Quantum-resistant secret handshakes with dynamic joining, leaving, and banishment: GCD revisited

Olivier Blazy
Computer Science Laboratory of the École Polytechnique
Paris, France
olivier.blazy@polytechnique.edu

Philippe Gaborit
Université de Limoges/XLIM/CNRS 7252
Limoges, France
gaborit@unilim.fr

Philippe Krejci
Université de Limoges/XLIM/CNRS 7252
Limoges, France
krejci@xlim.fr

Cristina Onete
Université de Limoges/XLIM/CNRS 7252
Limoges, France
maria-cristina.onete@unilim.fr

ABSTRACT

Secret handshakes, introduced by Balfanz *et al.* [3], allow users associated with various groups to determine if they share a common affiliation. These protocols ensure crucial properties such as fairness, affiliation privacy, and *result-hiding*. Over time, various secret-handshake schemes have been proposed, with a notable advancement being the modular GCD framework by Tsudik and Xu.

Building upon this modularity, we propose significant updates. By addressing hidden complexities and revising the security model, we enhance both the efficiency and the privacy guarantees of the protocol. Specifically, we achieve the novel property of Self distinction—the ability to distinguish between two users in a session without revealing their identities—by replacing the group signature primitive with a new construct, the List MAC. This primitive is inherently untraceable, necessitating adjustments to the original syntax to support stronger privacy guarantees. Consequently, we introduce the Traitor Catching paradigm, where the transcript of a handshake reveals only the identity of a traitor, preserving the anonymity of all other participants.

To showcase the flexibility and robustness of our updated framework, we present two post-quantum instantiations (a hash-based one and another based on lattices). Our approach not only corrects prior limitations but also establishes a new benchmark for privacy and security in secret handshakes.

KEYWORDS

anonymity, self-distinguishability, secret handshakes

1 INTRODUCTION

Secret handshakes were originally introduced by Balfanz *et al.* [3] as a means of allowing two (or more) users to ascertain whether they belong to the same group *without revealing their affiliations*, either to each other or to an external adversary. An extension of secret handshakes is affiliation-hiding key-agreement [14]: if all participants share an affiliation, they can establish a secure communications' channel.

Say two users, Alice and Bob, are registered on a given platform (*e.g.,* a social network). Though unacquainted, they are open to exchanging messages – on condition that they share a characteristic (a geographical area, a common cause, etc.). The more sensitive the affiliation, the more important it is to hide it until both users are sure they share it. For instance, two political dissidents in an autocratic regime might have a lot to lose if a state agent were to learn of their connections or political opinions.

In a secret-handshake context, various affiliations can co-exist simultaneously, each affiliation corresponding to a group. Users can gain new affiliations or voluntarily drop out of groups. An authority called a group manager, takes over the administrative duties in each group: new users joining, current users leaving or being revoked, as well as updates to the group keys [1]. Later, subgroups of user can interact in secret-handshake protocol sessions, which will end in the acceptance by all users (and the computation of a secure-channel key) if they belong to the same group. Even if they do not belong to the same group, protocol participants enjoy strong privacy properties, such as: session-unlinkability (the same user cannot be recognized or tracked across sessions), affiliation-hiding[2] (hiding the user's affiliation from users from different groups), result-hiding (users of any group, not taking part in the secret handshake, are unaware of its result), and handshake simulatability (a primitive form of deniability).

An established modular way of constructing secret handshakes due to Tsudik *et* Xu [20] relies on three different building blocks: a group-signature scheme, centralized key-agreement, and a distributed key-agreement protocol. While the GCD construction has some advantages has a *potential* for strong security and privacy guarantees, the framework is only partially proved secure. Moreover, the instantiation proposed by [20] features vulnerabilities and an unsound circularity in the use of the session key which ensures security cannot be proved under standard assumptions. Moreover, while Tsudik *et* Xu introduce *self-distinction* as a desirable property, their scheme does not guarantee it.

And yet, self-distinction can be a valuable property. In a nutshell, it allows an honest secret-handshake participant to learn that all the other handshake peers belong the original participant's group *and that they are distinct.* This gives a reliable lower bound on the number of users running a particular secret-handshake session. It can allow, for instance, a journalist to have an objective lower bound on the number of participants in a protest movement, or a

[1] However, we limit the power of group managers and ensure non-frameability even with respect to corrupt or malicious group managers.

[2] Also called detection-resistance.

candidate for a particular election to gauge public support before entering his, her, or their bid.

Another salient point is that of users misbehaving (and being banned upon doing so, against their will). In many schemes, banishment involves an external entity called a judge and reveals more than just the identity of the banished user.

Though secret handshakes abound in the literature, it is to this day challenging to design a secret-sharing protocol that combines:

- Strong privacy requirements, such as: unlinkability, result-hiding, and handshake-simulatability;
- The property of self-distinction;
- Dynamic handling of users joining and leaving groups;
- Handling of banishment with better privacy properties;
- Resistance to quantum attacks;
- Fully-formal security models and proofs.

Our contributions. In this paper we take up this challenge and provide a modular construction, which departs from the original GCD compiler, but replaces the initial group signature by a new primitive we dub ListMAC.

ListMACs are versions of list signatures, in turn related to group signatures. If we compare ListMACs to group signatures, we find a few significant differences:

Matching ListMACs allow group members to only authenticate a few messages anonymously within the group[3]. This property almost immediately provides both self-distinction and a new property of tracing banned or leaving users.

Local verification Group and list signatures are verified by using a publicly-known group key. In ListMACs, verification can only be done locally, by members of the same group. This renders it ideal for the problem of secret handshakes, in which authentication must only function when all users can prove they are members of the same group.

No Traceability Group (and list) signatures provide Traceability by means of an opening algorithm, which uses trapdoors in order to track down a signer and prove he is the signer of a message. ListMACs achieve a slight relaxation of this property that we dub Traitor Catching. The difference is subtle: whereas in Traceability an authority can identify all the participants in a secret handshake, in Traitor Catching a potentially malicious entity will then effectively be banned from the group against his, her, or their will, without all the participants' identities being compromised.

Since we aim for quantum resistance for our ListMACs, we instantiate them by modifying EPID (Enhanced Privacy ID [6]) signatures, which allows a trusted platform module (TPM) to create salve-keys that are distributed to new devices. In our instantiation, the group ListMAC manager creates keys jointly with the users.

In particular, we extend *lattice-based* [11] and *hash-based* EPID [9] for our construction.

Our second and main contribution is a modular protocol we dub LCA, which constructs secret handshake using three ingredients: ListMACs, a group channel which allow managers to broadcast and users to unicast, denoted CBU2, and an anonymous group key-agreement with fresh randomness AGKA-FR. These two latter primitives can be viewed as proper formalizations and extensions, respectively, of the centralized secure channel (CSC) and the decentralized group key-agreement (DGKA) protocols used by the GCD construction. An important contribution of our work is the formal definition of the CBU2 and AGKA-FR primitives required for secret handshakes.

Our second and main contribution is a modular protocol we dub LCA, which constructs secret handshake using three ingredients: ListMACs, a secure group channel with dynamic updates, which allow managers to broadcast and users to unicast, denoted CBU2, and an anonymous group key-agreement with fresh randomness AGKA-FR. These two latter primitives can be viewed as proper formalizations and extensions, respectively, of the centralized secure channel (CSC) and the decentralized group key-agreement (DGKA) protocols used by the GCD construction. An important contribution of our work is the formal definition of the CBU2 and AGKA-FR primitives required for secret handshakes.

Our secret handshake construction, LCA, is instantiated using quantum-resistant cryptography. The resulting scheme guarantees strong privacy properties, but also self-distinction, and can handle dynamic group cases. Our final, and equally-important, contribution consists of the proofs of the properties our scheme provides.

GCD vs. LCA. Our work extends existing results by Tsudik and Xu [20] – however, our contributions with respect to that existing framework are manyfold and, we believe, impactful. Amongst the main differences between our results and [20], we mention:

- Introducing ListMAC as a new cryptographic primitive, and instantiating it in the quantum-secure setting;
- Formally defining the CBU2 and AGKA-FR building blocks, which are non-trivial extensions of the Centralized Secure Channel and decentralized group key-agreement primitives in [20];
- Formally define the desired security and privacy properties for secret handshakes;
- Constructing the LCA scheme from ListMACs, CBU2, and AGKA-FR. Apart from its provable security, our scheme handles dynamic banishment, and provably provides properties unattained by GCD, such as self-distinction.
- We also introduce a new paradigm for catching banned users that increases innocent-user privacy.

We emphasize that, since the proofs of [20] contain unclear and circular arguments, our work is the first to provably confirm the validity of a modular construction paradigm for secret handshakes.

Related Work. The literature of secret handshakes is vast, and features constructions which are difficult to compare, as they provide very diverse properties. Some such schemes feature stronger privacy properties, others seek to provide better traceability. In some (*e.g.,* [3, 17], the authentication of members of the same group must take place prior to the establishment of a secure channel, whereas in later work, parties compute some partial key material before authentication is finished [20].

Our work in this paper comes closest to the GCD framework, proposed by Tsudik and Xu [20]. The framework is relatively generic, featuring group signatures (that provide Unlinkability), group

[3] In practice, as soon as a user authenticates the same message twice, that user is correctly identified as a cheater and can be traced.

key-agreement, and a permanent secure channel for group updates. Apart from standard properties of secret handshakes, such as handshake-simulatability, result-hiding, unlinkability, etc., this work introduces a new and interesting property, namely self-distinction. While [20] provides a modular, generic construction, we show in this paper that it does not achieve the properties it requires in a provable manner.

A recent instantiation of the GCD framework introduced the mCSH protocol [23], which uses blind signatures in order to authenticate, while hiding the affiliation publicly during the handshake. Moreover, the construction achieves quantum-resistance through the instantiation of GCD with quantum-secure primitives. Our generic construction, which improves the GCD framework, can also be instantiated in the quantum-secure setting, as we show in our paper.

However, GCD is not the only way to construct secret handshakes. In the following, we review some further approaches to constructing secret handshakes.

The first constructions of secret handshakes [3] featured one-time credentials – essentially pseudonyms generated by group managers in order to allow users to simultaneously identify each other and hide their identities. This scheme achieves properties such as Unlinkability and Full-unlinkability, but at significant storage costs and a large overhead in case of banned users.

A different approach featured using identity-based encryption and a description of user identities as a combination of various descriptive data [2].

A different protocol, called FSSH [1] is based on Zero-Knowledge Arguments of Knowledge (ZKAoKs), with users obtaining updatable authentication tokens from their group managers. This method provides many properties, but loses out in terms of modularity with respect to the GCD framework, and does not achieve unlinkability.

Table 1: Valid properties per protocols: ✓ pass, ✗ fail

	Unlinkability	User authentication	Traceability	Self distinction	Result-hiding	Handshake simulability	Non-frameability	Full-unlinkability	Traitor Catching
CSH [23]	✓	✓	✓	✗	✓	✓	✗	✓	✗
FSSH [1]	✗	✓	✓	✗	✓	✓	✓	✗	✗
GCD 1 [20]	✓	✓	✗	✗	✓	✓	✗	✓	✗
GCD 2 [20]	✓	✓	✗	✓	✓	✓	✗	✗	✗
Our	✓	✓	✗	✓	✓	✓	✓	✗	✓

2 PRELIMINARIES

2.1 Secret handshakes

We consider users[4] U_i, which may (or may not) be a part of some given group G. A special user, called a group authority or group manager GA, will be associated with each group. We denote by $\mathcal{GA}$ the set of group authorities and by $\mathcal{U}$ the set of all (regular) users, and demand that the two sets are disjoint[5].

Secret handshakes are interactive protocols run among a set Δ of users. Users $U_i \in \Delta$ may or may not be part of a common group G. At the end of the protocol run, each U_i either: (1) Concludes that all the $U_i \in \Delta$ belong to a group G. Each U_i computes a session key K; or (2) Concludes that some of the users do not share their affiliation; no key is then established.

We formally present the syntax of secret handshake protocols like GCD in Appendix A. In terms of properties, GCD, as well as other secret handshake schemes, aim to guarantee the following properties[6]:

User authentication (Auth): an outsider to a group G cannot convince a user $U \in G$, that the adversary $\mathcal{A} \in G$.

Handshake simulability (Hand-Sim): a group outsider with no knowledge of group data (*e.g.,* through corruption) cannot distinguish between a handshake with a group member and a handshake with a simulator.

Full-unlinkability (F-Unlink): no adversary can link two handshakes run with the same user U, even if U's private material was corrupted. A weaker form of this property, called just Unlinkability (Unlink) will not allow users to be corrupted.

Result-hiding (Res-Hide): a legitimate group member not participating in a handshake cannot know whether the handshake finished successfully or not.

Traceability (Trace): the group authority can trace all the users involved in a given handshake.

Non-frameability (NF): no collusion of malicious users, aided by the group authority, can frame an honest user of taking part in a session if that is not so.

Self distinction (S-Dist): a user can establish that all participants in a handshake are distinct.

2.2 The GCD Approach

The GCD approach. In [20], secret handshakes feature groups with dynamic joining and leaving mechanisms, and algorithms ensuring the well-running of secret handshakes. Finally, the GCD framework also incorporates a mechanism for tracing: given a handshake transcript from a successful handshake, the group authority can recover the identities of all the participants (and possibly prove this to a judge J).

2.2.1 The GCD Construction. In 2005, Tsudik and Xu proposed a modular GCD construction for secret handshakes, relying on three main cryptographic sub-components:

[4] We will require user to have unique identifiers. In practice, this could be implemented by an additional registration step including authentication.

[5] Note that in reality, some users can manage groups; our construction simply divides these roles and associates to each one potentially different credentials.

[6] We use a slightly different (but we believe more descriptive) name for classical security and privacy properties in Secret Handshakes. We do this to improve legibility and the impact on privacy community.

Group signatures: GCD requires a group signature scheme with dynamic user addition, leaving, and key-updates. The group authority both manages the group and traces problematic signatures and issues, to each user U_i, a pair of group signature keys $(\mathsf{GSig}.pk, \mathsf{GSig}.sk_i)$. Later, the users group-sign a value that is unique per user to obtain an unlinkable signature σ_i, traceable only by the group authority;

Centralized secure channel: The protocol relies on group members sharing a secret value k, which has to be regularly updated whenever users join or leave the group; this value is broadcast by GA over a group secure channel and is essentially half of the key-material for the secret handshake;

Decentralized group key-agreement: This allows handshake participants to compute an unauthenticated group key $\hat{k}$. This is the second half of the key-material for the secret handshake.

In addition, the GCD scheme also relies on the following, much less discussed primitives: an IND-CCA-secure public-key encryption scheme to ensure traceability; symmetric-key encryption whose properties, which are not specified, are discussed below and in Appendix A.1; a MAC scheme with no precise security requirements, which we return to again below and in Appendix A.1; and mutually-secure channels between the group authority and each of the group members.

The GCD scheme. We proceed to intuitively indicate how the GCD scheme works (for full details, please see [20]).

In this scheme, group authorities will each own a pair of PKE keys $(\mathsf{pk}_{\mathsf{GA}}, \mathsf{sk}_{\mathsf{GA}}$, as well as some group-signature opening trapdoor τ. Users U_i belonging to a group G managed by GA maintain secure channels to GA, over which they will receive group-signature keys $(\mathsf{GSig.pk}, \mathsf{GSig.sk}_i)$ and a common secret value k_G.

Group dynamics. Users can dynamically join, leave or be revoked from the group; in all cases, the group authority will refresh (via the centralized secure channel and mutually-authenticated user-to-GA channels) their group signature keys $(\mathsf{GSig.pk}, \mathsf{GSig.sk}_i)$ as well as a secret value, denoted k_G, which is known only to current group members.

Secret handhshake. Whenever users $U_1, \ldots, U_m$ run a secret handshake, the protocol begins with a decentralized group key-agreement between $U_1, \ldots, U_m$, so that users compute a session key k.

During the secret handshake, each user (belonging to some group G) will compute the value $k_i \leftarrow k_G \oplus k$, where k_G is the current secret value shared over the centralized channel of G and k is the key computed through decentralized key-agreement.

Once k_i is computed, each user U_i broadcasts a MAC $\mathtt{MAC}_{k_i}(s_i, i)$, on its index i within the set of handshake users, and s_i is a potentially-public, user-specific value (*e.g.*, some user-specific message). Having received everyone else's MAC tag, U_i verifies them.

The next step for each user U_i is to check all the received MAC values. Note that whenever U_i checks the tag of a user U_j which is in the same group as U_i, the tag verifies because the users computed the same key. When this is not the case, the MAC verification fails. From this point on, U_i simulates the protocol by outputting random values of the correct format, thus achieving result-hiding.

If all the MAC tags verify, each user computes a public-key encryption of k_i under GA's public key, then group-signs this value, and finally symmetric-encrypts the signature with k_i in order to hide whether it verifies or not. The user broadcasts the encryption of k_i and the symmetric-encryption of it.

Tracing. The group manager can trace the users involved in a handshake by decrypting all the encryptions of the keys k_i, some of which might not be correct (in this scenario, some users cheat). If at least one user was honest, the group authority decrypts, using the keys k_i, the encryption of the signature and traces those back to all the handshake participants using its trapdoor τ.

2.2.2 GCD Security Problems. Although the GCD paper aims to prove the security of its scheme, this is not quite achieved, as we point out below and, in more detail, in Appendix A.1.

One of the most problematic aspects of the GCD construction is the fact that the key k_i is first encrypted, then used as the key to encrypt the group signature. This constructional aspect creates a circularity that makes proving security difficult under standard assumptions. Since k_i is meant to be later used in order to secure a channel between the handshake participants, this further complicates a proof.

Another potential problem is the construction of GCD based on generic MAC schemes. The paper indicates no specific security assumption for its MACs, and the standard security notion for MACs is existential unforgeability under chosen-message attacks. In GCD k_i is used to generate a MAC, and under standard unforgeability assumptions, we have no guarantee that k_i is still indistinguishable from random after this step, which jeopardizes further security steps (such as symmetric-encrypting with k_i).

Another component with insufficient security assumptions is the decentralized group key-agreement, which is only required to hide user identities. This is insufficient, since sessions of given users can be linked even when nothing leaks about the user's identifier.

Finally, note that users can bypass tracing by simply choosing not to encrypt the correct key k_i. Since fellow handshake participants cannot decrypt a PKE ciphertext generated with the group authority's public key. We describe further weaknesses of this scheme in Appendix A.1.

2.3 Our approach

Our approach is four-fold in this paper, as we review three important aspects of the initial GCD construction:

- A new primitive: ListMACs The problems caused by the public verifiability of GCD's group signatures led us to define and employ a new primitive: List Message-Authentication Codes (ListMACs). Intuitively, these are a symmetric-key variant of List signatures, which ensure that, if a single user tags two messages in quick succession, this can be detected – thus guaranteeing self-distinction for free.
- Clean key-derivation We replace the key-derivation mechanism in the secret handshake to ensure that different keys are used in different operations. To that purpose we use a two-step KDF, such as HKDF.
- A new banishment paradigm Traitor Catching Assume a user has misbehaved and their behaviour was flagged by

another group member. The group authority decides to ban that user. In most secret-handshake schemes, this leads to revealing the identity not just of the misbehaving user, but also those of the other handshake participants. In our approach, non-misbehaving users retain their privacy, while still allowing the banished users from being noticed at subsequent handshake attempts.
- Full formalizations/proofs An important disadvantage of the paper by Tsudik *et* Xu is that their design is not accompanied by proper formalization of security assumptions and proofs. In our paper, we remedy this and provide formalizations of both the primitives we use, and of the properties we achieve.

We begin by presenting the protocol that results from merely replacing group signatures by ListMACs and introducing a clean, purpose-specific key-derivation. We then assess the security of resulting scheme and finally describe a potential enhancement.

Building blocks. We use two main new building blocks in our protocol: ListMACs (which we present in section 3 and a key-derivation function like HKDF, consisting of two steps: the extraction of a high-entropy, short secret s from a low-entropy secret sk and a value *salt*: ($\mathsf{HKDF.Ext}(\mathsf{sk}; salt)$), and the expansion of that high-entropy secret to a number of keys, using a potential input value *input* and an output length ℓ ($\mathsf{HKDF.XP}(s; input, \ell)$). In our notation, we will often omit the length at the evaluation step.

Assumptions and environment. We consider the environment presented in subsection 2.1, with users $U_i \in \mathcal{U}$. In this environment, groups G also exist, each group being associated with a single group authority $\mathsf{GA} \in \mathcal{GA}$. The set of users $\mathcal{U}$ is disjoint from the set of group authorities $\mathcal{GA}$.

Users U_i can join and leave groups of their choice, and they can also be banned by the group authority. In parallel, users of various groups can run a secret handshake together, whose result is:

- **Common group:** If the users taking part in the handshake are part of one common group (and identify themselves, during the handshake, as belonging to that group), then the result is an overall accept of the handshake, culminating in the establishment of a group secure channel;
- **Non-consensus:** If at least one participant to the handshake has no group in common with all the others (or is a member of that group, but chooses not to identify as being a part of it), then the result is an overall reject of the handshake, and no secure channel will be established.

We take, seemingly, the same approach as Tsudik *et* Xu and allow group authorities to control both registration/access to the group and *catching* responsibilities.

3 ListMAC

Group signature schemes, as used in the GCD framework, can provide the anonymity within the group of a specific user. In [20], the traceability property is also used in order to attain non-frameability. However, group signatures also come with several disadvantages, including potentially cumbersome dynamic updates of the group's keys and an instant recognition of a valid group signature since the verification algorithm is public.

We take a different approach to constructing secret handshakes. Since our goal is to obtain secret handshakes which are moreover self-distinguishable, we want a means of group authentication that provides unlinkability, but also a way to match signatures produced in the same handshake by the same entity.

Thus, we propose the concept of *List MACs*, akin to List Signatures [8] but only allowing group members to verify the validity of the produced signature. List MACs essentially allow signatures to remain unlinkable up to two types of matching, which serve a dual purpose: tracking malicious users, and achieving self-distinction.

Definition 1 (ListMAC). *A ListMAC scheme is a tuple of algorithms* $\mathsf{LM} = (\mathsf{LM.Setup}, \mathsf{LM.GenGroup}, \mathsf{LM.RegUser}, \mathsf{LM.Tag}, \mathsf{LM.Ver},$
$\mathsf{LM.Match}, \mathsf{LM.Match}, \mathsf{LM.MatchSet})$ *as follows:*

- $\mathsf{LM.Setup}(1^\lambda) \rightarrow \mathsf{param}$ *generates system parameters* param*;*
- $\mathsf{LM.GenGroup}(\mathsf{param}) \rightarrow (\mathsf{gmk}, \mathsf{gvk}, \mathsf{bsn})$ *generates a group master key* gmk*, a group verification key* gvk*, and a value* bsn *standing for group's basename.*
- $\mathsf{LM.RegUser}(U, \mathsf{gmk}) \rightarrow (\mathsf{ID}_U^{\mathsf{LM}}, \mathsf{sk}_U) \sqcup \bot$ *allows a new user U to join the group by interacting with the group manager* GA*. The user outputs a private value* sk_U*, allowing the user to generate and verify ListMAC tags on behalf of the group. The user also outputs a pseudonym* $\mathsf{ID}_U^{\mathsf{LM}}$ *also known by* GA*, which will make the user matchable for specific values of the* aux *value input to the tagging algorithm.*
- $\mathsf{LM.Tag}(\mathsf{sk}, m, \mathsf{aux}, S) \rightarrow (\tau, \pi)$ *: given a secret key* sk*, a message m, an auxiliary value* aux*, and a set S containing tags, the algorithm returns a tag τ and a proof π attesting that the private key used to generate τ was not used in generating any value in S. The set S is composed of tuples of the form* (τ, aux).
- $\mathsf{LM.Ver}(\mathsf{gvk}, m, \mathsf{aux}, \tau) \in \{0, 1\}$*: the algorithm returns a verification bit as follows: if τ was issued by a group member, the algorithm outputs* 1 *(i.e., the tag is valid for* (m, aux)*); else it outputs* 0.
- $\mathsf{LM.Match}(\mathsf{gvk}, m, m', \mathsf{aux}, \tau, \tau') \in \{0, 1\}$*: this algorithm returns* 1 *if* (m, aux, τ) *and* $(m', \mathsf{aux}, \tau')$ *were issued by the same issuer, and* 0 *otherwise. More specifically,*

$$\begin{aligned}
&\forall \mathsf{aux}, \forall (m, m') \in \mathcal{M}^2, \mathsf{LM.Match}(\mathsf{gvk}, m, m', \mathsf{aux}, \tau, \tau') = 1 \\
&\qquad \wedge\ \mathsf{LM.Ver}(\mathsf{gvk}, m, \mathsf{aux}, \tau, \bot, \emptyset) = 1 \\
&\qquad \wedge\ \mathsf{LM.Ver}(\mathsf{gvk}, m', \mathsf{aux}, \tau', \bot, \emptyset) = 1 \\
&\iff \mathsf{sk} = \mathsf{sk}',
\end{aligned}$$

where $(\tau, \bot) \leftarrow \mathsf{LM.Tag}(\mathsf{sk}, m, \mathsf{aux}, \emptyset)$ *and* $(\tau', \bot) \leftarrow \mathsf{LM.Tag}(\mathsf{sk}', m', \mathsf{aux}, \emptyset)$.
Moreover, we design the algorithm so that $(m, \mathsf{aux}, \tau) = (\bot, \mathsf{bsn}, \mathsf{ID}^{\mathsf{LM}})$ *matches any* $(m', \mathsf{bsn}, \tau')$ *issued by user with ID* $\mathsf{ID}^{\mathsf{LM}}$.
- $\mathsf{LM.MatchSet}(\mathsf{gvk}, m, \mathsf{aux}, \tau, \pi, S) \in \{0, 1\} \sqcup \bot$ *: this algorithm returns* 1 *if any tag in S was issued by the same user producing a valid tuple* (τ, π) *for* (m, aux)*. More formally,*

given$\mathsf{sk}_U, (m, \mathsf{aux}, \tau, \pi), S$*:*

$$\begin{aligned} &\exists(\tau', \mathsf{aux}') \in S, (\tau', \bot) \in \mathsf{LM.Tag}(\mathsf{sk}_U, \cdot, \mathsf{aux}', \emptyset) \\ &\quad \land \mathsf{LM.Tag}(\mathsf{sk}_U, m, \mathsf{aux}, S) \ni (\tau, \pi) \\ &\iff \mathsf{LM.MatchSet}(\mathsf{gvk}, m, \mathsf{aux}, \tau, \pi, S) = 1 \end{aligned}$$

otherwise 0*. However if the* π *isn't valid it returns* $\bot$*.*

The Correctness of ListMACs requires several properties:

- Honestly-generated tags always verify.
- Tags honestly generated by the same user match.
- Tags honestly generated by a user can always be matched with tags produced by the same user and included in S.

3.1 Adversarial model

The security model for List MACs combines unforgeability properties with matching and unlinkability. The adversary interacts with the environment by using several *oracles*.

Generating groups and users. The adversary will use *oracle access* to generate honest or corrupt group authorities for honest, resp. corrupt groups and resp users.

- $\mathrm{oGenGroup}^b(G) \to (\bot, \mathsf{gvk}, \mathsf{bsn}) \sqcup (\mathsf{gmk}, \mathsf{gvk}, \mathsf{bsn})$: the oracle generates a group, for either an honest group authority (for $b = 0$) or a corrupt one ($b = 1$). For $b = 1$, GA is added to a list $\mathcal{CGA}$ of corrupted authorities, and the adversary learns the master secret material gmk. Otherwise, GA is added to $\mathcal{HGA}$, and the oracle outputs public group material gvk, bsn.
- $\mathrm{oCorruptGA}(G) \to (\mathsf{gmk}, \{\forall i, \mathsf{ID}_i^{\mathsf{LM}}\})$ corrupt the group authority GA of G, adding GA to $\mathcal{CGA}$.
- $\mathrm{oJoin}^b(G, U)$ creates either a honest user added to the set $\mathcal{HU}$ if $b = 0$ or a corrupted user add to $\mathcal{CU}$ (whose keys leaked to the adversary) if $b = 1$.

Created users can also be corrupted:

- $\mathrm{oCorruptUser}(G, U)$, if $U \in \mathcal{HU}$, provides the specific secret key of this user for the specific G. The adversary can control this party, and therefore U is added to $\mathcal{CU}$;

Interactions. Once groups and users are created, the adversary interacts with the environment via the following oracles.

- $\mathrm{oTag}(U, m, \mathsf{aux}, S) \to (\tau, \pi) \sqcup \bot$: this oracle only works for $U \in \mathcal{HU}$, running the tagging algorithm as a black box to generate a tag on message m for the auxiliary aux and given S.
- $\mathrm{oTagLoR}^b(\{U_L, U_R\}, m, \mathsf{aux}, S) \to (\tau, \pi)$: this oracle emulates tag generation for one of two users U_L or U_R depending on a bit $b \in \{L, R\}$.
- $\mathrm{oVer}(G, m, \mathsf{aux}, \tau) \to \{0, 1\}$ runs $\mathsf{LM.Ver}$ as a black box, for group parameter gvk.
- $\mathrm{oMatch}(G, m, m', \mathsf{aux}, \tau, \tau')$ runs $\mathsf{LM.Match}$ as a black box with the corresponding G's gvk.
- $\mathrm{oMatchSet}(G, m, \mathsf{aux}, \tau, \pi, S) \to \{0, 1\} \sqcup \bot$ runs $\mathsf{LM.MatchSet}$ as a black box with the corresponding G's gvk.

We set $\mathcal{O}_{\mathsf{LM}}$ as being the set containing all the oracles defined for ListMAC.

```
$\mathsf{Exp}_{\mathsf{ListMAC}}^{EUF-CMA-AD}$
$\mathsf{param}_{\mathsf{ListMAC}} \leftarrow \mathsf{ListMAC.Setup}(1^\lambda)$
$(G, m, \mathsf{aux}, \tau, \pi, S) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{LM}} \setminus \{\mathrm{oTagLoR}\}}(\mathsf{param}_{\mathsf{ListMAC}})$
$\mathcal{A}$ wins $\iff \mathrm{oVer}(g.G, m, \mathsf{aux}, \tau) = 1 \land G.\mathrm{GA} \in \mathcal{HGA}$
$\land \mathbb{Q}^{\mathrm{oJoin}^1(g.G, \cdot)} = \emptyset \land \mathbb{Q}^{\mathrm{oCorruptUser}(g.G, \cdot)} = \emptyset$
$\land [\ \pi \neq \bot \implies \mathrm{oMatchSet}(g.G, m, \mathsf{aux}, S, \tau, \pi) = 1 \land S \neq \emptyset\ ]$
$\land [\ \forall t \in \mathbb{Q}^{\mathrm{oTag}},$
    $[t.S = S \implies t.m \neq m] \lor [t.m = m \implies t.S \neq S \land S \neq \emptyset]\ ]$
```

Figure 1: ListMAC Existential Unforgeability against Chosen Message Attacks with Auxiliary Data (EUF-CMA-AD) game

```
$\mathsf{Exp}_{\mathsf{ListMAC}}^{\mathsf{Unlink}}$
$b \xleftarrow{\$} \{L, R\}$
$\mathsf{param}_{\mathsf{ListMAC}} \leftarrow \mathsf{ListMAC.Setup}(1^\lambda)$
$d \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{LM}} \setminus \{\mathrm{oCorruptUser}\}}(\mathsf{param}_{\mathsf{ListMAC}})$
$\mathcal{A}$ wins $\iff b = d$
$\land \exists! h \in \mathbb{Q}^{\mathrm{oGenGroup}}, \forall e \in \mathbb{Q}^{\mathrm{oTagLoR}^b},$
    $e.\mathsf{aux} \neq h.\mathsf{bsn} \land \{e.U_L, e.U_R\} \subset (\mathcal{HU} \cap \mathsf{USet}_{h.G})$
$\land \forall f \in \mathbb{Q}^{\mathrm{oTagLoR}^b} \setminus \{e\},$
    $[\ \{e.U_L, e.U_R\} \cap \{f.U_L, f.U_R\} \neq \emptyset$
    $\implies e.\mathsf{aux} \neq f.\mathsf{aux} \land (f.\tau, f.\mathsf{aux}) \notin e.S\ ]$
$\land \forall g \in \mathbb{Q}^{\mathrm{oTag}},$
    $[\ g.U \in \{e.U_L, e.U_R\}$
    $\implies e.\mathsf{aux} \neq g.\mathsf{aux} \land (g.\tau, g.\mathsf{aux}) \notin e.S \land (e.\tau, e.\mathsf{aux}) \notin g.S\ ]$
$\land \forall s \in g.S \cup e.S \cup f.S,$
    $[\ \exists! a \in \mathbb{Q}^{\mathrm{oTag}}, (a.\tau, a.\mathsf{aux}) = (s.\tau, s.\mathsf{aux})\ ]$
    $\oplus [\ \exists! l \in \mathbb{Q}^{\mathrm{oTagLoR}^b}, (l.\tau, l.\mathsf{aux}) = (s.\tau, s.\mathsf{aux})\ ]$
```

Figure 2: ListMAC Unlinkability game

3.2 Security Notions

In the unforgeability game, the adversary's task is to forge a tag for a fresh message on behalf a group not containing any corrupted users. This definition is described below.

In the unlinkability game, the adversary seeks to link two signatures without using the matching functionality. This game is formally described below.

The non-frameability game concerns matching. While signatures created using the same private keys should be matchable, no one should be able to create a tag that matches to that of an honest user (either through tag-to-tag matching, or through tag-to-set matching).

Definition 2 (EUF-CMA-AD). *Consider a* ListMAC *instance* ListMAC*. For a probabilistic polynomial-time (PPT) adversary* $\mathcal{A}$ *let* $\mathsf{Adv}_{\mathsf{ListMAC}}^{\mathsf{EUF-CMA-AD}}(\mathcal{A})$ *be its advantage to win the* $\mathsf{Exp}_{\mathsf{ListMAC}}^{EUF-CMA-AD}$ *security game (cf.Figure 1):*

$$\mathsf{Adv}_{\mathsf{ListMAC}}^{\mathsf{EUF-CMA-AD}}(\mathcal{A}) = \frac{1}{1-\epsilon}\left|\Pr[\mathcal{A}\ wins\, \mathsf{Exp}_{\mathsf{ListMAC}}^{EUF-CMA-AD}] - \epsilon\right|$$

where ϵ *is negligeable compare to* λ*.*

```
$\mathsf{Exp}^{\mathsf{NF}}_{\mathsf{LM}}$
$\mathsf{param}_{\mathsf{ListMAC}} \leftarrow \mathsf{ListMAC.Setup}(1^{\lambda})$
$(G, m, \mathsf{aux}, \tau, \pi, S) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{LM}} \setminus \{\mathsf{oTagLoR}\}}(\mathsf{param}_{\mathsf{ListMAC}})$
$\mathcal{A}$ wins $\iff \mathsf{oVer}(G, m, \mathsf{aux}, \tau) = 1$
$\wedge \forall s \in S, \exists! q \in \mathbb{Q}^{\mathsf{oTag}}, (s.\tau, s.\mathsf{aux}) = (q.\tau, q.\mathsf{aux})$
$\wedge$ [ [$\mathsf{oMatchSet}(G, m, \mathsf{aux}, \tau, \pi, S) = 1$
        $\implies q.U \in \mathcal{HU} \wedge \mathbb{Q}^{\mathsf{oTag}(q.U,\cdot,\cdot,S)} = \emptyset$]
    $\vee$[$\mathsf{oMatchSet}(G, m, \mathsf{aux}, \tau, \pi, S) = 0 \implies q.U \in \mathcal{MU} \cup \mathcal{CU}$]
    $\vee$[ $\exists H \in \mathcal{HU}, \forall m' \in \mathcal{M}$,
        $\exists(\tau', \perp) \in \mathsf{oTag}(H, m', \mathsf{aux}, \emptyset)$,
        $\mathsf{oMatch}(G, m, m', \mathsf{aux}, \tau, \tau') = 1$ ] ]
```

Figure 3: ListMAC NF game : $\mathcal{A}$ isn't able to forge a tag that can be linked to an honest user

We call ListMAC *EUF-CMA-AD-secure if, and only if, any probabilistic polynomially bounded adversary $\mathcal{A}$ against* ListMAC *has at most an advantage is negligible compare to the security parameter λ.*

Definition 3 (Unlinkability). *Consider a ListMAC scheme* ListMAC. *The advantage of a PPT adversary$\mathcal{A}$ to win $\mathsf{Exp}^{\mathsf{Unlink}}_{\mathsf{ListMAC}}$ (cf. Figure 2) is defined as follows:*

$$\mathsf{Adv}^{\mathsf{Unlink}}_{\mathsf{ListMAC}}(\mathcal{A}) = 2\left|\Pr[\mathcal{A}\ wins\ \mathsf{Exp}^{\mathsf{Unlink}}_{\mathsf{ListMAC}}] - \frac{1}{2}\right|$$

We call ListMAC *Unlinkable if, and only if, any probabilistic polynomially bounded adversary $\mathcal{A}$ against* ListMAC *has at most an advantage is negligible compare to the security parameter λ.*

Remark 1. *(cf. Figure 2) Some may argue that the* Unlinkability *is a weakened game since we do not englobe the whole logical possibilities; in other words, we avoided in the winning conditions the logical predicate which states that the $\mathcal{A}$ is allowed to run* oTagLoR *with a set S containing U_L's and U_R's tag each in the S. For the sake of generality, since we may build a specific instance which doesn't give us only* 'if the user match a tag in a set'*, but also* 'which tag the user matches'. *Also it is possible to build a sequence of call to* oTag *in order to obtain the same information even without the* 'which tag the user matches'*, the proof is given in Proposition 2.*

Definition 4 (Non-Frameable). *The advantage of a PPT adversary $\mathcal{A}$ to win $\mathsf{Exp}^{\mathsf{NF}}_{\mathsf{ListMAC}}$ (cf. Figure 3) against a ListMAC scheme* ListMAC*, is defined as follows:*

$$\mathsf{Adv}^{\mathsf{NF}}_{\mathsf{ListMAC}}(\mathcal{A}) = \frac{1}{1-\epsilon}\left|\Pr[\mathcal{A}\ wins\ \mathsf{Exp}^{\mathsf{NF}}_{\mathsf{ListMAC}}] - \epsilon\right|$$

where ϵ is negligeable compare to λ.

