A Flexible Framework for Secret Handshakes

or: How to Achieve Multi-Party Interactive Anonymous Mutual Authentication

Gene Tsudik (Contact Author)[1] and Shouhuai Xu[2]

[1] Department of Computer Science
University of California, Irvine.
gts@ics.uci.edu

[2] Department of Computer Science
University of Texas, San Antonio.
shxu@cs.utsa.edu

Abstract. In the society increasingly concerned with the erosion of privacy, privacy-preserving techniques are becoming very important. Secret handshakes offer anonymous and unobservable authentication and serve as an important tool in the arsenal of privacy-preserving techniques. Relevant prior research focused on 2-party secret handshakes with one-time credentials, whereby two parties establish a secure, anonymous and unobservable communication channel only if they are members of the same group.

This paper breaks new ground on two accounts: (1) it shows how to obtain secure and efficient secret handshakes with reusable credentials, and (2) it provides the first treatment of *multi-party* secret handshakes, whereby $m \geq 2$ parties establish a secure, anonymous and unobservable communication channel if they all belong to the same group. An interesting new issue encountered in multi-party secret handshakes is the need to ensure that all parties are indeed distinct. (This is a real challenge since the parties cannot expose their identities.) We tackle this and other challenging issues in constructing **GCD** – a flexible secret handshake framework. **GCD** can be viewed as a "compiler" that transforms three main building blocks (a **G**roup signature scheme, a **C**entralized group key distribution scheme, and a **D**istributed group key agreement scheme) into a secure multi-party secret handshake scheme.

The proposed framework lends itself to multiple practical instantiations, and offers several novel and appealing features such as self-distinction and strong anonymity with *reusable* credentials. In addition to describing the motivation and step-by-step construction of the framework, this paper provides a security analysis and illustrates several concrete framework instantiations.

Keywords: secret handshakes, privacy-preservation, anonymity, credential systems, unobservability, key management.

Contact Author: Gene Tsudik, CS Dept, University of California, Irvine, CA 92697-3425. Phone: (949)824-3410. Fax: (949)824-4056. Email: *gts@ics.uci.edu*

NOTE: If not selected for regular presentation, please consider this submission for a brief announcement.

1 Introduction

Much of today's communication is conducted over public networks which naturally prompts a number of concerns about security and privacy. Communication security has been studied extensively and a number of effective and efficient security tools and techniques are available.

Unfortunately, privacy concerns have not been addressed to the same extent. Yet, it is quite obvious to anyone who keeps up with the news that our society is very concerned with privacy. At the same time, privacy is being eroded by (often legitimate) concerns about crime, terrorism and other malfeasances. Furthermore, the proliferation of wireless communication (among laptops, cell phones, PDAs, sensors and RFIDs) drastically lowers the bar for eavesdropping and tracking of both people and their devices.

Popular techniques to provide communication privacy include email MIX-es, anonymizing routers and proxy web servers as well as purely cryptographic tools, such as private information retrieval. Despite important advances, the privacy continuum has not been fully explored. One particular issue that has not been widely recognized is the need for unobservable, untraceable and anonymous authentication, i.e., **privacy-preserving authentication**. Such a notion might seem counter-intuitive at first, since authentication traditionally goes hand-in-hand with identification. However, in the context of groups or roles, authentication identifies not a distinct entity but a collection thereof. To this end, some advanced cryptographic techniques has been developed, such as group signatures [2] and privacy-preserving trust negotiation [50, 37].

We focus on *interactive privacy-preserving mutual authentication*; more specifically, on *secret handshakes*. A secret handshake scheme (SHS) allows two or more group members to authenticate each other in an anonymous, unlinkable and unobservable manner such that one's membership is not revealed unless every other party's membership is also ensured.[1]

In more detail, a secure handshake allows members of the same group to identify each other *secretly*, such that each party reveals its affiliation to others if only if the latter are also group members. For example, in a 2-party setting, an FBI agent (Alice) wants to authenticate to Bob only if Bob is also an FBI agent. Moreover, if Bob is *not* an FBI agent, he should be unable to determine whether Alice is one (and vice versa). This property can be further extended to ensure that group members' affiliations are revealed only to members who hold specific *roles* in the group. For example, Alice might want to authenticate herself as an agent with a certain clearance level *only if* Bob is also an agent with at least the same clearance level.

In a more general sense, secret handshakes offer a means for privacy-preserving mutual authentication with many possible applications, especially, in hostile environments.

Goals: We set out to develop techniques for supporting efficient **multi-party** secret handshakes while avoiding certain drawbacks present in some or all of the previous **2-party** secret handshake solutions. These drawbacks include: (1) use of one-time credentials or pseudonyms, (2) ability of the group authority to cheat users, (3) requirement to maintain information about many irrelevant groups (groups that one is not member of), and (4) lack of support for handshakes of three or more parties. Some of these drawbacks are self-explanatory, while others are clarified later in the paper.

1.1 Overview and Summary of Contributions

We are interested in multi-party secret handshakes, whereby $m \geq 2$ parties establish a secure, anonymous and unobservable communication channel provided that they are members of the same group. We achieve this by constructing a secret handshake framework called **GCD**. This framework is essentially a *compiler* that transforms three main ingredients – a **G**roup signature scheme, a **C**entralized group key distribution scheme, and a **D**istributed group key agreement scheme – into a secure secret handshake scheme. We formally specify this framework based on desired functionality and security properties.

From the functionality perspective, existing solutions are only able to support 2-party secret handshakes [4], [20, 49]. Our framework represnts the first result that supports truly *multi-party* secret hand-

[1] This informal definition broadens the prior version [4] which limited secret handshakes to two parties.

shakes. Moreover, our work is also the first to address the problem of *partially-successful* secret handshakes; this occurs if not all parties engaged in a handshake protocol are members of the same group.[2]

From the security perspective, our framework has two novel features. First, it can be resolved into concrete schemes that provide the novel and important `self-distinction` property which ensures the uniqueness of each handshake participant. In other words, it guarantees that the protocol is a multi-party computation with the exact number of players that claim to be participating. Without `self-distinction`, a malicious insider may arbitrarily impersonate any number of group members by simultaneously playing multiple roles in a handshake protocol.[3]

Second, in contrast with prior work [4, 20] which relies on one-time credentials to achieve `unlinkability` – which ensures that multiple handshake sessions involving the same participant(s) cannot be linked by an adversary – our approach provides `unlinkability` with multi-show (or reusable) credentials. This greatly enhances its usability. Moreover, our approach does not require users to be aware of other groups, in contrast with [49].

In addition, our framework has some interesting *flexibility* features. In particular, it is model-agnostic: if the underlying building blocks operate in the standard cryptographic model – rather than in the Random Oracle Model (ROM) – so does the resulting secret handshake scheme. Similarly, if the building blocks operate in the asynchronous communication model (with guaranteed delivery), so does the resulting secret handshake scheme. Also, it supports a set of selectable properties that can be tailored to application needs and semantics. Finally, it lends itself to many practical instantiations: we present three concrete examples where a handshake participant computes only $O(m)$ modular exponentiations and sends/receives $O(m)$ messages, where m is the number of handshake participants.

Organization: Section 2 presents our system model and definitions of secret handshake schemes. Then we proceed to discuss the design space and lay the foundation for the framework in Section 3 The models and definitions for the three building blocks are discussed in Sections 4, 5, and 6. Next, Section 7 presents the actual **GCD** framework and the analysis of its properties, followed by three concrete instantiations in Section 8. Related work is overviewed in Section 9.

Appendix: Due to space restrictions, some material has been placed in the Appendix. Sections A, B, C and D describe the formal security properties of, respectively: (1) secret handshake schemes, (2) group signature schemes, (3) group key distribution schemes, and (4) distributed group key agreement schemes. Sections E and F provide the proofs of security for the framework and one of its instantiations. Section G overviews a specific group signature scheme used for one of the instantiations and Section H discusses certain practical issues.

2 Secret Handshakes: Model and Definition

Let κ be a security parameter and $\mathcal{U}$ be a set of all users: $\mathcal{U} = \{U_i \mid 0 < i < n\}$ where n is bounded by $poly(\kappa)$. Let ℓ be an integer, and $\mathbf{G}$ be a set of all groups $\mathbf{G} = \{G_1, \ldots, G_\ell\}$ where each group[4] $G \in \mathbf{G}$ is a set of members managed by a *group authority* $\mathcal{GA}$, which is responsible for admitting members, revoking their membership and updating system state information. For simplicity's sake we assume that each user is a member of exactly one group. (Of course, all results can be easily generalized to the case that users are allowed to join multiple groups.) An adversary $\mathcal{A}$ is allowed to corrupt various participants. All participants (including $\mathcal{A}$) are modeled as probabilistic polynomial-time algorithms.

We assume the existence of *anonymous* channels between all the legitimates participants, where the term "anonymous" means that an outside attacker cannot determine identities of the $\mathcal{GA}$, group members, as well as the dynamics and size of a group, and that a malicious insider cannot determine the identities of other honest group members as well as the the dynamics and size of the group. This assumption is necessary

[2] For example, if 5 parties initiate a 5-way secret handshake protocol and 2 of them are members of group A, while the rest are members of group B, the desired outcome is for both the former and the latter to complete the secret handshake protocol and determine that their respective handshakes were performed with 2 and 3 members, respectively.

[3] This is reminiscent of the well-known Sybil attack [26].

[4] We use "group" to refer to a set of users, unless explicitly stated otherwise.

in any privacy-preserving authentication scheme (otherwise, anonymity is trivially compromised); we discuss practical issues in Appendix H.

A secret handshake scheme (SHS) consists of the algorithms and protocols shown in Figure 1:

Fig. 1. Definition of Multi-Party Secret Handshake Schemes

SHS.CreateGroup: executed by $\mathcal{GA}$ to establish a group G. It takes as input appropriate security parameters, and outputs a cryptographic context specific to this group. The context may include a certificate/membership revocation list, $\mathcal{CRL}$, which is originally empty. The cryptographic context is made public, while the $\mathcal{CRL}$ is made known only to current group members.

SHS.AdmitMember: executed by $\mathcal{GA}$ to admit a user to the group under its jurisdiction. We assume that $\mathcal{GA}$ admits members according to a certain admission policy. Specification and enforcement of such policy is out the scope of this paper. After executing the algorithm, group state information has been appropriately updated, the new member holds some secret(s) as well as a membership certificate(s), and existing members obtain updated system information from $\mathcal{GA}$ via the aforementioned authenticated anonymous channel.

SHS.RemoveUser: executed by $\mathcal{GA}$. It takes as input the current $\mathcal{CRL}$ and a user identity U_i such that $U_i \in \mathcal{U}$ and $U_i \in G$. The output includes an updated $\mathcal{CRL}$ which includes the newly revoked certificate for U_i. The state update information is sent to the existing group members through the authenticated anonymous channel.

SHS.Update: executed by each current group member upon receiving, via the authenticated anonymous channel, system state update information from $\mathcal{GA}$. It is used to update each member's system state information.

SHS.Handshake(Δ): executed by a set Δ of m users purporting to be members of a group G, where $\Delta = \{U_1, \ldots, U_m\}$ and $m \geq 2$. The input to this protocol includes the secrets of all users in Δ, and possibly some public information regarding the current state of the systems. At the end of a protocol execution, it is ensured that each $U_i \in \Delta$ determines that $\Delta \setminus \{U_i\} \subseteq G$ if and only if each $U_j \in \Delta$ $(j \neq i)$ discovers $\Delta \setminus \{U_j\} \subseteq G$.

SHS.TraceUser: executed by $\mathcal{GA}$. On input of a transcript of a successful secret handshake protocol SHS.Handshake(Δ), $\mathcal{GA}$ outputs the identities of all m participants involved in the handshake, i.e., $U_1, ..., U_m$.

We note that the above definition says nothing about the participants establishing a common key following (or during) a successful handshake. It is indeed straightforward to establish such a key if a secret handshake succeeds. However, allowing further communication based on a newly established key would require concealing the outcome of the handshake. (See also Appendix H for further discussion).

The above definition also does not ensure any form of "agreement" in the sense of [28], since the adversary is assumed to have complete control over all communication, and can allow corrupt parties.

Desired security properties are informally specified in Figure 2. Due to space limitations, the formal treatment is deferred to Appendix A.

Fig. 2. Informal Security for Multi-Party Secret Handshake Schemes

* `Correctness`: If all handshake participants $\{U_1, \ldots, U_m\}$ belong to the same group, the protocol returns "1"; otherwise, the protocol returns "0".
* `Resistance to impersonation`: a *passive* adversary $\mathcal{A} \notin G$ who does not corrupt any members of G has only a negligible probability in convincing an honest user $U \in G$ that $\mathcal{A} \in G$.
* `Resistance to detection`: no adversary $\mathcal{A} \notin G$ can distinguish between an interaction with an honest user $U \in G$ and an interaction with a simulator.
* `Unlinkability`: no adversary $\mathcal{A}$ is able to associate two handshakes involving a same honest user $U \in G$, even if $\mathcal{A} \in G$ and $\mathcal{A}$ participated in both executions.
* `Indistinguishability to eavesdroppers`: no adversary $\mathcal{A}$ who does not participate in a handshake protocol can distinguish between a successful handshake between $\{U_1, \ldots, U_m\} \subseteq G$ and an unsuccessful one, even if $\mathcal{A} \in G$.
* `Traceability`: $\mathcal{GA}$ can trace all users involved in a given handshake session, as long as at least one of them is honest.
* `No-misattribution`: no coalition of malicious parties (including any number of group members and the $\mathcal{GA}$) is able to frame an honest member as being involved in a secret handshake.
* `Forward-repudiability`: Suppose at some time t_1, a set of honest users in $\Delta = \{U_1, \ldots, U_m\}$ engage in a handshake. At time $t_2 > t_1$, it is impossible for any $U_i \in \Delta$ to prove to a third party that any $U \in \Delta \setminus \{U_i\}$ participated in the handshake session at time t_1, even if U_i reveals its own secrets.
* `Self-distinction`: each participant is ensured that all the participants are distinct.

