Randomizable Proofs and 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 credentials 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 a 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 instance of *controlled rerandomization* of non-interactive zero-knowledge proofs by a *third-party*. Our construction uses Groth-Sahai proofs (Eurocrypt 2008).

1 Introduction

Access control is one of the most fundamental 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 the 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 the burden on the system administrator if he had to approve every single moderator and user for every single forum.

We want cryptographic credentials to follow the same delegation model as access control follows in the real world. The system administrator can use his public key to sign a webmaster's public key, creating a credential of length 1. In general, a user with a level L credential can sign another user's public key and give him his credential chain, to create a level $L+1$ credential.

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. Our main contribution is the first efficient 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.

Pseudonymous systems. Prior work on anonymous credentials [CL02b,BCKL08] created systems where each user has one secret key but multiple "public keys." Given a secret key sk_A, Alice can create a new public key by choosing a random value *open* and publishing a commitment $pk_A = \mathsf{Commit}(sk_A, open)$. Alice could register pk_A with Oliver and pk'_A with Olga.

Oliver can give Alice a credential by signing the statement that the value in commitment pk_A has some attribute. Alice can then show pk'_A to Olga and prove that Oliver signed that the value in pk'_A had that attribute. This works because pk_A and pk'_A are commitments to the same value sk_A. The chief building block of an anonymous credential scheme is a signature scheme that lends itself 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.

Why delegation is a challenging problem. There is no straightforward transformation of anonymous credential schemes [Cha85,Bra99,LRSW99,CL01,BCKL08] into delegatable schemes. Suppose instead of giving Alice a direct credential, Oliver delegates his own credential to Alice. If Oliver gives Alice a *sig* on Oliver's secret key, then Alice could learn who gave Oliver his credential. Generalizing this approach would reveal to Alice the identity of every person in Oliver's credential chain.

Our approach. Instead of giving Alice his signature, Oliver gives Alice a non-interactive proof-of-knowledge of the signature. We show how Alice can (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 and 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 is still known only 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. However, 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.

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.

Randomizable proof systems. The key to our construction is a randomizable proof system that lets the prover (1) randomize the proof *without knowing the witness*, and (2) *control* the outcome of the randomization process. This is a fundamentally new notion, and one we think will be of independent interest. We give a formal definition and show that we can instantiate it by adding a randomization procedure to the pairing-based proof system of Groth and Sahai [GS08]. Our use of pairings is not merely a matter of efficiency – we do not know of any proof system based on general assumptions that can be randomized.

In fact the Groth-Sahai proofs allow us to go beyond merely randomizing the proof to actually change the statements we are proving. What do we mean by this? Groth-Sahai proofs make statements about the values inside commitments. Let $C = \mathsf{Commit}(x, open)$. A prover who knows $(x, open)$ can choose a new value $open'$ so that in the rerandomized proof, C is transformed to $C' = \mathsf{Commit}(x, open')$. Otherwise, the prover can choose whether to leave C unchanged or randomize it to C' that uses a some random $open'$ unknown to the prover. This fine level of control together with the basic randomization property gives a very useful building block, which is crucial in our application.

There has been prior work on some related notions: Burmester et al [BDI+99] show a third party can help randomize proofs during the execution of an *interactive* protocol to prevent subliminal channels. De Santis and Yung [DSY90] propose the notion of meta proofs, in which anyone who holds a proof for a given statement can generate a proof that there exists a proof for the statement. Neither of these approaches work for our scenario because we need to randomize

non-interactive proofs, and, unlike a meta-proof, the randomized proof must be indistinguishable from the original.

Our delegatable credentials construction. We construct delegatable credentials using randomizable proofs. By concatenating rerandomized credential chains, we can create a credential chain of length L that takes $O(L)$ space. Our strong anonymity properties are an immediate consequence of rerandomization: each showing of the credential is unlinkable and users do not learn the identities of delegators in their own credential chain.

Our solution (1) prevents adversarial users from mixing and matching pieces of different credential chains to create unauthorized credential chains and (2) protects the user's anonymity even when the adversary delegates *to* the user. We solve the second problem by creating an authentication scheme (symmetric signature scheme) that is secure even when the adversary gets a signature on the user's secret key.

Attributes. Our delegatable anonymous credentials 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 Crypto Forum" made him the "moderator of Crypto Forum". Our construction lets users add as many attributes as they want to each credential, allowing for the expressibility that we see in modern (non-anonymous) access control systems.