We call ListMAC *EUF-CMA-AD-secure if, and only if, any probabilistic polynomially bounded adversary $\mathcal{A}$ against* ListMAC *has at most an advantage is negligible compare to the security parameter λ.*

Theorem 1.

$$\forall \mathcal{R}, EUF-CMA-AD(\mathcal{R}) \implies \forall \mathcal{A}, \mathsf{NF}(\mathcal{A})$$

Proof. We demonstrate the following statement:

$$\exists \mathcal{A}, \neg\mathsf{NF}(\mathcal{A}) \implies \exists \mathcal{R}, \neg EUF-CMA-AD(\mathcal{R})$$

Let suppose an adversary $\mathcal{A}$ that wins $\mathsf{Exp}^{\mathsf{NF}}_{\mathsf{LM}}$. We build a reduction $\mathcal{R}$ that plays against $EUF-CMA-AD$, and let C be the $\mathcal{R}$'s challenger. Note that $\mathcal{R}$ simulates perfectly oracle queries, since $\mathcal{R}$ conveys the to C. Therefore the $\mathcal{A}$ can call oJoin as many time as it wishes, and frame a user then it sends the tag to $\mathcal{R}$ that conveys to C. Hence proving that the statement is correct. □

Lemma 1.

$$\mathsf{Adv}^{\mathsf{NF}}_{\mathsf{LM}} \leq \mathsf{Adv}^{EUF-CMA-AD}_{\mathsf{LM}}$$

4 FURTHER BUILDING BLOCKS

The GCD framework of Tsudik and Xu [20] required the use of three components: a group-signature scheme, a central secure channel, and a distributed (non-authenticated) group key-agreement. In this work, we replace group signatures by the ListMAC primitive, presented above. In the spirit of our approach, described in Section 2.3, we propose and formalize below the following building blocks:

- Centralized Broadcast and User Unicast(CBU2): this extension of the central secure channel of Tsudik and Xu will allow a group manager to communicate securely in broadcast mode, and each user to unicast separately to the group manager. We use this primitive in the secret handshake construction for two main purposes: the broadcast will allow group members to recognize each other; while the unicast will allow for the detection of malicious users.
- Anonymous Group Key-Agrement with Fresh Randomness(AGKA-FR): this extension of the distributed group key-agreement by Tsudik and Xu will allow a group of users to compute a master secret without revealing any information about their identity (sessions are unlinkable). In addition, we formalize a functionality requirement made informally by [20], specifically the presence of some information that allows the AGKA-FR session to be bound to that of the subsequent secret handshake.

4.1 Centralized broadcast and User Unicast (CBU2)

In our construction, we collapse the centralized secure-channel with the mutually-authenticated secure channels from managers to users into a single, complex element, allowing group authorities to broadcast to a multitude of users, but also user-to-central-party unicast, for the purposes of allowing users to leave or be banned.

We call this a Centralized-Broadcast-and-User-Unicast channel (in short CBU2). The full formal description of this primitive is provided in Appendix C.4.3, but we give a summary of it below.

Intuitively, CBU2 features a global setup algorithm outputting global private (spar) and public (ppar) parameters. Given the private parameters, so-called channel managers can use long-term credentials to register users. The managers can add and remove users from the channel (see algorithms `UAdd` and `URmv`), or trigger updates of session keys (algorithm `ChUpdate`).

Messages can be sent via broadcast (channel manager to users) or unicast (user to channel manager). Sending and receiving of broadcasts is done via the `BCast` and respectively `RecBCast` algorithms. While users cannot use the broadcast channel, they can unicast to the central manager (algorithms UCast and RecUCast) in a secure way with respect even to other legitimate CBU2 users.

A full description of the CBU2 syntax is in Appendix C.4.3.

Notice that while our syntax features global private parameters given to the channel managers, the latter still require private keys to set up and manage new channels. Users may additionally only join channels for which they are registered with the managers. Channel management is dynamic and takes place in installments (epochs). Only channel managers can trigger updates, and they do so when users are added or removed. Each modification triggers an update of key-material and user lists and can be perceived as being akin to the notion of "epoch" in secure-channel establishment with post-compromise security. To avoid confusion, we abuse vocabulary and call these installments "time", indexed by a discrete variable t. The phrase "the user group $\mathcal{UG}_{\mathsf{sid}}$ at time t" thus refers to the value of $\mathcal{UG}_{\mathsf{sid}}$ at the t-th installment of the group management process.

The CBU2 channel allows channel managers to broadcast using ms values, and users to unicast using sendK keys. The session identifier sid and the broadcast key material evolve upon adding or removing users from a channel session, or updating the key material. The same may apply to unicast keys, but this is not compulsory.

Channel manager and user states. Both users and channel managers maintain state, consisting of:

- Their long-term private and secret keys sk, pk
- (Users) A list ChList of channels they are registered to.
- (Managers) The database $\mathcal{UG}$ of all registered users, with respective keying materials. For each session of the channel, $\mathcal{UG}_{\mathsf{sid}}$ stores the up-to-date session state of sid (with its keys per epoch).
- The current epoch they are on, in each session, t_{sid}.
- Bits indicating the acceptance state of users.
- Lists of sent and received messages at each epoch, denoted respectively Snd and Rcv.
- Lists of keys KeyList for all sessions the users/channel managers are involved in, at various epochs.

Correctness. The definition of correctness for CBU2 is provided in Appendix C.4.3. Note that we require both a narrower and a broader notion of correctness than in typical multi-stage authenticated key-agreement, since CBU2 is centralized (key-updates are triggered only by the channel manager, acceptance/key computation on the channel manager will imply acceptance/key computation for the channel users), features some key-updates for the broadcast channel, but also includes a more classical end-to-end secure unicast channel with no key updates.

Adversary Model. We define the security of the CBU2 primitive in terms of authentication and security of the established channel (akin to the notion of (S)ACCE, originally introduced in the context of TLS 1.2 in [13]). Our choice is motivated by the fact that in the construction of secret handshakes, the property we require from the CBU2 is that nonces encrypted and sent through broadcast are indistinguishable from random, and tags sent through unicast are similarly indistinguishable from random. While this property can be achieved by the careful composition of AKE-secure forward-secure key exchange and authenticated-encryption, achieving the equivalent of ACCE security is not always immediate.

We will require the following properties:

- Centralized broadcast: We require the authentication of sessions, updates, and transmissions (messages should only originate with the channel manager), and the security of the broadcast channel, specifically:
  - **Authentication:** an attacker without access to the channel manager's private values cannot make a non-malicious user accept a message not sent at that epoch by the channel manager.
  - **Security:** an attacker without access to the session key at epoch t cannot break the ACCE security of a broadcast message, *i.e.,* only users legitimately in possession of the epoch's keys may distinguish a transmitted message from random.
- User unicast: We require the authentication of transmissions (the user is authenticated), and the security of the unicast channel, specifically:
  - **Authentication:** an attacker without access to an honest user's private values cannot make a channel manager accept a message not set at that epoch by that user.
  - **Security:** an attacker without access to the user's key at that epoch cannot break the ACCE security of unicast messages, *i.e.,* only the user and channel manager can distinguish from random the exchanged plaintexts.

The full security definitions are provided in Appendix C.4.3.

Insight: constructing CBU2**.** At its core, the CBU2 channel consists of a manager-to-users broadcast channel with key-evolution, and multiple user-to-manager unicast channel which does not necessarily have to feature key-evolution. In both cases, confidentiality must be ensured, and the authentication property demands that the communication only run one-way, which requires the use of EUF-CMA-secure authentication with non-repudiation: typically signature schemes.

There are many ways to construct such channels.

A typical start would be to provide broadcast communication via a group-communication channel with post-compromise security, such as MLS, combined with a signature scheme that would allow only the channel manager to effectively send messages. Note, however, that the functionality required here differs a little from the standard MLS architecture. For one thing, the only entity that will be proposing the addition or removal of users is the group manager. In addition, since we will be using the CBU2 protocol in the interest of a privacy-preserving scheme, note that the true identities of the users in the group will not be known (we will be using channel-specific identifiers within the Secret Handshake scheme). This partially violates one of the core MLS properties: the fact that users are aware who is in the group. No users will be able to make proposals or commits to the channel. Key-updates are also only triggered by the channel manager.

We note that the unicast channel key-material could be derived, via a secure PRF, from the group secrets at the epoch at which the user has joined as well as a nonce known only to the manager and the user, chosen uniformly and independently at random during the Joining procedure. The derivation needs to preserve certain security properties, but could essentially work as described by

Brzuska, Jacobsen, and Stebila in [7]. In this case as well, we would require the user to sign each sent message.

4.2 Anonymous group key-exchange (AGKA-FR)

Apart from CBU2, another critical component in the framework of Tsudik and Xu [20] is Distributed Group Key-Agreement (DGKA), meant to allow several users to compute a set of common secret keys without betraying their identities. Unfortunately, the basic DGKA syntax described in [20] only models limited privacy, which is insufficient to guarantee the strong privacy properties required by secret handshakes.

In our framework, we use *anonymous* group key-agreement with an additional feature: each honest user will employ fresh randomness during every session. We call our protocol Anonymous Group Key-Agreement with Fresh Randomness (AGKA-FR), and it has two main properties: the indistinguishability from random of session keys, and the *unlinkability* of protocol sessions. Due to space limitations, we include in Appendix C.4.4 the lion's share of the formalization, only including here a characterization that will help understand our secret handshake protocol.

The AGKA-FR primitive will be parametrized by a randomness superspace $\mathcal{R}$, essentially characterizing the randomness that is used (*e.g.,* group element, integer, etc.). Then, during AGKA-FR setup, a subset $\mathcal{R}^{\Pi}$ of $\mathcal{R}$ is chosen to become the set of randomness that is used in practice (*e.g.,* nonces of 128 bits). Honest protocol participants will abort AGKA-FR in case of colliding randomness[7].

In our secret handshakes, users first undergo basic group key-agreement. We need to bind information about that key-agreement later, when users employ list MACs to authenticate. We require two properties: each user in the handshake has a unique input during key-agreement, and the concatenation of the randomness of all users is in turn unique. For an in-depth discussion of our design choices and some alternatives, please see Appendix C.4.4.

AGKA-FR **syntax and environment.** We consider a set of users USet, each user associated with an identity U_i. The protocol is defined as a tuple of the following algorithms.

- $\mathsf{AGKA\text{-}FR.Setup}(1^{\lambda}, \mathcal{R}) \rightarrow (\mathsf{ppar}, \mathcal{K}, \mathcal{R}^{\Pi})$: The global setup algorithm outputs the subset of usable randomness $\mathcal{R}^{\Pi}$ used for the randomness, as well as a (surjective) set of possible keys and public parameters ppar.
- $\mathsf{AGKA\text{-}FR.Handshake}(\Delta) \rightarrow (\{\mathsf{state}_i, \mathsf{sid}, \mathsf{ms}_i\}_{U_i \in \Delta}$: This interactive handshake algorithm allows users in a set Δ to run the protocol, outputting: unique randomness $\mathsf{state}_i \in \mathcal{R}^{\Pi}$, a session identifier sid which is the concatenation of all state_i from smallest to largest; and a master secret ms.

Users U_i store a table AGKA-FR.SList indexed by session identifiers sid and containing: the number $\mathsf{n}_{\mathsf{sid}}$ of users in session sid, the randomness state the user used in sid, the user's acceptance bit α_{sid} for the correctness of the randomness generated by all the users during sid, the user's acceptance bit α of the protocol session, the key computed by the user ms, and a reveal bit ρ indicating whether the key has been revealed.

We require a complex notion of correctness: the honest use of correct and non-colliding randomness state (otherwise, users abort), and the computation of the same key by users taking part in the same handshake.

Security notions for AGKA-FR**.** Intuitively, AGKA-FR should guarantee the following properties: anonymity and the security of the computed keys. We define anonymity in terms of the unlinkability between two participating users, and prove that this property also implies a type of *simulatability, i.e.,* it is impossible to tell whether the protocol participant is a real user or a simulator.

Since users are not associated with private values, unlike in our secret handshake and CBU2 protocols, for AGKA-FR it makes no sense to consider corrupt users: just honest and malicious ones. The adversary can register either of these and allow them to interact in protocol sessions. In unlinkability, the adversary aims to link sessions of the same party. The security definition is a typical AKE left-or-right notion, in which the adversary aims to distinguish the real key, computed by an honest party interacting with other honest parties, from a random key from the keyspace.

5 SECRET HANDSHAKES FROM LIST MACS

Our protocol LCA corrects, improves, and extends the GCD framework of Tsudik and Xu [20]. We preface its presentation by a short description of the intuition behind it, then present the scheme itself.

5.1 Intuition

Our scheme makes use of the three main building blocks presented in Sections 3 and 4, and which give it its name: LCA for ListMAC-CBU2-AGKA-FR. The goal is to provably provide the cornerstone properties of Secret Handshake schemes, notably authentication, unlinkability, handshake-simulatability, result-hiding, non-frameability, and a measure of traceability. While GCD achieves a strong form of traceability, which leaks all the participants of a handshake in case of a dispute, we prefer a more privacy-friendly approach, in which misbehaving users are caught only if they use their credentials after misbehaving. We argue that our approach provides the same security/accountability guarantees, but at less cost to privacy.

At the core of our construction is a means of providing anonymous mutual authentication between group members, which remains undetectable and simulatable. This is achieved by a combination of several features:

- Group members establish a CBU2 session over which the group authority provides a fresh nonce at each epoch. Whenever group members are added, leave, or the channel is updated, the CBU2 key material is renewed, and a new nonce is broadcast. No one but current group members know the nonce.
- Secret handshake participants run an initial AGKA-FR session. This session yields key material that only participants will know, as well as public, but unique randomness bound to each participant.

[7] Note that, since most key-agreement protocols rely on the use of some randomness, restricting ourselves to schemes explicitly using such values is not a big restriction.

- ListMACs are computed by each member of the group that is present in the handshake, and each MAC is bound to the AGKA-FR execution. Moreover, ListMAC verification can only be achieved by members of the group, which helps the result-hiding property. By its matching properties, ListMACs help LCA achieve self-distinction.

Our protocol LCA takes place on 3 main fronts: group management, handshake execution, and tracing/banning. We briefly discuss each of these before proceeding to the protocol description.

Group management. Each group is run by a group authority, which establishes a unique CBU2 channel which runs permanently in the background. Whenever a user joins, it interacts with the group authority to obtain ListMAC credentials. The fact that the group authority does not know the user's ListMAC key is essential to providing non-frameability. Every time new users join or leave, the group manager updates the CBU2 session, computing new key materials and broadcasting a fresh nonce.

Handshakes. The handshake starts with a AGKA-FR session between protocol participants. The AGKA-FR anonymity is crucial here, as we want no information to be leaked about the participants: we just want the users present at the handshake to establish some common key material. We recall that AGKA-FR also compels participants to use fresh randomness, which is unique per party and per session. This binding material from AGKA-FR is part of a message tagged (using ListMAC) by each participant. Finally, using the key material derived from AGKA-FR, the user cleanly computes (through the judicious use of export keys and Extract-and-Expand PRFs) a key with which the tag is encrypted. In short, the encryption renders the value simulatable, while the ListMAC allows members of the same group to recognize each other.

ListMACs come with multiple perks. Given a verifiable ListMAC, users from the same group can ensure that Self-Distinction is guaranteed. Moreover, it becomes easy to check if any handshake participants are users banned from the group. Finally, the computation of a second ListMAC tag provides non-frameability.

Tracing/banning. Our scheme supports both users voluntarily leaving groups (via Leave) and users being banned against their will if they misbehave. User banishment is essentially ensured by the use of ListMACs. Banned users will have tags generated by them added to a special set of banned tags. During handshakes, users match received tags against the set of banned tags of their group, thus detecting potentially banned users. If a user misbehaves and the group authority is provided with that user's tag from a handshake, the authority uses CBU2 in order to request, from each user, a tag that will either match (in the case of the guilty party) or not (for innocent parties) the tag provided during the handshake[8].

We summarize in Table 2 the properties that our scheme inherits from its building blocks.

Intuition: multiple groups. Notice that users may actually be members of several groups simultaneously. The question then is which group they are meant to represent in each handshake. In this paper, we simplify such users' choice and note that users will simply draw at random one group identity amongst all their groups, and they will decide to run the protocol as a member of that group. We leave the question of how to optimize this process – which is not characteristic to our own scheme, but rather, a general problem in secret handshakes – as further work.

Table 2: How our building blocks provide the construction's properties: Unlinkability, Authentication, Self-Distinction, Handshake-Simulatability, Result-hiding, Non-Frameability, Full-Unlinkability, and Traitor-Catching: ✓: building block useful, ✗: building block prevents property.

	Unlink	Auth	S-Dist	Res-Hide	Hand-Sim	NF	F-Unlink	Catch
AGKA-FR	✓			✓	✓		✓	
CBU2		✓			✓	✓		✓
ListMAC	✓	✓	✓			✓	✗	✓

5.2 Proposed protocol

In this section we describe our LCA scheme in more detail. Due to space limitations, an even more detailed and formal description is provided in Appendix C.2.

Global setup: SHS.Setup:. This step is used to run the global setup of our main primitives, providing parameters $\mathsf{param}_{\mathsf{ListMAC}}$ for ListMACs, $\mathsf{param}_{\mathsf{CBU2}} = (\mathsf{spar}_{\mathsf{CBU2}}, \mathsf{ppar}_{\mathsf{CBU2}})$ for CBU2, and $\mathsf{param}_{\mathsf{AGKA\text{-}FR}}$ and the randomness space $\mathcal{R}^{\Pi}$ for the AGKA-FR.

Group creation: SHS.NewGroup:. The group authority GA creates a new group G, generates credentials for a new CBU2 channel, and generates ListMAC master and verification parameters for ListMACs (including a basename bsn used in identification). Initially there are no banished users. The GA creates a random $\mathsf{sid}_{\mathsf{ban}}$ (later used for catching and banning misbehaving users).

Channel update: SHS.Update:. If, for whatever reason, the group authority wants to renew CBU2 material, it runs an update algorithm, which executes CBU2.ChUpdate as a black box, thus renewing the key material of all the users. It then generates and sends a fresh nonce over the broadcast channel, and sends two current banned-user lists: that of voluntarily-leaving users KRL and that of banned users SRL. Updates are specifically also used whenever users join and leave the group.

New user joining: SHS.Join:. This step first starts with a registration step: GA interacts with the new user U on a secure channel in order to generate, together, the user's ListMAC secret key (known only to U) and its ListMAC identity $(\mathsf{ID}^{\mathsf{LM}}, \mathsf{sk}_U)$ (known to GA and U and stored by GA in a list DB). The group authority also gives U the verification key gvk and the value $\mathsf{sid}_{\mathsf{ban}}$. After successful ListMAC registration, U is registered, then added to the CBU2. If U is the first group member, CBU2 needs to first be instantiated. Else, the CBU2 session is updated and all users, including U receive new key material.

[8] The ListMACs nifty ability to match tags provided by the same user also comes into play if the guilty party tries to avoid detection by using the credentials of a corrupted party.

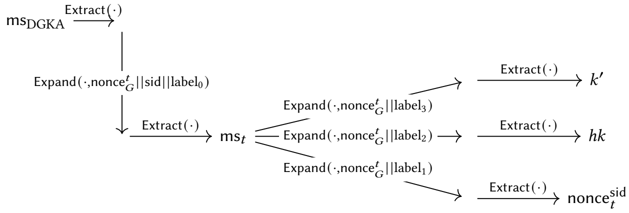
Thus, at the end of this step, U possesses ListMAC private and verification material, as well as broadcast secrets (in order to receive GA's CBU2 broadcasts) and unicast keys (in order to unicast to GA). Finally, the user receives from GA the updated nonce and the two lists of banned users KRL, SRL.

User leaving: SHS.Leave**:.** When users leave voluntarily, they unicast their ListMAC private keys over CBU2, and the latter is added to KRL. We note that a variant that achieves even better privacy involves the leaving user providing a tag that can be added to SRL. This ensures the user can no longer produce ListMAC tags on behalf of the group, but unfortunately does not take them off the CBU2. To identify which CBU2 credentials correspond to each users, GA chooses a random m and broadcasts Leave$||m$. Leaving users unicast a tag on that message, which will be identified and linked, thus allowing GA to sever the CBU2 connection. This tag is added to SRL. Finally, GA performs the updating procedure for the updated KRL, SRL) tuple.

Secret handshake: SHS.Handshake**:.** As described in the intuition, a secret handshake consists of several steps: the users first run AGKA-FR, using fresh randomness from $\mathcal{R}^{\Pi}$; then users authenticate by tagging their randomness, the session transcript, and a nonce derived from the current group randomness; this tag is encrypted, and the ciphertext broadcast; users verify all received tags and, if they all verify as valid (*i.e.,* they came from other group members), they match received tags to check for double tagging (to get self-distinction) and banned users; users compute a second tag over entire session transcript, to ensure non-frameability.

Note that at each step, failures can occur: in AGKA-FR users could notice colliding randomness, or perhaps users notice at least one other handshake participant not being in the same group. In all failure cases, the user will run a simulation of the handshake, producing random outputs of the correct format and length.

The protocol is described in Figure 4, with more details in Appendix C.2. An essential part of it is the key-schedule, allowing users to compute, through the use of HKDF [15], a number of indepdent key, by taking in input common secrets, but distinct labels, specifically (in order from label$_0$ to label$_3$): "`separation`", "`generate nonce`", "`generate hiding key`", "`generate session key`" These strings will serve as distinguishing labels in order to obtain independent key values from the same secret.


$\mathsf{ms}_{\mathrm{DGKA}}$
Extract(·)
Expand(·,nonce_G^t||sid||label$_0$)
Extract(·)
ms_t
Expand(·,nonce_G^t||label$_3$)
Expand(·,nonce_G^t||label$_2$)
Expand(·,nonce_G^t||label$_1$)
Extract(·)
k'
Extract(·)
hk
Extract(·)
$\mathsf{nonce}_t^{\mathsf{sid}}$

Pseudo Random Key	Salt	Input Key Material
k'	ms_t	$\mathsf{nonce}_G^t$$||$label$_3$
hk	ms_t	$\mathsf{nonce}_G^t$$||$label$_2$
$\mathsf{nonce}_t^{\mathsf{sid}}$	ms_t	$\mathsf{nonce}_G^t$$||$label$_1$
ms_t	$\mathsf{ms}_{\mathsf{AGKA\text{-}FR}}$	$\mathsf{nonce}_G^t$$||$KShare$||$label$_0$

Figure 5: KeySchedule**: description**

```
$U_A$
Get $(\mathsf{gvk}, \mathsf{sk}_A, \mathsf{KRL}, \mathsf{SRL}, \mathsf{nonce}_G^t)$ //Current group parameters
//Establish a common secret amongs users in $\Delta$
$(\mathtt{state}_A, \mathtt{sid}, \mathsf{ms}_{\mathsf{AGKA\text{-}FR}}) \leftarrow \mathsf{AGKA\text{-}FR.Handshake}(\Delta)$
//Compute keys $k'$, $hk$, randomness $\mathsf{nonce}_t^{\mathsf{sid}}$
$(k', hk, \mathsf{nonce}_t^{\mathsf{sid}}) \leftarrow \mathtt{KeySchedule}(\mathsf{ms}_{\mathsf{AGKA\text{-}FR}}, \mathsf{nonce}_G^t, \mathsf{sid})$
if $\mathsf{sid} = \mathsf{bsn}$ abort and simulate
//Compute tag and proof non-banned user.
$(\tau_A, \pi_A^{\mathsf{SRL}}) \leftarrow \mathsf{LM.Tag}(\mathsf{sk}_A, H(\mathsf{state}_A||\mathsf{nonce}_t^{\mathsf{sid}}), \mathsf{sid}, \mathsf{SRL})$
//Encrypt and send tag
$c_A^\tau \leftarrow \mathsf{SEnc}(hk, (\mathsf{state}_A, \tau_A))$
                                                        $\xrightarrow{c_A^\tau}$
                                                        $\xleftarrow{\{c_*^\tau\}}$
//Verify all tags, check matches for Self-Distinction
$D \leftarrow \mathsf{sid}$ //Recall sid is concatenation of state in AGKA-FR
foreach $i \in [[\#\Delta]]$
    $(\mathsf{state}_i, \tau_i) \leftarrow \mathsf{SDec}(hk, c_i^\tau)$
    if $\mathsf{LM.Ver}(\mathsf{gvk}, H(\mathsf{state}_i||\mathsf{nonce}_{\mathsf{sid}}^t), \mathsf{sid}, \tau_i) = 0 \lor \mathsf{state}_i \notin D$
        Abort and simulate remainder of protocol
    $D \leftarrow D\backslash < \mathsf{state}_i >$
//$E$ contains a generation of tags for sid that are issued from KRL
$E \leftarrow \{(0, m', \tau') : \forall \mathsf{sk}' \in \mathsf{KRL}, \tau' \leftarrow \mathsf{LM.Tag}(\mathsf{sk}', m', \mathsf{sid}, \emptyset)\}$
if $\mathtt{DetectSelfDistinction}(\mathsf{gvk}, \mathsf{sid},$
$E \cup \{\mathsf{state}_*, H(\mathsf{state}_i||\mathsf{nonce}_t^{\mathsf{sid}}), \tau_i)\}_{i\in[[n]]}) \neq \emptyset$
    abort and simulate
$c_A^\pi \leftarrow \mathsf{SEnc}(hk, (\mathsf{state}_A, \mathsf{pad}(\pi_A^{\mathsf{SRL}})))$
                                                        $\xrightarrow{c_A^\pi}$
                                                        $\xleftarrow{\{c_*^\pi\}}$
Decrypt all $c_*^\pi$, associate each $c_*^\pi$ with AGKA-FR nonces
//Check for banned users
foreach $i \in [[n]]$
    if$0 \neq \mathsf{LM.MatchSet}(\mathsf{gvk}, H(\mathsf{state}_i||\mathsf{nonce}_t^{\mathsf{sid}}), \mathsf{sid}, \tau_i, \pi_i^{\mathsf{SRL}}, \mathsf{SRL})$
        Abort and simulate remainder of protocol
//Compute second tag for NON-FRAMEABILITY from the accumulator
$\mathsf{acc} \leftarrow H(\mathsf{nonce}_t^{\mathsf{sid}}||\tau_1||\mathsf{state}_1||\ldots||\tau_n||\mathsf{state}_n)$
$(\tau_A^{\mathsf{acc}}, \perp) \leftarrow \mathsf{LM.Tag}(\mathsf{sk}_A, H(\mathsf{state}_A||\mathsf{acc}), \mathsf{sid}, \emptyset)$
$c_A^{\mathsf{acc}} \leftarrow \mathsf{SEnc}(hk, (\mathsf{state}_A, \tau_A^{\mathsf{acc}}))$
                                                        $\xrightarrow{c_A^{\mathsf{acc}}}$
                                                        $\xleftarrow{\{c_*^{\mathsf{acc}}\}}$
foreach $i \in [[n]]$
    Associate to the corresponding $\tau_i$ based on the $\mathsf{state}_i$
    if $\mathsf{LM.Ver}(\mathsf{gvk}, H(\mathsf{state}_i||\mathsf{acc}), \mathsf{sid}, \tau_i^{\mathsf{acc}}) = 0$
    $\lor \mathsf{LM.Match}(\mathsf{gvk}, H(\mathsf{state}_i||\mathsf{acc}), H(\mathsf{state}_i||\mathsf{nonce}_t^{\mathsf{sid}}), \mathsf{sid},$
    $\tau_i^{\mathsf{acc}}, \tau_i) = 0$
        Abort and Simulate
store $\mathsf{tr} = (\{(\mathsf{state}_i, \tau_i, \tau_i^{\mathsf{acc}})\}_{i\in[[n]]}, \mathsf{sid}, \mathsf{nonce}_t^{\mathsf{sid}})$
//If no aborted/simulated run, handshake is successful
$k'$ is the session key
```

Figure 4: SHS.Handshake**: Secret Handshake in** LCA

Also we need to use the function pad which pads the data up to the size of a constant, that symbolizes the maximum numbers of size that can acheive π^{SRL}. This helps to prevent against Result-hiding.

Banning a user: SHS.Ban**:.** If a user misbehaves in a secret handshake, other group members taking part in the handshake can report this, forwarding the session's transcript including the guilty user's tag τ_T. The group authority verifies that the tag is correct with respect to the transcript (*i.e.,* with respect to session identifier, randomness from AGKA-FR, the proofs are all verifiable, etc.). If the group authority decides to ban the user, GA updates $\mathsf{sid}_{\mathsf{ban}} \longleftarrow H(\mathsf{sid}_{\mathsf{ban}}||\mathsf{tr}.$
$\mathsf{sid}_T)$, adds the tag to SRL and broadcasts $\mathsf{sid}_{\mathsf{ban}}$, tr, τ_T, sid_T.

The value of tr, which is computed at the end of each successful handshake, is crucial in allowing all group users (not just those who took part in the handshake) to verify that the transcript is valid, that τ_T was generated in that session, and that the session was correctly run. All group members then add the tuple (τ_T, sid_T) in SRL. They also unicast a tag on $\mathsf{sid}_{\mathsf{ban}}$.

The group authority will check first that it has received as many responses as it expected (it bans all users that do not respond), and that no two tags match back to single entity (this happens in case of corruption or malicious users, and the user is banned). Finally, any user whose tag matches SRL is removed from CBU2.

5.3 Security Analysis

We present in this section a security statement for our protocol, and include a very brief intuition of the more salient points of the security proofs. The latter can be found in Appendix D.5 (each statement corresponds in the appendix to to a separate theorem).

Theorem 2. *Let* Π *be* LCA *scheme described in subsection 5.2 . The following statements hold for this protocol:*

- Π *achieves* Unlinkability *if* AGKA-FR *guarantees unlinkability and* ListMAC *guarantees unlinkability;*
- Π *achieves* User authentication *if the employed Symmetric Encryption scheme is correct, if* CBU2 *guarantees broadcast-security, broadcast-authentication, unicast-authentication, and if* ListMAC *guarantees EUF-CMA-AD and non-frameability;*
- Π *achieves* Non-frameability *if* ListMAC *guarantees non-frameability';*
- Π *achieves* Self distinction *if* ListMAC *guarantees non-frameability and EUF-CMA-AD, and if* AGKA-FR *guarantees correctness;*
- Π *achieves* Result-hiding *if* AGKA-FR *guarantees correctness, AKE-security, if HKDF acts as a PRF, and if the Symmetric Encryption guarantees IND-CPA;*
- Π *achieves* Handshake simulability *if* AGKA-FR *guarantees handshake-simulatability, AKE-security, and correctness, if the Symmetric Encryption guarantees IND-CPA-secure and correctness, and if* CBU2 *guarantees broadcast security;*
- Π *achieves* Traitor Catching *if* ListMAC *guarantees EUF-CMA-AD and non-frameability, and* CBU2 *guarantees broadcast authentication and unicast authentication.*

Sketch. In the unlinkability proof, the adversary gets to query a Left-or-Right oracle, which adds one of two "comparable" users to an existing set of users running handshake. In the proof, we replace, step by step, the values that might give any information to the adversary about which user is involved. The AGKA-FR is already unlinkable. Since ListMACs are also unlinkable, the adversary cannot use the content of the handshake's authentication messages to distinguish between the users.

In the user authentication proof, some more obvious reductions involve the adversary not being able to break the broadcast security of CBU2 for groups it is not a member of. Moreover, it is paramount that the adversary cannot broadcast over such a channel itself. However, a less obvious step is the reduction to unicast authentication for CBU2. An adversary that is able to unicast instead of an honest user is able to linger on the CBU2 even after requesting to Leave.

Non-frameability follows in a straight-forward manner from ListMACs.

In self-distinction, an obvious way for the adversary to cheat is to try to produce a List MAC on behalf of a user that is not taking part in the handshake (which counts as a forgery). Another strategy could be for the adversary to produce two MACs with the same key and hope matching fails to detect this – or alternatively, the adversary could try to produce a MAC with a key from KRL – a fact which again Match should detect. The soundness of the Match algorithm is captured by the Non-Frameability of ListMACs.

Result-hiding can also be perceived as a Left-or-Right indistinguishability game: the left option is a successful handshake, whereas the right option is an unsuccessful one. One way to distinguish is for the adversary to be faced with a collision between nonces in the AGKA-FR part of the protocol (since participants always abort if such an event occurs). Note that in this game, the adversary cannot influence the handshake and cause the collision itself. Furthermore, in order to hide the result, it is important for the protocol to match real executions with the simulation provided whenever an error occurs. In other words, we need real ciphertexts to be similar to simulated ones, which is true if the symmetric encryption is IND-CPA and the used keys are indistinguishable from random. Finally, note that in order to hide the result, it is imperative to pad the size of the proof of non-banishment, which would otherwise fluctuate depending on the size of the banished user set, thus betraying whether participants are from the same group or not.

For handshake simulatability, it should be possible to simulate the protocol such that a user not in a given group is unable to distinguish between a group member and the simulator. The AGKA-FR component's handshake simulatability is an easy first step. The simulator simulates the remainder of the protocol by generating random key material, simulating the ListMAC, and otherwise following protocol.

Finally, for the traitor-catching property, our proof will have to focus on the scheme's ability to either respond with a forged List MAC during the manager's challenge or to bypass matching algorithms. □

6 CONCLUSION

Our paper presents a new primitive, List MACs, it formalizes two further building blocks (CBU2 and AGKA-FR), and uses them to construct secret handshakes in a clean, modular fashion. The resulting secret handshake construction LCA achieves user-authentication, unlinkability, self-distinction, non-frameability, result-hiding, and handshake simulatability.

In addition, we put forth a new paradigm of accountability, called traitor catching. As opposed to traitor-tracing, which reveals not just dishonest participants, but all the participants to a handshake, in traitor-catching all innocent participants retain their full unlinkability.

We note that, given its ability to ensure the new property of traitor catching, List MACs are Privacy-Enhancing Technologies that can be of further interest to the community. We propose two quantum-resistant instantiations for this primitive, one based on hash functions and another, based on lattices.

A quirk of our protocol and of secret handshakes in general is that, when users have multiple affiliations, it is not clear in practice which group input it will use. In our work, we assume that each user randomly picks the group whose credentials it will use within the handshake. We leave as further work to find a better means of ensuring that users in multiple groups can better pick the group they will prove their affiliation to during the protocol run.

REFERENCES

[1] Zhiyuan An, Jing Pan, Yamin Wen, and Fangguo Zhang. Forward-secure revocable secret handshakes from lattices. In Jung Hee Cheon and Thomas Johansson, editors, *Post-Quantum Cryptography*, pages 453–479, Cham, 2022. Springer International Publishing.

[2] Giuseppe Ateniese, Jonathan Kirsch, and Marina Blanton. Secret handshakes with dynamic and fuzzy matching. In *Proceedings of NDSS*. The Internet Society, 2007.

[3] Dirk Balfanz, Glenn Durfee, Narendar Shankar, Diana K. Smetters, Jessica Staddon, and Hao-Chi Wong. Secret handshakes from pairing-based key agreements. In *Proceedings of (S&P 2003)*, pages 180–196. IEEE Computer Society, 2003.

[4] Dan Boneh, Özgür Dagdelen, Marc Fischlin, Anja Lehmann, Christian Schaffner, and Mark Zhandry. Random oracles in a quantum world. In *Advances in Cryptology–ASIACRYPT 2011: 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings 17*, pages 41–69. Springer, 2011.

[5] Ernie Brickell, Jan Camenisch, and Liqun Chen. Direct anonymous attestation. In *Proceedings of the 11th ACM conference on Computer and communications security*, pages 132–145, 2004.

[6] Ernie Brickell and Jiangtao Li. Enhanced privacy id: A direct anonymous attestation scheme with enhanced revocation capabilities. In *Proceedings of the 2007 ACM workshop on Privacy in electronic society*, pages 21–30, 2007.

[7] Christina Brzuska, Håkon Jacobsen, and Douglas Stebila. Safely exporting keys from secure channels: on the security of EAP-TLS and TLS key exporters. In *EuroCrypt*, 2016.

[8] Sébastien Canard, Berry Schoenmakers, Martijn Stam, and Jacques Traoré. List signature schemes. *Discrete Appl. Math.*, 154(2):189–201, 2006.

[9] Liqun Chen, Changyu Dong, Nada El Kassem, Christopher JP Newton, and Yalan Wang. A new hash-based enhanced privacy id signature scheme. In *International Conference on Post-Quantum Cryptography*, pages 37–71. Springer, 2024.

[10] Benny Chor, Amos Fiat, and Moni Naor. Tracing traitors. In Yvo Desmedt, editor, *Advances in Cryptology - CRYPTO '94, 14th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1994, Proceedings*, volume 839 of *Lecture Notes in Computer Science*, pages 257–270. Springer, 1994.

[11] Nada El Kassem, Luís Fiolhais, Paulo Martins, Liqun Chen, and Leonel Sousa. A lattice-based enhanced privacy id. In *Information Security Theory and Practice: 13th IFIP WG 11.2 International Conference, WISTP 2019, Paris, France, December 11–12, 2019, Proceedings 13*, pages 15–31. Springer, 2020.

[12] Hanwen Feng, Jianwei Liu, and Qianhong Wu. Secure stern signatures in quantum random oracle model. In *International Conference on Information Security*, pages 425–444. Springer, 2019.

[13] Tibor Jager, Florian Kohlar, Sven Schäge, and Jörg Schwenk. On the security of TLS-DHE in the standard model. In *Proceedings of CRYPTO 2012*, volume 7417 of *LNCS*, pages 273–293, 2012.

[14] Stanisław Jarecki, Jihye Kim, and Gene Tsudik. Beyond secret handshakes: Affiliation-hiding authenticated key exchange. In Tal Malkin, editor, *Topics in Cryptology – CT-RSA 2008*. Springer Berlin Heidelberg, 2008.

[15] Dr. Hugo Krawczyk and Pasi Eronen. Hmac-based extract-and-expand key derivation function (hkdf). Technical Report 5869, May 2010.

[16] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. *Journal of the ACM (JACM)*, 60(6):1–35, 2013.

[17] Mark Manulis, Bertram Poettering, and Gene Tsudik. Taming big brother ambitions: More privacy for secret handshakes. In *Proceedings of PETS*, volume 6205 of *Lecture Notes in Computer Science*, pages 149–165. Springer, 2010.

[18] The European Parliament and the Council of the European Union. on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/ec (general data protection regulation), 2016.

[19] The European Parliament and the Council of the European Union. Regulation (eu) 2018/1725 of the european parliament and of the council, 2018.

[20] Gene Tsudik and Shouhuai Xu. A flexible framework for secret handshakes. In *Proceedings of PETS*, volume 4258 of *Lecture Notes in Computer Science*, pages 295–315. Springer, 2006.

[21] Dominique Unruh. Non-interactive zero-knowledge proofs in the quantum random oracle model. In *Advances in Cryptology-EUROCRYPT 2015: 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II 34*, pages 755–784. Springer, 2015.

[22] Mark Zhandry. Secure identity-based encryption in the quantum random oracle model. *International Journal of Quantum Information*, 13(04):1550014, 2015.

[23] Zhuoran Zhang, Fangguo Zhang, and Haibo Tian. CSH: A post-quantum secret handshake scheme from coding theory. In *Proceedings of ESORICS*, volume 12309 of *Lecture Notes in Computer Science*, pages 317–335. Springer, 2020.

