Full version of an extended abstract published in Proceedings of Crypto 2009, Springer-Verlag, 2009. Available from the IACR Cryptology ePrint Archive as Report 2008/428.

Randomizable Proofs and Delegatable Anonymous Credentials

Mira Belenkiy, Microsoft, mibelenk@microsoft.com
Jan Camenisch, IBM Zurich Research Laboratory
Melissa Chase, Microsoft Research, melissac@microsoft.com
Markulf Kohlweiss, KU Leuven, ESAT-COSIC / IBBT, markulf.kohlweiss@esat.kuleuven.be
Anna Lysyanskaya, Brown University, anna@cs.brown.edu
Hovav Shacham, UC San Diego, hovav@cs.ucsd.edu

November 7, 2011

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 [Lys02, CL02b, CL04, 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, Dam90, Bra99, LRSW99, CL01, CL04, 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.

Our malleable proofs do not allow adversaries to forge credentials because we use an *extractable* proof system. This lets us extract a certification chain from any proof, no matter how randomized. We can use an adversary that forges a proof to attack the underlying authentication scheme.

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 $C = \mathsf{Commit}(x, \mathit{open})$. A prover who knows (x, open) can choose a new value open' so that in the rerandomized proof, C is transformed to $C' = \mathsf{Commit}(x, \mathit{open}')$. 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) anonymous 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.

Somewhat closer to our approach are anonymous proxy signatures [FP08]. There, the delegation of the signing rights is, however, non-interactive and requires that the delegator knows the public key and thus the identity of the delegee, while the parties in a delegatable anonymous credentials at all stages only know each other under unlinkable pseudonyms.

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 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 RandProof, and the witness w that is used to form a new proof of the same statement. Formally:

Definition 1 *We say that* Setup, Prove, VerifyProof, 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:*

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

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. This was also independently observed by Fuchsbauer and Pointcheval [FP09] for the variant of Groth-Sahai proofs based on composite order groups.

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 $e : G_1 \times G_2 \to G_T$, and g, h generators of G_1, G_2 respectively.[1]

Let $\{a_q\}_{q=1\ldots Q} \in G_1$, $\{b_q\}_{q=1\ldots Q} \in G_2, t \in G_T$, and $\{\alpha_{q,m}\}_{q=1\ldots Q, m=1\ldots M}$, $\{\beta_{q,n}\}_{q=1\ldots Q, n=1\ldots N} \in Z_p$ be the coefficients of a pairing product equation The prover in a Groth-Sahai proof system knows secret values $\{x_m\}_{m=1}^{M}$, $\{y_n\}_{n=1}^{N}$ and wants to prove that these values satisfy the pairing

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

product equation

$$\prod_{q=1}^{Q} e(a_q \prod_{m=1}^{M} x_m^{\alpha_{q,m}}, b_q \prod_{n=1}^{N} y_n^{\beta_{q,n}}) = t\ .$$

We now give some details about the internals of Groth-Sahai proof systems. 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. 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 $\eta_{h,i,j}$, $1 \le i \le I$, $1 \le j \le J$, and $1 \le h \le H$.

To create a Groth-Sahai commitment to $x \in G_1$, choose random opening $\mathit{open} = (r_1, \ldots, r_I) \leftarrow R^I$, 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 $\mathit{open} \in R^J$. For simplicity we assume that $\mathsf{GSCommit}(\mathit{params}_{PK}, m, \mathit{open})$ first determines whether $m \in G_1$ or $m \in G_2$ and then follows the appropriate instructions.

In a Groth-Sahai proof system, the prover creates or knows the openings of 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.

Groth and Sahai [GS08] show how to efficiently compute proofs $\{\pi_i\}_{i=1}^{I}$, $\{\psi_j\}_{j=1}^{J}$ that prove that the values in the commitments c_m and d_n satisfy a pairing product equation. The verifier computes, for all $1 \le q \le Q$, $\hat{c}_q \leftarrow \mu_1(a_q) \cdot \prod_{m=1}^{M} c_m^{\alpha_{q,m}}$ and $\hat{d}_q \leftarrow \mu_2(b_q) \cdot \prod_{n=1}^{N} d_n^{\beta_{q,n}}$. Then the verifier checks that $\prod_{q=1}^{Q} E(\hat{c}_q, \hat{d}_q) = \mu_T(t) \cdot \prod_{i=1}^{I} E(u_i, \pi_i) \cdot \prod_{j=1}^{J} E(\psi_j, v_j)$.

Depending on the relation that is proven, the commitments $\{c_m\}_{m=1\ldots M}$ and $\{d_n\}_{n=1\ldots N}$ are interpreted as being part of the instance or as part of the proof. For ease of exposition we for now assume that all commitments are internal commitments, i.e., the proof consists of $[(\pi_1, \ldots, \pi_I, \psi_1, \ldots, \psi_J), \Pi]$ where Π contains the internal commitments $c_1, \ldots, c_M$ and $d_1, \ldots, d_N$. We will show how to relax this condition in Section 2.2.

Randomizing Groth-Sahai proofs. $\mathsf{RandProof}$ 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 internal 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 $D'_q \leftarrow \mu_2(b_q) \cdot \prod_{n=1}^{N} d_n'^{\beta_{q,n}}$ 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.[2]

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

[2] This procedure becomes slightly more complex, if not all commitments are internal: as randomization is supposed to leave the instance unchanged, commitments that are interpreted as being part of the instance are not randomized, i.e. the $s_{m,i}$, $z_{n,j}$ for these commitments are 0. We treat changes to such commitments as a form of malleability, see Section 2.1.

$t_{i,j}, t_h \leftarrow R$, 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 Appendix D 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 it is randomizable and if for all ppt. $\mathcal{A}$ there exists a negligible ν such that:*

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

If we apply 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 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 $w = (x_1, \mathit{open}_1, \ldots, x_n, \mathit{open}_n, z)$, 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 $R = \{(\mathit{params}, y, w) | C_1 = \mathsf{Commit}(\mathit{params}, x_1, \mathit{open}_1) \wedge \ldots \wedge C_n = \mathsf{Commit}(\mathit{params}, x_n, \mathit{open}_n) \wedge \mathsf{Condition}(\mathit{params}, x_1, \ldots, x_n, z)\}$.

A proof system supports randomizable commitments if there exist efficient algorithms Y and P, such that on input (s, y, π), where $s = (\mathit{open}'_1, \ldots, \mathit{open}'_n)$ and $\pi \leftarrow \mathsf{Prove}(\mathit{params}, y, w)$, (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 Appendix D for details.

Remark 1 *To simplify notation,* 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 $(x_1, open_1, \ldots, x_n, open_n, z)$:

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

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 (in which all commitments are interpreted as being part of the instance) give us NIPK of the form:

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

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.[3]

$\mathsf{Issue}(params_{DC}, Nym_O, sk_I, Nym_I, aux(Nym_I), cred, Nym_U, L)$

$\leftrightarrow \mathsf{Obtain}(params_{DC}, Nym_O, sk_U, Nym_U, aux(Nym_U), Nym_I, L)$ 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 $cred_U$.

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

$\mathsf{CredVerify}(params_{DC}, Nym_O, credproof, Nym, L)$. 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 Appendix A. 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 ↔ Issue and the honest user gets a proper credential.

[3] 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.

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 ($\mathsf{SimSetup}$, $\mathsf{SimProve}$, $\mathsf{SimObtain}$, $\mathsf{SimIssue}$). $\mathsf{SimSetup}$ 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 $\mathsf{SimIssue}$ 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 $\mathsf{SimObtain}$ that is only given the authority, the length of the credential chain, and the pseudonyms of the issuer and user.
- The simulator $\mathsf{SimProve}$ can output a fake *credproof* that cannot be distinguished from a real credential, even when $\mathsf{SimProve}$ 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. 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 $\mathsf{Extract}$ 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.

In the game defining the property, 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 ExtSetup and Extract with five properties:

- ExtSetup and Setup output identically distributed *params*.
- Under these parameters, pseudonyms are perfectly binding for *sk*.
- 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, 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 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 Issue $\leftrightarrow$ 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 Extract algorithm to extract the chain of identities behind each credential being issued and store it on the list ValidCredentialChains. For details, see Appendix A.

The oracle responds to the following types of queries:

AddUser. The oracle runs $sk \leftarrow \mathsf{Keygen}(params_{DC})$. It stores $(sk, F(sk))$ in the user database, gives the adversary $F(sk)$, and stores $F(sk)$ in the list HonestUsers.

$\mathsf{FormNym}(f)$. The oracle looks up (sk, f) in its user database and terminates if it does not exist. It calls $(Nym, aux(Nym)) \leftarrow \mathsf{Nymgen}(params_{DC}, sk)$. The oracle stores $(sk, Nym, aux(Nym))$ in its pseudonym database and gives the adversary Nym.

$\mathsf{Issue}(Nym_I, Nym_U, cred_I, L, Nym_O)$. The oracle looks up $(sk_U, Nym_U, aux(Nym_U))$ and $(sk_I, Nym_I, aux(Nym_I))$ in its pseudonym database and outputs an error if they do not exist, or if $sk_U = sk_I$ (honest users cannot issue to themselves). The oracle runs $\mathsf{CredProve}(params_{DC}, Nym_O, cred_I, sk_I, Nym_I, aux(Nym_I), L)$ to obtain $credproof$ (for $L = 0$, $credproof = \epsilon$). It runs $\mathsf{Extract}(params_{DC}, td, credproof, Nym_O, Nym_I, L)$ to obtain $f_0, f_1, \ldots f_L$. The oracle runs $\mathsf{Issue}(params_{DC}, Nym_O, sk_I, Nym_I, aux(Nym_I), cred_I, Nym_U, L) \leftrightarrow \mathsf{Obtain}(params_{DC}, Nym_O, sk_U, Nym_U, aux(Nym_U), Nym_I, L) \rightarrow cred_U$. The oracle stores $(f_0, L+1, f_L, F(sk_U))$ in $\mathsf{ValidCredentialChains}$ and outputs $cred_U$ to the adversary.

$\mathsf{IssueToAdv}(Nym_I, cred_I, Nym, L, Nym_O)$. The oracle looks up $(sk_I, Nym_I, aux(Nym_I))$ in its database of pseudonyms, and outputs an error if they do not exist. The oracle generates a credential proof by running $\mathsf{CredProve}(params_{DC}, Nym_O, cred, sk_I, Nym_I, aux(Nym_I), L)$ to obtain $credproof$. It runs $\mathsf{Extract}(params_{DC}, td, credproof, Nym_I, Nym_O, L)$ to obtain $f_0, \ldots f_L$. It then identifies the recipient by running $\mathsf{Extract}(params_{DC}, td, \epsilon, Nym, Nym, 0)$ to obtain f_{L+1}. Finally the oracle executes the algorithm $\mathsf{Issue}(params_{DC}, Nym_O, sk_I, Nym_I, aux(Nym_I), cred_I, Nym, L)$ interacting with the adversary. If the protocol does not abort, the oracle stores $(f_0, L+1, f_L, f_{L+1})$ in $\mathsf{ValidCredentialChains}$.

$\mathsf{ObtainFromAdv}(Nym_A, Nym_U, Nym_O, L)$. The oracle looks up $(sk_U, Nym_U, aux(Nym_U))$ in its database of pseudonyms, and outputs an error if they do not exist. Then it runs $\mathsf{Obtain}(params_{DC}, Nym_O, sk_U, Nym_U, aux(Nym_U), Nym_A, L)$ with the adversary to get $cred$. It outputs $cred$.

$\mathsf{Prove}(Nym, cred, Nym_O, L)$. The oracle looks up $(sk, Nym, aux(Nym))$ in its pseudonym database, and outputs an error if they do not exist. The oracle then runs $\mathsf{CredProve}(params_{DC}, Nym_O, cred, sk, Nym, aux(Nym), L)$ to obtain $credproof$, and outputs this result.

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 $\vec{m}$ (in our basic scheme $|\vec{m}| = 2$)in the common parameters model: $\mathsf{AuthSetup}(1^k)$ outputs common parameters $params_A$, $\mathsf{AuthKg}(params_A)$ outputs a secret key sk, $\mathsf{Auth}(params_A, sk, \vec{m})$ outputs an authentication tag $auth$ that authenticates a vector of messages $\vec{m}$, and $\mathsf{VerifyAuth}(params_A, sk, \vec{m}, auth)$ accepts if $auth$ is a proper authenticator for $\vec{m}$ 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

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

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 the issuer did not supply $(sk_I, open_I)$ such that $Nym_I = \mathsf{Commit}(sk_I, open_I)$, or if the user did not supply $(sk_U, open_U)$ such that $Nym_U = \mathsf{Commit}(sk_U, open_U)$, 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 $credproof_I \circ \pi$. $\in$

$$\mathsf{NIZKPK}[sk_O \text{ in } Nym_O; sk_I \text{ in } Nym_I; sk_U \text{ in } S_U]\{(F(sk_O), F(sk_1), \ldots, F(sk_{L-1}), F(sk_I), F(sk_U), \\ auth_1, \ldots, auth_L, auth_{L+1}) : \mathsf{VerifyAuth}(params_A, sk_O, (sk_1, r_1), auth_1) \wedge \\ \mathsf{VerifyAuth}(params_A, sk_1, (sk_2, r_2), auth_2) \wedge \ldots \wedge \\ \mathsf{VerifyAuth}(params_A, sk_{L-1}, (sk_I, r_L), auth_L) \wedge \\ \mathsf{VerifyAuth}(params_A, sk_I, (sk_U, r_{L+1}), auth_{L+1})\} \ .$$

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 $(F(\vec{m}), auth)$ without first getting an authenticator on m. (We write $F(\vec{m}) = F(m_1, \ldots, m_n)$ 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:

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

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

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 Appendix C.2 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 $e : G_1 \times G_2 \to G_T$. 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* $x, c_1 \dots c_q \leftarrow Z_p$*. On input* $g, g^x, u \in G_1$*,* $h, h^x \in G_2$ *and the tuple* $\{g^{1/(x+c_\ell)}, c_\ell\}_{\ell=1\dots q}$*, it is computationally infeasible to output a new tuple* $(g^{1/(x+c)}, h^c, u^c)$*.*

Definition 4 (BB-CDH) *Let* $x, y, c_1 \dots c_q \leftarrow Z_p$*. On input* $g, g^x, g^y \in G_1$*,* $h, h^x \in G_2$ *and the tuple* $\{g^{1/(x+c_\ell)}, c_\ell\}_{\ell=1\dots q}$*, 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 Appendix C 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 Appendix C 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 $\vec{m} = (m_1, m_2)$, 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 Appendix C.3 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 Appendix C 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 Appendix E 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] 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.

