

A Flexible Framework for Secret Handshakes

(Multi-party Anonymous and Un-observable Authentication)

Gene Tsudik¹ and Shouhuai Xu²

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. This motivates research in cryptographic techniques offering built-in privacy. A secret handshake is a protocol whereby participants establish a secure, anonymous and unobservable communication channel only if they are members of the same group. This type of “private” authentication is a valuable tool in the arsenal of privacy-preserving cryptographic techniques. Prior research focused on 2-party secret handshakes with one-time credentials.

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 represents the first treatment of group (or *multi-party*) secret handshakes, thus providing a natural extension to the secret handshake technology. 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 framework for secret handshakes. The proposed **GCD** framework lends itself to many 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 thorough security analysis and illustrates two concrete framework instantiations.

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

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 have been developed, such as group signatures [1] and privacy-preserving trust negotiation [9,25].

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 and 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 a 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 This informal definition broadens the prior version [3] which limited secret handshakes to two parties.

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 [3,14,36]. Our framework represents the first result that supports truly *multi-party* secret handshakes. Moreover, our work is first to solve the problem of *partially-successful* secret handshakes.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 can easily impersonate any number of group members by simultaneously playing multiple roles in a handshake protocol.³ Thus, an honest participant may be fooled into making a wrong decision when the number of participating parties is a factor in the decision-making policy. We also note that self-distinction is trivial for 2-party secret handshakes. However, it becomes more challenging for handshakes of three or more, since the parties cannot simply expose their identities; otherwise, anonymity would be lost.

Second, in contrast with prior work [3,14] which relies on one-time credentials to achieve **unlinkability** – this 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 [36].

In addition, our framework has some interesting *flexibility* features. In particular, it is model-agnostic: 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

² A partially successful handshake occurs whenever not all parties engaged in a handshake protocol are members of the same group. For example, if 5 parties take part in a secret handshake 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. Our scheme achieves this desired goal.

³ This is reminiscent of the well-known Sybil attack [19], which is nevertheless different and not addressed in the present paper.

tailored to application needs and semantics (e.g., the two specific instantiations have two different sets of properties). Finally, it lends itself to many practical instantiations: we present two 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 two concrete instantiations in Section 8. Some practical issues are considered in Section 9 and related work is overviewed in Section 10. The paper concludes with the summary and future research directions. Due to space limitation, we placed some technical material into the full version of the present paper [32].

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 \mathbf{G} be a set of groups, where each group⁴ $G \in \mathbf{G}$ 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 in most privacy-preserving authentication schemes; otherwise, anonymity could be trivially compromised. However, we note that the fact that secret handshake protocols themselves rely on anonymous channels does *not* necessarily present a problem. This is because a typical secret handshake application would be in a wireless setting where all communication is done via *broadcast* which offers receiver anonymity as a “built-in” feature.⁵ (See Section 9 for further discussion of practical issues.)

⁴ We use “group” to refer to a set of users, unless explicitly stated otherwise.

⁵ This does not contradict our claim in Section 1 that wirelessness heightens privacy concerns. Although eavesdropping is easier in wireless networks, receiver anonymity is, at the same time, also easier to achieve in wireless (rather than in wired) networks.

Definition 1. A secret handshake scheme (SHS) consists of the following algorithms and protocols:

- 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, $CR\mathcal{L}$, which is originally empty. The cryptographic context is made public, while the $CR\mathcal{L}$ 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 $CR\mathcal{L}$ and a user identity U_i such that $U_i \in \mathcal{U}$ and $U_i \in G$. The output includes an updated $CR\mathcal{L}$ 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 $\Delta = \{U_1, \dots, U_m\}$ 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 $SHS.Handshake(\Delta)$, \mathcal{GA} outputs the identities of all m participants involved in the handshake, i.e., U_1, \dots, U_m .

We note that the 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 Section 9.) The definition also does not ensure any form of “agreement” in the sense of [20], since the adversary is assumed to have complete control over all communication, and can corrupt parties. This also explains why we only achieve a somewhat *weak* form of traceability.

Definition 2. Desired security properties are informally specified below (the formal treatment is deferred to [32]).

