.

Why anonymity? The system administrator clearly needs to hold webmasters accountable if they crash the server. However, moderators and forum users may want to be able to act anonymously in order to foster a free and lively exchange of ideas.

Our delegatable anonymous credential system lets users add human-readable attributes to each credential. Oliver can give Alice a level 1 credential with attribute "webmaster of Crypto Forum." Alice can then delegate her credential to Bob with attribute "moderator of Crypto Forum." As a result, Bob can log on to the server anonymously and prove that the "webmaster of the Crypto Forum" made him the "moderator of the Crypto Forum." Our construction lets users add as many attributes as they want to each credential, allowing for the flexibility and expressibility that we see in modern (non-anonymous) access control systems. In some cases, the issuer may want to explicitly add the user's identity to the attribute to create accountability. Or, he might want to include an expiration date for the credential, or other conditions controlling it's use.

Our contribution. We (1) define and construct extractable and randomizable proofs of knowledge of a witness to a certain class of relations based on the recent proof system for pairing product equations due to Groth and Sahai [GS08]; and (2) define and construct a delegatable anonymous credential system based on this and other building blocks. Our construction is efficient whenever the building blocks can be realized efficiently, and we give efficient instantiations of the building blocks under appropriate assumptions about bilinear groups. (We introduce two new assumptions, but we also justify these assumptions by providing proofs in the generic group model.)

Organization. In Section 2 we define a delegatable anonymous credential system. In Section 3 we define an appropriate authentication system and the appropriate zero-knowledge proof of knowledge system. We show how these building blocks can be instantiated under appropriate assumptions about groups with bilinear maps. In Section 4 we give a construction of anonymous delegatable credentials based on these building blocks.

2 Definition of Delegatable Credentials

An anonymous delegatable credential system has only one type of participant: users. Any user O can become an originator of a credential; all he needs to do is publish one of his pseudonyms Nym_O as its public key to act as a credential authority.[8] Each user will have one secret key, but many different pseudonyms corresponding to that secret key. Thus a user A with secret key sk_A can be known to authority O as $Nym_A^{(O)}$ and to user B as $Nym_A^{(B)}$. If authority O issues user A a credential for $Nym_A^{(O)}$, then user A can prove to user B that $Nym_A^{(B)}$ has a credential from authority O. We say that credentials received directly from the authority are level 1 credentials or basic credentials, credentials that have been delegated once are level 2 credentials, and so on. Thus user A can also delegate its credential to user B, and user B can

[8] We assume some kind of PKI for storing the authorities' public keys, but this is outside the scope of this paper.

then prove that he has a level 2 credential from authority O. A delegatable credential system consists of the following algorithms:

Setup(1^k) outputs the trusted public parameters of the system, $params_{DC}$.

Keygen($params_{DC}$) creates the secret key of a party in the system.

Nymgen($params_{DC}, sk$). On each run, the algorithm outputs a new pseudonym Nym and auxiliary info $aux(Nym)$ for secret key sk.[9]

Issue($params_{DC}, Nym_O, sk_I, Nym_I, aux(Nym_I), cred, Nym_U, L$)
$\leftrightarrow$ Obtain($params_{DC}, Nym_O, sk_U, Nym_U, aux(Nym_U), Nym_I, L$) are the interactive algorithms that let a user I issue a level L credential to a user U. The pseudonym Nym_O is the authority's public key, sk_I is the issuers's secret key, Nym_I is the issuer's pseudonym with auxiliary information $aux(Nym_I)$, $cred$ is the issuer's level L credential rooted at Nym_O, sk_U is the user's secret key, and Nym_U is the user's pseudonym with auxiliary information $aux(Nym_U)$. If $Nym_I = Nym_O$, then the issuer is the authority responsible for this credential, so $L = 0$ and $cred = \epsilon$. The issuer runs Issue with these inputs and gets no output, and he user runs Obtain and gets a credential $cred_U$.

CredProve($params_{DC}, Nym_O, cred, sk, Nym, aux(Nym), L$)**.** Takes as input a level L credential $cred$ from authority Nym_O, outputs a value $credproof$.

CredVerify($params_{DC}, Nym_O, credproof, Nym, L$)**.** Outputs accept if $credproof$ is a valid proof that the owner of pseudonym Nym possesses a level L credential with root Nym_O and reject otherwise.

We say that a function $\nu : \mathbb{Z} \rightarrow \mathbb{R}$ is negligible if, for all integers c, there exists an integer K such that $\forall k > K, |\nu(k)| < 1/k^c$. We use the standard GMR [GMR88] notation to describe probability spaces.

We formally define a secure delegatable credential system in the full version.Intuitively, the algorithms Setup, Keygen, Nymgen, NymProve, NymVerify, VerifyAux, Issue, Obtain, CredProve, and CredVerify constitute a secure anonymous delegatable credential scheme if the following properties hold:

Correctness. We say that a credential $cred$ is a proper credential, if for all of the user's pseudonyms, CredProve always creates a proof that CredVerify accepts.The delegatable credential system is correct if an honest user and an honest issuer can run Obtain $\leftrightarrow$ Issue and the honest user gets a proper credential. (The correctness property requires that Issue and CredProve check that their inputs are valid and that the credentials they delegate or prove are proper. This is necessary to avoid selective failure attacks in which users are given credentials that work for some identities/pseudonyms, but not all.)

Anonymity Intuitively, anonymity requires that the adversary's interactions with the honest parties in the real game should be indistinguishable from some ideal game in which pseudonyms, credentials and proofs are truly anonymous.

Specifically, there has to exists a simulator (SimSetup, SimProve, SimObtain, SimIssue) such that SimSetup produces parameters indistinguishable from those output by Setup, along with some simulation trapdoor sim. Under these parameters we require that the following properties hold when SimProve, SimObtain, SimIssue and even the distinguishers are given sim:

- When generated using the parameters output by SimSetup, Nym is distributed independently of sk.
- No distinguisher can tell if it is interacting with Issue run by an honest party with a valid credential, or with SimIssue which is not given the credential and the issuer's secret key, but only told the authority, length of the credential chain, and the pseudonyms of the issuer and user.
- No distinguisher can tell if it is interacting with Obtain run by an honest party with secret sk, or with SimObtain that is only given the authority, length of the credential chain, and the pseudonyms of the issuer and user (but not sk).

[9] We assume the existence of a predicate VerifyAux that accepts iff Nym is a valid pseudonym for $(sk, aux(Nym))$ and of a protocol NymProve $\leftrightarrow$ NymVerify that is a zero-knowledge proof of knowledge of $sk, aux(Nym)$ such that VerifyAux($params_{DC}, Nym, sk, aux(Nym)$) = accept.

– The simulator SimProve can output a fake credential proof $credproof$ that cannot be distinguished from a real credential proof, even when SimProve is only told the authority, length of the credential chain, and the pseudonym of the user (it is not given the user's secret key, or his private credentials).

Remark 1. Our definition, somewhat in the spirit of the composable zero-knowledge definition given in [GS08], requires that each individual protocol (SimIssue, SimObtain, SimProve), when run on a single adversarially chosen input produces output indistinguishable from the corresponding real protocol, even when the adversary is given the *simulation trapdoor*. A simple hybrid argument shows that this implies the more complex but weaker definition in which the adversary only controls the public inputs to the algorithm. This stronger definition is much easier to work with as we need only consider one protocol at a time, and only a single execution of each protocol.

Unforgeability. To define unforgeability we will have all of the honest parties controlled by a single oracle, and we will keep track of all honestly issued credentials. Then we will require that an adversary given access to this oracle should have only negligible probability of outputting a forged credential.