[BBG05] Dan Boneh, Xavier Boyen, and Eu-Jin Goh. Hierarchical identity based encryption with constant size ciphertext. In Ronald Cramer, editor, *Proceedings of Eurocrypt 2005*, LNCS. Springer, 2005.

[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.

[BDI$^+$99] Mike Burmester, Yvo Desmedt, Toshiya Itoh, Kouichi Sakurai, and Hiroki Shizuya. Divertible and subliminal-free zero-knowledge proofs for languages. *Journal of Cryptology*, 12(3):197–223, Nov 1999.

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

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications. In *STOC '88: Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 103–112, New York, NY, USA, 1988. ACM Press.

[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.*

[Boy08] Xavier Boyen. The uber-assumption family. In *Pairing '08: Proceedings of the 2nd international conference on Pairing-Based Cryptography*, pages 39–56, Berlin, Heidelberg, 2008. Springer-Verlag.

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

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

[CHK$^+$06] Jan Camenisch, Susan Hohenberger, Markulf Kohlweiss, Anna Lysyanskaya, and Mira Meyerovich. How to win the clonewars: efficient periodic n-times anonymous authentication. In *CCS '06: Proceedings of the 13th ACM conference on Computer and communications security*, pages 201–210, New York, NY, USA, 2006. ACM Press.

[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.

[CL02a] Jan Camenisch and Anna Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 61–76. Springer Verlag, 2002.

[CL02b] 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.

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

[DSY90] Alfredo De Santis and Moti Yung. Cryptographic applications of the non-interactive metaproof and many-prover systems. In Alfred J. Menezes and Scott A. Vanstone, editors, *CRYPTO 1990*, volume 537 of *LNCS*, pages 366–377. Springer Verlag, 1990.

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

[FP08] Georg Fuchsbauer and David Pointcheval. Anonymous proxy signatures. In Rafail Ostrovsky, Roberto De Prisco, and Ivan Visconti, editors, *SCN*, volume 5229 of *Lecture Notes in Computer Science*, pages 201–217. Springer, 2008.

[FP09] Georg Fuchsbauer and David Pointcheval. Proofs on encrypted values in bilinear groups and an application to anonymity of signatures. In Hovav Shacham and Brent Waters, editors, *Pairing*, volume 5671 of *Lecture Notes in Computer Science*, pages 132–149. Springer, 2009.

[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.

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

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

[GOS06] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Perfect non-interactive zero knowledge for np. In *EUROCRYPT*, pages 339–358, 2006.

[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/`.

[GS02] Craig Gentry and Alice Silverberg. Hierarchical ID-based cryptography. In Yuliang Zheng, editor, *ASIACRYPT 2002*, volume 2501 of *LNCS*, pages 548–566. Springer Verlag, 2002.

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

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

[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.

[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.

A Definition of Delegatable Credentials

We say that a function $\nu : \mathbb{Z} \to \mathbb{R}$ 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.

Correctness. We say that a credential *cred* is *a proper level L credential* for organization Nym_O with respect to $(params_{DC}, sk)$ iff it creates accepting proofs for all pseudonyms, or more formally

$$\begin{aligned}\Pr[&Nym, aux(Nym) \leftarrow \mathsf{Nymgen}(params_{DC}, sk);\\ &credproof \leftarrow \mathsf{CredProve}(params_{DC}, Nym_O, cred, sk, Nym, aux(Nym), L) :\\ &\mathsf{CredVerify}(params_{DC}, Nym_O, credproof, Nym, L) = \mathsf{accept}] = 1.\end{aligned}$$

We define the predicate $\mathsf{proper}(params_{DC}, cred, sk, Nym_O, L)$ to be true iff *cred* is a proper level L credential with respect to $params_{DC}$, *cred*, *sk*, and Nym_O. We require the following property:

(a). Obtain always either outputs a proper $L+1$ credential (in the sense above) or aborts.

(b). Users with proper level L credentials can delegate proper level $L+1$ credentials. Users with credentials that are not proper will abort.

(c). $\mathsf{Issue}(params_{DC}, Nym_O, sk_D, Nym_D, aux(Nym_D), cred, Nym_U, L)$ aborts without starting any communication if $\mathsf{proper}(params_{DC}, cred, sk_D, Nym_O, L) = 0$, or if there does not exist sk_U, $aux(Nym_U)$ such that $\mathsf{VerifyAux}(params_{DC}, Nym_U, sk_U, aux(Nym_U)) = 1$, or if $\mathsf{VerifyAux}(params_{DC}, Nym_D, sk_D, aux(Nym_D)) = 0$.

(d). $\mathsf{CredProve}(params_{DC}, Nym_O, cred, sk, Nym, aux(Nym), L)$ aborts without output if $\mathsf{proper}(params_{DC}, cred, sk, Nym_O, L) = 0$, or if $\mathsf{VerifyAux}(params_{DC}, Nym, sk, aux(Nym)) = 0$.

(e.) $\mathsf{VerifyAux}$ always accepts pseudonyms generated by Nymgen.

Anonymity During any protocol when a user reveals his pseudonym *Nym* but does not intentionally reveal $(sk, aux(Nym))$, the other user should learn no information about $(sk, aux(Nym))$. By no information, we mean that the user can be replaced by a simulator that does not know $(sk, aux(Nym))$, but still can execute the protocol. The simulator $\mathsf{SimSetup}$, $\mathsf{SimProve}$, $\mathsf{SimObtain}$, $\mathsf{SimIssue}$ has the following properties:

(a). The public parameters generated by $\mathsf{SimSetup}$ is indistinguishable from those output by Setup.

$$\begin{aligned}|\Pr[&params_{DC} \leftarrow \mathsf{Setup}(1^k); b \leftarrow \mathcal{A}(params_{DC}) : b = 1]\\ &- \Pr[(params_{DC}, sim) \leftarrow \mathsf{SimSetup}(1^k); b \leftarrow \mathcal{A}(params_{DC}) : b = 1]| < \nu(k)\end{aligned}$$

(b). A pseudonym *Nym* reveals no information about the identity *sk*. Let $params_{DC}, sim \leftarrow \mathsf{SimSetup}(1^k)$, $sk \leftarrow \mathsf{Keygen}(params_{DC})$, and $(Nym, aux(Nym)) \leftarrow \mathsf{Nymgen}(params_{DC}, sk)$. Then $(params_{DC}, sim, Nym)$ is information theoretically independent of *sk*.

(c). The simulator can output a fake credential *credproof* that cannot be distinguished from a real credential, even though the simulator does not have access to sk_U and *cred* (and sk_U and *cred* are chosen adversarially). Formally, for all PPTM adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, there exists a negligible function ν so that:

$$\begin{aligned}
&|\Pr[(\mathit{params}_{DC}, \mathit{sim}) \leftarrow \mathsf{SimSetup}(1^k);\\
&\quad (\mathit{Nym}_O, \mathit{cred}, \mathit{sk}, \mathit{Nym}, \mathit{aux}(\mathit{Nym}), L, \mathit{state}) \leftarrow \mathcal{A}_1(\mathit{params}_{DC}, \mathit{sim});\\
&\quad \pi \leftarrow \mathsf{CredProve}(\mathit{params}_{DC}, \mathit{Nym}_O, \mathit{cred}, \mathit{sk}, \mathit{Nym}, \mathit{aux}(\mathit{Nym}), L); b \leftarrow \mathcal{A}_2(\mathit{state}, \pi) : b = 1]\\
&- \Pr[(\mathit{params}_{DC}, \mathit{sim}) \leftarrow \mathsf{SimSetup}(1^k);\\
&\quad (\mathit{Nym}_O, \mathit{cred}, \mathit{sk}, \mathit{Nym}, \mathit{aux}(\mathit{Nym}), L, \mathit{state}) \leftarrow \mathcal{A}_1(\mathit{params}_{DC}, \mathit{sim});\\
&\quad \mathsf{flag} \leftarrow \mathsf{Check}(\mathit{params}_{DC}, \mathit{Nym}_O, \mathit{cred}, \mathit{sk}, \mathit{Nym}, \mathit{aux}(\mathit{Nym}), L);\\
&\quad \pi \leftarrow \mathsf{SimProve}(\mathit{params}_{DC}, \mathit{sim}, \mathit{Nym}_O, \mathit{Nym}, L, \mathsf{flag}); b \leftarrow \mathcal{A}_2(\mathit{state}, \pi) : b = 1]| < \nu(k).
\end{aligned}$$

$\mathsf{Check}(\mathit{params}_{DC}, \mathit{Nym}_O, \mathit{cred}, \mathit{sk}, \mathit{Nym}, \mathit{aux}(\mathit{Nym}), L)$ outputs accept if $\mathsf{VerifyAux}(\mathit{params}_{DC}, \mathit{Nym}, \mathit{sk}, \mathit{aux}(\mathit{Nym})) = 1$ and $\mathsf{proper}(\mathit{params}_{DC}, \mathit{cred}, \mathit{sk}, \mathit{Nym}_O, L) = 1$

(d). The adversary cannot tell if it is interacting with Obtain or SimObtain. Formally, for all PPTM adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, there exists a negligible function ν so that:

$$\begin{aligned}
&\Big|\Pr[\mathit{params}_{DC}, \mathit{sim} \leftarrow \mathsf{SimSetup}(1^k);\\
&\quad (\mathit{Nym}_O, \mathit{sk}, \mathit{Nym}, \mathit{aux}(\mathit{Nym}), L, \mathit{Nym}_A, \mathit{state}) \leftarrow \mathcal{A}_1(\mathit{params}_{DC}, \mathit{sim});\\
&\quad b \leftarrow \mathcal{A}_2(\mathit{state}) \leftrightarrow \mathsf{Obtain}(\mathit{params}_{DC}, \mathit{Nym}_O, \mathit{sk}, \mathit{Nym}, \mathit{aux}(\mathit{Nym}), \mathit{Nym}_A, L) : b = 1]\\
&- \Pr[\mathit{params}_{DC}, \mathit{sim} \leftarrow \mathsf{SimSetup}(1^k);\\
&\quad (\mathit{Nym}_O, \mathit{sk}, \mathit{Nym}, \mathit{aux}(\mathit{Nym}), L, \mathit{Nym}_A, \mathit{state}) \leftarrow \mathcal{A}_1(\mathit{params}_{DC}, \mathit{sim});\\
&\quad \mathsf{flag} \leftarrow \mathsf{Check}(\mathit{params}_{DC}, \mathit{sk}, \mathit{Nym}, \mathit{aux}(\mathit{Nym}));\\
&\quad b \leftarrow \mathcal{A}_2(\mathit{state}) \leftrightarrow \mathsf{SimObtain}(\mathit{params}_{DC}, \mathit{sim}, \mathit{Nym}_O, \mathit{Nym}, \mathit{Nym}_A, L, \mathsf{flag}) : b = 1]\Big| < \nu(k).
\end{aligned}$$

$\mathsf{Check}(\mathit{params}_{DC}, \mathit{sk}, \mathit{Nym}, \mathit{aux}(\mathit{Nym}))$ outputs accept in case $\mathsf{VerifyAux}(\mathit{params}_{DC}, \mathit{Nym}, \mathit{sk}, \mathit{aux}(\mathit{Nym})) = 1$ and reject otherwise.

(e). The adversary cannot tell if it is interacting with Issue or SimIssue. Formally, for all PPTM adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, there exists a negligible function ν so that:

$$\begin{aligned}
&|\Pr[(\mathit{params}_{DC}, \mathit{sim}) \leftarrow \mathsf{SimSetup}(1^k);\\
&\quad (\mathit{Nym}_O, \mathit{sk}, \mathit{Nym}, \mathit{aux}(\mathit{Nym}), \mathit{cred}, \mathit{Nym}_A, L, \mathit{state}) \leftarrow \mathcal{A}_1(\mathit{params}_{DC}, \mathit{sim});\\
&\quad \mathsf{Issue}(\mathit{params}_{DC}, \mathit{Nym}_O, \mathit{sk}, \mathit{Nym}, \mathit{aux}(\mathit{Nym}), \mathit{cred}, \mathit{Nym}_A, L) \leftrightarrow \mathcal{A}_2(\mathit{state}) \to b : b = 1]\\
&\quad - \Pr[(\mathit{params}_{DC}, \mathit{sim}) \leftarrow \mathsf{SimSetup}(1^k);\\
&\quad (\mathit{Nym}_O, \mathit{sk}, \mathit{Nym}, \mathit{aux}(\mathit{Nym}), \mathit{cred}, \mathit{Nym}_A, L, \mathit{state}) \leftarrow \mathcal{A}_1(\mathit{params}_{DC}, \mathit{sim});\\
&\quad \mathsf{flag} \leftarrow \mathsf{Check}(\mathit{params}_{DC}, \mathit{Nym}_O, \mathit{cred}, \mathit{sk}, \mathit{Nym}, \mathit{aux}(\mathit{Nym}), L);\\
&\quad \mathsf{SimIssue}(\mathit{params}_{DC}, \mathit{sim}, \mathit{Nym}_O, \mathit{Nym}, \mathit{Nym}_A, L, \mathsf{flag}) \leftrightarrow \mathcal{A}_2(\mathit{state}) \to b : b = 1]| < \nu(k)
\end{aligned}$$

$\mathsf{Check}(\mathit{params}_{DC}, \mathit{Nym}_O, \mathit{cred}, \mathit{sk}, \mathit{Nym}, \mathit{aux}(\mathit{Nym}), L)$ outputs accept iff $\mathsf{VerifyAux}(\mathit{params}_{DC}, \mathit{Nym}, \mathit{sk}, \mathit{aux}(\mathit{Nym})) = 1$ and $\mathsf{proper}(\mathit{params}_{DC}, \mathit{cred}, \mathit{sk}, \mathit{Nym}_O, L) = 1$.

Security of NymProve. NymProve must be a zero knowledge proof of knowledge of $\mathit{sk}, \mathit{aux}(\mathit{sk})$ such that $\mathsf{VerifyAux}(\mathit{params}_{DC}, \mathit{Nym}, \mathit{sk}, \mathit{aux}(\mathit{sk})) = 1$. Note that this can be an interactive proof system.

