The Signal Private Group System and Anonymous Credentials Supporting Efficient Verifiable Encryption

Melissa Chase
Microsoft Research
melissac@microsoft.com

Trevor Perrin
Signal Technology Foundation
trevp@signal.org

Greg Zaverucha
Microsoft Research
gregz@microsoft.com

Draft – December 6, 2019

Abstract

In this paper we present a system for maintaining a membership list of users in a group, designed for use in the Signal Messenger secure messaging app. The goal is to support *private groups* where membership information is readily available to all group members but hidden from the service provider or anyone outside the group. In the proposed solution, a central server stores the group membership in the form of encrypted entries. Members of the group authenticate to the server in a way that reveals only that they correspond to some encrypted entry, then read and write the encrypted entries.

Authentication in our design uses a primitive called a keyed-verification anonymous credential (KVAC), and we construct a new KVAC scheme based on an algebraic MAC, instantiated in a group $\mathbb{G}$ of prime order. The benefit of the new KVAC is that attributes may be elements in $\mathbb{G}$, whereas previous schemes could only support attributes that were integers modulo the order of $\mathbb{G}$. This enables us to encrypt group data using an efficient Elgamal-like encryption scheme, and to prove in zero-knowledge that the encrypted data is certified by a credential. Because encryption, authentication, and the associated proofs of knowledge are all instantiated in $\mathbb{G}$ the system is efficient, even for large groups.

1 Introduction

Secure messaging applications enable a user to send encrypted messages to one or more recipients. A notion of *groups* is often supported: messages sent to a group will be delivered to all users who are current members of the group. Typically a group is created by a user to contain an initial set of members. These members (and the group creator) are given privileges to add and remove other members and grant them privileges, and so on. The result is that group membership is managed by the members.

Maintaining this membership list introduces challenges. First, the membership list must be stored and made available to users. Second, changes to the membership list must

be subject to authentication and access control, to ensure that changes are only made by authenticated and authorized users.

The standard approach is to store the membership list, in plaintext, in a database on a server. This solves the problems of storage and availability, and the server enforces authentication and access control. The downside to this approach is that the server has a stored repository of associations between its users, and can easily insert malicious users into groups to receive messages. These are serious threats for an encrypted messaging system.

The Signal messaging app [Sig19] previously introduced a *private group* approach where the membership list is hidden from the server. In Signal's system the group membership list is maintained in a distributed fashion by each user [Mar14]. Users send encrypted group changes to each other when they want to modify group membership. Users then apply these changes to their local copy of the group state. This avoids problems with server trust, but introduces new problems around group consistency in the face of race conditions, lost messages, or malicious users. To reduce some consistency problems, clients can trust each other to provide copies of the group state, but this limits their ability to enforce access control and introduces other risks [RMS17].

To address the above problems with distributed private groups while maintaining their security, we introduce a new approach to private groups. In our new approach, group membership is stored on a server but in encrypted form.

This use of server storage addresses the availability and consistency challenges: there is always a single membership list for each group, stored on the server, which clients can query. The use of encryption reduces the trust placed in the server: in our design the entries that store each member's identity (called a *UID*) are encrypted with a group-specific secret key which all members share with each other. A malicious server could still learn something about the group by observing access patterns as entries are used for authentication, added, and deleted. A malicious server could also make limited changes by restoring old entries or returning incomplete information. However, the server is prevented from decrypting entries or forging new entries.

Another benefit of this approach is that all changes to the membership list go through the server. Thus, even though entries are encrypted, the server can reliably enforce access control rules regarding which entries are allowed to make which changes.

Using encryption in this manner introduces new requirements, which are the focus of the present work:

- *Anonymous authentication*: When a group member wishes to add or remove another user from the group, the existing member must first authenticate to the server so that the server can perform access control and determine whether the member is allowed to perform this operation. This is true for the standard "plaintext on server" approach and remains true in our proposal, except that the group entry being used for authentication and access control contains an encrypted UID rather than a plaintext

UID. The group member will need to anonymously authenticate using their encrypted entry by proving ownership of the encrypted UID, without the server learning the UID.

- *Deterministic encryption*: It is important that each plaintext UID in a group corresponds to a single encrypted UID in that group, and that an entry must not decrypt successfully unless it is the unique deterministic encryption of the underlying UID. If this requirement is not met, a single UID could be added to the group many times using different ciphertexts, and these ciphertexts would be treated as different members by the server. This would complicate access control and operations such as deletion.

- *Decryption and authentication consistency*: Because encrypted entries are used in two ways (decrypted by users to learn the group membership, and used for authentication), it is important for entries to decrypt successfully if and only if they can be used for authentication, even if entries are created by a malicious group member. In other words, successful decryption must imply that an entry can be used for authentication, and successful authentication must imply that an entry can be decrypted. If this requirement is not met, a malicious group member could cause an honest group member's view of group membership to diverge from the group membership that the server is using for access control. (Note that these requirements do not prevent a user from uploading an invalid ciphertext, provided that it cannot be used for authentication; nor do they prevent a user from encrypting an invalid UID.)

We satisfy the anonymous authentication requirement using server-issued *anonymous credentials*. In particular, we introduce a new form of *keyed-verification anonymous credentials*, extending the construction from [CMZ14] to support efficient zero-knowledge proofs compatible with *verifiable encryption*. These proofs assert that a verifiably-encrypted ciphertext decrypts to a plaintext P, and that the prover has a credential certifying P. The plaintext P can be any value that can be encoded as a group element[1]. Given this credential scheme, the server will issue users time-limited *auth credentials* for their UID, encoded as a group element. Because encryption of UIDs is deterministic, users can calculate their encrypted group entry without needing to retrieve it (so that the encrypted group membership list is not exposed to non-members). Users will then provide the server a zero-knowledge proof that they have a valid auth credential matching their encrypted entry. Because of the zero-knowledge property, the server has assurance that the user possesses such an auth credential without learning the UID certified by the credential, or other information that might link this use of the credential to other uses or to credential issuance.

We satisfy the deterministic encryption and decryption/authentication consistency requirements by (a) having the user prove to the server that their entry is a correct deterministic encryption of some UID at the same time as the user authenticates using that

[1]The term *group* here and elsewhere is used in its mathematical sense; we trust context will disambiguate.

entry, and (b) having decrypters check that entries are correct deterministic encryptions of the UID at the time of decryption.

Profile keys With the above building blocks we have a rudimentary management system for private groups, capable of storing and managing membership lists comprised of encrypted UIDs. We then build a more sophisticated system that additionally stores an encrypted *profile key* for each user. Profile keys are a notion from the Signal Protocol, where they are used to encrypt user-associated *profile data* such as avatar images and profile names that provide a more user-friendly view of a user's identity [Lun17]. Encrypted profile data is stored on the server and fetched and decrypted by users, but is not decryptable by the server. Users will share their profile key (and thus their profile data) with other users whom they trust.

To improve the Signal group experience we will also store encrypted profile keys in the group membership state so that users in any group will see a user-friendly, "profile-enhanced" view of the membership list, rather than simply a list of UIDs. If a user Alice would like to add Bob to a group and she knows Bob's profile key then she can encrypt the profile key and add the ciphertext to a group herself. If Alice would like to add Bob to a group without knowing his profile key she will *invite* Bob into the group. The server will record this invitation but Bob won't be considered a full member (and thus won't receive group messages) until he accepts Alice's invitation by adding his encrypted profile key.

Storing encrypted profile keys introduces a new requirement for *UID and profile key consistency*: it is important that the server only stores a pair (UID ciphertext, profile key ciphertext) if this pair correctly decrypt to a UID and its associated profile key, even if these ciphertexts are created by a malicious group member. If this requirement is not met, a malicious group member could cause an honest group member's view of other group members to be inaccurate or incomplete with respect to profile data.

We satisfy this requirement with an additional server-issued anonymous credential. Unlike the *auth credentials* discussed previously which are issued to the owners of UIDs, these *profile key credentials* are issued to any user who knows another user's profile key. Users will register a *profile key commitment* with the server. Then other users can prove to the server that they know another user's profile key without revealing it. We combine this proof with a *blinded credential issuance* since the server must issue a credential on a UID and profile key without knowing the profile key.

After a user (Alice) acquires a profile key credential for another user (Bob), she can add Bob to groups by providing UID and profile key ciphertexts for Bob along with a zero-knowledge proof that these ciphertexts are correctly associated (i.e., that they encrypt values which are certified by a profile key credential). The zero-knowledge property of the credential system means that the server does not learn which profile key credential was used, nor learn anything about the encrypted UID and encrypted profile key beyond the fact that they are associated.

1.1 System Overview

At this point we can summarize the main objects and interactions in the Signal Private Group System from the perspective of two users, Alice and Bob. More details on these objects and operations are presented in Section 5.

- Bob generates a *ProfileKey* and registers his *ProfileKeyCommitment* with the server. Bob uses his *ProfileKey* to encrypt profile data.[2]
- Bob trusts Alice to view his profile data and so shares his *ProfileKey* with Alice by sending her an encrypted message. [3]
- Alice and Bob are configured with *ServerPublicParams* which corresponds to the *ServerSecretParams* used by the server for credential issuance. Alice and Bob use the *ServerPublicParams* for verifying issued credentials and presenting credentials.
- Alice contacts the server, without identifying herself, and uses Bob's *ProfileKey* to fetch and decrypt Bob's profile data, and also to fetch a *ProfileKeyCredential* for Bob's UID and *ProfileKey*.
- Alice and Bob contact the server periodically to fetch *AuthCredentials* for their UID.
- Alice creates a new group containing her and Bob by generating a pair (*GroupSecretParams*, *GroupPublicParams*), contacting the server without identifying herself, and registering the *GroupPublicParams* with the server. Alice also uploads pairs of (*UidCiphertext*, *ProfileKeyCiphertext*) for herself and Bob. Alice proves these ciphertexts are correct by proving that she has an *AuthCredential* for her *UidCiphertext*, and by proving she has a *ProfileKeyCredential* for each pair of ciphertexts.
- Alice sends Bob the *GroupSecretParams* via an encrypted message. Bob can now authenticate to the group using his *AuthCredential* to download and decrypt the other group entries and learn the membership. If Bob's entry is authorized to add or delete members of the group, Bob can also authenticate to the server and request it to perform these operations.

Cryptography For efficiency and simplicity, our solution is designed to work using cryptography instantiated in a group $\mathbb{G}$ of prime order q. We augment this group with functions to hash to group elements or integers modulo the group order, and functions to encode and decode data into group elements.

Our encryption scheme is symmetric-key, deterministic, CCA-secure, and has a property we call *unique ciphertexts*, meaning that it is intractable to find two valid encryptions

[2] The details of profile data encryption (see [Lun17]) are not relevant to how groups are managed.

[3] This is an end-to-end encrypted message which we assume the secure messaging platform provides. For details of E2E encryption in Signal, see [Sig19].

of the same plaintext, even with knowledge of the key. Since it is a variant of Elgamal encryption in $\mathbb{G}$, it has small ciphertexts with efficient encryption and decryption. Moreover, it is compatible with the efficient zero-knowledge proof system we use for credential presentation, allowing us to prove that ciphertexts are well-formed with respect to a public commitment of the key, and that the plaintext is an attribute from a credential.

For the latter part of the proof, we need a credential system that supports attributes that are group elements. Previously known anonymous credentials and KVAC schemes only support attributes from $\mathbb{Z}_q$. Following the approach to constructing a KVAC scheme from [CMZ14], we first design an algebraic MAC where the message space is n-tuples of elements in $\mathbb{G}$. Note that we can still support attributes $m \in \mathbb{Z}_q$, by having a fixed base $G \in \mathbb{G}$ and computing the MAC on G^m, which is helpful since some ZK proofs are easier when attributes are in $\mathbb{Z}_q$. We prove our new MAC is secure in the random oracle model, assuming i) that DDH in $\mathbb{G}$ is hard, and ii) that a simpler MAC, called $\mathsf{MAC}_{\mathsf{GGM}}$, from [CMZ14] is secure. Our security analysis of our encryption scheme first defines the new properties required for the private groups application, then we prove the scheme is secure under the same assumptions as the security proof for our new MAC.

