# Group Secret Handshakes
# or
# Affiliation-Hiding Authenticated Group Key Agreement

Stanisław Jarecki, Jihye Kim, and Gene Tsudik
Computer Science Department
University of California, Irvine
{stasio, jihyek, gts}@ics.uci.edu

## Abstract

*Privacy concerns in many aspects of electronic communication trigger the need to re-examine – with privacy in mind – familiar security services, such as authentication and key agreement.*

*An* Affiliation-Hiding *Group Key Agreement (AH-AGKA) protocol (also known as* Group Secret Handshake*) allows a set of participants, each with a certificate issued by the same authority, to establish a common authenticated secret key. In contrast to standard AGKA protocols, an AH-AGKA protocol has the following privacy feature: If Alice, who is a member of a group $G$, participates in an AH-AGKA protocol, none of the other protocol participants* learn *whether Alice is a member of $G$, unless these participants are themselves members of group $G$. Such protocols are useful in suspicious settings where a set of members of a (perhaps secret) group need to authenticate each other and agree on a common secret key, without revealing their affiliations to outsiders.*

*In this paper we strengthen the prior definition of AH-AGKA so that the security and privacy properties are maintained under any composition of protocol instances. We also construct two novel AH-AGKA protocols secure in this new and stronger model under the RSA and Gap Diffie-Hellman assumptions, respectively. Each protocol involves only two communication rounds and few exponentiations per player (e.g., no bilinear map operations). Interestingly, these costs are essentially the same as those of the underlying (*unauthenticated*) group key agreement protocol. Finally, our protocols, unlike prior results, retain their security and privacy properties without the use of one-time certificates.*

Keywords: secret-handshakes, group key agreement, authenticated group key agreement, privacy, privacy-preserving authentication.

## 1 Introduction

A traditional authenticated group key agreement (AGKA) protocol is assumed to operate within the confines of a common Public Key Infrastructure (PKI). At the start, participants – who have no prior secrets in common – exchange their public key certificates (PKCs). This exchange leaks information; in particular, it always reveals a participant's public key certification authority (CA). However, exchange of credentials, such as PKCs, is part and parcel of any AGKA and it seems counter-intuitive to be concerned about information leakage. At the same time, in many applications, the identity of the certificate-issuing CA determines the certificate owner's *affiliation*. This is not an issue if affiliation by itself is not a sensitive attribute. However, in certain scenarios, affiliation must be kept private and protected from all unauthorized parties, most commonly, those with different affiliations. We consider two motivating examples.

- CIA agents often operate in hostile environments and their affiliation represents a closely-guarded secret. This is mandated by the rules of the agency. Therefore, if two or more CIA agents need to discover each other and establish a secure communication channel, affiliation-leaking information cannot be exchanged for fear of detection and unpleasant consequences.
- Federal air marshals routinely accompany civilian flights and are required to keep a very low profile, i.e., to blend in as much as possible. When two or more marshals in an airport (or any common vicinity) need to coordinate activities and set up a secure conference, they must do so in an unobservable and undetectable manner, i.e., their affiliations must be kept private.

In a two-party setting, affiliation hiding authentication schemes have been addressed in the past with so-called *secret handshake* protocols [1]. The initial work [1] introduced the notion of privacy in public key-based authentication schemes and proposed the first two-party secret handshake scheme based on bilinear maps and secure under the Gap Diffie-Hellman (GDH) assumption. A subsequent result by Castelluccia, et al. [9] developed a slightly more efficient secret handshake scheme secure under the Computational Diffie-Hellman (CDH) assumption. Both schemes can be used in two versions: If the players use one-time certificates, in addition to affiliation-hiding these protocols trivially attain a property of *unlinkability*, since in addition to not leaking their affiliations, any two instances of the same player cannot be linked with each other. If the players re-use their certificates, the protocols are affiliation-hiding but it's possible to trace multiple occurrences of the same party.

In this paper we consider affiliation hiding in a multi-party (two or more) setting, i.e. for Authenticated *Group* Key Agreement protocols (AGKA). We construct two practical Affiliation-Hiding AGKA protocols (AH-AGKA), wherein participants compute an authenticated common secret key as long as all participants have the same affiliation, i.e., possess certificates issued by the same CA. At the same time, in contrast to a standard AGKA, a party engaging in an AH-AGKA protocol is assured that its affiliation is revealed to only those other protocol participants that belong to the group governed by the same CA. Our protocols have similar properties as the two-party secret handshakes of [1, 9], i.e. they offer affiliation-hiding with standard re-usable certificates, and they can offer unlinkability only if the players use one-time certificates. They can also offer heuristic unlinkability, e.g if players limit the usage of one certificate based on their physical mobility.

Group (or multi-party) secret handshake protocols have been considered in prior work, notably [17] and [12]. In [17], Tsudik and Xu presented the first scheme supporting any number of protocol participants and reusable certificates. However, their approach assures that the participants in the AGKA protocol successfully compute a shared key only if their group revocation information is synchronized (in other words, only if each participant assumes the same revocation *epoch*).

Recently, Jarecki et al. [12] constructed a practical AH-AGKA protocol which avoids this synchronization assumption, based on the (unauthenticated) Burmester-Desmedt group key agreement protocol [7]. However, this AH-AGKA protocol is secure only with the use of one-time certificates.[1] Also, the model of security for AH-AGKA protocols considered in [12] is restricted to a *single instance* of an AH-AGKA protocol execution. Such a model makes sense if each protocol instance uses independent inputs, but it is insufficient in the standard PKI setting of re-usable certificates.

**Our contributions.** The contributions of this paper are as follows: First, we upgrade the notion of AH-AGKA in [12] with a more robust and thus more useful notion. The new notion assumes a standard PKI model of re-usable certificates and it is modeled on the standard – and very strong – notion for traditional AGKA protocols in [6, 14], which, in turn, comes from a long line of research on Authenticated 2-party Key Agreement protocols [3, 16, 8]. This upgraded security notion implies that each AH-AGKA protocol

[1] We want to point out that in addition to requiring more storage for group members, higher load on the issuing CA, and longer certificate revocation structures, a protocol that requires single-use certificates is vulnerable to *depletion attacks*, whereby the adversary repeatedly engages some user in the AH-AGKA protocol, thus depleting the latter's supply of one-time certificates.

session remains secure given arbitrary scheduling of protocol instances and any message-interleaving pattern between these instances, e.g., a man-in-the-middle attack. Also, the security of a protocol session is independent of the usage of keys produced by all *other* protocol sessions.

Second, we construct two AH-AGKA protocols that support standard re-usable certificates and satisfy the new strong notion of AH-AGKA security. Both protocols are implicitly-authenticated variants of the Burmester-Desmedt GKA protocol. These two protocols are secure under the RSA and the GDH assumptions, respectively, in the Random Oracle Model (ROM). (Moreover, the second protocol is secure also under the CDH assumption, but the security reduction from the CDH problem is weaker.) Each scheme involves only 2 communication rounds and few exponentiations per participant. From both communication and computation perspective, the protocol costs are the same as those of the *unauthenticated* Burmester-Desmedt group key agreement protocol [7] and lower than those of the (non affiliation-hiding) signature-based authenticated version of the Burmester-Desmedt protocol due to Katz and Yung [14]. Consequently, our protocols show that *Affiliation Hiding* for AGKA protocol can be achieved at essentially no additional cost. Note, however, that an AH-AGKA protocol guarantees success only if all participants are affiliated with the same CA, which is not the case in a standard AGKA. Moreover, we do not address perfect forward security in this paper.

Third, an independent consequence of our work is a variant of the Burmester-Desmedt GKA protocol which is secure (in ROM), although without perfect forward secrecy, even if the participants re-use their Diffie-Hellman key contributions. The standard Burmester-Desmedt GKA protocol is insecure unless each player uses a new contribution in every protocol instance. As a consequence of re-use of key contributions, this version of the Burmester-Desmedt protocol requires 2 exponentiations per player instead of 3.

**Organization.** The rest of this paper is organized as follows: Section 2 formally defines an AH-AGKA scheme and the desired security/privacy properties. Section 3 defines the cryptographic assumptions required by our constructions. Section 4 presents the RSA-based AH-AGKA protocol, and Section 5 presents the DH-based AH-AGKA protocol. The security proofs for the RSA-based scheme are given in detail, but because of space limitations we relegate the proofs of security for the DH-based scheme to the full version of this paper [13].

## 2 Affiliation-Hiding Authenticated Group Key Agreement: Model and Definitions

**Entities.** Our AH-AGKA model is based on the existing standard model for authenticated group key agreement protocols [6, 14]. The main difference is that the standard model assumes a global PKI where each entity has a private/public key-pair and a certificate issued by a CA which is part of the PKI. The PKI involves a *certification hierarchy*, where the integrity of the association between entities and their public keys is vouched by a chain of certificates all leading to some commonly trusted CA-s. In this model, it is assumed that certificates (which in many applications contain information about owners *affiliation*) are publicly available. In contrast, AH-AGKA protocols aim to protect affiliation privacy of the participants and certificates are kept private. Another distinctive feature of our model is its "flat" certification structure, i.e., certification hierarchies and chains are not allowed. There are only CA-s and entities certified by CA-s; there are no intermediate CA-s and no delegation of certificates.

An AH-AGKA *scheme* operates in an environment that includes a set of *users* $\mathcal{U}$ and a set of *groups* $\mathcal{G}$. Each group is administered by a CA responsible for creating the group, admitting entities as members and revoking membership. We assume upper bounds $m$ and $l$, respectively, on the total number of groups and the number of members in any given group, i.e., $|\mathcal{G}| \leq m$ and $|\mathcal{U}| \leq l$. We assume that each user can be a member of many groups. We denote the fact that user $U \in \mathcal{U}$ is a member of group $G \in \mathcal{G}$ as $U \prec G$. The main part of an AH-AGKA scheme is an AH-AGKA *protocol*, which is executed by any set of users $\Delta = \{U_1, ..., U_n\} \subseteq \mathcal{U}$, for any $n \geq 2$. (More on that below.) Hereafter, the term *group member* refers to a user who is a member of a particular group, whereas the terms *player* and *protocol participant* refer to a user who is currently taking part in some particular instance of an AH-AGKA protocol.

**Groups.** We note from the outset that the use of the term group is over-loaded in this paper. First, it denotes a set of users with the same affiliation (members of group $G$), i.e., with certificates issued by the same CA. Second, it refers to an ad-hoc group ($\Delta$) of AH-AGKA protocol participants who may or may not be all members of the same group $G$. We make the desired meaning unambiguous from the context. We use *protocol participants* or *set of players* when referring to the second meaning, and we use *group* only in the first meaning except when re-using the standard terminology of (Authenticated) *Group* Key Agreement, where the word *Group* refers to the set of players participating in an instance of the AGKE protocol.

**AH-AGKA Protocol.** Using this terminology, each player $U_i \in \Delta$ participating in an instance of the AH-AGKE protocol executes the protocol instructions on inputs a public key of some group $G \in \mathcal{G}$ s.t. $U_i \prec G$, and $U_i$'s certificate of membership in $G$. The purpose of the AH-AGKA protocol is for the players in $\Delta$ to establish an authenticated shared secret key as long as (1) each of them run the protocol on the same public key, i.e. the public key of the same group $G$, and (2) for each $U_i \in \Delta$ it holds that $U_i \prec G$. This key is secret and authenticated in the sense that it can be used for any subsequent secure communication, e.g., entity authentication or message encryption and/or authentication.

