Fully Distributed Group Signatures

Rosario Gennaro[1], Steven Goldfeder[2], and Bertrand Ithurburn[1,3]

[1] The City University of New York
[2] Cornell University
[3] ORBS

Abstract. A group signature scheme is a powerful primitive by which one can sign a message as a member of a well-defined group without leaking one's individual identity, but the group manager can revoke this anonymity and reveal the identity of the signer. All existing group signature schemes incorporate a centralized manager that is a single point of trust and failure. The group *issuing manager* is charged with issuing credentials, and can maliciously issue bogus credentials that can be used to sign but cannot be traced to any player. The group *tracing manager* can choose at-will to revoke the anonymity of any group member.
We present the first group signature schemes that are completely decentralized and have no single points of trust. To accomplish this, we design a (t, n) threshold scheme that distributes control of the issuing manager and the tracing manager. This removes the need to trust a single entity to perform the roles of adding, revoking and tracing group members, as these actions can only be done by a threshold quorum.
We present two highly efficient fully distributed group signature schemes in this paper. Our first scheme is a distributed version of the BBS group signature scheme (CRYPTO 2004), and our second scheme is a distributed variant of the Camenisch-Lysyanskaya anonymous credentialing scheme (CRYPTO 2004). Our schemes inherit all its efficiency features of the original centralized group signature schemes; in particular the signatures are concretely very short.

1 Introduction

There's often a tension between the need to authenticate oneself and the desire to retain privacy. Consider the case of trusted platform modules (TPMs), for example, where this conundrum manifests itself quite clearly. On the one hand, a trusted enclave needs to sign its attestations to prove that they're authentic. On the other hand, if each TPM were given a unique signing key, this would allow one to trace the activities of a specific enclave as all of its actions would be linked to that key.

Group signatures are an effective tool to address this scenario where authentication is necessary, but fine-grained authentication would present a privacy issue. Introduced by Chaum and Van Heyst in 1991 [20], a group signature scheme allows a form of anonymous authentication where the anonymity set is a specified group of users. Group members can sign messages, and anyone can verify that

the message was indeed signed by a member of the group. However, the identity of the individual group member who issued the signature remains private.

Without further constraints, constructing a group signature scheme that meets this requirement is trivial: take any secure signature scheme $\mathcal{S}$, distribute a single private key sk to all group members, and publish the corresponding public key pk. Using the shared private key, any group members can sign messages on behalf of the group. Non-members cannot sign but can verify the group's signatures using the group's public key, pk.

The complexity of group signature schemes stems from additional properties that are often desired, properties which the trivial solution does not provide. Two such properties are *traceability* and *revocability*. Traceability refers to the ability to selectively *trace* or de-anonymize a signature to reveal which individual member signed. Revocability is the ability to *revoke* a group member's credentials, thereby removing that individual from the group.

These properties are generally described in the context of **trusted managers**: in a traceable group signature scheme, a *tracing manager* can de-anonymize a given signature and unmask the individual signer who issued it. Similarly an *issuing manager* is responsible for maintaining membership in the group: the manager issues and revokes credentials to users, allowing or preventing them to sign on behalf of the group, without affecting other members.

Group signatures are useful in a variety of contexts. They have served as the basis for electronic voting schemes [4], auction protocols [50], early electronic cash protocols [41, 45], and, as previously mentioned, are a key primitive used in trusted platform modules such as Intel SGX [13, 14]. In these systems, TPMs are issued attestation keys with which they sign statements attesting that the execution was done correctly in the secure environment, and the ability to revoke credentials is crucial.

1.1 Eliminating trust

Group signatures rely on trusted managers, but for many applications, relying on a trusted manager is a considerable drawback. The issuing manager has complete centralized control of the group's membership. A corrupted issuing manager can issue credentials at will, and in particular can issue bogus credentials that will not be traceable to any known identity. Conversely, the manager can revoke a user at will, with or without legitimate reasons for doing so. Moreover, the anonymity properties are severely weakened by the fact that the tracing manager can unmask any signature at its discretion. In the event that the tracing manager is corrupted, anonymity is completely lost.

In this paper, we answer the following question in the affirmative: Can we build a group signature scheme that is both traceable and revocable but does not rely on any centralized trust?

Instead of having a single trusted manager, we replace the manager's functionality with a distributed (t, n) threshold protocol that is run by a set of servers. This set could be the group itself, a subset or superset of the group,

or an entirely independent set that is separate from the group members. Moreover, the tracing manager and issuing manager can themselves be independent threshold sets and need not be implemented by the same group.

Assume the issuing manager and tracing manager are distributed among a group of n nodes. In order to perform one of the manager's services (e.g. accept a new user, revoke a user's credentials, or trace a signature), $t+1$ members must participate. Any subset of $t+1$ or more players will be able to perform the service, whereas a group of t or fewer members are unable to do so. Compromising a single node will not put the scheme at risk, and malicious actions by the manager will require the collusion of $t+1$ nodes.

On a technical level, the problem of distributing the tracing manager is far simpler than distributing the issuing manager. While there have been previous works that have successfully distributed the tracing manager [9], there are no traceable group signature schemes with both distributed issuing and tracing, and this turns out to be a considerably difficult technical challenge. We propose the first group signature schemes that realize this goal of being fully distributed.

In this paper, we are exclusively interested in *succinct* group signatures in which the private key, public key, and signature are all of constant size. In particular, we do not consider solutions in which the public key and/or signature size are functions of the size of the group (See Section 1.2 for alternative proposals that do not fulfill this requirement.)

1.2 Related Work

Chaum and Van Heyst introduced the notion of group signatures in [20]. Following their initial work, many other protocols were presented. For the decade following their initial paper, the most efficient group signature schemes relied on the Strong RSA assumption (see Appendix A.3) for their security [1–3, 15, 54].

The initial proposals for group signature schemes often proposed their own set of definitions, both for desired properties as well as for what it meant to be secure. These definitions were often overlapping and inconsistent with one another. In a 2003 paper, Bellare *et al.* formalized the properties and security definitions for group signature schemes. They also proposed their own scheme, which was based on trapdoor permutations and proven secure in the standard model, but it was not practical [8]. See Appendix B where we recall these formal definitions of group signature schemes.

Kiayias *et al.* introduced the notion of *traceable signatures* [37], a related primitive in which one simultaneously can revoke the privacy of all signatures originating from a given user as well as selectively and independently trace their own signature. Their construction is also based on the Strong RSA assumption.

In concurrent work, Boneh, Boyen, and Shacham (BBS) [10] and Camenisch and Lysyanskaya (CL) [16] present highly efficient group signature schemes based on billinear maps, both proven secure in the random oracle model using security definitions based on the ones given in [8]. While the two schemes are different, they both were a significant advancement as the signatures in these schemes are significantly shorter than those based on the Strong RSA assumption. Indeed,

these two schemes serve as the respective starting points for the two schemes we present in this paper.

While several previous attempts have been made to eliminate the trusted center in group signature schemes, no prior scheme has succeeded in building a fully decentralized scheme. Blomer *et al.* modify the BBS scheme by replacing the tracing manager with a (t, n) threshold protocol [9]. However, they do not address the more difficult technical challenge of distributing the issuing manager, which is the core contribution of this paper.

Noting the drawback of having trusted group managers, Manulis introduced a variant of group signatures known as *democratic group signatures* [46]. In a democratic group signature scheme, **every** member of the group can single-handedly trace signatures. Put another way, there is no anonymity within the group, and anonymity of signatures is provided only with respect to non-members. Moreover, issuing a new key requires the participation of all n group members, and the public key is changed in the process. Similarly, removing a member from the group requires the participation of all remaining group members. Zheng *at al.* extend Manulis's construction to support threshold traceability [55].

We note that democratic group signatures have a very weak notion of anonymity (as any group member alone can trace a signature) and the schemes in [46, 55] are not succinct. In particular, the size of both the public key and the signature is linear in the number of members of the group.

We note that a democratic group signature scheme can be viewed as a special case of a (t, n) threshold group scheme where the tracing manager is distributed among the group members such that $t = 1$ and the issuing manager is distributed such that $t = n$. With this generalization in mind, we point out that using our generalized (t, n) group signature schemes we can instantiate highly efficient democratic signature schemes as well – i.e. succinct democratic signatures with short signatures.

Recently Sonnino *et al.* presented an anonymous credential scheme with a distributed issuance protocol [53]. Their scheme allows a user to aggregate and selectively disclose anonymous credentials. Unlike the group signature schemes considered in this paper, however, their scheme does not support traceability or revocability.

Also recently, Boneh *et al.* proposed a post-quantum secure group signature scheme that is constructed entirely from symmetric key primitives [11] based on the ZKB++ proof system and the Picnic signature scheme [19, 18]. This scheme however also relies on a centralized group manager, and it is an open problem of how to build a fully-distributed post-quantum group signature scheme.

1.3 Our contribution

In Section 4.1, we build a fully distributed group signature scheme based on BBS. In Section 5, we build a fully distributed version of the CL group signature scheme.

Anonymous Credentials Our techniques can also be used to build anonymous credential systems with distributed issuing authorities. In fact our techniques for

the BBS signature scheme extend to the BBS+ [5] anonymous credential system. Similarly the techniques for the CL group signature scheme can also be used for the CL anonymous credential system described in the same paper [16]. As such our new schemes are alternatives to the CoCoNut scheme in [53]. Details of the anonymous credential schemes will appear in the final paper.

1.4 Our Approach

As we mentioned earlier in the Introduction, Blomer *et al.* modify the BBS scheme by replacing the tracing manager with a (t, n) threshold protocol [9]. We focus on distributing the issuing manager.

First we point out that the public/secret key pair of the issuing manager in both BBS and CL schemes can be distributively generated by the player without the help of a trusted dealer. This can be achieved via a protocol which is based on [33]. At the end of this protocol each server holds a share w_i of the secret key w such that $w_i = f(i) \bmod q$ where f is a polynomial of degree t and $f(0) = w$.

The technical complication with sharing the issuing of credentials now comes from having to jointly compute a value of the form $g^{\frac{1}{\ell(w)}}$ (where ℓ is an affine function).

This can be done using a classic technique due to Bar-Ilan and Beaver [6] which requires one multiplication between $z = \ell(w)$ and a randomly shared element ρ. If shares are held in polynomial fashion this would require doubling the degree of the polynomial and at least $2t+1$ servers to cooperate to sign (this approach was followed in [32] when a similar problem arises in the computation of threshold DSA signatures).

We are interested in a solution that only requires $t + 1$ servers – the minimal number – to cooperate to issue credentials. Here we use a technique that originates with Gilboa [35] but has been used several time since (e.g. [30, 36, 40]). The idea is to write the product $z\rho$ as the sum of products of secret values held by the servers. For example if $\ell(w) = z = \sum z_i$ and $\rho = \sum_j \rho_j$, then $z\rho = \sum_{ij} z_i\rho_j$, and then each pair of players P_i, P_j engages in a *share conversion* protocol which on input z_i, ρ_j respectively returns values a_{ij} to P_i and b_{ij} to P_j such that $a_{ij} + b_{ij} = z_i\rho_j$. This in turn would allow the players to hold additive shares of $z\rho$ and eventually compute $g^{\frac{1}{\ell(w)}}$.

Our protocols are proven secure against a malicious adversary who controls up to t players. Since we only assume that $n \geq t+1$ (i.e. dishonest majority) we can only guarantee that the protocol either terminates with the correct solution or aborts.

Security for the distributed BBS scheme is proven via a full simulation argument, where the view of the adversary can be efficiently simulated without knowledge of the input of the honest players. To that extent we present two protocols, both fully simulatable, which present a tradeoff between efficiency and security assumptions.

The security of the distributed CL scheme is proven in a weaker game-based definition where we prove that an adversary controlling only t players cannot issue new credentials.

2 Preliminaries

2.1 Communication model

We assume a group of n members which are connected via a complete network of point-to-point channels, as well as a broadcast channel. We assume that the network is *partially synchronous*.

2.2 Adversarial model

We prove security against a adversary, who may corrupt up to t players. We assume a possible *dishonest majority*, and thus the only restriction on t is $t \leq n$. We assume *static corruptions*, meaning the adversary must choose at the beginning of the protocol which players to corrupt.

The *view of the adversary* is comprised of all private information and state held by the corrupted players as well as the public view of the protocol. We assume a *malicious* adversary who may arbitrarily deviate from the protocol. We also assume a *rushing adversary*, meaning that the honest players speak last and, in particular, may choose their response adaptively after seeing the adversaries messages.

2.3 Group signatures

We refer the reader to [8] for the formal definition of group signatures. We point out that in [10] the BBS scheme described below is proven secure according to the definitions in [8].

In this paper, we don't engage directly with these security definition as we use a simulation argument to prove security based on the assumed security of the BBS scheme (which is itself proven secure using the definitions of [8]). We prove that our threshold credential issuing protocol is simulatable and therefore inherits all the security properties of the original BBS scheme. Moreover, for threshold traceability, we rely on the simulatable threshold decryption scheme in [9] for BBS (see Section 4.5) and on the Canetti Goldwasser cryptosystem for CL (see Section 5.4).

For completeness, we include the formal requirements for a group signature schemes as well as the formal security definitions from [8] in Appendix B.

2.4 Threshold Secret sharing

Threshold secret sharing is a method of distributing a secret to n participants (commonly called *players*) with a specified threshold t such that $t + 1$ players can jointly reconstruct the secret, whereas t or fewer players cannot. The most prominent threshold secret sharing scheme is due to Shamir [52]. In Shamir's scheme, the secret shares are points on a degree t polynomial:

$$p(x) = \sigma + a_1 x + a_2 x^2 + \cdots + a_t x^t \bmod q$$

Let each player P_i be associated with a unique public index z_i (commonly $z_i = i$). Then P_i's share of the secret σ is $p(z_i)$.

Any group of $t + 1$ players can reconstruct the secret via Lagrange interpolation.