- * **Correctness:** *If all handshake participants $\{U_1, \dots, U_m\}$ belong to the same group, the protocol returns “1”; otherwise, the protocol returns “0”.*
- * **Resistance to impersonation:** *an 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 $\mathcal{A} \in G$. This remains to be true even if \mathcal{A} plays the roles of multiple participants.*
- * **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. This remains to be true even if \mathcal{A} plays the roles of multiple participants.*
- * **Full-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, and U has been corrupt. This remains to be true even if \mathcal{A} plays the roles of multiple participants.*
- * **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. This remains to be true even if \mathcal{A} plays the roles of multiple participants.*
- * **Indistinguishability to eavesdroppers:** *no adversary \mathcal{A} who does not participate in a handshake protocol can distinguish between a successful handshake between $\{U_1, \dots, U_m\} \subseteq G$ and an unsuccessful one, even if $\mathcal{A} \in G$.*
- * **Traceability:** *$\mathcal{G}\mathcal{A}$ can trace all users involved in the handshake session of a given transcript.*
- * **No-misattribution:** *no coalition of malicious parties (including any number of group members and the $\mathcal{G}\mathcal{A}$) is able to frame an honest member as being involved in a secret handshake.*
- * **Self-distinction:** *each participant is ensured that all the participants are distinct.*

Remark 1. If needed, our definitions of **resistance to impersonation** and **resistance to detection** can be naturally extended to capture the case when \mathcal{A} corrupts some group members but does not use their secrets in the subsequent handshake protocols.

We notice that for certain applications **full-unlinkability** may be desirable, while for certain other applications **unlinkability** and **self-distinction** may be desirable. In other words, the framework specifies the important properties, while leaving the decision on which subset of the properties to satisfy to the specific applications.

The flavor of **traceability** achieved in the framework is relatively weak since the protocol participant who is last to send out the values (to facilitate traceability) can always neglect to do so. However, we observe that this holds in other schemes, even in those based on one-time credentials [3,14]. The subtle issue is that the last sender could always use a “fake” token before other (honest) participants can verify its validity. This is inevitable because of the basic impossibility result in [20]. While there are some purely theoretical ways to mitigate this problem, we are interested in efficient (i.e., practical) solutions. Consequently, we are prepared to tolerate some unfairness, which, nevertheless, only exists

between *legitimate* users. As a result, the achieved traceability is still valuable for investigating activities of group members before they become corrupt.

3 Design Space

As mentioned earlier, the **GCD** framework is essentially a compiler that outputs a multi-party secret handshake scheme satisfying all desired properties specified 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 (broadcast encryption) 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 encryption. (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 (or even 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 [3]), 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.

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 undermines their version of the unlinkability property. (Unlinkability in group signatures is different from that in group secret handshakes; see Section 8.2.) 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 [12]. 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 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 signature⁶. 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**, consists of the following algorithms.

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 $(pk_{\mathcal{GM}}, sk_{\mathcal{GM}}) \leftarrow \text{Setup}(1^\kappa)$.

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

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 U . Their common output includes the user's unique membership public key pk_U , and perhaps some updated information that indicates the current state of the system. The user's output includes a membership secret key sk_U . This procedure may be denoted by $(pk_U, sk_U, certificate_U; pk_U, certificate_U) \leftarrow \text{Join}[U \leftrightarrow \mathcal{GM}]$, where $\text{Join}[U \leftrightarrow \mathcal{GM}]$ denotes an interactive protocol between U and \mathcal{GM} , $pk_U, sk_U, certificate_U$ is the output of U , and $pk_U, certificate_U$ 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_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_U, pk_U) and a message M , outputs a group signature σ of M . This procedure may be denoted by $\sigma \leftarrow \text{Sign}(pk_{\mathcal{GM}}, pk_U, sk_U, M)$.

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 $\text{TRUE}/\text{FALSE} \leftarrow \text{Verify}(pk_{\mathcal{GM}}, M, \sigma)$.

Open: an algorithm executed by the group manager \mathcal{GM} . It takes as input of a message M , a group signature σ , $pk_{\mathcal{GM}}$ 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_U and a proof) that allows anyone to identify the actual signer. This procedure may be denoted, without loss of generality, by $U \leftarrow \text{Open}(pk_{\mathcal{GM}}, sk_{\mathcal{GM}}, M, \sigma)$ if $\text{TRUE} \leftarrow \text{Verify}(pk_{\mathcal{GM}}, M, \sigma)$.

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 [4,7,23], we say a group signature scheme is **secure** if it satisfies the following three properties (see[32] 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.