We then give protocols for credential (blind) issuance and presentation, to construct a complete KVAC system satisfying the security properties defined in [CMZ14]. The resulting credential scheme and proof protocols are efficient, and can be instantiated using well-known non-interactive generalized Schnorr proofs of knowledge.

2 Preliminaries and Related Work

Notation We use capital letters to denote group elements, and lower case letters to denote integers modulo the group order.

2.1 Group Description and Hardness Assumptions

The new cryptographic primitives in this paper are designed to work in a cyclic group, denoted $\mathbb{G}$, of prime order q. We require that $\mathbb{G}$ has three associated functions.

1. A function $\mathsf{HashTo}\mathbb{G} : \{0,1\}^* \to \mathbb{G}$ that hashes strings to group elements. This should be based on a cryptographic hash function; we will model it as a random oracle.
2. A function $\mathsf{HashTo}\mathbb{Z}_q : \{0,1\}^* \to \mathbb{Z}_q$, also based on a cryptographic hash function.
3. A function $\mathsf{EncodeTo}\mathbb{G} : \{0,1\}^\ell \to \mathbb{G}$, that maps ℓ-bit strings to elements of $\mathbb{G}$ in a reversible way. The parameter ℓ depends on the size of $\mathbb{G}$ and the encoding.

For our security analysis, we will assume that the decisional Diffie-Hellman (DDH) is hard in $\mathbb{G}$, i.e., given (G^a, G^b, C) decide if $C = G^{ab}$. This implies that the discrete logarithm problem (DLP) is also hard in $\mathbb{G}$, i.e., given $Y = G^x$ it is hard to find x. We also require

that $\mathsf{MAC}_{\mathsf{GGM}}$ is uf-cma-secure, and the only known proofs are in the generic group model, so we inherit this assumption as well.

2.2 Keyed-Verification Anonymous Credentials (KVAC)

An anonymous credential system [Cha85, CV02, PZ13] is a set of cryptographic protocols: A *credential issuance* protocol provides users with credentials that "certify" some set of attributes. A *credential presentation* protocol enables the user to prove that they possess a credential whose attributes satisfy some predicate without revealing the credential or anything else about it (a zero-knowledge proof). There is a vast literature on anonymous credentials, a good starting point on the subject is [RCE15].

Traditional anonymous credentials designs are based on public key signatures: the credential Alice holds is a special type of signature on the attributes. When she presents the credential to Bob, she proves (in zero-knowledge) that her credential is a valid signature with respect to the credential issuer's public key. The benefit of signature-based credentials is that Alice may present her credential to anyone in possession of the issuer's public key, but the drawback is that known constructions are either relatively expensive, being based on the strong RSA assumption [CL03] or groups with a pairing [CL04], or if the credentials are efficient (using prime order groups) [PZ13, BL13] they do not support *multi-show unlinkability*. This means that if Alice presents her credential to Bob twice, he can link these presentations (effectively making Alice pseudonymous, rather than anonymous, or requiring Alice to use a fresh credential for each presentation).

With *keyed-verification anonymous credentials* (KVAC) [CMZ14], the issuer and verifier are the same party (or share a key), and so the design can use a MAC in place of a signature scheme. It is then possible to have an efficient credential system constructed in a group of prime order, with multi-show unlinkability. In the present scenario the issuer and verifier are the same party, so a KVAC system is a natural fit.

2.3 MACs and Algebraic MACs

Many popular MAC algorithms are constructed using symmetric-key primitives like hash functions (e.g., HMAC [KBC97]) and block ciphers (e.g., Poly1305-AES [Ber05]). Unlike algebraic MACs, these MACs do not have efficient zero-knowledge proofs associated to them, allowing one to prove possession of a MAC authenticating a message. We use the term algebraic MAC to mean a MAC constructed using group operations. Dodis et al. [DKPW12] study many algebraic MACs, and Chase et al. [CMZ14] show that certain algebraic MACs can be used to construct an efficient type of anonymous credential.

We describe a particular algebraic MAC, called $\mathsf{MAC}_{\mathsf{GGM}}$, that we use as a building block in our new encryption and MAC schemes.

Definition 1. *The* $\mathsf{MAC}_{\mathsf{GGM}}$ *construction [DKPW12, CMZ14] is an algebraic MAC constructed in a group* $\mathbb{G}$ *of prime order* q*, with the following algorithms.*

KeyGen: *choose random* $(x_0, x_1) \in {\mathbb{Z}_q}^2$*, output* $sk = (x_0, x_1)$.

$\mathsf{MAC}(sk, m)$: *choose random* $U \in G$*, output* $\sigma = (U, U^{x_0+x_1 m})$.

$\mathsf{Verify}(sk, (U, U'), m)$: *recompute* $U'' = U^{x_0+x_1 m}$*, output "valid" if* $U'' = U'$*, and "invalid" otherwise.*

In [DKPW12] it is shown that $\mathsf{MAC}_{\mathsf{GGM}}$ satisfies a weak notion of MAC security called selective security (where the adversary must specify the message that will be forged in advance), assuming DDH. In [CMZ14, ABS16], it is shown that $\mathsf{MAC}_{\mathsf{GGM}}$ is uf-cmva secure in the generic group model.

The security notions for algebraic MACs are the same as for traditional MACs.

uf-cma: unforgeability under chosen message attacks

suf-cma: strong unforgeability under chosen message attacks

suf-cmva: strong unforgeability under chosen message and verification attacks

Definition 2. *For a MAC with algorithms* $(\mathsf{KeyGen}, \mathsf{MAC}, \mathsf{Verify})$*, consider the following security game between a challenger* $\mathcal{C}$ *and an attacker* $\mathcal{A}$*.*

1. $\mathcal{C}$ *uses* KeyGen *to generate* sk*. If the MAC has public parameters,* $\mathcal{C}$ *gives them to* $\mathcal{A}$*.*
2. $\mathcal{A}$ *makes queries to* $\mathcal{C}$*.*
   - *MAC query:* $\mathcal{A}$ *submits* m *and* $\mathcal{C}$ *outputs* $\sigma = \mathsf{MAC}(sk, m)$*.* $\mathcal{C}$ *stores* (m, σ) *in a set* M*.*
   - *Verify query:* $\mathcal{A}$ *submits* (σ, m) *and* $\mathcal{C}$ *outputs* $\mathsf{Verify}(sk, \sigma, m)a$
3. $\mathcal{A}$ *outputs* (σ^*, m^*)

We say that $\mathcal{A}$ *wins the* uf-cma *game if no Verify queries are made, and* m^* *is not in* M*. We say that* $\mathcal{A}$ *wins the* suf-cma *security game if no Verify queries are made, and* $(m^*, \sigma^*) \notin M$*. We say that* $\mathcal{A}$ *wins the* suf-cmva *game if* $(m^*, \sigma^*) \notin M$*. The MAC is* uf-cma*-secure if no polynomial-time* $\mathcal{A}$ *wins the* uf-cma *game with probability that is non-negligible in* κ *(and* suf-cma*,* suf-cmva *security are defined analogously).*

A proof of the following lemma is in [BS17, Theorem 6.1]. Basically it says that verification queries don't help an attacker, when looking only at asymptotic security.

Lemma 3. *Let* $\mathcal{M}$ *be a MAC scheme. The security notions* suf-cma *and* suf-cmva *are equivalent. If* $\mathcal{M}$ *is* suf-cma *secure, then it is also* suf-cmva *secure (and vice-versa).*

2.4 Zero-Knowledge Proofs

In multiple places our constructions use zero-knowledge (ZK) proofs to prove knowledge of discrete logarithms and of representations of elements in $\mathbb{G}$. We use the notation introduced by Camenisch and Stadler [CS97]. A non-interactive proof of knowledge π is described by:

$$\pi = \text{PK}\{(x, y, \ldots,) : \text{Predicates using } x, \text{ } y \text{ and public values}\}$$

which means that the prover is proving knowledge of $(x, y, \ldots)$ (all elements of $\mathbb{Z}_q$), such that the predicates are satisfied. Predicates we will use in this paper are knowledge of a discrete logarithm, e.g., $\text{PK}\{(x) : Y = G^x\}$ for public Y and G, and knowledge of a representation using two or more bases, e.g., $\text{PK}\{(x_1, \ldots, x_n) : Y = \prod_{i=1}^{n} G_i{}^{x_i}\}$. We also use multiple predicates, and require that they all be true, e.g., $\text{PK}\{(x, y) : Y = G^x \ \wedge \ Z = G^y H^x\}$. Given two proofs we can combine them by merging the list of secrets and predicates, e.g., proofs $\pi_1 = \text{PK}\{(x) : Y = G^x\}$ and $\pi_2 = \text{PK}\{(x, y) : Z = G^x H^y\}$ combine to give $\pi_3 = \text{PK}\{(x, y) : Y = G^x \ \wedge \ Z = G^x H^y\}$.

There are multiple ways to instantiate the proofs of knowledge we need. The Signal implementation uses a generalization of Schnorr's protocol [BS17, Ch. 19], made non-interactive with the Fiat-Shamir transform [FS87]. As in previous work, we must also assume that the proof system has a strong extraction property; see discussion in [CMZ14, Appendix D].

2.5 Secure Messaging and Signal

In a secure messaging application such as Signal, users send each other encrypted messages with the aid of a server. For the purposes of this document, most details of the Signal Protocol [Sig19] can be abstracted away, leaving a few points which are crucial for understanding the Signal Private Group System in Section 5.

Users can contact the Signal server over a mutually-authenticated secure channel, or over a secure channel that only authenticates the server. For simplicity, we'll describe the former case as an *authenticated channel*, and the latter case as an *unauthenticated channel*. Unauthenticated channels are used when the user wishes to interact with the server without revealing their identity, and thus will be used extensively in the protocols described here.

When users in a Signal group send encrypted messages to the group, they encrypt and send the message to each group member, individually, with end-to-end encryption. The server is given no explicit indication of the difference between group and non-group encrypted messages, apart from traffic analysis.

Users are identified by some *UID*. Users send their profile key attached to encrypted text messages if the recipient is trusted, which we interpret to mean either: the recipient is in the sender's address book; or the sender initiated the conversation; or the sender opted in to sharing profile data with the recipient. Given a user's UID and profile key, you can fetch and decrypt profile data they have uploaded for themselves.

3 A New KVAC and Protocols

In this section we define our new keyed-verification anonymous credential system. We start with the new algebraic MAC that the scheme is based on, then describe protocols for credential issuance and presentation.

3.1 A New Algebraic MAC

Our new MAC construction is constructed in a group $\mathbb{G}$ of prime order q. A new feature that is important for our use case is that the list of attributes may contain a mix of elements of $\mathbb{G}$ (*group attributes*), or integers in $\mathbb{Z}_q$ (*scalar attributes*), while in previous work attributes were restricted to being chosen from $\mathbb{Z}_q$. When using generalized Schnorr proofs in a cyclic group (the most common ZK proof system), the types of statements that can be proven about attributes in $\mathbb{G}$ are limited, however, it does allow us to prove knowledge of the plaintext corresponding to an Elgamal ciphertext, and that the plaintext is certified by a MAC.

Parameters Let κ be a security parameter. The scheme is defined in a group $\mathbb{G}$ of prime order q, written multiplicatively.

The number of attributes in the message space is denoted n. We write $\vec{x}$ to denote a list of values. The scheme requires the following fixed set of group elements:

$$G, G_w, G_{w'}, G_{x_0}, G_{x_1}, G_{y_1}, \ldots, G_{y_n}, G_{m_1}, \ldots, G_{m_n}, G_V$$

generated so that the relative discrete logarithms are unknown, e.g., $G_{m_1} = \mathsf{HashTo}\mathbb{G}(\text{“}\mathtt{m1}\text{”})$.

$\mathsf{KeyGen}(params)$ The secret key is $sk := (w, w', x_0, x_1, (y_1, \ldots, y_n))$ all elements of $\mathbb{Z}_q$. We will write $W := {G_w}^w$, and W is considered part of sk. Optionally, compute the *issuer parameters iparams* (C_W, X, Y_i) as follows:

$$C_W = {G_w}^w {G_{w'}}^{w'}, \;\; I = \frac{G_V}{{G_{x_0}}^{x_0} {G_{x_1}}^{x_1} {G_{y_1}}^{y_1} \ldots {G_{y_n}}^{y_n}}$$

The *iparams* are optional for basic use of the MAC, but are required when the MAC is used in the protocols we consider, therefore we assume *iparams* is always present.