To avoid any misunderstanding, we stress that such protocol does not in general imply an efficient solution for an (affiliation-hiding) *group discovery* problem, where each participating player starts a protocol on a *set* of its certificates of membership in a *set* of groups, and the protocol succeeds, for example, as long as all the certificates are valid and all these sets have a non-empty union. In contrast, our AH-AGKA schemes are most practical in scenarios where each user is a member of at most one group. However, we stress that if a user is a member of many groups, this would affect execution efficiency (or robustness), but it would not affect *security* and *affiliation-hiding* of our schemes. Indeed, in the definitions that follow we assume w.l.o.g. that each user is a member of every group.

**Public Information and Network Assumptions.** In our environment, all groups $G \in \mathcal{G}$ are publicly known. Their CA public keys and certificate revocation lists (CRL-s) maintained by CA-s are publicly accessible. Before any group can be created, a common security parameter must be publicly chosen, and a public $\mathsf{Setup}$ procedure is executed on that parameter. The $\mathsf{Setup}$ procedure creates common cryptographic parameters which are used as inputs in all subsequent protocols. We stress that the $\mathsf{Setup}$ procedure does not need to be executed by a trusted authority: It can be executed by anyone, for example by one of the CAs, and everyone can verify the validity of its outputs.

We assume that communication between users and CA-s, i.e. the certificate issuance process and the CRL retrieval, are conducted over anonymous and authenticated channels. In practice, a user might communicate with the CA, e.g., while retrieving the most recent CRL for its group, over an anonymous channel such as TOR [11]. Alternatively, the CRL-s of all groups can be combined and stored at some highly-available site where they can be either retrieved in bulk (if small) or via some Private Information Retrieval (PIR) protocol, e.g., [10].

We assume that all communication within the AH-AGKA protocol takes place over a broadcast channel. We assume weak synchrony, i.e., the protocol executes in rounds. In practice, this assumption implies that the protocol is started by a broadcast message indicating the number of participants. Weak synchrony among the participants also assumes that the length of the time window assigned to each protocol round is large enough to accommodate clock skews and reasonable communication delays. The broadcast channel is *not* assumed to be authenticated. In fact, the broadcast channel is used for purely notational convenience since we make no assumptions about its reliability. Specifically, when a participant broadcasts a message, it could just as well send a copy of this message to every other participant over a point-to-point link. In our model, the adversary is assumed to have full control of the underlying network: it sees the messages broadcasted by each participant in a given round, and decides which messages will be *delivered* to each participant in that round. The adversary can delete, modify or substitute any message and it can choose to deliver different messages to different participants.

As a consequence of this model, the security and privacy (Affiliation Hiding) properties of our AH-AGKA protocols hold, by definition, given *any* adversarial interference in the protocol. However, we stress that we do not claim any *robustness* properties of our protocols, apart from the basic correctness, i.e. that the protocols succeeds if the players execute the protocols on matching inputs and there is no active adversarial interference in the protocol. Indeed, constructing AH-AGKA protocols which are robust against protocol interference is an open issue.

**Player Instances and Protocol Sessions.** In line with prior work [8, 6, 14], our model allows for multiple executions of the AH-AGKA protocol scheduled in an arbitrary way, each involving any set of participants. We model this in the usual way, by assuming that every user $U \in \mathcal{U}$ can run multiple *instances* of the protocol. We denote the $\tau$-th instance of user $U$ as $\Pi_U^\tau$. When player $U$ starts a new instance of the AH-AGKA protocol, it creates a new instance $\Pi_U^\tau$ for a locally unique value $\tau$. Such instances can run on shared state, e.g., certificates and CRLs held by player $U$, but each instance also keeps separate state. Each player instance can either reject or accept and output a key. We say that an instance $\Pi_U^\tau$ runs a *protocol session*, and we use *player instance* and *protocol session* interchangeably, denoting both as $\Pi_U^\tau$. When referring to a specific user $U_i$ we use $\Pi_i^\tau$ as a short-hand version of $\Pi_{U_i}^\tau$, to denote $\tau$-th instance of user $U_i \in \mathcal{U}$. Each instance $\Pi_i^\tau$ keeps a state variable, $\mathsf{sid}_i^\tau$ which can be thought of as a *session id*. (However, see the remark below.) This variable is protocol-dependent, but in our protocols it is always set to an entirety of the communication sent and received by instance $\Pi_i^\tau$.

**AH-AGKA Syntax.** We define an AH-AGKA scheme as a collection of the following algorithms:

- $\mathsf{Setup}$: on input of security parameter $\kappa$, it generates public parameters $\mathsf{params}$.
- $\mathsf{KGen}$: executed by the group CA, on input $\mathsf{params}$, it outputs the group public key $\mathcal{PK}$ and the corresponding secret key $\mathcal{SK}$ for this group, and an empty certificate revocation list $\mathcal{CRL}$. We denote the group corresponding to the public key $\mathcal{PK}$ as $\mathsf{Group}(\mathcal{PK})$. If $\mathcal{PK}$ was generated by the CA that maintains group $G$ then $\mathsf{Group}(\mathcal{PK}) = G$.
- $\mathsf{Add}$: executed by the CA of group $G$, on input $\mathcal{SK}$ and $U \in \mathcal{U}$, it adds $U$ to $G$ by generating a certificate for $U$, denoted $\mathsf{cert}$. If $\mathsf{cert}$ is issued under a public key $\mathcal{PK}$, we say that $\mathsf{cert} \in \mathsf{Certs}(\mathcal{PK})$.
- $\mathsf{Revoke}$: executed by the group CA, on input $U \in \mathcal{U}$, it retrieves the corresponding $\mathsf{cert} \in \mathsf{Certs}(PK)$ issued for $U$, and revokes it by adding a new entry to the group $\mathcal{CRL}$ which uniquely identifies $\mathsf{cert}$. If $\mathsf{cert}$ is revoked, we say that $\mathsf{cert} \in \mathsf{RevokedCerts}(\mathcal{CRL})$.
- $\mathsf{Handshake}$: this is the AH-AGKA *protocol* itself, which is an interactive protocol executed by some set of participants $\Delta = \{U_1, ..., U_n\} \subseteq \mathcal{U}$. Each $U_i$ uses its distinct new instance $\Pi_i^\tau$ and runs session $\Pi_i^\tau$ of the protocol on some inputs:
$$(\mathsf{cert}_i^\tau, \mathcal{PK}_i^\tau, \mathcal{CRL}_i^\tau)$$
where $\mathcal{PK}_i^\tau$ is the public key of the group which, in $U_i$'s view, sets the context for the protocol, $\mathsf{cert}_i^\tau$ is $U_i$'s certificate in $\mathsf{Group}(PK_i^\tau)$, and $CRL_i^\tau$ is the CRL of this group.[2] An instance $\Pi_i^\tau$ either rejects or outputs an authenticated secret key $K_i^\tau$.

**Remark:** Our syntax, though adopted from earlier AGKA models of [6, 14], is slightly different from that used in some other work on Key Agreement protocols, e.g., [8], where a protocol instance takes as additional input, a so-called *session-id* (different from the $sid_i^\tau$ value introduced above). In this alternative model, creation of a fresh and locally-unique session-id's, common to all players engaging in the protocol,

[2] As in standard authentication protocols in the PKI model, the more recent CRL, the better. However, we do not assume that a player has the most recent group CRL.

is assumed to be done before the protocol starts. In contrast, in the model of [6, 14], which we adopt, no such agreed-upon value is assumed. (However, our protocols, similarly to the AGKA protocol in [14], in the first protocol round create a value $s$ which plays the role of such unique session-id input. As a side remark, we point out that unlike the protocol of [14], our AH-AGKA protocol manages to piggyback the creation of this session-id $s$ onto the first round of the protocol, thus saving one communication round.)

**Partnering.** The purpose of the Handshake protocol is to allow a set of participants with *matching inputs*, i.e. specifying the same group $G$, to establish a common key. We use the term *session partnering* to denote protocol instances that run on matching inputs and where all protocol messages between them are properly delivered. Namely, we say that a set of protocol instances $\{\Pi_1^{\tau_1}, \Pi_2^{\tau_2}, ..., \Pi_n^{\tau_n}\}$ is *partnered* if there exists a single public key $\mathcal{PK}$ and a single value $\mathsf{sid}$ such that, for each session $\Pi_i^{\tau_i}$, in this set it holds that $\mathcal{PK}_i^{\tau_i} = \mathcal{PK}$ and $\mathsf{sid}_i^{\tau_i} = \mathsf{sid}$. The latter implies complete agreement among these player instances with regard to the set of messages sent and delivered between these instances.

**Correctness.** We say that an AH-AGKA scheme is *correct* if, assuming that all keys, certificates and CRL-s are generated by following the Setup, KGen, Add and Revoke procedures, the following holds:

> For any set of *partnered* sessions $\Pi_1^{\tau_1}, \Pi_2^{\tau_2}, ..., \Pi_n^{\tau_n}$ where $\mathsf{cert}_i^{\tau_i} \in \mathsf{Certs}(\mathcal{PK}_i^{\tau_i})$ for each $i$, and $\mathsf{cert}_i^{\tau_i} \notin \mathsf{RevokedCerts}(CRL_j^{\tau_j})$ for all pairs $(i, j)$, there exists a single unique bit-string $K$ of length $\kappa$ such that each session $\Pi_i^{\tau_i}$ accepts and outputs $K_i^{\tau_i} = K$.

We define AH-AGKA security similarly to standard AGKA protocols in the PKI model, but we must adapt these security notions to our setting. In the setting of an AH-AGKA scheme, the protocol participants, instead of recognizing one another by individual public keys, want to establish authenticated sessions with *any* other participants as long as all these participants are non-revoked members of the same group. This is reflected in the fact that a user starts an AH-AGKE protocol instance on just his certificate and the public key of some chosen group $G$. One implication this bears for the AH-AGKA security definition is that, unlike in a standard AGKA protocol in the PKI model, our notion of security must explicitly include admission and revocation actions of the CA's which manages the groups.

AH-AGKA security is defined via a game between an adversary and a set of users communicating over a network. In this game, the adversary gets to see the public keys of all groups, and some number of certificates in each group, corresponding to all corrupted players and leaked secrets. The adversary then schedules any number of Handshake protocol instances, involving any combination of honest users and groups. The adversary has complete control of the network, i.e., it sees all messages and can delay, delete, modify, or inject any messages received by the honest players. The adversary can also request that some key established in some protocol session be *revealed*. We say that the AGKA protocol is **secure** if, for each (unrevealed) session executed by an honest player, the adversary cannot distinguish the key output by the player on that session from a random bitstring of the same length. (As discussed below, the only exception is if the adversary previously requested that a key be revealed for some protocol session *partnered* with the one at hand.)

Formally, security is defined via an interaction of an adversarial algorithm $\mathcal{A}$ and a challenger $\mathcal{C}$ on common inputs $(\kappa, l, m)$. The interaction starts with $\mathcal{C}$ generating $\mathsf{params}$ via $\mathsf{Setup}(\kappa)$, and initializing $m$ groups $G_1, ..., G_m$, by running the $\mathsf{KGen}(\mathsf{params})$ algorithm $m$ times. $\mathcal{C}$ initializes all members in these groups, by running the $\mathsf{Add}(\mathcal{SK}_j)$ algorithm, for each $\mathcal{SK}_j$, $j = 1, ..., m$, for $l$ times. This way, $\mathcal{C}$ generates $m$ certificates for every $U \in \mathcal{U}$, thus making every user a member of every group. The adversary $\mathcal{A}$ gets all generated public keys $\mathcal{PK}_1, ..., \mathcal{PK}_m$. It then chooses any subset $\mathsf{Rev} \subseteq \mathcal{U}$ of initially corrupted players and gets the set of their certificates $\{\mathsf{cert}_i{}^{(j)} \mid U_i \in \mathsf{Rev}, j \in \{1, ..., m\}\}$. For each group $G$ in $\mathcal{G}$, the challenger runs the Revoke algorithm to revoke all corrupted members $U \in \mathsf{Rev}$, and outputs the resulting CRL-s for each group, i.e., $\mathcal{CRL}_1, ..., \mathcal{CRL}_m$.