Advanced abuse prevention mechanisms. Our construction shows how to efficiently implement maximum anonymity at all levels and all roles in the delegation chain—with the exception of the credential authority. Some applications will not require this full anonymity. Indeed, a large number of abuse prevention mechanisms for anonymous credentials (anonymity revocation [CL01], credential revocation [CL02a], limited show [CHK$^+$06]) aim at striking a balance between privacy and accountability. Concerning our own scheme we make three simple observations: (i) global traceability can be achieved by providing a trusted tracing authority with the extraction trapdoors for the common parameters of the Groth Sahai (GS) proof system; (ii) at the last level of the delegation chain, we can make use of all abuse prevention mechanisms known for traditional anonymous credentials; (iii) many abuse prevention mechanisms known from the literature can be adapted to our construction by replacing traditional sigma proofs [Dam02] with GS proofs [GS08].

Other related work. At first glance, our delegatable credentials scenario might resemble the HIBE or HIBS settings [GS02,BBG05], where a root delegator can issue decryption or signing keys to recipients, who in turn can delegate subkeys to lower level participants. There are two key differences between such a HIBE/HIBS scheme and anonymous credential schemes: (1) In HIBE/HIBS two users with the same attributes are completely interchangeable while an anonymous credentials system gives them distinct sets of pseudonyms and (2) anony-

mous credentials allow a user to show that he has obtained valid credentials from two independent authorities.

In a somewhat different direction, Barak [Bar01] presented a general (inefficient) construction for delegatable signatures. In that work, the goal was a signature scheme which would allow the signer to delegate signing rights for a restricted message space, such that signatures generated with the delegated key are indistinguishable from the originals. In our setting, we want the opposite: once we delegate a credential, the delegatee should be able to issue lower level credentials to any users he chooses, however we require that credentials at different levels be clearly distinguishable. Finally, the definition in [Bar01] does not consider anonymity between the delegator and the delegatee, while we do.

Our contribution and organization of the paper. We (1) define and construct a randomizable NIZKPK (Section 2) and (2) define and construct an efficient delegatable anonymous credential system (Section 3). We also create an appropriate message authentication scheme and some other additional building blocks for the delegatable credentials scheme. We show how these building blocks can be instantiated under appropriate assumptions about groups with bilinear maps.

2 Randomizable NIZK proof systems

Let $R(params, y, w)$ be any polynomial-time computable relation. A non-interactive proof system for relation R allows the prover to convince a verifier that for some instance y there exists a witness w such that $R(params, y, w)$, where *params* is a common (public) reference string. The prover generates a proof $\pi \leftarrow \mathsf{Prove}(params, y, w)$, the verifier checks it via $\mathsf{VerifyProof}(params, y, \pi)$. A trusted third party runs $params \leftarrow \mathsf{Setup}(1^k)$ once to initialize the system.

Informally, zero-knowledge captures the notion that a verifier learns nothing from the proof but the truth of the statement. Witness indistinguishability merely guarantees that the verifier learns nothing about which witness was used in the proof. Soundness means an adversary cannot convince an honest verifier of a false statement. Completeness means all honest verifiers accept all correctly computed proofs. See [GMR89,Gol00,BFM88,FLS99] for formal definitions.

We define randomizable proof systems, which have an additional algorithm $\mathsf{RandProof}$ that takes as input a proof π for instance y in relation R, and produces a new proof for the same statement y. The resulting proof must be indistinguishable from a new proof for y. We allow the adversary to choose the instance y, the proof π that is used as input for $\mathsf{RandProof}$, and the witness w that is used to form a new proof of the same statement. Formally:

Definition 1. *We say that* $\mathsf{Setup}, \mathsf{Prove}, \mathsf{VerifyProof}, \mathsf{RandProof}$ *constitute a randomizable proof system if the following property holds. For all ppt.* $(\mathcal{A}_1, \mathcal{A}_2)$ *there exists a negligible function* ν *such that:*

$$Pr[params \leftarrow \mathsf{Setup}(1^k); (y, w, \pi, state) \leftarrow \mathcal{A}_1(params);$$
$$\pi_0 \leftarrow \mathsf{Prove}(params, y, w); \pi_1 \leftarrow \mathsf{RandProof}(params, y, \pi);$$
$$b \leftarrow \{0,1\}; b' \leftarrow \mathcal{A}_2(state, \pi_b) :$$
$$R_L(y, w) \wedge \mathsf{VerifyProof}(params, y, \pi) = 1 \wedge b = b'] \leq 1/2 + \nu(k) \ .$$

2.1 Instantiating a randomizable proof system

Randomization is a fundamentally new property. It is not clear how one might randomize proofs in any of the existing NIZK proof systems [BDMP91,KP98,FLS99] *without knowing the witness.* The one exception is the recent proof system of Groth and Sahai [GS08] (an extension of [GOS06]), which gives witness indistinghishable (and in some cases zero-knowledge) NIPKs. We will show how to add a randomization procedure to Groth-Sahai proofs.

