Posted as Report 2013/516 on 19 August 2013; revised 31 October 2013.

Algebraic MACs and Keyed-Verification Anonymous Credentials

Melissa Chase
Microsoft Research Redmond
melissac@microsoft.com

Sarah Meiklejohn*
UC San Diego
smeiklej@cs.ucsd.edu

Greg Zaverucha
Microsoft Research Redmond
gregz@microsoft.com

Abstract

We consider the problem of constructing anonymous credentials for use in a setting where the issuer of credentials is also the verifier, or more generally where the issuer and verifier have a shared key. In this setting we can use message authentication codes (MACs) instead of public key signatures as the basis for the credential system.

To this end, we construct two algebraic MACs in prime-order groups, along with efficient protocols for issuing credentials, asserting possession a credential, and proving statements about hidden attributes (e.g., the age of the credential owner). We prove the security of the first scheme in the generic group model, and prove the security of the second scheme — using a dual-system-based approach — under decisional Diffie-Hellman (DDH). Our MACs are of independent interest, as they are the only uf-cmva-secure MACs with efficient proofs of knowledge.

Finally, we compare the efficiency of our new systems to two existing constructions of anonymous credentials: U-Prove and Idemix. We show that the performance of the new schemes is competitive with U-Prove (which is not provably secure, whereas ours is based on DDH), and many times faster than Idemix.

1 Introduction

An anonymous credentials system [12, 8, 10] allows for a landscape in which users can be known in different context by different pseudonyms. For example, a user Alice might be known to Bob under one pseudonym nym, and to Carol under a different pseudonym nym'. Her behavior under these two pseudonyms should be *unlinkable*, meaning no one can discern that the two pseudonyms belong to the same user, yet she should be able to prove possession of a credential issued to one given pseudonym to any other user, without revealing the pseudonym (e.g., if Carol issued a credential to nym', Alice should nevertheless be able to prove to Bob — using nym — that she is in possession of the credential). Beyond proving basic possession of a credential, Alice may want to furthermore prove that it captures certain *attributes* about her (e.g., her age).

Many of the potential applications of anonymous credentials involve authentication. For example, a transit authority might issue monthly passes and then check for possession of such a pass when the user boards a bus; similarly, a university might issue keycards to access certain buildings, and then require a user to swipe his card upon entering one of these buildings. For each of these situations, the authority needs to know only that an authorized user is gaining entry, yet current implementations of such access systems allow the authority to learn the patterns of each individual participants (e.g., who

*Work done as an intern at Microsoft Research Redmond

is entering which building and when). Anonymous credentials present a solution that simultaneously preserves the integrity of the system through an *unforgeability* guarantee that a user cannot prove possession of credentials he wasn't issued (e.g., cannot gain access to a building without having been granted access) but also preserves the *anonymity* of the individual participants. If the anonymity is preserved throughout multiple presentations of the credential (e.g., each time a user boards the bus), it is said to satisfy *multi-show unlinkability*.

Traditionally, the models for anonymous credentials have assumed that a user must be prepared to prove possession of his credentials to many other participants in the system; as such, all existing constructions are built on top of public-key primitives such as digital signatures. In both of the above examples, however, the issuer and verifier are in fact the same entity; e.g., the transit authority both sells (i.e., issues) monthly passes and then verifies them when users enter the system. In these scenarios, and more generally in any setting in which the party controlling access to a resource also manages the accounts of authorized parties, constructions can take advantage of this symmetry to use symmetric-key primitives — which are typically significantly more efficient than their public-key counterparts — to construct a *keyed-verification* credential.

If we define keyed-verification credentials as allowing the issuer and verifier to share access to some secret key, then symmetric-key primitives can be further adopted beyond the setting in which the issuer and verifier are the exact same entity. As long as a user does not need to authenticate himself to *any* other user in the system, the issuer can share a separate secret key with each verifier, and then issue credentials specific to each of these verifiers. In a purely non-interactive setting this might require the issuer to provide credentials for each verifier all at once (which might become fairly unattractive beyond a small number of verifiers), but in a more online setting the user might request credentials from the issuer as needed.

More generally, it is also possible to translate a publicly verifiable credential into a more efficient keyed-verification credential with the same attributes and functionality. Another use case is thus as follows: when a user wishes to interact with a new verifier, he first enrolls with the verifier by presenting a traditional credential; the verifier checks this credential, then issues a new credential that only he can verify. When the user returns, the efficient keyed-verification credential is used, and it is impossible to link this use with previous presentations of the credential or with the user's initial enrollment. Translating credentials in this manner provides the appealing trade-off that public verifiability is still possible when necessary, but credential use becomes more efficient with repeat verifiers.

Our contributions. In this paper, we introduce keyed-verification credentials, which formalize the intuition outlined above. By using message authentication codes (MACs) in place of more traditional public-key signatures, we show that we can achieve performance improvements over existing constructions of anonymous credentials.

In order to integrate nicely with primitives such as zero-knowledge proofs (which are typically needed to construct anonymous credentials), we require an *algebraic MAC*, meaning a MAC constructed using group operations rather than block ciphers or hash functions. In Section 3, we present two such MACs, both constructed in prime-order groups. The first, $\mathsf{MAC}_{\mathsf{GGM}}$, is a generalization of a MAC presented by Dodis et al. [16]. While Dodis et al. show that this MAC provides only a very weak notion of security under the DDH assumption, we generalize the scheme to allow for blocks of messages, and then prove it satisfies the standard notion (uf-cmva security) of MAC unforgeability, albeit in the generic group model. The second, $\mathsf{MAC}_{\mathsf{DDH}}$, is uf-cmva secure under the DDH assumption.

Both MACs are of potential independent interest, as they avoid techniques such as collision-resistant hash functions and bit-wise decompositions, which have often been relied upon to construct efficient MACs. $\mathsf{MAC}_{\mathsf{GGM}}$ is additionally quite efficient, while $\mathsf{MAC}_{\mathsf{DDH}}$ is based on a mild assumption (and still quite efficient). Interestingly, the proof of security for $\mathsf{MAC}_{\mathsf{DDH}}$ follows the dual-system

approach introduced by Waters [31], which makes it (to the best of our knowledge) the first application of this technique outside of the pairing-based setting.

Next, in Section 4, we present keyed-verification credentials. We first present a formal security model, and then present two constructions, one based on each of our MACs. For each construction, we describe how to efficiently issue and prove possession of credentials. Our constructions consider credentials for various attributes to reflect situations with a complex access policy (e.g., in the transit setting, a monthly pass could encode an expiration date, the area of usage, etc.); allow for *blind* issuance of credentials, in which one or more of the attributes may remain hidden from the issuer (e.g., a senior citizen might need to provide their date of birth or other sensitive information in order to receive a discount transit pass); and allow for presentation of credentials with attributes satisfying a given statement. We then prove the security of our $\mathsf{MAC}_{\mathsf{DDH}}$-based construction under DDH.

Finally, in Section 5, we provide a detailed efficiency comparison of our new keyed-verification credentials to U-Prove [7, 24] and Idemix [9, 21], the two most efficient anonymous credential schemes to date. Our comparison indicates that, depending on the parameters of the presentation, our new constructions both have the same or slightly higher cost when compared to U-Prove, and are always many times faster than Idemix (by our estimates, anywhere between 4 and 16 times faster).

Related work. The state of the art in MACs based on number-theoretic assumptions are the schemes by Dodis et al. [16]. (Their paper also contains a survey of prior work. Of the nine MACs presented in [16], all either (1) satisfy a weaker security notion than uf-cmva, or (2) use hash functions or bitwise decomposition of the message, thus preventing an efficient proof of knowledge. Since our keyed-verification credential constructions require both of these properties, we cannot use these existing MACs directly. Section 3 describes the differences in more detail.

Anonymous credentials were introduced by Chaum [12] as a way to provide individuals more control over the disclosure of personal data. U-Prove [24] is a credential system constructed from a blind version of Schnorr signatures [7]. It is defined in a prime-order group, and is thus very computationally efficient. A U-Prove credential is constructed as a number of tokens, where each token may be used once unlinkably, so the size of U-Prove credentials is linear in the number of unlinkable uses. A recent paper of Baldimtsi and Lysyanskaya [2] presents a construction with efficiency similar to U-Prove (and similarly with no multi-show unlinkability), but with a security proof assuming the DDH assumption in the random oracle model (U-Prove does not have a formal proof of security).

Idemix [21] is based on the Camenisch-Lysyanskaya [10] signature scheme (CL signatures). In terms of performance, Idemix and U-Prove credentials have an opposite trade-off: Idemix credentials have constant size, but are considerably more expensive to present. The computational cost is increased because the underlying signature scheme is constructed in a group where the strong RSA problem (SRSA) is hard. While there are no guidelines for choosing parameters for the strong RSA problem, they must be at least as large as RSA parameters, e.g., 3072 bits for 128-bit security.[1] With multiple attributes, and advanced presentation proof predicates, this cost quickly becomes too high for lightweight provers such as smartcards [5].

There are also versions of the CL signature scheme defined in bilinear groups [1, 10], and Belenkiy et al. [3] construct anonymous credentials that support delegation. However, the algorithms in this setting are considerably more expensive, and the computational costs of creating a presentation proof and verifying it are still significantly greater than in U-Prove. The standardization of cryptographic schemes based on SRSA and bilinear groups also lags further behind prime-order groups, presenting another hurdle to deployment.

[1] Note the optimizations that apply to the RSA signing operation are only available to the issuer in Idemix, not the user or verifiers, as in that case the group order is unknown and exponents must be large to satisfy privacy requirements.

Given the trade-offs of each system, our design goal is a credential system with the strengths of U-Prove (efficient presentation and standard parameters), and those of Idemix (constant credential size and multi-show unlinkability).

2 Preliminaries

Notation. We use the notation $x \in_R X$ or $x \xleftarrow{\$} X$ to mean x is chosen uniformly at random from the set X. The notation $\{x_i\}_1^n$, $\sum_1^n x_i$, and $\prod_1^n x_i$ are shorthand for $\{x_i\}_{i=1}^n$, $\sum_{i=1}^n x_i$, and $\prod_{i=1}^n x_i$ respectively. This shorthand is used only when the set, sum or product has a single index. The notation $\vec{x}$ is used to denote the vector $(x_1, \ldots, x_n)$, where n will be clear from the context.

We use games in the definition of MAC security and in proofs. A game G has a MAIN procedure whose output is the output of the game. $\Pr[\mathsf{G}]$ denotes the probability that this output is 1.

Zero-Knowledge Proofs. The protocols that comprise our credential system make use of zero-knowledge (ZK) proofs to prove knowledge of, and relations between, discrete logarithms. We abstract these protocols with a notation introduced by Camenisch and Stadler [11]. Proofs are expressed with the notation

$$\mathrm{PK}\{(x, y, \ldots) : \textit{statements} \text{ about } x, y, \ldots\}$$

where $(x, y, \ldots)$ are secrets (discrete logarithms) which satisfy *statements*. The prover is asserting knowledge of $(x, y, \ldots)$, and all other values in the protocol are public.

There are many choices to implement these protocols, especially since the types of statements required by our protocols are relatively simple (knowledge of a representation and proof of logarithm equality). In particular, all the statements we prove can be captured by efficient sigma protocols.

For our application, we need a proof system that is zero knowledge and satisfies a weak form of online extraction [18]. We propose two approaches to instantiate the proof system. The first is to use the Damgård protocol [15], which converts any sigma protocol into a three-round interactive zero-knowledge proof of knowledge secure under concurrent composition. This protocol requires trusted parameters, but this restriction can be omitted in the random oracle model. The second option is to make the assumption that Fiat-Shamir based proofs [17] in the random oracle model satisfy the required extraction property. For more discussion, see Appendix D.

Parameter Generation. Some of the parameters of our constructions include a group element h, chosen such that $\log_g h$ is unknown, where g is a generator of the group. In practice, this can be done by deterministically deriving h from arbitrary public information using a cryptographic hash function. All protocol participants may then verify that h was derived correctly by repeating the derivation process. One such derivation procedure is specified in [24]. Formally, we model this as a trusted setup algorithm which generates g, h where $\log_g h$ is unknown to all parties.

Cryptographic Assumptions. The *decisional Diffie-Hellman problem* (DDH) is the following: Let G be a cyclic group of prime order p with generator g and let $a, b, c \in_R \mathbb{F}_p$; given $(A = g^a, B = g^b, C) \in G^3$, determine whether $C = g^{ab}$ or $C = g^c$. The DDH assumption is that this problem is intractable for all polynomial time adversaries.

For some of our constructions we will also give security results in the *generic group model* (GGM). Intractability results in this model essentially mean that problems are intractable provided the adversary only performs a series of group operations. The GGM was first used by Shoup to prove lower bounds on DDH and related problems [6, 27].

Concrete examples of groups that are thought to satisfy these assumptions are certain elliptic curve groups over $\mathbb{F}_p$, such as those standardized by NIST in [23].

3 MACs in Prime-Order Groups

In this section we present two MACs constructed using a cyclic group of prime order. Both schemes use the same system parameters, created with the following algorithm.

$\mathsf{Setup}(1^k)$**.** Choose a group G with order p, where p is a k-bit prime. Let g and h be generators of G such that $\log_g h$ is unknown. The system parameters are $params := (G, p, g, h)$.

In addition to the Setup algorithm, MACs have a key generation function KeyGen, a MAC function MAC that produces an authentication tag on a message, and a verify function Verify that verifies a tag is valid with respect to a key and message. While we do not include it as an explicit parameter, the MAC and Verify functions are assumed to have *params*. This could easily be captured by including it in the secret key; we omit it to simplify the descriptions. The message space of both schemes is $\mathbb{F}_p^n$, where $n > 0$ is a parameter.

We say that $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{MAC}, \mathsf{Verify})$ is a secure MAC if it is existentially unforgeable under chosen message attack, given a verification oracle (defined as uf-cmva in [16]). We augment the definition slightly to guarantee security even when the signer publishes some parameters *iparams* associated with his secret key. In our application to anonymous credentials, *iparams* are the issuer parameters and we use them to implement an efficient presentation protocol.