After this initialization, $\mathcal{A}$ schedules any number of Handshake protocols, arbitrarily manipulates their messages, requests the keys on any number of the (accepting) sessions, and optionally corrupts any number of additional players, all of which can be modeled by $\mathcal{A}$ issuing any number of the commands listed below. Finally, $\mathcal{A}$ stops and outputs a single bit $b'$. The commands the adversary can issue, and the way the challenger $\mathcal{C}$ responds to them, are listed below. In all commands we assume that $U \in \mathcal{U} \setminus \mathsf{Rev}$.

- $\mathsf{Start}(U, G)$: If $U = U_i$ for some $U_i \in \mathcal{U} \backslash \mathsf{Rev}$ and $G = G_j$ for some $G_j \in \mathcal{G}$, the challenger retrieves key $\mathcal{PK}_j$ for group $G_j$, certificate $cert_i{}^{(j)}$ issued to player $U_i$ for group $G_j$, and the $\mathcal{CRL}_j$ for group $G_j$, and initiates instance $\Pi_U^\tau$, where $\tau$ is an index that has not been used by user $U$ before. The challenger follows the Handshake protocol on behalf of instance $\Pi_U^\tau$ on inputs $(\mathsf{cert}_i{}^{(j)}, \mathcal{PK}_j, \mathcal{CRL}_j)$, forwarding any message generated by this instance to $\mathcal{A}$. The challenger keeps the state of all initiated instances. We denote the group upon which $\Pi_U^\tau$ is initiated as $\mathsf{Group}(\Pi_U^\tau)$. If $\Pi_U^\tau$ is triggered on $G_j$ then $\mathsf{Group}(\Pi_U^\tau) = G_j$. $\mathcal{C}$ also hands to $\mathcal{A}$ the index $\tau$ of this instance.

- $\mathsf{Send}(U, \tau, \mathcal{M})$: If instance $\Pi_U^\tau$ has been initiated and is still active, $\mathcal{C}$ delivers a *set* $\mathcal{M}$ of messages to this instance. The set $\mathcal{M}$ should normally contain $n - 1$ messages $M_2, ...M_n$, for $n \geq 2$, which models the messages that instance $\Pi_U^\tau$ receives in the current round of this protocol. The instance interprets these messages as broadcasted by $n - 1$ distinct instances of the protocol in the same round. ($\mathcal{A}$ could send an empty set $\mathcal{M}$, but an instance would invariable immediately abandon the protocol as a result.) $\mathcal{C}$ forwards to $\mathcal{A}$ any message $\Pi_U^\tau$ generates in response. If $\Pi_U^\tau$ outputs a key, $\mathcal{C}$ stores this key with the session state.

- $\mathsf{Reveal}(U, \tau)$: If $\Pi_U^\tau$ outputs a session key $K$, $\mathcal{C}$ sends $K$ to $\mathcal{A}$. If the session has either not completed yet or has been rejected, $\mathcal{C}$ sends $\mathcal{A}$ a null value.

- $\mathsf{Test}(U, \tau)$: This query is allowed only once. If session $\Pi_U^\tau$ has output a session key $K$, $\mathcal{C}$ picks a random bit $b$. If $b = 1$, then $\mathcal{C}$ sends $K$ to $\mathcal{A}$. If $b = 0$ then $\mathcal{C}$ sends to $\mathcal{A}$ a random $\kappa$-bit long value $K'$, instead of $K$. If the session does not exist, failed, or is still active, the challenger ignores this command.

**Session Freshness and Legitimate Adversaries.** We call an active session $\Pi_U^\tau$ of an uncorrupted player $U$ *fresh*, if for all sessions $\Pi_{U'}^{\tau'}$ partnered with $\Pi_U^\tau$ the adversary has not queried $\mathsf{Reveal}(U, \tau)$ or $\mathsf{Reveal}(U', \tau')$. Note that the adversary knows whether any two sessions are partnered or not. We call an adversary $\mathcal{A}$ *legitimate* if it poses a Test query on a fresh session $\Pi_i^\tau$, and afterwards $\mathcal{A}$ does not issue a Reveal query on $\Pi_U^\tau$ or any $\Pi_{U'}^{\tau'}$ partnered with $\Pi_U^\tau$.

**Definition 1.** *Denote the final output of adversary $\mathcal{A}$ in the above interaction with the challenger $\mathcal{C}$ on common inputs $(\kappa, l, m)$ as $\langle \mathcal{A}, \mathcal{C}(b) \rangle(\kappa, l, m)$. We define the adversary's advantage in the security game as*

$$\mathsf{Adv}_{\mathcal{A}}^{\text{sec}} = |\Pr[b = b' \mid b' \leftarrow \langle \mathcal{A}, \mathcal{C}(b) \rangle(\kappa, l, m)] - 1/2|$$

*where the probability is taken over the random coins used by $\mathcal{A}$ and $\mathcal{C}$ and a random choice of the challengers bit $b$.*

*We call an AH-AGKA scheme $(\epsilon, t, q_s, q_H, l, m)$*-secure *in the Random Oracle Model if for all legitimate adversaries $\mathcal{A}$ who run in time $t$, start $q_s$ AH-AGKA sessions, and make $q_H$ hash function queries, it holds that $\mathsf{Adv}_{\mathcal{A}}^{\text{sec}} \leq \epsilon$.*

We define the affiliation-hiding property using a similar game as in the security definition in the previous section. However, the adversary's goal in the affiliation-hiding game is not to violate semantic security of some session key (as in the security game above) but to learn the participants' affiliation. We model the property of the attacker's *inability* to learn the affiliation by comparing two executions of the adversary: one where the challenger follows the protocols faithfully on behalf of all honest participants, and the other where the adversary interacts with a *simulator*, instead of the real users. The simulator attempts to follow the adversary's instructions, except that it is never told the groups for which the (scheduled by the adversary) $\mathsf{Handshake}$ protocol instances are executed, i.e., if the adversary issues a $\mathsf{Start}(U, G)$ query, the simulator gets only an identifier $(\hat{id})$ which is uniquely but arbitrarily assigned to the pair $(U, G) \in \mathcal{U} \times \mathcal{G}$.

Consequently, these inputs are also the only thing that the adversary can possibly learn from the interaction with a simulator. The simulated protocol messages can reveal only whether or not two sessions involve *the same* (user,group) pair. However, the adversary does not learn which group, nor can he decide if two instances of two different users belong to the same group. Note that we allow the adversary to be able to *link* instances which involve the same (user,group) pair because the simulator gets the same $\hat{id}$ for such instances. Indeed, all AH-AGKA schemes we propose in this paper are linkable in this sense.

Formally, the game between $\mathcal{A}$ and $\mathcal{C}^{\mathsf{ah}}$, on common inputs $\kappa, l, m$, starts exactly as the game between $\mathcal{A}$ and $\mathcal{C}$ in the security definition above. Namely, $\mathcal{C}^{\mathsf{ah}}$ runs $\mathsf{Setup}(\kappa) \rightarrow \mathsf{params}$, then runs $m$ instances of $\mathsf{KGen}(\mathsf{params}) \rightarrow (\mathcal{PK}_j, \mathcal{SK}_j)$, for $j = 1, .., m$, then $lm$ instances of the $\mathsf{Add}$ algorithm, $\mathsf{Add}(\mathcal{SK}_j) \rightarrow \mathsf{cert}_i{}^{(j)}$, for $i = 1, ..., l$ and $j = 1, ..., m$, which generate $l$ certificates for each of the $m$ groups. $\mathcal{C}^{\mathsf{ah}}$ gives to $\mathcal{A}$ all public keys $\mathcal{PK}_j$ and the certificates of all corrupted users: $\{\mathsf{cert}_i{}^{(j)} \mid i \in \mathsf{Rev}, j \in \{1, ..., m\}\}$, revokes all of these certificates, and finally publishes the resulting CRL-s.

After this initialization, $\mathcal{A}$ schedules any number of $\mathsf{Handshake}$ instances and manipulates their messages in arbitrary ways. We model this interaction between $\mathcal{A}$ and $\mathcal{C}^{\mathsf{ah}}$ by allowing $\mathcal{A}$ any number of queries $\mathsf{Start}(U, G)$ and $\mathsf{Send}(U, \tau, \mathcal{M})$ to $\mathcal{C}$, as in the security game.

However, $\mathcal{A}$ does not make a $\mathsf{Test}$ query in this game. Instead, $\mathcal{C}^{\mathsf{ah}}$ picks a random bit $b$ at the beginning of the execution and performs $\mathcal{A}$'s commands depending on the value of $b$. If $b = 0$, $\mathcal{C}^{\mathsf{ah}}$ responds to $\mathcal{A}$'s commands $\mathsf{Start}(U, G)$ and $\mathsf{Send}(U, s, \mathcal{M})$ by following the corresponding protocol on behalf of the user, exactly as in the security game in above. Otherwise ($b = 1$), $\mathcal{C}^{\mathsf{ah}}$ replies to $\mathcal{A}$ with messages produced by the simulator $\mathcal{SIM}$, which is an interactive machine which runs only on inputs $\mathsf{params}$, and, instead of $\mathsf{Start}(U, G)$ and $\mathsf{Send}(U, s, \mathcal{M})$, it gets on-line inputs $\mathsf{Start}(\hat{id})$ and $\mathsf{Send}(\hat{id}, \mathcal{M})$, respectively, where $\hat{id}$ is a unique (and random) string assigned to this $(U, G)$ pair. At the end of the game, the adversary outputs a bit $b'$.

**Definition 2.** *Denote the final output of adversary $\mathcal{A}$ in the above interaction with the challenger $\mathcal{C}^{\mathsf{ah}}$ on common inputs $(\kappa, l, m)$ as $\langle \mathcal{A}, \mathcal{C}^{\mathsf{ah}}(b) \rangle(\kappa, l, m)$. We define the adversarial advantage in the affiliation-hiding game as*

$$\mathsf{Adv}^{\mathsf{ah}}_{\mathcal{A}}(\kappa, l, m) = |\Pr[b = b' \mid b' \leftarrow \langle \mathcal{A}, \mathcal{C}^{\mathsf{ah}}(b) \rangle(\kappa, l, m)] - 1/2|$$

*where the probability is taken over the random coins used by $\mathcal{A}$ and $\mathcal{C}^{\mathsf{ah}}$ and a random choice of the challengers bit $b$.*

*We call an AH-AGKA scheme* affiliation-hiding *if for any probabilistic polynomial-time adversary $\mathcal{A}$, for parameters $l$ an $m$ polynomially related to $\kappa$, the adversarial advantage $\mathsf{Adv}^{\mathsf{ah}}_{\mathcal{A}}(\kappa, l, m)$ is a negligible function of $\kappa$.*

**Remark on the Affiliation-Hiding Notion.** First, note that the above definition restricts $\mathcal{A}$ to only $\mathsf{Start}$ and $\mathsf{Send}$ queries. This results in a restricted notion of affiliation-hiding, which can and should be strengthened to include the information the $\mathcal{A}$ can gain about session keys from higher-level protocols, modeled by

Reveal queries. Such strengthening is in fact necessary in practice because without the Reveal queries, $\mathcal{A}$'s view does not even contain information on whether a given session instance failed or succeeded, which is something that a network adversary can very often learn in practice. We leave consideration of stronger notions of affiliation-hiding to the full version of the paper. Second, an exact-security version of the above notion can be easily extrapolated, and this too will be included in the full version, together with the exact security bounds on affiliation-hiding for the two AH-AGKA schemes presented in this paper.