For unforgeability of credentials to make sense, we have to define it in a setting where pseudonyms are completely binding, i.e. for each pseudonym there is exactly one valid corresponding secret key. Thus, there must exist some (potentially exponential-time) extraction algorithm which takes as input a pseudonym and outputs the corresponding secret key. A forgery of a level L credential occurs when the adversary can "prove" that Nym has a level L credential when such a credential was never issued to any pseudonym owned by $sk_L = \mathsf{Extract}(Nym)$. Our definition is slightly stronger, in that we require an *efficient* algorithm Extract that produces $F(sk_L)$ for some bijection F, and so we get F-extraction.

Suppose we can extract an entire chain of secret keys from the credential proof. Then we can say a forgery occurs when the adversary produces a proof of a level L credential with authority O from which we extract $sk_1, \ldots, sk_{L-1}, sk_L$ such that a level L credential rooted at O was never delegated by sk_{L-1} to sk_L. Thus, we are not concerned with exactly which set $sk_2, \ldots, sk_{L-2}$ are extracted. In practical terms, this means that once sk_{L-1} has delegated a level L credential from authority O to sk_L, we don't care if the adversary can forge credentials with different credential chains as long as they have the same level, are from the same authority, and are for the same sk_L. (Obviously, we can also consider functions of the secret keys $F(sk_i)$ in this discussion).

Of course, this only makes sense if sk_{L-1} belongs to an honest user; otherwise we have no way of knowing what credentials he issued. But what if the owner of sk_{L-1} is adversarial and the owner sk_{L-2} is honest? Then the owner of sk_L should be able to prove possession of a credential if and only if sk_{L-2} delegated a level $L-1$ credential rooted at authority O to user sk_{L-1}. Generalizing this idea, our definition says a forgery is successful if we extract $sk_0, \ldots, sk_L$ such that there is a prefix $sk_O, \ldots, sk_i$ where sk_{i-1} is honest, but sk_{i-1} never issued a level i credential from root O to sk_i.

Let F be an efficiently computable bijection and a one-way function. There exists a PPT ExtSetup, and a deterministic Extract with five properties:

– The parameters output by ExtSetup are distributed identically to those output by Setup.
– Under these parameters, pseudonyms are perfectly binding, i.e. for any Nym, there exists at most one sk for which there exists $aux(Nym)$ such that VerifyAux accepts.
– Given an honestly generated level L credential proof, Extract can always extract the correct chain of L identities. I.e. if the credential is formed by using sk_0 to delegate to sk_1 which delegates to sk_2 and so on until sk_L, Extract will produce $(f_0, \ldots, f_L) = (F(sk_0), \ldots, F(sk_L))$. Note that this must hold for level 0 as well: for any valid pseudonym Nym_O, Extract will produce $f_0 = F(sk_O)$ where sk_O corresponds to Nym_O.
– Given an adversarially generated level L credential proof $credproof$ from authority Nym_O for the pseudonym Nym, Extract will always produce either the special symbol $\perp$ or $f_0, \ldots f_L$ such that Nym_O is a pseudonym for $F^{-1}(f_0)$ and Nym is a pseudonym for $F^{-1}(f_L)$.

– No adversary can output a valid credential proof from which an unauthorized chain of identities is extracted:

$$\Pr[(params_{DC}, td) \leftarrow \mathsf{ExtSetup}(1^k);$$
$$(credproof, Nym, Nym_O, L), \leftarrow \mathcal{A}^{\mathcal{O}(params_{DC},\cdot,\cdot)}(params_{DC}, td);$$
$$(f_0, \ldots, f_L) \leftarrow \mathsf{Extract}(params_{DC}, td, credproof, Nym, Nym_O, L) :$$
$$\mathsf{CredVerify}(params_{DC}, Nym_O, credproof, Nym, L) = \mathsf{accept} \wedge$$
$$(\exists i \text{ such that } (f_0, i, f_{i-1}, f_i) \notin \mathsf{ValidCredentialChains} \wedge f_{i-1} \in \mathsf{HonestUsers})] \leq \nu(k)$$

where $\mathcal{O}(params_{DC}, command, input)$ describes all possible ways for the adversary $\mathcal{A}$ to interact with the delegatable credentials system: $\mathcal{A}$ can ask the oracle to add new honest users; the oracle generates $sk \leftarrow \mathsf{Keygen}(params_{DC})$, stores it in the list $\mathsf{HonestUsers}$, and returns $F(sk)$ as the handle.[10] $\mathcal{A}$ can ask for new pseudonyms for existing honest users, referenced by $F(sk)$, and he can provide a credential and ask an honest user to generate the corresponding proof. Finally, he can run the $\mathsf{Issue} \leftrightarrow \mathsf{Obtain}$ protocols on credentials of its choice, either between themselves, or with an adversarial issuer or obtainer. In this case, we need to keep track of which credentials are being issued, so that we will be able to identify a forgery. To do this, we use the $\mathsf{Extract}$ algorithm to extract and store the chain of identities behind each credential being issued on the list $\mathsf{ValidCredentialChains}$. For details, see the full version.

Remark 2. We let the adversary track honest users' credentials and pseudonyms (but, of course, not their secret keys). Our definition is strictly stronger than one that uses a general oracle that does not reveal the credentials of honest users to the adversary. This approach results in simpler definition and analysis.

3 Building Blocks

In sections 3.1 and 3.3 we define two new concepts: certification-secure authentication schemes and randomizable proofs. In Sections 3.2 and 3.4 we recap existing work about non-interactive proofs and define important notation. Finally, in section 3.5, we give new efficient protocols for randomizable proofs of an authenticator. As part of one of these protocols, we give an efficient two party protocol for generating a Boneh-Boyen signature[BB04] on a committed message, which may be of independent interest.

Bilinear Maps and Assumptions We use standard notation for groups with a computable bilinear map $e : G_1 \times G_2 \rightarrow G_T$. See, e.g., [BLS04,GPS06]. The security of our scheme is based on strengthened versions of the SDH [BB04] and CDH assumptions. See full version for a more detailed discussion.

Definition 1 (HSDH [BW07]). *On input $g, g^x, u \in G_1$, $h, h^x \in G_2$ and $\{g^{1/(x+c_\ell)}, h^{c_\ell}, u^{c_\ell}\}_{\ell=1\ldots q}$, it is computationally infeasible to output a new tuple $(g^{1/(x+c)}, h^c, u^c)$.*

Definition 2 (BB-HSDH). *Let $c_1 \ldots c_q \leftarrow Z_p$. On input $g, g^x, u \in G_1$, $h, h^x \in G_2$ and the tuple $\{g^{1/(x+c_\ell)}, c_\ell\}_{\ell=1\ldots q}$, it is computationally infeasible to output a new tuple $(g^{1/(x+c)}, h^c, u^c)$.*

Definition 3 (BB-CDH). *On input $g, g^x, g^y \in G_1$, $h, h^x \in G_2$, $c_1, \ldots, c_q \leftarrow Z_q$ and $g^{\frac{1}{x+c_1}}, \ldots, g^{\frac{1}{x+c_q}}$, it is computationally infeasible to output a g^{xy}.*

We give a generic group proof for BB-HSDH in the full version.It is easy to see that the BB-HSDH implies HSDH, thus our proof also establishes generic group security for HSDH. We show that SDH implies BB-CDH (see full version). Since SDH is secure in the generic group model, so is BB-CDH.

Groth-Sahai proofs use either the DLIN or SXDH assumption [GS08]. (See full version.)

[10] Since we are assuming that the adversary is given the extraction trapdoor td, he will be able to extract $F(sk)$ from any pseudonym for this user. Note that $F(sk)$ must not reveal sk.