Definition 1 (uf-cmva security)**.** *For a MAC* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{MAC}, \mathsf{Verify})$*, define* $\mathbf{Adv}^{\textit{uf-cmva}}_{\textit{mac},A}(k) = Pr[\mathsf{G}^A_{\textit{uf-cmva}}(k)]$*, where* $\mathsf{G}^A_{\textit{uf-cmva}}(k)$ *is defined as follows:*

$$\frac{\text{MAIN } \mathsf{G}^A_{\textit{uf-cmva}}(k)}{}$$
$$Q \leftarrow \emptyset;\ params \xleftarrow{\$} \mathsf{Setup}(1^k);\ (iparams, sk) \xleftarrow{\$} \mathsf{KeyGen}(params)$$
$$(m, \sigma) \xleftarrow{\$} A^{\text{MAC},\text{VERIFY}}(params, iparams)$$
$$\textit{return } (m \notin Q) \wedge (\mathsf{Verify}(sk, m, \sigma) = 1)$$

$$\frac{\textit{Procedure } \text{MAC}_{sk}(m)}{}$$
$$Q \leftarrow Q \cup \{m\}$$
$$\textit{return } \mathsf{MAC}(sk, m)$$

$$\frac{\textit{Procedure } \text{VERIFY}_{sk}(m, \sigma)}{}$$
$$\textit{return } \mathsf{Verify}(sk, m)$$

Then the MAC is uf-cmva secure *if for all PPT adversaries A, there exists a negligible function* $\nu(\cdot)$ *such that* $\mathbf{Adv}^{\textit{uf-cmva}}_{\textit{mac},A}(k) < \nu(k)$*.*

A stronger security notion for MACs is sometimes used, where A may win by outputting (m, σ), even if $m \in Q$, provided σ was not output by the MAC oracle for m. The schemes we present were expressly designed *not* to provide this type of security, to allow tags to be *re-randomized* (or blinded) and thus allow for more efficient zero-knowledge proofs of possession of a MAC.

3.1 $\mathsf{MAC}_{\mathsf{GGM}}$

Our first MAC is a generalization of a scheme due to Dodis et al. [16]. The original MAC works in a cyclic group G of prime order p, and the secret key is a pair $(x_0, x_1) \in \mathbb{F}_p^2$. To compute the MAC of

a message $m \in \mathbb{F}_p$, choose $u \in_R G$, and compute $(u, u^{mx_1+x_0})$ as the tag. To verify a tag (u, u') for a message m, check whether $u^{mx_1+x_0} = u'$.

We extend the scheme to support n attributes, where the secret key becomes $(x_0, x_1, \ldots, x_n)$ and tags are computed as $(u, u^{x_1m_1+\ldots+x_nm_n+x_0})$. Note that $m_1, \ldots, m_n$ are n messages, each in $\mathbb{F}_p$, rather than the binary decomposition of a single message m. We refer to this scheme as $\mathsf{MAC}_{\mathsf{GGM}}$ (the single message and binary message schemes were respectively called $\mathsf{MAC}_{\mathsf{hwPRF}}$ and $\mathsf{MAC}_{\mathsf{WhwPRF}}$ in [16]). KeyGen has an optional step that is required only when $\mathsf{MAC}_{\mathsf{GGM}}$ is used for keyed-verification credentials.

In what follows (including for $\mathsf{MAC}_{\mathsf{DDH}}$), we use $\vec{m} = (m_1, \ldots, m_n)$ to mean a list of n messages in $\mathbb{F}_p$, and use $H_x(\vec{m}) := x_0 + \sum_i^n \vec{x}_i\vec{m}_i$.

KeyGen(*params*)**:** Choose a secret key $sk := \vec{x} \in_R \mathbb{F}_p^{n+1}$. Optionally, choose $\tilde{x_0} \in_R \mathbb{F}_p$ and compute $C_{x_0} := g^{x_0}h^{\tilde{x_0}}$ and $(X_1 := h^{x_1}, \ldots, X_n := h^{x_n})$, and publish the *issuer parameters*, denoted $iparams := (C_{x_0}, X_1, \ldots, X_n)$.

MAC$(sk, \vec{m})$**:** Choose $u \in_R G \setminus \{1\}$ and compute the tag $\sigma = (u, u')$, where $u' := u^{H_x(\vec{m})}$.

Verify$(sk, \vec{m}, \sigma)$**:** Parse $\sigma = (u, u') \in G^2$. Accept if $u \neq 1$ and $u' = u^{H_x(\vec{m})}$.

Dodis et al. [16] prove that under the DDH assumption, $\mathsf{MAC}_{\mathsf{GGM}}$ is suf-cma secure when $n = 1$. In this definition, security is called *selective unforgeability*, because the attacker must select the message he will use in a forgery *before* seeing any tags, and is not allowed verification queries. However, for our credential system, we require uf-cmva security. (Selective unforgeability gives only very limited protection against misbehaving adversaries, and verification queries are inherent in anonymous credentials as the adversary is always able to present credentials and observe the verifier's reaction.)

We stress that Dodis et al. give no evidence that $\mathsf{MAC}_{\mathsf{GGM}}$ is not in fact uf-cmva secure. Rather, it appears that their proof technique does not extend to also prove security under the stronger definition. A simple (but inefficient) reduction exists between uf-cma and suf-cma. A uf-cma adversary is transformed into an suf-cma adversary by an algorithm which guesses the message to be forged by the uf-cma adversary. The success probability of the new adversary is $\epsilon/|M|$ where M is the message space of the scheme, and ϵ is the success probability of the uf-cma adversary. If the size of M is constrained, the loss in security may be acceptable (i.e., it may be acceptable to use an suf-cma secure scheme). This may be of use in our application, in the very limited setting where credentials contain a small number of attributes from a small set, known to the issuer, and where during presentation the user is required to prove that all the attributes in his credential are within this set.

To ensure security in the more realistic case of unconstrained messages (attributes), and when verification queries are allowed (as in a credential system), we prove that $\mathsf{MAC}_{\mathsf{GGM}}$ is uf-cmva secure in the generic group model. Additionally, we include *iparams* in our analysis. A proof of the following theorem is given in Appendix A.1.

Theorem 2. *In the generic group model, a uf-cmva adversary attacking the* $\mathsf{MAC}_{\mathsf{GGM}}$ *scheme, succeeding with non-negligible probability, performs* $\Omega(\sqrt{p})$ *group operations.*

3.2 $\mathsf{MAC}_{\mathsf{DDH}}$

In this section, we describe another MAC construction, called $\mathsf{MAC}_{\mathsf{DDH}}$. Recall that *params* are created by Setup(1^k) and are assumed to be available to all algorithms, that $\vec{m} = (m_1, \ldots, m_n)$ is a list of n messages in $\mathbb{F}_p$, and that the optional step in KeyGen is required only when $\mathsf{MAC}_{\mathsf{DDH}}$ is used for keyed-verification credentials.

$\mathsf{KeyGen}(params)$**:** Pick $z, x_0, y_0 \ldots, x_n, y_n \xleftarrow{\$} \mathbb{F}_p$. Output $sk := (\vec{x}, \vec{y}, z)$. Optionally, compute $X_i := h^{x_i}$ and $Y_i := h^{y_i}$ for each $i \in \{0, \ldots n\}$, and publish $iparams := (\vec{X}, \vec{Y})$.

$\mathsf{MAC}(sk, \vec{m})$**:** Pick $r \xleftarrow{\$} \mathbb{F}_p$ and set $\sigma_w := g^r$, $\sigma_x := g^{rH_x(\vec{m})}$, $\sigma_y := g^{rH_y(\vec{m})}$, and $\sigma_z := g^{zr}$. Output $(\sigma_w, \sigma_x, \sigma_y, \sigma_z)$.

$\mathsf{Verify}(sk, \vec{m}, \sigma)$**:** Parse $\sigma = (\sigma_w, \sigma_x, \sigma_y, \sigma_z) \in G^4$. Check that $\sigma_w \neq 1$, $\sigma_x = \sigma_w^{H_x(\vec{m})}$, $\sigma_y = \sigma_w^{H_y(\vec{m})}$, and $\sigma_z = \sigma_w^z$. Accept if these checks pass and reject otherwise.

Theorem 3. *If the DDH assumption holds in G, then $\mathsf{MAC}_{\mathsf{DDH}}$ is uf-cmva secure.*

A proof of this theorem is given in Appendix A.2. Briefly, the proof approximately follows the standard dual-system approach [31], which proceeds (roughly) as follows: first, the scheme exists in a subgroup of some larger group, and a "shadow" copy of the scheme is added into a new subgroup. This addition goes unnoticed by *subgroup hiding*, a computational assumption. Next, this shadow copy is randomized, so that it becomes decoupled from the scheme in the original subgroup. This goes unnoticed by *parameter hiding*, which is typically a statistical property of the group. Finally, the additional randomness from the new subgroup is folded back into the original subgroup (again, by subgroup hiding), at which point it can be used to, e.g., blind information.

In our setting, in which we are working in a cyclic group, we have no nontrivial subgroups. We can nevertheless think of two different "groups": one using g as a generator, and one using h. Our first step in the proof is therefore folding in values using h into the MACs, and switching to a different verification procedure to accommodate this change. This change goes unnoticed, not by a computational subgroup hiding argument (because, again, we have no subgroups), but rather by a statistical argument.

Next, we continue to follow the dual-system approach by decoupling the randomness used in the h component from the g component. Whereas in the standard dual-system approach this change is statistical, we now use a computational assumption (DDH) to argue that the change goes unnoticed. We can now use this extra randomness to mask the secret values, at which point outputting a valid MAC becomes statistically hard.

4 Keyed-Verification Credentials

In this section we first describe the set of algorithms that form a keyed-verification credential scheme. We then informally describe the desired security and privacy properties (formal definitions are in Appendix B), present constructions of keyed-verification credentials based on $\mathsf{MAC}_{\mathsf{GGM}}$ and $\mathsf{MAC}_{\mathsf{DDH}}$, and prove our $\mathsf{MAC}_{\mathsf{DDH}}$-based construction secure assuming DDH. The proof of security for our $\mathsf{MAC}_{\mathsf{GGM}}$-based construction is a trivial simplification of the $\mathsf{MAC}_{\mathsf{DDH}}$-based proof, so we omit it.

A keyed-verification credential system consists of the following algorithms:

$\mathsf{Setup}(1^k)$ defines the system parameters *params*. We will assume that *params* is available to all algorithms, and that all parties have assurance it was created correctly.

$\mathsf{CredKeygen}(params)$ is run by the issuer on input *params* to generate a secret key sk and (public) issuer parameters *iparams*.

$\mathsf{BlindIssue}(sk, S) \leftrightarrow \mathsf{BlindObtain}(iparams, (m_1, \ldots, m_n))$ is a potentially interactive protocol where a user can obtain a credential on attributes $(m_1, \ldots, m_n)$ from an issuer who is only given some subset S of those attributes.

$\mathsf{Show}(\mathit{iparams}, \mathit{cred}, (m_1, \ldots, m_n), \phi) \leftrightarrow \mathsf{ShowVerify}(sk, \phi)$ is an interactive protocol between a user and a verifier. Show is run by a user to generate a proof of possession π of a credential *cred* certifying some set of attributes $(m_1, \ldots, m_n)$ satisfying a set of statements ϕ under the key corresponding to *iparams*, and ShowVerify is run by the verifier in possession of sk to verify proof π claiming knowledge of a credential satisfying the statements ϕ.

While we defined our presentation protocol in terms of a single credential, we could generalize our definitions and constructions to allow the user to prove relationships between attributes across multiple credentials that he owns. We chose the above variant because it allows for fairly simple definitions, yet still allows us to consider properties of a credential scheme as it would be used.

Note that the standard approach of requiring that the Show protocol be a proof of knowledge of a credential cannot be directly applied here because the verifier must know the issuer secret key in order to verify the credential. This is somewhat similar to a designated verifier proof [22], but it has the additional complication that the statement (validity of the credential) depends on the verifier's secret key.

4.1 Security properties

A keyed-verification credential system should have the following security properties (defined formally in Appendix B). Informally, *correctness* requires that every credential generated by Issue for attribute set $\{m_1, \ldots, m_n\}$ can be used to generate a proof for any statement satisfied by that attribute set. *Unforgeability* requires that an adversary cannot produce an accepting proof for a statement ϕ that is not satisfied by any of the attribute sets for which it has received credentials. *Anonymity* requires that the proofs produced by Show reveal nothing more than the statement being proved. *Blind issuance* requires that BlindIssue, BlindObtain define a secure two-party protocol for generating credentials on the user's attributes. Finally, *key-parameter consistency* requires that the probability that an adversary can find two secret keys that correspond to the same set of issuer parameters is negligible; this guarantees that the issuer cannot use different secret keys with different users and thus compromise their anonymity.

4.2 Keyed-verification credentials from $\mathsf{MAC}_{\mathsf{GGM}}$

We now give a construction of a keyed-verification credential system from $\mathsf{MAC}_{\mathsf{GGM}} = (\mathsf{Setup}_{\mathsf{GGM}}, \mathsf{KeyGen}_{\mathsf{GGM}}, \mathsf{MAC}_{\mathsf{GGM}}, \mathsf{Verify}_{\mathsf{GGM}})$. We define the following setup algorithms for the credential system.

$\mathsf{Setup}(1^k)$**:** Output $(G, p, g, h) \xleftarrow{\$} \mathsf{Setup}_{\mathsf{GGM}}(1^k)$.

$\mathsf{CredKeygen}(\mathit{params})$**:** Parse *params* as (G, p, g, h). Output $((C_{x_0}, \vec{X}), (\vec{x}, \tilde{x}_0)) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{GGM}}(\mathit{params})$.

Issuance. To issue a credential with the n attributes $(m_1, \ldots, m_n) \in \mathbb{F}_p^n$, the issuer computes $(u, u') \leftarrow \mathsf{MAC}_{\mathsf{GGM}}(sk, (m_1, \ldots, m_n))$ and returns (u, u') and π to the user, where

$$\pi := \mathrm{PK}\{(x_0, x_1, \ldots, x_n, \tilde{x_0}) : u' = u^{x_0} \prod_{i=1}^{n} (u^{m_i})^{x_i} \ \wedge \ C_{x_0} = g^{x_0} h^{\tilde{x}_0} \\ \wedge \ X_i = h^{x_i} \ \forall i \in \{1, \ldots, n\}\}$$

The proof π proves that (u, u') is well-formed with respect to the system and issuer parameters. If this proof verifies, the user accepts and outputs (u, u'). Otherwise it rejects with output $\perp$.

To alternatively keep some subset $\mathcal{H} \subseteq \{1, \ldots, n\}$ of the attributes hidden from the issuer, we can proceed as follows: The user generates an ElGamal keypair $(d, \gamma := g^d)$, then creates an encryption of g^{m_i} for each hidden attribute m_i as $E_i = (g^{r_i}, g^{m_i}\gamma^{r_i})$ for all $i \in \mathcal{H}$, using $r_i \in_R \mathbb{F}_p$. The user sends these ciphertexts to the issuer, along with a proof of knowledge of $\{r_i, m_i\}_{i \in \mathcal{H}}$. The issuer chooses $b \in_R \mathbb{F}_p$. It then computes $u = g^b$, and uses the homomorphic properties of ElGamal to generate an encryption $E_{u'}$ of $u' = g^{bx_0} \prod_1^n (g^{m_i})^{bx_i}$, and to randomize this encryption to obtain $E'_{u'}$ (by multiplying with an encryption of 0 using randomness $r' \in_R \mathbb{F}_p$). It sends $u, E'_{u'}$ to the user and gives a proof that these values have been generated correctly with respect to $(C_{x_0}, \{X_i\}_1^n)$ (i.e. a proof of knowledge of the appropriate $\{x_i\}_0^n, \tilde{x}_0, b$, and randomizing factors r'). If the proof does not verify, the user outputs $\perp$. Otherwise, the user decrypts $E'_{u'}$ to get u', and outputs (u, u').