A BACKGROUND ON SECRET HANDSHAKES

We present in this appendix further formal definitions regarding the type of classical secret handshakes that are instantiated by GCD [20]. Note that our work additionally accounts for user banishment, which forces us to slightly change this syntax.

The GCD approach is an instantiation of a secret handshake scheme that can be defined as $\mathsf{SHS} = (\mathsf{SHS.Setup}, \mathsf{SHS.NewGroup}, \mathsf{SHS.Join}, \mathsf{SHS.Leave}, \mathsf{SHS.Update}, \mathsf{SHS.Handshake}, \mathsf{SHS.Trace}, \mathsf{SHS.Judge})$ such that:

$(\mathsf{msk}, \mathsf{ppar}) \leftarrow \mathsf{SHS.Setup}(1^\lambda)$: this global setup algorithm outputs some universal private parameters (given to all group authorities) and public parameters ppar, implicitly taken in input to all other algorithms.

$(\mathsf{spar}_G, \mathsf{ppar}_G) \leftarrow \mathsf{SHS.NewGroup}(G)$: a group authority runs this algorithm for a unique G to output private group parameters spar_G (only known to GA) and public group parameters ppar_G. The group authority sets $\mathsf{USet}_G \leftarrow \emptyset$ and keeps track of state value state(*e.g.,* a group private key, member identities, etc.).

$((\mathsf{sk}_U, \mathsf{pk}_U), \{(\mathsf{sk}_V, \mathsf{pk}_V)\}_{V \in \mathsf{USet}_G}, \mathsf{USet}_G) \leftarrow \mathsf{SHS.Join}(U, \mathsf{spar}_G)$: a user U interacts with the group authority GA to yield, on the user's side, private and public keys $(\mathsf{sk}_U, \mathsf{pk}_U)$, updated keys $(\mathsf{sk}_V, \mathsf{pk}_V)$ to all other group users, and an updated set of users for GA.

$(\bot, \{(\mathsf{sk}_V, \mathsf{pk}_V)\}_{V \in \mathsf{USet}_G \setminus U}, \mathsf{USet}_G) \leftarrow \mathsf{SHS.Leave}(U, \mathsf{spar}_G)$: this is an interactive algorithm run between a user U, which either leaves the group or is forcibly revoked, and the group authority (on input spar_G). At the end, the leaving user produces no output, the other users in the group possibly update new parameters $\mathsf{sk}_V, \mathsf{pk}_V$, and the group authority outputs an updated user set USet_G.

$\{(\mathsf{sk}_U, \mathsf{pk}_U)\}_{U \in \mathsf{USet}_G} \leftarrow \mathsf{SHS.Update}(\mathsf{spar}_G)$: the group authority can run this algorithm (on input spar_G), which results in updated keys for each user $U \in \mathsf{USet}_G$.

$(\{(k^i_U, \tau^i_U)\}_{U \in \mathsf{SHSet}}) \leftarrow \mathsf{SHS.Handshake}(\{\mathsf{sk}_U\}_{U \in \mathsf{SHSet}})$: a set of users SHSet runs this algorithm (using their private keys);

each user ends up outputting a private session key k_U^i (which can take a special value $\perp$), and a transcript τ_U^i.

$((\perp, \perp) \cup \{U, \pi_U^i\}_{U \in \mathsf{SHSet}}) \leftarrow \mathsf{SHS.Trace}(\mathsf{spar}_G, \{\tau_U^i\}_{U \in \mathsf{SHSet}})$: on input the private group parameters spar_G, and a list of handshake transcripts, this algorithm outputs either a single couple of elements $(\perp, \perp)$, indicating that the handshake had ended in failure, or a set of elements (U, π_U^i) for each user $U \in \mathsf{SHSet}$, consisting of a user identity and a proof π_U^i.

$(0 \cup 1) \leftarrow \mathsf{SHS.Judge}(\mathsf{msk}, \pi_U^i)$: on input a proof π_U^i and the judge's master paramters msk, this algorithm outputs either 1 (the proof is valid) or 0 (it is not).

The basic correctness definition requires that, assuming that we run in sequence : $(\mathsf{msk}, \mathsf{ppar}) \leftarrow \mathsf{SHS.Setup}(1^\lambda)$; $(\mathsf{spar}_G, \mathsf{ppar}_G) \leftarrow \mathsf{SHS.NewGroup}(G)$; $\forall U \in \mathsf{SHSet}$, we run
$(\{\mathsf{sk}_U, \mathsf{pk}_U\}, \{\mathsf{sk}_V, \mathsf{pk}_V\}_{V \in \mathsf{USet}_G}, \mathsf{USet}_G) \leftarrow \mathsf{SHS.Join}(U, \mathsf{spar}_G)$, where we set
$(\{k_U^i, \tau_U^i\}_{U \in \mathsf{SHSet}}) \leftarrow \mathsf{SHS.Handshake}(\{\mathsf{sk}_U\}_{U \in \mathsf{SHSet}})$, and then:

- There exists a value $k \neq \perp$ such that $\forall U \in \mathsf{SHSet}$ it holds that: $k = k_U^i$ (*i.e.,* all the keys computed in a successful handshake are identical);
- $\forall U \in \mathsf{SHSet}$, it holds that: $(\{U, \pi_U^i\}_{U \in \mathsf{SHSet}}) \leftarrow \mathsf{SHS.Trace}(\mathsf{spar}_G, \{\tau_U^i\}_{U \in \mathsf{SHSet}})$ and $\forall \pi_U^i$ generated in this way: $1 \leftarrow \mathsf{SHS.Judge}(\mathsf{msk}, \pi_U^i)$ (*i.e.,* the transcripts generated in a successful handshake are all traceable to the correct identities, and moreover, the proofs of the tracing are valid).

A.1 A cleaner handshake design

Although the GCD paper by Tsudik and Xu aims to provide formalizations and security proofs, this is not quite achieved. In this section, we point out several problems and disadvantages in their construction, beginning with the ones we deem the most serious.

Message-dependent security. Recall that in the secret handshake, users compute $k_i = k \oplus \hat{k}$ as their potential secret key. Following MAC verification, if all the MACs verify, each U_i will: PK encrypt k_i with the public key of the group authority: $\delta_i \leftarrow \mathsf{PKE.Enc}_{\mathsf{pk}_{\mathsf{GA}}}(k_i)$; *and* use k_i as a private key in the symmetric-key encryption of the group signature σ_i.

In particular, we notice that the key k_i is the message encrypted in δ_i, but also the key used in deriving θ_i, which would make a formal proof of security difficult under standard assumptions. Finally, note that k_i is meant to be later used in order to secure a channel between the handshake participants – which further complicates a potential proof.

Insufficient MAC security. The theorems stating the properties of the two GCD constructions fail to indicate which assumptions are required for both the MAC scheme that they use, and for the symmetric-encryption scheme. However, the authors indicate that any MAC scheme is sufficient – which seems to imply that the GCD scheme only relies on existential unforgeability under chosen-message attacks.

However, recall that following the distributed key-agreement, the parties running the secret handshake use a MAC keyed with the session key k_i to create a tag on a user-specific value s and their index i. Under standard unforgeability assumptions, we have no guarantee that k_i is still indistinguishable from random after this step, which jeopardizes further security steps (such as symmetric-encrypting with k_i).

Insufficient group key-agreement security. Tsudik and Xu strongly rely in their GCD construction on the security of a building block they call distributed group key-agreement. The latter is only assumed secure against a passive adversary, which is moreover not part of the group key-agreement (at least within the targeted session).

While the emphasis in the paper is that the group key-agreement *need not* be authenticated – this is not enough to ensure that it is, at the very minimal, affiliation hiding. Consider a protocol in which each handshake participant prefaces its messages in the group key-agreement by a plaintext unique identifier of its group. Clearly this cannot count as an authentication – yet, it is sufficient to damage important properties of Tsudik and Xu's security aims (such as Result-hiding and Unlinkability).

Overcomplicated traceability. In the traceability game, the parties running a handshake can potentially cheat, for instance computing a $\hat{\delta}_i$ value that is not correctly generated: either by encrypting under some $\hat{\mathsf{pk}} \neq \mathsf{pk}_{\mathsf{GA}}$, or by encrypting some value r under the correct $\mathsf{pk}_{\mathsf{GA}}$. Yet, it is crucial for GA to retrieve k_i, since it must later decrypt θ_i to retrieve σ_i.

In the GCD paper, since GA cannot be sure that what it decrypts from a δ_i value is the genuine key, the authority has to decrypt *all* the δ_i value of all the participants, then exhaustively try out all the retrieved keys to decrypt the signatures. It will be convinced that the key is genuine when, having decrypted the corresponding signature, the latter also verifies.

The degree of obtained traceability is thus relatively weak. For one thing, traceability is only obtained if all the parties running the secret handshake belong to the same group (otherwise, only random signature values are returned). For another, GA has to perform potentially $2n$ decryptions and n signature-verifications in order to decide what the correct k_i was, which is prohibitive for large values of n.

Full-unlinkability is not achieved if users misbehave. The authors present two schemes. In the second scheme, the group signature is claimed to have Self distinction, and it is argued that Full-unlinkability is maintained because the signature is encrypted with the session key, which is not stored by the user. However, if users misbehave and secret keys are corrupted, then the attacker can use self-distinction in order to determine if it has already interacted in some previous session with the corrupted user.

Missing assumptions. In contrast to the standard assumptions stated in their security theorems, Tsudik and Xu actually also rely on several unstated assumptions.

For instance, there is no requirement specified for the symmetric encryption used to obtain θ_i. As described above, simple IND-CPA (or even IND-CCA) security is insufficient: we would need resistance to key-dependent messages.

We recall that a similar problem occurs for the MACs deployed.

Finally, note that the schemes rely on secure channels always existing between the group manager and each of the users – a relatively-strong assumption which might be used more – for instance during the handshake, in order to just forward the session key and the group signature, without encrypting it. This would enable much quicker traceability.

Rekeying used as a Key Management methodology. Each time a user joins or leaves the group, the GA needs to rekey for all user and for GA itself, in order to change the verification key in case a malicious user could communicate with an insider and therefore manage to authenticate. The new keys "are *somehow* sent to the legitimate users through the authenticated channels (depending on concrete schemes)" [20]. This might potentially lead to cumbersome calculation, but more importantly, the term '*somehow*' indicates a potentially large attack surface; concretely, since the identities of user doesn't exist outside the group how can we ensure the continuity?

B LISTMAC: TOWARDS PQ-SECURE SECRET HANDSHAKES

We begin by reminding the reader that the ListMAC has the purpose to achieve those following properties: (1) Self distinction (detect if a user has signed twice during the same session *i.e.,* for the same aux), (2) Unlinkability (it's not possible to know if we have already interacted with the user in a previous session), and (3) Non-frameability (an adversary aided by the GA and some malicious users cannot forge a signature involving an honest user) ; More formal definitions and security properties can be found in subsection D.4. To obtain Self distinction – the main feature of ListMAC – the *naive* idea is that a tag issued from ListMAC has a traceable word (also called a nym) which is unique and combine the secret of the user and the aux in a such way that the users can compare and state whether this traceable word, nym, looks like another one which represent the same tuple.

B.0.1 Consequences.

Proposition 1 (multi-set checking). *Since ListMAC possesses two types of matching we can combine them and obtain some properties such as many set matching due to a type of transitiveness given with* LM.Match.

To check if a user is matchable in two sets, let's say S and S', we can do the following:

1. *Prover P tags for the first set* $(\tau, \pi) \longleftarrow \mathsf{LM.Tag}(\mathsf{sk}_P, m, \mathsf{aux}, S)$
2. *Prover P tags for the second set* $(\tau', \pi') \longleftarrow \mathsf{LM.Tag}(\mathsf{sk}_P, m', \mathsf{aux}, S')$ *but needs to tag with the same* aux *as previously.*
3. *Prover P conveys* $(\mathsf{aux}, (m, \tau, \pi, S), (m', \tau', \pi', S'))$ *to the verifier V*
4. *Verifier V checks if the tags are valid (i.e.,* LM.Ver*)*
5. *Verifier V checks if the tags were issued from the same user (i.e.,* LM.Match*)*
6. *Verifier V can check independently which set is matching with the prover P by running:*
   - $\mathsf{LM.MatchSet}(\mathsf{gvk}, m, \mathsf{aux}, \tau, \pi, S)$
   - $\mathsf{LM.MatchSet}(\mathsf{gvk}, m', \mathsf{aux}, \tau', \pi', S')$

Remark 2. *Some words about item 5, a contrario of naive methods where we would checks with $O(n^2)$, with n being the numbers of tags, here we can check with many-to-one tag therefore decreasing to $O(n)$.*

Proposition 2 (Leaking matching tag in list). *There exists a sequence of call to* oTag *such that, it is possible to leak the tag which is matching in the list S.*

Proof. Also, due to the Proposition 1 it is possible for an adversary to determine the same information by using sets that are disjoint to each other except for few elements (therefore obtaining more equation than variables) according to t-design theory. *e.g.,* Let say we test those following sets[9] composed of the tag of the user U_x as follows τ_x:

$$S_1 = \{(\tau_2, \mathsf{aux}_2), (\tau_3, \mathsf{aux}_3)\}$$
$$S_2 = \{(\tau_1, \mathsf{aux}_1), (\tau_3, \mathsf{aux}_3)\}$$

Consider the issued tags of those set, let's say $(m, \mathsf{aux}, \tau, \pi_1)$ tagged with S_1 (*idem* for $(m', \mathsf{aux}', \tau', \pi_2)$ with S_2), with the following:

$$\mathsf{LM.MatchSet}(\mathsf{gvk}, m_1, \mathsf{aux}, \tau, \pi_1, S_1) = r_1$$
$$\mathsf{LM.MatchSet}(\mathsf{gvk}, m_2, \mathsf{aux}', \tau', \pi_2, S_2) = r_2$$

therefore we can deduce the following and obtaining the identity of the user:

$$r_1 = 1 \wedge r_2 = 1 \implies U_b = U_3$$
$$r_1 = 1 \wedge r_2 = 0 \implies U_b = U_2$$
$$r_1 = 0 \wedge r_2 = 1 \implies U_b = U_1$$
$$\text{otherwise}\, U_b \notin \{U_1, U_2, U_3\}$$

□

B.1 Rewriting from EPID to ListMAC

We notice that ListMAC has similitude with EPID (Enhanced Privacy ID [6]) signature. In fact, the GA could be seen as the TPM[10] in this paradigm. We propose a new instantiation using Lattice-based mechanism (based on the following work [11]).

We need to specify first the scheme of an EPID, and then explain how we convert it in a ListMAC instance.

Definition 5 (EPID). *Historically Intel proposed DAA (Direct Anonymous Attestation [5]) and upgrade it to EPID. DAA is aimed to be an anonymous group signature, where the group is managed by the* TPM. *An EPID [6] is composed of an issuer (or* TPM*) which manages a group and members which can sign anonymously on behalf of the group. "An EPID has the following four procedures:*

[9] The reader may see a reference to block design theory. The instance described is for an incidence structure $\mathcal{D} = (\mathcal{P}, \mathcal{B}, \mathcal{I})$ is a $2 - (3, 2, 1)$design (*i.e.,* $t - (v, k, \lambda)$design) where:

$$\mathcal{P} = \{(\tau_1, \mathsf{aux}_1), \ldots, (\tau_3, \mathsf{aux}_3)\}$$
$$\forall B_i \in \mathcal{B}, B_i = \mathcal{P} \setminus \{(\tau_i, \mathsf{aux}_i)\}$$
$$\forall \{a, b\} \subset \mathcal{P}, \exists i \in \{1, 2, 3\} \setminus \{a, b\}, \{a, b\} \in B_i$$

In our example B_i is dubbed S_i. Therefore, we need $t = 2$ blocks to determine precisely which one ($\lambda = 1$) of the three users it was.

[10] A TPM stands for Trusted Platform Module and is a hardware chip aiming to accelerate and strengthen the cryptographic operations. This chip contains a hard coded key usually considered as non corruptible.

- `Setup`*: In this procedure, the issuer creates a group public key and a group issuing private key. The issuer publishes the group public key.*
- JoinGroup*: This is a protocol between the issuer and a user that results in the user becoming a new group member. At the end of this protocol, the user obtains a membership private key from the issuer.*
- Verify *i.e., Proof of Membership: In this protocol, a prover interacts with a verifier to convince the verifier that he is a member of the group in good standing (i.e., without being revoked). [...]*
- Revocation*: The revocation manager puts a group member into the revocation list. There are three types of revocations: (1) private-key based revocation in which the revocation manager revokes a user based on the user's membership private key, (2) signature based revocation in which the revocation manager revokes a user based on the signatures created by the user, and (3) issuer based revocation in which the revocation manager revokes a user based on the recommendation from the issuer." – [6]*

Rewriting. From that, we notice that the first three methods are almost the same as in ListMAC[11], but the Revocation will be adapted and broken down to obtain the so-called `LM.Match` and `LM.MatchSet` to obtain Self distinction. One of the idea in the Revocation function is to use a pseudonym that is linked to a word and the secret. Therefore, having to prove that the user isn't banned, the latter needs to prove that when generating the pseudonym of the same tuple the outcome will be far enough/different from the referential pseudonym. If we fix the word to be the equivalent of our aux therefore we have a sort of `LM.Match`, as described previously.

Difference. Lattice-based and hash-based approaches differ both in their construction and underlying properties. Throughout the detailed instantiation, the main distinction may not be immediately apparent. In lattice-based methods, the output consists of *match tokens* that are close but not exactly identical, which necessitates a function `LM.Match` that performs pairwise comparisons. This results in a time complexity of $O(n^2)$ for a set of n elements. In contrast, hash-based methods produce *match tokens* that are exactly the same, allowing the use of a key-value database; this enables the `LM.Match` function to operate in $O(n)$ time.

B.2 Lattice-based

B.2.1 Preliminaries.

Definition 6 (Lattice). *A* lattice $\mathcal{L}$ *is a discrete additive subgroup of* $\mathbb{R}^n$*. More concretely, it is the set of all integer linear combinations of a set of linearly independent vectors* $\mathbf{b}_1, \ldots, \mathbf{b}_m \in \mathbb{R}^n$*:*

$$\mathcal{L}(\mathbf{B}) = \left\{ \sum_{i=1}^{m} z_i \mathbf{b}_i \mid z_i \in \mathbb{Z} \right\}$$

where $\mathbf{B} = [\mathbf{b}_1, \ldots, \mathbf{b}_m]$ *is called a* basis *of the lattice and* $m \leq n$ *is the rank of the lattice.*

Definition 7 (Ring LWE). *First of all we define the polynomial rings* $R_q = \mathbb{Z}_q[X]/ < X^n + 1 >$ *where* $\mathbb{Z}_q$ *is the quotient ring* $\mathbb{Z}/q\mathbb{Z}$*. We define* $A_{s,\chi}$ *the distribution that yields* (u, v) *such that,* $u \xleftarrow{\$} R_q$ *and* $v \leftarrow u \cdot s + e$ *with* $e \xleftarrow{\$} \chi$*.*

Definition 8 (Search Ring-LWE). *Given a polynomial number of* $A_{s,\chi}$*'s samples and adversary finds* s*.*

$$\Pr[\mathsf{Exp}_{\mathsf{RLWE}}] = \Pr\left[s = s' : \begin{array}{c} s \xleftarrow{\$} Rq \\ \mathcal{A}^{\mathsf{oA}^1_{\mathsf{s},\chi}}() \end{array} \right]$$

Definition 9.

$$\mathsf{Adv}^{\mathsf{RLWE}} = \frac{1}{1-\epsilon} \left| \Pr[\mathsf{Exp}_{\mathsf{RLWE}} = 1] - \epsilon \right|$$

with ϵ *negligeable.*

Definition 10 (Decisional Ring-LWE-RoR).

$$\Pr[\mathsf{Exp}^b_{\mathsf{DRLWE-RoR}} = 1] = \Pr\left[b = d : \begin{array}{c} s \xleftarrow{\$} R_q \\ (u_0, v_0) \xleftarrow{\$} R_q^2 \\ (u_1, v_1) \xleftarrow{\$} A_{s,\chi} \\ d \leftarrow \mathcal{A}(u_b, v_b) \end{array} \right]$$

Definition 11.

$$\mathsf{Adv}^{\mathsf{DRLWE-RoR}} = \left| \Pr[\mathsf{Exp}^1_{\mathsf{DRLWE-RoR}} = 1] - \Pr[\mathsf{Exp}^0_{\mathsf{DRLWE-RoR}} = 0] \right|$$

We use the game of Decisional Ring-LWE-RoR in the average case decision such as defined in [16].

Definition 12 (Decisional Ring-LWE-RoR, Average case decision).

$$\Pr[\mathsf{Exp}^b_{\mathsf{avgDRLWE-RoR}} = 1] = \Pr\left[b = d : \begin{array}{c} s \xleftarrow{\$} R_q \\ d \leftarrow \mathcal{A}^{\mathsf{oA}^{\mathsf{b}}_{\mathsf{s},\chi}}() \end{array} \right]$$

where the oracle is defined as follow:

$$\begin{array}{l} \hline \mathsf{oA}^{\mathsf{b}}_{\mathsf{s},\chi} \\ \hline (u_0, v_0) \xleftarrow{\$} R_q^2 \\ (u_1, v_1) \xleftarrow{\$} A_{s,\chi} \\ \textbf{\textit{return}}\ (u_b, v_b) \\ \hline \end{array}$$

Definition 13.

$$\mathsf{Adv}^{\mathsf{avgDRLWE-RoR}} = \Big| \Pr[\mathsf{Exp}^1_{\mathsf{avgDRLWE-RoR}} = 1] \\ - \Pr[\mathsf{Exp}^0_{\mathsf{avgDRLWE-RoR}} = 0] \Big|$$

Definition 14 (Decisional Ring-LWE-LoR, Average case decision).

$$\Pr[\mathsf{Exp}^b_{\mathsf{avgDRLWE-LoR}} = 1] = \Pr\left[b = d : \begin{array}{c} (s_0, s_1) \xleftarrow{\$} R_q^2 \\ d \leftarrow \mathcal{A}^{\mathsf{oA}^1_{\mathsf{s}_0,\chi_0}, \mathsf{oA}^1_{\mathsf{s}_1,\chi_1}, \mathsf{oA}^1_{\mathsf{s}_\mathsf{b},\chi_\mathsf{b}}}() \end{array} \right]$$

Theorem 3.

$$\mathsf{Adv}^{\mathsf{avgDRLWE-LoR}} \leq 2\mathsf{Adv}^{\mathsf{avgDRLWE-RoR}}$$

Proof. It is equivalent to prove the following:

$$A_{s_0,\chi_0} \overset{C.I.}{\equiv} R_q^2 \wedge A_{s_1,\chi_1} \overset{C.I.}{\equiv} R_q^2 \implies A_{s_0,\chi_0} \overset{C.I.}{\equiv} A_{s_1,\chi_1}$$

[11] Considering breaking down `Setup` into `LM.Setup` and `LM.GenGroup`. Plus JoinGroup is dubbed as `LM.RegUser` and Verify is dubbed as `LM.Ver`.

Thus by the triangular inequality we obtain:

$$
\begin{aligned}
&\mathsf{Adv}^{\mathsf{avgDRLWE-LoR}}\\
&=\left|\Pr[\mathsf{Exp}^1_{\mathsf{avgDRLWE-RoR}(s_0)}=1]-\Pr[\mathsf{Exp}^0_{\mathsf{avgDRLWE-RoR}(s_1)}=0]\right|\\
&=\Big|\Pr[\mathsf{Exp}^1_{\mathsf{avgDRLWE-RoR}(s_0)}=1]-\mathsf{Exp}^0_{\mathsf{avgDRLWE-RoR}(s_0)}=0]\\
&\quad+\Pr[\mathsf{Exp}^0_{\mathsf{avgDRLWE-RoR}(s_1)}=0]-\Pr[\mathsf{Exp}^1_{\mathsf{avgDRLWE-RoR}(s_1)}=1]\Big|\\
&\le 2\mathsf{Adv}^{\mathsf{avgDRLWE-RoR}}
\end{aligned}
$$

□

B.2.2 Instanciation. Here, we present a flavor that is based on the following work [11]. For the reader we boxed all modifications inside the functions.

$\mathsf{LM.Setup}(1^\lambda) \longrightarrow \mathsf{param}$

First the GA instantiates all common functions and strings:

$$\mathsf{param}=\{\lambda,t,q,n,\bar{m},\beta,l,r,s,\xi,\mathcal{R}_q,\mathcal{H}_p,\mathcal{H},H,H'\}$$

Where "λ,t are positive integer security parameters, β is a positive real number such that $\beta < q$, l is the length of the users' identifiers, and r,s and ξ represent standard deviations of Gaussian distributions" [11]. Alsos the common function $\mathcal{H}_p : \{0,1\}^* \longrightarrow \mathcal{R}_q$ a hash function mapping a string into a polynomial (contained into the ring $\mathcal{R}_q$), $\mathcal{H} : \{0,1\}^* \longrightarrow \{1,2,3\}^t$ and $H : \{0,1\}^* \longrightarrow \{0,1,2,\ldots,2n-1\}$. And furthermore a hash function $H' : \{0,1\}^* \longrightarrow \{y \in \mathcal{D}_s : ||y||_\infty \le \beta\}$ maps a binary word into a polynomial issued from a gaussian distribution centered in s of infinite norm less or equal to β.

$\mathsf{LM.GenGroup}(\mathsf{param}) \longrightarrow (\mathsf{gmk},\mathsf{gvk},\mathsf{bsn})$

$$
\begin{aligned}
\mathsf{gvk}&=(b,\hat{A}_{\mathcal{I}},\hat{A}_0,\hat{A}_1,\ldots,\hat{A}_l,\mathbf{u},\pi_{\mathcal{I}})\\
\mathsf{gmk}&=\hat{T}_{\mathcal{I}}\\
\mathsf{bsn}&\overset{\$}{\longleftarrow}\{0,1\}^*
\end{aligned}
$$

For the gvk note that $\forall i \in [[0,l]] \cup \{\mathcal{I}\}, \hat{A}_i \in \mathcal{R}_q^{\bar{m}}$ and $\mathbf{b}$ and $\mathbf{u}$ are in $\mathcal{R}_q$. For the gmk we have $\hat{T}_{\mathcal{I}}$ is the GA's private key which is the trapdoor of $\hat{A}_{\mathcal{I}}$ with $||\hat{T}||_\infty \le \beta$. $\pi_{\mathcal{I}}$ is the proof that the latter key is well-formed.

$\mathsf{LM.RegUser}(U,\mathsf{gmk}) \longrightarrow (\mathsf{ID}_U^{\mathsf{LM}},\mathsf{sk}_U) \sqcup \perp$

So first GA generates a nonce $\rho \overset{\$}{\longleftarrow} \{0,1\}^\lambda$. The user requests a registration as follows:

(1) Samples a private key $\mathbf{x}_1 \overset{\text{Gauss}}{\longleftarrow} \mathcal{D}_s$ and $(\mathbf{x}_2,\ldots,\mathbf{x}_{\bar{m}+1}) \overset{\text{Gauss}}{\longleftarrow} \mathcal{D}_r^{\bar{m}}$. [12] Let $\hat{X}_t = (\mathbf{x}_1,\ldots,\mathbf{x}_{\bar{m}+1})$ corresponds to the user's secret key with the condition $||(\mathbf{x}_2,\ldots,\mathbf{x}_{\bar{m}+1})||_\infty \le \beta/2$ and $||\mathbf{x}_1||_\infty \le \beta$. The user samples at random $k_e \overset{\$}{\longleftarrow} \{0,1\}^*$ which is used as a PRF key.

(2) The user "computes its public key $\mathbf{u}_t = [\mathbf{b}|\hat{A}_{\mathcal{I}}]\hat{X}_t \mod q$, a link token $\mathsf{ID}_U^{\mathsf{LM}} = \mathcal{H}_p(\mathsf{bsn})\mathbf{x}_1 + \mathbf{e}_{\mathcal{I}} \mod q$ [also referred as its identity in our scheme ListMAC] for some error $\mathbf{e}_{\mathcal{I}}$ such that $||\mathbf{e}_{\mathcal{I}}||_\infty \le \beta$." [11]. Here in order to avoid key recover we define $\mathbf{e}_{\mathcal{I}}$ as the result of $H'(k_e||\mathsf{bsn})$.

(3) The user generates the following proof:

$$
\begin{aligned}
\pi_{\mathbf{u}_t}=&\mathsf{SPK}\{(\mathsf{param},\mathbf{u}_t,\mathsf{bsn},\mathsf{ID}_U^{\mathsf{LM}});(\hat{X}_t,\mathbf{e}_{\mathcal{I}}):\\
&\mathbf{u}_t=[b|\hat{A}_{\mathcal{I}}]\hat{X}_t \mod q\\
&\wedge\ ||\hat{X}_t/\mathbf{x}_1||_\infty \le \beta/2\ \wedge\ ||\mathbf{x}_1|| \le \beta\\
&\wedge\ \mathsf{ID}_U^{\mathsf{LM}}=\mathcal{H}_p(\mathsf{bsn})\mathbf{x}_1+\mathbf{e}_{\mathcal{I}} \mod q\\
&\wedge\ ||\mathbf{e}_{\mathcal{I}}||_\infty \le \beta\}(\rho)
\end{aligned}
$$

(4) It sends $(\mathsf{ID}_U^{\mathsf{LM}},\mathbf{u}_t,\pi_{\mathbf{u}_t})$ to the GA

Upon receiving it, the GA is proceeding as follows to allow or reject the user:

(1) First the GA checks if no user $\mathsf{ID}_U^{\mathsf{LM}}$ exists in his, her or their, database by checking the following condition:

$$\forall \mathsf{ID}^{\mathsf{LM}} \in DB, ||\mathsf{ID}_U^{\mathsf{LM}} - \mathsf{ID}^{\mathsf{LM}}||_\infty > 2\beta$$

if true then the GA continues otherwise returns a $\perp$ (implying that this user already exists in the database).

(2) GA associates an entry token id $\overset{\$}{\longleftarrow} \{0,1\}^l$

(3) GA computes the vector of polynomials $\hat{A}_h = [\hat{A}_{\mathcal{I}}|\hat{A}_0 + \sum_{i=1}^{l} \mathsf{id}_i\hat{A}_i] \in \mathcal{R}_q^{2\bar{m}}$

(4) GA samples, using the GA's private key $\mathsf{gmk} = \hat{T}_{\mathcal{I}}$, a preimage $\hat{X}_h = [\hat{X}_{h_1}|\hat{X}_{h_2}] = (\mathbf{y}_2,\ldots,\mathbf{y}_{2\bar{m}+1}) \in \mathcal{D}_r^{\bar{m}} \times \mathcal{D}_s^{\bar{m}}$ of $\mathbf{u} - \mathbf{u}_t$ such that $\hat{A}_h\hat{X}_h = \mathbf{u}_h = \mathbf{u} - \mathbf{u}_t \mod q$ and $||\hat{X}_{h_1}||_\infty \le \beta/2$ and $||\hat{X}_{h_2}||_\infty \le \beta$

When the user U receives $(\hat{X}_h,\mathbf{u}_h)$, the user checks if its valid (checking the boundaries 4) and further checks the equalities (mentioned before 4) If everything happens correctly therefore we should have the following:

$$
\begin{aligned}
\mathsf{ID}_U^{\mathsf{LM}}&=\mathcal{H}_p(\mathsf{bsn})\mathbf{x}_1+\mathbf{e}_{\mathcal{I}}\\
\mathsf{sk}_U&=(\mathsf{id},\hat{X},\mathbf{u},k_e)
\end{aligned}
$$

In order to compute $\hat{X}$ the user does the following:

$$
\begin{aligned}
\hat{X}=(&\mathbf{x_1},\forall_{i=(2,\ldots,\bar{m}+1)}\mathbf{x}_i := \mathbf{x}_i+\mathbf{y}_i,\\
&\forall_{j=(\bar{m}+2,\ldots,2\bar{m}+1)}\mathbf{x}_i := \mathbf{y}_i)
\end{aligned}
$$

$\mathsf{LM.Tag}(\mathsf{sk},m,\mathsf{aux},S) \longrightarrow (\tau,\pi)$

To tag the user generates the following proof:

$$
\begin{aligned}
\tau:\mathsf{SPK}\{&(\mathsf{gvk},\mathsf{nym},\boxed{\mathsf{aux}});\\
&(\hat{X}=(x_1,\ldots,x_{2d+1}),\mathsf{id},\boxed{\mathbf{e}_{\mathsf{aux}}}):\\
&[\mathbf{b}|\hat{A}_h]\hat{X}=\mathbf{u}\ \wedge\ ||\hat{X}|| \le \beta\\
&\wedge\ \boxed{\mathsf{nym}=\mathcal{H}_p(\mathsf{aux})x_1+\mathbf{e}_{\mathsf{aux}}}\\
&\wedge\ ||\mathbf{e}_{\mathsf{aux}}||_\infty \le \beta\}(m)
\end{aligned}
$$

where $\mathbf{e}_{\mathsf{aux}} = H'(k_e||\mathsf{aux})$ is always the same value, in order to avoid key recovery. Contrary to the original work [11] using the Fiat-Shamir Heuristic (not suitable for QROM model), the zero-knowledge proof is done by the Unruh's transform [21] proved in QROM for Stern in [12].

If $S \neq \emptyset$, then the user generates π. For each $(\mathsf{nym}_i,\mathsf{aux}_i) \in S$ the user computes the commitment:

- $\mathbf{o}_i = \mathcal{H}_p(\mathsf{aux}_i)\mathbf{q}_i + \mathbf{l}'_i$ where $\mathbf{q}_i, \mathbf{l}'{}_i \overset{\text{Gauss}}{\longleftarrow} \mathcal{D}_s$
- $\mathbf{k}_i = \mathbf{o}_i\mathbf{x}_1 + \mathbf{l}''_i$ where $\mathbf{l}''_i \overset{\text{Gauss}}{\longleftarrow} \mathcal{D}_s$

[12] $x \overset{\text{Gauss}}{\longleftarrow} \mathcal{D}_s^h$ *i.e.,* sampling x over the guassian distribution of standard deviation s such that $\Pr_{x \overset{\text{Gauss}}{\longleftarrow} \mathcal{D}_s^h}[||x|| > s\sqrt{2h}] \le 2^{-h/4}$

- $\mathbf{d}_i = \mathcal{H}_p(\mathsf{aux}_i)\mathbf{q}_i + \mathbf{l}'''_i$ where $\mathbf{l}'''_i \xleftarrow{\text{Gauss}} \mathcal{D}_s$
- $\mathbf{t}_{\mathsf{nym}} = \mathcal{H}_p(\mathsf{aux})\mathbf{r}_{x_1} + \mathbf{r}_e$ where $\mathbf{r}_{x_1}, \mathbf{r}_e \xleftarrow{\text{Gauss}} \mathcal{D}_s$
- $\mathbf{t}_{o_i} = \mathcal{H}_p(\mathsf{aux}_i)\mathbf{r}_{q_i} + \mathbf{r}_{l'_i}$ where $\mathbf{r}_{q_i}, \mathbf{r}_{l'_i} \xleftarrow{\text{Gauss}} \mathcal{D}_s$
- $\mathbf{t}_{k_i} = \mathbf{o}_i\mathbf{r}_{x_1} + \mathbf{r}_{l''_i}$ where $\mathbf{r}_{l''_i} \xleftarrow{\text{Gauss}} \mathcal{D}_s$
- $\mathbf{t}_{d_i} = \mathsf{nym}_i\mathbf{r}_{q_i} + \mathbf{r}_{l'''_i}$ where $\mathbf{r}_{l'''_i} \xleftarrow{\text{Gauss}} \mathcal{D}_s$

Therefore the user respond to a challenge c_v by computing $(\mathbf{s}_{x_1}, \mathbf{s}_e, \mathbf{s}_{q_i}, \mathbf{s}_{l'_i}, \mathbf{s}_{l''_i}, \mathbf{s}_{l'''_i})$ for each $\mathbf{s}_a$ computed as follow : $\mathbf{s}_a = \mathbf{r}_a + X^{c_v}\mathbf{a}$. Abort if one sample is rejected (out of bound).
Importantly the non-interactive proof here should authenticate (τ, S) – ensuring that the values computed were played with the challenge containing the committed values (τ, S)[13]. Again, since this non-interactive proof has the property of special soundness and honest verifier zero-knowledge property [11], then the Unruh's transformation guarantees zero-knowledge and simulation-sound online extractability property in the QROM [21].
the user yields $\pi = (\mathsf{nym}, \mathsf{aux}, \{\forall i \in [[1, \#S]], \mathbf{o}_i, \mathbf{k}_i, \mathbf{d}_i, \mathbf{s}_{x_1}, \mathbf{s}_e, \mathbf{s}_{q_i}, \mathbf{s}_{l'_i}, \mathbf{s}_{l''_i}, \mathbf{s}_{l'''_i}\})$.

$\mathsf{LM.Ver}(\mathsf{gvk}, m, \mathsf{aux}, \tau) \in \{0, 1\}$

First the user checks if the zero-knowledge is correct and verifies the statement. Next if $\pi \neq \perp \wedge S \neq \emptyset$ the user do the following:

(1) Computes:

$$\begin{aligned}
\mathbf{t}'_{k_i} &= \mathbf{o}_i\mathbf{s}_{x_1} + \mathbf{s}_{l''_i} - X^{c_v}\mathbf{k}_i \\
\mathbf{t}'_{d_i} &= \mathsf{nym}_i\mathbf{s}_{q_i} + \mathbf{s}_{l'''_i} - X^{c_v}\mathbf{d}_i \\
\mathbf{t}'_{o_i} &= \mathcal{H}_p(\mathsf{aux}_i)\mathbf{s}_{q_i} + \mathbf{s}_{l'_i} - X^{c_v}\mathbf{o}_i \\
\mathbf{t}'_{\mathsf{nym}} &= \mathcal{H}_p(\mathsf{aux})\mathbf{s}_{x_1} + \mathbf{s}_e - X^{c_v}\mathsf{nym}
\end{aligned}$$

(2) checks $2||d_i - k_i|| < \Gamma$ where Γ is a function of β if true then outputs 0 otherwise 1.

$\mathsf{LM.Match}(\mathsf{gvk}, m, m', \mathsf{aux}, \tau, \tau') \in \{0, 1\}$

Parse τ and τ' and returns 1 if $||\tau.\mathsf{nym} - \tau'.\mathsf{nym}|| \leq 2\beta$ otherwise returns 0 for non-match state. The reader may notice a similar equation in [11] during the joining procedure.

B.3 Proofs

B.3.1 Unlinkability cf. Figure 2.

