MAC Schemes with Efficient Protocols and Keyed-Verification Anonymous Credentials

Melissa Chase, Greg Zaverucha
Microsoft

August 19, 2013

Abstract

We consider the problem of constructing anonymous credentials for use in a setting where the issuer of credentials is also the verifier, or 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 of the credential system.

To this end, we construct two algebraic MAC schemes in prime order groups, along with efficient protocols for issuing credentials, asserting possession a credential, and proving statements about the attributes. Security of the first scheme is proven in the generic group model, and we show that the second is secure under the decisional Diffie-Hellman (DDH) assumption, using a dual system-based approach.

Finally, we compare the efficiency of our new systems to two traditional credential systems, U-Prove and Idemix. We show that performance of the new schemes are competitive with U-Prove, and many times faster than Idemix. This brings together the best aspects of these two existing systems: the efficiency of U-Prove combined with the multi-show unlinkability of Idemix.

1 Introduction

Traditionally, anonymous credentials systems are constructed from public-key signature schemes. To issue a credential to a user certifying a set of *attributes*, the issuer generates a signature on those attributes. Later, when the user wants to use his credential, he presents a *zero knowledge proof of knowledge* of a valid signature on appropriate attributes; this allows him to demonstrate possession of a credential without revealing unnecessary attributes, and ensures that even if the issuer and verifier cooperate it remains infeasible to link a run of the presentation protocol using a given credential to the run of the issuance protocol in which that credential was generated.

In the U-Prove system [6, 21] (sometimes also called "Brands credentials") the issuer and user jointly compute the zero knowledge proof, so each proof requires an interaction with the issuer. In the Idemix system [7, 18] (Camenisch-Lysyanskaya signatures), the user can generate proofs on his own. This means that each time he presents his credential the user can generate a fresh proof, so that one credential can be presented many times, and the verifier will be unable to distinguish many presentations of the same credential from presentations of many different credentials with appropriate attribute sets. This property is often referred to as *multi-show unlinkability.*

Because they are constructed from signature schemes, known anonymous credential schemes are publicly verifiable, i.e., any party may verify a credential given the issuer's public key and the system parameters.

MAC based credentials. In many authentication scenarios, the party controlling access to a resource also manages the accounts of authorized parties. For example, a transit company will typically be responsible for issuing and managing transit cards. Online marketplaces and other online services may also choose to manage their own accounts, rather than accept credentials from outside identity providers. In these types of systems, public verifiability is not a requirement.

In this paper we introduce the notion of *keyed-verification* credentials to address the functionality needed in these systems. The main observation is that in many of these settings, it is natural to assume that the verifier and the issuer will possess a shared secret. As we will show, focusing on this simpler setting, without public verifiability, can lead to performance improvements. Our new constructions use MAC schemes as a basis for the credential. These MAC schemes are algebraic, constructed in prime order groups, rather than being constructed with block ciphers or hash functions. While algebraic MACs are much slower than traditional MACs, the additional structure of the tags allows for efficient proofs of knowledge using sigma protocols, for proving possession of a MAC on a (committed) message, and for securely obtaining a MAC on potentially hidden messages. Naturally, issuance and verification both require the secret key of the MAC scheme, which motivates the name keyed-verification credentials. By comparison, credential systems with multi-show unlinkability constructed from signature schemes seem to require either a strong RSA group [7], or a group with a bilinear pairing [1, 2, 8], resulting in slower protocols and much larger credentials. Among other things, this limits the devices which can practically use these credentials; for example Bischel et al. [4] report that a partial implementation of the Idemix presentation proof on a smart card required 7.4 seconds to complete (at a security level below 80 bits).

Other use cases. We stress that it is also possible to use keyed-verification credentials in the more traditional anonymous credential scenario with multiple verifiers. For a small number of verifiers, the issuer can issue separate keyed-verification credentials with the same attributes under separate secret keys shared with each of the verifiers. A user may obtain credentials for all verifiers at once, or if the user and issuer are online, credentials may be obtained as needed. Interestingly, if the user needs to return to the issuer to request credentials for additional verifiers, he can do so anonymously, and using blind issuance the new credential can be made to have the same attributes as the old one, without revealing these attributes to the issuer.

As we will describe, it is possible using our scheme to translate a publicly verifiable credential into a more efficient keyed-verification credential with the same attributes and functionality. Thus another use case is 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 which only he can verify. Importantly, this protocol does not require that the user reveal the attributes in the traditional credential, or allow the issuer and verifier together to link the credential used during enrollment with a run of the issuance protocol. When the user returns, the efficient keyed-verification credential is used, and again it will be impossible to link this use with previous presentations or with the enrollment. In the case of U-Prove, this allows a single-use credential to be used to bootstrap multiple unlinkable credential uses. In the case of Idemix, subsequent uses of the keyed-verification credential remain unlinkable, but are significantly more efficient. Translating credentials in this manner provides an appealing tradeoff; public verifiability is still possible when necessary, but credential use becomes more efficient with repeat verifiers.

1.1 Summary of Contributions and Paper Outline

The main contributions of this paper are the following.

New MAC schemes We present two new MAC schemes and analyze their security (§3). The first (§3.1), called $\mathsf{MAC}_{\mathsf{GGM}}$, is a generalization of a scheme presented by Dodis et al. [13] who proved that it satisfies a very weak notion of security. We generalize the scheme to support blocks of messages, and prove it satisfies standard MAC unforgeability (uf-cmva) in the generic group model. Our second scheme (§3.2), $\mathsf{MAC}_{\mathsf{DDH}}$, is new. We prove that $\mathsf{MAC}_{\mathsf{DDH}}$ is uf-cmva secure assuming DDH is hard.

The advantage of these constructions, when compared to the uf-cmva MAC schemes from [13] is that they are entirely algebraic, without collision resistant hash functions or bit-wise decompositions, which allows us to construct very efficient protocols for them. For this reason we feel that $\mathsf{MAC}_{\mathsf{DDH}}$ and $\mathsf{MAC}_{\mathsf{GGM}}$ will be useful primitives in other privacy protocols. The advantage of $\mathsf{MAC}_{\mathsf{DDH}}$ is that its provable security relies on weaker assumptions, while the advantage of $\mathsf{MAC}_{\mathsf{GGM}}$ is its simplicity which results in extra efficiency gains.

Another potentially interesting aspect of the $\mathsf{MAC}_{\mathsf{DDH}}$ construction is that its proof of security follows the outline of the dual system approach introduced by Waters [28, 16], making it the first (to our knowledge) use of dual system techniques in a setting without pairings.

Protocols We also demonstrate how these MAC schemes can be used to construct an efficient keyed-verification anonymous credential system. In particular, we show efficient protocols (§4) for issuing credentials on hidden attributes (*blind issuance*), and for proving possession of a credential with attributes satisfying a given statement (*credential presentation*). As the keyed-verification setting is somewhat different from that of previous credential systems, we also present formal definitions for the security required in this setting, and prove that our protocols satisfy these definitions. (This is deferred to Appendices B and C for lack of space.)

Efficiency Comparison We provide a detailed efficiency comparison of our new keyed-verification schemes to U-Prove and Idemix (§5). Our estimates are supported by an implementation of the dominant operations in each group for the cost of a presentation protocol. The results confirm that our goals are met; depending on the parameters of the presentation, our new schemes have the same or slightly higher cost when compared to U-Prove, and are always many times faster than Idemix (by our estimates at least 4 times, and up to 16 times faster).

1.2 Related Work on Anonymous Credentials

U-Prove [21] is a credential system constructed from a blind version of Schnorr signatures [6]. U-Prove 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. Therefore, the size of U-Prove credentials are linear in the number of unlinkable uses.

Idemix [18] is based on the Camenisch-Lysyanskaya [8] signature scheme (CL signatures). In terms of performance Idemix and U-Prove credentials make the 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 is hard (SRSA). 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 [4]).

There are also versions of the CL signature scheme defined in bilinear groups [1, 8], and Belenkiy [2] et. al. construct a signature scheme with Groth-Sahai proofs of knowledge supporting delegation. However, the applicable signature schemes 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.

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

2 Preliminaries

Notation We use the notation $x \in_R X$ or $x \leftarrow X$ 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 only used when the set, sum or product has only a single index.

[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.

Zero Knowledge Proofs The protocols that form our credential system make use of zero knowledge (ZK) proofs to prove knowledge of, and relations between discrete logarithms. In our presentation we abstract these protocols with a notation introduced by Camenisch and Stadler [9]. 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 the protocols in this paper 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 [15]. We propose two approaches to instantiate the proof system. The first is to use the Damgård protocol [12], 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 [14] in the random oracle model satisfy the required extraction property. For more discussion, see Appendix C.5.

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 [21]. Formally, we will 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 $a, b, c \in_R \mathbb{Z}_p$; 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 a precise definition, see [5, 13].

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 [5, 24].

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 [20].

3 MAC Schemes in Prime Order Groups

In this section we present two MAC schemes 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 $\mathit{params} := (G, p, g, h)$.