Unforgeability. Let F be an efficiently computable bijection. There exists an extractor ($\mathsf{ExtSetup}$, $\mathsf{Extract}$) with four properties:

(a). The parameters generated by $\mathsf{ExtSetup}$ are distributed identically as those generated by Setup.

(b). Under thse parameters pseudonyms are perfectly binding. I.e. for all $(\mathit{params}_{DC}, \mathit{td}) \leftarrow \mathsf{ExtSetup}$, for all Nym, if there exists $\mathit{aux}(\mathit{Nym}), \mathit{aux}(\mathit{Nym})'$ such that $\mathsf{VerifyAux}(\mathit{params}_{DC}, \mathit{Nym}, \mathit{sk}, \mathit{aux}(\mathit{Nym})) = 1$ and $\mathsf{VerifyAux}(\mathit{params}_{DC}, \mathit{Nym}, \mathit{sk}', \mathit{aux}(\mathit{Nym})') = 1$, then $\mathit{sk}' = \mathit{sk}$

(c). Given an honestly generated level L credential, $\mathsf{Extract}$ can always extract the corresponding chain of L identities.

Also, for any valid pseudonym, $\mathsf{Extract}$ will produce the appropriate $F(\mathit{sk}) \leftarrow \mathsf{Extract}(\mathit{params}_{DC}, \mathit{td}, \perp, \mathit{Nym}, \mathit{Nym}, 0)$. This can be seen as the extaction of a level 0 credential.

(d). Given an adversarially generated credential, $\mathsf{Extract}$ will always produce the correct values for f_0, f_L, or produce $\perp$.

$$\begin{aligned}
&\Pr[(\mathit{params}_{DC}, \mathit{td}) \leftarrow \mathsf{ExtSetup}(1^k);\\
&\quad (\mathit{credproof}, \mathit{Nym}, \mathit{Nym}_O, L), \leftarrow \mathcal{A}(\mathit{params}_{DC}, \mathit{td});\\
&\quad (f_0, \ldots, f_L) \leftarrow \mathsf{Extract}(\mathit{params}_{DC}, \mathit{td}, \mathit{credproof}, \mathit{Nym}, \mathit{Nym}_O, L) :\\
&\quad (f_0, \ldots, f_L) \neq \perp \wedge\\
&\quad ((\exists \mathit{sk}_{\mathcal{U}}{}' \, \exists \mathit{aux}(\mathit{Nym})' : \mathsf{VerifyAux}(\mathit{params}_{DC}, \mathit{Nym}, \mathit{sk}_{\mathcal{U}}{}', \mathit{aux}(\mathit{Nym})') \wedge F(\mathit{sk}_{\mathcal{U}}{}') \neq f_L)\\
&\quad \vee (\exists \mathit{sk}'_O \, \exists \mathit{aux}(\mathit{Nym}_O)' : \mathsf{VerifyAux}(\mathit{params}_{DC}, \mathit{Nym}_O, \mathit{sk}'_O, \mathit{aux}(\mathit{Nym}_O)') \wedge F(\mathit{sk}'_O) \neq f_0))]\\
&\quad \leq \nu(k) \ .
\end{aligned}$$

(e). No adversary can output a valid credential from which the extractor extracts an unauthorized chain of identities.

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

where $\mathcal{O}(\mathit{params}_{DC}, \mathit{command}, \mathit{input})$ behaves as follows:

$\mathsf{AddUser}$. The oracle runs $\mathit{sk} \leftarrow \mathsf{Keygen}(\mathit{params}_{DC})$. It stores $(\mathit{sk}, F(\mathit{sk}))$ in the user database and gives the adversary $F(\mathit{sk})$. Store $F(\mathit{sk})$ in the list $\mathsf{HonestUsers}$.

$\mathsf{FormNym}(y)$. The oracle looks up (sk, y) in its user database and terminates if it does not exist. It calls $(\mathit{Nym}, \mathit{aux}(\mathit{Nym})) \leftarrow \mathsf{Nymgen}(\mathit{params}_{DC}, \mathit{sk})$. The oracle stores $(\mathit{sk}, \mathit{Nym}, \mathit{aux}(\mathit{Nym}))$ in its pseudonym database and gives the adversary Nym.

$\mathsf{Issue}(\mathit{Nym}_D, \mathit{Nym}_U, \mathit{cred}_D, L, \mathit{Nym}_O)$. The oracle looks up $(\mathit{sk}_U, \mathit{Nym}_U, \mathit{aux}(\mathit{Nym}_U))$ and $(\mathit{sk}_D, \mathit{Nym}_D, \mathit{aux}(\mathit{Nym}_D))$ in its pseudonym database and outputs an error if they do not exist. The oracle then generates a credential proof by running $\mathsf{CredProve}(\mathit{params}_{DC}, \mathit{Nym}_O,$

$cred_D, sk_D, Nym_D, aux(Nym_D), L)$ to obtain $credproof_D$ (for $L = 0$, $credproof_D = \perp$). It runs $\mathsf{Extract}(params_{DC}, td, credproof_D, Nym_O, Nym_D, L)$ to obtain $f_0, f_1, \ldots f_L$. The oracle then runs

$$\mathsf{Issue}(params_{DC}, Nym_O, sk_D, Nym_D, aux(Nym_D), cred_D, Nym_U, L) \\ \leftrightarrow \mathsf{Obtain}(params_{DC}, Nym_O, sk_U, Nym_U, aux(Nym_U), Nym_D, L) \to cred_U.$$

Finally, the oracle stores $(f_0, L+1, f_L, F(sk_U))$ in ValidCredentialChains and outputs $cred_U$ to the adversary.

IssueToAdv($Nym_D, cred_D, Nym, L, Nym_O$). The oracle looks up $(sk_D, Nym_D, aux(Nym_D))$ in its pseudonym database, and outputs an error if they do not exist. The oracle generates a credential proof by running CredProve($params_{DC}, Nym_O, cred, sk_D, Nym_D, aux(Nym_D)$, L) to obtain $credproof_D$. It runs Extract($params_{DC}, td, credproof_D, Nym_D, Nym_O, L$) to obtain $f_0, \ldots f_L$. It then identifies the recipient by running Extract($params_{DC}, td, \perp, Nym$, $Nym, 0$) to obtain f_{L+1}. Finally the oracle executes the algorithm Issue($params_{DC}, Nym_O$, $sk_D, Nym_D, aux(Nym_D), cred_D, Nym, L$) interacting with the adversary. If the protocol does not abort, the oracle stores $(f_0, L+1, f_L, f_{L+1})$ in ValidCredentialChains.

ObtainFromAdv(Nym_A, Nym_U, Nym_O, L). The oracle looks up $(sk_U, Nym_U, aux(Nym_U))$ in its pseudonym database, and outputs an error if they do not exist. Then it runs Obtain($params_{DC}$,
$Nym_O, sk_U, Nym_U, aux(Nym_U), Nym_A$) with the adversary to get *cred*. It outputs *cred*.

Prove($Nym, cred, Nym_O, L$) The oracle looks up $(sk, Nym, aux(Nym))$ in its pseudonym database, and outputs an error if they do not exist. The oracle then runs CredProve($params_{DC}$, $Nym_O, cred, sk, Nym, aux(Nym), L$) to obtain $credproof_D$, and outputs this result.

B Security Proof for Delegatable Credential Construction

Claim 1 (Correctness) *The delegatable credential construction given in Section 3.2 satisfies the Correctness property*

Proof.

(a). Note that Obtain aborts if the *credproof* that it receives from the issuer does not pass CredVerify. Then Obtain only completes without aborting if the two party computation completes successfully. In this case, by the security of the two party computation, we are certain that π_L is indistinguishable from an honestly generated proof of knoweldge of an honestly generated authenticator. Then by the correctness of the NIZKPK system and correctness of the authentication scheme, we know that π_L will pass the NIZKPK verification. Thus, the new *cred* that results from combining *credproof* with π_L will pass the NIZKPK verification. Furthermore, by correctness of rerandomization, this means that any rerandomization of this proof will pass the NIZKPK verificaiton and thus be accepted by CredVerify, so *cred* is proper.

(b). Now, if Issue is interacting with Obtain with the appropriate inputs, then it will produce *credproof* that will be accepted by the honest user. Then the two party computation will be successful by the 2PC correctness property. That means Obtain will not abort, so by property (a). it will produce a proper credential.

(c). If the *cred* is not a proper credential, then by the properties of the rerandomization, CredVerify will not accept the resulting *credproof*, so Issue will abort. Issue also aborts if Nym_U is not valid, or if sk_I, $open_I$ do not satisfy VerifyAux.

(d). Note that if *cred* passes the CredVerify, then by correctness of rerandomization, it must be a proper credential. Thus, if *cred* is not a proper credential, or if $Nym, sk, aux(Nym)$ does not pass VerifyAux, then CredProve aborts. If both of thse are correct, then *credproof* is a rerandomization of a proper credential. That means that the proof must verify, so CredVerify will accept.

(e). Follows from the definition of VerifyAux and Nymgen.

□

Claim 2 (Anonymity) *The delegatable credential construction given in Section 3.2 satisfies the* Anonymity *property under the assumption that our building blocks are secure.*

Proof. Define the simulator algorithms as follows:

SimSetup(1^k). Uses AuthSetup(1^k) to generate $params_A$ for an F-unforgeable certificaton secure authentication scheme and then uses GSSimSetup to choose corresponding $params_P$ for a rerandomizable commitment scheme with a partially extractable randomizable composeable NIZKPK proof system, and to choose the appropriate trapdoor sim_{NIZK}. Finally, the setup chooses a collision resistant hash function H whose range is the message space of the authentication scheme and outputs public parameters $params_{DC} = (params_A, params_P, H)$, $sim = sim_{\text{NIZK}}$.

SimProve($params_{DC}, sim, Nym_O, Nym, L, flag$). If $flag$ = reject, abort and return $\perp$.

Otherwise let $Nym_0 = Nym_O$ and $Nym_L = Nym$, generate commitments $Nym_1, \ldots Nym_{L-1}$ to random values, and computes $r_1, \ldots r_L$ where $r_i = H(Nym_O, i)$.

Then we use the NIZKPK simulator and the simulation trapdoor sim to simulate proofs $\pi_1 \ldots \pi_L$, where π_i is a simulated proof of the form

$$\pi_i \leftarrow \mathsf{SimNIZKPK}[sk_{i-1} \text{ in } Nym_{i-1}; sk_i \text{ in } Nym_i]\{(F(sk_{i-1}), F(sk_i), auth) : \\ \mathsf{VerifyAuth}(params, sk_{i-1}, (sk_i, r_i), auth) = \mathsf{accept}\}$$

Finally, we output $Nym_0, \ldots Nym_L$, and $\pi_1 \circ \cdots \circ \pi_L$.

SimObtain($params_{DC}, Nym_O, Nym_U, Nym_A, L, flag$). If $flag$ = reject, abort and return $\perp$.

It then proceeds as follows:

1. Receives *credproof* from the adversary.
2. Runs CredVerify($params_{DC}, Nym_O, credproof, Nym_I, L$) to checks, that *credproof* is correct. If the checks fail, it aborts.

   Otherwise, it computes $r_L = H(Nym_O, L))$, the committed message $C_{m_1} = (Nym_U)$, and the public message $m_2 = r_L$.

3. Now we must simulate the two-party computation protocol. We will do this by using the 2PC simulator which interacts with a corrupt signer. (Note that this simulator expects no input from the trusted functionality.)

$\mathsf{SimIssue}(params_{DC}, sim, Nym_O, Nym_D, Nym_A, L, flag)$. If $flag = \mathsf{reject}$, abort and return $\perp$.

Otherwise let $Nym_0 = Nym_O$, $Nym_L = Nym_D$ and $Nym_{L+1} = Nym_A$, generate commitments $Nym_1, \ldots Nym_{L-1}$ to random values, and computes $r_1, \ldots r_{L+1}$ where $r_i = H(Nym_O, i)$.

Then we use the NIZKPK simulator and the simulation trapdoor sim to simulate proofs $\pi_1 \ldots \pi_{L+1}$, where π_i is a simulated proof of the form

$$\pi_i \leftarrow \mathsf{SimNIZKPK}[sk_{i-1} \text{ in } Nym_{i-1}; sk_i \text{ in } Nym_i]\{(F(sk_{i-1}), F(sk_i), auth) : \\ \mathsf{VerifyAuth}(params, sk_{i-1}, (sk_i, r_i), auth) = \mathsf{accept}\}$$

1. Send $Nym_0, \ldots Nym_L, \pi_1, \ldots \pi_L$ to the adversary
2. Receive C_{m_1} and check that it matches Nym_A.
3. Now we must simulate the two-party computation protocol. We will do this by using the 2PC simulator which interacts with a corrupt recipient. Note that this simulator expects to be provided with a proof of knowledge of the appropriate authenticator. Thus, we will give it the proof π_{L+1} computed above.

Now we will prove that these algorithms satisfy the required properties:

(a). Holds by the composable zero knowledge properties of the underlying NIZK proof system.

(b). Holds by the hiding property of the underlying composable commitment scheme.

(c). Note that the difference between SimProve and CredProve is that SimProve generates Nym_1, $\ldots, Nym_{L-1}$ as random commitments and uses SimNIZKPK to generated simulated proofs $\pi_1, \ldots, \pi_L$. These commitments are identical to the honest commitments by the strong computational hidingproperty, and the simulated proofs are indistinguishable from the honestly rerandomized proofs by the randomization property of the randomizable proof system.

(d). Note that the difference between SimObtain and Obtain is that SimObtain uses the simulator to run the two party computation. This should be indistinguishable from the honest Obtain protocol by the security of the 2PC.

(e). SimIssue differs in three ways from Issue. First, the initial *credproof* that is sent to the user is formed using SimNIZKPK instead of by rerandomizing a real *cred*. Second, SimIssue uses the simulator to run the two party computation. Third, the resulting π_{L+1} is the output of SimNIZKPK and not the result of a valid NIZKPK of an Auth computed by the 2PC.

We can prove that SimIssue and Issue are indistinguishable by considering several hybrid algorithms.

Hybrid 1 will be given the same input as Issue. It will verify that *cred* is proper, that it has been given a correct sk, Nym, $aux(Nym)$ and that Nym_A is a valid pseudonym. It will compute *credproof* honestly. Then it will use the 2PC simulator to run the two party protocol. This simulator will extract sk_A, $aux(Nym_A)$ from the user, and expect to receive a corresponding authenticator proof. We will use sk to form an authenticator on sk_A, r_{L+1}, and then we will use the honest NIZKPK to generate π_{L+1} from it. Finally, we will pass π_{L+1} to the 2PC simulator which will complete the protocol.