2.5 Verifiable Secret Sharing

A *verifiable secret sharing (VSS)* scheme has the additional property that players can independently verify the validity of their share, meaning that ant $t+1$ players will reconstruct the same unique secret. Again, let the secret be defined by the polynomial

$$p(x) = \sigma + a_1 x + a_2 x^2 + \cdots + a_t x^t \bmod q$$

and assume that a dealer D is distributing shares of this secret. In Feldman's scheme, D distributes $v_i = g^{a_i}$ in G for each $i \in [1, t]$ as well as $v_0 = g^{\sigma}$, where G denotes a cyclic group and g denotes an element of G. Player P_i uses these values to check the validity of its share σ_i by calculating:

$$g^{\sigma_i} \stackrel{?}{=} \prod_{j=0}^{t} v_j^{{z_i}^j} \text{ in } G$$

If any player finds that the above equation does not hold, then that player complains and the protocol terminates. The original Feldman system assumes an honest majority and in the case where a (minority of) player(s) raised a false complaint, it allows for recovery. However, in this paper we are assuming a dishonest majority and recovery therefore may not be possible. Instead the protocol aborts if any player raises a complaint.

Feldman's scheme is not information theoretically secure, and indeed it clearly leaks g^{σ}. However, it can be shown via simulation argument that it does not leak any other information. We omit the simulation details here, but briefly it involves Lagrange interpolation in the exponent by which given just g^{σ} and the shares of t corrupted player, one can recover all other published information in the protocol.

2.6 Commitment Schemes

A trapdoor commitment scheme allows a sender to commit to a message without revealing any information about the committed message. In other words, a receiver who possesses a transcript of the commitment exchange cannot guess the committed message any better than random, even if the receiver has infinite computational power. However, the scheme's trapdoor makes it possible to open the commitment. A proper implementation will have a hard-to-compute trapdoor, but if a user has knowledge of the trapdoor, then that user can open the commitment. We call this type of scheme *equivocable*.

A non-interactive trapdoor commitment scheme typically has four algorithms, described below:

- KG denotes the key generation algorithm. It receives the security parameter as input and outputs a pair pk, tk where pk is the public key associated with the commitment scheme and tk is the trapdoor.
- Com denotes the commitment algorithm. It receives input pk and message M and outputs $[C(M), D(M)] = Com(\mathsf{pk}, M, R)$, where R denotes coin tosses. $C(M)$ refers to the commitment string, and $D(M)$ refers to the decommitment string. The latter remains secret until opening takes place.
- Ver denotes the verification algorithm. It receives as input C, D, and pk and outputs a message M if verification succeeds or $\perp$ if it fails.
- Equiv denotes the equivocation algorithm that opens the commitment to any selected message if given the trapdoor. It takes as input pk, M, R with $[C(M), D(M)] = \mathsf{Com}(\mathsf{pk}, \mathsf{M}, \mathsf{R})$, a message $M' \neq M$, and a string T. If $T = \mathsf{tk}$, then the algorithm outputs D' such that $\mathsf{Ver}(\mathsf{pk}, C(M), D') = M'$.

If the sender refuses to open a commitment, we can set $D = \perp$ and $\mathsf{Ver}(\mathsf{pk}, C, \perp) = \perp$. A trapdoor commitment scheme has the following properties:

Correctness If $[C(M), D(M)] = \mathsf{Com}(\mathsf{pk}, \mathsf{M}, \mathsf{R})$, then $\mathsf{Ver}(\mathsf{pk}, C(M), D(M)) = M$.

Information Theoretic Security For every message pair M, M', the distributions $C(M)$ and $C(M')$ are statistically close.

Secure Binding Adversary A wins if it produces C, D, and D' such that $\mathsf{Ver}(\mathsf{pk}, C, D) = M$, $\mathsf{Ver}(\mathsf{pk}, C, D') = M'$, and $M \neq M'$. To achieve secure binding, any efficient A should have a negligible probability of winning with respect to the security parameter.

Given a commitment C to messages m, if no adversary A can produce another commitment C' after seeing the opening of C to m and successfully decommit to a related message m', then the commitment achieves *non-malleability* [24, 25]. For the purposes of this paper, the non-malleable commitment scheme must have concurrent security, meaning that security must hold when the adversary has seen multiple commitments. Multiple schemes offer this level of security [22, 29, 44]. Alternatively, an implementation can use a secure hash function H to define the commitment to x as $h = H(x, r)$ for a uniformly-chosen r of length λ. H acts as random oracle in this version.

2.7 Paillier's Encryption Scheme

An additively homomorphic encryption scheme $\mathcal{E}$ is an encryption scheme that has the additional property that it is homomorphic modulo an integer N. Let $E_{pk}(\cdot)$ denote the case the encryption algorithm using public key pk. Additive homomorphism means that given two ciphertexts $c_1 = E_{pk}(a)$ and $c_2 = E_{pk}(b)$, there is an efficiently computable operation $\oplus$ such that

$$c_1 \oplus c_2 = E_{pk}(a + b \bmod N)$$

Repeating this addition operation naturally gives rise to a corresponding scalar multiplication operation $\otimes$, by which a ciphertext can be multiplied by a

scalar. Given an integer $s \in Z_N$ and a ciphertext $c = E_{pk}(a)$, then

$$c \otimes s = E_{pk}(as \bmod N)$$

Informally, an additively homomorphic encryption scheme $\mathcal{E}$ achieves semantic security if an adversary cannot distinguish between the probability distributions of the encryptions of any two messages. In this paper, we rely on an additively homomorphic encryption scheme. We instantiate our protocols with Paillier's scheme [47], although in principle any additively homomorphic encryption scheme will work.

We now provide the details of the Paillier encryption scheme:

- Key-Gen:
  - choose two large primes P, Q of equal length
  - calculate $N = PQ$
  - calculate $\lambda(N) = lcm(P-1, Q-1)$
  - choose $\Gamma \in Z^*_{N^2}$ such that it has an order that is a multiple of N
  - output public key (N, Γ) and secret key $\lambda(N)$
- Encryption on input message $m \in Z_N$:
  - choose $x \in_R Z^*_N$
  - output $c = \Gamma^m x^N \bmod N^2$
- Decryption on input $c \in Z_{N^2}$:
  - let L denote a function defined over the set $\{u \in Z_{N^2} : u = 1 \bmod N\}$ where $L(u) = (u-1)/N$
  - output $L(c^{\lambda(N)})/L(\Gamma^{\lambda(N)}) \bmod N$
- Homomorphic Properties: Given two ciphertexts $c_1, c_2 \in Z_{N^2}$ such that $c_i = E(m_i)$, define $c_1 \oplus c_2 = c_1 c_2 \bmod N^2$ and $c_1 \oplus c_2 = E(m_1 + m_2 \bmod N)$. Likewise, given a ciphertext $c = E(m) \in Z_{N^2}$ and a number $a \in Z_n$, define $a \otimes c = c^a \bmod N^2 = E(am \bmod N)$.

Paillier relies on the N-decisional composite residuosity assumption (DCRA) to prove its security [47]. Briefly, this assumption says that a distinguisher cannot distinguish between random N-residues and random group elements in $Z^*_{N^2}$.

3 Multiplicative-to-additive share conversion protocol

Assume that we have two parties Alice and Bob with associated public keys $A = g^a$ and $B = g^b$ and respective secret keys $a, b \in Z_q$ (where g is the generator of a group G of prime order q). Consider the Diffie-Hellman transform of the two public keys $X = g^x$ with $x = ab \bmod q$. We can think of a, b as multiplicative shares of $x = ab \bmod q$.

Alice and Bob would like to compute secret additive shares α, β of x, that is random values such that $\alpha + \beta = x = ab \bmod q$ with Alice holding α and Bob holding β. Moreover Alice's and Bob's output should also include the values $\hat{A} = g^\alpha$ and $\hat{B} = g^\beta$.

Here we show a protocol based on an additively homomorphic scheme which has appeared many times before in the literature (e.g. [23, 30, 35, 36, 38, 40, 43] but that we adapt to our needs.

In our protocol we enforce that Alice and Bob use the correct values via a ZK proof. Moreover we assume that Alice and Bob have "registered" their public keys by at some point proving possession of the associated secret key via a zero-knowledge proof of knowledge.

We assume that Alice is associated with a public key E_A for an additively homomorphic scheme $\mathcal{E}$ over an integer N. Slightly abusing notation we denote with $c = E_A(m)$ the encryption of m (with random uniform coins) and with $D_A(c)$ the value that results by decrypting c. Let $K > q$ also be a bound which will be specified later.

In the following we will refer to this protocol as MtAwc (for *Multiplicative to Additive with checks*) share conversion protocol.

1. Alice initiates the protocol by
   - sending $c_A = E_A(a)$ to Bob
   - proving in ZK that $D_A(c_A) < K$ and $g^{D_A(c_A)} = A$
2. Bob computes the ciphertext $c_B = b \otimes c_A \oplus E_A(\beta') = E_A(ab + \beta')$ where β' is chosen uniformly at random in Z_N. Bob sets his share to $\beta = -\beta' \bmod q$. He responds to Alice by
   - sending c_B and $\hat{B} = g^\beta$
   - proving in ZK that he knows b, β such that $B = g^b$, $b < K$, $\hat{B} = g^\beta$, $c_B = b \otimes c_A \oplus E_A(\beta')$, and $\beta = -\beta' \bmod q$
3. Alice decrypts c_B to obtain $\alpha' = D_A(c_B)$. She sets her share to $\alpha = \alpha' \bmod q$ and reveals $\hat{A} = g^\alpha$.
4. Both parties check that $\hat{A} \cdot \hat{B} = g^{ab}$ which Alice (resp. Bob) can compute as B^a (resp. A^b) and abort if that does not hold.

Correctness. Assume $N > K^2 q$ and that neither party aborts during the protocol. Then note that Alice decrypts the value $\alpha' = ab + \beta' \bmod N$. Note that if $\beta' < N - ab$ the reduction mod N is not executed. Conditioned on this event, the protocol correctly computes α, β such that $\alpha + \beta = x \bmod q$.

Since $ab \leq K^2$ and $N > K^2 q$ we have that $\beta' \geq N - ab$ with probability at most $1/q$ (i.e. negligible).

Simulation of Bob. If the adversary corrupts Alice, then Bob's message can be simulated without knowledge of its input b. We sketch here how a simulator for Bob will work. First of all we assume that the simulator knows a since it can extract it from the proof of knowledge during the registration phase. Then the simulator can set $c_B = E(\alpha')$ for $\alpha' \in_R Z_N$ and set $\hat{B} = B^a \cdot g^{-\alpha'}$ (and simulate the ZK proof of correctness w.r.t. B). The distribution of the message decrypted by Alice in this simulation is close to that of the message decrypted when Bob uses the real b, because

- the value a sent encrypted by Alice is smaller than K due to the ZK proof
- the "noise" β' is uniformly distributed in $Z_N > K^2 q$

so the distribution of α' is statistically close to the uniform one over Z_N in both the real and simulated executions.

Simulation of Alice. If the adversary corrupts Bob, then Alice's message can be simulated without knowledge of its input a. Indeed a simulator can just choose a random $a' \in Z_q$ and act as Alice (simulating the ZK proof of correctness w.r.t. A). Moreover the simulator can set $\hat{A} = A^b \hat{B}^{-1}$ (as before we assume that the simulator knows b since it can extract it from the proof of knowledge during the registration phase). In this case the view of Bob is computationally indistinguishable from the real one due to the semantic security of the encryption scheme $\mathcal{E}$. Note that the simulation of Alice does not require knowledge of the secret key of her encryption scheme (otherwise the reduction above to the semantic security of $\mathcal{E}$ could not be carried out).

Remark. *On the ZK proofs and the size of the modulus N.* For the ZK proofs required in the protocol we use simplified versions of similar ZK proofs presented in [43] and already used in [31]). These protocols are presented in the Appendix.

These are ZK arguments with security holding under the Strong RSA Assumption. Moreover they require $K \sim q^3$ which in turn require $N > q^7$ which in practice is not an issue.

3.1 Removing the ZK Proofs

In the above MtAwc protocol, the ZK proofs are the most expensive step of the protocol because they require the prover to prove a "complicated" statement about secrets which operate mod$N$ when encrypted while the result has to be valid mod q.

We now present a protocol which removes the expensive ZK proofs and replaces them with simple proofs of knowledge of certain discrete logarithms (these proofs of knowledge are not required for the security of the "standalone" protocol when run by itself, but will be useful to prove the security of our threshold issuing protocol). These proofs can be implemented much more efficiently using a variant of Schnorr's proof [51]. Throughout, when we refer to Schnorr-style proofs, we require our proofs to be concurrently extractable so we instantiate Schnorr's proof with Fischlin's [27] transformation in place of Fiat-Shamir.

This gain in efficiency is offset by requiring additional computational assumption to prove the security of the protocol, as we discuss below.

In the following we will refer to this protocol as MtA (for *Multiplicative to Additive*) share conversion protocol.

Registering the Public Keys. In the MtAwc protocol we made the assumption that Alice and Bob proved knowledge of a, b respectively during a prior registration phase. For the security of MtA we need to also require that during this phase, Alice proves knowledge of the secret key D_A of her encryption key E_A. This would allow us to assume that the simulator for Bob knows D_A since it could have extracted it during a simulation of the registration phase. Jumping ahead, this will happen in the distributed key generation phase of our protocol.

Input Distribution. We also assume that the inputs a, b are drawn from the uniform distribution in Z_q (this requirement was not needed in MtAwc, but it is in any case satisfied by our application of threshold issuance of group credentials).

1. Alice initiates the protocol by
   - sending $c_A = E_A(a)$ to Bob
2. Bob computes the ciphertext $c_B = b \otimes c_A \oplus E_A(\beta') = E_A(ab + \beta')$ where β' is chosen uniformly at random in Z_N. Bob sets his share to $\beta = -\beta' \bmod q$. He responds to Alice by
   - sending c_B and $\hat{B} = g^\beta$
   - proving in ZK that he knows β such that $\hat{B} = g^\beta$
3. Alice decrypts c_B to obtain $\alpha' = D_A(c_B)$. She sets her share to $\alpha = \alpha' \bmod q$. She checks that $g^\alpha \cdot \hat{B} = B^a$. If the check fails she aborts and stops. Otherwise she sends to Bob $\hat{A} = g^\alpha$.
4. Bob checks that $\hat{A} \cdot \hat{B} = A^b$ and abort if that does not hold.