Theorem 4.

$$\mathsf{Adv}^{\mathsf{Unlink}}_{\mathsf{LM}} \leq \varepsilon^2 \cdot \mathsf{Adv}^{\mathsf{avgDRLWE-LoR}}$$

Proof. [$\mathbb{G}_0$:Orig.] In this original game $\mathbb{G}_0$, the adversary plays the experience $\mathsf{Exp}^{\mathsf{Unlink}}_{\mathsf{LM}}$, playing with oracle $\mathsf{oTagLoR}^b$ trying to guess b. We have the following probability :

$$\mathsf{Adv}^{\mathsf{Unlink}}_{\mathsf{LM}} = 2\left|\Pr[\mathrm{S}_0] - \frac{1}{2}\right|$$

[$\mathbb{G}_1$:Bridging] In this game $\mathbb{G}_1$ the user $U_b \in \{U_L, U_R\}$ uses the ${k_e}^L$. Since we place our selves in the QROM, therefore quering the hash function leads to choose a random element, choosing betwen either ${k_e}^L$ or ${k_e}^R$ result in a perfect indistinguishability. Hence the probability is:

$$\Pr[\mathrm{S}_1] = \Pr[\mathrm{S}_0]$$

[$\mathbb{G}_2$:Indig.] To bridge the unlinkability experiment with the decisional RLWE-based Left-or-Right game $\mathsf{Exp}_{\mathsf{avgDRLWE-LoR}}$ in the Quantum Random Oracle Model (QROM), we introduce a semi-constant random oracle, following the technique of Zhandry [22].

Let $\mathcal{A}$ be an adversary that succeeds in breaking the unlinkability property, and let $\mathcal{R}$ be a reduction that uses $\mathcal{A}$ to distinguish between left and right RLWE samples in the $\mathsf{Exp}_{\mathsf{avgDRLWE-LoR}}$ game. The reduction interacts with a challenger C who returns samples – either Left or Right, or according to the oracle to a specific s – encoded as quantum states.

The goal of $\mathcal{R}$ is to respond to tagging queries from $\mathcal{A}$, which involve messages of the form (m, aux, S), while embedding the sample (u, v) as part of the nym, such that the adversary cannot distinguish which distribution it came from. A key challenge is that the nym depends on a hash function evaluation $\mathcal{H}_p(\mathsf{aux})$, and the reduction does not control u in advance.

To resolve this, we instantiate the hash function $\mathcal{H}_p$ as a *semi-constant random oracle* $\mathcal{H}^{\epsilon}_p$, which behaves as follows:

- For a randomly selected subset $T \subset \mathcal{X}$ of inputs of size ϵ, we set $\mathcal{H}^{\epsilon}_p(x_i) = \bar{u}_i$ for all $x_i \in T$;
- For all other inputs $x \notin T$, $\mathcal{H}^{\epsilon}_p(x)$ is sampled uniformly at random.

The reduction measures the first register of the $\mathsf{avgDRLWE} - \mathsf{LoR}$ sample provided by the challenger in the form $|u\rangle|v = u \cdot s + e\rangle$, which collapses the full state to the classical value $(\bar{u}, \bar{v} = \bar{u} \cdot s + e)$. The reduction then uses this value $\bar{u}$ to program $\mathcal{H}^{\epsilon}_p$, ensuring that the oracle outputs $\bar{u}$ for a small, non-negligible fraction of the inputs.

Next, for each oracle query, the reduction creates a dedicated response oracle:

- $\mathcal{N}^{\epsilon}_i$ is used for the tagging oracle $\mathrm{oTag}(U_i, \cdot, \cdot, \cdot)$, which uses the embedding $(\bar{u}, \bar{v})$ in the simulated proof τ;
- $\mathcal{N}^{\epsilon}_{\mathsf{LoR},(i,j)}$ is used for the tagging oracle $\mathrm{oTagLoR}((U_i, U_j), \cdot, \cdot, \cdot)$, where the reduction uses $(\bar{u}, \bar{v})$ for the nym and constructs the unlinkable proof π using the second challenge sample $|u', v'\rangle$.

Each oracle $\mathcal{N}^{\epsilon}_i$ or $\mathcal{N}^{\epsilon}_{\mathsf{LoR},(i,j)}$ ensures that the mapping between $\bar{u}$ and $\bar{v}$ is consistent across queries with a chance of $\frac{1}{\epsilon}$ per queries to link it correctly.

Importantly, the second RLWE sample $|u', v'\rangle$, which is also obtained from the challenger, remains unmeasured and is used in simulating the zero-knowledge proof π under the assumption that the proof system is simulatable.

Since the adversary queries $\mathcal{H}^{\epsilon}_p$ in superposition, the reduction cannot directly observe these inputs. However, by Zhandry's analysis of semi-constant oracles it follows that the adversary has a non-negligible probability $\varepsilon = \frac{q_{\mathcal{H}_p}}{\epsilon}$ of querying a point $x \in T$, causing the reduction to embed the RLWE sample $(\bar{u}, \bar{v})$ in a way that makes it indistinguishable from a real RLWE challenge.

Thus, the adversary's advantage in distinguishing between the left and right nyms directly translates into a non-negligible advantage for the reduction in breaking the $\mathsf{avgDRLWE} - \mathsf{LoR}$ assumption. Hence, we obtain:

$$|\Pr[\mathrm{S}_2] - \Pr[\mathrm{S}_1]| = \varepsilon^2 \cdot \mathsf{Adv}^{\mathsf{avgDRLWE-LoR}}$$

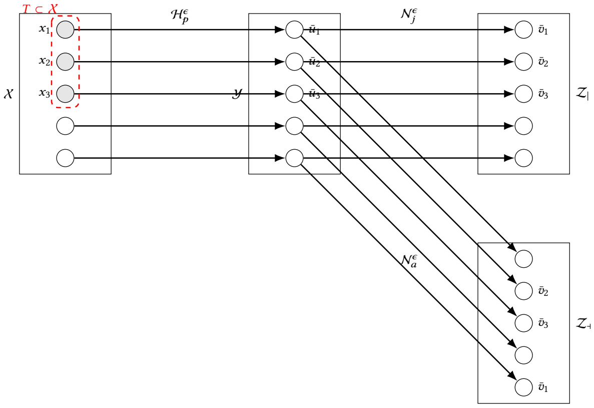


Figure 6: $\forall x_i \in T, \mathcal{H}_p^\epsilon \colon x_i \mapsto u_i \qquad \forall u_i \in Y, \mathcal{N}_j^\epsilon \colon u_i \mapsto v_i$

[$\mathbb{G}_2$:Final] The game $\mathbb{G}_2$ gives no more oportuninties to the adversary to win. Hence giving $\Pr[S_2] = \frac{1}{2}$.

□

B.3.2 *EUF-CMA-AD cf. Figure 1.*

Theorem 5.

$$\mathsf{Adv}_{\mathsf{LM}}^{EUF-CMA-AD} \leq \mathsf{Adv}^{\mathsf{PRF}} + \mathsf{Adv}^{\mathsf{RLWE}} + \mathsf{Adv}_{ZK}^{HVZK}$$

Sketch. [$\mathbb{G}_0$:Orig.] In this original game $\mathbb{G}_0$, the adversary plays the experience $\mathsf{Exp}_{\mathsf{LM}}^{\mathsf{Unlink}}$, playing with oracle $\mathsf{oTagLoR}^b$ trying to guess b. We have the following probability :

$$\mathsf{Adv}_{\mathsf{LM}}^{\mathsf{Unlink}} = \frac{1}{1-\epsilon} \left|\Pr[S_0] - \epsilon\right|$$

[$\mathbb{G}_1$:Indig.] We transform the hash oracle into a programable quantum random oracle this allows to latter simulates the proofs. The reduction $\mathcal{R}$ is in charge of this. The $\mathcal{A}$ notice the difference with the following probability:

$$|\Pr[S_1] - \Pr[S_0]| \leq \mathsf{Adv}^{\mathsf{PRF}}$$

[$\mathbb{G}_2$:Indig.] We use the Unruh's extractor from the underlining sigma protocol. Here we have two proofs τ that is 3-special sound and π that is special sound. Both of them would require to break the search-RLWE.

$$|\Pr[S_2] - \Pr[S_1]| \leq \mathsf{Adv}^{\mathsf{RLWE}}$$

[$\mathbb{G}_3$:Indig.] The game $\mathbb{G}_3$ differs from the previous game $\mathbb{G}_2$, by always simulating the ZKAoK (τ, π). This relies on $\mathcal{R}$ to simulate it, sending then to $\mathcal{A}$. This ones replies, implies that it can differentiate between the real and simulated one. In other words the probability is:

$$|\Pr[S_3] - \Pr[S_2]| = \mathsf{Adv}_{ZK}^{HVZK}$$

[$\mathbb{G}_3$:Final] Since the proofs are entirely simulated the $\mathcal{A}$ has no chances to win. Therefore the Theorem 5 is true. □

B.4 Hash-based

Here, we present a flavor that is based on the following work [9]. For the reader we boxed all modifications inside the functions.

$\mathtt{LM.Setup}(1^\lambda) \longrightarrow \mathsf{param}$

First the GA chooses parameters (d, k) for the hyper-tree M-FORS scheme and instanciates the following functions:

$$H_1 : \{0,1\}^* \longrightarrow \{0,1\}^n$$
$$H_2 : \{0,1\}^* \longrightarrow \{0,1\}^{d \cdot k}$$
$$H_2 : \{0,1\}^* \longrightarrow \{0,1\}^{d \cdot k + (\log_2 q) \cdot h}$$

And a keyed pseudo random function $F : \{0,1\}^n \times \{0,1\}^n \longrightarrow \{0,1\}^n$.

$\mathtt{LM.GenGroup}(\mathsf{param}) \longrightarrow (\mathsf{gmk}, \mathsf{gvk}, \mathsf{bsn})$

The GA generate the SPHINCS tree by executing $(\mathsf{sk}_{\mathsf{SPHINCS}}, rpk, gp) \longleftarrow \mathsf{F-SPHINCS+.keyGen(n, q, h)}$. The GA obtains the following:

$$\mathsf{gmk} = \mathsf{sk}_{\mathsf{SPHINCS}}$$
$$\mathsf{gvk} = (gp, rpk, H_1, H_2, H_3, F, \mathsf{prf})$$
$$gp = (n, q, h, d, k)$$
$$\mathsf{bsn} = H_1(rpk)$$

$\mathtt{LM.RegUser}(U, \mathsf{gmk}) \longrightarrow (\mathsf{ID}_U^{\mathsf{LM}}, \mathsf{sk}_U) \sqcup \bot$

First the user U chooses $x_U \overset{\$}{\longleftarrow} \{0,1\}^n$ and computes its identity with $\mathsf{ID}_U^{\mathsf{LM}} = F(x_U, \mathsf{bsn})$ (we recall that the user can recompute $\mathsf{bsn} = H_1(rpk)$) with its proof: $\pi_U : \mathcal{P}\{gp, \mathsf{bsn}, \mathsf{ID}_U^{\mathsf{LM}}; x_U : \mathsf{ID}_U^{\mathsf{LM}} = F(x_U, \mathsf{bsn})\}$. The user conveys $(\mathsf{ID}_U^{\mathsf{LM}}, \pi_U)$.

If $\mathsf{ID}_U^{\mathsf{LM}}$ is absent in the database (*i.e.*, haven't been registered before) then the GA proceeds as follows, first checks if the π_U is valid and then the GA computes the group credential $(gr_U, \Sigma) \longleftarrow \mathsf{F-SPHINCS+.sign}(\mathsf{ID}_U^{\mathsf{LM}} || gr_U, \mathsf{gmk}, gp)$ and adds $(\mathsf{ID}_U^{\mathsf{LM}}, gr_U, \Sigma)$ to its database (with $gr_U \overset{\$}{\longleftarrow} \{0,1\}^n$); otherwise rejects. The user if accepted, receives from GA (gr_U, Σ), and yeilds the following:

$$\mathsf{sk}_U = (x_U, gr_U, \Sigma)$$
$$\mathsf{ID}_U^{\mathsf{LM}} = F(x_U, \mathsf{bsn})$$

$\mathsf{LM.Tag}(\mathsf{sk}_U, m, \mathsf{aux}, S) \longrightarrow (\tau, \pi)$

The user generate an MPC proof that guarantees the following:

$$\begin{aligned}
&\tau : \mathcal{P}\{(gp, rpk, \mathsf{bsn}, \boxed{\mathsf{aux}}, \boxed{m}, \boxed{ssm}, \mathsf{nym}, com);\\
&\quad (x_U, \boxed{\mathsf{ID}_U^{\mathsf{LM}}}, gr_U, \boxed{sigm}, s, \Sigma = \{\sigma_h, \ldots, \sigma_0\}) :\\
&\quad \mathsf{nym} = F(x_U, \boxed{\mathsf{aux}})\\
&\quad \wedge \boxed{sigm = F(x_U, m)}\\
&\quad \wedge \boxed{ssm = F(sigm, \mathsf{nym})}\\
&\quad \wedge \boxed{\mathsf{ID}_U^{\mathsf{LM}}} = F(x_U, \mathsf{bsn})\\
&\quad \wedge mt_U || idx = H_3(\boxed{\mathsf{ID}_U^{\mathsf{LM}}} || gr_U)\\
&\quad \wedge pk_h = \mathsf{recoverPK}(\sigma_h, mt_U, (n, d, k, (h, idx)))\\
&\quad \wedge pk_{h-1} = \mathsf{recoverPK}(\sigma_{h-1}, pk_h, (n, d, k,\\
&\qquad\qquad\qquad (h-1, \lfloor \frac{idx}{q} \rfloor)))\\
&\quad \wedge \ldots\\
&\quad \wedge rpk = \mathsf{recoverPK}(\sigma_0, pk_1, (n, d, k, (0,0)))\\
&\quad \wedge com = H_1(s || pk_h || \ldots || rpk)\}
\end{aligned}$$

And from [9], we are able to seperate the proof that is used for S only:

$$\begin{aligned}
&\pi : \mathcal{P}\{(\ gp, \mathsf{aux}, \mathsf{nym}, r, \tau, \mathsf{nonce}\\
&\quad \{\forall j \in [\#S], (\mathsf{nym}_j, \mathsf{aux}_j) \in S, A_j\ \}\});\\
&\quad (x_U) :\\
&\quad r = H(S || \mathsf{nym} || \tau || \mathsf{nonce})\\
&\quad\ \wedge \mathsf{nym} = F(x_U, \mathsf{aux})\\
&\quad \bigwedge_{j \in [\#S]} \left(A_j = F(F(x_U, \mathsf{aux}_j), r)\right)\}
\end{aligned}$$

Where $r = H(S || \mathsf{nym} || \tau || \mathsf{nonce})$ allows to use it as a commitment over the tupple (S, τ).[13]

$\mathsf{LM.Ver}(\mathsf{gvk}, m, \mathsf{aux}, \tau) \in \{0, 1\}$

Check the proof τ.

$\mathsf{LM.Match}(\mathsf{gvk}, m, m', \mathsf{aux}, \tau, \tau) \in \{0, 1\}$

returns 1 if $\mathsf{nym} = \mathsf{nym}'$ otherwise returns 0 for non-match state.

$\mathsf{LM.MatchSet}(\mathsf{gvk}, m, \mathsf{aux}, \tau, \pi, S) \in \{0, 1\}$

First checks the $r = H(S || \mathsf{nym} || \tau || \mathsf{nonce})$. Then verifies the consistency of the proof π and checks $\pi.\mathsf{nym} = \tau.\mathsf{nym}$ and furthermore we have:

$$\begin{cases} 1 & \Leftarrow \exists j \in [\#S], A_j = F(s_j.\tau.\mathsf{nym}, r) \\ 0 & \text{otherwise} \end{cases}$$

where s_j being the j-th element of S.

C SECRET HANDSHAKES: BUILDING BLOCKS AND FORMALIZATION

C.1 Algorithms

Here we formally explicit each of the algorithms of the SHS primitive.

`DetectSelfDistinction`. Without a loss of generality, this subroutine enables a user to detect who tagged twice (at least) in the set $\mathcal{W}$. The following set is composed of a vector containing a pseudo-identity ID (*e.g.,* it can be linked to a nonce), a message m and, a tag τ, aiming to return a set containing pseudo-identities that misbehaved.

The complexity of this function is $O(n^2)$ calls to LM.Match with n the numbers of elements in $\mathcal{W}$.

```
$B \leftarrow$ DetectSelfDistinction$(\mathsf{gvk}, \mathsf{aux}, \mathcal{W})$
$B \leftarrow \emptyset$
foreach $e$ in $\mathcal{W}$:
    foreach $f$ in $\mathcal{W} \setminus \{e\}$:
        if LM.Match$(\mathsf{gvk}, e.m, f.m, \mathsf{aux}, e.\tau, f.\tau) = 1$
        $\wedge$LM.Ver$(\mathsf{gvk}, e.m, \mathsf{aux}, e.\tau) = 1$
        $\wedge$LM.Ver$(\mathsf{gvk}, f.m, \mathsf{aux}, f.\tau) = 1$ :
            $B \leftarrow B \cup \{e, f\}$
return $B$
```

`VerTr`. This function evaluates if the transcript was correctly generated by checking first if the tags are valid and also if all the users has accepted to communicate (represented by τ^{acc}).

```
$\{0,1\} \ni$ VerTr$(\mathsf{gvk}, \mathsf{tr} = (\{(\mathsf{state}_i, \tau_i, \tau_i^{\mathsf{acc}})_{i \in [[n]]}\}, \mathsf{sid}, \mathsf{nonce}_t^{\mathsf{sid}}))$
$\mathsf{acc} \leftarrow H(\mathsf{nonce}_t^{\mathsf{sid}} || \tau_1 || \mathsf{state}_1 || \ldots || \tau_n || \mathsf{state}_n)$
if $\emptyset \neq$ DetectSelfDistinction$(\mathsf{gvk}, \mathsf{sid},$
$\{(H(\mathsf{state}_i || \mathsf{acc}), \tau_i^{\mathsf{acc}})\}_{i \in [[n]]})$
    return 0
foreach $i \in [[n]]$
    $m_i \leftarrow H(\mathsf{state}_i || \mathsf{nonce}_t^{\mathsf{sid}})$
    $\mathsf{acc}_i \leftarrow H(\mathsf{state}_i || \mathsf{acc})$
    if LM.Ver$(\mathsf{gvk}, m_i, \mathsf{sid}, \tau_i) = 0$
    $\vee$LM.Ver$(\mathsf{gvk}, \mathsf{acc}_i, \mathsf{sid}, \tau_i^{\mathsf{acc}}) = 0$
    $\vee$LM.Match$(\mathsf{gvk}, \mathsf{acc}_i, m_i, \mathsf{sid}, \tau_i^{\mathsf{acc}}, \tau_i) = 0$ :
        return 0
return 1
```

`DetectTwiceID`. This function filters out all the identities that are present at least twice in a set, $\mathcal{W}$, of arbitrary lengthed vector with its first coordinate annotated as identity.

[13] Two attacks are feasible without those commited values: (1) Without this tweak, an adversary could get a valid (τ, π) for a specific S. Then the adversary could forge easily by creating a subset of S, let's call it S' and a fresh π' simply by removing the corresponding item hence to be valid with the set S'. Here the (r, nonce) is used to prevent this attack. (2) If we do not commit the value τ therefore, according to some security models, an adversary can call an oracle to tag twice to obtain two tupples (τ_0, π_0) and (τ_1, π_1) then the adversary can states to have forge a new valid tag with (τ_0, π_1) (or even (τ_1, π_0)) since it is not corresponding to any oracle output.

```
$C \leftarrow \texttt{DetectTwiceID}(\mathcal{W})$
$C \leftarrow \emptyset$
foreach $e$ in $\mathcal{W}$:
    foreach $f$ in $\mathcal{W} \setminus \{e\}$:
        if $e.\mathrm{ID} = f.\mathrm{ID}$
            $C \leftarrow C \cup \{e.\mathrm{ID}, f.\mathrm{ID}\}$
return $C$
```

`VerID`**.** This function aims to verify the validity of the identities in otherwords if there is a bound between $\mathsf{ID}^{\mathsf{CBU2}}$ and $\mathsf{ID}^{\mathsf{LM}}$.

```
$\{0,1\} \ni \texttt{VerID}(\mathsf{gvk}, \mathsf{bsn}, v)$
$\mathsf{CBU2}.\texttt{RecUCast}(v.\mathsf{sid}_{\mathsf{CBU2}}, v.\mathsf{sendK}, v.\mathsf{pk}^{\mathsf{CBU2}}, (v.c^{\sigma}, v.c, v.\mathsf{aux}))$
$\rightarrow (m, \tau, \pi)$
if $\neg[v.\tau = \tau^{id} \wedge v.\pi^{id} = \pi \wedge m = (v.\mathsf{ID}^{\mathsf{LM}} || v.\mathsf{ID}^{\mathsf{CBU2}} || v.\mathsf{pk}^{\mathsf{CBU2}})]$
    return 0
return [ $\mathsf{LM}.\texttt{Ver}(\mathsf{gvk}, m, \mathsf{bsn}, v.\tau^{id}) = 1$
  $\wedge \mathsf{LM}.\texttt{Ver}(\mathsf{gvk}, \perp, \mathsf{bsn}, v.\mathsf{ID}^{\mathsf{LM}}) = 1$
  $\wedge \mathsf{LM}.\texttt{Match}(\mathsf{gvk}, m, \perp, \mathsf{bsn}, \tau^{id}, \mathsf{ID}^{\mathsf{LM}}) = 1$ ]
```

`VerBan`**.** This function is a triplet of algorithms that verifies if the banishment has proceed correctly.

```
$\{0,1\} \ni \texttt{VerBan}^0(\mathsf{gvk}, \mathsf{bsn}, \mathsf{sid}_{\mathsf{ban}}, \mathsf{SRL}, \mathcal{W})$
foreach $e$ in $\mathcal{W}$:
    // Checks if the user is revoked
    // Then checks if the link to CBU2 is correct
    if $\neg$[ $\mathsf{LM}.\texttt{Ver}(\mathsf{gvk}, e.m, \mathsf{sid}_{\mathsf{ban}}, e.\tau) = 1$
    $\wedge$ $\mathsf{LM}.\texttt{MatchSet}(\mathsf{gvk}, e.m, \mathsf{sid}_{\mathsf{ban}}, e.\tau, e.\pi, \mathsf{SRL}) = 1$
    $\wedge$ $\mathsf{CBU2}.\texttt{RecBCast}(\mathsf{sid}_{\mathsf{CBU2}}, e.\mathsf{sendK}, e.pk^{\mathsf{CBU2}}, (e.c^{\sigma}, e.c, e.\mathsf{aux}))$
      $= (e.m, e.\tau, e.\pi)$
    $\wedge$ $\texttt{VerID}(\mathsf{gvk}, \mathsf{bsn}, e.DB[e.\mathsf{ID}^{\mathsf{CBU2}}]) = 1$
    $\wedge e.\mathsf{pk}^{\mathsf{CBU2}} = e.DB[e.\mathsf{ID}^{\mathsf{CBU2}}]$
        return 0
return 1
```

```
$(\{0,1\}, \mathsf{SRL}) \leftarrow \texttt{VerBan}^1(\mathsf{gvk}, \mathsf{bsn}, \mathsf{SRL}, \mathcal{W})$
$s \leftarrow 1$
foreach $(e, f)$ in $\mathcal{W}$:
    if $\mathsf{LM}.\texttt{Ver}(\mathsf{gvk}, e.m, e.\mathsf{sid}_{\mathsf{ban}}, e.\tau) = 1$
    $\wedge$ $\mathsf{LM}.\texttt{Ver}(\mathsf{gvk}, f.m, e.\mathsf{sid}_{\mathsf{ban}}, f.\tau) = 1$
    $\wedge \mathsf{LM}.\texttt{MatchSet}(\mathsf{gvk}, e.m, e.\mathsf{sid}_{\mathsf{ban}}, e.\tau, e.\pi, \mathsf{SRL}) = 0$
    $\wedge \mathsf{LM}.\texttt{MatchSet}(\mathsf{gvk}, f.m, e.\mathsf{sid}_{\mathsf{ban}}, f.\tau, f.\pi, \mathsf{SRL}) = 0$
    $\wedge$ $\mathsf{LM}.\texttt{Match}(\mathsf{gvk}, e.m, f.m, e.\mathsf{sid}_{\mathsf{ban}}, e.\tau, f.\tau) = 1$
    $\wedge$ $\mathsf{CBU2}.\texttt{RecBCast}(\mathsf{sid}_{\mathsf{CBU2}}, e.\mathsf{sendK}, e.pk^{\mathsf{CBU2}}, (e.c^{\sigma}, e.c, e.\mathsf{aux}))$
      $= (e.m, e.\tau, e.\pi)$
    $\wedge$ $\mathsf{CBU2}.\texttt{RecBCast}(\mathsf{sid}_{\mathsf{CBU2}}, f.\mathsf{sendK}, f.pk^{\mathsf{CBU2}}, (f.c^{\sigma}, f.c, f.\mathsf{aux}))$
      $= (f.m, f.\tau, f.\pi)$
    $\wedge$ $\texttt{VerID}(\mathsf{gvk}, \mathsf{bsn}, e.DB[e.\mathsf{ID}^{\mathsf{CBU2}}]) = 1$
    $\wedge$ $\texttt{VerID}(\mathsf{gvk}, \mathsf{bsn}, f.DB[f.\mathsf{ID}^{\mathsf{CBU2}}]) = 1$
    $\wedge$ $e.\mathsf{pk}^{\mathsf{CBU2}} \neq f.\mathsf{pk}^{\mathsf{CBU2}}$
    $\wedge$ $e.\tau \neq f.\tau$ :
        $\mathsf{SRL} \leftarrow \mathsf{SRL} \cup \{(e.\mathsf{ID}^{\mathsf{LM}}, \mathsf{bsn}), (f.\mathsf{ID}^{\mathsf{LM}}, \mathsf{bsn}), (e.\tau, e.\mathsf{sid}_{\mathsf{ban}})\}$
    else
        $s \leftarrow 0$
return $(s, \mathsf{SRL})$
```

```
$(\{0,1\}, \mathsf{SRL}) \leftarrow \texttt{VerBan}^2(\mathsf{gvk}, \mathsf{bsn}, \mathsf{SRL}, \mathcal{W})$
$s \leftarrow 1$
foreach $(e, f)$ in $\mathcal{W}$:
    if $\mathsf{LM}.\texttt{Ver}(\mathsf{gvk}, e.m, e.\mathsf{sid}_{\mathsf{ban}}, e.\tau) = 1$
    $\wedge$ $\mathsf{LM}.\texttt{Ver}(\mathsf{gvk}, f.m, e.\mathsf{sid}_{\mathsf{ban}}, f.\tau) = 1$
    $\wedge \mathsf{LM}.\texttt{MatchSet}(\mathsf{gvk}, e.m, e.\mathsf{sid}_{\mathsf{ban}}, e.\tau, e.\pi, \mathsf{SRL}) = 0$
    $\wedge \mathsf{LM}.\texttt{MatchSet}(\mathsf{gvk}, f.m, e.\mathsf{sid}_{\mathsf{ban}}, f.\tau, f.\pi, \mathsf{SRL}) = 0$
    $\wedge$ $\mathsf{LM}.\texttt{Match}(\mathsf{gvk}, e.m, f.m, e.\mathsf{sid}_{\mathsf{ban}}, e.\tau, f.\tau) = 0$
    $\wedge$ $\mathsf{CBU2}.\texttt{RecBCast}(\mathsf{sid}_{\mathsf{CBU2}}, e.\mathsf{sendK}, e.pk^{\mathsf{CBU2}}, (e.c^{\sigma}, e.c, e.\mathsf{aux}))$
      $= (e.m, e.\tau, e.\pi)$
    $\wedge$ $\mathsf{CBU2}.\texttt{RecBCast}(\mathsf{sid}_{\mathsf{CBU2}}, f.\mathsf{sendK}, f.pk^{\mathsf{CBU2}}, (f.c^{\sigma}, f.c, f.\mathsf{aux}))$
      $= (f.m, f.\tau, f.\pi)$
    $\wedge$ $e.\mathsf{pk}^{\mathsf{CBU2}} = f.\mathsf{pk}^{\mathsf{CBU2}} \wedge e.m \neq f.m \wedge e.\tau \neq f.\tau$ :
        $\mathsf{SRL} \leftarrow \mathsf{SRL} \cup \{(e.\mathsf{ID}^{\mathsf{LM}}, \mathsf{bsn}), (f.\mathsf{ID}^{\mathsf{LM}}, \mathsf{bsn}),$
            $(e.\tau, e.\mathsf{sid}_{\mathsf{ban}}), (f.\tau, e.\mathsf{sid}_{\mathsf{ban}})\}$
    else
        $s \leftarrow 0$
return $(s, \mathsf{SRL})$
```

Remark 3. *We notice that the* GA *can collude with a malicious user. We still keep manage to obtain* Self distinction *since it relies mainly on the revocation list. But for the corrupted user there is still a possibility to frame honest users as corrupted since the* GA *openning the challenge can resend it to a malicious user in order to re-wrap it. Making it seems corrupted even though it isn't. Which does not hurt the definition of* Self distinction*.*

C.2 Secret handshakes from list MACs

This appendix provides a full, formal description of the LCA scheme.

C.2.1 Protocol details. SHS.Setup**.** Formally, the following steps are executed: $\mathsf{LM}.\texttt{Setup}(1^{\lambda}) \rightarrow \mathsf{param}_{\mathsf{ListMAC}}$, $\mathsf{CBU2}.\texttt{Setup}(1^{\lambda})$

```
$U$
// Interactive algorithm between GA & $U$ to create a ListMAC key
// $\mathsf{ID}_U^{\mathsf{LM}}$ is known by $U$ & GA ; $\mathsf{sk}_U$ is only known by $U$
$(\mathsf{ID}_U^{\mathsf{LM}}, \mathsf{sk}_U) \leftarrow \mathsf{LM.RegUser}(U, \mathsf{gmk})$
// Generate credential for secure channel
$(\mathsf{sk}_U^{\mathsf{CBU2}}, \mathsf{pk}_U^{\mathsf{CBU2}}) \leftarrow \mathsf{CBU2.NewCred}(\text{“user”})$
GA
// Interactive algorithm between GA and $U$ to register the user
$(\mathsf{ID}_U^{\mathsf{CBU2}}, \mathsf{status}) \leftarrow \mathsf{CBU2.RegCred}(U(<\mathsf{sk}_U^{\mathsf{CBU2}}, \mathsf{pk}_U^{\mathsf{CBU2}}>),$
    $\mathsf{CM}(<\mathsf{CM.sk}, \mathsf{CM.pk}>))$
if status $\neq$ OK:
    $\mathsf{SRL} \leftarrow \mathsf{SRL} \cup \{(\mathsf{ID}_U^{\mathsf{LM}}, \mathsf{bsn})\}$
    return $\perp$
if $\mathsf{USet}_G = \emptyset$ :
    $\mathsf{CBU2.ChInit}(\{U\}, \mathsf{CM.sk})$
      $\rightarrow (<\mathsf{ms}^0_{\mathsf{sid}_{\mathsf{CBU2}}}, \mathsf{mpk}^0_{\mathsf{sid}_{\mathsf{CBU2}}}, \mathsf{sid}_{\mathsf{CBU2}}, \mathsf{sendK}_U>,$
      $<\mathsf{sid}_{\mathsf{CBU2}}, \mathsf{ms}^0_{\mathsf{sid}_{\mathsf{CBU2}}}, \mathsf{mpk}^0_{\mathsf{sid}_{\mathsf{CBU2}}}, \mathsf{sid}_{\mathsf{CBU2}},$
      $\{\mathsf{sendK}_i\}_{i \in \mathsf{USet}_G}>)$
    // The user and GA are the only one to share $\mathsf{sendK}_U$
else :
    //For sake of readability we do not details the output
    $\mathsf{CBU2.UAdd}(\mathsf{sid}_{\mathsf{CBU2}}, \mathsf{CM.sk}, U)$
$\mathsf{CBU2.ChUpdate}(\mathsf{sid}_{\mathsf{CBU2}}, \mathsf{CM.sk})$
$U$
//This aims to build a strong bound between $\mathsf{ID}_U^{\mathsf{CBU2}}$ and $\mathsf{ID}_U^{\mathsf{LM}}$
$(\tau_U^{id}, \pi_U^{id}) \leftarrow \mathsf{LM.Tag}(\mathsf{sk}_U, (\mathsf{ID}_U^{\mathsf{LM}} || \mathsf{ID}_U^{\mathsf{CBU2}} || \mathsf{pk}_U^{\mathsf{CBU2}}), \mathsf{bsn}, \mathsf{SRL})$
$(c^\sigma, c, \mathsf{aux}) \leftarrow \mathsf{CBU2.UCast}(\mathsf{sid}_{\mathsf{CBU2}}, \mathsf{sendK}_U, \mathsf{sk}_U,$
    $((\mathsf{ID}_U^{\mathsf{LM}} || \mathsf{ID}_U^{\mathsf{CBU2}} || \mathsf{pk}_U^{\mathsf{CBU2}}), \tau_U^{id}, \pi_U^{id}))$
GA
$\mathsf{CBU2.RecUCast}(\mathsf{sid}_{\mathsf{CBU2}}, \mathsf{sendK}_U, \mathsf{pk}_U^{\mathsf{CBU2}}, (c^\sigma, c, \mathsf{aux}))$
    $\rightarrow (m, \tau_U^{id}, \pi_U^{id}))$
//Checking bounding
if $(\mathsf{ID}_U^{\mathsf{LM}} || \mathsf{ID}_U^{\mathsf{CBU2}} || \mathsf{pk}_U^{\mathsf{CBU2}}) = m$
$\wedge \mathsf{LM.Ver}(\mathsf{gvk}, m, \mathsf{bsn}, \tau_U^{id}) = 1$
$\wedge \mathsf{LM.Match}(\mathsf{gvk}, m, \perp, \mathsf{bsn}, \tau_U^{id}, \mathsf{ID}_U^{\mathsf{LM}}) = 1$
$\wedge \mathsf{LM.MatchSet}(\mathsf{gvk}, m, \mathsf{bsn}, \tau_U^{id}, \pi_U^{id}, \mathsf{SRL}) = 0$ :
    $DB[\mathsf{ID}_U^{\mathsf{CBU2}}] \leftarrow (c^\sigma, c, \mathsf{aux}, \mathsf{pk}_U^{\mathsf{CBU2}}, \mathsf{sendK}_U)$
else :
    $\mathsf{CBU2.URmv}(\mathsf{sid}_{\mathsf{CBU2}}, \mathsf{CM.sk}, U)$
    //Removing the $\mathsf{ID}_U^{\mathsf{LM}}$ since GA may recieve $\perp$ or random
    $\mathsf{SRL} \leftarrow \mathsf{SRL} \cup \{(\mathsf{ID}_U^{\mathsf{LM}}, \mathsf{bsn})\}$
```

Figure 7: SHS.Join: **User's registration in** LCA

$\rightarrow \mathsf{param}_{\mathsf{CBU2}} = (\mathsf{spar}_{\mathsf{CBU2}}, \mathsf{ppar}_{\mathsf{CBU2}})$, $\mathsf{AGKA\text{-}FR.Setup}(1^\lambda, \mathcal{R}) \rightarrow (\mathsf{ppar}_{\mathsf{AGKA\text{-}FR}}, \mathcal{K}, \mathcal{R}^\Pi) = \mathsf{param}_{\mathsf{AGKA\text{-}FR}}$ and returns $\mathsf{param} \leftarrow (\mathsf{param}_{\mathsf{ListMAC}}, \mathsf{param}_{\mathsf{CBU2}}, \mathsf{param}_{\mathsf{AGKA\text{-}FR}})$. *We recall to the reader, that $\mathcal{R}$ here needs to be the same amongst all the group because it describes the type of randomness.*

```
GA
// Choose random message
$m \xleftarrow{\$} \{0,1\}^*$
// Broadcast LEAVE || m
$\mathsf{CBU2.BCast}(\mathsf{sid}_{\mathsf{CBU2}}, \mathsf{CM.sk}, \mathsf{ms}^t_{\mathsf{sid}}, (\text{“LEAVE"} || m))$
$U_A$
Parse broadcast
$(\text{“LEAVE"} || m') \leftarrow \mathsf{CBU2.RecBCast}(\mathsf{sid}_{\mathsf{CBU2}}, \mathsf{ms}^t_{\mathsf{sid}_{\mathsf{CBU2}}}, (c, \mathsf{aux}))$
// Tag LEAVE || m and unicast result
$(\tau, \pi) \leftarrow \mathsf{LM.Tag}(\mathsf{sk}_A, m', \mathsf{bsn}, \mathsf{SRL})$
$\mathsf{CBU2.UCast}(\mathsf{sid}_{\mathsf{CBU2}}, \mathsf{sendK}_A, (\tau, \pi))$
GA
// Parse unicast, trace user ID in CBU2
$(\tau, \pi) \leftarrow \mathsf{CBU2.RecUCast}(\mathsf{sid}_{\mathsf{CBU2}}, \mathsf{sendK}_A, (c, \mathsf{aux}))$
$\mathsf{ID}_A^{\mathsf{LM}} \leftarrow DB[\mathsf{ID}_A^{\mathsf{CBU2}}]$
// Remove user from CBU2
$\mathsf{CBU2.URmv}(\mathsf{sid}_{\mathsf{CBU2}}, \mathsf{CM.sk}, \mathsf{ID}_A^{\mathsf{CBU2}})$
// Update banned lists
$\mathsf{SRL}' \leftarrow \mathsf{SRL}$
$\mathsf{SRL} \leftarrow \mathsf{SRL} \cup \{(\mathsf{ID}_A^{\mathsf{LM}}, \mathsf{bsn})\}$
// Check tag validity, check matching
if $\mathsf{LM.Ver}(\mathsf{gvk}, \text{“LEAVE"} || m, \mathsf{bsn}, \tau) = 1$
$\wedge \mathsf{LM.Match}(\mathsf{gvk}, \perp, \text{“LEAVE"} || m, \mathsf{bsn}, \mathsf{ID}_A^{\mathsf{LM}}, \tau) = 0$ :
    $\mathsf{SRL} \leftarrow \mathsf{SRL} \cup \{(\tau, \mathsf{bsn})\}$
    // Detect the true issuer
    $E \leftarrow \{\mathsf{ID}^{\mathsf{LM}} : \exists \mathsf{ID}^{\mathsf{LM}}, \mathsf{ID}^{\mathsf{LM}} \neq \mathsf{ID}_A^{\mathsf{LM}}$
    $\wedge \mathsf{LM.Match}(\mathsf{gvk}, \perp, \text{“LEAVE"} || m, \mathsf{bsn}, \mathsf{ID}^{\mathsf{LM}}, \tau) = 1\}$
    foreach $\mathsf{ID}^{\mathsf{LM}} \in E$
        $\mathsf{CBU2.URmv}(\mathsf{sid}_{\mathsf{CBU2}}, \mathsf{CM.sk}, DB[\mathsf{ID}^{\mathsf{LM}}])$
    $\mathsf{SRL} \leftarrow \mathsf{SRL} \cup E$
$\mathsf{CBU2.BCast}(\mathsf{sid}_{\mathsf{CBU2}}, \mathsf{CM.sk}, \mathsf{ms}^t_{\mathsf{sid}_{\mathsf{CBU2}}}, (\mathsf{KRL}, \mathsf{SRL}))$
```