## 3 Cryptographic Assumptions

**Definition 3.** *Let* $\mathsf{S\text{-}RSA\text{-}IG}(\kappa)$ *be an algorithm that outputs so-called safe RSA instances, i.e. pairs* $(n, e)$ *where* $n = pq$*,* $e$ *is a small prime that satisfies* $gcd(e, \phi(n)) = 1$*, and* $p, q$ *are randomly generated* $\kappa$*-bit primes subject to the constraint that* $p = 2p' + 1$*,* $q = 2q' + 1$ *for prime* $p', q'$*,* $p' \neq q'$*.*

*We say that the RSA problem is* $(\epsilon, t)$*-hard on* $2\kappa$*-bit safe RSA moduli, if for every algorithm* $\mathcal{A}$ *that runs in time* $t$ *we have*

$$\Pr[(n, e) \leftarrow \mathsf{S\text{-}RSA\text{-}IG}(\kappa), g \leftarrow \mathbb{Z}_n^* : \mathcal{A}(n, e, g) = z \text{ s.t. } z^e = g \pmod{n}] \leq \epsilon.$$

**Definition 4.** *Let* $\mathbb{G}$ *be a cyclic group of prime order* $q$ *with a generator* $g$*. We say that the Square Diffie-Hellman Problem (SDH) in* $\mathbb{G}$ *is* $(\epsilon, t)$*-hard if for every algorithm* $A$ *running in time* $t$ *we have*

$$\Pr[x \leftarrow \mathbb{Z}_q : A(g, g^x) = g^{x^2}] \leq \epsilon.$$

**DDH oracle:** A DDH oracle in group $\mathbb{G}$ is an algorithm that returns 1 on queries of the form $(g, g^x, g^y, g^z)$ where $z = xy \bmod q$, and 0 on queries of the form $(g, g^x, g^y, g^z)$ where $z \neq xy \bmod q$.

**Definition 5.** *We say that the Gap Square Diffie-Hellman Problem (GSDH) in group* $\mathbb{G}$ *is* $(\epsilon, t)$*-hard if for every algorithm* $A$ *running in time* $t$ *on access to the DDH oracle* $\mathsf{DDH}_{\mathbb{G}}$ *in group* $\mathbb{G}$ *we have*

$$\Pr[x \leftarrow \mathbb{Z}_q : A^{\mathsf{DDH}_{\mathbb{G}}}(g, g^x) = g^{x^2}] \leq \epsilon.$$

It is well known that the SDH problem is equivalent to the computational Diffie-Hellman (DH) problem. Just note that $g^{xy} = (g^{(x+y)^2} / (g^{x^2} g^{y^2}))^{2^{-1}}$, and that oracle errors can be easily corrected since both the SDH and the DH problems are random self-reducible. Similarly, the GSDH problem is equivalent to the Gap Diffie-Hellman problem (GDH), which is believed to be hard in many prime-order groups. In particular, generic group algorithms cannot solve it in time better than $\Omega(\sqrt{q})$ [5].

## 4 AH-AGKE Scheme based on the RSA Assumption

- $\mathsf{Setup}$: On security parameter $\kappa$, the $\mathsf{Setup}$ procedure picks two other parameters $\kappa'$, and $\kappa''$. Parameter $\kappa$ is the length of the key output by the key agreement protocol $\mathsf{Handshake}$, $\kappa'$ is an additional parameter which in practice can be 160, and $\kappa''$ is chosen so that the RSA problem for $2\kappa''$-bit safe RSA moduli has at least $\kappa$-bit security (see theorem 1 for exact bounds). Whenever we say that two distributions $D_1, D_2$ are statistically close we mean that the statistical difference between them is bounded by $O(2^{-min(\kappa, \kappa', \kappa'')})$. The setup procedure also chooses a $\kappa'$-bit prime $\hat{q}$ and defines hash functions $H_{\hat{q}} : \{0,1\}^* \rightarrow Z_{\hat{q}}^*$ and $H : \{0,1\}^* \rightarrow \{0,1\}^\kappa$.

- $\mathsf{KGen}$: Generate a $2\kappa''$-bit safe RSA modulus $n = pq$, where $p = 2p' + 1$, $q = 2q' + 1$, and $p, q, p', q'$ are primes. Pick a random element $g$ s.t. $g$ generates a maximum subgroup in $Z_n^*$, i.e. $ord(g) = 2p'q'$, and s.t. $-1 \notin \langle g \rangle$. (This holds for about half of the elements in $Z_n^*$, and it is easily tested.) Note that in this case $Z_n^* \equiv \langle -1 \rangle \times \langle g \rangle$. Therefore, in particular, if $x \leftarrow Z_{2p'q'}$ and $b \leftarrow \{0, 1\}$ then $(-1)^b g^x$ is distributed uniformly in $Z_n^*$. RSA exponents $(e, d)$ are chosen in the standard way, as a small prime $e$ and $d = e^{-1} \pmod{\phi(n)}$. The secret key is $(p, q, d)$ and public key is $(n, g, e)$. Key generation also fixes a hash function $H_n : \{0,1\}^* \rightarrow Z_n$, specific to the group modulus $n$.[3]

- Add: To add user $U$ to the group, the manager picks a random string $id \leftarrow \{0,1\}^{\kappa'}$ and computes a (full-domain hash) RSA signature on $id$, $\sigma = h^d \pmod{n}$, where $h = H_n(id)$. $U$'s certificate is $\mathsf{cert} = (id, \sigma)$.

- Revoke: To remove user $U$ from the group, the manager appends string $id$ to the group $\mathcal{CRL}$, where $(\sigma, id)$ is $U$'s certificate in this group.

- Handshake: This is an AGKA protocol executed by some set $\Delta = \{U_1, ..., U_n\}$ of players. Each player $U_i$ starts a session $\Pi_i^{\tau_i}$ for a (locally) fresh $\tau_i$, on *some* inputs $(\mathsf{cert}_i, (n, e, g), \mathcal{CRL}_i)$ s.t. $(n, e, g)$ is some public key, $\mathsf{cert}_i = (id_i, \sigma_i)$ is $U_i$'s certificate for this public key $(n, e, g)$, i.e. $\mathsf{cert}_i \in \mathsf{Certs}(n, e, g)$, and $\mathcal{CRL}_i$ is the (hopefully recent) CRL for group $\mathsf{Group}(n, e, g)$. The Handshake protocol is in Figure 1 below (see also the note below).

**Notational Simplifications.** In figure 1, we make several assumptions to simplify the notation. First, we denote the set of participating players as simply $U_1, ..., U_n$, even though they can be any $n$ users $U_{i_1}, ..., U_{i_n}$ among $\mathcal{U} = \{U_1, ..., U_l\}$, for any $n \geq 2$. Secondly, we assume that the order between the players, which in the full protocol is determined on-line according to the players' messages in Round 1, is simply $U_1, ..., U_n$.[4] For simplicity of notation, we assume that the indices cycle modulo $n$, i.e. $U_{n+1} = U_1$. We also assume that each instance $\Pi_i^{\tau}$ starts on the same public key $(n, e, g)$. (Since we are not concerned with robustness properties in this paper, we do not concern ourselves with what happens with executions of instances which are not *partnered*, and in particular do not run on the same public keys.)

Affiliation hiding property of the protocol in figure 1 depends crucially on the fact that if the distribution of variable $\bar{\theta}_i$ is indistinguishable from uniform over $Z_n$ then the distribution of $\theta_i = \bar{\theta}_i + kn$ is statistically close to $U_{2^{2\kappa''+\kappa}}$. There is an alternative way that we can use to hide the range of $\theta_i$, which does not take the $\kappa$ bandwidth overhead [2], which is to repeat picking $\bar{\theta}_i$ until $\bar{\theta}_i \in \{0,1\}^{2\kappa''-1}$. However, the expected running time of such procedure is twice larger than ours. Moreover, such procedure can be subject to timing attacks. Note that the overhead of $\kappa$ bits our procedure incurs is small compared to $|\bar{\theta}_i| = |n|$.

**Protocol Correctness.** To see that the protocol Handshake in Figure 1 is correct, note that if some $n$ sessions $\Pi_1^{\tau_1}, ..., \Pi_n^{\tau_n}$ are partnered then they all run on the same public key $(n, e, g)$, and all the values $(\theta_1, id_1, \mu_1, X_1)$,...,$(\theta_n, id_n, \mu_n, X_n)$ are exchanged between them without interference. Therefore, first of all, each participating player will create the same order among the participants, and hence each player labels all the exchanged values in the same way, so we can assume for simplicity that this ordering coincides with the original labels $i = 1, ..., n$. Each player also computes also the same value $s$ and $\mathsf{sid}_i$. To see that each player computes the same value $k_i$ and hence the same key $K_i$, note that for each $j$ we have

[3] Selecting separate hash function $H_n$ for every group is done purely for notational convenience. A family of hash functions $H_n : \{0,1\}^* \rightarrow Z_n$ s.t. each $H_n$ is statistically close to a random function with range $Z_n$, can be easily implemented in the random oracle model with a single hash function with range $2^{2\kappa''+\kappa}$. E.g., $H_n(m) = H(n, m) \bmod n$.

[4] This ordering is done as follows: In the protocol each player $U_i$ picks a long-enough random nonce $\mu_i$ (see Round 1 in figure 2), which $U_i$ then includes in *all* its messages in the protocol. After receiving some set of messages in Round 1 (note that we assume weak synchrony), every receiver sorts all the received messages by the increasing order of these $\mu_i$ values. Each player then (re)labels all the participants and the messages received in Round 1, including its own messages and its inputs, according to this order. The actual protocol runs exactly as the simplified protocol in figure 2 in the case that this ordering of players created in Round 1 coincides with the original labels $i = 1, ..., n$ of the participants $U_1, ..., U_n$ assumed for simplicity in figure 2.

The inputs of instance $\Pi_i^\tau$ of player $U_i$ are cert$_i = (\sigma_i, id_i)$, $(n, g, e)$, and $\mathcal{CRL}_i$. Note that $\sigma_i^e = H_n(id_i) \bmod n$.

**[Round 1]**: $U_i$ picks random values $b_i \leftarrow \{0,1\}$, $t_i \leftarrow Z_{n/2}$, and $\mu_i \leftarrow \{0,1\}^{3\kappa}$, computes $\bar{\theta}_i = (-1)^{b_i}\sigma_i g^{t_i} \pmod{n}$, sets $\theta_i = \bar{\theta}_i + \nu n$ for random $\nu \leftarrow [0, ..., \lfloor 2^{2\kappa''+\kappa}/n \rfloor]$, and $U_i$ broadcasts $(\theta_i, id_i, \mu_i)$.

- Assume that player $U_i$ received $n$-1 messages $(\theta_1, id_1, \mu_1)$ , ..., $(\theta_{i-1}, id_{i-1}, \mu_{i-1})$, $(\theta_{i+1}, id_{i+1}, \mu_{i+1})$, $(\theta_n, id_n, \mu_n)$ in Round 1. (This is a simplification. In the real protocol each receiver $U_i$ orders the received messages, and the players which sent them, according to values $\mu$ these messages contain. See footnote 4.)
  If any two messages contain the same value $id_j$ or the same value $\mu_j$, player $U_i$ rejects.
- $U_i$ sets $s = ((n, g, e), \{\theta_j, id_j, \mu_j\}_{j=1,...,n})$
- If $id_j \in \mathcal{CRL}_i$ for any $j$ then $U_i$ picks a random value $X_i$ in $Z_{\hat{q}}^*$ and sets reject $= T$. Otherwise, $U_i$ computes $X_i = H_{\hat{q}}((z_{i+1})^{t_i}, s)/H_{\hat{q}}((z_{i-1})^{t_i}, s) \pmod{\hat{q}}$ , where $z_{i+1} = (\theta_{i+1})^{2e}(h_{i+1})^{-2} \pmod{n}$ and $z_{i-1} = (\theta_{i-1})^{2e}(h_{i-1})^{-2} \pmod{n}$.
  (Note that if $(id_j, \sigma_j)$ is a certificate for public key $(n, e, g)$ and $\theta_j = (-1)^{b_j}\sigma_j g^{t_j} + \nu n$ then $z_j = g^{2et_j}$.)