$\mathsf{MAC}(sk, \vec{M})$ The MAC is calculated over a collection of group attributes and scalar attributes. For a given MAC key, each of the n attribute positions is fixed as either a group attribute or scalar attribute. *Group attributes* are elements in $\mathbb{G}$, denoted as M_i, and *scalar attributes* are elements m_j in $\mathbb{Z}_q$, written as $M_j = {G_{m_j}}^{m_j}$. Choose random $t \in \mathbb{Z}_q$, $U \in \mathbb{G}$, and compute

$$V = WU^{x_0+x_1 t}\left(\prod_{i=1}^{n} M_i^{y_i}\right)$$

Output (t, U, V) as the MAC on $\vec{M}$.

$\mathsf{Verify}(sk, \vec{M}, (t, U, V))$ Recompute V as in MAC (denote it V') and accept if $V \stackrel{?}{=} V'$.

3.2 Credential Issuance and Presentation

Here we describe how credentials are issued and presented. To start, we describe issuance when there are no blind attributes, and describe blind issuance in Section 5.9.

Credential Issuance A credential is a MAC from Section 3.1 computed by the issuer on the attributes. The MAC is the triple $(t, U, V) \in \mathbb{Z}_q \times \mathbb{G} \times \mathbb{G}$ where $t \in_R \mathbb{Z}_q$, $U \in_R \mathbb{G}$, and

$$V = WU^{x_0+x_1 t}\left(\prod_{i=1}^{n} M_i^{y_i}\right)$$

The issuer proves knowledge of the secret key, and that (t, U, V) is correct relative to $iparams = (C_W, I)$, with the following proof of knowledge.

$$\begin{aligned}\pi_I = \mathrm{PK}\{(w, w', x_0, x_1, y_1, \ldots, y_4) : \\ C_W = {G_w}^w {G_{w'}}^{w'} \wedge \\ I = \frac{G_V}{{G_{x_0}}^{x_0} {G_{x_1}}^{x_1} {G_{y_1}}^{y_1} \ldots {G_{y_n}}^{y_n}} \wedge \\ V = G^w (U^{x_0})(U^t)^{x_1}\left(\prod_{i=1}^{n} M_i^{y_i}\right)\}\end{aligned}$$

Credential Presentation To present the credential (t, U, V) on attributes $\vec{M}$, a user creates the following proof (called a Show protocol). On its own this only proves that the user holds a valid credential, so we will always add additional predicates that prove more about the attributes. Attributes may be revealed to the verifier, or kept hidden, in which case the user proves knowledge of hidden attributes.

1. Choose $z \in_R \mathbb{Z}_q$ and compute (i ranges from 1 to n):

$$C_{y_i} = \begin{cases} {G_{y_i}}^z M_i \text{ if } i \text{ is a hidden group attribute} \\ {G_{y_i}}^z {G_{m_i}}^{m_i} \text{ if } i \text{ is a hidden scalar attribute} \\ {G_{y_i}}^z \text{ if } i \text{ is a revealed attribute} \end{cases}$$
$$C_{x_0} = {G_{x_0}}^z U$$
$$C_{x_1} = {G_{x_1}}^z U^t$$
$$C_V = {G_V}^z V$$

   along with the value $z_0 = -tz \pmod q$. Let $\mathcal{H}_s$ denote the set of hidden scalar attributes.

2. Compute the following proof of knowledge:

$$\begin{aligned} \pi = \text{PK}\{ (z, z_0, \{m_i\}_{i \in \mathcal{H}_s}, t) : \\ Z = I^z \ \wedge \\ C_{x_1} = {C_{x_0}}^t {G_{x_0}}^{z_0} {G_{x_1}}^z \ \wedge \\ C_{y_i} = \begin{cases} {G_{y_i}}^z {G_{m_i}}^{m_i} \text{ if } i \text{ is a hidden scalar attribute} \\ {G_{y_i}}^z \text{ if } i \text{ is a revealed attribute} \end{cases} \\ \} \end{aligned}$$

3. Output $(C_{x_0}, C_{x_1}, C_{y_1}, \ldots, C_{y_n}, C_V, \pi)$

4. Let $\mathcal{H}$ denote the set of all hidden attributes. The verifier computes

$$Z = \frac{C_V}{(W {C_{x_0}}^{x_0} {C_{x_1}}^{x_1} \prod_{i \in \mathcal{H}} {C_{y_i}}^{y_i} \prod_{i \notin \mathcal{H}} (C_{y_i} M_i)^{y_i})}$$

   using the secret key $(W, x_0, x_1, y_1, \ldots, y_n)$ and revealed attributes, and then verifies π.

4 Verifiable Encryption

Since our credential system supports attributes that are group elements, we can use the Elgamal encryption scheme to create an efficient verifiable encryption scheme [CD00]. By *verifiable*, we mean that we can prove properties about the plaintext in zero-knowledge. In particular, we show how to prove that the plaintext is certified by a credential from Section 3.

Suppose we have a credential certifying a group attribute M_1, and let $Y = G^y$ be an Elgamal public key. The encryption of M_1 with Y is $(E_1, E_2) = (G^r, Y^r M_1)$. To prove

that the plaintext is certified, we add two predicates to the credential presentation proof:

$$E_1 = G^r \wedge C_{y_1}/E_2 = G_{y_1}^z/Y^r \ ,$$

and prove knowledge of r in the same proof. Previous verifiable encryption schemes did not allow us to efficiently encrypt group elements, and thus required more expensive techniques, such as a variant of Paillier's encryption scheme [CS03], or groups with bilinear maps [CHK+11]. We caveat that this is only a promising direction for a new (public-key) verifiable encryption scheme, since the above basic Elgamal scheme is not CCA secure, and we have not carefully analyzed its security. Therefore, for many applications, the previous schemes may be more appropriate.

Since it will be sufficient for our application, we focus on *symmetric-key* verifiable encryption that is CCA secure, and leave construction of a public-key CCA-secure verifiable encryption scheme compatible with our credential scheme as an open problem.

Symmetric-key verifiable encryption with unique ciphertexts Informally, we will need a symmetric-key encryption scheme that (i) has *unique ciphertexts*, meaning that for every plaintext there is at most one ciphertext that will decrypt correctly, (ii) has public verifiability, meaning that we can prove that a ciphertext encrypts a certified plaintext with a key that is consistent with some public parameters, and (iii) is correct under adversarially chosen keys, meaning that it is hard to find a key and message that cause decryption to fail.

4.1 Construction

The notation we use here is chosen to be consistent with later sections.

System parameters A cyclic group $\mathbb{G}$ of prime order q, with associated hash and encode functions (as described in §2.1). Recall that $\mathsf{EncodeTo}\mathbb{G}$ is a function that encodes strings as group elements, that $\mathsf{HashTo}\mathbb{G}$ and $\mathsf{HashTo}\mathbb{Z}_q$ are cryptographic hash functions that hash strings to elements of $\mathbb{G}$ and $\mathbb{Z}_q$ (respectively). We define a fourth function $\mathsf{Derive} : \{0,1\}^{2\kappa} \to (\mathbb{Z}_q)^3$, used to derive three keys from a master key. Derive should also be a cryptographic hash function. Group elements G_a, G_{a_0}, G_{a_1} are chosen such that the relative discrete logs are unknown.

$\mathsf{KeyGen}(1^\kappa)$ Choose the secret key k_0 at random from $\{0,1\}^{2\kappa}$, and derive $k = \mathsf{Derive}(k_0) = (a, a_0, a_1) \in (\mathbb{Z}_q)^3$. We assume that honest parties will not use k that was not derived from a k_0 in this way. Compute the public parameters $pk := {G_a}^a {G_{a_0}}^{a_0} {G_{a_1}}^{a_1}$.

$\mathsf{Enc}(k, m)$ Compute $M_1 = \mathsf{EncodeTo}\mathbb{G}(m)$, $M_2 = \mathsf{HashTo}\mathbb{G}(m)$ and $m_3 = \mathsf{HashTo}\mathbb{Z}_q(m)$. Compute

$$E_1 = {M_2}^{a_0 + a_1 m_3}$$
$$E_2 = (E_1)^a M_1$$

$\mathsf{Dec}(k, E_1, E_2)$ First compute $m' = \mathsf{DecodeFrom}\mathbb{G}(E_2/E_1^a)$. If

$$E_1 = \mathsf{HashTo}\mathbb{G}(m')^{a_0 + a_1 \mathsf{HashTo}\mathbb{Z}_q(m')}$$

output m, otherwise output $\perp$.

$\mathsf{Prove}(k, pk, E_1, E_2, \vec{C})$ To prove that the ciphertext (E_1, E_2) encrypts the plaintext committed in the list of commitments $\vec{C}$:

$$C_{y_1} := M_1 {G_{y_1}}^z, C_{y_2} := M_2 {G_{y_2}}^z, \text{ and } C_{y_3} := {G_{m_3}}^{m_3} {G_{y_3}}^z,$$

first compute ${C_{y_2}}' := {C_{y_2}}^{a_1}$ and the scalar $z_1 = -z(a_0 + a_1 m_3)$, then create the proof

$$\begin{aligned}
\pi_{\mathsf{Enc}} = \mathrm{PK}\{(a, a_0, a_1, m_3, z, z_1) : \\
& pk = {G_a}^a {G_{a_0}}^{a_0} {G_{a_1}}^{a_1} \ \wedge \\
& C_{y_1}/E_2 = {G_{y_1}}^z / {E_1}^a \ \wedge \qquad //\text{plaintext is } M_1 \\
& {C_{y_2}}' = {C_{y_2}}^{a_1} \ \wedge \\
& E_1 = {C_{y_2}}^{a_0} ({C_{y_2}}')^{m_3} {G_{y_2}}^{z_1} \ \wedge \qquad //E_1 \text{ is well-formed} \\
& C_{y_3} = {G_{y_3}}^z {G_{m_3}}^{m_3} \ \}
\end{aligned}$$

$\mathsf{Verify}(pk, \pi_{\mathsf{Enc}}, \vec{C}, C'_{y_2})$ Accept if π_{Enc} verifies, otherwise reject.

Discussion When we use the Prove function, it will be combined with the credential presentation proof, which creates the triple of commitments $\vec{C}$ (and this is why they use the same z value). The E_1 part of the ciphertext can be seen as part of a $\mathsf{MAC}_{\mathsf{GGM}}$ authentication tag on m_3, computed with key (a_0, a_1), where the MAC function's choice of a group element is derandomized by hashing the message. When $\mathsf{HashTo}\mathbb{G}$ is modelled as a random oracle, security of this 1-element MAC reduces to the security of $\mathsf{MAC}_{\mathsf{GGM}}$ and may be of independent interest, as it halves the tag size.

The Derive function in key generation serves two purposes. First, when sharing group keys amongst themselves, group members can share a short master key, saving bandwidth. Second, it ensures that (a, a_0, a_1) are not a degenerate value (such as all zero), that might be used to break the correctness under adversarially chosen keys property (Definition 8).

5 The Signal Private Group System

In the next sections we provide a high-level description of the data objects and operations in the the Signal Private Group System. Following this we describe these objects, and their associated proofs of knowledge, in detail.

5.1 General Data Objects

UID: A 16-byte UUID (universally unique identifier) representing a user.

ServerSecretParams: A set of secret values the server uses to issue credentials and verify zero-knowledge proofs about credentials.

ServerPublicParams: A set of public values which are derived from *ServerSecretParams* and which are known to all users. The *ServerPublicParams* are used by users to verify issued credentials and produce zero-knowledge proofs about credentials.

5.2 Data Objects for Authentication

AuthCredentialResponse: A message sent from the server to a user containing an *AuthCredential* and a zero-knowledge proof that this credential was constructed correctly. Since the corresponding request is trivial, we omit it.

AuthCredential: A credential with attributes based on the *UID* and a redemption time which specifies the day on which this credential is valid. Since none of the attributes are secret (known only to the user), the credential itself must be kept secret, and issued over a secure channel.

AuthCredentialPresentation: A message sent from a user to the server containing a *UidCiphertext*, a redemption time, and the credential presentation proof π_A from Section 5.11.

5.3 Data Objects for Profile Keys

The data objects in this section are all related to profile keys, commitment and credentials.