Note that Hybrid 1 is indistinguishable from the game involving the real Issue by the security of the 2PC.

Hybrid 2 will be given the same input as Issue. It will verify that *cred* is proper, that it has been given a correct sk, Nym, $aux(Nym)$ and that Nym_A is a valid pseudonym. It will compute *credproof* honestly. Then it will use the 2PC simulator to run the two party protocol. This simulator will extract sk_A, $aux(Nym_A)$ from the user, and expect to receive a corresponding authenticator proof. This time, we will use SimNIZKPK to simulate the proof of knowledge of an authenticator on for Nym_A. We will pass π_L to the 2PC simulator which will complete the protocol.

Note that Hybrid 2 is indistinguishable from Hybrid 1 by the zero knowledge properties of SimNIZKPK.

Finally, note that the difference between Hybrid 2 and game with SimIssue is that Hybrid 2 generates *credproof* by rerandomizing *cred*, while SimIssue uses the SimNIZKPK. This means the two games are indistinguishable by the rerandomization properties of the NIZKPK system.

□

Claim 3 (Unforgeability) *The delegatable credential construction given in Section 3.2 satisfies the unforgeability property under the assumption that our building blocks are secure.*

Proof. Let ExtSetup be the same as Setup except that when generating $params_{PK}$ it uses the extraction setup of the partially extractable randomizable composeable NIZKPK proof system.

(a) Follows from the indistinguishability of the extraction setup of the proof system.

(b) The commitment schemes used with the NIZKPK proof system is perfectly binding, so are our pseudonyms. Let F correspond to the F of the F-unforgeable authentication cheme and let Extract be an algorithm that verifies the credential and aborts with output $\perp$ if CredVerify rejects, otherwise it uses the extractability features of the NIZKPK proof system to extract all $F(sk_i)$ values.

(c) An honestly generated level L credential is a proof of knowledge of a certification chain from sk_0 to sk_L. It allows us to extract $(f_0, \ldots, f_L) = (F(sk_0), \ldots, F(sk_L))$. If the length of certification chain is 0, Extract extracts $f_0 = F(sk_O)$ from any valid commitment Nym_O using the extraction property of the commitment.

(d) If CredVerify accepts, then the NIPK extractor will succeed in extracting the appropriate values.

(e) Let Q be the maximum number of users in a credential system. We consider two games. In Game 1 the adversary plays the real unforgeability game. In Game 2 we pick a random $q \in \{1, \ldots, Q\}$. Game 2 is the same as Game 1, except that the oracle queries with command IssueToAdv and ObtainFromAdv are answered differently for the qth user.

Let sk^* be the secret key generated for the qth AddUser query.

$\mathsf{IssueToAdv}(Nym_I, cred_I, Nym, L, Nym_O)$. If Nym is not a valid pseudonym for sk, $aux(Nym)$, the oracle terminates. The oracle looks up $(sk_I, Nym_I, aux(Nym_I))$ in its pseudonym database, and outputs an error if they do not exist. If $sk_I \neq sk^*$ it proceeds as in Game 1. Otherwise the oracle follows the Issue protocol, but uses the simulator for the two-party protocol to simulate interaction with the adversarial user. Note that the simulator will extract the adversary's message m, and expect to get an authenticator proof as input. In this game we know sk^*, so we can generate this by simply computing the authenticator.

$\mathsf{ObtainFromAdv}(Nym_A, Nym_U, Nym_O, L)$. Similarly, the oracle looks up $(sk_U, Nym_U, aux(Nym_U))$ in its pseudonym database, and outputs an error if they do not exist. If $sk_U \neq sk^*$ the oracle runs $\mathsf{Obtain}(params_{DC}, Nym_O, sk_U, Nym_U, aux(Nym_U), Nym_A)$ with the adversary to get $cred$ (the same as Game 1). Otherwise it follows the Obtain protocol, and now uses the simulator for the two-party protocol to simulate the interaction with the adversarial issuer. It outputs $cred$.

By a simple hybrid argument either Game 1 and Game 2 are computationally indistinguishable or we break the security of the two-party computation.

Next we give a reduction to show that an adversary $\mathcal{A}$ that can win in Game 2 can be used to break the security of our authentication scheme. The reduction gets $params_A$, $f^* = F(sk^*)$ and access to $\mathcal{O}_{\mathsf{Auth}}(params_A, sk^*, .), \mathcal{O}_{\mathsf{Certify}}(params_A, ., (sk^*, ., \dots))$ for a challenge secret key sk^* from the authentication scheme's unforgeability game. It creates matching proof system parameters $params_{PK}$ and a trapdoor td, and combines them into $params_{DC}$. It hands $params_{DC}$ and td to $\mathcal{A}$.

The reduction answers $\mathcal{A}$s oracle queries as follows:

$\mathsf{AddUser}$. The oracle keeps a counter indicating the number of times it was queried, otherwise it behaves as the original oracle, except for the qth query. To answers the qth query the oracle stores $(\Box, F(sk^*))$ in the user database and adds $F(sk^*)$ in the list $\mathsf{HonestUsers}$. It returns $F(sk^*)$ to $\mathcal{A}$. Note that we use the special token '$\Box$' to indicate the unknown challenge key.

$\mathsf{FormNym}(f)$. The oracle looks up (sk, f) in its user database and terminates if it does not exist (we explicitly allow $sk = \Box$). If $f \neq f^*$ the oracle behaves as the original oracle. Otherwise it picks a random $aux(Nym)$ and computes $Nym = \mathsf{Commit}'(f, aux(Nym))$. The oracle stores $(\Box, Nym, aux(Nym))$ in its pseudonym database and gives the adversary Nym.

$\mathsf{Issue}(Nym_I, Nym_U, cred_I, L, Nym_O)$. The oracle looks up $(sk_U, Nym_U, aux(Nym_U))$ and $(sk_I, Nym_I, aux(Nym_I))$ in its pseudonym database and outputs an error if they do not exist or if $sk_U = sk_I$. If $sk_U \neq \Box$ and $sk_I \neq \Box$ the oracle behaves as the original oracle. Otherwise we distinguish two cases (note that $sk_U = \Box$ and $sk_I = \Box$ is not possible as honest users do not issue to themselves):
(a) $sk_U = \Box$: If $L > 0$, it runs $\mathsf{Extract}(params_{DC}, td, credproof_I, Nym_I)$ to obtain $f_0, f_1, \dots f_L$. If $f_L \neq F(sk_I)$, it aborts. Otherwise the oracle computes $cred_U$ in the same way as $\mathsf{Issue} \leftrightarrow \mathsf{Obtain}$ except that it uses a query to $\mathcal{O}_{\mathsf{Certify}}(params_A, sk_I, (sk^*, H(Nym_O, L)))$ to obtain the witness for proof π which it then can compute itself. The oracle stores $(f_0, L, f(sk_I), f^*)$ in $\mathsf{ValidCredentialChains}$ and outputs $cred_U$ to the adversary.
(b) $sk_I = \Box$: If $L > 0$,it runs $\mathsf{Extract}(params_{DC}, td, credproof_I, Nym_I)$ to obtain $f_0, f_1, \dots f_L$. If $f_L \neq f^*$, it aborts. Otherwise the oracle computes $cred_U$ in the same way as $\mathsf{Issue} \leftrightarrow \mathsf{Obtain}$ except that it uses a query to $\mathcal{O}_{\mathsf{Auth}}(params_A, sk^*, (sk_U, H(Nym_O, L)))$ to obtain the witness for proof π which it then can compute itself. The oracle stores $(f_0, L, f^*, F(sk_U))$ in $\mathsf{ValidCredentialChains}$ and outputs $cred_U$ to the adversary.

$\mathsf{IssueToAdv}(Nym_I, cred_I, Nym, L, Nym_O)$. If Nym is not a valid pseudonym for sk, $aux(Nym)$, the oracle terminates. The oracle looks up $(sk_I, Nym_I, aux(Nym_I))$ in its pseudonym database, and outputs $\perp$ if they do not exist. If $sk_I \neq \Box$ the oracle behaves as the original oracle. Otherwise the oracle follows the Issue protocol, but uses the simulator for the two-party protocol to simulate interaction with the adversarial user. We can simulate the ideal functionality of the two-party protocol with a $\mathcal{O}_{\mathsf{Auth}}(params_A, sk^*, H(Nym_O, L)))$ query. Note that the simulator of the two-party protocol provides us with sk_U. If the oracle's output is not $\perp$, the oracle stores $(f_0, L, f^*, f(sk_U))$ in $\mathsf{ValidCredentialChains}$.

$\mathsf{ObtainFromAdv}(Nym_A, Nym_U, Nym_O)$. Similarly, the oracle looks up $(sk_U, Nym_U, aux(Nym_U))$ in its pseudonym database, and outputs an error if they do not exist. If $sk_U \neq \Box$ the oracle behaves normally. Otherwise it follows the Obtain protocol, and now uses the simulator for the two-party protocol to simulate the interaction with the adversarial issuer. After a successful protocol execution the oracle outputs $cred = \mathcal{O}_{\mathsf{Certify}}(params_A, sk_I, (sk^*, H(Nym_O, L)))$.

$\mathsf{Prove}(Nym, cred, Nym_O)$. The prove protocol does not require a user's secret key, and is answered as the original oracle.

This simulation of Game 2 is perfect. We now need to show that a forgery in the credential system can be turned into a forgery for the authentication scheme.

A successful adversary outputs $(credproof, Nym, Nym_O, L)$ such that $\mathsf{CredVerify}(params_{DC}, Nym_O, credproof, Nym, L) = \mathsf{accept}$ and $(f_0, \ldots, f_L) \leftarrow \mathsf{Extract}(params_{DC}, td, credproof, Nym, Nym_O)$.

If we guessed correcly for which honest user $\mathcal{A}$ would create the forgery, $f_{i-1} = f^*$ and we can extract an $auth_i$ such that $\mathsf{VerifyAuth}(params_A, sk^*, (sk_i, H(Nym_O, i)), auth_i) = 1$. As $(f_0, i, f_i) \notin \mathsf{ValidCredentialChains}$, this message was never signed before and consitutes a valid authentication forgery. $\Box$

C A Certification-secure F-unforgeable Authentication Scheme

C.1 Intuition behind Construction

Our authentication scheme is loosely inspired by the weak Boneh Boyen signature scheme [BB04], in which the signature under a secret key sk on a message m is computed as $\mathsf{BBsig}(sk, m) = g^{\frac{1}{sk+m}}$. Our scheme is more complicated for several reasons. First, we must have a scheme which allows us to authenticate several messages at once, without increasing the size of the secret key (the secret key space must be a subset of the message space).

Second, as described in Section 3.2, we need a scheme which is unforgeable even when given access to a certification oracle. Note that the basic BB scheme is not secure in this sense – the response to a certification query with adversary key sk_A directly provides a forgery on the message $m = sk_A$.

Also, we want this scheme to be F-unforgeable, in the sense that it is also hard to produce $F(m)$ and an authenticator on m for an m which has not been signed. In this case our bijection F will be the information which we can extract from a commitment to a message, which will be in the group G_1, while if we emulate the BB scheme, m must be in the exponent space, Z_p. Thus the trivial bijection $F(m) = m$ will not work, and F unforgeability must be a stronger definition than standard unforgeability. Note, also that if $F(m) = g^m$, and if the adversary is given $v = F(sk)$ (as he is in our certification definition), the above authentication scheme is not F unforgeable – for any $f \in Z_p^*$, the value $g^{1/f}$ is a valid BB signature for $F(m) = g^f/v = g^{f-sk}$. Thus, our bijection F must be somewhat more complex.

Finally, the weak BB scheme is only secure for a previously determined polynomial sized message space. We want to be able to sign arbitrary messages, since we must be able to sign any randomly generated secret key. (This could be 'solved' by using an interactive assumption, however, as interactive assumptions are generally considered very strong, we would like to avoid them.)

For simplicity, we will describe an authentication scheme for authenticating two messages at once. It can be trivially extended to authenticate an arbitrary (polynomial) number of messages.

First, to solve the F-unforgeability problem, we use an approach similar to that given in the construction for P-signatures [BDMP91]. We add an element $u \in G_1$ to the parameters and set $F(m) = (h^m, u^m)$. For this F, we can show that the weak BB signature on random messages is F-unforgeable under the HSDH assumption. We will see soon how we allow arbitrary message. Security will rely on the stronger (but still non-interactive) BB-HSDH assumption.

Next, we need to authenticate many messages at once, using the BB scheme which allows us to sign only one message. A simple way to do this is to choose a new temporary key K^* for each authentication. Suppose we want to sign the message pair (m_1, m_2) under secret key sk. We use the BB signature to sign this key under the real secret key sk. Then, we use K^* to sign each of the messages. As, each K^* is only used in a single authentication, the adversary will only get BB signatures under K^* on m_1 and on m_2, thus, it will not be possible for him to use K^* in a forgery that involves any other messages. In some sense, K^* is used to tie together two messages. Note that we must also include $F(K^*)$: since the adversary cannot forge $F(K^*)$, $\mathsf{BBsig}(sk, K^*)$ pairs, we can conclude that he must reuse a previous m_1, m_2 message pair.

Thus, such an authentication would consist of $\mathsf{BBsig}(sk, K^*)$, $F(K^*)$, $\mathsf{BBsig}(K^*, m_1)$, and $\mathsf{BBsig}(K^*, m_2)$, which would be implemented as $(g^{\frac{1}{sk+K^*}}, h^{K^*}, u^{K^*}, g^{\frac{1}{K^*+m_1}}, g^{\frac{1}{K^*+m_2}})$.

These temporary keys also allow us (at least intuitively) to expand beyond random messages. We choose a random intermediate key K^*, and now sk is used to sign this random value, rather than one of the adversary's choice. Then K^* is used to sign the desired message. (Note that the security proof is actually much more complicated, as we must show that the BB signature generated under K^* is also unforgeable, but because it is only used to sign one message, this is somewhat easier. We will need to add additional temporary keys before we can fully prove security, but they will be necessary in any case as we will see below).