Figure 8: SHS.Leave: **Leave in** LCA

SHS.NewGroup**.** The GA creates a new group G. First the GA executes $\mathsf{CBU2.NewCred(manager)} \rightarrow (\mathsf{CM.sk}, \mathsf{CM.pk})$ and runs $(\mathsf{gmk}, \mathsf{gvk}, \mathsf{bsn}) \leftarrow \mathsf{LM.GenGroup}(\mathsf{param})$ to obtain the master key gmk kept secret by the GA, the group verification key gvk known to the members, and bsn the base name of the group (used later on for identification). The GA initializes $\mathsf{KRL} = \emptyset$, where KRL is an (initially empty) set, which will store private keys of banned users. Similarly, for SRL, the list of tags of revoked users. Finally, the GA chooses an initial random nonce $\mathsf{nonce}_G^t \xleftarrow{\$} \{0,1\}^*$, which is a global group specific nonce which will be updated regularly (whenever users choose to Join or Leave, when they are banned, or simply when the GA performs an Update operation). The GA creates a random $\mathsf{sid}_{\mathsf{ban}}$ *later used during banishment.*

SHS.Update**.** First the GA runs $\mathsf{CBU2.ChUpdate}(\mathsf{sid}, \mathsf{CM.sk})$, then sends the following updated values through CBU2.BCast: $t \leftarrow t+1$ and $\mathsf{nonce}_G^t \xleftarrow{\$} \{0,1\}^*$.

```
GA
if $\mathsf{VerTr}(\mathsf{gvk}, \mathsf{tr}) = 0$
    return $\perp$
$\mathsf{SRL} \leftarrow \mathsf{SRL} \cup \{(\mathsf{tr}.\tau_T, \mathsf{tr}.\mathsf{sid}_T)\}$
$\mathsf{sid}_{\mathsf{ban}} \leftarrow H(\mathsf{sid}_{\mathsf{ban}} || \mathsf{tr}.\mathsf{sid}_T)$
$\mathsf{CBU2.BCast}(\mathsf{sid}_{\mathsf{CBU2}}, \mathsf{CM.sk}, \mathsf{ms}^t_{\mathsf{sid}}, (\mathsf{sid}_{\mathsf{ban}}, \mathsf{tr}, \mathsf{tr}.\tau_T, \mathsf{tr}.\mathsf{sid}_T))$
  $\rightarrow (c, \mathsf{aux})$

$U_A$
$\mathsf{CBU2.RecBCast}(\mathsf{sid}_{\mathsf{CBU2}}, \mathsf{ms}^t_{\mathsf{sid}_{\mathsf{CBU2}}}, (c, \mathsf{aux}))$
  $\rightarrow (\mathsf{sid}'_{\mathsf{ban}}, \mathsf{tr}', \tau'_T, \mathsf{sid}'_T)$
// Check transcript validity and the new $\mathsf{sid}_{\mathsf{ban}}$
if $\mathsf{VerTr}(\mathsf{gvk}, \mathsf{tr}) = 0 \vee \tau'_T \notin \{\mathsf{tr}.\tau_*\}$
    return $\perp$
if $\mathsf{sid}' = \mathsf{bsn} \wedge H(\mathsf{sid}_{\mathsf{ban}} || \mathsf{sid}') \neq \mathsf{sid}'_{\mathsf{ban}}$
    return $\perp$
$(\mathsf{SRL}, \mathsf{sid}_{\mathsf{ban}}) \leftarrow (\mathsf{SRL} \cup \{(\tau_T, \mathsf{sid}_T)\}, \mathsf{sid}'_{\mathsf{ban}})$
$m \overset{\$}{\leftarrow} \{0,1\}^*$
// Tag with the updated $\mathsf{SRL}$
$(\tau_A, \pi_A) \leftarrow \mathsf{LM.Tag}(\mathsf{sk}_A, m, \mathsf{sid}_{\mathsf{ban}}, \mathsf{SRL})$
$\mathsf{CBU2.UCast}(\mathsf{sid}_{\mathsf{CBU2}}, \mathsf{sendK}_A, \mathsf{sk}^{\mathsf{CBU2}}_A, (m, \tau_A, \pi_A))$
  $\rightarrow (c^{\sigma}_A, c_A, \mathsf{aux}_A)$

GA
$E \leftarrow \emptyset$
$K \leftarrow \{(0, m', \tau) : \forall \mathsf{sk}_b \in \mathsf{KRL}(\tau,) \leftarrow \mathsf{LM.Tag}(\mathsf{sk}_b, m', \mathsf{sid}_{\mathsf{ban}}, \emptyset)\}$
foreach $U_i \in \mathcal{UG}^{\mathsf{CBU2}}_{\mathsf{sid}_{\mathsf{CBU2}}}$
    $\mathsf{CBU2.RecUCast}(\mathsf{sid}_{\mathsf{CBU2}}, \mathsf{sendK}_i, \mathsf{pk}^{\mathsf{CBU2}}_i, (c^{\sigma}_i, c_i, \mathsf{aux}_i))$
      $\rightarrow (m_i, \tau_i, \pi_i)$
    if $\mathsf{LM.Ver}(\mathsf{gvk}, m_i, \mathsf{sid}_{\mathsf{ban}}, \tau_i) = 1$
    $\wedge \mathsf{DetectSelfDistinction}(\mathsf{gvk}, \mathsf{sid}_{\mathsf{ban}}, K \cup \{(m_i, \tau_i, \pi_i)\}) = \emptyset$:
        $E \leftarrow E \cup \{(\mathsf{ID}^{\mathsf{CBU2}}_i, m_i, \tau_i, c^{\sigma}_i, c_i, \mathsf{aux}_i, \mathsf{pk}^{\mathsf{CBU2}}_i, \mathsf{sendK}_i,$
          $DB[\mathsf{ID}^{\mathsf{CBU2}}_i])\}$
// Remove all those can't tag properly, those that are revoked
$F \leftarrow \{e \in E : \mathsf{LM.MatchSet}(\mathsf{gvk}, e.m, \mathsf{sid}_{\mathsf{ban}}, e.\tau, e.\pi, \mathsf{SRL}) = 1)\}$
// Set of absent users
$A \leftarrow \mathcal{UG}^{\mathsf{CBU2}}_{\mathsf{sid}_{\mathsf{CBU2}}} \setminus \{e.\mathsf{ID}^{\mathsf{CBU2}} : e \in E\}$
// Determine those users who were corrupted or that colluded
$B \leftarrow \mathsf{DetectSelfDistinction}(\mathsf{gvk}, \mathsf{sid}_{\mathsf{ban}}, E)$   // via ListMAC
$C \leftarrow \mathsf{DetectTwiceID}(E)$  // via CBU2
$\mathsf{CBU2.BCast}(\mathsf{sid}_{\mathsf{CBU2}}, \mathsf{CM.sk}, \mathsf{ms}^t_{\mathsf{sid}}, (F, B, C))$
foreach $\mathsf{ID}^{\mathsf{CBU2}} \in F \cup A \cup B \cup C$:  // Remove users from CBU2
    $\mathsf{CBU2.URmv}(\mathsf{sid}_{\mathsf{CBU2}}, \mathsf{CM.sk}, \mathsf{ID}^{\mathsf{CBU2}})$
$\mathsf{SRL} \leftarrow \mathsf{SRL} \cup \{(e.\mathsf{ID}^{\mathsf{LM}}, \mathsf{bsn}), (f.\mathsf{ID}^{\mathsf{LM}}, \mathsf{bsn}),$
  $(e.\tau, \mathsf{sid}_{\mathsf{ban}}) : \forall (e, f) \in B\}$
$\mathsf{SRL} \leftarrow \mathsf{SRL} \cup \{(e.\mathsf{ID}^{\mathsf{LM}}, \mathsf{bsn}), (f.\mathsf{ID}^{\mathsf{LM}}, \mathsf{bsn}), (e.\tau, \mathsf{sid}_{\mathsf{ban}}),$
  $(f.\tau, \mathsf{sid}_{\mathsf{ban}}) : \forall (e, f) \in F\}$

$U_A$
$(F, B, C) \leftarrow \mathsf{CBU2.RecBCast}(\mathsf{sid}_{\mathsf{CBU2}}, \mathsf{ms}^t_{\mathsf{sid}_{\mathsf{CBU2}}}, (c', \mathsf{aux}))$
// Convincing that the user is the traitor
if $0 = \mathsf{VerBan}^0(\mathsf{gvk}, \mathsf{bsn}, \mathsf{sid}_{\mathsf{ban}}, \mathsf{SRL}, F)$:
    return $\perp$
// Convincing that the user is corrupted
$(s_1, \mathsf{SRL}) \leftarrow \mathsf{VerBan}^1(\mathsf{gvk}, \mathsf{bsn}, \mathsf{sid}_{\mathsf{ban}}, \mathsf{SRL}, B)$
$(s_2, \mathsf{SRL}) \leftarrow \mathsf{VerBan}^2(\mathsf{gvk}, \mathsf{bsn}, \mathsf{sid}_{\mathsf{ban}}, \mathsf{SRL}, B)$:
if $0 = s_1 \vee 0 = s_2$
    return $\perp$
```

Figure 9: SHS.Ban**: Banishment in** LCA

SHS.Join**.** (*cf.* Figure 7) The GA interacts with the user U over a secure channel. During this interaction they both generate the user's ListMAC secret key and ListMAC identity $(\mathsf{ID}^{\mathsf{LM}}, \mathsf{sk}_U) \leftarrow \mathsf{LM.RegUser}(U, \mathsf{gmk})$. Recall that sk_U is only known to the user ; the values $\mathsf{ID}^{\mathsf{LM}}$ and gvk are known by the user and the GA. Note that joining fails if a user tries to re-join the group.

Afterwards, GA needs to add the user to the corresponding CBU2 channel. User U generates credentials $(\mathsf{sk}^{\mathsf{CBU2}}_U, \mathsf{pk}^{\mathsf{CBU2}}_U) \leftarrow \mathsf{CBU2.NewCred}(\text{“user”})$. The user then registers to the GA : $\mathsf{CBU2.RegCred}(U(<\mathsf{sk}^{\mathsf{CBU2}}_U, \mathsf{pk}^{\mathsf{CBU2}}_U>), \mathsf{CM}(<\mathsf{CM.sk}, \mathsf{CM.pk}>)) \rightarrow (\mathsf{ID}^{\mathsf{CBU2}}_U, \mathsf{OK})$.

One of the following cases holds:

(1) **If** $\mathsf{USet}_G = \emptyset$, GA creates and initiates the CBU2 session $\mathsf{CBU2.ChInit}(\{U\}, \mathsf{CM.sk}) \rightarrow (<\mathsf{ms}^0_{\mathsf{sid}_{\mathsf{CBU2}}}, \mathsf{mpk}^0_{\mathsf{sid}_{\mathsf{CBU2}}}, \mathsf{sid}_{\mathsf{CBU2}}, \mathsf{sendK}_1>, <\mathsf{sid}_{\mathsf{CBU2}}, \mathsf{ms}^0_{\mathsf{sid}_{\mathsf{CBU2}}}, \mathsf{mpk}^0_{\mathsf{sid}_{\mathsf{CBU2}}}, \mathsf{sid}_{\mathsf{CBU2}}, \{\mathsf{sendK}_1\}>)$;

(2) **Otherwise**, the GA adds the user to the corresponding $\mathsf{sid}_{\mathsf{CBU2}}$ *i.e.,* $\mathsf{CBU2.UAdd}(\mathsf{sid}_{\mathsf{CBU2}}, \mathsf{CM.sk}, U)$.

In both cases broadcast and unicast keys are generated. Then the authority updates the channel.

Now, the goal is to create a strong link between $\mathsf{ID}^{\mathsf{CBU2}}_U$ and $\mathsf{ID}^{\mathsf{LM}}_U$. To do this, the idea is that the user certifies $\mathsf{pk}^{\mathsf{CBU2}}_U$ using their ListMAC key and then sends it. This results in the certificate being certified with $\mathsf{pk}^{\mathsf{CBU2}}_U$. Thus, the user tags $(\mathsf{ID}^{\mathsf{LM}}_U, \mathsf{ID}^{\mathsf{CBU2}}_U, \mathsf{pk}^{\mathsf{CBU2}}_U)$ for the auxiliary value bsn and sends it via unicast to the GA. From there, the GA verifies that the tag matches the identity. If that is the case, then it adds to its database what it has received, namely $(c^{\sigma}, c, \mathsf{aux}, \mathsf{sendK}_U)$, since it is already certified with $\mathsf{sk}^{\mathsf{CBU2}}_U$. If it fails then removes the user, further details in Figure 7.

SHS.Leave**.** (*cf.* Figure 8) *If a user A wants to leave a group voluntarily, it is needed either to reveal his, her, or their ListMAC secret key (and the* GA *adds it in the* KRL *set and broadcast it) or either by adding one of his, her, or their tag to the* SRL*. The first method isn't enough private since it reveals to the lasting user if they already exchanged with this user or not in the past* [14]*. We choose to focus on the second solution to obtain a strong privacy.*

After receiving an ACK message for leaving, the GA creates a random message m and broadcast it to every user prefixed with “LEAVE". The users that would like to leave just needs to tag the message with LM.Tag for the special auxiliary set to bsn and sends it to GA. Since the GA is able to link the identity $\mathsf{ID}^{\mathsf{CBU2}}_A$ of those interesting to leave to the corresponding $\mathsf{ID}^{\mathsf{LM}}_A$. For safety purpose there is a need to immediately remove the one whose identity is linked to CBU2 (also meaning to add his, her or their $\mathsf{ID}^{\mathsf{LM}}$in the SRL), because it implies that if the user didn't give to order, then the user is partially corrupted. To avoid any replay attack, we need to check if the tag is valid, meaning, the tag is verified and that the tag isn't issued from a user that already have one of his, her or their instance in the SRL if it is the case GA return $\perp$; this is done to punish all kind of attackers who would like to leave the group with someone else without his, her, or their consent. Lastly if the tag is valid but doesn't correspond to the ListMAC identity linked

[14] Even though this use case can be helpful in a specific context, where the user can learn if they have exchanged with it and if they receives false information for example in more concrete scenario.

to the CBU2 one, then the GA tries to find the corresponding user ListMAC identity to the CBU2 identity one, trivially by matching with all the ListMAC identities in the database. Nevertheless, the τ is used is added in SRL plus all the corresponding identities. Again all the matched identities are removed of the CBU2. Finally, GA updates, by broadcasting, the new KRL, SRL) tuple.

$\mathsf{SHS.Ban}$. (*cf.* Figure 9) The GA receives a complaint of a user, which is composed of a transcript tr and the specific τ_T that is wished to be banned, implicitly labeling the targeted user to ban as T. First we ensure that the transcript is valid. Here the protocol will ban the user based on the TRAITOR CATCHING method by using the revoked tags list SRL, as follows:

(1) the GA adds a tag issued from a banned user in SRL.
(2) the GA sends to all members the values to complete the challenge; the GA sends the tuple $(\mathsf{sid}_{\mathsf{ban}}, \mathsf{tr}, \tau_T, \mathsf{sid}_T)$. Those values are defined as follows: $\mathsf{sid}_{\mathsf{ban}} \leftarrow H(\mathsf{sid}_{\mathsf{ban}}||\mathsf{tr}.\mathsf{sid}_T)$, which prevents the GA from misbehaving to be curious, in fact this is in a way used as a commitment,
(3) therefore users act only if the $\mathsf{sid}_{\mathsf{ban}}$ was correctly generated and if the received tr is valid and that the (τ_T, sid_T) are present in the tr. If so, the users add the tuple (τ_T, sid_T) in SRL.
(4) the members have to tag a dummy message with the $\mathsf{sid}_{\mathsf{ban}}$ and send it back to the GA *i.e.,* $m_i \xleftarrow{\$} \{0,1\}^*$; $\mathsf{LM.Tag}(\mathsf{sk}_i, m_i, \mathsf{sid}_{\mathsf{ban}}, \mathsf{SRL}) \longrightarrow (\tau_i, \pi_i)$
(5) the GA keep the user in the group if the response is successful, which means that the tag is correctly generated (which also means that the tagger does not possess any element in SRL nor his, her or their secret key in KRL) and doesn't match with any other tags issued for this $\mathsf{sid}_{\mathsf{ban}}$ due to SELF DISTINCTION. In other words the GA receives all the user's tuple conveyed throughout the unicast *i.e.,* $\mathsf{CBU2.RecUCast}$, and from there deduce the following information:

(1) if $\mathsf{LM.Ver}$ is successful and $\mathsf{LM.MatchSet}$ outputs 1, it implies that the user was the concerned targeted user resulting in his, her or their exclusion using $\mathsf{CBU2.URmv}$.
(2) if $\mathsf{LM.Ver}$ succeeded and $\mathsf{LM.MatchSet}$ outputs 0, it implies that the user is partially valid therefore we enter in one of the two situations:
    (a) if a match is detected between the user and an another, this implies that the user tried to outsmart by playing with someone else's valid ListMAC key and with his, her or their own CBU2 key. Therefore, the $\mathsf{ID}^{\mathsf{LM}}$ linked to $\mathsf{ID}^{\mathsf{CBU2}}$ is retrieved and added to the $\mathsf{SRL} \cup \{\mathsf{ID}^{\mathsf{LM}}, \mathsf{bsn}\}$.
    (b) the protocol also checks if the users that are solving the challenge are all distinct in terms of $\mathsf{ID}^{\mathsf{CBU2}}$, if not the GA adds the corresponding $\mathsf{ID}^{\mathsf{LM}}$ into SRL.
    (c) if no match is detected, this implies that the user is legit.

For all the banned collateral users (those who have been corrupted for example), since the GA adds the tag generated through their $\mathsf{ID}^{\mathsf{LM}}$, therefore broadcasts to the users some new elements to add in SRL with the particularities that the added tags are those with an auxiliary data equal to bsn – therefore users can check that the added elements in SRL are those 'collateral' users. The banishment finish when the GA sends `FINISHEDBAN`. Note, that this method isn't invasive in a sense that the identities of other members whose had participated in the handshake corresponding to the transcript tr aren't disclosed to the GA, plus the commitment value prevents any misbehaving from a potential curious GA.

$\mathsf{SHS.Handshake}$. (*cf.* Figure 4) The secret handshake can be broken down in four distinct phases, as follows:

(1) First, the users establish a common secret that they authenticate by tagging it;
(2) Then, when receiving the tags of each user they obtained, they seperatly checks if the tag is valid for their concerned group;
(3) Next, they check if none of them have been banned or left the group, if so they accept to communicate by tagging the contribution of each user;
(4) Finally, they check if they all accepted, and therefore they can securely communicate, as the handshake was successful.

We recall, the protocol is run amongst user in a set that we dub Δ – those users may or may not be in the same group. Since then, they establish a common secret $\mathsf{ms}_{\mathsf{AGKA\text{-}FR}}$ and a common value sid by using the building block AGKA-FR. This also help them to obtain a unique identifier, which is, for the sake of simplicity, their share's part *i.e.,* state_A for user U_A. In order to separate the user in Δ amongst them by their affiliation to their group, they each run `KeySchedule` by binding the $\mathsf{ms}_{\mathsf{AGKA\text{-}FR}}$ (common to all Δ's users) with nonce^t_G (common to all G's members) therefore obtaining a secret tuple $(k', hk, \mathsf{nonce}^{\mathsf{sid}}_t)$. Those values have different uses:

k' is used to cipher the conversation as would a session key,
hk referred as *hiding key* helps to cipher all values needed for the authentication, prior that all parties trust them each other,
$\mathsf{nonce}^{\mathsf{sid}}_t$ is used as a nonce in the first phase.

First each user tags with their secret keys, the auxiliary data set to KShare, and a message which is for example for user U_A, we have $H(\mathsf{state}_A||\mathsf{nonce}^{\mathsf{sid}}_t)$; This helps to obtain a unique message per users and therefore avoid the issue of bit-leakage as mentioned in subsection A.1. Then the pseudo-identifier state_A and the tag τ_A is symmetrically encrypted with the key hk.

In the second phase the users decrypt the cipher with their computed hk, therefore obtain a right tag if in the same group otherwise couldn't go further. The decryption makes the verification of the tag possible, plus we check if we have the correspondence between one tag and one state_*. Plus we check that no two tags could be signed by the same user by running `DetectSelfDistinction`, therefore obtaining SELF DISTINCTION. The revealed keys stored in KRL are also checked that they weren't used to tag, with the same mechanism. If all is respected then the users send to each other their π^{τ}_*, again concatenated with the corresponding state_* and ciphered with $\mathsf{SEnc}(hk, \cdot)$.

In the third phase all participants after decryption have the corresponding π^{SRL} which proves if a SRL's tag has been tagged by the same user; in our scheme it corresponds to check if the user is banned. Therefore, each user checks if they do not discuss with a banned user or with a user who sent an invalid π^{SRL}. If so, then the users accepts the handshake by hashing the concatenation of all signatures into acc $\leftarrow H(\mathsf{nonce}_t^{\mathsf{sid}}||\tau_1||\mathsf{state}_1||...|\tau_{\#\Delta}||\mathsf{state}_{\#\Delta})$. Then each user tags it with their associated state_i always with the same auxiliary data sid, ciphers it and conveys it to the other users.

In the fourth phase after reception and decryption, the users obtains τ_i^{acc} checks the validity of the tag. But also checks if it matches with the corresponding τ_i supposedly issued by the same user; satisfying the fact that no users have been substituted due to Self distinction.

Note to the reader, we haven't specified, but if a fail occurs therefore the function `SimuleStage` is run to simulate an interaction, by specifying the number of the phase and a secret common to all the Δ's users.

Remark 4. *Without loss of generality, since ListMAC doesn't guarantee the confidentiality of the messages, we propose to cipher this over the network.*

Remark 5. *To keep Result-hiding, Handshake simulability and to avoid any threats concerning the affiliation, since the π^{SRL} can vary from group to group, it's send separately from the tag τ.*

Remark 6. *One may argue that in the second phase the use of* `DetectSelfDistinction` *over the* KRL*, may vary from groups to groups. But since the publication in* KRL *is discouraged in the protocol, therefore there should be a very few numbers of keys in* KRL*. Nevertheless, it is easy to 'import' from* KRL *to* SRL*, by simply generate a tag and then put it in* SRL*.*

C.3 Costs

We would like to talk about the costs, to sum up GCD [20] is pretty heavy. As we have seen before, during a secret handshake in the two first phases there aren't any identity system, and we have the group signature in the third phase that have a sort of identity system, but a user during a secret handshake doesn't identify himself/herself in order to keep Unlinkability and Full-unlinkability, this leads to an optimization issue mainly in the management of banned users. In facts, when the GA would like to ban a user since the GA isn't able to revoke this user, the GA regenerates new key to all members. This is a heavy operation to do. The same happens for SHS.`Update` for the same reasons, and SHS.`Join` since SHS.`Join` calls SHS.`Update`.

In terms of the comparative complexity of this scheme with respect to the original GCD scheme, we can list the following improvements. Improvements have been made on the SHS.`Join` by avoiding regenerating again all the users' secret key, therefore the complexity drop from regenerating keys for all users to only generate a key for a new user. Same improvements have been made to SHS.`Leave` by avoiding sending new sets of keys ; But to ban a user in the GCD original handshake, the GA traces back the users and then suppress the access of the user to the CBU2, and then runs SHS.`Update`, meaning regenerating secret long term keys for all the remaining users, here we act with a different paradigm where the GA prepares a trap and waits for the traitor to ask for temporal material before getting trapped, due to the revocation style. Therefore, the complexity also drops, and we lost access to the function made for tracing and helps us achieved a better privacy requirements.

C.4 Building blocks

C.4.1 Symmetric encryption.

Definition 15 (SE). *Syntax:*

- $\mathsf{SEKGen}(1^\lambda) \to k$ *Generates a key k for a given security parameter (in general it is equivalent to sample a random binary vector of size λ)*
- $\mathsf{SEnc}(k, m) \to c$ *For a given key k and a message it generates a ciphertext c*
- $\mathsf{SDec}(k, c) \to m$ *For a given key k and a message it gives a message m*

Correctness.

$$\mathsf{SDec}(k', \mathsf{SEnc}(k, m)) = m \implies k' = k$$

Security. We require that the SE is NM-CPA.

Corollary 1.

$$\mathsf{NM-CPA} \implies \mathsf{IND-CPA}$$

C.4.2 Hash.

Definition 16 (Hash function H). *Let H be a deterministic hash function, it is defined as follows:*

$$H : \{0,1\}^* \to \{0,1\}^l$$

It takes a binary word of any size and outputs a binary word of size l.

Definition 17 (2nd – preimg).

$$\Pr\left[\begin{array}{lcl} x \neq x' & & k \xleftarrow{\$} \mathcal{K} \\ \wedge H(k||x) = H(k||x') & : & x \xleftarrow{\$} D \\ x \in D & & x' \xleftarrow{\$} \mathcal{A}(k, x) \end{array}\right] = \mathsf{Adv}_H^{\mathsf{2nd-preimg}}$$

*C.4.3 Centralized Broadcast and User Unicast (*CBU2*).* This appendix further details the notion of Centralized Broadcast and User Unicast, providing fully-formal definitions and a security model.

The CBU2 primitive supposes the existence of several users U_i, associated with some registered credentials, who may be added or removed from sessions of CBU2 channels, by special super-users called managers, denoted as CM. We assume that no manager plays the role of user elsewhere, and vice-versa.

- CBU2.`Setup`$(n) \to$ (spar, ppar): This global setup algorithm takes in input a security parameter (in unary) and outputs global private (spar) and public (ppar) parameters. The private parameters spar are given in input to all channel managers CM. the public parameters ppar are taken implicitly in input to all the following algorithms.
- CBU2.`NewCred`(role) $\to$ (sk, pk): This auxiliary algorithm can be run by users (with role = user) or by channel managers (with role = manager) to obtain a pair or private and public credentials (denoted, respectively, sk or pk).

- $\mathsf{CBU2.RegCred}(U\langle U.\mathsf{sk}, U.\mathsf{pk}\rangle, \mathsf{CM}\langle \mathsf{CM.sk}, \mathsf{CM.pk}\rangle) \rightarrow (\langle \mathsf{ID}^{\mathsf{CBU2}} \cup \bot\rangle, \langle \mathsf{OK} \cup \bot\rangle)$: The interactive $\mathsf{RegCred}$ algorithm enables a user to register its public key (and implicitly its private key) $(U.\mathsf{sk}, U.\mathsf{pk})$ with a channel manager CM. The output of the algorithm on the user side is a unique, potentially-public identifier $\mathsf{ID}^{\mathsf{CBU2}}$, while on the channel manager side, the output is just a success/failure bit. The channel manager will keep track of a user list $\mathcal{UG}$, containing tuples of the form $(U, U.\mathsf{pk}, \mathsf{ID}^{\mathsf{CBU2}}, \{(\mathsf{sid}, \mathsf{ms}, \mathsf{sendK})\}^*)$, whenever the channel manager outputs OK. The last value is a list of three-value tuples. Each three-value tuple corresponds to a session in which U takes part, and will be used to store, respectively, session identifiers, the current master secret value for channel session sid, and a unicast key that is only known by the channel manager and the user itself (these three values are originally set to $\bot$ here and updated the first time upon channel creation or users joining an established channel). We explicitly assume that the channel manager outputs OK after ensuring: (i) that the same key pk is not registered for another $U' \in \mathcal{UG}$; (ii) that both U and CM have both confirmed that their communication partners know the private keys corresponding to their respective public keys; (iii) that potentially out-of-band verification is performed to ensure that the keys legitimately belong to the user in question (equivalent to verification of identities and certificates).
- $\mathsf{CBU2.ChInit}(\Delta, \mathsf{CM.sk}) \rightarrow \langle \mathsf{ms}^0_{\mathsf{sid}}, \mathsf{mpk}^0_{\mathsf{sid}}, \mathsf{sid}, \mathsf{sendK}_i\rangle_{i=1}^{|\Delta|}, \langle \mathsf{sid}, \mathsf{ms}^0_{\mathsf{sid}}, \mathsf{mpk}^0_{\mathsf{sid}}, \{\mathsf{sendK}_i\}_{i=1}^{|\Delta|}\rangle$: This interactive algorithm is always triggered by a channel manager CM for a given set of registered users $\Delta := \{U_1, U_2, \ldots, U_\Delta\}$, such that $\forall i : U_i \in \mathcal{UG}$. It outputs, for each user U_i, a tuple of the form $\mathsf{ms}^0_{\mathsf{sid}}, \mathsf{mpk}^0_{\mathsf{sid}}, \mathsf{sid}, \mathsf{sendK}_i$, consisting of the (private) initial master session key $\mathsf{ms}^0_{\mathsf{sid}}$, the public initial master public key $\mathsf{mpk}^0_{\mathsf{sid}}$, a unique session identifier sid, and the private unicast key of the user U_i, denoted sendK_i. The two former parameters will be shared amongst all the group users, being used for broadcasts by the channel manager, and will evolve through updates (changing the upper index from 0 to 1, 2, etc.). By contrast, the sending key will only be known to the user and the channel manager, thus allowing for unicast. The channel manager receives the master session parameters (both public and private) and all the user unicast keys. For each $U_i \in \mathcal{UG}$, the channel manager inserts a tuple $(\mathsf{sid}, \mathsf{ms} = \mathsf{ms}^0_{\mathsf{sid}}, \mathsf{sendK} = \mathsf{sendK}_i)$ in the entry indexed by U_i in $\mathcal{UG}$. The channel manager will also keep track of the identities of users participating in each session sid; these groups of identities are stored in lists denoted $\mathcal{UG}_{\mathsf{sid}}$.
- $\mathsf{CBU2.UAdd}(\mathsf{sid}, \mathsf{CM.sk}, U_j) \rightarrow \langle \mathsf{ms}^t_{\mathsf{sid}}, \mathsf{mpk}^t_{\mathsf{sid}}, \mathsf{sid}, \mathsf{sendK}_j\rangle$, $\langle \mathsf{ms}^t_{\mathsf{sid}}, \mathsf{mpk}^t_{\mathsf{sid}}$, $\mathsf{sidsendK}_i\rangle^{U_i \in \mathcal{UG}_{\mathsf{sid}}}, \langle \mathsf{sid}, \mathsf{ms}^t_{\mathsf{sid}}, \mathsf{mpk}^t_{\mathsf{sid}}$, $\{\mathsf{sendK}_i\}^{U_i \in \mathcal{UG}_{\mathsf{sid}} \cup U}, \mathsf{sid}, \mathcal{UG}_{\mathsf{sid}}\rangle$: This protocol is used by the channel manager to add a registered user U to an existing session sid. The user obtains private/public master session credentials $\mathsf{ms}^t_{\mathsf{sid}}, \mathsf{mpk}^t_{\mathsf{sid}}$ (where t is an index accounting for the number of adds/removals/updates performed since the setup of the channel), as well as the session identifier and its unicast key. All other existing users in the channel update their own key-materials. Finally, the channel manager also updates its session and unicast parameters, the session identifier for that session, the current list of users participating in session sid denoted $\mathcal{UG}_{\mathsf{sid}}$, and adds a tuple of the form $(\mathsf{sid}, \mathsf{ms} = \mathsf{ms}^t_{\mathsf{sid}}, \mathsf{sendK} = \mathsf{sendK}_j)$ for the entry indexed U_j in $\mathcal{UG}$.
- $\mathsf{CBU2.URmv}(\mathsf{sid}, \mathsf{CM.sk}, U_j) \rightarrow \langle \mathsf{ms}^t_{\mathsf{sid}}, \mathsf{mpk}^t_{\mathsf{sid}}, \mathsf{sid}, \mathsf{sendK}_i\rangle^{U_i \in \mathcal{UG}_{\mathsf{sid}} \setminus U_j}$, $\langle \mathsf{sid}, \mathsf{ms}^t_{\mathsf{sid}}, \mathsf{mpk}^t_{\mathsf{sid}}, \{\mathsf{sendK}_i\}^{U_i \in \mathcal{UG}_{\mathsf{sid}} \setminus U}, \mathsf{sid}, \mathcal{UG}_{\mathsf{sid}}\rangle$: The channel manager may also decide to remove a user from the group. Analogously to the algorithm that allows it to add a new user, the channel manager triggers updates to each remaining user's session key-material, as well as to the list $\mathcal{UG}_{\mathsf{sid}}$ and potentially the session identifier itself. Finally, CM removes a tuple of the form $(\mathsf{sid}, \cdot, \cdot)$ from the entry indexed by U_j in $\mathcal{UG}$.
- $\mathsf{CBU2.ChUpdate}(\mathsf{sid}, \mathsf{CM.sk}) \rightarrow \langle \mathsf{ms}^t_{\mathsf{sid}}, \mathsf{mpk}^t_{\mathsf{sid}}, \mathsf{sid}, \mathsf{sendK}_i\rangle^{U_i \in \mathcal{UG}_{\mathsf{sid}}}, \langle \mathsf{sid}, \mathsf{ms}^t_{\mathsf{sid}}, \mathsf{mpk}^t_{\mathsf{sid}}, \{\mathsf{sendK}_i\}^{U_i \in \mathcal{UG}_{\mathsf{sid}}}, \mathsf{sid}\rangle$: The channel manager may decide to trigger at any point an update to the session-specific keys of all the users. Each update increases by 1 the value of the index t of the master private and public keys (used for broadcast). We note that while the unicast keys may also be updated at this time, this is not compulsory. After each update, for each $U \in \mathcal{UG}_{\mathsf{sid}}$, the channel manager updates the entry $(\mathsf{sid}, \mathsf{ms} = \mathsf{ms}^t_{\mathsf{sid}}, \mathsf{sendK} = \mathsf{sendK}_i$ of that user in the database $\mathcal{UG}$.
- $\mathsf{CBU2.BCast}(\mathsf{sid}, \mathsf{CM.sk}, \mathsf{ms}^t_{\mathsf{sid}}, msg) \rightarrow (c, \mathsf{aux})$: The channel manager may choose to (securely) broadcast messages *msg* by using its private key CM.sk as well as the current private broadcast key $\mathsf{ms}^t_{\mathsf{sid}}$. The result is a ciphertext c, and potentially some auxiliary information aux (which might store AEAD data, or potential key-updating information).
- $\mathsf{CBU2.UCast}(\mathsf{sid}, \mathsf{sendK}_i, msg) \rightarrow (c, \mathsf{aux})$: Users may not use the broadcasting algorithm, but any user U_i can unicast messages *msg* to the channel manager, by using their current unicast key sendK_i. The result is a ciphertext c, and potentially some auxiliary information aux (which might store AEAD data, or potential key-updating information).
- $\mathsf{CBU2.RecBCast}(\mathsf{sid}, \mathsf{ms}^t_{\mathsf{sid}}, (c, \mathsf{aux})) \rightarrow msg \cup \bot$: This algorithm allows a user to receive a broadcast from the channel manager, by using the current session keys $\mathsf{ms}^t_{\mathsf{sid}}$ in order to decrypt the ciphertext c and auxiliary information aux to either a message *msg* or to an error symbol $\bot$. If the output is $\bot$, we say that the user has rejected the broadcast message.
- $\mathsf{CBU2.RecUCast}(\mathsf{sid}, \mathsf{sendK}_i, (c, \mathsf{aux})) \rightarrow msg \cup \bot$: This algorithm allows the channel manager to receive a unicast from a user U_i, by using the corresponding user's current unicast key sendK_i in order to decrypt the ciphertext c and auxiliary information aux to either a message *msg* or to an

error symbol $\bot$. If the output is $\bot$, we say that the channel manager has rejected the unicast message.

Notice that our construction involves a global setup algorithm, which will output global public and private parameters spar and ppar. These values will be provided to all the channel managers. Each channel managers can create various sessions of a channel, by using both these global parameters are the credentials generated for the channel manager itself (through the NewCred algorithm). In other words, whereas sessions of two different channels are somewhat compatible to each other, cryptographically speaking, because they employ the same global parameters, it is impossible for one channel manager to hijack the channel of another manager (except by corruption), because each channel manager needs to use its own private credentials for channel creation and later management.

Another important note concerns the registration step for the users. We include this algorithm as a statement of intent, on the part of the user, that he, she, or they wish to join the channel managed by CM. In particular, from the point of view of the user, the process of deciding to joint a $\mathsf{CBU2}$ channel goes as follows: first, the user generates credentials for role = user. Then, it registers those credentials in an interactive protocol with the channel manager CM of the desired channel. The channel manager can then choose to either add the user to an existing channel session sid or to create a new channel session sid' taking in input a set of users made up of at least the requesting user (and maybe also other, previously-registered users).

Channel management is dynamic and takes place in various installments across vast periods of time. The channel manager is the only entity that can trigger an update, whenever users are added or removed, or the channel is updated. Each modification triggers an update of key-material and user lists and can be perceived as being akin to the notion of "epoch" in secure-channel establishment with post-compromise security. To avoid confusion, we abuse vocabulary and call these installments "time", indexed by a discrete variable t. Thus, the phrase "the user group $\mathcal{UG}_{\mathsf{sid}}$ at time t" is used to refer to the value of $\mathcal{UG}_{\mathsf{sid}}$ at the t-th installment of the group management process.

The lion's share of the key-management is done by the channel manager, which maintains updated user lists $\mathcal{UG}_{\mathsf{sid}}$ of user identifiers U_i for users taking part in the channel session which has a session identifier of sid, though we note that in practice CM might want to store, for each user, a tuple of values $(U_i, \mathsf{ID}_i^{\mathsf{CBU2}})$ – note that this correspondence is stored in the database $\mathcal{UG}$ that stores the cumulative data for all the sessions and all the users involved, notably tuples of the form $(U, U.\mathsf{pk}, \mathsf{ID}^{\mathsf{CBU2}}, \{(\mathsf{sid}, \mathsf{ms}, \mathsf{sendK})\}^*)$, as mentioned in the description of the credential-registration algorithm. The values sid, ms, and sendK evolve over time, in one or more of the following ways:

- Whenever a registered user U' is added an existing session sid of a channel managed by CM: the entry corresponding to U' in $\mathcal{UG}$ (which was created upon registration) is enriched with a tuple containing key-material and the session identifier for session sid; all entries corresponding to users $U \in \mathcal{UG}_{\mathsf{sid}}$ is updated with new values for the session identifier and the key-material.
- Whenever a registered user U' leaves an existing session sid of a channel managed by CM: the entry corresponding to U' in $\mathcal{UG}$ (which was created upon registration) has the entry of the form $(\mathsf{sid}, \cdot, \cdot)$ removed; all entries corresponding to users $U \in \mathcal{UG}_{\mathsf{sid}}$ have the values $(\mathsf{sid}, \mathsf{ms}, \mathsf{sendK})$ updated.
- Whenever the channel manager decides to update the session sid, without adding or removing users: all entries corresponding to users $U \in \mathcal{UG}_{\mathsf{sid}}$ have the values $(\mathsf{sid}, \mathsf{ms}, \mathsf{sendK})$ updated.