**[Round 2]**:

$U_i$ broadcasts $(X_i, \mu_i)$.

- If in Round 2 player $U_i$ receives $n$-1 values $X_j$ accompanied by $\mu_j$'s that match the $\mu_1, ..., \mu_{i-1}, \mu_{i+1}, ..., \mu_n$ values above, if $\prod_{j=1}^n X_j = 1$, and if reject $\neq T$, then $U_i$ computes $k_i = H_{\hat{q}}((z_{i-1})^{t_i}, s)^n \cdot (X_i)^{n-1} \cdot (X_{i+1})^{n-2} \cdots X_{i-2} \pmod{\hat{q}}$ and outputs $K_i = H(k_i, \mathsf{sid}_i)$, where $\mathsf{sid}_i = ((n, g, e), \{\theta_j, id_j, \mu_j, X_j\}_{j=1,...,n})$. Otherwise $U_i$ rejects.

**Figure 1. RSA-based Affiliation-Hiding AGKE protocol**

$z_j = \theta_j^{2e} h_j^{-2} = g^{2et_j}$, and therefore, each $X_i = H_{\hat{q}}(g^{2et_i t_{i+1}}, s)/H_{\hat{q}}(g^{2et_{i-1}t_i}, s) \pmod{\hat{q}}$. Note also that $H_{\hat{q}}((z_{i-1})^{t_i}, s) = H_{\hat{q}}(g^{2et_{i-1}t_i}, s)$. It follows that for every $i$ we have

$$k_i = H_{\hat{q}}(g^{2et_{i-1}t_i}, s) * H_{\hat{q}}(g^{2et_i t_{i+1}}, s) * ... * H_{\hat{q}}(g^{2et_{i-2}t_{i-1}}, s) \bmod \hat{q}$$

Therefore all the keys $K_i$ are the same as well.

**Theorem 1.** *Assuming that the RSA problem is* $(\epsilon', t')$*-hard on random safe RSA moduli of length* $2\kappa''$*, the above tuple of algorithms* $(\mathsf{Setup}, \mathsf{KGen}, \mathsf{Add}, \mathsf{Revoke}, \mathsf{Handshake})$ *is an* $(\epsilon, t, q_s, q_H, l, m)$*-secure AH-AGKE scheme in the Random Oracle Model as long as*

$$\begin{aligned} \epsilon &\approx m * (2\epsilon' + 2lq_H 2^{-\kappa'} + q_s^2 2^{-3\kappa} + q_s 2^{-\kappa''+2}) \\ t &\approx t' - (m * t_{kg} + q_s * q_H * t_{exp}) \end{aligned}$$

*where* $t_{kg}$ *is the time to generate an RSA private/public key pair and* $t_{exp}$ *is the time of (multi)exponentiation modulo* $n$*, for* $2\kappa''$*-bit RSA moduli.*

*Proof.* Assume a legitimate PPT adversary $\mathcal{A}$ interacting with challenger $\mathcal{C}$ as described in the security definition (definition 1). Assume that there are $m$ groups and $l$ users in the universe, and that $\mathcal{A}$ runs in time $t$, starts at most $q_s$ sessions, and makes at most $q_H$ queries to the hash functions $H_n$, $H_{\hat{q}}$, and $H$. Assume w.l.o.g. that $\mathcal{A}$ always makes a test query on some session. Denote $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{sec}} = |Pr[b' = b]|$, i.e. the advantage of the adversary $\mathcal{A}$ in the interaction with $\mathcal{C}$, by $\epsilon$. We split the security proof into two parts. First we describe the simulation procedure, $\mathcal{SIM}$, which using $\mathcal{A}$, attempts to solve for $z$ s.t. $z^e = g \bmod n$ on an RSA challenge $(n, e, g)$. This simulation procedure will run in time $t'$ approximately $t + (m * t_{kg} + q_s * q_H * t_{exp})$. We will then argue that the probability of $\mathcal{SIM}$'s success in solving the RSA challenge is at least $\epsilon' \geq \epsilon/m - (2lq_H 2^{-\kappa'} + q_s^2 2^{-3\kappa} + q_s 2^{-\kappa''+2})$, assuming that element $g$ in $\mathcal{SIM}$'s challenge is such that $\langle -1 \rangle \times \langle g \rangle = Z_n^*$. Note that if $n$ is a safe RSA modulus then for a random $g \in Z_n^*$

this holds with probability $1/2 - O(2^{-|n|/2}) \approx 1/2$. Therefore, the success of $\mathcal{SIM}$ on solving a random $g \in Z_n^*$ is (statistically close to) at least half the above expression, which completes the proof.

PART I: CONSTRUCTION OF A SIMULATOR

**Setup.** Given the RSA challenge $(n, e, g)$, $\mathcal{SIM}$ follows the $\mathsf{Setup}$ algorithm with parameters $\kappa, \kappa'$, and $\kappa'' = |n|/2$. As mentioned above, we assume that $\langle -1 \rangle \times \langle g \rangle = Z_n^*$.

**Initialization of all groups.** Let $G^* \in \mathcal{G}$ be a group s.t. the probability that the adversary $\mathcal{A}$ tests on $\Pi_i^\tau$ s.t. $\mathsf{Group}(\Pi_i^\tau) = G^*$ is not less than $1/m$. (Recall that we assume $\mathcal{A}$ always tests some session.) Simulator $\mathcal{SIM}$ initializes all the groups in $\mathcal{G}$ except $G^*$ as in the real protocol. $\mathcal{SIM}$ also creates $l$ certificates for each of these groups by following the $\mathsf{Add}$ procedure, and in the rest of the simulation $\mathcal{SIM}$ simply follows the $\mathsf{Handshake}$ protocol on behalf of all instances $\Pi_i^\tau$ s.t. $\mathsf{Group}(\Pi_i^\tau) \neq G^*$. Thus, in the rest of the simulation description we will only describe $\mathcal{SIM}$'s actions with regard to instances $\Pi_i^\tau$ s.t. $\mathsf{Group}(\Pi_i^\tau) = G^*$.

For group $\mathcal{G}^*$, $\mathcal{SIM}$ sets its public key as $(n, e, g)$, and creates the certificates for each revoked player $U_i \in \mathsf{Rev}$ by simulating an RSA signature $(id_i, \sigma_i)$ under key $(n, e, g)$. Namely, $\mathcal{SIM}$ picks two random values $id_i \leftarrow \{0,1\}^{\kappa'}$ and $\sigma_i \leftarrow Z_n^*$, and assigns $H_n(id_i)$ to $\sigma_i^e \pmod{n}$. If $\mathcal{A}$ has already queried $H_n$ on any $id_i$'s chosen by $\mathcal{SIM}$ in this way, $\mathcal{SIM}$ abandons the simulation. For each $U_i \notin \mathsf{Rev}$ in $G^*$, $\mathcal{SIM}$ picks a random value $id_i \leftarrow \{0,1\}^{kappa'}$. $\mathcal{SIM}$ hands to $\mathcal{A}$ all the public keys and the certs of the corrupted players.

**Hash queries to $H_n$, $H_{\hat{q}}$ and $H$.** For each query $x$ to $H_n$, $\mathcal{SIM}$ picks random $a \leftarrow Z_n^*$ and sets $H_n(x) = a^e \cdot g^{-1} \pmod{n}$. W.l.o.g, assume that $H_n$ is queried on each $id_i$ for $U_i \notin \mathsf{Rev}$. Denote value $a$ chosen above for $x = id_i$ as $a_i$, and $H_n(id_i) = a_i^e g^{-1}$ as $h_i$. For the queries to $H$ and $H_{\hat{q}}$, $\mathcal{SIM}$ simply passes these queries to $H$ and $H_{\hat{q}}$, respectively. However, for each query $(r, s)$ to $H_{\hat{q}}$, $\mathcal{SIM}$ also tries to solve the RSA challenge as we describe below.

After the above initialization, $\mathcal{SIM}$ must provide responses for $\mathcal{A}$'s queries $\mathsf{Start}$, $\mathsf{Send}$, $\mathsf{Reveal}$, and $\mathsf{Test}$, which would look to $\mathcal{A}$ as the real execution, i.e. as in $\mathcal{A}$'s interaction with challenger $\mathcal{C}$. For notational convenience assume that the local index $\tau$ of each instance $\Pi_i^\tau$ is globally unique (e.g., assume that $\tau$ in $\Pi_i^\tau$ has a suffix $i$). In the following description, we add as a superscript the instance index $\tau$ to all values related to $\Pi_i^\tau$. For example, $\theta_i^\tau$, $X_i^\tau$ will refer to messages $\theta_i$, $X_i$ sent by instance $\Pi_i^\tau$. As mentioned above, $\mathcal{SIM}$ responds to $\mathcal{A}$'s commands relating to instances $\Pi_i^\tau$ s.t. $\mathsf{Group}(\Pi_i^\tau) \neq G^*$ by simply following the honest players' protocol. However, for queries involving group $G^*$, simulator $\mathcal{SIM}$ respondes as follows:

**Start commands.** For the $\mathsf{Start}(U_i, G^*)$ command, $\mathcal{SIM}$ initializes instance $\Pi_i^\tau$. $\mathcal{SIM}$ picks $b_i^\tau \leftarrow \{0,1\}$, $\gamma_i^\tau \leftarrow Z_{n/2}$, and computes $\bar{\theta}_i^\tau = (-1)^{b_i^\tau} \cdot a_i \cdot g^{\gamma_i^\tau} \pmod{n}$. Notice that the distribution of $\bar{\theta}_i^\tau$ in this simulation and in the real protocol are statistically close because both are statistically close to uniform in $Z_n^*$. Note that since $h_i = (a_i)^e/g$, therefore $\bar{\theta}_i^\tau = (-1)^{b_i^\tau} \cdot (h_i g)^d \cdot g^{\gamma_i^\tau} = (-1)^{b_i^\tau} \cdot h_i^d \cdot g^{d+\gamma_i^\tau} = (-1)^{b_i^\tau} \cdot h_i^d \cdot g^{t_i^\tau}$ $\pmod{n}$, where $t_i^\tau = \gamma_i^\tau + d \pmod{\phi(n)/2}$. The simulator does not know either $d$ or $t_i^\tau$, but will use the above relation to solve for $g^d$ later. $\mathcal{SIM}$ also chooses $\mu_i^\tau \leftarrow \{0,1\}^{3\kappa}$, $\nu_i^\tau \leftarrow [0, ..., \lfloor 2^{2\kappa''+\kappa}/n \rfloor]$, sets $\theta_i^\tau = \bar{\theta}_i^\tau + \nu_i^\tau$ and replies with $(\theta_i^\tau, id_i, \mu_i^\tau)$.

