Deniable Unique Group Authentication
CS380S Project Report

Matteo Pontecorvi, Subhashini Venugopalan

Abstract

Deniable unique authentication is desirable in a number of scenarios. Electronic voting would be a prime example where it is necessary for a user to be anonymous and still be able to cast a vote not more than once. A stronger definition of anonymity is deniability; here we would like an honest voter to remain anonymous even when the secret keys of all users (including the voter herself, and the servers) are compromised after the election. The primary challenge is to provide deniability along with uniqueness in the presense of malicious entities. In this project we study a number of cryptographic primitives that satisfy some combination of the properties of anonymity, uniqueness and deniability. We analyze the protocol proposed in [11] and demonstrate an attack on deniability. This report also provides a fix and proves a modification of the protocol to be both unique and deniable.

1 Introduction

The use of anonymous message protocols is extremely important in many real-world applications. Consider, for example, an online voting scheme. Each voter can choose between a number of preferences, and some specific properties should be guaranteed at the end of the scheme (this is a list of the most important):

- each voter can decide to participate or not to the voting scheme,
- every regular vote submitted must be anonymous,
- each voter (malicious or not) can vote at most once,
- it's impossible for any observer (malicious or not) to distinguish if an honest voter participated the voting scheme (even if the voter regularly participated),
- it's impossible, for an adversary, to forge or modify the vote of any honest voter.

These conceptual properties can be formally expressed using several security games and we'll discuss them in the next sections. For the moment we give the first conceptual definitions that captures our goals:

Anonymity - The anonymity property we would like to achieve is that, the messages must appear to have come from any entity in the network, i.e. it must not be possible to identify (from the messages) that some client is a non-participant. This is called strong anonymity. We can also consider a relaxed definition of anonymity where it is sufficient for the messages to appear to be from participating clients, instead of strong anonymity, where it's likely for the messages to have come from any entity in the network.

Deniability - It must not be possible for any entity to prove the participation of any client in the protocol. This property must be achieved even if the long term secrets of all entities are revealed at the end of the round.

Uniqueness - In a single round of the protocol, no member must be able to participate more than once.

Unforgeability - It's impossible for an adversary to forge a message for an inactive honest client.

The goal of this project is to investigate the existence of a *complete protocol* that satisfies all the security properties (we call it CMP-protocol), or to identify which subset of properties can be satisfied by a *partial protocol* (we call it PRT-protocol) using modern cryptographic techniques.

1.1 Previous Work

There are many cryptographic primitives and protocols available for each of the properties listed previously, however, none of them have been able to achieve all the three together. Infact, achieving both Uniqueness and Deniability in the same context appears to be challenging. We report in table 1 some classes of available schemes and the properties they satisfy.

Primitive/Properties	**Uniqueness**	**Anonymity**	**Deniability**	**Comments**
Ring Signatures	×	✓	×	
Group Signatures	✓	✓ (But not from group manager)	×	Group anonymity only.
Linkable Ring Signs	✓	✓	×	
Deniable Ring Signs	×	✓	✓	
e-Cash, digital coins (blind signatures)	✓ (Indirectly)[1]	✓	✓	Assuming coin distribution is easy.
Dissent	✓	✓	×	

Table 1: Schemes and properties satisfied.

Generic signature schemes satisfying the desired properties:

Properties	**Uniqueness**	**Anonymity**	**Deniability**
Uniqueness	e-cash, digital coins	Linkable/Unique Ring, Untraceable e-cash	
Anonymity	Linkable/Unique ring Untraceable e-cash	Ring Signature	Deniable Ring
Deniability		Deniable Ring	Deniable authentication

Table 2: Schemes for a combination of properties.

Properties	**Uniqueness**	**Anonymity**	**Deniability**
Uniqueness	[6]	[7, 2, 16]	
Anonymity	[7, 2, 16]	[14]	[13, 1]
Deniability		[13, 1]	[10]

Table 3: Sample schemes satisfying different properties.

Specific references to signature schemes for a combination of the desired properties in Table 2 is presented in Table 3. The table only presents a few representative samples of the generic schemes based on the desired properties. There are several constructions for each of the authentication classes mentioned in table 2.

Organization This report is organized as follows: in section 2, we describe a practical network model and the security games for all the main properties. In section 3 present some of the related cryptographic schemes that achieve a subset of properties providing PRT-protocols. In section 4, we fully describe the protocol in [11] for the described network model. The complete analysis of the protocol is given in section 5; an attack is presented in section 6. In section 7 we show a modified version of the original algorithm and we prove that it's a CMP-Protocol. Conclusions for this project are in section 8.

2 Definitions

2.1 Network and Security Model

The system (based on [11]) is a communication network where there are n clients and m servers ($n+m$ *nodes* or *entities* in total). The communication network is *scrambled* meaning that an adversary cannot directly check the full route of messages sent from honest entities

[1]By giving only one coin to each participant, we can ensure uniqueness. It is also important that the coins are all identical otherwise it can break anonymity.

by performing network analysis, otherwise an adversary could immediately tell if a honest client is participating to the scheme. An entity in the network can send a message to another specific entity, or it's allowed to *broadcast* the message to all the nodes. In general we assume that

Definition 2.1. In a *scrambled* network, if the adversary $\mathcal{A}$ intercepts a message m then $\mathcal{A}$ cannot learn the source and/or destination of m by performing only network analysis.

Another strong assumption is that the network is *reliable*. In this scenario an adversary can only read the information that are travelling on the network but cannot change them. It's easy to see that without this assumption a powerful adversary can always delete every message from the network denying any possible service: a D.o.S attack (which is out of scope for this project).

Definition 2.2. In a *reliable* network, every sent message always reaches its destination unmodified.

Every node i in the network has a long term key-pair $K_i = (SK_i, PK_i)$ where SK_i is the long term secret key for node i, and PK_i is the long term public key for node i. Also, every node i has a unique network location L_i; this allows the messages to be delivered to specific nodes. We call the set of clients nodes $Clt = \{C_1, C_2, \ldots, C_n\}$ and the set of servers nodes $Srv = \{S_1, S_2, \ldots, S_m\}$. The following vectors are public and they are known to all entities in the network.

$$\begin{aligned} \overrightarrow{PK} &= (PK_{C_1}, \ldots, PK_{C_n}, PK_{S_1}, \ldots, PK_{S_m}) \\ \overrightarrow{L_S} &= (L_{S_1}, \ldots, L_{S_m}) \end{aligned}$$

If a client C_i participates in the protocol then we say that C_i is *active*, otherwise C_i is *inactive* (performs no communication). All the servers are active, so they always participate in every step of the protocol. The adversary is a PPT algorithm $\mathcal{A}$ that has full control on a proper subset of servers $MalSrv \subset Srv$ (at least one server is honest), and full control of at most $n-2$ clients

$$MalClt \subset Clt, \ |MalClt| \leq n-2$$

When a node C_i is controlled by an adversary $\mathcal{A}$ we write $C_i^{\mathcal{A}}$ with $1 \leq i \leq n$; similarly we write $S_j^{\mathcal{A}}$ with $1 \leq j \leq m$.

The goal of a CMP-protocol (PRT-protocol) is to complete at least one round of communication trying to maintain all (some of) the properties stated in section 1 and formally defined in subsection 2.3. During this round, each entity in the network is allowed to generate ephemeral random values (or keys) in order to achieve security; moreover each entity can delete all such values before the end of the round. After the end of the round, the adversary $\mathcal{A}$ compromises all the nodes in the network (clients and servers, active or inactive), learning all the informations (including the long term secret keys of the honest nodes) not erased before the end of the round.

We present,in this network model, the formal security games of the conceptual properties mentioned in the introduction. Let $\mathcal{M}$ be the message space for the protocol, and *Loc* be the set of locations. We always refer to a single message m in the following definitions, but a protocol could exchange more than a message in order to vote; in this case m will represent the vector of messages sent by the client to establish a vote (or in general a communication). Let ν be the security parameter for the protocol. The oracles involved in the following games are:

- $\mathcal{MO} : Clt \rightarrow \mathcal{M}$. The oracle $\mathcal{MO}$ receives a client ID C_i and returns the message $m \in \mathcal{M}$ sent by C_i during the protocol round.
- $\mathcal{LO} : Clt \rightarrow Loc$. The oracle $\mathcal{LO}$ receives a client ID C_i and returns the unique network location $L_i \in Loc$ of client C_i.
- $\mathcal{VO} : Clt \times \overrightarrow{\mathcal{R}} \times \mathcal{M} \rightarrow \mathcal{M}$. The oracle $\mathcal{VO}$ receives a client ID C_i, the vector $\overrightarrow{\mathcal{R}}$ of public round variables generated by the protocol and a message m. It returns a valid message $m' \in \mathcal{M}$ where m' has the same *meaning*[1] of m, but is generated by client C_i.

2.2 Authentication scheme definitions

Digital Signature Scheme A digital signature scheme is a triplet (Gen, Sign, Vrfy) where

- $\text{Gen}(1^\lambda) \rightarrow (PK, SK)$. Gen is a key generating algorithm that takes as input 1^λ, where λ is the security paramater, and returns the public key PK and secret key SK.
- $\text{Sign}(SK, m) \rightarrow \sigma$. Sign is a signing algorithm that takes as input the secret key SK and message m and returns a signature σ.
- $\text{Vrfy}(PK, m, \sigma) \rightarrow \{0, 1\}$. The verification algorithm takes as input the public key PK, the message m and the signature σ and outputs 1 if the signature is valid and 0 otherwise.
- Correctness: For all key pairs (PK, SK) that Gen might produce, and for all messages m, $\Pr[\text{Vrfy}(PK, m, \text{Sign}(SK, m)) = 1] = 1$.

The security for signature schemes is defined in terms of the chosen message attack game. The adversary is given a public key, access to a signing oracle, and is challenged to produce a forgery on any message for which he did not already query the oracle. No probabilistic polynomial time (PPT) adversary must be able to succeed in the game with better than negligible probability.

[1] In case of a voting scheme, m' contains the same vote of m.

Chosen Message Attack A digital signature scheme (Gen, Sign, Vrfy) is secure against existential forgery under adaptive chosen-message attacks if for any PPT adversary $\mathcal{A}$ the probability $p(\lambda)$ that $\mathcal{A}$ succeeds in the following game satisfies $p(\lambda) \leq negl(\lambda)$:

- (PK, SK) $\leftarrow$ Gen(1^λ). PK is given to $\mathcal{A}$.
- $\mathcal{A}$ has access to a signing oracle SO (adaptively queried) that receives a secret key SK and a message m, and outputs a signature $\sigma \leftarrow$Sign(SK, m) on message m.
- $\mathcal{A}$ returns a (forged) signature σ^* for a message m^* and a public key PK and succeeds if m^* was never queried to the SO and Vrfy$(PK, m^*, \sigma^*) = 1$

2.3 Security Games

This subsection contains the principal properties that a complete protocol should satisfy.

Anonymity Defines the impossibility to link a message (or a vote) to a specific client (or voter). An adversary $\mathcal{A}$ cannot distinguish if a message m has been sent from C_i rather than C_j. Formally speaking we consider the following game $\mathcal{G}_a$:

1. The adversary $\mathcal{A}$ can query the oracle $\mathcal{LO}$ for any client in Clt.
2. The adversary $\mathcal{A}$ chooses two messages m_0 and m_1 and two honest clients C_i and C_j, with $1 \leq i < j \leq n$, and send them to an oracle $\mathcal{O}$.
3. The oracle $\mathcal{O}$ chooses a random bit b and gives message m_b to C_i and $m_{\bar{b}}$ to C_j.
4. The protocol runs a full round where client C_i sends message m_b and client C_j send message $m_{\bar{b}}$ on then network.
5. At the end of the round, the adversary $\mathcal{A}$ learns all the information not erased by the honest entities (including every long term key pair in the protocol). Moreover $\mathcal{A}$ can query the oracle $\mathcal{MO}$ for every honest client in $Clt - \{C_i, C_j\}$.
6. The adversary $\mathcal{A}$ outputs a bit value b'.

The adversary $\mathcal{A}$ wins the anonymity game $\mathcal{G}_a$ if

$$\Pr[b' = b] > \frac{1}{2} + \text{negl}(\nu)$$

Roughly speaking, the adversary can only randomly guess what message has been sent from one of the two honest clients. We also observe that $\mathcal{A}$ can immediately ask for the location of all the clients which are not publicly available.

