Privacy Preserving Grapevines: Capturing Social Network Interactions Using Delegatable Anonymous Credentials

Vijay A. Balasubramaniyan[1], Younho Lee[2], and Mustaque Ahamad[1]

[1]College of Computing, Georgia Institute of Technology, USA
[2]Department of Information and Communication Engineering, Yeungnam University, Korea

Abstract

A wide variety of services allow users to meet online and communicate with each other, building new social relationships and reinforcing older ones. Unfortunately, malicious entities can exploit such services for fraudulent activities such as spamming. It is critical that these services protect users from unwanted interactions, especially when new relationships are being established - the introduction problem. The problem of assessing that a social network connection is no longer beneficial is also important due to the dynamic nature of such networks. A large number of new connections are established through existing, weak social ties (for example, friend of a friend). On the other hand, the willingness of a user to continue interactions with an existing relationship is an indication of his or her endorsement of that relationship. The interaction history of a user provides valuable information about both new social network connections and the validity of established ones. However, capturing this interaction history is rife with privacy concerns. In this paper, we create a transferable token framework, based on delegatable anonymous credentials (DAC - Crypto 2009), that captures interaction history in a privacy preserving manner. By using the Groth Sahai proof system, we extend DACs to allow for single use tokens with the ability to identify token double spenders. We show that such tokens can, simultaneously, demonstrate the existence of a social network path and capture the continued validity of a social network connection. We present an implementation of this DAC based token framework and utilize it in a Voice over IP (VoIP) setting to enable legitimate user interactions in the presence of a spammer threat model. Our results indicate that we are able to achieve low false positive and false negative rates for realistic threat scenarios without disclosing a user's social network connections.

1 Introduction

Social relationships are growing with a wide variety of online systems that allow users to communicate with each other. Instant Messaging (IM) services such as Yahoo Messenger[24] and AIM[3] offer feature rich text messaging services. Telecommunications have also been revolutionized with the introduction of Voice over IP (VoIP) systems such as Skype[19] and Google Talk[20]. The number of active users on these systems is an indication of their success. As of 2008, Yahoo Messenger boasted of over 200 million users[4] and currently there are over 400 million Skype users[1]. The biggest advantage of these systems is that users can communicate, share and collaborate with other users, irrespective of their actual physical location, providing a platform for building new relationships and renewing old ones.

In such systems, malicious users are constantly trying to exploit a user's willingness to communicate with new users. For the system to ensure beneficial interactions, it should try to determine whether a user initiating contact for the first time is legitimate or malicious - the introduction problem. For example, in IM systems such as Yahoo Messenger or Google Talk, users explicitly invite people that they would like to chat with, and subsequently are able to chat with all those who accept the invite. Google Talk goes a step further and allows users who have had prior email correspondence to automatically chat with each other. Automatic introduction is especially important in systems like VoIP, where the possibility of prior user interaction is low. To illustrate,

consider a scenario where your father's friend's son, say Bob, would like to talk to you about admission to a program at your university (or career openings at your workplace). Social network (SN) theory suggests that higher the number of such weak ties between users, higher the likelihood of of a new direct tie being established between them[21, 6]. The legitimacy of Bob's weak tie should not be determined by active user participation as that can also be misused by spammers/telemarketers to provide unsolicited information. For example, in IM based systems such as AIM[3], spammers provide unsolicited content as part of the initial invite request itself. We, thus, need an automated framework that is able to *demonstrate the existence of a social network/interaction path between two users that are trying to establish communication for the first time.*

Along with determining a user's willingness to interact with another user for the first time, it is equally important to determine whether a user would like to continue interactions with people that he has been introduced to. To illustrate, consider a user who calls a travel agent and requests for travel quotes to India. If the travel agent does not specialize in flight tickets to India, he could get fellow travel agents (his SN) to talk to the user. As long as the user is interested in flight tickets, he will continue interacting with the travel agent and/or his SN. This interaction stops after he has bought the ticket. However, it is likely that the travel agents continue to contact him with promotional offers. This information is unsolicited by the user, and constitutes spam. Therefore, in addition, to being able to determine social network paths, the framework should be able to *capture the willingness of a user to continue interactions with a particular user*.

In essence, to provide good user experiences, a system needs to address two different challenges. The first allows users without a direct link to communicate with each other, and the second monitors the quality and validity of a link that exists between two users. Both these aspects can, in fact, be determined by capturing the interaction history of users. A weak social tie exists between two users, if there exists some interaction path between them. Similarly a users' willingness to continue interacting with another user is an indication of the strength of their social tie. Many systems use tokens[26, 17, 2] to capture interaction/transaction information. However, in social networks, capturing the interaction history is rife with privacy concerns as the interactions may reveal sensitive information. To illustrate these privacy concerns we consider CallRank[5], a system that combats the VoIP spam problem.

Application Scenario - VoIP Spam: VoIP spam is unsolicited phone calls through the use of VoIP systems. Unlike email spam, voice spam cannot filter on content as media content, in addition to being real time, is received only after a ringing phone is picked up. A spam engine that filters based on the media content, however successful, cannot prevent phones from ringing. To address this, our previous work, CallRank[5] uses call duration and call initiation direction, to differentiate between a legitimate user and a spammer. It is motivated by the observation that a legitimate user, on average, makes and receives calls of significant call durations. A spammer on the other hand is trying to disseminate information to as many people as possible and therefore makes a large number of calls. These calls are brief as spammers find it hard to engage legitimate users in a conversation. In addition, spammers hardly receive any calls. Thus, the calling pattern for a spammer is largely one way while it is bidirectional for legitimate users. The CallRank system realizes this asymmetry by using a token/credential mechanism. As shown in figure 1, at the end of a VoIP call between Alice(caller) and Bob(call recipient) that lasted 10 minutes, Alice issues a digitally signed call token to Bob, represented by $T^{A \rightarrow B}$. At a later instance when Bob talks to Charlie for 20 minutes, he similarly hands him another token $T^{B \rightarrow C}$ as shown. Now when Charlie wants to talk to Alice, with whom he has had no previous direct interaction, he presents $T^{B \rightarrow C}$ to prove to Alice that someone in her social network was willing to talk to him. The factors that influence Alice's decision to accept this call token include how well she knows Bob and the call duration information embedded. This provides Alice great control on the calls she accepts. However, she also gets to know precisely when and for how long Bob and Charlie talked, something that violates their privacy. CallRank restricts itself to friends and friends of friends(two hop) as the loss in privacy, illustrated above, is aggravated as number of hops increase. The CallRank setting clearly shows how SN interaction

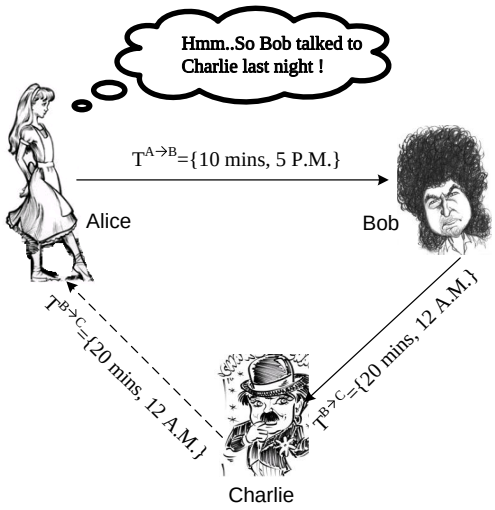

Figure 1: Privacy Leak in CallRank

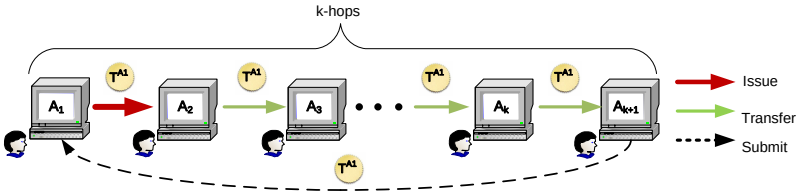

Figure 2: Multi-hop Token Transfer

history provides a valuable mechanism to differentiate between regular users and spammers. However these tokens are not privacy preserving. In addition, since they only allow a two hop SN, they are restrictive and lose valuable weak tie information that can be obtained from a larger hop SN.

In this paper, we create a token framework that uses delegatable anonymous credentials [7] to create N hop transferable tokens without sacrificing user privacy. The framework allows a user to prove the existence of a token transfer path between him and the user he is trying to initiate contact with, without actually revealing the path. If token transfers are associated with a user's SN interactions then the token transfer path is essentially equivalent to a SN path. For example, in CallRank, if tokens are issued and transferred as part of call signaling, the transfer path represents a chain of calls between two callers. This information can be used by legitimate users to prove the existence of a weak social tie (father's friend's son) between them and the user they are trying to call. In addition, we use the Groth Sahai (GS) proof system[22], to extend DACs to create single use tokens with the ability to identify token double spenders. This allows tokens to capture a user's continued endorsement of a direct link(strong social tie). We implement the token framework using the Pairing Based Cryptography (PBC) library[27] and utilize it in the VoIP setting to explore its performance in the presence of a spammer threat model. Other communication systems and social network based services can also use this framework with minor modifications.

Our paper makes the following contributions:

1. We identify the requirements for a framework that allows a new user, Bob, to prove the existence of an interaction path between him and Alice, without revealing the actual path. In addition, the framework allows us to capture a user Alice's willingness towards continued interactions with Bob.
2. We create a transferable single use token mechanism that enhances delegatable anonymous credentials[7] to realize this framework.
3. We provide an implementation of this framework using the PBC library and experimentally evaluate the costs associated with its operations.
4. We apply this framework to the real world application setting of VoIP Spam to demonstrate that it can combat the spam problem with low false positive and false negative rates.

The rest of the paper is organized as follows. In section 2 we discuss the requirements of the desired framework, followed by possible approaches in section 3. We show that none of these approaches satisfy all the requirements and we develop our solution by first discussing the building blocks, the GS proof system and

DACs in section 4. In section 5 we discuss how we extend DACs to create our single use privacy preserving transferable token scheme. We discuss implementation details, and results that include operation times of our scheme and the performance of the scheme with respect to the VoIP spam threat model in section 7. We finally conclude and discuss future work in section 8.

2 Requirements Discussion

The two broad goals for our framework is that, in a privacy preserving manner, it is able to: (1) demonstrate the existence of a social network/interaction path between two users that are trying to establish communication for the first time and (2) capture the willingness of a user to continue interactions with a particular user and his SN. One way of demonstrating an interaction path is showing that there exists a sequence of interactions starting from one particular user and ending at another. This is advantageous as (2) gauges continued interactions, reducing the basic primitive for both these requirements to that of capturing user-user interaction. We assume this interaction can be captured through tokens. If tokens now need to be able to capture a sequence of interactions they have to be transferred between users in the path with interaction details appended to the token as it is transferred. We use the multi-hop transferable token example shown in figure 2 to understand the requirements of a system that will satisfy our goals. Towards (1), the token needs to satisfy the following requirements:

Unforgeability: Interaction paths demonstrate the existence of a weak tie to a particular user, specifically, the token issuer. In figure 2, A_1 is the token issuer and any user in the token path is trying to demonstrate a weak tie to A_1, by possession of the token. Therefore, the token that A_1 issues should be unforgeable, that is no other user should be able to issue a token on A_1's behalf. The transfer path itself should be unforgeable, which means a user can only prove a transfer if that transfer actually occurred. We note that without an all observing trusted third party there is no way of ensuring token transfers are tied to an interaction or the interaction details embedded in the token are real. We, therefore, make the assumption that honest users transfer tokens only during social interactions, and mutually agree on the embedded interaction details. Malicious entities may choose to transfer the tokens without an interaction. Despite this, a malicious entity should not be able to forge a token that is issued by an honest party or one that is transferred by an honest user.

Verifiability: The unforgeability of the token should be verifiable by any user in the transfer path. In figure 2, for a transfer leg $A_{i-1} \rightarrow A_i$, A_i before accepting a token A_i should be able to verify that the token is issued by A_1 and that A_{i-1} is the rightful owner of the token. To do this A_{i-1} will need to prove that the token was issued by A_1, and subsequently there were a set of transfers culminating at him. The unforgeability requirement states that A_{i-1} cannot do this successfully unless A_1 issued the token and all subsequent transfers leading to A_{i-1} did indeed occur. This prevents a solution from verifying a token only when it is finally submitted to an issuer. Such a solution can lead to a lot of bogus tokens in the system, each of them being discarded only when they are submitted to the issuer. In such a system, users will never be able judge the validity of a token, ultimately rendering tokens useless.

Privacy: For honest users, a token is transferred during a social interaction and contains sensitive interaction details. In achieving the above two requirements the scheme should ensure that details of an interaction should only be known to users directly participating in that interaction. Consider a transfer leg $A_{i-1} \rightarrow A_i \rightarrow A_{i+1}$ for token T^{A_1}. In this case when A_i transfers the token to A_{i+1}, A_{i+1} should only be able to identify that the token was issued originally by A_1 and that it has been verifiably transferred at each hop culminating at A_i. The token should not reveal the identities of previous holders of the token including the fact that it was transferred from A_{i-1} to A_i. Therefore, a user in the transfer chain should only know who the token issuer is, who the token was received from and to who it is transferred. A user not in the transfer chain(example, someone who snoops a token off the wire) can at most know the identity of the token issuer as this information does not reveal

any of his interactions. This notion of privacy should be preserved even if a user is involved in multiple legs of the token chain. We assume that the token issuer, for tokens issued by him, never acts maliciously. For a more detailed and formal definition of our anonymity requirements please refer to section 6.3.3.