**Send queries.** Consider an instance $\Pi_i^\tau$ created by the $\mathsf{Start}$ command above. We denote the $\mathsf{Send}$ command to this instance corresponding to Round 1 of the protocol by $\mathsf{Send}_1$, and the $\mathsf{Send}$ command corresponding to Round 2 of the protocol by $\mathsf{Send}_2$. In the following statement, just like we did in the description of the protocol, we assume that the index of player $U_i$ involved in session $\Pi_i^\tau$ belongs to set $i \in \{1, .., n\}$. (In general $i \in \{1, .., l\}$ where $l = |\mathcal{U}| > n$, but the proof in the general case is easy to extrapolate from the proof we give here.)

For the $\mathsf{Send}_1(U_i, \tau, \{\hat{\theta}_j^\tau, \hat{id}_j^\tau, \hat{\mu}_j^\tau\}_{j=1,..,n,j\neq i})$ command, unless there are collisions in $id$'s or $\mu$'s, $\mathcal{SIM}$ sets $s_i^\tau$ as in the protocol. If any $\hat{id}_j^\tau$'s are on $\mathcal{CRL}_i$ then $\mathcal{SIM}$ sets $X_i^\tau \leftarrow Z_{\hat{q}}^*$ and $\mathsf{reject}_i^\tau = T$. Otherwise,

$\mathcal{SIM}$ sets $X_i^\tau = c_{i,i+1}^\tau / c_{i,i-1}^\tau \pmod{\hat{q}}$ where values $c_{i,j}^\tau$ for $j = i-1$ and $j = i+1$ are chosen as follows. If $\exists$ some $\Pi_{i'}^{\tau'}$ which received the $\mathsf{Send}_1$ query s.t.:

1. $(\theta_{i'}^{\tau'}, id_{i'}^{\tau'}, \mu_{i'}^{\tau'}) = (\hat{\theta}_j^\tau, \hat{id}_j^\tau, \hat{\mu}_j^\tau)$
2. $(\hat{\theta}_{j'}^{\tau'}, \hat{id}_{j'}^{\tau'}, \hat{\mu}_{j'}^{\tau'}) = (\theta_i^\tau, id_i^\tau, \mu_i^\tau)$ for $j' = i'+1$ or $j' = i'-1$
3. $s_{i'}^{\tau'} = s_i^\tau$ and $\mathsf{reject}_{i'}^{\tau'} \neq T$

then $\mathcal{SIM}$ assigns $c_{i,j}^\tau \leftarrow c_{i',j'}^{\tau'}$, where $c_{i',j'}^{\tau'}$ is a value $\mathcal{SIM}$ has previously chosen when dealing with the $\mathsf{Send}_1$ command to session $\Pi_{i'}^{\tau'}$. Note that this case corresponds to an adversary who honestly routes the messages of matching instances $\Pi_i^\tau$ and $\Pi_{i'}^{\tau'}$ from one to another. In such case in the real execution these two instances would compute the same value $c_{i,j}^\tau = c_{i',j'}^\tau$, where

$$c_{i,j}^\tau = H_{\hat{q}}((\hat{z}_j^\tau)^{t_i^\tau}, s_i^\tau) \quad \text{and} \quad c_{i',j'}^{\tau'} = H_{\hat{q}}((\hat{z}_{j'}^{\tau'})^{t_{i'}^{\tau'}}, s_{i'}^{\tau'})$$

If any of these conditions are not met, which corresponds to the case where there is no instance $\Pi_{i'}^{\tau'}$ which runs on matching inputs as $\Pi_i^\tau$, or when the adversary actively interferes in the communication between these two instances, $\mathcal{SIM}$ picks a fresh random value $c_{i,j}^\tau \leftarrow Z_{\hat{q}}^*$. In both cases $\mathcal{SIM}$ stores $[j, \Pi_i^\tau, s_i^\tau, (\hat{\theta}_j^\tau, \hat{id}_j^\tau, \hat{\mu}_j^\tau), c_{i,j}^\tau]$ in a table denoted $T_{H_{\hat{q}}}$. Finally, $\mathcal{SIM}$ replies with $(X_i^\tau, \mu_i^\tau)$.

For all the $\mathsf{Send}_2(U_i, \tau, \{\hat{X}_j^\tau, \hat{\mu}_j^\tau\}_{j=1,..,n,j\neq i})$ commands, $\mathcal{SIM}$ abandons $\Pi_U^\tau$ if values $\hat{\mu}_j^\tau$ are not correct or $\Pi_{j=1}^n \hat{X}_j^\tau \neq 1$, where $\hat{X}_i^\tau = X_i^\tau$; otherwise $\mathcal{SIM}$ sets $\mathsf{sid}_i^\tau$ as in the protocol, computes

$$k_i^\tau = (c_{i,i-1}^\tau)^n \cdot (\hat{X}_i^\tau)^{n-1} \cdot (\hat{X}_{i+1}^\tau)^{n-2} \cdots (\hat{X}_{i-2}^\tau) \pmod{\hat{q}} \tag{1}$$

and outputs $K_i^\tau = H(k_i^\tau, \mathsf{sid}_i^\tau)$.

**Reveal queries.** On $\mathsf{Reveal}(U_i, \tau)$, if instance $\Pi_i^\tau$ has output a session key $K_i^\tau$, $\mathcal{SIM}$ delivers it to $\mathcal{A}$.

**Test query.** Finally, if adversary issues command $\mathsf{Test}(i, \tau)$ then $\mathcal{SIM}$ picks a random bit $b$ as $\mathcal{C}$ does, and if $b = 1$ then $\mathcal{SIM}$ replies with $K_i^\tau$ to $\mathcal{A}$. Otherwise, $\mathcal{SIM}$ returns a random value in $\{0,1\}^\kappa$.

**Computing the RSA challenge.** Every time $\mathcal{A}$ makes a query $(r, s)$ to $H_{\hat{q}}$, $\mathcal{SIM}$ attempts to solve its RSA challenge as follows. For each entry $[j, \Pi_i^\tau, s_i^\tau, (\hat{\theta}_j^\tau, \hat{id}_j^\tau, \hat{\mu}_j^\tau), c_{i,j}^\tau]$ in table $T_{H_{\hat{q}}}$ s.t. $s_i^\tau = s$, $\mathcal{SIM}$ wants to check if

$$r = ((\hat{\theta})^{2e}(\hat{h})^{-2})^t = (\hat{\theta})^{2e(\gamma+d)}(a^e/g)^{-2(\gamma+d)} = (\hat{\theta}/a)^{2(e\gamma+1)} g^{2\gamma} g^{2d} \pmod{n} \tag{2}$$

where $\hat{\theta} = \hat{\theta}_j^\tau$, $a$ is the value s.t. $\hat{h} = H_n(\hat{id}_j^\tau) = a^e \cdot g^{-1} \pmod{n}$, and $(t, \gamma) = (t_i^\tau, \gamma_i^\tau)$ defined when session $\Pi_i^\tau$ was started. Note that if $r = (\hat{z}_j^\tau)^{t_i^\tau}$, i.e. $\mathcal{A}$ queries $H_{\hat{q}}$ on pair $(r, s) = ((\hat{z}_j^\tau)^{t_i^\tau}, s_i^\tau)$, where $\hat{z}_j^\tau = (\hat{\theta}_j^\tau)^{2e}(\hat{h})^{-2}$ and $t_i^\tau$ is the value that satisfies $\bar{\theta}_i^\tau = (-1)^{b_i^\tau}(h_i)^d g^{t_i^\tau}$, then $r = ((\hat{\theta}_j^\tau)^{2e}(\hat{h})^{-2})^{t_i^\tau}$.

The way $\mathcal{SIM}$ can verify if equation (2) holds is to compute

$$w = r \cdot (\hat{\theta}/a)^{-2(1+e\gamma)} g^{-2\gamma} \pmod{n} \tag{3}$$

and test if $w^e = g^2$. If this holds then $\mathcal{SIM}$ extracts $g^d$ by computing $w^\beta g^\alpha$, where $\alpha, \beta$ satisfy $e\alpha + 2\beta = 1$.

PART II: ANALYSIS OF THE SIMULATION

First note that if the adversary $\mathcal{A}$ runs in time $t$ then the running time $t'$ of the above simulator $\mathcal{SIM}$ is dominated by $t + m * t_{kg} + q_s * q_H * t_{exp}$, where $t_{kg}$ is the time to generate an RSA private/public key pair and $t_{exp}$ is the time of an exponentiation modulo $n$.

Denote as $N_b$ the real network as executed by the challenger $\mathcal{C}$ with a fixed bit $b$. Recall that if $b = 0$ then $\mathcal{C}$ sends to $\mathcal{A}$ a random $\kappa$-bit long value and if $b = 1$ then $\mathcal{C}$ delivers the session key of the tested instance. We also denote as $\mathcal{SIM}_b$ an execution of the above simulator $\mathcal{SIM}$ with a fixed bit $b$ on challenge $(n, e, g)$ where $g$ satisfies $\langle -1 \rangle \times \langle g \rangle = Z_n^*$.

We define the following events:

$\mathsf{NE_b}$: $\mathcal{A}$ outputs 1 on interaction with $N_b$.

$\mathsf{NE_{G,b}}$: $\mathcal{A}$ outputs 1 and tests session $\Pi_i^\tau$ s.t. $\mathsf{Group}(\Pi_i^\tau) = G$, on interaction with $N_b$.

$\mathsf{SE_{G,b}}$: $\mathcal{A}$ outputs 1 and tests session $\Pi_i^\tau$ s.t. $\mathsf{Group}(\Pi_i^\tau) = G$, on interaction with $\mathcal{SIM}_b$.

$\mathsf{sCollision}$: There is a user $U_i$ s.t. $s_i^{\tau_1} = s_i^{\tau_2}$ for some $\tau_1 \neq \tau_2$, either in an execution or in a simulation.

$\mathsf{H_nFailure}$: $\mathcal{A}$ queries $H_n$ on $id_i$ for some $U_i \in \mathsf{Rev}$ *before* this value is chosen, by $\mathcal{C}$ in an execution and by $\mathcal{SIM}$ in a simulation.

$\bar{\mathsf{N}}\mathsf{E_{G^*,b}} = \mathsf{NE_{G^*,b}} \wedge \neg(\mathsf{H_nFailure} \vee \mathsf{sCollision})$

$\bar{\mathsf{S}}\mathsf{E_{G^*,b}} = \mathsf{GE_{G^*,b}} \wedge \neg(\mathsf{H_nFailure} \vee \mathsf{sCollision})$

$\mathsf{H_{\hat{q}}Query}$: There is a session $\Pi_i^\tau$ s.t. $\mathcal{A}$ queries $H_{\hat{q}}$ on pair $((\hat{z}_j^\tau)^{t_i^\tau}, s_i^\tau)$, for $j = i{-}1$ or $j = i{+}1$, which relates to this $\Pi_i^\tau$ session, i.e. $\hat{z}_j^\tau = (\hat{\theta}_j^\tau)^{2e}(H_n(\hat{id}_j^\tau))^{-2}$, and $t_i^\tau$ satisfies $g^{t_i^\tau} = (\theta_i^\tau)^{2e}(H_n(id_i))^{-2}$.