In addition to the Setup algorithm, MAC schemes 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 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 $\mathbb{Z}_p^n$, 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 [13]). We augment the definition slightly to guarantee security even when the signer publishes some parameters *IParams* associated with his secret key. In our application, we will use *IParams* to implement an efficient presentation protocol.

Definition 1 (uf-cmva security)**.** $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{MAC}, \mathsf{Verify})$ *is a secure MAC if for any PPT adversary $\mathcal{A}$, there exists a negligible function ν such that for all k*

$$\begin{aligned}\Pr\big[&\mathit{params} \leftarrow \mathsf{Setup}(1^k), (\mathit{IParams}, \mathit{sk}) \leftarrow \mathsf{KeyGen}(\mathit{params});\\ &(m, \sigma) \leftarrow \mathcal{A}^{\mathsf{MAC}(\mathit{sk},\cdot), \mathsf{Verify}(\mathit{sk},\cdot,\cdot)}(\mathit{params}, \mathit{IParams})\\ &\textit{such that } m \notin Q \wedge \mathsf{Verify}(\mathit{sk}, m, \sigma) = 1\big] = \nu(k)\end{aligned}$$

where Q is the list of messages $\mathcal{A}$ queried to the oracle $\mathsf{MAC}(\mathit{sk}, \cdot)$.

A stronger security notion for MAC schemes is sometimes used, where $\mathcal{A}$ 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), to 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 from [13]. The original MAC scheme works in a cyclic group G of prime order p, and the secret key is a pair $(x_0, x_1) \in \mathbb{Z}_p^2$. To compute the MAC of a message $m \in \mathbb{Z}_p$, 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 $\mathbb{Z}_p$; this is not the binary decomposition of a single message m. We refer to this scheme as $\mathsf{MAC}_{\mathsf{GGM}}$ (the single message version was called $\mathsf{MAC}_{\mathsf{hwPRF}}$ in [13]). The optional step in KeyGen is only required when $\mathsf{MAC}_{\mathsf{GGM}}$ is used for keyed-verification credentials.

$\mathsf{KeyGen}(\mathit{params})$ Choose a secret key $sk := (x_0, x_1, \ldots, x_n) \in_R (\mathbb{Z}_p)^{n+1}$. Optionally, choose $\tilde{x_0} \in_R \mathbb{Z}_p$ 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

$$\mathit{IParams} := (C_{x_0}, X_1, \ldots, X_n)\ .$$

$\mathsf{MAC}(sk, m)$ Parse m as $(m_1, \ldots, m_n) \in \mathbb{Z}_p^n$, and sk as $(x_0, x_1, \ldots, x_n) \in \mathbb{Z}_p^{n+1}$. Choose $u \in_R G \setminus \{1\}$ and compute the tag $\sigma = (u, u')$ where

$$u' := u^{x_0 + \sum_1^n m_i x_i}\ .$$

$\mathsf{Verify}(sk, m, \sigma)$ Parse m as $(m_1, \ldots, m_n) \in \mathbb{Z}_p^n$, and sk as $(x_0, x_1, \ldots, x_n) \in \mathbb{Z}_p^{n+1}$, and σ as $(u, u') \in G^2$. Accept iff $u \neq 1$ and

$$u' = u^{x_0 + \sum_1^n m_i x_i}\ .$$

Security Dodis et al. [13] prove that under the DDH assumption, $\mathsf{MAC}_{\mathsf{GGM}}$ is suf-cma secure. In this definition, security is called *selectively unforgeable*, 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 as the adversary will always be 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. 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 $\mathsf{Setup}(1^k)$, defined at the beginning of this section, and are assumed to be available to all algorithms, and that $(m_1, \ldots, m_n)$ is a list of n messages in $\mathbb{Z}_p$.

$\mathsf{KeyGen}(params)$**:** Choose random values $\{x_i\}_0^{n+1}, \{y_i\}_0^{n+1}, z$ from $\mathbb{Z}_p$. Store $sk := (\{x_i, y_i\}_0^{n+1},\ z)$. Optionally compute $X_i := h^{x_i}$ and $Y_i := h^{y_i}$ for each $i \in \{1, \ldots n+1\}$, and publish $IParams := (\{X_i, Y_i\}_1^{n+1})$.

$\mathsf{MAC}(sk, m)$**:** On input $m = (m_1, \ldots, m_n) \in \mathbb{Z}_p^n$, choose $u \in_R G \setminus \{1\}$, $m_{n+1} \in_R \mathbb{Z}_p$. Compute and output

$$\sigma := (u, u^{x_0 + \Sigma_1^{n+1} x_i m_i}, u^{y_0 + \Sigma_1^{n+1} y_i m_i}, u^z, m_{n+1})\ .$$

$\mathsf{Verify}(sk, m, \sigma)$**:** Parse σ as $(\sigma_W, \sigma_X, \sigma_Y, \sigma_Z, m_{n+1}) \in G^4 \times \mathbb{Z}_p$. Accept if $\sigma_W \neq 1$ and

$$\sigma_X = {\sigma_W}^{x_0 + \sum_1^{n+1} x_i m_i} \text{ and } \sigma_Y = {\sigma_W}^{y_0 + \sum_1^{n+1} y_i m_i} \text{ and } \sigma_Z = {\sigma_W}^{z} .$$

The optional step in KeyGen is only required when $\mathsf{MAC_{DDH}}$ is used for keyed-verification credentials. The randomly chosen value m_{n+1} is not needed if, in a particular application, one of the messages is chosen by the signer at random. Another possible optimization to reduce the size of the tag is to derive m_{n+1} from u with a cryptographic hash function; in the random oracle model, this will be equivalent since u is chosen at random for each tag.

Theorem 3. $\mathsf{MAC_{DDH}}$ *is uf-cmva secure, assuming the DDH problem is hard in G.*

Proof. First, we define a few alternate algorithms which we will use in the proof. The first, KeyGen', is a modified key generation algorithm that produces *IParams* and *sk* distributed identically to KeyGen, but also produces some additional values sk'. MAC' uses sk' and produces tags identical to those produced by MAC. SimMAC is a simulated MAC algorithm in which σ_W and σ_Z are not correctly formed. $\mathsf{FinalVerify}$ is designed to accept tags produced by MAC and reject those produced by SimMAC. Finally, $\mathsf{SimVerify}$ is designed to accept tags produced by either MAC or SimMAC.

Briefly, the proof will begin with the real unforgeability game, then argue that the adversary's success probabiliy will not decrease too much if verification queries are evaluated using $\mathsf{SimVerify}$, and the final tag is considered a success if it is accepted by $\mathsf{SimVerify}$ and $\mathsf{FinalVerify}$ rather than Verify. Then we argue that the adversary's success probability will be roughly the same if he is given tags issued by SimMAC instead of MAC. Finally, we observe that once the adversary is only given access to oracles for SimMAC and $\mathsf{SimVerify}$ rather than MAC and Verify, it has only negligible probability of producing a tag which is accepted by $\mathsf{SimVerify}$ and $\mathsf{FinalVerify}$.

In the description of these algorithms and in the rest of the proof, we will abuse notation: for $m = m_1, \ldots, m_{n+1} \in \mathbb{Z}_p^n$ we will write $H_a(m)$ to denote $a_0 + \sum_1^{n+1} a_i m_i$.

$\mathsf{KeyGen}'(params)$**:** Choose values $z', s, t, \{x'_i\}_0^{n+1}, \{y'_i\}_0^{n+1}, \{v_i\}_0^{n+1} \in_R \mathbb{Z}_p$. Implicitly let $w = \log_g h$.[2] Compute $Z = g^{z'} h^{-t}$, and for each $i \in \{0, \ldots n+1\}$ compute $X_i = g^{x'_i} h^{v_i}$, and $Y_i = g^{y'_i} X_i^{-s}$.

Also compute $z = z'/w - t$, and for each $i \in \{0, \ldots n+1\}$ compute $x_i = x'_i/w + v_i$ and $y_i = y'_i/w - s x_i$.

Set $IParams = (\{X_i, Y_i\}_1^{n+1})$, and optionally publish it.

[2]This discrete logarithm will only be used in our information theoretic proof steps.

Store $sk' = (\{x'_i, y'_i, v_i\}_0^{n+1}, s, z', t, X_0, Y_0, Z, \mathit{IParams})$, and $sk = (\{x_i, y_i\}_0^{n+1}, z, \mathit{IParams})$.