3.1 F-Unforgeable Certification Secure Message Authentication Scheme

A message authentication scheme is similar to a signature scheme. However, there is no public-key, and a user uses the *secret key* to verify that a message has been authenticated. We need an authentication scheme for a vector of messages $\boldsymbol{m}$. This is reminiscent of signatures on blocks of messages [CL02]. In the common parameters model such a scheme consists of four protocols: $\mathsf{AuthSetup}(1^k)$ outputs common parameters $params_A$. $\mathsf{AuthKg}(params_A)$ outputs a secret key sk. $\mathsf{Auth}(params_A, sk, \boldsymbol{m})$ outputs an authentication tag $auth$ that authenticates a vector of messages $\boldsymbol{m}$. $\mathsf{VerifyAuth}(params_A, sk, \boldsymbol{m}, auth)$ accepts if $auth$ is a proper authenticator for $\boldsymbol{m}$ under key sk.

Security. Just like a signature scheme, an authentication scheme must be complete and unforgeable. However, we need to strengthen the unforgeability property to fit our application: *F-Unforgeability* introduced by Belenkiy et al. [BCKL08] requires that for some well defined bijection F, no adversary can output $(F(\boldsymbol{m}), auth)$ without first getting an authenticator on m. We need this stronger definition because in our construction of delegatable credentials, we are only able to extract a function of the message. We write $F(\boldsymbol{m}) = F(m_1, \ldots, m_n)$ to denote $(F(m_1), \ldots, F(m_n))$. In addition, we require the authentication scheme to be unforgeable even if the adversary learns a signature on the challenge secret-key. This is because an adversary in a delegatable credentials system might give a user a credential – i.e., sign the user's secret-key. We call this *certification security*.

An authentication scheme is F-unforgeable and certification secure if for all ppt. adversaries $\mathcal{A}$:

$$\begin{aligned}\Pr[&params_A \leftarrow \mathsf{AuthSetup}(1^k); sk \leftarrow \mathsf{AuthKg}(params_A);\\ &(\boldsymbol{y}, auth) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Auth}}(params_A, sk, .), \mathcal{O}_{\mathsf{Certify}}(params_A, ., (sk, ., \ldots))}(params_A, F(sk)) :\\ &\mathsf{VerifyAuth}(params_A, sk, F^{-1}(\boldsymbol{y}), auth) = 1 \wedge F^{-1}(\boldsymbol{y}) \notin Q_{\mathsf{Auth}}] \leq \nu(k)\ ,\end{aligned}$$

where the oracle $\mathcal{O}_{\mathsf{Auth}}(params_A, sk, \boldsymbol{m})$ outputs $\mathsf{Auth}(params_A, sk, \boldsymbol{m})$ and stores $\boldsymbol{m}$ on Q_{Auth}, and oracle $\mathcal{O}_{\mathsf{Certify}}(params_A, sk^*, (sk, m_2, \ldots, m_n))$ outputs $\mathsf{Auth}(params_A, sk^*, (sk, m_2, \ldots, m_n)$.

Construction. Our authentication scheme is similar to the Boneh-Boyen weak signature scheme [BB04], where $\mathsf{Sign}_{sk}(m) = g^{1/(sk+m)}$. Belenkiy et al. showed that the Boneh-Boyen signature scheme is F-unforgeable for the bijection $F(m) = (g^m, u^m)$, and that the Groth-Sahai proof system can be used to prove knowledge of such a signature. However, in the Boneh-Boyen construction certification-security does not hold because $\mathsf{Sign}_{sk}(m) = \mathsf{Sign}_m(sk)$. In addition, we also need to sign a vector of messages.

We use the Boneh-Boyen signature scheme as a building block. For example, to sign message (m_1, m_2), we choose random keys K^*, K_1, K_2 and compute $(\mathsf{Sign}_{sk}(K^*), \mathsf{Sign}_{K^*}(K_1), \mathsf{Sign}_{K^*}(K_2), \mathsf{Sign}_{K_1}(m_1), \mathsf{Sign}_{K_2}(m_2), F(K^*), F(K_1), F(K_2))$. At a high level, this complex structure eliminates any symmetries between $\mathsf{Auth}_{sk}(m)$ and $\mathsf{Auth}_m(sk)$. More formally:

$\mathsf{AuthSetup}(1^k)$ generates groups G_1, G_2, G_T of prime order p (where $|p|$ is proportional to k), bilinear map $e : G_1 \times G_2 \rightarrow G_T$, and group elements $g, u, u^*, u_1, \ldots, u_n \in G_1$ and $h \in G_2$. It outputs $params_A = (G_1, G_2, G_T, e, p, g, u, u^*, u_1, \ldots, u_n, h)$. Note that the element u is only needed to define F; it is not used in creating the authenticator.

$\mathsf{AuthKg}(params_A)$ outputs a random $sk \leftarrow Z_p$.

$\mathsf{Auth}(params_A, sk, (m_1, \ldots m_n))$ chooses random $K^*, K_1, \ldots K_n \leftarrow Z_p$. It outputs

$$auth = (g^{\frac{1}{sk+K^*}}, h^{K^*}, {u^*}^{K^*}, \{g^{\frac{1}{K^*+K_i}}, h^{K_i}, u_i^{K_i}, g^{\frac{1}{K_i+m_i}}\}_{1 \leq i \leq n})\ .$$

$\mathsf{VerifyAuth}(params_A, sk, (m_1, \ldots, m_n), auth)$. Parse $auth = (A^*, B^*, C^*, \{A_i, B_i, C_i, D_i\}_{1 \leq i \leq n})$. Verify that $e(A^*, h^{sk} B^*) = e(g, h)$, and that $e(u^*, B^*) = e(C^*, h)$. For $1 \leq i \leq n$ verify that $e(A_i, B^* B_i) = e(g, h)$, that $e(u_i, B_i) = e(C_i, h)$, and that $e(D_i, B_i h^{m_i}) = e(g, h)$. Accept if and only if all verifications succeed.

Theorem 1. *The message authentication scheme above is F-unforgeable and certification secure for $F(m_i) = (h^{m_i}, u^{m_i})$ under the BB-HSDH and BB-CDH assumptions.*
See full version for proof. The signature scheme obtained by setting $pk = h^{sk}$ may be of independent interest.

3.2 Composable Non-Interactive Proofs

We review composable non-interactive proof systems. Let $R(\cdot,\cdot)$ be any polynomial-time computable relation. A non-interactive proof system for a NP language allows a prover to convince a verifier about the truth of the statement $\exists x : R(s,x)$ using witness x. Non-interactive proof systems use a common reference string $params$ as output by $\mathsf{Setup}(1^k)$ that is common input to both the $\pi \leftarrow \mathsf{Prove}(params, s, x)$ and $\mathsf{accept/reject} \leftarrow \mathsf{VerifyProof}(params, x, \pi)$ algorithms. This notion can be generalized for a relations $R(params, s, x)$ parameterized by $params$.

Informally, zero-knowledge captures the notion that a verifier learns nothing from the proof but the truth of the statement. Witness indistinguishability is a weaker notion that guarantees that the verifier learns nothing about which witness was used in the proof.

In a *composable* (under the definition of Groth and Sahai [GS08]) non-interactive *witness indistinguishable* proof system there exists an algorithm SimSetup that outputs $params$ together with a trapdoor sim, such that (1) $params$ output by SimSetup is indistinguishable from those output by Setup; (2) the output of Prove using these parameters is perfectly witness-indistinguishable (in other words, even if there are two witnesses to a statement, they induce identical distributions on the proofs). Composable non-interactive *zero-knowledge* further means that there exists an algorithm SimProve that outputs a simulated proof using sim and the

output of SimProve is distributed identically to that of Prove when given the simulated parameters. The big advantage of a composable definition is that it is fairly simple and easy to work with, and yet it still implies the standard multi-theorem definitions.