3 Design Space

As mentioned earlier, the SHS framework is essentially a compiler that outputs a multi-party secret handshake scheme satisfying all definitions in Section 2. Its input includes:

– A group signature scheme (GSIG): a scheme that allows any group member to produce signatures on behalf of the group in an anonymous and unlinkable manner; only a special entity (called a group manager) is able to revoke anonymity and "open" a group signature thereby revealing the signer's identity. (See Section 4.)
– A centralized group key distribution scheme (CGKD): a key management scheme for large one-to-many groups that handles key changes due to dynamic group membership and facilitates secure broadcast encryprion. (See Section 5.)
– A distributed group key agreement scheme (DGKA): a scheme that allows a group of peer entities to dynamically (on-the-fly) agree on a common secret key to be used for subsequent secure communication within that group. (See Section 6.)

We now discuss the choices made in designing **GCD**.

As a first try, one might be tempted to construct a secret handshake scheme directly upon a CGKD that enables secure multicast. It is easy to see that $m \geq 2$ members can conduct efficient secret handshakes based on a group key k. However, this approach would have some significant drawbacks:

(1) No indistinguishability-to-eavesdroppers. A *passive* malicious (honest-but-curious) group member can detect, by simply eavesdropping, whenever other members are conducting a secret handshake.
(2) No traceability. A dishonest member who takes part in a handshake (or is otherwise malicious) can not be traced and held accountable.
(3) No self-distinction. For handshakes of more than two parties, self-distinction is not attained since a rogue member can play multiple roles in a handshake.

Alternatively, one could employ a GSIG scheme as a basis for a secret handshake scheme. This would avoid the above drawback (2), however, drawback (1) remains. Also, resistance to detection attacks would be sacrificed, since (as noted in [4]), group signatures are verifiable by anyone in possession of the group public key.

A natural next step is to combine a CGKD with a GSIG. This way, the GSIG group public key is kept secret among all current group members (along with the CGKD group-wide secret key k), and – during the handshake – group signatures would be encrypted under the group-wide key k. Although traceability would be re-gained, unfortunately, drawbacks (1) and (3) would remain.

In order to avoid (1), we need the third component, an interactive distributed key agreement protocol. With it, any member who wants to determine if other parties are members (or are conducting a secret handshake) is forced to participate in a secret handshake protocol. As a result, the group signatures are encrypted with a key derived from both: (a) the group-wide key and (b) the freshly established key. Moreover, we can thus ensure that, as long as a group signature is presented by a corrupt member, the traceability feature enables the group authority to hold that member accountable.[5]

As pertains to drawback (3) above (no self-distinction), we defer the discussion to later in the paper. Suffice it to say, that group signature schemes do not provide self-distinction by design, since doing so would undermine their version of unlinkability, which is different from the unlinkability in our context. To remedy the situation, we need some additional tools, as described in Section 8 below.

Since our approach involves combining a group signature scheme with a centralized group key distribution scheme, it is natural to examine potentially redundant components. In particular, both GSIG and CGKD schemes include a revocation mechanism. Furthermore, revocation in the former is quite expensive, usually based on dynamic accumulators [16]. Thus, it might seem worthwhile to drop the revocation of component of GSIG altogether in favor of the more efficient revocation in CGKD. This way, a revoked

[5] If the adversary is always the last to send its encrypted group signature, the traceability feature is lost. However, the achieved traceability is still valuable, for example, for investigating activities of members before they become corrupt.

member would simply not receive the new group-wide key in CGKD but would remain un-revoked as far as the underlying GSIG is concerned. To illustrate the problem with this optimization, consider an attack whereby a malicious but unrevoked member reveals the CGKD group-wide key to a revoked member. The latter can then take part in secret handshakes and successfully fool legitimate members. Whereas, if both revocation components are in place, the attack fails since the revoked member's group signature (exchanged as part of the handshake) would not be accepted as valid.

4 Building Block I: Group Signature Schemes

Let $\mathcal{U}$ be the universe of user identities. In a group signature scheme, there is an authority called a group manager ($\mathcal{GM}$) responsible for admitting users and identifying the actual signer of a given group signature[6]. There is also a set of users who can sign on behalf of the group. In addition, there is a set of entities called verifiers. All participants are modeled as probabilistic polynomial-time algorithms.

A group signature scheme, denoted by GSIG, is specified in Figure 4.

Fig. 3. Definition of Dynamic Group Signature Schemes

Setup: a probabilistic polynomial-time algorithm that, on input of a security parameter κ, outputs the specification of a cryptographic context including the group manager's public key $pk_{\mathcal{GM}}$ and secret key $sk_{\mathcal{GM}}$. This procedure may be denoted by $(pk_{\mathcal{GM}}, sk_{\mathcal{GM}}) \leftarrow \text{Setup}(1^\kappa)$.

Join: a protocol between $\mathcal{GM}$ and a user (conducted over a private and authenticated channel) that results in the user becoming a group member $\mathcal{U}$. Their common output includes the user's unique membership public key $pk_{\mathcal{U}}$, and perhaps some updated information that indicates the current state of the system. The user's output includes a membership secret key $sk_{\mathcal{U}}$. This procedure may be denoted by $(pk_{\mathsf{U}}, sk_{\mathsf{U}}, certificate_{\mathsf{U}}; pk_{\mathsf{U}}, certificate_{\mathsf{U}}) \leftarrow \text{Join}[\mathsf{U} \leftrightarrow \mathcal{GM}]$, where $\text{Join}[\mathsf{U} \leftrightarrow \mathcal{GM}]$ denotes an interactive protocol between U and $\mathcal{GM}$, $pk_{\mathsf{U}}, sk_{\mathsf{U}}, certificate_{\mathsf{U}}$ is the output of U, and $pk_{\mathsf{U}}, certificate_{\mathsf{U}}$ is the output of $\mathcal{GM}$. Besides, there may be some system state information that is made public to all participants.

Revoke: an algorithm that, on input of a group member's identity (and perhaps her public key $pk_{\mathcal{U}}$), outputs updated information that indicates the current state of the system after revoking the membership of a given group member.

Update: a deterministic algorithm that may be triggered by any Join or Revoke operation. It is run by each group member after obtaining system state information from the group manager.

Sign: a probabilistic algorithm that, on input of: key $pk_{\mathcal{GM}}$, $(sk_{\mathcal{U}}, pk_{\mathcal{U}})$ and a message m, outputs a group signature σ of m. This procedure may be denoted by $\sigma \leftarrow \mathsf{Sign}(pk_{\mathcal{GM}}, pk_{\mathsf{U}}, sk_{\mathsf{U}}, m)$.

Verify: an algorithm that, on input of: $pk_{\mathcal{GM}}$, an alleged group signature σ and a message m, outputs a binary value TRUE/FALSE indicating whether σ is a valid group signature (under $pk_{\mathcal{GM}}$) of m. This procedure may be denoted by TRUE/FALSE $\leftarrow \text{Verify}(pk_{\mathcal{GM}}, m, \sigma)$.

Open: an algorithm executed by the group manager $\mathcal{GM}$. It takes as input of a message m, a group signature σ, pk_{GM} and $sk_{\mathcal{GM}}$. It first executes Verify on the first three inputs and, if the output of Verify is *TRUE*, outputs some incontestable evidence (e.g., a membership public key $pk_{\mathcal{U}}$ and a proof) that allows anyone to identify the actual signer. This procedure may be denoted, without loss of generality, by $\mathsf{U} \leftarrow \mathsf{Open}(pk_{\mathcal{GM}}, sk_{\mathcal{GM}}, m, \sigma)$ if TRUE $\leftarrow \text{Verify}(pk_{\mathcal{GM}}, m, \sigma)$.

Informally, we require a group signature scheme to be `correct`, i.e., any signature produced by an honest group member using Sign is always accepted by Verify.

Following notable prior work [5, 10, 34], we say a group signature scheme is **secure** if it satisfies the following three properties (see Appendix B for a formal definition): (1) `full-traceability` – any valid group signature can be traced back to its actual signer, (2) `full-anonymity` – no adversary can identify the actual signer of a group signature, even if the actual signer's secret has been compromised, and (3) `no-misattribution` – no malicious group manager can misattribute a group signature to an honest group member.

[6] Sometimes, the two functionalities are assigned to two separate entities.

5 Building Block II: Centralized Group Key Distribution Scheme

Let κ be a security parameter, and $\mathbb{ID}$ be the set of possible group members (i.e., principals) such that $|\mathbb{ID}|$ is polynomially-bounded. There is a special entity called a key server or *Group Controller* ($\mathcal{GC}$) such that $\mathcal{GC} \notin \mathbb{ID}$.

To simplify the presentation, we assume that during system initialization (i.e., Setup below) $\mathcal{GC}$ can communicate with each legitimate member U through a *private* (i.e. untappable) channel, that $\mathcal{GC}$ can establish a common secret, if needed, with a joining user, and that, after the system initialization, $\mathcal{GC}$ can communicate with any $U \in \mathbb{ID}$ through an *authenticated* channel. By the term "authenticated" we mean that only $\mathcal{GC}$ can insert information into this channel. (We discuss these (and other) issues in Appendix H.) In line with notable prior work ([7, 9, 19, 6]), a centralized group key distribution scheme (CGKD) is defined in Figure 4.

Fig. 4. Definition of Centralized Group Key Distribution Schemes

Setup: The group controller $\mathcal{GC}$ generates a set of keys $K_{\mathcal{GC}}$, and distributes them to the current group members, $\Delta \subseteq \mathbb{ID}$, through the authenticated private channels. Each member $U_i \in \Delta$ holds a set of keys denoted by $K_{U_i} \subset K_{\mathcal{GC}}$, and there is a key, k (called group key or sometimes session key) that is common to all the current members, namely $k \in K_{\mathcal{GC}} \cap K_{U_1} \cap \ldots \cap K_{U_{|\Delta|}}$.

Since $\mathcal{GC}$ can execute the following Join and Leave protocols multiple times, we denote an instance i of $\mathcal{GC}$ as $\Pi^i_{\mathcal{GC}}$, where $i \in \mathbb{N}$ is (for example) a counter maintained by $\mathcal{GC}$ itself. Moreover, each instance $\Pi^i_{\mathcal{GC}}$ is associated with variables $\mathsf{sid}^i_{\mathcal{GC}}$ and $\mathsf{sk}^i_{\mathcal{GC}}$. Initially, $\mathsf{sk}^0_{\mathcal{GC}} \leftarrow k$ and $\forall\, i \in \mathbb{N}$, $\mathsf{sid}^i_{\mathcal{GC}} \leftarrow$ UNDEFINED, $\mathsf{sk}^i_{\mathcal{GC}} \leftarrow$ UNDEFINED.

Similarly, we denote an instance i of $U \in \mathbb{ID}$ as Π^i_U, where $i \in \mathbb{N}$ could be a counter maintained by U itself. Each instance Π^i_U is associated with variables acc^i_U, sid^i_U and sk^i_U. Initially, (1) $\forall\, U \in \mathbb{ID}, i \in \mathbb{N}$, $\mathsf{acc}^i_U \leftarrow$ FALSE, $\mathsf{sid}^i_U \leftarrow$ UNDEFINED, $\mathsf{sk}^i_U \leftarrow$ UNDEFINED, (2) $\forall\, U \in \Delta$, $\mathsf{sk}^0_U \leftarrow k$, and (3) $\forall\, U \in \mathbb{ID} \setminus \Delta$, $\mathsf{sk}^0_U \leftarrow$ UNDEFINED.

Join: This algorithm is executed by group controller $\mathcal{GC}$ (we abstract away the out-of-band authentication and establishment of an individual key for each of the new members). It takes as input: (1) identities of current group members, Δ, (2) identities of newly admitted group members, $\Delta' \subseteq \mathbb{ID} \setminus \Delta$, (3) keys held by the group controller, $K_{\mathcal{GC}}$, and (4) keys held by group members, $\{K_{U_i}\}_{U_i \in \Delta} = \{K_{U_i} : U_i \in \Delta\}$.

It outputs updated system state information, including: (1) identities of new group members, $\Delta^{new} \leftarrow \Delta \cup \Delta'$, (2) new keys for $\mathcal{GC}$ itself, $K^{new}_{\mathcal{GC}}$, (3) new keys for new group members, $\{K^{new}_{U_i}\}_{U_i \in \Delta^{new}}$, which are *somehow* sent to the legitimate users through the authenticated channels, (4) session id, $\mathsf{sid}^i_{\mathcal{GC}}$, of instance $\Pi^i_{\mathcal{GC}}$, i.e., a protocol-specific function of all communications sent by $\Pi^i_{\mathcal{GC}}$ (e.g., set $\mathsf{sid}^i_{\mathcal{GC}}$ as the concatenation, or ordered set, of all messages sent by $\Pi^i_{\mathcal{GC}}$ during the course of its execution), and (5) new group key $\mathsf{sk}^i_{\mathcal{GC}} \leftarrow k$, where $k \in K^{new}_{\mathcal{GC}}$.

Formally, denote it by $(\Delta^{new}, K^{new}_{\mathcal{GC}}, \{K^{new}_{U_i}\}_{U_i \in \Delta^{new}}, \mathsf{sid}^i_{\mathcal{GC}}, \mathsf{sk}^i_{\mathcal{GC}}) \leftarrow \mathsf{Join}(\Delta, \Delta', K_{\mathcal{GC}}, \{K_{U_i}\}_{U_i \in \Delta})$.