$\mathsf{MAC}'(sk', m)$**:** Extract $\mathit{IParams}, Z, X_0, Y_0$ from sk'. Parse $m = m_1, \ldots m_n$. Choose $r \in_R \mathbb{Z}_p^*$, $m_{n+1} \in_R \mathbb{Z}_p$. Compute and output

$$\sigma = (h^r, (X_0 \prod_{i=1}^{n+1} X_i^{m_i})^r, (Y_0 \prod_{i=1}^{n+1} Y_i^{m_i})^r, Z^r, m_{n+1}).$$

$\mathsf{SimMAC}(sk', m)$**:** Extract $\{y'_i\}_0^{n+1}, s, z', t$ from sk'. Parse $m = m_1, \ldots m_n$. Choose $r, m_{n+1} \in_R \mathbb{Z}_p$ and $\sigma_W \in_R G \setminus \{1\}$ and $\sigma_X \in_R G$, let $\hat{m} = (m_1, \ldots, m_{n+1})$ and output

$$\sigma = (\sigma_W, \sigma_X, (g^{H_{y'}(\hat{m})})^r {\sigma_X}^{-s}, g^{z'r} {\sigma_W}^{-t}, m_{n+1}).$$

$\mathsf{FinalVerify}(sk', m, \sigma)$**:** Extract $\{x'_i, y'_i, v_i\}_0^{n+1}, s$ from sk' and ignore the rest. Parse σ as $(\sigma_W, \sigma_X, \sigma_Y, \sigma_Z, m_{n+1})$ and m as $m_1, \ldots m_n$ and let $\hat{m} = (m_1, \ldots, m_{n+1})$. Verify σ_X, σ_Y w.r.t. σ_W, i.e., accept iff

$$({\sigma_W}^{-H_v(\hat{m})} \sigma_X)^{H_{y'}(\hat{m})} = ({\sigma_X}^s \sigma_Y)^{H_{x'}(\hat{m})}$$

$\mathsf{SimVerify}(sk', m, \sigma)$**:** Extract $\{y'_i\}_0^{n+1}, z', s, t$ from sk' and ignore the rest. Parse σ as $(\sigma_W, \sigma_X, \sigma_Y, \sigma_Z, m_{n+1})$ and m as $m_1, \ldots m_n$, and let $\hat{m} = (m_1, \ldots, m_{n+1})$. Accept iff $\sigma_W \neq 1$ and

$$(\sigma_Y {\sigma_X}^s)^{z'} = (\sigma_Z {\sigma_W}^t)^{H_{y'}(\hat{m})}.$$

Then our proof will proceed by considering the following sequence of games:

Real Game: The real uf-cmva game using $\mathsf{KeyGen}, \mathsf{MAC}, \mathsf{Verify}$ as in Definition 1.

Game G_1: As above except that keys are generated by KeyGen', tags are generated by MAC', and when the adversary produces a forgery, the adversary succeeds only if $\mathsf{Verify}(sk, m, \sigma) = 1$ and $\mathsf{FinalVerify}(sk', m, \sigma) = 1$.

Claim 4. *The adversary's success probability in Game G_1 will be identical to that in the Real Game.* This follows directly from the observation that KeyGen and KeyGen' produce identically distributed $\mathit{IParams}, sk$, and the output of MAC' on sk' is identical to the output of MAC on sk, and the observation that any tag that is accepted by Verify will also be accepted by $\mathsf{FinalVerify}$.

Game G_2: As in game G_1 except that Verify is replaced by SimVerify both in the adversary's verification queries, and in the final verification of the adversary's forgery.

Claim 5. *The adversary's success probability in games G_1 and G_2 differ by at most a negligible amount.* We will show that this follows from an information theoretic argument: since (s, t) are information theoretically hidden from $\mathcal{A}$, it has only negligible probability of finding a tag on which Verify and SimVerify behave differently. See Appendix A.2 for details.

Game H_i: As in game G_2 except that the first i MAC queries made by the adversary are answered using SimMAC. (Note that games H_0 and G_2 are identical.)

Claim 6. *Let Q be a polynomial upper bound on the number of MAC queries made by the adversary. For all $i \in \{1, \dots Q\}$, the adversary's success probability in games H_{i-1} and H_i differs by at most a negligible amount.* We will show that this follows from DDH. At a high level, view the values in the ith tag as $\sigma_W = h^r$, $\sigma_X = (g^r)^{H_{x'}(\hat{m})}(h^r)^{H_v(\hat{m})}$, $\sigma_Y = (g^r)^{H_{y'}(\hat{m})}{\sigma_X}^{-s}$, $\sigma_Z = (g^r)^{z'}{\sigma_W}^{-t}$. First choose m_{n+1} such that $H_v(\hat{m}) = 0$, and view (h, g^r, h^r) as a DDH triple so that σ_W can be replaced by a random element. Then choose m_{n+1} at random and note that again we have a DDH triple (h, g^r, h^r) where now h^r only appears in σ_X, so we can replace σ_X with a random element. See Appendix A.2 for details.

Game G_3: In this game all of the adversary's MAC queries are answered using SimMAC, and as in game G_2, the verification queries are answered using SimVerify, and the forgery is considered a success iff it is accepted by both SimVerify and FinalVerify. (Note that games G_3 and H_Q are identical.)

Claim 7. *The adversary's success probability in game G_3 is at most negligible.* We will show this through an information-theoretic argument: informally this follows from the fact that x_0 (and thus x'_0, v_0 and $H_{x'}(m), H_v(m)$) is information theoretically hidden from $\mathcal{A}$ in this game. See Appendix A.2 for details.

We conclude that the adversary's success probability in the real game is at most negligible. □

4 Protocols for Keyed-Verification Credentials

In this section we first describe the set of algorithms that form a keyed-verification credential scheme. Then we informally describe the desired security and privacy properties (formal definitions are in Appendix B). We present a construction based on $\mathsf{MAC}_{\mathsf{GGM}}$ (§4.1), and one based on $\mathsf{MAC}_{\mathsf{DDH}}$ (Appendix C). We give a formal proof of security for the scheme from $\mathsf{MAC}_{\mathsf{DDH}}$ (§C.4); 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 in §C.4.

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.

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

$\mathsf{BlindIssue}(sk, S) \leftrightarrow \mathsf{BlindObtain}(\mathit{IParams}, (m_1, \ldots, m_n))$ 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 $\mathsf{ShowVerify}$ is run by the verifier in possession of *sk* to verify proof π claiming knowledge of a credential satisfying the statements ϕ.

Discussion We defined our presentation protocol in terms of a single credential, however we could generalize our definitions and constructions to allow the user to prove relationships between attributes across multiple credentials that they own. We chose the above variant because it allows for fairly simple definitions, and 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 [19], but it has the additional complication that the statement (validity of the credential) depends on the verifier's secret key.

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 he has received credentials. *Anonymity* says that the proofs produced by Show reveal nothing more than the statement being proved. *Blind Issuance* requires that $\mathsf{BlindIssue}, \mathsf{BlindObtain}$ define a secure two party protocol for generating credentials on the user's attributes. Finally, we require *Key-Parameter Consistency*, which says 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.

4.1 A Keyed-Verification Credential Scheme From $\mathsf{MAC}_{\mathsf{GGM}}$

We will now give a construction of a keyed-verification credential system using $\mathsf{MAC}_{\mathsf{GGM}}$. Let $\mathsf{Setup}_{\mathsf{GGM}}, \mathsf{KeyGen}_{\mathsf{GGM}}, \mathsf{MAC}_{\mathsf{GGM}}, \mathsf{Verify}_{\mathsf{GGM}}$ be the algorithms described in Section 3.1. Then we define the following setup algorithms for the credential system.

$\mathsf{Setup}(1^k)$ will be the same as $\mathsf{Setup}_{\mathsf{GGM}}$.

$\mathsf{CredKeygen}(\mathit{params})$**:** Parse params as (G, p, g, h). Run $\mathsf{KeyGen}_{\mathsf{GGM}}(\mathit{params})$ to obtain $\mathit{IParams}_{\mathsf{GGM}} = (\{X_i\}_1^n, C_{x_0})$ and $sk_{\mathsf{GGM}} = (\{x_i\}_0^n, \tilde{x}_0)$. Output $\mathit{IParams}$ and sk_{GGM}.

4.1.1 Issuance

Here we describe the protocols $\mathsf{BlindIssue}$ and $\mathsf{BlindObtain}$ for issuing credentials. We first consider a simplified issuing protocol that can be used if all of the attributes are known to the issuer. Then we present the more general protocol allowing for hidden attributes. Section 2 has a discussion of some possible instantiations of the ZK proofs.

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

$$\begin{aligned}\pi = \mathrm{PK}\{(x_0, x_1, \ldots, x_n, \tilde{x_0}) : u' &= (u^{m_1})^{x_1} \cdots (u^{m_n})^{x_n} u^{x_0} \\ &\wedge\ X_1 = h^{x_1}\ \wedge\ \ldots\ \wedge\ X_n = h^{x_n} \\ &\wedge\ C_{x_0} = g^{x_0} h^{\tilde{x_0}}\}\end{aligned}$$

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 $\perp$.

Issuance with hidden attributes To keep some 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 follows: $E_i = (g^{r_i}, g^{m_i}\gamma^{r_i})$, using $r_i \in_R Z_p$. The ciphertexts are sent to the issuer (along with a proof of knowledge of $\{r_i, m_i\}$). The issuer chooses $b \in_R Z_p^*$. 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 $r' \leftarrow Z_p$). 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 proof that ciphertexts E_i are well formed, the user can include proof 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, perhaps one which is more expensive to use (such as an Idemix credential), or one that cannot be presented multiple times unlinkably (such as a U-Prove credential).