Deniability Defines the ability of an honest client to deny participation in the protocol. An adversary $\mathcal{A}$ cannot prove if a client C_i participated to the round even if the client C_i really participated. Formally speaking we consider the following game $\mathcal{G}_d$:

1. The adversary $\mathcal{A}$ chooses two honest clients C_i and C_j, with $1 \leq i < j \leq n$, and send them to an oracle $\mathcal{O}$.

2. The oracle $\mathcal{O}$ chooses a random bit b and computes $b_1 = bi + \bar{b}j$ and $b_2 = \bar{b}i + bj$. Then $\mathcal{O}$ set client C_{b_1} active and client C_{b_2} inactive.

3. The adversary $\mathcal{A}$ can query the oracle $\mathcal{LO}$ for any honest client in $Clt - \{C_i, C_j\}$. The adversary also receives from $\mathcal{LO}$ the location L of the active client in the set $\{C_i, C_j\}$.

4. The protocol runs a full round where all the active clients send a message.

5. At the end of the round, the adversary $\mathcal{A}$ learns all the information not erased by the honest entities (including every long term key pair in the protocol). Moreover $\mathcal{A}$ can query the oracle $\mathcal{MO}$ for every honest client in $Clt - \{C_i, C_j\}$. The adversary also receives from $\mathcal{MO}$ the message m sent by the active client in the set $\{C_i, C_j\}$.

6. The adversary $\mathcal{A}$ outputs a bit value b'.

The adversary $\mathcal{A}$ wins the deniability game $\mathcal{G}_d$ if

$$\Pr[b' = b] > \frac{1}{2} + \text{negl}(\nu)$$

Roughly speaking, the adversary can only randomly guess what is the state for one of the two honest clients. We also observe that $\mathcal{A}$ can ask for the location and the message sent by the active client in $\{C_i, C_j\}$ without receiving any additional information about the state of the two honest clients.

Uniqueness Define the impossibility for a malicious entity to send more then a vote for an inactive client. Formally speaking we consider the following games $\mathcal{G}_u$:

1. The adversary $\mathcal{A}$ collect any information (Long term key pairs, ephemeral states, locations) from all the clients and all the servers except for the long term key pair and the private ephemeral state of a honest server.

2. The adversary $\mathcal{A}$ runs a protocol round impersonating all the clients and outputs $n+1$ messages on the network.

The adversary wins the uniqueness game $\mathcal{G}_u$ if all the $n+1$ messages are valid for the protocol, meaning that every subset of size n of them is always accepted in the same protocol round.

Unforgeability Define the impossibility for a malicious entity to forge a vote acting as a honest inactive client. Formally speaking we consider the following games $\mathcal{G}_u$:

1. The adversary $\mathcal{A}$ choose a honest inactive client C_i.
2. The adversary $\mathcal{A}$ collect any information (Long term key pairs, private ephemeral state, location) from all the clients and all the servers except from C_i and a honest server S_j.
3. The adversary can query the oracle $\mathcal{VO}$ for any client in $Clt - \{C_i\}$, any message m and any vector of public variables R. Let M' be the set of messages generated by the oracle $\mathcal{VO}$ in this step.
4. The adversary $\mathcal{A}$ runs a protocol where $Clt = C_i$ (C_i is the only client) and outputs a messages $m \notin M'$ on the network.

The adversary wins the unforgeability game $\mathcal{G}_{uf}$ if the message is accepted by the protocol.

2.4 Other Security Games

This subsection contains all that properties that are less important for many applications (i.e. voting or election schemes) or can be easily achieved with the use of standard techniques.

Location Anonymity Defines the impossibility to link a location to a specific client. We consider the following game $\mathcal{G}_{la}$:

1. The adversary $\mathcal{A}$ chooses two locations L_0 and L_1 and send them to an oracle $\mathcal{O}$.
2. The oracle $\mathcal{O}$ chooses two honest clients C_i and C_j with $1 \leq i < j \leq n$ and a random bit b. Then $\mathcal{O}$ sets the location of C_i to be L_b, and the location of C_j to be $L_{\bar{b}}$. The oracle $\mathcal{O}$ communicates the chosen clients C_i and C_j to $\mathcal{A}$.
3. The adversary $\mathcal{A}$ can query the oracle $\mathcal{LO}$ for any client in $Clt - \{C_i, C_j\}$.
4. The protocol runs a full round where C_i sends messages from location L_b and C_j sends messages from location $L_{\bar{b}}$.
5. At the end of the round, the adversary $\mathcal{A}$ learns all the information not erased by the honest entities. Moreover $\mathcal{A}$ can query the oracle $\mathcal{MO}$ for every client in Clt.
6. The adversary $\mathcal{A}$ outputs a bit value b'.

The adversary $\mathcal{A}$ wins the location anonymity game $\mathcal{G}_{la}$ if

$$\Pr[b' = b] \leq \frac{1}{2} + \text{negl}(\nu)$$

Roughly speaking, the adversary can only randomly guess the location from where one of the two honest clients is sending messages.

3 Related Schemes and Properties

In this section, we present some well-known cryptographic schemes that can achieve some of the important properties (Anonymity, Deniability, Uniqueness, Unforgeability) in their settings. For clarification, we also define the properties when they vary from our definitions.

3.1 Ring Signature

A ring signature, introduced by Rivest *et al.* [14], is a form of digital signature which can be produced by any member of a ring/group of signers, all of who have a set of public/secret key pairs. The ring-signature only says that the message was signed by one member of the ring; moreover, it will not reveal anything about the signer itself. Additionally, the members of the ring can be arbitrary and the ring can be easily extended to include more people as long as they have their own public/private key pairs. In fact there is no need for the ring members to be endowed with any special property at all. The fact that ring signatures can maintain the anonymity of a single signer makes them extremely appealing for anonymous communication protocols.

Bender *et al.* [1], presented the first ring signature scheme that doesn't depend on random oracles for the security. They also introduced stronger definitions of anonymity, and were the first to propose the notions of anonymity against adversarially chosen public keys and anonymity against full key exposure.

We provide the formal definition of ring signature scheme. The structure is quite similar to digital signatures, however in addition to the secret key during signing procedure, one also uses the public keys of other users that will define the ring.

Ring Signature Scheme A ring signature scheme is a triple of PPT algorithms (Gen, Sign, Vrfy) where

- Gen$(1^\lambda)\rightarrow (PK, SK)$. Gen is a key generating algorithm that takes as input 1^λ, where λ is the security paramater, and returns the public key PK and secret key SK.

- Sign$(SK_{i^*},\ m,\ R = (PK_1, \ldots, PK_l)) \rightarrow \sigma$. Sign is a signing algorithm that takes as input a message m, a ring $R = (PK_1, \ldots, PK_l)$ of public keys and a secret key SK_{i^*} such that $PK_{i^*} \in R$ and returns a signature σ.

- Vrfy$(R = (PK_1, \ldots, PK_l),\ m,\ \sigma)\rightarrow \{0, 1\}$. The verification algorithm takes as input the ring of public keys R, the message m and the signature σ and outputs 1 if the signature is valid (was signed using the secret corresponding to one of the public keys in R) and 0 otherwise.

- Correctness: For any set of key pairs $\{(PK_i, SK_i)\}_{i=1}^{l}$ that Gen might produce, and for any messages m and user $i^* \in [1, l]$, Pr[Vrfy$(R, m, Sign(SK_{i^*}, m, R))$=1]=1, where $R = (PK_1, \ldots, PK_l)$.

We now give the strongest unforgeability definition given in [1], which is referred to as unforgeability with respect to insider corruption. Informally, the definition is meant to capture even the case where the adversary adaptively corrupts honest participants and obtains their secret keys and can query for signatures on any message on any ring as any user of the ring. The requirement is that, the adversary succeeds in producing a forgery on a message and ring (none of whose secret keys the adversary has) with only negligible probability.

Unforgeability A ring signature scheme (Gen, Sign, Vrfy) is unforgeable with respect to insider corruption if for any PPT adversary $\mathcal{A}$, the probability $p(\lambda)$ that $\mathcal{A}$ succeeds in the following game satisfies $p(\lambda) \leq negl(\lambda)$:

- Key pairs $\{(PK_i, SK_i)\}_{i=1}^{l}$ are generated using $\text{Gen}(1^\lambda)$, and the set of public keys $PK = \{PK_i\}_{i=1}^{l}$ is given to $\mathcal{A}$.
- $\mathcal{A}$ has access to a signing oracle SO which it can query adaptively as $\text{SO}(SK_s, m, R)$ to get a signature $\sigma \leftarrow \text{Sign}(SK_s, m, R)$ on any message m where $PK_s \in R$.
- $\mathcal{A}$ is also given a $Corrupt$ oracle where $Corrupt(i)$ outputs SK_i.
- $\mathcal{A}$ returns a (forged) signature (R^*, m^*, σ^*) for a message m^* and succeeds only if $\text{Vrfy}(R^*, m^*, \sigma^*) = 1$ and $\mathcal{A}$ never queried the signing oracle for (SK^*, m^*, R^*) and $R^* \subseteq PK \setminus C$, where C is the set of corrupted users.

Anonymity When we normally refer to anonymity, we assume that there are atleast two honest users in the ring (whose keys the adversary doesn't know) and hence the adversary cannot identify the exact signer. This definition is logical because if the adversary had keys for all except one signer, and he knows that he did not sign the message then this automatically identifies the signer of the message. We now present the strong anonymity definition presented in [1], anonymity against full key exposure. In this game, the adversary is given a signing oracle as in in the unforgeability game and also has access to the secret keys of all honest users in the ring.

Anonymity against full-key exposure (Deniability) A ring signature scheme (Gen, Sign, Vrfy) is anonymous against attribution attacks if for any PPT $\mathcal{A}$ and polynomial l, the probability $p(\lambda)$ that $\mathcal{A}$ succeeds in the following game satisfies $|p(\lambda) - \frac{1}{2}| \leq negl(\lambda)$

- Key pairs $\{(PK_i, SK_i)\}_{i=1}^{l} \leftarrow Gen(1^\lambda; \omega_i)$ are generated randomly, and the set of public keys $S = \{PK_i\}_{i=1}^{l}$ is given to $\mathcal{A}$. (ω_i is the randomness used to generate keys for each user i).
- $\mathcal{A}$ has access to a signing oracle SO which it can query adaptively as $\text{SO}(SK_s, m, R)$ to get a signature $\sigma \leftarrow \text{Sign}(SK_s, m, R)$ on any message m as long as $PK_s \in R$.

- $\mathcal{A}$ chooses a message m, a ring R, and indices of two users i_0 and i_1 such that PK_{i_0} and PK_{i_1} are both in R.

- A random bit β is selected and $\mathcal{A}$ is given a signature $\sigma \leftarrow Sign(SK_{i_\beta}, m, R)$ as well as $\{\omega_i\}_{i=1}^{l}$ (includes $i = i_0$ and $i = i_1$, and all other users in the ring).

- The adversary outputs a bit β', and succeeds if $\beta' = \beta$

Note that the above definition also encompasses deniability in the strong sense. i.e. Even when the user's secret keys are exposed, the signature cannot be linked to the user.

3.2 Deniable Ring Signature

In this section we present a sample deniable ring signature which satisfies both anonymity and deniability (for full key exposure) for a 2 user ring. The construction is from [1] and uses Waters' signature scheme [17] which we will describe briefly.

Waters Signature

Key Generation. Choose $\alpha \leftarrow \mathbb{Z}_q$ and set $g_1 = g^\alpha$. Then, choose elements $h, u, u_1, \ldots, u_n \xleftarrow{R} \mathbb{G}$, uniformly at random. Choose e, a bi-linear map function that takes two inputs from the group $\mathbb{G}$ and outputs an element in group $\mathbb{G}_1$. The public key $PK = (g_1, h, u, u_1, \ldots, u_n)$ and the secret key $SK = h^\alpha$.

Signing. To sign an $n-$bit message m, first compute $w = u \cdot \prod_{i:m_i=1} u_i$. Choose a random value $r \xleftarrow{R} \mathbb{Z}_q$ and output the signature $\sigma = \langle \sigma_0, \sigma_1 \rangle = \langle h^\alpha \cdot w^r, g^r \rangle$.