Composable proofs about commitments. The prover and verifier frequently get some set of commitments $(C_1, \ldots, C_n)$ as common input. The prover wants to show that a statement $s = (C_1, \ldots, C_n, \mathsf{Condition})$ holds. The witness to the statement is $(x_1, open_1, \ldots, x_n, open_n, y)$, where $(x_i, open_i)$ is the opening of commitment C_i, while y is some value that has nothing to do with the commitments. The relation is $R = \{(params, s, x) | C_1 = \mathsf{Commit}(params, x_1, open_1) \wedge \ldots \wedge C_n = \mathsf{Commit}(params, x_n, open_n) \wedge \mathsf{Condition}(params, x_1, \ldots, x_n, y)\}$.

We can make two transformations on composeable proofs about commitments. The *concatentation* of two proofs π and π' is a proof $\pi \circ \pi'$ that combines all the commitments and proves the AND of the two conditions. If a proof π proves a condition about a set of commitments $\mathcal{C}$, a *projection* $\pi' = \pi \circ \mathcal{S}$ proves a condition about the contents of the subset $\mathcal{C} \setminus \mathcal{S}$ of commitments. A projected proof π' is obtained by removing the commitments in $\mathcal{S}$ from the statement and appending them to the proof.

3.3 Randomizable non-interactive proofs.

We say that a proof system about commitments is randomizable if there exists an efficient algorithm RandProof that on input $(C_1, \ldots, C_n)$, $(open'_1, \ldots, open'_n)$ and π outputs a randomized proof π' and commitments $(C'_1, \ldots, C'_n)$ such that, informally, the randomized proof π' is indistinguishable from a freshly generated proof with fresh commitments.

More formally, let $\mathsf{Commit}(params_{PK}, \cdot, \cdot) : X \times Y \mapsto \{0,1\}^*$ be a non-interactive commitment scheme. X denotes the input domain of the commitment, and Y denotes the domain for the randomness (both may depend on $params$). We require that Y is an efficiently samplable group with efficiently computable '0' element and '+' and '-' operations. Let Setup, Prove, VerifyProof constitute a composable witness indistinguishable (or zero-knowledge) proof system as defined above.

We consider two experiments. In both experiments we are given $params$ chosen by $\mathsf{SimSetup}(1^k)$, and also $(x_1, \ldots, x_n)$, y, $(open_1, \ldots, open_n)$, $(open'_1, \ldots, open'_n)$, Condition, and π such that (a) the $\mathsf{Condition}(params, x_1, \ldots, x_n, y)$ accepts, and (b) π is a valid proof (accepted by VerifyProof) for the statement $(C_1, \ldots, C_n, \mathsf{Condition})$. Let $C_i = \mathsf{Commit}(params, x_i, open_i)$ and $C'_i = \mathsf{Commit}(params, x_i, open_i + open'_i)$.

1. Experiment: Proof π'_1 is computed by Prove on input the commitments $\{C'_i\}$, their openings $open_i + open'_i$, and the values $(x_1, \ldots, x_n, y)$.
2. Experiment: Proof π'_2 is created by $\mathsf{RandProof}(params, (C_1, \ldots, C_n), (open'_1, \ldots, open'_n), \pi)$.

We say that a proof system is randomizable, if with all but negligible probability over the choice of $params$, for all π, $(x_1, \ldots, x_n)$, y, $(open_1, \ldots, open_n)$, $(open'_1, \ldots, open'_n)$, and Condition that satisfy (a) and (b) above, π'_1 is distributed identically to π'_2 even given π, $(x_1, \ldots, x_n)$, y, $(open_1, \ldots, open_n)$, $(open'_1, \ldots, open'_n)$.

We say that RandProof is a correct rerandomization algorithm if given parameters $params$ chosen by the real Setup, whenever a proof π passes the verification algorithm, then it's rerandomization will pass the verification algorithm. Note that this must hold with high probability by the indistinguishability of the two setups and the above property. However, here we require this to hold with probability 1.

Summary of Groth-Sahai proofs. Groth and Sahai [GS08] give a composeable witness-indistinguishable proof system system that lets us efficiently prove statements in the context of groups with bilinear maps. Let $params_{BM} = (p, G_1, G_2, G_T, e, g, h)$ be the setup for pairing groups of prime order p.

In a Groth-Sahai proof, the prover and the verifier both know values $\{a_q\}_{q=1\ldots Q} \in G_1$, $\{b_q\}_{q=1\ldots Q} \in G_2$, $t \in G_T$, and $\{\alpha_{q,m}\}_{q=1\ldots Q, m=1\ldots M}, \{\beta_{q,n}\}_{q=1\ldots Q, n=1\ldots N} \in Z_p$. In addition, they both know commitments $\{C_m\}_{m=1\ldots M}$ and $\{D_n\}_{n=1\ldots N}$ to values in G_1 and G_2 respectively. For each commitment C_m and D_n the prover knows the opening information and the committed value $x_m \in G_1$ or $y_n \in G_2$ respectively ($m = 1\ldots M$, $n = 1\ldots N$).

Groth-Sahai proofs prove that the values in these commitments fulfill the pairing product equation $\prod_{q=1}^{Q} e(a_q \prod_{m=1}^{M} x_m^{\alpha_{q,m}}, b_q \prod_{n=1}^{N} y_n^{\beta_{q,n}}) = t$.

Let M_1, M_2, and M_T be R-modules for some ring R, and let $E\colon M_1 \times M_2 \to M_T$ be a bilinear map. Also let μ_1, μ_2, μ_T be efficiently computable embeddings that map elements of G_1, G_2, G_T into M_1, M_2, M_T, respectively.[11] The public parameter $params_{PK}$ contain a common reference string with elements $u_1, \ldots, u_I \in M_1$, $v_1, \ldots, v_J \in M_2$ and values $\eta_{h,i,j}$, $1 \le i \le I$, $1 \le j \le J$, and $1 \le h \le H$ as defined by Groth and Sahai [GS08, Section 5].

To commit to an element $x \in G_1$, choose random opening $open = (r_1, \ldots, r_I) \leftarrow R^I$, and compute $C = \mu_1(x) \cdot \prod_{i=1}^{I} u_i^{r_i}$. Elements $y \in G_2$ are committed to in the same way using μ_2 and $v_1, \ldots, v_J \in M_2$, and an opening vector $open \in R^J$. For simplicity we assume that $\mathsf{GSCommit}(params_{PK}, m, open)$ first determines whether $m \in G_1$ or $m \in G_2$ and then follows the appropriate instructions.

Groth and Sahai [GS08, Section 5] show how to efficiently compute proofs $\{\pi_i\}_{i=1}^{I}$, $\{\psi_j\}_{j=1}^{J}$ that prove that the openings of the C_m and D_n satisfy a pairing product equation. To verify such a proof the verifier computes, for all $1 \le q \le Q$, $C_q \leftarrow \mu_1(a_q) \cdot \prod_{m=1}^{M} C_m^{\alpha_{q,m}}$ and $D_q \leftarrow \mu_2(b_q) \cdot \prod_{n=1}^{N} D_n^{\beta_{q,n}}$. Then the verifier checks that $\prod_{q=1}^{Q} E(C_q, D_q) = \mu_T(t) \cdot \prod_{i=1}^{I} E(u_i, \pi_i) \cdot \prod_{j=1}^{J} E(\psi_j, v_j)$.

GS proofs are randomizable. RandProof gets as input commitments $(\hat{C}_1, \ldots, \hat{C}_{\hat{n}})$, $(open'_1, \ldots, open'_{\hat{n}})$, and the proof $[(\pi_1, \ldots, \pi_I, \psi_1, \ldots, \psi_J), \Pi]$. Π contains the internal commitments $C_1, \ldots, C_M$ and

[11] In our applications, $R = Z_q$. However, the randomization techniques also work for Groth-Sahai proofs in composite order groups.

$D_1, \ldots, D_N$. An internal commitment C_m or D_n may or may not correspond to an explicit commitments $\hat{C}_i$.[12] The opening increments $open'_i$ correspond to some $(s_{m,1}, \ldots, s_{m,I})$ or $(z_{n,1}, \ldots, z_{n,J})$.