In order to achieve secret handshakes of **self-distinction**, we may also adopt group signature schemes achieving a somewhat weaker privacy notion. Specifically, we can substitute the following weaker notion of **anonymity** for the above **full-anonymity**: (2') **anonymity** – no adversary can identify the actual signer of a group signature, as long as the actual signer's secret has *not*

been compromised. As we will see, our specific handshake scheme achieving **self-distinction** is based on the variant group signature scheme of [22], which fulfills the above **anonymity** rather than **full-anonymity**.

5 Building Block II: Centralized Group Key Distribution Scheme

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

Since a (stateful) group communication scheme is driven by “rekeying” events (because of joining or leaving operations below), it is convenient to treat the events occur at “virtual time” $t = 0, 1, 2, \dots$, because the group controller is able to maintain such an execution history. At time t , let $\Delta^{(t)}$ denote the set of legitimate group members, $k^{(t)} = k_{\mathcal{GC}}^{(t)} = k_{U_1}^{(t)} = \dots$ the group (or session) key, $K_{\mathcal{GC}}^{(t)}$ the set of keys held by \mathcal{GC} , $K_U^{(t)}$ the set of keys held by $U \in \Delta^{(t)}$, $\text{acc}_U^{(t)}$ the state indicating whether $U \in \Delta^{(t)}$ has successfully received the rekeying message. Initially, $\forall U \in \mathbb{I}\mathbb{D}, t \in \mathbb{N}$, set $\text{acc}_U^{(t)} \leftarrow \text{FALSE}$. We assume that \mathcal{GC} treat joining and leaving operation separately (e.g., first fulfilling the leaving operation and then immediately the joining one), even if the requests are made simultaneously. This strategy has indeed been adopted in the group communication literature.

To simplify the presentation, we assume that during system initialization (i.e., **Setup** described below) \mathcal{GC} can communicate with each legitimate member U through an *authenticated private* channel. In practice, this assumption can be implemented with a two-party authenticated key-exchange protocol. Further, we assume 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 $U \in \mathbb{I}\mathbb{D}$ through an *authenticated* channel.

A centralized group key distribution scheme (CGKD) is specified below. It is adopted from [35].

Setup: The group controller \mathcal{GC} generates a set of keys $K_{\mathcal{GC}}^{(0)}$, and distributes them to the current group members (that may be determined by the adversary), $\Delta^{(0)} \subseteq \mathbb{I}\mathbb{D}$, through the authenticated private channels. (If some users were corrupted before this setup procedure, we may let the adversary select the keys held by the corrupt users.) Each member $U_i \in \Delta^{(0)}$ holds a set of keys denoted by $K_{U_i}^{(0)} \subset K_{\mathcal{GC}}^{(0)}$, and there is a key, $k^{(0)}$ that is common to all the current members, namely $k^{(0)} \in K_{\mathcal{GC}}^{(0)} \cap K_{U_1}^{(0)} \cap \dots \cap K_{U_{|\Delta^{(0)}|}}^{(0)}$.

Join: This algorithm is executed by the group controller \mathcal{GC} at certain time t following a join request by a prospective member. (We abstract away the out-of-band authentication and establishment of an individual key for each new member). It takes as input: (1) a set of identities of current group members

– $\Delta^{(t-1)}$, (2) identities of newly admitted group member, $\Delta' \subseteq \mathbb{ID} \setminus \Delta^{(t-1)}$, (3) keys held by the group controller, $K_{\mathcal{GC}}^{(t-1)}$, and (4) keys held by group members, $\{K_{U_i}^{(t-1)}\}_{U_i \in \Delta^{(t-1)}} = \{K_{U_i}^{(t-1)} : U_i \in \Delta^{(t-1)}\}$.

It outputs updated system state information, including: (1) identities of new group members, $\Delta^{(t)} \leftarrow \Delta^{(t-1)} \cup \Delta'$, (2) new keys for \mathcal{GC} itself, $K_{\mathcal{GC}}^{(t)}$, (3) new keys for new group members, $\{K_{U_i}^{(t)}\}_{U_i \in \Delta^{(t)}}$, which are *somehow* sent to the legitimate users through the authenticated channels (depending on concrete schemes), (4) new group key $k^{(t)} \in K_{\mathcal{GC}}^{(t)} \cap K_{U_1}^{(t)} \cap \dots \cap K_{U_{|\Delta^{(t)}|}}^{(t)}$. Denote it by $(\Delta^{(t)}, K_{\mathcal{GC}}^{(t)}, \{K_{U_i}^{(t)}\}_{U_i \in \Delta^{(t)}}) \leftarrow \text{Join}(\Delta^{(t-1)}, \Delta', K_{\mathcal{GC}}^{(t-1)}, \{K_{U_i}^{(t-1)}\}_{U_i \in \Delta^{(t-1)}})$.