The resulting scheme still has two remaining problems. First, it is pretty clear that this scheme is forgeable because the order of the messages m_1, m_2 is not enforced – an authenticator on m_1, m_2 is also an authenticator on m_2, m_1. Second, the resulting authentication scheme is not certification secure: a certification query response $\mathsf{BBsig}(sk_A, K^*)$, $F(K^*)$, $\mathsf{BBsig}(K^*, sk)$, and $\mathsf{BBsig}(K^*, m_2)$, combined with the given value $F(sk)$ can easily be used to produce an authenticator under sk on (sk_A, m_2) (Note that $\mathsf{BBsig}(sk, K^*) = \mathsf{BBsig}(K^*, sk)$ and $\mathsf{BBsig}(K^*, sk_A) = \mathsf{BBsig}(sk_A, K^*)$).

It turns out that we can solve all of these problems at once. The solution is to add an additional level to our BB signature tree by adding two more temporary keys K_1, K_2 for each authentication. Now, sk will be used to BB sign K^*, which will in turn be used to BB sign K_1 and K_2. Finally K_1 will be used to sign m_1 and K_2 will be used to sign m_2. In order to prevent a forger from swapping the positions of K_1, m_1 and K_2, m_2, we add two additional values to our parameters, u_1, u_2. Let $F_1(x) = (h^x, u_1^x)$ and let $F_2(x) = (h^x, u_2^x)$. Similarly, we add a third value u^*, define $F^*(x) = (h^x, u^{*x})$, and include $F^*(K^*)$ in the authentication tag (instead of $F(K^*)$). The final authentication under key sk on messages (m_1, m_2) will be $\mathsf{BBsig}(sk, K^*)$, $F^*(K^*)$, $\mathsf{BBsig}(K^*, K_1)$,

$F_1(K_1)$, $\mathsf{BBsig}(K_1, m_1)$, $\mathsf{BBsig}(K^*, K_2)$, $F_2(K_2)$, and $\mathsf{BBsig}(K_2, m_2)$, which would be implemented as $(g^{\frac{1}{sk+K^*}}, h^{K^*}, u^{*K^*}, g^{\frac{1}{K^*+K_1}}, h^{K_1}, u_1^{K_1}, g^{\frac{1}{K_1+m_1}}, g^{\frac{1}{K^*+K_2}}, h^{K_2}, u_2^{K_2}, g^{\frac{1}{K_2+m_2}})$.

Note that because we are given only $F_1(K_1)$, and $F_2(K_2)$, the adversary will not be able to use an authentication of (m_1, m_2) to forge an authentication of (m_2, m_1) by the BB-CDH assumption (which is implied by SDH). Furthermore, we can actually show that the resulting scheme is certification secure. When the adversary makes a certification query for sk_A, it receives the values $F_1(K_1)$ and $\mathsf{BBsig}(K_1, sk)$ (as well as the rest). However, it cannot use this to forge a new authentication – an authentication using $\mathsf{BBsig}(sk, K_1)$ would need to be accompanied by $F^*(K_1)$, but the adversary will only know $F_1(K_1)$.

Thus our final authentication scheme to sign n messages at once is as follows:

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

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

C.2 Proof of Certification F-unforgeability of Authentication Scheme

Suppose we have an adversary which, can break the certification F-unforgeability property, by outputting a forgery of the form: $(F(m_1), \ldots, F(m_n)), (A^*, B^*, C^*, \{A_i, B_i, C_i, D_i\}_{1 \leq i \leq n})$.

If this is a valid forgery, then the verification equations hold, which implies that $\exists b^*, b_1, \ldots b_n$ such that: $A^* = g^{\frac{1}{sk+b^*}}, B^* = h^{b^*}, C^* = u^{*b^*}$ and for all $i \in 1, \ldots n$, $A_i = g^{\frac{1}{b^*+b_i}}, B_i = h^{b_i}, C_i = u_i^{b_i}, D_i = g^{\frac{1}{b_i+m_i}}$. We consider the following cases:

Case 1:

Suppose that with nonnegligible probability, the adversary produces a forgery with $b^* = K^*$, where K^* was used in the response to a previous authentication query. We divide this into three subcases:

Case 1a: Suppose that with nonnegligible probability, the adversary produces a forgery with $b^* = K^*$, where $K^*, K_1, \ldots, K_n$ were used together in the response to a previous authentication query, but there exists i such that for all $j \in \{1, \ldots, n\}$, $b_i \neq K_j$. In this case we will show that the BB-HSDH assumption with $q = n + 1$ does not hold. Our reduction proceeds as follows:

Setup: We are given the groups G_1, G_2, G_T of order p with bilinear map e, group elements $g, X_1 = g^x, v \in G_1$ and $h, X_2 = h^x \in G_2$ and the pairs $\{T_\ell = g^{\frac{1}{x+c_\ell}}, c_\ell\}_{1 \leq \ell \leq n+1}$. We need to produce a new tuple $g^{\frac{1}{x+c}}, h^c, v^c$. We make a guess i^* for which b_i will satsify the above condition. The parameters for the authentication scheme are computed as follows follows: choose random $z \leftarrow Z_p$, random $u \leftarrow G_1$, and random $u_j \leftarrow G_1$ for all $j \neq i^*$, and set $u_{i^*} = v$ and $u^* = g^z$. $\mathit{params} = (G_1, G_2, G_T, e, p, g, h, u, u^*, u_1, \dots u_n)$. Next, we set secret key $\mathit{sk} = c_{n+1}$ and send $F(\mathit{sk}) = h^{\mathit{sk}}, u^{\mathit{sk}}$ to the adversary.

Queries: Then we must answer the adversary's signing and certification queries. We make a guess λ for which query's K^* the adversary will attempt to reuse. Now, for signing queries other than query λ, the adversary sends message set $(m_1, \dots m_n)$. We compute $\mathsf{Auth}(\mathit{params}, \mathit{sk}, (m_1, \dots m_n))$ and send the result to the adversary.

For the λth signing query, we will implicitly set $K^* = x$. We set $K_j = c_j$ for all $j \in \{1, \dots, n\}$. Then we can compute the authentication: $\hat{A}^* = T_{n+1}, \hat{B}^* = X_2, \hat{C}^* = X_1^z, \{\hat{A}_j = T_j, \hat{B}_j = h^{c_j}, \hat{C}_j = u_j^{c_j}, \hat{D}_j = g^{\frac{1}{c_j+m_j}}\}_{1 \leq j \leq n}$.

For certification queries, the adversary sends key sk_A, and messages $m_2, \dots m_n$. We compute and return $\mathsf{Auth}(\mathit{params}, \mathit{sk}_A, (\mathit{sk}, m_2, \dots m_n))$.

Forgery: When the adversary produces a forgery, with nonegligible probability it will be of the form $F(m_1), \dots, F(m_n), A* = g^{\frac{1}{sk+b^*}}, B^* = h^{b^*}, C^* = u^{*b^*}, \{A_i = g^{\frac{1}{b^*+b_i}}, B_i = h^{b_i}, C_i = u_i^{b_i}, D_i = g^{\frac{1}{b_i+m_i}}\}_{1 \leq i \leq n}$. with $b^* = \hat{K}^*$, where $\hat{K}^*, \hat{K}_1, \dots \hat{K}_n$ were used together in the response to previous authentication query, but there exists i such that for all j, $b_i \neq \hat{K}_j$.

With nonnegligible probability, we will have correctly guessed λ such that $b^* = K^*$ for the K^* returned in the λth authentication query, and correctly guessed i^*. That means $b^* = x$, and A_i, B_i, C_i is a fresh HSDH triple.

Case 1b: Suppose that with nonnegligible probability, the adversary produces a forgery with $b* = K^*$, where $K^*, K_1, \dots, K_n$ were used together in the response to a previous authentication query, but there exists $i, j, i \neq j$ such that $b_i = K_j$. In this case we will show that the BB-CDH assumption with $q = 2$ does not hold. Our reduction proceeds as follows:

Setup: We are given the groups G_1, G_2, G_T of order p with bilinear map e, group elements $g, X_1 = g^x, Y = g^y \in G_1$ and $h, X_2 = h^x \in G_2$, integers $c_1, c_2 \in Z_p$, and values $T_1 = g^{\frac{1}{x+c_1}}$ and $T_2 = g^{\frac{1}{x+c_2}}$. We need to produce the value g^{xy}. We first make a guesses j^* and i^* for which K_j and b_i will satsify the above condition. We will give the adversary parameters for the authentication scheme as follows: we choose random $z \leftarrow Z_p$, random $u, u^* \leftarrow G_1$ and random $u_j \leftarrow G_1$ for all $j \neq j^*, j \neq i^*$, and we set $u_{i^*} = Y$ and $u_{j^*} = g^z$. $\mathit{params} = (G_1, G_2, G_T, e, p, g, h, u^*, u_1, \dots u_n)$. Next, we choose a random key sk and send $F(\mathit{sk}) = h^{\mathit{sk}}, u^{\mathit{sk}}$ to the adversary.

Queries: Then we must answer the adversary's signing and certification queries. First, we make a guess λ for which query's K^* the adversary will attempt to reuse.

Now, for signing queries other than query λ, when the adversary sends message set $(m_1, \dots m_n)$, we compute $\mathsf{Auth}(\mathit{params}, \mathit{sk}, (m_1, \dots m_n))$ and send the result to the adversary. For the λth signing query we set $K^* = c_2 - c_1 + m_{j^*}$ and implicitly set $K_{j^*} = x + c_1 - m_{j^*}$. (Thus, $K^* + K_{j^*} = x + c_2$ and $K_{j^*} + m_{j^*} = x + c_1$.) We randomly choose $K_j = z_j$ for all $j \in \{1, \dots, n\}$. Then we can compute the authentication: $\hat{A}^* = g^{\frac{1}{sk+c_2-c_1+m_{j^*}}}, \hat{B}^* = h^{sk}, \hat{C}^* = u^{*sk}$, $\{\hat{A}_j = g^{\frac{1}{c_2-c_1+m_{j^*}+z_j}}, \hat{B}_j = h^{z_j}, \hat{C}_j = u_j^{z_j}, \hat{D}_j = g^{\frac{1}{z_j+m_j}}\}_{1 \leq j \leq n, j \neq j^*}$. $\{\hat{A}_{j^*} = T_2, \hat{B}_{j^*} = X_2 h^{c_1 - m_{j^*}}, \hat{C}_j = X_1^z u_j^{c_1 - m_{j^*}}, \hat{D}_j = T_1$

For certification queries, the adversary sends key sk_A, and messages $m_2, \dots m_n$. We compute and return $\mathsf{Auth}(\mathit{params}, \mathit{sk}_A, (\mathit{sk}, m_2, \dots m_n))$.

Forgery: When the adversary produces a forgery, with nonegligible probability it will be of the form $F(m_1), \ldots, F(m_n), A* = g^{\frac{1}{sk+b^*}}, B^* = h^{b^*}, C^* = u^{*b^*}, \{A_i = g^{\frac{1}{b^*+b_i}}, B_i = h^{b_i}, C_i = u_i^{b_i}, D_i = g^{\frac{1}{b_i+m_i}}\}_{1 \leq i \leq n}$. with $b* = K^*$, where $K^*, K_1, \ldots, K_n$ were used together in the response to a previous authentication query, but there exists $i, j, i \neq j$ such that $b_i = K_j$.

With nonnegligible probability, we will have correctly guessed λ such that $b^* = K^*$ for the K^* returned in the λth authentication query, and correctly guessed i^*, j^*. Let M_{j^*} be the message that was signed by K_{j^*} in the λth authentication query. That means $b^* = K^*$, and $b_{i^*} = K_{j^*}$ from the λth triple, so $b_i^* = x + c_1 - M_{j^*}$, which means we have $C_{i^*} = u_{i^*}^{x+c_1-M_{j^*}} = Y^{x+c_1-M_{j^*}}$. Finally, we compute and return $C_{i^*}/Y^{c_1-M_{j^*}} = Y^x = g^{xy}$.

Case 1c: Suppose that with nonnegligible probability, the adversary produces a forgery with $b* = K^*$ where $K^*, K_1, \ldots K_n$ were used together in the response to a previous authentication query, and for all $i \in \{1, \ldots, n\}$, $b_i = K_i$. Suppose further that keys $K^*, K_1, \ldots K_n$ were originally used to sign message set $(M_1, \ldots M_n)$. Then since this is a forgery, there must exist at least one index j such that $m_j \neq M_j$. We break this into two further cases:

Case 1ci: The above forgery is such that $m_j \neq K^*$. In this case we will show that the BB-HSDH assumption with $q = 2$ does not hold. Our reduction proceeds as follows:

Setup: We are given the groups G_1, G_2, G_T of order p with bilinear map e, group elements $g, X_1 = g^x, v \in G_1$ and $h, X_2 = h^x \in G_2$ and the pairs $\{T_\ell = g^{\frac{1}{sk+c_\ell}}, c_\ell\}_{\ell=1,2}$. We need to produce a new tuple $g^{\frac{1}{sk+c}}, h^c, v^c$. We make a guess j^* for which m_j will satsify the above condition. We will give the adversary parameters for the authentication scheme as follows: we choose random $z \leftarrow Z_p$, random $u^* \leftarrow G_1$, and random $u_j \leftarrow G_1$ for all $j \neq j^*$, and we set $u_{j^*} = g^z$ and $u = v$. $params = (G_1, G_2, G_T, e, p, g, h, u, u^*, u_1, \ldots u_n)$. Next, we randomly choose secret key $sk \leftarrow Z_q$ and send $F(sk) = h^{sk}, u^{sk}$ to the adversary.

Queries: Then we must answer the adversary's signing and certification queries. We make a guess λ for which query's K^* the adversary will attempt to reuse. For signing queries other than query λ, when the adversary sends message set $(m_1, \ldots m_n)$, we compute $\mathsf{Auth}(params,\ sk, (m_1, \ldots, m_n))$ and send the result to the adversary.

For the λth signing query, we will implicitly set $K_{j^*} = x+c_1-m_{j^*}$. We set $K^* = c_2-c_1+m_{j^*}$ and randomly choose $K_j \leftarrow Z_p$ for all $j \in \{1, \ldots, n\}, j \neq j^*$. Then we can compute the authentication: $\hat{A}^* = g^{\frac{1}{sk+c_2-c_1+m_{j^*}}}, \hat{B}^* = h^{c_2-c_1+m_{j^*}}, \hat{C}^* = u^{*c_2-c_1+m_{j^*}}$, $\{\hat{A}_j = g^{\frac{1}{c_2-c_1+m_{j^*}+K_j}}, \hat{B}_j = h^{K_j}, \hat{C}_j = u_j^{K_j}, \hat{D}_j = g^{\frac{1}{K_j+m_j}}\}_{1 \leq j \leq n, j \neq j^*}$, $\hat{A}_{j^*} = Y_2, \hat{B}_{j^*} = X_2 h^{c_1-m_{j^*}}, \hat{C}_{j^*} = X_1^z u_{j^*}^{c_1-m_{j^*}}, \hat{D}_{j^*} = Y_1$.