A scheme that satisfies the above requirements can address the introduction problem. On the other hand, a user's willingness (or unwillingness) to continue interactions with another user, is useful in capturing the evolving nature of social networks. Social network connections can be fleeting as in the travel agent example, or can be deactivated after a longer association (for example, relationships gone sour). Deactivation can occur even on account of malicious entities. For example a malicious entity might gain the trust of users and then start behaving maliciously. Essentially, a scheme that assumes a user's behavior is going to always remain the same fails to realize any of these scenarios. It is in this light that being able to capture a user's continued endorsement of a social tie is a desirable goal. Towards this the token system needs to satisfy the following requirement:

Limiting Token Reuse: If a token needs to be able to capture a user's willingness for continued interaction it cannot be reused infinitely. Revisiting the travel agent example, we see that a non revocable token cannot capture the user's unwillingness to talk to the travel agent and his SN. Therefore tokens need to either be single use, have a restricted lifetime or support a revocation policy. Single use tokens can provide a fair exchange for users' interaction time. In the VoIP setting, if A_1 talked to A_2 for 10 minutes then the token T^{A_1} provides A_2 or his social network the ability to talk to A_1 for a proportional period of time. If A_1 is no longer willing to talk to A_2, as in the travel agent example, then A_2 only has a fixed supply of $A_1's$ tokens that will eventually run out. A more time sensitive approach is the use of token lifetimes where tokens expire after a specified time limit. However, determining what is a good token lifetime is hard, particularly when tokens are transferable, as the time between token issue and token use will vary. In addition, token lifetimes reveal information about the time of token issue. A more elaborate mechanism is incorporating token revocation. Anytime a token issuer would like to deactivate a link to another user and his SN, he could send a token revocation to the user. However, this would require the user to keep token state of all the other users to whom that token was transferred. All users who received this token would also need to maintain similar state and an elaborate revocation propagation mechanism would need to be put in place. Single use tokens seem a practical token control mechanism that can be tailored to maintain a high level of privacy and provide a fair prediction of a user's willingness to interact. Single use tokens also have the desirable property that they favor new user introductions through more established interaction paths.

A scheme that satisfies the above requirements will need to create a token framework that has single use transferable tokens that can capture social network interactions in a privacy preserving manner.

3 Possible Approaches and Related Work

Before arriving at our proposed solution, we considered a number of possible approaches. None of them satisfy all the requirements but provide an insight into the challenges of creating a feasible scheme.

The simplest construct would be creating a token using a message authentication code (MAC) under the secret key of the issuer. The message can contain a serial number that ensures the uniqueness of the token. The token issuer can always verify the validity of the token once it is submitted back to him. However, none of the other users can verify that the MAC was indeed generated by the issuer. To address this, we can create tokens using digital signatures (DS)[5, 14]. The token issuer signs the token with his secret key and any user can verify that the token is generated by the issuer. However, this does not verifiably prove the existence of a transfer path. A malicious entity can snoop the token off the wire and make many copies of the token and transfer it across different paths. The serial number would prevent the token from being reused but there will be no way of identifying the user who made copies (double spent) of the token. Since the token double spender cannot be caught, tokens themselves become useless and cannot really ensure fair use of the system.

To prove the existence of a transfer path, user certificates could be employed to validate the transfer. If a user A_1 wants to issue a token T^{A_1} to user A_2, he can associate a certificate with the token by signing A_2's public key with his secret key, $Cert_{A_1}(pk_{A_2})$. A_2 can use a signature with his secret key to prove to any user that he holds a valid certificate from A_1. A_2 can transfer the token to A_3 and in doing so, needs to provide a similar certificate for A_3, $Cert_{A_2}(pk_{A_3})$. A_3 now holds the token and the associated certificate chain $(Cert_{A_1}(pk_{A_2}), Cert_{A_2}(pk_{A_3}))$ to prove that he is the valid owner of the token. To prove the validity of a token any user must show an associated certificate chain that leads up to him. However, this clearly reveals all the interactions that have occurred so far. For example, when user A_3 further transfers the token to A_4, he has to reveal the certificate to prove token validity, which in turn reveals the interaction $A_2 \rightarrow A_3$.

To hide the identity of a user in a certificate chain, we can assume each user belongs to a group and use group signatures. Since we don't have to hide the token issuer's identity, A_1 can issue a certificate signing A_2's group public key with his secret key, yielding $Cert_{A_1}(pk_{A_2^G})$, where A_2^G is the group A_2 belongs to. When A_2 transfers the token to A_3, he can prove that he is part of group A_2^G by providing a signature using the group secret key. He then provides a certificate of the form $Cert_{A_2^G}(pk_{A_3^G})$ to complete the token transfer. The certificate chain $(Cert_{A_1}(pk_{A_2^G}), Cert_{A_2^G}(pk_{A_3^G}))$ allows A_3, who is part of the group A_3^G to prove he has a valid token. When A_3 wants to further transfer the token to A_4 he can reveal this certificate chain. A_4 only gets to know that the original issuer of the token is A_1, and that some member of group A_2^G transferred the token to A_3. He no longer gets to know the identity of A_2. This scheme seems to capture the transfer path in a privacy preserving manner except for one problem. A_4 can decide to transfer the token to A_2 as he does not know that A_2 was previously an owner of this token. Though the associated credential chain is of the form $(Cert_{A_1}(pk_{A_2^G}), Cert_{A_2^G}(pk_{A_3^G}), Cert_{A_3^G}(pk_{A_4^G}))$, due to the uniqueness of the embedded information in the token(for example, the serial number), A_2 knows this is the same token that he transferred to A_3. Due to the deterministic nature of the way the token grows in size (this cannot be avoided[16]), A_2 also knows that this token has undergone only one transfer and therefore knows $A_3 \rightarrow A_4$, again a loss in privacy. In addition, for group signatures, clients have the overhead of creating sub groups and electing group managers. We could use ring signatures but since members of a ring need not voluntarily participate, the trustworthiness of a transfer path significantly degrades. We could avoid using groups or rings completely by using a zero knowledge (ZK) proof system to hide the identity of previous owners of a token. However, just like the group signature scheme, when a previous owner of a token sees the token again, he will be able to glean private interaction information. Detecting a cycle is impossible as a privacy preserving solution cannot reveal previous owners of a token. However, if cycle detection is impossible then we need to limit the maximum number of hops that a token can be transferred. An added requirement to our token system is that, in order to avoid looping in a cycle forever, the token system should be able to restrict the number of hops.

The occurrence of cycles in a token transfer path forms the hardest challenge in determining a solution that provides accountability and does not leak privacy. The key observation here is that for a token to not leak privacy, *all the information it carries must be sufficiently randomized at each transfer* such that a user who has seen a token previously cannot identify it when it is transferred to him again. We use randomizable proofs[22] and delegatable anonymous credentials [7] to achieve this.

Other Related Work: Tokens have been used to ensure fairness and provide accountability in P2P systems[26, 17, 2]. Anagnostakis et al[2] advocate the notion of transferable tokens and shows the improvements in scalability and redundancy afforded by introducing such tokens. An alternative to tokens for accountability in P2P systems is the use of micropayments[35, 32, 25, 23]. However none of these schemes support both anonymity and unforgeability of the transfer path. Our system is similar to [32] in the sense that each user can issue his own currency, however [32] is an online system with the issuer having to partake whenever his currency is being transferred.

Adding privacy requirements to incentive mechanisms(like tokens) has been studied extensively in repu-

Algorithm Name	Description
$\mathsf{GSSetup}(1^k)$	Generates common parameters, par^{GS}, shared by all users.
$\mathsf{GSCommit}(x, o, par^{GS})$	Creates a commitment of secret, x with opening, o.
$\mathsf{GSGenProof}((C_i^{x_i}(x_i, r_i))_{1\leq i\leq n}, \mathsf{cond}, par^{GS})$	Makes an efficient NIZK proof, π which shows that the values committed in $C_i^{x_i}$ satisfy cond.
$\mathsf{GSVerify}((C_i^{x_i})_{1\leq i\leq n}, \pi, \mathsf{pubinfo}, \mathsf{cond}, par^{GS})$	If π is a correct proof of cond, it will return 1 else 0. pubinfo gives public information that tells the verifier that the commitments satisfying the proof hide some meaningful value.
$\mathsf{GSRand}((C_i^{x_i})_{1\leq i\leq n}, \pi, \mathsf{pubinfo}, \mathsf{cond}, par^{GS})$	Generates random opening values $(o'_i)_{1\leq i\leq n}$ and updates $C_i^{x_i}$ using o'_i, to give $C_i'^{x_i}$ whose updated opening values are $o_i + o'_i$. It also updates the proof with o'_i yielding $(C_i'^{x_i})_{1\leq i\leq n}, \pi'$. π' can also be verified using GSVerify, if o'_i are known. π and π' are unlinkable.

Table 1: GS Proof System Cheat Sheet

tation and recommender systems [29] and social networks[14], utilizing a host of cryptographic techniques. Laurent et al[10] use group signatures, while Carminati et al[14] use digital signatures to provide anonymity. Kai et al[34] use group signatures to add anonymity to the micropayment scheme proposed in[35]. For our setting, we have demonstrated how any of these techniques would still leak privacy. Belenkiy et al[8] try to achieve accountability without losing privacy by using an e-cash mechanism to provide a currency model in P2P. However their system requires a bank and does not support transferable coins. A transferable e-cash mechanism using the meta proof technique is described in Canard et al[13]. However, the meta proof technique is a general circuit based proof that is inefficient in practice. Since currency is meaningful only to the resource/service provider, we allow users to act as banks in their own right, creating, issuing and transferring tokens to each other. Enhancing delegatable anonymous credentials allows for such a system while providing strong privacy guarantees.

4 Building Blocks

4.1 Efficient Non Interactive Zero Knowledge (NIZK) Randomizable Proof System

ZK proof systems can prove the existence of a transfer path while maintaining anonymity of previous owners of the token. However anonymity is lost if the same user is involved in multiple legs (interactions) of the transfer. To achieve anonymity in this case, the underlying proof system needs to be randomizable such that the proofs when presented in different legs of the transfer are unlinkable to each other. The Groth Sahai NIZK proof system[22] has this desirable property. It allows the translation of proof statements arising in real world settings to be expressed as relations between group elements. The proof system is summarized by the set of algorithms shown in table 1. Chase et al[7] showed that a proof, π, in this proof system can be randomized without knowing the original committed secrets or their openings, which is captured by the algorithm GSRand in table 1. They use this notion of randomizability to create delegatable anonymous credentials, which we discuss next.

4.2 Delegatable Anonymous Credentials

Delegatable anonymous credentials (DACs) can be used to delegate access rights repeatedly without revealing the identity of the participants. In order to achieve a high level of anonymity, it requires an underlying proof system that offers randomizability, like the GS proof system. As described in section 3, a user certificate can be used to delegate access rights but leaks privacy. DACs provide similar functionality as a user certificate but ensure that the certificates do not reveal identity and can be randomized. Towards achieving this, Chase et al[7]

Algorithm Name	**Description**
$\mathsf{AuthSetup}(1^k)$	Generates groups G_1, G_2, G_T of prime order p whose bit length is proportional to k, a bilinear map $e : G_1 \times G_2 \rightarrow G_T$, and group elements $g, u, u^*, u_1, \cdots, u_n \in G_1$ and $h \in G_2$. It outputs the complete parameter list $par_A = (G_1, G_2, e, p, g, u, u^*, u_1, \cdots, u_n, h)$.
$\mathsf{Auth}(sk, \vec{m} = (m_1, \cdots, m_n), par_A)$	Generates $sk \xleftarrow{\$} \mathbb{Z}_p$ and $pk \leftarrow h^{sk}$, and returns (sk, pk).
$\mathsf{VerifyAuth}(pk, \vec{m} = (m_1, \cdots, m_n), auth^{sk \rightarrow \vec{m}}, par_A)$	Parses $auth^{sk \rightarrow \vec{m}} = (A^*, B^*, C^*, \{A_i, B_i, C_i, D_i\}_{1 \leq i \leq n}$ and verifies $\{e(A^*, pk \cdot B^*) \cdot e(g, h^{-1}) = 1 \wedge e(u^*, B^*) \cdot e(C^*, h^{-1}) = 1 \wedge_{1 \leq i \leq n} (e(D_i, B_i h^{m_i}) \cdot e(g, h^{-1}) = 1)\}$. Returns 1, if all equations match, else 0.

Table 2: DAC Cheat Sheet

create an authentication scheme with certain properties.

4.2.1 Authentication Scheme

The authentication scheme creates a tag that authenticates a vector of messages under a secret key. For example, user A_1 can authenticate a set of messages, $\vec{m}$ under his secret key sk^{A_1}. If $\vec{m}$ includes the secret key of another user A_2, sk^{A_2}, the tag becomes a user certificate from A_1 to A_2. The scheme is summarized by the set of algorithms shown in table 2. Using Auth, a user A_1 can authenticate the secret key of user A_2. However, since A_2's secret key should not be revealed to A_1 we carry out a secure two party computation (2PC) of the authentication scheme between A_1 and A_2, summarized below:

- $\mathsf{2PCAuth}(\mathcal{I}(sk_{\mathcal{I}}, \{m_i\}_{1 \leq i \leq l}), \mathcal{O}(pk_{\mathcal{I}}, \{m_i\}_{1 \leq i \leq n}))$ is a secure two party computation between an authentication issuer $\mathcal{I}$ and a message owner $\mathcal{O}$ such that $\mathcal{I}$ does not get any information about $(m_i)_{l+1 \leq i \leq n}$ as well as $\{g^{\frac{1}{K_i + m_i}}\}_{l+1 \leq i \leq n}$.