Summary of Groth-Sahai proofs. Let $params_{BM} = (p, G_1, G_2, G_T, e, g, h)$ be the setup for pairing groups of prime order p, with pairing $e : G_1 \times G_2 \rightarrow G_T$, and g, h generators of G_1, G_2 respectively.[7]

The instance consists of the coefficients of a pairing product equation: $\{a_q\}_{q=1\ldots Q} \in G_1$, $\{b_q\}_{q=1\ldots Q} \in G_2$, $t \in G_T$, and $\{\alpha_{q,m}\}_{q=1\ldots Q, m=1\ldots M}$, $\{\beta_{q,n}\}_{q=1\ldots Q, n=1\ldots N} \in Z_p$. The prover knows $\{x_m\}_{m=1}^M$, $\{y_n\}_{n=1}^N$ that satisfy the pairing product equation $\prod_{q=1}^Q e(a_q \prod_{m=1}^M x_m^{\alpha_{q,m}}, b_q \prod_{n=1}^N y_n^{\beta_{q,n}}) = t$.

The prover creates perfectly binding, computationally hiding commitments $\{c_m\}_{m=1\ldots M}$ and $\{d_n\}_{n=1\ldots N}$ for all values x_m, y_n in G_1 and G_2 respectively. The instance is the pairing product equation (e.g. its coefficients) and the above commitments, while the witness, known only to the prover, is the values *and openings* of these commitments.

We now describe how to construct the proof. Let M_1, M_2, and M_T be R-modules for some ring R, and let $E\colon M_1 \times M_2 \rightarrow M_T$ 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. The public parameters $params_{PK}$ contain elements $u_1, \ldots, u_I \in M_1$, $v_1, \ldots, v_J \in M_2$ and values $\eta_{h,i,j}$, $1 \leq i \leq I$, $1 \leq j \leq J$, and $1 \leq h \leq H$.

To create a Groth-Sahai commitment to $x \in G_1$, 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] 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. The verifier computes, for all $1 \leq q \leq Q$, $\hat{c}_q \leftarrow \mu_1(a_q) \cdot \prod_{m=1}^M c_m^{\alpha_{q,m}}$ and $\hat{d}_q \leftarrow \mu_2(b_q) \cdot \prod_{n=1}^N d_n^{\beta_{q,n}}$. Then the verifier checks that $\prod_{q=1}^Q E(\hat{c}_q, \hat{d}_q) = \mu_T(t) \cdot \prod_{i=1}^I E(u_i, \pi_i) \cdot \prod_{j=1}^J E(\psi_j, v_j)$.

[7] For simplicity, we do not consider Groth-Sahai proofs for composite order groups.

Randomizing Groth-Sahai proofs. RandProof gets as input an instance with the $a_q, b_q, t, \alpha_{q,m}, \beta_{q,n}$ values as well as the proof $[(\pi_1, \ldots, \pi_I, \psi_1, \ldots, \psi_J), \Pi]$. Π contains the internal commitments $c_1, \ldots, c_M$ and $d_1, \ldots, d_N$.

The algorithm first chooses randomization exponents $(s_{1,1}, \ldots, s_{M,I})$ and $(z_{1,1}, \ldots, z_{N,J})$ at random from Z_p. It then rerandomizes the commitments c_m and d_n to $c'_m = c_m \cdot \prod_{i=1}^{I} u_i^{s_{m,i}}$ and $d'_n = d_n \cdot \prod_{j=1}^{J} v_j^{z_{n,j}}$. Then it computes $\hat{s}_{q,i} = \sum_{m=1}^{M} s_{m,i} \cdot \alpha_{q,m}$, $\hat{z}_{q,j} = \sum_{n=1}^{N} z_{n,j} \cdot \beta_{q,n}$, and $D'_q \leftarrow \mu_2(b_q) \cdot \prod_{n=1}^{N} d'^{\beta_{q,n}}_n$ and $C_q \leftarrow \mu_1(a_q) \cdot \prod_{m=1}^{M} c_m^{\alpha_{q,m}}$. 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] for a detailed explanation.

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 $d'_1, \ldots, d'_N$. See full version [BCC+08] for details. A similar approach works for composite order groups.

Composable Proofs. Groth-Sahai proofs are *composable* witness indistinguishable, and in some cases composable zero-knowledge. To simplify our definitions and proofs, we use a similar notion for randomizability.