Verification. A signature σ on a message m is verified by first computing $w = u \cdot \prod_{i:m_i=1} u_i$ (from the public parameters PK). Then check if $e(g_1, h) \cdot e(\sigma_1, w) \stackrel{?}{=} e(\sigma_0, g)$ (where e is a bi-linear map function).

Two users deniable ring signature Consider the following construction from [1] for a two user deniable ring signature:

Key Generation. Choose $\alpha \leftarrow \mathbb{Z}_q$ and set $g_1 = g^\alpha$. Then, choose elements $u, u_1, \ldots, u_n \xleftarrow{R} \mathbb{G}$, uniformly at random. Choose $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_1$. The public key $PK = (g_1, u, u_1, \ldots, u_n)$ and the secret key $SK = \alpha$. Public parameters *params*=$\{q, \mathbb{G}, \mathbb{G}_1, e, g,$ and $PK\}$

Signing. To sign an $n-$bit message m with respect to the ring $R = \{PK, PK'\}$ using secret key α (where *alpha* is the secret corresponding to PK) is done as follows: Let $PK = (g_1, u, u_1, \ldots, u_n)$ and $PK' = (g_1', u', u_1', \ldots, u_n')$. First compute $w = u \cdot \prod_{i:m_i=1} u_i$ and $w' = u' \cdot \prod_{i:m_i=1} u_i'$. Choose a random value $r \xleftarrow{R} \mathbb{Z}_q$ and output the signature $\sigma = \langle \sigma_0, \sigma_1 \rangle = \langle g_1'^\alpha \cdot (ww')^r, g^r \rangle$.

Verification. A signature σ on a message m with respect ring $R = \{PK, PK'\}$ is verified by first computing $w = u \cdot \prod_{i:m_i=1} u_i$ (from the public parameters PK) and then computing $w' = u' \cdot \prod_{i:m_i=1} u'_i$ (from the public parameters PK'). Then check if $e(g_1, g'_1) \cdot e(\sigma_1, (ww')) \stackrel{?}{=} e(\sigma_0, g)$ (where e is a bi-linear map function).

Proof Sketch Deniability and Anonymity The scheme is unconditionally anonymous even on full key exposure. This can be observed from the fact that $g_1'^{\alpha} = g^{\alpha\alpha'} = g_1^{\alpha'}$. Thus, the same signature can be generated using either of the 2 secret keys $SK = \{\alpha\}$ or $SK' = \{\alpha'\}$.

Unforgeability (Proof Sketch) We can show that if there exists a PPT adversary $\mathcal{A}$ who can forge the ring signature without having the secret key of any member in the ring then we can create a polynomial time algorithm $\mathcal{B}$ that can solve the computational Diffie-Hellman assumption with non-negligible probability.

Let $\mathcal{B}$ receive the CDH challenge as $(g, X = g^x, Y = g^y)$. It then picks random values $p, p_1, \ldots, p_n, q, q_1, \ldots, q_n \stackrel{R}{\leftarrow} \mathbb{Z}_p^*$ and gives $\mathcal{A}$ the public keys of the 2 ring members as follows: $PK = (X, u = g^p, u_1 = g^{p_1}, \ldots, u_n = g^{p_n})$ and $PK' = (Y, u' = g^q, u'_1 = g^{q_1}, \ldots, u'_n = g^{q_n})$. Adversary $\mathcal{A}$ then submits a forgery $\sigma^* = \langle \sigma_0^*, \sigma_1^* \rangle$ for a message m^*. $\mathcal{B}$ checks that the signature is valid by computing $w = u \cdot \prod_{i:m_i=1} u_i = g^{p+\sum_{i:m_i^*=1} p_i}$ and $w' = u' \cdot \prod_{i:m_i^*=1} u'_i = g^{q+\sum_{i:m_i^*=1} q_i}$. Then checks, $e(X, Y) \cdot e(\sigma_1, (ww')) \stackrel{?}{=} e(\sigma_0, g)$. $\mathcal{B}$ then solves the CDH problem by getting $Z' = \sigma_0 / (\sigma_1)^{p+q \sum_{i:m_i^*=1} (p_i+q_i)}$.

3.3 Group Signature

A group signature scheme is a quadruple consisting of algorithms (Gen, Sign, Vrfy, Open) where

- Gen$(1^\lambda) \rightarrow (PK, \overrightarrow{SK}, SK_{mgr})$. The Gen algorithm takes as input 1^λ, where λ is the security paramater, and returns the group public key PK, and individual secret keys SK for each group member, and an administrative key for the group manager.

- Sign$(SK, m) \rightarrow \sigma$. Sign is a signing algorithm that takes as input a group member's secret key SK and message m and returns a signature σ on m.

- Vrfy$(PK, m, \sigma) \rightarrow \{0, 1\}$. The verification algorithm takes as input the group's public key PK, the message m and the signature σ and outputs 1 if the signature is valid and 0 otherwise.

- Open$(\sigma, SK_{mgr}) \rightarrow (ID, prf)$. The open algorithm takes as input a signature and the secret key of the group manager and returns a proof prf and the identity ID of the group member who produced that signature.

A group signature needs to satisfy the following properties:

- Unforgeability: Only group members should be able to produce valid signatures.

- Anonymity: Given a signature and the public parameters, it must not be possible to identify which group member signed the message. Note: Only a group manager can reveal the identity of the signer.

- Unlinkability: It must not be possible to link/decide whether two signatures were produced by the same group member.

- Framing attack security: Group members cannot circumvent the opening of a signature - group members including group managers cannot sign on behalf of other group members.

Comments We compare group signatures with respect to the properties that we desire:

1. Anonymity: Not satisfied. Group signatures give anonymity but this can be revoked by a group manager.

2. Uniqueness: Satisfied. It is easy to achieve uniqueness with the help of a trusted manager.

3. Deniability: Not Satisfied. This will not offer deniability if the user's secret key is revealed. Moreover, the manager will always be able to identify the signer. Thus, if just the manager's secret keys are compromised, no member of the group can have deniability.

3.4 Linkable Ring Signature

A linkable ring signature scheme is a quadruple consisting of algorithms (Gen, Sign, Vrfy, Link) where

- Gen$(1^\lambda) \rightarrow (PK, SK)$. The Gen algorithm takes as input 1^λ, where λ is the security paramater, and outputs a public key PK, a secret keys SK.

- Sign$(SK, R, m) \rightarrow \sigma$. Sign is a signing algorithm that takes as input one secret key SK, and a ring consisting of n public keys which includes the public key PK corresponding to SK, and a message m and produces a signature σ on m.

- Vrfy$(R, m, \sigma) \rightarrow \{0, 1\}$. The verification algorithm takes as input the ring of public keys R, the message m and the signature σ and outputs 1 if the signature is valid (was signed using the secret corresponding to one of the public keys in R) and 0 otherwise. i.e. Vrfy$(R, m,$ Sign$(SK', R, m))= 1$, if $(SK', PK') \leftarrow$ Gen(1^λ) and $PK' \in R$.

- Link$(R, m_1, m_2, \sigma_1, \sigma_2) \rightarrow \{0, 1\}$. The Link algorithm takes as input a ring R of public keys, and two signatures σ_1 and σ_2 along with their corresponding messages and

outputs 1 (for linked) or 0 (for unlinked). It is required that $\text{Vrfy}(R, m_1, \sigma_1) = 1$ and $\text{Vrfy}(R, m_2, \sigma_2) = 1$. Now, for any $\sigma_1 \leftarrow \text{Sign}(SK_1, R, m_1)$ and $\sigma_2 \leftarrow \text{Sign}(SK_2, R, m_2)$:

$$Link(R, m_1, m_2, \sigma_1, \sigma_2) = \begin{cases} 1 & \text{if } SK_1 = SK_2 \\ 0 & \text{otherwise} \end{cases}$$

A linkable ring signature needs to satisfy the following properties:

- Unforgeability: The unforgeability property of a linkable ring signature is the same as for ring signatures. It must not be possible to produce a valid linkable ring signature without having the secret key corresponding to at least one of the public keys in the ring.

- Anonymity: Given a signature and the public parameters, it must not be possible to identify the signer with an advantage non-negligibly larger than $1/(n-k)$ where n is the size of the ring and k is the number of known secret keys. $(\forall n, k : n - k \geq 2)$.

- Linkability: This has two properties:

  – If two signatures were produced using the same secret key SK, they must never return 0 on Link.

  – (Framing) It must not be possible for a signer with a different key pair (PK', SK') to produce a signature that links ($\mathsf{link} = 1$) with the signature of a different signer with keys (PK, SK) when $PK \neq PK'$ and $SK \neq SK'$.

Comments

1. Anonymity: Satisfied. Linkable ring signatures gives a user full anonymity as long as the user does not sign more than once (some variants specify a threshold k).

2. Uniqueness: Satisfied. It is easy to enforce uniqueness here since it possible to link 2 signatures by the same member.

3. Deniability: Not Satisfied. This will not offer deniability in case that the user's secret key is revealed. An attacker with the secret key can trivially break deniability by creating another signature and checking if it links with some signature on the network.

3.5 E-Cash Schemes

As seen from Table 1, E-Coins (or in general E-Cash schemes) can solve the problem under specific conditions: the coin distribution is easy, it doesn't reveal any information about the coin receiver and it's possible to distribute exactly one coin to each client.

There are many E-cash schemes with different properties. The first scheme introduced by Chaum, Fiat and Naor (CFN) [8] is based on blind signatures introduced by Chaum [9]. It's

based on RSA and allows double-spending (uniqueness) detection; it uses cut-and-choose proof techniques and an e-coin takes $O(k^2)$ bits, where k is the security parameter. The second scheme, called Brand's e-cash [3], uses groups of prime order and proves the security using the more reliable Sigma Protocol, and security reductions. An e-coins take only $O(k)$ bits. The last big improvement along this lines is the Camenisch, Hohenberger, Lysyanskaya (CHL) scheme [4]. This scheme allows the clients to withdraw W coins from the bank and store them in $O(k + logW)$ bits space; moreover the security of this scheme is proved using Zero-Knowledge proofs which provides greater privacy to the users. CHL is the first E-cash scheme proved to be secure. Another difference between CHL and the other two schemes is that, CHL is built using cryptographic building blocks in a black-box approach.

The main security goals for an E-Cash system are:

Unlinkability Suppose two honest users U_0 and U_1 both withdraw an e-coin from a malicious bank. We flip a coin $b \in \{0, 1\}$ and have U_b give the e-coin to a malicious merchant via the Spend protocol. Even if the malicious merchant and the malicious bank work together, they should not be able to guess whether the spent e-coin came from U_0 or U_1. This is a weak concept then pure Deniability, in fact in this case merchant and bank doesn't know the secret state of the client.

Anonymity Suppose an honest user withdraws an e-coin from a malicious bank. We flip a coin $b \in \{0, 1\}$; if $b = 0$, the user gives the e-coin to a malicious merchant, while if $b = 1$, we create a simulator that does not know anything about the e-coin, but has special powers (such as special knowledge about the system parameters), and the simulator gives an e-coin to the merchant. Even if the malicious merchant and the malicious bank work together, they should not be able to guess whether the e-coin came from a real user or a simulator.

Generic Balance or Uniqueness Suppose a group of malicious users and merchants execute a sequence of n withdraws. Then they should not be able to successfully execute $n + 1$ deposits.

Serial Number Balance or Unforgeability Suppose a group of malicious users and merchants execute a series of withdraws, and these executions are associated with serial numbers (e-coins IDs) $S_1, S_2, \ldots S_n$. Then, even if malicious merchants and users work together, they should not be able to successfully deposit an e-coin with serial number $S \notin \{S_1, S_2, \ldots S_n\}$.

There are other properties which are strictly related to the E-Cash world such as Identification, Strong and Weak Exculpability, but they are out of scope for our project. Unfortunately the current E-cash schemes does not satisfy all the properties: for example the CNF e-cash and Brand's e-cash are unlinkable while CHL e-cash is anonymous.

Every cash scheme is based on four cryptographic building blocks: proofs of knowledge, commitments, blind signatures and secret sharing. We give a brief description for each one

of them.