Leave: This algorithm is executed by the group controller $\mathcal{GC}$. It takes as input: (1) identities of current group members, Δ, (2) identities of leaving group members, $\Delta' \subseteq \Delta$, (3) keys held by the controller, $K_{\mathcal{GC}}$, and (4) keys held by group members, $\{K_{U_i}\}_{U_i \in \Delta}$.

It outputs updated system state information, including: (1) identities of new group members, $\Delta^{new} \leftarrow \Delta \setminus \Delta'$, (2) new keys for $\mathcal{GC}$, $K^{new}_{\mathcal{GC}}$, (3) new keys for new group members, $\{K^{new}_{U_i}\}_{U_i \in \Delta^{new}}$, which are *somehow* sent to the legitimate users through the authenticated channels, (4) session id, $\mathsf{sid}^i_{\mathcal{GC}}$, of instance $\Pi^i_{\mathcal{GC}}$, i.e., a protocol-specified function of all communications sent by $\Pi^i_{\mathcal{GC}}$ (e.g., set $\mathsf{sid}^i_{\mathcal{GC}}$ as the concatenation, or ordered set, of all messages sent by $\Pi^i_{\mathcal{GC}}$ during the course of its execution), and (5) new group key $\mathsf{sk}^i_{\mathcal{GC}} \leftarrow k$, where $k \in K^{new}_{\mathcal{GC}}$.

Formally, denote it by $(\Delta^{new}, K^{new}_{\mathcal{GC}}, \{K^{new}_{U_i}\}_{U_i \in \Delta^{new}}, \mathsf{sid}^i_{\mathcal{GC}}, \mathsf{sk}^i_{\mathcal{GC}}) \leftarrow \mathsf{Leave}(\Delta, \Delta', K_{\mathcal{GC}}, \{K_{U_i}\}_{U_i \in \Delta})$.

Rekey: This algorithm is executed by the new group members, i.e., instances $\Pi^j_{U_i}$ for $U_i \in \Delta^{new}$, which is one of the outputs of the corresponding Join or Leave event. In other words, $U_i \in \Delta^{new}$ runs this algorithm upon receiving the message from $\mathcal{GC}$ over the authenticated channel. The algorithm takes as input the received message and U_i's secrets, and is supposed to output the updated keys for the group member. If the execution of the algorithm is successful, $\Pi^j_{U_i}$ sets: (1) $\mathsf{acc}^j_{U_i} \leftarrow$ TRUE, (2) $\mathsf{sid}^j_{U_i}$ as the session id of instance $\Pi^j_{U_i}$, i.e., a protocol-specified function of all communication received by $\Pi^j_{U_i}$ (e.g., set $\mathsf{sid}^j_{U_i}$ as the concatenation, or ordered set, of all messages received by $\Pi^j_{U_i}$ during the course of the execution), (3) $K^{new}_{U_i}$, and (4) $\mathsf{sk}^j_{U_i} \leftarrow k$ as the new group key, where $k \in K^{new}_{\mathcal{GC}} \cap K^{new}_{U_1} \cap \ldots \cap K^{new}_{U_{|\Delta^{new}|}}$.

If the rekey is caused by a join event, every $U \in \Delta^{new}$ erases K_U and any temporary storage after obtaining K^{new}_U. If the rekey event is incurred by a leave event, every $U \in \Delta^{new}$ erases K_U and any temporary storage after obtaining K^{new}_U, and every *honest* leaving group member $U \in \Delta'$ erases K_U (although a *corrupt* one does not have to follow this protocol).

We require for a CGKD scheme to be `correct`, meaning that after each rekey process, all the group members share a common key with the group controller, and `secure`, meaning that no adversary (who does not compromise any legitimate group members) learns any information about the new group key (see Appendix C for a formal definition).

6 Building Block III: Distributed Group Key Agreement

Let κ be a security parameter. We assume a polynomial-size set $\mathbb{ID}$ of potential players. Any subset of $\mathbb{ID}$ may decide at any point to invoke distributed group key agreement. A distributed group key agreement scheme, DGKA, is specified in Figure 5 in accordance with the results in [6], [14, 12, 13] and [32].

Fig. 5. Definition of Distributed Key Agreement Schemes

Setup: a probabilistic polynomial-time algorithm that, on input of the security parameter κ, outputs the specification of a cryptographic context wherein the key agreement protocol operates, and a group-wide long-lived key LLK that is sent to all participants via an authenticated private channel. Note that LLK could be a symmetric key, or a set of public keys, such that, for each public key in the set, the corresponding private key is held by exactly one of the participants.
Since each principal can execute Setup multiple times, we denote an instance i of $U \in \mathbb{ID}$ as Π_U^i. Each instance Π_U^i is associated with variables $\mathsf{acc}_U^i, \mathsf{sid}_U^i, \mathsf{pid}_U^i, \mathsf{sk}_U^i$. Initially, $\forall\ U \in \mathbb{ID}$ and $i \in \mathbb{N}$, $\mathsf{acc}_U^i \leftarrow$ FALSE and $\mathsf{sid}_U^i, \mathsf{pid}_U^i, \mathsf{sk}_U^i \leftarrow$ UNDEFINED.

AuthKeyAgreement: a protocol that performs distributed group agreement between a subset of principals. After executing the protocol, each party outputs an indication of the protocol outcome (success or failure), and some secret information, in case of success. In more detail, the protocol is executed by m instances, $\Pi_{U_1}^{i_1}, \ldots, \Pi_{U_m}^{i_m}$, where $\{U_1, \ldots, U_m\} \subseteq \mathbb{ID}$. If the execution of $\Pi_{U_j}^{i_j}$ is successful, it sets (1) $\mathsf{acc}_{U_j}^{i_j} \leftarrow$ TRUE, (2) $\mathsf{sid}_{U_j}^{i_j}$ as the session id of instance $\Pi_{U_j}^{i_j}$, namely a protocol-specified function of all communication sent and received by $\Pi_{U_j}^{i_j}$ (e.g., we can simply set $\mathsf{sid}_{U_j}^{i_j}$ as the concatenation of all messages sent and received by $\Pi_{U_j}^{i_j}$ in the course of its execution), (3) $\mathsf{pid}_{U_j}^{i_j}$ as the session id of instance $\Pi_{U_j}^{i_j}$, namely the identities of the principals in the group with whom $\Pi_{U_j}^{i_j}$ intends to establish a session key (including U_j itself), and (4) $\mathsf{sk}_{U_j}^{i_j}$ as the newly established session key.

To achieve more flexibility, we also specify a "raw" or un-authenticated group key agreement component.

RawKeyAgreement: same as AuthKeyAgreement except that there is *no authentication* in protocol, even if there is a long-term key LLK. This is a straight-forward extension to the traditional *un-authenticated* Diffie-Hellman, e.g., [45]. Of course, we are aware of the dangers posed by man-in-the-middle attacks, but those can be remedied at a later phase, through the use of CGKD.

Informally speaking (see Appendix D for a formal definition), we require for a scheme to have correctness and security. Correctness means that the participants must obtain a new session secret (key) that can be used as a session key and `security` means that an adversary – who does not compromise any principals during the execution – does not learn any information about the new group session key.

7 GCD Secret Handshake Framework

The **GCD** framework has the following components:

GCD.**CreateGroup:** The group authority ($\mathcal{GA}$) plays the roles of both group manager in GSIG and group controller in CGKD.
- $\mathcal{GA}$ executes GSIG.Setup. This initializes a group signature scheme.
- $\mathcal{GA}$ executes CGKD.Setup. Thie initializes a centralized group key distribution scheme.
- $\mathcal{GA}$ generates a pair of public/private keys (pk_T, sk_T) with respect to an IND-CCA2 secure public key cryptosystem. This pair of keys enables $\mathcal{GA}$ to identify handshake participants in any handshake transcript.
- With regard to DGKA.Setup, we assume that there is a set of system-wide cryptographic parameters specified for the DGKA schemes; at least the honest users must insist on these parameters.

GCD.AdmitMember: $\mathcal{GA}$ executes CGKD.Join and GSIG.Join, except: (1) the updated system state information corresponding to GSIG is encrypted under the new group session key, and distributed as part of CGKD's state information updating message, and (2) the updating messages are published via an authenticated anonymous channel, e.g., a public bulletin board.

GCD.RemoveUser: $\mathcal{GA}$ executes CGKD.Leave and GSIG.Revoke, except: (1) the updated system state information corresponding to GSIG is encrypted under CGKD's new group session key, and distributed as part of CGKD's state information updating message, and (2) the updating messages are distributed via an authenticated anonymous channel.

GCD.Update: All non-revoked members execute GSIG.Update and CGKD.Rekey, except: (1) the updated system state information is obtained from an authenticated anonymous channel, and (2) if CGKD.Rekey succeeds, the update information corresponding to GSIG is decrypted using CGKD's new group key.

GCD.Handshake: Suppose m (≥ 2) users want to determine if they belong to the same group. We denote their group keys with respect to the CGKD as: $k_1, \ldots, k_m$, respectively. Note that, if they belong to the same group, then $k_1 = \ldots = k_m$.

Phase I: Preparation: All m parties jointly execute the DGKA.RawKeyAgreement protocol. We denote the resulting keys as: $k_1^*, \ldots, k_m^*$, respectively. If the execution is successful, then $k_1^* = \ldots = k_m^*$, and each party computes $k_i' = k_i^* \oplus k_i$.

Phase II: Preliminary Handshake: Each party publishes a tag $\mathsf{MAC}(k_i', s)$ corresponding to a message authentication code MAC (e.g., HMAC-SHA1), where s is a string unique to the party (e.g., the message it sent in the DGKA.RawKeyAgreement execution).[7]

Phase III: Full Handshake: There are two cases:

CASE 1: If all message authentication tags are valid (i.e., they belong to the same group), each party executes the following:

1. Encrypt k_i' to obtain ciphertext δ_i under the group authority's tracing public key pk_T; $\delta_i \leftarrow \mathsf{ENC}(pk_T, k_i')$.
2. Generate a group signature σ_i on the concatenation of δ_i via GSIG.Sign.
3. Encrypt σ_i using a symmetric key encryption algorithm and key k_i' to obtain a ciphertext θ_i; $\theta_i \leftarrow \mathsf{SENC}(k_i', \sigma_i)$.
4. Publish (θ_i, δ_i).
5. Upon receiving (θ_i, δ_i), execute the following:
   – Obtain the group signature by performing symmetric key decryption algorithm using k_i'; $\sigma_i \leftarrow \mathsf{SDEC}(k_i', \theta_i)$.
   – Run GSIG.Verify to check if σ_i is a valid group signature. If all group signatures are deemed valid, the party concludes that the corresponding parties all belong to the same group and stores the transcript including $\{(\theta_i, \delta_i)\}_{1 \leq i \leq m}$.

CASE 2: If at least one message authentication tag is invalid, each of party picks a pair (θ_i, δ_i) randomly selected from the ciphertext spaces corresponding to the symmetric key and public key cryptosystems, respectively.

GCD.TraceUser: Given a transcript of a secret handshake instance: $\{(\theta_i, \delta_i)\}_{1 \leq i \leq m}$, the group authority $\mathcal{GA}$ decrypts all δ_i's to obtain the corresponding session keys: $k_1', \ldots, k_m'$. In the worst case, the authority needs to try to search the right session key and decrypt all θ_i's to obtain the cleartext group signatures. Then, it executes GSIG.Open to identify the handshake parties.

Remark: In order to enable modular construction, we specify the handshake protocol as a three-phase protocol. Thus, the resulting framework is flexible, i.e., tailorable to application semantics. For example, if traceability is not required, a handshake may only involve Phase I and Phase II. Further, Phase II can be merged with Phase I; indeed this is the case for one of our concrete constructions below.

The following theorem is proved in Appendix E.

Theorem 1. *Assumming that* GSIG, DGKA, CGKD *are secure with respect to their corresponding definitions in Sections 4-6, the* **GCD** *framework is* **secure** *with respect to definitions in Section 2, except* `self-distinction`.

[7] If a broadcast channel is available, the tag is sent on it; else, it is sent point-to-point.

8 Three Instantiations

We now present three concrete secret handshake schemes. The first scheme employs "raw" (unauthenticated) contributory group key agreement, whereas, the second scheme uses authenticated contributory group key agreement, and the third scheme ensures that all handshake participants are distinct. These examples illustrate the flexibility of the **GCD** framework.

8.1 Example Scheme 1

This is a straight-forward instantiation of the **GCD** framework. We simply use an unauthenticated DGKA similar to [15, 45, 32], CGKD based on [48, 39], and GSIG based on [2, 16]. Theorem 1 immediately implies that this instantiation satisfies all properties specified in Section 2, except `self-distinction`.

The computational complexity for each party is the sum of the respective complexities incurred in each of the three building blocks. We remark that each party only needs to compute $O(m)$ modular exponentiations in total. Moreover, the total communication complexity is $O(m)$ (per-user) in terms of messages, where m is the number of protocol parties.

8.2 Example Scheme 2: Fast Handshakes

The next scheme has an attractive feature of authenticating membership as early as possible, by merging Phase I and Phase II of the handshake protocol. Since all protocols, except for the actual handshake, are the same as in the framework, we only focus on the handshake protocol illustrated in Figure 6.

Fig. 6. Example Scheme 2: Fast Handshakes

GCD.Handshake: Suppose m (≥ 2) users are the protocol participants and let their group keys with respect to CGKD be denoted as: $k_1, \ldots, k_m$. As before, if they all belong to the same group, then $k_1 = \ldots = k_m$.