Credential translation. In addition to proving that the ciphertexts E_i are well formed, the user can include proofs about the attributes the ciphertexts encrypt. For example, the user may prove that some of the attributes m_i are the same as in another credential, such as one that is more expensive to use (e.g., an Idemix credential), or one that cannot be presented multiple times unlinkably (e.g., a U-Prove credential).

Credential presentation. Here we present a construction for Show and $\mathsf{ShowVerify}$. The details of (one possible way of) instantiating the proof of knowledge are given in Appendix E.

$\mathsf{Show}(\mathit{params}, \mathit{iparams}, \phi, \mathit{cred}, \{m_i\}_i^n)$**:** The prover chooses $r, z_1, \ldots, z_n \in_R \mathbb{F}_p$ and parses $\mathit{cred} = (u, u')$. It then computes $\{C_{m_i} := u^{m_i} h^{z_i}\}_{i=1}^n$ and $C_{u'} := u' g^r$ and sends $\sigma = (u, \{C_{m_i}\}_i^n, C_{u'})$ and a proof of knowledge π, which it computes as

$$\pi = \mathrm{PK}\{(\vec{m}, \vec{z}, -r) : \phi(m_1, \ldots, m_n) = 1 \ \wedge \ C_{m_i} = u^{m_i} h^{z_i} \ \forall i \in \{1, \ldots, n\} \ \wedge \ V = g^{-r} \prod_{i=1}^{n} X_i^{z_i}\}.$$

$\mathsf{ShowVerify}(\mathit{params}, \mathit{iparams}, \phi, \{x_i\}_i^n, z, \sigma, \pi)$**:** The verifier parses $\sigma = (u, \{C_{m_i}\}_i^n, C_{u'})$, computes V as

$$V = \frac{u^{x_0} \prod_{i=1}^n {C_{m_i}}^{x_i}}{C_{u'}}$$

and verifies the proof π using V. If the proof is valid, it outputs $(C_{m_1}, \ldots, C_{m_n})$, and otherwise it outputs $\perp$.

Security. To see that the $\mathsf{MAC}_{\mathsf{GGM}}$ protocol works when $n = 1$ and both parties are honest, note that the verifier computes

$$V = \frac{{C_{m_1}}^{x_1} u^{x_0}}{C_{u'}} = \frac{u^{m_1 x_1} h^{x_1 z_1} u^{x_0}}{u^{m_1 x_1 + x_0} g^r} = h^{x_1 z_1} g^{-r} = X_1^{z_1} g^{-r},$$

which matches the statement in the proof π. The security of the credential scheme is obtained as a special case of the $\mathsf{MAC}_{\mathsf{DDH}}$-based construction (as this is a strictly simpler construction).

4.3 Keyed-verification credentials from $\mathsf{MAC}_{\mathsf{DDH}}$

We now give a construction of a keyed-verification credential system from $\mathsf{MAC}_{\mathsf{DDH}} = (\mathsf{Setup}_{\mathsf{DDH}}, \mathsf{KeyGen}_{\mathsf{DDH}}, \mathsf{MAC}_{\mathsf{DDH}}, \mathsf{Verify}_{\mathsf{DDH}})$. We define the following setup algorithms for the credential system.

$\mathsf{Setup}(1^k)$**:** Output $(G, p, g, h) \xleftarrow{\$} \mathsf{Setup}_{\mathsf{DDH}}(1^k)$.

$\mathsf{CredKeygen}(params)$**:** Compute $((\vec{X},\vec{Y}),(\vec{x},\vec{y},z)) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{DDH}}(params)$. Pick $\tilde{x},\ \tilde{y},\ \tilde{z} \in_R \mathbb{F}_p$ and form commitments $C_{x_0} := g^{x_0}h^{\tilde{x}}$, $C_{y_0} := g^{y_0}h^{\tilde{y}}$, and $C_z := g^z h^{\tilde{z}}$. Output $iparams = (\vec{X},\vec{Y},C_{x_0}, C_{y_0},C_z)$ and $sk = (\vec{x},\vec{y},z,\tilde{x},\tilde{y},\tilde{z})$.

Issuance. To issue a credential with the attributes $(m_1,\ldots,m_n) \in \mathbb{F}_q^n$, the issuer chooses $\sigma = (\sigma_w,\sigma_x,\sigma_z) \xleftarrow{\$} \mathsf{MAC}_{\mathsf{DDH}}(sk,(m_1,\ldots,m_n))$, and returns σ to the user with a proof π, where

$$\begin{aligned}\pi := \mathrm{PK}\{(\vec{x},\vec{y},z,\tilde{x},\tilde{y},\tilde{z}) \ :\ & \sigma_x = \sigma_w^{x_0}\prod_1^n({\sigma_w}^{m_i})^{x_i} \ \wedge\ \sigma_y = \sigma_w^{y_0}\prod_1^n(\sigma_w^{m_i})^{y_i} \ \wedge\ \sigma_z = {\sigma_w}^z \\ & \wedge\ C_{x_0} = g^{x_0}h^{\tilde{x}} \wedge C_{y_0} = g^{y_0}h^{\tilde{y}} \ \wedge\ C_z = g^z h^{\tilde{z}} \\ & \wedge\ X_i = h^{x_i} \ \wedge\ Y_i = h^{y_i}\ \forall i \in \{1,\ldots,n\}\}.\end{aligned}$$

The proof π proves that the credential is well-formed with respect to the system and issuer parameters. If this proof verifies, the user outputs σ; otherwise it outputs $\perp$.

If some of the attributes must be hidden, we can first proceed as we did with $\mathsf{MAC}_{\mathsf{GGM}}$, to the point where the user sends the ciphertexts E_i and proofs of knowledge of $\{r_i,m_i\}_{i\in\mathcal{H}}$ to the issuer. The issuer now chooses $b \in_R \mathbb{F}_p$, computes $\sigma_w = g^b$, $\sigma_z = \sigma_w^z$, and uses the homomorphic properties of ElGamal to generate an encryption E_x of $\sigma_x = g^{bx_0}\prod_1^n(g^{m_i})^{bx_i}$ and an encryption E_y of $\sigma_y = g^{by_0}\prod_1^n(g^{m_i})^{bx_i}$. It then randomizes these to obtain E'_x (by multiplying with an encryption of 0 using randomness $r_x \in_R \mathbb{F}_p$) and E'_y. It sends $(\sigma_w,\sigma_z,E'_x,E'_y)$ to the user and gives a proof that these values have been generated correctly with respect to $(\vec{X},\vec{Y},C_{x_0},C_{y_0},C_z)$ (i.e., a proof of knowledge of the appropriate $(\vec{x},\vec{y},z,\tilde{x},\tilde{y},\tilde{z},b,r_x,r_y)$). If the proof does not verify, the user outputs $\perp$. Otherwise, the user decrypts E'_x and E'_y to get σ_x and σ_y respectively, and outputs $\sigma = (\sigma_w,\sigma_x,\sigma_y,\sigma_z)$.

As with the $\mathsf{MAC}_{\mathsf{GGM}}$ scheme, credential translation is also possible with the $\mathsf{MAC}_{\mathsf{DDH}}$ scheme.

Credential presentation. Here we present a construction for Show and ShowVerify.

$\mathsf{Show}(params,iparams,\phi,cred,\{m_i\}_i^n)$**:** The prover chooses $r,r_x,r_y,z_1,\ldots,z_n \in_R \mathbb{F}_p$ and parses $cred = (\sigma_w,\sigma_x,\sigma_y,\sigma_z)$. It first randomizes the credential by computing $\sigma_w = \sigma_w^r$, $\sigma_x = \sigma_x^r$, $\sigma_y = \sigma_y^r$, and $\sigma_z = \sigma_z^r$, and then computes

$$\{C_{m_i} := \sigma_w^{m_i}h^{z_i}\}_{i=1}^n,\ C_{\sigma_x} := \sigma_x g^{r_x},\ C_{\sigma_y} := \sigma_y g^{r_y},\ V_x := g^{-r_x}\prod_{i=1}^n X_i^{z_i},\ \text{and } V_y := g^{-r_y}\prod_{i=1}^n Y_i^{z_i}.$$

It then sends $\sigma = (\sigma_w,\sigma_z,C_{\sigma_x},C_{\sigma_y},V_x,V_y,\{C_{m_i}\}_i^n)$ along with a proof of knowledge π, which it computes as

$$\begin{aligned}\pi = \mathrm{PK}\{\vec{m},\vec{z},-r_x,-r_y) \ :\ & \phi(m_1,\ldots,m_n) = 1 \ \wedge\ C_{m_i} = \sigma_w^{m_1}h^{z_1}\ \forall\ i \in \{1,\ldots,n\} \\ & \wedge\ V_x = g^{-r_x}\prod_{i=1}^n X_i^{z_i} \ \wedge\ V_y = g^{-r_y}\prod_{i=1}^n Y_i^{z_i}\}.\end{aligned}$$

$\mathsf{ShowVerify}(params,iparams,\phi,\{x_i,y_i\}_i^n,z,\sigma,\pi)$**:** The verifier parses $\sigma = (\sigma_w,\sigma_y,\sigma_z,V_x,V_y,\{C_{m_i}\}_i^n,C_{\sigma_x}, C_{\sigma_y})$ and verifies that

$$V_x = \frac{\sigma_w^{x_0}\prod_{i=1}^n C_{m_i}^{x_i}}{C_{\sigma_x}} \quad \text{and} \quad V_y = \frac{\sigma_w^{y_0}\prod_{i=1}^n C_{m_i}^{y_i}}{C_{\sigma_y}}.$$

It then verifies the proof π. If the proof is valid and if $\sigma_z = \sigma_w^z$ it accepts and outputs $(C_{m_1},\ldots,C_{m_n})$, and otherwise it rejects and outputs $\perp$.

Security. We give a formal proof of the following theorem in Appendix C.

Theorem 4. *If DDH holds and the proof system is a zero-knowledge proof of knowledge, the above algorithms* (CredKeygen, Issue, CredVerify, , Show, ShowVerify, , BlindIssue, BlindObtain) *make up a secure keyed-verification credential system.*

Intuitively, credential unforgeability follows from the unforgeability of the MAC (which is based on DDH); credential anonymity follows from the zero knowledge property of the proof; blind issuance follows from the extractability of the proof and the IND-CPA security of the encryption scheme (which, for ElGamal, follows from DDH); and key-parameter consistency follows from the binding property of the commitment scheme (which, for Pedersen commitments, follows from the discrete log assumption, which is implied by DDH). In Appendix D we discuss several possible instantiations of the zero-knowledge proof of knowledge.

As the protocols for $\mathsf{MAC}_{\mathsf{GGM}}$ are essentially a simplified version of those for $\mathsf{MAC}_{\mathsf{DDH}}$, the proof for $\mathsf{MAC}_{\mathsf{GGM}}$ is a straightforward simplification of the proof, so we omit it.

5 Efficiency

In this section we compare the efficiency of our new schemes to U-Prove and Idemix. We focus on the computational cost of creating a presentation proof, as this operation typically must be done by the largest range of devices. We consider the $\mathsf{MAC}_{\mathsf{GGM}}$- and $\mathsf{MAC}_{\mathsf{DDH}}$-based schemes, where the proof system is implemented with Fiat-Shamir (full details of $\mathsf{MAC}_{\mathsf{GGM}}$ are given in Appendix E, and $\mathsf{MAC}_{\mathsf{DDH}}$ is very similar). Using the proof system from [15] will have essentially the same computational cost (not including communication time). Complete descriptions of Idemix and U-Prove are available in [21] and [24] respectively. We did not include the bilinear CL signature schemes [1, 10], or the recent scheme of Baldimtsi and Lysyanskaya [2], in our comparison, as detailed specifications (including parameter choices) are not available.

Credential Size. Table 1 shows the size of a credential in all four schemes, both asymptotically, and for a concrete choice of parameters. The parameter s is the number of times the credential may be shown unlinkably (which is relevant for U-Prove). The size only counts the cryptographic components of the credential, the metadata and attribute values are assumed to be the same for all systems. The overhead of $\mathsf{MAC}_{\mathsf{GGM}}$ is the least, followed by $\mathsf{MAC}_{\mathsf{DDH}}$, which is the size of a single U-Prove token. The size of SRSA group elements makes Idemix credentials larger than $\mathsf{MAC}_{\mathsf{GGM}}$ and $\mathsf{MAC}_{\mathsf{DDH}}$, however, once $s > 5$, Idemix credentials are smaller than U-Prove credentials.

	Credential size for s shows	
	Asymptotic	Concrete (in bits)
U-Prove	$O(s)$	$1024s$
Idemix	$O(1)$	5369
$\mathsf{MAC}_{\mathsf{GGM}}$	$O(1)$	512
$\mathsf{MAC}_{\mathsf{DDH}}$	$O(1)$	1024

Table 1: Comparison of credential sizes of U-Prove, Idemix, $\mathsf{MAC}_{\mathsf{GGM}}$ and $\mathsf{MAC}_{\mathsf{DDH}}$. The number of times the credential may be shown is denoted s. U-Prove, $\mathsf{MAC}_{\mathsf{GGM}}$ and $\mathsf{MAC}_{\mathsf{DDH}}$ use a 256-bit elliptic curve group. Idemix uses a 2048-bit modulus.

Computation Cost for Presentation. We estimate the cost of creating a presentation proof and compare the four schemes. Our estimate is formed by counting the number of multi-exponentiations required to create a presentation proof. We use the notation ℓ-exp to denote computing the product of ℓ powers. To realistically estimate the performance of Idemix, the bitlengths of the exponents must also be considered, so we use the notation ℓ-exp$(b_1, \ldots, b_\ell)$ to denote the product of ℓ powers when the bitlengths of the exponents are $b_1, \ldots, b_\ell$. These bitlengths are calculated from the Idemix specification [21]. For U-Prove, $\mathsf{MAC}_{\mathsf{GGM}}$ and $\mathsf{MAC}_{\mathsf{DDH}}$ the bitlength of the exponent is always the length of the group order (256-bits in our comparison).