4.1.2 The Show and ShowVerify Protocols

Here we present a construction for Show and ShowVerify. The details of (one possible way of) instantiating the proof of knowledge are given in Appendix D.

The input to Show must be a rerandomized $\mathsf{MAC}_{\mathsf{GGM}}$ credential. To rerandomize (u, u'), choose $r \in_R \mathbb{Z}_q$, and output the new credential $(u^r, (u')^r)$. The value r can be deleted.

Show The inputs are *params* and *IParams*, the statement ϕ, the rerandomized credential (u, u'), and the attribute values $m_1, \ldots, m_n$. The Prover chooses values $r, z_1, \ldots, z_n, \in_R \mathbb{Z}_q$, computes $\{C_{m_i} := u^{m_i} h^{z_i}\}_{i=1}^{n}$, $C_{u'} := u' g^r$, sends $\sigma = (u, C_{m_1}, \ldots, C_{m_n}, C_{u'})$ and gives a proof of knowledge

$$\begin{aligned}\mathrm{PK}\{(m_1, \ldots, m_n, z_1, \ldots, z_n, -r) : C_{m_1} = u^{m_1} h^{z_1} \ \wedge\ \ldots\ \wedge\ C_{m_n} = u^{m_n} h^{z_n} \\ \wedge\ V = {X_1}^{z_1} \cdots {X_n}^{z_n} g^{-r} \ \wedge\ \phi(m_1, \ldots, m_n) = 1\}\ .\end{aligned}$$

ShowVerify The inputs are *params*, the statement ϕ and the secret key values $x_1, \ldots, x_n$ and y. The verifier computes V as

$$V = \frac{{C_{m_1}}^{x_1} \cdots {C_{m_n}}^{x_n} u^y}{C_{u'}}\ ,$$

then verifies the proof using V. If the proof is valid, output $(C_{m_1}, \ldots, C_{m_n})$ otherwise output $\perp$.

Correctness To see that the 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 predicate in the proof π.

5 Efficiency

In this section we compare the efficiency of our new schemes to U-Prove and Idemix. We will 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 D, and $\mathsf{MAC}_{\mathsf{DDH}}$ is very similar). Using the proof system from [12] will have essentially the same computational cost (not including communication time). Complete descriptions of Idemix and U-Prove are available in [18] and [21] respectively. We did not include the bilinear CL signature schemes [1, 8] 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
U-Prove	$O(s)$	$1024s$ bits
Idemix	$O(1)$	5369 bits
$\mathsf{MAC}_{\mathsf{GGM}}$	$O(1)$	512 bits
$\mathsf{MAC}_{\mathsf{DDH}}$	$O(1)$	1024 bits

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 [18]. 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 [18]. 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.

		Time when $(n, c, r) =$	
	Number of exponentiations	(10,2,2)	(10,10, 0)
U-Prove	1 $(n - r + 1)$-exp, $2c$ 2-exp	3.38 ms	12.43 ms
$\mathsf{MAC}_{\mathsf{GGM}}$	3 1-exp, 1 $(n - r + 1)$-exp $2(n - r)$ 2-exp	11.42 ms	13.93 ms
$\mathsf{MAC}_{\mathsf{DDH}}$	6 1-exp, 2 $(n - r + 2)$-exp $2(n - r + 1)$ 2-exp,	15.31 ms	18.10 ms
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 ms	226.79 ms

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, will 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 $\mathsf{MAC}_{\mathsf{DDH}}$ will 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. The parameter set used for Idemix is not believed to provide 128-bit security, and this favors Idemix in the comparison. For RSA, a 3072-bit modulus is required for 128-bit security, 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 parameter choices (n, c, r), which will be different in almost every application. 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] 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.

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

[4] P. Bichsel, J. Camenisch, T. Groß, and V. Shoup. Anonymous Credentials on a Standard Java Card. *Proceedings ACM CCS'09*, ACM Press, (2009) 600–610.

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

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

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

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

[9] 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.

[10] 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.

[11] 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.

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

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

[14] 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.

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

[16] 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.

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

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

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

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

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

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

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

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

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

[26] 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.

[27] 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.

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

A Security of MACs

This section completes the proofs of Theorems 2 and 3.

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 $\mathbb{Z}_p$, however it may easily be generalized to the message space $\mathbb{Z}_p^n$. 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 $\log_g a \in \mathbb{Z}_q$. We encode elements of G as random strings in a set $S \in \{0,1\}^*$ with the function $\zeta : \mathbb{Z}_p \to S$ (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 $(x, y, h) \in_R \mathbb{Z}_q^3$. 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 $\mathbb{Z}_p[\overline{x}, \overline{y}, \overline{h}, \overline{z_1}, \ldots, \overline{z_{q_t}}]$, 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 $(F_i, \zeta_i) \in \mathbb{Z}_p[\overline{x}, \overline{y}, \overline{h}, \overline{z_1}, \ldots, \overline{z_{q_t}}] \times S$. 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 $m_i \in \mathbb{Z}_p$. 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 rerandomize algorithm by repeated calls to the group operation oracle.

Verify query: The input from A is $(m, \zeta, \zeta') \in \mathbb{Z}_p \times S \times S$. 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 $i, j \leq q$. 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 $(x, y, h, z_1, \ldots, z_{q_t}) \in \mathbb{Z}_p^{3+q_t}$, 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 probability that each of (1) or (2) holds is $2/p$, since the degree of the polynomial in each case is at most 2 (see [24, Lemma 1]). Therefore the probability over all pairs (i, j) is

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

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

$$\begin{aligned}\epsilon &= \binom{q}{2} \cdot \frac{4}{p} \\ &\le \frac{4q^2}{p}\end{aligned}$$

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

$\square$

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

Here we prove the claims stated in Section 3.2 to complete the proof of Theorem 3.

Proof of Claim 5. Here we consider a series of games V_i. Let Q_v be an upper bound on the number of verification queries that an adversary can make. In all the games, keys will be generated with KeyGen', and MAC queries will be answered using MAC. In the ith game, the first i verification queries will be answered using Verify, and the remaining $Q_v + 1 - i$ by $\mathsf{SimVerify}$. When the adversary produces a forgery, we will verify it using either $\mathsf{SimVerify}$ or Verify (we will consider this to be the $Q_v + 1$th verification query), and then again using $\mathsf{FinalVerify}$, and the adversary succeeds if both verifications accept. Note that the Q_v+1th game is identical to G_1, and because MAC and MAC' produce identically distributed tags the 0th game is identical to G_2.

Now we argue that for all $i \in 1, \ldots, Q_v + 1$, the adversary's success probability in game V_i differs from his success probability in V_{i-1} by at most a $1/2^k$.

To see this, note that the two games are identical until the ith verification. Let E_i be the event that in the ith verification, the adversary produces a forgery for which $\mathsf{SimVerify}$ accepts but Verify doesn't. Then, the games are identical as long as E_i does not occur, so the adversary's success probability can differ by at most $\Pr[E_i]$. (Note that $\mathsf{SimVerify}$ will always accept queries that are accepted by Verify.) We argue that this probability must be at most $1/2^k$: let $m = (m_1, \ldots, m_n), \sigma = (\sigma_W, \sigma_X, \sigma_Y, \sigma_Z, m_{n+1})$ be the message and tag in the adversary's ith verification query, and let $\hat{m} = (m_1, \ldots, m_{n+1})$. Let R_X, R_Y, R_Z be such that

$$\sigma_X = R_X {\sigma_W}^{H_x(\hat{m})} \text{ and } \sigma_Y = R_Y {\sigma_W}^{H_y(\hat{m})} \text{ and } \sigma_Z = R_Z {\sigma_W}^{z}.$$

Note that Verify accepts iff R_X, R_Y, R_Z are all 1. Now, if $\mathsf{SimVerify}$ accepts, we know that $(\sigma_Y {\sigma_X}^s)^{z'} = (\sigma_Z {\sigma_W}^t)^{H_{y'}(\hat{m})}$; rewriting using the choice of $z, \{y_i\}$ given by KeyGen' we get $(\sigma_Y {\sigma_X}^s)^{z'} = (\sigma_Z {\sigma_W}^{z'/w - z})^{wH_y(\hat{m}) + swH_x(\hat{m})}$; substituting the values above, this means $(R_Y {R_X}^s)^{z'} = {R_Z}^{wH_y(\hat{m}) + swH_x(\hat{m})}$. Finally, we observe that the adversary's view up until the ith query, and the values $w, z, \{x_i, y_i\}_0^{n+1}$ are independent of the pair (s, z'). (Note that the distribution of *IParams*, *sk* is independent of s, z', and the game up until the ith verification only requires *IParams*, *sk*.) Thus, the probability that the adversary produces $(R_X, R_Y, R_Z) \neq (1,1,1)$ such that $(R_Y {R_X}^s)^{z'} = {R_Z}^{wH_y(\hat{m})+swH_x(\hat{m})}$ is at most $1/2^k$, which means that the probability of event E_i, i.e., that the adversary produces an ith verification query which is accepted by SimVerify and rejected by Verify is also $1/2^k$.