For certification queries, the adversary sends key sk_A, and messages $m_2, \ldots m_n$. We compute and return $\mathsf{Auth}(params, sk_A, (sk, m_2, \ldots m_n))$.

Forgery: When the adversary produces a forgery, with nonegligible probability it will be of the form $F(m_1), \ldots, F(m_n), A* = g^{\frac{1}{sk+b^*}}, B^* = h^{b^*}, C^* = u^{*b^*}, \{A_i = g^{\frac{1}{b^*+b_i}}, B_i = h^{b_i}, C_i = u_i^{b_i}, D_i = g^{\frac{1}{b_i+m_i}}\}_{1 \leq i \leq n}$. with $b^* = \hat{K}^*$, where $\hat{K}^*, \hat{K}_1, \ldots \hat{K}_n$ were used together in the response to previous authentication query to sign message set $M_1, \ldots M_n$, and for all i, $b_i = \hat{K}_i$, and there exists j such that $m_j \neq M_j$ and $m_j \neq \hat{K}^*$.

With nonnegligible probability, we will have correctly guessed λ such that $b^* = K^*$ for the K^* returned in the λth authentication query, and correctly guessed j^*. That means $b_{j^*} = x + c_1 - m_{j^*}$, and $D_{j^*} = g^{\frac{1}{b_{j^*}+m_{j^*}}} = g^{\frac{1}{x+c_1-M_{j^*}+m_{j^*}}}$. Since we have said that $m_{j^*} \neq M_{j^*}$ and that $m_{j^*} \neq K^* =$

$c_2 - c_1 + M_{j^*}$, we are guaranteed that $D_{j^*}, h^{c_1 - M_{j^*} + m_{j^*}}, u_{j^*}^{c_1 - M_{j^*} + m_{j^*}}$ is a fresh HSDH triple.
Case 1cii: The above forgery is such that $m_j = K^*$. In this case we will show that the BB-CDH assumption does not hold for $q = n + 1$. Our reduction proceeds as follows:

Setup: We are given the groups G_1, G_2, G_T of order p with bilinear map e, group elements $g, X_1 = g^x, Y = g^y \in G_1$ and $h, X_2 = h^x \in G_2$, and $n + 1$ pairs $\{c_\ell,\ g^{\frac{1}{sk+c_\ell}}\}_{1 \leq \ell \leq n+1}$. We need to produce the value g^{xy}. We make a guess j^* for which m_j will satsify the above condition. We give the adversary parameters for the authentication scheme as follows: we choose random $z \leftarrow Z_p$, random $u \leftarrow G_1$ and random $u_j \leftarrow G_1$ for all $j \in \{1, \ldots, n\}$, and we set $u^* = X_1^z$ and $u = Y$. $params = (G_1, G_2, G_T, e, p, g, h, u, u^*, u_1, \ldots u_n)$. Next, we set secret key $sk = c_{n+1}$ and send $F(sk) = h^{sk}, u^{sk}$ to the adversary.

Queries: Then we must answer the adversary's signing and certification queries. We make a guess λ for which query's K^* the adversary will attempt to reuse. For signing queries other than query λ, when the adversary sends message set $(m_1, \ldots m_n)$, we compute $\mathsf{Auth}(params, sk, (m_1, \ldots, m_n))$ and send the result to the adversary. For the λth signing query, we set $K_j = c_j$ for $j \in \{1, \ldots, n\}$ and implicitly set $K^* = x$. Then we can compute the authentication: $\hat{A}^* = T_{n+1}, \hat{B}^* = X_2, \hat{C}^* = X_1^z$, $\{\hat{A}_j = T_j, \hat{B}_j = h^{c_j}, \hat{C}_j = u_j^{c_j}, \hat{D}_j = g^{\frac{1}{c_j + m_j}}\}_{1 \leq j \leq n}$.

For certification queries, the adversary sends key sk_A, and messages $m_2, \ldots m_n$. We compute and return $\mathsf{Auth}(params, sk_A, (sk, m_2, \ldots m_n))$.

Forgery: When the adversary produces a forgery, with nonegligible probability it will be of the form $F(m_1) = (h^{m_1}, u^{m_1}), \ldots, F(m_n) = (h^{m_n}, u^{m_n}), A* = g^{\frac{1}{sk+b^*}}, B^* = h^{b^*}, C^* = u^{*b^*}, \{A_i = g^{\frac{1}{b^*+b_i}}, B_i = h^{b_i}, C_i = u_i^{b_i}, D_i = g^{\frac{1}{b_i+m_i}}\}_{1 \leq i \leq n}$. with $b^* = \hat{K}^*$, where $\hat{K}^*, \hat{K}_1, \ldots \hat{K}_n$ were used together in the response to previous authentication query to sign message set $M_1, \ldots M_n$, and for all i, $b_i = \hat{K}_i$, and there exists j such that $m_j = \hat{K}^*$.

With nonnegligible probability, we will have correctly guessed λ such that $b^* = K^*$ for the K^* returned in the λth authentication query, and correctly guessed i^*, j^*. That means $m_j = K^* = x$, and so we can simply return $u^{m_j} = u^x = Y^x = g^{xy}$.

Case 2:

Suppose that with nonnegligible probability, the adversary produces a forgery with $b* = K_i$, where K_i was returned by a previous certification query. In this case we will show that the BB-CDH assumption does not hold for $q = 2$. Our reduction proceeds as follows:

Setup: We are given the groups G_1, G_2, G_T of order p with bilinear map e, group elements $g, X_1 = g^x, Y = g^y \in G_1$ and $h, X_2 = h^x \in G_2$, integers c_1, c_2, and the values $T_1 = g^{\frac{1}{x+c_1}}$ and $T_2 = g^{\frac{1}{x+c_2}}$. We need to produce the value g^{xy}. We make a guess i^* for which K_i will satsify the above condition. We will give the adversary parameters for the authentication scheme as follows: we choose random $z \leftarrow Z_p$, random $u \leftarrow G_1$ and random $u_j \leftarrow G_1$ for all $j \in \{1, \ldots, n\}, j \neq i^*$, and we set $u^* = Y$ and $u_{i^*} = X^z$. $params = (G_1, G_2, G_T, e, p, g, h, u, u^*, u_1, \ldots u_n)$. Next, we set secret key $sk = c_2$ and send $F(sk) = h^{sk}, u^{sk}$ to the adversary.

Queries: We must answer the adversary's signing and certification queries. We make a guess λ for which certification query's K_{i^*} the adversary will attempt to reuse.

Now, for signing queries the adversary sends message set $(m_1, \ldots m_n)$. We compute $\mathsf{Auth}(params, sk, (m_1, \ldots m_n))$ and send the result to the adversary.

For certification queries other than λ, the adversary sends key sk_A, and messages $m_2, \ldots m_n$

and we compute and return $\mathsf{Auth}(params, sk_A, (sk, m_2, \ldots m_n))$. For the λth certificaiton query we will implicitly set $K_{i^*} = x$, and we will set $K^* = c_1$. We randomly choose $K_j \leftarrow Z_p$ for $j \in \{1, \ldots, n\}, j \neq i^*$. Then we can compute the certification: $\hat{A}^* = g^{\frac{1}{sk_A + c_1}}, \hat{B}^* = h^{c_1}, \hat{C}^* = u^{*c_1}$, $\{\hat{A}_j = g^{\frac{1}{c_1 + K_j}}, \hat{B}_j = h^{K_j}, \hat{C}_j = u_j^{K_j}, \hat{D}_j = g^{\frac{1}{K_j + m_j}}\}_{1 \leq j \leq n, j \neq i^*}$, $\hat{A}_{i^*} = T_1, \hat{B}_{i^*} = X_2, \hat{C}_{i*} = X_1^z, \hat{D}_{i^*} = T_2$.

Forgery: When the adversary produces a forgery, with nonegligible probability it will be of the form $F(m_1), \ldots, F(m_n), A* = g^{\frac{1}{sk + b^*}}, B^* = h^{b^*}, C^* = u^{*b^*}, \{A_i = g^{\frac{1}{b^* + b_i}}, B_i = h^{b_i}, C_i = u_i^{b_i}, D_i = g^{\frac{1}{b_i + m_i}}\}_{1 \leq i \leq n}$. with $b* = K_i$, where K_i was returned by a previous certification query.

With nonnegligible probability, we will have correctly guessed λ such that $b^* = K_i$ for the K_i returned in the λth certification query, and correctly guessed i^*. That means $b^* = K_{i^*} = x$, and so we can simply return $B^* = u^{*x} = Y^x = g^{xy}$.

Case 3:

Suppose that with nonnegligible probability, the adversary produces a forgery where $b^* =\neq K^*$ for any K^* returned by a previous authentication query, and where $b^* =\neq K_i$ for any K_i returned by a previous certification query. Let q be an upper bound on the number of authentication or certification queries that the adversary makes. In this case we will show that the BB-HSDH assumption does not hold for this q. Our reduction proceeds as follows:

Setup: We are given the groups G_1, G_2, G_T of order p with bilinear map e, group elements $g, X_1 = g^x, v \in G_1$ and $h, X_2 = h^x \in G_2$ and the pairs $\{T_i = g^{\frac{1}{x + c_\ell}}, c_\ell\}_{1 \leq \ell \leq q}$. We need to produce a new tuple $g^{\frac{1}{x+c}}, h^c, v^c$. We will give the adversary parameters for the authentication scheme as follows: we choose random $z \leftarrow Z_p$, and random $u_j \leftarrow G_p$, and we set $u^* = v$ and $u = g^z$. $params = (G_1, G_2, G_T, e, p, g, h, u, u^*, u_1, \ldots u_n)$. We implicitly secret key $sk = x$. We send $F(sk) = X_2, X_1^z$ to the adversary.

Queries: Then we must answer the adversary's signing and certification queries.

For the γth query: If it is an authentication query: The adversary sends message set $(m_1, \ldots m_n)$. We will set $K^* = c_\gamma$ We choose random $K_j \leftarrow Z_p$ for all $j \in \{1, \ldots, n\}$. Then we can compute the authentication: $\hat{A}^* = T_i, \hat{B}^* = h^{c_\gamma}, \hat{C}^* = u^{*c_\gamma}$, $\{\hat{A}_j = g^{\frac{1}{c_\gamma + K_j}}, \hat{B}_j = h^{K_j}, \hat{C}_j = u_j^{K_j}, \hat{D}_j = g^{\frac{1}{K_j + m_j}}\}_{1 \leq j \leq n}$.

If it is a certification query: The adversary sends key sk_A, and messages $m_2, \ldots m_n$. We will set $K_1 = c_\gamma$ We choose random $K_j \leftarrow Z_p$ for all $j \in \{1, \ldots, n\}, j \neq 1$, and random $K^* \leftarrow Z_p$. Then we can compute the authentication: $\hat{A}^* = g^{\frac{1}{sk_A + K^*}}, \hat{B}^* = h^{K^*}, \hat{C}^* = u^{*K^*}$, $\{\hat{A}_j = g^{\frac{1}{K^* + K_j}}, \hat{B}_j = h^{K_j}, \hat{C}_j = u_j^{K_j}, \hat{D}_j = g^{\frac{1}{K_j + m_j}}\}_{2 \leq j \leq n}$, $\hat{A}_1 = g^{\frac{1}{K^* + c_\gamma}}, \hat{B}_1 = h^{c_\gamma}, \hat{C}_1 = u_1^{c_\gamma}, \hat{D}_1 = T_\gamma$.

Forgery: When the adversary produces a forgery, with nonegligible probability it will be of the form $F(m_1), \ldots, F(m_n), A* = g^{\frac{1}{sk + b^*}}, B^* = h^{b^*}, C^* = u^{*b^*}, \{A_i = g^{\frac{1}{b^* + b_i}}, B_i = h^{b_i}, C_i = u_i^{b_i}, D_i = g^{\frac{1}{b_i + m_i}}\}_{i = 1 \ldots n}$. where $b^* =\neq K^*$ for any K^* returned by a previous authentication query, and where $b^* =\neq K_i$ for any K_i returned by a previous certification query. That means for all $\ell \in \{1, \ldots, q\}$, $b^* \neq c_\ell$, so A^*, B^*, C^* is a fresh HSDH triple.

C.3 Proof of knowledge of an authenticator.

We need a zero-knowledge proof of knowledge of an unforgeable authenticator for messages $\vec{m} = (m_1, \ldots, m_n)$, where the first ℓ values $\vec{m}_h$ are hidden in commitments $C_{\vec{m}_h}$ and the remaining $n - \ell$ values $\vec{m}_o$ are publicly known. In our notation, this is:

$$\mathsf{NIZKPK}[\mathit{sk} \text{ in } C_{\mathit{sk}}; \vec{m}_h \text{ in } C_{\vec{m}_h}]\{(F(\mathit{sk}), F(\vec{m}_h), \mathit{auth}) : \mathsf{VerifyAuth}(\mathit{params}_A, \mathit{sk}, \vec{m}, \mathit{auth}) = 1\}.$$

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

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

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

Groth and Sahai [GS08] show that witness indistinguishable proofs like π_{sk} and π_{m_i} 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 $\pi_{\mathit{sk}}, \pi_{m_i}$ are zero-knowledge, then the composite proof π will be zero knowledge: The simulator first picks sk' and $\vec{m}_h$ 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_i} are to the same values as the original commitments C_{sk} and C_{m_i}.

D Proof that Groth-Sahai Proofs Are Randomizable

Correctness. Suppose an honest prover gets as input commitments $\{c_m\}$ and $\{d_n\}$, and a valid proof $\{\pi_i\}$ and $\{\psi_j\}$. The prover rerandomizes it and sends the verifier the new commitments $\{c'_m\}$ and $\{d'_n\}$, the new proof $\{\pi'_i\}$ and $\{\psi'_i\}$, and the original pairing product equation.