Table 2 gives the number of multi-exponentiations in terms of three parameters: n is the number of attributes in a credential, r is the number of *revealed* attributes in a presentation proof, and c is the number of *committed* attributes. For each committed attribute m, a separate Pedersen commitment is output. As a further comparison, Table 2 includes the time required to compute these multi-exponentiations for a given choice of parameters (n, c, r). Our multi-exponentiation implementation in G uses the NIST 256-bit elliptic curve, and for Idemix uses the parameters in [21]. The benchmarks were computed on an Intel Xeon CPU (E31230, quad core, 3.2 GHz) on an HP Z210 workstation running Windows 7 (64-bit). The times are in milliseconds, and are the average of 100 runs.

The times given in Table 2 show that the new schemes are competitive with U-Prove, especially when most of the attributes are committed, and that they are much faster than Idemix. In particular, in the first benchmark (when $(n, c, r) = (10, 2, 2)$), $\mathsf{MAC}_{\mathsf{GGM}}$ is 6.28 times faster than Idemix, and $\mathsf{MAC}_{\mathsf{DDH}}$ is 4.7 times faster than Idemix. Compared to U-Prove, $\mathsf{MAC}_{\mathsf{GGM}}$ and $\mathsf{MAC}_{\mathsf{DDH}}$ are 3.4 and 4.5 times slower, much less than the 21.2 times slowdown for Idemix.

In the second benchmark, when $(n, c, r) = (10, 10, 0)$, the performance of U-Prove, $\mathsf{MAC}_{\mathsf{GGM}}$ and $\mathsf{MAC}_{\mathsf{DDH}}$ are very similar. $\mathsf{MAC}_{\mathsf{GGM}}$ and $\mathsf{MAC}_{\mathsf{DDH}}$ are only 1.04 and 1.5 times slower than U-Prove. Idemix is 18.2, 16.3 and 12.5 times slower than U-Prove, $\mathsf{MAC}_{\mathsf{GGM}}$ and $\mathsf{MAC}_{\mathsf{DDH}}$, respectively.

	Number of exponentiations	Time (in ms) when $(n, c, r) =$ (10,2,2)	(10,10, 0)
U-Prove	1 $(n - r + 1)$-exp, $2c$ 2-exp	3.38	12.43
$\mathsf{MAC}_{\mathsf{GGM}}$	3 1-exp, 1 $(n - r + 1)$-exp $2(n - r)$ 2-exp	11.42	13.93
$\mathsf{MAC}_{\mathsf{DDH}}$	6 1-exp, 2 $(n - r + 2)$-exp $2(n - r + 1)$ 2-exp	15.31	18.10
Idemix	1 1-exp(2048) c 2-exp(256, 2046) c 2-exp(592, 2385) 1 $(n - r + 2)$-exp(456,3060,592,...,592)	71.72	226.79

Table 2: Comparison of estimated presentation proof generation cost. U-Prove, $\mathsf{MAC}_{\mathsf{GGM}}$ and $\mathsf{MAC}_{\mathsf{DDH}}$ use 256-bit elliptic curve parameters, and Idemix uses a 2048-bit modulus.

Discussion. These performance estimates show that the new schemes do provide a considerable performance advantage when compared to Idemix, and a small decrease compared to U-Prove. The other protocols, namely issuance and verification, have similar relative performance (for the user and issuer). In the case of issuance, our new schemes are expected to have slightly higher computational cost than issuing a single U-Prove token, but with one less round of interaction (when implemented with Fiat-Shamir proofs). When issuing multiple tokens, $\mathsf{MAC}_{\mathsf{GGM}}$ and $\mathsf{MAC}_{\mathsf{DDH}}$ have the best performance.

In all protocols, the cost of verification is within a small factor of the cost of proof generation.

We note some limitations of our comparison. First, the comparison is limited to applications where the issuer and verifier share a key. The parameter set used for Idemix is not believed to provide 128-bit security, so this favors Idemix in the comparison. For RSA, a 3072-bit modulus is required for 128-bit security, and for strong RSA we are unaware of any published guidance on choosing the modulus size. (Idemix would need *at least* a 3072-bit modulus for 128-bit security.) Another limitation is our choices of (n, c, r), which will be different across applications. Once an application is fixed, optimizations may be possible, such as creating a single commitment to multiple attributes, or re-using the same commitment in multiple presentations (e.g., when the commitment is used as a pseudonym).

References

[1] M. Au, W. Susilo, and Y. Mu. Constant-size dynamic k-TAA. *Proceedings of SCN 2006, LNCS* **4116** (2006), 111-125.

[2] F. Baldimtsi and A. Lysyanskaya. Anonymous Credentials Light. *Proceedings of ACM CCS 2013*, ACM Press, (2013). To appear.

[3] M. Belenkiy, J. Camenisch, M. Chase, M. Kohlweiss, A. Lysyanskaya and H. Shacham. Randomizable proofs and delegatable anonymous credentials. *Proceedings of CRYPTO 2009, LNCS* **5677**, (2009), 108–125.

[4] M. Bellare and O. Goldreich. On Defining Proofs of Knowledge. *Proceedings of CRYPTO 1992, LNCS* **740** (1993), 390–420

[5] P. Bichsel, J. Camenisch, T. Groß, and V. Shoup. Anonymous Credentials on a Standard Java Card. *Proceedings ACM CCS 2009*, ACM Press, (2009) 600–610.

[6] D. Boneh. The Decision Diffie-Hellman Problem. *Proceedings of ANTS-III, LNCS* **1423** (1998), 48–63.

[7] S. Brands. *Rethinking Public Key Infrastructures and Digital Certificates.* The MIT Press, August 2000.

[8] J. Camenisch and A. Lysyanskaya. An Efficient System for Non-Transferable Anonymous Credentials with Optional Anonymity Revocation. *Proceedings of EUROCRYPT 2001, LNCS* **2045** (2001), 93–118.

[9] J. Camenisch and A. Lysyanskaya. A Signature Scheme with Efficient Protocols. *Proceedings of SCN 2003, LNCS* **2576** (2003), 268–289.

[10] J. Camenisch and A. Lysyanskaya. Signature Schemes and Anonymous Credentials from Bilinear Maps. *Proceedings of CRYPTO 2004, LNCS* **3152** (2004), 56–72.

[11] J. Camenisch and M. Stadler. Proof Systems for General Statements About Discrete Logarithms. Technical Report TR 260 (1997), Institute for Theoretical Computer Science, ETH Zurich.

[12] D. Chaum. Security without Identification: Transaction Systems to Make Big Brother Obsolete. *Communications of the ACM* **28(10)** (1985), 1030–1044.

[13] J. Chen, H. Lim, S. Ling, H. Wang, and H. Wee. Shorter IBE and Signatures via Asymmetric Pairings. *Proceedings of Pairing 2012, LNCS* **7708**,(2012), 122–140.

[14] R. Cramer, R. Gennaro, and B. Schoenmakers. A Secure and Optimally Efficient Multi-Authority Election Scheme. *Proceedings of EUROCRYPT'97, LNCS* **1233**, (1997), 103–118.

[15] I. Damgård. Efficient Concurrent Zero-Knowledge in the Auxiliary String Model. *Proceedings of EUROCRYPT 2000, LNCS* **1807** (2000), 418430.

[16] Y. Dodis, E. Kiltz, K. Pietrzak, D. Wichs. Message Authentication, Revisited. *Proceedings of EUROCRYPT'12, LNCS* **7237** (2012), 355–374.

[17] A. Fiat and A. Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. *Proceedings of CRYPTO 1986, LNCS* **263** (1987), 186-194.

[18] M. Fischlin. Communication-Efficient Non-Interactive Proofs of Knowledge with Online Extractors. *Proceedings of CRYPTO 2005, LNCS* **3621** (2005), 152–168.

[19] M. Gerbush, A. Lewko, A. O'Neill, and B. Waters Dual Form Signatures: An Approach for Proving Security from Static Assumptions. *Proceedings of ASIACRYPT 2012, LNCS* **7658** (2012), 25–42.

[20] O. Goldreich. *The Foundations of Cryptography - Volume 2 Basic Applications.* Cambridge University Press, New York, 2004.

[21] IBM. Specification of the Identity Mixer Cryptographic Library (Revised version 2.3.0). IBM Research Report RZ 3730, April 2010.

[22] M. Jakobsson, K.Sako, and R. Impagliazzo. Designated Verifier Proofs and Their Applications. *Proceedings of EUROCRYPT96, LNCS* **1070** (1996) 143–154.

[23] NIST. *FIPS 186-3: Digital Signature Standard (DSS)*, Federal Information Processing Standards Publication (2009).

[24] C. Paquin and G. Zaverucha. U-Prove Cryptographic Specification V1.1 (Revision 2). April 2013. Available online: `www.microsoft.com/uprove`.

[25] C. Schnorr. Efficient Signature Generation by Smart Cards. *Journal of Cryptology* **4** (1991), 161-174.

[26] C. Schnorr. Security of Blind Discrete Log Signatures Against Interactive Attacks. *ICICS 2001, LNCS* **2229** (2001), 1–12.

[27] V. Shoup. Lower Bounds for Discrete Logarithms and Related Problems. *Proceedings of EUROCRYPT'97, LNCS* **1233** (1997), 256–266.

[28] Y. Tsiounis and M. Yung On the security of ElGamal based encryption. *Proceedings of PKC 1998, LNCS* **1431** (1998), 117–134.

[29] N. Smart. The Exact Security of ECIES in the Generic Group Model. *Proceedings of Cryptography and Coding, IMA Int. Conf., LNCS* **2260** (2001), 73–84.

[30] R. Steinfeld, J. Pieprzyk and H. Wang. How to strengthen any weakly unforgeable signature into a strongly unforgeable signature. *Proceedings of CT-RSA 2007, LNCS* **4377** (2007), 357–371.

[31] B. Waters. Dual System Encryption: Realizing Fully Secure IBE and HIBE under Simple Assumptions. *Proceedings of CRYPTO 2009, LNCS* **5677** (2009), 619–636.

A MAC Security

A.1 Security of $\mathsf{MAC}_{\mathsf{GGM}}$

In this section we give the proof of Theorem 2, that $\mathsf{MAC}_{\mathsf{GGM}}$ is uf-cmva secure in the generic group model. The proof is for the message space $\mathbb{F}_p$, however it may easily be generalized to the message space $\mathbb{F}_p^n$. Since the system parameter C_y hides y perfectly and unconditionally, we omit it from this analysis.

Proof. Let g be a fixed generator of a generic group G, and let G be written multiplicatively. We then represent elements $a \in G$ as $\log_g a \in \mathbb{F}_p$. We encode elements of G as random strings in a set $S \in \{0,1\}^*$ with the function $\zeta : \mathbb{F}_p \to S$ (i.e., $\zeta(\log_g a)$ gives the encoding of $a \in G$ as an element of S). The choice of S is not important, provided $|S| \geq q$.

Let A denote a uf-cmva attacker. A refers to elements of G only using their representation as elements of S. The attacker refers to elements in the message space directly.

We describe an algorithm B, which interacts with A, implementing oracles for group operations, as well as MAC and verification queries. B chooses the secret values $(x, y, h) \in_R \mathbb{F}_p^3$. The inputs B gives A are the system parameters: $g, H = g^h$, and $X = H^x = g^{hx}$, encoded as $\zeta(1), \zeta(h)$, and $\zeta(xh)$.

B maintains a list L of polynomials in $\mathbb{F}_p[\overline{x}, \overline{y}, \overline{h}, \overline{z_1}, \ldots, \overline{z_{q_t}}]$, where q_t is the number of tag queries made by A. The indeterminates $(\overline{x}, \overline{y}, \overline{h}, \overline{z_1}, \ldots, \overline{z_{q_t}})$ correspond to the secrets (x, y, h) and the random values z_i used to create tags. Each polynomial in L corresponds to a group element at each step of A's computation. The list contains pairs $(F_i, \zeta_i) \in \mathbb{F}_p[\overline{x}, \overline{y}, \overline{h}, \overline{z_1}, \ldots, \overline{z_{q_t}}] \times S$. A second list Q maintains the set of queried messages. Both lists are initially empty.

B counts the number of group oracle queries by q_G, and the number of tag queries with q_t, both initialized to zero. The number of verification queries are not counted (but is assumed to be polynomial in the security parameter). The total number of group operations is $q = q_G + 2q_t$, since each tag query requires two group operations to answer.

Group operation. A provides input $(\zeta_i, \zeta_j, \pm)$ where $\pm$ corresponds to multiply/divide, and $i, j < q_G$. Then B sets $F_{q_G} = F_i \pm F_j$. If $F_{q_G} = F_\ell$ for $\ell < q_G$, B sets $\zeta_{q_G} = \zeta_\ell$, otherwise B sets $\zeta_{q_G} \in_R S$ distinct from $\zeta_0, \ldots, \zeta_{q_G - 1}$. B adds (F_{q_G}, ζ_{q_G}) to L and outputs ζ_{q_G} to A. Finally, B increments q_G.

MAC operation. On the i-th query, A provides input $m_i \in \mathbb{F}_p$. B sets $F_{q_G} = \overline{z_i}$ and $\zeta_{q_G} \in_R S$. Then B computes $F_{q_G+1} = \overline{z_i}(m_i\overline{x}+\overline{y}) = \overline{z_i}m_i\overline{x}+\overline{z_i}\overline{y}$. If $F_{q_G+1} = F_\ell$ for $\ell < q_G$, then B sets $\zeta_{q_G+1} = \zeta_\ell$, otherwise B sets $\zeta_{q_G+1} \in_R S$ (distinct from $\zeta_0, \ldots, \zeta_{q_G}$). The output to A is $(\zeta_{q_G}, \zeta_{q_G+1})$. Finally B adds two to q_G, one to q_t, and m_i is added to Q.

Note that we do not assume each MAC query is distinct: A may request multiple tags for the same m. A may also implement the re-randomize algorithm by repeated calls to the group operation oracle.

Verify query. The input from A is $(m, \zeta, \zeta') \in \mathbb{F}_p \times S \times S$. If either of ζ, ζ' are not in L, return "invalid". Then $\zeta = \zeta_i$ and $\zeta' = \zeta_j$ for some $i, j < q_G$. If

$$F_i \cdot (\overline{x}m + \overline{y}) = F_j$$

then return "valid", and otherwise return "invalid". Note that this operation does not change any of B's state, it only lets A query L.

At any time during the game, the polynomials in L are of degree (in $\overline{x}$, $\overline{y}$, $\overline{z_i}$) at most two: G-queries compute $F_i \pm F_j$, which does not increase degree, the initial polynomials have degree one, and MAC queries add a polynomial of degree 1 and of degree 2 to L.