Leave: This algorithm is executed by the group controller \mathcal{GC} at time, say, t due to leave or revocation operation(s). It takes as input: (1) identities of previous group members, $\Delta^{(t-1)}$, (2) identities of leaving group members, $\Delta' \subseteq \Delta^{(t-1)}$, (3) keys held by the controller, $K_{\mathcal{GC}}^{(t-1)}$, and (4) keys held by group members, $\{K_{U_i}^{(t-1)}\}_{U_i \in \Delta^{(t-1)}}$.

It outputs updated system state information, including: (1) identities of new group members, $\Delta^{(t)} \leftarrow \Delta^{(t-1)} \setminus \Delta'$, (2) new keys for \mathcal{GC} , $K_{\mathcal{GC}}^{(t)}$, (3) new keys for new group members, $\{K_{U_i}^{(t)}\}_{U_i \in \Delta^{(t)}}$, which are *somehow* sent to the legitimate users through the authenticated channels (depending on concrete schemes), (4) new group key $k^{(t)} \in K_{\mathcal{GC}}^{(t)} \cap K_{U_1}^{(t)} \cap \dots \cap K_{U_{|\Delta^{(t)}|}}^{(t)}$. Denote it by $(\Delta^{(t)}, K_{\mathcal{GC}}^{(t)}, \{K_{U_i}^{(t)}\}_{U_i \in \Delta^{(t)}}) \leftarrow \text{Leave}(\Delta^{(t-1)}, \Delta', K_{\mathcal{GC}}^{(t-1)}, \{K_{U_i}^{(t-1)}\}_{U_i \in \Delta^{(t-1)}})$.

Rekey: This algorithm is executed by the legitimate group members at some time t , namely all $U_i \in \Delta^{(t)}$ where $\Delta^{(t)}$ is derived from a Join or Leave event. In other words, $U_i \in \Delta^{(t)}$ 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, U_i sets: (1) $\text{acc}_{U_i}^{(t)} \leftarrow \text{TRUE}$, (2) $K_{U_i}^{(t)}$, where $k_{U_i}^{(t)} \in K_{U_i}^{(t)}$ is supposed to be the new group key.

If the rekeying event is incurred by a Join event, every $U_i \in \Delta^{(t)}$ erases $K_{U_i}^{(t-1)}$ and any temporary storage after obtaining $K_{U_i}^{(t)}$. If the rekeying event is incurred by a Leave event, every $U_i \in \Delta^{(t)}$ erases $K_{U_i}^{(t-1)}$ and any temporary storage after obtaining $K_{U_i}^{(t)}$, and every *honest* leaving group member $U_j \in \Delta'$ erases $K_{U_j}^{(t-1)}$ (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 learns any information about a group key at time t_1 , even if there are corrupt users at time $t_2 > t_1$. This is the strongest notion, called *strong-security in the active outsider attack model* in [35] (somewhat surprisingly, existing popular group communication schemes do not

achieve this property, but many of them can be made secure in this sense without incurring any significant extra complexity [35]). We defer formal definition and discussions to [32].

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 below; it follows the results in [5] and [21].

Environment: Since each principal can take part in many runs of the **GroupKeyAgreement** protocol (described below), we denote an instance i of $U \in \mathbb{ID}$ as Π_U^i . Each instance Π_U^i is associated with variables $\text{acc}_U^i, \text{sid}_U^i, \text{pid}_U^i, \text{sk}_U^i$. Initially, $\forall U \in \mathbb{ID}$ and $i \in \mathbb{N}$, $\text{acc}_U^i \leftarrow \text{FALSE}$ and $\text{sid}_U^i, \text{pid}_U^i, \text{sk}_U^i \leftarrow \text{UNDEFINED}$.