4.2.2 Making a NIZK proof of the authenticator

If A_1 and A_2 ran 2PCAuth, A_2 possesses an authenticator $auth^{A_1 \rightarrow A_2}$ from A_1, essentially a certificate on his secret key. Such an authenticator itself is unchanging and therefore reveals the identity of a user. The DAC system uses the notion of user pseudonyms to get around this. In pseudonym systems[28], a user has a single secret key but multiple public keys. User A_2 who has secret key sk^{A_2}, can choose a random value o and use the commitment $Commit(sk^{A_2}, o)$ as a public key. Different values of o result in different public keys or pseudonyms for the same user. A_2 can be known to user A_1 with public key pk^{A_2} and to user A_3 with public key pk'^{A_2}. Though an adversary cannot link pk^{A_2} and pk'^{A_2}, user A_2 can prove that they are actually commitments to the same secret. In this case A_1 rather than provide the authenticator directly, provides an NIZK proof for the authenticator, $\pi^{A_1 \rightarrow A_2}$ authenticating the contents of pseudonym pk^{A_2}. When A_2 wants to delegate his access right to A_3, he randomizes the pseudonym, proof pair $(pk^{A_2}, \pi^{A_1 \rightarrow A_2})$ to $(pk'^{A_2}, \pi'^{A_1 \rightarrow A_2})$ where $\pi'^{A_1 \rightarrow A_2}$ authenticates the contents of pk'^{A_2}. It uses the GSRand algorithm to achieve this. The new pseudonym, proof pair can be further randomized by users who don't know the underlying contents, using the GSRand algorithm, and this provides the strong unlinkability guarantees when the token is transferred hop by hop. The actual commitment scheme used by DACs is a deterministic double commitment scheme, $\mathsf{DoubleCommit}(x, open = (o_1, o_2))$ that outputs (C^x, π). $C^x = (C_1^x, C_2^x) = (\mathsf{GSCommit}(g^x, o_1), \mathsf{GSCommit}(h^x, o_2))$. π is NIZK GS proof that proves both GS commitments committed the same secret. This allows f-extractability of such proofs[7].

Though DACs provide the desirable level of anonymity, they do not support the notion of single use tokens. When access rights are delegated to a user, the user can repeatedly delegate the rights to any number of other users. In the VoIP setting, a user who has a token can transfer it to a large number of other users, all of whom can call the token issuer. In addition, once a user has a token, he can reuse it multiple times, regardless of the issuer's interaction experiences with that user. To address this we extend DACs to create single use tokens.

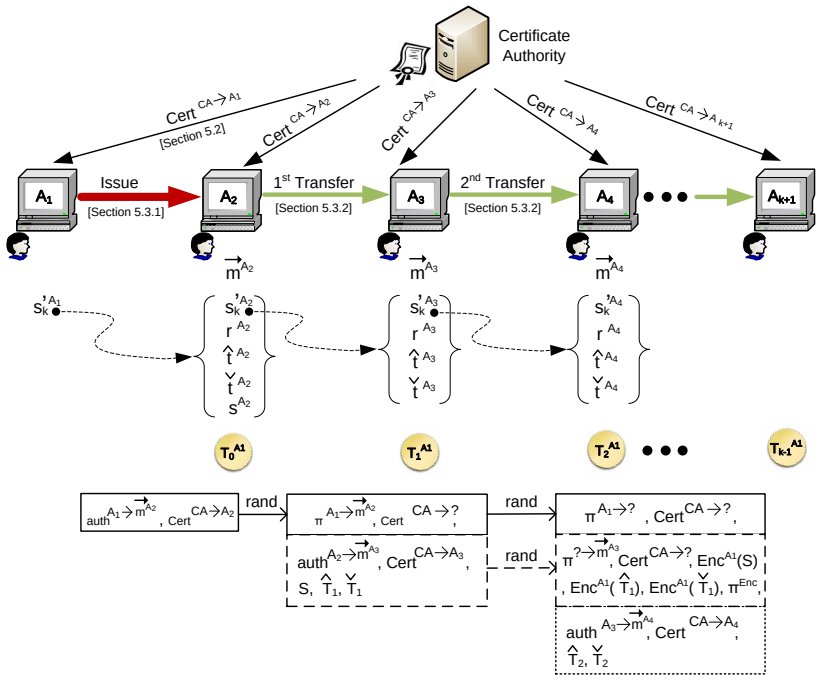

Figure 3: Single Use Anonymous Transferable Token Scheme

5 Single Use Anonymous Transferable Token Scheme

DACs lend themselves well to providing tokens that are both transferable and privacy preserving. To support single use tokens, we extend DACs by using techniques employed in e-cash schemes to create single use coins[11, 13]. In e-cash each coin is associated with a serial number and tags. The serial number is used to prevent coin duplication, the tags are used to catch the double spender responsible for this duplication. The primary contribution of our scheme is (1) showing how to create serial number and tags using the DAC's authenticator, that ensure single use of tokens and double spender identification and (2) how to randomize these new components such that token unlinkability is still preserved.

In the DAC setting, users have one secret key and many public keys. In order to catch double spenders we need to have at least one of these public keys registered with a certificate authority (CA). In VoIP systems, the authentication server of VoIP providers like Skype, Google Talk and Vonage can play the role of the CA. In fact, in Skype, user accounts are already associated with a public key, which they use for communication. We, however need a new certificate issuing protocol because a conventional certificate reveals the identity of the certificate holder and we need one that reveals the identity only in direct interaction with a user and when the user behaves dishonestly (for example, double spends a token). The overall scheme is shown in figure 3. We start by describing the cryptographic setting including the keys that each user needs to generate in section 5.1. Thereafter, section 5.2 describes the certificate issuing protocol and section 5.3 describes the token framework and the structure of the serial number and tags used to make single use tokens.

5.1 Cryptographic Preliminaries

We use the GS proof system, the DAC authentication scheme and ElGamal encryption to meet the different requirements of the token framework. These constructs require common parameters, described in ParamGen, that are shared across all users. The clients also need to generate a set of keys for different parts of the scheme, and this is described in KeyGen.

• $\mathsf{ParamGen}(1^k)$ is probabilistic algorithm that outputs the common parameters for the token scheme, par^{TS}. It runs $\mathsf{GSSetup}(1^k)$(section 4.1) to get par^{GS} and $\mathsf{AuthSetup}(1^k)$(section 4.2) to get par^{A}. Then it generates generators $\bar{g} \in G_1$ and $\bar{h} \in G_2$ for the ElGamal encryption. It returns $par^{TS} = (par^{GS}, par^{A}, \bar{g}, \bar{h})$. These

common parameters are shared by all users of the system and are used for all the token operations.
• $\mathsf{KeyGen}(par^{TS})$ is a probabilistic algorithm that outputs the key pair for a user, A_i, (sk^{A_i}, pk^{A_i}) and is run by each user after they obtain the common parameters. This algorithm parses par^{TS} and uses par^A to generate $(sk'^{A_i}, pk_1'^{A_i}) \leftarrow \mathsf{AuthKg}(par^A)$ (section 4.1). Remember $pk_1'^{A_i} \leftarrow h^{(sk_1'^{A_i})}$. It then computes another public key, $pk_2'^{A_i} \leftarrow u^{sk'^{A_i}} (\in G_1)$. The certificate issuing protocol (section 4.2) and the token issue protocol is run with only these keys. The algorithm then generates $\bar{sk}^{A_i} \xleftarrow{\$} G_1$. It uses this secret key to compute $\bar{pk}_1^{A_i} \leftarrow \bar{g}^{\bar{sk}^{A_i}}$ and $\bar{pk}_2^{A_i} \leftarrow \bar{h}^{\bar{sk}^{A_i}}$. These keys are used for ElGamal encryption whenever A_i is a token issuer. Finally, $sk^{A_i} \leftarrow (sk'^{A_i}, \bar{sk}^{A_i})$ and $pk^{A_i} \leftarrow (pk'^{A_i}, \bar{pk}^{A_i}) \leftarrow ((pk_1'^{A_i}, pk_2'^{A_i}), (\bar{pk}_1^{A_i}, \bar{pk}_2^{A_i}))$.

5.2 Certificate Issuing Protocol

The certificate issued by a certificate authority, CA, needs to have the following properties: (1) When two users are directly interacting with each other, the certificate for each user should clearly reveal their identities. This ensures that each user knows that the other has been certified by the CA. (2) It should be randomizable so that it can be transferred with a token. On randomization the certificate should no longer reveal the identity of the user but should still verifiably prove that the certificate has been generated by CA. (3) The certificate information should help identify a token double spender. In this section, we show how we can satisfy (1) and (2). After introducing the rest of the scheme we show how (3) can also be satisfied.

To achieve (1) and (2), we use the authentication scheme of DAC. The details of the protocol between the CA and a user A_i are shown in protocol 1. For brevity, all protocols described hereafter do not explicitly specify the steps in case certain conditions are not met (for example, when proofs don't verify, sub protocols do not complete successfuly). In all such cases the protocol is immediately aborted. We assume that through some previous interaction the CA has the user's public key, pk'^{A_i}. As a first step, (Line 1) CA and A_i run a 2PC for creating a NIZKPK of an authenticator ([7]) with A_i as the authentication issuer, the secret key of CA as the message being authenticated, and with zero value openings. This yields $\pi^{sk'^{A_i} \rightarrow \{sk'^{CA}\}}$ which is a proof that the commitment or the pseudonym $(C_1^{sk'^{A_i}}, C_2^{sk'^{A_i}}) \leftarrow \mathsf{DoubleCommit}(sk'^{A_i}, (0,0))$ authenticates the secret key of the CA. From the definition of DoubleCommit we can see that the $(C_1^{sk'^{A_i}}, C_2^{sk'^{A_i}})$ are exactly the public keys of A_i, $(pk_1'^{A_i}, pk_2'^{A_i})$. CA verifies that this is the case and if so stores A_i's information and the proof, $\pi^{sk'^{A_i} \rightarrow \{sk'^{CA}\}}$ as the certificate and then sends this certificate to A_i (Line 3 and 4). $\pi^{sk'^{A_i} \rightarrow \{sk'^{CA}\}}$ clearly reveals the identity of A_i and hence achieves (1). Since the DAC authentication scheme uses the GS proof system, the proofs themselves can be randomized using GSRand. On randomization the proof only shows that some user's secret key was certified by the CA but does not reveal the identity corresponding to the secret key. This gives other users a way to reveal A_i's certificate without revealing that they interacted with him, achieving (2). In figure 3, the CA initially certifies all users. When A_2 interacts with A_3 directly, he includes his unanonymized certificate to A_3 in the token. However after A_3 randomizes the certificate, it no longer reveals the identity of A_2.

Protocol 1 $\mathsf{IssueCertificate}(CA(sk'^{CA}, pk'^{A_i}), A_i(sk'^{A_i}, pk'^{CA}))$

1: Run 2PC Protocol for generating NIZKPK of authenticator with zero value openings, CA gets $\pi^{sk'^{A_i} \rightarrow \{sk'^{CA}\}}$
2: **if** (2 PC protocol finishes successfully) and (CA verifies $\pi^{sk'^{A_i} \rightarrow \{sk'^{CA}\}}$ is generated correctly) **then**
3: CA: Store $(\mathsf{A_i}, pk'^{A_i}, \pi^{sk'^{A_i} \rightarrow \{sk'^{CA}\}})$.
4: CA: Send $\pi^{sk'^{A_i} \rightarrow \{sk'^{CA}\}}$ to A_i.
5: A_i: Set certificate $\mathsf{Cert}^{\mathsf{CA} \rightarrow \mathsf{A_i}}$ as $\pi^{sk'^{A_i} \rightarrow \{sk'^{CA}\}}$
6: **end if**

5.3 Token Framework

5.3.1 Token Issue

A user issues a token to other users after a direct interaction with them. To ensure tokens are single use we use serial numbers, similar to e-cash[11, 13]. To preserve privacy, the serial number should not be generated by the token issuer. To see this, consider a token that is issued by A_1 and follows the path $A_1 \rightarrow A_2 \rightarrow A_3$ after which A_3 submits the token back to A_1. If A_1 generated the serial number, S, when it is submitted back, along with the deterministic increase in token length[15], A_1 gets to know the interaction $A_2 \rightarrow A_3$. To prevent this, we need A_2 to generate the serial number. Single use of tokens can still be ensured as long as A_1 caches the serial numbers of all tokens submitted to him. The complete token issue protocol is shown in protocol 2. Initially (line 1), A_2 generates a set of random values, s^{A_2}, which is used to create the serial number and $r^{A_2}, \hat{t}^{A_2}, \check{t}^{A_2}$, which are used to create the tags. We use 2PCAuth so that A_1 authenticates these random values without getting to know them (line 2). This allows A_2 to create the serial numbers without revealing them to A_1. A_2 verifies the resulting authenticator and stores the authenticator and his certificate as a direct (0-transfer) token, $\mathsf{Token}_0^{A_1}$ (line 4), abbreviated as $T_0^{A_1}$ in figure 3.