ProfileKey: A 32-byte key used for symmetric-key encryption of profile data. A user shares their *ProfileKey* with users they trust, but not with the server. At any point in time a UID is associated with a single profile key, but the profile key associated with a UID could be changed at the user's discretion. The uses of profile data are outside the scope of this document, but two examples are a user's screen name and profile picture.

ProfileKeyCommitment: A deterministic commitment to a *ProfileKey*.

ProfileKeyVersion: A collision-resistant hash of the *ProfileKeyCommitment*.

ProfileKeyCredentialRequest: A message sent from a user to the server to request a *ProfileKeyCredential*. The message contains a *ProfileKeyVersion*, a proof of knowledge of the corresponding *ProfileKey*, and some additional data to help the server perform a blinded credential issuance.

ProfileKeyCredentialResponse: A message sent from the server to a user containing a *ProfileKeyCredential* and the blind issuance proof π_B from Section 5.9.

ProfileKeyCredential: A credential with attributes based on a *UID* and *ProfileKey*. Note that an *AuthCredential* for a UID is issued only to the user who owns that UID, whereas *ProfileKeyCredentials* are issued to anyone who knows the profile key for a UID. Since profile keys are shared between users, many users will hold a *ProfileKeyCredential* certifying a user's *ProfileKey*.

ProfileKeyCredentialPresentation: A message sent from a user to the server containing a *UidCiphertext*, a *ProfileKeyCiphertext*, and a zero-knowledge proof of knowledge of some *ProfileKeyCredential* issued over the encrypted *UID* and *ProfileKey*.

5.4 Data Objects for Groups

The data objects in this section exist for a specific group.

GroupMasterKey: A random value which the *GroupSecretParams* are derived from. When a new user is added or invited to a group, the user adding them will send the new member the group's *GroupMasterKey* via an encrypted message, so the new member can derive the *GroupSecretParams*. Each encrypted message sent within the group will also contain a copy of the *GroupMasterKey*, in case the initial message fails to arrive. Note that a user who has acquired a group's *GroupMasterKey* and then leaves the group (or is deleted) retains the ability to collude with a malicious server to encrypt and decrypt group entries. We deem this risk acceptable for now due to the complexities in rapid and reliable rekey of the *GroupMasterKey*.

GroupSecretParams: A set of secret values which group members use to encrypt and decrypt *UidCiphertexts* and *ProfileKeyCiphertexts*, as well as construct zero-knowledge proofs about these ciphertexts.

GroupPublicParams: A set of public values which are derived from some *GroupSecretParams*. The *GroupPublicParams* are stored on the server to represent a group. The server uses the *GroupPublicParams* to verify zero-knowledge proofs about *UidCiphertexts* and *ProfileKeyCiphertexts*.

GroupIdentifier: A collision-resistant hash of the *GroupPublicParams*.

UidCiphertext: A deterministic encryption of a *UID* using *GroupSecretParams*.

ProfileKeyCiphertext: A deterministic encryption of a *ProfileKey* using *GroupSecretParams*.

Role: A value specifying what access privileges a user has to modify the group. For example, a user with an administrator role may have more privileges than other users. Discussion of specific roles is out of scope of this document. Roles are enforced by the server, not by a cryptographic mechanism.

5.5 Operations for Credentials

Here we describe the protocols used to get profile and authentication credentials.

CommitToProfileKey

1. The user generates a random *ProfileKey*, computes a *ProfileKeyCommitment* from the *ProfileKey*, and computes a *ProfileKeyVersion* by hashing their *ProfileKeyCommitment*.
2. The user sends the (*ProfileKeyVersion*, *ProfileKeyCommitment*) pair over the authenticated channel to the server.
3. The server stores the *ProfileKeyCommitment* associated with the authenticated user's *UID* and the *ProfileKeyVersion*.
4. Note that the server does not check anything about the *ProfileKeyCommitment* or *ProfileKeyVersion*. If a user registers an invalid commitment, that will have the same effect as distributing an invalid profile key to other users, which the user cannot be prevented from doing.

GetProfileKeyCredential This operation provisions a user with a *ProfileKeyCredential* for some (*UID*, *ProfileKey*) if and only if the user knows a *ProfileKey* matching a *ProfileKeyCommitment* that was previously sent to the server via the *CommitToProfileKey* operation.

1. The user has a *ProfileKey* for a target *UID* (their UID or another user's UID).
2. The user derives a *ProfileKeyVersion* from the *ProfileKey*, and computes a *ProfileKeyCredentialRequest* from the *ProfileKey*.

3. The user sends the (UID, $ProfileKeyVersion$, $ProfileKeyCredentialRequest$) over an unauthenticated channel to the server.

4. If the server has a stored $ProfileKeyCommitment$ for the specified UID and $ProfileKeyVersion$, the server verifies the proof of knowledge in the $ProfileKeyCredentialRequest$.

   (a) If verification succeeds the server generates a $ProfileKeyCredentialResponse$.

   (b) Otherwise (if the lookup fails or verification of $ProfileKeyCredentialRequest$ fails) the server returns an error.

5. The user verifies the proof of knowledge in the $ProfileKeyCredentialResponse$, and if verification succeeds the user stores a $ProfileKeyCredential$ for the target UID.

GetAuthCredential

1. The user contacts the server over an authenticated channel and requests an $AuthCredential$ for some redemption date.

2. If the date is in the allowed range, the server returns an $AuthCredentialResponse$ for the date, and returns an error otherwise.

3. The user verifies the proof of knowledge in the $AuthCredentialResponse$ and stores an $AuthCredential$ if the verification succeeds.

5.6 Operations for Group Management

In this section we describe the operations used to manage groups.

CreateGroup

1. The user generates a pair ($GroupSecretParams$, $GroupPublicParams$) for the new group.

2. The user encrypts their UID into a $UidCiphertext$ using the $GroupSecretParams$.

3. The user creates an $AuthCredentialPresentation$ containing the $UidCiphertext$ and a proof of knowledge of an $AuthCredential$ matching the encrypted UID.

4. For each (UID, $ProfileKey$) that will be members of the new group, including this user (the group creator), the user:

   (a) Encrypts the UID and $ProfileKey$ into a $UidCiphertext$ and $ProfileKeyCiphertext$ using the $GroupSecretParams$.

(b) Creates a *ProfileKeyCredentialPresentation* containing these ciphertexts and a proof of knowledge of a *ProfileKeyCredential* matching the ciphertexts.

5. The user contacts the server over an unauthenticated channel and sends:
   - *GroupPublicParams*
   - *AuthCredentialPresentation* for itself
   - List of initial members including the creator, each containing a *ProfileKeyCredentialPresentation* and a *Role*.
6. The server verifies
   (a) The proof of knowledge in the *AuthCredentialPresentation* using the *UidCiphertext* from the same *AuthCredentialPresentation*.
   (b) The proof of knowledge in each *ProfileKeyCredentialPresentation* using the *UidCiphertext* and *ProfileKeyCiphertext* from the same *ProfileKeyCredentialPresentation*.
   (c) That the *AuthCredentialPresentation* contains a *UidCiphertext* that matches some *UidCiphertext* in a *ProfileKeyCredentialPresentation*.
   (d) That the *GroupIdentifier* derived from these *GroupPublicParams* is not used for any existing group.

   If all these verifications succeed, the server stores the *GroupPublicParams* and a list of (*UidCiphertext*, *ProfileKeyCiphertext*, *Role*) tuples associated to the *GroupIdentifier*. Otherwise the server returns an error.

AuthAsGroupMember This operation uses an unauthenticated channel so that the server does not learn the user's *UID*, but the channel is authenticated to a particular *UidCiphertext* within a group. This operation is used by other operations for group management.

1. The user recomputes their *UidCiphertext* for the group and creates an *AuthCredentialPresentation* to prove knowledge of an *AuthCredential* matching the encrypted *UID*.
2. The user contacts the server over an unauthenticated channel and sends:
   - *GroupPublicParams* identifying a particular group, and
   - The *AuthCredentialPresentation*
3. The server verifies the proof of knowledge in the *AuthCredentialPresentation* using the transmitted *GroupPublicParams* and the *UidCiphertext* which is contained within the *AuthCredentialPresentation*.

4. If this verification succeeds, and the *GroupPublicParams* corresponds to an actual group, and the *AuthCredentialPresentation's UidCiphertext* corresponds to a user in the group, then the user is authenticated as the corresponding group member. Otherwise an error is returned.

AddInvitedGroupMember This operation is used when a group member would like to add some target user to the group but doesn't know the target's *ProfileKey*. The target will be sent a *GroupMasterKey* and invited to join the group, and can accept the invitation and join by using the *UpdateProfileKey* operation.

1. The user and server execute $AuthAsGroupMember$, and abort if it fails.
2. The user encrypts another user's *UID* as a *UidCiphertext* using the *GroupSecretParams*, and sends this ciphertext, and a *Role* for the new user, to the server.
3. The server verifies that the *UidCiphertext* does not already exist in the group, and that the user's *Role* allows them invite other users.
4. The server stores the *UidCiphertext* and *Role* in the group with no associated *ProfileKeyCiphertext* (this user is only an invited member, not a full member, of the group). Note that the server does not check anything about the *UidCiphertext*, aside from checking for duplicates.

AddGroupMember

1. The user and server execute $AuthAsGroupMember$, and abort if it fails.
2. The user encrypts the new user's (*UID*, *ProfileKey*) into a *UidCiphertext* and *ProfileKeyCiphertext* using the *GroupSecretParams*, then creates a *ProfileKeyCredentialPresentation* proving these ciphertexts are well-formed and consistent, and sends it to the server, along with a *Role* for the new user.
3. The server verifies that
   (a) The authenticated user's *Role* allows them to add other users.
   (b) The *UidCiphertext* does not already exist in the group as a full member. If the *UidCiphertext* exists in the group as an invited member (i.e., missing *ProfileKeyCiphertext*), then this operation proceeds and adds the user as a full member.
   (c) The proof of knowledge in the *ProfileKeyCredentialPresentation* is valid

   If these checks succeed, the server stores the tuple (*UidCiphertext*, *ProfileKeyCiphertext*, *Role*) in the group. Otherwise the server returns an error.

DeleteGroupMember

1. The user and server execute $AuthAsGroupMember$, and abort if it fails.
2. The user sends the *UidCiphertext* of another user or themselves (the target user).
3. The server checks whether the authenticated user's *Role* allows them to delete the target user. If not, this operation fails.
4. The corresponding entry in the group membership is deleted.
5. If all members are deleted then the group is deleted.

FetchGroupMembers

1. The user and server execute $AuthAsGroupMember$, and abort if it fails.
2. All of the (*UidCiphertext*, *ProfileKeyCiphertext*) pairs are returned for full members as well as invited members (members without a *ProfileKeyCiphertext*).

UpdateProfileKey

1. The user and server execute $AuthAsGroupMember$, and abort if it fails.
2. The user encrypts their own *UID* and *ProfileKey* into a *UidCiphertext* and *ProfileKeyCiphertext* using the *GroupSecretParams*, creates a $ProfileKeyCredentialPresentation$ containing these ciphertexts and a proof of knowledge of a *ProfileKeyCredential* matching these ciphertexts, and then sends the $ProfileKeyCredentialPresentation$ to the server.
3. The server verifies
   (a) The proof of knowledge in the $ProfileKeyCredentialPresentation$.
   (b) That the *UidCiphertext* in the $AuthCredentialPresentation$ is the same as the *UidCiphertext* in the $ProfileKeyCredentialPresentation$ and matches an entry in the group's membership list.

   If these checks succeed the *ProfileKeyCiphertext* is replaced (or added in case of an previously invited user), otherwise an error is returned.

5.7 System Parameters and Server Parameters

The Signal Private Group System involves two types of credentials, *AuthCredentials* and *ProfileKeyCredentials*. They are each issued with a separate issuer key and *iparams*.

System Parameters In addition to the parameters of the MAC scheme, the group elements $(G_a, G_{a_0}, G_{a_1}, G_b, G_{b_0}, G_{b_1}, H, G, H_1, H_2)$ are generated so that the relative discrete logarithms are unknown, e.g., $G_{a_0} = \mathsf{HashTo}\mathbb{G}(\text{“a0”})$.