Channel manager and user states. We assume that the channel manager CM maintains state consisting of:

- $\mathsf{CM}.\mathsf{sk}, \mathsf{CM}.\mathsf{pk}$: Its private and public long-term parameters.
- $\mathcal{UG}$: The database of all registered users, described in detail in previous paragraphs.
- $\mathcal{UG}_{\mathsf{sid}}$: For each session sid of the channel, a constantly updated list of users, which explicitly includes the up-to-date session state sid, the up-to-date master secret $\mathsf{ms}^t_{\mathsf{sid}}$, and for each user $U \in \mathcal{UG}_{\mathsf{sid}}$, its unicast key sendK_i.
- t_{sid}: For each channel session sid, a variable indicating the current time (epoch) of that session.
- Snd: For each channel session sid, this is a list of entries of the type $(t, \mathcal{UG}_{\mathsf{sid}}, msg, c, \mathsf{aux})$, consisting of: the time (epoch) t at which the message msg is broadcast by CM; the users $\mathcal{UG}_{\mathsf{sid}}$ taking part in session sid managed by CM at time t; the content of the message msg; as well as the ciphertext c and auxiliary value aux obtained by running the BCast algorithm for message msg. We denote by $\mathsf{Snd}_{\mathsf{sid}}$ the part of this database indexed by sid.
- Rcv: A database, indexed by session identifiers sid, containing lists of tuples $(t_{\mathsf{sid}}, U, c, \mathsf{aux}, m)$, which keep track of ciphertexts (c, aux) received at each epoch from users $U \in \mathcal{UG}_{\mathsf{sid}}$, decrypted to a value m (which can take a special value $\bot$ if decryption fails). We denote by $\mathsf{Snd}_{\mathsf{sid}}[U_i]$ the part of this database indexed by sid and restricted to messages received from user U_i.

A user may register its credentials to several channel managers and be part of several sessions of each channel. Administering these values will require users U to keep track of the following values:

- $U.\mathsf{sk}, U.\mathsf{pk}$: Its private and public long-term parameters.
- ChList: A database of elements of the form $(\mathsf{CM}, \mathsf{CM}.\mathsf{pk}, \mathsf{ID}^{\mathsf{CBU2}}, \{(\mathsf{sid}, \mathsf{ms}, \mathsf{sendK})\}^*)$, where the two first components indicate the identity and public key of the channel manager, the value $\mathsf{ID}^{\mathsf{CBU2}}$ stores the identity of the user within the channel managed by the channel manager, and the last element is a list of credentials per session of the channel. This last element is updated each time the channel manager triggers an evolution of the key, which is to say, at each epoch.
- γ: A corrupt bit, initially set to 0, indicating whether the user has been corrupted (the bit is set to 1 and can never be reverted back to 0) or not (the bit is 0).
- t_{sid}: For each channel session sid, a variable indicating the current time (epoch) of that session.
- α_{sid}: For each channel session sid, a variable indicating whether the user is in an accepting state at the current epoch (indicated by t_{sid}).

- Snd: For each channel session sid, this is a list of entries of the type $(t, msg, c, \mathsf{aux})$, consisting of: the time (epoch) t at which the message msg is unicast by the user; the content of the message msg; as well as the ciphertext c and auxiliary value aux obtained by running the `UCast` algorithm for message msg in session sid. We denote by $\mathsf{Snd}_{\mathsf{sid}}$ the part of this database indexed by sid.
- Rcv: A database, indexed by session identifiers sid, containing lists of tuples $(t_{\mathsf{sid}}, c, \mathsf{aux}, m)$, which keep track of ciphertexts (c, aux) received through broadcast at each epoch from the manager of the channel to which sid belongs, decrypted to a value m (which can take a special value $\perp$ if decryption fails). We denote by $\mathsf{Rcv}_{\mathsf{sid}}$ the part of this database indexed by sid.

Correctness. For the purposes of the security model, both the users and the channel managers must keep track of additional states (real-or-random key bits, as well as past keys).

For the channel manager CM, the additional state consists of:

- γ: A corrupt bit, initially set to 0, indicating whether the channel manager has been corrupted (the bit is set to 1 and can never be reverted back to 0) or not (the bit is 0).
- b: This real-or-random bit will be used for the security games, and will indicate whether the game will return a real key for the test oracle, or a random key chosen from the same distribution.
- $\mathsf{KeyList}$: A list indexed by session identifiers, containing elements of the form $(\mathsf{sid}, \{t, \mathcal{UG}^t_{\mathsf{sid}}, \alpha^t_{\mathsf{sid}}, (\mathsf{ms}, \rho^t_{\mathsf{ms}}), \{(\mathsf{sendK}_i, \rho^t_{\mathsf{sendK}_i})\}^{U_i \in \mathcal{UG}^t_{\mathsf{sid}}}\})$, in which, for each channel value of $t = 0, 1, \ldots, t_{\mathsf{sid}}$, the channel manager stores credentials ms and sendK at epoch t, as well as the list of users present on the channel session at epoch t, denoted $\mathcal{UG}^t_{\mathsf{sid}}$. In addition, the channel manager also keeps track of a reveal bit for each of those keys, denoted ρ^t_{ms} and respectively ρ^t_{sendK}, and an accept bit α^t_{sid} indicating whether it is in an accepting state at epoch t.

For a user U, the additional state consists of:

- γ: A corrupt bit, initially set to 0, indicating whether the user has been corrupted (the bit is set to 1 and can never be reverted back to 0) or not (the bit is 0).
- b: This real-or-random bit will be used for the security games, and will indicate whether the game will return a real key for the test oracle, or a random key chosen from the same distribution.
- $\mathsf{KeyList}$: A list indexed by session identifiers[15], containing elements of the form $(\mathsf{sid}, \{t, \alpha^t_{\mathsf{sid}}, (\mathsf{ms}, \rho^t_{\mathsf{ms}}), (\mathsf{sendK}, \rho^t_{\mathsf{sendK}})\})$, in which, for each channel value of $t = 0, 1, \ldots, t_{\mathsf{sid}}$, the user stores credentials ms and sendK at epoch t. In addition, the user also keeps track of a reveal bit for each of those keys, denoted ρ^t_{ms} and respectively ρ^t_{sendK}, and an accept bit α^t_{sid} indicating whether it is in an accepting state at epoch t.

Notice that the session administration we propose differs somewhat in syntax from typical Bellare-Rogaway terminology (we omit the notion of instance, though we store instance-specific state information in global databases that include entries indexed by session identifiers, *e.g.,* ChList, $\mathcal{UG}$, etc. We do this mostly in order to render notation easy to understand even for readers not well-versed in complex secure-channel establishment; in addition, this allows us to more easily use the CBU2 channel in the context of secret handshakes. This notation affects how security is defined, but direct parallels with more classical notations are evident.

The notion of correctness is multi-faceted.

We require both a narrower and a broader notion of correctness than in typical multi-stage authenticated key-agreement, since CBU2 is centralized (key-updates are triggered only by the channel manager, acceptance/key computation on the channel manager will imply acceptance/key computation for the channel users), features some key-updates for the broadcast channel, but also includes a more classical end-to-end secure unicast channel with no key updates.

This is formally defined below.

Definition 18 (Correctness of CBU2). *A* CBU2 *protocol run in the presence of a set of managers* MngSet *and a set of users* USet *should provide the following notions of correctness:*

- *Centralized broadcast: For every* CM*, every* sid*, and every value t, if there exists* $(\mathsf{sid}, \{t, \mathcal{UG}^t_{\mathsf{sid}}, \alpha^t_{\mathsf{sid}}, (\mathsf{ms}, \cdot), \{(U_i, \mathsf{sendK}_i, \cdot)\}^{U_i \in \mathcal{UG}^t_{\mathsf{sid}}}\}) \in \mathsf{CM.KeyList}$ *with* $\mathcal{UG}^t_{\mathsf{sid}} \neq \emptyset$*, and denoting, by abuse of notation* $\mathsf{CM.sid} = \mathsf{sid}$, $\mathsf{CM}.\alpha^t_{\mathsf{sid}} := \alpha^t_{\mathsf{sid}}$, $\mathsf{CM.ms}^t_{\mathsf{sid}} = \mathsf{ms}$ *from the entry above, and for every* $U_i \in \mathcal{UG}^t_{\mathsf{sid}}$, $\mathsf{CM.sid.sendK}_i = \mathsf{sendK}_i$ *from the entry above, the following holds, in the absence of an adversary:*
  - *For every user* $U_i \in \mathcal{UG}^t_{\mathsf{sid}}$*, there exists an entry* $(\mathsf{sid}, \{t, \alpha^t_{\mathsf{sid}}, (\mathsf{ms}, \cdot), (\mathsf{sendK}, \cdot)\}) \in U_i.\mathsf{KeyList}$*. Informally, if the channel manager manages a session* sid *of the channel such that, at time t, it believes the broadcast receivers are users* $U_i \in \mathcal{UG}^t_{\mathsf{sid}}$*, then all the users* $U_i \in \mathcal{UG}^t_{\mathsf{sid}}$ *also agree on this and store session data to prove it. By abuse of notation, we denote* $U_i.\mathsf{sid} = \mathsf{sid}$, $U_i.\alpha^t_{\mathsf{sid}} = \alpha^t_{\mathsf{sid}}$, $U_i.\mathsf{ms}^t_{\mathsf{sid}} = \mathsf{ms}$ *from the user's database entry as described above, and* $U_i.\mathsf{sid.sendK} = \mathsf{sendK}$ *from the entry above.*
  - *For every user* $U_j \notin \mathcal{UG}^t_{\mathsf{sid}}$*, there exists no entry of the form* $(\mathsf{sid}, \{t, \cdot, \cdot, \cdot\}) \in U_i.\mathsf{KeyList}$*.*
  - *For every user* $U_i \in \mathcal{UG}^t_{\mathsf{sid}}$*, either* $\mathsf{CM.sid} = U_i.\mathsf{sid}$ *at time t or it is a prefix thereof at time t, while being equal at time* $t - 1$*.*
  - *For every user* $U_i \in \mathcal{UG}^t_{\mathsf{sid}}$*, it holds that* $\mathsf{CM}.\alpha^t_{\mathsf{sid}} = U_i.\alpha^t_{\mathsf{sid}}$*.*
  - *For every user* $U_i \in \mathcal{UG}^t_{\mathsf{sid}}$*, if* $\mathsf{CM}.\alpha^t_{\mathsf{sid}} = U_i.\alpha^t_{\mathsf{sid}} = 1$ *at time t, then* $\mathsf{CM.ms}^t_{\mathsf{sid}} = U_i.\mathsf{ms}^t_{\mathsf{sid}} \neq \perp$*.*
  - *If* CM *has an entry* $\mathsf{CM.Snd}_{\mathsf{sid}}$*, for each entry* $(t, msg, c, \mathsf{aux}) \in \mathsf{CM.Snd}_{\mathsf{sid}}$*, there exists a corresponding entry* $(t, c, \mathsf{aux}, m) \in U_i.\mathsf{Rcv}_{\mathsf{sid}}$ *for every user* $U_i \in \mathcal{UG}^t_{\mathsf{sid}}$*.*
- *User unicast: Using the same notations as for the first bullet-point, for every channel manager* CM*, every session* sid*, every timestamp t, and every user* $U_i \in \mathcal{UG}_{\mathsf{sid}}$*, in the absence of an adversary, the following statements hold, in addition to those in the first bullet-point:*
  - *If* $\mathsf{CM}.\alpha^t_{\mathsf{sid}} = U_i.\alpha^t_{\mathsf{sid}} = 1$ *at time t, then* $\mathsf{CM.sendK}_i = U_i.\mathsf{sid.sendK} \neq \perp$*.*

[15] Note that session identifiers are meant to be globally unique, which means that a session identifier will implicitly also point to other useful information such as the identity of the group manager CM, the current epoch of the channel, etc.

- *If a user $U_i \in \mathcal{UG}_{\mathsf{sid}}$ has an entry $U_i.\mathsf{Snd}_{\mathsf{sid}}$, for each entry $(t, msg, c, \mathsf{aux}) \in U_i.\mathsf{Snd}_{\mathsf{sid}}$, then there exists a corresponding entry $(t_{\mathsf{sid}}, U, c, \mathsf{aux}, m) \in \mathsf{CM}.\mathsf{Rcv}_{\mathsf{sid}}[U_i]$.*

Adversary Model. We define the security of the CBU2 primitive in terms of authentication and security of the established channel (akin to the notion of (S)ACCE, originally introduced in the context of TLS 1.2 in [13]). Our choice is motivated by the fact that in the construction of secret handshakes, the property we require from the CBU2 is that nonces encrypted and sent through broadcast are indistinguishable from random, and tags sent through unicast are similarly indistinguishable from random. While this property can be achieved by the careful composition of AKE-secure forward-secure key exchange and authenticated-encryption, achieving the equivalent of ACCE security is not always immediate.

We will require the following properties:

- Centralized broadcast: We require the authentication of sessions, updates, and transmissions (messages can only be accepted if they originate with the channel manager), and the security of the broadcast channel, specifically:
  - **Authentication:** an attacker without access to the channel manager's private values cannot make a non-malicious user accept a message that was not sent at that epoch by the channel manager itself.
  - **Security:** an attacker without access to the session key at some epoch t cannot break the ACCE security of a message sent through centralized broadcast – that is, only the users legitimately in possession of the epoch's keys may distinguish a transmitted message from random.
- User unicast: We require the authentication of transmissions (the user is authenticated), and the security of the unicast channel, specifically:
  - **Authentication:** an attacker not having access to a specific, honest user's private values cannot make a channel manager accept a message that was not set at that epoch by that user.
  - **Security:** an attacker without access to the user's key at that epoch cannot break the ACCE security of transmitted messages (in other words, only the user and the channel manager can distinguish from random the exchanged plaintexts).

Throughout the security games, the adversary will be given access to the following oracles:

- $\mathsf{oRegCred}(U, \mathsf{CM}, b_U, b_{\mathsf{CM}})$: On input a user U, a group manager CM, and a pair of corrupt bits b_U, b_{CM}, this oracle first checks that user U has the credentials required for registration and, if this is not the case, the oracle generates credentials by using the algorithm NewCred as a black box. We assume that all public credentials are provided to all parties. If moreover $b_U = 1$, the user is immediately corrupted and the adversary also obtains the private key corresponding to the freshly generated credentials. The situation is analogous for the intended channel manager CM. Once the credentials are available, the oracle runs CBU2.RegCred as a black box in order to register the user with the specific manager. The output is provided to the adversary.
- $\mathsf{oChSession}(\mathsf{CM}, \Delta)$: On input a channel manager CM (which implicitly identifies a channel managed by CM), as well as a set Δ of already-registered users, this oracle runs the ChInit algorithm on these inputs in order to start a new session of that channel. The adversary receives, as an output, the public initial session key as well as the session identifier. In addition, for every corrupt party (user or channel manager), the adversary also gets its private state (for a user this consists of the master secret of the broadcast channel and sending key for its unicast channel; for a channel manager, the state consists of the master secret of the broadcast channel, as well as all the keys for all the users' unicast channels).
- $\mathsf{oUAdd}(U, \mathsf{sid})$: On input a user U and a session identifier sid, this oracle runs the UAdd algorithm for U, sid, and the channel manager CM whose session sid is, at the current epoch t. The adversary gains access to: the (potentially updated) session identifier sid, the public key for the broadcast channel, as well as private broadcast session credentials and sending key for each corrupted user and the entire database of private credentials at epoch t if the channel manager is corrupted.
- $\mathsf{oURmv}(U, \mathsf{sid})$: On input a user U and a session identifier sid, this oracle runs the URmv algorithm for U, sid, and the channel manager CM whose session sid is, at the current epoch t. The adversary gains access to: the (potentially updated) session identifier sid, the public key for the broadcast channel, as well as private broadcast session credentials and sending key for each corrupted user and the entire database of private credentials at epoch t if the channel manager is corrupted.
- $\mathsf{oChUpdate}(\mathsf{sid})$: On input a session identifier sid, the oracle first identifies the channel manager whose session sid is. Then it runs the ChUpdate algorithm as a black box for that session and channel manager, at current epoch t. The adversary gains access to: the (potentially updated) session identifier sid, the public key for the broadcast channel, as well as private broadcast session credentials and sending key for each corrupted user and the entire database of private credentials at epoch t if the channel manager is corrupted.
- $\mathsf{oReveal}(\mathsf{sid}, P, \mathsf{type}, t)$: On input a session identifier sid, a party P, which can be either a user U or a channel manager CM, key type $\mathsf{type} \in \{\mathsf{UCast}, \mathsf{BCast}\}$, and an epoch t which must necessarily predate the epoch of sid at the time of the query (else the oracle returns an error symbol $\perp$), this oracle proceeds as follows. If $\mathsf{type} = \mathsf{UCast}$ and $P \in \mathsf{MngSet}$, then the output is an error symbol $\perp$. Else, if $\mathsf{type} = \mathsf{UCast}$ and P is a user U_i which is present at epoch t in sid, then the oracle retrieves and returns the user's unicast key at that epoch, sendK_i. The user sets $\rho^t_{\mathsf{sendK}_i}) = 1$ in the attribute KeyList. If $\mathsf{type} = \mathsf{BCast}$, and if the party P takes part in sid at epoch t, then the oracle returns the broadcast master secret at epoch t, ms, setting, for that party, the reveal bit $\rho^t_{\mathsf{ms}} := 1$ in KeyList.
- $\mathsf{oCorrupt}(\mathsf{CM}, P)$: This oracle is typically run with an input consisting of a channel manager CM and a party P (either a user or a channel manager), in which case it returns either

the long-term credentials of party P that were registered with the channel run by CM, or a special symbol $\perp$. Exceptionally, it can also be run with a special input $\mathsf{CM} \leftarrow \mathsf{MngSet}$ and a party P, in which case the credentials registered for that users with respect to all channel managers are targeted and returned. If P is a channel manager, $\mathsf{CM} \neq \mathsf{MngSet}$, and $P \neq \mathsf{CM}$, then the oracle returns $\perp$. For the case where P is a user U_i, while the oracle works even if U_i is not currently in the user-group sid run by CM, note that in some cases, the protocol might proceed to the deletion of key materials that are no longer in use, in which case the oracle will return $\perp$.

- $\mathsf{oPromptSend}(\mathsf{sid}, \mathsf{src}, \mathsf{type}, M)$: This oracle allows the adversary to prompt the honest sending of messages in the following two ways (in both cases the adversary receives the messages sent across the channel, such as ciphertexts and auxiliary information, and the message is delivered at the session's current epoch t):

  $\mathsf{oPromptSend}(\mathsf{sid}, \perp, \mathsf{BCast}, M)$: In this variant, the input includes $\mathsf{src} = \perp$ and $\mathsf{type} = \mathsf{BCast}$. This allows an attacker to prompt the channel manager CM of session sid to send (potentially in encrypted form), the message M. The message is delivered to all the users U_i that are taking part in sid at time t, *i.e.*, $\forall U_i \in \mathcal{UG}^t_{\mathsf{sid}}$, and for each user U_i the algorithm `RecBCast` is run as a black box for that message and sid. The message is added to the Snd of the channel manager CM.

  $\mathsf{oPromptSend}(\mathsf{sid}, U_i, \mathsf{UCast}, M)$: In this variant, the input to the oracle includes $\mathsf{type} = \mathsf{UCast}$, and a source $\mathsf{src} = U_i$, which indicates on which unicast channel the message is sent. The query runs `RecUCast` as a black box on the source user, the session identifier, and the provided message. Note that if U_i is not part of session sid at the time t when the query is made, the receiving algorithm might return an error symbol $\perp$. The message is added to the Snd of the user U_i if the user is present in sid at the current epoch.

- $\mathsf{oInjectSend}(\mathsf{sid}, \mathsf{src}, \mathsf{type}, M)$: This oracle can be used by the attacker to inject a message either into the broadcast or into a unicast channel in the following two ways:

  $\mathsf{oInjectSend}(\mathsf{sid}, \perp, \mathsf{BCast}, M)$: In this variant, the input includes $\mathsf{src} = \perp$ and $\mathsf{type} = \mathsf{BCast}$. This allows an attacker to inject a message on the broadcast link at the current epoch t (as though he were the group manager CM. The message is delivered to all the users U_i that are taking part in sid at time t, *i.e.*, $\forall U_i \in \mathcal{UG}^t_{\mathsf{sid}}$, and for each user U_i the algorithm `RecBCast` is run as a black box for that message and sid. Note that, if for instance M is incorrectly encrypted, running `RecBCast` might yield errors on the sides of the receivers.

  $\mathsf{oInjectSend}(\mathsf{sid}, U_i, \mathsf{UCast}, M)$: In this variant, the input to the oracle includes $\mathsf{type} = \mathsf{UCast}$, and a source $\mathsf{src} = U_i$, which indicates on which unicast channel the message is sent. The query runs `RecUCast` as a black box on the source user, the session identifier, and the provided message. Note that if U_i is not part of session sid at the time t when the query is made, the receiving algorithm might return an error symbol $\perp$.

- $\mathsf{oSendRoR}_b(\mathsf{sid}, \mathsf{src}, \mathsf{type}, M)$: This oracle can be used by the attacker only in some security games in order to force (without knowing the pertinent secret keys) the (authenticated) encryption and sending of either the input message M or a random message M_R of the same length, from the message space, depending on a bit b, which is either 0 (encrypt M_R) or 1 (encrypt M). If $\mathsf{type} = \mathsf{BCast}$, the real-or-random message is encrypted using ms at the current epoch of sid, whereas if $\mathsf{type} = \mathsf{UCast}$, it is encrypted using the sending unicast key sendK_i of user $U_i = \mathsf{src}$. In both cases, the attacker is given the resulting ciphertext c and potential auxiliary material aux, and the oracle $\mathsf{oInjectSend}(\mathsf{sid}, \perp, \mathsf{BCast}, (c, \mathsf{aux}))$ is run as a black box.

Security definitions. We define the security notions informally discussed above.

```
Game $\mathsf{Exp}_{\Pi}^{\mathsf{BCast\text{-}Auth}}(\mathcal{A})$
Let $\mathcal{O} = \{\mathsf{oRegCred}, \mathsf{oChSession}, \mathsf{oUAdd}, \mathsf{oURmv}, \mathsf{oChUpdate}\}$
        $\cup\{\mathsf{oReveal}, \mathsf{oCorrupt}, \mathsf{oPromptSend}, \mathsf{oInjectSend}\}$
$(\mathsf{spar}, \mathsf{ppar}) \leftarrow \mathsf{CBU2.Setup}(n)$
$\mathsf{done} \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{ppar})$;
$\mathcal{A}$ wins iff. $\exists U_i$ s.t. $\exists[\mathsf{sid}, (t_{\mathsf{sid}}, c, \mathsf{aux}, m)] \in U_i.\mathsf{Rcv}$
and (simultaneously):
    $m \neq \perp$
    $[\mathsf{sid}, (t_{\mathsf{sid}}, \cdot, \cdot, \cdot, m)] \notin \mathsf{CM.Snd}$
    The unique $\mathsf{CM}$ managing sid created via $\mathsf{oRegCred}(\cdot, \mathsf{CM}, \cdot, 0)$
    No $\mathsf{oCorrupt}(\mathsf{CM}, \mathsf{CM})$ query for the manager $\mathsf{CM}$ of sid
```

Figure 10: The Broadcast-authentication security game of $\mathsf{CBU2}$**.**

Definition 19 (Broadcast-Authentication Security). *Consider a* $\mathsf{CBU2}$ *channel denoted* $\mathsf{CBU2}$*. For a probabilistic polynomial-time (PPT) adversary* $\mathcal{A}$ *we define its advantage* $\mathsf{Adv}_{\mathsf{CBU2}}^{\mathsf{BCast\text{-}Auth}}(\mathcal{A})$ *to win the* $\mathsf{Exp}_{\Pi}^{\mathsf{BCast\text{-}Auth}}(\mathcal{A})$ *security game presented in Figure 10 as follows:*

$$\mathsf{Adv}_{\mathsf{CBU2}}^{\mathsf{BCast\text{-}Auth}}(\mathcal{A}) = \Pr[\mathcal{A} \textit{ wins } \mathsf{Exp}_{\Pi}^{\mathsf{BCast\text{-}Auth}}(\mathcal{A})]. \qquad (1)$$

The channel $\mathsf{CBU2}$ *is* ϵ*-Broadcast-Authentication-Secure if, and only if, any PPT adversary* $\mathcal{A}$ *against* $\mathsf{CBU2}$ *has at most an advantage of* ϵ *to win the* $\mathsf{Exp}_{\Pi}^{\mathsf{BCast\text{-}Auth}}(\mathcal{A})$ *security game. Asymptotically,* $\mathsf{CBU2}$ *is called* Broadcast-Authentication-Secure *if* ϵ *is negligible as a function of the security parameter* n*.*

Definition 20 (Unicast-Authentication Security). *Consider a* $\mathsf{CBU2}$ *channel denoted* $\mathsf{CBU2}$*. For a probabilistic polynomial-time (PPT) adversary* $\mathcal{A}$ *we define its advantage* $\mathsf{Adv}_{\mathsf{CBU2}}^{\mathsf{UCast\text{-}Auth}}(\mathcal{A})$ *to win the* $\mathsf{Exp}_{\Pi}^{\mathsf{UCast\text{-}Auth}}(\mathcal{A})$ *security game presented in Figure 11 as follows:*

$$\mathsf{Adv}_{\mathsf{CBU2}}^{\mathsf{UCast\text{-}Auth}}(\mathcal{A}) = \Pr[\mathcal{A} \textit{ wins } \mathsf{Exp}_{\Pi}^{\mathsf{UCast\text{-}Auth}}(\mathcal{A})]. \qquad (2)$$

The channel $\mathsf{CBU2}$ *is* ϵ*-Unicast-Authentication-Secure if, and only if, any PPT adversary* $\mathcal{A}$ *against* $\mathsf{CBU2}$ *has at most an advantage of* ϵ *to win the* $\mathsf{Exp}_{\Pi}^{\mathsf{UCast\text{-}Auth}}(\mathcal{A})$ *security game. Asymptotically,* $\mathsf{CBU2}$ *is called* Unicast-Authentication-Secure *if* ϵ *is negligible as a function of the security parameter* n*.*

```
Game $\mathsf{Exp}_{\Pi}^{\mathsf{UCast\text{-}Auth}}(\mathcal{A})$
Let $O$ = {oRegCred, oChSession, oUAdd, oURmv, oChUpdate}
        $\cup${oReveal, oCorrupt, oPromptSend, oInjectSend}
(spar, ppar) $\leftarrow$ CBU2.Setup($n$)
done $\leftarrow \mathcal{A}^{O}$(ppar);
$\mathcal{A}$ wins iff. $\exists$CM s.t. $\exists$[sid, ($t_{\mathsf{sid}}$, $U_i$, $c$, aux, $m$)] $\in$ CM.Rcv
 and (simultaneously):
    $m \neq \perp$
    [sid, ($t_{\mathsf{sid}}$, $\cdot$, $\cdot$, $m$)] $\notin U_i$.Snd
    User $U_i$ created via oRegCred($U_i$, CM, 0, $\cdot$)
    CM created via oRegCred($\cdot$, CM, $\cdot$, 0)
    No oCorrupt(CM, $U_i$) query
```

Figure 11: The Unicast-authentication security game of CBU2.

Definition 21 (Broadcast-Security). *Consider a* CBU2 *channel denoted* CBU2. *For a probabilistic polynomial-time (PPT) adversary* $\mathcal{A}$ *we define its advantage* $\mathsf{Adv}_{\mathsf{CBU2}}^{\mathsf{BCast\text{-}Sec}}(\mathcal{A})$ *to win the* $\mathsf{Exp}_{\Pi}^{\mathsf{BCast\text{-}Sec}}(\mathcal{A})$ *security game presented in Figure 12 as follows:*

$$\mathsf{Adv}_{\mathsf{CBU2}}^{\mathsf{BCast\text{-}Sec}}(\mathcal{A}) = \left|\Pr[\mathcal{A} \textit{ wins } \mathsf{Exp}_{\Pi}^{\mathsf{BCast\text{-}Sec}}(\mathcal{A})] - \frac{1}{2}\right|. \quad (3)$$

The channel CBU2 *is* ϵ*-Broadcast-Secure if, and only if, any PPT adversary* $\mathcal{A}$ *against* CBU2 *has at most an advantage of* ϵ *to win the* $\mathsf{Exp}_{\Pi}^{\mathsf{BCast\text{-}Sec}}(\mathcal{A})$ *security game. Asymptotically,* CBU2 *is called* Broadcast-Secure *if* ϵ *is negligible as a function of the security parameter* n*.*

```
Game $\mathsf{Exp}_{\Pi}^{\mathsf{BCast\text{-}Sec}}(\mathcal{A})$
$b \xleftarrow{\$} \{0,1\}$
Let $O$ = {oRegCred, oChSession, oUAdd, oURmv, oChUpdate}
        $\cup${oReveal, oCorrupt, oSendRoR$_b$, oInjectSend}
(spar, ppar) $\leftarrow$ CBU2.Setup($n$)
$d \leftarrow \mathcal{A}^{O}$(ppar);
$\mathcal{A}$ wins iff. $d = b$ and (simultaneously) for any epoch $t$
and session sid s.t.
if oSendRoR$_($sid, $\perp$, BCast, $\cdot$) was queried, simultaneously:
    No oReveal(sid, $\cdot$, BCast, $t$) queried
    CM created via oRegCred($\cdot$, CM, $\cdot$, 0)
    If $U_i$ created via oRegCred($U_i$, CM, 0, $\cdot$), $U_i \notin \mathcal{UG}_{\mathsf{sid}}$ at epoch
$t$
```

Figure 12: The Broadcast-security game of CBU2.

Definition 22 (Unicast-Security). *Consider a* CBU2 *channel denoted* CBU2. *For a probabilistic polynomial-time (PPT) adversary* $\mathcal{A}$ *we define its advantage* $\mathsf{Adv}_{\mathsf{CBU2}}^{\mathsf{UCast\text{-}Sec}}(\mathcal{A})$ *to win the* $\mathsf{Exp}_{\Pi}^{\mathsf{UCast\text{-}Sec}}(\mathcal{A})$ *security game presented in Figure 13 as follows:*

$$\mathsf{Adv}_{\mathsf{CBU2}}^{\mathsf{UCast\text{-}Sec}}(\mathcal{A}) = \left|\Pr[\mathcal{A} \textit{ wins } \mathsf{Exp}_{\Pi}^{\mathsf{UCast\text{-}Sec}}(\mathcal{A})] - \frac{1}{2}\right|. \quad (4)$$

The channel CBU2 *is* ϵ*-Unicast-Secure if, and only if, any PPT adversary* $\mathcal{A}$ *against* CBU2 *has at most an advantage of* ϵ *to win the* $\mathsf{Exp}_{\Pi}^{\mathsf{UCast\text{-}Sec}}(\mathcal{A})$ *security game. Asymptotically,* CBU2 *is called* Unicast-Secure *if* ϵ *is negligible as a function of the security parameter* n*.*

```
Game $\mathsf{Exp}_{\Pi}^{\mathsf{UCast\text{-}Sec}}(\mathcal{A})$
$b \xleftarrow{\$} \{0,1\}$
Let $O$ = {oRegCred, oChSession, oUAdd, oURmv, oChUpdate}
        $\cup${oReveal, oCorrupt, oSendRoR$_b$, oInjectSend}
(spar, ppar) $\leftarrow$ CBU2.Setup($n$)
$d \leftarrow \mathcal{A}^{O}$(ppar);
$\mathcal{A}$ wins iff. $d = b$ and (simultaneously) for any epoch $t$ and
session sid s.t.
if oSendRoR$_($sid, $U_i$, UCast, $\cdot$) was queried, simultaneously:
    No oReveal(sid, $P$, UCast, $t$) queried for $P \in \{U_i, \mathsf{CM}\}$
    CM created via oRegCred($\cdot$, CM, $\cdot$, 0)
    $U_i$ created via oRegCred($U_i$, CM, 0, $\cdot$)
```

Figure 13: The Unicast-security game of CBU2.

Insight: constructing CBU2**.** At its core, the CBU2 channel consists of a manager-to-users broadcast channel with key-evolution, and multiple user-to-manager unicast channel which does not necessarily have to feature key-evolution. In both cases, confidentiality must be ensured, and the authentication property demands that the communication only run one-way, which requires the use of EUF-CMA-secure authentication with non-repudiation: typically signature schemes.

There are many ways to construct such channels.

A typical start would be to provide broadcast communication via a group-communication channel with post-compromise security, such as MLS, combined with a signature scheme that would allow only the channel manager to effectively send messages. Note, however, that the functionality required here differs a little from the standard MLS architecture. For one thing, the only entity that will be proposing the addition or removal of users is the group manager. In addition, since we will be using the CBU2 protocol in the interest of a privacy-preserving scheme, note that the true identities of the users in the group will not be known (we will be using channel-specific identifiers within the Secret Handshake scheme). This partially violates one of the core MLS properties: the fact that users are aware who is in the group. No users will be able to make proposals or commits to the channel. Key-updates are also only triggered by the channel manager.

We note that the unicast channel key-material could be derived, via a secure PRF, from the group secrets at the epoch at which the user has joined as well as a nonce known only to the manager and the user, chosen uniformly and independently at random during the Joining procedure. The derivation needs to preserve certain security properties, but could essentially work as described by Brzuska, Jacobsen, and Stebila in [7]. In this case as well, we would require the user to sign each sent message.

C.4.4 Anonymous Group Key-Agreement with Fresh Randomness (AGKA-FR). In this section we formalize in detail the AGKA-FR primitive, as described in Section 4.2. We consider mostly-generic anonymous group-key agreement, with a single functional constraint: the use, during the protocol run, of a single fresh random value, taken from a (large) set, which will eventually be used as state in the secret handshakes protocol.

More technically, the anonymous group key-agreement protocol is parametrized by a randomness superspace $\mathcal{R}$, which essentially imposes the *type* of randomness that is used (for instance, a group element, an integer, etc.). Then, during setup, depending on the security parameter, a subset of $\mathcal{R}$, denoted $\mathcal{R}^{\Pi}$, is chosen to become the set from which each protocol participant will output fresh randomness at each session. Moreover, honest protocol participants will abort AGKA-FR sessions if two distinct parties output the same randomness. At the end of a successful session, each party retains the fresh randomness, along with the computed session secret, as state to be used in the remainder of the secret handshake protocol run.

We discuss our choice of constraining the class of anonymous group key-agreement protocols in this way.

Intuition: why AGKA-FR **with fresh randomness.** First, it should be noted that the great majority of key-agreement protocols rely on the use of randomness: either in the form of nonces or in the form of random group elements. Therefore, while *theoretically* our choice restricts the class of group-key agreements that could be used in constructing secret handshakes, in *practice* the pick of protocols is not very restricted.

In our constructions of Secret Handshakes, we specifically need some means of binding the group key-agreement of users taking part in a secret handshake with the group membership of those users. Later in the protocol, we need that binding information in the construction of a list MAC, which should take in unique input per user. Finally, the concatenation of those random values will count as a session identifier, which must, in its own turn, be unique. On the other hand, we need to keep the group key-agreement unlinkable, even with respect to other handshake participants – thus precluding binding through authentication. Thus, we chose to bind the two phases of the secret-handshake protocol by means of a piece of randomness, generated uniformly and independently at random from a set whose size depends on the security parameter, and whose nature depends on the parameter $\mathcal{R}$. This, however, is not the only choice we considered.

An alternative could have been to explicitly require that the session key in the group key-agreement is then used to *generate* binding information. Yet, in order for the binding information to be unique the generation must use fresh randomness, which all the other users must know (hence, it must be exchanged during the protocol). Yet, that only seems to reduce this second approach to the first one.

An alternative, potentially valid approach is to enforce uniqueness of each user's state by associating them randomly with an index between 1 and the number of participants to the handshake. In order to furthermore ensure that AGKA-FR session identifiers are unique, one could furthermore associate each session with a single pseudorandom value, to which users will append their indexes. This potential solution, however, raises the question of how to ensure that the single pseudorandom value associated with each session is unique across all the sessions.

AGKA-FR **syntax and environment.** We consider a set of users USet, with individual users associated with identities U_i. The protocol is defined as a tuple of the following algorithms.

- $\mathsf{AGKA\text{-}FR.Setup}(1^{\lambda}, \mathcal{R}) \rightarrow (\mathsf{ppar}, \mathcal{K}, \mathcal{R}^{\Pi})$: This global setup algorithm takes in input a security parameter (in unary) and the superset of random values $\mathcal{R}$ and outputs global public parameters ppar and a subset $\mathcal{R}^{\Pi}$ of $\mathcal{R}$, which must be used to produce randomness. The public parameters ppar are taken implicitly in input to all the following algorithms. Finally, the algorithm also outputs the set $\mathcal{K}$ of possible computed master-secret values (we require that this set be surjective).
- $\mathsf{AGKA\text{-}FR.Handshake}(\Delta) \rightarrow (\{\mathsf{state}_i, \mathsf{sid}, \mathsf{ms}_i\}_{U_i \in \Delta}$: Given a set of users Δ, this interactive protocol outputs, for each participating user $U_i \in \Delta$, a set of three values: a piece of randomness $\mathsf{state}_i \in \mathcal{R}^{\Pi}$, which users choses uniformly and independently at random and broadcast at the beginning of the session; a session identifier sid consisting of the concatenation of all state_i values, from smallest to largest; and a master secret value ms. All of these values can take a special error value $\perp$.

Users U_i stores a table AGKA-FR.SList indexed by session identifiers sid and containing the following values:

$\mathsf{n}_{\mathsf{sid}}$: the number of users taking part in handshake sid, that is $\mathsf{n}_{\mathsf{sid}} = |\Delta_{\mathsf{sid}}|$.

state : the randomness used by that party during session with identifier sid. Note that $\mathsf{sid} \in (\mathcal{R}^{\Pi})^{\mathsf{n}_{\mathsf{sid}}}$ and that either $\mathsf{state} \in \mathsf{sid}$ or $\mathsf{state} = \perp$.

α_{sid} : the user's acceptance bit for the randomness generated during session sid. This bit is initially set to 0 and changes to 1 if the user accepts the validity of the session state sid, *i.e.*,, sid consists of $\mathsf{n}_{\mathsf{sid}}$ unique random values.