The verifier computes $c'_q \leftarrow \mu_1(a_q) \cdot \prod_{m=1}^{M} {c'}_m^{\alpha_{q,m}}$ and $d'_q \leftarrow \mu_2(b_q) \cdot \prod_{n=1}^{N} {d'}_n^{\beta_{q,n}}$. Because the prover is honest, we know that for all $q : c'_q = \mu_1(a_q) \cdot \prod_{m=1}^{M} {c'}_m^{\alpha_{q,m}} = \mu_1(a_q) \cdot \prod_{m=1}^{M} \left(c_m \prod_{i=1}^{I} u_i^{s_{m,i}}\right)^{\alpha_{q,m}}$ $= \left(\mu_1(a_q) \cdot \prod_{m=1}^{M} c_m^{\alpha_{q,m}}\right) \cdot \left(\prod_{m=1}^{M} \prod_{i=1}^{I} u_i^{s_{m,i}\alpha_{q,m}}\right) = c_q \cdot \prod_{i=1}^{I} u_i^{\hat{s}_{q,i}}$. Similarly, for all $q : d'_q = d_q \cdot \prod_{j=1}^{J} v_j^{\hat{z}_{q,j}}$.

Before the prover fully randomizes the $\{\pi'_i\}$ and $\{\psi'_j\}$, we have that

$$\pi'_i = \pi_i \cdot \prod_{q=1}^{Q} (d'_q)^{\hat{s}_{q,i}} \qquad\qquad \psi'_j = \psi_j \cdot \prod_{q=1}^{Q} (c_q)^{\hat{z}_{q,j}}.$$

Due to the results of Groth and Sahai, we know that if a prover starts with a valid proof $\{\pi_i\}$ and $\{\psi_j\}$ and multiplies each π_i by $\prod_{j=1}^{J} v_j^{t_{i,j}}$ and each ψ_j by $\prod_{i=1}^{I} u_i^{\sum_{h=1}^{H} t_h \eta_{h,i,j}} \prod_{i=1}^{I} u_i^{t_{i,j}}$ the result is still a valid proof. Therefore, all we need to show is that if the prover skips this multiplication step, that the verification formula $\prod_{q=1}^{Q} E(c'_q, d'_q) = \mu_T(t) \cdot \prod_{i=1}^{I} E(u_i, \pi'_i) \cdot \prod_{j=1}^{J} E(\psi'_j, v_j)$. holds.

$$\begin{aligned}
\prod_{q=1}^{Q} E(c'_q, d'_q) &= \prod_{q=1}^{Q} E(c_q \prod_{i=1}^{I} u_i^{\hat{s}_{q,i}}, d_q \prod_{j=1}^{J} v_j^{\hat{z}_{q,j}}) \\
&= \Big[\prod_{q=1}^{Q} E(c_q, d_q)\Big] \cdot \Big[\prod_{q=1}^{Q} E(c_q, \prod_{j=1}^{J} v_j^{\hat{z}_{q,j}})\Big] \cdot \Big[\prod_{q=1}^{Q} E(\prod_{i=1}^{I} u_i^{\hat{s}_{q,i}}, d_q)\Big] \cdot \Big[\prod_{q=1}^{Q} E(\prod_{i=1}^{I} u_i^{\hat{s}_{q,i}}, \prod_{j=1}^{J} v_j^{\hat{z}_{q,j}})\Big] \\
&= \Big[\prod_{q=1}^{Q} E(c_q, d_q)\Big] \cdot \Big[\prod_{j=1}^{J}\prod_{q=1}^{Q} E(c_q, v_j^{\hat{z}_{q,j}})\Big] \cdot \Big[\prod_{i=1}^{I}\prod_{q=1}^{Q} E(u_i^{\hat{s}_{q,i}}, d_q)\Big] \cdot \Big[\prod_{i=1}^{I}\prod_{q=1}^{Q} E(u_i^{\hat{s}_{q,i}}, \prod_{j=1}^{J} v_j^{\hat{z}_{q,j}})\Big] \\
&= \Big[\prod_{q=1}^{Q} E(c_q, d_q)\Big] \cdot \Big[\prod_{j=1}^{J} E(\prod_{q=1}^{Q} c_q^{\hat{z}_{q,j}}, v_j)\Big] \cdot \Big[\prod_{i=1}^{I} E(u_i, \prod_{q=1}^{Q} d_q^{\hat{s}_{q,i}})\Big] \cdot \Big[\prod_{i=1}^{I} E(u_i, \prod_{q=1}^{Q}\prod_{j=1}^{J} v_j^{\hat{z}_{q,j}\hat{s}_{q,i}})\Big] \\
&= \Big[\mu_T(t) \cdot \prod_{i=1}^{I} E(u_i, \pi_i) \cdot \prod_{j=1}^{J} E(\psi_j, v_j)\Big] \cdot \Big[\prod_{j=1}^{J} E(\prod_{q=1}^{Q} c_q^{\hat{z}_{q,j}}, v_j)\Big] \cdot \Big[\prod_{i=1}^{I} E(u_i, \prod_{q=1}^{Q}(d_q \prod_{j=1}^{J} v_j^{\hat{z}_{q,j}})^{\hat{s}_{q,i}})\Big] \\
&= \mu_T(t) \cdot \Big[\prod_{i=1}^{I} E(u_i, \pi_i) \prod_{i=1}^{I} E(u_i, \prod_{q=1}^{Q}(d_q \prod_{j=1}^{J} v_j^{\hat{z}_{q,j}})^{\hat{s}_{q,i}})\Big] \cdot \Big[\prod_{j=1}^{J} E(\psi_j, v_j) \prod_{j=1}^{J} E(\prod_{q=1}^{Q} c_q^{\hat{z}_{q,j}}, v_j)\Big] \\
&= \mu_T(t) \cdot \Big[\prod_{i=1}^{I} E(u_i, \pi_i \prod_{q=1}^{Q} d_q'^{\hat{s}_{q,i}})\Big] \cdot \Big[\prod_{j=1}^{J} E(\psi_j \prod_{q=1}^{Q} c_q^{\hat{z}_{q,j}}, v_j)\Big] \\
&= \mu_T(t) \cdot \prod_{i=1}^{I} E(u_i, \pi'_i) \cdot \prod_{j=1}^{J} E(\psi'_j, v_j)
\end{aligned}$$

Thus we see that the verification equation holds.

Malleability. The same argument used for correctness also shows that when commitments are considered part of the instance, the instance can be mauled by randomizing these commitments.

Extractability. The Groth-Sahai extractor works by using the trapdoor to "decrypt" the commitments. Recall that a commitment is of the form $c = \mathsf{Commit}(x, (r_1, \ldots, r_I)) = x \cdot \prod_{i=1}^{I} u_i^{r_i}$ for an *arbitrary* vector $(r_1, \ldots, r_I)$. When we rerandomize c, we compute

$$c' \leftarrow c \cdot \prod_{i=1}^{I} u_i^{s_i} = x \cdot \prod_{i=1}^{I} u_i^{r_i} \cdot \prod_{i=1}^{I} u_i^{s_i} = x \cdot \prod_{i=1}^{I} u_i^{r_i + s_i} .$$

Thus, we still have a commitment to x.

Randomness. First we consider the randomization of commitments. As we saw above, each new commitment is set to $c' \leftarrow x \cdot \prod_{i=1}^{I} u_i^{r_i+s_i}$. The randomness of c' depends on how the s_i are chosen. If the s_i are chosen at random, then c' is a randomly chosen commitment for x. If the s_i are not chosen at random (especially if they are all set to 0!), then we have no such guarantees.

Now we argue about the randomization of proofs. When the prover computes $\pi'_i \leftarrow \pi_i \cdot \prod_{q=1}^{Q}(d'_q)^{\hat{s}_{q,i}}$ and $\psi'_j \leftarrow \psi_j \cdot \prod_{q=1}^{Q}(c_q)^{\hat{z}_{q,j}}$, the result is a valid proof (as we demonstrated above).

Then the prover multiplies the $\{\pi'_i\}$ and $\{\psi'_i\}$ by a certain factor. Groth and Sahai show that the result is a randomly chosen proof from the space of all valid proofs, given that the commitments and pairing product equation are fixed. Since, as we showed above, the commitments are completely rerandomized, the result is a randomly chosen proof given a fixed solution $\{x_m\}, \{y_n\}$ to a particular pairing product equation.

E 2PC Protocol for Creating a NIZKPK of an Authenticator

Our construction of delegatable credentials is secure for any 2PC protocol that securely generates a NIZKPK of an authenticator. In this section we provide an efficient instantiation of such a protocol. However, we do not know how to built an efficient protocol that does not use either the simulation or the extraction trapdoor in the proof of security. As this is not covered by the standard definition of secure 2PC we first define trapdoor-secure two-party computation. We then argue that our delegatable credential system remain secure if the 2PC protocol is replaced by a trapdoor-secure two-party protocol. We then describe and prove the concrete protocol that we propose.

E.1 Trapdoor-Secure Two-Party Computation

In a two-party protocol, we have A interacting with B; they both receive the same public input p; A has private input a and B has private input b. This is written $A(p, a) \leftrightarrow B(p, b)$. Recall the standard definition for what it means for $A(p, a) \leftrightarrow B(p, b)$ to securely compute a function $f(p, a, b)$ (see, for example, Lindell and Pinkas [LP07]).

In this paper, the protocols take advantage of the fact that there are appropriately generated public parameters *params*, which is one of the public inputs to the parties. The notion of $A((\mathit{params}, p), a) \leftrightarrow B((\mathit{params}, p), b)$ securely computing the function $f(\mathit{params}, p, a, b)$ for $\mathit{params} \leftarrow \mathsf{Setup}(1^k)$ is a generalization of the standard secure 2PC notion to the common parameters model.

Let $\mathsf{Trapdoor}$ be any function (not necessarily efficiently computable). We want to weaken the notion of security of 2PC in the common parameters model by requiring that security is preserved only when the simulator has additional information about *params*, namely $\mathsf{Trapdoor}(\mathit{params})$. This is captured as follows: $A((\mathit{params}, p), a) \leftrightarrow B((\mathit{params}, p), b)$ $\mathsf{Trapdoor}(\cdot)$-securely compute the function $f(\mathit{params}, p, a, b)$ for $\mathit{params} \leftarrow \mathsf{Setup}(1^k)$ if $A'((\mathit{params}, \mathsf{Trapdoor}(\mathit{params}), p), a) \leftrightarrow B'((\mathit{params}, \mathsf{Trapdoor}(\mathit{params}), p), b)$ securely compute the same f, where A' and B' ignore $\mathsf{Trapdoor}(\mathit{params})$ and run A and B respectively.

Lemma 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 construction in Section 3.2 constitutes a secure anonymous delegatable credential scheme.*

Proof. In our reduction we only use the 2PC simulator in situations in which the trapdoors are available. The proof of security of the delegatable credential system thus carries through without changes. □

E.2 An efficient two-party computation protocol for computing a NIZKPK of an authentication tag

An efficient two-party computation protocol for computing a non-interactive zero-knowledge proof of knowledge (NIZKPK) of an authentication tag is a protocol between a user and a issuer. The user's private input is the vector $\vec{m_h}$ of committed messages that are to be authenticated and opening vector $open_{\vec{m_h}}$. The issuer's private input is secret key sk and $open_{sk}$. Both parties agree on the public input: the public parameters $params_{DC}$ for the proof system and the authentication scheme, the commitments $C_{\vec{m_h}}$, the public messages $\vec{m_o}$, and C_{sk}. The commitments are double commitments as described in Appendix C.3. We also define a commitment to a vector of messages to be the list of commitments to its elements. The user's output of the protocol is a proof $\pi \leftarrow \mathsf{NIZKPK}[sk \text{ in } C_{sk}; \vec{m_h} \text{ in } \mathsf{Commit}(\vec{m_h}, \vec{0})]\{(F(sk), F(\vec{m_h}), auth) :$ $\mathsf{VerifyAuth}(params_A, sk, \vec{m_h}, \vec{m_o},$
$auth) = 1\}$ as described in Appendix C.3. The issuer only learns about success or failure of the protocol and outputs nothing. If $(\vec{m_h}, open_{\vec{m_h}})$, or $(sk, open_{sk})$ are inconsistend with $C_{\vec{m_h}}, C_{sk}$ respectively, the functionality reports failure; otherwise it returns a correctly formed random proof π.

1. The user proves that she knows $\vec{m_h}$ and $open_{\vec{m_h}}$ for $C_{\vec{m_h}}$. The issuer aborts if the proof fails.

2. The issuer does a proof of knowledge of sk and $open_{sk}$ for C_{sk}. The user aborts if the proof fails.

3. The issuer chooses random $K^*, K_1, \dots K_n \leftarrow Z_p$. He computes a partial authentication tag $auth' = (g^{\frac{1}{sk+K^*}}, h^{K^*}, u^{*K^*}, \{g^{\frac{1}{K^*+K_i}}, h^{K_i}, u_i^{K_i}\}_{1\le i\le n})$.

4. Then the issuer computes a fresh commitment C'_{sk} to sk and creates a NIZKPK π' for a partial authenticator $auth'$:

$$\begin{aligned}\pi_2 \leftarrow \mathsf{NIZKPK}_{GS}[h^{sk} \text{ in } {C'}^{(1)}_{sk}; u^{sk} \text{ in } {C'}^{(2)}_{sk}; B_i \text{ in } \mathsf{GSCommit}(B_i, 0)]&\\ \{(F(sk), (A^*, B^*, C^*, \{A_i, B_i, C_i\}_{1\le i\le n}, \{D_i\}_{\ell+1\le i\le n}), h^\theta) :&\\ e(u^{sk}/u^{sk'}, h^\theta) = 1 \wedge e(g, h^\theta) = e(g,h) \wedge e(h^{sk}, u) = e(h, u^{sk}) \wedge&\\ e(A^*, h^{sk}B^*) = e(g,h) \wedge e(B^*, u^*) = e(h, C^*) \wedge&\\ \{e(A_i, B^*B_i) = e(g,h) \wedge e(B_i, u_i) = e(h, C_i)\}_{1\le i\le n} \wedge&\\ \{e(D_i, B_i h^{m_i}) = e(g,h)\}_{\ell+1\le i\le n}\}.&\end{aligned}$$

   Note that all authenticator values use random commitments, except for the B_i values, which are committed using 0 openings. This means that $B_i = h^{K_i}$ can be learned from the proof.