Since Q_v is at most polynomial in k, the claim follows. □

Proof of Claim 6. Consider an additional game H_i' which behaves like H_i and H_{i-1} except that the ith tag is generated as follows:

SimMAC$'(sk', m, m_{n+1})$: Extract *IParams*, $\{x_i, y_i, v_i\}_0^{n+1}, s, z', t$ from sk' and ignore the rest. Parse $m = m_1, \dots m_n$ and let $\hat{m} = (m_1, \dots, m_{n+1})$. Choose random $r \leftarrow Z_p^*$ and random $\sigma_W \leftarrow G \setminus \{1\}$. Compute and output

$$\sigma = (\sigma_W, (X_0 \prod_{i=1}^{n+1} {X_i}^{m_i})^r, (g^{H_{y'}(\hat{m})}(X_0 \prod_{i=1}^{n+1} {X_i}^{m_i})^{-s})^r, g^{z'r}{\sigma_W}^{-t}, m_{n+1})$$

Then we complete the proof in two steps. First, we argue by DDH that the adversary's success probability in games H_{i-1} and H_i' differ at most negligibly, and then we argue again by DDH that his success probability in games H_i' and H_i is also at most negligibly different.

H_{i-1} and H_i' First, observe that in game H_i' (and game H_{i-1}) the output of the game will be exactly the same if we make the following modification: On the ith MAC query, instead of choosing $m_{n+1}^{(i)}$ at random, we choose $m_{n+1}^{(i)}$ so that if we set $\hat{m}^{(i)} = (m_1^{(i)}, \dots, m_{n+1}^{(i)})$ (where $m^{(i)} = (m_1^{(i)}, \dots, m_n^{(i)})$ is the message the adversary queried on), we get $H_v(\hat{m}^{(i)}) = 0$. To see that this will produce the same output note that *IParams* and the values $X_0, Y_0, \{y_i'\}_0^{n+1}, z', s, t$ needed to run MAC$'$, SimMAC, SimMAC$'$ and SimVerify are all independent of $\{v_i\}_0^{n+1}$ (because $\{x_i'\}_0^{n+1}$ are uniformly random), so this $m_{n+1}^{(i)}$ will be uniformly distributed, independent of these values in the adversary's view. Furthermore, his chance of success on output forgery with $\hat{m}^* = (m_1^*, \dots, m_n^*, m_{n+1}^*)$ (i.e. where the forgery message is $m^* = (m_1^*, \dots m_n^*)$ and m_{n+1}^* is the last element of the forged tag) depends only on $H_{y'}(\hat{m}^*), H_v(\hat{m}^*), s, H_{x'}(\hat{m}^*)$ and these values will be independent of $m_{n+1}^{(i)}$. (We have observed that $\{y'\}_0^{n+1}, s$ are independent of $m_{n+1}^{(i)}$. Then we note that $H_v(\hat{m}^*)$ will also be independent of $m_{n+1}^{(i)}$ as long as the adversary produces a forgery on a new message $m^* \neq m^{(i)}$, because $H_v(\cdot)$ is a pairwise independent function, and that $H_{x'}(\hat{m}^*)$ is completely determined by

$H_v(\hat{m}^*), w, \mathit{IParams}$.) Thus, the distribution of all the values in the game (including the adversary's probability of success) will be identical to what it would have been in the original H'_i (or H_{i-1}).

Suppose that there exists an adversary for which the success probability in games H'_i and H_{i-1} is non-negligibly different. Then we will construct an adversary $\mathcal{B}$ for DDH as follows: $\mathcal{B}$ receives as input a DDH challenge $(A, B, C) = (g^a, g^b, g^c)$ and must determine whether $c = ab$. It sets $h = A$, and then generates all of the other elements according to KeyGen'. Note that MAC', SimMAC, $\mathsf{SimVerify}$, and $\mathsf{FinalVerify}$ all work without knowing the discrete log w of h, so $\mathcal{B}$ can answer all the verification queries, generate the first $i-1$ (simulated) tags, and the last $Q_v - i$ (real) tags, and evaluate the adversary's forgery exactly as in games H'_i and H_{i-1}.

For the ith tag if $\mathcal{A}$ queries for message $m = (m_1, \ldots, m_n)$, $\mathcal{B}$ will choose m_{n+1} such that if we let $\hat{m} = (m_1, \ldots, m_{n+1})$, we will get $H_v(\hat{m}) = 0$. (As noted above, this does not affect the adversary's view or chances of success.) It then sets $\sigma_X = B^{H_{x'}(\hat{m})}$, $\sigma_Y = B^{H_{y'}(\hat{m}) - sH_{x'}(\hat{m})}$, $\sigma_W = C$, and $\sigma_Z = B^{z'}C^{-t}$. Implicitly, this is equivalent to setting $r = b$. Now, if $C = g^{ab}$, this will be identical to the output of MAC' and otherwise it will be identical to the output of SimMAC' (in both cases conditioned on the appropriate choice for m_{n+1}).

Thus, $\mathcal{B}$ will continue the game until $\mathcal{A}$ outputs a forgery. It will evaluate the forgery by running $\mathsf{SimVerify}$ and $\mathsf{FinalVerify}$, and if both accept it will output 1, and otherwise it will output 0.

Since $\mathcal{B}$ exactly simulates either H_{i-1} or H'_i depending on the DDH challenge, $\mathcal{B}$'s advantage will be exactly the difference in $\mathcal{A}$'s success probabilities between the two games. We conclude that $\mathcal{A}$'s success probabilities differ by at most a negligible amount.

H'_i and H_i Suppose that there exists an adversary for which the success probability in games H'_i and H_i is non-negligibly different. Then we will construct an adversary $\mathcal{B}$ for DDH as follows: $\mathcal{B}$ receives as input a DDH challenge $(A, B, C) = (g^a, g^b, g^c)$ and must determine whether $c = ab$. It sets $h = A$, and then generates all of the other elements according to KeyGen'. Note that MAC', SimMAC, $\mathsf{SimVerify}$, and $\mathsf{FinalVerify}$ all work without knowing the discrete log w of h, so $\mathcal{B}$ can answer all the verification queries, generate the first $i-1$ (simulated) tags, and the last $Q_v - i$ (real) tags, and evaluate the adversary's forgery exactly as in games H'_i and H_{i-1}.

For the ith tag if $\mathcal{A}$ queries for message $m = (m_1, \ldots, m_n)$, $\mathcal{B}$ will choose random $m_{n+1} \leftarrow Z_p$ and random $\sigma_W \leftarrow G$, let $\hat{m} = (m_1, \ldots, m_{n+1})$, and set $\sigma_X = B^{H_{x'}(\hat{m})}C^{H_v(\hat{m})}$, $\sigma_Y = B^{H_{y'}(\hat{m})}\sigma_X{}^{-s}$, and $\sigma_Z = B^{z'}\sigma_W{}^{-t}$. Implicitly, this is equivalent to setting $r = b$. Now, if $C = g^{ab}$, this will be identical to the output of SimMAC' and if it is a random element of G it will be statistically close to the output of SimMAC. (If $H_v(\hat{m}) \neq 0$, then this will be identical to the output of SimMAC, and this case occurs with all but negligible probability because here m_{n+1} is chosen at random.)

Thus, $\mathcal{B}$ will continue the game until $\mathcal{A}$ outputs a forgery. It will evaluate the forgery by running SimVerify and FinalVerify, and if both accept it will output 1, and otherwise it will output 0.

Since $\mathcal{B}$ exactly simulates either H_i' or H_i depending on the DDH challenge, $\mathcal{B}$'s advantage will be exactly the difference in $\mathcal{A}$'s success probabilities between the two games. We conclude that $\mathcal{A}$'s success probabilities differ by at most a negligible amount.

□

Proof of Claim 7. Let $m = (m_1, \dots, m_n), \sigma = (\sigma_X, \sigma_Y, \sigma_W, \sigma_Z, m_{n+1})$ be the adversary's forgery, and let $\hat{m} = (m_1, \dots, m_{n+1})$.

Now, if m, σ is accepted by FinalVerify, then the following equation holds:

$$({\sigma_W}^{-H_v(\hat{m})}\sigma_X)^{H_{y'}(\hat{m})} = ({\sigma_X}^s\sigma_Y)^{H_{x'}(\hat{m})}$$

Note that x_0', v_0' are independent and information theoretically hidden from $\mathcal{A}$, even given all the secrets other than X_0, Y_0, x_0, y_0 (which are not used in Game G_3). This in turn means that $H_{x'}(\hat{m}), H_v(\hat{m})$ are information theoretically hidden.