α : the user's acceptance bit for the validity of the entire protocol run. This value is initially set to 0 and may change to 1 if the user accepts the validity of the protocol run. We demand that $\alpha = 0$ whenever $\alpha_{\mathsf{sid}} = 0$.

ms : the master secret computed by the user in session sid of AGKA-FR, which can also potentially be equal to $\perp$.

ρ : a reveal bit, initially set to 0, which will be set to 1 if the key is revealed during an attack.

We require the following notion of correctness.

Definition 23 (AGKA-FR Correctness). *Let* $\mathsf{AGKA\text{-}FR} = (\mathsf{AGKA\text{-}FR.Setup}, \mathsf{AGKA\text{-}FR.Handshake})$ *be a* AGKA-FR *protocol parametrized with randomness superspace* $\mathcal{R}$ *and used by a set* USet*. For every tuple of values* $(\mathsf{ppar}, \mathcal{R}^{\Pi}) \leftarrow \mathsf{AGKA\text{-}FR.Setup}(1^{\lambda}, \mathcal{R})$*, for every* $\Delta \in \mathsf{USet}$ *and* $\mathsf{AGKA\text{-}FR.Handshake}(\Delta) \rightarrow (\{\mathsf{state}_i, \mathsf{sid}, \mathsf{ms}_i\}_{U_i \in \Delta}$*, the following statements hold simultaneously:*

- *All users* $U_i \in \Delta$ *choose and broadcast* state *independently and uniformly at random from* $\mathcal{R}^{\Pi}$.
- *If* $\exists \mathsf{state}, \mathsf{state}' \in \mathsf{sid}$ *with* $\mathsf{state} = \mathsf{state}'$*, then* $\alpha_{\mathsf{sid}} = 0\ \forall U_i \in \Delta$*;*
- *If* $\forall \mathsf{state}, \mathsf{state}' \in \mathsf{sid}$ *it holds that* $\mathsf{state} \neq \mathsf{state}'$*, then* $\alpha_{\mathsf{sid}} = 1\ \forall U_i \in \Delta$*;*
- $\forall U_i \in \Delta$*, if* $\alpha_{\mathsf{sid}} = 0$*, then* $\alpha = 0$*;*
- $\forall U_i \in \Delta$*, if* $\alpha = 1$*, then all* U_i *compute the same values* ms *and* sid *at the end of the session;*

Security notions for AGKA-FR. Intuitively, AGKA-FR should guarantee the following properties: anonymity and the security of the computed keys. We define anonymity in terms of the unlinkability between two participating users, and prove that this property also implies a type of *simulatability*, *i.e.,* it is impossible to tell whether the protocol participant is a real user or a simulator.

Since users are not associated with private values, unlike in our secret handshake and CBU2 protocols, for AGKA-FR it makes no sense to consider corrupt users: just honest and malicious ones. We describe the following oracles, which will allow the adversary to interact with its environment.

- $\underline{U_i \leftarrow \mathsf{oNewUser}(\mathsf{mal})}$: on input a bit mal, indicating whether a new user is created honest ($\mathsf{mal} = 0$) or malicious ($\mathsf{mal} = 1$), this oracle creates a new user U_i, whose identity is output to the adversary. The challenger will keep track of which users are malicious and which are honest.
- $\underline{(\mathsf{sid}, \tau) \leftarrow \mathsf{oNewSession}(\Delta)}$: on input a set of users Δ of size at least 2, which may consist of both honest and malicious users, this algorithm runs the handshake as a black box with the indicated users. The attacker receives the transcript of the session, τ, as well as the session identifier sid. Note that, because of the way sid is defined above, it is a value that can be retrieved directly from τ. The adversary explicitly gets this value so that it can use it later for potential key-retrieval.
- $\underline{\mathsf{ms} \cup \perp \leftarrow \mathsf{oReveal}(\mathsf{sid}, U_i)}$: on input the session identifier sid of a prior session, as well as the identity of a user U_i, this oracle first checks that U_i is honest (otherwise, the oracle outputs $\perp$). For an honest user U_i, this oracle outputs the value stored by this user as the master secret of sid.
- $\underline{(\mathsf{sid}, \tau) \leftarrow \mathsf{oLoRNewSession}^b(\hat{\Delta}, U, U')}$: On input a core set of users $\hat{\Delta}$ and two honest users U, U' such that $\hat{\Delta} \cap \{U, U'\} = \emptyset$, this oracle, parametrized by a bit b, it sets $\Delta = \hat{\Delta} \cup \{U\}$ for $b = 0$ and $\Delta = \hat{\Delta} \cup \{U'\}$ for $b = 1$. It then runs a handshake between users in Δ, producing the transcript τ, which is returned to the adversary together with the session identifier sid.
- $\underline{\mathsf{ms} \cup \perp \leftarrow \mathsf{oRoRHandshake}_b(\mathsf{sid}, U_i)}$: on input the session identifier sid corresponding to a prior session, and a user U_i, and parametrized by a bit b, this oracle first checks that user U_i has stored in its database an entry indexed by the identifier sid. If that is not the case, the oracle outputs $\perp$. If U_i has such an entry, the oracle checks that the entry stored by U_i for session sid is a value $\mathsf{ms} \neq \perp$. Finally, it also checks that all users involved in sid are honest. Then, if $b = 0$, the oracle outputs the true entry ms stored by U_i for sid; else, if $b = 1$, the oracle outputs a random value s chosen at uniformly at random from the key set $\mathcal{K}$.

Definition 24 (AKE-Security for AGKA-FR). *Consider a* AGKA-FR *protocol. For a probabilistic polynomial-time (PPT) adversary $\mathcal{A}$ we define its advantage $\mathsf{Adv}^{\mathsf{GKA\text{-}AKE}}_{\mathsf{AGKA\text{-}FR}}(\mathcal{A})$ to win the $\mathsf{Exp}^{\mathsf{GKA\text{-}AKE}}_{\Pi}(\mathcal{A})$ security game presented in Figure 14 as follows:*

$$\mathsf{Adv}^{\mathsf{GKA\text{-}AKE}}_{\mathsf{AGKA\text{-}FR}}(\mathcal{A}) = \left|\Pr[\mathcal{A} \text{ wins } \mathsf{Exp}^{\mathsf{GKA\text{-}AKE}}_{\Pi}(\mathcal{A})] - \frac{1}{2}\right|. \quad (5)$$

The protocol AGKA-FR *is ϵ-AKE-Secure if, and only if, any PPT adversary $\mathcal{A}$ against the AKE-security of* AGKA-FR *has at most an advantage of ϵ to win the $\mathsf{Exp}^{\mathsf{GKA\text{-}AKE}}_{\Pi}(\mathcal{A})$ security game. Asymptotically,* AGKA-FR *is called* AKE-Secure *if ϵ is negligible as a function of the security parameter 1^λ.*

Definition 25 (Unlinkability in AGKA-FR). *Consider a* AGKA-FR *protocol. For a probabilistic polynomial-time (PPT) adversary $\mathcal{A}$ we define its advantage $\mathsf{Adv}^{\mathsf{GKA\text{-}Unlink}}_{\mathsf{AGKA\text{-}FR}}(\mathcal{A})$ to win the $\mathsf{Exp}^{\mathsf{GKA\text{-}Unlink}}_{\Pi}(\mathcal{A})$ security game presented in Figure 14 as follows:*

$$\mathsf{Adv}^{\mathsf{GKA\text{-}Unlink}}_{\mathsf{AGKA\text{-}FR}}(\mathcal{A}) = \left|\Pr[\mathcal{A} \text{ wins } \mathsf{Exp}^{\mathsf{GKA\text{-}Unlink}}_{\Pi}(\mathcal{A})] - \frac{1}{2}\right|. \quad (6)$$

The protocol AGKA-FR *is ϵ-Unlinkable if, and only if, any PPT adversary $\mathcal{A}$ against the Unlinkability of* AGKA-FR *has at most an advantage of ϵ to win the $\mathsf{Exp}^{\mathsf{GKA\text{-}Unlink}}_{\Pi}(\mathcal{A})$ security game. Asymptotically,* AGKA-FR *is called* Unlinkable *if ϵ is negligible as a function of the security parameter 1^λ.*

D SECURITY ANALYSIS

In this section, we begin by formalizing the security properties we want our scheme to achieve, and then proceed to prove that the latter are guaranteed.

D.1 Winning condition notations

Definition 26 (Queries). *Let O denote a set of oracles, and let $\mathbb{Q}$ refer to a set of queries. For every $o \in O$, we write $o : i \mapsto r$ to indicate that i is input to the oracle o and r, the oracle's response. Whenever a query $o : i \mapsto r$ is made, a value (r, i) is added to a list $\mathbb{Q}^o$, or more formally:*

$$\forall o \in O, o : I \longrightarrow R, \forall (r, i) \in R \times I,$$
$$o : i \mapsto r \equiv (r, i) \in \mathbb{Q}^o$$

Definition 27 (Entities sets). *We denote by $\mathcal{MU}, \mathcal{HU}, \mathcal{CU}$ the sets of Malicious, respectively Honest and Corrupted users. Sets of users, agnostically of their respective group, are defined as follows:*

$$U \in \mathcal{U} = \mathcal{MU} \sqcup \mathcal{CU} \sqcup \mathcal{HU}$$

The sets of group authorities are similarly denoted and,

$$\mathsf{GA} \in \mathcal{GA} = \mathcal{MGA} \sqcup \mathcal{CGA} \sqcup \mathcal{HGA}$$

Definition 28 (User entity). *For a user U, we use the notations:*

- accept_i *indicates whether the user accepted the handshake and has computed the key for the i-th handshake*
- $\mathsf{acceptban}_j$ *indicates wether the user is convinced that the banned user are correctly justified (if the proof of their implication in a valid transcript is generated correctly)*
- group_i *denotes the group the user played with during the* i-th *handshake*
- groups_i *denotes all the groups the user has joined prior to the* i-th *handshake.*

Definition 29 (USet_G). *We propose two valid notations for* USet_G *to ease the notation, by fixing either the i counter:*

$$\mathsf{USet}_{G,i} = \{q.U : q \in \mathbb{Q}^{\mathsf{oJoin}(G,\cdot)}, q.i < i\}$$

or also the j counter:

$$\mathsf{USet}_{G,i,j} = \{q.U : q \in \mathbb{Q}^{\mathsf{oJoin}(G,\cdot)}, q.j \leq j\}$$

```
Game $\mathsf{Exp}_{\Pi}^{\mathsf{GKA\text{-}AKE}}(\mathcal{A})$
$(\mathsf{ppar}, \mathcal{K}, \mathcal{R}^{\Pi}) \leftarrow \mathtt{AGKA\text{-}FR.Setup}(1^{\lambda}, \mathcal{R})$
$b \overset{\$}{\leftarrow} \{0,1\}$
Let $O = \{\mathsf{oNewUser}, \mathsf{oNewSession}, \mathsf{oReveal}, \mathsf{oRoRHandshake}_b\}$
$d \leftarrow \mathcal{A}^{O}(\mathsf{ppar}, \mathcal{K}, \mathcal{R});$
$\mathcal{A}$ wins iff. $d = b$
        and no sid output by oRoRHandshake input to oReveal
```

```
Game $\mathsf{Exp}_{\Pi}^{\mathsf{GKA\text{-}Unlink}}(\mathcal{A})$
$(\mathsf{ppar}, \mathcal{K}, \mathcal{R}^{\Pi}) \leftarrow \mathtt{AGKA\text{-}FR.Setup}(1^{\lambda} s, \mathcal{R})$
$b \overset{\$}{\leftarrow} \{0,1\}$
$O = \{\mathsf{oNewUser}, \mathsf{oNewSession}, \mathsf{oReveal}, \mathsf{oLoRNewSession}_b\}$
$d \leftarrow \mathcal{A}^{O}(\mathsf{ppar}, \mathcal{K}, \mathcal{R});$
$\mathcal{A}$ wins iff. $d = b$
```

Figure 14: The AKE (left) and Unlink (right) security notions for AGKA-FR.

Definition 30 ($\mathcal{CU}_{(\cdot)}$). *Therefore, we can add a notation to $\mathcal{CU}$, which is :*

$$\forall s \in \mathbb{Q}^{\mathsf{oSHandshake}},$$

$$\mathcal{CU}_{s.i} = \{c.U : c \in \mathbb{Q}^{\mathsf{oCorruptUser}}, c.i < s.i\}$$

For sets not specified we have:

$$\mathcal{CU} = \mathcal{CU}_{\#\mathbb{Q}^{\mathsf{oSHandshake}}}$$

For other sets we have:

$$\mathcal{HU}_i = \mathcal{U} \setminus (\mathcal{CU}_i \cup \mathcal{MU}_i)$$

where,

$$\mathcal{MU}_i = \{r.U : r \in \mathbb{Q}^{\mathsf{oUReg}(\cdot,1)}, r.i < s.i\}$$

And so on for $\mathcal{CGA}$, $\mathcal{HGA}$, and $\mathcal{MGA}$.

D.2 Adversarial model

We define security by means of a number of *security games*, played between the challenger and an adversary. A peculiarity of the security of secret handshakes is that the adversary can always register malicious users, or even create its own groups; most security notions, however, will concern groups for which the adversary does not (at the moment of the challenge phase) control any users.

Honest, malicious, and corrupted entities. We distinguish between three types of entities. *Honest* users (whose identities are stored in a list $\mathcal{L}_{\mathsf{honest}}$) are not controlled by the adversary (though the latter might gain oracle access to them), and their private parameters (keys, state) remains unknown to $\mathcal{A}$. At the opposite extreme, *malicious* users (whose identities are stored in a list $\mathcal{L}_{\mathsf{malicious}}$) are fully controlled by the adversary, which in addition knows all their private parameters and internal state. Finally, *corrupted* users (whose identities are stored in $\mathcal{L}_{\mathsf{corrupt}}$) begin as honest, but are at some point corrupted by the attacker, which learns the user's private key. However, as opposed to malicious users, corrupt users retain privacy of some of their state (such as accept/reject bits, session keys, or intermediate handshake-protocol values), and are not controlled by the adversary (except through oracle queries). We do, however, allow the adversary to learn their private keys even after updates.

Group authorities can also be honest, malicious, or corrupted. Note that, as opposed to users, group authorities cannot be revoked – as a result, corruption of a previously honest authority essentially renders the group malicious.

Party state and attributes. For administrative purposes, the parties involved in our model will retain some state, which may be updated according to their actions (for instance, when users join a new group, leave a group, or perform a new handshake). We consider essentially two distinct types of parties, belonging to two disjoint sets: common users and group authorities. Each of these will maintain a distinct state.

Regular users will have to keep state relating to both long-term group membership, and to the handshakes they ran. Thus, users U maintain a *group-related state*

Call-oracle management. Consider that each query are sequential, therefore we can order them in a particular sequence and check if it were respected by the $\mathcal{A}$. In our model we introduce two counters, one is called the universal counter and the other one is called group counter. In facts, we have two types of main oracles the ones who are internal to groups such as oJoin, oLeave, or oUpdate, and another type which runs action over users from different groups such as oSHandshake. Therefore, those types of oracles will have each two counters therefore we can manipulate easily our model to know the order of action that were run. We note that for group-crossing-users oracles (*e.g.,* oSHandshake,oSHandshakeLoR) only use the universal counter.

Oracles. The challenger of each security game embodies all the honest parties and knows their private keys. All the parties, including the adversary $\mathcal{A}$, will know the public parameters and keys involved in the scheme. Moreover, we allow $\mathcal{A}$ to query the following oracles:

- $U \cup \bot \leftarrow \mathsf{oUReg}(U, b)$: This oracle allows an adversary to register a user U as honest ($b = 0$) or malicious ($b = 1$). In either case, the oracle first verifies whether the identity U has already been queried to the oracle before and, if that is the case, it responds with $\bot$. Otherwise, it returns to the adversary the handle U, and internally adds U to an (initially empty) list $\mathcal{L}_{\mathsf{honest}}$ – if $b = 0$ – or to an (initially empty) list $\mathcal{L}_{\mathsf{malicious}}$ – if $b = 0$.
- $(G, \mathsf{GA}, \mathsf{spar}_G, \mathsf{ppar}_G) \leftarrow \mathsf{oNewGroup}(b)$: This oracle allows for the creation of a group G (with a unique identifier G). The group can be either honestly created (the input bit is $b = 0$) or malicious ($b = 1$). Whenever a new group is created, a new manager identity GA is also created and returned to the adversary. Finally, the oracle returns $(\mathsf{spar}_G, \mathsf{ppar}_G)$. If $b = 0$ and the group is thus honest, the returned spar_G value is set to $\bot$ (the adversary only learns the public group parameters). The challenger keeps track of all the groups by means of a database, whose entries are of the form $(G, b, \mathsf{GA}, \mathsf{spar}_G, \mathsf{ppar}_G, \mathsf{USet}_G)$. Initially, each group

will be associated with an empty user-set, *i.e.,* $\mathsf{USet}_G := \emptyset$. If, for such an entry, the bit $b = 1$ (the group was maliciously created), then the adversary is assumed to have taken control of the group authority GA.

- $(i, j, \{\mathsf{sk}_U, \mathsf{pk}_U\}, \{\mathsf{sk}_V, \mathsf{pk}_V\}_{V \in \mathsf{USet}_G}, \mathsf{USet}_G) \leftarrow \mathsf{oJoin}(U, G)$: This oracle allows an adversary to prompt the honest group authority GA of a group G to permit a user U (honest or malicious) to join up. As a result, U is added (by the challenger) to G and the parameters of all the members of the new group (including those of U) are output. For each honest user $V \in \mathsf{USet}_G$, the adversary only gets the public key pk_V, whereas the output sk_V is set to $\perp$. By contrast, for each $V \in \mathsf{USet}_G$ that is malicious or has previously been corrupted, the adversary receives the genuine private key sk_V. The counters i and j are returned, where $i = \#\mathbb{Q}^{\mathsf{oSHandshake}}$ and $j = \#\mathbb{Q}^{\mathsf{oUpdate}(G)} + \#\mathbb{Q}^{\mathsf{oJoin}(\cdot,G)} + \#\mathbb{Q}^{\mathsf{oLeave}(\cdot,G)}$.
- $\perp \sqcup (i, j, \mathsf{USet}_G, \{U_L < \mathsf{sk}_L, \mathsf{pk}_L >\}_{L \in \mathsf{USet}_{G,i,j-1} \setminus \mathsf{USet}_G}) \leftarrow \mathsf{oLeave}(G, U)$: First, this oracle, disallow to input U with all secret parameters of another user. Therefore, the security model always assume that at least one of the U's secret material is used, otherwise return $\perp$. This oracle returns, depending on the protocol's design, whether or not the secret material of the users who have left the group G and the USet_G obtained with the counters i, and j which is incremented.
- $(i, j, B, D, \{U_V < \mathsf{sk}_V, \mathsf{pk}_V >\}_{V \in \mathsf{USet}_G}, \{U_B < \mathsf{sk}_B, \mathsf{pk}_B >\}_{B \notin \mathsf{USet}_G}, \mathsf{USet}_G, \pi^{\mathsf{BAN}}) \leftarrow \mathsf{oBan}(G, \mathsf{tr}, p)$: This oracle bans a user from a given transcript addressed with a specific index p to point out which part of the transcript is revelant for the banishing including the data use to specify which participant is faulty (*e.g.,* it could be the τ issued during the handshake or even the KShare_U as in our protocol) and *idem* returns counters i and j. It also returns two sets B and D. The first one B is the set of users that are marked as banned, while D is the set of users marked as users which secret materials had leaked and been used during oBan. In our protocol LCA, the set B is in LCA F, and the set D corresponds in LCA to B and C.
- $(i, j, \{\mathsf{sk}_U, \mathsf{pk}_U\}_{U \in \mathsf{USet}_G}, \mathsf{USet}_G) \leftarrow \mathsf{oUpdate}(G)$: This oracle allows the adversary to exploit the key-updating process of the secret handshake, even outside of joining and leaving actions. For each $U \in \mathsf{USet}_G \cap \mathcal{L}_{\mathsf{honest}}$, the adversary is given only the new pk_U, while the output for sk_U is $\perp$. For users $U \in \mathsf{USet}_G \cap (\mathcal{L}_{\mathsf{corrupt}} \cup \mathcal{L}_{\mathsf{malicious}})$, both values are honestly output. Also, the counters are returned *idem* for counters i and j.

Remark 7. *Order of oracles operating over a specific group can be determined with counters. e.g.,* oLeave *is always followed by* oUpdate *could be written as:*

$$\forall f \in \mathbb{Q}^{\mathsf{oLeave}(,G)}, \exists u \in \mathbb{Q}^{\mathsf{oUpdate}(,G)}, f.j < u.j$$

- $(i, \mathsf{sk}_U) \leftarrow \mathsf{oCorruptUser}(U)$: The adversary is given the ability to corrupt users $U \in \mathcal{L}_{\mathsf{honest}}$. If $U \notin \mathcal{L}_{\mathsf{honest}}$, the oracle returns $\perp$. Else, the oracle returns sk_U, removes U from $\mathcal{L}_{\mathsf{honest}}$, then adds it to $\mathcal{L}_{\mathsf{corrupt}}$, and returns with it i which is equal to the number of calls of oSHandshake made. This index i allows us to know when the corruption were made.
- $(\mathsf{spar}_G, \mathsf{state}) \leftarrow \mathsf{oCorruptGA}(G)$: The adversary can also corrupt group authorities of groups G which were honestly created (the group database has $b = 0$, marking that the group is honest), thus receiving their full state and private group parameters spar_G. The bit b is flipped for that group in the database, to $b = 1$ (they will be considered malicious from now on).

The handshake oracle. A special oracle will be provided to the adversary in order to allow it to run secret handshake session with a number of honest users. In particular we assume that no honest or corrupted user will actually engage of its own volition in a handshake prompted by a malicious party: in order to achieve this, the adversary has to query the oSHandshake oracle that we describe below.

At each session i, each user U that is not adversarially-controlled will keep track of the following values: a session identifier sid_U^i, a point-of-view transcript for the session tr_U^i, a session key k_U^i, potentially set to $\perp$ while the handshake is running and if the handshake is rejected by that party, and finally, a number of partners n_U^i.

- $\underline{i, \mathsf{tr}_i \leftarrow \mathsf{oSHandshake}(\mathsf{SHSet})}$: Given a set of users noted SHSet, which may include malicious, honest, and corrupted users, this oracle allows the parties to run a new session of the secret handshake protocol. The challenger will play the part of the honest and corrupted parties, whereas the adversary controls the malicious parties. We index calls to this oracle by a counter, initialized at 1 for the first query, and output the counter i and a transcript tr (consisting of messages exchanged over a public channel) back to the adversary. Note that if the adversary has included malicious or corrupted parties in the handshake, it will also update its state in terms of the information it has been provided throughout the protocol run.

D.3 Discussion about the new paradigm Traitor Catching

The Traitor Tracing paradigm, called traitor tracing, appears first in [10], where the main idea is to essentially watermark sensitive data to make it traceable. "If only one person is told about some secret, and this next appears in the evening news, then the guilty party is evident. A more complex situation arises if the set of people that have access to the secret is large. The problem of determining guilt or innocence is (mathematically) insurmountable if all people get, the exact same data and one of them behaves treacherously and reveals the secret." – [10]. We note that this is the standard mechanism used in the literature of secret handshake, finding its roots in the original article from Balfanz *et al.* in [3] by means of traceable signatures.

Here we propose a shift to a methodology that we call Traitor Catching. We can illustrate it as follows. Consider the perpetrator of a felony and a law-enforcement officer who is unable to narrow down a set of suspects. If this is the status quo, then the scheme

```
$\mathsf{Exp}^{\mathsf{Auth}}_{\mathsf{SHS}}()$
----------------------------------------
$(\mathsf{msk}, \mathsf{ppar}) \leftarrow \mathsf{SHS.Setup}(1^\lambda)$
Let $\mathcal{O}_{\mathsf{Auth}}$:= {oUReg, oNewGroup, oJoin, oLeave, oUpdate,
    oCorruptUser, oCorruptGA, oSHandshake}
$\mathsf{DONE} \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Auth}}}(1^\lambda)$
----------------------------------------
$\mathcal{A}$ wins $\iff \exists s \in \mathbb{Q}^{\mathsf{oSHandshake}}, s.\mathsf{Win} = 1$
```

Figure 15: The user authentication game.

fails to guarantee Traitor Tracing. On the other hand, even if the officer initially does not know who the traitor is, they can lay a trap and try to find the perpetrator in that way. This is Traitor Catching. Informally, Traitor Catching is defined as an adversary being unable to ensure a handshake with at least one honest user is successful after being banned.

Traitor Tracing vs. Traitor Catching:. The two approaches each have their own pros and cons. The Traitor Catching property has a greater complexity. On the other hand, it also offers better privacy, and also prevents potential intrusion within the group after the tracing is done. In our construction, we rely on a combination of self-distinction and traitor catching, which ensures that the group authority can also trace corrupted and malicious users if the adversary tries to hide during traitor catching.

D.4 Security Definitions

```
$\mathsf{Exp}^{\mathsf{Unlink}}_{\mathsf{SHS}}$
----------------------------------------
Let $\mathcal{O}_{\mathsf{Unlink}}$ := {oUReg, oNewGroup, oJoin, oLeave, oUpdate,
    $\mathsf{oSHandshakeLoR}^b$}
$(\mathsf{ms}, \mathsf{ppar}) \leftarrow \mathtt{Setup}(1^\lambda)$
$b \xleftarrow{\$} \{``L", ``R"\}$
$b' \leftarrow \mathcal{A}^{\mathcal{O}}_{\mathsf{Unlink}}(\mathsf{ppar})$
----------------------------------------
$\mathcal{A}$ wins $\iff b = b'$
$\wedge \forall q \in \mathbb{Q}^{\mathsf{oSHandshakeLoR}}, \{q.U_R, q.U_L\} \cap (\mathcal{MU} \cup \mathcal{CU}) = \emptyset$
$\wedge U_L.\mathsf{groups}_{q.i} = U_R.\mathsf{groups}_{q.i}$
$\wedge \{U_L.\mathsf{group}_{q.i}.\mathsf{GA}, U_R.\mathsf{group}_{q.i}.\mathsf{GA}\} \cap (\mathcal{CGA}_{q.i} \cup \mathcal{MGA}_{q.i}) = \emptyset$
```

Figure 16: Experiment Unlinkability

```
$\mathsf{oSHandshakeLoR}^b(\mathsf{SHSet}, (U_L, U_R))$
----------------------------------------
if $\mathsf{SHSet} \cap \{U_L, U_R\} \neq \emptyset$
    return $\perp$
$\Delta \leftarrow \mathsf{SHSet} \cup < U_b >$
$(i, \mathsf{tr}) \leftarrow \mathsf{Handshake}(\Delta)$
$DB_{LR} \leftarrow DB_{LR} \cup \{(i, \mathsf{tr}, U_L, U_R)\}$
```

Figure 17: Oracle oSHandshakeLoR

```
$\mathsf{Exp}^{\mathsf{NF}}_{\mathsf{SHS}}$
----------------------------------------
Let $\mathcal{O}_{\mathsf{NF}}$ := {oUReg, oNewGroup, oJoin, oLeave, oBan, oUpdate,
    oSHandshake, oCorruptGA, oCorruptUser}
$(\mathsf{ms}, \mathsf{ppar}) \leftarrow \mathtt{Setup}(1^\lambda)$
$(G, \mathsf{tr}, \tau) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{NF}}}(\mathsf{ppar})$
----------------------------------------
$\mathcal{A}$ wins $\iff \exists r \in \mathbb{Q}^{\mathsf{oBan}(G,\mathsf{tr},\mathsf{pointer})}, \exists U \in \mathcal{HU}, \forall V \in \mathcal{HU},$
$U \in r.B \wedge V.\mathsf{acceptban}_{r.j}$
$\wedge \neg(\exists s \in \mathbb{Q}^{\mathsf{oSHandshake}}, U \in s.\Delta \wedge \mathsf{tr} = s.\mathsf{tr})$
```

Figure 18: Experiment Non-frameability

```
$\mathsf{Exp}^{\mathsf{Hand\text{-}Sim}}_{\mathsf{SHS}}$
----------------------------------------
Let $\mathcal{O}_{\mathsf{Hand\text{-}Sim}}$ := {oUReg, oNewGroup, oJoin, oLeave, oUpdate,
    $\mathsf{oSHandshakeLoR}^b(\cdot, (\cdot, \mathtt{SIM}))$}
$(\mathsf{ms}, \mathsf{ppar}) \leftarrow \mathtt{Setup}(1^\lambda)$
$b \xleftarrow{\$} \{``L", ``R"\}$
$b' \leftarrow \mathcal{A}^{\mathcal{O}}_{\mathsf{Hand\text{-}Sim}}(\mathsf{ppar})$
----------------------------------------
$\mathcal{A}$ wins $\iff b = b'$
$\wedge \forall q \in \mathbb{Q}^{\mathsf{oSHandshakeLoR}(\cdot,\cdot,\mathtt{SIM})}, q.U_L \notin \mathcal{MU} \cup \mathcal{CU}$
$\wedge q.\mathsf{group}.\mathsf{GA} \in \mathcal{HGA} \wedge \#\mathsf{USet}_{q.U_L.\mathsf{group},q.i} \geq 2$
```

Figure 19: Experiment Handshake simulability

```
$\mathsf{Exp}^{\mathsf{S\text{-}Dist}}_{\mathsf{SHS}}$
----------------------------------------
Let $\mathcal{O}_{\mathsf{S\text{-}Dist}}$ := {oUReg, oNewGroup, oJoin, oLeave, oUpdate,
    , oCorruptUser, oCorruptGA, oSHandshake, oTrace}
$(\mathsf{ms}, \mathsf{ppar}) \leftarrow \mathtt{Setup}(1^\lambda)$
$\mathsf{DONE} \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{S\text{-}Dist}}}(1^\lambda)$
----------------------------------------
$\mathcal{A}$ wins $\iff \exists e \in \mathbb{Q}^{\mathsf{oSHandshake}}, \exists U \in e.\Delta \cap \{\mathcal{HU} \cup \mathcal{CU}\},$
$\#(U.\mathsf{plist}_{e.i}) < \#e.\Delta \wedge U.\mathsf{accept}_{e.i} = 1$
```

Figure 20: Experiment Self distinction

User authentication. The security definition for this property in [20] is somewhat imprecise, both in terms of oracle formalisations and in terms of the security game itself. As a result, we modify it and (hopefully) add the necessary precision.

We will define a Win predicate for each oSHandshake query, which will intuitively evaluate to 1 if, and only if, the adversary is able to break authentication for that particular call, and 0 otherwise.

Definition 31 (Win for Auth.). *Let* oSHandshake(SHSet) *be the adversary's* i-th *query to the* oSHandshake *oracle, with the entire protocol execution that leads to completion. We evaluate the predicate* Win *for that handshake to* 0*, except if all the following conditions hold (in which case,* Win *evaluates to* 1*):*

- $\exists U \in \mathsf{SHSet} \cap (\mathcal{L}_{\mathsf{honest}} \cup \mathcal{L}_{\mathsf{corrupt}})$ *which has computed, in that session a key* $k_U^i \neq \perp$*;*
- $\nexists G$ *such that* $\forall U \in \mathsf{SHSet}$*, a query* $\mathsf{oJoin}(U, G)$ *was made, but no* $\mathsf{oLeave}(U, G)$ *query was also made, prior to the execution of the* i-th *handshake;*
- *Letting* $Gr := \{G : \forall U \in \mathsf{SHSet} \cap (\mathcal{L}_{\mathsf{honest}} \cup \mathcal{L}_{\mathsf{corrupt}})\}$*:*
  - $\mathcal{A}$ *has never queried* $\mathsf{oCorruptGA}(G)$ *for* $G \in Gr$*;*

$\mathsf{Exp}^{\mathsf{Catch}}_{\mathsf{SHS}}$

Let $\mathcal{O}_{\mathsf{Catch}} := \{\mathsf{oUReg}, \mathsf{oNewGroup}, \mathsf{oJoin}, \mathsf{oLeave}, \mathsf{oUpdate}, \mathsf{oCorruptUser}, \mathsf{oCorruptGA}, \mathsf{oSHandshake}\}$
$\mathsf{DONE} \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Catch}}}(1^\lambda)$

$$\begin{aligned}
&\mathcal{A} \text{ wins} \iff \\
&\exists s \in \mathbb{Q}^{\mathsf{oSHandshake}}, \exists H \in s.\Delta \cap (\mathcal{HU} \cup \mathcal{CU}), \\
&H.\mathsf{accept}_{s.i} = 1 \wedge H.\mathsf{group}_{s.i}.\mathsf{GA} \in \mathcal{HGA}_{s.i} \\
&\wedge \mathcal{MU}_{s.i} \cap \mathsf{USet}_{H.\mathsf{group}_{s.i}, s.i} = \\
&\quad (\{V \in \mathcal{MU}_{s.i} : \exists e \in \mathbb{Q}^{\mathsf{oJoin}(V, H.group_{s.i})}, \\
&\qquad \exists f \in \mathbb{Q}^{\mathsf{oLeave}(V, H.\mathsf{group}_{s.i})}, \exists u \in \mathbb{Q}^{\mathsf{oUpdate}(H.group_{s.i})}, \\
&\qquad e.i \leq f.i \leq u.i < s.i \ \wedge \ e.j < f.j < u.j\} \\
&\quad \cup \{I \in m.\Delta \cap \mathcal{MU}_{s.i} : \exists m \in \mathbb{Q}^{\mathsf{oSHandshake}}, \\
&\qquad \exists r \in \mathbb{Q}^{\mathsf{oJoin}(I, H.\mathsf{group}_{s.i})}, \exists b \in \mathbb{Q}^{\mathsf{oBan}(H.\mathsf{group}_{s.i}, m.\mathsf{tr}, m.\tau_I)} \\
&\qquad \exists p \in \mathbb{Q}^{\mathsf{oUpdate}(H.\mathsf{group}_{s.i})}, \\
&\qquad r.i \leq b.i \leq p.i \leq m.i \ \wedge \ r.j < b.j < p.j\}) \\
&\quad \cap \mathsf{USet}_{H.\mathsf{group}_{s.i}, s.i}
\end{aligned}$$

Figure 21: Experiment Traitor Catching

$\mathsf{Exp}^{\mathsf{Res\text{-}Hide}}_{\mathsf{SHS}}$

Let $\mathcal{O}_{\mathsf{Res\text{-}Hide}} := \{\mathsf{oUReg}, \mathsf{oNewGroup}, \mathsf{oJoin}, \mathsf{oLeave}, \mathsf{oUpdate}, \mathsf{oSHandshakeLoR}^b(\cdot, (\cdot, \mathtt{SIM}))\}$
$(\mathsf{ms}, \mathsf{ppar}) \leftarrow \mathtt{Setup}(1^\lambda)$
$b \xleftarrow{\$} \{\text{“}L\text{”}, \text{“}R\text{”}\}$
$b' \leftarrow \mathcal{A}^{\mathcal{O}}_{\mathsf{Res\text{-}Hide}}(\mathsf{ppar})$

$$\begin{aligned}
&\mathcal{A} \text{ wins} \iff b = b' \\
&\wedge [\forall s \in \mathbb{Q}^{\mathsf{oSHandshakeLoR}}, \forall V \in s.\mathsf{SHSet}, \\
&s.U_R.\mathsf{groups}_{s.i} \neq V.\mathsf{groups}_{s.i} = s.U_L.\mathsf{groups}_{s.i} \\
&\wedge \{V, s.U_L, s.U_R\} \subset \mathcal{HU}]
\end{aligned}$$

Figure 22: Experiment Result-hiding

– *if $\mathcal{A}$ queried* $\mathsf{oCorruptUser}(U)$ *for any user U for which* $\mathsf{oJoin}(U, G)$ *was queried with $G \in Gr$, then* $\mathsf{oLeave}(U, G)$ *was also queried prior to beginning the i-*th $\mathsf{oSHandshake}$ *interaction. Notice that in this case U need not be part of* SHSet.

A formal definition of this predicate translates to:

$$\begin{aligned}
&\mathcal{A} \text{ wins} \\
\iff &\exists s \in \mathbb{Q}^{\mathsf{oSHandshake}}, \exists H \in s.\Delta \cap \{\mathcal{HU} \cup \mathcal{CU}\}, \\
&H.\mathsf{accept}_{s.i} = 1 \wedge H.\mathsf{group}_{s.i}.\mathsf{GA} \in \mathcal{HGA}_{s.i} \\
&\wedge (\forall U \in s.\Delta, \forall e \in \mathbb{Q}^{\mathsf{oJoin}(U, H.\mathsf{group}_{s.i})}, \\
&\exists f \in \mathbb{Q}^{\mathsf{oLeave}(U, H.\mathsf{group}_{s.i})}, \exists u \in \mathbb{Q}^{\mathsf{oUpdate}(H.\mathsf{group}_{s.i})} \\
&e.i \leq f.i \leq u.i < s.i \wedge e.j < f.j < u.j)) \\
&\wedge (\forall G \in \{H.\mathsf{group}_{s.i}\}, \\
&\forall C \in (\mathcal{MU}_{s.i} \cup \mathcal{CU}_{s.i}) \cap \mathsf{USet}_{G, s.i}, \\
&\exists l \in \mathbb{Q}^{\mathsf{oLeave}(C, G)}, \exists p \in \mathbb{Q}^{\mathsf{oUpdate}(H.\mathsf{group}_{s.i})}, \\
&l.i \leq p.i < s.i \wedge l.j < p.j)
\end{aligned}$$

There are a few differences between our notion and that of [20]:

- Tsudik and Xu use a notation in which the adversary may insert itself at any spot in the participant vector and play the handshake. Instead, in our framework the attacker can choose to use malicious users (one or more) as part of any handshake.
- We formulate the winning conditions of our authentication game in terms of the predicate Win, which can be computed after each execution of the Secret Handshake protocol (following a query to oSHandshake). The predicate will be set to 1 if the adversary has managed to convince at least one user which is a current, legitimate member of a group to accept (and compute a key) with a set of users, at least one of which is not a current, legitimate member of the same group.

D.5 Proofs

For the reader, we define S_i as the event of winning the game $\mathbb{G}_i$.

Full-unlinkability. The sk_U of the victim user U leaked then an attacker $\mathcal{A}$ which has recorded all the transcript can check if $\mathcal{A}$ has already encounter U in some previous sessions by running `DetectBannedUser` by replacing the set KRL by $\{\mathsf{sk}_U\}$; By exploiting the property of Self distinction.