In a composable (under the definition of Groth and Sahai [GS08]) non-interactive proof system there exists an algorithm SimSetup that outputs *params* together with a trapdoor *sim*, such that *params* output by SimSetup is indistinguishable from those output by Setup. Composable witness-indistinguishability (or zero-knowledge) requires that, under these parameters, the witness-indistinguishability (resp. zero-knowledge) property holds *even when the adversary is given the trapdoor sim*. Groth-Sahai commitments are perfectly hiding under the simulated parameters. (Under the honest parameters they are perfectly binding.) In the same spirit, we say the *composable* randomizability property must hold even when the distinguisher is given the trapdoor *sim*.

2.2 Malleable proofs and randomizable commitments

For our application, randomizing proofs is not sufficient. We also need to randomize (anonymize) the statement that we are proving. Consider a family of transformations $\{Y_s, P_s\}_{s \in S}$ that transform the instance and the proof respectively (for us, S is the set of all possible commitment openings). We require that $\forall (y, \pi), \forall s \in S$, if π is a valid proof for y, then $P_s(\pi)$ is a valid proof for $Y_s(y)$.

Definition 2. *We say that* Setup, Prove, VerifyProof, RandProof, $\{Y_s, P_s\}_{s \in S}$, *constitute a Y-malleable randomizable proof system, if for all ppt. $\mathcal{A}$ there exists a negligible ν such that:*

$$Pr[params \leftarrow \mathsf{Setup}(1^k); (y, \pi, s) \leftarrow \mathcal{A}(params) :$$
$$\mathsf{VerifyProof}(params, y, \pi) = 1 \wedge \mathsf{VerifyProof}(params, Y_s(y), P_s(\pi)) = 0] = \nu(k) \ .$$

If we apply $\mathsf{RandProof}$ to $P_s(\pi)$, then the result will be indistinguishable from a random fresh proof for $Y_s(y)$.

Groth-Sahai proofs can be used to prove that *the values in a given set of commitments* form a solution to a specific set of pairing product equations; the commitments can be part of the proof or the instance y. In our application, we will need to anonymize not only the proof, but also the commitments in the instance.

Suppose a prover wants to show that some $\mathsf{Condition}$ holds for the values inside commitments $C_1, \ldots, C_n$. Then the instance is $y = (\mathsf{Condition}, C_1, \ldots, C_n)$, and the witness is $w = (x_1, open_1, \ldots, x_n, open_n, z)$, where $(x_i, open_i)$ is the opening of commitment C_i, while z is some value that has nothing to do with the commitments. We define the relation $R = \{(params, y, w) | C_1 = \mathsf{Commit}(params, x_1, open_1) \wedge \ldots \wedge C_n = \mathsf{Commit}(params, x_n, open_n) \wedge \mathsf{Condition}(params, x_1, \ldots, x_n, z)\}$. A proof system supports randomizable commitments if there exist efficient algorithms Y and P, such that on input (s, y, π), where $s = (open'_1, \ldots, open'_n)$ and $\pi \leftarrow \mathsf{Prove}(params, y, w)$, (1) $Y(s, y)$ outputs instance $y' = (\mathsf{Condition}, C'_1, \ldots, C'_n)$, where $C'_i = Commit(params, x_i, open_i + open'_i)$, (2) $P(s, \pi)$ outputs a proof π' for instance y', and (3) Y and P fulfill the malleability requirements of Definition 2.

Lemma 1. *The Groth-Sahai proof system is malleable with respect to the randomness in the commitments.* See full version [BCC+08] for details.

Remark 1. To simplify notation, $\mathsf{RandProof}$ will take $s = (open'_1, \ldots, open'_n)$ as input, apply P_s, and then run the randomization algorithm. To leave C_i unchanged, we set $open'_i = 0$.

2.3 Partially Extractable Non-interactive Proofs of Knowledge

A NIPK system is a non-interactive proof system that is *extractable*. We recall the notion of *f-extractability* [BCKL08], which is an extension of the original definition of extractability [SCP00]. In an extractable proof system, there exists a ppt. extractor $(\mathsf{PKExtractSetup}, \mathsf{PKExtract})$. $\mathsf{PKExtractSetup}(1^k)$ outputs $(td, params)$ where $params$ is distributed identically to the output of $\mathsf{Setup}(1^k)$. For all polynomial time adversaries $\mathcal{A}$, the probability that $\mathcal{A}(1^k, params)$ outputs (y, π) such that $\mathsf{VerifyProof}(params, y, \pi) = \mathsf{accept}$ and $\mathsf{PKExtract}(td, y, \pi)$ fails to extract a witness w such that $R(params, y, w) = \mathsf{accept}$ 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 w' such that $\exists w : R(params, y, w) = \mathsf{accept} \wedge w' = f(params, w)$. If $f(params, \cdot)$ is the identity function, we get the usual notion of extractability.