It is easy to see that if the players are honest the protocol is correct. We now describe the simulation first for Bob and then for Alice. In each case, before we describe how to simulate the above protocol we first point out where the simulation strategy for MtAwc fails in this case.

Simulation of Bob. If Alice is corrupted, in the previous simulation of MtAwc the Bob-simulator encrypted a random $\alpha' \in Z_N$ and published $\hat{B} = B^a g^{-\alpha'}$ guaranteeing that the view of Alice is compatible with a protocol that successfully completes. But this is OK because we know that (i) Alice really encrypted a small number (so there is no reduction $\bmod N$) and (ii) Alice really encrypted $\log_g A$ (so the output of Bob's step is correct). However we don't have this guarantee in MtA. Indeed a malicious Alice could encrypt a large number and the probability of wrapping $\bmod N$ depends on the size of Bob's input which is not available to the simulator. So in this new simulation we use knowledge of Alice's decryption key to approximate the distribution of α' as described below.

Remember that the Bob-simulator knows a and D_A Alice's decryption key (since it can extract both from the proofs of knowledge during the registration phase). The simulator chooses a random $\hat{b} \in Z_q$ and sets $\alpha' = \hat{b} \cdot D(c_A) + \beta'$ for $\beta' \in Z_N$ and returns $c_B = E(\alpha')$ and $\hat{B} = B^a \cdot g^{-\alpha'}$. Note that in this simulation the message that Alice decrypts might be reduced mod N and her check that $g^{D_A(c_B)} \cdot \hat{B} = B^a$ might fail, as in the real protocol. The only difference with the real protocol is that Bob uses the real $b = \log_g B$ to multiply Alice's value, while the simulator uses a random $\hat{b}$.

Notice, however, that the simulation is indistinguishable from the real one. For any two messages $m_0, m_1 \in Z_N$ and for $\beta \in_R Z_N$, it is clear that $m_0 + \beta \bmod N$ and $m_1 + \beta \bmod N$ are identically distributed. In particular then, letting $m_0 = b \cdot a \bmod N$ and $m_1 = \hat{b} \cdot a \bmod N$, it is clear that the value that Alice decrypts is identically distributed in the real protocol and in the simulation.

Simulation for Alice. If the adversary corrupts Bob, the simulation in the previous case proceeds with the simulator encrypting a random value and relying

on the semantic security of E_A. However in our simplified protocol, Alice checks Bob's behavior by performing a test on the decryption of c_B a ciphertext provided by the possibly malicious Bob. This turns Alice into a form of decryption oracle, and simple semantic security is not sufficient anymore. Moreover we can't use a CCA2-secure encryption scheme [25] because we need E to be additively homomorphic.

In our simulation we assume that we are given access to an oracle $Oc_A(c_B, b, \beta)$ which answers 1 if and only if $Dec(c_B) = b \cdot Dec(c_A) + \beta \bmod q$. Since the simulator can extract b, β from the malicious Bob's proof of knowledge, then the simulator can query $O_{c_A}(c_B, b, \beta)$ and accepts if the oracle answers 1.

Security cannot be based on the semantic security of E_A anymore since the presence of the oracle immediately implies that E_A is not semantically secure anymore. However consider the following experiment:

- Generate a key (E, D)
- Generate two random values $a_0, a_1 \in_R Z_q$ and publish $A = g^{a_0}$
- Choose a random bit b and publish $c = E(a_b)$
- Let b' be the output of the adversary who is allowed *restricted* access to the oracle O – by restricted we mean that the oracle will stop working after it outputs 0.

Assumption 1 *We say that the* **Enc-ECR** *assumption holds if for every PPT adversary, the probability that $b = b'$ is negligible.*

Under the Enc-ECR assumption we can prove that no adversary given g^{a_0} can distinguish if the MtA protocol was run with a_0 or a_1 (with both values being "high entropy" in particularly randomly chosen) which is what's needed to complete the simulation above.

We note that our Enc-ECR assumption is a generalized version of the Paillier-ECR assumption introduced in [30] just for Paillier's encryption (we stated it for any additively homomorphic encryption E). It is a weaker version of the Paillier-EC assumption in [38] shown to be false in [39] (the extended version of [38]). In the latter the oracle access is not restricted, which makes the assumption much stronger. In our case it is sufficient to consider the restricted oracle since the real protocol stops if Alice detects cheating. This variation prevents the attack shown in [39] to complete successfully, and indeed no attack is currently known on the ECR assumption.

4 Fully distributed group signatures from BBS

In this section, we present a fully distributed variant of the BBS group signature scheme [10]. We begin by recalling the details of the BBS group signature scheme

(Section 4.1) and then proceed to show how the issuing manager can be replaced by a (t, n) threshold protocol using the MtAwc protocol as a primitive (Section 4.2). We then prove the security of the threshold issuing protocol by reducing it to the security of the centralized BBS signature scheme (Section 4.3). Next, we show how we can realize a simpler protocol by replacing the more expensive MtAwc with the lightweight MtA protocol (Section 4.4). Finally, we recall how the tracing manager can also be distributed in a straightforward manner and present other extensions to our protocol (Section 4.5).

4.1 The BBS group signature scheme

We now provide a detailed description of the BBS group signature scheme [10]. Let (G_1, G_2) denote a bilinear group pair of order q generated by g_1, g_2 respectively. Let ψ denote a computable isomorphism from G_2 to G_1, such that $\psi(g_2) = g_1$. Let $e : G_1 \times G_2 \to G_T$ denote a non-degenerate, bilinear map, where bilinearity and non-degeneracy have the following definitions:

- **Bilinearity**: for all $z_1 \in G_1$, $z_2 \in G_2$, and $d_1, d_2 \in Z$, $e(z_1^{d_1}, z_2^{d_2}) = e(z_1, z_2)^{d_1 d_2}$
- **Non-degeneracy**: $e(g_1, g_2) \neq 1$

Also, let $H : \{0,1\}^* \to Z_p$ denote a hash function which will be modeled as a random oracle.

- Key-Gen On input n, the number of members in the group,
  - choose generator $g_2 \in_R G_2$
  - compute $g_1 = \psi(g_2)$
  - *Issuing Secret Key:* choose $\gamma \in_R Z_q^*$
  - *Issuing Public Key:* $w = g_2^{\gamma}$
  - *Tracing Private Key:* choose $\xi_1, \xi_2 \in_R Z_q^*$
  - *Tracing Public Key* choose $h \in_R G_1 \setminus \{1_{G_1}\}$ and compute $u, v \in G_1$ such that $u^{\xi_1} = v^{\xi_2} = h$
  - output the group public key $gpk = (g_1, g_2, h, u, v, w)$ and the group private key $gmsk = (\gamma, \xi_1, \xi_2)$
- Issue Credential: For every user U: choose $x \in_R Z_p$ and set $A = g_1^{\frac{1}{x+\gamma}}$. The user secret key is $sk_U : (A, x)$.
- Sign On input gpk, sk_U, and message $M \in \{0,1\}^*$, the user U
  - *Encrypts* A: choose $\alpha, \beta \in_R Z_q$ and compute $T_1 = u^{\alpha}$, $T_2 = v^{\beta}$, $T_3 = A_i h^{\alpha+\beta}$;
  - *Proves in ZK that encryption is correct:*
    - computes $\delta_1 = x_i \alpha$, and $\delta_2 = x_i \beta$
    - choose r_α, r_β, r_x, r_{δ_1}, $r_{\delta_2} \in_R Z_q$
    - compute $R_1 = u^{r_\alpha}$, $R_2 = v^{r_\beta}$, $R_3 = e(T_3, g_2)^{r_x} \cdot e(h, w)^{-r_\alpha - r_\beta} \cdot e(h, g_2)^{-r_{\delta_1} - r_{\delta_2}}$, $R_4 = T_1^{r_x} \cdot u^{-r_{\delta_1}}$, and $R_5 = T_2^{r_x} \cdot v^{-r_{\delta_2}}$
    - compute $c = H(M, T_1, T_2, T_3, R_1, R_2, R_3, R_4, R_5) \in Z_q$
    - compute $s_\alpha = r_\alpha + c\alpha$, $s_\beta = r_\beta + c\beta$, $s_x = r_x + cx_i$, $s_{\delta_1} = r_{\delta_1} + c\delta_1$, and $s_{\delta_2} = r_{\delta_2} + c\delta_2$

  - Output $\sigma = (T_1, T_2, T_3, c, s_\alpha, s_\beta, s_x, s_{\delta_1}, s_{\delta_2})$
- Verify On input gpk, M, and σ,
  - compute $\tilde{R}_1 = u^{s_\alpha}$, $\tilde{R}_2 = v^{s_\beta} \cdot T_2^{-c}$, $\tilde{R}_3 = e(T_3, g_2)^{s_x} \cdot e(h, w)^{-s_\alpha - s_\beta} \cdot e(h, g_2)^{-s_{\delta_1} - s_{\delta_2}} \cdot (e(T_3, w)/e(g_1, g_2))^c$, $\tilde{R}_4 = T_1^{s_x} \cdot u^{-s_{\delta_1}}$, and $\tilde{R}_5 = T_2^{s_x} \cdot v^{-s_{\delta_2}}$
  - Accept if $c = H(M, T_1, T_2, T_3, \tilde{R}_1, \tilde{R}_2, \tilde{R}_3, \tilde{R}_4, \tilde{R}_5)$
- Trace On input gpk, $gmsk$, M, and σ,
  - verify that σ is valid on M
  - *Decrypt* A: compute $A = T_3/(T_1^{\xi_1} \cdot T_2^{\xi_2})$
  - use A_i to identify the signer

4.2 Distributing the issuing manager

We now describe our threshold protocol for distributing the BBS issuing manager. This consists of two protocols: one for the generating the master issuing key and then a second protocol for using that distributed master key to issue keys to group members (or *players*).

The players run on input $G_1, G_2, G_3, g_1, g_2, e$. The private issuing key is $\gamma \in_r Z_p^*$. The public key is $w = g_2^\gamma$. We also assume that each player P_i is associated with a public key E_i for an additively homomorphic encryption scheme $\mathcal{E}$. In our instantiation, we will use Paillier, but in principle any additively homomorphic encryption scheme will work.

First (in Section 4.2.1) we describe the protocol for generating the issuing key pair in a distributed manner. Then (in Section 4.2.2) we present the protocol to issue credentials to users in a distributed threshold fashion.

4.2.1 Issuing key generation protocol

- Round 1. Each Player P_i selects $u_i \in_R Z_q$; computes $[KGC_i, KGD_i] = \mathsf{Com}(g_1^{u_i})$ and broadcasts
  - KGC_i
  - E_i, their public Paillier key
- Round 2. Each Player P_i broadcasts KGD_i. Let w_i be the value decommitted by P_i (i.e. for honest players, $w_i = g^{u_i}$). The player P_i performs a (t, n) Feldman-VSS of the value u_i, with w_i as the "free term in the exponent" The public key is set to $w = \prod_i w_i$. Each player adds the private shares received during the n Feldman VSS protocols. The resulting values γ_i are a (t, n) Shamir's secret sharing of the secret key $\gamma = \sum_i u_i$. Note that the values $\Gamma_i = g_1^{\gamma_i}$ are public.
- Round 3 Let $N_i = p_i q_i$ be the RSA modulus associated with E_i. Each player P_i proves in zero knowledge that he knows γ_i using Schnorr's protocol [51]

4.2.2 Issuing signing keys

Recall that a user U's credential in the BBS group signature scheme is a private signing key (x, A), where $A = g_2^{\frac{1}{\gamma+x}}$. Recall moreover that when user U signs, he will encrypt (x, A) (under the tracing manager public key) and prove in ZK that it is a valid credential (i.e. that $e(g_1^x w, A) = e(g_1, g_2)$)

Note that at the time of signing in order to maintain anonymity, both x_i and A_i must remain private. However, there is no need for the mapping between the user's identity U and (x, A) to be private, and indeed we will need the mapping to be public in order for the scheme to be traceable. We assume that each player chooses and publishes a value $x_i \in Z_p^*$ that will be used as its identity in the protocol.

We now describe the protocol for issuing credentials to users with which they can sign messages. The issuing protocol is a t-out-of-n threshold protocol. In particular, the issuing key is shared among n players using (t, n) Shamir secret sharing. In order to issue shares to a new player, $t + 1$ of these players must participate. As in the centralized BBS scheme, let $H : \{0,1\}^* \to Z_p$ denote a hash function which will be modeled as a random oracle.

Let $S \subseteq [1..n]$ be the set of players participating in the key issuance protocol. We assume that $|S| = t'$ where $t < t' \leq n$. During the key issuing protocol, we can share any ephemeral secrets using a (t', t') secret sharing scheme, and do not need to use the general (t, n) structure.

We note that using the appropriate Lagrangian coefficients $\lambda_{i,S}$ each player in S can locally map its own (t, n) share γ_i of γ into a (t', t') share $z_i = \lambda_{i,S}\gamma_i$ of γ, i.e. $\gamma = \sum_{i \in S} z_i$. Since $\Gamma_i = g_1^{\gamma_i}$ and $\lambda_{i,S}$ are public values, all the players can compute $Z_i = g_1^{z_i} = \Gamma_i^{\lambda_{i,S}}$.

When a user requests a credential he submits $x' \in Z_p$ to the players who jointly compute $x = H(x')$ and then compute A. Note that we do not allow the user to choose x directly but instead we use $H(x')$ so that x is randomly chosen in the random oracle model. This is to be consistent with the centralized BBS scheme, in which the issuing manager randomly chooses $x \in Z_p$ but the users do not have freedom over the choice of x.

Let $s = \gamma + x$. Since x is public, each player P_i can use z_i to produce s_i, a share of $s = \gamma + x$ and everyone can compute $S_i = g_1^{s_i}$.