Phase I & II: Preparation. All m parties jointly execute an instance of the DGKA.AuthKeyAgreement protocol, with authentication based on the group key (i.e., those parties with invalid message authentication tags are not taken into consideration). We denote the resulting keys as: $k'_1, \ldots, k'_m$. After a successful protocol run: $k'_1 = \ldots = k'_m$.

Phase III: Full Handshake. execute the following:

- encrypt k'_i to obtain ciphertext δ_i under the group authority's tracing public key pk_T
- generate a group signature σ_i on the concatenation of δ_i using GSIG.Sign
- encrypt σ_i using a symmetric key encryption algorithm and key k'_i to obtain a ciphertext $\theta_i \leftarrow \mathsf{SENC}(\mathsf{k}'_i, \sigma_i)$, and
- publish (θ_i, δ_i).
- upon receiving (θ_i, δ_i), execute the following:
  - obtain $\sigma_i \leftarrow \mathsf{SDEC}(\mathsf{k}'_i, \theta_i)$
  - invoke GSIG.Ver to check whether each σ_i is a valid group signature (if all signatures are valid, it concludes that they all belong to the same group and records the protocol transcript).

The proof of the following theorem is similar to the proof of Theorem 1.

Theorem 2. *If the underlying components:* GSIG, DGKA, CGKD *are secure (with respect to their corresponding definitions in Sections 4-6), then the above scheme satisfies all definitions in Section 2, except* `self-distinction`.

The computational and communication complexities of this scheme are essentially the same as in the previous example scheme, except for lower communication complexity.

8.3 Example Scheme 3: Self-Distinction

Neither previous instantiation guarantees that all handshake parties are distinct, i.e., all or some of the m handshake parties could in fact be "played" by the same party. We now sketch out a scheme based on a variant of a group signature scheme in [33] (see Appendix G). We only illustrate the handshake protocol (in Figure 7) since it is the only distinctive feature of this scheme.

A proof sketch of the following theorem can be found in Appendix F.

Fig. 7. Example Scheme 3: Self-Distinction

GCD.Handshake: Suppose m (≥ 2) users are taking part in the protocol. We denote their group keys with respect to the CGKD by $k_1, \ldots, k_m$, respectively. As before, if they belong to the same group, $k_1 = \ldots = k_m$.

Phase I: **Preparation:** m parties jointly execute the DGKA.RawKeyAgreement protocol. Let the resulting keys be denoted as: $k_1^*, \ldots, k_m^*$, respectively. (After a successful run $k_1^* = \ldots = k_m^*$.) Then, each party computes $k_i' = k_i^* \oplus k_i$

Phase II: **Preliminary Handshake:** Each party publishes a pair $\mathsf{MAC}(k_i', s)$, where s is the concatenation of all messages in executing DGKA.RawKeyAgreement. (Note that these two phases can be merged via DGKA.AuthKeyAgreement.)

Phase III: **Full Handshake:** We consider two cases:

Case 1: If all message authentication tags are valid (i.e., they belong to the same group), each party executes as follows:

1. Encrypts k_i' under the public key pk_T to obtain ciphertext δ_i
2. Generates a variant group signature σ_i on δ_i on s via GSIG.Sign, where the group signature is the same as what is called a traceable signature in [33], except the following:
   The value $T_7 = g^{k'}$ is jointly computed by all parties. (For example, by running a Pedersen-style [41] commitment protocol and then de-committing the values. Alternatively, k' can be obtained by running an instance of DGKA.AuthKeyAgreement.) As a result, the number of distinct T_6 values corresponds to the number of distinct parties.
3. Encrypt σ_i using a symmetric key encryption algorithm and key k_i' to obtain ciphertext $\theta_i \leftarrow \mathsf{SENC}(k_i', \sigma_i)$.
4. Publish (θ_i, δ_i).
5. Upon receiving (θ_i, δ_i), execute as follows:
   (a) Obtain $\sigma_i \leftarrow \mathsf{SDEC}(k_i', \theta_i)$
   (b) Run GSIG.Verify to check if each σ_i is a valid group signature (if all group signatures are valid, it concludes that they all belong to the same group and records the transcript).

Case 2: If at least one message authentication tag is invalid, each party picks a pair of (θ_i, δ_i) randomly selected from the ciphertext spaces corresponding to the symmetric key and public key cryptosystems, respectively.

Theorem 3. *If the underlying components:* GSIG, DGKA, CGKD *are secure (with respect to their corresponding definitions in Sections 4-6) and T_7 is uniformly distributed, then the above scheme satisfies all definitions in Section 2*

The computational complexity in terms of modular exponentiations (per-user) remains $O(m)$ and the communication complexity (per-user) is no more than double the complexity of the first instantiation above, where m is the number of handshake participants.

9 Related Work

The first secret handshake scheme [4] is based on the protocol of Sakai et al. [44], which targets the key exchange problem. Indeed, a secret handshake can be appropriately turned into an authenticated key exchange, but an authenticated key exchange does not necessarily imply a secret handshake, e.g., the two-party Diffie-Hellman key agreement scheme [25] does not lend itself to solving the secret handshake problem; see [4]. The scheme in [4] is based on bilinear maps in the setting of elliptic curves and its security is based on the associated assumptions. This scheme uses one-time pseudonyms to achieve unlinkability and does not offer the *No-misattribution* property.

A more recent follow-on result is due to Castelluccia, et al. [20]. This work constructs several handshake schemes in more standard cryptographic settings (avoiding bilinear maps) and provides some extensions for satisfying *No-misattribution.* However, it still relies on one-time pseudonyms to satisfy unlinkability. Another recent results by [49] requires each player to be aware of the information of other groups and offers weaker anonymity (referred to as k-anonymity).

References

1. W. Aiello, S. Bellovin, M. Blaze, J. Ioannidis, O. Reingold, R. Canetti, and A. Keromytis. Efficient, dos-resistant, secure key exchange for internet protocols. *ACM CCS 2002.*

2. G. Ateniese, J. Camenisch, M. Joye, and G. Tsudik. A practical and provably secure coalition-resistant group signature scheme. *CRYPTO 2000*.
3. G. Ateniese and G. Tsudik. Some open issues and new directions in group signatures. *Financial Cryptography 1999*.
4. D. Balfanz, G. Durfee, N. Shankar, D. Smetters, J. Staddon, and H. Wong. Secret handshakes from pairing-based key agreements. *IEEE SoSP 2003*.
5. M. Bellare, D. Micciancio, and B. Warinschi. Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. *EUROCRYPT 2003*.
6. M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. *EUROCRYPT 2000*.
7. M. Bellare and P. Rogaway. Entity authentication and key distribution. *CRYPTO 1993*.
8. M. Bellare and P. Rogaway. Optimal asymmetric encryption. *EUROCRYPT 1994*.
9. M. Bellare and P. Rogaway. Provably secure session key distribution– the three party case. *ACM STOC 1995*.
10. M. Bellare, H. Shi, and C. Zhang. Foundations of group signatures: The case of dynamic groups. Cryptology ePrint Archive, Report 2004/077, 2004.
11. C. Boyd, W. Mao, and K. Paterson. Deniable authenticated key establishment for internet protocols. *IEEE SoSP 2003*.
12. E. Bresson, O. Chevassut, and D. Pointcheval. Provably authenticated group Diffie-Hellman key exchange: the dynamic case. *Asiacrypt 2001*.
13. E. Bresson, O. Chevassut, and D. Pointcheval. Dynamic group Diffie-Hellman key exchange under standard assumptions. *Eurocrypt 2002*.
14. E. Bresson, O. Chevassut, D. Pointcheval, and J.-J. Quisquater. Provably authenticated group Diffie-Hellman key exchange. *ACM CCS 2001*.
15. M. Burmester and Y. Desmedt. A secure and efficient conference key distribution system. *EUROCRYPT 1994*.
16. J. Camenisch and A. Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. *CRYPTO 2002*.
17. J. Camenisch and M. Michels. A group signature scheme with improved efficiency. *ASIACRYPT 1998*.
18. J. Camenisch and M. Stadler. Efficient group signature schemes for large groups. *CRYPTO 1997*.
19. R. Canetti, T. Malkin, and K. Nissim. Efficient communication-storage tradeoffs for multicast encryption. *EUROCRYPT 1999*.
20. C. Castelluccia, S. Jarecki, and G. Tsudik. Secret handshakes from ca-oblivious encryption. *ASIACRYPT 2004*.
21. D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *CACM*, 24:84–88, February 1981.
22. D. Chaum. Blind signatures for untraceable payments. *CRYPTO 1982*.
23. D. Chaum and E. Van Heyst. Group signatures. *Eurocrypt 1991*.
24. R. Cramer and V. Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. *CRYPTO 1998*.
25. W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE ToIT*, IT-22:644–654, November 1976.
26. J. Douceur. The Sybil Attack. *IPTPS 2002*.
27. D. Dolev, C. Dwork, and M. Naor. Non-malleable cryptography. *ACM STOC 1991*.
28. M. Fischer, N. Lynch, and M. Patterson. Impossibility of distributed consensus with one faulty process. *JACM 1991*.
29. E. Fujisaki, T. Okamoto, D. Pointcheval, and J. Stern. Rsa-oaep is secure under the rsa assumption. *CRYPTO 2001*.
30. O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *JACM*, 33(4):792–807, October 1986.
31. S. Goldwasser and S. Micali. Probabilistic encryption. *JCSS*, 28(2):270–299, April 1984.
32. J. Katz and M. Yung. Scalable protocols for authenticated group key exchange. *CRYPTO 2003*.
33. A. Kiayias, Y. Tsiounis, and M. Yung. Traceable signatures. *EUROCRYPT 2004*.
34. A. Kiayias and M. Yung. Group signatures: Provable security, efficient constructions and anonymity from trapdoor-holders. Cryptology ePrint Archive, Report 2004/076, 2004.
35. J. Kilian and E. Petrank. Identity escrow. *CRYPTO 1998*.
36. H. Krawczyk. Sigma: The 'sign-and-mac' approach to authenticated diffie-hellman and its use in the ike-protocols. *CRYPTO 2003*.
37. N. Li, W. Du, and D. Boneh. Oblivious signature-based envelope. *ACM PODC 2003*.
38. M. Naor. Deniable ring authentication. *CRYPTO 2002*.
39. D. Naor, M. Naor and J. Lotspiech. Revocation and Tracing Schemes for Stateless Receivers. *CRYPTO 2001*.
40. M. Naor and M. Yung. Public-key cryptosystems provably secure against chosen ciphertext attack. *ACM STOC 1990*.
41. T. Pedersen. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. *CRYPTO 1991*.
42. C. Rackoff and D. R. Simon. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. *CRYPTO 1991*.
43. R. Rivest, A. Shamir, and Y. Tauman. How to leak a secret. *ASIACRYPT 2001*.
44. R. Sakai, K. Ohgishi, and M. Kasahara. Cryptosystems based on pairing. *ISCIS 2002*.
45. M. Steiner, G. Tsudik, and M. Waidner. Key agreement in dynamic peer groups. *IEEE TPDS*, 11(8):769–780, 2000.
46. Y. Sun and K. Liu. Securing dynamic membership information in multicast communications. *IEEE Infocom 2004*.
47. D. Wallner, E. Harder, and R. Agee. Key management for multicast: Issues and architecture. Internet Draft, Sept. 1998.
48. C. Wong, M. Gouda, and S. Lam. Secure group communication using key graphs. *IEEE/ACM ToN*, Vol. 8, 2000.
49. S. Xu and M. Yung. k-anonymous secret handshake with reusable credentials. *ACM CCS 2004*.
50. R. Bradshaw, J. Holt, and K. Seamons. Concealing Complex Policies with Hidden Credentials. *ACM CCS 2004*.

A Security Properties of Secret Handshake Schemes

Consider a probabilistic polynomial-time adversary $\mathcal{A}$ that is allowed access to the following oracles:

- $\mathcal{O}_{CG}(\cdot)$: activate a new $\mathcal{GA}$ to create a group via $\mathsf{SHS.CreateGroup}$. The identity, $\mathcal{GA}$, may be given by $\mathcal{A}$ as input. We assume that a $\mathcal{GA}$ is not under $\mathcal{A}$'s control before the group is established. However, $\mathcal{GA}$ may be corrupted immediately after its establishment, i.e., before any users are admitted into the group.
- $\mathcal{O}_{AM}(\cdot,\cdot)$: input includes the identity of a $\mathcal{GA}$ and, optionally, the identity U of a user under $\mathcal{A}$'s control. In case of $\mathcal{O}_{AM}(\mathcal{GA}, U)$, $\mathcal{GA}$ may admit the corrupt user U by executing $\mathsf{SHS.AdmitMember}$; in case of $\mathcal{O}_{AM}(\mathcal{GA})$, $\mathcal{GA}$ executes $\mathsf{SHS.AdmitMember}$ to admit an honest user and assign it a unique pseudonym U, if needed.
- $\mathcal{O}_{RU}(\cdot,\cdot)$: input includes the identity of a $\mathcal{GA}$ and a pseudonym U. The oracle activates $\mathsf{SHS.RemoveUser}$ to place U onto the corresponding $\mathcal{CRL}$. The system state information is appropriately updated.
- $\mathcal{O}_{Update}()$: whenever $\mathcal{O}_{AM}(\cdot,\cdot)$ or $\mathcal{O}_{RU}(\cdot,\cdot)$ is called, this oracle is also called so that the corresponding non-revoked group members can update their system state information.
- $\mathcal{O}_{HS}(\cdot,\ldots,\cdot)$: activate $\mathsf{SHS.Handshake}$ between members $U_1,\ldots,U_m$, where none or some (but not all) are under A's control. The honest members execute according to the protocol.
- $\mathcal{O}_{TU}(\cdot)$: on input of a successful handshake transcript, $\mathcal{GA}$ identifies the participating members.
- $\mathcal{O}_{Corrupt}(\cdot,\cdot)$: takes as input the identity of $\mathcal{GA}$ and, optionally, a pseudonym in $\mathcal{U}\cup\{\bot,\top\}$. In case of $\mathcal{O}_{Corrupt}(\mathcal{GA}, U \in \mathcal{U})$, the oracle returns U's current internal state information (including all secrets) to $\mathcal{A}$; in case of $\mathcal{O}_{Corrupt}(\mathcal{GA},\bot)$, the oracle returns $\mathcal{GA}$'s current internal state information for admitting members to $\mathcal{A}$; in case of $\mathcal{O}_{Corrupt}(\mathcal{GA},\top)$, the oracle returns $\mathcal{GA}$'s current internal state information for tracing members to $\mathcal{A}$. Once $\mathcal{GA}$ or U is corrupt, it executes according to $\mathcal{A}$'s wishes until the corruption is detected by some outside mechanism (e.g., intrusion detection systems). When the corruption of user U is detected, it is excluded from the group via $\mathsf{SHS.RemoveUser}$; when the corruption of $\mathcal{GA}$ is detected, the corresponding group is simply excluded from the system.