GroupKeyAgreement: a protocol that performs distributed unauthenticated (or “raw”) group agreement between any set of $m \geq 2$ parties. 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: $\Pi_{U_1}^{i_1}, \dots, \Pi_{U_m}^{i_m}$, where $\{U_1, \dots, U_m\} \subseteq \mathbb{ID}$. If the execution of $\Pi_{U_j}^{i_j}$ is successful, it sets:

1. $\text{acc}_{U_j}^{i_j} \leftarrow \text{TRUE}$,
2. $\text{sid}_{U_j}^{i_j}$ as the session id of instance $\Pi_{U_j}^{i_j}$, namely a protocol-specific function of all communication sent and received by $\Pi_{U_j}^{i_j}$ (e.g., we can simply set $\text{sid}_{U_j}^{i_j}$ as the concatenation of all messages sent and received by $\Pi_{U_j}^{i_j}$ in the course of its execution),
3. $\text{pid}_{U_j}^{i_j}$ 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) $\text{sk}_{U_j}^{i_j}$ as the newly established session key.

Remark: We stress that this definition does not offer any *authentication*, i.e., it does not capture *authenticated group key agreement*. For example, the above definition can be satisfied (instantiated) with a straight-forward extension to any of the several group Diffie-Hellman protocols, such as BD or GDH [11,30]. Of course, we are aware that unauthenticated key agreement protocols are susceptible to man-in-the-middle (MITM) attacks; this is addressed later, through the use of our second building block – CGKD.

Informally speaking (see [32] for a formal definition), we require for a scheme to have **correctness** and **security**. **Correctness** means that all participants must obtain the same new session secret (key), and **security** means that a passive adversary – who does not compromise any principal during protocol 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**. This initializes a centralized group key distribution (broadcast encryption) 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.
- Note that no real group-specific setup is required for initializing the distributed group key agreement component – **DGKA**. We assume that there is a set of system-wide (not group-specific) cryptographic parameters for the **DGKA** scheme, e.g., all groups use the same group key agreement protocol with the same global parameters. (More on this below.)

GCD.AdmitMember: \mathcal{GA} executes **CGKD.Join** and **GSIG.Join**.

CGKD.Join results in a new group key and **GSIG.Join** causes an update to **GSIG** state information. The updated **GSIG** state information is encrypted under the new **CGKD** group key and distributed to all group members through an authenticated anonymous channel, e.g., posted on 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 update 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 **CGKD** as: k_1, \dots, k_m , respectively. Note that, if they belong to the same group, then $k_1 = \dots = k_m$.

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

Phase II: Preliminary Handshake: Each party publishes a tag $\text{MAC}(k'_i, s, i)$ corresponding to a message authentication code **MAC** (e.g., **HMAC-SHA1**), where s is a string unique to party i , e.g., the message(s) it sent in the **DGKA.GroupKeyAgreement** execution.⁷

Phase III: Full Handshake: There are two cases:

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

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 ; $\delta_i \leftarrow \text{ENC}(pk_T, k'_i)$.
2. Generate a group signature σ_i on 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 ; $\theta_i \leftarrow \text{SENC}(k'_i, \sigma_i)$.
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 ; $\sigma_i \leftarrow \text{SDEC}(k'_i, \theta_i)$.
 - 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 and publishes 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, \dots, 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 2. 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.

The following theorems are proved in [32].

Theorem 1. *Assume GSIG possesses the properties specified in Section 4, namely correctness, full-traceability, full-anonymity, and no-misattribution. Assume also that DGKA and CGKD are secure with respect to their corresponding definitions in Sections 5-6. Then, the GCD framework possesses the properties specified in Section 2, namely correctness, resistance to impersonation, resistance to detection, full-unlinkability, indistinguishability to eavesdroppers, traceability, and no-misattribution.*

Theorem 2. *Assume GSIG possesses the properties specified in Section 4, namely correctness, full-traceability, anonymity, and no-misattribution. Assume DGKA and CGKD are secure with respect to their corresponding definitions in Sections 5-6. Then, the GCD framework possesses the properties specified in Section 2, namely correctness, resistance to impersonation, resistance to detection, unlinkability, indistinguishability to eavesdroppers, traceability, and no-misattribution.*

Extension: A natural extension of the above framework can fulfill the aforementioned *partially-successful* secret handshakes, namely that all such $\Delta \subset \{1, \dots, m\}$ that consists of $|\Delta| > 1$ members of a same group can always succeed in their handshakes without incurring any extra complexity. Each participant i can immediately tell the Δ such that $i \in \Delta$ as i knows which message authentication tags are valid.

8 Two Concrete Instantiations

We now present two concrete secret handshake schemes. The first scheme employs “raw” (unauthenticated) contributory group key agreement, and the second scheme ensures that all handshake participants are distinct.