First it appropriately rerandomizes the commitments C_m and D_n to $C'_m = C_m \cdot \prod_{i=1}^{I} u_i^{s_{m,i}}$ and $D'_n = C_n \cdot \prod_{j=1}^{J} v_j^{z_{n,j}}$, and stores the $s_{m,i}$ and $z_{n,j}$ it used. If an explicit commitment $\hat{C}_{\hat{m}}$ exists it uses $open_{\hat{m}}$; otherwise it assignes fresh random values. Then it computes $\hat{s}_{q,i} = \sum_{m=1}^{M} s_{m,i} \cdot \alpha_{q,m}$, $\hat{z}_{q,j} = \sum_{n=1}^{N} z_{m,j} \cdot \beta_{q,m}$, and $D'_q \leftarrow \mu_2(b_q) \cdot \prod_{n=1}^{N} D_n'^{\beta_{q,n}}$. Next, the prover sets $\pi'_i \leftarrow \pi_i \cdot \prod_{q=1}^{Q} (D'_q)^{\hat{s}_{q,i}}$ and $\psi'_j \leftarrow \psi_j \cdot \prod_{q=1}^{Q} (C_q)^{\hat{z}_{q,j}}$. These π'_i and ψ'_j will satisfy the verification equation for the new commitments.

Now the prover must make a certain technical step to fully randomize the proof. Intuitively, for every set of commitments, there are many proofs $(\pi_1, \ldots, \pi_I, \psi_1, \ldots, \psi_J)$ that can satisfy the verification equation. Given one such proof, we can randomly choose another: The prover chooses $t_{i,j}, t_h \leftarrow R$, and multiplies each $\pi_i := \pi_i \cdot \prod_{j=1}^{J} v_j^{t_{i,j}}$ and each $\psi_j := \psi_j \cdot \prod_{i=1}^{I} u_i^{\sum_{h=1}^{H} t_h \eta_{h,i,j}} \prod_{i=1}^{I} u_i^{t_{i,j}}$. See [GS08, Section 5, Equation 1] for a detailed explanation of this operation.

The algorithm outputs the new proof $[(\pi'_1, \ldots, \pi'_I, \psi'_1, \ldots, \psi'_J), \Pi']$ where Π' contains the internal commitments $C'_1, \ldots, C'_M$ and $D'_1, \ldots, D'_N$.

3.4 Non-interactive Proofs of Knowledge

A NIPK system is a non-interactive proof system with algorithms $\mathsf{PKSetup}$, $\mathsf{PKProve}$, and $\mathsf{PKVerify}$ that correspond to the Setup, Prove, and $\mathsf{VerifyProof}$ we saw in Section 3.2 of a non-interactive proof system. In addition, such a system must be *extractable*.

We recall the notion of *f-extractability* defined by Belenkiy et al. [BCKL08], which is an extension of the original extractability (as defined by De Santis et al [SCP00]). In an extractable proofs system there exists a polynomial-time extractor $(\mathsf{PKExtractSetup}, \mathsf{PKExtract})$. $\mathsf{PKExtractSetup}(1^k)$ outputs $(td, params_{PK})$ where $params_{PK}$ is distributed identically to the output of $\mathsf{PKSetup}(1^k)$. For all PPT adversaries $\mathcal{A}$, the probability that $\mathcal{A}(1^k, params_{PK})$ outputs (s, π) such that $\mathsf{PKVerify}(params_{PK}, s, \pi) = \mathsf{accept}$ and $\mathsf{PKExtract}(td, s, \pi)$ fails to extract a witness x such that $R(params_{PK}, s, x) = \mathsf{accept}$ is negligible in k. We have *perfect* extractability if this probability is 0. f-Extractability means that the extractor $\mathsf{PKExtract}$ only has to output a y such that $\exists x : R(params_{PK}, s, x) = \mathsf{accept} \wedge y = f(params_{PK}, x)$. If $f(params_{PK}, \cdot)$ is the identity function, we get the usual notion of extractability.

NIPK notation We are interested in NIPK about commitments. Let $\mathsf{Commit}(params_{PK}, \cdot, \cdot)$ be an unconditionally binding non-interactive commitment scheme (possibly parameterized by $params_{PK}$) over input domain X that, on input $x \in X$ and a random value $open$, outputs a commitment C. By 'x in C' we denote that there exists $open$ such that $C = \mathsf{Commit}(x, open)$. Following Camenisch and Stadler [CS97] and Belenkiy et al. [BCKL08], we use the following notation to express an f-extractable NIPK of statement $(C_1, \ldots, C_n, \mathsf{Condition})$ with witness $(x_1, open_1, \ldots, x_n, open_n, y)$:

$$\pi \leftarrow \mathsf{NIPK}[x_1 \text{ in } C_1, \ldots, x_n \text{ in } C_n]\{(\ f(params_{PK},\ (x_1, open_1, \ldots, x_n, open_n, y)\ )\ ) : \mathsf{Condition}(params_{PK}, x_1, \ldots, x_n, y)\}.$$

The f-extractability of a NIPK ensures that, with overwhelming probability over the choice of $params_{PK}$, if $\mathsf{PKVerify}$ accepts then we can extract $f(params_{PK}, (x_1, open_1, \ldots, x_n, open_n, y))$ from π, such that x_i is the content of the commitment C_i, and $\mathsf{Condition}(params_{PK}, x_1, \ldots, x_n, y)$ is satisfied.[13]

[12] The GS proof system creates internal commitments while generating the proof; the only difference between explicit and internal commitments is that explicit commitments have names. Verification and randomization treats these commitments identically.

[13] In out construction we will use proofs based on extractable commitments. These proofs let us extract a function $F(x_i)$ of the committed value x_i, but not the opening $open_i$. In this case $f(params_{PK}, (x_1, open_1, \ldots, x_n, open_n, y)) = (F(x_1), \ldots, F(x_n), y)$.

In our notation, $\pi \in \mathsf{NIPK}[\dots]$ means that $\mathsf{PKVerify}$ accepts the proof π for statement $(C_1, \dots, C_n, \mathsf{Condition})$. To further abbreviate notation, we omit $params_{PK}$ and assume that $\mathsf{Condition}$ is clear from the context, and so the sole inputs to $\mathsf{PKVerify}$ are $(C_1, \dots, C_n)$ and π. If the proof is zero-knowledge instead of merely witness indistinguishable, we will write NIZKPK.

Groth-Sahai proofs are f-extractable. The Groth-Sahai proof system generates NIPK proofs of the form $\mathsf{NIPK}_{\mathsf{GS}}[\{x_m \text{ in } C_m\}_{m=1}^{M}, \{y_n \text{ in } D_n\}_{n=1}^{N}]\{(x_1, \dots, x_M, y_1, \dots, y_N) : \prod_{q=1}^{Q} e(a_q \prod_{m=1}^{M} x_m^{\alpha_{q,m}}, b_q \prod_{n=1}^{N} y_n^{\beta_{q,n}}) = t\}$.

3.5 Some additional protocols

We describe three additional protocols which we will need in our construction. First, we introduce a commitment scheme with which we can commit to messages m in the secret key space of the authentication scheme. We show that given an extraction trapdoor, we will be able to extract $F(m)$ (as defined in Section 3.1) from such a commitment.

Next, we describe a protocol for proving knowledge of an authenticator. It is possible that an authenticator may leak information about a user's secret key, which would compromise anonymity. Thus, rather than present an authenticator as part of a credential, the users in our system will generate a zero knowledge proof of knowledge of such an authenticator. We can use the Groth-Sahai composable witness indistinguishable proof system to form this proof.

Finally, we provide a secure two party protocol for jointly computing a proof of knowledge of an authenticator. If Bob wants to prove he has a credential from Alice, he needs to show that he has an authenticator under her secret key on his secret key. However, neither user has enough information to form this authenticator, let alone the corresponding proof. Thus, we also provide a secure two party protocol for jointly computing a proof of knowledge of an authenticator.