5. The issuer sends π' to the user.

6. The user checks the proof and aborts if the verification fails. Otherwise, she runs ℓ instances of an efficient two-party computation protocol with the issuer. On public input C_{m_i} and secret input m_i by the user and K_i by the issuer $(1 \le i \le \ell)$ the protocol computes the missing $D_i = g^{\frac{1}{K_i+m_i}}$. The output is obtained only by the user. We give an efficient implementation of this 2PC protocol using additively homomorphic encryption in Section 3.3.

7. The user checks that the D_i were computed for the correct K_i using the B_i from the proof π', computes commitments $C^{(1)}_{m_i} = \mathsf{GSCommit}(h^{m_i}, 0)$, $C^{(2)}_{m_i} = \mathsf{GSCommit}(u^{m_i}, 0)$ and fresh commitments ${C'}^{(1)}_{m_i}, {C'}^{(2)}_{m_i}$ for each m_i and computes the proof $\pi'' \leftarrow \mathsf{NIPK}_{GS}[h^{m_1} \text{ in } {C'}^{(1)}_{m_1}; u^{m_1} \text{ in } {C'}^{(2)}_{m_1}; \ldots; h^{m_\ell} \text{ in } {C'}^{(1)}_{m_\ell}; u^{m_\ell} \text{ in } {C'}^{(2)}_{m_\ell}); B_i \text{ in } \mathsf{GSCommit}(B_i, 0)]\{(F(\vec{m}_h) = (\{F(m_i)\}_{1 \leq i \leq \ell}), \{B_i, D_i\}_{1 \leq i \leq \ell}) : \{e(D_i, B_i h^{m_i}) = e(g, h) \wedge e(h^{m_i}, u) = e(h, u^{m_i})\}_{1 \leq i \leq \ell}\}$, as well as the zero-knowledge proofs of equality of committed values in $C^{(1)}_{m_i}$ and ${C'}^{(1)}_{m_i}$, $\pi_{m_1}, \ldots \pi_{m_\ell}$.

8. Finally the user computes $\pi = \pi' \circ \pi'' \circ \pi_{m_1} \circ \ldots \circ \pi_{m_\ell}$ and randomizes the combined proof with opening values $\mathit{open}_{sk} = 0$ and $\mathit{open}_{\vec{m}_h} = 0$, and all other openings chosen at random. The resulting proof π' is the user's output.

Theorem 4 *The above construction is a secure two-party computation for the parameters and the trapdoor generated by* ExtSetup *and* SimSetup *if the underlying proof system and the homomorphic encryption scheme are secure.*

Proof. We assume the existance of a secure two party protocol for computing $g^{1/(sk+m)}$, i.e., $g^{1/(K_i+m_i)}$. We need to do a simulation for both ExtSetup and SimSetup.
ExtSetup: A simulator that simulates a dishonest user $\tilde{\mathcal{U}}$ proceeds as follows:

- The simulator runs $\tilde{\mathcal{U}}$ as a blackbox. He uses the extractor of the proof of knowledge to obtain $\vec{m}$ and $\mathit{open}_{\vec{m}}$. If they are consistent with $C_{\vec{m}}$ he sends them to the ideal functionality. He aborts otherwise.

- Next, the simulator uses the zero-knowledge simulator to simulate his own proof of knowledge. The ideal functionality returns a NIZKPK π of an authenticator.

- The simulator uses the trapdoor to extract the authenticator. He uses a subset of the values to build proof π'.

- Next the simulator uses the simulator of the 2PC subprotocol for weak BB signatures to finish the simulation. Note that an honest issuer never outputs anything, even in case of protocol failure.

- The simulator outputs whatever $\tilde{\mathcal{U}}$ outputs.

As the simulation of the environment for $\tilde{\mathcal{U}}$ up to the 2PC subprotocol is perfect, $\tilde{\mathcal{U}}$ will behave exactly in the same way as in the real world. The simulatability of the 2PC subprotocol complets the proof.

A simulator that simulates a dishonest issuer $\tilde{\mathcal{I}}$ proceeds as follows:

- The simulator runs $\tilde{\mathcal{I}}$ as a black box. The simulator uses the zero-knowledge simulator to simulate the proof of knowledge.

- The simulator uses the extractor of the proof of knowledge. If they are inconsistent with $C_{\vec{m}}$, he aborts.

- Now the malicious issuer provides additional values, that the simulator checks in the same way as the user would.

- Then it runs the simulator of the 2PC with the issuer. If the checks or the 2PC fail, he sends $\perp$ to the ideal functionality to force the ideal user to abort. If they pass, he sends $\vec{m}$ and $open_{\vec{m}}$ to the ideal functionality.
- The simulator outputs whatever $\tilde{\mathcal{I}}$ outputs.

The simulator aborts, when an honest user would abort, and both the ideal world user as well as the real world user output a random proof. This completes the proof for the extraction parameters. $\mathsf{SimSetup}$: A simulator that simulates a dishonest user $\tilde{\mathcal{U}}$ proceeds as follows:

- The simulator runs $\tilde{\mathcal{U}}$ as a blackbox. He uses the extractor of the proof of knowledge to obtain $\vec{m}$ and $open_{\vec{m}}$. If the values are consistent with $C_{\vec{m}}$, he sends $\vec{m}$ and $open_{\vec{m}}$ to the ideal functionality; otherwise he aborts.
- Next, the simulator uses the zero-knowledge simulator to simulate his own proof of knowledge. The ideal functionality returns a NIZKPK π of an authenticator.
- Now, however, we cannot use the returned proof π for the simulation. Luckily, the simulation trapdoor allows us to fake the proof of equality between the commitments C_{sk} and C'_{sk}, and the simulator can use a new random key sk' to compute the partial authenticator $auth'$ and the corresponding proof.
- The rest of the proof proceeds as for $\mathsf{ExtSetup}$.

The simulation for the dishonest issuer $\tilde{\mathcal{I}}$ is the same as for $\mathsf{ExtSetup}$.

E.3 Security of protocol for computing $g^{1/(sk+m)}$:

We show that the protocol presented in Section 3.3 is secure. We first simulate a malicious user. Recall that we must define a simulator $\mathcal{S}$ which gets as input parameters $params$, commitment C to the message to be signed, and a signature σ on that message under secret key sk from the ideal functionality, and must impersonate an honest issuer without knowing sk. Consider the following simulator:

1. $\mathcal{S}$ honestly generates a key pair $(sk_{hom}, pk_{hom}) \leftarrow \mathsf{Keygen}(1^k)$. It computes $e_1 = \mathsf{Enc}(pk_{hom}, 0)$. It sends pk_{hom}, e_1 to the adversary.
2. $\mathcal{S}$ receives e_2 from the adversary.
3. $\mathcal{S}$ acts as the verifier for the proof that e_2 was computed correctly. He runs the proof of knowledge extraction algorithm and extracts r_1 (note that this might include rewinding the adversary, but not farther than the beginning of this step). Finally it computes $\sigma^* = \sigma^{1/r_1}$ ands sends it to the adversary.

Now we show that this $\mathcal{S}$ does a successful simulation: Consider the following series of games:

- In the first game, $sk, pk, m, open, state, C$ are generated as in the definition of the protocol, and then the adversary $\mathcal{A}_2(state)$ interacts with the real world party as defined above.

- In the second game, $sk, pk, m, open, state, C$ are generated the same way, but now $\mathcal{A}_2(state)$ interacts with a $\mathcal{S}'$, which behaves as the real protocol for steps 1 and 2, but then behaves as $\mathcal{S}$ for step 3. The only difference then is that this simulator extracts r_1 from the proof, and uses r_1 and σ to form σ^*. Note that if the proof is sound, then this σ^* will be identical to that produced in the previous game. Thus this is indistinguishable from the previous game by the extraction property of the ZK proof system.

- In the last game, $sk, pk, m, open, state, C$ are generated the same way, and then $\mathcal{A}_2(state)$ interacts with $\mathcal{S}$. This differs from the second game only in that the initial encryption e_1 is generated by encrypting 0. Thus, this is indistinguishable from the second game by the security of the encryption scheme.

Since the first game is indistinguishable from the third, the probability that the adversary $\mathcal{A}_2$ can output 1 in each game can differ only negligibly. Thus, the simulation is successful.

Next, we consider a malicious signer. Recall that we must define a simulator $\mathcal{S}$ which gets as input parameters *params*, the public key *pk* of the signer, and a commitment C to the message to be signed, and must impersonate an honest user without knowing the message contained in the commitment m. Consider the following simulator:

1. $\mathcal{S}$ receives pk_{hom}, e_1 from the adversary.

2. $\mathcal{S}$ chooses a random value $t \leftarrow [0, 2^k p^2]$. It computes $e_2 = e_1 \oplus \mathsf{Enc}(pk_{hom}, t)$, and sends e_2 to the adversary.

3. $\mathcal{S}$ uses the simulator for the zero knowledge proof to interact with the adversary. (details?)

4. $\mathcal{S}$ receives σ^* from the adversary and checks it is a valid signature on $m' = t \bmod p$.

Now we show that this $\mathcal{S}$ does a successful simulation: Consider the following series of games:

- In the first game, $pk, m, open, state, C$ are generated as in the real protocol, and then the adversary $\mathcal{A}_2(state)$ interacts with an honest user as defined above.

- In the second game, $sk, pk, m, open, state, C$ are generated the same way, but now $\mathcal{A}_2(state)$ interacts with a $\mathcal{S}'$, which behaves as in the real protocol for steps 1 and 2, but then behaves as $\mathcal{S}$ for step 3. The only difference then is that here we use the zero-knowledge simulator to do the interactive proof. Thus this is indistinguishable from the previous game by the zero knowledge property of the ZK proof system.

- In the last game, $sk, pk, m, open, state, C$ are generated the same way, and then $\mathcal{A}_2(state)$ interacts with $\mathcal{S}$. This differs from the second game only in that e_2 is generated by computing $e_1 \oplus enc(t)$ rather than by computing $e_2 = ((e_1 \oplus enc(m)) * r_1) \oplus \mathsf{Enc}(r_2 p)$. As t is chosen from $[0, 2^k p]$ and e_1 encrypts a value from Zp, the value encrypted in $e_1 \oplus enc(t)$ will be distributed statistically close to the uniform distribution over $[0, 2^k p]$. Similarly, the value encrypted in $e_2 = ((e_1 \oplus enc(m)) * r_1) \oplus \mathsf{Enc}(r_2 p)$ will be distributed statistically close to the uniform distribution over $[0, 2^k p]$ and hence this game is indistinguishable from the previous one.

Since the first game is indistinguishable from the third, the probability that the adversary $\mathcal{A}_2$ can output 1 in each game can differ only negligibly. □

The following theorem states that it is possible to replace the standard 2PC protocol with the trapdoor secure protocol described above without compromising security.

E.4 Efficient Implementation using Pallier Encryption

We show how the two party computation protocol for computing $g^{(sk+m)}$ that was presented in Appendix E.2 can be implemented efficiently using the Pallier encryption scheme.

For simplicity of this exposition, we assume that there as an RSA modulus $\mathfrak{n}$ available such that neither the prover or the verifier know its factors. Moreover assume further that $\mathfrak{g}$ and $\mathfrak{h}$ from $Z_{\mathfrak{n}}^*$ are available such that $\log_{\mathfrak{g}} \mathfrak{h}$ is unknown and that $\mathfrak{g} \in \langle \mathfrak{h} \rangle$. (Alternatively the prover and the verifier can generate their own modules and then use in the protocol, e.g., as proposed by Bangerter et al. [BCM05].)

Assume that the commitment of the recipient's secret key be $comm = \mu_1(g)^m \prod u_i^{s_i}$. We will use the Paillier encryption scheme. The user and the issuer run the following protocol:

1. The issuer generates an n RSA modulus of size at least $2^{3k}p^2$, where k is our security parameter. Further let $\mathsf{h} = \mathsf{n} + 1$ and g be an element of order $\phi(\mathsf{n})$ modulo n^2. Next, the issuer computes $e_1 = \mathsf{h}^{\mathsf{n}/2+sk}\mathsf{g}^r \bmod \mathsf{n}^2$ for a random $r \in Z_{\phi(\mathsf{n})}$ and $\mathfrak{c} = \mathfrak{g}^{sk}\mathfrak{h}^{r'} \bmod \mathfrak{n}$ for a random $r \in Z_{\phi\mathfrak{n}}$.

   Now the signer and the recipient run the following protocol with each other:

   $$PK\{(sk, r, r') : e_1/\mathsf{h}^{\mathsf{n}/2} = \mathsf{h}^{sk}\mathsf{g}^r \pmod{\mathsf{n}^2} \ \wedge\ \mathfrak{c} = \mathfrak{g}^{sk}\mathfrak{h}^{r'} \pmod{\mathfrak{n}} \ \wedge\ sk \in [-p2^{160}, p2^{160}]\}$$

2. The user chooses $r_1 \leftarrow Z_p$ and $r_2 \leftarrow \{0, \ldots 2^k p\}$ and computes

   $$e_2 = (e_1/\mathsf{h}^{\mathsf{n}/2)})^{r_1}\mathsf{h}^{(\mathsf{n}/2+m)r_1+r_2 p}\mathsf{g}^{\bar{r}} \bmod \mathsf{n}^2$$

   as well as the commitment $\mathfrak{c}' = \mathfrak{g}^m\mathfrak{h}^{\bar{r}} \bmod \mathfrak{n}$, with $\bar{r}$ chosen randomly from $[0, \mathfrak{n}2^k]$.

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, and that r_1, r_2 are in the appropriate ranges:

   $$PK\{(m, r_1, r_2, s_i, m', u, \bar{r}) : e_2/\mathsf{h}^{\mathsf{n}/2} = (e_1/\mathsf{h}^{\mathsf{n}/2})^{r_1}\mathsf{h}^{m'}(\mathsf{h}^p)^{r_2}\mathsf{g}^{\bar{r}} \pmod{\mathsf{n}^2} \wedge \mathfrak{c}' = \mathfrak{g}^m\mathfrak{h}^{\bar{r}} \pmod{\mathfrak{n}}$$
   $$\wedge\ 1 = \mathfrak{c}'^{r_1}(1/\mathfrak{g})^{m'}\mathfrak{h}^u \ \wedge\ comm = \mu_1(g)^m \prod u_i^{s_i} \ \wedge\ m, r_1 \in [-p2^{160}, p2^{160}]\}$$

4. The issuer decrypts $x = \mathsf{Dec}(e_2) - n/2$, 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 signature on m.

We see that issuer and the user have both to perform about 10 exponentiations (in different groups). The security of the PK protocol follows straight forward from known works such as [CS03]).