Protocol 2 IssueToken$(A_1(sk'^{A_1}, pk'^{A_2}), A_2(sk'^{A_2}, pk'^{A_1}))$

1: A_2: Generate $s^{A_2}, r^{A_2}, \hat{t}^{A_2}, \check{t}^{A_2} \xleftarrow{\$} \mathbb{Z}_p$.
2: Run 2PCAuth$(A_1(sk'^{A_1}, \{\}), A_2(pk_1'^{A_1}, \vec{m}^{A_2} = \{sk'^{A_2}, s^{A_2}, r^{A_2}, \hat{t}^{A_2}, \check{t}^{A_2}\}))$, A_2 gets $auth^{sk'^{A_1} \rightarrow \vec{m}^{A_2}}$
3: **if** (2PCAuth successfully finishes) and (VerifyAuth($pk_1'^{A_1}$, $\{sk'^{A_1}\}$, $auth^{sk'^{A_1} \rightarrow \vec{m}^{A_2}}$)) **then**
4: A_2: Set token $\mathsf{Token}_0^{A_1} \leftarrow (auth^{sk'^{A_1} \rightarrow \vec{m}^{A_2}}, Cert^{CA \rightarrow A_2})$.
5: **end if**

5.3.2 Token Transfer

Tokens are transferred to allow new users to interact with the token issuer. If tokens are transferred only on interactions, the transfer path represents the interaction path. For example, in the VoIP setting, if tokens are transferred only at the end of a call, the transfer path represents the SN path from the token issuer. This path can then be used to prove higher order constructs such as weak social ties. On direct interaction the identity information of interacting entities does not need to be anonymized. However, as the token is transferred all previous interaction information in the token needs to be randomized to achieve unlinkability. The crux of token transfer is to show how we can randomize the different parts of the token. Towards this we divide the protocol into three cases: (1) the first transfer, that occurs between users who are one hop and two hops away, (2) the second transfer, between users two hops and three hops away and (3) N^{th} transfer, any transfer after the second transfer.

First transfer (Case $i = 0$): In figure 3 when user A_2 transfers a token to user A_3, the *first transfer* protocol described in protocol 3 is carried out. The token contains the authenticator from A_1 to A_2, $auth^{sk'^{A_1} \rightarrow \vec{m}^{A_2}}$ and A_2's certificate. To randomize the authenticator, A_2 creates a NIZKPK for it using commitments of the secrets that were authenticated (Line 1 and 2). To create a ZK proof two independent commitments, committing to the same secret are required ([22]), something that only A_2 can do. Since there is nothing to hide at this point, A_2 makes commitments with 0 opening values, as then the commitment, $C^{sk'^{A_2}}$ is essentially his public key and this can be verified with his certificate. In addition his certificate is randomized to yield $Cert^{CA \rightarrow ?}$ (Line 3). The above steps make sure that the previous form of the token, namely Token_0, has been suitably randomized. A_2 uses the authenticated set of values to create serial numbers and tags. The serial number is obtained from s, using the Dodis and Yampolskiy PRF[18], as $S \leftarrow g^{\frac{1}{sk'^{A_2}+s^{A_2}}}$ (Line 4). The remaining elements of the

authenticated message vector(except the secret key) are used to compute tags, to identify a token duplicator. A token is duplicated if a user tries to transfer the same token (same serial number) twice. A tag then should reveal the identity of a user only when he has duplicated a token and should not leak any identity information otherwise. We achieve this by constructing a tag that has contributions from both users in the interaction. The user initiating the token transfer creates partial tags and the user receiving the token completes the tags such that if the same token was transferred twice, the two resulting completed tags are different enough to yield the token duplicator's identity. Towards this end, when user A_2 transfers the token to A_3, he creates tags of the form, $\hat{T}_1^{A_2} \leftarrow g^{\frac{1}{sk'^{A_2}+\hat{t}^{A_2}}}$, $\hat{T}_2^{A_2} \leftarrow pk_2'^{A_2} \cdot (\hat{T}_1^{A_2})^{r^{A_2}}$ (Line 4).

The NIZKPK of the authenticator, the certificate, the commitments and the serial number and partial tags are all sent to A_3 (Line 5). A_3 verifies that $\pi^{sk'^{A_1} \rightarrow \vec{m}^{A_2}}$ is a valid proof of the commitments (Line 6). In addition it checks that the serial numbers and tags are formed correctly using the same commitments, through a set of equations (Line 9). These equations are based on one of the properties of a bilinear group, namely $e(g^a, h^b) = e(g, h)^{ab}$. These verifications along with a check to verify A_2's certificate(Line 7) ensures that A_2 has a valid authenticator from A_1 and that only the secrets authenticated were used to create the serial number and tags. In order to make sure that A_2 does not double spend, A_3 creates the completed tags by generating a random number, r^{A_3} and then calculating $\hat{T}_{11} \leftarrow \hat{T}_2^{A_2} \cdot (\hat{T}_1^{A_2})^{sk'^{A_3}}$, $\hat{T}_{12} \leftarrow pk_1'^{A_3} \cdot C^{r^{A_2}}$, $\check{T}_{11} \leftarrow pk_2'^{A_2} \cdot (\check{T}^{A_2})^{r^{A_3}}$, and $\check{T}_{12} \leftarrow h^{r^{A_3}}$(Line 12). The public key that satisfies the following equation is the public key of the double spender, and this is explained subsequently.

$$\frac{e(\hat{T}_{l1}, \hat{T'}_{l2})}{e(\hat{T'}_{l1}, \hat{T}_{l2})} = e(pk^{DS}, \frac{\hat{T'}_{l2}}{\hat{T}_{l2}}) \tag{1}$$

To see why this is the case, consider a user A_i transferring the same token to both A_j and A_k.

For A_j

$$\hat{T}_{l1} \leftarrow pk_2'^{A_i} \cdot (\hat{T}_1^{A_i})^{sk'^{A_j}+r^{A_i}}$$
$$\hat{T}_{l2} \leftarrow (h)^{sk'^{A_j}+r^{A_i}}$$

For A_k

$$\hat{T'}_{l1} \leftarrow pk_2'^{A_i} \cdot (\hat{T}_1^{A_i})^{sk'^{A_k}+r^{A_i}}$$
$$\hat{T'}_{l2} \leftarrow (h)^{sk'^{A_k}+r^{A_i}}$$

The random values, $r^{A_2}, \hat{t}^{A_2}, \check{t}^{A_2}$ used to generate the token tags will be the same when A_i transfers the token to both A_j and A_k. Plugging in these values into the left hand side of equation 1 gives:

$$\frac{e(\hat{T}_{l1}, \hat{T'}_{l2})}{e(\hat{T'}_{l1}, \hat{T}_{l2})} = \frac{e(pk_2'^{A_i} \cdot (\hat{T}_1^{A_i})^{sk'^{A_j}+r^{A_i}}, (h)^{sk'^{A_k}+r^{A_i}})}{e(pk_2'^{A_i} \cdot (\hat{T}_1^{A_i})^{sk'^{A_k}+r^{A_i}}, (h)^{sk'^{A_j}+r^{A_i}})} = \frac{e(pk_2'^{A_i}, (h)^{sk'^{A_k}+r^{A_i}})}{e(pk_2'^{A_i}, (h)^{sk'^{A_j}+r^{A_i}})} = e(pk_2'^{A_i}, \frac{\hat{T'}_{l2}}{\hat{T}_{l2}})$$

From this, $pk^{DS} = pk_2'^{A_i}$ and thus, A_i will be correctly identified as the double spender. This explains the need for the first set of tags, $\hat{T}_{l1}$ and $\hat{T}_{l2}$. The second set of tags helps catch the token double spender if he transfers the same token twice to the same user. This is useful, as users never need to store the details of a token once they have transferred it. In this case, consider the user A_i who transfers the same token twice to A_j. The first set of tags will both be of the form $(pk_2'^{A_i} \cdot (\hat{T}_1^{A_i})^{sk'^{A_j}+r^{A_i}}, (h)^{sk'^{A_j}+r^{A_i}})$. On the other hand, the second set of tags, $(\check{T}_{l1}, \check{T}_{l2})$ will be different as A_j generates a new random number, r^{A_j} for each token transfer. In this case a similar equation to equation 1 can be used to identify A_i as the double spender, and is shown in equation 2.

$$\frac{e(\check{T}_{l1}, \check{T'}_{l2})}{e(\check{T'}_{l1}, \check{T}_{l2})} = e(pk^{DS}, \frac{\check{T'}_{l2}}{\check{T}_{l2}}) \tag{2}$$

Finally, to complete the token transfer A_2 needs to authenticate A_3's secrets to transfer token ownership, which is done by running 2PCAuth (Line 13). As shown in figure 3 the authenticated message vector contains

A_3's secret key, sk'^{A_3}, and the tag creating secrets $r^{A_3}, \hat{t}^{A_3}, \check{t}^{A_3}$. The tag secrets will be used to create the tags for the next transfer. Since the serial number for the token has already been created we do not need another secret for it. The new token Token_1 is as shown in figure 3 ($T_1^{A_1}$) and contains the randomized components of the old token, and the newly computed components appended to it(Line 15).

Protocol 3 $\mathsf{TransferToken}(A_2(sk'^{A_2}, pk'^{A_1}, pk'^{A_3}, \mathsf{Token}_i^{A_1}), A_3(sk'^{A_3}, pk'^{A_1}, pk'^{A_2}))$ $i = 0$ case

1: A_2: Make 0 opening value commitments for $sk'^{A_2}, s^{A_2}, r^{A_2}, \hat{t}^{A_2}, \check{t}^{A_2}$, represented by $C^{sk'^{A_2}}, C^{s^{A_2}}, C^{r^{A_2}}, C^{\hat{t}^{A_2}}, C^{\check{t}^{A_2}}$.
2: A_2: Use commitments to create an NIZK proof of $auth^{sk'^{A_1} \rightarrow \vec{m}^{A_2}}$ using $\mathsf{GSGenProof}$, yielding $\pi^{sk'^{A_1} \rightarrow \vec{m}^{A_2}}$.
3: A_2: Get randomized certificate, $Cert^{CA \rightarrow ?}$ by executing $\mathsf{RandomizeCertificate}(Cert^{CA \rightarrow A_2})$.
4: A_2: Compute serial number, $S \leftarrow g^{\frac{1}{sk'^{A_2}+s^{A_2}}}$ and partial tags, $\hat{T}_1^{A_2} \leftarrow g^{\frac{1}{sk'^{A_2}+\hat{t}^{A_2}}}$, $\hat{T}_2^{A_2} \leftarrow pk_2'^{A_2} \cdot (\hat{T}_1^{A_2})^{r^{A_2}}$, $\check{T}^{A_2} \leftarrow g^{\frac{1}{sk'^{A_2}+\check{t}^{A_2}}}$.
5: A_2: Send $(C^{sk'^{A_2}}, C^{s^{A_2}}, C^{r^{A_2}}, C^{\hat{t}^{A_2}}, C^{\check{t}^{A_2}}, \pi^{sk'^{A_1} \rightarrow \vec{m}^{A_2}}, Cert^{CA \rightarrow ?}, S, \hat{T}_1^{A_2}, \hat{T}_2^{A_2}, \check{T}^{A_2})$ to A_3.
6: A_3: Verify $\pi^{sk'^{A_1} \rightarrow \vec{m}^{A_2}}$ is a valid proof for the commitments using $\mathsf{GSVerify}$.
7: A_3: Verify $Cert^{CA \rightarrow ?}$ using $\mathsf{VerifyCertificate}$.
8: A_3: Parse public key of A_2, pk'^{A_2} as $(pk_1'^{A_2}, pk_2'^{A_2})$.
9: A_3: Use the commitments and A_2's public key information to check the correctness of the serial number and tags generated using the following equations: $e(S, pk_1'^{A_2} \cdot C^{s^{A_2}}) = e(g,h) \wedge e(\hat{T}_1^{A_2}, pk_1'^{A_2} \cdot C^{\hat{t}^{A_2}}) = e(g,h) \wedge e(\hat{T}_2^{A_2}, pk_1'^{A_2} \cdot C^{\hat{t}^{A_2}}) = e(pk_2'^{A_2}, pk_1'^{A_2} \cdot C^{\hat{t}^{A_2}}) \cdot e(g, C^{r^{A_2}}) \wedge e(\check{T}^{A_2}, pk_1'^{A_2} \cdot C^{\check{t}^{A_2}}) = e(g,h)$.
10: **if** (all verification procedures succeed) **then**
11: A_3: Generate $r^{A_3} \xleftarrow{\$} \mathbb{Z}_p$
12: A_3: Contribute A_3 specific information in computing final tags, $\hat{T}_{11} \leftarrow \hat{T}_2^{A_2} \cdot (\hat{T}_1^{A_2})^{sk'^{A_3}}$, $\hat{T}_{12} \leftarrow pk_1'^{A_3} \cdot C^{r^{A_2}}$, $\check{T}_{11} \leftarrow pk_2'^{A_2} \cdot (\check{T}^{A_2})^{r^{A_3}}$, and $\check{T}_{12} \leftarrow h^{r^{A_3}}$
13: Run $\mathsf{2PCAuth}(A_2(sk'^{A_2}, \{\}), A_3(pk_1'^{A_2}, \vec{m}^{A_3} = \{sk'^{A_3}, r^{A_3}, \hat{t}^{A_3}, \check{t}^{A_3}\}))$, A_3 gets $auth^{sk'^{A_2} \rightarrow \vec{m}^{A_3}}$
14: **if** ($\mathsf{2PCAuth}$ finishes successfully) **then**
15: A_3: Set token $\mathsf{Token}_1 \leftarrow ((\pi^{sk'^{A_1} \rightarrow \vec{m}^{A_2}}, Cert^{CA \rightarrow ?}), (auth^{sk'^{A_2} \rightarrow \vec{m}^{A_3}}, Cert^{CA \rightarrow A_3}, S, \hat{T}_1 = \{\hat{T}_{11}, \hat{T}_{12}\}, \check{T}_1 = \{\check{T}_{11}, \check{T}_{12}\}))$
16: **end if**
17: **end if**