Proof of Knowledge In the course of executing an e-cash protocol, the user often has to prove that he *knows* something. We can express this in terms of a relation $\mathcal{R}$. The user gives the verifier a statement s and proves that he knows a witness w such that $(s, w) \in \mathcal{R}$. For example, the user might prove that he knows the pre-image of a hash function H. In this case, $\mathcal{R} = \{(s, w) : s = H(w)\}$. Proofs of Knowledge are usually achieved using cut-and-choose techniques, Sigma Protocols or Zero Knowledge Proofs.

Commitments A commitment is the cryptographic equivalent of a sealed envelope. Looking at a commitment reveals no information about the value inside. However, when the commitment's creator opens the commitment, he can take out only the value he put inside.

Blind Signatures Blind signatures allow the user to get a signature from the bank on a value without revealing that value to the bank. That value becomes the basis for constructing a unique serial number that the bank won't be able to link to a specific withdraw.

Secret Sharing E-cash systems use secret sharing to catch double-spenders. The client C breaks his public key PK_C into n pieces, of which at least two are needed to recover his identity. If he spends an e-coin with the same serial number more than once, the bank can compute PK_C. The most common secret sharing schemes are Arithmetic Secret Sharing and Shamir Secret Sharing [15].

We describe the CNF in details. Remember that in a E-Cash protocol there are clients (Alice), merchants (Bob) and a bank.

Setup : Two collision resistant hash functions h_1 and h_2, both $\mathbb{Z}_n^* \times \mathbb{Z}_n^* \to \mathbb{Z}_n^*$, are given and a security parameter k. Each entity (clients, merchants and bank) publishes its RSA public key.

Withdraw : Alice (client) withdraw one e-coin from the bank.

1. Alice contacts bank via some authenticated channel. The bank gives Alice PK_U that is a unique identifier of the withdraw phase (for example an account number concatenated with a counter).

2. Alice chooses $a_i, c_i, d_i \in \mathbb{Z}_n^*$, for $1 \leq i \leq 2k$, and computes $2k$ messages $(x_1, y_1), \ldots, (x_{2k}, y_{2k})$ where
$$x_i = h_2(a_i, c_i) \qquad y_i = h_2(a_i \oplus (PK_U||i), d_i)$$
This is basically the Arithmetic Secret Sharing construction. Then Alice blinds her messages: for each i, she chooses $r_i \in \mathbb{Z}_n^*$ and computes $M_i = r_i^3(h_1(x_i, y_i))$. Alice sends $M_1, \ldots, M_{2k}$ to the bank.

3. The bank uses cut-and-choose to verify that Alice properly encoded a_i and $a_i \oplus (PK_U||i)$. To do that, it chooses k random indices I and sends them to Alice. Alice takes each index $i \in I$ and sends (a_i, c_i, d_i, r_i) to the bank. The bank can now verify M_i and in general k such messages with the information received by Alice.

4. The bank now trust Alice and will sign the remaining messages M_j that Alice didn't revealed to the bank. Let $R = \{1, \ldots, 2k\} \setminus I$ the set of remaining indices. Observe that $|R| = k$. The bank computes and send to Alice S' where

$$S' = \prod_{i \in R} M_i^{\frac{1}{3}}$$

5. Alice finally computes the signature S on her e-coin (we are just signing one e-coin).

$$S = \frac{S'}{\prod_{i \in R} r_i^{\frac{1}{3}}} = \prod_{i \in R} h_1(x_i, y_i)^{\frac{1}{3}}$$

   Practically, the e-coin in Alice's wallet will appear as S together with $(a_i, c_i, d_i, r_i)_{i \in R}$.

Spend : Alice (client) gives one e-coin to Bob (merchant).

1. Alice send the e-coin signature S to Bob.

2. Bob sends Alice a cut-and-choose challenge of k random bits $b_1, \ldots, b_k$.

3. Alice constructs a *response* proving the signature S is properly formed.

   - If $b_i = 0$, Alice reveals x_i and sends Bob $x_i, a_i \oplus (PK_U||i), d_i$.
   - If $b_i = 1$, Alice reveals y_i and sends Bob y_i, a_i, c_i.

4. Since h_1 and h_2 are publicly available Bob has all the information to reconstruct x_i and y_i for every $i \in R$ and then compute S itself. Bob can finally check if the S sent by Alice is valid.

Deposit : Bob (merchant) deposit one e-coin in the bank.

1. Bob sends S, his challenge $\overrightarrow{b} = (b_1, \ldots, b_k)$ and the entire *response* from Alice (to that challenge), to the bank.

2. The bank checks if S is fresh. If S is new, the bank credits Bob's account and stores $(S, \overrightarrow{b}, \textit{response})$ in the bank database. If S is already in the database, this means that some other merchant (Charlie) has deposit the same e-coin before. If the challenges of Bob $\overrightarrow{b}$ and Charlie $\overrightarrow{b}'$ are different, then Bob and Charlie (the merchants) are honest.

Identify : The bank wants to punish a malicious client.

1. The bank already has $(S, \overrightarrow{b}, response)$ in its database, and receives $(S, \overrightarrow{b}', response')$ from a merchant. Then there exists at least one index i, for $1 \leq i \leq k$ where $b_i \in \overrightarrow{b}$ is different from $b'_i \in \overrightarrow{b}'$.

2. The bank knows, both a_i and $a_i \oplus (PK_U||i)$. Thus it can compute

$$a_i \oplus (a_i \oplus (PK_U||i)) = PK_U||i$$

and punish Alice, since the banks know to which client's account the value PK_U is associated.

Comments This scheme in Unlinkable according to the definitions previously given. Unfortunately unlinkability is not as strong as deniability. Regarding Uniqueness and Unforgeability, CFN e-cash system make a heuristic argument that every valid e-coin must have a valid signature, and the bank signs only one serial number per execution of the Withdraw protocol. Moreover for Anonymity, there is no proof that is impossible to distinguish between a real client and a ad-hoc simulator when a coin is given to a merchant.
E-cash schemes do not achieve Anonymity, Deniability and Uniqueness in the general sense.

4 The Randomness Scrubbing Scheme

We now present the protocol proposed in [11]. The network model and definitions used are as per section 2.

We assume that every node in the network knows the generator g for group $\mathbb{Z}_p$ of prime order p, where $p = 2q + 1$ for a sufficiently large prime q. Every operation described below is $\pmod p$.

4.1 Setup

Each client $C_i \in Clt$ has a private key $SK_{C_i} = x_i$ and a public key $PK_{C_i} = g^{x_i}$. Similarly each server $S_j \in Srv$ has a private key $SK_{S_j} = y_j$ and a public key $PK_{S_j} = g^{y_j}$. Let $\overrightarrow{PK}_{Clt} = (PK_{C_1}, \ldots, PK_{C_n})$ and similarly $\overrightarrow{PK}_{Srv} = (PK_{S_1}, \ldots, PK_{S_m})$ be the vectors of all clients and servers' public keys. We define the following vector $\overrightarrow{PK} = (\overrightarrow{PK}_{Clt}, \overrightarrow{PK}_{Srv})$.

At the beginning of the round, each server S_j picks a per-round random value r_j, then broadcasts a signed message $R_j = g^{r_j}$. Each active entity can easily compute the following vector $\overrightarrow{R} = (R_1, \ldots, R_m)$ by checking the signature attached to the messages. The vector $\overrightarrow{R}$ will serve as a *round identifier* for this round.

4.2 The Clients

From now on we refer only to a generic active client.

1. Client C_i picks a per-round generator h_i such that it's hard to establish the logarithmic relationship between g and any h_i. Moreover, for every i and i' with $1 \leq i < i' \leq n$, is hard to establish the logarithmic relationship between h_i and $h_{i'}$. Such value can be computed by each client, for example, with an hash function of a unique well-known subset of public variables for each client.

2. Client C_i picks a random ephemeral key z_i and computes $Z_i = g^{z_i}$. For each server S_j, the client picks the public key PK_{S_j} of server S_j and computes a secret exponent

$$s_j = H((PK_{S_j})^{z_i}) = H(g^{y_j z_i})$$

where $H : \mathbb{Z}_p^* \to \mathbb{Z}_p^*$ is a cryptographic hash function from a group element to an exponent.

3. Client C_i computes a linkage tag $T_i(0) = h_i^{s_1 s_2 \dots s_m}$ and for each $1 \leq j \leq m$ computes a set of commits for the servers $[SC_i(0), SC_i(1), \dots, SC_i(j), \dots, SC_i(m)]$ where

$$SC_i(0) = g \quad SC_i(1) = g^{s_1} \quad \dots \quad SC_i(j) = g^{s_1 s_2 \dots s_j} \quad \dots \quad SC_i(m) = g^{s_1 s_2 \dots s_m}$$

Let $\overrightarrow{SC}_i = (Z_i, SC_i(0), SC_i(1), \dots, SC_i(m))$ be the vector of commits sent by client C_i on the network.

4. Finally, client C_i chooses is message m_i and sends the following message to server S_1.

$$M_i(0) = \left(\Gamma_i(0) = \underbrace{(m_i, \overrightarrow{PK}, \overrightarrow{R}, \overrightarrow{SC}_i)}_{\text{context}}, \underbrace{T_i(0)}_{\text{tag}}, \underbrace{P_i(0)}_{\text{proof}} \right)$$

The context $\Gamma_i(0)$ and the tag $T_i(0)$ are clearly defined at this point. The proof $P_i(0)$ is necessary to prove the soundness of the entire message and is generated as follows.

Client Proof For a client C_i we define $s = s_1 s_2 \dots s_m$ to be the product of the secret exponents generated by C_i during step 2 of its protocol. Given the list of public keys $PK_{C_i} = g^{x_i}$ and corresponding per-round generators h_i for all clients C_i, the initial linkage tag $T_0 = h_{\hat{i}}^s$ of a particular authenticating client $C_{\hat{i}}$ and that client per-servers shared secret commits $\overrightarrow{SC}_{\hat{i}}$, the client $C_{\hat{i}}$ needs to prove to the server that there exists a client secret key $x \in \{x_1, x_2, \dots, x_n\}$ such that one of the following condition is satisfied

$$(X_1 = g^x \wedge SC_{\hat{i}}(m) = g^s \wedge T_0 = h_1^s) \vee \dots \vee (X_n = g^x \wedge SC_{\hat{i}}(m) = g^s \wedge T_0 = h_n^s) \equiv POK(x, s)$$

This is called the proof of knowledge ($POK(x, s)$) for x and s. This is given by client $C_{\hat{i}}$ to basically tell the server that it knows x and s, where x is the secret key corresponding to one of the client public keys and the associated h_i has been used to create the tag. This proof is based on [5] and can be computed by a client $\hat{i}$ as follows:

1. Choose $\overrightarrow{W} = (w_1, \cdots, w_n)$ where $w_{\hat{i}} = 0$ and $w_i \xleftarrow{R} \mathbb{Z}_p^*$ for $i \neq \hat{i}$.

2. Choose $\overrightarrow{V} = (v_{1.0}, v_{1.1}, \ldots, v_{n.0}, v_{n.1})$ where $v_{i.0} \xleftarrow{R} \mathbb{Z}_p^*$ and $v_{i.1} \xleftarrow{R} \mathbb{Z}_p^*$ for $1 \leq i \leq n$.

3. Compute the commitments for $1 \leq i \leq n$ as follows:

$$Q_{i.0} = X_i^{w_i} g^{v_{i.0}} \qquad Q_{i.10} = SC_i(m)^{w_i} g^{v_{i.1}} \qquad Q_{i.11} = T_0^{w_i} h_i^{v_{i.1}}$$

   Let $\overrightarrow{Q} = (Q_{1.0}, Q_{1.10}, Q_{1.11}, \ldots, Q_{n.0}, Q_{n.10}, Q_{n.11})$

4. Compute the challenge $\overrightarrow{C} = (c_1, \ldots, c_n)$ as:

$$c_i = \begin{cases} \mathcal{H}(\Gamma_{\hat{i}}(0), T_{\hat{i}}(0), \overrightarrow{Q}) - \sum_{k=1}^{n} w_k \pmod{p}, & \text{for } i = \hat{i} \\ w_i & \text{otherwise} \end{cases}$$

   where $\mathcal{H}$ is a collision resistant hash function such that $\mathcal{H} : \{0,1\}^* \rightarrow \mathbb{Z}_p^*$

5. Compute the responses $\overrightarrow{R} = (r_{1.0}, r_{1.1}, \ldots, r_{n.0}, r_{n.1})$ as follows: Set $x_{\hat{i}.0} = x_{\hat{i}}$ and $x_{\hat{i}.1} = s$. For all $i \neq \hat{i}$ let $x_{i.0} = x_{i.1} = 0$. Then compute $r_{i.k} = v_{i.k} - c_i x_{i.k} \pmod{p}$, for $1 \leq i \leq n$ and $k \in \{0, 1\}$.