8.1 Example Scheme 1

This is a straight-forward instantiation of the **GCD** framework. We simply plug in unauthenticated group key agreement (DGKA) derived from any of [11,30,21], CGKD based on [34,26], and **GSIG** based on [1,12]. Theorem 1 immediately implies that this instantiation satisfies all properties specified in Section 2, excluding **self-distinction**.

Computational complexity for each party is the sum of the respective complexities incurred in each of the three building blocks. Note that, in an m -party handshake, each party only needs to compute $O(m)$ modular exponentiations in total. Moreover, the communication complexity is $O(m)$ per-user in number of messages.

8.2 Example Scheme 2

As mentioned above, the first instantiation does not offer self-distinction, i.e., some of the m parties in a handshake protocol could in fact be “played” by the same party. We now discuss the basic idea for attaining the self-distinction property. Naturally, neither group key agreement nor centralized key distribution (i.e., the CGKD and DGKA components) can provide self-distinction. Thus, we turn to group signatures to obtain it. However, group signature schemes do not natively offer self-distinction since it runs against one of their basic tenets, informally summarized as:

Given any two group signatures it should be impossible to determine with any certainty whether the same signer (or two distinct signers) generated both signatures

Nonetheless, the need for self-distinction in group signatures (not in secret handshakes) has been recognized prior to this paper. In particular, the work in [2] introduces the concept of *subgroup signatures* motivated by certain applications, such as anonymous petitions. (In an anonymous petition, t group members want to sign a document in a way that any verifier can determine with certainty that

all t signers are distinct.) The example technique for constructing sub-group signatures in [2] involves slight modifications to the underlying group signature scheme. This is very similar to what we need to achieve self-distinction in the proposed framework.

Unfortunately, we cannot use the example in [2] since it is based on a group signature scheme [13] which is inefficient and not provably secure. However, we can modify a more recent (as well as much more efficient and provably secure) group signature scheme by Kiayas and Yung [22]. In this scheme each group signature is accompanied by a pair: $\langle T_6 = g^{\beta\alpha}, T_7 = g^\alpha \rangle$, where β is the signer's secret, and α is the signer's randomness for this specific signature. This structure has a nice feature that T_7 serves only as an "anonymity shield" in the sense that the signer does not even need to prove the knowledge of α . Instead, it is crucial that $T_6 = T_7^\beta$. Intuitively, this allows us to obtain **self-distinction** if we let each handshake participant use the same T_7 , since they should all provide distinct T_6 's. This can be achieved by simply utilizing an idealized hash function [6] $\mathcal{H} : \{0, 1\}^* \rightarrow \mathbf{R}$ to the input of, for instance, the concatenation of all messages sent by the handshake participants.⁸ This ensures that, as long as there is at least one honest participant, the resulting T_7 is uniformly distributed over \mathbf{R} , and the security proof in [22] remain sufficient for our purposes.

We now present a scheme based on the modified version of [22]. In what follows we only illustrate the handshake protocol since it is the only distinctive feature of this scheme.

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

Phase I: Preparation: m parties jointly execute DGKA.GroupKeyAgreement.

Let the resulting keys be denoted as: k_1^*, \dots, k_m^* , respectively. (After a successful run $k_1^* = \dots = k_m^*$.) Then, each party computes $k'_i = k_i^* \oplus k_i$

Phase II: Preliminary Handshake: Each party publishes a pair $\text{MAC}(k'_i, s, i)$, where s is a string unique to i .

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. Encrypt k'_i under the public key pk_T to obtain ciphertext δ_i .
2. Generate a variant group signature σ_i on δ_i on s via GSIG.Sign , which is the same as in [22] except that T_7 is chosen using an ideal hash function as discussed above, and the *same* T_7 is common to all handshake participants. (We stress that **self-distinction** is obtained from requiring all participants to use the same T_7 which forces them to compute distinct T_6 values.)
3. Encrypt σ_i using a symmetric key encryption algorithm and key k'_i to obtain ciphertext $\theta_i \leftarrow \text{SENC}(k'_i, \sigma_i)$.

⁸ While it would suffice for \mathbf{R} to be $QR(n)$, what is needed in [22] is in fact that $g \in QR(n)$ and α is chosen from an appropriate interval, i.e., $\mathbf{R} \subset QR(n)$.

4. Publish (θ_i, δ_i) .
5. Upon receiving (θ_i, δ_i) , execute as follows:
 - (a) Obtain $\sigma_i \leftarrow \text{SDEC}(k'_i, \theta_i)$
 - (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 simulates the above execution of the Pederson protocol and then picks a pair of (θ_i, δ_i) randomly selected from the ciphertext spaces corresponding to the symmetric key and public key cryptosystems, respectively.