After q queries ($q = q_G + 2q_t$), A outputs (m, ζ, ζ') for some $m \notin Q$, and $(\zeta, \zeta') = (\zeta_i, \zeta_j)$ for some $i, j \le q$. If A succeeds,

$$F_i \cdot (m\overline{x} + \overline{y}) = F_j, \text{ or equivalently, } F_i \cdot (m\overline{x} + \overline{y}) - F_j = 0\ . \tag{1}$$

Given the operations available to A, we have that

$$F_j = a(\overline{x}\overline{h}) + \sum_{i=1}^{q_t} b_i\overline{z}_i + \sum_{i=1}^{q_t} c_i\overline{z_i}(m_i\overline{x} + \overline{y})\ ,$$

where $\overline{z_i}$ indeterminates representing the random values chosen in each MAC query, and a, b_i and c_i are integers. Note that the only way we can have $F_j = F_i \cdot (m\overline{x}+\overline{y})$ for such an F_j is if $F_i = f_j = 0$ or if $m = m_\ell$ for some $m_\ell \in m_1, \ldots, m_{q_t}$. In either case this will not be a valid forgery. (In the first case the forgery will be rejected by Verify, in the second case this is not a new message.) Therefore, the polynomial in (1) is a non-zero polynomial of degree 2 (in $\overline{x}, \overline{y}, \overline{h}, \overline{z_i}$), and the adversary will succeed in his forgery only if the evaluation of this polynomial on the randomly chosen $(x, y, z_1, \ldots, z_{q_t})$ is 0. (Event (1).)

If, for a particular choice of $(x, y, h, z_1, \ldots, z_{q_t}) \in \mathbb{F}_p^{3+q_t}$, we have $F_i(x, y, h, z_1, \ldots, z_{q_t}) = F_j(x, y, h, z_1, \ldots, z_{q_t})$, but $F_i \neq F_j$, the simulation is invalid because B presented two elements to A as distinct, but they were in fact equal. This condition is described as:

$$F_i(x, y, h, z_1, \ldots, z_{q_t}) - F_j(x, y, h, z_1, \ldots, z_{q_t}) = 0\ . \tag{2}$$

Clearly, this second condition can only hold for an unfortunate random choice of $(x, y, h, z_1, \ldots, z_{q_t})$, and cannot be influenced by A. The success probability of A is bounded by the probability of events (1) and (2).

For fixed $i, j < q$ the Schwartz-Zippel lemma tells us that the probability that each of (1) or (2) holds is $2/p$, since the degree of the polynomial in each case is at most 2. Therefore the probability over all pairs (i, j) is

$$\binom{q}{2} \cdot \frac{4}{p}$$

$\underline{\text{MAIN } \boxed{\mathsf{G}^A_{\text{uf-cmva}}(k)} \;/\; \mathsf{G}^A_0(k) \;/\; \mathsf{G}^A_1(k)}$

$(p, G, g, h) \xleftarrow{\$} \mathsf{Setup}(1^k)$; $Q \leftarrow \emptyset$; $x_0, y_0, \ldots, x_n, y_n, z \xleftarrow{\$} \mathbb{F}_p$, $s, t, v_0, \ldots, v_n \xleftarrow{\$} \mathbb{F}_p$

$sk \leftarrow ((p, G, g, h), s, t, \vec{v}, \vec{x}, \vec{y}, z)$, $\boxed{sk \leftarrow ((p, G, g, h), \vec{x}, \vec{y}, z)}$

$iparams \leftarrow (\{h^{x_i}\}_i^n, \{h^{y_i}\}_i^n)$

$(\vec{m}^*, \sigma^*) \xleftarrow{\$} A^{\text{MAC},\text{VERIFY}}((p, G, g, h), iparams)$

$b \leftarrow ((\sigma_y \sigma_x^s)^z = (\sigma_z \sigma_w^t)^{H_y(\vec{m}^*)}) \wedge ((\sigma_w^{-H_v(\vec{m}^*)} \sigma_x)^{H_y(\vec{m}^*)} = (\sigma_x^s \sigma_y)^{H_x(\vec{m}^*)}) \wedge (\sigma_w \neq 1)$,
$\boxed{b \leftarrow ((\sigma_x = \sigma_w^{H_x(\vec{m}^*)}) \wedge (\sigma_y = \sigma_w^{H_y(\vec{m}^*)}) \wedge (\sigma_z = \sigma_w^z) \wedge (\sigma_w \neq 1))}$

return $((\vec{m}^* \notin Q) \wedge b)$

$\underline{\text{Procedure } \text{MAC}_{sk}(\vec{m})}$ // $\mathsf{G}^A_{\text{uf-cmva}}(k)$ / $\boxed{\mathsf{G}^A_0(k)}$

$Q \leftarrow Q \cup \{\vec{m}\}$

$r \xleftarrow{\$} \mathbb{F}_p$; $\sigma_w \leftarrow g^r$, $\boxed{\sigma_w \leftarrow h^r}$; $\sigma_x \leftarrow g^{r H_x(\vec{m})}$, $\boxed{\sigma_x \leftarrow (g^{H_x(\vec{m})} h^{H_v(\vec{m})})^r}$;
$\sigma_y \leftarrow g^{r H_y(\vec{m})}$, $\boxed{\sigma_y \leftarrow (g^{H_y(\vec{m}) - s H_x(\vec{m})} h^{-s H_v(\vec{m})})^r}$; $\sigma_z \leftarrow g^{zr}$, $\boxed{\sigma_z \leftarrow (g^z h^{-t})^r}$

return $(\sigma_w, \sigma_x, \sigma_y, \sigma_z)$

$\underline{\text{Procedure } \text{MAC}_{sk}(\vec{m})}$ // $\mathsf{G}^A_0(k)$ / $\boxed{\mathsf{G}^A_1(k)}$

$r \xleftarrow{\$} \mathbb{F}_p$, $\boxed{\omega, \chi \xleftarrow{\$} \mathbb{F}_p}$; $\sigma_w \leftarrow h^r$, $\boxed{\sigma_w \leftarrow h^\omega}$; $\sigma_x \leftarrow (g^{H_x(\vec{m})} h^{H_v(\vec{m})})^r$, $\boxed{\sigma_x \leftarrow h^\chi}$;
$\sigma_y \leftarrow (g^{H_y(\vec{m}) - s H_x(\vec{m})} h^{-s H_v(\vec{m})})^r$, $\boxed{\sigma_y \leftarrow g^{r H_y(\vec{m})} h^{-\chi s}}$; $\sigma_z \leftarrow (g^z h^{-t})^r$, $\boxed{\sigma_z \leftarrow g^{zr} h^{-\omega t}}$

$\underline{\text{Procedure } \text{VERIFY}_{sk}(\vec{m}, \sigma)}$ // $\boxed{\mathsf{G}^A_{\text{uf-cmva}}(k)}$ / $\mathsf{G}^A_0(k)$ / $\mathsf{G}^A_1(k)$

$\boxed{\text{return } ((\sigma_x = \sigma_w^{H_x(\vec{m})}) \wedge (\sigma_y = \sigma_w^{H_y(\vec{m})}) \wedge (\sigma_z = \sigma_w^z) \wedge (\sigma_w \neq 1))}$,
return $((\sigma_y \sigma_x^s)^z = (\sigma_z \sigma_w^t)^{H_y(\vec{m})}) \wedge (\sigma_w \neq 1)$

Figure 1: Games for the proof of Theorem 3. The boxed game uses the boxed code and the other games do not.

Therefore A's success probability after q queries is at most

$$\epsilon = \binom{q}{2} \cdot \frac{4}{p} \leq \frac{4q^2}{p}$$

To have a constant $\epsilon > 0$ requires $\Omega(\sqrt{p})$ operations in G. □

A.2 Security of $\mathsf{MAC}_{\mathsf{DDH}}$

Proof. Let A be a PT adversary playing game $\mathsf{G}^A_{\text{uf-cmva}}(k)$ that makes q_m MAC queries and q_v VERIFY queries, where $q_m = q_m(k)$ and $q_v = q_v(k)$ for polynomials $q_m(\cdot)$ and $q_v(\cdot)$. We provide a PT adversary B and negligible functions $\nu_0(\cdot)$ and $\nu(\cdot)$ such that

$$\mathbf{Adv}^{\text{uf-cmva}}_{\text{mac},A}(k) \leq q_v \nu_0(k) + q_m \mathbf{Adv}^{\text{ddh}}_B(k) + \nu(k)$$

for all $k \in \mathbb{N}$, from which the theorem follows. To do this, we build B, $\nu_0(\cdot)$, and $\nu(\cdot)$ such that for all $k \in \mathbb{N}$ we have

$$\Pr[\mathsf{G}^A_{\text{uf-cmva}}(k)] - \Pr[\mathsf{G}^A_0(k)] \leq q_v \nu_0(k) \tag{3}$$
$$\Pr[\mathsf{G}^A_0(k)] - \Pr[\mathsf{G}^A_1(k)] \leq q_m \mathbf{Adv}^{\text{ddh}}_B(k) \tag{4}$$
$$\Pr[\mathsf{G}^A_1(k)] \leq \nu(k). \tag{5}$$

We then have

$$\begin{aligned}
\mathbf{Adv}^{\text{uf-cmva}}_{\text{mac},A}(k) &= \Pr[\mathsf{G}^A_{\text{uf-cmva}}(k)] \\
&= (\Pr[\mathsf{G}^A_{\text{uf-cmva}}(k)] - \Pr[\mathsf{G}^A_0(k)]) + (\Pr[\mathsf{G}^A_0(k)] - \Pr[\mathsf{G}^A_1(k)]) + \Pr[\mathsf{G}^A_1(k)] \\
&\leq q_v \nu_0(k) + q_m \mathbf{Adv}^{\text{ddh}}_B(k) + \nu(k).
\end{aligned}$$

Equation 3.

To first prove Equation 3, we consider a modified version of $\mathsf{G}^A_{\text{uf-cmva}}(k)$ as an intermediate game: rather than pick $\vec{x}$ and $\vec{y}$ randomly, pick $x'_i, y'_i, v_i \xleftarrow{\$} \mathbb{F}_p$ and set $x_i := x'_i/\beta + v_i$ and $y_i := y'_i/\beta - s x_i$ for all i, $0 \leq i \leq n$; furthermore, rather than pick $z \xleftarrow{\$} \mathbb{F}_p$, pick $z' \xleftarrow{\$} \mathbb{F}_p$ and set $z := z'/\beta - t$. (Recall that $\beta = \log_g(h)$.) In the MAC oracle, use $r := r'\beta$ for $r' \xleftarrow{\$} \mathbb{F}_p$ rather than $r \xleftarrow{\$} \mathbb{F}_p$. These values are distributed identically to the values used in $\mathsf{G}^A_{\text{uf-cmva}}(k)$, so the distribution in this modified game is identical. Furthermore, we have

$$\begin{aligned}
\sigma'_w &= g^r = g^{r'\beta} = h^{r'} \\
\sigma'_x &= g^{rH_x(\vec{m})} = g^{r'\beta(H_{x'}(\vec{m})/\beta + H_v(\vec{m}))} = (g^{H_{x'}(\vec{m})} h^{H_v(\vec{m})})^{r'} \\
\sigma'_y &= g^{rH_y(\vec{m})} = g^{r'\beta((H_{y'}(\vec{m}) - sH_{x'}(\vec{m}))/\beta - sH_v(\vec{m}))} = (g^{H_{y'}(\vec{m}) - sH_{x'}(\vec{m})} h^{-sH_v(\vec{m})})^{r'} \\
\sigma'_z &= g^{rz} = g^{r'\beta(z'/\beta - t)} = (g^{z'} h^{-t})^{r'},
\end{aligned}$$

so the MAC responses in the modified game are identical to those in $\mathsf{G}^A_0(k)$ in which $x'_i, y'_i, z', r' \xleftarrow{\$} \mathbb{F}_p$.

To address the changes in verification, we proceed through a series of hybrids: define $\mathsf{H}^A_i(k)$ to be a game in which the first i VERIFY queries are answered using Verify, the last $q_v - i$ are answered using the verification procedure defined in $\mathsf{G}^A_0(k)$ (referred to in the sequel as $\mathsf{SimVerify}$), and the verification at the end is considered the $q_v + 1$st query; then $\mathsf{H}^A_{q_v+1}(k)$ is the intermediate game and $\mathsf{H}^A_0(k)$ is $\mathsf{G}^A_0(k)$. To transition from $\mathsf{H}^A_i(k)$ to $\mathsf{H}^A_{i-1}(k)$ for $i < q_v + 1$, we therefore need only consider the i-th query $(\vec{m}, \sigma)$ (as the two games are identical both before and after this query), and the probability that $\mathsf{Verify}(sk, \vec{m}, \sigma) \neq \mathsf{SimVerify}(sk, \vec{m}, \sigma)$; i.e., that $\mathsf{H}^A_{i-1}(k)$ and $\mathsf{H}^A_i(k)$ produce different responses on the i-th query. We refer to this event as E_i, and show that $\Pr[E_i] \leq \nu_0(k)$ for a negligible function $\nu_0(k)$.