We now specify the desired security properties for a secret handshake scheme. We note that they form a superset of security properties found in prior work [4, 49, 20].

`Correctness`: Suppose a set of participants, $\Delta = \{U_1,\ldots,U_m\}$, are conducting the secret handshake protocol. If all the U_i's belong to the same group, then $\mathsf{Handshake}(\Delta)$ always returns "1"; otherwise, it always returns "0".

`Resistance to impersonation`: no *passive* adversary $\mathcal{A} \notin G$ who does not corrupt any members of G has only a negligible probability in convincing an honest user $U \in G$ that $\mathcal{A} \in G$. Formally, consider the following game or experiment.

```
Experiment $\mathsf{RIA}_{\mathsf{SHS},\mathcal{A}}(1^\kappa)$:
    $(\mathcal{GA}, \Delta' = \{U'_i\}_{i\in\mathbb{I}}, b) \leftarrow \mathcal{A}^{\mathcal{O}_{CG}(\cdot),\mathcal{O}_{AM}(\cdot,\cdot),\mathcal{O}_{RU}(\cdot,\cdot),\mathcal{O}_{Update}(),\mathcal{O}_{HS}(\cdot,\cdot),\mathcal{O}_{TU}(\cdot),\mathcal{O}_{Corrupt}(\cdot,\cdot)}(1^\kappa)$
    Return "1" if the following holds and "0" otherwise:
        (1) $\forall\ i \in \mathbb{I}$: $U'_i$ belongs to the group managed by $\mathcal{GA}$
        (2) there is no $\mathcal{O}_{Corrupt}(\mathcal{GA})$ query
        (3) $\forall\ i \in \mathbb{I}$: there is no $\mathcal{O}_{RU}(\mathcal{GA}, U'_i)$ query
        (4) if there is an $\mathcal{O}_{AM}(\mathcal{GA}, X \in \mathcal{U})$ query, then there is also an $\mathcal{O}_{RU}(\mathcal{GA}, X)$ query
        (5) if there is an $\mathcal{O}_{Corrupt}(\mathcal{GA}, X \in \mathcal{U})$ query, then there is also an $\mathcal{O}_{RU}(\mathcal{GA}, X)$ query
        (6) $|\Delta'| = m-1$ and $\Delta = \{\mathcal{A}\} \cup \Delta' = \{U_1,\ldots,U_{b-1},\mathcal{A},U_{b+1},\ldots,U_m\}$
        (7) $\mathsf{SHS.Handshake}(\Delta)$ returns "1"
```

Let $\mathsf{AdvRIA}_{\mathsf{SHS},\mathcal{A}}(1^\kappa) = \Pr[\mathsf{RIA}_{\mathsf{SHS},\mathcal{A}}(1^\kappa)$ returns "1"$]$, which is the probability that the experiment $\mathsf{RIA}_{\mathsf{SHS},\mathcal{A}}(1^\kappa)$ returns "1", where the probability is taken over the coins used for selecting the cryptographic parameters, the coins used by the oracles, and the coins used by the adversary. A secret handshake scheme SHS is "`resistant to impersonation attacks`" if for $\forall\mathcal{A}$, $\mathsf{AdvRIA}_{\mathsf{SHS},\mathcal{A}}(1^\kappa)$ is negligible in κ.

`Resistance to detection`: no adversary $\mathcal{A} \notin G$ can tell it is interacting with an honest user $U \in G$ or a simulator. Formally, consider the following game or experiment.

```
Experiment $\mathsf{RDA}_{\mathsf{SHS},\mathcal{A}}(1^\kappa)$:
    $\mathsf{bit} \in_R \{0,1\}$
    $(\mathcal{GA}, \Delta' = \{U'_i\}_{i\in\mathbb{I}}, b, \mathsf{stateinfo}) \leftarrow \mathcal{A}^{\mathcal{O}_{CG}(\cdot),\mathcal{O}_{AM}(\cdot,\cdot),\mathcal{O}_{RU}(\cdot,\cdot),\mathcal{O}_{Update}(),\mathcal{O}_{HS}(\cdot,\cdot),\mathcal{O}_{TU}(\cdot),\mathcal{O}_{Corrupt}(\cdot,\cdot)}(1^\kappa)$
        where $|\Delta'| = m-1$ and $\Delta' \subseteq G$ for $G$ managed by $\mathcal{GA}$
    set $\Delta = \Delta' \cup \{\mathcal{A}\} = \{U_1,\ldots,U_{b-1},\mathcal{A},U_{b+1},\ldots,U_m\}$
    if $\mathbf{bit} == 0$ execute $\mathsf{SHS.Handshake}(U_1,\ldots,U_{b-1},\mathcal{A},U_{b+1},\ldots,U_m)$
        else execute $\mathsf{SHS.Handshake}(\mathsf{SIM},\ldots,\mathsf{SIM},\mathcal{A},\mathsf{SIM},\ldots,\mathsf{SIM})$ where $\mathsf{SIM}$ is a simulator
    $\mathsf{bit}' \leftarrow \mathcal{A}^{\mathcal{O}_{CG}(\cdot),\mathcal{O}_{AM}(\cdot,\cdot),\mathcal{O}_{RU}(\cdot,\cdot),\mathcal{O}_{Update}(),\mathcal{O}_{HS}(\cdot,\cdot),\mathcal{O}_{TU}(\cdot),\mathcal{O}_{Corrupt}(\cdot,\cdot)}(\mathsf{stateinfo})$
    Return "1" if the following hold and "0" otherwise
        (1) $\mathsf{bit} == \mathsf{bit}'$
        (2) there is no $\mathcal{O}_{Corrupt}(\mathcal{GA})$ query
        (3) $\forall\ i \in \mathbb{I}$: there is no $\mathcal{O}_{RU}(\mathcal{GA}, U'_i)$ query
        (4) if there is an $\mathcal{O}_{AM}(\mathcal{GA}, X \in \mathcal{U})$ query, then there is also an $\mathcal{O}_{RU}(\mathcal{GA}, X)$ query
        (5) if there is an $\mathcal{O}_{Corrupt}(\mathcal{GA}, X \in \mathcal{U})$ query, then there is also an $\mathcal{O}_{RU}(\mathcal{GA}, X)$ query
```

Let $\mathsf{AdvRDA}_{\mathsf{SHS},\mathcal{A}}(\kappa) = |\Pr[\mathsf{RDA}_{\mathsf{SHS},\mathcal{A}}(\kappa)$ returns "1"$|\mathsf{bit} = 1] - \Pr[\mathsf{RDA}_{\mathsf{SHS},\mathcal{A}}(\kappa)$ returns "1"$|\mathsf{bit} = 0|$. A scheme SHS is "`resistant to detection`" if for $\forall\mathcal{A}$, $\mathsf{AdvRDA}_{\mathsf{SHS}}(\mathcal{A})$ is negligible in κ.

`Unlinkability`: no adversary $\mathcal{A}$ is able to associate two handshakes involving a same honest user $U \in G$, even if $\mathcal{A} \in G$ and $\mathcal{A}$ participated in both executions. Formally, consider the following experiment.

```
Experiment $\mathsf{Unlink}_{\mathsf{SHS},\mathcal{A}}(1^\kappa)$:
    $\mathsf{bit} \in_R \{0,1\}$
    $(\varDelta_0 = \{U_1,\dots,U_{m_0}\}, \mathsf{stateinfo}_0) \leftarrow \mathcal{A}^{\mathcal{O}_{CG}(\cdot),\mathcal{O}_{AM}(\cdot,\cdot),\mathcal{O}_{HS}(\cdot,\cdot),\mathcal{O}_{RU}(\cdot,\cdot),\mathcal{O}_{Corrupt}(\cdot,\cdot)}(1^\kappa)$
    $(\varDelta_1 = \{U'_1,\dots,U'_{m_1}\}, \mathsf{stateinfo}_1) \leftarrow \mathcal{A}^{\mathcal{O}_{CG}(\cdot),\mathcal{O}_{AM}(\cdot,\cdot),\mathcal{O}_{HS}(\cdot,\cdot),\mathcal{O}_{RU}(\cdot,\cdot),\mathcal{O}_{Corrupt}(\cdot,\cdot)}(\mathsf{stateinfo}_0)$
    if $\mathsf{bit} == 0$ then execute
            $\mathsf{Handshake}(\{U_1,\dots,U_{i-1},U_i,U_{i+1},U_{m_0}\})$ and
            $\mathsf{Handshake}(\{U'_1,\dots,U'_{j-1},U'_j,U'_{j+1},\dots,U'_{m_1}\})$
        else execute
            $\mathsf{Handshake}(\{U_1,\dots,U_{i-1},\mathsf{SIM},U_{i+1},U_{m_0}\})$ and
            $\mathsf{Handshake}(\{U'_1,\dots,U'_{j-1},\mathsf{SIM},U'_{j+1},\dots,U'_{m_1}\})$
    Return "1" if the following hold and "0" otherwise:
        (1) $\mathsf{bit}' \leftarrow \mathcal{A}^{\mathcal{O}_{CG}(\cdot),\mathcal{O}_{AM}(\cdot,\cdot),\mathcal{O}_{HS}(\cdot,\cdot),\mathcal{O}_{RU}(\cdot,\cdot),\mathcal{O}_{Corrupt}(\cdot,\cdot)}(\mathsf{stateinfo}_0, \mathsf{stateinfo}_1)$
        (2) $\forall U \in \varDelta_0$: there is no $\mathcal{O}_{RU}(\mathcal{GA}, U)$ query
                where $\mathcal{GA}$ is the group authority of the group to which $U$ belongs
        (3) $\forall U' \in \varDelta_1$: there is no $\mathcal{O}_{RU}(\mathcal{GA}', U')$ query
                where $\mathcal{GA}'$ is the group authority of the group to which $U'$ belongs
        (4) $U_i = U'_j$
        (5) there is no $\mathcal{O}_{AM}(\mathcal{GA}, U_i)$ query
        (6) there is no $\mathcal{O}_{Corrupt}(\mathcal{GA}, U_i)$ query
        (7) $\mathsf{bit} == \mathsf{bit}'$
```

Let $\mathsf{AdvUnlink}_{\mathsf{SHS},\mathcal{A}}(1^\kappa) = |\Pr[\mathsf{Unlink}_{\mathsf{SHS},\mathcal{A}}(1^\kappa)$ returns "1"$|\mathsf{bit} = 1] - \Pr[\mathsf{Unlink}_{\mathsf{SHS},\mathcal{A}}(1^\kappa)$ returns "1"$|\mathsf{bit} = 0]|$, where the probability is taken over all tossed coins. A scheme SHS is "`unlinkable`" if for $\forall\mathcal{A}$, $\mathsf{AdvUnlink}_{\mathsf{SHS},\mathcal{A}}(1^\kappa)$ is negligible in κ.

`Indistinguishability to eavesdroppers`: no adversary $\mathcal{A}$ who does not participate in a handshake protocol can tell a successful handshake between $\{U_1,\dots,U_m\}$ from an unsuccessful one, even if $\mathcal{A} \in G$. Formally, consider the following experiment.

```
Experiment $\mathsf{INDeav}_{\mathsf{SHS},\mathcal{A}}(1^\kappa)$:
    $\mathsf{bit} \in_R \{0,1\}$
    $(\mathcal{GA}, \varDelta = \{U_1,\dots,U_m\}, \mathsf{stateInfo}) \leftarrow \mathcal{A}^{\mathcal{O}_{CG}(\cdot),\mathcal{O}_{AM}(\cdot,\cdot),\mathcal{O}_{HS}(\cdot,\cdot),\mathcal{O}_{RU}(\cdot,\cdot),\mathcal{O}_{Corrupt}(\cdot,\cdot)}(1^\kappa)$
    if $\mathsf{bit} == 0$ then execute $\mathsf{SHS.Handshake}(U_1,\dots,U_m)$
        else execute $\mathsf{SHS.Handshake}(\mathsf{SIM},\dots,\mathsf{SIM})$ where $\mathsf{SIM}$ is a simulator
    $\mathsf{bit}' \leftarrow \mathcal{A}^{\mathcal{O}_{CG}(\cdot),\mathcal{O}_{AM}(\cdot,\cdot),\mathcal{O}_{HS}(\cdot,\cdot),\mathcal{O}_{RU}(\cdot,\cdot),\mathcal{O}_{Corrupt}(\cdot,\cdot)}(\mathsf{stateinfo})$
    Return "1" if the following hold and "0" otherwise:
        (1) $\mathsf{bit} == \mathsf{bit}'$
        (2) $\forall U \in \varDelta$, there is no $\mathcal{O}_{RU}(\mathcal{GA}, U)$ query
        (3) there is no $\mathcal{O}_{Corrupt}(\mathcal{GA}, \top)$ query
        (4) if there is an $\mathcal{O}_{AM}(\mathcal{GA}, X)$ query, then $X \notin \varDelta$ query
```

Let $\mathsf{AdvINDeav}_{\mathsf{SHS},\mathcal{A}} = |\Pr[\mathsf{INDeav}_{\mathsf{SHS},\mathcal{A}}(1^\kappa)$ returns "1"$]-1/2|$, where the probability is taken over all flipped coins. A scheme SHS is "`indistinguishable to eavesdroppers`" if for $\forall\mathcal{A}$, $\mathsf{AdvINDeav}_{\mathsf{SHS}}(\mathcal{A})$ is negligible in κ.