Let C be an unconditionally binding commitment. By '$x \,\mathrm{in}\, C$' we mean $\exists open : C = \mathsf{Commit}(params, x, open)$. We use NIPK notation [CS97,BCKL08],

to denote an f-extractable NIPK for instance $(C_1, \ldots, C_n, \mathsf{Condition})$ with witness $(x_1, open_1, \ldots, x_n, open_n, z)$:

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

The f-extractability property ensures that if $\mathsf{VerifyProof}$ accepts then we can extract $f(params, (x_1, open_1, \ldots, x_n, open_n, z))$ from π, such that x_i is the content of the commitment C_i, and $\mathsf{Condition}(params, x_1, \ldots, x_n, z)$ is satisfied.

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

The *concatenation* 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 instance and appending them to the proof.

Groth-Sahai proofs give us NIPK of the form:

$$\mathsf{NIPK}_{\mathsf{GS}}[\{x_m \text{ in } c_m\}_{m=1}^{M}, \{y_n \text{ in } d_n\}_{n=1}^{N}]\{(x_1, \ldots, x_M, y_1, \ldots, y_N) : \prod_{q=1}^{Q} e(a_q \prod_{m=1}^{M} x_m^{\alpha_{q,m}}, b_q \prod_{n=1}^{N} y_n^{\beta_{q,n}}) = t\}.$$

3 Delegatable Anonymous Credentials

An anonymous delegatable credential system has only one type of participant: users. Each user has a single secret key and uses it to generate different pseudonyms. User A with secret key sk_A can be known to user O as $Nym_A^{(O)}$ and to user B as $Nym_A^{(B)}$. Any user O can become an originator of a credential; all he needs to do is publish one of his pseudonyms Nym_O as his public key. 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. Credentials received directly from the authority are level 1 credentials, credentials that have been delegated once are level 2 credentials, etc. A delegatable credential system consists of the following algorithms:

$\mathsf{Setup}(1^k)$ outputs the trusted public parameters of the system, $params_{DC}$.
$\mathsf{Keygen}(params_{DC})$ creates the secret key of a party in the system.
$\mathsf{Nymgen}(params_{DC}, sk)$. On each run, the algorithm outputs a new pseudonym Nym with auxiliary info $aux(Nym)$ for secret key sk.[8]

[8] We do not address how to prove ownership of a pseudonym; in our constructions this involves interactively proving knowledge of the opening of a commitment.

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 $L+1$ credential to a user U. The pseudonym Nym_O is the authority's public key, sk_I is the issuer'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 $L = 0$ then $cred = \epsilon$. The issuer gets no output, and the user 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.

3.1 Security Definition of Delegatable Credentials

We formally define a secure delegatable credential system in the full version [BCC+08]. Intuitively, the algorithms Setup, Keygen, Nymgen, 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.

Anonymity. 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 independent of the user's identity and delegation chain. The adversary should not even recognize a credential he delegated.

There must exist a simulator (SimSetup, SimProve, SimObtain, SimIssue). 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 even the adversary is given *sim*:

- *Nym* is distributed independently of *sk*.
- No adversary can tell if it is interacting with Issue run by an honest party with a proper credential, or with SimIssue which is not given the credential and the issuer's secret key, but only the name of the authority, the length of the credential chain, and the pseudonyms of the issuer and user.
- No adversary 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, the length of the credential chain, and the pseudonyms of the issuer and user.
- The simulator SimProve can output a fake *credproof* that cannot be distinguished from a real credential, even when SimProve is only told the authority, the length of the credential chain, and the pseudonym of the user.

Remark 2. Our definition implies the more complex but weaker definition in which the adversary only controls the public inputs to the algorithm. Our definition is easier to work with as we need only consider one protocol at a time, and only a single execution of each protocol.

Unforgeability. Each credential defines a specific delegation chain. We cannot monitor delegation between adversarial parties. However, we require that whenever the delegation chain shows that an honest player delegated a level L credential to some user, that delegation actually occurred.

In this game, all of the honest parties are controlled by a single oracle that keeps track of all honestly issued credentials. An adversary given access to this oracle should have only negligible probability of outputting a forged credential.