6. The client $C_{\hat{i}}$ proof is submitted as $P_{\hat{i}}(0) = (\overrightarrow{C}, \overrightarrow{R})$.

Client Proof Verification Server S_1 verifies the proof $P_{\hat{i}}(0) = (\overrightarrow{C}, \overrightarrow{R})$ for the context $\Gamma_{\hat{i}}(0)$ received in the message $M_{\hat{i}}(0)$ sent by active client $C_{\hat{i}}$ as follows:

1. The server reconstructs $\overrightarrow{Q'}$ (for $1 \leq i \leq n$) as below:

   - $Q'_{i.0} = X_i^{c_i} g^{r_{i.0}}$
   - $Q'_{i.10} = SC_i(m)^{c_i} g^{r_{i.1}}$
   - $Q'_{i.11} = T_0^{c_i} h_i^{r_{i.1}}$

   where $\overrightarrow{Q'} = (Q'_{1.0}, Q'_{1.10}, Q'_{1.11}, \ldots, Q'_{n.0}, Q'_{n.10}, Q'_{n.11})$.

2. Then it verifies that, $\mathcal{H}(\Gamma_{\hat{i}}(0), T_{\hat{i}}(0), \overrightarrow{Q'}) = \sum_{i=1}^{n} c_i \pmod{p}$

4.3 The Servers

When the server S_1 receives a message from an active client C_i, it replaces the ephemeral randomness of the client applied on the tag, with the ephemeral round identifier. Then it sends a new well-formed signed message to server S_2. Server S_2 will perform the same task and will send a new message to S_3 and so on.

1. Server S_j receives a message $M_i(j-1) = (\ \Gamma_i(j-1),\ T_i(j-1),\ P_i(j-1)\ )$ from the client C_i if $j = 1$ or from the Server S_{j-1} if $2 \leq j \leq m$.

2. Server S_j checks the message context $\Gamma_i(j-1)$ and its proof $P_i(j-1)$, and rejects the message if invalid.

3. Server S_j takes the value $Z_i \in \overrightarrow{SC}_i$, where $\overrightarrow{SC}_i \in \Gamma_i(j-1)$, and computes using its secret key $SK_{S_j} = y_j$ the ephemeral value $s_j = H(g^{y_j z_i}) = H((Z_i)^{y_j})$. The server verifies if the commit $SC_i(j) \in \overrightarrow{SC}_i$ is well-formed by checking that $SC_i(j) = (SC_i(j-1))^{s_j}$.

4. Server S_j updated the linkage tag by computing

$$T_i(j) = (T_i(j-1))^{\frac{r_j}{s_j}} = h_i^{r_1 r_2 \ldots r_j s_{j+1} \ldots s_m}$$

   and generates a valid proof $P_i(j)$ for the new message.

5. Server S_j forms a new message $M_i(j) = (\ \Gamma_i(j),\ T_i(j),\ P_i(j)\ )$ where the new context is equal to the incoming message $M_i(j-1)$, meaning $\Gamma_i(j) = M_i(j-1)$. We observe that each server's message includes all prior messages, and ultimately the initial context $\Gamma_i(0)$ of the client C_i.

6. Server S_j finally sends $M_i(j)$ to server S_{j+1} (with a signature for such message) if $j < m$, or to all servers if $j = m$.

Server Proof Each server S_j receives $M_i(j-1) = (\Gamma_i(j-1), T_i(j-1) = h_i^{r_1 \ldots r_{j-1} s_j \ldots s_m}, P_i(j-1))$ from the previous server and must provide a proof of correctness for the new tag $T_i(j) = h_i^{r_1 \ldots r_j s_{j+1} \ldots s_m}$ which it computes with respect to its per-round commitment R_j and the client's shared secret commits $SC_i(j-1)$ and $SC_i(j)$. The server gives a proof of knowledge of secret values r_j and s_j as follows:

$$(R_j = g^{r_j} \wedge SC_i(j) = SC_i(j-1)^{s_j} \wedge T_i(j-1)^{r_j} = T_i(j)^{s_j}) \equiv POK(r_j, s_j)$$

As in the client's POK, this proof is based on [5] and can be computed by a server $\hat{j}$ as follows:

1. Choose $\overrightarrow{V} = (v_0, v_1)$ where $v_b \overset{R}{\leftarrow} \mathbb{Z}_p^*$ with $b \in \{0, 1\}$.

2. Compute the commitments as follows:

$$Q_0 = g^{v_0} \qquad Q_1 = (SC_i(j-1))^{v_1} \qquad Q_2 = T_i(j)^{v_1}/(T_i(j-1))^{v_0}$$

Let $\overrightarrow{Q} = (Q_0, Q_1, Q_2)$

3. Compute the challenge $c = \mathcal{H}(\Gamma_i(j-1), T_i(j), \overrightarrow{Q})$ where $\mathcal{H}$ is a collision resistant hash function such that $\mathcal{H} : \{0,1\}^* \rightarrow \mathbb{Z}_p^*$

4. Compute the responses $\overrightarrow{A} = (a_0, a_1)$ as follows: compute $a_0 = v_0 - cr_j \pmod p$ and $a_1 = v_1 - cs_j \pmod p$.

5. The server S_j submits the proof as $P_i(j) = (c, \overrightarrow{A})$.

Server Proof Verification The proof $P_i(j) = (c, \overrightarrow{A})$ from server S_j which is received by server S_{j+1} for the context $\Gamma_i(j)$ and message $M_i(j)$ is verified as follows:

1. The server reconstructs $\overrightarrow{Q'}$ as below:

   - $Q'_0 = R_j^c g^{a_0}$
   - $Q'_1 = (SC_i(j))^c (SC_i(j-1))^{a_1}$
   - $Q'_2 = (T_i(j))^{a_1}/(T_i(j-1))^{a_2}$

   where $\overrightarrow{Q'} = (Q'_0, Q'_1, Q'_2)$.

2. Then it verifies that, $\mathcal{H}(\Gamma_i(j-1), T_i(j), \overrightarrow{Q'}) = c \pmod p$

5 Protocol security properties

We discuss the security of the randomness scrubbing protocol below:

5.1 Client Proof is a Proof of Knowledge

Theorem 5.1.

$POK(x,s) \equiv \{(X_1 = g^x \wedge S_{\hat{i}}(m) = g^s \wedge T_0 = h_1^s) \vee \cdots \vee (X_n = g^x \wedge S_{\hat{i}}(m) = g^s \wedge T_0 = h_n^s)\}$

Given g, $\{h_i\}_{1 \leq i \leq n}$, and the context $=(\overrightarrow{PK}, \overrightarrow{R}, \overrightarrow{SC_i})$ (all public values) and the tag $T_i(0)$, the client must convince the server that it knows a secret x and value s such that $POK(x,s)$ is satisfied, but the server does not learn which clause is satisfied (or anything else about the client's secret values).

(Proof Sketch) The proof is based on [5]. We argue that the client's proof is a zero knowledge proof of knowledge of (x,s) which satisfies the POK formula.

- **Completeness**: A client which knows the pair (x, s) satisfying the boolean formula, must be able to convince any honest verifier of the same with high probability $(1 - negl(k))$.
  It is easy to see that, any client with appropriate x_i and s, can create the challenge and response values as mentioned in the protocol (section 4.2) and pass the verification to convince an honest server.

- **Soundness**: No cheating (PPT) client i.e. client without an appropriate (x, s), should be able to convince a verifier of the proof with better than negligible probability.

  We first present an informal argument and then present the knowledge extractor:

  Say that a client does not have the appropriate x and s, it needs to create c_i and x_i to satisfy the following equations:

$$Q_{i.0} = X_i^{w_i} g^{v_{i.0}} = X_i^{c_i} g^{r_{i.0}} \tag{1}$$
$$Q_{i.10} = SC_i(m)^{w_i} g^{v_{i.1}} = SC_i(m)^{c_i} g^{r_{i.1}} \tag{2}$$
$$Q_{i.11} = (h_i^s)^{w_i} h_i^{v_{i.1}} = (h_i^s)^{c_i} h_i^{r_{i.1}} \tag{3}$$
$$\mathcal{H}(\Gamma_i(0), T_i(0), \overrightarrow{Q}) = \sum_{i=1}^{n} c_i \mod p \tag{4}$$

  In order to satisfy equations (1), (2) and (3), the client can set $c_i = w_i$ and $v_{i.*} = r_{i.*}$. However, this is unlikely to satisfy equation (4) since the w_i values need to be fixed (to give $\overrightarrow{Q}$) before computing the hash $\mathcal{H}$, and the probability of this value being equal to $\sum w_i (= \sum c_i$ for this case) is negligible.

  Say, the cheating client knows one value (x_i or s) but not both, it can still not generate appropriate response values to the challenge. This is because, the client needs to create both $r_{i.0} = v_{i.0} + c_i(x_i)$ and $r_{i.1} = v_{i.1} + c_i(s)$, it will not be able to do this with out knowing both x_i and s. And as we have seen in the previous case, the probability of the hash value matching the choice of w_i is negligible.

  Now consider a cheating client who tries to use his secret x_i with a different per-round generator h_j $(i \neq j)$ to try to create a different linkage-tag. In that case, the client must be able to compute commitments such that:

$$Q_{i.10} = SC_i(m)^{w_i} g^{v_{i.1}} = SC_i(m)^{c_i} g^{r_{i.1}} \tag{5}$$
$$Q_{i.11} = (h_j^s)^{w_i} h_i^{v_{i.1}} = (h_j^s)^{c_i} h_i^{r_{i.1}} \tag{6}$$

  In order to do this it needs to know the discrete log of h_j with respect to h_i, if $log_{h_i} h_j = b$, then it must satisfy: $bsw_i + v_{i.1} = bsc_i + r_{i.1}$. But since $w_i = 0$, it must set $r_{i.1} = v_{i.1} - bsc_i$, however, this contradicts the assumption that no entity knows the discrete log of any h_i with respect to another h_j.

  Below we construct a knowledge extractor to show that the argument is sound.

- **Knowledge extractor**

Theorem 5.2. *From any (possibly cheating) prover (client) that is able to convince the verifier with good enough probability in the above protocol, we can construct an algorithm $\mathcal{B}$ to extract a valid (x', s') from the prover with non-negligible probability.*

1. $\mathcal{B}$ uses a random oracle to model the hash function.
2. $\mathcal{B}$ runs the prover (client) and receives the message and proof:

$$M_i(j) = (\Gamma_i(j), T_i(j), P_i(j) = \{\overrightarrow{C^*}, \overrightarrow{R^*}\})$$

3. Then $\mathcal{B}$ rewinds the prover until the point where it has not yet queried the hash i.e. it has fixed values $(\overrightarrow{W}, \overrightarrow{V}, \overrightarrow{Q})$. Now, $\mathcal{B}$ runs the random oracle hash function to return a different value when queried for $\mathcal{H}(\Gamma_{(0)}, \overrightarrow{Q})$.
4. Now $\mathcal{B}$ replays the prover from that point, to get a different challenge and response pair $(\overrightarrow{C'}, \overrightarrow{R'})$. Thus we have,

$$\overrightarrow{C^*} = (c_1, \ldots, c_t, \ldots, c_n) \qquad \overrightarrow{R^*} = (r_{1.0}, r_{1.1}, \ldots, r_{t.0}, r_{t.1}, \ldots, r_{n.0}, r_{n.1})$$

$$\overrightarrow{C'} = (c_1, \ldots, c'_t, \ldots, c_n) \qquad \overrightarrow{R^*} = (r_{1.0}, r_{1.1}, \ldots, r'_{t.0}, r'_{t.1}, \ldots, r_{n.0}, r_{n.1})$$

5. To solve the discrete log problem for x, $\mathcal{B}$ considers $r_{t.0} = v_{t.0} - c_t x'_t$ and $r'_{t.0} = v_{t.0} - c'_t x'_t$

$$r_{t.0} - r'_{t.0} = x'_t(c'_t - c_t)$$
$$x'_t = \frac{(r_{t.0} - r'_{t.0})}{(c'_t - c_t)}$$

6. To solve the discrete log problem for s, $\mathcal{B}$ considers $r_{t.1} = v_{t.1} - c_t s'$ and $r'_{t.1} = v_{t.1} - c'_t s'$

$$r_{t.1} - r'_{t.1} = s'(c'_t - c_t)$$
$$s' = \frac{(r_{t.1} - r'_{t.1})}{(c'_t - c_t)}$$

7. Thus $\mathcal{B}$ extracts values (witness) (x'_t, s') that satisfies a clause (and hence the entire formula) in the client proof.