- Phase 1. Each Player P_i selects $\rho_i \in_R Z_p$; computes $[C_i, D_i] = \mathsf{Com}(g_2^{\rho_i})$ and broadcast C_i.
- Phase 2. Each Player P_i broadcasts D_i. Let Ω_i be the values decommitted by P_i who proves in ZK that he knows ρ_i s.t. $\Omega_i = g_2^{\rho_i}$ using Schnorr's protocol [51].
  The players compute
  $$\Omega = [\prod_{i \in S} \Omega_i] = g_2^{\rho}$$
  where $\rho = \sum_{i \in S} \rho_i$. Note that
  $$s\rho = \sum_{i,j \in S} s_i \rho_j \bmod p$$

– Phase 3. Every pair of players P_i, P_j engages in the following multiplicative-to-additive share conversion subprotocol. Notice that since each player will play both roles, every pair of players will actually engage in **two** MtAwc subprotocols of the following form:

  P_i, P_j run MtA with shares s_i, ρ_j respectively. Let α_{ij} [resp. β_{ij}] be the share received by player P_i [resp. P_j] at the end of this protocol, i.e.

$$s_i \rho_j = \alpha_{ij} + \beta_{ij}$$

  Player P_i sets $\tau_i = s_i \rho_i + \sum_{j \neq i} \alpha_{ij} + \sum_{j \neq i} \beta_{ji}$. Note that the τ_i are a (t', t') additive sharing of $s\rho = \sum_{i \in S} \tau_i$

  Note that at the end of this phase the values $S_i = g_1^{s_i}$, $\Omega_i = g_2^{\rho_i}$, $\hat{A}_{ij} = g_2^{\alpha_{ij}}$ and $\hat{B}_{ij} = g_2^{\beta_{ij}}$ are public.

– Phase 4. Every player P_i sends τ_i and the entire transcript of the protocol to the user U. The user can check if τ_i is correct since $T_i = g_2^{\tau_i} \prod_{j \neq i} \hat{A}_{ij}^{-1} \prod_{j \neq i} \hat{B}_{ji}^{-1}$ should be equal to $g_2^{s_i \rho_i}$ which can be checked via the bilinear map, i.e. by checking that $e(g_1, T_i) = e(S_i, \Omega_i)$. If some check fails, the user aborts. Otherwise he reconstructs $\tau = \sum_{i \in S} \tau_i = \rho s = \rho(\gamma + x)$ and $\tau^{-1} \bmod p$ and

$$A = \Omega^{\tau^{-1}} = (g_2^{\frac{1}{\gamma + x}})$$

4.3 Proof of Security

We prove security according to the simulation paradigm. We show that there exists a simulator that without knowing the input of the honest players can produce a view for the adversary that is indistinguishable from the real protocol. Because we are in the dishonest majority case we use "simulation with abort" where there is no guarantee that the honest players will receive the output of the computation due to aborts.

For the key generation protocol we prove that the protocol can be simulated for any randomly chosen public key w. This guarantee that if the real protocol does not abort then the distribution of the public keys generated by our threshold protocol is the same as the distribution of the centralized BBS scheme.

For the credential issuing protocol, similarly, we prove that the simulator given the output (x, A) of the centralized BBS issuing algorithm will generate a indistinguishable view that results in (x, A) as the output.

Theorem 1. *Assuming that*

– *The Strong RSA Assumption holds;*
– KG, Com, Ver, Equiv *is a non-malleable equivocable commitment scheme;*

Our threshold credential issuing protocol is simulatable.

We assume that the adversary controls players $P_2, \ldots, P_{t+1}$ and that P_1 is the honest player. We point out that because we use concurrently non-malleable commitments (where the adversary can see many commitments from the honest

players) the proof also holds if the adversary controls less than t players and we have more than 1 honest player. So the above assumption is without loss of generality.

Because we are assuming a rushing adversary, P_1 always speaks first at each round. Our simulator will act on behalf of P_1 and interact with the adversary controlling $P_2, \ldots, P_n$. Recall how $\mathcal{A}$ works: it first participates in the key generation protocol to generate a public key w for the threshold scheme. Then it requests the group of players to issue credentials to several users which result in running the protocol on inputs $x_1, \ldots, x_\ell$, and the group engages in the signing protocol on those messages.

4.3.1 Simulating the key generation protocol

We need to prove that the key generation protocol results in a public key w which is uniformly distributed in G_1. So the simulator $\mathcal{S}$ will run on input $w = g^\gamma$ sampled uniformly over G_1 and will produce a simulation that outputs w or aborts.

The simulation Sim-Key-Gen is described below, where $\mathcal{S}$ plays the role of P_1.

$\mathcal{S}$ also runs on input a public key E for which he does not know the matching secret key (this is necessary for when we have to make a reduction to the semantic security of the Paillier encryption scheme).

Simulation:

- $\mathcal{S}$ (as P_1) selects a random value $u_1 \in Z_q$, computes $[KGC_1, KGD_1] = \mathsf{Com}(g_1^{u_1})$ and broadcasts KGC_1. $\mathcal{A}$ broadcast commitments KCG_i for $i > 1$;
- Each player P_i broadcasts KGD_i; let w_i be the decommitted value and the accompanying Feldman-VSS ($\mathcal{S}$ will follow the protocol instructions). Each player broadcasts E_i. $\mathcal{S}$ broadcasts $E_1 = E$.
- Let y_i the revealed commitment values of each party. $\mathcal{S}$ *rewinds* the adversary to the decommitment step and
  - changes the opening of P_1 to $\hat{KGD}_1$ so that the committed value revealed is now $\hat{w_1} = w \cdot \prod_{i=2}^{n} y_i^{-1}$.
  - simulates the Feldman-VSS with free term $\hat{w_1}$
- The adversary $\mathcal{A}$ will broadcasts $\hat{KGD}_i$. Let $\hat{w_i}$ be the committed value revealed by $\mathcal{A}$ at this point (this could be $\perp$ if the adversary refused to decommit).
- The players compute $\hat{w} = \prod_{i=1}^{n} \hat{w_i}$ (set to $\perp$ if any of the $\hat{w_i}$ are set to $\perp$ in the previous step).
- The players perform the proofs of knowledge for their secret keys γ_i, p_i, q_i. The simulator $\mathcal{S}$ will simulate these proofs for P_1, and extract the values γ_i, p_i, q_i for $i > 1$).

The only differences between the real and the simulated views is that P_1 runs a simulated Feldman-VSS with free term in the exponent $\hat{y_1}$ for which it does not know the discrete log. But we have shown in Section 2.5 that this simulation is identically distributed from the real Feldman-VSS. So the simulation of the protocol is perfect.

If the simulation does not abort, then it terminates with output w except with negligible probability. This is a consequence of the non-malleability property of the commitment scheme. Indeed, if $\mathcal{A}$ correctly decommits KGC_i twice it must do so with the same string, no matter what P_1 decommits too (except with negligible probability)[4]. Therefore $\hat{w}_i = w_i$ for $i > 1$ and therefore $\hat{w} = w$.

4.3.2 Simulating the issuing protocol

We point out that $\mathcal{S}$ does not know the secret values associated with P_1: its correct share s_1 of the secret key, and the secret key of its public key E_1. The latter is necessary in order to reduce unforgeability to the semantic security of the encryption scheme. However $\mathcal{S}$ knows the secret keys of all the other players, and their shares s_j. It also knows the "public key" of P_1, $S_1 = g^{s_1}$ from the simulation of the key generation protocol.

In the following simulation $\mathcal{S}$ aborts whenever the protocol is supposed to abort.

On input x the protocol should terminate with either $A = g_2^{\frac{1}{\gamma+x}}$ or abort. The simulator chooses a random value $\tau \in Z_q$ and sets $\Omega = A^\tau \in G_2$.

- Phase 1 All the players execute the protocol by broadcasting C_i ($\mathcal{S}$ runs the protocol correctly for P_1).
- Phase 2
  - Each player P_i broadcasts KGD_i; let Ω_i be the decommitted value of each player.
  - $\mathcal{S}$ *rewinds* the adversary to the decommitment step and changes the opening of P_1 to $\hat{KGD}_1$ so that the committed value revealed is now $\hat{\Omega}_1 = \Omega \cdot \prod_{i=2}^{n} \Omega_i^{-1}$.
  - Because of non-malleability $\mathcal{A}$ must open with the same Ω_i or the simulation aborts.
  - The players prove knowledge of ρ_i such that $\Omega_i = g^{\rho_i}$. $\mathcal{S}$ extracts the values ρ_i (for $i > 1$) and simulates the proof for P_1.

  Note that $\mathcal{S}$ does not know ρ_1 s.t. $\hat{\Omega}_1 = g_2^{\rho_1}$.
- Phase 3 All the players execute the MtAwc protocol for s and ρ.
  $\mathcal{S}$ runs the simulation for P_1 since it does not know s_1, ρ_1.
  More specifically, when P_1 interacts with P_i on input s_1, ρ_i, $\mathcal{S}$ will run the "Alice simulation". Note that in this case $\mathcal{S}$ can extract β_{1i}, the output share for P_i, from his ZK proof of knowledge.
  When P_1 interacts with P_i on input s_i, ρ_1, $\mathcal{S}$ will run the "Bob simulation". Note that in this case $\mathcal{S}$ knows the output share α_{i1} for P_i since $\mathcal{S}$ sets it as a random element in Z_N.
  The bottom line is that at this point in the simulation $\mathcal{S}$ knows $\tau_{\mathcal{A}} = \sum_{i>1} \tau_i$ the sum of the values that the adversary should broadcast in the next round. Indeed

$$\tau_{\mathcal{A}} = \sum_{i,j>1} s_i \rho_j + \sum_{i>1} \alpha_{i1} + \sum_{i>1} \beta_{1i}$$

[4] This property is actually referred to as independence. Introducedin [34] as a stronger version of non-malleability, it was later proven equivalent in [12].

and $\mathcal{S}$ knows all the values on the right end side of the equation.
$\mathcal{S}$ sets $\tau_1 = \tau - \tau_{\mathcal{A}}$. If the simulation has not aborted up to this point then it is not hard to verify that $e(g_1, T_1) = e(S_1, \hat{\Omega}_1)$ as required.

– Phase 4. Every player P_i sends τ_i to U. If all the τ_i are correct U reconstructs

$$\Omega^{\tau^{-1}} = A = (g_2^{\frac{1}{\gamma+x}})$$

The only difference between the real and the simulated execution is that we are running the simulation of the MtAwc protocol, which we have shown to be indistinguishable from real executions under the semantic security of $\mathcal{E}$.

The indistinguishability of the simulation is also predicated on the soundness of the ZK proofs (implied by the Strong RSA Assumption), and the non-malleability of the commitment scheme.

4.4 A Simplified Protocol

To improve the efficiency of the protocol, we replace the MtAwc protocol with the simpler MtA one. Recall that the latter requires that the inputs are randomly distributed (which is true in our case since the inputs are the local shares s_i and ρ_i) and that the users have proven knowledge of the secret keys of their additively homomorphic encryption scheme.

Therefore we also modify the Issuing Key Generation Protocol by requiring that each player proves knowledge of D_i (in the case of Paillier this can be done by proving knowledge of the factorization of N_i via a protocol such as [48]).

Theorem 2. *Assuming that*

– *The Enc-ECR and Dlog-ECR Assumptions hold;*
– KG, Com, Ver, Equiv *is a non-malleable equivocable commitment scheme;*

Then our simplified threshold credential issuing protocol is simulatable.

The proof follows the same lines of the proof of Theorem 1 except that we replace the simulation of MtAwc with the simulation of MtA.

4.5 Additional properties

As we mentioned in the introduction, in order to be fully distributed, we need to build threshold protocols for traceability and revocation as well. Although the main technical challenge towards building a fully distributed group signature scheme is thresholding the issuing manager, for completeness, we recall how the other aspects of the protocol can also be thresholdized in a straightforward manner.

4.6 Threshold Traceability

As we discussed in Section 4.1 in the BBS group signature scheme a user secret key (x, A) can be thought of as a signature A on the message x under the group manager's issuing key. To sign, the user encrypts this key under a linear encryption scheme and proves in ZK that the signature is valid.

The key under which the signature is encrypted is held by the group tracing manager who can de-anonymize a signature by simply decrypting the secret key.

The linear encryption used in the BBS scheme is a variation of the ElGamal encryption scheme [26] that works in groups which admits bilinear maps (on these groups ElGamal encryption cannot be used since the Decisional Diffie-Hellman assumption does not hold).

The decryption procedure for the linear encryption scheme is a simple exponentiation and therefore can be easily distributed and indeed this has been presented in [9].

4.7 Revocation

If a user abuses his roles or has her key compromised, the group may decide to revoke her key. In the BBS paper [10] a simple revocation scheme is described which is easily adapted to our distributed scenario.

In the BBS revocation scheme when the issuing manager wants to revoke a key (x, A) (with $A = g_1^{\frac{1}{x+\gamma}}$) the manager has to compute $A* = g_2^{\frac{1}{x+\gamma}}$ which can be done in a distributed fashion in the same way we compute A.

This is a feature of our protocol as a user can be revoked only if a threshold of issuing managers agrees on that, preventing abuses of revocation power.

4.8 Exculpability

In [4, 8] *exculpability* for group signatures is informally defined as follows: no member of the group, not even the group tracing manager, can produce signatures on behalf of the group. *Strong exculpability* means that not even the group issuing manager can sign on behalf of a user.

The BBS scheme is naturally exculpable, but in the centralized scheme it is not *strongly exculpable*. We first point out that by distributing the power of the issuing manager among several servers we provide an intermediate form of exculpability: only a group of issuing managers larger than then the threshold can frame a user.

Yet we can also produce full strong exculpability (where even the entire set of issuing managers cannot frame a user) by considering the strong exculpable variation of the BBS scheme described in [10]. This requires a variation of the issuing protocol where A – the secret key of user U – is computed as $g_U^{\frac{1}{x+\gamma}}$ rather than for a fixed g_1. Then the ZK proof is modified to take into account that the user must prove that he knows a secret associated with g_U (namely U proves that he knows y_U such that $g_1 = g_U h^{y_U}$ for another fixed parameter h). It is not

hard to see that the distributed portion of the protocol does not change except for running it with basis g_U instead of g_1.

4.9 Alternative share conversion protocols

We can use an alternative protocol to convert the shares from multiplicative to additive, which is based on oblivious transfer (as initially proposed by Gilboa [35]). The main disadvantage of that scheme is the increase in communication complexity.