If $\mathsf{Verify}(sk, \vec{m}, \sigma) = \mathsf{accept}$ then

$$\begin{aligned}
(\sigma_y \sigma_x^s)^{z'} &= (\sigma_w^{H_y(\vec{m})} \sigma_w^{sH_x(\vec{m})})^{z'} \\
&= (\sigma_w^{(H_{y'}(\vec{m}) - sH_{x'}(\vec{m}))/\beta - sH_v(\vec{m})} \sigma_w^{s(H_{x'}(\vec{m})/\beta + H_v(\vec{m}))})^{z'} \\
&= (\sigma_w^{H_{y'}(\vec{m})/\beta})^{\beta(z+t)} \\
&= (\sigma_w^z \sigma_w^t)^{H_{y'}(\vec{m})} \\
&= (\sigma_z \sigma_w^t)^{H_{y'}(\vec{m})},
\end{aligned}$$

so $\mathsf{SimVerify}(sk, \vec{m}, \sigma) = \mathsf{accept}$ and E_i happens only if $\mathsf{SimVerify}(sk, \vec{m}, \sigma) = \mathsf{accept}$ but $\mathsf{Verify}(sk, \vec{m}, \sigma) = \mathsf{reject}$. Consider R_x, R_y, and R_z such that $\sigma_x = R_x \sigma_w^{H_x(\vec{m})}$, $\sigma_y = R_y \sigma_w^{H_y(\vec{m})}$, and $\sigma_z = R_z \sigma_w^z$; then Verify accepts only if $R_x = R_y = R_z = 1$. If $\mathsf{SimVerify}(sk, \vec{m}, \sigma) = \mathsf{accept}$ then, since $H_y(\vec{m}) + sH_x(\vec{m}) = H_{y'}(\vec{m})/\beta$ and $z' = \beta(z + t)$, we have

$$\begin{aligned}
(\sigma_y \sigma_x^s)^{z'} &= (\sigma_z \sigma_w^t)^{H_{y'}(\vec{m})} \\
(R_y \sigma_w^{H_y(\vec{m})} R_x \sigma_w^{sH_x(\vec{m})})^{z'} &= (R_z \sigma_w^z \sigma_w^t)^{H_{y'}(\vec{m})} \\
(R_x R_y \sigma_w^{H_y(\vec{m}) + sH_x(\vec{m})})^{z'} &= (R_z \sigma_w^{z+t})^{H_{y'}(\vec{m})} \\
(R_x R_y \sigma_w^{H_{y'}(\vec{m})/\beta})^{\beta(z+t)} &= R_z^{H_{y'}(\vec{m})} \sigma_w^{(z+t)H_{y'}(\vec{m})} \\
(R_x R_y)^{\beta(z+t)} \sigma_w^{(z+t)H_{y'}(\vec{m})} &= R_z^{H_{y'}(\vec{m})} \sigma_w^{(z+t)H_{y'}(\vec{m})} \\
(R_x R_y)^{z'} &= R_z^{H_{y'}(\vec{m})}.
\end{aligned}$$

As up until the i-th query A has never seen the values of z' and $\vec{y}'$, however, it has at most a negligible probability (in fact, probability $1/2^k$) of coming up with R_x, R_y, and R_z that satisfy this equation but such that it is not the case that $R_x = R_y = R_z = 1$. We therefore have that $\Pr[E_i] \leq \nu_0(k)$ for a negligible function $\nu_0(\cdot)$, for all queries to the VERIFY oracle. To finally address the case of $i = q_v + 1$, in which we additionally check that $(\sigma_w^{-H_v(\vec{m}^*)} \sigma_x)^{H_y(\vec{m}^*)} = (\sigma_x^s \sigma_y)^{H_x(\vec{m}^*)}$, we observe that if $\mathsf{Verify}(sk, \vec{m}^*, \sigma^*) = \mathsf{accept}$ then

$$\begin{aligned}
(\sigma_w^{-H_v(\vec{m}^*)} \sigma_x)^{H_{y'}(\vec{m}^*)} &= (\sigma_w^{-H_v(\vec{m}^*)} \sigma_w^{H_x(\vec{m}^*)})^{H_{y'}(\vec{m}^*)} \\
&= (\sigma_w^{-H_v(\vec{m}^*)} \sigma_w^{H_{x'}(\vec{m}^*)/\beta + H_v(\vec{m}^*)})^{H_{y'}(\vec{m}^*)} \\
&= (\sigma_w^{H_{y'}(\vec{m}^*)} / \beta)^{H_{x'}(\vec{m}^*)} \\
&= (\sigma_w^{sH_x(\vec{m})} \sigma_w^{H_{y'}(\vec{m^*})/\beta - sH_x(\vec{m})})^{H_{x'}(\vec{m}^*)} \\
&= (\sigma_x^s \sigma_w^{H_y(\vec{m}^*)})^{H_{x'}(\vec{m}^*)} \\
&= (\sigma_x^s \sigma_y)^{H_{x'}(\vec{m}^*)}.
\end{aligned}$$

We must also show that if this equality holds then $\mathsf{Verify}(sk, \vec{m}^*, \sigma^*) = \mathsf{accept}$; we do this by an argument similar to the one for $\mathsf{SimVerify}$, so we have

$$\begin{aligned}
(\sigma_w^{-H_v(\vec{m}^*)} \sigma_x)^{H_{y'}(\vec{m}^*)} &= (\sigma_x^s \sigma_y)^{H_{x'}(\vec{m}^*)} \\
(\sigma_w^{-H_v(\vec{m}^*)} R_x \sigma_w^{H_x(\vec{m}^*)})^{H_{y'}(\vec{m}^*)} &= (R_x R_y \sigma_w^{sH_x(\vec{m}^*)} \sigma_w^{H_y(\vec{m}^*)})^{H_{x'}(\vec{m}^*)} \\
(R_x \sigma_w^{-H_v(\vec{m}^*)} \sigma_w^{H_{x'}(\vec{m}^*)/\beta + H_v(\vec{m}^*)})^{H_{y'}(\vec{m}^*)} &= (R_x R_y \sigma_w^{sH_x(\vec{m}^*)} \sigma_w^{H_{y'}/\beta - sH_x(\vec{m}^*)})^{H_{x'}(\vec{m}^*)} \\
(R_x \sigma_w^{H_{x'}(\vec{m}^*)/\beta})^{H_{y'}(\vec{m}^*)} &= (R_x R_y \sigma_w^{H_{y'}(\vec{m}^*)/\beta})^{H_{x'}(\vec{m}^*)} \\
R_x^{H_{y'}(\vec{m}^*)} &= (R_x R_y)^{H_{x'}(\vec{m}^*)}.
\end{aligned}$$

As A again has at most a negligible probability of coming up with values R_x and R_y that satisfy this equation but such that either $R_x \neq 1$ or $R_y \neq 1$, $\Pr[E_{q_v+1}] \leq \nu_0(k)$ as well, which proves Equation 3.

Equation 4.

We now prove Equation 4. To do this, we consider a series of hybrids: in each hybrid $\mathsf{H}_i^A(k)$, the first i queries use the MAC values from $\mathsf{G}_1^A(k)$, and the last $q_m - i$ use the values from $\mathsf{G}_0^A(k)$; $\mathsf{H}_0^A(k)$ is then equivalent to $\mathsf{G}_0^A(k)$, and $\mathsf{H}_{q_m}^A(k)$ is equivalent to $\mathsf{G}_1^A(k)$. To argue that $\mathsf{H}_i^A(k)$ is indistinguishable

Procedure $\textsc{Mac}_{sk}(\vec{m})$ // $\mathsf{H}^A_{i-1}(k)$ / $\boxed{\mathsf{H}^A_{i,1}(k)}$

if $(j < i)$ then $r, \omega, \chi \xleftarrow{\$} \mathbb{F}_p$; $\sigma_w \leftarrow h^\omega$; $\sigma_x \leftarrow h^\chi$; $\sigma_y \leftarrow g^{rH_y(\vec{m})} h^{-\chi s}$; $\sigma_z \leftarrow g^{zr} h^{-\omega t}$

if $(j = i)$ then $r, \boxed{\omega} \xleftarrow{\$} \mathbb{F}_p$; $\sigma_w \leftarrow h^r$, $\boxed{\sigma_w \leftarrow h^\omega}$; $\sigma_x \leftarrow (g^{H_x(\vec{m})} h^{H_v(\vec{m})})^r$, $\boxed{\sigma_x \leftarrow g^{rH_x(\vec{m})} h^{\omega H_v(\vec{m})}}$;
$\sigma_y \leftarrow (g^{H_y(\vec{m}) - sH_x(\vec{m})} h^{-sH_v(\vec{m})})^r$, $\boxed{\sigma_y \leftarrow g^{r(H_y(\vec{m}) - sH_x(\vec{m}))} h^{-s\omega H_v(\vec{m})}}$;
$\sigma_z \leftarrow (g^z h^{-t})^r$, $\boxed{\sigma_z \leftarrow g^{zr} h^{-\omega t}}$

if $(j > i)$ then $r \xleftarrow{\$} \mathbb{F}_p$; $\sigma_w \leftarrow h^r$; $\sigma_x \leftarrow (g^{H_x(\vec{m})} h^{H_v(\vec{m})})^r$; $\sigma_y \leftarrow (g^{H_y(\vec{m}) - sH_x(\vec{m})} h^{-sH_v(\vec{m})})^r$;
$\sigma_z \leftarrow (g^z h^{-t})^r$

Procedure $\textsc{Mac}_{sk}(\vec{m})$ // $\mathsf{H}^A_{i,1}(k)$ / $\boxed{\mathsf{H}^A_{i,2}(k)}$

if $(j = i)$ then $r, \omega, \boxed{\chi} \xleftarrow{\$} \mathbb{F}_p$; $\sigma_w \leftarrow h^\omega$; $\sigma_x \leftarrow g^{rH_x(\vec{m})} h^{\omega H_v(\vec{m})}$, $\boxed{\sigma_x \leftarrow g^{rH_x(\vec{m})} h^\chi}$;
$\sigma_y \leftarrow g^{r(H_y(\vec{m}) - sH_x(\vec{m}))} h^{-s\omega H_v(\vec{m})}$, $\boxed{\sigma_y \leftarrow g^{r(H_y(\vec{m}) - sH_x(\vec{m}))} h^{-\chi s}}$; $\sigma_z \leftarrow g^{zr} h^{-\omega t}$

Procedure $\textsc{Mac}_{sk}(\vec{m})$ // $\mathsf{H}^A_{i,2}(k)$ / $\boxed{\mathsf{H}^A_{i}(k)}$

if $(j = i)$ then $r, \omega, \chi \xleftarrow{\$} \mathbb{F}_p$; $\sigma_w \leftarrow h^\omega$; $\sigma_x \leftarrow g^{rH_x(\vec{m})} h^\chi$, $\boxed{\sigma_x \leftarrow h^\chi}$;
$\sigma_y \leftarrow g^{r(H_y(\vec{m}) - sH_x(\vec{m}))} h^{-\chi s}$, $\boxed{\sigma_y \leftarrow g^{rH_y(\vec{m})} h^{-\chi s}}$; $\sigma_z \leftarrow g^{zr} h^{-\omega t}$

Figure 2: Games for the transition from $\mathsf{H}^A_{i-1}(k)$ to $\mathsf{H}^A_i(k)$. The boxed game uses the boxed code and the other games do not.

---

from $\mathsf{H}^A_{i-1}(k)$, we cannot proceed in a single step. Instead, we gradually change the value of the i-th query across two additional games $\mathsf{H}^A_{i,1}(k)$ and $\mathsf{H}^A_{i,2}(k)$, as shown in Figure 2.

We then show that

$$\Pr[\mathsf{H}^A_{i,1}(k)] - \Pr[\mathsf{H}^A_{i-1}(k)] \leq \mathbf{Adv}^{\mathrm{ddh}}_B(k) \tag{4.1}$$

$$\Pr[\mathsf{H}^A_{i,2}(k)] - \Pr[\mathsf{H}^A_{i,1}(k)] = 0 \tag{4.2}$$

$$\Pr[\mathsf{H}^A_{i}(k)] - \Pr[\mathsf{H}^A_{i,2}(k)] = 0 \tag{4.3}$$

for all $k \in \mathbb{N}$, from which Equation 4 follows.

To prove Equation 4.1, the construction of B is as follows:

$\underline{B(p, G, g, A, B, C)}$
$Q \leftarrow \emptyset$; $h \leftarrow A$
$X_i \leftarrow g^{x_i} h^{v_i}$; $Y_i \leftarrow g^{y_i} X_i^{-s}\ \forall\ i$; $iparams \leftarrow (\vec{X}, \vec{Y})$
$(\vec{m}^*, \sigma^*) \xleftarrow{\$} A^{\textsc{SimMac},\textsc{Verify}}((p, G, g, h), iparams)$
$b \leftarrow ((\sigma_y \sigma_x^s)^z = (\sigma_z \sigma_w^t)^{H_y(\vec{m}^*)}) \wedge ((\sigma_w^{-H_v(\vec{m}^*)} \sigma_x)^{H_y(\vec{m}^*)} = (\sigma_x^s \sigma_y)^{H_x(\vec{m}^*)}) \wedge (\sigma_w \neq 1)$
return $b' = ((\vec{m}^* \notin Q) \wedge b)$

$\underline{\text{Procedure } \textsc{SimMac}_{sk}(\vec{m})}$
$\sigma_w \leftarrow C$; $\sigma_x \leftarrow B^{H_x(\vec{m})} C^{H_v(\vec{m})}$; $\sigma_y \leftarrow B^{H_y(\vec{m}) - sH_x(\vec{m})} C^{-sH_v(\vec{m})}$; $\sigma_z \leftarrow B^z C^{-t}$
return $(\sigma_w, \sigma_x, \sigma_y, \sigma_z)$

$\underline{\text{Procedure } \textsc{Verify}_{sk}(\vec{m}, \sigma)}$
return $((\sigma_y \sigma_x^s)^z = (\sigma_x \sigma_w^t)^{H_y(\vec{m})}) \wedge (\sigma_w \neq 1)$

To see that B successfully simulates the Mac oracle, observe that if we implicitly use $r = b$ then, if $C = g^{ab}$, we have

$$\begin{aligned}
\sigma_w &= g^{ab} = h^b = h^r \\
\sigma_x &= g^{bH_x(\vec{m})} g^{abH_v(\vec{m})} = (g^{H_x(\vec{m})} h^{H_v(\vec{m})})^r \\
\sigma_y &= g^{b(H_y(\vec{m}) - sH_x(\vec{m}))} g^{-absH_v(\vec{m})} = (g^{H_y(\vec{m}) - sH_x(\vec{m})} h^{-sH_v(\vec{m})})^r \\
\sigma_z &= g^{bz} g^{-abt} = (g^z h^{-t})^r,
\end{aligned}$$

which are distributed identically to the values in $\mathsf{H}_{i-1}^A(k)$. If instead C is random, then in particular we can write it as $C = h^\omega$ for some $\omega \xleftarrow{\$} \mathbb{F}_p$. In thise case

$$\begin{aligned}
\sigma_w &= h^\omega \\
\sigma_x &= g^{rH_x(\vec{m})} h^{\omega H_v(\vec{m})} \\
\sigma_y &= g^{r(H_y(\vec{m}) - sH_x(\vec{m}))} h^{-s\omega H_v(\vec{m})} \\
\sigma_z &= g^{zr} h^{-\omega t},
\end{aligned}$$

which are distributed identically to the values in $\mathsf{H}_{i,1}^A(k)$.

To prove Equation 4.2, we remind ourselves of the transition: in both games, the first $i-1$ queries are answered using h^{χ_j} for $\chi_j \xleftarrow{\$} \mathbb{F}_p$ and the last $q_m - i$ queries are answered using $(g^{H_x(\vec{m}_j)} h^{H_v(\vec{m}_j)})^{r_j}$ for $r_j \xleftarrow{\$} \mathbb{F}_p$. The i-th query then uses either $g^{r_i H_x(\vec{m}_i)} h^{\chi_i}$ (in $\mathsf{H}_{i,2}^A(k)$) or $g^{r_i H_x(\vec{m}_i)} h^{\omega H_v(\vec{m}_i)}$ (in $\mathsf{H}_{i,1}^A(k)$); if we can argue that the value of $H_v(\vec{m}_i)$ is independent of any other values in the game, then in particular $H_v(\vec{m}_i)$ could take on any value and the distribution over these two values is identical. To do this, we first observe that $\mathsf{SimVerify}$, and thus Verify, is independent of $\vec{v}$. The first $i-1$ Mac responses are also independent of $\vec{v}$, so we must prove only two properties: (1) the value of $(g^{H_x(\vec{m}_j)} h^{H_v(\vec{m}_j)})^{r_j}$ in the last $q_m - i$ Mac responses is independent of $\vec{v}$, and (2) the value of $H_v(\vec{m}_i)$ is independent of the value of $H_v(\vec{m}^*)$, and thus changing the i-th query does not affect the distribution at the end of the game.