ServerSecretParams and ServerPublicParams The *ServerSecretParams* contains two secret keys for the MAC scheme. The *ServerPublicParams* contains the corresponding issuer parameters, denoted $iparams_A$ (for auth credentials) and $iparams_P$ (for profile key credentials).

5.8 Auth Credentials

An *AuthCredential* has four attributes:

1. $M_1 := \mathsf{EncodeTo}\mathbb{G}(UID)$, a reversible encoding of *UID*,
2. $M_2 := \mathsf{HashTo}\mathbb{G}(UID)$,
3. $M_3 := {G_3}^{m_3}$, where $m_3 = \mathsf{HashTo}\mathbb{Z}_q(UID)$,
4. $M_4 := {G_4}^{m_4}$, where $m_4 \in \mathbb{Z}_q$, is a "redemption time".

An *AuthCredentialResponse* contains an algebraic MAC for the credential, and also the proof of issuance π_I. The user verifies this proof in the *GetAuthCredential* operation, using attribute values (M_1, M_2, m_3) which the user derives from their own UID.

5.9 Profile Key Commitments and Credentials

A *ProfileKeyCredential* links a *UID* and a *ProfileKey*. The first three credential attributes encode the UID and are the same as the *AuthCredential*, and the last three encode the *ProfileKey*:

1. $N_1 = M_1 = \mathsf{EncodeTo}\mathbb{G}(UID)$,
2. $N_2 = M_2 = \mathsf{HashTo}\mathbb{G}(UID)$,
3. $N_3 = M_3 = {G_{m_3}}^{m_3}$ where $m_3 = \mathsf{HashTo}\mathbb{Z}_q(UID)$,
4. $N_4 = \mathsf{EncodeTo}\mathbb{G}(ProfileKey)$, an encoding of the profile key,
5. $N_5 = \mathsf{HashTo}\mathbb{G}(ProfileKey, UID)$, a hash of the profile key and UID, and
6. $N_6 = {G_{m_6}}^{n_6}$, where $n_6 = \mathsf{HashTo}\mathbb{Z}_q(ProfileKey, UID)$, a hash of the profile key and UID.

A *ProfileKeyCommitment* commits to the three values N_4, N_5 and N_6. Since N_4 and N_5 are group elements and not scalars, we can't simply use Pedersen's commitment scheme. Instead, a *ProfileKeyCommitment* is the triple of values $(J_1, J_2, J_3) = (G^{n_6}, {H_1}^{n_6} N_4, {H_2}^{n_6} N_5)$. Note that this commitment scheme is not perfectly hiding, but since *ProfileKeys* are assumed to have high min-entropy, this is sufficient. Further, the

commitment is deterministic since (N_4, N_5, n_6) is derived from the *ProfileKey*, thus any user with a *ProfileKey* can reconstruct the *ProfileKeyCommitment*.

The $ProfileKeyVersion$ is a collision-resistant hash of the *ProfileKeyCommitment* and is used as an identifier for the *ProfileKey*.

Blind Issuance Issuance of $ProfileKeyCredentials$ differs from $AuthCredentials$ because the $ProfileKeyCredential$ attribute values are not all known to the server, so the server can't simply calculate and return a MAC to the user.

Instead, the user and server will perform a blind issuance protocol, based on the same idea as in [CMZ14]. The *ProfileKeyCredentialRequest* will contain an Elgamal encryption of the blinded attributes (N_4, N_5, N_6) and a proof that these values match the *ProfileKeyCommitment*. The server will use the homomorphic properties of Elgamal encryption to create an encrypted MAC and return it to the user along with a proof of correctness, in a *ProfileKeyCredentialResponse*. The user will verify the proof and then decrypt the MAC to recover their $ProfileKeyCredential$.

To generate the *ProfileKeyCredentialRequest* the user generates an Elgamal key pair $(y, Y = G^y)$, where G is a generator of $\mathbb{G}$. The blind attributes (N_4, N_5, n_6) are encrypted as

$$\begin{aligned}(D_1, D_2) &= (G^{r_1}, Y^{r_1} N_4) \\ (E_1, E_2) &= (G^{r_2}, Y^{r_2} N_5) \\ (F_1, F_2) &= (G^{r_3}, Y^{r_3} {G_{m_6}}^{n_6})\end{aligned}$$

for random r_1, r_2 and r_3. The *ProfileKeyCredentialRequest* contains the ciphertexts, the public key Y, and a proof that the encrypted values match the commitment $(J_1, J_2, J_3) = (G^{n_6}, {H_1}^{n_6} N_4, {H_2}^{n_6} N_5)$:

$$\begin{aligned}\pi_B = \mathrm{PK}\{(y, r_1, r_2, r_3, n_6) : \\ & Y = G^y \wedge D_1 = G^{r_1} \wedge E_1 = G^{r_2} \wedge F_1 = G^{r_3} \wedge J_1 = G^{n_6} \wedge \\ & D_2 / J_2 = Y^{r_1} / {H_1}^{n_6} \wedge \\ & E_2 / J_3 = Y^{r_2} / {H_2}^{n_6} \wedge \\ & F_2 = Y^{r_3} {G_{m_6}}^{n_6}\}\end{aligned}$$

To create a *ProfileKeyCredentialResponse* after verifying the *ProfileKeyCredentialRequest* the server will create a partial credential (t, U, V') that covers the unblinded attributes, and encrypt V' with the user's public key Y to get $(R_1, R_2) = (G^{r'}, Y^{r'} V')$ for a random r'. Then the server will compute

$$(S_1, S_2) = (D_1{}^{y_4} E_1{}^{y_5} F_1{}^{y_6} R_1, D_2{}^{y_4} E_2{}^{y_5} F_2{}^{y_6} R_2) \ .$$

Because Elgamal encryption is homomorphic, the ciphertext (S_1, S_2) is an encryption of V for a credential (t, U, V) which covers both blinded and revealed attributes. With the

attributes $(N_1, \ldots, N_6)$ as described above, (S_1, S_2) will be:

$$\begin{aligned} S_1 &= G^{y_4 r_1 + y_5 r_2 + y_6 r_3 + r'}, \\ S_2 &= Y^{y_4 r_1 + y_5 r_2 + y_6 r_3 + r'} W U^{x_0 + t x_1} \prod_{i=1}^{6} N_i^{y_i} \\ &= Y^{y_4 r_1 + y_5 r_2 + y_6 r_3 + r'} V \end{aligned}$$

The server can prove that (S_1, S_2) were calculated correctly by modifying the issuance proof to be the following proof π_{BI}:

$$\begin{aligned} \mathrm{PK}\{(w, w', y_1, \ldots, y_6, x_0, x_1, r') : \\ C_W &= {G_w}^{w} {G_{w'}}^{w'} \ \wedge \\ I &= \frac{G_V}{{G_{x_0}}^{x_0} {G_{x_1}}^{x_1} {G_{y_1}}^{y_1} \ldots {G_{y_6}}^{y_6}} \ \wedge \\ S_1 &= {D_1}^{y_4} {E_1}^{y_5} {F_1}^{y_6} G^{r'} \ \wedge \\ S_2 &= {D_2}^{y_4} {E_2}^{y_5} {F_2}^{y_6} Y^{r'} {G_w}^{w} (U^{x_0})(U^t)^{x_1} {M_1}^{y_1} {M_2}^{y_2} {M_3}^{y_3} \} \end{aligned}$$

The server sends $(S_1, S_2, t, U, \pi_{BI})$ to the user, and if π_{BI} is valid, the user decrypts $V = S_2 / {S_1}^{y}$ and outputs the credential (t, U, V) with attributes $(N_1, \ldots, N_6)$.

5.10 Verifiable Encryption of UIDs and Profile Keys

Encryption of UIDs and *ProfileKeys* is done with the symmetric-key scheme from Section 4.1. Both encryption and decryption require the secret key (*GroupSecretParams*). The public key (*GroupPublicParams*) only exists to allow users to prove to the server that ciphertexts are well-formed.

GroupSecretParams and GroupPublicParams The *GroupSecretParams* are $(a, a_0, a_1, b, b_0, b_1) \in {\mathbb{Z}_q}^6$ derived from a randomly-chosen *GroupMasterKey*. The *GroupPublicParams* are $pk := {G_a}^{a} {G_{a_0}}^{a_0} {G_{a_1}}^{a_1} {G_b}^{b} {G_{b_0}}^{b_0} {G_{b_1}}^{b_1}$.

Encryption of UIDs Recall that $M_1 = \mathsf{EncodeTo}\mathbb{G}(UID)$, $M_2 = \mathsf{HashTo}\mathbb{G}(UID)$, and $m_3 = \mathsf{HashTo}\mathbb{Z}_q(UID)$. To encrypt a *UID* to a *UidCiphertext* (E_{A_1}, E_{A_2}) calculate:

$$\begin{aligned} E_{A_1} &= {M_2}^{(a_0 + a_1 m_3)} \\ E_{A_2} &= {E_{A_1}}^{a} M_1 \end{aligned}$$

To decrypt the *UidCiphertext* first compute:

$$M_1' = E_{A_2} / {E_{A_1}}^a$$

then decode M_1' to get UID', compute $M_2' = \mathsf{HashTo}\mathbb{G}(UID')$, and $m_3' = \mathsf{HashTo}\mathbb{Z}_q(UID')$. Then perform the following checks and return UID' if they succeed, $\perp$ otherwise:

$$E_{A_1} \stackrel{?}{=} (M_2')^{a_0 + a_1 m_3'}$$
$$E_{A_1} \stackrel{?}{\neq} 1$$

Encryption of ProfileKeys Recall that $N_4 = \mathsf{EncodeTo}\mathbb{G}(\mathit{ProfileKey})$, $N_5 = \mathsf{HashTo}\mathbb{G}(\mathit{ProfileKey}, \mathit{UID})$ and $n_6 = \mathsf{HashTo}\mathbb{Z}_q(\mathit{ProfileKey}, \mathit{UID})$. To encrypt a *ProfileKey* as a *ProfileKeyCiphertext* (E_{B_1}, E_{B_2}) calculate:

$$E_{B_1} = {N_5}^{(b_0 + b_1 n_6)}$$
$$E_{B_2} = {E_{B_1}}^b N_4$$

To decrypt the *ProfileKeyCiphertext* first compute:

$$N_4' = E_{B_2} / {E_{B_1}}^b$$

then decode N_4' to get $\mathit{ProfileKey}'$, and compute $N_5' = \mathsf{HashTo}\mathbb{G}(\mathit{ProfileKey}', \mathit{UID})$ and $n_6' = \mathsf{HashTo}\mathbb{Z}_q(\mathit{ProfileKey}', \mathit{UID})$. Then perform the following checks and return $\mathit{ProfileKey}'$ if they succeed, $\perp$ otherwise:

$$E_{B_1} \stackrel{?}{=} (N_5')^{b_0 + b_1 n_6'}$$
$$E_{B_1} \stackrel{?}{\neq} 1$$

5.11 Presenting an AuthCredential

An *AuthCredentialPresentation* contains a *UidCiphertext*, a redemption time, and a proof of knowledge calculated as follows:

1. Recompute (E_{A_1}, E_{A_2}) from UID and (a, a_0, a_1) as described in Section 5.10.

2. Choose $z \in_R \mathbb{Z}_q$ and compute

$$\begin{aligned} C_{y_1} &= {G_{y_1}}^z M_1 & C_{x_0} &= {G_{x_0}}^z U \\ C_{y_2} &= {G_{y_2}}^z M_2 & C_{x_1} &= {G_{x_1}}^z U^t \\ C_{y_3} &= {G_{y_3}}^z {G_{m_3}}^{m_3} & C_V &= {G_V}^z V \\ C_{y_4} &= {G_{y_4}}^z & {C_{y_2}}' &= {C_{y_2}}^{a_1} \end{aligned}$$

along with two values in $\mathbb{Z}_q$: $z_0 = -tz$ and $z_1 = -z(a_0 + a_1 m_3)$.