5 Distributed Camenisch-Lysyanskaya Signature Scheme from Bilinear Maps

We now present our protocol for a distributed version of Camenisch and Lysyanskaya's (DCL) group signature scheme from bilinear maps [16]. First, we recall the details of the CL scheme (Section 5.1). Next, we show how to distribute the issuing manager via a (t, n) threshold protocol (Section 5.2. We then prove security of this protocol by reducing it to the security of the centralized CL scheme (Section 5.3). Finally, we show how the tracing manager can be distributed using known techniques (Section 5.4).

5.1 Camenisch and Lysyanskaya Anonymous Credentialing Scheme

Camenisch and Lysyanskaya (CL) proposed an efficient signature scheme based on a discrete-logarithm assumption from earlier work by Lysyanskaya, Rivest, Sahai, and Wolf (LRSW) [16, 42]. The scheme employs bilinear maps to allow a party to obtain a credential and prove in zero-knowledge the knowledge of a credential.

Before presenting the LRSW assumption, we first describe a setup algorithm $(q, G, \mathsf{G}, g, \mathsf{g}, e) \leftarrow \mathit{Setup}$ used in the assumptions description. $G =< g >$ and $\mathsf{G} =< \mathsf{g} >$ describe two different groups of the same prime order q such that $\mathsf{g} = e(g, g)$. e is a bilinear map.

The LRSW assumption follows. Suppose that $G =< g >$ denotes a group chosen by a setup algorithm *Setup*. Let $X, Y \in G$ such that $X = g_x, Y = g^y$. Also, let $O_{X,Y}(\cdot)$ be an oracle that, on input $m \in Z$, outputs a triple $A = (a, a^y, a^{x+mxy})$ for a random a. Then, for all probabilistic poly-time adversaries $\mathcal{A}$, the function $\nu(k)$, defined as follows, is negligible:

$$\begin{aligned} &Pr[(q, G, \mathsf{G}, g, \mathsf{g}, e) \leftarrow \mathit{Setup}(1^k); x \in Z_q; y \in Z_q; X = g^x; Y = g^y; \\ &\quad (m, a, b, c) \leftarrow \mathcal{A}^{O_{X,Y}}(q, G, \mathsf{G}, g, \mathsf{g}, e, X, Y) : m \notin Q \wedge m \in Z_q \wedge \\ &\quad m \neq 0 \wedge a \in G \wedge b = a^y \wedge c = a^{m+mxy}] = \nu(k), \end{aligned}$$

where Q denotes a set of queries that $\mathcal{A}$ made to $O_{X,Y}(\cdot)$.

The CL signature scheme follows. Note that for the purposes of this paper, the signature is equivalent to a credential:

- Key-Gen:
  - run $(q, G, \mathsf{G}, g, \mathsf{g}, e) \leftarrow Setup(1^k)$
  - choose $x \in_R Z_q$ and $Y \in_R Z_q$
  - set $sk = (x, y)$ and $pk = (q, G, \mathsf{G}, g, \mathsf{g}, e, X, Y)$ where $X = g^x$ and $Y = g^y$
  - output sk and pk
- Sign On input message m, secret key $sk = (x, y)$, and public key $pk = (q, G, \mathsf{G}, g, \mathsf{g}, e, X, Y)$:
  - choose $a \in_R G$
  - output the signature $\sigma = (a, a^y, a^{x+mxy})$
- Verfiy On input $pk = (q, G, \mathsf{G}, g, \mathsf{g}, e, X, Y)$, message m, and purported signature $\sigma = (a, b, c)$:
  - accept if the following equations hold: $e(a, Y) = e(g, b)$ and $e(X, a) \cdot e(X, b)^m = e(g, c)$

CL also has a protocol for proving knowledge of a signature (credential). The prover and verifier in this protocol both take as input the public key $pk = (q, G, \mathsf{G}, g, \mathsf{g}, e, X, Y)$. The prover also takes as input message $m \in Z_q$ and signature $\sigma = (a, b, c)$. The protocol follows:

- Phase 1. The prover does the following
  - choose $r, r' \in_R Z_q$
  - compute a blinded version of the signaturee $\tilde{\sigma} = (a^{r'}, b^{r'}, c^{r'r}) = (\tilde{a}, \tilde{b}, \tilde{c}^r) = (\tilde{a}, \tilde{b}, \hat{c})$
  - send $\tilde{\sigma}$ to the verifier
- Phase 2. Let $v_x = e(X, \tilde{a})$, $v_{xy} = e(X, \tilde{b})$, and $v_s = e(g, \hat{c})$.
  - the prover and verifier compute these values locally
  - the prover and verifier carry out a zero-knowledge proof protocol of (μ, ρ) such that $v_s^\rho = v_x v_{xy}^\mu$
  - the verifier accepts if the above proof holds and $e(\tilde{a}, Y) = e(g, \tilde{b})$

5.2 Distributing the CL issuing manager

We now present a (t, n) threshold protocol for distributing the issuing manger in the CL group signature scheme. This protocol has two subprotocols: one for key generation, and one for issuing credentials.

The players run on input $q, G, \mathsf{G}, g, \mathsf{g}, e$. q denotes the prime order of groups $G = \langle g \rangle$ and $\mathsf{G} = \langle \mathsf{g} \rangle$. e denotes a bilinear, non-degenerate, and efficient mapping $e : G \times G \rightarrow \mathsf{G}$. The public key is $pk = (q, G, \mathsf{G}, g, \mathsf{g}, e, X, Y)$, where X and Y come from running a distributed key generation (DKG) protocol among the players. The secret key $sk = (x, y)$ also comes from the DKG protocol such that $X = g^x$ and $Y = g^y$.

In section 5.2.1, we describe the key generation protocol. In section 5.2.2, we describe the protocol for issuing credentials.

5.2.1 Key generation protocol

This protocol uses a DKG subprotocol to create shared secrets and the MtA protocol to keep the combined shares additive.

- Phase 1. All players run two DKG protocols to generate the secret values x and y such that each P_i has share x_i of x and y_i of y where $\sum_i x_i = x$ and $\sum_i y_i = y$ for $1 \leq i \leq n$. The values $X = g^x$ and $Y = g^y$ are public.
- Phase 2. Each P_i, P_j runs the MtAwc protocol to get additive shares such that $x_i y_j = \gamma_{ij} + \delta_{ij}$.
- Phase 3. P_i sets $t_i = x_i y_i + \sum_{i \neq j} \gamma_{ij} + \sum_{i \neq j} \delta_{ij}$. It follows that $xy = \sum_i t_i$. Let $pk = (q, G, \mathsf{G}, g, \mathsf{g}, e, X, Y)$ where $X, Y \in G$, and $\mathsf{G} = \langle \mathsf{g} \rangle$ is the target group of e. Let $sk = (x, y)$.

5.2.2 Credential issuing protocol

The credentialing algorithm runs on input $U = g^u$, sk, and pk. U refers to the user's identity. The value u is secret to the user. $\sigma = (a, b, c)$ denotes the credential the user receives at the end of the issuing protocol. Note that $a = g^\alpha, b = a^y, c = a^x U^{\alpha xy}$. We point out that a credential can be verified by the user by checking that

$$e(a, y) = e(b, g) \quad \text{and} \quad e(a, X) \cdot e(b, X) = e(c, g) \tag{1}$$

We point out that the value α has to be kept secret (it is not hard to see that if the adversary knew α he could forge new credentials). Therefore the value α has to be generated in a shared form by the players. In turn this implies that we need to run another MtA protocol to compute additive shares of the value αxy where each player uses the shares α_i, t_i. However in this case we cannot immediately run the MtAwc protocol because we do not have the values g^{t_i} public that allow the ZK proof of correctness to be run (and again it's not hard to see that if the values g^{t_i} were made public at the end of the Key Generation phase, then the scheme would become insecure because the value $g^{xy} = \prod_i g^{t_i}$ would also be public).

So in this case we run the basic MtA protocol to compute shares of αxy. Note that this means that we can't check if the adversary is inputting the correct values into the computation and this makes simulating the protocol difficult (intuitively we can't rule out an adversarial strategy where, by "messing up" in the MtA protocol, secret information held by the honest players is leaked)[5].

To deal with this issue we have the players distributively check if the resulting credential is valid or not, i.e. if Eq. 1 is satisfied or not. If the credential is not valid, then the check results in the honest players revealing pseudorandom values and therefore leaking no private information. Details appear in the security proof.

- Phase 1. The players select $a = g^\alpha$ via a DKG protocol. Each player P_i holds share a_i such that $\sum_i a_i = \alpha$.

[5] Compare this to the threshold version of the BBS group signature in the previous Section, where we forced the adversary to behave honestly at each step.

- Phase 2. Each P_i, P_j runs the MtA protocol to get additive shares $\tilde{\gamma}_{ij}, \tilde{\delta}_{ij}$ such that $a_i t_j = \tilde{\gamma}_{ij} + \tilde{\delta}_{ij}$.
  P_i sets $z_i = a_i t_i + \sum_{i \neq j} \tilde{\gamma}_{ij} + \sum_{i \neq j} \tilde{\delta}_{ij}$. It follows that $\alpha xy = \sum_i z_i$.
- Phase 3. Each player P_i calculates $b_i = a^{y_i}$ and $c_i = a^{x_i} U^{z_i}$. Player P_i broadcasts b_i.
  Let $b = \prod_i b_i$, $c = \prod_i c_i$, $\mathsf{c}_i = e(c_i, g)$, $\mathsf{c} = e(c, g) = \prod_i \mathsf{c}_i$, $\mathsf{a} = e(a, X)$, $\mathsf{b} = e(b, X)$ and $\mathsf{U} = e(U, g)$
- Phase 4. Player P_i chooses $\ell_i, \rho_i \in_R Z_q$ computes $\mathsf{V}_i = \mathsf{c}_i \mathsf{g}^{\ell_i}$, $\mathsf{A}_i = \mathsf{g}^{\rho_i}$, and $[\hat{C}_i, \hat{D}_i] = \mathsf{Com}(\mathsf{V}_i, \mathsf{A}_i)$ and broadcasts $\hat{C}_i$.
  Let $\ell = \sum_i \ell_i$ and $\rho = \sum_i \rho_i$.
- Phase 5. Player P_i broadcasts $\hat{D}_i$ and proves in ZK that he knows x_i, z_i, ℓ_i, ρ_i such that $V_i = \mathsf{a}^{x_i} \mathsf{U}^{z_i} \mathsf{g}^{\ell_i}$ and $A_i = \mathsf{g}^{\rho_i}$. Let $\mathsf{V} = \mathsf{a}^{-1} \mathsf{b}^{-1} \prod_i \mathsf{V}_i$ (this should be $\mathsf{V} = \mathsf{g}^{\ell}$) and $\mathsf{A} = \prod_i \mathsf{A}_i$.
- Phase 6. Player P_i computes $\mathsf{W}_i = \mathsf{V}^{\rho_i}$ and $\mathsf{T}_i = \mathsf{A}^{\ell_i}$. It commits $[\tilde{C}_i, \tilde{D}_i] = \mathsf{Com}(\mathsf{W}_i, \mathsf{T}_i)$ and broadcasts $\tilde{C}_i$.
- Phase 7. Player P_i broadcasts $\tilde{D}_i$ to decommit to $\mathsf{W}_i, \mathsf{T}_i$.
  If $\prod_i \mathsf{T}_i \neq \prod_i \mathsf{W}_i$ the protocol aborts.
- Phase 8. Otherwise player P_i broadcasts c_i. The user computes the credential (a, b, c) where $c = \prod_i c_i$. If Equation 1 is satisfied the user accepts, otherwise the protocol aborts.

5.3 Sketch of security proof

In this section we prove that our credential issuing protocol is *unforgeable*. In other words we prove that the adversary cannot create a new credential even after requesting many credential for users U of its choice (i.e. we prove that if the original Camenisch-Lysyanskaya credentail is a signature which is existentially unforgeable under adaptively chosen message attack, then so it is our distributed version).

The proof of our protocol proceeds by simulation. First we simulate the key generation protocol: on input a public key $pk = (q, G, \mathsf{G}, g, \mathsf{g}, e, X, Y)$ the simulator runs the adversary to generate a transcript that outputs the same public key. This is done by simply simulating the two DKG protocols to output X and Y, and then simulate the MtAwc protocol (which however has no bearing on the output public key). We note that at the end of this simulation, the simulator knows all the inputs held by the adversary but does not know x_1, y_1, or t_1 for the honest player P_1.

The next step is to simulate the credential issuing protocol every time the adversary requests a credential for a user U. Here the simulator is allowed to receive a valid credential (a, b, c) for U, and will simulate the protocol to output this credential if the protocol is successful. One problem is that the adversary may cause the protocol to fail and the simulator may not be aware if this is a "correct" execution resulting in a correct credential or a "messed up" execution resulting in an abort. However we note that there can be only one aborting simulation (since if that happens the system is reset), so the simulator can guess which execution it is and be correct with non-negligible probability.

More specifically, let Q be the number of "credential queries" made by the adversary. The simulator chooses an index $i \in [1 \dots Q+1]$ and executes a good simulation on queries $1, \dots, i-1$ and an aborting one for the i^{th} query (setting $i = Q+1$ means all executions are correct). The simulator guess will be correct with probability $1/(Q+1)$ which is non-negligible.

More details on the simulation follow. The simulator runs for player P_1 on input the public key pk and the credential a, b, c.

Simulation of executions $1, \dots, i-1$.