To first prove this latter property, we observe that $H_v(\cdot)$ is a pairwise independent function. As the winning conditions of the game require that $m^* \notin Q$ and thus $\vec{m}_i \neq \vec{m}^*$, this means that for any $\alpha_1, \alpha_2 \in \mathbb{F}_p$, $\Pr[H_v(\vec{m}_i) = \alpha_1 \wedge H_v(\vec{m}^*) = \alpha_2] = 1/p^2$, which in turn implies that the values of $H_v(\vec{m}_i)$ and $H_v(\vec{m}^*)$ are independent as desired.

To prove the former property, we perform a similar argument to that in the proof of Equation 3: consider a modified game in which $x_i = x'_i - \beta v_i$; then for the last $q_m - i$ queries,

$$(g^{(H_x(\vec{m}_j)}h^{H_v(\vec{m}_j)})^{r_j} = g^{r_j(H_x(\vec{m}_j+\beta H_v(\vec{m}_j)))} = g^{r_j(H_{x'}(\vec{m}_j)-\beta H_v(\vec{m}_j)+\beta H_v(\vec{m}_j))} = g^{r_j H_{x'}(\vec{m}_j)},$$

so these values information-theoretically hide $\vec{v}$. For the i-th query, however, if we use $g^{r_i H_x(\vec{m}_i)}h^{\omega_i H_v(\vec{m}_i)}$ then we have

$$g^{r_i(H_{x'}(\vec{m}_i)-\beta H_v(\vec{m}_i))}g^{\beta\omega_i H_v(\vec{m}_i)} = g^{r_i H_{x'}(\vec{m}_i)}h^{(\omega_i - r_i)H_v(\vec{m}_i)},$$

which, using instead $\omega'_i = \omega_i - r_i$, is distributed identically to the value in $\mathsf{H}^A_{i,1}(k)$.

To prove Equation 4.3, we consider a modified version of $\mathsf{H}^A_i(k)$ in which, rather than pick $\chi \xleftarrow{\$} \mathbb{F}_p$, pick $\chi' \xleftarrow{\$} \mathbb{F}_p$ and set $\chi := rH_x(\vec{m})/\beta + \chi'$. Then the distribution over χ is still uniformly random and thus identical to the distribution in $\mathsf{H}^A_i(k)$, and

$$h^{\chi} = h^{rH_x(\vec{m})/\beta+\chi'} = g^{\beta(rH_x(\vec{m})/\beta+\chi')} = g^{rH_x(\vec{m})}h^{\chi'},$$

so the distribution over σ_x in the modified game is identical to that in $\mathsf{H}^A_{i,2}(k)$ in which $\chi' \xleftarrow{\$} \mathbb{F}_p$ (and, consequently, so is the distribution over σ_y).

Equation 5.

Finally, we prove Equation 5. If the forgery $(\vec{m}^*, \sigma^*)$ output by A passes verification, then by definition $(\sigma_w^{-H_v(\vec{m}^*)}\sigma_x)^{H_y(\vec{m}^*)} = (\sigma_x^s\sigma_y)^{H_x(\vec{m}^*)}$. Since $\vec{x}$ and $\vec{v}$ are not used in any values given to A and thus $H_x(\vec{m}^*)$ and $H_v(\vec{m}^*)$ are information-theoretically hidden, however, A has a negligible probability of producing $(\vec{m}^*, \sigma^*)$ such that this equality holds and $\sigma_w \neq 1$, meaning the probability that it passes final verification is bounded by $\nu(k)$ for a negligible function $\nu(\cdot)$. □

B Formal Security Definitions for Keyed-Verification Credentials

In this section we formally define the security properties of keyed-verification credential scheme, introduced in Section 4.

To simplify the definition somewhat, we first consider the setting where the issuer sees all of the user's attributes when it issues the credential, and define correctness, unforgeability, and anonymity in this setting. Then we require the existence of a *blind issuing protocol*, which is a secure two party computation allowing the user to obtain credentials identical to those generated by Issue, while keeping a subset of his attributes private.

We also include two algorithms which are used to define security for the system:

$\mathsf{Issue}(sk, (m_1, \ldots, m_n))$ uses the secret key to generate a credential for attributes $(m_1, \ldots, m_n)$. This can be run directly, if the issuer is trusted to behave honestly and knows all the user's attributes, otherwise $\mathsf{BlindIssue}$ and $\mathsf{BlindObtain}$ should be used, as these allow the user to guarantee that the credential received is valid, and to hide some of his attributes.

$\mathsf{CredVerify}(sk, (m_1, \ldots, m_n), cred)$ uses the secret key to verify a credential. This is never run (because it reveals the attributes $(m_1, \ldots, m_n)$ as well as *cred* which may compromise the user's privacy), but is used to define the set of valid credentials for attributes $(m_1, \ldots, m_n)$ under the *sk*.

For security, we require the following five properties to hold.

Definition 5 (Correctness)**.** *Let Φ be the set of statements supported by a credential system, and $\mathcal{U}$ be the universe of attribute sets. Then a keyed-verification credential system* (CredKeygen, Issue, CredVerify, Show, ShowVerify) *is* correct *for Φ,$\mathcal{U}$ if for all for all $(m_1, \ldots, m_n) \in \mathcal{U}$, for all sufficiently large k,*

$$\Pr\big[params \xleftarrow{\$} \mathsf{Setup}(1^k); (sk, iparams) \xleftarrow{\$} \mathsf{CredKeygen}(params); \\ cred \xleftarrow{\$} \mathsf{Issue}(sk, (m_1, \ldots, m_n)) \ : \ \mathsf{CredVerify}(sk, (m_1, \ldots, m_n), cred) = 0\big] = 0$$

and for all $\phi \in \Phi$, $(m_1, \ldots, m_n) \in \mathcal{U}$ such that $\phi(m_1, \ldots, m_n) = 1$, for all sufficiently large k,

$$\Pr\big[params \xleftarrow{\$} \mathsf{Setup}(1^k); (sk, iparams) \xleftarrow{\$} \mathsf{CredKeygen}(params); cred \xleftarrow{\$} \mathsf{Issue}(sk, (m_1, \ldots, m_n)); \\ \mathsf{Show}(iparams, cred, (m_1, \ldots, m_n), \phi) \leftrightarrow \mathsf{ShowVerify}(sk, \phi) \to b \ : \ b = 0\big] = 0$$

The unforgeability property ensures an adversary cannot produce an accepting proof for a statement ϕ unless at least one of the attribute sets that he requested a credential for satisfies ϕ.

Definition 6 (Unforgeability)**.** *A presentation protocol* Show, ShowVerify *for keyed-verification credentials scheme* CredKeygen, Issue *is* unforgeable *if for all PPT adversaries A, there exists a negligible function ν such that for all k,*

$$\Pr\big[params \xleftarrow{\$} \mathsf{Setup}(1^k); \\ (iparams, sk) \xleftarrow{\$} \mathsf{CredKeygen}(params); \\ (state, \phi) \xleftarrow{\$} A(params, iparams)^{\mathsf{Issue}(sk,\cdot),\mathsf{ShowVerify}(sk,\cdot)} \\ A(state) \leftrightarrow \mathsf{ShowVerify}(sk, \phi) \to b \\ \text{such that } b = 1 \wedge (\forall (m_1, \ldots, m_n) \in Q, \phi(m_1, \ldots, m_n) = 0)\big] = \nu(k)$$

where Q is the list of all attribute sets $(m_1, \ldots, m_n)$ queried to the $\mathsf{Issue}(sk, \cdot)$ oracle, and all executions of ShowVerify *are required to be sequential.*

Definition 7 (Anonymity)**.** *A presentation protocol* Show, ShowVerify *for keyed-verification credentials scheme* CredKeygen, Issue *is* anonymous *if for all PPT adversaries A, there exists an efficient algorithm* SimShow*, and a negligible function ν such that for all k, for all $\phi \in \Phi$ and $(m_1, \ldots, m_n) \in \mathcal{U}$ such that $\phi(m_1, \ldots, m_n) = 1$, and for all params $\xleftarrow{\$} \mathsf{Setup}(1^k)$ and all $(iparams, sk) \xleftarrow{\$} \mathsf{KeyGen}(params)$, for all cred such that $\mathsf{CredVerify}(sk, (m_1, \ldots, m_n), cred) = 1$:*

$$\{\mathsf{Show}(iparams, cred, (m_1, \ldots, m_n), \phi) \leftrightarrow A \to state\} \approx \{\mathsf{SimShow}(iparams, sk, \phi)\},$$

i.e., the adversary's view given the proof can be simulated by SimShow *given only ϕ and a valid secret key corresponding to iparams.*

Note that the statement ϕ is known to A and may contain information about the attribute values, which may identify the user. Definition 7 ensures that the keyed-verification credential scheme's protocols are anonymous, modulo information revealed in ϕ.

Definition 8 (Blind issuance)**.** *Here we consider a setting where the user wishes to obtain credentials for attributes $(m_1, \ldots, m_n)$, and the issuer knows only some subset S of those attributes. Then we consider the following function: $f((S, params, iparams), (sk, r), (m_1, \ldots, m_n))$ on shared input $(S, params, iparams)$, issuer input (sk, r), and user input $(m_1, \ldots, m_n)$, returns $\perp$ to the issuer and returns to the user "params error" if $(iparams, sk)$ are not in the range of* $\mathsf{CredKeygen}(params)$*,*

"attribute error" if S does not agree with $(m_1, \ldots, m_n)$,and cred $\xleftarrow{\$}$ $\mathsf{Issue}(sk, (m_1, \ldots, m_n); r)$ *if neither of these errors occurs.*[2]

We say that an issuance protocol $\mathsf{BlindIssue}, \mathsf{BlindObtain}$ *is a* blind issuance *protocol for* Issue *if it is a secure two-party computation (against malicious adversaries) for the above function. See [20, Chapter 7] for a definition of secure two-party computation.*

Definition 9 (Key-parameter consistency)**.** *The key generation algorithm* $\mathsf{CredKeygen}$ *satisfies* key-parameter consistency *if for any PPT adversary A, the probability that A given params* $\xleftarrow{\$}$ $\mathsf{Setup}(1^k)$ *can produce $(iparams, sk_1, sk_2)$ such that $(iparams, sk_1)$ and $(iparams, sk_2)$ are both in the range of* $\mathsf{CredKeygen}(params)$ *is negligible (where the probability is over the choice of params and the random coins of A).*

Definition 10 (Secure keyed-verification credential system)**.** *We say that* $(\mathsf{CredKeygen}, \mathsf{CredVerify}, \mathsf{Issue}, \mathsf{BlindIssue}, \mathsf{BlindObtain}, \mathsf{Show}, \mathsf{ShowVerify})$ *is a secure keyed-verification credential system if these algorithms satisfy correctness, unforgeability, anonymity, blind issuance, and key-parameter consistency as defined above.*

C A Proof of Theorem 4

We present the following algorithms, which will be used to specify the form of valid credentials when we prove security of the scheme.

$\mathsf{Issue}(sk, (m_1, \ldots, m_n))$**:** Output *cred* $\xleftarrow{\$}$ $\mathsf{MAC}_{\mathsf{DDH}}(sk, (m_1, \ldots, m_n),)$.

$\mathsf{CredVerify}(sk, (m_1, \ldots, m_n), cred)$**:** Output the result of $\mathsf{Verify}_{\mathsf{DDH}}(sk, (m_1, \ldots, m_n), cred)$.

Proof. We will show that these algorithms satisfy correctness, unforgeability, anonymity, and blind issuance.

Correctness. For correctness we need to show two properties. The first follows directly from correctness of the MAC. To see the second, consider the following:

$\mathsf{Issue}(sk, (m_1, \ldots, m_n))$ generates credentials of the form $(u, u^{x_0+\sum_1^n x_i m_i}, u^{y_0+\sum_1^n y_i m_i}, u^z)$. Then if both Show and $\mathsf{ShowVerify}$ are executed honestly, then the proof π will be accepting by completeness of the proof system. Also, the honest Show will compute :

$$\begin{aligned}
V_x &= \frac{{\sigma_w}^{x_0} \prod_1^n {C_{m_i}}^{x_i}}{C_{\sigma_x}} \\
&= \frac{{\sigma_w}^{x_0} \prod_1^n {\sigma_w}^{m_i x_i} h^{x_i w_i}}{\sigma_x g^{r_x}} \\
&= \frac{{\sigma_w}^{x_0} \prod_1^n {\sigma_w}^{m_i x_i}}{\sigma_x g^{r_x}} \prod_{i=1}^n X_i^{w_i} \\
&= \frac{u^{x_0} \prod_1^n u^{m_i x_i}}{g^{r_x} u^{x_0+\sum_1^n x_i m_i}} \prod_{i=1}^n X_i^{w_i} \\
&= g^{-r_x} \prod_{i=1}^n X_i^{w_i}
\end{aligned}$$

so the verifier's check on V_x will succeed. A similar equality holds for V_y. Finally, since Issue produces $\sigma_z = u^z$, the verifier's final check will succeed and the verifier will accept.

[2]Here $\mathsf{Issue}(sk, (m_1, \ldots, m_n); r)$ means running $\mathsf{Issue}(sk, (m_1, \ldots, m_n))$ with randomness r.

Unforgeability. We have shown (Theorem 3) that $\mathsf{MAC}_{\mathsf{DDH}}$ is unforgeable under DDH. Suppose there exists an adversary A who can break the unforgeability property of our credential system. Then we can construct an algorithm B that breaks unforgeability of $\mathsf{MAC}_{\mathsf{DDH}}$ as follows:

B receives $params, iparams_{\mathsf{DDH}}$ and chooses random $C_{x_0}, C_{y_0}, C_z \xleftarrow{\$} Z_p$. It then sends $params$, $iparams = (iparams_{\mathsf{DDH}}, C_{x_0}, C_{y_0}, C_z)$ to A.

When A queries the Issue oracle, B forwards the query to its MAC oracle and returns the resulting tag.

When A queries the $\mathsf{ShowVerify}$ oracle: A sends $\sigma_w, \sigma_z, C_{m_1}, \ldots C_{m_n}, C_{\sigma_x}, C_{\sigma_y}$, and gives a proof π. If the proof π is invalid, B will return $\perp$. Otherwise B will run the proof of knowledge extractor to extract $\{m_i\}_1^n, r_x, r_y$. Then it will compute $\sigma_x = C_{\sigma_x} g^{-r_x}$ and $\sigma_y = C_{\sigma_y} g^{-r_y}$. Finally, it will query its Verify oracle with $(m_1, \ldots m_n), (\sigma_w, \sigma_x, \sigma_y, \sigma_z)$, and output the result.