3. Compute the following proof of knowledge:

$$\begin{aligned} \pi_A = \mathrm{PK}\{(z, sk, z_0, z_1, m_3, t) : \\ & Z = I^z \ \wedge \\ & C_{x_1} = {C_{x_0}}^t {G_{x_0}}^{z_0} {G_{x_1}}^z \ \wedge \\ & pk = {G_a}^a {G_{a_0}}^{a_0} {G_{a_1}}^{a_1} {G_b}^b {G_{b_0}}^{b_0} {G_{b_1}}^{b_1} \ \wedge \\ & C_{y_1}/E_{A_2} = {G_{y_1}}^z / {E_{A_1}}^a \ \wedge \qquad //\text{plaintext is } M_1 \\ & {C_{y_2}}' = {C_{y_2}}^{a_1} \ \wedge \\ & E_{A_1} = {C_{y_2}}^{a_0} ({C_{y_2}}')^{m_3} {G_{y_2}}^{z_1} \ \wedge \qquad //E_{A_1} \text{ is well-formed} \\ & C_{y_3} = {G_{y_3}}^z {G_{m_3}}^{m_3} \ \wedge \\ & C_{y_4} = {G_{y_4}}^z \ \} \end{aligned}$$

4. Output $(C_{x_0}, C_{x_1}, C_{y_1}, \ldots, C_{y_4}, C_V, {C_{y_2}}', E_{A_1}, E_{A_2}, \pi_A)$

5. The server computes

$$Z = C_V / (W {C_{x_0}}^{x_0} {C_{x_1}}^{x_1} {C_{y_1}}^{y_1} {C_{y_2}}^{y_2} {C_{y_3}}^{y_3} (C_{y_4} {G_{m_4}}^{m_4})^{y_4})$$

using the timestamp m_4 and the secret key $(W, x_0, x_1, y_1, \ldots, y_4)$, and then verifies π_A.

5.12 Presenting a ProfileKeyCredential

A *ProfileKeyCredentialPresentation* contains a *UidCiphertext*, a *ProfileKeyCiphertext*, and a proof of knowledge calculated as follows:

1. Choose random z, r then compute

$$\begin{aligned} C_{y_i} &= {G_{y_i}}^z N_i \text{ for } i = 1, \ldots, 6 & C_V &= {G_V}^z V \\ C_{x_0} &= {G_{x_0}}^z U & {C_{y_2}}' &= {C_{y_2}}^{a_1} \\ C_{x_1} &= {G_{x_1}}^z U^t & {C_{y_5}}' &= {C_{y_5}}^{b_1} \end{aligned}$$

along with three values in Z_q: $z_0 = -tz$, $z_1 = -z(a_0 + a_1 m_3)$, and $z_2 = -z(b_0 + b_1 n_6)$.

2. Then compute the proof of knowledge

$$\begin{aligned}\pi_P = \mathrm{PK}\{(sk, & m_3, n_6, z, z_0, z_1, z_2, t) : \\ & Z = I^z \wedge \\ & C_{x_1} = {C_{x_0}}^t {G_{x_0}}^{z_0} {G_{x_1}}^z \wedge \\ & pk = {G_a}^a {G_{a_0}}^{a_0} {G_{a_1}}^{a_1} {G_b}^b {G_{b_0}}^{b_0} {G_{b_1}}^{b_1} \ \wedge \\ & C_{y_1}/E_{A_2} = {G_{y_1}}^z / {E_{A_1}}^a \ \wedge \qquad //\text{plaintext is } N_1 \\ & {C_{y_2}}' = {C_{y_2}}^{a_1} \ \wedge \\ & E_{A_1} = {C_{y_2}}^{a_0} ({C_{y_2}}')^{m_3} {G_{y_2}}^{z_1} \ \wedge \qquad //E_{A_1} \text{ is well-formed} \\ & C_{y_3} = {G_{y_3}}^z {G_{m_3}}^{m_3} \ \wedge \\ & C_{y_4}/E_{B_2} = {G_{y_4}}^z / {E_{B_1}}^b \ \wedge \qquad //\text{plaintext is } N_4 \\ & {C_{y_5}}' = {C_{y_5}}^{b_1} \ \wedge \\ & E_{B_1} = {C_{y_5}}^{b_0} ({C_{y_5}}')^{n_6} {G_{y_5}}^{z_2} \ \wedge \qquad //E_{B_1} \text{ is well-formed} \\ & C_{y_6} = {G_{y_6}}^z {G_{m_6}}^{n_6} \\ & \}\end{aligned}$$

and output $(\{C_{y_i}\}_{i=1}^6, C_{x_0}, C_{x_1}, C_V, {C_{y_2}}', {C_{y_5}}', \pi_P)$.

3. The server computes

$$Z = C_V / (W {C_{x_0}}^{x_0} {C_{x_1}}^{x_1} \prod_{i=1}^{6} {C_{y_i}}^{y_i})$$

using the secret key $(W, x_0, x_1, y_1, \ldots, y_5)$, and then verifies π_P.

6 Security analysis

In this section we analyze the security of: the encryption scheme defined in Section 4.1, our new algebraic MAC from Section 3.1, and the security of the keyed-verification anonymous credential system build on top of the MAC.

6.1 Security of Encryption

We first give a definition of weak pseudorandom functions (wPRF) [NR95], tailored to our setting.

Definition 4. *Let $\mathbb{G}$ be a group of prime order q. A function $f_k : \mathbb{G} \to \mathbb{G}$ with key $k \in \mathbb{Z}_q$ is said to be a weak pseudorandom function (wPRF), if the following two sequences (of length polynomial in κ) are indistinguishable*

$$(x_1, f_k(x_1), (x_2, f_k(x_2)), \ldots$$

and

$$(x_1, r_1), (x_2, r_2), \ldots$$

where x_i and r_i are sampled from the uniform distribution on $\mathbb{G}$.

Weak PRFs are useful because they are PRFs when the inputs are chosen at random. The specific wPRF we use in our security analysis is the function $f_k : \mathbb{G} \to \mathbb{G}$ defined as $f_k(x) = x^k$. The fact that f_k is a wPRF is known in the literature (e.g,. [DKPW12]).

First we show that the encryption scheme of §4.1 is CPA secure. Our definition uses a real-or-random experiment [BDJR97], and to model the deterministic property, the encryption oracle can only be queried once per plaintext.

Definition 5. *For a deterministic symmetric key cipher with public verifiability* ($\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}$)*, we define CPA security by the following security game.*

- *The challenger selects $(k, pk) \leftarrow \mathsf{KeyGen}(1^\kappa)$, and a random bit b.*
- *The attacker is given pk and an oracle $\mathcal{O}_k(\cdot)$ that outputs $\mathsf{Enc}_K(\cdot)$ when $b = 0$ and $\mathsf{Enc}_K(r)$ for uniformly random r (of the same length) when $b = 1$. $\mathcal{O}_k$ outputs $\perp$ if the input was previously queried.*
- *$\mathcal{A}$ outputs a guess bit b' and wins if $b = b'$.*

In the following proof and throughout this section, we use the shorthand H to denote $\mathsf{HashTo}\mathbb{G}$, and h to denote $\mathsf{HashTo}\mathbb{Z}_q$.

Theorem 6. *The encryption scheme of Section 4.1 is CPA secure, in the random oracle model, assuming the DDH problem is hard in $\mathbb{G}$.*

Proof. Let $\mathcal{A}$ be an attacker in the CPA game. We construct a DDH distinguisher $\mathcal{B}$ that uses $\mathcal{A}$ as a subroutine. We proceed with a hybrid argument. Let G_i be the probability that $\mathcal{A}$ outputs 1 in Game i. Suppose that DDH is ϵ_{ddh}-hard in $\mathbb{G}$, i.e., no polynomial time algorithm exists for DDH that succeeds with probability better than ϵ_{ddh}.

Game 0 This is the real CPA game, where $\mathcal{B}$ is the challenger, and H is modeled as a random oracle. The probability that $\mathcal{A}$ breaks CPA security of the scheme is G_0.

Game 1 is the same as Game 0, but $\mathcal{B}$ replaces pk with a random value. We claim that $\mathsf{G}_1 - \mathsf{G}_0 \leq \epsilon_{\text{ddh}}$. Let $\mathcal{B}$ have a DDH triple as input, $(A, B, C) = (G^{a_0}, G^b, G^{a_0 b}$ or $G^z)$ for a random $z \in \mathbb{Z}_q$. $\mathcal{B}$ chooses a, a_1 at random and creates pk, by first programming H so that $G_{a_0} = B = G^b$ (this is possible since G_{a_0} is derived using H). Then $\mathcal{B}$ computes $pk = {G_a}^a C {G_{a_1}}^{a_1}$. On hash queries $H(M)$, $\mathcal{B}$ outputs G^r for random r and stores (M, r). To answer $\mathsf{Enc}(M)$ queries, $\mathcal{B}$ programs H (or has already) so that $H(M) = G^r$, then $A^r = G^{a_0 r} = H(M)^{a_0}$. $\mathcal{B}$ outputs $(E_1, E_2) = (A^r H(M)^{a_1 h(m)}, (E_1)^a M)$.

When the DDH triple is real, games G_0 and G_1 are identical, and when the triple is random, pk is uniformly distributed in $\mathbb{G}$. The output of Enc queries is always the same in both games because it doesn't depend on C. Therefore, $\mathsf{G}_1 - \mathsf{G}_0 \leq \epsilon_{\text{ddh}}$.

Game 2 is the same as Game 1, except $\mathcal{B}$ replaces E_1 with a random value when $\mathcal{A}$ makes an Enc query. $\mathcal{B}$ chooses (a, a_1) at random, and acts as a wPRF attacker, for an instance where a_0 is the secret. When $\mathcal{A}$ queries $H(m)$, $\mathcal{B}$ queries the wPRF oracle to get (U, U'). $\mathcal{B}$ programs $H(m) = U$, then outputs $(E_1, E_2) = (U'U^{a_1 h(m)}, (E_1)^a m)$. Note that since m never repeats, U is a fresh random group element for every Enc query with overwhelming probability. When the wPRF oracle outputs real pairs then $\mathcal{B}$ outputs $E_1 = U'U^{a_1 h(m)} = U^{a_0 + a_1 h(m)} = H(m)^{a_0 + a_1 h(m)}$, and $\mathsf{G}_2 = \mathsf{G}_1$. When the wPRF output is random, then E_1 is uniformly distributed in $\mathbb{G}$. Therefore $\mathcal{B}$ is a distinguisher for the wPRF game (and hence DDH) with probability $\mathsf{G}_2 - \mathsf{G}_1 \leq \epsilon_{\text{ddh}}$.

Game 3 is the same as 2 but now E_2 is replaced with a random value. $\mathcal{B}$ does not use (a_0, a_1), and again plays the wPRF game, this time for an instance with secret a. When $\mathcal{A}$ makes an $\mathsf{Enc}(m)$ query, $\mathcal{B}$ queries the wPRF oracle to get (U, U'), and $\mathcal{B}$ outputs $(E_1, E_2) = (U, U'm)$. E_1 is uniformly distributed in both games 2 and 3. For E_2, when the wPRF output is real, we have $E_2 = (E_1)^a m$, exactly as in Game 2, and when the wPRF output is random, E_2 is uniformly distributed. Therefore $\mathsf{G}_3 - \mathsf{G}_2 \leq \epsilon_{\text{ddh}}$.

In Game 3 $\mathcal{B}$ no longer uses m. By a union bound

$$\Pr[\mathcal{A} \text{ wins the CPA game }] \leq 3\epsilon_{\text{ddh}}$$

□

Now we formally define the unique ciphertexts property.

Definition 7. *We say a symmetric-key encryption scheme* $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *has* unique ciphertexts *if for all polynomial-time* $\mathcal{A}$*,*

$$\Pr[(k, c_1, c_2) \leftarrow \mathcal{A}(1^\kappa) : c_1 \neq c_2 \ \wedge \ \mathsf{Dec}_k(c_1) = \mathsf{Dec}_k(c_2) \neq \perp]$$

is negligible in κ*.*

Next we define correctness under adversarially chosen keys. Our definition refers to the Derive hash function from our construction, used to derive the secret key from a seed.

Definition 8. *We say a symmetric-key encryption scheme* $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *is* correct under adversarially chosen keys *if for all polynomial-time* $\mathcal{A}$*,*

$$\Pr[(k_0, m) \leftarrow \mathcal{A}(1^\kappa) : sk = \mathsf{Derive}(k_0) \ \wedge \ \mathsf{Dec}_{sk}\mathsf{Enc}_{sk}(m) = \perp]$$

is negligible in κ *for all polynomial time* $\mathcal{A}$*.*

Now we prove that our encryption scheme has unique ciphertexts (Definition 7) and is correct under adversarially chosen keys(Definition 8).