Thus, the probability that $\mathcal{A}$ can produce σ with ${\sigma_X}^s\sigma_Y \neq 1$ that satisfies FinalVerify is $1/2^k$. Furthermore, suppose that ${\sigma_X}^s\sigma_Y = 1$. Then if FinalVerify accepts, it must be the case that ${\sigma_W}^{-H_v(\hat{m})}\sigma_X = 1$ as well. Now, consider the probability that $\mathcal{A}$ can produce σ such that this equation holds. Recall that $H_v(\hat{m})$ is also information theoretically hidden from $\mathcal{A}$. So the probability that $\mathcal{A}$ produces $\sigma_W \neq 1$ and FinalVerify accepts is also $1/2^k$. The only remaining case will be $\sigma_W = 1$, which is rejected by SimVerify. We conclude that the probability $\mathcal{A}$ succeeds in game G_3 is at most $1/2^k$. □

B Formal Security Definitions for Protocols

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

To simplify the definition somewhat, we will 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 will 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 will be 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 will 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*.

Security properties of a keyed-verification credential system We require the following five properties hold.

Definition 8 (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* ($\mathsf{CredKeygen}, \mathsf{Issue}, \mathsf{CredVerify}, \mathsf{Show}, \mathsf{ShowVerify}$) *is correct for $\Phi, \mathcal{U}$ if for all for all $(m_1, \ldots, m_n) \in \mathcal{U}$, for all sufficiently large k,*

$$\begin{aligned}\Pr\big[&params \leftarrow \mathsf{Setup}(1^k);\\ &(sk, IParams) \leftarrow \mathsf{CredKeygen}(params);\\ &cred \leftarrow \mathsf{Issue}(sk, (m_1, \ldots, m_n));\\ &\textit{such that } \mathsf{CredVerify}(sk, (m_1, \ldots, m_n), cred) = 0\big] = 0\end{aligned}$$

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,

$$\begin{aligned}\Pr\big[&params \leftarrow \mathsf{Setup}(1^k);\\ &(sk, IParams) \leftarrow \mathsf{CredKeygen}(params);\\ &cred \leftarrow \mathsf{Issue}(sk, (m_1, \ldots, m_n));\\ &\mathsf{Show}(IParams, cred, (m_1, \ldots, m_n), \phi) \leftrightarrow \mathsf{ShowVerify}(sk, \phi) \rightarrow b\\ &\textit{such that } b = 0\big] = 0\end{aligned}$$

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 9 (Unforgeability)**.** *A presentation protocol* $\mathsf{Show}, \mathsf{ShowVerify}$ *for keyed-verification credentials scheme* $\mathsf{CredKeygen}, \mathsf{Issue}$ *is* unforgeable *if for all PPT adversaries $\mathcal{A}$, there exists a negligible function ν such that for all k,*

$$\begin{aligned}\Pr\big[&params \leftarrow \mathsf{Setup}(1^k);\\ &(IParams, sk) \leftarrow \mathsf{CredKeygen}(params);\\ &(state, \phi) \leftarrow \mathcal{A}(params, IParams)^{\mathsf{Issue}(sk,\cdot),\mathsf{ShowVerify}(sk,\cdot)}\\ &\mathcal{A}(state) \leftrightarrow \mathsf{ShowVerify}(sk, \phi) \rightarrow b\\ &\textit{such that } b = 1 \wedge (\forall (m_1, \ldots, m_n) \in Q, \phi(m_1, \ldots, m_n) = 0)\big] = \nu(k)\end{aligned}$$

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* $\mathsf{ShowVerify}$ *are required to be sequential.*

Definition 10 (Anonymity)**.** *A presentation protocol* $\mathsf{Show}, \mathsf{ShowVerify}$ *for keyed-verification credentials scheme* $\mathsf{CredKeygen}, \mathsf{Issue}$ *is* anonymous *if for all PPT adversaries $\mathcal{A}$, there exists an efficient algorithm* $\mathsf{SimShow}$*, 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 $params \leftarrow \mathsf{Setup}(1^k)$ and all $(IParams, sk) \leftarrow \mathsf{KeyGen}(params)$, for all $cred$ such that $\mathsf{CredVerify}(sk, (m_1, \ldots, m_n), cred) = 1$:*

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

i.e., the adversary's view given the proof can be simulated by $\mathsf{SimShow}$ *given only ϕ and a valid secret key corresponding to $IParams$.*

Note that the statement ϕ is known to $\mathcal{A}$ and may contain information about the attribute values, which may identify the user. Definition 10 ensures that the keyed-verification credential scheme's protocols are anonymous, modulo information revealed in ϕ.

Definition 11 (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 $\leftarrow \mathsf{Issue}(sk, (m_1, \ldots, m_n); r)$ if neither of these errors occurs.*[3]

We will 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 [17, Chapter 7] for a definition of secure 2PC.*

Definition 12 (Key-Parameter Consistency)**.** *The key generation algorithm* $\mathsf{CredKeygen}$ *satisfies Key-Parameter Consistency if for any PPT adversary $\mathcal{A}$, the probability that $\mathcal{A}$ given params $\leftarrow \mathsf{Setup}(1^k)$ can produce $(\mathit{IParams}, sk_1, sk_2)$ such that $(\mathit{IParams}, sk_1)$ and $(\mathit{IParams}, sk_2)$ are both in the range of* $\mathsf{CredKeygen}(\mathit{params})$ *is negligible (where the probability is over the choice of params and the random coins of $\mathcal{A}$).*

Definition 13 (Secure keyed-verification credential system)**.** *We say that* ($\mathsf{CredKeygen}$, $\mathsf{CredVerify}$, Issue, $\mathsf{BlindIssue}$, $\mathsf{BlindObtain}$, 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 Keyed-Verification Credential Scheme from $\mathsf{MAC}_{\mathsf{DDH}}$

C.1 Basic algorithms: Setup, CredKeygen, Issue, CredVerify

Let $\mathsf{Setup}_{\mathsf{DDH}}, \mathsf{KeyGen}_{\mathsf{DDH}}, \mathsf{MAC}_{\mathsf{DDH}}, \mathsf{Verify}_{\mathsf{DDH}}$ be the algorithms described in Section 3.2. Then we define the following setup algorithms for the credential system.

$\mathsf{Setup}(1^k)$ will be the same as $\mathsf{Setup}_{\mathsf{DDH}}$.

$\mathsf{CredKeygen}(\mathit{params})$**:** Parse $\mathit{params} = (G, p, g, h)$. Run $\mathsf{KeyGen}_{\mathsf{DDH}}(\mathit{params})$ to obtain $\mathit{IParams}_{\mathsf{DDH}} = \{X_i, Y_i\}_1^{n+1})$ and $sk_{\mathsf{DDH}} = (\{x_i, y_i\}_0^{n+1}, z, \mathit{IParams}_{\mathsf{DDH}})$. Choose $\tilde{x}, \tilde{y}, \tilde{z} \leftarrow Z_p$ and form commitments $C_{x_0} = g^{x_0} h^{\tilde{x}}, C_{y_0} = g^{y_0} h^{\tilde{y}}, C_z = g^z h^{\tilde{z}}$. Output $\mathit{IParams} = (\mathit{IParams}_{\mathsf{DDH}}, C_{x_0}, C_{y_0}, C_z)$ and $sk = (sk_{\mathsf{DDH}}, \tilde{x}, \tilde{y}, \tilde{z})$.

We also 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 $\mathit{cred} \leftarrow \mathsf{MAC}_{\mathsf{DDH}}(sk, (m_1, \ldots, m_n),)$.

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

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

C.2 Presentation Proof

Here we present a construction for Show, ShowVerify using a zero knowledge proof of knowledge. See Section 2 for a discussion of some possible instantiations for this proof system.

Proof Generation The inputs are the system and issuer parameters (G, p, g, h) and $(C_{x_0}, C_{y_0}, C_z, \{X_i\}_1^{n+1}, \{Y_i\}_1^{n+1})$, the credential $(\sigma_W, \sigma_X, \sigma_Y, \sigma_Z, m_{n+1})$, the statement ϕ, and the attribute values $\{m_i\}$. The prover will first randomize the credential: choose $r \in_R \mathbb{Z}_p*$ and compute $\sigma_W := {\sigma_W}^r$, $\sigma_X := {\sigma_X}^r$, $\sigma_Y := {\sigma_Y}^r$, and $\sigma_Z := {\sigma_Z}^r$.