Second transfer (Case $i = 1$): After the first transfer, in addition to posessing two sets of authenticators and certificates, the token, Token_1, now contains the serial number and tags. When this token needs to be transferred the components need to be randomized yet again. Randomizing the authenticator and the certificate can be done as before. However, the serial number and tags, due to the entropy they carry, also need to be randomized. In addition, the token issuer, A_1 needs to be able to retrieve the original serial number and the tags, to detect and catch a double spender. To satisfy these requirements, we encrypt the serial number and tags with the public key of the issuer, $\bar{pk}^{A_1}$, a technique introduced in [12]. As shown in figure 3, when A_3 transfers the token to A_4, $(S, \hat{T}_{11}, \hat{T}_{12}, \check{T}_{11}, \check{T}_{12})$ gets encrypted to $(Enc^{A_1}(S), Enc^{A_1}(\hat{T}_{11}), Enc^{A_1}(\hat{T}_{12}), Enc^{A_1}(\check{T}_{11}), Enc^{A_1}(\check{T}_{12}))$, the two sets being unlinkable with each other. As older components of the token need to be randomized every transfer, the serial number and tags can be encrypted each time, still maintaining unlinkability of tokens. We use the ElGamal encryption as the token issuer requires a single decryption operation even if the contents (serial number and tags) are encrypted multiple times. Specifically, $Enc^{A_1}(S) \leftarrow ((\bar{pk}_2^{A_1})^{\bar{r}_1} \cdot S, \bar{g}^{\bar{r}_1})$, $Enc^{A_1}(\hat{T}_{11}) \leftarrow ((\bar{pk}_2^{A_1})^{\bar{r}_2} \cdot S, \bar{g}^{\bar{r}_2})$, $Enc^{A_1}(\hat{T}_{12}) \leftarrow ((\bar{pk}_1^{A_1})^{\bar{r}_3} \cdot S, \bar{g}^{\bar{r}_3})$, $Enc^{A_1}(\check{T}_{11}) \leftarrow ((\bar{pk}_2^{A_1})^{\bar{r}_4} \cdot S, \bar{g}^{\bar{r}_4})$, $Enc^{A_1}(\check{T}_{12}) \leftarrow ((\bar{pk}_1^{A_1})^{\bar{r}_5} \cdot S, \bar{g}^{\bar{r}_5})$, where $\bar{r}_1, \cdots, \bar{r}_5 \xleftarrow{\$} \mathbb{Z}_p$.

Encrypting the serial number and tags makes it hard to check if they have been generated correctly. To resolve this, we attach the ZK proofs that show that the encrypted serial number and tags are generated correctly. For example, for $Enc^{A_1}(S)$ whose first term is of the form $P \cdot S$, A_3 generates commitments of P and S, C^P and C^S respectively. It uses GSGenProof to create a proof that shows C^P and C^S have been generated correctly. It also creates a proof that the multiplication of C^P and C^S is the commitment to $P \cdot S$. Similar proofs are generated for the other encryptions. Using the GS proof system allows us to concatenate[22] all of the proofs generated, into one final proof, π_1^{Enc}. Once the token has been randomized, A_3 creates partial tags representing this interaction similar to *First Transfer* and sends all this information to A_4. Once A_4 verifies all the information, and completes the tags the two users run 2PCAuth to complete the token transfer. The final token is as shown in figure 3 ($T_2^{A_1}$)

Beyond Second Transfer (Case $i > 1$): As before, randomizing the components is the first step and generating new tags (representing this transfer) and the authenticator is the next step. Randomization is done as in the previous transfer cases, except that we have to re-encrypt the serial number and tags to preserve unlinkability. The ciphertexts encrypting the serial number and tags is of the form $(A = (\bar{g}^x)^r \cdot m, B = \bar{g}^r)$, where $\bar{g}^x$ is the public key of the decryptor. Then a user who has the ciphertext and the decryptor's public key can re-encrypt it again by computing $((\bar{g}^x)^{r'} \cdot A, \bar{g}^{r'} \cdot B)$. We can then modify the commitments and the proof according to the new random value r' using GSRand.

5.3.3 Token Submit

A user who submits a token makes a claim that there is a transfer/interaction path between the issuer and him. This claim, if verified, provides the token issuer with the ability to make decisions on whether to accept any further interaction. Token submission is similar to the corresponding token transfer as the submitter needs to randomize all the previous components of the token. For example, if A_2 submits the token that he directly received from A_1, he needs to carry out the same procedure as the *first transfer* protocol and create the serial number and tags for the token. This is because A_2, after submitting, could still duplicate the token by transferring it to another user. Similarly, any user A_j submitting the token should randomize the tokens by creating a proof for the authenticator, randomizing commitments, proof of authenticators, certificates and (re)encrypting serial number and tags. The only difference is that the authentication procedure between submitter and issuer need not be carried out. The issuer A_1 on receipt of the token must verify all the components to ensure that it is a valid token. If the token is not valid, then it is rejected and the interaction is aborted. If valid, A_1 needs to decrypt all the serial number and tags with $\bar{sk}^{A_1}$. The issuer sees the serial number only when the token is submitted back and detects a duplicate when two tokens with the same serial number have been submitted. Towards this end, the token issuer must maintain a database of all tokens submitted indexed by their serial number. When A_j submits a token, it checks to see if the token already exists in its database. If it is a new token, the complete token information along with the serial number is stored in the database. If the token already exists then the two tokens are retrieved and sent to the CA to identify the double spender based of the public key that satisfies either equations 1 or 2.

Different parts of this token scheme contribute towards satisfying our requirements. The certificates and the keys ensure unforgeability and verifiability, privacy is supported by randomizing the token at each stage and the serial number and tags are used to make the token single use. The security evaluation of our scheme is provided in the next section.

6 Security Evaluation

6.1 Algorithms and protocols

We formalize and summarize the algorithms described in the previous sections as follows:

- $\mathsf{ParamGen}(1^k)$ is probabilistic algorithm that outputs the system parameters par^{TS}.
- $\mathsf{KeyGen}(par^{TS})$ is a probabilistic algorithm that outputs the key pair of user, A_i: (sk^{A_i},pk^{A_i}). This pair represents all the keys that are generated.
- $\mathsf{IssueToken}(A_1(sk^{A_1}, pk^{A_2}), A_2(sk^{A_2}, pk^{A_1}))$ is an interactive protocol where A_1 issues a token to A_2. After this protocol ends, A_1 gets either its view $\mathcal{V}_{A_1}^{\mathsf{issue}}$ or $\perp$, and A_2 gets either a token $\mathsf{Token}_0^{A_1}$ or $\perp$.
- $\mathsf{TransferToken}(A_i(sk^{A_i}, pk^{A_1}, pk^{A_{i+1}}, \mathsf{Token}_{i-2}^{A_1}), A_{i+1}(sk^{A_{i+1}}, pk^{A_1}, pk^{A_i}))$ is an interactive protocol between A_i and A_{i+1}. pk^{A_1} is the public key of the issuer of $\mathsf{Token}_{i-2}^{A_1}$'. At the end, A_i has its view $\mathcal{V}_{A_i}^{\mathsf{transfer}}$ or $\perp$, and A_{i+1} has either a token $\mathsf{Token}_{i-1}^{A_1}$ or $\perp$.
- $\mathsf{SubmitToken}(A_{k+1}(sk^{A_{k+1}}, pk^{A_1}, \mathsf{Token}_{k-1}^{A_1}), A_1(sk^{A_1}, pk^{A_{k+1}}, D^{A_1}))$ is an interactive protocol between A_{k+1} and A_1. A_1 will accept $\mathsf{Token}_{k-1}^{A_1}$ if it was correctly issued by A_1 and has never been submitted before. D^{A_1} represents A_1's token database. At the end of this protocol, A_{k+1} gets either its view $\mathcal{V}_{A_{k+1}}^{submit}$ or $\perp$, and A_1 gets either an updated list D'^{A_1}, or two tokens $\mathsf{Token}_{k+1}^{A_1}$ and $\mathsf{Token}_{l}^{A_1}$ which have the same serial number, or $\perp$.
- $\mathsf{Identify}(\mathsf{Token}_{l}^{A_1}, \mathsf{Token}_{l'}^{A_1})$ is a deterministic algorithm. If both $\mathsf{Token}_{l}^{A_1}$ and $\mathsf{Token}_{l'}^{A_1}$ come from the same $\mathsf{Token}_0^{A_1}$, it outputs the public key of the token double spender. Otherwise it returns $\perp$.
- $\mathsf{VerifyGuilt}(pk^{A_i}, \Pi)$ is a deterministic algorithm which outputs 0 if Π is a correct proof that the owner of pk^{A_i} double spent the token, or 1 otherwise.

6.2 Correctness

We say a token submit is correct if an honest issuer gets an updated database as part of running protocol SubmitToken with the token submitter, only when the submitter submits a valid token. We say that a token issue and token transfer are correct if a honest user gets a valid token by running IssueToken or TransferToken protocol respectively, such that the token can be submitted or transferred and the submitter on running SubmitToken with the issuer, will never have the issuer outputting $\perp$.

$$
\begin{aligned}
Pr[\{\mathcal{V}_{A_1}^{submit}, D'^{A_1}\} \xleftarrow{\$} \mathsf{SubmitToken}(A_{k+1}(sk^{A_{k+1}}, pk^{A_1}, \mathsf{Token}_{k-1}^{A_1}), A_1(sk^{A_1}, pk^{A_{k+1}}, D^{A_1})) : \\
par^{TS} \xleftarrow{\$} \mathsf{ParamGen}(1^k); \\
(sk^{A_1}, pk^{A_1}), ..., (sk^{A_{k+1}}, pk^{A_{k+1}}) \xleftarrow{\$} \mathsf{KeyGen}(par^{TS}); \\
\{\mathcal{V}_{A_1}^{\mathsf{issue}}, \mathsf{Token}_0^{A_1}\} \xleftarrow{\$} \mathsf{IssueToken}(A_1(sk^{A_1}, pk^{A_2}), A_2(sk^{A_2}, pk^{A_1})); \\
\{\{\mathcal{V}_{A_i}^{\mathsf{transfer}}, \mathsf{Token}_{i-1}^{A_1}\} \xleftarrow{\$} \mathsf{TransferToken}(A_i(sk^{A_i}, pk^{A_1}, pk^{A_{i+1}}, \mathsf{Token}_{i-2}^{A_1}), A_{i+1}(sk^{A_{i+1}}, pk^{A_1}, pk^{A_i}))\}_{i=2,\cdots,k}] = 1,
\end{aligned}
$$

6.3 Security and anonymity

This section shows the security and anonymity model that any token transfer scheme needs to satisfy. It then provides the security proofs of our token transfer scheme under this model.

6.3.1 Definition of oracles

We follow a similar approach as [12]. Suppose that the parameter par^{TS} is given to the oracles. All the users' public keys and secret keys are initially created and managed by the oracles in databases PK and SK. They also manage the set of views of tokens. There are three tables IT, OT and ST. The tokens issued from the oracles are stored in IT, those issued to, or transferred from or to the oracles in OT, and those submitted to the oracles in ST. To evaluate the security of our scheme we use the following oracles:

- $\mathsf{OCreateUser}(i)$ executes $\mathsf{KeyGen}(par^{TS})$ and stores the output public key pk^{A_i} in $\mathsf{PK}[i]$ and the secret key sk^{A_i} in $\mathsf{SK}[i]$.
- $\mathsf{OCorrupt}(i)$ outputs sk^{A_i} and sets $\mathsf{SK}[i] = \perp$. When an adversary executes this oracle he gets all of A_i's tokens. After this protocol is run, the adversary can act as A_i as well as any of the other users that he has corrupted.
- $\mathsf{OIssuel}(pk^{A_1}, pk^{A_2})$ runs IssueToken protocol playing the token issuer. The adversary should have the secret key sk^{A_2} to execute this oracle. The oracle stores $\mathcal{V}_{A_1}^{\mathsf{issue}}$ in $\mathsf{IT}[1]$.
- $\mathsf{OIssueU}(pk^{A_1}, pk^{A_2})$ runs IssueToken playing the token receiver, A_2's side. The adversary should have sk^{A_1} to execute this oracle. The oracle stores the resulting token in $\mathsf{OT}[2]$
- $\mathsf{OIssuel\&U}(pk^{A_1}, pk^{A_2})$ runs IssueToken protocol playing both the token issuer and receiver. If the result of the protocol is $\mathcal{V}_{A_1}^{\mathsf{issue}}$ and $\mathsf{Token}_0^{A_1}$, they are stored in $\mathsf{IT}[1]$ and $\mathsf{OT}[2]$, respectively. The adversary should have neither sk^{A_1} nor sk^{A_2}.
- $\mathsf{OTransferS}(pk^{A_i}, \mathsf{Token}_{i-2}^{A_1}, pk^{A_{i+1}})$ runs TransferToken protocol playing the user who is transferring the token. The adversary should have secret key $sk^{A_{i+1}}$ to execute this oracle. If $\mathsf{OT}[i]$ does not have the token, the protocol is aborted. If the protocol is successful then $\mathsf{Token}_{i-2}^{A_1}$ is removed from $\mathsf{OT}[i]$ and sent to the adversary. $\mathsf{OT}[i]$ is updated with the view $\mathcal{V}_{A_i}^{transfer}$.
- $\mathsf{OTransferR}(pk^{A_i}, \mathsf{Token}_{i-2}^{A_1}, pk^{A_{i+1}})$ runs TransferToken protocol playing the token receiver, A_{i+1}'s side. The adversary should have sk^{A_i} and $\mathsf{Token}_{i-2}^{A_1}$ before executing this oracle. If the protocol completes successfully, the resulting $\mathsf{Token}_{i-1}^{A_1}$ is stored in $\mathsf{OT}[i+1]$.
- $\mathsf{OTransferS\&R}(pk^{A_i}, \mathsf{Token}_{i-2}^{A_1}, pk^{A_{i+1}})$ runs TransferToken protocol playing both sides. If $\mathsf{OT}[i]$ does not have the token, the protocol is aborted. Otherwise, after running the protocol, $\mathsf{Token}_{i-2}^{A_1}$ is removed from $\mathsf{OT}[i]$ and sent to A_{i+1}. $\mathsf{Token}_{i-1}^{A_1}$ is now stored in $\mathsf{OT}[i+1]$. A_i's output, $\mathcal{V}_{A_i}^{transfer}$ is now stored in $\mathsf{OT}[i]$.
- $\mathsf{OSubmitS}(pk^{A_{k+1}}, \mathsf{Token}_{k-1}^{A_1}, pk^{A_1})$ runs SubmitToken protocol playing A_{k+1}. The adversary should have sk^{A_1} to execute this oracle. If the protocol is not aborted $\mathsf{OT}[k+1]$ is updated with A_{k+1}'s view of the protocol, $\mathcal{V}_{A_{k+1}}^{submit}$. If SubmitToken outputs $\mathsf{Token}_l^{A_1}, \mathsf{Token}_{k-1}^{A_1}$, it runs $\mathsf{Identify}(\mathsf{Token}_l^{A_1}, \mathsf{Token}_{k-1}^{A_1},)$ and outputs the resulting public key.
- $\mathsf{OSubmitR}(pk^{A_{k+1}}, \mathsf{Token}_{k-1}^{A_1}, pk^{A_1})$ runs SubmitToken protocol playing the issuer's side. The adversary should have both $\mathsf{Token}_{k-1}^{A_1}$ and $sk^{A_{k+1}}$ to run this oracle. sk^{A_1} should not belong to the adversary. It updates ST if the protocol completes successfully. If SubmitToken outputs $\mathsf{Token}_l^{A_1}, \mathsf{Token}_{k-1}^{A_1}$, it runs $\mathsf{Identify}(\mathsf{Token}_l^{A_1}, \mathsf{Token}_{k-1}^{A_1},)$ and outputs the resulting public key.