Note that by the assumption that $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{sec}} = |Pr[b' = b]| \geq \epsilon$ we have $|\Pr[\mathsf{NE_1}] - \Pr[\mathsf{NE_0}]| \geq 2\epsilon$. Also, since $\Pr[\mathsf{E_b}] = \sum_{G \in \mathcal{G}} \Pr[\mathsf{NE_{G,b}}]$, let $G^* \in \mathcal{G}$ be a group s.t.

$$|\Pr[\mathsf{NE_{G^*,1}}] - \Pr[\mathsf{NE_{G^*,0}}]| \geq 2\epsilon/m \tag{4}$$

Assume that this is a group chosen by the simulator $\mathcal{SIM}$ above. (Note that $\mathcal{SIM}$ could also guess $\mathcal{G}^*$ with $1/m$ probability.) We will argue the following four facts:

$$\begin{aligned}
|\Pr[\mathsf{NE_{G^*,b}}] - \Pr[\bar{\mathsf{N}}\mathsf{E_{G^*,b}}]| &\leq \Pr[\mathsf{H_nFailure} \wedge \mathsf{sCollision}] \;\text{ for } b = 0,1 & (5)\\
\Pr[\mathsf{H_nFailure} \wedge \mathsf{sCollision}] &\leq lq_H 2^{-\kappa'} + q_s^2 \cdot 2^{-3\kappa} & (6)\\
|\Pr[\bar{\mathsf{S}}\mathsf{E_{G^*,1}}] - \Pr[\bar{\mathsf{S}}\mathsf{E_{G^*,0}}]| &\leq q_H 2^{-\kappa'} & (7)\\
|\Pr[\bar{\mathsf{N}}\mathsf{E_{G^*,b}} \mid \neg\mathsf{H_{\hat{q}}Query}] - \Pr[\bar{\mathsf{S}}\mathsf{E_{G^*,b}} \mid \neg\mathsf{H_{\hat{q}}Query}]| &\leq q_s 2^{-\kappa''+2} \;\text{ for } b = 0,1 & (8)
\end{aligned}$$

Note that by inequalities (4)-(7) it follows that for either $b = 0$ or $b = 1$ we have:

$$|\Pr[\bar{\mathsf{N}}\mathsf{E}_{G^*,b}] - \Pr[\bar{\mathsf{S}}\mathsf{E}_{G^*,b}]| \geq \epsilon/m - (lq_H 2^{-\kappa'} + q_s^2 2^{-3\kappa} + q_H 2^{-\kappa'}/2) \geq \epsilon/m - (2lq_H 2^{-\kappa'} + q_s^2 2^{-3\kappa})$$

Together with (8), this inequality implies that

$$\Pr[\mathsf{H_{\hat{q}}Query}] \geq \epsilon/m - (2lq_H 2^{-\kappa'} + q_s^2 2^{-3\kappa} + q_s 2^{-\kappa''+2})$$

Since whenever event $\mathsf{H_{\hat{q}}Query}$ happens the simulator $\mathcal{SIM}$ solves its RSA challenge, this implies our claim that $\epsilon' \geq \epsilon/m - (2lq_H 2^{-\kappa'} + q_s^2 2^{-3\kappa} + q_s 2^{-\kappa''+2})$

It remains for us to argue that statements (5)-(8) above indeed hold. Note that inequality (5) follows immediately from the definition of $\mathsf{NE_{G^*,b}}$. For inequality (6) observe that $\Pr[\mathsf{H_nFailure}] \leq lq_H 2^{-\kappa'}$ because $|\mathsf{Rev}| \leq l$ and the response on each query to $H_n$ is a random element in the set of size $\{0,1\}^{\kappa'}$. Also $\Pr[\mathsf{sCollision}] \leq q_s^2 \cdot 2^{-3\kappa}$ because a collision in $s_i^\tau$ values for any user $U_i$ can only happen if two sessions

$\Pi_i^{\tau_1}$ and $\Pi_i^{\tau_2}$ of this user choose the same value $\mu_i^{\tau_1} = \mu_i^{\tau_2}$. Since every session chooses its $\mu_i^{\tau}$ value at random in a set of size $2^{-3\kappa}$, and there are at most $q_s$ sessions, the above bound follows.

Equality (8) is also straightforward to see. First note that the statistical difference between all the values $\theta_i^{\tau}$ in the execution and the simulation is $q_s 2^{-\kappa''+2}$, because for each $\Pi_i^{\tau}$, the difference between distribution of $t_i^{\tau}$ chosen as in the execution as $t_i^{\tau} \leftarrow Z_{n/2}$, and the distribution of values $t_i^{\tau} = \gamma_i^{\tau} + d \pmod{\phi(n)/2}$ for $\gamma_i^{\tau}$ uniform in $Z_{n/2}$ (recall that this is how value $t_i^{\tau}$ is defined in the simulation), is at most $2^{-\kappa''+2}$. Everything else in the execution and the simulation is distributed in the same way, provided $g$ is correct and event $\mathsf{H_nFailure}$ does not happen, except for the way values $c_{i,j}^{\tau}$ are computed. Now, if $\mathsf{H_{\hat{q}}Query}$ does not happen, i.e. if for all sessions $\Pi_i^{\tau}$, adversary $\mathcal{A}$ does not query the hash function $H_{\hat{q}}$ on the proper pair $((\hat{z}_j^{\tau})^{t_i^{\tau}}, s_i^{\tau})$ that corresponds to the $\Pi_i^{\tau}$ session, then the way $c$'s are computed in the execution (as outputs of $H_{\hat{q}}$) and the way they are picked in the simulation (at random in $Z_{\hat{q}}^*$ except if two sessions are partnered) are the same from $\mathcal{A}$'s point of view. The reason that's the case is that the only case in the protocol execution when two sessions $\Pi_i^{\tau}, \Pi_{i'}^{\tau'}$ compute two $c$ values on the same input is if $s_i^{\tau} = s_{i'}^{\tau'}$. But if there is no collisions in s values (event $\mathsf{sCollision}$) then this implies in particular that that the adversary re-routed messages of these two sessions between each other, and in this case the simulator $\mathcal{SIM}$ also makes the two $c$ values equal to one another.

It remains to argue that inequality (7) holds. Note that the only difference in these two interactions is that in $\mathsf{S}\bar{\mathsf{E}}_{\mathsf{G}^*,1}$ $\mathcal{A}$ gets key $K_i^{\tau} = H(k_i^{\tau}, s_i^{\tau})$ on tested $\Pi_i^{\tau}$, while in $\mathsf{S}\bar{\mathsf{E}}_{\mathsf{G}^*,0}$ $\mathcal{A}$ gets a random $\kappa$-bit value instead of $K_i^{\tau}$. Note that in $\mathcal{A}$'s interaction with $\mathsf{S}\bar{\mathsf{E}}$, if we disregard for a moment the information $\mathcal{A}$ gets from queries to $H(k_{i'}^{\tau'}, s_{i'}^{\tau'})$ for any $\Pi_{i'}^{\tau'}$ (this information is contained in the answers of $\mathsf{Test}$ and $\mathsf{Reveal}$ queries), then value $k_i^{\tau}$ is hidden from $\mathcal{A}$ in an information-theoretic way, i.e. it's uniformly distributed in $Z_{\hat{q}}^*$ independently from everything else $\mathcal{A}$ sees. The reason that's the case is because, by equation (1), for each $\Pi_i^{\tau}$, value $k_i^{\tau}$ is distributed independently from $\mathcal{A}$'s view as long as $c_{i,i-1}^{\tau}$ is independent from $\mathcal{A}$'s view. Disregarding $\mathcal{A}$'s queries to $H$, the only way value $c_{i,i-1}^{\tau}$ enters into the information $\mathcal{A}$ gets in the simulation is via $X_i^{\tau} = c_{i,i+1}^{\tau}/c_{i,i-1}^{\tau} \pmod{\hat{q}}$, where the $c_{i,i+1}^{\tau}$ is chosen independently from $c_{i,i-1}^{\tau}$, *except* if $\Pi_i^{\tau}$ is partnered by $\mathcal{A}$'s $\mathsf{Send}_1$ commands with some other session $\Pi_{i'}^{\tau'}$ (see the three conditions on sessions $\Pi_i^{\tau}$ and $\Pi_{i'}^{\tau'}$ in the procedure for $\mathcal{SIM}$ on $\mathsf{Send}_1$ query). In that case we have $c_{i,j}^{\tau} = c_{i',j'}^{\tau'}$ for some $j = i \pm 1$ and $j' = i' \pm 1$, and thus we have to ask if $c_{i,i-1}^{\tau}$ is still perfectly uniform given $X_i^{\tau}, X_{i'}^{\tau'}$. Let us call a pair $(\Pi_i^{\tau}, \Pi_{i'}^{\tau'})$ *related* if this is the case and assume $j = i + 1$ and $j' = i' - 1$ (in general there are three other cases for $\mathcal{A}$ to pair up these sessions, but the argument given here can be extended to this general case). Let $\Pi_{i_1}^{\tau_{i_1}}, ..., \Pi_{i_n}^{\tau_{i_n}}$ be sessions s.t. for each $j$, sessions $\Pi_{i_j}^{\tau_{i_j}}$ and $\Pi_{i_{j+1}}^{\tau_{i_{j+1}}}$ are related in the above way. However, even in this case, each variable $c_{i_j, i_{j-1}}$, taken by itself, is still uniformly distributed in $Z_{\hat{q}}^*$ (although *not* independently from one another) given $\mathcal{A}$'s view $X_{i_1}^{\tau_{i_1}}, ..., X_{i_n}^{\tau_{i_n}}$, because each $X_{i_1}$ sets one constraint between two $c$'s but there are $n + 1$ independently chosen $c$'s involved.

Finally, let us put back the additional information related to any of these $c_{i_j, i_j - 1}$ values that $\mathcal{A}$ gets from hash function outputs $H(k_{i_j}^{\tau_{i_j}}, s_{i_j}^{\tau_{i_j}})$. Note that $\mathcal{A}$ gets to see these outputs from both its $\mathsf{Reveal}$ and $\mathsf{Test}$ queries, and $\mathcal{A}$ can query $H$ to search for the matching value $k_{i_j}^{\tau_{i_j}}$ for any $\Pi_{i_j}^{\tau_{i_j}}$ in a chain of related sessions $\Pi_{i_1}^{\tau_{i_1}}, ..., \Pi_{i_n}^{\tau_{i_n}}$. Learning any such $k_{i_j}^{\tau_{i_j}}$ value implies learning the corresponding $c_{i_j, i_j - 1}^{\tau_{i_j}}$ value, and together with $X_{i_1}^{\tau_{i_1}}, ..., X_{i_n}^{\tau_{i_n}}$ this leads to recovery of *all* values $c_{i_1, i_1 - 1}^{\tau_{i_1}}, ..., c_{i_n, i_n - 1}^{\tau_{i_n}}$. However, $\mathcal{A}$ can only make $q_H$ hash queries to $H$, and since each of these values is (individually) uniform in $Z_{\hat{q}}^*$, the ability to query $H$ can leak information on any of these values with probability at most $q_h 2^{-\kappa'}$, because $\hat{q}$ is a $\kappa'$-bit prime. This implies the $q_h 2^{-\kappa'}$ bound on the distance between the two simulations, for $b = 0$ and $b = 1$. □

**Theorem 2.** *The AH-AGKE protocol defined by the above tuple* ($\mathsf{Setup}$, $\mathsf{KGen}$, $\mathsf{Add}$, $\mathsf{Revoke}$, $\mathsf{Handshake}$) *is Affiliation-Hiding.*

*Proof.* Since the $id$ values are chosen independently of the group, the only values which can reveal something about the group membership of the honest player are the $\theta$ values. We claim that the distribution of value $\theta$ sent by an honest user in this protocol is statistically close to a uniform distribution on $(2\kappa'' + \kappa)$-bit strings, denoted $Z_{2^{2\kappa''+\kappa}}$. Recall that for all groups $G_1, ..., G_m$ we have $2\kappa'' = |n_1| = ... = |n_m|$. We use $U \approx_S V$ to denote that distribution $U$ is statistically close to $V$ in the sense that the difference between these distributions is at most $O(2^{-min(\kappa,\kappa'')})$.