Then the Prover chooses $r_x, r_y, w_1, \ldots w_{n+1} \in_R \mathbb{Z}_p$, and computes

$$C_{\sigma_X} := \sigma_X g^{r_x} \text{ and } C_{\sigma_Y} := \sigma_Y g^{r_y},$$
$$V_x := g^{-r_x} \prod_{i=1}^{n+1} X_i^{w_i} \text{ and } V_y := g^{-r_y} \prod_{i=1}^{n+1} Y_i^{w_i},$$
$$\text{and } C_{m_i} := {\sigma_W}^{m_i} h^{w_i} \text{ for all } i \in 1, \ldots, n+1 \; .$$

He sends $\sigma_W, \sigma_Z, C_{\sigma_X}, C_{\sigma_Y}, V_x, V_y, C_{m_1}, \ldots, C_{m_{n+1}}$ to the verifier[4] and gives a proof of knowledge:

$$\begin{aligned} \text{PK}\{(\{m_i\}_1^{n+1}, \{w_i\}_1^{n+1}, -r_x, -r_y,) : C_{m_i} &= {\sigma_W}^{m_i} h^{w_i} \text{ for all } i \in 1, \ldots n+1 \\ &\wedge\ V_x = g^{-r_x} \prod_{i=1}^{n+1} X_i^{w_i} \ \wedge\ V_y = g^{-r_y} \prod_{i=1}^{n+1} Y_i^{w_i} \\ &\wedge\ \phi(m_1, \ldots, m_n) = 1\} \; . \end{aligned}$$

Proof Verification The inputs are the system and issuer parameters (G, q, g, h) and $(C_{x_0}, C_{y_0}, C_z, \{X_i\}_1^{n+1}, \{Y_i\}_1^{n+1})$, the secret values $\{x_i\}_0^{n+1}, \{y_i\}_0^{n+1}, z$ and the presentation proof γ.

The verifier receives $(\sigma_W, \sigma_Z, C_{m_1}, \ldots, C_{m_{n+1}}, C_{\sigma_X}, C_{\sigma_Y})$ from the prover and verifies that

$$V_x = \frac{{\sigma_W}^{x_0} \prod_1^{n+1} {C_{m_i}}^{x_i}}{C_{\sigma_X}} \; ,$$
$$V_y = \frac{{\sigma_W}^{y_0} \prod_1^{n+1} {C_{m_i}}^{y_i}}{C_{\sigma_Y}} \; ,$$

[4]If we have a proof of knowledge system where the extractor can extract a witness without being given the statement, then V_x, V_y can be omitted from this message; in that case the honest verifier will compute $V_x = \frac{{\sigma_W}^{x_0} \prod_1^{n+1} {C_{m_i}}^{x_i}}{C_{\sigma_X}}$ and $V_y = \frac{{\sigma_W}^{y_0} \prod_1^{n+1} {C_{m_i}}^{y_i}}{C_{\sigma_Y}}$ and use that to verify the proof.

then verifies π. If π is valid and if $\sigma_Z = {\sigma_W}^z$, accept, otherwise reject.

C.3 Issuance

Here we consider the protocols $\mathsf{BlindIssue}, \mathsf{BlindObtain}$ for issuing credentials. We first consider a simplified issuing protocol that can be used if all of the attributes are known to the issuer, then present the more general protocol allowing for hidden attributes. Again, see Section 2 for a discussion of some possible instantiations for the proof of knowledge system.

Issuance with public attributes To issue a credential with the attributes $(m_1, \ldots, m_n) \in \mathbb{Z}_q^n$, the issuer chooses $\sigma = (\sigma_W, \sigma_X, \sigma_Y, \sigma_Z, m_{n+1}) \leftarrow \mathsf{MAC}_{\mathsf{DDH}}(sk, (m_1, \ldots, m_n))$, and returns σ to the user and gives a proof π, where

$$\begin{aligned}\pi := \mathrm{PK}\{(\{x_i\}_0^{n+1}, \{y_i\}_0^{n+1}, z, \tilde{x}, \tilde{y}, \tilde{z}) :&\sigma_X = {\sigma_W}^{x_0} \prod_1^{n+1} ({\sigma_W}^{m_i})^{x_i} \ \wedge\ \sigma_Y = {\sigma_W}^{y_0} \prod_1^{n+1} ({\sigma_W}^{m_i})^{y_i} \\ &\wedge\ \sigma_Z = {\sigma_W}^z \ \wedge\ C_{x_0} = g^{x_0} h^{\tilde{x}} \ \wedge\ C_{y_0} = g^{y_0} h^{\tilde{y}} \\ &\wedge\ C_z = g^z h^{\tilde{z}} \ \wedge\ \forall i \in 1, \ldots, n+1, X_i = h^{x_i} \\ &\wedge\ \forall i \in 1, \ldots, n+1, Y_i = h^{y_i}\}\end{aligned}$$

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 $\perp$.

Issuance with hidden attributes If some of the attributes must be hidden, we can proceed as follows: Let $\mathcal{H} \subseteq \{1, \ldots, n\}$, be the indices of the attributes that must remain hidden. 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 follows: $E_i = (g^{r_i}, g^{m_i}\gamma^{r_i})$, using $r_i \in_R Z_p$. The ciphertexts are sent to the issuer (along with a proof of knowledge of $\{r_i, m_i\}_i \in \mathcal{H}$). The issuer chooses $b \in_R Z_p^*$, and $m_{n+1} \in_R Z_p$.[5] It computes $\sigma_W = g^b$, $\sigma_Z = \sigma_W^z$, and uses the homomorphic properties of Elgamal to generate an encryption E_x of $\sigma_X = g^{bx_0 + bm_{n+1}x_{n+1}} \prod_1^n (g^{m_i})^{bx_i}$ and encryption E_y of $\sigma_Y = g^{by_0 + bm_{n+1}y_{n+1}} \prod_1^n (g^{m_i})^{by_i}$, and to randomize both of these encryptions to obtain E'_x, E'_y (by multiplying with encryptions of 0 using randomness $r_x \leftarrow Z_p$ and $r_y \leftarrow Z_p$ respectively). It sends $\sigma_W, \sigma_Z, E'_x, E'_y, m_{n+1}$ to the user and gives a proof that these values have been generated correctly with respect to $(C_{x_0}, C_{y_0}, C_z, \{X_i, Y_i\}_1^{n+1})$ (i.e.

[5]As discussed in section 3.2, in practice we can choose $m_{n+1} = H(\sigma_W)$ if we assume H is a random oracle.

a proof of knowledge the appropriate $\{x_i, y_i\}_0^{n+1}, z, \tilde{x}, \tilde{y}, \tilde{z}, b$, and randomizing factors r_x, r_y). If the proof does not verify, the user outputs $\perp$. Otherwise, the user decrypts E'_x, E'_y to get σ_X, σ_Y, and outputs $\sigma = (\sigma_W, \sigma_X, \sigma_Y, \sigma_Z, m_{n+1})$.

C.4 Security

Theorem 14. *If the DDH assumption holds and the proof system used is a zero knowledge proof of knowledge, then the above algorithms* ($\mathsf{CredKeygen}$, Issue, $\mathsf{CredVerify}$, Show, $\mathsf{ShowVerify}$, $\mathsf{BlindIssue}$, $\mathsf{BlindObtain}$) *make up a secure keyed-verification credential system.*

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 $(u, u^{x_0+\sum_1^{n+1} x_i m_i}, u^{y_0+\sum_1^{n+1} y_i m_i}, u^z, m_{n+1})$. 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 :

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

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.

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

$\mathcal{B}$ receives $params, IParams_{\mathsf{DDH}}$ and chooses random $C_{x_0}, C_{y_0}, C_z \leftarrow Z_p$. It then sends $params, IParams = (IParams_{\mathsf{DDH}}, C_{x_0}, C_{y_0}, C_z)$ to $\mathcal{A}$.

When $\mathcal{A}$ queries the Issue oracle, $\mathcal{B}$ forwards the query to it's MAC oracle and returns the resulting tag.

When $\mathcal{A}$ queries the $\mathsf{ShowVerify}$ oracle: $\mathcal{A}$ sends $\sigma_W, \sigma_Z, C_{m_1}, \ldots C_{m_{n+1}}, C_{\sigma_X}, C_{\sigma_Y}$, and gives a proof π. If the proof π is invalid, $\mathcal{B}$ will return $\perp$. Otherwise $\mathcal{B}$ will run the proof of knowledge extractor to extract $\{m_i\}_1^{n+1}, r_x, r_y$. 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 it's Verify oracle with $(m_1, \ldots m_n), (\sigma_W, \sigma_X, \sigma_Y, \sigma_Z, m_{n+1})$, and output the result.

In the final show protocol, $\mathcal{B}$ will again, extract $\{m_i\}_1^{n+1}, r_x, r_y$, and output $(m_1, \ldots m_n)$, $(\sigma_W, C_{\sigma_X} g^{-r_x}, C_{\sigma_Y} g^{-r_y}, \sigma_Z, m_{n+1})$ as it's forgery.