- **ZK-ness**: The scheme is honest verifier zero knowledge. The server learns nothing except that the proof is true. The proof of this can be shown in the random oracle model by taking the verifier as a blackbox, this is similar to the unforgeability of a signature. The View generated by the simulator is the same as the signature transcript given by the sign oracle using the random oracle hash function. The unforgeability of the client proof is included in appendix A.

5.2 Server Proof is a Proof of Knowledge

Theorem 5.3.

$$POK(r_j, s_j) \equiv (R_j = g^{r_j} \wedge SC_i(j) = SC_i(j-1)^{s_j} \wedge T_i(j-1)^{r_j} = T_i(j)^{s_j})$$

Given g, the context $=(\overrightarrow{PK}, \overrightarrow{R}, \overrightarrow{SC_i})$ and the tag $T_i(j-1)$ and all public values, the server must convince the verifier that it knows a secret values r_j and s_j such that $POK(r_j, s_j)$ is satisfied, but the verifier does not learn which clause is satisfied (or anything else about the server's secrets).

(Proof Sketch) This proof is similar to the client's proof. We argue that the server's proof is a zero knowledge proof of knowledge of (r_j, s_j) which satisfies the POK formula.

- **Completeness**: A server which knows the pair (r_j, s_j) satisfying the boolean formula, must be able to convince any honest verifier of the same with high probability $(1 - negl(k))$. It is easy to see that, the server with appropriate r_j and s_j, can create the challenge and response values as mentioned in the protocol (section 4.3) and pass the verification to convince an honest verifier.

- **Soundness (Knowledge Extractor)**: This is proved using the knowledge extractor for proof of knowledge.

  Theorem 5.4. *From any (possibly cheating) prover (server) that is able to convince the verifier with good enough probability in the above protocol, we can construct an algorithm $\mathcal{B}$ to extract a valid (r_j, s_j) from the prover with non-negligible probability.*

  1. $\mathcal{B}$ uses a random oracle to model the hash function.
  2. $\mathcal{B}$ runs the prover (server) and receives the message and proof:

  $$M_i(j-1) = (\Gamma_i(j), T_i(j) = h_i^{r_1 \ldots r_j s_{j+1} \ldots s_m}, P_i(j) = (c^*, \overrightarrow{A^*}))$$

  3. Then $\mathcal{B}$ rewinds the prover until the point where it has not yet queried the hash i.e. it has fixed values $(\overrightarrow{V}, \overrightarrow{Q})$. Now, $\mathcal{B}$ runs the random oracle hash function to return a different value when queried for $\mathcal{H}(\Gamma_{(j-1)}, \overrightarrow{Q})$.
  4. Now $\mathcal{B}$ replays the prover from that point, to get a different challenge and response pair $(c', \overrightarrow{A'})$. Thus we have,

  $$\overrightarrow{C^*} = (c^*) \qquad \overrightarrow{A^*} = (a_0^*, a_1^*)$$

  $$\overrightarrow{C'} = (c') \qquad \overrightarrow{A'} = (a_0', a_1')$$

5. To solve the discrete log problem with respect to r_j, $\mathcal{B}$ considers $a_0^* = v_0 - c^* r_j$ and $a_0' = v_0 - c' r_j$

$$a_0^* - a_0' = r_j(c' - c^*)$$
$$r_j = \frac{(a_0^* - a_0')}{(c' - c^*)}$$

6. To solve the discrete log problem for s_j, $\mathcal{B}$ considers $a_1^* = v_1 - c^* s_j$ and $a_1' = v_1 - c' s_j$

$$a_1^* - a_1' = s_j(c' - c^*)$$
$$s_j = \frac{(a_1^* - a_1')}{(c' - c^*)}$$

7. Thus $\mathcal{B}$ extracts values (a witness) (r_j, s_j) that satisfies a clause (and hence the entire formula) in the server's proof.

- **ZK-ness**: The scheme is honest verifier zero knowledge. The server learns nothing except that the proof is true. The proof of this can be shown in the random oracle model by taking the verifier as a blackbox, this is similar to the unforgeability of a signature. The View generated by the simulator is the same as the signature transcript given by the sign oracle using the random oracle hash function. The unforgeability of the server proof is included in appendix A.1.

5.3 Uniqueness Proof

As defined in section 2.3 we consider the game where the adversary has secret keys of all n clients and has to generate $n+1$ valid messages. With reference to our protocol, this means that the adversary should be able to create $n+1$ linkage tags. Note that, an adversary can create multiple valid $M_i(0)$ messages (for the same h_i), however, at the end of the server circulation all such messages will get the same linkage tag. We prove uniqueness as follows:

Theorem 5.5. *If there exists a PPT adversary $\mathcal{A}$ who can create messages to produce $n+1$ linkage tags with n client secret keys with a non-negligible advantage, then we can construct a polynomial time algorithm $\mathcal{B}$ which contradicts the assumption that discrete log of h_i to the base g (or some other h_j) is not known to anyone.*

To prove this theorem, we consider the game in two cases and we use a hybrid argument to arrive at the attack. Let's consider the following cases, when an adversary tries to produce $n+1$ different linkage tags, then it must be that:

1. the adversary uses a different h_y such that $h_y \notin \{h_1, \ldots, h_n\}$ (with secret keys for all clients it is easy for an adversary to create tags for all $h_y \in \{h_1, \ldots, h_n\}$). In order to create a tag using h_y , the adversary requires $log_{h_i} h_y$ (where $i \in [1, n]$) (if not it breaks the soundness argument in the client's POK in section 5). We show how a client d ($d \in [1, n]$) who has $b = log_{h_d} h_y$ can create a valid proof:

- The client computes server commitments as follows:

$$s_j = H((PK_{S_j})^{z_d}) \qquad 1 \leq j \leq m$$

$SC_d(0) = g \quad SC_d(1) = g^{s_1} \quad \ldots \quad SC_d(e) = g^{s_1 s_2 \ldots s_e b} \quad \ldots \quad SC_d(m) = g^{bs_1 s_2 \ldots s_m}$

Thus, $b \cdot s = s_1 s_2 \ldots bs_e \ldots s_m = bs_1 \ldots s_m$ and the tag $T_d(0) = h_i^{bs_1 \ldots s_m} = h_y^{s_1 \ldots s_m}$.

- To generate the proof, the client follows the same protocol until it computes the challenge vector $\overrightarrow{C}$. Now,for the response vector, it must be able to compute commitments such that:

$$Q_{d.0} = X_d^{w_d} g^{v_{d.0}} = X_d^{c_d} g^{r_{d.0}} \tag{7}$$
$$Q_{d.10} = SC_d(m)^{w_d} g^{v_{d.1}} = SC_d(m)^{c_d} g^{r_{d.1}} \tag{8}$$
$$Q_{d.11} = (h_y^s)^{w_d} h_d^{v_{d.1}} = (h_y^s)^{c_d} h_d^{r_{d.1}} \tag{9}$$

It sets $r_{i.*} = v_{i.*}$ for $i \neq d$. Then set, $r_{d.0} = v_{d.0} - c_d x_d$.

Now to satisfy the above equations, it's necessary to get: $bsw_d + v_{d.1} = bsc_d + r_{d.1}$. But since $w_d = 0$, it must set $r_{d.1} = v_{d.1} - bsc_d$, which is possible since it knows b which is the discrete log h_y with respect to h_d.

- The submitted proof is actually for values (x_d, bs) and it is a valid proof of knowledge as the values (x_d, bs) satisfy the POK boolean formula.
- Now, each server will check if it is indeed true that $S_{j-1}^{s_j} = S_j$. If all servers were honest then it is easy to see that the client would be caught since $\exists S_j := S_j = S_{j-1}^{bs_j} \neq S_{j-1}^{s_j}$.
- Now we claim that even in the presence of malicious servers, it is possible to catch the cheating client. An evil server e has to create the proof for :

$$(R_e = g^{r_e} \wedge SC_d(e) = SC_d(e-1)^{s_e} \wedge T_d(e-1)^{r_e} = T_d(e)^{s_e}) \equiv POK(r_e, s_e)$$

But it has the tuple:

$$(R_e = g^{r_e} \wedge SC_d(e) = SC_d(e-1)^{bs_e} \wedge T_d(e-1)^{r_e} = T_d(e)^{s_e})$$

and $\nexists(r, s)$ that satisfies both tuples. Hence, the server proof will fail. And no cheating server can create an appropriate proof (from the soundness argument in server proof of section 5.1)

2. Consider the other case where a malicious server tries to create an additional linkage tag for $h_i^{r_1 \cdots r_m}$ where some r_j is changed by the server to r'_j. But the ability to do this again implies breaking the server proof ($POK \equiv (r_j, s_j)R_j = g^{r_j} \wedge SC_i(j) = SC_i(j-1)^{s_j} \wedge T_i(j-1)^{r_j} = T_i(j)^{s_j}$)

Thus it is not possible for any collusion of malicious entities to construct a new linkage tag or more than one linkage tag for a client, and hence the protocol satisfies uniqueness.

6 Attacks

We show that the randomness scrubbing protocol (in its first form)s does not satisfy deniability. We show how an attacker who has all messages on the network along with the long-term secrets can break deniability. In the next subsection we also propose a slightly relaxed definition for deniabilty that this protocol satisfies.

6.1 Deniability Attack

We show that the proposed randomness scrubbing protocol is not deniable. More specifically, we show that an attacker who learns all the messages on the network including the client and server long term secret keys (which are not erased after the round) can win in the Deniability (and Anonymity) game. The adversary $\mathcal{A}$ does the following:

1. Adversary $\mathcal{A}$ learns all the public keys $\overrightarrow{PK}$ of servers and clients and also learns per-round generators $\{h_1, \ldots, h_n\}$ corresponding to each client.

2. Adversary $\mathcal{A}$ keeps a record of all messages sent on the network during the round, in particular messages $M(0) = \Big(\Gamma(0), T(0), P(0) = (\overrightarrow{C}, \overrightarrow{R})\Big)$ sent from honest active client C to server S_1.

3. After the protocol round finishes, $\mathcal{A}$ also learns all the long term secret values $\overrightarrow{SK}$ of the clients $\{x_1, \ldots, x_n\}$ and servers $\{y_1, \ldots, y_m\}$.

4. Now, using Z ($\in \overrightarrow{SC} \in \Gamma(0)$) and the servers' private keys $\{y_1, \ldots, y_m\}$, the adversary $\mathcal{A}$ computes for $1 \leq i \leq n$:

$$s'_j = H(Z^{y_j}) \qquad \forall\ j,\ 1 \leq j \leq m$$

5. At this point the adversary knows a full set of secret exponents for some active honest client but it doesn't know which one. To find the right client the adversary brute forces on every $h \in \{h_1, \ldots, h_n\}$ until he finds a valid i such that:

$$T(0) = h_i^{s'_1 s'_2 \ldots s'_m}$$

6. Adversary $\mathcal{A}$ knows that $T(0) = T_i(0)$ since the tag has been generated using h_i which is associated with client C_i. Since the message $M(0)$ has been accepted by the protocol, then $M(0) = M_i(0)$. This finally reveals that the client C_i participated in the protocol round.

6.2 Relaxed Deniability

In the previous section, we saw an attack on deniability when there was a compromise in the servers' long term secret keys. If we relax the notion of deniability where we only compromise client keys (long term) and all honest clients and servers erase their ephemeral data, then the randomness scrubbing construction is deniable. (A stronger claim is that the scheme is deniable as long as atleast one honest server's long term keys are protected)

Theorem 6.1. *The randomness scrubbing scheme provides deniability and anonymity on full client key exposure.*

The proof of this can be observed from the construction of the message

$$M_x(0) = \left(\quad \Gamma_x(0) = \underbrace{(m_x, \overrightarrow{PK}, \overrightarrow{R}, \overrightarrow{SC}_x)}_{\text{context}}, \quad \underbrace{T_x(0)}_{\text{tag}}, \quad \underbrace{P_x(0)}_{\text{proof}} \right)$$

We show that it is possible for two different clients C_i and C_j to both generate the same message tuple $M_x(0)$.