6.3.2 Unforgeability

As in [12], the unforgeability requirement reduces to the fact that any set of users should not be able to spend more tokens than those issued or transferred to them.

Game. Suppose an adversary Adv is a probabilistic polynomial-time Turing Machine that has access to all of the user's public keys in PK and $par^{TS} \leftarrow \mathsf{ParamGen}(1^k)$. Adv can play with the oracles OCreateUser, OCurrupt, OIssuel, OIssuel&U, OTransferS, OTransferR, OTransferS&R and OSubmitR, as many times as he wants. Adv wins the game if $q_I + q_R < q_S$ where q_I is the number of successful queries to the oracle OIssuel, q_R is the number of successful queries to the oracle OTransferS, and q_S is the number of successful queries to the oracle OTransferR.

Theorem 6.1 *The proposed scheme is unforgeable*

Proof: (Sketch) Suppose the adversary, Adv succeeds in forging a token in the unforgeability game. This means Adv produces at least one new token that is acceptable by the oracle OTransferR. Based on the number of transfers that the token has undergone, we can divide this into three cases. If the new token is a directly issued token, then the entire token is essentially a delegatable anonymous credential. The existence of the new token means breaking F-unforgeability[7], which is a contradiction based on the computational assumption in [7]. If the new token has undergone a single transfer then it consists of the delgatable anonymous credential, a serial number, and a tag. The existence of the new token then breaks F-unforgeability, or violates the weak BB assumption [9]. Based on the assumptions in [7], this is infeasible. The final case is where the new token has undergone more than one transfer. In this case, the new token is a GS-NIZK proof. Because of the extractability of the GS-NIZK proof, we can extract the witness of the proof. Thus, like the second case, we can show that this means breaking the F-unforgeability or violating the weak BB assumption. Therefore, the proposed scheme is unforgeable. ∎

6.3.3 Anonymity

For the token scheme to be privacy preserving, in our setting, we need it to have strong anonymity guarantees. In this section we define the exact anonymity requirements, and call it *interaction anonymity*. We define the *interaction anonymity* game analogous to the one in [12]. In [12], the adversary, Adv runs the e-cash credential transfer protocol (spending protocol) with a challenged user i_b, where b could be either 0 or 1, and has to determine b. In our case, since the identity of a user is known in a direct interaction, Adv can easily win the same game. We, therefore, modify the game such that the challenged user i_b runs the token transfer protocol with an intermediate user A_j first. A_j, then, transfers it to Adv, who tries to determine b. This game captures the concept of *interaction anonymity* where the concern is the privacy of previous interactions. We have previously used A_* to define all our users. We use i_0 and i_1 to maintain a similar notation as [12], enabling us to highlight the difference between the two anonymity games. i_0 and i_1 could represent any two users. We define the anonymity game more precisely as follows:

Game$^{\mathsf{anonymity}}(1^k)$

1 $par^{TS} \xleftarrow{\$} \mathsf{ParamGen}(1^k)$

2 SK, PK, IT, OT, and ST are created.

3 $pk^{i_0}, pk^{i_1}, pk^{A_j}, \mathsf{Token}^{A_1}, \mathsf{Token}'^{A_1} \xleftarrow{\$} Adv^{\mathsf{SetofOracles}}$, where $\mathsf{SK}[1] \neq \perp$, $\mathsf{SK}[j] \neq \perp$, $\mathsf{SK}[i_0] \neq \perp$, $\mathsf{SK}[i_1] \neq \perp$. Token^{A_1}, Token'^{A_1} have the same length.

4 Suppose Token^{A_1} belongs to A_m, and Token'^{A_1} belongs to A_n, where both users could be corrupted by Adv. $\mathsf{OTransferR}(pk^{A_m}, pk^{i_0}, \mathsf{Token}^{A_1})$ and $\mathsf{OTransferR}(pk^{A_n}, pk^{i_1}, \mathsf{Token}'^{A_1})$ are executed.

5 $b \xleftarrow{\$} \{0,1\}$ and $\mathsf{OTransferS\&R}(pk^{i_b}, pk^{A_j})$ is executed.

6 $\mathsf{OTransferS}(pk^{A_j}, pk^{A^{Adv}})$ is executed, where A^{Adv} can be any user who is corrupted by Adv.

7 $b' \leftarrow Adv^{\mathsf{SetofOracles}'}$

8 If $b = b'$ return 1. Else, return 0.

(*) Adv cannot use OSubmitR more than once for each token Token^{A_1} and Token'^{A_1} through the whole experiment, even when they are transferred to other users controlled by oracles.

(*) SetofOracles means Adv can play with all the oracles.

(*) SetofOracles$'$ means Adv can play with all oracles except $\mathsf{OTransferS}(pk^{i_0}, \mathsf{Token}^{A_1}, \cdot)$, $\mathsf{OTransferS}(pk^{i_1}, \mathsf{Token}'^{A_1}\cdot)$, $\mathsf{OSubmitS}(pk^{i_0}, \mathsf{Token}^{A_1}, A_1)$, and $\mathsf{OSubmitS}(pk^{i_1}, \mathsf{Token}'^{A_1}, A_1)$ are not allowed.

In the above game, the following inequality should hold for a scheme if it has to meet *interaction anonymity*:

$$|Pr[\mathbf{Game}^{\mathsf{anonymity}}(1^k) = 1] - Pr[\mathbf{Game}^{\mathsf{anonymity}}(1^k) = 0]| < \frac{1}{p(k)}$$

Theorem 6.2 *The proposed scheme preserves interaction anonymity*

Proof: (Sketch) From the anonymity experiment, the token has undergone at least 3 transfers after which the adversary, Adv needs to determine whether i_0 or i_1 owned it previously. This means that the token is composed of GS-Proofs, a serial number, and tags. The serial number and tags are encrypted with the issuer's public key. The harder case is if the adversary has seen the token before, by corrupting users A_m and A_n. Since the GS-proofs are randomized [7], and the serial number and tags are re-encrypted with a new random number at every transfer, both of which ensure unlinkability, Adv cannot link the token that he obtains to any of the tokens that he previously owned. More precisely speaking, the randomizability of GS-proofs[22] shows that the randomized GS-proof cannot be distinguishable from a simulated GS-proof that is generated based on simulated parameters even though the adversary knows the trapdoor information of the proof. This means the GS-proofs that were part of the token owned by the Adv are unlinkable to the GS-proof in the token that he obtains at the end of the experiment. As far as the serial number and tags are concerned, any of the re-encrypted Elgamal ciphertexts are indistinguishable from the two random element tuple (g^{r_1}, g^{r_2}), where $r_1, r_2 \xleftarrow{\$} \mathbb{Z}_q$ based on DDH assumption. Therefore, the serial number and tags previously seen by the adversary are unlinkable to the ones that are part of the token that he obtains at the end of the experiment. The proposed scheme therefore preserves *interaction anonymity*. ∎

6.3.4 Identification of double spender

No user can double spend or transfer a token twice without revealing his identity. We define this requirement through the following game:

Game. Let an adversary Adv be a polynomial probabilistic Turing Machine that has access to all of the users' public keys in PK and par^{TS}. Adv can play any number of times with all of the oracles. Then Adv

chooses a challenge token $Token$ that belongs to one of the users that he has corrupted, A_i. After that, Adv uses $Token$ twice using either OTransferR or OSubmitR. Adv can again play with all of the oracles any number of times. Adv wins the game if, on running OSubmitR, the Submit protocol outputs $Token', Token''$, where both tokens come from $Token$, and the output of Idenitfy$(Token', Token'')$ is not a public key whose secret key is $\perp$ in SK.

Theorem 6.3 *The proposed scheme identifies double spenders*

Proof: (Sketch) We divide the double spending into two cases. The first case is where a user A_i transfers his token to two different users, A_m and A_n. A_m and A_n use their public keys to make the first set of tags ($\hat{T}_{l1}$, $\hat{T}_{l2}$). Therefore, these two tags are different ensuring that when the issuer receives both these tokens, no matter how many transfers the tokens have undergone, the double spending will be detected as shown in equation 1. The only way the tags are not different is if A_m and A_n use the same public key, which is not possible, as for them to be regarded as different entities, their (registered) public keys should be different from each other. If they are the same entity the situation is considered in the next case.

The second case is if A_i transfers the same token twice, as the receiving user does not have access to the serial number of the token he receives. In this case, the second set of tags ($\check{T}_{l1}$, $\check{T}_{l2}$) for the two copies of the double spent token are different as the receiving user will use a different random number for each interaction. When these tokens are submitted to the issuer, ($\check{T}_{l1}$, $\check{T}_{l2}$) will reveal the double spender, A_i as shown in equation 2. A_i can transfer the token to another user corrupted by Adv, A_j who uses the same random number for both interactions. A_j will then transfer these tokens to some other set of users. However when these tokens are submitted to the issuer, A_j will instead be caught as the double spender. ▌

7 Implementation and Evaluation

Our scheme is built on a large number of relatively new cryptographic primitives (DAC - Crypto 2009, GS Proof System - Crypto 2008) and we are aware of no implementations that exist for them. We, therefore, first built these primitives, in C, on top of the PBC library[27] which performs pairing based mathematical operations. We use type D curves with group order of 159-bit length. We then implemented the different algorithms of our token scheme on top of these primitives. The complete token framework was implemented in ≈ 12000 lines of code. We rewrote our existing VoIP simulator code to use Google protocol buffers to pass messages between entities. The protocol buffers offer fast serialization and deserialization, with minimal header overheads, and allow changes to the message structure without breaking older code. This required an additional ≈ 2500 lines of code. Since the token framework implements a wide variety of cryptographic primitives we first study the performance of the token framework with respect to the time taken by operations and the message lengths (network bandwidth) generated by the interactive protocols.

7.1 Operation Costs

7.1.1 Startup Costs

We initially need to generate the common parameters using the PBC library[27], as described in the algorithm ParamGen in section 5.1. These include the bilinear groups and all the group generators. Since we use the SXDH assumption [9] we create an asymmetric bilinear pairing, that is $G1 \neq G2$. These common parameters need to be shared across all clients. In our system, without loss of generality, we let the CA generate these parameters. The operation costs that follow were measured on an Intel Xeon 5160 with a 3 GHz processor. Each

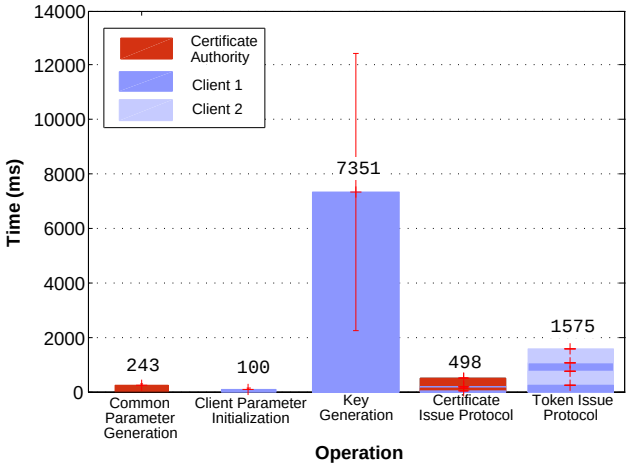

Figure 4: Time - Operation Preliminaries

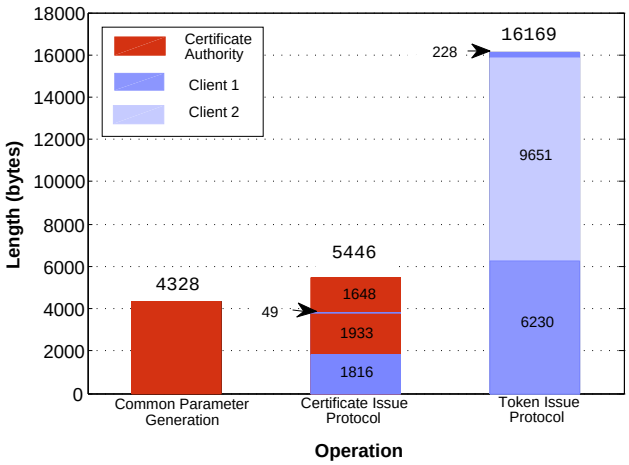

Figure 5: Length - Operation Preliminaries

operation was run 10 times and the mean and standard deviation of the time taken and the message lengths generated by these operations are as shown in figure 4 and 5. Many of the operations include two entities (example, Client and CA) and the length captures the network bandwidth utilized during this operation. The time in such interactive operations does not include the network latency. The parameter generation measurements are as shown in the block marked *Common Parameter Generation*. The operation takes around $240\,ms$ and the parameters themselves can be encapsulated in a message of length $\approx 4\,KB$. All clients on startup contact the CA to obtain the common parameters as a $4\,KB$ message and use it to initialize their global parameters structure. The client initialization is relatively quick and takes around $100\,ms$ and is shown as block marked *Client Parameter Initialization* in figure 4. The CA also needs to initialize its own global parameters structure.