Commitment scheme. A commitment $C_x = \mathsf{Commit}(x, open_x)$ to an exponent x consists of two GS commitments to group elements such that $\mathsf{Commit}(x, (o_1, o_2)) = (C^{(1)}, C^{(2)}) = (\mathsf{GSCommit}(h^x, o_1), \mathsf{GSCommit}(u^x, o_2))$ and a NIPK_{GS} proof that these commitments are correctly related. This allows us to extract $F(x) = (h^x, u^x)$.

Proof of knowledge of an authenticator. We need a zero-knowledge proof of knowledge of an unforgeable authenticator for messages $\boldsymbol{m} = (m_1, m_2)$, where the first value is hidden in commitment C_{m_1} and the second values m_2 is publicly known. In our notation, this is:

$$\mathsf{NIZKPK}[sk \text{ in } C_{sk}; m_1 \text{ in } C_{m_1}]\{(F(sk), F(m_1), auth) : \mathsf{VerifyAuth}(params_A, sk, (m_1, m_2), auth) = 1\}.$$

We use Groth-Sahai witness indistinguishable proofs as a building block. We create a concatenation of three proofs: $\pi = \pi_{auth} \circ \pi_{sk} \circ \pi_{m_1}$:

$$\begin{aligned}\pi_{auth} \leftarrow \mathsf{NIPK}_{GS}[&h^{sk} \text{ in } C'^{(1)}_{sk}; u^{sk} \text{ in } C'^{(2)}_{sk}; h^{m_1} \text{ in } C'^{(1)}_{m_1}; u^{m_1} \text{ in } C'^{(2)}_{m_1}] \\ &\{F(sk), F(m_1), A^*, B^*, C^*, \{A_i, B_i, C_i, D_i\}_{1\le i\le 2}) : e(u, h^{sk}) = e(u^{sk}, h) \wedge \\ &e(A^*, h^{sk} B^*) = e(g, h) \wedge e(u^*, B^*) = e(C^*, h) \wedge e(u, h^{m_1}) = e(u^{m_1}, h,) \wedge \\ &\{e(A_i, B^* B_i) = e(g, h) \wedge e(u_i, B_i,) = e(C_i, h) \wedge e(D_i, B_i h^{m_i}) = e(g, h)\}_{1\le i\le 2}\}.\end{aligned}$$

Proofs π_{sk}, π_{m_1} are proofs that two commitments contain the same value. Let $x = sk, m_1$, respectively. Then the two proofs are of the form $\pi_x \leftarrow \mathsf{NIPK}_{GS}[x \text{ in } C_x^{(1)}; x' \text{ in } C'^{(1)}_x]\{(x, x', h^\theta) : e(x/x', h^\theta) = 1 \wedge e(g, h^\theta) = e(g, h)\}$.

Groth and Sahai [GS08] show that witness indistinguishable proofs like π_{sk} and π_{m_1} are also zero knowledge. A simulator that knows the simulation trapdoor sim for the GS proof system can simulate

the two conditions by setting θ to 0 and 1 respectively. In this way he can fake the proofs for arbitrary commitments.

Note that if π_{sk}, π_{m_1} are zero-knowledge, then the composite proof π will be zero knowledge: The simulator first picks sk' and m_1 at random and uses them to generate an authentication tag. It uses the authentication tag as a witness for the witness indistinguishable proof π_{auth} and then fakes the proofs that the commitments C'_{sk}, and C'_{m_1} are to the same values as the original commitments C_{sk} and C_{m_1}.

Two-party protocol for creating a proof of knowledge of an authenticator. The issuer chooses random $K^*, K_1, \ldots K_2 \leftarrow Z_p$. He computes a partial authentication tag $auth' = (g^{\frac{1}{sk+K^*}}, h^{K^*}, u^{*K^*}, \{g^{\frac{1}{K^*+K_1}}, h^{K_i}, u_i^{K_i}\}_{1\leq i\leq 2}, g^{\frac{1}{K_2+m_2}})$. Then the issuer creates π_{sk} and a NIZK$_{GS}$ proof π'_{auth} for this partial authenticator $auth'$.

Let $\mathsf{Keygen}, \mathsf{Enc}, \mathsf{Dec}$ be an additively homomorphic semantically secure encryption scheme, let "$\oplus$" denote the homomorphic operation on ciphertexts; for e a ciphertext and r an integer, $e \otimes r$ denotes "adding" e to itself r times. (For a specific efficient instantiation using the Paillier encryption scheme, see full version). Recall that p is the order of the bilinear groups. The user and the issuer run the following protocol:

1. The issuer generates $(sk_{hom}, pk_{hom}) \leftarrow \mathsf{Keygen}(1^k)$ in such a way that the message space is of size at least $2^k p^2$. He then computes $e_1 = \mathsf{Enc}(pk_{hom}, K_1)$ and sends e_1, pk_{hom} to the user and engages with her in an interactive zero knowledge proof that e_1 encrypts to a message in $[0, p]$.
2. The user chooses $r_1 \leftarrow Z_p$ and $r_2 \leftarrow \{0, \ldots 2^k p\}$, then computes $e_2 = ((e_1 \oplus \mathsf{Enc}(pk_{hom}, m_1)) \otimes r_1) \oplus \mathsf{Enc}(pk_{hom}, r_2 p)$ and sends e_2 to the user.
3. The issuer and the user perform an interactive zero knowledge proof in which the user shows that e_2 has been computed correctly using the message in C_{m_1}, and that r_1, r_2 are in the appropriate ranges.
4. The issuer decrypts $x = \mathsf{Dec}(sk_{hom}, e_2)$, computes $\sigma^* = g^{1/x}$ and sends it to the user.
5. The user computes $\sigma = \sigma^{*r_1}$ and verifies that it is a correct weak BB signature on m_1. The issuer obtains no information about m_1.

Based on $g^{\frac{1}{K_1+m_1}}$ the user computes a proof π''_{auth} to extend the proof of the partial authenticator to a proof of a full authenticator and the proof π_{m_1}. Finally the user computes $\pi = \pi'_{auth} \circ \pi''_{auth} \circ \pi_{sk} \circ \pi_{m_1}$ and randomizes the combined proof.

4 Construction of Delegatable Credentials

We construct delegatable credentials using the tools defined in Section 3. The parameters of the system combine the parameters $params_A$ from the authentication scheme and $params_{PK}$ from the composable and randomizable NIZKPK proof system and its associated commitment scheme Commit. The keyspace of the authenticator must be a subset of the input space of the commitment scheme. Each user U has a secret key $sk_U \leftarrow \mathsf{AuthKg}(params_A)$, and forms his pseudonyms using Commit: $Nym_U = \mathsf{Commit}(sk_U, open_U)$. U can create arbitrarily many different pseudonyms by choosing new random values $open_U$. A user can act as an authority (originator) for some type of a credential by making his pseudonym Nym_O publicly available.

How the proof system fits with the construction. Let sk_0 be the secret key of a credential authority A whose public key is $Nym_O = \mathsf{Commit}(sk_0, open_0)$; and let sk_1 be the secret key of a user U whose pseudonym is $Nym_U = \mathsf{Commit}(sk_1, open_1)$. Let $\mathsf{VerifyAuth}$ be as defined in Section 3.1; let F be a function such that the message authentication scheme is F-unforgeable. Then a level 1 credential from A to U will be

$$\pi_1 \leftarrow \mathsf{NIZKPK}[sk_0 \text{ in } Nym_O, sk_1 \text{ in } Nym_U]\{(F(sk_0), F(sk_1), auth) : \mathsf{VerifyAuth}(params_A, sk_0, (sk_1, r_1), auth)\},$$

where the values r_1 will be defined later. Note that A and U will have to compute π_1 jointly using the protocol defined in section 3.5. While neither can compute $auth$ independently, they jointly have the information to compute $auth$, and therefore, they jointly have the information to compute a proof of knowledge of $auth$.