- Phase 1. The simulator runs a simulation of the DKG that results in a. Note that the simulator does not know a_1.
- Phase 2. The simulator simulates P_1 in its executions of the MtA protocols. P_i sets $z_i = a_i t_i + \sum_{i \neq j} \tilde{\gamma}_{ij} + \sum_{i \neq j} \tilde{\delta}_{ij}$. It follows that $\alpha xy = \sum_i z_i$.
- Phase 3. The simulator broadcasts $b_1 = b \cdot \prod_{i \neq 1} a^{-y_i}$ (which he can do since he knows the y_i held by the adversary) for player P_1. Note that at this point the simulator does not know the correct c_1.
- Phase 4. Let R_1 be a random element in G. Player P_1 chooses $\ell_1, \rho_1 \in_R Z_q$ computes $\mathsf{V}_1 = \mathsf{R}_1 \mathsf{g}^{\ell_1}$, $\mathsf{A}_i = \mathsf{g}^{\rho_1}$, and $[\hat{C}_1, \hat{D}_1] = \mathsf{Com}(\mathsf{V}_1, \mathsf{A}_1)$ and broadcasts $\hat{C}_1$.
- Phase 5. All the players P_i broadcast $\hat{D}_i$ and prove in ZK that they know x_i, z_i, ℓ_i, ρ_i such that $V_i = \mathsf{a}^{x_i} \mathsf{U}^{z_i} \mathsf{g}^{\ell_i}$ and $A_i = \mathsf{g}^{\rho_i}$. The simulator simulates the ZK proof for P_1 and extracts the values x_i, z_i for the adversary.
  The simulator computes $c_1 = c \cdot \prod_{i \neq 1} a^{-x_i} U^{-z_i}$. Let $\mathsf{c}_1 = e(c_1, g)$
  It rewinds the simulation and it now opens the commitment for P_1 as $\mathsf{V}_1 = \mathsf{c}_1 \mathsf{g}^{\ell_1}$
- Phase 6. Player P_1 computes $\mathsf{W}_1 = \mathsf{V}^{\rho_1}$ and $\mathsf{T}_i = \mathsf{A}^{\ell_1}$. It commits $[\tilde{C}_1, \tilde{D}_1] = \mathsf{Com}(\mathsf{W}_1, \mathsf{T}_1)$ and broadcasts $\tilde{C}_1$.
- Phase 7. Player P_1 broadcasts $\tilde{D}_i$ to decommit to $\mathsf{W}_1, \mathsf{T}_1$.
  If $\prod_i \mathsf{T}_i \neq \prod_i \mathsf{W}_i$ the protocol aborts.
- Phase 8. Otherwise player P_i broadcasts c_1. The user computes the credential (a, b, c) where $c = \prod_i c_i$. If Equation 1 is satisfied the user accepts, otherwise the protocol aborts.

It is not hard to see that this simulation is identically distributed to a real protocol in which the correct signature is computed (unless "noticeable" aborts happen)

Simulation of the last execution.

- Phase 1. The simulator runs a simulation of the DKG that results in a. Note that the simulator does not know a_1.
- Phase 2. The simulator simulates P_1 in its executions of the MtA protocols. P_i sets $z_i = a_i t_i + \sum_{i \neq j} \tilde{\gamma}_{ij} + \sum_{i \neq j} \tilde{\delta}_{ij}$. It follows that $\alpha xy = \sum_i z_i$.
- Phase 3. The simulator broadcasts $b_1 = b \cdot \prod_{i \neq 1} a^{-y_i}$ (which he can do since he knows the y_i held by the adversary) for player P_1. Note that at this point the simulator does not know the correct c_1.

- Phase 4. Let R_1 be a random element in G. Player P_1 chooses $\ell_1, \rho_1 \in_R Z_q$ computes $\mathsf{V}_1 = \mathsf{R}_1 \mathsf{g}^{\ell_1}$, $\mathsf{A}_i = \mathsf{g}^{\rho_1}$, and $[\hat{C}_1, \hat{D}_1] = \mathsf{Com}(\mathsf{V}_1, \mathsf{A}_1)$ and broadcasts $\hat{C}_1$.
- Phase 5. All the players P_i broadcast $\hat{D}_i$ and prove in ZK that they know x_i, z_i, ℓ_i, ρ_i such that $V_i = \mathsf{a}^{x_i} \mathsf{U}^{z_i} \mathsf{g}^{\ell_i}$ and $A_i = \mathsf{g}^{\rho_i}$. The simulator simulates the ZK proof for P_1.
- Phase 6. Player P_1 computes $\mathsf{W}_1 = \mathsf{V}^{\rho_1}$ and $\mathsf{T}_i = \mathsf{A}^{\ell_1}$. It commits $[\tilde{C}_1, \tilde{D}_1] = \mathsf{Com}(\mathsf{W}_1, \mathsf{T}_1)$ and broadcasts $\tilde{C}_1$.
- Phase 7. Player P_1 broadcasts $\tilde{D}_i$ to decommit to $\mathsf{W}_1, \mathsf{T}_1$.
  Since $\prod_i \mathsf{T}_i \neq \prod_i \mathsf{W}_i$ the protocol aborts.

In this case the simulation in indistinguishable from the real one under the DDH assumption on the group G. Indeed the distributed check on Phases 4-7 only reveals if the shares c_i held by the players reconstruct a correct credential or not, but reveal no additional information about each individual share. Indeed each share c_i is "masked" by the value g^{ℓ_i} which is pseudorandom under the DDH (since all we see is g^{ρ_i} and $\mathsf{g}^{\ell_i \rho_i}$.

Completing the proof The simulation above guarantees that if an adversary can forge a credential in the distributed protocol then he can forge credentials directly in the Camenisch-Lysyanskaya centralized scheme (since the adversary could simulate the entire distributed protocol itself). Note that the success probability of the adversary degrades by at least a factor of $1/(Q+1)$ which however is non-negligible.

5.4 Distributing the CL tracing manager

As with the previous scheme, we show how to achieve distributed traceability, thus resulting in a fully distributed group signatrue scheme. As before, distributing the tracing manager is straightforward and the main technical contribution is the protocol for distributing the issuing manager.

The CL scheme achieves privacy by encrypting the identity of the user the Cramer-Shoup cryptosystem [21]. The tracing manager has the Cramer-Shoup decryption key and thus can trace the signer by decrypting their identity.

Canetti and Goldwasser show how to build a threshold variant of Cramer-Shoup encryption [17]. Using the techniques of Canetti and Goldwasser directly, the tracing manager can be replaced by a threshold protocol. Combined with the distributed issuing manager, this yields a fully distributed variant of the CL group signature scheme.

References

1. Ateniese, G., Camenisch, J., Joye, M., Tsudik, G.: A practical and provably secure coalition-resistant group signature scheme. In: Annual International Cryptology Conference. pp. 255–270. Springer (2000)

2. Ateniese, G., de Medeiros, B.: Efficient group signatures without trapdoors. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 246–268. Springer (2003)
3. Ateniese, G., Song, D., Tsudik, G.: Quasi-efficient revocation of group signatures. In: International Conference on Financial Cryptography. pp. 183–197. Springer (2002)
4. Ateniese, G., Tsudik, G.: Some open issues and new directions in group signatures. In: International Conference on Financial Cryptography. pp. 196–211. Springer (1999)
5. Au, M.H., Susilo, W., Mu, Y.: Constant-size dynamic k-taa. In: International Conference on Security and Cryptography for Networks. pp. 111–125. Springer (2006)
6. Bar-Ilan, J., Beaver, D.: Non-cryptographic fault-tolerant computing in constant number of rounds of interaction. In: Proceedings of the eighth annual ACM Symposium on Principles of distributed computing. pp. 201–209. ACM (1989)
7. Barić, N., Pfitzmann, B.: Collision-free accumulators and fail-stop signature schemes without trees. In: International Conference on the Theory and Applications of Cryptographic Techniques. pp. 480–494. Springer (1997)
8. Bellare, M., Micciancio, D., Warinschi, B.: Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In: International Conference on the Theory and Applications of Cryptographic Techniques. pp. 614–629. Springer (2003)
9. Blömer, J., Juhnke, J., Löken, N.: Short group signatures with distributed traceability. In: International Conference on Mathematical Aspects of Computer and Information Sciences. pp. 166–180. Springer (2015)
10. Boneh, D., Boyen, X., Shacham, H.: Short group signatures. In: Franklin, M. (ed.) Advances in Cryptology – CRYPTO 2004. pp. 41–55. Springer Berlin Heidelberg, Berlin, Heidelberg (2004)
11. Boneh, D., Eskandarian, S., Fisch, B.: Post-quantum epid group signatures from symmetric primitives. Tech. rep., Cryptology ePrint Archive, Report 2018/261, 2018. https://eprint. iacr. org/2018/261 (2018)
12. Boneh, D., Gennaro, R., Goldfeder, S.: Using level-1 homomorphic encryption to improve threshold dsa signatures for bitcoin wallet security. In: Latincrypt (2017)
13. Brickell, E., Camenisch, J., Chen, L.: Direct anonymous attestation. In: Proceedings of the 11th ACM conference on Computer and communications security. pp. 132–145. ACM (2004)
14. Brickell, E., Chen, L., Li, J.: A new direct anonymous attestation scheme from bilinear maps. In: International Conference on Trusted Computing. pp. 166–178. Springer (2008)
15. Camenisch, J., Lysyanskaya, A.: A signature scheme with efficient protocols. In: Security in communication networks, pp. 268–289. Springer (2002)
16. Camenisch, J., Lysyanskaya, A.: Signature schemes and anonymous credentials from bilinear maps. In: Annual International Cryptology Conference. pp. 56–72. Springer (2004)
17. Canetti, R., Goldwasser, S.: An efficient threshold public key cryptosystem secure against adaptive chosen ciphertext attack. In: International Conference on the Theory and Applications of Cryptographic Techniques. pp. 90–106. Springer (1999)
18. Chase, M., Derler, D., Goldfeder, S., Orlandi, C., Ramacher, S., Rechberger, C., Slamanig, D., Zaverucha, G.: The picnic signature algorithm specification (2017)
19. Chase, M., Derler, D., Goldfeder, S., Orlandi, C., Ramacher, S., Rechberger, C., Slamanig, D., Zaverucha, G.: Post-quantum zero-knowledge and signatures from

symmetric-key primitives. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 1825–1842. ACM (2017)
20. Chaum, D., Van Heyst, E.: Group signatures. In: Workshop on the Theory and Application of of Cryptographic Techniques. pp. 257–265. Springer (1991)
21. Cramer, R., Shoup, V.: A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In: Annual International Cryptology Conference. pp. 13–25. Springer (1998)
22. Damgard, I., Groth, J.: Non-interactive and reusable non-malleable commitment schemes. In: Proceedings of the thirty-fifth annual ACM symposium on Theory of computing. pp. 426–437. ACM (2003)
23. Damgård, I., Keller, M., Larraia, E., Miles, C., Smart, N.P.: Implementing aes via an actively/covertly secure dishonest-majority mpc protocol. In: International Conference on Security and Cryptography for Networks. pp. 241–263. Springer (2012)
24. Di Crescenzo, G., Ishai, Y., Ostrovsky, R.: Non-interactive and non-malleable commitment. In: Proceedings of the thirtieth annual ACM symposium on Theory of computing. pp. 141–150. ACM (1998)
25. Dolev, D., Dwork, C., Naor, M.: Non-malleable cryptography,". In: Proceedings of the 23rd Annual Symposium on the Theory of Computing, ACM (1991)
26. ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE transactions on information theory **31**(4), 469–472 (1985)
27. Fischlin, M.: Communication-efficient non-interactive proofs of knowledge with online extractors. In: Annual International Cryptology Conference. pp. 152–168. Springer (2005)
28. Fujisaki, E., Okamoto, T.: Statistical zero knowledge protocols to prove modular polynomial relations. In: Annual International Cryptology Conference. pp. 16–30. Springer (1997)
29. Gennaro, R.: Multi-trapdoor commitments and their applications to proofs of knowledge secure under concurrent man-in-the-middle attacks. In: Annual International Cryptology Conference. pp. 220–236. Springer (2004)
30. Gennaro, R., Goldfeder, S.: Fast multiparty threshold ecdsa with fast trustless setup. In: Proceedings of the 25th ACM Conference on Computer and Communications Security. ACM (2018)
31. Gennaro, R., Goldfeder, S., Narayanan, A.: Threshold-optimal dsa/ecdsa signatures and an application to bitcoin wallet security. In: International Conference on Applied Cryptography and Network Security. pp. 156–174. Springer (2016)
32. Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Robust threshold dss signatures. Information and Computation **164**(1), 54–84 (2001)
33. Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Secure distributed key generation for discrete-log based cryptosystems. J. Cryptology **20**(1), 51–83 (2007)
34. Gennaro, R., Micali, S.: Independent zero-knowledge sets. In: International Colloquium on Automata, Languages, and Programming. pp. 34–45. Springer (2006)
35. Gilboa, N.: Two party rsa key generation. In: Annual International Cryptology Conference. pp. 116–129. Springer (1999)
36. Keller, M., Pastro, V., Rotaru, D.: Overdrive: making spdz great again. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 158–189. Springer (2018)
37. Kiayias, A., Tsiounis, Y., Yung, M.: Traceable signatures. In: International Conference on the Theory and Applications of Cryptographic Techniques. pp. 571–589. Springer (2004)

38. Lindell, Y.: Fast secure two-party ecdsa signing. In: Annual International Cryptology Conference. pp. 613–644. Springer (2017)
39. Lindell, Y.: Fast secure two-party ecdsa signing. Tech. rep., Cryptology ePrint Archive, Report 2017/552, 2018. https://eprint. iacr. org/2017/552 (2017)
40. Lindell, Y., Nof, A.: Fast secure multiparty ecdsa with practical distributed key generation and applications to cryptocurrency custody. In: Proceedings of the 25th ACM Conference on Computer and Communications Security. ACM (2018)
41. Lysyanskaya, A., Ramzan, Z.: Group blind digital signatures: A scalable solution to electronic cash. In: International Conference on Financial Cryptography. pp. 184–197. Springer (1998)
42. Lysyanskaya, A., Rivest, R.L., Sahai, A., Wolf, S.: Pseudonym systems. In: Heys, H., Adams, C. (eds.) Selected Areas in Cryptography. pp. 184–199. Springer Berlin Heidelberg, Berlin, Heidelberg (2000)
43. MacKenzie, P., Reiter, M.K.: Two-party generation of dsa signatures. In: Annual International Cryptology Conference. pp. 137–154. Springer (2001)
44. MacKenzie, P., Yang, K.: On simulation-sound trapdoor commitments. In: International Conference on the Theory and Applications of Cryptographic Techniques. pp. 382–400. Springer (2004)
45. Maitland, G., Boyd, C.: Fair electronic cash based on a group signature scheme. In: International Conference on Information and Communications Security. pp. 461–465. Springer (2001)
46. Manulis, M.: Democratic group signatures: on an example of joint ventures. In: Proceedings of the 2006 ACM Symposium on Information, computer and communications security. pp. 365–365. ACM (2006)
47. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: International Conference on the Theory and Applications of Cryptographic Techniques. pp. 223–238. Springer (1999)
48. Poupard, G., Stern, J.: Short proofs of knowledge for factoring. In: Public Key Cryptography, Third International Workshop on Practice and Theory in Public Key Cryptography, PKC 2000, Melbourne, Victoria, Australia, January 18-20, 2000, Proceedings. pp. 147–166 (2000)
49. Rivest, R.L., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM **21**(2), 120–126 (1978)
50. Sakurai, K., Miyazaki, S.: An anonymous electronic bidding protocol based on a new convertible group signature scheme. In: Australasian Conference on Information Security and Privacy. pp. 385–399. Springer (2000)
51. Schnorr, C.: Efficient signature generation by smart cards. J. Cryptology **4**(3), 161–174 (1991)
52. Shamir, A.: How to share a secret. Communications of the ACM **22**(11), 612–613 (1979)
53. Sonnino, A., Al-Bassam, M., Bano, S., Danezis, G.: Coconut: Threshold issuance selective disclosure credentials with applications to distributed ledgers. arXiv preprint arXiv:1802.07344 (2018)
54. Tsudik, G., Xu, S.: Accumulating composites and improved group signing. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 269–286. Springer (2003)
55. Zheng, D., Li, X., Ma, C., Chen, K., Li, J.: Democratic group signatures with threshold traceability. IACR Cryptology ePrint Archive (2008)

A The ZK Proofs for the MtAwc protocol

In this section we describe the ZK proofs that are needed in the MtAwc protocol (see Section 3). The proofs are based on similar ones from [43]: specifically we prove statements that are simpler than the ones needed in [43].