Theorem 9. *The encryption scheme of Section 4.1 has unique ciphertexts, and is correct under adversarially chosen keys assuming* $\mathsf{HashTo}\mathbb{G}$ *is a random oracle,* $\mathsf{HashTo}\mathbb{Z}_q$ *is collision-resistant,* Derive *is a random function and the DLP is hard in* $\mathbb{G}$*.*

Proof. First we prove that the scheme has unique ciphertexts. Because decryption recomputes E_1' from M, and this step is deterministic, there is exactly one E_1 value for each M, such that decryption will succeed. For a pair of ciphertexts (E_1, E_2) and (E_1', E_2') that decrypt successfully to the same M, since $E_1 = E_1'$, and $E_2/(E_1)^a = E_2'/(E_1')^a$ it must that $E_2 = E_2'$.

Now for correctness under adversarially chosen keys. Suppose $\mathcal{A}$ has output k_0 such that $k = \mathsf{Derive}(k_0)$ and M such that $\mathsf{Dec}_k(\mathsf{Enc}_k(M)) = \perp$. By definition Enc_k succeeds for any k and M in range, therefore Enc outputs some (E_1, E_2). $\mathsf{Dec}_k(\cdot)$ can only fail if it recomputes some $E_1' \neq E_1$. Let M' be the value computed during the first step of Dec, i.e., $M' = E_2/(E_1)^a$. We prove that $E_1 \neq E_1'$ if and only if $M' \neq M$.

If $E_1 \neq E_1'$ then $M \neq M'$ since encryption is deterministic.

If $M \neq M'$ then $E_1 \neq E_1'$ since encryption is deterministic, and it's hard to find collisions in the function used to compute E_1. More specifically, it should be hard to find $M \neq M'$ such that

$$H(M)^{a_0+a_1h(M)} = H(M')^{a_0+a_1h(M')} .$$

Suppose $\mathcal{A}$ is an adversary that outputs such a pair, we construct $\mathcal{B}$ that solves the DLP instance $Y = G^x$ using $\mathcal{A}$, in the random oracle model. For each query $H(M)$ made by $\mathcal{A}$, $\mathcal{B}$ responds with Y^sG^t for random (s,t), and stores (M,s,t). When $\mathcal{A}$ outputs colliding $M \neq M'$, $\mathcal{B}$ has

$$(Y^sG^t)^{a_0+a_1h(M)} = (Y^{s'}G^{t'})^{a_0+a_1h(M')} \tag{1}$$

Let $v = a_0 + a_1h(M)$ and $v' = a_0 + a_1h(M')$. Solving equation 1 gives $Y^{sv-s'v'} = G^{t'v'-tv}$. Note that s, s' are statistically hidden from the adversary, so the probability that he can find M, M', a_0, a_1 such that $s'v' - sv = 0$ is negligible. That means $\mathcal{B}$ can output the discrete log as $\frac{t'v'-tv}{sv-s'v'}$. □

To relate the security properties of our new encryption to the more standard notion of CCA security, we first note that unique ciphertexts and plaintext integrity (PI, see [BS17, Excercise 9.15]) implies ciphertext integrity (CI, see [BS17, Definition 9.1]). In PI and CI, an adversary is given an encryption oracle, and must either create a new ciphertext that decrypts correctly (CI), or create a ciphertext that decrypts to a message that was never queried (PI).

To see that our scheme provides PI security, note that E_1 is a $\mathsf{MAC}_{\mathsf{GGM}}$ tag on the message $\mathsf{HashTo}\mathbb{Z}_q(M)$, with secret key (a_0, a_1). Then a PI attacker can be used to create a forgery on $\mathsf{MAC}_{\mathsf{GGM}}$. The reduction will choose a, and use it compute E_2, for E_1 it

will query a $\mathsf{MAC}_{\mathsf{GGM}}$ oracle to get (U, U') and program $\mathsf{HashTo}\mathbb{G}(M) = U$ and output $E_1 = U'$. When the PI attacker outputs E_1^* as part of a ciphertext on a never queried M^*, the reduction outputs $(M^*, H(M), E_1)$ as a forgery in the $\mathsf{MAC}_{\mathsf{GGM}}$ uf-cma-security game. This fails if there was a collision in $\mathsf{HashTo}\mathbb{Z}_q$, so we assume it is collision resistant.

Now we argue that PI and unique ciphertexts implies CI. When the CI attacker outputs a new ciphertext, PI security ensures that it is an encryption of a previously queried message. However, each previously queried message has only one ciphertext by the unique ciphertext security. Therefore, the attacker cannot output a valid ciphertext without breaking one of the two security properties.

Finally, if an encryption scheme is both CI and CPA secure, then it is CCA secure [BS17, §9.2.3].

6.2 Security of our New MAC

In this section we prove that our new MAC is secure.

Theorem 10. *The MAC defined in Section 3.1 is* suf-cmva *secure, assuming the DDH problem is hard in* $\mathbb{G}$*, and that the* $\mathsf{MAC}_{\mathsf{GGM}}$ *construction is* uf-cma *secure.*

Proof. Using Lemma 3, we can ignore verification queries and prove that the MAC is suf-cma secure.

We consider three possible types of forgeries, and show that each can occur with at most negligible probability. Recall that the forged MAC on message M^* consists of three values (t^*, U^*, V^*), and let M_i and (t_i, U_i, V_i) be the message used and MACs resulting from the adversary's MAC oracle queries. In Type 1 forgeries, $t^* \neq t_i$ for any i. In Type 2 forgeries, there exists a previous query i such that $t^* = t_i$, but $M^* \neq M_i$. (Note that since t is chosen freshly at random for each MAC produced by the oracle, there will be at most one such i.) Finally, in Type 3 forgeries, there exists a previous query i such that $t^* = t_i$ and $M^* = M_i$, but $U^* \neq U_i$. Note that since V^* is fully defined by M^*, U^*, t^*, this covers all possible forgeries in the suf-cma game. Let $\mathcal{A}$ be an attacker who plays the suf-cma security game.

Type 1 (t^* was not output by the MAC oracle) Suppose that there is an attacker $\mathcal{A}$ that can produce a Type 1 forgery in the suf-cma game with non-negligible probability. In this case, $\mathcal{A}$'s forgery uses a new tag value t^*, i.e., one that has not been output by in response to a MAC query from $\mathcal{A}$.

We construct an algorithm $\mathcal{B}$ that uses $\mathcal{A}$ as a subroutine. $\mathcal{B}$ will be a uf-cma forger for $\mathsf{MAC}_{\mathsf{GGM}}$. $\mathcal{B}$ plays the uf-cma security game for $\mathsf{MAC}_{\mathsf{GGM}}$ with a challenger, who provides a MAC oracle. First $\mathcal{B}$ generates part of the MAC key and the issuer parameters. $\mathcal{B}$ chooses $(w, y_1, \ldots, y_n)$ at random and computes C_W. Then $\mathcal{B}$ chooses a random z and queries $\mathsf{MAC}_{\mathsf{GGM}}(z)$ to get a MAC (U, U'). Since $U' = U^{x_0 + x_1 z} = U^{x_0} U^{x_1 z}$, when the random oracle used to generate parameters is programmed to output $G_{x_0} = U$ and $G_{x_1} = U^z$, $\mathcal{B}$

can create $I = G_V/({G_{y_1}}^{y_1} \ldots {G_{y_n}}^{y_n} {G_{x_0}}^{x_0} {G_{x_1}}^{x_1})$ as $I = G_V/({G_{y_1}}^{y_1} \ldots {G_{y_n}}^{y_n} U')$. Now $\mathcal{B}$ initializes $\mathcal{A}$ with the issuer parameters.

For MAC queries, $\mathcal{B}$ computes $\tilde{M} = \prod_{i=0}^{n} M_i^{y_i}$. $\mathcal{B}$ chooses a random t, queries $\mathsf{MAC}_{\mathsf{GGM}}(t)$ to get (U, U') and computes the MAC as $(t, U, V = \tilde{M}WU')$. Since $U' = U^{x_0 + x_1 t}$, this MAC is computed correctly.

When $\mathcal{A}$ outputs a forgery, $\mathcal{B}$ can compute $(U^*, V^*/\tilde{M}W)$, and output this as a $\mathsf{MAC}_{\mathsf{GGM}}$ forgery on the message t^*. If t^* was never output in a MAC created by $\mathcal{B}$ it was never queried to the $\mathsf{MAC}_{\mathsf{GGM}}$ oracle, and is therefore a valid $\mathsf{MAC}_{\mathsf{GGM}}$ forgery, if σ^* is a valid MAC and a Type 1 forgery.

Type 2 or Type 3 (t^* was output by the MAC oracle) Suppose that there is an attacker $\mathcal{A}$ that can produce a Type 2 or Type 3 forgery in the suf-cma game with non-negligible probabilty. We argue that this will allow us to construct a reduction $\mathcal{B}$ to break DDH. We proceed via a series of games:

Game 0 This is the real suf-cma game, with the modification that the adversary wins if the forgery is valid and is of Type 2 or Type 3. By assumption $\mathcal{A}$ produces a Type 2 or Type 3 forgery with non-negligible probability ϵ.

Game 1 This game proceeds as the suf-cma game with the following exception: the game first chooses a random $i^* \in 1 \ldots Q$, where Q is the the maximum number of queries that $\mathcal{A}$ can make, and the adversary wins only if $t^* = t_{i^*}$. The adversary will win this game with probability at least ϵ/Q.

Game 2 This game proceeds as in Game 1 with the following exceptions: First, the issuer parameters are chosen at random. Then, on the i^*th query that $\mathcal{A}$ makes to its MAC oracle, the game will respond by running the MAC algorithm. For all other MAC oracle queries, the game will return three random values (t, U, V) in the appropriate groups. The adversary wins if the MAC verifies and $t^* = t_{i^*}$ but either $\vec{M}$ or U is new. Suppose that the adversary's success probability in Game 2 is non-negligibly lower than in Game 1. In this case we build an algorithm $\mathcal{B}$ that breaks DDH.

Let (R, X_1, Z) be a DDH instance in $\mathbb{G}$, that $\mathcal{B}$ will use $\mathcal{A}$ to answer. We use the notation (G^r, G^{x_1}, G^{rx_1}) for a real DDH triple, and replace G^{rx_1} with G^z when Z is random. The base $G \in \mathbb{G}$ is assumed to be different from the parameters used by the MAC scheme.

$\mathcal{B}$ no longer chooses x_0, x_1 in the secret key. Instead $\mathcal{B}$ chooses random $d, t_{i^*} \in Z_q$. The value of x_1 is fixed by X_1 in the DDH instance, and the value x_0 used by $\mathcal{B}$ when creating MACs will be implicitly defined as $x_0 = d - x_1 t^*$.

To create I in the issuer parameters without (x_0, x_1), $\mathcal{B}$ first programs the random oracle so that G_{x_1} is derived as $R^a G^b$ for a random a, b. Then $\mathcal{B}$ computes the term ${G_{x_1}}^{x_1}$ as $Z^a X_1^b$. Similarly, for the term ${G_{x_0}}^{x_0}$, $\mathcal{B}$ programs the random oracle to derive $G_{x_0} = R^{a'} G^{b'}$

for a random a', b'. Then $\mathcal{B}$ computes ${G_{x_0}}^{x_0} = ((R^{a'}G^{b'})^d(Z^{a'}X_1^{b'})^{-t^*}$. Finally $\mathcal{B}$ chooses C_W at random; since this is a perfectly hiding commitment the distribution is identical to that in the real parameters.

For MAC query i^*, $\mathcal{B}$ chooses random a_{i^*}, b_{i^*} and outputs the MAC

$$(R^{a_{i^*}}G^{b_{i^*}}, \tilde{M}W(R^{a_{i^*}}G^{b_{i^*}})^d, t^*) \tag{2}$$

and for the other query $i \neq i^*$ $\mathcal{B}$ chooses random a_i, b_i, t and outputs

$$(R^{a_i}G^{b_i}, \tilde{M}W(R^{a_i}G^{b_i})^d(Z^{a_i}X_1^{b_i})^{t-t^*}, t) \tag{3}$$

Note that (2) is a special case of (3), since when $t = t^*$ part of (3) cancels out.