A level L credential from A to U_L (whose pseudonym is Nym_L) will be

$$\pi_L \leftarrow \mathsf{NIZKPK}[sk_0 \text{ in } Nym_O, sk_L \text{ in } Nym_L]\{(F(sk_0), F(sk_1), \ldots, F(sk_L), auth_1, \ldots, auth_L) : \mathsf{VerifyAuth}(params_A, sk_0, (sk_1, r_1), auth) \wedge \ldots \wedge \mathsf{VerifyAuth}(params_A, sk_{L-1}, (sk_L, r_L)\}.$$

The extractor can extract $F(sk_i)$ from the credential π_L, where sk_i is the secret key of any intermediate delegator U_i, $1 \leq i < L$.

The value π_{L+1} needs to be computable by U_L and U_{L+1}, without contacting anyone else. This is tricky because, in fact, neither of them know any of the values $(sk_0, \ldots, sk_{L-1})$ and $(auth_1, \ldots, auth_L)$! So how can they possibly compute a proof of knowledge of these values?

Our first observation is that, although they do not know any of the secret keys and authentication tags, U_L already has π_L which is distributed correctly. So, in fact, if U_L and U_{L+1} can jointly compute

$$\pi'_{L+1} \leftarrow \mathsf{NIZKPK}[sk_L \text{ in } Nym_L, sk_{L+1} \text{ in } Nym_{L+1}]\{(F(sk_L), F(sk_{L+1}), auth_{L+1}) : \mathsf{VerifyAuth}(params_A, sk_L, (sk_{L+1}, r_{L+1}), auth_{L+1})\}$$

and then compute the concatenated proof $\pi_{L+1} = \pi_L \circ \pi'_{L+1}$, they will in fact arrive at π_{L+1} from which all the right values can be extracted, and whose zero-knowledge properties are preserved as well.

However, this method of computing π_{L+1} is inadequate for our purposes, because π_{L+1} can be linked to π_L, and, as a result, violates the anonymity requirements. This is why it is crucial that the underlying proof system be *randomizable*.

Full construction. We denote a user's private credential as $cred$. To show or delegate the credential, the user randomizes $cred$ to form $credproof$. In our construction, $cred$ is in fact an NIZKPK of a statement about U's specific *secret* pseudonym $S_U = \mathsf{Commit}(sk_U, 0)$ (this specific pseudonym does not in fact hide sk_U since it is formed as a deterministic function of sk_U) while $credproof$ is a statement about a proper pseudonym, $Nym_U = \mathsf{Commit}(sk_U, open)$ for a randomly chosen $open$. So U randomizes $cred$ to obtain $credproof$ using the $\mathsf{RandProof}$ algorithm described in Section 3.

Suppose a user with secret key sk_U has a level L credential from some authority A, and let $(sk_O, sk_1, \ldots, sk_{L-1}, sk_U)$ be the keys such that the owner of sk_i delegated the credential to sk_{i+1} (we let $sk_0 = sk_O$ and $sk_L = sk_U$). A *certification chain* is a list of authenticators $auth_1, \ldots, auth_L$, such that sk_i generated authenticator $auth_{i+1}$ on sk_{i+1}.

To make sure that pieces of different certification chains cannot be mixed and matched, we add a label r_i to each authenticator. The labels have to be unique for each authority and delegation level. Let H be a collision resistant hash function with an appropriate range. For a credential chain rooted at Nym_O, we set $r_i = H(Nym_O, i)$. Each $auth_i$ is an output of $\mathsf{Auth}(params_A, sk_{i-1}, (sk_i, r_i))$. The user U's level L private credential $cred$ is therefore a proof

$$cred \in \mathsf{NIZKPK}[sk_0 \text{ in } Nym_O; sk_L \text{ in } S_U]\{(F(sk_0), \ldots, F(sk_L), auth_1, \ldots, auth_L) : \mathsf{VerifyAuth}(params_A, sk_0, (sk_1, r_1), auth_1) \wedge \ldots \wedge \mathsf{VerifyAuth}(params_A, sk_{L-1}, (sk_L, r_L), auth_L)\} \ .$$

where, F is a bijection such that the authentication scheme is F-unforgeable.

We now give the full construction. Let $\mathsf{PKSetup}, \mathsf{PKProve}, \mathsf{PKVerify}$ be a proof system and let $\mathsf{AuthSetup}, \mathsf{AuthKg}, \mathsf{Auth}, \mathsf{VerifyAuth}$ be an authentication scheme, and let $H : \{0,1\}^* \rightarrow Z_p$ be a hash function.

$\mathsf{Setup}(1^k)$. Use $\mathsf{AuthSetup}(1^k)$ to generate $params_A$ and $\mathsf{PKSetup}(1^k)$ to generate $params_{PK}$; choose the hash function H (as explained above); and output $params_{DC} = (params_A, params_{PK}, H)$.

$\mathsf{Keygen}(params_{DC})$. Run $\mathsf{AuthKg}(params_A)$ and output the secret key sk.

$\mathsf{Nymgen}(params_{DC}, sk)$. Choose a random $open \in Y$ (where Y is the domain of openings of the commitment scheme Commit associated with $params_{PK}$, as explained in Section 3). Compute $Nym = \mathsf{Commit}(params_{PK}, sk, open)$ and output pseudonym Nym and auxiliary information $open$.

$\mathsf{CredProve}(params_{DC}, Nym_O, cred, sk_U, Nym_U, open_U, L)$. Recall that $cred$ should be an NIZKPK of a certification chain (above, we already gave the NIZKPK formula for it with the correct Condition and extraction function f). If $\mathsf{PKVerify}(params_{PK}, (Nym_O, \mathsf{Commit}(sk_U, 0)), cred)$ rejects, or if $Nym_U \neq \mathsf{Commit}(sk_U, open_U)$, abort. Otherwise, set $credproof \leftarrow \mathsf{RandProof}((Nym_O, Nym_U), (0, open_U), cred)$. Note that, by the randomization properties of the proof system,

$$credproof \in \mathsf{NIZKPK}[sk_0 \text{ in } Nym_O; sk_L \text{ in } Nym_U]\{(F(sk_0), \ldots, F(sk_L), auth_1, \ldots, auth_L) : \mathsf{VerifyAuth}(params_A, sk_0, (sk_1, r_1), auth_1) \wedge \ldots \wedge \mathsf{VerifyAuth}(params_A, sk_{L-1}, (sk_L, r_L), auth_L)\} \ .$$

$\mathsf{CredVerify}(params_{DC}, Nym_O, credproof, Nym_U, L)$ runs $\mathsf{PKVerify}$ to verify the above.