A proof sketch of the following theorem can be found in [32].

Theorem 3. *Assume that GSIG possesses the properties specified in Section 4, namely correctness, full-traceability, anonymity, and no-misattribution. Assume also that DGKA and CGKD are secure with respect to their corresponding definitions in Sections 5-6. Then, the above instantiation possesses the properties of correctness, resistance to impersonation, resistance to detection, unlinkability, indistinguishability to eavesdroppers, no-misattribution, traceability, and self-distinction specified in Section 2.*

Computational complexity in number of modular exponentiations (per-user) remains $O(m)$ and communication complexity (also per-user) in number of messages also $O(m)$, where m is the number of participants.

9 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 [24,8,16,17,28,27]. 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 [24]. 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 [3,24,8]) can provide provable privacy-preserving authentication, whereby two (or in our case, more) participants authenticate each other's membership *simultaneously*. Otherwise, all schemes attain heuristic or *best-effort* anonymity.

10 Related Work

The first secret handshake scheme [3] is based on the protocol of Sakai et al. [29], 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 [18] does not lend itself to solving the secret handshake problem; see [3]. The scheme in [3] 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 result is due to Castelluccia, et al. [14]. 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 result by [36] requires each player to be aware of the information of other groups and offers weaker anonymity (referred to as k -anonymity).

11 Conclusions and Future Work

To summarize, we present **GCD**— a flexible secret handshake framework. **GCD** is a compiler that can transform a group signature scheme, a centralized group key distribution scheme, and a distributed group key agreement scheme into a secure secret handshake scheme. As illustrated by three concrete examples, **GCD** lends itself to actual practical instantiations and offers several interesting new features. **GCD** avoids the use of one-time pseudonyms and, unlike prior techniques, supports handshakes among an arbitrary number of parties. Furthermore, **GCD** can be instantiated to support the important new property of **self-distinction** important in handshakes of more than two participants.

We believe that the work described in this paper is a *first* step towards achieving practical anonymous interactive multi-party authentication protocols. Much remains to be done. First, the **GCD** framework needs to be implemented and experimented with. Second, we have made no attempt to optimize the efficiency of the framework. Further investigation is clearly called for. Third, efficient constructions are needed for those settings where the **GCD** framework does not apply (because, e.g., the lack of a centralized group key distribution scheme).

References

1. G. Ateniese, J. Camenisch, M. Joye, and G. Tsudik. A practical and provably secure coalition-resistant group signature scheme. In *Proc. CRYPTO 2000*, pages 255–270. Springer, 2000. Lecture Notes in Computer Science No. 1880.
2. G. Ateniese and G. Tsudik. Some Open Issues and New Directions in Group Signatures. In *Financial Cryptography'99*. Lecture Notes in Computer Science No. 1880.
3. D. Balfanz, G. Durfee, N. Shankar, D. Smetters, J. Staddon, and H. Wong. Secret handshakes from pairing-based key agreements. In *24th IEEE Symposium on Security and Privacy*, May 2003.
4. M. Bellare, D. Micciancio, and B. Warinschi. Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In E. Biham, editor, *Advances in Cryptology - EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 614–629. Springer, 2003.
5. M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In *Proc. EUROCRYPT 2000*, pages 139–155. Springer, 2000. Lecture Notes in Computer Science No. 1807.
6. M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *First ACM Conference on Computer and Communications Security*, pages 62–73, Fairfax, 1993. ACM.
7. M. Bellare, H. Shi, and C. Zhang. Foundations of group signatures: The case of dynamic groups. Cryptology ePrint Archive, Report 2004/077, 2004. <http://eprint.iacr.org/>.
8. C. Boyd, W. Mao, and K. Paterson. Deniable authenticated key establishment for internet protocols.
9. R. Bradshaw, J. Holt, and K. Seamons. Concealing complex policies with hidden credentials. In *Proceedings of the 11th ACM conference on Computer and communications security (CCS'04)*, pages 146–157. ACM Press, 2004.
10. E. Bresson, O. Chevassut, and D. Pointcheval. Dynamic group Diffie-Hellman key exchange under standard assumptions. In L. R. Knudsen, editor, *Proc. of Eurocrypt 02*, volume 2332 of *LNCS*, page 321–336.
11. M. Burmester and Y. Desmedt. A secure and efficient conference key distribution system. In A. D. Santis, editor, *Proc. EUROCRYPT 94*, pages 275–286. Springer, 1994. Lecture Notes in Computer Science No. 950.
12. J. Camenisch and A. Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In M. Yung, editor, *Proc. CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 61–76.
13. J. Camenisch and M. Stadler. Efficient group signature schemes for large groups. In B.S. Kaliski Jr., editor, *Proc. CRYPTO 1997*, volume 1294 of *Lecture Notes in Computer Science*, pages 410–424. Springer-Verlag, 1997.