Let F be an efficiently computable bijection and a one-way function. There exists a ppt. algorithms $\mathsf{ExtSetup}$ and $\mathsf{Extract}$ with five properties:

- $\mathsf{ExtSetup}$ and Setup output identically distributed $params$.
- Under these parameters, pseudonyms are perfectly binding for sk.
- $\mathsf{Extract}$ always extracts the correct chain of L identities from an honestly generated level L $credproof$.
- Given an adversarially generated level L credential proof $credproof$ from authority Nym_O for the pseudonym Nym, $\mathsf{Extract}$ 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. More formally we require that for all ppt. $\mathcal{A}$ there exists a negligible ν such that:

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

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. $\mathcal{A}$ 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 his choice, either between honest users, 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 the chain of identities behind each credential being issued and store it on the list $\mathsf{ValidCredentialChains}$. For details, see full version [BCC+08].

Remark 3. 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 a simpler definition and analysis.

3.2 Construction of Delegatable Credentials

We construct delegatable credentials using a randomizable NIZK proof system with randomizable commitments (as described in Section 2) and a message authentication scheme for a vector of messages $\boldsymbol{m}$ (in our basic scheme $|\boldsymbol{m}| = 2$)in the common parameters model: $\mathsf{AuthSetup}(1^k)$ 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 $\boldsymbol{m}$, and $\mathsf{VerifyAuth}(params_A, sk, \boldsymbol{m}, auth)$ accepts if $auth$ is a proper authenticator for $\boldsymbol{m}$ under key sk. (We will discuss the properties we will require from this authentication scheme after we present our delegatable credentials construction.)

The parameters of the delegatable credentials system combine the parameters $params_A$ from the authentication scheme and $params_{PK}$ from the composable and randomizable NIZKPK system and its associated commitment scheme Commit. We assume that all algorithms are aware of these parameters and omit them when appropriate to simplify our notation.

Intuition behind our construction. 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 credentials by making his pseudonym Nym_O publicly available.

The user's secret credential *cred* is a 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). To show or delegate the credential, the user randomizes and mauls *cred* to obtain *credproof* using the $\mathsf{RandProof}$ algorithm described in Section 2. The resulting *credproof* is a proof about a proper pseudonym, $Nym_U = \mathsf{Commit}(sk_U, open)$ for a randomly chosen $open$.

Suppose a user with secret key sk_U has a level L credential from some authority O, 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 was used to generate authenticator $auth_{i+1}$ on message 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 then an output of $\mathsf{Auth}(params_A, sk_{i-1}, (sk_i, r_i))$. Let

F be an efficiently computable bijection. The user U's level L private credential *cred* is a proof of the form

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

Full construction. Let $\mathsf{PKSetup}, \mathsf{PKProve}, \mathsf{PKVerify}$, and $\mathsf{RandProof}$ be a randomizable NIPK system and let $\mathsf{AuthSetup}, \mathsf{AuthKg}, \mathsf{Auth}, \mathsf{VerifyAuth}$ be an authentication scheme, and let $H : \{0,1\}^* \to Z_p$ 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 random $open$, compute $Nym = \mathsf{Commit}(params_{PK}, sk, open)$ and output pseudonym Nym and auxiliary information $open$.

$\mathsf{CredProve}(params_{DC}, Nym_O, cred, sk_U, Nym_U, open_U, L)$. 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. Return $credproof \leftarrow \mathsf{RandProof}((Nym_O, Nym_U), (0, open_U), cred)$.

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

$\mathsf{Issue}(params_{DC}, Nym_O, sk_I, Nym_I, open_I, cred, Nym_U, L)$
$\leftrightarrow \mathsf{Obtain}(params_{DC}, Nym_O, sk_U, Nym_U, open_U, Nym_I, L)$. Abort if $L = 0$ and $Nym_O \neq Nym_I$. The issuer verifies *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:

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

In Section 3.3 we give an efficient instantiation of such a 2PC protocol for the specific authentication and NIZKPK schemes we use.
If $L = 0$, then the user outputs $cred_U = \pi$. Otherwise, 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 a 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 needs to project $credproof_I \circ \pi$ into a proof about (Nym_O, S_U) instead of Nym_I.

Remark 4. We can attach public attributes to each level of the credential. We 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 display all the attributes associated with each level.

Message authentication scheme. Just like a signature scheme, an authentication scheme must be complete and unforgeable. For our application we need to strengthen the unforgeability property in two ways. First, we require *F-Unforgeability* [BCKL08], which guarantees that for some well-defined bijection F, no adversary can output $(F(\boldsymbol{m}), auth)$ without first getting an authenticator on m. (We write $F(\boldsymbol{m}) = F(m_1, \ldots, m_n)$ to denote $(F(m_1), \ldots, F(m_n))$.) Second we require a new property which we call *certification security*; the authenticator is unforgeable even if the adversary learns a signature on the challenge secret key. An authentication scheme is F-unforgeable and certification secure if for all ppt. adversaries $\mathcal{A}$ there exists negligible ν such that:

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

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 the signature $\mathsf{Auth}(params_A, sk^*, (sk, m_2, \ldots, m_n))$.