1. Note that the *context* $= \Gamma_x(0) = (m_x, \overrightarrow{PK}, \overrightarrow{R}, \overrightarrow{SC})$ is constructed without using client secret keys and can be created by any client. This makes it client independent and therefore deniable.

2. The tag $T_x(0) = h_x^{s_1 \cdots s_m}$ (where $s_j = H((PK_{S_j})^{z_i}) = H(g^{y_j z_i})$), and the adversary does not know the long term secret of atleast one honest server k and hence cannot compute s_k. Thus, $\exists s_\gamma, s_{\bar{\gamma}}$ such that:

   $$T_x(0) = h_i^{s_1 \cdots s_\gamma \cdots s_m} = h_j^{s_1 \cdots s_{\bar{\gamma}} \cdots s_m}$$

   Solving for s_γ and $s_{\bar{\gamma}}$ is as hard as the DL problem. And even if the attacker solves that, it cannot confirm if $s_k = s_\gamma$ or $s_k = s_{\bar{\gamma}}$ without knowing the long term secret key of the server.

3. The proof $P_x(0) = (\overrightarrow{C}, \overrightarrow{R})$, could have been generated by either clients as shown below: Generating challenge $\overrightarrow{C}$ does not involve secret keys of clients (even the context and tags which are inputs to the hash functions do not require the client secret for computation). Thus, knowing $\overrightarrow{C}$ does not reveal anything about the client. According to the protocol:

$$c_i = \begin{cases} \mathcal{H}(\Gamma_{\hat{i}}(0), T_{\hat{i}}(0), \overrightarrow{Q}) - \sum_{k=1}^{n} w_k \pmod p, & \text{for } i = \hat{i} \\ w_i & \text{otherwise} \end{cases}$$

The w_i values are entirely random (when $i \neq \hat{i}$). Also, since the message passes verification, it must be the case that $\mathcal{H}(\Gamma_x(0), T_x(0), \overrightarrow{Q}) = \sum_{i=1}^{n} c_i \pmod p$. This implies that

$$\forall c_j : c_j = \mathcal{H}(\Gamma_x(0), T_x(0), \overrightarrow{Q}) - \sum_{i=1; i\neq j}^{n} c_i \qquad 1 \leq j \leq n$$

4. Now we show that the same $\overrightarrow{R}$ can be generated by both clients i and j. If $\overrightarrow{R} = (r_{1.0}, r_{1.1}, \ldots, r_{n.0}, r_{n.1})$. We have $r_{i.k} = v_{i.k} - c_i x_{i.k} \pmod p$, for $1 \leq i \leq n$ and $k \in \{0, 1\}$.

$$r_{i.0} = v_{i.0} - c_i x_i \pmod p$$

$$r_{i.1} = v_{i.1} - c_i s \pmod p$$

If the attacker changes x_i to x_j, he will only get another set of $v'_{i.0}$ which will also satisfy the equations of the proof but these cannot be checked with the original values which were deleted by the honest client. For $r_{i.1}$, the attacker knows neither s nor $v_{i.1}$ and hence the values will appear to be entirely random. Thus, $\overrightarrow{R}$ can be generated by either client.

Hence, all messages sent by honests clients are sender-deniable.

6.3 Anonymity

Anonymity follows from the proof of sender-deniability.

7 Modified Randomness Scrubbing Scheme

We thank Prof. Shmatikov for pointing out this clever modification that increases the security of the protocol providing full deniability. The modified scheme is described below:

We assume that every node in the network knows the generator g for group $\mathbb{Z}_p$ of prime order p, where $p = 2q + 1$ for a sufficiently large prime q. Every operation described below is $\pmod p$.

7.1 Setup

Each client $C_i \in Clt$ has a private key $SK_{C_i} = x_i$ and a public key $PK_{C_i} = g^{x_i}$. Similarly each server $S_j \in Srv$ has a private key $SK_{S_j} = y_j$ and a public key $PK_{S_j} = g^{y_j}$. Let $\overrightarrow{PK}_{Clt} = (PK_{C_1}, \ldots, PK_{C_n})$ and similarly $\overrightarrow{PK}_{Srv} = (PK_{S_1}, \ldots, PK_{S_m})$ be the vectors of all clients and servers' public keys. We define the following vector $\overrightarrow{PK} = (\overrightarrow{PK}_{Clt}, \overrightarrow{PK}_{Srv})$.

Ephemeral keys. At the beginning of the round, each server S_j picks two per-round random values r_j and l_j, then broadcasts signed messages $R_j = g^{r_j}$ and $L_j = g^{l_j}$. Each active entity can easily compute the following vector $\overrightarrow{R} = (R_1, \ldots, R_m)$ by checking the signature attached to the messages. The vector $\overrightarrow{R}$ will serve as a *round identifier* for this round and the values $\overrightarrow{L} = (L_1, \ldots, L_m)$ will serve as the round-public keys.

7.2 The Clients

From now on we refer only to a generic active client.

1. Client C_i picks a per-round generator h_i such that it's hard to establish the logarithmic relationship between g and any h_i. Moreover for every i and i' with $1 \leq i < i' \leq n$, it is hard to establish the logarithmic relationship between h_i and $h_{i'}$. Such values can be computed, for example, with an hash function of a unique subset of public variables for each client. Each client C_i broadcast a message containing h_i.

2. Client C_i picks a random ephemeral key z_i and computes $Z_i = g^{z_i}$. For each server S_j, the client picks the round public key L_j of server S_j and computes a secret exponent

$$s_j = H((L_j)^{z_i}) = H(g^{l_j z_i})$$

   where $H : \mathbb{Z}_p^* \to \mathbb{Z}_p^*$ is a cryptographic hash function from a group element to an exponent.

3. Client C_i computes a linkage tag $T_i(0) = h_i^{s_1 s_2 \ldots s_m}$ and for each $1 \leq j \leq m$ computes a set of commits for the servers $[SC_i(0), SC_i(1), \ldots, SC_i(j), \ldots, SC_i(m)]$ where

$$SC_i(0) = g \quad SC_i(1) = g^{s_1} \quad \ldots \quad SC_i(j) = g^{s_1 s_2 \ldots s_j} \quad \ldots \quad SC_i(m) = g^{s_1 s_2 \ldots s_m}$$

   Let $\overrightarrow{SC}_i = (Z_i, SC_i(0), SC_i(1), \ldots, SC_i(m))$ be the vector of commits sent by client C_i on the network.

4. Finally, client C_i chooses it's message m_i and sends the following message to server S_1.

$$M_i(0) = \left(\quad \Gamma_i(0) = \underbrace{(m_i, \overrightarrow{PK}, \overrightarrow{R}, \overrightarrow{SC}_i)}_{\text{context}}, \quad \underbrace{T_i(0)}_{\text{tag}}, \quad \underbrace{P_i(0)}_{\text{proof}} \quad \right)$$

The context $\Gamma_i(0)$ and the tag $T_i(0)$ are clearly defined at this point. The proof $P_i(0)$ is a proof of knowledge of secret values (x, s) such that $(X_k = g^x \wedge S_k(m) = g^s \wedge T_k(0) = h_1^s)$ (where $1 \leq k \leq n$) and is computed as before. This completes the client interaction.

7.3 The Servers

When the server S_1 receives a message from an active client C_i, it replaces the ephemeral randomness s_j used by the client on the tag, with the ephemeral round identifier r_j. Then it sends a new well-formed signed message to server S_2. Server S_2 will perform the same task and will send a new message to S_3 and so on.

1. Server S_j receives a message $M_i(j-1) = (\ \Gamma_i(j-1),\ T_i(j-1),\ P_i(j-1)\ )$ from the client C_i if $j = 1$ or from the Server S_{j-1} if $2 \leq j \leq m$.

2. Server S_j checks the message context $\Gamma_i(j-1)$ and its proof $P_i(j-1)$, and rejects the message if invalid.

3. Server S_j takes the value $Z_i \in \overrightarrow{SC}_i$, where $\overrightarrow{SC}_i \in \Gamma_i(j-1)$, and computes using its round secret key l_j the ephemeral value $s_j = H(g^{l_j z_i}) = H((Z_i)^{l_j})$. The server verifies if the commit $SC_i(j) \in \overrightarrow{SC}_i$ is well-formed by checking that $SC_i(j) = (SC_i(j-1))^{s_j}$.

4. Server S_j updated the linkage tag by computing

$$T_i(j) = (T_i(j-1))^{\frac{r_j}{s_j}} = h_i^{r_1 r_2 \ldots r_j s_{j+1} \ldots s_m}$$

   and generates a valid proof $P_i(j)$ for the new message.

5. Server S_j forms a new message $M_i(j) = (\ \Gamma_i(j),\ T_i(j),\ P_i(j)\ )$ where the new context is equal to the incoming message $M_i(j-1)$, meaning $\Gamma_i(j) = M_i(j-1)$. We observe that each server's message includes all prior messages, and ultimately the initial context $\Gamma_i(0)$ of the client C_i.

6. Server S_j finally sends $M_i(j)$ (with a signutare on such message) to server S_{j+1} if $j < m$, or to all servers if $j = m$.

7.4 Security

Theorem 7.1. *The modified randomness scrubbing scheme satisfies, uniqueness, deniability and anonymity*

1. **Uniqueness** The proof for uniqueness remains unchanged as before.

   - A cheating client cannot send a tag $T_i(0)$ with h_j^s if it does not know $log_{h_i} h_j$ or the secret key (x_i) associated with h_i.
   - A cheating client may send a tag using $h_y \notin (h_1, \ldots, h_n)$ and can create a POK if it knows the discrete log of h_y with respect to any h_i. If $b = log_{h_i} h_y$, then it has to send $SC_i(m) = g^{sb}$ and there will exist a j $(1 \leq j \leq m)$ such that $SC_i(j) = g^{s_1 \ldots s_{j-1} b s_j}$ and $SC_i(j-1) = g^{s_1 \ldots s_{j-1}}$. This implies that either an honest server will spot the cheating client or a dishonest will fail to create a proof for it's new tag since $T_i(j-1)^{r_j} = T_i(j)^{s_j}$ but $SC_i(j) \neq SC_i(j-1)^{s_j}$.

- A cheating server will not be able to replace s_j with some other value $r'_j \neq r_j$ since it will be hard to give a valid proof for it's tag.

2. **Deniability** As before, to show the deniability of the modified protocol, one can examine each of the components in the message sent by a client and show that the same message could have been generated by two different honest clients (c and d, w.l.o.g let $1 \leq c < d \leq n$).

$$M_x(0) = \left(\quad \Gamma_x(0) = \underbrace{(m_x, \overrightarrow{PK}, \overrightarrow{R}, \overrightarrow{SC}_x)}_{\text{context}}, \quad \underbrace{T_x(0)}_{\text{tag}}, \quad \underbrace{P_x(0)}_{\text{proof}} \quad \right)$$

- $\overrightarrow{PK}, \overrightarrow{R}$ are public values available to all, m_x can be any message string and can be generated by anyone in the network.
- Computation of $\overrightarrow{SC}_x$ do not require long-term secrets and can be computed by any entity with the appropriate ephemeral keys. Since the honest clients are required to delete their ephemeral secrets, the same $\overrightarrow{SC}_x$ values could have been computed by either honest clients.
- Tag $T_x(0) = h_x^{s_1 \cdots s_m}$. In order to identify the client x, it is necessary to obtain all s_j $(1 \leq j \leq m)$ values correctly. If the attacker does not know at least one s_j, then $\exists (s, s')$ such that $h_c^s = h_d^{s'}$ where $s = s_1 \ldots s_j \ldots s_m$ and $s' = s_1 \ldots s'_j \ldots s_m$.
- In the network setup where there is at least one honest server, we claim that there exists at least one value s_j $(1 \leq j \leq m)$, that is not known to the attacker at the end of the protocol.