As we noted in the construction, values $(-1)^b g^t \pmod{n}$ are uniformly distributed in $Z_n^*$ for $(b,t) \leftarrow \{0,1\} \times Z_{2p'q'}$. Take any $h \in Z_n^*$ and $\sigma = h^d \bmod n$. Define a random variable $\bar{\theta}_{b,t} = (-1)^b g^t \sigma \bmod n$. Since multiplication by $\sigma$ is a permutation in $Z_n^*$, we have

$$\{\bar{\theta}_{b,t}\}_{(b,t)\leftarrow\{0,1\}\times Z_{2p'q'}} \equiv Z_n^*$$

Since $Z_{n/2} \approx_S Z_{2p'q'}$, the above implies that

$$\{\bar{\theta}_{b,t}\}_{(b,t)\leftarrow\{0,1\}\times Z_{n/2}} \approx_S Z_n^*$$

Because the proportion of elements in $Z_n$ which are divisible by $p'$ or $q'$ is $O(2^{-\kappa''})$, we have $Z_n^* \approx_S Z_n$. Therefore

$$\{\bar{\theta}_{b,t}\}_{(b,t)\leftarrow\{0,1\}\times Z_{n/2}} \approx_S Z_n$$

Finally, if $k$ is chosen uniformly in $[0, ..., \lfloor 2^{2\kappa''+\kappa}/n \rfloor]$, this implies that for every $2\kappa''$-bit $n$ we have

$$\{\bar{\theta}_{b,t} + kn\}_{(b,t,k)\leftarrow\{0,1\}\times Z_{n/2}\times Z_{\lfloor 2^{2\kappa''+\kappa}/n\rfloor}} \approx_S \{0,1\}^{2\kappa''+\kappa}$$

Therefore the difference between the distribution of values $\theta_i^\tau$ in the protocol execution and a simulation where these values are chosen uniformly among $(2\kappa'' + \kappa)$-bit strings, is at most $O(2^{-min(\kappa,\kappa'')})$. Since there are $q_s$ sessions the difference between the distribution of adversary's view of an interaction with the network and an interaction with a simulator is at most $q_s O(2^{-min(\kappa,\kappa'')})$. □

## 5 Affiliation-Hiding AGKA Scheme based on the Diffie-Hellman Problem

We present the DH-based AH-AGKA scheme. Due to space constraints we present only the scheme, a sketch of correctness, and the statements of theorems about its security and affiliation-hiding. The proofs will be included in a full version of this paper [13]. We note that the proof of security of this AH-AGKE protocol follows a similar logic to the proof of security of the RSA-based AH-AGKA protocol in the previous section, but it includes rewinding.

- Setup: The setup algorithm outputs the standard discrete logarithm parameters $(p, q, g)$, i.e., primes $p, q$ of size polynomial in $\kappa$, s.t. $g$ is a generator of a subgroup in $Z_p^*$ of order $q$. We also define hash functions $H_q : \{0,1\}^* \rightarrow Z_q$, $\bar{H}_q : \{0,1\}^* \rightarrow Z_q$, and $H : \{0,1\}^* \rightarrow \{0,1\}^\kappa$.

- KGen: The secret key is chosen as a random number $x \in Z_q$ and the public key is $y = g^x \pmod{p}$.

- Add: For any user $U$ in the group, $\mathcal{CA}$ computes the certificate $\mathsf{cert}$ as a Schnorr signature [15] on an empty message under the key $y$, namely $\mathsf{cert} = (w, t)$ where $w = g^r \pmod{p}$, and $t = r + xH_q(w) \pmod{q}$, for random $r \leftarrow Z_q$. Note that $(w, t)$ satisfies equation $g^t = wy^{H_q(w)} \pmod{p}$.

- Revoke: To revoke user $U$, the $\mathcal{CRL}$ is appended with (hash of) $w$, where $(w, t)$ was $U$'s certificate.

- Handshake: This is an AGKA protocol executed by some set $\Delta = \{U_1, ..., U_n\}$ of players. Each player $U_i$ starts a session $\Pi_i^{\tau_i}$ for a (locally) fresh $\tau_i$, on *some* inputs $(\mathsf{cert}_i, y, \mathcal{CRL}_i)$ s.t. $y$ is some public key, $\mathsf{cert}_i = (w_i, t_i)$ is $U_i$'s certificate for this public key $y$, i.e. $\mathsf{cert}_i \in \mathsf{Certs}(y)$, and $\mathcal{CRL}$ is the (hopefully recent) CRL for group $\mathsf{Group}(y)$. The Handshake protocol is in Figure 2 below.

The inputs of instance $\Pi_i^{\mathcal{T}}$ of player $U_i$ are $\mathsf{cert}_i = (w_i, t_i)$, $y$, and $\mathcal{CRL}_i$. Note that $g^{t_i} = w_i y^{H_q(w_i)}$.

**[Round 1]**: Player $U_i$ picks $\mu_i \leftarrow \{0,1\}^{3\kappa}$, and broadcasts $(w_i, \mu_i)$

- Assume that player $U_i$ received $n$-1 messages $(w_1, \mu_1)$, ..., $(w_{i-1}, \mu_{i-1})$, $(w_{i+1}, \mu_{i+1})$, ..., $(w_n, \mu_n)$ in Round 1. (This is a simplification as in Figure 1. See footnote 4.)
  If any two messages contain the same value $\mu_j$ or the same value $w_j$, player $U_i$ rejects.
- $U_i$ sets $s = (y, \{w_j, \mu_j\}_{j=1,\ldots,n})$.
- If $w_j \in \mathcal{CRL}_i$ for any $j$ then $U_i$ picks a random value $X_i$ in $Z_q$ and sets $\mathsf{reject} = T$. Otherwise, $U_i$ computes $X_i = \bar{H}_q((z_{i+1})^{t_i}, s)/\bar{H}_q((z_{i-1})^{t_i}, s) \pmod q$ where $z_{i-1} = w_{i-1} y^{H_q(w_{i-1})}$ and $z_{i+1} = w_{i+1} y^{H_q(w_{i+1})}$.
  (Note that if $(w_{i-1}, t_{i-1})$ and $(w_{i+1}, t_{i+1})$ are certificates under key $y$ then $z_{i\pm1} = g^{t_{i\pm1}}$.)

**[Round 2]**: Player $U_i$ broadcasts $(X_i, \mu_i)$.

- If in Round 2 player $U_i$ receives $n$-1 values $X_j$ accompanied by $\mu_j$'s that match the $\mu_1, ..., \mu_{i-1}, \mu_{i+1}, ..., \mu_n$ values above, and if $\mathsf{reject} \neq T$, then $U_i$ computes $k_i = \bar{H}_q((z_{i-1})^{t_i}, s)^n \cdot X_i^{n-1} \cdot X_{i+1}^{n-2} \cdots X_{i-2} \pmod q$ and outputs $K_i = H(k_i, \mathsf{sid}_i)$, where $\mathsf{sid}_i = (y, \{w_j, \mu_j, X_j\}_{j=1,\ldots,n})$. Otherwise $U$ rejects.

**Figure 2. DH-based Affiliation-Hiding AGKA protocol**

**Protocol Correctness.** Similarly to the correctness argument for the RSA-based protocol, if $n$ instances $\Pi_i^{\mathcal{T}}$ are executed on the same public key $y$ and their messages are properly exchanges, they output same values $s_i^{\mathcal{T}}$, $\mathsf{sid}_i^{\mathcal{T}}$, and they all compute the same key material

$$k_i^{\mathcal{T}} = \bar{H}_q(g^{t_{i-1}t_i}, s) * \bar{H}_q(g^{t_i t_{i+1}}, s) * ... * \bar{H}_q(g^{t_{i-2}t_{i-1}}, s) \bmod q$$

where $t_i$'s are defined by the first message as in Figure 2. Therefore all partnered sessions also output the same keys $K_i^{\mathcal{T}}$.

**Theorem 3.** *The AH-AGKA scheme defined by the above tuple* $(\mathsf{Setup}, \mathsf{KGen}, \mathsf{Add}, \mathsf{Revoke}, \mathsf{Handshake})$ *is affiliation-hiding.*

**Theorem 4.** *Assuming that the GSDH problem in a subgroup generated by $g$ in $Z_p^*$ is $(\epsilon', t')$-hard, the AH-AGKA scheme defined by the above tuple* $(\mathsf{Setup}, \mathsf{KGen}, \mathsf{Add}, \mathsf{Revoke}, \mathsf{Handshake})$ *is $(\epsilon, t, q_s, q_H, l, m)$-secure in the Random Oracle Model for*

$$\begin{aligned} \epsilon &\approx c_\epsilon * (\epsilon' + (lmq_H/q + (q_s)^2 2^{-3\kappa})) \\ t &\approx t' - c_t * lq_s(q_H)^2 t_{\mathsf{exp}} \end{aligned}$$

*where $t_{\mathsf{exp}}$ is a cost of exponentiation in the subgroup generated by $g$ and $c_t, c_\epsilon$ are small constants, assuming the cost of accessing the DDH oracle is constant.*

## References

[1] 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*, Oakland, CA, May 2003.

[2] M. Bellare, A. Boldyreva, A. Desai and D. Pointcheval. Key-privacy in public-key encryption. In *Advances in Cryptology - ASIACRYPT 2001*, 2001.

[3] M. Bellare, R. Canetti, and H. Krawczyk. A modular approach to the design and analysis of authentication and key-exchange protocols. In *30th STOC'01*, 2001.

[4] M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated Key Exchange Secure Against Dictionary Attacks In *Advances in Cryptology - EUROCRYPT 2000*, 2000.

[5] D. Boneh, H. Shacham, and B. Lynn. Short signatures from the Weil pairing. In *J. of Cryptology*, vol. 17, no. 4, pp. 297-319, 2004.

[6] E. Bresson, O. Chevassut, D. Pointcheval, and J. Quisquater. Provably Authenticated Group Diffie-Hellman Key Exchange In *Proceedings of the 8th ACM conference on Computer and communications security (CCS'01)*, 2001

[7] M. Burmester and Y. Desmedt. A secure and efficient conference key distribution system. In *Advances in Cryptology - EUROCRYPT 1994*, 1994.

[8] R. Canetti and H. Krawczyk. Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels. In *Advances in Cryptology - CRYPTO 2001*, 2001.

[9] C. Castelluccia, S. Jarecki, and G. Tsudik. Secret handshakes from ca-oblivious encryption. In *Advances in Cryptology - ASIACRYPT 2004*, 2004.

[10] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan. Private Information Retrieval In *Journal of the ACM*, Volume 45, Issue 6, Pages:965-981, 1998

[11] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The Second-Generation Onion Router In *13th USENIX Security Symposium*, August 2004.

[12] S. Jarecki, J. Kim, and G. Tsudik. Authentication for Paranoids: Multi-Party Secret Handshakes. In *ACNS'06*, June 2006.

[13] S. Jarecki, J. Kim, and G. Tsudik. Group Secret Handshakes or Affiliation-Hiding Authenticated Group Key Agreement. To appear on IACR eprint archives (http://eprint.iacr.org), 2007.

[14] J. Katz and M. Yung. Scalable Protocols for Authenticated Group Key Exchange In *Advances in Cryptology - ASIACRYPT 2003*. 2003

[15] C. Schnorr. Efficient identification and signatures for smart cards. In *Advances in Cryptology - CRYPTO 1989*, 1989.

[16] V. Shoup. On Formal Models for Secure Key Exchange. In *Theory of Cryptography Library*, 1999.

[17] G. Tsudik and S. Xu. A Flexible Framework for Secret Handshakes. In *Privacy-Enhancing Technologies Workshop (PET'06)*, June 2006. Earlier version appeared as a Brief Announcement in *ACM PODC'05*, August 2005.