In the final show protocol, B will again, extract $\{m_i\}_1^n, r_x, r_y$, and output $(m_1, \ldots m_n)$, $(\sigma_w, C_{\sigma_x} g^{-r_x}$, $C_{\sigma_y} g^{-r_y}$, $\sigma_z)$ as its forgery.

First, note that B's response to Issue queries will be identical to the honest Issue algorithm. Then, we argue that its response to $\mathsf{ShowVerify}$ queries will also with overwhelming probability be identical to the output of the honest algorithm. To see this, note that the proof of knowledge property guarantees that the extractor will succeed in producing a valid witness with all but negligible probability. Furthermore, if the extractor gives valid $\{m_i\}_1^n, r_x, r_y$ then

$$
\begin{aligned}
V_x &= \frac{{\sigma_w}^{x_0} \prod_1^n {C_{m_i}}^{x_i}}{C_{\sigma_x}} \\
\iff & g^{-r_x} \prod_1^n {X_i}^{w_i} = \frac{{\sigma_w}^{x_0} \prod_1^n ({\sigma_w}^{m_i} h^{w_i})^{x_i}}{C_{\sigma_x}} \\
\iff & g^{-r_x} \prod_1^n (h^{x_i})^{w_i} = \frac{{\sigma_w}^{x_0 + \sum_1^n m_i x_i} \prod_1^n h^{w_i x_i}}{C_{\sigma_x}} \\
\iff & C_{\sigma_x} g^{-r_x} = {\sigma_w}^{x_0 + \sum_1^n m_i x_i}
\end{aligned}
$$

And similarly $V_y = \frac{C_{r_y} {\sigma_w}^{y_0} \prod_1^n {C_{m_i}}^{y_i}}{C_{\sigma_y}}$ iff $C_{\sigma_y} g^{-r_y} = {\sigma_w}^{y_0 + \sum_1^n y_i m_i}$. The final check that the honest verifier makes guarantees that $\sigma_z = {\sigma_w}^z$. Thus, the honest verifier algorithm will accept iff $(\sigma_w, C_{\sigma_x} g^{-r_x}, C_{\sigma_y} g^{-r_y}, \sigma_z)$ would be accepted by $\mathsf{Verify}_{\mathsf{DDH}}$ for message $(m_1, \ldots, m_n)$.

Similarly, we can argue that B will extract a valid MAC from the final show protocol whenever $\mathsf{ShowVerify}$ would have output 1. Thus, if A can cause $\mathsf{ShowVerify}$ to accept for some statement ϕ that is not satisfied by any of the attribute sets queried to Issue, then B will extract a new message $(m_1, \ldots, m_n)$ and a valid tag for that message.

Anonymity. Let $\phi \in \Phi$ and $(m_1, \ldots, m_n) \in \mathcal{U}$ be such that $\phi(m_1, \ldots, m_n) = 1$. Let $(iparams, sk)$ be in the range of $\mathsf{CredKeygen}$, and let $cred$ be such that $\mathsf{CredVerify}(sk, cred, (m_1, \ldots, m_n)) = 1$.

Then $\mathsf{SimShow}(sk, \phi)$ behaves as follows: It chooses random values $\sigma_w, C_{\sigma_x}, C_{\sigma_y}, C_{m_1}, \ldots, C_{m_n} \xleftarrow{\$} G$. It then uses $\{x_i, y_i\}_0^n, z$ from sk to compute $\sigma_z = {\sigma_w}^z$, $V_x = \frac{{\sigma_w}^{x_0} \prod_1^n {C_{m_i}}^{x_i}}{C_{\sigma_x}}$, and $V_y = \frac{{\sigma_w}^{y_0} \prod_1^n {C_{m_i}}^{y_i}}{C_{\sigma_y}}$. It will run A with these values as the first message, and then simulate the proof of knowledge, and output whatever A outputs at the end of the proof.

First note $C_{\sigma_x}, C_{\sigma_y}, C_{m_1}, \ldots, C_{m_n}$ are distributed identically to those produced by Show. Next, note that for any $cred$ such that $\mathsf{CredVerify}(sk, cred, (m_1, \ldots, m_n)) = 1$, randomizing the credential will produce the same distribution as choosing random σ_w and computing $\sigma_x = {\sigma_w}^{x_0 + \sum_1^n x_i m_i}$, $\sigma_y =$

${\sigma_w}^{y_0+\sum_1^n y_i m_i}$, and $\sigma_z = {\sigma_w}^z$ for the values $z, \{x_i, y_i\}_0^n$ in sk. Thus, σ_w, σ_z will also be distributed identically to those produced by Show.

Finally, note that if we define $r_x, r_y, \{w_i\}$ to be the values such that $C_{\sigma_x} = {\sigma_w}^{x_0+\sum_1^n x_i m_i} g^{r_x}$, $C_{\sigma_y} = {\sigma_w}^{y_0+\sum_1^n y_i m_i} g^{r_y}$, and $C_{m_i} = u^{m_i} h^{w_i}$ for the random values $C_{\sigma_x}, C_{\sigma_y}, C_{m_1}, \ldots, C_{m_n}$ chosen by SimShow, then the calculation above in the proof of correctness shows that the V_x, V_y that SimShow computes will be identical to those that the honest Show would have produced.

By the zero knowledge property of the proof of knowledge, we conclude that the resulting view will be indistinguishable to that produced by the adversary interacting with Show.

Blind issuance. First, we consider the setting where all of the attributes are known to the issuer and we use the simpler algorithm. Consider the case where the user is corrupt. Then our 2PC simulator on shared input $(S, iparams)$ will receive the user's list of attributes $(m_1, \ldots, m_n)$ and forward it to the functionality. The functionality will return "attribute error" if $S \neq (m_1, \ldots, m_n)$ and otherwise it will return $cred$. If the error does not occur, the 2PC simulator will then send $cred$ and run the proof of knowledge ZK simulator to simulate the proof of correctness for $cred$. By zero knowledge, this will be indistinguishable from the real world.

Next, we consider the case where the issuer is corrupt. In this case our 2PC simulator will receive $cred = (\sigma_w, \sigma_x, \sigma_y, \sigma_z)$ from the issuer and run the verifier for the proof system. If the proof accepts, it will run the proof of knowledge extractor to extract $sk = (\{x_i\}_0^n, \{y_i\}_0^n, \tilde{x}, \tilde{y}, \tilde{z})$ and $r = \sigma_w$. It will send (sk, r) to the ideal functionality. By the proof of knowledge property, the credential sent in the real world is $\sigma_w, {\sigma_w}^{x_0} \prod_1^n ({\sigma_w}^{m_i})^{x_i}, \sigma_y = {\sigma_w}^{y_0} \prod_1^n ({\sigma_w}^{m_i})^{y_i}, \sigma_z = {\sigma_w}^z)$ which is exactly what would be produced by the ideal functionality on input the (sk, r) described above.

Then, we consider the more complex algorithm which allows hidden attributes. Consider the case where the user is corrupt. Then our 2PC simulator on shared input $(S, iparams)$ will receive the user's list of ciphertexts $(E_1, \ldots, E_n)$, and run the verification for the proof of knowledge. If the proof accepts, it will then use the proof of knowledge extractor to extract $\{m_i\}_{i \in \mathcal{H}}$ and send it along with the set S to the functionality. The functionality will return $cred = (\sigma_w, \sigma_x, \sigma_y, \sigma_z)$. The 2PC simulator will then compute an encryption E'_x of σ_x and an encryption E'_y of σ_y, send $(\sigma_w, \sigma_z, E'_x, E'_y)$ to the user, and use the ZK simulator to simulate the correctness proof. Note that in the real BlindIssue protocol, if $E_1, \ldots, E_n$ are encryptions of $g^{m_1}, \ldots, g^{m_n}$, then the resulting E_x, E_y will be distributed identically to a fresh encryption of ${\sigma_w}^{x_0} \prod_1^n ({\sigma_w}^{m_i})^{x_i}, {\sigma_w}^{y_0} \prod_1^n ({\sigma_w}^{m_i})^{y_i}$. Thus, these will be identical to what the simulator produces.

Next, we consider the case where the issuer is corrupt. In this case our 2PC simulator will generate encryptions E_i of 1 for all $i \in \mathcal{H}$, send them to A, and simulate the proof. It will then receive $cred = (\sigma_w, \sigma_x, \sigma_y, \sigma_z)$ from A and run the verification of the proof of knowledge; if the proof is accepting, it will run the proof of knowledge extractor to extract $sk = (\{x_i\}_0^n, \{y_i\}_0^n, \tilde{x}, \tilde{y}, \tilde{z})$ and $r = b$. It will send (sk, r) to the ideal functionality. To see that this will be indistinguishable from the real game, consider the following series of games. The first game G_1 is identical to the real game, except that instead of computing σ_x, σ_y by decrypting the ciphertexts E_x, E_y, we run the proof of knowledge extractor to extract sk, r and use those to form the credential by running Issue. By the proof of knowledge property and correctness and homomorphic properties of the encryption scheme, the credential sent in the real world is $(\sigma_w = g^b, {\sigma_w}^{x_0} \prod_1^n ({\sigma_w}^{m_i})^{x_i}, {\sigma_w}^{y_0} \prod_1^n ({\sigma_w}^{m_i})^{y_i}, {\sigma_w}^z, m_n)$ which is exactly what would be produced by the ideal functionality on input the (sk, r) described above. Next, in game G_2 we replace the proof of knowledge of the messages in E_i with a simulated proof - by zero knowledge this is indistinguishable. Finally, we note that the only difference between this game and the simulated game is that E_i is generated as an encryption of g^{m_i} rather than 1; thus the two games are indistinguishable by CPA-security of Elgamal encryption (which follows from DDH [28]).

Key-parameter consistency. This follows under the discrete log assumption from the binding property of the Pedersen commitment scheme. (Note that the discrete log assumption is implied by DDH.) □

D Instantiating Proofs of Knowledge

For our application we need a proof system that is zero knowledge and satisfies a strong proof of knowledge property. In our setting we propose two approaches to instantiating the proof system. The first is to use the Damgård protocol [15], which converts any sigma protocol into a three-round interactive zero-knowledge proof of knowledge secure under concurrent composition. This protocol requires trusted parameters but this restriction can be omitted in the random oracle model. The second option is to make the assumption that the stronger extraction property holds for Fiat-Shamir based proofs [17] in the random oracle model.

In particular, we need that the proof of knowledge property hold even when the adversary is given some information about previously extracted values, which can be modeled as access to an extraction oracle. (This comes up, for example, in the credential unforgeability proof, when we need to extract in order to answer the user's ShowVerify queries. For standard model proof protocols, when proofs are executed sequentially, this follows directly from the standard proof of knowledge property [4]. In the random oracle model, however, we don't know of any such implication. (See [18, p. 152] for a discussion of some of the issues in this setting.)

In our setting we propose two approaches to instantiating the proof system. The first is to use the Damgård protocol [15], as described above. To see that trusted parameters can be avoided in the random oracle model, consider the commitment scheme that chooses r and computes the commitment as $H(m; r)$. It is clear that in the random oracle model this will be a trapdoor commitment, since control of the random oracle can be used to open such a commitment to any message. Implementing the trapdoor commitment this way means we do not need any trusted setup besides the establishment of a secure hash function that can be modeled as a random oracle.[3]

The second option is to make the assumption that the stronger extraction property holds for Fiat-Shamir based proofs [17] in the random oracle model. While it is not obvious how to show that this property holds in the random oracle model, it seems like a reasonable assumption in the combined random oracle and generic group model, following along the lines of [26, 29]. Since our analysis for the $\mathsf{MAC}_{\mathsf{GGM}}$ scheme already uses this model, this may be a good choice for use with that scheme.

E Detailed Description of Show with $\mathsf{MAC}_{\mathsf{GGM}}$

We describe an instantiation of our presentation protocol and corresponding verification when the ZK proofs are implemented using non-interactive Schnorr proofs. This is the same proof system used in U-Prove and Idemix. This protocol does not include proof of any additional predicates ϕ, but outputs commitments which may be used as input to further proof protocols. H will denote a cryptographic hash function.

E.1 Proof generation

Inputs: *params*, a credential u_0, u_0', and attribute values $m_1, \ldots, m_n$.

1. (*Re-randomize*) Choose $a \in_R \mathbb{F}_p$, compute $u = {u_0}^a$ and $u' = {u_0'}^a$. Delete a.

[3] For alternative trapdoor commitment schemes that do not require a random oracle, see [15, Section 4]. These alternatives require trusted setup of a common reference string,

2. (*Form commitments*)

   (a) Choose $r, z_1, \ldots z_n \in_R \mathbb{F}_p$.

   (b) Compute $\{C_{m_i} := u^{m_i} h^{z_i}\}_{i=1}^n$, $C_{u'} := u' g^r$.

3. (*Create proof* π)

   (a) Choose $\tilde{z_1}, \ldots, \tilde{z_n}, \tilde{r}, \tilde{m_1}, \ldots, \tilde{m_n} \in_R \mathbb{F}_p$.

   (b) Compute $\{\tilde{C}_{m_i} := u^{\tilde{m_i}} h^{\tilde{z_i}}\}_{i=1}^n$, and $\tilde{V} = X^{\tilde{z_1}} \cdots X^{\tilde{z_n}} g^{\tilde{r}}$.

   (c) Form the challenge
   $$c = H(param \| \{C_{m_i}\}_{i=1}^n \| C_{u'} \| \{\tilde{C}_{m_i}\}_{i=1}^n \| \tilde{V})$$

   (d) Compute responses (all mod p), $\{s_{m_i} = \tilde{m_i} - c m_i, s_{z_i} = \tilde{z_i} - c z_i\}_{i=1}^n$, and $s_r = \tilde{r} + rc$. Let S denote the set of responses.

   (e) Output $\pi = (c, S)$.

4. (*Output*) Output the presentation proof $P = (u, \{C_{m_i}\}_{i=1}^n, C_{u'}, \pi)$.

E.2 Proof verification

Inputs: Presentation proof P, issuer and system parameters *param*, private key elements $x_0, x_1, \ldots, x_n$.

1. Parse P as $(u, C_{m_1}, \ldots, C_{m_n}, C_{u'}, \pi)$.

2. Compute
   $$V = \frac{{C_{m_1}}^{x_1} \cdots {C_{m_n}}^{x_n} u^{x_0}}{C_{u'}} ,$$

3. (*Verify* π)

   (a) Parse π as (c, S) where S contains the responses computed in Step 3d of proof generation.

   (b) Compute
   $$c' = H(param \| \{C_{m_i}\}_{i=1}^n \| C_{u'} \| \{C_{m_i} g^{s_{m_i}} h^{s_{z_i}}\}_{i=1}^n \| V X^{s_{z_1}} \cdots X^{s_{z_n}} g^{s_r})$$

   (c) Accept π as valid if $c' = c$, otherwise reject.

4. (*Output*) If π is valid, output $\{C_{m_i}\}_{i=1}^n$