By the definition of d, the MAC in (2) is valid and distributed identically to the output of the MAC algorithm. When $Z = G^{rx_1}$, it can be checked that (3) is also valid and distributed identically to the output of the MAC algorithm, as are the issuer parameters. When Z is random, $Z^aX_1^b$ is random and independent of R^aG^b, so (3) consists of 3 independent random values. Similarly in this case I is random as well.

When A outputs a forgery (M^*, t^*, U^*, V^*), by assumption it will use a tag output by $\mathcal{B}$, and if the tag is not t^* output in query i^*, $\mathcal{B}$ aborts. If the tag is t^*, B computes $\tilde{M}$ from M^*, uses $\vec{y_i}$ and checks whether $V^*/(U^*)^d = \tilde{M}W$. If the comparison fails, the forgery is invalid, and $\mathcal{B}$ outputs "random" to the DDH instance, if it succeeds, $\mathcal{B}$ outputs "DDH tuple". So if $\mathcal{A}$'s forgery probability changes between games 0 and 1, $\mathcal{B}$'s DDH advantage changes by the same amount. Thus games 1 and 2 are indistinguishable assuming DDH is hard in $\mathbb{G}$.

Success probability in Game 2 Now we argue that $\mathcal{A}$'s forgery probability in Game 2 is negligible. First consider a Type 2 forgery. Note that a forgery on a new message $\vec{M}'$ must have $\tilde{M}' = \prod_{i=0}^{n} M_i'^{y_i}$. If $\vec{M}' = G^{\vec{m'}}$, the logarithm of $\tilde{M}'W$ to the base G is $y_1m_1' + \ldots + y_nm_n' + w \pmod q$. Note that this is a pairwise independent function of $m_1', \ldots, m_n'$. Since $\mathcal{A}$ has only received one value using $\vec{y}$ and w (in the response to the i^*th MAC query), the adversary can produce this value with probability at most $1/q$. Next, we consider a Type 3 forgery. Let $d = x_0 + x_1t^*$. Then the one MAC that $\mathcal{B}$ has output using the secret key has $V = \tilde{M}G^{w+ud}$ and the forgery has $V^* = \tilde{M}G^{w+u^*d}$, where u, u^* are the discrete logs of U and U^* from the i^*th query and the forgery respectively. Again, note that this is a pairwise independent function of u, and since $\mathcal{A}$ only has one MAC using w, d, the adversary has only negligible probability of producing the right value for u^*.

□

6.3 Credential Security

Referring to the definition in [CMZ14], a keyed-verification anonymous credential scheme has the following protocols: CredKeygen, BlindIssue, BlindObtain, Show, ShowVerify. The

key generation, (non-blind) issuance, and show protocols are described in Section 3, and the blind issuance protocol is described in Section 5.9

The following security properties are for formally defined in [CMZ14]. In this section we review them briefly and argue that a similar analysis applies here.

Correctness The first part of correctness (that credentials always verify) follows from correctness of the MAC. Correctness of the second part (that Show always succeeds for valid credentials), follows from the correctness of the zero-knowledge proof system, and the equation $Z = I^z$. Using our 4-attribute example where (M_1, M_2, M_3) are hidden and M_4 is revealed, when Z is computed honestly, we have

$$\begin{aligned} Z &= \frac{C_V}{{C_{y_1}}^{y_1}{C_{y_2}}^{y_2}{C_{y_3}}^{y_3}(M_4 C_{y_4})^{y_4} W {C_{x_0}}^{x_0}{C_{x_1}}^{x_1}} \\ &= \frac{V {G_V}^z}{(M_1 {G_{y_1}}^z)^{y_1} \cdots (M_4 {G_{y_4}}^z)^{y_4} W (U {G_{x_0}}^z)^{x_0} (U^t {G_{x_1}}^z)^{x_1}} \\ &= \frac{V {G_V}^z}{({M_1}^{y_1} \cdots {M_4}^{y_4} W U^{x_0 + t x_1}) {G_1}^{z y_1} \cdots {G_4}^{z y_4} {G_{x_1}}^{z x_1} {G_{x_0}}^{z x_0}} \\ &= \frac{{G_V}^z}{{G_{y_1}}^{z y_1} \cdots {G_{y_4}}^{z y_4} {G_{x_0}}^{z x_0} {G_{x_1}}^{z x_1}} = I^z \end{aligned}$$

and it can be checked similarly that this also holds with more than four attributes.

Unforgeability Intuitively, credential unforgeability means that an adversary cannot create a valid proof for a statement not satisfied by the credentials they have been issued. This follows from the unforgeability of the MAC (proven in Section 6.2), and the extractability of the proof system. If the adversary outputs a proof based on a MAC with attributes that were not output by Issue, then we can extract a forgery for the MAC scheme.

For example, referring the to the proof of knowledge used for authentication in Section 5.11, note that from a successful prover we can extract (z, m_3, sk), then use these to compute (t, U, V), which is a valid MAC on the attributes (M_1, M_2, m_3, m_4) since it satisfies the verification equation (assured by the proof statement $Z = I^z$). If the MAC was created by the issuer, authentication should succeed. If not, and the MAC is new, it is a forgery and the MAC scheme is broken.

Anonymity This requires that the proofs output when presenting a credential reveal only the statement being proven. Below we sketch a proof that the authentication proof is zero-knowledge, and this proof includes the statements common to any credential presentation.

To show that the proof of Section 5.11 is zero-knowledge, we first need to show that the commitments are hiding (which is nontrivial since they all share the same random value z). Note that in the random oracle model, the bases G_{y_i} are a random set, that are then

input to the wPRF f_z, so ${G_{y_i}}^z$ is a PRF output under the DDH assumption. Therefore, the commitments $(C_{y_1}, \ldots, C_{y_4}, C_{x_0}, C_{x_1}, C_V)$ are hiding, and they can be simulated with random group elements. Similarly, since ${C_{y_2}}'$ is a PRF output assuming DDH (since C_{y_2} is input to the wPRF f_{a_1}), it can be simulated with a random group element. Since the ciphertext (E_{A_1}, E_{A_2}) is CPA secure, it can also be simulated with random values, and since the proof π_A is zero-knowledge, a simulator exists to simulate it.

Blind Issuance This property requires that blind issuance be a secure two-party protocol, between the user, who has the blind attributes as private input, and the issuer, who has the issuer secret key as private input. Our blind issuance protocol based on homomorphic Elgamal encryption is unchanged from [CMZ14], and security follows from CPA security of Elgamal (implied by DDH) and the privacy and extractability of the zero-knowledge proof system. Note that non-blind issuance is the special case where no attributes are hidden.

Key-parameter consistency This property ensures that an issuer cannot use different secret keys with different users, in order to link an instance of BlindIssue with an instance of Show.

We consider two cases, starting with the key consistency of (w, w'). From an issuer that creates two proofs π_I with different (w, w'), we can extract two openings to the Pedersen commitment $C_W = {G_W}^w {G_{W'}}^{w'}$, breaking the binding property. Given such a malicious issuer, we can construct an algorithm for the DLP in $\mathbb{G}$. Given a DLP instance $Y = G^x$, set $G_W = Y$ and $G_{W'} = G^{r_1}$. Then given two distinct openings of the commitment C_W, and knowledge of r_1, we can solve for x.

Now consider the secrets used in the I value of *iparams*. Similarly, the product ${G_{y_1}}^{y_1} \ldots {G_{y_n}}^{y_n} {G_{x_0}}^{x_0} {G_{x_1}}^{x_1}$ is a binding commitment under the DLP assumption in $\mathbb{G}$, and by the same argument no malicious issuer can prove knowledge of distinct openings if the DLP is hard in $\mathbb{G}$.

7 Acknowledgements

We extend our thanks to Dan Boneh, Isis Lovecruft, Henry de Valence, Jan Camenisch, and the Signal team for helpful discussions.

References

[ABS16] Miguel Ambrona, Gilles Barthe, and Benedikt Schmidt. Automated unbounded analysis of cryptographic constructions in the generic group model. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 822–851. Springer, Heidelberg, May 2016.

[BDJR97] Mihir Bellare, Anand Desai, Eric Jokipii, and Phillip Rogaway. A concrete security treatment of symmetric encryption. In *38th FOCS*, pages 394–403. IEEE Computer Society Press, October 1997.

[Ber05] Daniel J. Bernstein. The poly1305-AES message-authentication code. In Henri Gilbert and Helena Handschuh, editors, *FSE 2005*, volume 3557 of *LNCS*, pages 32–49. Springer, Heidelberg, February 2005.

[BL13] Foteini Baldimtsi and Anna Lysyanskaya. Anonymous credentials light. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 1087–1098. ACM Press, November 2013.

[BS17] Dan Boneh and Victor Shoup. A graduate course in applied cryptography, 2017. Available online `https://crypto.stanford.edu/~dabo/cryptobook/`.

[CD00] Jan Camenisch and Ivan Damgård. Verifiable encryption, group encryption, and their applications to separable group signatures and signature sharing schemes. In Tatsuaki Okamoto, editor, *ASIACRYPT 2000*, volume 1976 of *LNCS*, pages 331–345. Springer, Heidelberg, December 2000.

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

[CHK+11] Jan Camenisch, Kristiyan Haralambiev, Markulf Kohlweiss, Jorn Lapon, and Vincent Naessens. Structure preserving CCA secure encryption and applications. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 89–106. Springer, Heidelberg, December 2011.

[CL03] Jan Camenisch and Anna Lysyanskaya. A signature scheme with efficient protocols. In Stelvio Cimato, Clemente Galdi, and Giuseppe Persiano, editors, *SCN 02*, volume 2576 of *LNCS*, pages 268–289. Springer, Heidelberg, September 2003.

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

[CMZ14] Melissa Chase, Sarah Meiklejohn, and Greg Zaverucha. Algebraic MACs and keyed-verification anonymous credentials. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 2014*, pages 1205–1216. ACM Press, November 2014.

[CS97] J. Camenisch and M. Stadler. Proof systems for general statements about discrete logarithms. Technical Report TR 260, Institute for Theoretical Computer Science, ETH Zürich, 1997.

[CS03] Jan Camenisch and Victor Shoup. Practical verifiable encryption and decryption of discrete logarithms. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 126–144. Springer, Heidelberg, August 2003.

[CV02] Jan Camenisch and Els Van Herreweghen. Design and implementation of the idemix anonymous credential system. In Vijayalakshmi Atluri, editor, *ACM CCS 2002*, pages 21–30. ACM Press, November 2002.

[DKPW12] Yevgeniy Dodis, Eike Kiltz, Krzysztof Pietrzak, and Daniel Wichs. Message authentication, revisited. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 355–374. Springer, Heidelberg, April 2012.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Heidelberg, August 1987.

[KBC97] Hugo Krawczyk, Mihir Bellare, and Ran Canetti. HMAC: Keyed-hashing for message authentication. IETF Internet Request for Comments 2104, February 1997.

[Lun17] Joshua Lund. Encrypted profiles for Signal now in public beta, September 2017. `https://signal.org/blog/signal-profiles-beta/`.

[Mar14] Moxie Marlinspike. Private group messaging, May 2014. `https://signal.org/blog/private-groups/`.

[NR95] Moni Naor and Omer Reingold. Synthesizers and their application to the parallel construction of pseudo-random functions. In *36th FOCS*, pages 170–181. IEEE Computer Society Press, October 1995.

[PZ13] C. Paquin and G. Zaverucha. U-prove cryptographic specification v1.1 (revision 2), 2013. Available online: `www.microsoft.com/uprove`.

[RCE15] Kai Rannenberg, Jan Camenisch, and Ahmad Sabouri (Editors). *Attribute-based credentials for trust, identity in the information society*. Springer, 2015. `https://doi.org/10.1007/978-3-319-14439-9`.

[RMS17] Paul Rösler, Christian Mainka, and Jörg Schwenk. More is less: How group chats weaken the security of instant messengers signal, WhatsApp, and threema. Cryptology ePrint Archive, Report 2017/713, 2017. `http://eprint.iacr.org/2017/713`.

[Sig19] Signal. Technical information (specifications and software libraries), 2019. `https://www.signal.org/docs/`.