14. C. Castelluccia, S. Jarecki, and G. Tsudik. Secret handshakes from ca-oblivious encryption. In *Advances in Cryptology - ASIACRYPT 2004*, volume 3329 of *Lecture Notes in Computer Science*, pages 293–307. Springer, 2004.
15. D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24:84–88, Feb. 1981.
16. D. Chaum. Blind signatures for untraceable payments. In R. L. Rivest, A. Sherman, and D. Chaum, editors, *Proc. CRYPTO 82*, pages 199–203.
17. D. Chaum and E. V. Heyst. Group signatures. In D. W. Davies, editor, *Advances in Cryptology — Eurocrypt '91*, pages 257–265, Berlin, 1991. Springer-Verlag. Lecture Notes in Computer Science No. 547.
18. W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Trans. Inform. Theory*, IT-22:644–654, Nov. 1976.
19. J. Douceur. The sybil attack. In *Proceedings of the First International Workshop on Peer-to-Peer Systems (IPTPS'01)*, pages 251–260, 2002. Springer-Verlag.
20. M. Fischer, N. Lynch, and M. Patterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, 1985.
21. J. Katz and M. Yung. Scalable protocols for authenticated group key exchange. In D. Boneh, editor, *Proc. CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 110–125. Springer-Verlag, 2002.
22. A. Kiayias, Y. Tsiounis, and M. Yung. Traceable signatures. In C. Cachin and J. Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 571–589. Springer, 2004.
23. A. Kiayias and M. Yung. Group signatures: Provable security, efficient constructions and anonymity from trapdoor-holders. Cryptology ePrint Archive, Report 2004/076, 2004. <http://eprint.iacr.org/>.
24. H. Krawczyk. Sigma: The 'sign-and-mac' approach to authenticated diffie-hellman and its use in the ike-protocols. In D. Boneh, editor, *Proc. CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 400–425. Springer-Verlag, 2002.
25. N. Li, W. Du, and D. Boneh. Oblivious signature-based envelope. In *Proceedings of 22nd ACM Symposium on Principles of Distributed Computing (PODC)*, pages 182–189. ACM, 2003.
26. D. Naor, M. Naor, and J. Lotspiech. Revocation and tracing schemes for stateless receivers. In J. Kilian, editor, *Proc. CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 41–62. Springer-Verlag, 2001.
27. M. Naor. Deniable ring authentication. In M. Yung, editor, *Proc. CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 481–498. 2002.
28. R. Rivest, A. Shamir, and Y. Tauman. How to leak a secret. In *Advances in Cryptology-ASIACRYPT '2001*, pages 552–565.
29. R. Sakai, K. Ohgishi, and M. Kasahara. Cryptosystems based on pairing. In *Proceedings of the Symposium on Cryptography and Information Security (SCIS)*, 2002.
30. M. Steiner, G. Tsudik, and M. Waidner. Key agreement in dynamic peer groups. *IEEE Trans. on Parallel and Distributed Systems*, 11(8):769–780, 2000.
31. Y. Sun and K. Liu. Securing dynamic membership information in multicast communications. In *IEEE Infocom'04*.
32. G. Tsudik and S. Xu. A Flexible Framework for Secret Handshakes. Full version of the present paper (available from the authors).
33. D. Wallner, E. Harder, and R. Agee. Key management for multicast: Issues and architectures. Internet Draft, Sept. 1998.

34. C. Wong, M. Gouda, and S. Lam. Secure group communication using key graphs. *IEEE/ACM Transactions on Networking* (Preliminary version in SIGCOMM'98), 8, 2000.
35. S. Xu. On the security of group communication schemes based on symmetric key cryptosystems. In *Proceedings of the Third ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN'05)*, 2005.
36. S. Xu and M. Yung. k-anonymous secret handshakes with reusable credentials. In *Proceedings of the 11th ACM conference on Computer and communications security (CCS'04)*, pages 158–167. ACM Press, 2004.