`Traceability`: $\mathcal{GA}$ can trace all users involved in a given handshake session, as long as one of them is honest. Formally, consider the following experiment.

```
Experiment $\mathsf{TraceUser}_{\mathsf{SHS},\mathcal{A}}(1^\kappa)$:
    $(\mathcal{GA}, \varDelta = \{U_1,\dots,U_m\}, \mathsf{stateInfo}) \leftarrow \mathcal{A}^{\mathcal{O}_{CG}(\cdot),\mathcal{O}_{AM}(\cdot,\cdot),\mathcal{O}_{RU}(\cdot,\cdot),\mathcal{O}_{Update}(),\mathcal{O}_{HS}(\cdot,\cdot),\mathcal{O}_{TU}(\cdot),\mathcal{O}_{Corrupt}(\cdot,\cdot)}(1^\kappa)$
        where $\varDelta \subseteq G$ for $G$ managed by $\mathcal{GA}$
    execute $\mathsf{Handshake}(U_1,\dots,U_m)$ to obtain a transcript $\tau$
    Return "1" if the following hold and "0" otherwise
        (1) there is no $\mathcal{O}_{Corrupt}(\mathcal{GA}, \top)$ query
        (2) there exists $U \in \varDelta$ such that $\nexists\mathcal{O}_{Corrupt}(\mathcal{GA}, U)$ or $\nexists\mathcal{O}_{AM}(\mathcal{GA}, U)$
        (3) $\varDelta' \leftarrow \mathsf{TraceUser}(\tau)$
        (4) $\varDelta' \setminus \varDelta \neq \emptyset$
```

Let $\mathsf{AdvTraceUser}_{\mathsf{SHS},\mathcal{A}} = |\Pr[\mathsf{TraceUser}_{\mathsf{SHS},\mathcal{A}}(1^\kappa)$ returns "1"$|$, where the probability is taken over the choice of the cryptographic context, and the coins used in the experiment. A scheme SHS achieves "`traceability`" if for $\forall\mathcal{A}$, $\mathsf{AdvTraceUser}_{\mathsf{SHS},\mathcal{A}}$ is negligible in κ.

`No-misattribution`: no participant, no coalition of malicious parties (including any number of group members and the $\mathcal{GA}$) is able to frame an honest member as being involved in a secret database. Formally, consider the following experiment.

```
Experiment $\mathsf{Misattribution}_{\mathsf{SHS},\mathcal{A}}(1^\kappa)$:
    $(\mathcal{GA}, \mathit{\Delta} = \{U_1, \ldots, U_m\}) \leftarrow \mathcal{A}^{\mathcal{O}_{CG}(\cdot),\mathcal{O}_{AM}(\cdot,\cdot),\mathcal{O}_{RU}(\cdot,\cdot),\mathcal{O}_{Update}(),\mathcal{O}_{HS}(\cdot,\cdot),\mathcal{O}_{TU}(\cdot),\mathcal{O}_{Corrupt}(\cdot,\cdot)}(1^\kappa)$
        where $\mathit{\Delta} \subseteq G$ for $G$ managed by $\mathcal{GA}$
    execute $\mathsf{Handshake}(U_1, \ldots, U_m)$ to obtain a transcript $\tau$
    Return "1" if the following hold and "0" otherwise
        (1) $\mathit{\Delta}' \leftarrow \mathsf{TraceUser}(\tau)$
        (2) $U \in \mathit{\Delta}' \setminus \mathit{\Delta}$
        (3) there is no $\mathcal{O}_{Corrupt}(U)$ or $\mathcal{O}_{AM}(\mathcal{GA}, U)$ query
```

Let $\mathsf{AdvMisattribution}_{\mathsf{SHS},\mathcal{A}} = |\Pr[\mathsf{Misattribution}_{\mathsf{SHS},\mathcal{A}}(1^\kappa)$ returns "1"$]|$, where the probability is taken over the choice of the cryptographic context, and the coins used in the experiment. A scheme SHS achieves "`No-misattribution`" if for $\forall \mathcal{A}$, $\mathsf{Misattribution}_{\mathsf{SHS},\mathcal{A}}$ is negligible in κ.

`Forward-repudiability`: Suppose at some time t_1, a set of honest users in $\mathit{\Delta} = \{U_1, \ldots, U_m\}$ engage in a handshake. At time $t_2 > t_1$, it should not be possible for any $U_i \in \mathit{\Delta}$ to prove to a third party that any $U \in \mathit{\Delta} \setminus \{U_i\}$ participated in the handshake session at time t_1, even if U_i reveals its own secrets. Formally, we require that for any successful handshake session $\mathsf{Handshake}(\mathit{\Delta} = \{U_1, \ldots, U_m\})$ with transcript τ, there exists a polynomial-time algorithm that is able to simulate τ.

`Self-distinction`: if more than 2 parties are involved in a handshake, each participant is ensured that all other participants are different. Formally, consider the following experiment.

```
Experiment $\mathsf{SelfDist}_{\mathsf{SHS},\mathcal{A}}(1^\kappa)$:
    $(\mathcal{GA}, \mathit{\Delta} = \{U_1, \ldots, U_m\}) \leftarrow \mathcal{A}^{\mathcal{O}_{CG}(\cdot),\mathcal{O}_{AM}(\cdot,\cdot),\mathcal{O}_{RU}(\cdot,\cdot),\mathcal{O}_{Update}(),\mathcal{O}_{HS}(\cdot,\cdot),\mathcal{O}_{TU}(\cdot),\mathcal{O}_{Corrupt}(\cdot,\cdot)}(1^\kappa)$
        where $\mathit{\Delta} \subseteq G$ for $G$ managed by $\mathcal{GA}$
    execute $\mathsf{Handshake}(U_1, \ldots, U_m)$ to obtain a transcript $\tau$
    $\mathit{\Delta}' \leftarrow \mathsf{TraceUser}(\tau)$
    Return "1" if the following holds and "0" otherwise
        (1) there is no $\mathcal{O}_{Corrupt}(\mathcal{GA}, \top)$ query
        (2) $\exists U \in \mathit{\Delta}$ s.t. there is no $\mathcal{O}_{Corrupt}(\mathcal{GA}, U)$ query
        (3) $|\mathit{\Delta}'| < |\mathit{\Delta}|$
```

Let $\mathsf{AdvSelfDist}_{\mathsf{SHS},\mathcal{A}}(1^\kappa) = |\Pr[\mathsf{SelfDist}_{\mathsf{SHS},\mathcal{A}}(1^\kappa)$ returns "1"$]|$, where the probability is taken over the choice of the cryptographic context, and the coins used in the experiment. A scheme SHS achieves "`self-distinction`" if for $\forall \mathcal{A}$, $\mathsf{AdvSelfDist}_{\mathsf{SHS},\mathcal{A}}(1^\kappa)$ is negligible in κ.

B Security Properties of Group Signature Schemes

In this paper, we follow the definitions presented in [5, 10, 34] which include three properties: `full-traceability`, `full-anonymity`, and `no-misattribution`. In order to meaningfully specify the security properties, we first consider the oracles an adversary is allowed to have access to.

- $\mathcal{O}_{Setup}(1^\kappa)$: activate $\mathcal{GM}$ to execute $\mathsf{GSIG.Setup}$ on security parameter 1^κ.
- $\mathcal{O}_{Join}(U)$: result in a user U becoming a member of the group.
- $\mathcal{O}_{Revoke}(U)$: result in the membership of U being revoked.
- $\mathcal{O}_{Update}()$: result in the non-revoked members updating their system state information.
- $\mathcal{O}_{Sign}(M)$: activate a group member U to sign a message M by executing the Sign algorithm.
- $\mathcal{O}_{Open}(\sigma)$: activate the group manager $\mathcal{GM}$ to execute the Open algorithm on signature σ.
- $\mathcal{O}_{Corrupt}(U)$: return U's private information.
- $\mathcal{O}_{Corrupt}(\mathcal{GM}, \bot)$: return $\mathcal{GM}$'s private information used to admit group members (i.e., the secrets used to issue membership certificates).
- $\mathcal{O}_{Corrupt}(\mathcal{GM}, \top)$: return $\mathcal{GM}$'s private information used to Open group signatures (i.e., the secrets used to revoke anonymity).

Remark. First, a user or (part of) a group manager may be corrupted before becoming part of a group signature scheme. This is captured by allowing an adversary to immediately corrupt an entity after its initialization such as being admitted as a group member. Second, we differentiate the case that an adversary corrupts a group manager's capability for admitting members from the case that an adversary corrupts a group manager's capability for opening group signatures. This reflects the good practice that one server is responsible for admitting members, and another is responsible for opening group signatures.

`Full-traceability`: any group signature can be traced back to its actual signer. Formally, consider the following game or experiment.

```
Experiment $\mathsf{FullTrace}_{\mathsf{GSIG},\mathcal{A}}(1^\kappa)$
    $(pk_{\mathcal{GM}}, sk_{\mathcal{GM}}) \leftarrow \mathsf{Setup}(1^\kappa)$
    $(M, \sigma) \leftarrow \mathcal{A}^{\mathcal{O}_{Join}(\cdot),\mathcal{O}_{Sign}(\cdot,\cdot),\mathcal{O}_{Open}(\cdot),\mathcal{O}_{Corrupt}(\cdot),\mathcal{O}_{Corrupt}(\mathcal{GM},\top)}()$
```

```
Return "1" if
    (1) $\mathsf{Verify}(M, \sigma)$ returns TRUE
    (2) $\Delta = \{U' \in \mathbb{ID} \mid \exists\ \mathcal{O}_{Corrupt}(U')\}$
    (3) $U \leftarrow \mathsf{Open}(pk_{\mathcal{GM}}, sk_{\mathcal{GM}}, M, \sigma)$
    (4) $U \notin \Delta$
```

A group signature scheme fulfills `full-traceability` if $\forall \mathcal{A}$, $\Pr[\mathsf{FullTrace}_{\mathsf{GSIG},\mathcal{A}}(1^\kappa)$ returns "1"] is negligible in κ.

`Full-anonymity`: no adversary can identify the actual signer of a group signature, even if the actual signer's secret has been compromised. Formally, consider the following game or experiment.

```
Experiment $\mathsf{FullAnon}_{\mathsf{GSIG},\mathcal{A}}(1^\kappa)$
    $(pk_{\mathcal{GM}}, sk_{\mathcal{GM}}) \leftarrow \mathsf{Setup}(1^\kappa)$
    $b \in_R \{0, 1\}$
    $(U_0 \in \mathbb{ID}, U_1 \in \mathbb{ID}, M, \mathsf{stateinfo}) \leftarrow \mathcal{A}^{\mathcal{O}_{Join}(\cdot),\mathcal{O}_{Sign}(\cdot,\cdot),\mathcal{O}_{Open}(\cdot),\mathcal{O}_{Corrupt}(\cdot),\mathcal{O}_{Corrupt}(\mathcal{GM},\perp)}(1^\kappa)$
    Return "1" if
        (1) $\sigma \leftarrow \mathsf{Sign}(pk_{\mathcal{GM}}, pk_{U_b}, sk_{U_b}, M)$
        (2) $b' \leftarrow \mathcal{A}^{\mathcal{O}_{Join}(\cdot),\mathcal{O}_{Sign}(\cdot,\cdot),\mathcal{O}_{Open}(\cdot),\mathcal{O}_{Corrupt}(\cdot),\mathcal{O}_{Corrupt}(\mathcal{GM},\perp)}(\mathsf{stateinfo})$,
        (3) there is no $\mathcal{O}_{Open}(\sigma)$ query
        (4) $b' = b$
```

A group signature scheme fulfills `full-anonymity` if $|\Pr[\mathsf{FullAnon}_{\mathsf{GSIG},\mathcal{A}}(1^\kappa)$ returns "1"$] - 1/2|$ is negligible in κ.

`No-misattribution`: a compromised group manager cannot misattribute a group signature to an honest group member. Formally, consider the following game or experiment.

```
Experiment $\mathsf{NoMisAttri}_{\mathsf{GSIG},\mathcal{A}}(1^\kappa)$
    $(pk_{\mathcal{GM}}, sk_{\mathcal{GM}}) \leftarrow \mathsf{Setup}(1^\kappa)$
    $(M, U \in \mathbb{ID}, \sigma) \leftarrow \mathcal{A}^{\mathcal{O}_{Join}(\cdot),\mathcal{O}_{Sign}(\cdot,\cdot),\mathcal{O}_{Open}(\cdot),\mathcal{O}_{Corrupt}(\cdot),\mathcal{O}_{Corrupt}(\mathcal{GM},\perp),\mathcal{O}_{Corrupt}(\mathcal{GM},\top)}(1^\kappa)$
    Return "1" if
        (1) $\sigma$ is returned by $\mathcal{O}_{Sign}(U, M)$
        (2) there is no $\mathcal{O}_{Corrupt}(U)$ query
        (3) $U \leftarrow \mathsf{Open}(pk_{\mathcal{GM}}, sk_{\mathcal{GM}}, M, \sigma)$
```

A group signature scheme fulfills `no-misattribution` if $\Pr[\mathsf{NoMisAttri}_{\mathsf{GSIG},\mathcal{A}}(1^\kappa)$ returns "1"] is negligible in κ.