  Recall that, $s_j = H((L_j)^{z_x}) = H((Z_x)^{l_j}) = H(g^{l_j z_x})$. Let $Y = g^{l_j z_x}$. At the end of the protocol the attacker has access to all values $\overrightarrow{L}$ and Z_x. But since there exists atleast one honest server which deletes it's ephemeral per-round secret key l_j and the honest clients delete the ephemral values z_c, z_d, an attacker cannot compute atleast one value s_j. To prove this, assume the contrary.

  Theorem 7.2. *If there exists an attacker who can compute an honest server's s_j (or all s_j $(1 \leq j \leq m)$), then we can construct a PPT algorithm that can either find a collision in the hash function H or solve the Computational Diffie Hellman (CDH) problem using this attacker.*

  Assume the hash function as a random oracle; if the adversary has computed s_j correctly then he must have queried for the hash of some value Y such that $H(Y) = s_j$. Now, there are two possibilities: if $Y' = Y$, then we have a solution for the CDH problem, where given $Z_x = g^{z_x}, L_j = g^{l_z}$, we have computed (with the help of the attacker) a $Y' = g^{l_j z_x} = Y$. In the other case, if $Y' \neq Y$, then we have two input values Y, Y' such that $H(Y) = s_j = H(Y')$ and $Y \neq Y'$, which is a collision on the hash function.

- The proof $P_x(0) = (\overrightarrow{C}, \overrightarrow{R})$. The argument for this remains unchanged as in the case of relaxed deniability (section 6.1). Computation of $\overrightarrow{C}$ does not involve secret keys and knowing these values does not reveal anything about the sender of a message. In the case of $\overrightarrow{R}$, we have seen that it is possible for 2 different honest clients with different secret keys to create the same response tuple $\overrightarrow{R}$.

3. **Anonymity** Anonymity of the scheme follows from deniability. Since the sender of a message cannot be identified even on full key exposure, the sender remains anonymous.

8 Conclusions

In this project we explored many protocol and schemes. The idea of achieving Anonymity, Deniability and Uniqueness seems hard to reach, in fact previous schemes are not able to achieve even Deniability and Uniquess in the same context. We carefully study the protocol proposed in [11] and, we found different attacks on such preliminary draft. Fortunately all the attacks seems to be fixable, and we finally provided proofs on how to build a CMP-Protocol out of that scheme. The new idea, highlighted by Prof. Shmatikov, relies on the ability to build a shared secret key between clients and servers using a double linked ephemeral state: the ephemeral state of each client is itself determined by a part of the ephemeral state of each server. In this way, any attack based on the recovery of shared secret keys for the clients trivially fails, even if all the long term key pairs are compromised by the attacker.

References

[1] Bender, A., Katz, J., and Morselli, R. Ring signatures: Stronger definitions, and constructions without random oracles. In *TCC* (2006), pp. 60–79.

[2] Brands, S. Untraceable off-line cash in wallets with observers (extended abstract). In *CRYPTO* (1993), pp. 302–318.

[3] Brands, S. A. An efficient off-line electronic cash system based on the representation problem. Tech. rep., Amsterdam, The Netherlands, The Netherlands, 1993.

[4] Camenisch, J., Hohenberger, S., and Lysyanskaya, A. Compact e-cash. In *In EUROCRYPT, volume 3494 of LNCS* (2005), Springer-Verlag, pp. 302–321.

[5] Camenisch, J., and Stadler, M. Proof systems for general statements about discrete logarithms. Tech. rep., 1997.

[6] Chaum, D. Blind signatures for untraceable payments. In *CRYPTO* (1982), pp. 199–203.

[7] Chaum, D., Fiat, A., and Naor, M. Untraceable electronic cash. In *Advances in Cryptology CRYPTO 88*, S. Goldwasser, Ed., vol. 403 of *Lecture Notes in Computer Science*. Springer New York, 1990, pp. 319–327.

[8] Chaum, D., Fiat, A., and Naor, M. Untraceable electronic cash. In *Proceedings of the 8th Annual International Cryptology Conference on Advances in Cryptology* (London, UK, UK, 1990), CRYPTO '88, Springer-Verlag, pp. 319–327.

[9] Chaum, D. L. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM 24*, 2 (1981), 84–90.

[10] Dwork, C., Naor, M., and Sahai, A. Concurrent zero-knowledge. *J. ACM 51*, 6 (2004), 851–898.

[11] Ford, B., and Shmatikov, V. Notes on deniable anonymous group authentication. 2009.

[12] Herranz, J., and Sez, G. Forking lemmas for ring signature schemes. In *Progress in Cryptology - INDOCRYPT 2003*, T. Johansson and S. Maitra, Eds., vol. 2904 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2003, pp. 266–279.

[13] Naor, M. Deniable ring authentication. In *Advances in Cryptology CRYPTO 2002*, M. Yung, Ed., vol. 2442 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2002, pp. 481–498.

[14] Rivest, R. L., Shamir, A., and Tauman, Y. How to leak a secret. In *Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology* (2001), Springer-Verlag, pp. 554–567.

[15] Shamir, A. How to share a secret. *Commun. ACM 22*, 11 (Nov. 1979), 612–613.

[16] Tsang, P. P., and Wei, V. K. Short linkable ring signatures for e-voting, e-cash and attestation. In *ISPEC* (2005), pp. 48–60.

[17] Waters, B. Efficient identity-based encryption without random oracles. In *EUROCRYPT* (2005), pp. 114–127.

A Appendix

A.1 Unforgeability of the Client Proof

Theorem A.1. *If there exists a PPT adversary $\mathcal{A}$ who can forge a client proof with a non-negligible advantage, then we can construct a polynomial time algorithm $\mathcal{B}$ which can solve the discrete log problem with non-negligible advantage.*

The proof will use a random oracle to model the hash function $\mathcal{H}$ and the idea of forking lemmas for ring signatures [12], to solve the discrete log problem. Algorithm $\mathcal{B}$ does the following in the unforgeability game:

1. Algorithm $\mathcal{B}$ receive the discrete log challenge $(g, X_t = g^{x_t})$ and it sets the honest client's public key as X_t.
2. Sign Oracle Queries: When $\mathcal{A}$ requests for proof on some message $M_0 = (\Gamma(0), T_0)$.
   (a) $\mathcal{B}$ chooses $c_i, r_{i.0}, r_{i.1} \xleftarrow{R} \mathbb{Z}_q$, $\forall i \in [1, n]$ and then sets $\overrightarrow{C} = (c_1, \ldots, c_n)$ and $\overrightarrow{R} = (r_{1.0}, r_{1.1}, \ldots, r_{n.0}, r_{n.1})$.
   (b) $\mathcal{B}$ computes
   $$Q_{i.0} = X_i^{w_i} g^{v_{i.0}} \qquad Q_{i.10} = SC_i(m)^{w_i} g^{v_{i.1}} \qquad Q_{i.11} = T_0^{w_i} h_i^{v_{i.1}}$$
   (c) It then sets $\mathcal{H}(\Gamma_{(0)}, \overrightarrow{Q}) = \sum_{i=1}^{n} c_i \pmod q$
   (d) It returns $\overrightarrow{C}, \overrightarrow{R}$ as the proof.
3. *Hash oracle queries.* When $\mathcal{A}$ makes queries to the hash oracle, if the query is new $\mathcal{B}$ returns a random value $r \in \mathbb{N}$ and maintains a list containing $\left\langle \Gamma_{(0)}, \overrightarrow{Q}, r \right\rangle$. If, $\mathcal{A}$ requests for a hash of a value that was already computed then it returns it from the list.
4. Finally, $\mathcal{A}$ submits a forgery $\overrightarrow{C^*}, \overrightarrow{R^*}$ on some message M^*. By using the forking lemma - i.e. rewinding the attacker and changing our hash response to $\mathcal{A}$'s queries, we can get two forgeries on M^* such that:
   $$\overrightarrow{C^*} = (c_1, \ldots, c_t, \ldots, c_n) \qquad \overrightarrow{R^*} = (r_{1.0}, r_{1.1}, \ldots, r_{t.0}, r_{t.1}, \ldots, r_{n.0}, r_{n.1})$$
   $$\overrightarrow{C'} = (c_1, \ldots, c'_t, \ldots, c_n) \qquad \overrightarrow{R^*} = (r_{1.0}, r_{1.1}, \ldots, r'_{t.0}, r'_{t.1}, \ldots, r_{n.0}, r_{n.1})$$
   Note: When we rewind attacker, we only rewind it and return a different hash value for the query. Therefore, this changes only the value of c_t and r_t, all the other values that the attacker computes is already fixed (by his choice of $\overrightarrow{V}$ and $\overrightarrow{W}$). Thus, the computation of a second forgery by rewinding can be done in polynomial time.

5. To solve the discrete log problem for x, $\mathcal{B}$ considers $r_{t.0} = v_{t.0} - c_t x'_t$ and $r'_{t.0} = v_{t.0} - c'_t x'_t$

$$r_{t.0} - r'_{t.0} = x'_t(c'_t - c_t)$$
$$x'_t = \frac{(r_{t.0} - r'_{t.0})}{(c'_t - c_t)}$$

6. To solve the discrete log problem for s, $\mathcal{B}$ considers $r_{t.1} = v_{t.1} - c_t s'$ and $r'_{t.1} = v_{t.1} - c'_t s'$

$$r_{t.1} - r'_{t.1} = s'(c'_t - c_t)$$
$$s' = \frac{(r_{t.1} - r'_{t.1})}{(c'_t - c_t)}$$

7. $\mathcal{B}$ returns it's response x'_t and s' to the DL challenger.

A.2 Unforgeability of the Server Proof

Theorem A.2. *If there exists a PPT adversary $\mathcal{A}$ who can forge a server proof with a non-negligible advantage, then we can construct a polynomial time algorithm $\mathcal{B}$ which can solve the discrete log problem with non-negligible advantage.*

The unforgeability for server proof is similar to that of the client proof unforgeability. The proof will use a random oracle to model the hash function $\mathcal{H}$.

1. Algorithm $\mathcal{B}$ receive the discrete log challenge $(g, R_t = g^{r_t})$ and it sets the server's public key as R_t.

2. Sign Oracle Queries: When $\mathcal{A}$ requests for proof on some tag

$$(R_t, SC_x(t-1), SC_x(t), T_x(t-1), T_x(t))$$

.

   (a) $\mathcal{B}$ chooses $c, a_0, a_1 \xleftarrow{R} \mathbb{Z}_q$, $\forall i \in [1, n]$ and then sets $\overrightarrow{C} = c$ and $\overrightarrow{A} = (a_0, a_1)$.

   (b) $\mathcal{B}$ computes

$$Q_0 = R_t^c g^{a_0} \qquad Q_1 = (SC_x(t))^c (SC_x(t-1))^{a_1} \qquad Q_2 = \frac{(T_x(t))^{a_1}}{(T_x(t-1))^{a_2}}$$

   (c) It then sets $\mathcal{H}(\Gamma_{(t)}, T_x(t), \overrightarrow{Q}) = c \pmod q$.

   (d) It returns $(c, \overrightarrow{A})$ as the proof.

3. *Hash oracle queries.* When $\mathcal{A}$ makes queries to the hash oracle, if the query is new $\mathcal{B}$ returns a random value $r \in \mathbb{Z}_p^*$ and maintains a list containing $\left\langle \Gamma, T, \overrightarrow{Q}, r \right\rangle$. If, $\mathcal{A}$ requests for a hash of a value that was already computed then it returns it from the list.

4. When $\mathcal{A}$ submits a forgery(C^*, A^*) . By rewinding the adversary and giving a different hash value, we can get two challenge-responses for the same message where

$$\overrightarrow{C^*} = (c) \qquad \overrightarrow{A^*} = (a_0, a_1)$$

$$\overrightarrow{C'} = (c') \qquad \overrightarrow{A'} = (a'_0, a'_1)$$

To solve the discrete log problem with respect to r_t, $\mathcal{B}$ considers $a_0 = v_0 - cr_t$ and $a'_0 = v_0 - c'r_t$

$$a_0 - a'_0 = r_t(c' - c)$$
$$r_t = \frac{(a_0 - a'_0)}{(c' - c)}$$

Similarly, to solve the discrete log problem with respect to s_t, $\mathcal{B}$ considers $a_1 = v_1 - cs_t$ and $a'_1 = v_1 - c's_t$

$$a_1 - a'_1 = s_t(c' - c)$$
$$s_t = \frac{(a_1 - a'_1)}{(c' - c)}$$