Using the common parameters, the clients and the CA generate the secret keys and the public keys, (sk'^{A_i}, pk'^{A_i}) and ($\bar{sk}^{A_i}$, $\bar{pk}^{A_i}$) as described in algorithm KeyGen in section 5.1. Strictly speaking, the CA does not need to generate the secret key and the corresponding public keys that are used for ElGamal encryption, as it never issues a token. The key generation is a costly operation as seen in the block marked *Key Generation* and takes around $\approx 7\,s$ with a standard deviation of $\approx 5\,s$. The clients then need to certify their public keys pk'^{A_i}, from the CA as described in IssueCertificate. The client and the CA engage in a 2PC protocol for creating the NIZKPK proof of the authenticator. The details of the 2PC protocol are provided in [7]. The lengths of the messages exchanged between the client and the CA are shown in the block marked *Certificate Issue Protocol*. All the messages are under $2\,KB$, and the overall time of the operation notwithstanding network latencies is roughly $.5\,s$. All these operations are performed by the clients at startup time, following which clients can interact with each other to issue, transfer or submit tokens.

7.1.2 Cost of Token Operations

The token issue protocol is a three way exchange between the token issuer and the receiver. They carry out a secure 2PC of the issuer authenticating the receiver's secrets as described in IssueToken. The time taken and the message length generated are depicted by the block *Token Issue* in figures 4 and 5. A token issue takes on average $1.5\,s$ to complete, with little variance. Once a token is issued the receiving client can submit it back or transfer the token further. As described earlier, the submit or the transfer operations vary based on the number of times this token has already been transferred. The length of the randomized token(essentially all the proofs) is shown in figure 7. Chase et al[7] was the first to introduce DACs with proof size $O(Lk)$, where L is the number of hops from the token issuer and k is the security parameter. We rely heavily on the authentication scheme provided by DACs and we add the notion of single use tokens through the creation of serial number and

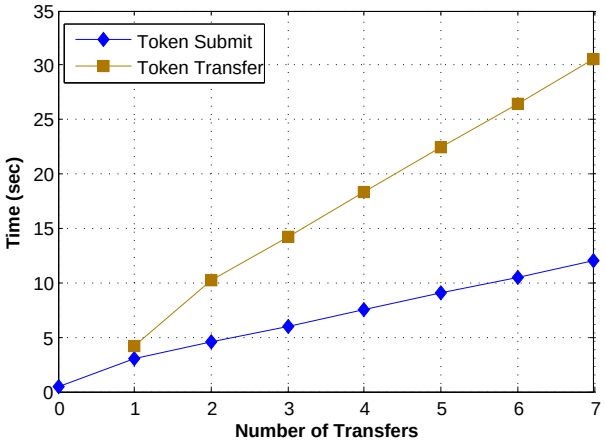

Figure 6: Time - Coin Transfer and Submit

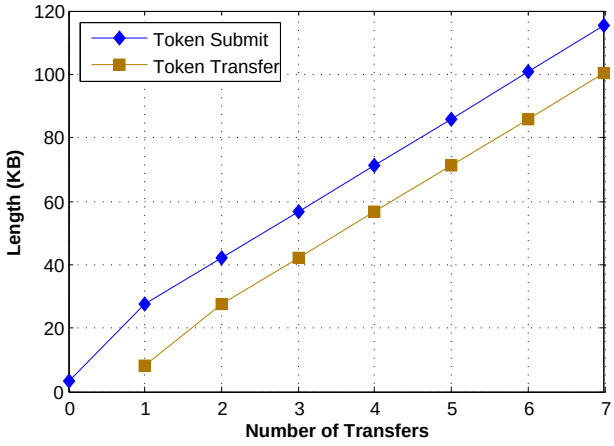

Figure 7: Length - Coin Transfer and Submit

tags. Our construction is also linear in the size of the proof and this can be seen in figure 7. The close similarity between token submit after L hops ($L \geq 2$) and token transfer after $L+1$ hops is that in order to submit a L hop transferred token, the owner needs to randomize the token in the exact fashion as a token transfer. For token submit, token lengths increases by $15\,KB$ each transfer. This is important as we plan to use this framework in a VoIP setting where call signaling and call teardown is performed over both UDP and TCP. In IPv4 for UDP, the maximum packet size is 65507, dictated by the maximum IP packet size of 65535 less 20 bytes for the IP header and 8 bytes for the UDP header. This can be overcome by splitting the token or using TCP or using IP jumbograms in the case of IPv6 networks. In this paper, we do not split the tokens and only allow tokens of size less than 65KB. We therefore restrict ourselves to only allowing 4 hop token transfers (3-transfer case) which have token lengths of size 58 KB.

As seen in figure 7 the length of *Token Submit 0* is $\approx 3\,KB$ and of *Token Submit 1* is $\approx 28\,KB$. *Token Submit 0* is when a token directly issued to a user is submitted back. Since we assume that the two entities engaged in an interaction know each other we simplify the *Token Submit 0* operation such that the contents are not randomized and the token is submitted back as is, except that the serial number and tags for the token are still calculated (the token can still be double spent). In all other cases the token needs to be randomized as this prevents the issuer from determining who was the penultimate owner of the token. In our example when a k hop token issued by A_1 is submitted through path $A_k \rightarrow A_{k+1} \rightarrow A_1$, randomization prevents A_1 from knowing $A_k \rightarrow A_{k+1}$. For *Token Submit 0* there is no previous owner to the submitter. In the token transfer case too there is an increase in token length from *Token Transfer 1* and *Token Transfer 2*. In this case from *Token Transfer 2*, the serial number and tags need to be encrypted and the proofs for their correctness need to be added. This causes the increase in token length.

Figure 6 depicts the most significant time activity for coin transfers and tokens over 10 runs (mean value, the variance is low). This includes verifying correctness of tokens and in the case of token transfers performing the final stage of the 2PC for creating NIZKPK of authenticator. The overall time taken for token transfer is significantly more due to the 2PC protocol. The time taken for both coin transfer and coin submit increases linearly with the number of hops. The token submit operations increase by $1.5\,s$ per hop and the token transfers increase by $4\,s$ per hop. These values will dictate feasibility of the token framework in a particular application setting. In the next section we apply the token framework to the VoIP setting and study its performance in preventing VoIP spam.

7.2 Applying The Token Framework To Prevent VoIP Spam

To evaluate our token framework we applied it to a VoIP setting to assess caller legitimacy based on social network interactions. VoIP is the umbrella term given to a set of protocols that allow the routing of voice calls over the internet (IP network). The signaling can be enabled through a variety of ways, though in this paper, we specifically consider the Session Initiation Protocol (SIP)[30]. For two users to communicate with each other using SIP, they need to know each other's SIP Uniform Resource Identifier (SIP URI). SIP uses a three-way handshake to establish a call and a two way handshake to teardown a call. Call duration as required by Callrank[5] and our system is the time between the end of call setup to the start of call teardown.

We piggyback our token mechanisms on top of the call signaling messages. Token submit just requires a single message from one user to the other. We therefore add token details to the call initiation message ($INVITE$) of call setup. Based on the validity of the submitted token, the call can be accepted ($200\ OK$) or rejected ($440\ REJECT$). E.721[33] recommends an average delay of no more than 3.0, 5.0 or $8.0\ s$, for local, toll and international calls, with the 95th percentiles set at 6.0, 8.0 and $11.0\ s$, respectively. Looking at the token submit times from figure 6 we see that other than for direct token submits (*Token Submit 0*), token submit times are greater than $3\ s$ and increase by $1.5\ s$ every hop. This implies that direct tokens offer acceptable call setup delays while tokens that have undergone one or two transfers will fall within the 95th percentile. Tokens that have undergone three transfers (four hops away) and beyond will have unacceptable call setup times. However, these calls will be infrequent as they introduce new users, who are four hops away and will be incurred only once for any such introduction. Taking this and the UDP packet size limit into account, we only allow up to three transfers (users four hops away) in this implementation. Removing these restrictions is part of future work, including the use of a different set of curves to obtain faster bilinear pairing operations.

Token issue and transfer can be piggybacked onto the call teardown message (BYE). Although token transfer takes significantly more time than the other operations, call teardown does not have strict time constraints. With the BYE message we can also calculate the duration of the call and embed it within the token as a public attribute[7]. Since the token does not reveal identities of the interacting parties, the call duration in itself does not leak privacy. SIP teardowns are two-way handshakes, a BYE message followed by a $200\ OK$. However, our token issue and transfer protocols are three-way, necessitating the need for a non piggybacked message. There can be multiple policies to decide, after a call, whether a caller issues a new token and or transfers a pre-owned one, to the call recipient. In this paper, each user maintains a max-heap of tokens, ordered on number of tokens. When user A_1 calls user A_2, he issues tokens if A_2 has lesser than a threshold number of A_1's tokens, or if A_1 does not have sufficient number of tokens of any other user to transfer. In all other cases, A_1 transfers the token of a user from whom it has collected the maximum number of tokens.

To evaluate the combined system, we setup a simulation with 4 domains, each serviced by a proxy that handles 50 users, a total of 200 users in the system. In addition, we have a DNS server, a cryptography server and a statistics server. The DNS server translates domain names to the correct proxy IP address. The cryptography server generates the common parameters and doubles up as the CA. The statistics server calculates statistics including true positives, true negatives, false positives and false negatives. Initially each client requests the cryptography server for the common parameters and uses them to generate keys. It gets the keys certified by the CA and then it is ready to make and receive calls. The distribution with which it makes calls is dependent on the type of user the client represents. Clients can behave either as an honest user or one of two types of spammers: (1) *engaging spammers* are able to engage users with a certain probability both when they receive calls and when they make calls, (2) *fleeting association spammers* are able to engage users only till the completion of some activity. Honest users makes calls to other phones with inter call and call duration values that are Poisson distributed. The choice of call recipient is Zipfian distributed. Spammers make calls to as many other users as possible. Honest users issue or transfer tokens based on a threshold call duration strategy. Spammers issue or transfer tokens to increase the number of spam calls. All spammers are inclined to collude with other

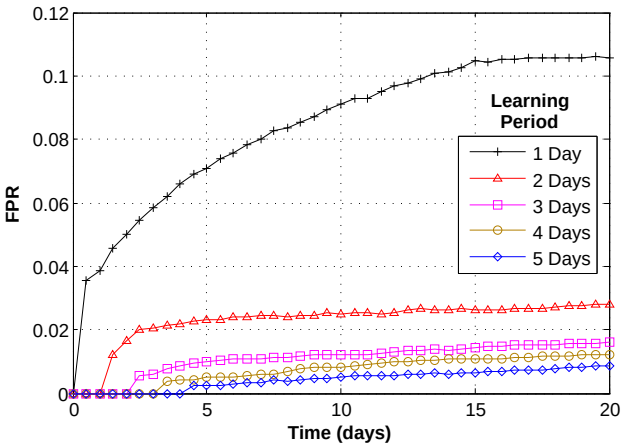

Figure 8: Learning Period - False Positive Rate

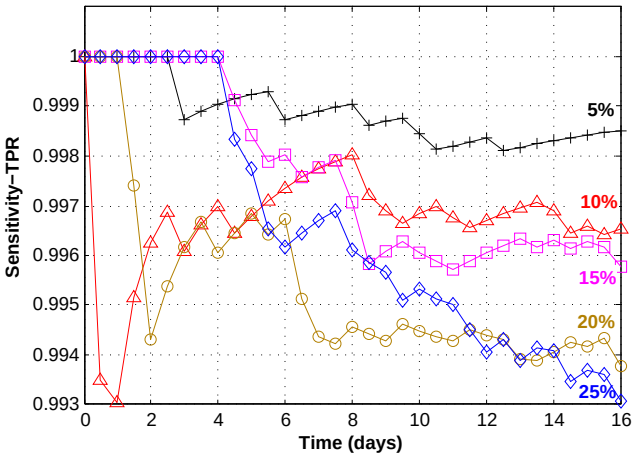

Figure 9: Engaging Spammers - True Positive Rate

spammers. In the simulation, 100 s of simulation time is equivalent to 1 day of real time. Each run lasts 20 days (2000 s).