C Security Properties of Centralized Key Distribution Schemes

We consider an adversary that has complete control over all the communications in the network. To simplify the definition, we assume that the group controller is never compromised; this is not necessarily a restriction because an adversary could have compromised all the group members (and thus have obtained the secrets the group controller holds). An adversary's interaction with principals in the network (more specifically, with the various instances) is modeled by allowing it to have access to the following oracles:

- $\mathcal{O}_{Send}(U, i, M, action)$: Send a message M to instance Π_U^i and outputs the reply generated by the instance, where $U \in \{\mathcal{GC}\} \cup \mathbb{ID}$, $action \in \{\mathsf{Setup}, \mathsf{Join}, \mathsf{Leave}, \mathsf{Rekey}\}$ meaning that the instance will execute according to the corresponding protocol, and M specifies the needed information for executing the protocol.
- $\mathcal{O}_{Reveal}(U, i)$: Output session key sk_U^i, where $U \in \mathbb{ID}$.
- $\mathcal{O}_{Corrupt}(U)$: Output K_U of player U, where $U \in \mathbb{ID}$. Note that the session key always belongs to K_U. Moreover, this oracle access accommodates the case that some users were corrupted before the system setup.
- $\mathcal{O}_{Test}(U, i)$: This oracle may be queried only once, at any time during the adversary's execution. A random bit b is generated: if $b = 1$ the adversary is given sk_U^i where $U \in \mathbb{ID} \cup \{\mathcal{GC}\}$, and if $b = 0$ the adversary is given a random session key.

`Correctness`. We require that any group communication scheme satisfy the following `correctness`: $\forall\, U \in \Delta^{new} \subseteq \mathbb{ID}, i, j \in \mathbb{N}$, if $\mathsf{sid}_U^i \subseteq \mathsf{sid}_{\mathcal{GC}}^j$ and $\mathsf{acc}_U^i = \text{TRUE}$, then $\mathsf{sk}_U^i = \mathsf{sk}_{\mathcal{GC}}^j$ and $K_U^{new} \subset K_{\mathcal{GC}}^{new}$.

`Security`. Intuitively, it means that an adversary who does not compromise any legitimate group member learns no information about the *new* group key. Formally, consider the following event Succ:

(1) The adversary queries the $\mathcal{O}_{Test}(U, i)$ oracle with respect to instance Π_U^i (with $\mathsf{acc}_U^i = \text{TRUE}$ if $U \in \mathbb{ID}$) and correctly guesses the bit b used by the $\mathcal{O}_{Test}(U, i)$ oracle in answering this query.
(2) There is no $\mathcal{O}_{Reveal}(V, \ell)$ query for $V \in \Delta^{new}$, where Δ^{new} is the legitimate group members corresponding to the instance Π_U^i such that $\mathsf{sid}_V^\ell \subseteq \mathsf{sid}_{\mathcal{GC}}^j \supseteq \mathsf{sid}_U^i$.
(3) Before the $\mathcal{O}_{Send}(\mathcal{GC}, j, *, \mathsf{Join})$ or $\mathcal{O}_{Send}(\mathcal{GC}, j, *, \mathsf{Leave})$ query such that $\mathsf{sid}_{\mathcal{GC}}^j \supseteq \mathsf{sid}_U^i$, if there was any $\mathcal{O}_{Corrupt}(V)$ query, then there must have been an $\mathcal{O}_{Send}(\mathcal{GC}, *, V, \mathsf{Leave})$ query.

(4) After the $\mathcal{O}_{Send}(\mathcal{GC}, j, *, \mathsf{Join})$ or $\mathcal{O}_{Send}(\mathcal{GC}, j, *, \mathsf{Leave})$ query such that $\mathsf{sid}^j_{\mathcal{GC}} \supseteq \mathsf{sid}^i_U$, there is no $\mathcal{O}_{Corrupt}(V)$ query.

The advantage of the adversary $\mathcal{A}$ in attacking the group communication scheme is defined as $\mathsf{Adv}_{\mathcal{A}}(\kappa) = |2 \cdot \Pr[\mathsf{Succ}] - 1|$, where $\Pr[\mathsf{Succ}]$ is the probability that the event Succ occurs, and the probability is taken over the coins used by $\mathcal{GC}$ and by $\mathcal{A}$. We say a scheme is a `secure` if for all probabilistic polynomial-time adversary $\mathcal{A}$, $\mathsf{Adv}_{\mathcal{A}}(\kappa)$ is negligible in κ.

Instantiations: We note that the authenticated private channels can be implemented using standard cryptographic techniques – this has been achieved in many specific schemes such as [48, 39] – except for the following: an adversary observing the communication channels may still be able to identify the membership of the non-compromised members. This can be resolved by letting the group controller publish information over a broadcast channel, or post it on a bulletin board to which all the group members can have anonymous access (e.g., via the standard MIX channels [21]). We stress that any centralized group key distribution scheme satisfying the above functionality and security requirements, such as [48, 47, 39], can be integrated into our secret handshake framework.

There is one other issue that needs to be addressed: we may need to ensure that the system update information published by the group controller does does not allow a passive adversary to discover the group identity, or even group dynamics. Although this may not be a real concern for many secret handshake applications (since a compromised group member has to know the identity of the group controller) we note that enhancement techniques suggested in [46] can be utilized to conceal the group dynamics of many popular centralized key distribution schemes, including [48, 47, 39].

D Security Properties of Distributed Group Key Agreement Schemes

We allow an adversary to have complete control over all the communications in the network. An adversary's interaction with the principals in the network (more specifically, with the various instances) is modeled by allowing it to have access to the following oracles:

- $\mathcal{O}_{Send}(U, i, M)$: Send message M to instance Π^i_U, and outputs the reply generated by this instance. Moreover, we even allow the adversary to prompt the unused instances Π^i_U to initiate the protocol with partners $U_2, \ldots, U_m$ by calling $\mathcal{O}_{Send}(U, i, \langle U_2, \ldots, U_m \rangle)$.
- $\mathcal{O}_{Execute}(U_1, \ldots, U_m)$: Execute the protocol between unused instances of players $U_1, \ldots, U_m \in \mathbb{ID}$ and outputs the transcript of the execution. The number of participants and their identities are chosen by the adversary.
- $\mathcal{O}_{Reveal}(U, i)$: Output session key sk^i_U, where $U \in \mathbb{ID}$.
- $\mathcal{O}_{Corrupt}(U)$: Output the long-term secret LLK_U of principal $U \in \mathbb{ID}$.
- $\mathcal{O}_{Test}(U, i)$: This query is only allowed once, at any time during the adversary's execution. A random bit b is generated; if $b = 1$ the adversary is given sk^i_U, and if $b = 0$ the adversary is given a random session key.

`Correctness`. We require that for all U, U', i, j such that:

$$\mathsf{sid}^i_U = \mathsf{sid}^i_{U'}, \mathsf{pid}^i_U = \mathsf{pid}^j_{U'} \mathsf{and} \mathsf{acc}^i_U = \mathsf{acc}^j_{U'} = \text{TRUE}$$

to be the case that: $\mathsf{sk}^i_U = \mathsf{sk}^j_{U'} \neq \mathsf{null}$.

`Security`. Intuitively, an adversary that does not compromise any principals during the execution does not learn any information about the new group session key. Formally, consider the following event Succ:

(1) The adversary queries $\mathcal{O}_{Test}(U, i)$ on instance Π^i_U for which $acc^i_U = \text{TRUE}$ and correctly guesses the bit b used by the oracle in answering this query.
(2) For any $\Pi^{i'}_{U'}$ such that $\mathsf{pid}^i_U = \mathsf{pid}^{i'}_{U'}$ and $\mathsf{sid}^i_U = \mathsf{sid}^{i'}_{U'}$, there is no $\mathcal{O}_{Reveal}(U, i)$ or $\mathcal{O}_{Reveal}(U', i')$ query.
(3) No query $\mathcal{O}_{Corrupt}(U')$ (with $U' \in \mathsf{pid}^i_U$) was asked before a query of the form $\mathcal{O}_{Send}(U', i', *)$. Note that, while (3) is not necessary in general, it is required in some specific schemes.

The advantage of the adversary $\mathcal{A}$ attacking the scheme DGKA is defined as: $\mathsf{Adv}_{\mathsf{DGKA},\mathcal{A}}(1^\kappa) = |2 \cdot \Pr[\mathsf{Succ}] - 1|$, where $\Pr[\mathsf{Succ}]$ is the probability that the event Succ occurs. We say a distributed key agreement scheme DGKA is *secure* if, for any probabilistic polynomial-time adversary $\mathcal{A}$, $\mathsf{Adv}_{\mathsf{DGKA},\mathcal{A}}(1^\kappa)$ is negligible in κ.

Instantiations: Any distributed group key agreement scheme satisfying the above requirements can be integrated into our framework. Popular schemes include [15, 45] and [13]. In particular, the scheme by Burmester and Desmedt [15] and its later variant [32] are efficient in computation, i.e., each participant needs to compute a constant number of modular exponentiations.

E Proof Sketch for Theorem 1

Proof. (SKETCH) `Correctness`. When all m parties belong to the same group, the handshake always succeeds; otherwise, the handshake succeeds only with a negligible probability (see `resistance to impersonation` below).

`Resistance to impersonation`. We observe that

$$\begin{aligned}\Pr[\mathsf{RIA}_{\mathsf{SHS},\mathcal{A}}(1^\kappa) \text{ returns "1"}] &= \Pr[\mathcal{A} \text{ provides a valid MAC tag} \wedge \mathcal{A} \text{ provides a valid group signature}]\\ &\leq \Pr[\mathcal{A} \text{ provides a valid MAC tag}]\\ &\leq \Pr[\mathcal{A} \text{ knows the MAC key}] +\\ &\quad \Pr[\mathcal{A} \text{ provides a valid MAC tag} | \mathcal{A} \text{ doesn't know the MAC key}].\end{aligned}$$

Suppose $\mathcal{A}$ breaks the `resistance to impersonation` property. This means that Pr[SHS.Handshake($\varDelta$) returns "1"] is non-negligible, and thus that either Pr[$\mathcal{A}$ knows the MAC key] or Pr[$\mathcal{A}$ provides a valid MAC tag|$\mathcal{A}$ doesn't know the MAC key] is non-negligible.

If Pr[$\mathcal{A}$ knows the MAC key] is non-negligible, then we can construct a polynomial-time algorithm $\mathcal{B}$ to break the CGKD scheme. Algorithm $\mathcal{B}$ operates as follows: It establish a simulated SHS environment as in the real-life system, except that the challenge CGKD environment is associated with a group G that is chosen uniformly at random, and $\mathcal{B}$ maintains the DGKA and GSIG with respect to G by itself. The simulated SHS environment operates exactly the same as in the real life system, except the following: Whenever $\mathcal{A}$ makes a query to the SHS environment with respect to group G, if the query cannot be answered by $\mathcal{B}$ based on its information regarding the DGKA and GSIG with respect to G, $\mathcal{B}$ makes the same queries to the challenge CGKD environment. Clearly, the simulated SHS environment is perfectly the same as in a real-life system. Since $\mathcal{A}$ has a non-negligible probability in learning a MAC key corresponding to group G, so is $\mathcal{B}$ because $\mathcal{B}$ also participates in DGKA.RawKeyAgreement. The fact that $\mathcal{A}$ does not corrupt any member when the MAC key is valid implies that $\mathcal{B}$ does not corrupt any member. Conditioned on the fact that there are polynomial many groups, we conclude that $\mathcal{B}$ breaks the challenge CGKD environment with a non-negligible probability.

If Pr[$\mathcal{A}$ provides a valid MAC tag|$\mathcal{A}$ doesn't know the MAC key] is non-negligible, then we can construct a polynomial-time algorithm $\mathcal{B}$ to break the MAC scheme. Algorithm $\mathcal{B}$ operates as follows: It establish a simulated SHS environment as in the real-life system, except that the challenge MAC scheme and an oracle scheme CGKD are associated with a group G that is chosen uniformly at random, and $\mathcal{B}$ maintains the DGKA and GSIG with respect to G by itself. The simulated SHS environment operates exactly the same as in the real life system, except the following: Whenever $\mathcal{A}$ makes a query to the SHS environment with respect to group G, if the query cannot be answered by $\mathcal{B}$ based on its information regarding the DGKA and GSIG with respect to G, $\mathcal{B}$ makes the same queries to the challenge MAC scheme or the oracle CGKD environment. Clearly, the simulated SHS environment is perfectly the same as in a real-life system. Since $\mathcal{A}$ has a non-negligible probability in forging, perhaps after seeing valid tags via $\mathcal{B}$ by having oracle access to the MAC scheme, a MAC tag corresponding to a key that is unknown to $\mathcal{A}$ (and thus unknown to $\mathcal{B}$), the forgery is also a successful attack by $\mathcal{B}$. Conditioned on the fact that there are polynomial many groups and there are polynomial many instances of SHS.Handshake, we conclude that $\mathcal{B}$ breaks the challenge MAC scheme with a non-negligible probability.