In these proofs the Verifier uses an auxiliary RSA modulus $\tilde{N}$ which is the product of two safe primes $\tilde{P} = 2\tilde{p} + 1$ and $\tilde{Q} = 2\tilde{q} + 1$ with $\tilde{p}, \tilde{q}$ primes. The Verifier also uses two values $h_1, h_2 \in Z^*_{\tilde{N}}$ according to the commitment scheme in [28]. Security is based on the assumption that the Prover cannot solve the Strong RSA problem over $\tilde{N}$.

Therefore our initialization protocol must be augmented with each player P_i generating an additional RSA modulus $\tilde{N}_i$, and values h_{1i}, h_{2i}, together with a proof that they are of the correct form (see [28]).

A.1 Range Proof

This proof is run by Alice (the initiator) in the MtAwc protocol.

The input for this proof is a Paillier public key N, Γ, a value $c \in Z_{N^2}$, and a value $M \in G$ a cyclic group of order q generated by g. The Prover knows $m \in Z_q$ and $r \in Z^*_N$ such that $c = \Gamma^m r^N \bmod N^2$, and $M = g^m \in G$.

At the end of the protocol the Verifier is convinced that the Prover knows such an m and that $m \in [-q^3, q^3]$.

- The Prover selects $\alpha \in_R Z_{q^3}$, $\beta \in_R Z^*_N$, $\gamma \in_R Z_{q^3\tilde{N}}$ and $\rho \in_R Z_{q\tilde{N}}$.
  The Prover computes $A = g^\alpha$, $z = h_1^m h_2^\rho \bmod \tilde{N}$, $u = \Gamma^\alpha \beta^N \bmod N^2$, $w = h_1^\alpha h_2^\gamma \bmod \tilde{N}$.
  The Prover sends z, u, w to the Verifier.
- The Verifier selects a challenge $e \in_R Z_q$ and sends it to the Prover.
- The Prover computes $s = r^e \beta \bmod N$, $s_1 = em + \alpha$ and $s_2 = e\rho + \gamma$ and sends s, s_1, s_2 to the Verifier.
- The Verifier checks that $s_1 \leq q^3$, $g^{s_1} = AM^e$, $u = \Gamma^{s_1} s^N c^{-e} \bmod N^2$ and $h_1^{s_1} h_2^{s_2} z^{-e} = w \bmod \tilde{N}$.

Completeness. By inspection.

Soundness. Let $\tilde{N}, \tilde{s}$ be our Strong RSA challenge. We show how to solve it using a Prover who succeeds on incorrect instances (i.e. where $|m| > q^3$).

Let $h_2 = \tilde{s}$ and $h_1 = h_2^\chi$ for a random $\chi \in Z_{q\tilde{N}}$. It is not hard to see that the distribution of these values is indistinguishable from the real one with sufficiently high probability.

Run the Prover on a successful execution over a challenge e and then rewind him and find a successful execution with challenge $\hat{e}$. Therefore we have the same first message z, u, w and two set of answers s, s_1, s_2 for challenge e, and $\hat{s}, \hat{s}_1, \hat{s}_2$ for challenge $\hat{e}$ both satisfying the verification equations. Let $\Delta_E = e - \hat{e}$, $\Delta_{s1} = s_1 - \hat{s}_1$ and $\Delta_{s2} = s_2 - \hat{s}_2$.

Let $\lambda = GCD(\Delta_{s2} + \chi\Delta_{s1}, \Delta_E)$. Assume $\lambda \neq \Delta_E$: denote with $\lambda_s = (\Delta_{s2} + \chi\Delta_{s1})/\lambda$ and $\lambda_E = \Delta_E/\lambda > 1$. Then we find μ, ν such that $\mu\lambda_s + \nu\lambda_E = 1$.

Then the solution to the Strong RSA challenge is $\tilde{x} = z^{\mu}\tilde{s}^{\nu} \bmod \tilde{N}, \lambda_E$. Indeed note that

$$w = h_1^{s_1} h_2^{s_2} z^{-e} = h_1^{\hat{s}_1} h_2^{\hat{s}_2} z^{-\hat{e}} \bmod \tilde{N}$$

therefore

$$z^{\Delta_E} = h_1^{\Delta_{s1}} h_2^{\Delta_{s2}} = \tilde{s}^{\Delta_{s2}+\chi\Delta_{s1}} \bmod \tilde{N}$$

which implies

$$z^{\lambda_E} = \tilde{s}^{\lambda_S} \bmod \tilde{N}$$

Concluding

$$\tilde{s} = \tilde{s}^{\mu\lambda_s+\nu\lambda_E} = [z^{\mu}\tilde{s}^{\nu}]^{\lambda_E} \bmod \tilde{N}$$

We now need to prove that the case $\lambda = \Delta_E$ cannot happen with high probability.

Consider first the case $\lambda = \Delta_E$ but Δ_E does not divide Δ_{s1}. Write $\chi = \chi_0 + \chi_1\tilde{p}\tilde{q}$ with χ_1 chosen uniformly at random from a set of size $> q$. Note that the value χ_1 is information theoretically secret from the adversary (who only has h_1, h_2). We have that

$$\Delta_{s2} + \chi\Delta_{s1} = \Delta_{s2} + \chi_0\Delta_{s1} + \chi_1\Delta_{s1}\tilde{p}\tilde{q}$$

Then there is a prime power a^b (with $a \geq 2$) such that $a^b|\Delta_E$, $a^{b-1}|\Delta_{s1}$ but a^b does not divide Δ_{s1}. Note that this implies that $a^{b-1}|\Delta_{s2}$. Set $c_0 = (\Delta_{s2} + \chi_0\Delta_{s1})/a^{b-1}$ and $c_1 = \Delta_{s1}\tilde{p}\tilde{q}/a^{b-1}$. We have that $c_0 + \chi_1 c_1 = 0 \bmod a$ and $c_1 \neq 0 \bmod a$. The number of elements χ_1 for which this equivalence holds is at most $q/a + 1$ and thus the probability of this holding for a random choice of χ_1 is at most $\frac{1}{a} + \frac{1}{q}$ which is at most $\frac{1}{2} + \frac{1}{q}$. Otherwise we are in the case above with $\lambda \neq \Delta_E$.

Now consider the case $\lambda = \Delta_E$ and $\Delta_E|\Delta_{s1}$. Note that this implies that $\Delta_E|\Delta_{s2}$ as well. Define $m_1 = \Delta_{s1}/\Delta_E$, $\rho_1 = \Delta_{s2}/\Delta_E$, $\alpha_1 = (e\hat{s}_1 - \hat{e}s_1)/\Delta_E$, $\gamma_1 = (e\hat{s}_2 - \hat{e}s_2)/\Delta_E$.

These ensure that $z = h_1^{m_1} h_2^{\rho_1} \bmod \tilde{N}$, $w = h_1^{\alpha_1} h_2^{\gamma_1} \bmod \tilde{N}$, $s_1 = em_1 + \alpha_1$ and $\hat{s}_1 = \hat{e}m_1 + \alpha_1$.

Finally denote with $m_1' = \Delta_{s1}\Delta_E^{-1} \bmod N$ and $\alpha_1' = (e\hat{s}_1 - \hat{e}s_1)\Delta_E^{-1} \bmod N$. Note that since $m_1' = m_1 \bmod N$ and $\alpha_1' = \alpha_1 \bmod N$, there must be $r_1, \beta' \in Z_N^*$ such that

$$c = \Gamma^{m_1'} r_1^N \quad \text{and} \quad u = \Gamma^{\alpha_1'} (\beta')^N \bmod N^2$$

At this point we know the following facts

$$s_1 < q^3 \quad s_1 = em_1 + \alpha_1 \quad s_1 = em_1' + \alpha_1 \bmod N$$

$$\hat{s}_1 < q^3 \quad \hat{s}_1 = \hat{e}m_1 + \alpha_1 \quad \hat{s}_1 = \hat{e}m_1' + \alpha_1 \bmod N$$

Therefore we can prove that $m_1 \in [-q^3, q^3]$ since $|m_1| \leq |\Delta_{s1}| \leq q^3$. But this implies that $m_1' \in [-q^3, q^3]$ since $m_1' = m_1 \bmod N$ and $N > q^7$.

Note also that $g^{m_1} = M$ via a standard Schnorr-like extraction argument.

Honest-Verifier Zero-Knowledge. The simulator proceeds as in [43]. Choose z, s, s_1, s_2, e according to the appropriate distribution and set $u = \Gamma^{s_1} s^N c^{-e} \bmod N$ and $w = h_1^{s_1} h_2^{s_2} z^{-e} \bmod \tilde{N}$.

A.2 Respondent ZK Proof for MtAwc

This proof is run by Bob (the responder) in the MtAwc protocol.

The input for this proof is a Paillier public key N, Γ and two values $c_1, c_2 \in Z_{N^2}$, together with a value X in $\mathcal{G}$ the DSA group.

The Prover knows $x \in Z_q$, $y \in Z_N$ and $r \in Z_N^*$ such that $c_2 = c_1^x \Gamma^y r^N \bmod N^2$, and $X = g^x \in \mathcal{G}$, where q is the order of the DSA group.

At the end of the protocol the Verifier is convinced of the above and that $x \in [-q^3, q^3]$.

- The Prover selects $\alpha \in_R Z_{q^3}$, $\rho \in_R Z_{q\tilde{N}}$, $\rho' \in_R Z_{q^3\tilde{N}}$, $\sigma \in Z_{q\tilde{N}}$, $\beta \in_R Z_N^*$, $\gamma \in_R Z_N^*$ and $\tau \in_R Z_{q\tilde{N}}$.

  The Prover computes $u = g^\alpha$, $z = h_1^x h_2^\rho \bmod \tilde{N}$, $z' = h_1^\alpha h_2^{\rho'} \bmod \tilde{N}$, $t = h_1^y h_2^\sigma \bmod \tilde{N}$, $v = c_1^\alpha \Gamma^\gamma \beta^N \bmod N^2$, and $w = h_1^\gamma h_2^\tau \bmod \tilde{N}$.

  The Prover sends $u, z.z', t, v, w$ to the Verifier.
- The Verifier selects a challenge $e \in_R Z_q$ and sends it to the Prover.
- The Prover computes $s = r^e \beta \bmod N$, $s_1 = ex + \alpha$, $s_2 = e\rho + \rho'$, $t_1 = ey + \gamma$ and $t_2 = e\sigma + \tau$.

  The Prover sends s, s_1, s_2, t_1, t_2 to the Verifier.
- The Verifier checks that $s_1 \leq q^3$, $g^{s_1} = X^e u \in \mathcal{G}$, $h_1^{s_1} h_2^{s_2} = z^e z' \bmod \tilde{N}$, $h_1^{t_1} h_2^{t_2} = t^e w \bmod \tilde{N}$, and $c_1^{s_1} s^N \Gamma^{t_1} = c_2^e v \bmod N^2$.

Completeness. By inspection.

Soundness. Let $\tilde{N}, \tilde{s}$ be our Strong RSA challenge. We show how to solve it using a Prover who succeeds on incorrect instances (i.e. where $|x| > q^3$).

Let $h_2 = \tilde{s}$ and $h_1 = h_2^\chi$ for a random $\chi \in Z_{q\tilde{N}}$. It is not hard to see that the distribution of these values is indistinguishable from the real one with sufficiently high probability.

Run the Prover on a successful execution over a challenge e and then rewind him and find a successful execution with challenge $\hat{e}$. Therefore we have the same first message u, z, z', t, v, w and two set of answers s, s_1, s_2, t_1, t_2 for challenge e, and $\hat{s}, \hat{s}_1, \hat{s}_2, \hat{t}_1, \hat{t}_2$ for challenge $\hat{e}$ both satisfying the verification equations. Let $\Delta_E = e - \hat{e}$, $\Delta_{s1} = s_1 - \hat{s}_1$, $\Delta_{s2} = s_2 - \hat{s}_2$, $\Delta_{t1} = t_1 - \hat{t}_1$ and $\Delta_{t2} = t_2 - \hat{t}_2$.

Let $\lambda = GCD(\Delta_{s2} + \chi\Delta_{s1}, \Delta_E)$. Assume $\lambda \neq \Delta_E$: denote with $\lambda_s = (\Delta_{s2} + \chi\Delta_{s1})/\lambda$ and $\lambda_E = \Delta_E/\lambda > 1$. Then we find μ, ν such that $\mu\lambda_s + \nu\lambda_E = 1$.