7.2.1 Choice of Learning Period

The learning period is a duration of time just after a user is introduced into the system. During this time, the user accepts all calls to obtain a sizeable starting set of tokens from his SN. These tokens enable the user to call his SN and also disseminate his tokens so that others can call back. The learning period ensures that when a honest user is introduced into the system, he becomes selective about the calls he accepts only when he has a significant supply of tokens from members of his SN and his SN has a significant supply of his tokens. We assume that during this learning period spammers do not discover the user and therefore cannot spam the user. The graph in figure 8 shows the false positive rate (FPR) for 200 users, all honest, for different initial learning periods. The time axis starts 1 day into the running of the system as this is the minimum learning period that was used. The stabilized false positive rate shows an exponential drop with increasing learning periods. For learning periods of 1, 2 and 3 days it is $\approx 11\%$, $\approx 3\%$, $\approx 1.7\%$, respectively and thereafter stays around $\approx 1.5\%$ for higher learning periods. After learning periods of 3 days or more, users have a significant supply of tokens and can obtain tokens of users who are four hops away through the token transfer mechanism, resulting in a low false positive rate. Shue et al[31] studied the onset of spam and found that accounts that post their addresses on less popular websites will be discovered and receive spam only after 3 days. If we assume this holds for VoIP addresses too, then as long as users do not aggressively broadcast their addresses a learning period of 3 days is feasible and provides a low enough FPR. In addition, since learning periods of more than 3 days do not reduce the FPR significantly, we use a 3 day learning period for the rest of the simulations.

7.2.2 Spammers that Engage Users In Conversation

In our system, users issue or transfer tokens only when a call lasts for more than a threshold duration. Spammers without the ability to engage users will never get tokens, even if they do manage to get users to inadvertently call them. However spammers thrive because some honest users are fooled into believing the legitimacy of the spam content. To model this, we associate with all users a value between 0 and 1 that represents the ability to engage another user in conversation. This value is set high for honest users and we configure spammers with various levels of engagability. Based on the engagability of the spammer, a user will inadvertently either issue or transfer tokens to spammers. Spammers can collude and therefore can collaboratively glean tokens. Spammers are introduced into the system immediately after the learning period (3 days). The results for different values

of spammer engagability for a system with 20% spammers are shown in figure 9. For all cases our token framework is able to achieve a high sensitivity, of over 99%, in blocking spam calls. Spammers will find it hard to engage users in conversation and even when they do, the single use tokens only allows a limited number of calls. On the other hand, an honest user, due to his ability to carry on a conversation will first receive tokens of his immediate SN, and then receive tokens from his extended SN. From figure 9, small values of engagability (5%, 10% and 15%) result in a low false negative rate (FNR) and this rate stabilizes early in the run. However spammers with a higher ability to engage users (25%), are able to make more calls at an ever increasing FNR, largely due to the collusions with other spammers. For spammers with 25% enagagability the final stabilized FNR was close to 1% and for 35% the FNR was close to 1.3%.

7.2.3 Fleeting Association Spammers

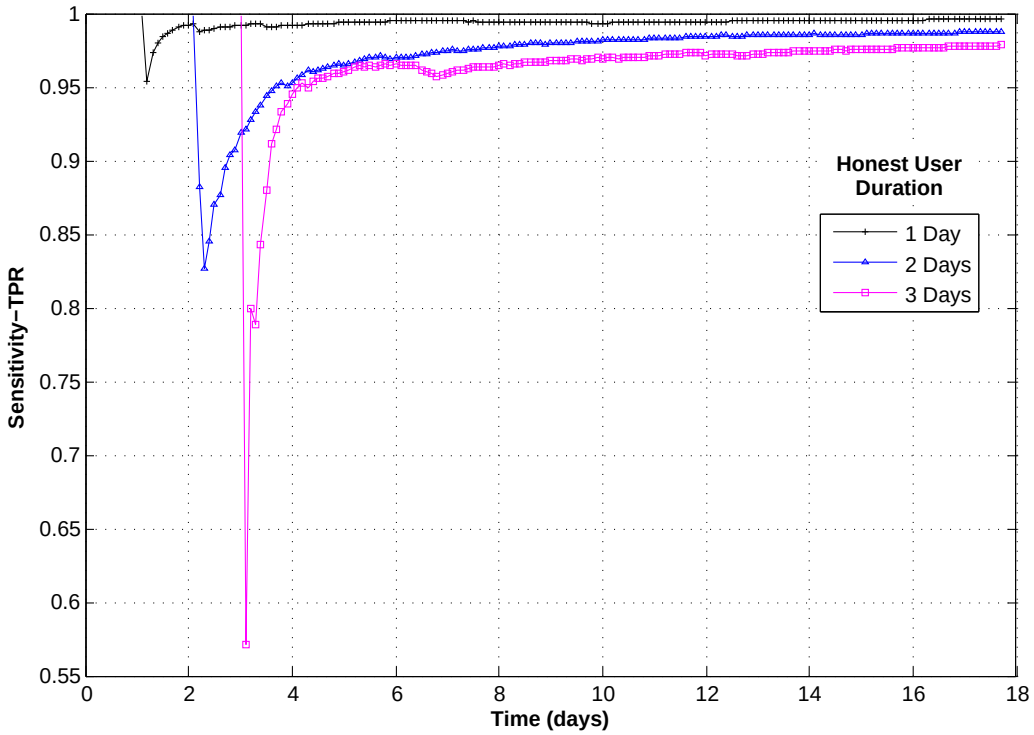

Figure 10: Fleeting Association Spammer - True Positive Rate

We also studied the effects of introducing 10% *fleeting association spammers* who behave dichotomously. When just introduced to a user, they behave legitimately but soon start spamming the user. We configure these users to behave normally for periods of 1, 2 and 3 days. Figure 10 shows the results. We find that the longer the spammer behaves normally the more effective they are in spamming. Spammers who behaved normally for 1, 2 and 3 days and then started spamming were able to achieve a FNR of $\approx 4.5\%$, $\approx 17\%$ and $\approx 43\%$, respectively. These values show that the success of a spammer increases significantly with the amount of time he is able to behave normally. These values are particularly alarming as spammers who behave normally for 3 days are able to amass a significant number of tokens and can use it to spam all the users who issued those tokens. However, the system realises this quickly and within a day reduces the FNR for each of the cases to under 10%. This reduction is not time based but token based. If a spammer starts making spam calls at a higher rate his success rate will also drop faster as he quickly run out of tokens (that no longer get replenished). The

stabilized FNRs for spammers who behaved normally for 1, 2 and 3 days is $\approx .3\%, 1\%, 2\%$. In addition, for the spammer to be successful again, he needs to engage each user that he wants to spam for for at least 2 days, after which his success rate is high only for the first day resulting in diminishing returns for the spammer.

The results clearly demonstrate that reasonable performance can be achieved for the token transfer scheme. In addition, the scheme is effective against the spammer threat model. The transfer mechanism of the token helps reduce the false positives for feasible values of the learning period. The single use feature protects against spammers that can engage users in conversation and hence obtain tokens, and users who behave normally and amass tokens and then start spamming. In both these cases the limited supply of tokens ensures that the spammers effectiveness rapidly decreases with time. The anonymity and unforgeability features ensure that users only engage with new users who have some weak social link without revealing what the link is.

8 Conclusion and Future Work

In this paper, we created a single use transferable token framework that captures interaction history in a privacy preserving manner by enhancing delegatable anonymous credentials. We implemented the framework and showed the feasibility of using it in a VoIP setting to prevent VoIP spam. Our future work includes extending the system to provide multiple tokens in a single interaction (compact token scheme), and ways on improving the efficiency of the system.

References

[1] Presentation on Q1 2009 Earning Report of Ebay Inc. http://www.slideshare.net/earningreport/presentation-on-q1-2009-earning-report-of-ebay-inc. Last accessed Sep. 18, 2009.

[2] K. G. Anagnostakis and M. B. Greenwald. Exchange-based incentive mechanisms for peer-to-peer file sharing. In *24th International Conference on Distributed Computing Systems (ICDCS)*, 2004.

[3] AOL. AOL Instant Messenger. http://dashboard.aim.com/aim. Last accessed. Sep 18, 2009.

[4] L. Baker. Yahoo to Support OpenID for its 248 Million Users, OpenID to Support Yahoo IDs. http://www.searchenginejournal.com/yahoo-to-support-openid-for-its-248-million-users-openid-to-support-yahoo-ids/6258/. Last accessed Sp. 18, 2009.

[5] V. Balasubramaniyan, M. Ahamad, and H. Park. Callrank: Combating spit using call duration, social networks and global reputation. In *CEAS 2007 - The Fourth Conference on Email and Anti-Spam, 2-3 August 2007, Mountain View, California, USA*, 2007.

[6] A.-L. Barabási. *Linked: The New Science of Networks*. Perseus Books Group, May 2002.

[7] M. Belenkiy, J. Camenisch, M. Chase, M. Kohlweiss, A. Lysyanskaya, and H. Shacham. Randomizable proofs and delegatable anonymous credentials. In *Crypto*, volume 5677 of *LNCS*, pages 108–25. Springer-Verlag, Aug. 2009.

[8] M. Belenkiy, M. Chase, C. C. Erway, J. Jannotti, A. Küpçü, A. Lysyanskaya, and E. Rachlin. Making p2p accountable without losing privacy. In *ACM workshop on Privacy in electronic society (WPES)*, pages 31–40, New York, NY, USA, 2007. ACM.

[9] D. Boneh, X. Boyen, and H. Shacham. Short group signatures. In *Crypto*, LNCS. Springer-Verlag, 2004.

[10] L. Bussard, Y. Roudier, and R. Molva. Untraceable secret credentials: Trust establishment with privacy. In *PERCOMW '04: Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops*, page 122, Washington, DC, USA, 2004. IEEE Computer Society.

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

[12] S. Canard and A. Gouget. Anonymity in transferable e-cash. In *Applied Cryptography and Network Security, 6th International Conference, ACNS 2008, New York, NY, USA, June 3-6, 2008. Proceedings*, pages 207–223, 2008.

[13] S. Canard, A. Gouget, and J. Traoré. Improvement of efficiency in (unconditional) anonymous transferable e-cash. pages 202–214, 2008.

[14] B. Carminati and E. Ferrari. Privacy-aware collaborative access control in web-based social networks. In *Proceeedings of the 22nd annual IFIP WG 11.3 working conference on Data and Applications Security*, pages 81–96, Berlin, Heidelberg, 2008. Springer-Verlag.

[15] D. Chaum, A. Fiat, and M. Naor. Untraceable electronic cash. In *CRYPTO '88: Proceedings on Advances in cryptology*, pages 319–327, New York, NY, USA, 1990. Springer-Verlag New York, Inc.

[16] D. Chaum and T. P. Pedersen. Transferred cash grows in size. In *EUROCRYPT*, pages 390–407, 1992.

[17] L. P. Cox and B. D. Noble. Samsara: honor among thieves in peer-to-peer storage. *SIGOPS Operating Systems Review*, 37(5):120–132, December 2003.

[18] Y. Dodis and A. Yampolskiy. A verifiable random function with short proofs and keys. In *Proceedings of the Workshop on Theory and Practice in Public Key Cryptography*, 2005.

[19] eBay. Skype. http://www.skype.com/.

[20] Google. Google talk. http://www.google.com/talk/.

[21] M. Granovetter. The strength of weak ties: A network theory revisited. *Sociological Theory*, 1:201–233, 1983.

[22] J. Groth and A. Sahai. Efficient non-interactive proof systems for bilinear groups. In *Advances in Cryptology EUROCRYPT 2008*, pages 415–432. 2008.

[23] D. Hausheer. *PeerMart: Secure Decentralized Pricing and Accounting for Peer-to-Peer Systems*. PhD thesis, ETH Zurich, Aachen, Germany, Mar. 2006.

[24] Y. Inc. Yahoo! messenger. http://messenger.yahoo.com/.

[25] J. Ioannidis, S. Ioannidis, A. D. Keromytis, and V. Prevelakis. Fileteller: Paying and getting paid for file storage. In *Sixth International Conference on Financial Cryptography*, pages 282–299, 2002.

[26] N. Liebau, V. Darlagiannis, A. Mauthe, and R. Steinmetz. *Token-Based Accounting for P2P-Systems*. 2005.

[27] B. Lynn. Pairing based cryptography library. http://crypto.stanford.edu/pbc/, 2006.

[28] A. Lysyanskaya, R. L. Rivest, and A. Sahai. Pseudonym systems. In *Proceedings of SAC 1999, volume 1758 of LNCS*, pages 184–199. Springer Verlag, 1999.

[29] N. Ramakrishnan, B. J. Keller, B. J. Mirza, A. Y. Grama, and G. Karypis. Privacy risks in recommender systems. *IEEE Internet Computing*, 5(6):54–62, November 2001.

[30] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session Initiation Protocol. RFC 3261 (Proposed Standard), June 2002. Updated by RFCs 3265, 3853, 4320, 4916, 5393.

[31] C. A. Shue, M. Gupta, J. J. Lubia, C. H. Kong, , and A. Yuksel. Spamology: A study of spam origins. In *Conference on Email and Anti Spam (CEAS)*, 2009.

[32] D. A. Turner. A lightweight currency paradigm for the p2p resource market. In *7th International Conference on Electronic Commerce Research*, 2003.

[33] I. T. Union. Network grade of service parameters and target values for circuit-switched services in the evolving isdn, 2004.

[34] K. Wei, A. J. Smith, Y.-F. R. Chen, and B. Vo. Whopay: A scalable and anonymous payment system for peer-to-peer environments. In *ICDCS '06: Proceedings of the 26th IEEE International Conference on Distributed Computing Systems*, page 13, Washington, DC, USA, 2006. IEEE Computer Society.

[35] B. Yang and H. Garcia-Molina. Ppay: micropayments for peer-to-peer systems. In *CCS '03: Proceedings of the 10th ACM conference on Computer and communications security*, pages 300–310, New York, NY, USA, 2003. ACM.