`Resistance to detection`. We notice that we already proved that Pr[$\mathsf{RDA}_{\mathsf{SHS},\mathcal{A}}(1^\kappa)$ returns "1"|bit $= 0$] is negligible. Therefore, if $\mathsf{AdvRDA}_{\mathsf{SHS},\mathcal{A}}$ is non-negligible, then Pr[$\mathsf{RDA}_{\mathsf{SHS},\mathcal{A}}(1^\kappa)$ returns "1"|bit $= 1$] is non-negligible. If this probability is non-negligible, then we can construct a simulator $\mathcal{B}$ to break the MAC scheme. Algorithm $\mathcal{B}$ operates as follows: It establish a simulated SHS environment as in the real-life system. Note that $\mathcal{B}$ has access to a challenge MAC environment. The simulated SHS environment operates exactly the same as in the real life system, except the following: When $\mathcal{A}$ incurs the $\mathsf{RDA}_{\mathsf{SHS},\mathcal{A}}$, $\mathcal{B}$ chooses a party $U' \in \varDelta'$ uniformly at random and provides its MAC tag via oracle query to the challenge MAC environment; for any other $U \in \varDelta' \setminus \{U'\}$, $\mathcal{B}$ chooses a MAC key uniformly at random. If $\mathcal{A}$ outputs a valid MAC tag with respect to the MAC tag of U', then this is also a successful forgery of $\mathcal{B}$ against the challenge MAC scheme; otherwise, $\mathcal{B}$ fails. Conditioned on the fact that $|\varDelta'|$ is polynomially bounded, $\mathcal{B}$ breaks the challenge MAC scheme with a non-negligible probability.

`Unlinkability`. Suppose an adversary can link two handshake sessions involving a same honest party. Without loss of generality, we assume that and both involve the same malicious party. There are two cases:

- CASE 1: Both sessions are successful handshakes. First, recall that no information in the transcript, except θ_i, contains any information specific to the identity of the honest party. If the adversary can assert that the same party is involved in the two instances, then we claim that `full-anonymity` of GSIG is violated. This is so because the adversary's decision can only be based upon the two group signatures produced by the same party.

  Specifically, we can construct a polynomial-time algorithm $\mathcal{B}$, which simulates the SHS environment as in the real life, except that $\mathcal{B}$ only has oracle access to a challenge group signature scheme GSIG. Whenever $\mathcal{A}$ makes a query to the SHS environment, if the query cannot be answered by $\mathcal{B}$ based on its information regarding the CGKD and DGKA $\mathcal{B}$ makes the same queries to the challenge GSIG scheme. Clearly, the simulated SHS environment is perfectly the same as in a real-life system. Finally, when $\mathcal{A}$ outputs the two handshake sessions involving the same person, $\mathcal{B}$ decrypt the corresponding θ's and output the pair of group signatures.
- CASE 2: At least one of them is an unsuccessful handshake. This is impossible since, in Phase III of the unsuccessful session, the honest party's output is wholely based on random simulations (i.e., independent of the honest party's secrets).

`Indistinguishability to eavesdroppers`. The only difference in the adversary's view is whether $\{(\theta_i, \delta_i)\}_{1 \leq i \leq m}$ are "real" or random, where δ_i is the symmetric encryption of group signature σ_i under the key $k'_i = k^*_i \oplus k_i$ and θ_i is the encryption of k_i' under $\mathcal{GA}$'s public key pk_T. (Note that this is indeed the so-called hybrid encryption). We observe that

$$\begin{aligned}
&\Pr[\mathsf{AdvINDeav}_{\mathsf{SHS},\mathcal{A}}(1^\kappa) \text{ returns "1"}] \\
&= \Pr[\mathcal{A} \text{ knows } k_i' \vee \mathcal{A} \text{ can distinguish encryption of group signature}] \\
&\le \Pr[\mathcal{A} \text{ knows } k_i'] + \Pr[\mathcal{A} \text{ can distinguish encryption of group signature}]
\end{aligned}$$

We claim that $\Pr[\mathcal{A}$ knows $k_i']$ is negligible; otherwise, such an adversary can immediately break the above proved `resistance to detection`. If $\Pr[\mathsf{AdvINDeav}_{\mathsf{SHS},\mathcal{A}}(1^\kappa)$ returns "1"] is negligible, so is $\Pr[\mathcal{A}$ can distinguish encryption of group signature], which is the probability that $\mathcal{A}$ can tell the encryption of a valid group signature ($\mathcal{A}$ knows the group public key in the worst case) from the encryption of a random string, even if $\mathcal{A}$ does not have any information about sk_T or k_i'. This immediately reduce to the security of the hybrid encryption scheme, namely that we can construct another polynomial-time algorithm to break either a challenge public key cryptosystem (with respect to T) or a challenge symmetric key cryptosystem (with respect to k_i').

`Traceability`. As long as the transcript includes the encryption of a valid session key k' under the $\mathcal{GA}$'s public key pk_T, $\mathcal{GA}$ can always recover k'. As a result, given any pair (θ_i, δ_i), $\mathcal{GA}$ is also able to recover the encrypted group signature, and therefore the identify of the actual signer. If there is a valid group signature that cannot be opened, then `full-traceability` with respect to GSIG is broken.

Specifically, we construct an algorithm $\mathcal{B}$ that simulates the SHS environment by having oracle access to a challenge group signature scheme GSIG. The execution of SHS is exactly the same as in a real-life system, except that whenever $\mathcal{A}$ presents a query against the GSIG, the query is forwarded by $\mathcal{B}$ to the challenge GSIG environment. Clearly, $\mathsf{AdvTraceUser}_{\mathsf{SHS},\mathcal{A}}$ is non-negligible means that $\mathcal{A}$ can provide a group signature in the $\mathsf{TraceUser}_{\mathsf{SHS},\mathcal{A}}$ experiment such that the signer is an innocent legitimate member. This is $\mathcal{B}$'s successful attack against the challenge GSIG environment.

`No-misattribution`. If an adversary can frame an honest member (link to a handshake instance) we claim that `no-misattribution` of group signature scheme is violated. Specifically, we construct an algorithm $\mathcal{B}$ which simulates a SHS environment by having oracle access to a challenge GSIG scheme. The simulated SHS environment executes exactly as in a real system, except that an oracle query provided by $\mathcal{A}$ (which cannot be directly answered by $\mathcal{B}$) is answered after $\mathcal{B}$ forwards the same query to the challenge GSIG environment. In particular, the group manager in the challenge GSIG environment can be corrupted. Clearly, if $AdvMisattribution_{\mathsf{SHS},\mathcal{A}}$ is non-negligible, then $\mathcal{B}$ violates `no-misattribution` of the challenge GSIG environment with the same probability.

`Forward-repudiability`. We claim that, if this property does not hold, neither the `full-anonymity` of the underlying GSIG. Specifically, we construct an algorithm $\mathcal{B}$, which simulates a SHS environment by having oracle access to a challenge GSIG scheme. The simulated SHS environment execute exactly the same as in a real-life system, except that an oracle query provided by $\mathcal{A}$, which cannot be directly answered by $\mathcal{B}$, is answered after $\mathcal{B}$ forwards the same query to the challenge GSIG environment. Note that the group manager in the challenge GSIG environment is never corrupted. Therefore, if $\mathcal{A}$ can convince an honest verifier that a group signature involved in a handshake session is produced by an honest member, then $\mathcal{B}$ breaks the `full-anonymity` of the challenge GSIG environment.

F Proof Sketch for Theorem 3

We now sketch the proof of `self-distinction`, as the proof of the other properties are similar to the proof of Theorem 1.

We observe that T_7 is the base common to all the participating parties and $T_6 = T_7^{x'}$, where x' is a member's secret. Suppose $\mathsf{AdvSelfDist}_{\mathsf{SHS},\mathcal{A}}$ is non-negligible, then $\mathcal{A}$ must be able to provide two group signatures with different x''s. Without loss of generality, suppose $\mathcal{A}$ knows (A, e, x, x') such that $A^e = a_0 a^x b^{x'}$. Denote by X the event that $\mathcal{A}$ holds (A, e, x, x') and $(A_\mathcal{A}, e, x_\mathcal{A}, x'_\mathcal{A})$ such that $A^e = a_0 a^x b^{x'} = (A_\mathcal{A})^e = a_0 a^{x_\mathcal{A}} b^{x'_\mathcal{A}}$. Note that

$$\begin{aligned}
&\Pr[\mathsf{SelfDist}_{\mathsf{SHS},\mathcal{A}}(1^\kappa) \text{ returns "1"}] \\
&= \Pr[\mathsf{SelfDist}_{\mathsf{SHS},\mathcal{A}}(1^\kappa) \text{ returns "1"}|X] \cdot \Pr[X] + \Pr[\mathsf{SelfDist}_{\mathsf{SHS},\mathcal{A}}(1^\kappa) \text{ returns "1"}|\neg X] \cdot \Pr[\neg X] \\
&\le \Pr[X] + \Pr[\mathsf{SelfDist}_{\mathsf{SHS},\mathcal{A}}(1^\kappa) \text{ returns "1"}|\neg X]
\end{aligned}$$

Since $\Pr[\mathsf{SelfDist}_{\mathsf{SHS},\mathcal{A}}(1^\kappa)$ returns "1"] is non-negligible, so is $\Pr[X]$ or $\Pr[\mathsf{SelfDist}_{\mathsf{SHS},\mathcal{A}}(1^\kappa)$ returns "1"$|\neg X]$. We claim that the `full-traceability` of the GSIG scheme is broken.

Specifically, we construct an algorithm $\mathcal{B}$, which simulates a SHS environment by having oracle access to a challenge GSIG scheme. The simulated SHS environment execute exactly the same as in a real-life system, except that an oracle query provided by $\mathcal{A}$, which cannot be directly answered by $\mathcal{B}$, is answered after $\mathcal{B}$ forwards the same query to the challenge GSIG environment. Note that the group manager in the challenge GSIG environment is never corrupted. Therefore, if $\mathcal{A}$ can provides two group signatures with $A^e = a_0 a^x b^{x'} = (A_\mathcal{A})^e = a_0 a^{x_\mathcal{A}} b^{x'_\mathcal{A}}$, $\mathcal{B}$ breaks the `full-traceability` of the challenge GSIG environment.

G Overview of the [33] Signature Scheme

For completeness, we now briefly overview the signature scheme of [33], which is an extension of the [2] group signature scheme. We summarize the parameters as:

- Two primes p', q' with $p = 2p' + 1$, $q = 2q' + 1$ also primes. The modulus is set to $n = pq$.
- The group manager selects $a, a_0, b, g, h \in_R QR(n)$.
- The group manager secret key is set to $p, q, \theta \in_R \mathbb{Z}_{p'q'}$.
- The system public key is n, a, a_0, b, y, g, h, where $y = g^{\theta} \mod n$.
- A member's private key is (A, e, x, x') such that $A^e = a_0 a^x b^{x'} \mod n$, where e is an appropriate prime number, x is an appropriate random number known also to the member, and x' is an appropriate random number known only to the member.

In order to sign a message, a group member computes:

$$T_1 = Ay^r,\ T_2 = g^r,\ T_3 = g^e h^r,\ T_4 = g^{xk},\ T_5 = g^k,\ T_6 = g^{x'k'},\ T_7 = g^{k'}$$

where r, k, k' are randomly selected from an appropriate interval. The signature is derived from the following proof of knowledge of appropriate x, x', e, r, h':

$$T_2 = g^r,\ T_3 = g^e h^r,\ T_2^e = g^{h'},\ T_5^x = T_4,\ T_7^{x'} = T_6,\ a_0 a^x b^{x'} y^{h'} = T_1^e.$$

Notice that (T_1, T_2) allows the group manager to open a group signature because of A, (T_4, T_5) allows one who knows x to trace the member's signatures, and (T_6, T_7) allows one to claim its signatures.

H Discussion

There are several practical issues that need to be addressed. First, if there is only a single group that uses a secret handshake scheme, an adversary can simply figure out that the handshake peers belong to that group. In fact, if a secret handshake scheme is implemented as a TLS or IKE cipher suite, then the parties will exchange a cipher suite designator that clearly shows that they wish to engage in a secret handshake. Second, in any secret handshake scheme, utilizing one-time or reusable credentials alike, it is assumed that there is no easy way to identify the party who sent or received a certain message; otherwise, it is easy for an adversary to discover who is interacting with whom. This assumption is also true in privacy-preserving authentication mechanisms [36, 1, 11, 22, 23, 43, 38]. Third, if an adversary observes that handshake participants continue communicating after finishing the handshake protocol, it can deduce that they belong to the same group. (This applies to any secret handshake scheme utilizing one-time or reusable credentials.)

The above issues can be mitigated by various means. First, it is reasonable to assume that there are many groups, as long as it is not illegal to conduct secret handshakes. Second, there may be settings where the identity (for the purpose of authentication) of a party is not directly derivable from the address that must appear in the clear in protocol messages. A common example is the case of mobile devices wishing to prevent an attacker from correlating their (changing) locations with the device's logical identity [36]. Furthermore, some form of anonymous communication could make it hard to decide exactly who is engaging in a secret handshake. Third, protection against traffic analysis (e.g., an adversary simply observing continued communication after a handshake) can be achieved by utilizing mechanisms such as steganographic techniques, or anonymous communication channels.

In summary, if all assumptions are satisfied, then our secret handshake schemes (as well as [4, 36, 1, 11]) can provide provable privacy-preserving authentication, whereby two (or in our case, more) participants authenticate each other's membership *simultaneously*. Otherwise, all schemes implement heuristic *best-effort* anonymity.