Theorem 1. *Let* $\mathsf{AuthSetup}, \mathsf{AuthKg}, \mathsf{Auth}, \mathsf{VerifyAuth}$ *be an F-unforgeable certification-secure authentication scheme, H be a collision resistant hash function, and* $\mathsf{PKSetup}, \mathsf{PKProve}, \mathsf{PKVerify}$ *be a randomizable, partially extractable, composable zero-knowledge non-interactive proof of knowledge system. Then the above construction constitutes a secure anonymous delegatable credential scheme. See full version [BCC+08] for proof.*

We will construct our authentication scheme based the BB-CDH and BB-HSDH assumptions (defined in Section 3.3). Groth-Sahai proofs require either the SXDH assumption or the Decision Linear Assumption [GS08]. Our two party protocol requires a homomorphic encryption scheme.

3.3 Building Block Instantiations

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. BB-CDH is implied by SDH; [Boy08] describes how to extend Generalized Diffie Hellman [BBG05] to cover these two assumptions and prove their generic group security.

Definition 3 (BB-HSDH). *Let $x, c_1 \ldots c_q \leftarrow Z_p$. 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 4 (BB-CDH). *Let $x, y, c_1 \ldots c_q \leftarrow Z_p$. On input $g, g^x, g^y \in G_1$, $h, h^x \in G_2$ and the tuple $\{g^{1/(x+c_\ell)}, c_\ell\}_{\ell=1\ldots q}$, it is computationally infeasible to output g^{xy}.*

F-Unforgeable Certification Secure Message Authentication Scheme. Our authentication scheme is based on 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 $F(m) = (g^m, u^m)$ (under a very strong assumption), and that the Groth-Sahai proof system can be used to prove knowledge of such a signature. Boneh-Boyen signatures are not certification secure because $\mathsf{Sign}_{sk}(m) = \mathsf{Sign}_m(sk)$. We show how to achieve certification security; we also authenticate a vector of messages and weaken the underlying security assumption. The construction is as follows: $\mathsf{Auth}(sk, m_1||m_2)$ chooses random keys K^*, K_1, K_2 and returns $(\mathsf{Sign}_{sk}(K^*), \mathsf{Sign}_{K^*}(K_1), \mathsf{Sign}_{K^*}(K_2), \mathsf{Sign}_{K_1}(m_1), \mathsf{Sign}_{K_2}(m_2), F(K^*), F(K_1), F(K_2))$. At a high level, this construction eliminates any symmetries between $\mathsf{Auth}_{sk}(m)$ and $\mathsf{Auth}_m(sk)$. See full version [BCC+08] for details.

Theorem 2. *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 [BCC+08] for proof. The signature scheme obtained by setting $pk = h^{sk}$ may be of independent interest.

Commitment scheme. A commitment to $x \in Z_p$ consists of two GS commitments $\mathsf{GSCommit}(h^x, o_1)$, $\mathsf{GSCommit}(u^x, o_2))$ and a NIPK_{GS} proof that these are commitments to the same value x. This allows us to extract $F(x) = (h^x, u^x)$.

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

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

Since Boneh-Boyen signatures are verified using pairing product equations, we can use Groth-Sahai proofs, see full version [BCC+08] for details.

Creating a NIZKPK of an authenticator. The issuer chooses K^*, K_1, K_2 and can generate most of the proof. Then, the issuer and user need to jointly compute a NIZKPK of a Boneh-Boyen signature on the user's secret key. We outline the protocol, see full version [BCC+08] for details.

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. The user with input m_1, and the issuer with input K_1 run the following protocol to compute $\mathsf{Sign}_{K_1}(m_1) = g^{1/(K_1+m_1)}$:

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 $x = \mathsf{Dec}(sk_{hom}, e_2)$, sends the user $\sigma^* = g^{1/x}$.
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.

Theorem 3. *The above is a secure two-party computation for computing Boneh-Boyen signatures.* (See full version [BCC+08] for proof.)