$\mathsf{Issue}(params_{DC}, Nym_O, sk_I, Nym_I, open_I, cred, Nym_U, L)$
$\leftrightarrow \mathsf{Obtain}(params_{DC}, Nym_O, sk_U, Nym_U, open_U, Nym_I, L)$. If $L = 0$ and $Nym_O \neq Nym_I$, then this protocol is aborted. The issuer verifies his $cred$ using $\mathsf{CredVerify}$ and if it does not verify or if $Nym_I \neq \mathsf{Commit}(sk_I, open_I)$ or Nym_U is not a valid pseudonym, the issuer aborts.
Else, the issuer and the user both compute $r_{L+1} = H(Nym_O, L{+}1)$. The issuer and the user run a two-party protocol with the following specifications: the public input is (Nym_I, Nym_U, r_{L+1}); the issuer's private input is $(sk_I, open_I)$ and the user's private input is $(sk_U, open_U)$. The output of the protocol is as follows: if $(sk_I, open_I)$ and $(sk_U, open_U)$ do not appropriately correspond to Nym_I, Nym_U, the protocol aborts; otherwise, the issuer receives no output while the user receives as output the value π computed as:

$$\pi \leftarrow \mathsf{NIZKPK}[sk_I \text{ in } Nym_I; sk_U \text{ in } \mathsf{Commit}(sk_U, 0)]\{(F(sk_I), F(sk_U), auth) : \mathsf{VerifyAuth}(params_A, sk_I, (sk_U, r_{L+1}), auth)\} \ .$$

We need the 2PC protocol to remain secure even if the adversary is given some trapdoor about $params_{DC}$. (See full version for formal statement.) In Section 3.5 we gave an efficient instantiation of such a 2PC protocol for the specific authentication and NIZKPK schemes we use.
If $L = 0$, then the user outputs $cred_U = \pi$. On the other hand, if $L > 0$, the issuer obtains $credproof_I \leftarrow \mathsf{CredProve}(params_{DC}, Nym_O, cred, sk_I, Nym_I, open_I, L)$ and sends it to the user. Let $S_U = \mathsf{Commit}(sk_U, 0)$. Intuitively, $credproof_I$ is proof that the owner of Nym_I has a level L credential under public key Nym_O, while π is proof that the owner of Nym_I delegated to the owner of S_U. The user concatenates $credproof_I$ and π to obtain $credproof_I \circ \pi$ To get $cred_U$, U now needs to project $credproof_I \circ \pi$ so it becomes a proof about (Nym_O, S_U) and not about Nym_I.

Theorem 2. *If* $\mathsf{AuthSetup}, \mathsf{AuthKg}, \mathsf{Auth}, \mathsf{VerifyAuth}$ *is an F-unforgeable certification-secure authentication scheme, and if H is a collision resistant hash function, and if* $\mathsf{PKSetup}, \mathsf{PKProve}, \mathsf{PKVerify}$ *is a randomizable, perfectly extractable, composable zero knowledge non-interactive proof of knowledge system with simulation setup* $\mathsf{SimSetup}$ *and extraction setup* $\mathsf{ExtSetup}$*, and if the two party protocol is trapdoor secure for the simulation trapdoors generated by* $\mathsf{SimSetup}$ *and* $\mathsf{ExtSetup}$*, then the above construction constitutes a secure anonymous delegatable credential scheme. (See full versionfor proof.)*

Remark 3. We can easily extend our construction to attach public attributes to each level of the credential. The main modification is that we now compute $r_\ell = H(sk_O, \ell, \mathsf{attr}_1, \ldots, \mathsf{attr}_\ell)$, where attr_i is the *set* of attributes added by the ith delegator in the delegation chain. When the user shows or delegates a credential, he must then display the all the attributes associated with each level.

References

[Bar01] Boaz Barak. Delegatable signatures. Technical report, Weizmann Institute of Science, 2001.

[BB04] Dan Boneh and Xavier Boyen. Short signatures without random oracles. In *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 54–73, 2004.

[BBS04] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures using strong diffie hellman. In *CRYPTO 2004*, LNCS. Springer Verlag, 2004.

[BCKL08] Mira Belenkiy, Melissa Chase, Markulf Kohlweiss, and Anna Lysyanskaya. P-signatures and noninteractive anonymous credentials. In *Theory of Cryptography Conference*, LNCS, pages 356–374. Springer Verlaag, 2008.

[BCM05] Endre Bangerter, Jan Camenisch, and Ueli M. Maurer. Efficient proofs of knowledge of discrete logarithms and representations in groups with hidden order. In Serge Vaudenay, editor, *Public Key Cryptography*, volume 3386 of *LNCS*, pages 154–171. Springer, 2005.

[BDMP91] Manuel Blum, Alfredo De Santis, Silvio Micali, and Guiseppe Persiano. Non-interactive zero-knowledge. *SIAM Journal of Computing*, 20(6):1084–1118, 1991.

[BGdMM] Lucas Ballard, Matthew Green, Breno de Medeiros, and Fabian Monrose. Correlation-Resistant Storage. Johns Hopkins University, CS Technical Report # TR-SP-BGMM-050705. `http://spar.isi.jhu.edu/~mgreen/correlation.pdf`, 2005.

[BLS04] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. *J. Cryptology*, 17(4):297–319, September 2004. Extended abstract in *Proceedings of Asiacrypt 2001*.

[Bra99] Stefan Brands. *Rethinking Public Key Infrastructure and Digital Certificates— Building in Privacy*. PhD thesis, Eindhoven Inst. of Tech. The Netherlands, 1999.

[BW07] Xavier Boyen and Brent Waters. Full-domain subgroup hiding and constant-size group signatures. In *Public Key Cryptography*, pages 1–15, 2007.

[Cha85] David Chaum. Security without identification: Transaction systems to make big brother obsolete. *Communications of the ACM*, 28(10):1030–1044, October 1985.

[CL01] Jan Camenisch and Anna Lysyanskaya. Efficient non-transferable anonymous multi-show credential system with optional anonymity revocation. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 93–118. Springer Verlag, 2001.

[CL02] Jan Camenisch and Anna Lysyanskaya. A signature scheme with efficient protocols. In *SCN 2002*, volume 2576 of *LNCS*, pages 268–289, 2002.

[CL04] Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In *CRYPTO 2004*, volume 3152 of *LNCS*, pages 56–72, 2004.

[CL06] Melissa Chase and Anna Lysyanskaya. On signatures of knowledge. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 78–96, 2006.

[CS97] Jan Camenisch and Markus Stadler. Efficient group signature schemes for large groups. In Burt Kaliski, editor, *CRYPTO '97*, volume 1296 of *LNCS*, pages 410–424. Springer Verlag, 1997.

[CS03] Jan Camenisch and Victor Shoup. Practical verifiable encryption and decryption of discrete logarithms. In *CRYPTO '03*, volume 2729 of *LNCS*, pages 126–144, 2003.

[Dam90] Ivan Bjerre Damgård. Payment systems and credential mechanism with provable security against abuse by individuals. In Shafi Goldwasser, editor, *CRYPTO '88*, volume 403 of *LNCS*, pages 328–335. Springer Verlag, 1990.

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, April 1988.

[GPS06] S.D. Galbraith, K.G. Paterson, and N.P. Smart. Pairings for cryptographers. Cryptology ePrint Archive, Report 2006/165, 2006. `http://eprint.iacr.org/`.

[GR04] S. Galbraith and V. Rotger. Easy decision diffie-hellman groups. *Journal of Computation and Mathematics*, 7:201–218, 2004.

[GS08] Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In Nigel Smart, editor, *EUROCRYPT 2008*, 2008.

[LP07] Yehuda Lindell and Benny Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *EUROCRYPT 2007*, 2007.

[LRSW99] Anna Lysyanskaya, Ron Rivest, Amit Sahai, and Stefan Wolf. Pseudonym systems. In Howard Heys and Carlisle Adams, editors, *Selected Areas in Cryptography*, volume 1758 of *LNCS*, 1999.

[Lys02] Anna Lysyanskaya. *Signature Schemes and Applications to Cryptographic Protocol Design*. PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, September 2002.

[Sco02] Mike Scott. Authenticated id-based key exchange and remote log-in with insecure token and pin number. `http://eprint.iacr.org/2002/164`, 2002.

[SCP00] Alfredo De Santis, Giovanni Di Crescenzo, and Giuseppe Persiano. Necessary and sufficient assumptions for non-interactive zero-knowledge proofs of knowledge for all NP relations. In Ugo Montanari, José P. Rolim, and Emo Welzl, editors, *Proc. 27th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 1853 of *LNCS*, pages 451–462. Springer Verlag, 2000.

[Ver04] Eric R. Verheul. Evidence that xtr is more secure than supersingular elliptic curve cryptosystems. *J. Cryptology*, 17(4):277–296, 2004.