Remark 8. *One may think that Self distinction and Full-unlinkability are incompatible but a protocol with a special one-time credential could meet both of them for example.*

Unlinkability. Let $\mathfrak{P}$ be the protocol described in subsection 5.2 with the following parameters:

Theorem 6. *Suppose there exists an attacker $\mathcal{A}$ against the Unlinkability of* LCA*, which wins with advantage* $\mathsf{Adv}^{\mathsf{Unlink}}_{\mathsf{LCA}}(\mathcal{A})$*. Then, there exist adversaries (called reductions) $\mathcal{R}_1, \mathcal{R}_2$ against respectively, the Unlinkability of* $\mathsf{AGKA\text{-}FR}$ *winning with* $\mathsf{Adv}^{\mathsf{GKA\text{-}Unlink}}_{\mathsf{AGKA\text{-}FR}}(\mathcal{R}_1)$*, and the Unlinkability of* $\mathsf{ListMAC}$ *winning with* $\mathsf{Adv}^{\mathsf{Unlink}}_{\mathsf{ListMAC}}(\mathcal{R}_2)$*,* *such that:*

$$\mathsf{Adv}^{\mathsf{Unlink}}_{\mathsf{LCA}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{GKA\text{-}Unlink}}_{\mathsf{AGKA\text{-}FR}}(\mathcal{R}_1) + \binom{q_{\mathsf{oSHandshake}}}{2} 2^{-|\mathsf{KShare}_{U_b}|} + \mathsf{Adv}^{\mathsf{Unlink}}_{\mathsf{ListMAC}}(\mathcal{R}_2)$$

Proof. [$\mathbb{G}_0$:Orig.] In this original game $\mathbb{G}_0$, the adversary plays with the oracle $\mathsf{oSHandshakeLoR}^b$ trying to guess the value of b. We have the following probability :

$$\mathsf{Adv}^{\mathsf{Unlink}}_{\mathsf{SHS}}(\mathcal{A}) = 2\left|\Pr[\mathsf{S}_0] - \frac{1}{2}\right|$$

[$\mathbb{G}_1$:Bridging] We introduce a game $\mathbb{G}_1$ which have a slight modification compare to $\mathbb{G}_0$; In $\mathbb{G}_1$ the user U_b uses the KShare_{U_L}. We state that the probability is:

$$\Pr[S_0] = \Pr[S_1] \because \mathsf{state}_{U_b} \overset{C.I.}{\equiv} \mathsf{state}_{U_L}$$

Since the value state_{U_b} and state_{U_L} are picked independently uniformly from $\{0,1\}^*$. Hence, $\mathsf{state}_{U_b} \overset{C.I.}{\equiv} \mathsf{state}_{U_L}$ *i.e.,* the probability space doesn't differ from the $\mathbb{G}_0$.

[$\mathbb{G}_2$:Indig.] We introduce Game $\mathbb{G}_2$, which is a slight modification in the oSHandshakeLoR of the previous game $\mathbb{G}_1$. During the AGKA-FR the user U_L is always chosen and shares the state of U_R with U_b:

We claim that the probability is:

$$|\Pr[\mathsf{S}_2] - \Pr[\mathsf{S}_1]| = \mathsf{Adv}^{\mathsf{GKA\text{-}Unlink}}_{\mathsf{AGKA\text{-}FR}}(\mathcal{R})$$

Let $\mathcal{A}$ be an adversary that wins the Unlink for SHS and let $\mathcal{R}$ be an advarsary against Unlink for AGKA-FR. Let C be $\mathcal{R}$'s challenger. We show how $\mathcal{R}$ perfectly simulates the experiment for $\mathcal{A}$ to win its own experiment.

$\mathcal{R}$ picks a game uniformly, either $\mathbb{G}_1$ or either $\mathbb{G}_2$, labeled under the bit $b_{\mathcal{R}}$. C picks randomly $b_C \overset{\$}{\leftarrow} \{\text{“}L\text{”}, \text{“}R\text{”}\}$. Whenever $\mathcal{A}$ sends a tuple $(\mathsf{SHSet}, U_L, U_R)$ to $\mathcal{R}$, the latter forward it to C. C answers by running $\mathsf{AGKA\text{-}FR.oLoRNewSession}^{b_C}$ and returns the transcript $\mathsf{tr}_{\mathsf{AGKA\text{-}FR}}$ to $\mathcal{R}$ if $b_{\mathcal{R}} = \mathbb{G}_1$; otherwise if $b_{\mathcal{R}} = \mathbb{G}_2$, C returns the transcript of the $\mathsf{AGKA\text{-}FR.oLoRNewSession}^L$ played under the $\mathbb{G}_2$ *i.e.,* it runs the AGKA-FR.KA only with U_L.

Since $\mathcal{A}$ is able to win the Unlink for SHS, $\mathcal{A}$ guess $d_{\mathcal{A}}$ which user was played with. Notice the only difference between the two games is that the AGKA-FR.oNewSession is called or not with the U_{b_C} or U_L. Therefore, we state that the probability for $\mathcal{A}$ to win in $\mathbb{G}_1$ *is almost the same as* to win in $\mathbb{G}_2$ except $\mathcal{A}$ couldn't use the information of the AGKA-FR in $\mathbb{G}_2$ but can still use the information leaking of other building blocks.

$$\begin{aligned} |\Pr[\mathsf{S}_2] - \Pr[\mathsf{S}_1]| &= |\Pr[\mathcal{A} \text{ wins } \mathbb{G}_2] - \Pr[\mathcal{A} \text{ wins } \mathbb{G}_1]| \\ &= |\Pr[\mathcal{A} \text{ wins } \mathbb{G}_2] - \Pr[\mathcal{A} \text{ wins } \mathbb{G}_2] - \epsilon| \\ &= \epsilon \end{aligned}$$

[$\mathbb{G}_3$:Indig.] We introduce the following game $\mathbb{G}_3$, where we do not allow anymore the re-use of the same sid during a handshake in oSHandshakeLoR.

We describe how this condition is useful for $\mathcal{A}$, and how it can occur to draw the probability.

$$\begin{aligned} &\forall q \in \mathbb{Q}^{\mathsf{oSHandshakeLoR}}, \\ &\quad (\forall a \in \mathbb{Q}^{\mathsf{oSHandshakeLoR}} \setminus \{q\}, a.\mathsf{tr.sid} \neq q.\mathsf{tr.sid}) \\ &\quad \wedge (\forall b \in \mathbb{Q}^{\mathsf{oSHandshake}}, b.\mathsf{tr.sid} \neq a.\mathsf{tr.sid}) \end{aligned}$$

It means that during the AGKA-FR all users agreed on a value such that it produces the same KShare as in a previous session. The aftermath is that $\mathcal{A}$ can guess which user have been played with, since we have Self distinction. Here's the following steps to guess:

(1) $\mathcal{A}$ call the oracle $\mathsf{oSHandshakeLoR}^b$ with SHSet, U_L, U_R
(2) $\mathcal{A}$ call the oracle oSHandshake with $\mathsf{SHSet} \cup U_L$ and manages to make agreed all on the same KShare
(3) $\mathcal{A}$ having participating in both sessions (the left-or-right one and the usual handshake as explain in item 2, therefore by running the algorithm LM.Match on the tags of the given transcripts. If it matches therefore $\mathcal{A}$ determines that $b = \text{“}L\text{”}$ otherwise $b = \text{“}R\text{”}$

We consider that the probability that $\mathcal{A}$ manages to generate twice the same sid lies on the construction of its generation. First, each user involved in the handshake publish a key share *i.e.,* $\forall U \in \Delta, \mathsf{state}_U$. Then according to a certain ordinal operation they agreed on a concatenation. Since there's at least one honest user in Δ it becomes *almost impossible* to agree on the same sid, but not impossible. The $\mathcal{A}$ can tries to run as many handshakes as possible hopping that the user U_L will generate the same state_{U_L}. The probability to occur is, as described with the differential lemma:

$$|\Pr[\mathsf{S}_3] - \Pr[\mathsf{S}_2]| \leq \binom{q_{\mathsf{oSHandshake}}}{2} 2^{-|\mathsf{state}_{U_b}|}$$

[$\mathbb{G}_4$:Indig.] We introduce a new game $\mathbb{G}_4$ with a slight modification at $\mathbb{G}_4$.oSHandshakeLoR where the tag (τ_U, π_U) is always issued by user U_L *i.e.,* using sk_{U_L}.

Let $\mathcal{A}$ be an adversary that wins the Unlink for SHS and let $\mathcal{R}$ be an adversary against anon for ListMAC. Let C be $\mathcal{R}$'s challenger. We show how $\mathcal{R}$ perfectly simulates the experiment for $\mathcal{A}$ to win its own experiment.

First C toss $b_C \overset{\$}{\leftarrow} \{\text{“}L\text{”}, \text{“}R\text{”}\}$ to decide which user will be used. The $\mathcal{R}$ chooses a bit $b_{\mathcal{R}} \overset{\$}{\leftarrow} \{\mathbb{G}_3, \mathbb{G}_4\}$ to choose which game to simulate. Whenever $\mathcal{A}$ calls the oracle oSHandshakeLoR with SHSet, U_L, U_R, then $\mathcal{R}$ forward the tupple $(U_L, U_R, m, \mathsf{KShare})$ with m choosen by $\mathcal{R}$ in the space of allowed messages for ListMAC and KShare public from the handshake, to C to the oracle oTagLoR.

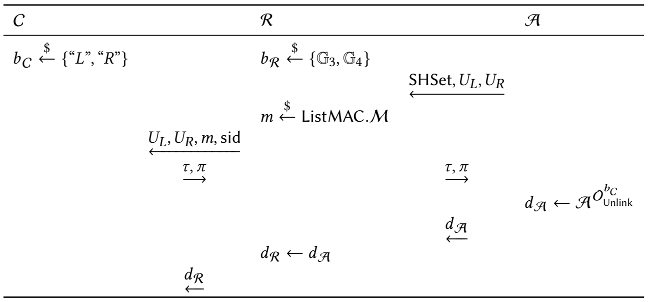

C
$\mathcal{R}$
$\mathcal{A}$
$b_C \xleftarrow{\$} \{“L”, “R”\}$
$b_{\mathcal{R}} \xleftarrow{\$} \{\mathbb{G}_3, \mathbb{G}_4\}$
$\xleftarrow{\mathsf{SHSet}, U_L, U_R}$
$m \xleftarrow{\$} \mathsf{ListMAC}.\mathcal{M}$
$\xleftarrow{U_L, U_R, m, \mathsf{sid}}$
$\xrightarrow{\tau, \pi}$
$\xrightarrow{\tau, \pi}$
$d_{\mathcal{A}} \leftarrow \mathcal{A}^{O^{b_C}_{\mathsf{Unlink}}}$
$\xleftarrow{d_{\mathcal{A}}}$
$d_{\mathcal{R}} \leftarrow d_{\mathcal{A}}$
$\xleftarrow{d_{\mathcal{R}}}$

Figure 23: Simulation during a call to oSHandshakeLoR **with a reduction $\mathcal{R}$ against** $\mathsf{Exp}^{\mathsf{Unlink}}_{\mathsf{ListMAC}}$

Therefore, we obtain:

$$\begin{aligned}|\Pr[\mathsf{S}_4] - \Pr[\mathsf{S}_3]| &= |\Pr[\mathcal{A}\text{ wins }\mathbb{G}_4] - \Pr[\mathcal{A}\text{ wins }\mathbb{G}_3]| \\ &= \mathsf{Adv}^{\mathsf{Unlink}}_{\mathsf{ListMAC}}(\mathcal{R})\end{aligned}$$

[$\mathbb{G}_4$:Final] This $\mathbb{G}_4$ will always have a probability of $\frac{1}{2}$ beacause even if C chooses the the user “L” or “R”, $\mathcal{A}$ cannot conclude since the game is only played with the user “L”.

Therefore the Theorem 6 is true. □

Non-frameability.

Theorem 7. *Suppose there exists an attacker $\mathcal{A}$ against the Non-frameability of* LCA, *which wins with advantage* $\mathsf{Adv}^{\mathsf{NF}}_{\mathsf{LCA}}(\mathcal{A})$*. Then there exist adversaries (called reductions) $\mathcal{R}_1, \mathcal{R}_2$ against respectively, the Non-frameability of* ListMAC *winning with* $\mathsf{Adv}^{\mathsf{Unlink}}_{\mathsf{ListMAC}}(\mathcal{R}_1)$*, such that:*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{NF}}_{\mathsf{LCA}}(\mathcal{A}) \leq &\mathsf{Adv}^{\mathsf{NF}}_{\mathsf{ListMAC}}(\mathcal{R}_1) + \frac{(2q+1)^4}{2^n} \\ &+ \mathsf{Adv}^{\mathsf{2nd-preimg}}_{H} + \mathsf{Adv}^{\mathsf{UCast-auth}}_{\mathsf{CBU2}}\end{aligned}$$

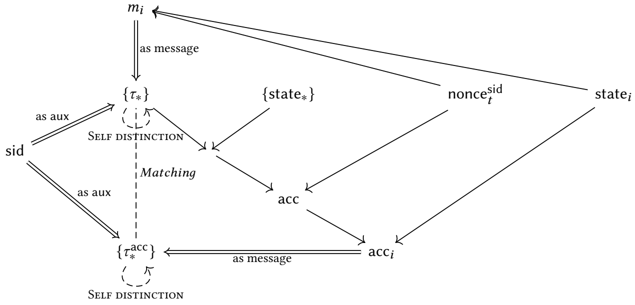


Figure 24: Big picture of interdependencies between all the values in a SHS **secret handshake transcript**

Proof. [$\mathbb{G}_0$:Orig.] In this original game $\mathbb{G}_0$, the adversary $\mathcal{A}$ plays $\mathsf{Exp}^{\mathsf{NF}}_{\mathsf{LCA}}$ trying to forge a valid transcript tr. We have the following probability:

$$\mathsf{Adv}^{\mathsf{NF}}_{\mathsf{LCA}} = \Pr[\mathsf{S}_0]$$

[$\mathbb{G}_1$:Indig.] We introduce a new game S_1 with a slight modification which idealize the ListMAC. Let $\mathcal{A}$ be an adversary that wins the NF for LCA and let $\mathcal{R}$ be an adversary against NF for ListMAC. Let C be $\mathcal{R}$'s challenger. We show how $\mathcal{R}$ perfectly simulates the experiment for $\mathcal{A}$ to win its own experiment. $\mathcal{R}$ chooses a bit $b_{\mathcal{R}} \xleftarrow{\$} \{\mathbb{G}_1, \mathbb{G}_0\}$ to choose which game to simulate. Whenever $\mathcal{A}$ forge a transcript tr, $\mathcal{A}$ forwards it to $\mathcal{R}$. The latter extract the target $(\mathsf{state}_T, \mathsf{nonce}^{\mathsf{sid}}_t, \tau_T, \mathsf{sid}_T)$ from the transcript tr and forwards it to C. Hence, the probability is:

$$|\Pr[\mathsf{S}_1] - \Pr[\mathsf{S}_0]| \leq \mathsf{Adv}^{\mathsf{NF}}_{\mathsf{ListMAC}}(\mathcal{R})$$

[$\mathbb{G}_2$:Failure] We introduce a condition to abort if we detect a collision. If $\mathcal{A}$ return a transcript that contains the same value acc_T as one issued during oSHandshake but with differents tags, therefore we detect a collision-finding. Since we use QROM the obtained probability is:

$$|\Pr[\mathsf{S}_2] - \Pr[\mathsf{S}_1]| \leq \frac{(2q_{\mathsf{H}_{\mathsf{QROM}}} + 1)^4}{2^l}$$

where l is the output size of the hash function H. To give some more explaination, since we are in NF-game, the $\mathcal{A}$ can collude with the GA and therefore forge as many new tags as possible in order to generate a new set of tags such that

$$\mathsf{acc} = H(\mathsf{state}_T||\tau'_1||\mathsf{state}'_1||...||\tau_T||\mathsf{state}_T||...||\tau'_x||\mathsf{state}'_x)$$

For a targeted user T.

[$\mathbb{G}_3$:Failure] We give the same arguments as in $\mathbb{G}_2$, but since the oracle's H evaluation time on a quantum Turing machine may happen to be longer than the classical Turing machine one [4]. Therefore we state the following probability:

$$|\Pr[\mathsf{S}_3] - \Pr[\mathsf{S}_2]| \leq \mathsf{Adv}^{\mathsf{2nd-preimg}}_{H}$$

[$\mathbb{G}_4$:Failure] We introduce the following game $\mathbb{G}_4$, in CBU2.`RecUCast` when `VerID` is inputed with an element that isn't in the *DB*, therefore the game fails. This leads to the straight forward probability:

$$|\Pr[\mathsf{S}_4] - \Pr[\mathsf{S}_3]| \leq \mathsf{Adv}^{\mathsf{UCast-auth}}_{\mathsf{CBU2}}$$

[$\mathbb{G}_4$:Final] From now there's no further possibilities to win the game unless to generate a random value that is valid, we bound this success by ϵ which is negligeable and varies according to λ. therefore Theorem 7 is true. □

Result-hiding. Let $\mathfrak{P}$ be the protocol described in subsection 5.2 with the following parameters:

Theorem 8.

$$\begin{aligned}\mathsf{Adv}^{\mathsf{Res\text{-}Hide}}_{\mathsf{LCA}}(\mathcal{A}) \leq &\mathsf{Adv}^{\textit{Correctness}}_{\mathsf{AGKA\text{-}FR}} + \frac{2}{|\mathsf{sid}|}q^{\mathsf{oSHandshakeLoR}} \\ &+ \mathsf{Adv}^{\mathsf{AKE-security}}_{\mathsf{AGKA\text{-}FR}} + \mathsf{Adv}^{\mathsf{PRF}}_{HKDF} + \mathsf{Adv}^{\mathsf{IND-CPA}}_{\mathsf{SEnc}}\end{aligned}$$

Proof. [$\mathbb{G}_0$:Orig.] In this original game $\mathbb{G}_0$, the adversary plays with the oracle $\mathsf{oSHandshakeLoR}^b$ trying to guess the value of b.

We have the following probability :

$$\mathsf{Adv}^{\mathsf{Res\text{-}Hide}}_{\mathsf{SHS}}(\mathcal{A}) = 2\left|\Pr[\mathrm{S}_0] - \frac{1}{2}\right|$$

[$\mathbb{G}_1$:Indig.] We introduce a new game $\mathbb{G}_1$ that differs slightly from the previous game $\mathbb{G}_0$, where the correctnes's AGKA-FR is idealized.

Let $\mathcal{A}$ be an adversary that wins the Res-Hide for SHS and let $\mathcal{R}$ be an adversary against Correctness for AGKA-FR. Let C be $\mathcal{R}$'s challenger. We show how $\mathcal{R}$ perfectly simulates the experiment for $\mathcal{A}$ to win its own experiment.

According to the game played the Correctness's AGKA-FR is either idealized or not by the C. $\mathcal{R}$ simulates the rest of the SHS.Handshake for $\mathcal{A}$ and conveys it. $\mathcal{A}$ has an advantage to determine if the secret handshake is successfull or not according to Correctness, hence having:

$$|\Pr[\mathbb{G}_1] - \Pr[\mathbb{G}_0]| = \mathsf{Adv}^{\textsc{Correctness}}_{\mathsf{AGKA\text{-}FR}}$$

[$\mathbb{G}_2$:Failure] We introduce a new game $\mathbb{G}_2$ that detects if a collision happens on $\mathsf{sid} = \mathsf{bsn}_{U_R} \vee \mathsf{sid} = \mathsf{bsn}_{U_R}$. Since the reduction is sumalating the SHS.Handshake it detectes wether or not this event has happened.

$$|\Pr[\mathbb{G}_2] - \Pr[\mathbb{G}_1]| \leq \frac{2}{|\mathsf{sid}|}(q^{\mathsf{oSHandshakeLoR}})$$

[$\mathbb{G}_3$:Indig.] Suppose $\mathcal{A}$ an adversary winning against Result-hiding for SHS.

Let $\mathcal{R}$ a reduction playing against AGKA-FR $\mathsf{AKE-security}$, and C be $\mathcal{R}$'s challenger. We show how $\mathcal{R}$ perfectly simulates the experiment for $\mathcal{A}$ to win its own experiment.

Let $\mathcal{A}$ asks $\mathcal{R}$ for the triplet $(\mathsf{SHSet}, U_L, U_R)$. Then $\mathcal{R}$ asks C a handshake and C replies with $\mathsf{ms}_{\mathsf{AGKA\text{-}FR}}$ rela or random. $\mathcal{R}$ derives now $\mathsf{ms}_{\mathsf{AGKA\text{-}FR}}$ according to LCA. $\mathcal{R}$ appends to the AGKA-FR's transcript the rest of the simulated secret handshake. $\mathcal{A}$ determines if the handshake is successfull or not, and sends its reply to $\mathcal{R}$. If the handshake is successfull therefore it returns that the $\mathsf{ms}_{\mathsf{AGKA\text{-}FR}}$ was real, otherwise random.

$$|\Pr[\mathbb{G}_3] - \Pr[\mathbb{G}_2]| = \mathsf{Adv}^{\mathsf{AKE-security}}_{\mathsf{AGKA\text{-}FR}}$$

[$\mathbb{G}_4$:Indig.] We introduce a new game $\mathbb{G}_4$ that slightly differs from $\mathbb{G}_3$ by idealizing the HKDF function with a Random Oracle.

Let $\mathcal{A}$ be an adversary that wins the Res-Hide for SHS and let $\mathcal{R}$ be an adversary against PRF for HKDF. Let C be $\mathcal{R}$'s challenger. We show how $\mathcal{R}$ perfectly simulates the experiment for $\mathcal{A}$ to win its own experiment.

Let $\mathcal{A}$ asks $\mathcal{R}$ for the triplet $(\mathsf{SHSet}, U_L, U_R)$. Then $\mathcal{R}$ simulates AGKA-FR and for the corresponding $\mathsf{ms}_{\mathsf{AGKA\text{-}FR}}$ asks to C accordingly to the `KeySchedule` function. In $\mathbb{G}_4$ C replies with a truly random value *via* the Random Oracle, while in $\mathbb{G}_3$ C replies with a value computed from the HKDF function. $\mathcal{R}$ simulates the rest, and sends it to $\mathcal{A}$. If $\mathcal{A}$ is able to determine that the handshake it means that the HKDF wasn't enough pseudo random and manage to do some hypothesis.

$$|\Pr[\mathbb{G}_4] - \Pr[\mathbb{G}_3]| = \mathsf{Adv}^{\mathsf{PRF}}_{HKDF}$$

[$\mathbb{G}_5$:Indig.] We introduce a new game $\mathbb{G}_5$ that slightly differs from $\mathbb{G}_4$ by replacing the output of SEnc by uniformly random values of the same size.

Let $\mathcal{A}$ be an adversary that wins the Res-Hide for SHS and let $\mathcal{R}$ be an adversary against $\mathsf{IND-CPA}$ for SEnc. Let C be $\mathcal{R}$'s challenger. We show how $\mathcal{R}$ perfectly simulates the experiment for $\mathcal{A}$ to win its own experiment.

Using the same structure as before, the $\mathcal{R}$ asks to C, according to the game, C returns either SEnc or random values. $\mathcal{R}$ simulates the rest of the SHS.Handshake, and conveys it to $\mathcal{A}$. Hence the probability is as follows:

$$|\Pr[\mathbb{G}_5] - \Pr[\mathbb{G}_4]| = \mathsf{Adv}^{\mathsf{IND-CPA}}_{\mathsf{SEnc}}$$

[$\mathbb{G}_5$:Final] Since all the parts have been idealized it is not possible anymore for the $\mathcal{A}$ to win. Therefore we conclude that Theorem 8 is true.

□

Self distinction.

Theorem 9. *Suppose there exists an attacker $\mathcal{A}$ against the Self distinction of* LCA*, which wins with advantage* $\mathsf{Adv}^{\mathsf{S\text{-}Dist}}_{\mathsf{LCA}}(\mathcal{A})$*. Then there exist adversaries (called reductions) $\mathcal{R}_1, \mathcal{R}_2$ against respectively, the Non-frameability of* ListMAC *winning with* $\mathsf{Adv}^{\mathsf{NF}}_{\mathsf{ListMAC}}(\mathcal{R}_1)$*, and the* $\mathsf{EUF-CMA-AD}$ *of* ListMAC *winning* $\mathsf{Adv}^{\mathsf{EUF-CMA-AD}}_{\mathsf{ListMAC}}(\mathcal{R}_2)$*, such that:*

$$\mathsf{Adv}^{\mathsf{S\text{-}Dist}}_{\mathsf{LCA}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{NF}}_{\mathsf{ListMAC}}(\mathcal{R}_1) + \mathsf{Adv}^{\mathsf{EUF-CMA-AD}}_{\mathsf{ListMAC}}(\mathcal{R}_2)$$

Proof. [$\mathbb{G}_0$:Orig.] In this original game $\mathbb{G}_0$, the adversary $\mathcal{A}$ plays $\mathsf{Exp}^{\mathsf{S\text{-}Dist}}_{\mathsf{LCA}}$ trying to forge a valid transcript tr. We have the following probability:

$$\mathsf{Adv}^{\mathsf{S\text{-}Dist}}_{\mathsf{LCA}} = \frac{1}{1-\epsilon}\,|\Pr[\mathrm{S}_0] - \epsilon|$$

[$\mathbb{G}_1$:Indig.] We introduce a new game S_1 with a slight modification which idealize the ListMAC. Let $\mathcal{A}$ be an adversary that wins the S-Dist for LCA and let $\mathcal{R}$ be an adversary against NF for ListMAC. Let C be $\mathcal{R}$'s challenger. We show how $\mathcal{R}$ perfectly simulates the experiment for $\mathcal{A}$ to win its own experiment.

$\mathcal{R}$ chooses a bit $b_{\mathcal{R}} \overset{\$}{\leftarrow} \{\mathbb{G}_1, \mathbb{G}_0\}$ to choose which game to simulate. Whenever $\mathcal{A}$ forge a transcript tr, $\mathcal{A}$ forwards it to $\mathcal{R}$. The latter extract the targets that contains the two tags issued from the sk_U but that doesn't match and forwards them to C. Hence, the probability is:

$$|\Pr[\mathrm{S}_1] - \Pr[\mathrm{S}_0]| \leq \mathsf{Adv}^{\mathsf{NF}}_{\mathsf{ListMAC}}(\mathcal{R})$$

[$\mathbb{G}_2$:Indig.] We introduce a new game S_2 with a slight modification which idealize the ListMAC. Let $\mathcal{A}$ be an adversary that wins the S-Dist for LCA and let $\mathcal{R}$ be an adversary against S-Dist for ListMAC. Let C be $\mathcal{R}$'s challenger. We show how $\mathcal{R}$ perfectly simulates the experiment for $\mathcal{A}$ to win its own experiment.

$\mathcal{R}$ chooses a bit $b_{\mathcal{R}} \overset{\$}{\leftarrow} \{\mathbb{G}_2, \mathbb{G}_1\}$ to choose which game to simulate. Whenever $\mathcal{A}$ forge a transcript tr, $\mathcal{A}$ forwards it to $\mathcal{R}$. The latter extract the target that contains the a tags that corresponds to none of the issued sk nor KRL and forwards them to C. Hence, the probability is:

$$|\Pr[\mathsf{S}_2] - \Pr[\mathsf{S}_1]| \leq \mathsf{Adv}^{\mathsf{EUF-CMA-AD}}_{\mathsf{ListMAC}}(\mathcal{R})$$

[$\mathbb{G}_2$:Final] From now there's no further possibilities to win the game unless to generate a random value that is valid, we bound this success by ϵ which is negligeable and varies according to λ. therefore Theorem 9 is true. □

Traitor Catching.

Theorem 10. *Suppose there exists an attacker $\mathcal{A}$ against the* Traitor Catching *of* LCA*, which wins with advantage* $\mathsf{Adv}^{\mathsf{Catch}}_{\mathsf{LCA}}(\mathcal{A})$*. Then there exist adversaries (called reductions)* $\mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_3$ *against respectively, the* Non-frameability *of* $\mathsf{ListMAC}$ *winning with* $\mathsf{Adv}^{\mathsf{Unlink}}_{\mathsf{ListMAC}}(\mathcal{R}_1)$*, the* $\mathsf{EUF-CMA-AD}$ *with* $\mathsf{Adv}^{\mathsf{EUF-CMA-AD}}_{\mathsf{ListMAC}}(\mathcal{R}_2)$*, and the useruni-cast authentication with* $\mathsf{Adv}^{\mathsf{UCast\text{-}Auth}}_{\mathsf{CBU2}}(\mathcal{R}_3)$*, such that:*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{Catch}}_{\mathsf{LCA}}(\mathcal{A}) \leq &\mathsf{Adv}^{\mathsf{NF}}_{\mathsf{ListMAC}}(\mathcal{R}_1) + \mathsf{Adv}^{\mathsf{EUF-CMA-AD}}_{\mathsf{ListMAC}}(\mathcal{R}_2)\\ &+ \mathsf{Adv}^{\mathsf{UCast\text{-}Auth}}_{\mathsf{CBU2}}(\mathcal{R}_3)\end{aligned}$$

[$\mathbb{G}_0$:Orig.] In this original game $\mathbb{G}_0$, the adversary $\mathcal{A}$ plays $\mathsf{Exp}^{\mathsf{Catch}}_{\mathsf{LCA}}$ trying to forge a valid transcript tr. We have the following probability:

$$\mathsf{Adv}^{\mathsf{Catch}}_{\mathsf{LCA}} = \frac{1}{1-\epsilon}\,|\Pr[\mathsf{S}_0] - \epsilon|$$

[$\mathbb{G}_1$:Indig.] We introduce a new game S_1 with a slight modification which idealize the $\mathsf{ListMAC}$. Let $\mathcal{A}$ be an adversary that wins the Catch for LCA and let $\mathcal{R}$ be an adversary against NF for $\mathsf{ListMAC}$. Let C be $\mathcal{R}$'s challenger. We show how $\mathcal{R}$ perfectly simulates the experiment for $\mathcal{A}$ to win its own experiment. $\mathcal{R}$ chooses a bit $b_{\mathcal{R}} \xleftarrow{\$} \{\mathbb{G}_1, \mathbb{G}_0\}$ to choose which game to simulate. Whenever $\mathcal{A}$ forge a transcript tr, $\mathcal{A}$ forwards it to $\mathcal{R}$. The latter extract the target tag from the transcript transcript $\mathsf{SHS.Ban}$ and forwards it to C. Hence, the probability is:

$$|\Pr[\mathsf{S}_1] - \Pr[\mathsf{S}_0]| \leq \mathsf{Adv}^{\mathsf{NF}}_{\mathsf{ListMAC}}(\mathcal{R})$$

[$\mathbb{G}_2$:Indig.] We introduce a new game S_2 with a slight modification which idealize the $\mathsf{ListMAC}$. Let $\mathcal{A}$ be an adversary that wins the S-Dist for LCA and let $\mathcal{R}$ be an adversary against S-Dist for $\mathsf{ListMAC}$. Let C be $\mathcal{R}$'s challenger. We show how $\mathcal{R}$ perfectly simulates the experiment for $\mathcal{A}$ to win its own experiment.

$\mathcal{R}$ chooses a bit $b_{\mathcal{R}} \xleftarrow{\$} \{\mathbb{G}_2, \mathbb{G}_1\}$ to choose which game to simulate. Whenever $\mathcal{A}$ forge a transcript tr, $\mathcal{A}$ forwards it to $\mathcal{R}$. The latter extract the target that contains the a tags that corresponds to none of the issued sk nor KRL and forwards them to C. Hence, the probability is:

$$|\Pr[\mathsf{S}_2] - \Pr[\mathsf{S}_1]| \leq \mathsf{Adv}^{\mathsf{EUF-CMA-AD}}_{\mathsf{ListMAC}}(\mathcal{R})$$

[$\mathbb{G}_3$:Indig.] Here we present an attack that forces to idealize the $\mathsf{CBU2}$. In this new game $\mathbb{G}_3$ the modification is to idealize $\mathsf{CBU2.UCast}$ and $\mathsf{CBU2.RecUCast}$ present in $\mathsf{SHS.Join}$ algorithm. So the $\mathcal{A}$ is able to register its τ^{id} under another identity. Therefore $\mathcal{A}$ can still corrupt or register a valid $\mathsf{ListMAC}$. Following this attacks this breaks the $\mathsf{CBU2}$'s user unicast authentication. Hence giving the following probability:

$$|\Pr[\mathsf{S}_3] - \Pr[\mathsf{S}_2]| \leq \mathsf{Adv}^{\mathsf{UCast\text{-}Auth}}_{\mathsf{CBU2}}(\mathcal{R})$$

[$\mathbb{G}_3$:Final] From now there's no further possibilities to win the game. therefore Theorem 10 is true.

E MITIGATION FOR DATA MINIMIZATION

The principle of Data Minimization means that a data controller should limit the collection of personal information to what "is directly relevant and necessary to accomplish a specified purpose" [18, 19]. As mentioned earlier, previous model used the paradigm of Traitor Tracing in which, given a transcript, the group manager trace the user that participated in it to afterward ban the concerned misbehaving/politically incorrect user. This gives to the group manager too much information such as the identities of each concerned secret handshake's participants.

At our known, compare to the current literature we state that our protocol achieves the Data Minimization – keep in mind that the definition isn't very formal and stands only for legal reasons – with our new paradigm, Traitor Catching. The data that are transmitted to the group manager are : (1) an excerpt of the conversation[16] (it can implicitly draw the relation between the participants *e.g.*, who is a leader, *etc*) but even though it avoids judgement based on out-of-context; (2) the identity of the banned user if the banishment is done, relies on the Traitor Catching.

On the other hand, the information that are kept secret to the users are: (1) the number of participants, *except of what is mentioned in messages* since the users have the possibility to trim the transcript; (2) the identity of users, *except the banned one*; (3) the identity of the plaintiff (without a loss of generality we state that Obliviousness – see definition 32 – couldn't be respected in a real world; even though we propose a method to mitigate it[17]).

Definition 32 (Obliviousness). *Based on network information such as IP address, size packages, time, speed, etc, a monitor can deduce interaction between users and some information to them e.g., someone using the same IP could be deanonymized, an IP receiving many connections could be a server.*

To sum up, the breakthrough in privacy that we gain with our new paradigm – Traitor Catching – is that only the concerned misbehaved user identity is revealed to the group manager *i.e.,* the identities of the participants, except the misbehaved one, in the examined handshake is kept unknown to the group manager. To do so the group manager deploys a 'trap', and only the targeted user will not be able to pass it. The 'trap' consists of a commit-challenge-response. First a complaining user publish the transcript which is seen as a commit, then the group manager instantiate a trap which is seen as a challenge, and users tries to go through to obtain a new nonce by sending their responses. We discuss a lot more at subsection D.3.

Messaging. To avoid any malleability in our messaging *i.e.,* Non-frameability, we propose to sign as many messages as possible.

[16] Participants are producing a conversation that is chained with the possibility to extract blocks of conversation for further reading see subsection 5.2.

[17] We propose a contamination mechanism where the plaintiff at the next successful handshake for the same group will send the complaint (ciphered) to other users

This could have harmful performances in terms of size and operation. Therefore one of the simplest solution is to chain our messages (as in a "naive" blockchain) and sign them at a specific period T_r, refer to Figure 25a and Figure 25b. We notice that the (β) chain follows a KDF Chain described in [?], but it isn't used as an update of the session key. In other words the (β) chain isn't used for confidentiality but only for integrity and authenticity purposes.

$$\begin{tikzcd}
\beta_i \arrow[rr, "\mathsf{Extract}(\mathsf{Expand}(\beta_i, m_{i+1}))"] & {} & \beta_{i+1} \\
& m_{i+1} \arrow[u] &
\end{tikzcd}$$

(a) Chaining system $\forall i, i \not\equiv 0 \pmod{T_r}$

$$\begin{tikzcd}
\beta_i \arrow[rr, "\mathsf{Extract}(\mathsf{Expand}(\beta_i, m_{i+1}))"] & {} & \beta_{i+1} \arrow[rr, "\mathsf{ListMAC.Tag}(\mathsf{sk}_U, \beta_{i+1} || m_{i+1})"] & {} & \sigma_{i+1} \\
& m_{i+1} \arrow[u] \arrow[rr] & & {} \arrow[u] &
\end{tikzcd}$$

(b) Chaining system $\forall i, i \equiv 0 \pmod{T_r}$

Figure 25: Chaining system for messages

Therefore for every $\forall i, i \not\equiv 0 \pmod{T_r}$ the user sends $\beta_{i+1} || m_{i+1}$. And for every $\forall i, i \equiv 0 \pmod{T_r}$ the user sends $\beta_{i+1} || \sigma_{i+1} || m_{i+1}$.

Note, that there is a privacy concern with the length of each message. The trade-off here is the optimization. If we want to sign every message therefore it is still interesting to keep the chain to avoid any $\mathcal{A}$ to manipulate the order of the messages at his will. Even though one of the countermeasure is to pad messages of the same length.

In case of a desynchronization, here the chaining is "naive" and doesn't need to be bothered by others users. Therefore, if a desynchronization happened, for example the β_i doesn't correspond, then a resynchronization is run. The interlocutor sends the last β_i and then the user sends back the rest of messages $\{m_k | \forall k > i\}$ and sign them.

`trim`**.** This algorithm helps to build a correct complaint without revealing the number of participants nor the whole conversation *etc.* We notice that the transcript is a set of tags that are almost independent to each others; in fact this allows us to cherry-pick only the tag of the misbehaved user. Also in the same scope, by the helps of messaging, we can also cherry-pick a block of the conversation, in respect of the T_r. Still note that `trim` isn't designed to modify the content of messages, this can leak some information such as the number of participants or some identities.

Contamination. As mentioned before for a banishment, due to the identification used in CBU2.Send it is easy for the GA to trace back the plaintiff. Therefore, we propose a contamination methodology to reduce the risks for the plaintiff. One of the possibility that doesn't involve any cryptographic parts are based on offer & demand market:

(1) everyone has to send one packet to the nonce+ 1 (therefore the transmission isn't exponential)
(2) each time a user rerandomize it or replace it with its complain;
(3) the package received and send it to someone of the group whose nonce is the closest of the obtained hash of the complaint;
(4) this user has the choice to report it or contaminate the next person in another handshake.

Therefore, the market regulates itself, between those who are complaining, those whose transmit and those whose report it. This solution prevents from over flooding the network by keeping the number of packages constant. Even though seeing its complaint unsatisfied the plaintiff could renew. The numbers of instance can reveal something on the seniority of the user. Since then the package is replicated many times over the network.

To sum up, during a period of high complaining, less will reach the GA; in low complaining many will reach the GA *i.e.,* regulation as in free market. If one user is flooding its packages repeatedly therefore its complain has a better chance to reach the GA.

It needs Enc, Dec, Rerand with $\mathsf{pk}_{\mathsf{GA}}$. Plus it respects Obliviousness inside of a group it prevents the GA to reveal the plaintiff (could be a potential whistleblower).