Acknowledgements. Jan Camenisch was supported in part by the European Commission through the ICT and IST programmes under contract ICT-216483 PRIMELIFE. Markulf Kohlweiss was supported in part by the Concerted Research Action (GOA) Ambiorics 2005/11 of the Flemish Government, by the IAP Programme P6/26 BCRYPT of the Belgian State (Belgian Science Policy), and in part by the European Commission through the ICT and IST programmes under the following contracts: ICT-216483 PRIMELIFE and ICT-216676 ECRYPT II. Hovav Shacham was supported by an AFOSR MURI grant and, while at the Weizmann Institute of Science, by a Koshland Scholars Program postdoctoral fellowship. Mira Belenkiy, Melissa Chase and Anna Lysyanskaya acknowledge the support of NSF grants 0831293, 0627553 and 0347661. The information in this document reflects only the authors' views.

References

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

[BB04] D. Boneh and X. Boyen. Short signatures without random oracles. In *Eurocrypt '04*.

[BBG05] D. Boneh, X. Boyen, and E. Goh. Hierarchical identity based encryption with constant size ciphertext. In *Eurocrypt '05*.

[BCC+08] M. Belenkiy, J. Camenisch, M. Chase, M. Kohlweiss, A. Lysyanskaya, and H. Shacham. Randomizable proofs and delegatable anonymous credentials. Cryptology ePrint Archive, Report 2008/428, 2008. `http://eprint.iacr.org/2008/428`.

[BCKL08] M. Belenkiy, M. Chase, M. Kohlweiss, and A. Lysyanskaya. P-signatures and noninteractive anonymous credentials. In *TCC '08*.

[BDI+99] M. Burmester, Y. Desmedt, T. Itoh, K. Sakurai, and H. Shizuya. Divertible and subliminal-free zero-knowledge proofs for languages. *Journal of Cryptology*, 12(3):197–223, Nov 1999.

[BDMP91] M. Blum, A. De Santis, S. Micali, and G. Persiano. Non-interactive zero-knowledge. *SIAM Journal of Computing*, 20(6):1084–1118, 1991.

[BFM88] M. Blum, P. Feldman, and S. Micali. Non-interactive zero-knowledge and its applications. In *STOC '88*.

[BLS04] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. *J. Cryptology*, 17(4):297–319, September 2004.

[Boy08] X. Boyen. The uber-assumption family. In *Pairing '08*.

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

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

[CHK+06] J. Camenisch, S. Hohenberger, M. Kohlweiss, A. Lysyanskaya, and M. Meyerovich. How to win the clonewars: efficient periodic n-times anonymous authentication. In *CCS '06*.

[CL01] J. Camenisch and A. Lysyanskaya. Efficient non-transferable anonymous multi-show credential system with optional anonymity revocation. In *Eurocrypt '01*.

[CL02a] J. Camenisch and A. Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In *Crypto '02*.

[CL02b] J. Camenisch and A. Lysyanskaya. A signature scheme with efficient protocols. In *SCN '02*.

[CL06] M. Chase and A. Lysyanskaya. On signatures of knowledge. In *Crypto '06*.

[CS97] J. Camenisch and M. Stadler. Efficient group signature schemes for large groups. In *Crypto '97*.

[Dam02] I. Damgård. On σ-protocols. Available at `http://www.daimi.au.dk/~ivan/Sigma.ps`, 2002.

[DSY90] A. De Santis and M. Yung. Cryptographic applications of the non-interactive metaproof and many-prover systems. In *Crypto '90*.

[FLS99] U. Feige, D. Lapidot, and A. Shamir. Multiple noninteractive zero knowledge proofs under general assumptions. *SIAM Journal on Computing*, 29(1):1–28, 1999.

[GMR89] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989.

[Gol00] O. Goldreich. *Foundations of Cryptography: Basic Tools*. Cambridge University Press, New York, NY, USA, 2000.

[GOS06] J. Groth, R. Ostrovsky, and A. Sahai. Perfect non-interactive zero knowledge for np. In *Eurocrypt '06*.

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

[GS02] C. Gentry and A. Silverberg. Hierarchical ID-based cryptography. In *Asiacrypt '02*.

[GS08] J. Groth and A. Sahai. Efficient non-interactive proof systems for bilinear groups. In *Eurocrypt '08*.

[KP98] J. Kilian and E. Petrank. An efficient non-interactive zero-knowledge proof system for np with general assumptions. *J. of Cryptology*, 1998.

[LRSW99] A. Lysyanskaya, R. Rivest, A. Sahai, and S. Wolf. Pseudonym systems. In *Selected Areas in Cryptography '99*.

[SCP00] A. De Santis, G. Di Crescenzo, and G. Persiano. Necessary and sufficient assumptions for non-interactive zero-knowledge proofs of knowledge for all NP relations. In *ICALP '00*.