Then the solution to the Strong RSA challenge is $\tilde{x} = z^\mu \tilde{s}^\nu \bmod \tilde{N}, \lambda_E$. Indeed note that

$$z' = h_1^{s_1} h_2^{s_2} z^{-e} = h_1^{\hat{s}_1} h_2^{\hat{s}_2} z^{-\hat{e}} \bmod \tilde{N}$$

therefore

$$z^{\Delta_E} = h_1^{\Delta_{s1}} h_2^{\Delta_{s2}} = \tilde{s}^{\Delta_{s2} + \chi\Delta_{s1}} \bmod \tilde{N}$$

which implies

$$z^{\lambda_E} = \tilde{s}^{\lambda_S} \bmod \tilde{N}$$

Concluding

$$\tilde{s} = \tilde{s}^{\mu\lambda_s + \nu\lambda_E} = [z^\mu \tilde{s}^\nu]^{\lambda_E} \bmod \tilde{N}$$

Let $\lambda' = GCD(\Delta_{t2} + \chi\Delta_{t1}, \Delta_E)$. In a similar way as above we can prove that if $\lambda' \neq \Delta_E$ then we can solve our Strong RSA challenge.

Therefore we can limit ourselves to the case $\lambda = \lambda' = \Delta_E$.

Consider first the case $\lambda = \lambda' = \Delta_E$ but Δ_E does not divide Δ_{s1}. Write $\chi = \chi_0 + \chi_1\tilde{p}\tilde{q}$ with χ_1 chosen uniformly at random from a set of size $> q$. Note that the value χ_1 is information theoretically secret from the adversary (who only has h_1, h_2). We have that

$$\Delta_{s2} + \chi\Delta_{s1} = \Delta_{s2} + \chi_0\Delta_{s1} + \chi_1\Delta_{s1}\tilde{p}\tilde{q}$$

Then there is a prime power a^b (with $a \geq 2$) such that $a^b|\Delta_E$, $a^{b-1}|\Delta_{s1}$ but a^b does not divide Δ_{s1}. Note that this implies that $a^{b-1}|\Delta_{s2}$. Set $c_0 = (\Delta_{s2} + \chi_0\Delta_{s1})/a^{b-1}$ and $c_1 = \Delta_{s1}\tilde{p}\tilde{q}/a^{b-1}$. We have that $c_0 + \chi_1 c_1 = 0 \bmod a$ and $c_1 \neq 0 \bmod a$. The number of elements χ_1 for which this equivalence holds is at most $q/a + 1$ and thus the probability of this holding for a random choice of χ_1 is at most $\frac{1}{a} + \frac{1}{q}$ which is at most $\frac{1}{2} + \frac{1}{q}$. Otherwise we are in the case above with $\lambda \neq \Delta_E$.

In a similar fashion we can remove the case in which $\lambda = \lambda' = \Delta_E$ but Δ_E does not divide Δ_{t1}.

Now consider the case $\lambda = \lambda' = \Delta_E$ with $\Delta_E|\Delta_{s1}$ and $\Delta_E|\Delta_{t1}$. Note that this implies that $\Delta_E|\Delta_{s2}$ and $\Delta_E|\Delta_{t2}$as well.

Define $x_1 = \Delta_{s1}/\Delta_E$, $\rho_1 = \Delta_{s2}/\Delta_E$, $\alpha_1 = (e\hat{s}_1 - \hat{e}s_1)/\Delta_E$, $\rho_1' = (e\hat{s}_2 - \hat{e}s_2)/\Delta_E$, $y_1 = \Delta_{t1}/\Delta_E$, $\sigma_1 = \Delta_{t2}/\Delta_E$, $\gamma_1 = (e\hat{t_1} - \hat{e}t_1)/\Delta_E$ and $\tau_1 = (e\hat{t_2} - \hat{e}t_2)/\Delta_E$.

Define $x_1' = x_1 \bmod N$ and $y_1' = y_1 \bmod N$. Note that by definition

$$c_1^{x_1'} \Gamma^{y_1'} \kappa^N = c_2 \bmod N^2$$

for some κ as needed. And $g^{x_1} = X \in \mathcal{G}$. So we have extracted the required x, y. As in the previous proof we can establish that $x_1, x_1' \in [-q^3, q^3]$.

Honest-Verifier Zero-Knowledge. The simulator proceeds as in [43] and in the previous ZK proof.

A.3 The Strong RSA Assumption

Since the ZK Proofs rely on the Strong RSA Assumption, we recall it here.

Let N be the product of two safe primes, $N = pq$, with $p = 2p' + 1$ and $q = 2q' + 1$ with p', q' primes. With $\phi(N)$ we denote the Euler function of N, i.e. $\phi(N) = (p-1)(q-1) = p'q'$. With Z_N^* we denote the set of integers between 0 and $N-1$ and relatively prime to N.

Let e be an integer relatively prime to $\phi(N)$. The RSA Assumption [49] states that it is infeasible to compute e-roots in Z_N^*. That is, given a random element $s \in_R Z_N^*$ it is hard to find x such that $x^e = s \bmod N$.

The Strong RSA Assumption (introduced in [7]) states that given a random element s in Z_N^* it is hard to find $x, e \neq 1$ such that $x^e = s \bmod N$. The assumption differs from the traditional RSA assumption in that we allow the

adversary to freely choose the exponent e for which she will be able to compute e-roots.

We now give formal definitions. Let $SRSA(n)$ be the set of integers N, such that N is the product of two $n/2$-bit safe primes.

Assumption 2 *We say that the Strong RSA Assumption holds, if for all probabilistic polynomial time adversaries $\mathcal{A}$ the following probability*

$$Prob[\ N \leftarrow SRSA(n)\ ;\ s \leftarrow Z_N^*\ :\ \mathcal{A}(N,s) = (x,e)\ \ s.t.\ \ x^e = s \bmod N\ ]$$

is negligible in n.

B Group Signature formal definitions from [8]

As mentioned in Section 2.3, Bellare *et al.* formalize the properties and security definitions relating to group signature schemes [8].Our security arguments in this paper do not deal with these definitions directly. Instead we show by simulation that our distributed protocols are secure by reducing to the security of the centralized schemes, which are in turn have previously proven secure by these definitions. Nevertheless, for completeness, we recall the formal definitions from [8] here.

Group signatures have two formal security requirements, full-anonymity and full-traceability, the imply a number of other informal requirements that together ensure security of the scheme [8]. Full-anonymity's definition employs an indistinguishability-based formalization in which the adversary produces a message and a pair of identities that belong to the group in question. The adversary then receives a signature on the message under a random one of the two identities. Finally, the adversary tries to determine which identity associates to the signature. To achieve security, the adversary must have no more than a negligible advantage over one-half in deciding the correct identity. Adding to the situation, the adversary has access to the secret keys of every group member and can see the results of the group manager attempting to open arbitrary signatures chosen by the adversary, except for the challenge signature.

Full-traceability requires that a collusion of group members cannot create a valid signature by pooling their secret keys without the signature being vulnerable to being caught by the group manager. The opening algorithm must be able identify a member of the group as the owner of the signature in the event of collusion. This requirement must hold even if the colluding group knows the group manager's secret key for opening signatures.

Before formalizing a definition for group signature schemes, we first define negligibility. We say a function $f : N \rightarrow N$ is *nice* if it is polynomially bound and computable in polynomial time. The notion of a function $\nu : N \rightarrow N$ begin negligible is standard. The negligibility definition in this paper applies a two-argument function $\mu : N \times N \rightarrow N$. μ is negligible if for every nice function $n : N \rightarrow N$ the function $\mu_n : N \rightarrow N$ is negligible, where $\mu_(k) = \mu(k, n(k))$ for all $k \in N$.

We now formulate a definition of group signatures. Let $\mathcal{GS} = (\mathsf{Key-Gen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Open})$ denote a group signature scheme:

- Key-Gen denotes a key generation algorithm that takes as input 1^k, 1^n, where $k \in N$ denotes the security parameter and $n \in N$ denotes the number of members in the group. This algorithm outputs a tuple $(gpk, gmsk, \boldsymbol{gsk})$, where gpk is the group public key, $gmsk$ is the group secret key, and $\boldsymbol{gsk}$ is an n-vector of keys such that $\boldsymbol{gsk}[i]$ is a secret signing key for player $i \in [n]$.
- Sign is a random algorithm that takes as input a secret signing key $\boldsymbol{gsk}[i]$ and message m. It returns a signature of m under $\boldsymbol{gsk}[i]$.
- Verify is a deterministic algorithm that on input group public key gpk, message m and candidate signature σ returns 1 if it σ is valid on m or 0 if not.
- Open is a deterministic algorithm that takes as input group manager secret key $gmsk$, message m, and signature σ and returns identity i or, if it fails, $\perp$.

The scheme has the following corrections requirement: for all $k \in N$, all $(gpk, gmsk, \boldsymbol{gsk} \in [\mathsf{Key-Gen}(1^k, 1^n)])$, all $i \in [n]$, and all $m \in \{0,1\}^*$:

$$\mathsf{Verify}(gpk, m, \mathsf{Sign}(\boldsymbol{gsk}[i], m)) = 1$$

and

$$\mathsf{Open}(gmsk, m, \mathsf{Sign}(\boldsymbol{gsk}[i], m)) = i$$

In words, the verification requirement ensures that valid signatures with the appropriate message verify, and the open requirement ensures that the Open algorithm finds the identity when provided with a valid signature.

We now formalize full-anonymity. Recall that the adversary has the secret keys of every group member. Let $\mathsf{Open}(gmsk, \cdot)$ denote an opening oracle that when queried with message m and signature σ answers with $\mathsf{Open}(gmsk, m, \sigma)$. The adversary has access to this oracle. For any group signature scheme $\mathcal{GS}(\mathsf{Key-Gen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Open})$, adversary A, and bit b, the following experiment applies:

- Experiment $\mathbf{Exp}^{\text{anon-b}}_{\mathcal{GS},A}(k, n)$
  - $(gpk, gmsk, \boldsymbol{gsk}) \xleftarrow{R} \mathsf{Key-Gen}(1^k, 1^n)$
  - $(St, i_0, i_1, m) \xleftarrow{R} A^{\mathsf{Open}(gmsk,\cdot,\cdot)}(\mathsf{choose}, gpk, \boldsymbol{gsk}); \sigma \xleftarrow{R} \mathsf{Sign}(\boldsymbol{gsk}[i_b], m)$
  - $d \xleftarrow{R} A^{\mathsf{Open}(gmsk,\cdot,\cdot)}(\mathsf{guess}, St, \sigma)$
  - If A did not query its oracle with m, σ, in the guess stage then return d EndIf
  - Return 0

To explain the experiment, we start with the adversary. A has two stages: choose and guess:

- Phase1.(choose) On input group member secret key gpk
  - A can query $\mathsf{Open}(gmsk, \cdot)$ on signatures of its choice

  - A outputs two valid identities $1 \leq i_0, i_1 \leq n$, message m, and some state information St
- Phase2.(guess) On input the state information St and a signature on m produced using the secret key of either i_0 or i_1 chosen at random
  - A can query the oracle, but not on the challenge signature
  - A returns its guess of the identity

A's advantage in breaking full-anonymity is:

$$\mathbf{Adv}^{\text{anon}}_{\mathcal{GS},A}(k,n) = Pr[\mathbf{Exp}^{\text{anon-1}}_{\mathcal{GS},A}(k,n) = 1] - Pr[\mathbf{Exp}^{\text{anon-0}}_{\mathcal{GS},A}(k,n) = 1]$$

If the two-argument function $\mathbf{Adv}^{\text{anon}}_{\mathcal{GS},A}(\cdot,\cdot)$ is negligible with respect to the definition of negligibility above for any polynomial time adversary, then $\mathcal{GS}$ is fully-anonymous.

We now formalize full-traceability. Recall that for full-traceability, no colluding set S of group members can create an unopenable signature. Note that this requirement implies than any signature generated by the colluders must trace back to a member of the group when running the Open algorithm. Further note that the adversary has the group manager's secret key. Full-traceability's definition uses to the following experiment:

- $\mathbf{Exp}^{\text{trace}}_{\mathcal{GS},A}(k,n)$
  - $(gpk, gmsk, \boldsymbol{gsk}) \xleftarrow{R} \mathsf{Key-Gen}(1^k, 1^n)$
  - $St \leftarrow (gmsk, gpk); \mathcal{C} \leftarrow \emptyset; K \leftarrow \epsilon; Cont \leftarrow \mathsf{true}$
  - While $(Cont = \mathsf{true})$ do
    - $(Cont, St, j) \xleftarrow{R} A^{\mathsf{Sign}(\boldsymbol{gsk}[\cdot],\cdot)}(\mathsf{choose}, St, K)$
    - If $Cont = \mathsf{true}$ then $\mathcal{C} \leftarrow \mathcal{C} \cup \{j\}; K \leftarrow \boldsymbol{gsk}[j]$ EndIf
  - EndWhile
  - $(m, \sigma) \xleftarrow{R} A^{\mathsf{Sign}(\boldsymbol{gsk}[\cdot],\cdot)}(\mathsf{guess}, St)$
  - If $\mathsf{Verify}(gpk, m, \sigma) = 0$ then return 0; If $\mathsf{Open}(gmsk, m, \sigma) = \perp$ return 1
  - If there exists $i \in [n]$ such that the following are true then return 1 else return 0
    1. $\mathsf{Open}(gmsk, m, \sigma) = i$
    2. $i \notin \mathcal{C}$
    3. i, m was not queried by A to its oracle

Once again, the adversary has a choose stage and a guess stage:

- Phase1.(choose) On input group public key gpk and group manager secret key $gmsk$
  - A corrupts a set $\mathcal{C}$ of group members
  - $\mathcal{C}$ identity numbers of corrupted members
- Phase2.(guess)
  - A outputs a forgery (m, σ)

A wins if σ is a valid group signature on m and the opening algorithm returns $\perp$ or i such that $i \notin \mathcal{C}$. Like in full-anonymity, the experiment returns 1 if A wins and 0 if A loses. A's advantages has the following description:

$$\mathbf{Adv}^{\text{trace}}_{\mathcal{GS},A}(k,n) = Pr[\mathbf{Exp}^{\text{trace}}_{\mathcal{GS},A}(k,n) = 1]$$

If the two-argument function $\mathbf{Adv}^{\text{trace}}_{\mathcal{GS},A}(\cdot,\cdot)$ is negligible, then $\mathcal{GS}$ is fully-traceable.