First, note that $\mathcal{B}$'s response to Issue queries will be identical to the honest Issue algorithm. Then, we argue that it's 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 $\{m_i\}_1^{n+1}, r_x, r_y$ then

$$
\begin{aligned}
V_x &= \frac{{\sigma_W}^{x_0} \prod_1^{n+1} {C_{m_i}}^{x_i}}{C_{\sigma_X}} \\
\iff g^{-r_x} \prod_1^{n+1} {X_i}^{w_i} &= \frac{{\sigma_W}^{x_0} \prod_1^{n+1} ({\sigma_W}^{m_i} h^{w_i})^{x_i}}{C_{\sigma_X}} \\
\iff g^{-r_x} \prod_1^{n+1} (h^{x_i})^{w_i} &= \frac{{\sigma_W}^{x_0 + \sum_1^{n+1} m_i x_i} \prod_1^{n+1} h^{w_i x_i}}{C_{\sigma_X}} \\
\iff C_{\sigma_X} g^{-r_x} &= {\sigma_W}^{x_0 + \sum_1^{n+1} m_i x_i}
\end{aligned}
$$

And similarly $V_y = \frac{C_{r_y} {\sigma_W}^{y_0} \prod_1^{n+1} {C_{m_i}}^{y_i}}{C_{\sigma_Y}}$ iff $C_{\sigma_Y} g^{-r_y} = {\sigma_W}^{y_0 + \sum_1^{n+1} y_i m_i}$. The final check that the honest verifier makes guarantees that $\sigma_Z = {\sigma_W}^z$. Thus, the honest verifier algorithm will accept iff $(\sigma_W, C_{\sigma_X} g^{-r_x}, C_{\sigma_Y} g^{-r_y}, \sigma_Z, m_{n+1})$ would be accepted by $\mathsf{Verify}_{\mathsf{DDH}}$ for message $(m_1, \ldots, m_n)$.

Similarly, we can argue that $\mathcal{B}$ will extract a valid MAC from the final show protocol whenever $\mathsf{ShowVerify}$ would have output 1. Thus, if $\mathcal{A}$ can cause $\mathsf{ShowVerify}$

to accept for some statement ϕ that is not satisfied by any of the attribute sets queried to Issue, then $\mathcal{B}$ 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 $(\mathit{IParams}, \mathit{sk})$ be in the range of $\mathsf{CredKeygen}$, and let cred be such that $\mathsf{CredVerify}(\mathit{sk}, \mathit{cred}, (m_1, \ldots, m_n)) = 1$.

Then $\mathsf{SimShow}(\mathit{sk}, \phi)$ will behave as follows: It will choose random values $\sigma_W, C_{\sigma_X}, C_{\sigma_Y}, C_{m_1}, \ldots, C_{m_n} \leftarrow G$. It will then use $\{x_i, y_i\}_0^{n+1}, z$ from sk to compute $\sigma_Z = {\sigma_W}^z$, $V_x = \frac{{\sigma_W}^{x_0} \prod_1^{n+1} {C_{m_i}}^{x_i}}{C_{\sigma_X}}$, and $V_y = \frac{{\sigma_W}^{y_0} \prod_1^{n+1} {C_{m_i}}^{y_i}}{C_{\sigma_Y}}$. It will run $\mathcal{A}$ with these values as the first message, and then simulate the proof of knowledge, and output whatever $\mathcal{A}$ 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 $\mathsf{CredVerify}(\mathit{sk}, \mathit{cred}, (m_1, \ldots, m_n)) = 1$, randomizing the credential will produce the same distribution as choosing random σ_W and computing $\sigma_X = {\sigma_W}^{x_0 + \sum_1^{n+1} x_i m_i}$, $\sigma_Y = {\sigma_W}^{y_0 + \sum_1^{n+1} y_i m_i}$, and $\sigma_Z = {\sigma_W}^z$ for the values $z, \{x_i, y_i\}_0^{n+1}$ 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 $C_{\sigma_X} = {\sigma_W}^{x_0 + \sum_1^{n+1} x_i m_i} g^{r_x}$, $C_{\sigma_Y} = {\sigma_W}^{y_0 + \sum_1^{n+1} y_i m_i} g^{r_y}$, 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 $\mathsf{SimShow}$, then the calculation above in the proof of Correctness shows that the V_x, V_y that $\mathsf{SimShow}$ 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 $(S, \mathit{IParams})$ 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 $\mathit{cred} = (\sigma_W, \sigma_X, \sigma_Y, \sigma_Z, m_{n+1})$ 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 $sk = (\{x_i\}_0^{n+1}, \{y_i\}_0^{n+1}, \tilde{x}, \tilde{y}, \tilde{z})$ and $r = (\sigma_W, m_{n+1})$. It will send (sk, r) to the ideal functionality. By the proof of knowledge property, the credential sent in the real world is $\sigma_W, {\sigma_W}^{x_0} \prod_1^{n+1} ({\sigma_W}^{m_i})^{x_i}, \sigma_Y = {\sigma_W}^{y_0} \prod_1^{n+1} ({\sigma_W}^{m_i})^{y_i}, \sigma_Z = {\sigma_W}^z, m_{n+1})$ 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 $(S, \mathit{IParams})$ 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 $\mathit{cred} = (\sigma_W, \sigma_X, \sigma_Y, \sigma_Z, m_{n+1})$. The 2PC simulator will then compute an encryption E'_x of σ_X and an encryption E'_y of σ_Y, send $(\sigma_W, \sigma_Z, E'_x, E'_y, m_{n+1})$ 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 ${\sigma_W}^{x_0} \prod_1^{n+1} ({\sigma_W}^{m_i})^{x_i}, {\sigma_W}^{y_0} \prod_1^{n+1} ({\sigma_W}^{m_i})^{y_i}$. 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 $\mathcal{A}$, and simulate the proof. It will then receive $\mathit{cred} = (\sigma_W, \sigma_X, \sigma_Y, \sigma_Z, m_{n+1})$ from $\mathcal{A}$ and run the verification of the proof of knowledge; if the proof is accepting, it will run the proof of knowledge extractor to extract $sk = (\{x_i\}_0^{n+1}, \{y_i\}_0^{n+1}, \tilde{x}, \tilde{y}, \tilde{z})$ and $r = (b, m_{n+1})$. 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 $(\sigma_W = g^b, {\sigma_W}^{x_0} \prod_1^{n+1} ({\sigma_W}^{m_i})^{x_i}, {\sigma_W}^{y_0} \prod_1^{n+1} ({\sigma_W}^{m_i})^{y_i}, {\sigma_W}^z, m_{n+1})$ 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 [25]).

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.)

□

C.5 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 [12], 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 [14] 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 [3]. In the random oracle model however, we don't know of any such implication. (See [15] pp.152–153 for 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 [12], which converts any sigma protocol into a three round interactive zero knowledge proof of knowledge secure under concurrent composition. This protocol requires trusted parameters for a trapdoor commitment scheme, which can be efficiently implemented (without a trusted party) in the random oracle model: Consider the commitment scheme which commits by choosing r and computing 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. [6]

The second option is to make the assumption that the stronger extraction property holds for Fiat-Shamir based proofs [14] in the random oracle model. While it

[6] For alternative trapdoor commitment schemes that do not require a random oracle, see [12, §4]. These alternatives require trusted setup of a common reference string,

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 [23, 26]. Since our analysis for the $\mathsf{MAC}_{\mathsf{GGM}}$ scheme already uses this model, this maybe a good choice for use with that scheme.

D 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.

D.1 Proof Generation

Inputs: *params*, a credential u_0, u'_0, and attribute values $m_1, \ldots, m_n$.

1. (*Re-randomize*) Choose $a \in_R \mathbb{Z}_p$, compute $u = {u_0}^a$ and $u' = {u'_0}^a$. Delete a.
2. (*Form commitments*)
   (a) Choose $r, z_1, \ldots z_n \in_R \mathbb{Z}_p$.
   (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 $\tilde{z_1}, \ldots, \tilde{z_n}, \tilde{r}, \tilde{m_1}, \ldots, \tilde{m_n} \in_R \mathbb{Z}_p$.
   (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), $\{s_{m_i} = \tilde{m_i} - cm_i, s_{z_i} = \tilde{z_i} - cz_i\}_{i=1}^n$, and $s_r = \tilde{r} + rc$. Let S denote the set of responses.
   (e) Output $\pi = (c, S)$.
4. (*Output*) Output the presentation proof $P = (u, \{C_{m_i}\}_{i=1}^n, C_{u'}, \pi)$.

D.2 Proof Verification

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

1. Parse P as $(u, C_{m_1}, \dots, C_{m_n}, C_{u'}, \pi)$.
2. Compute
$$V = \frac{{C_{m_1}}^{x_1} \cdots {C_{m_n}}^{x_n} u^y}{C_{u'}} ,$$
3. (*Verify* π)
    (a) Parse π as (c, S) where S contains the responses computed in Step 3d of proof generation.
    (b) Compute
    $$c' = H(\mathit{param} \| \{C_{m_i}\}_{i=1}^n \| C_{u'} \| \{C_{m_i} g^{s_{m_i}} h^{s_{z_i}}\}_{i=1}^n \| V X^{s_{z_1}} \cdots X^{s_{z_n}} g^{s_r})$$
    (c) Accept π as valid if $c' = c$, otherwise reject.
4. (*Output*) If π is valid, output $\{C_{m_i}\}_{i=1}^n$