

TIMED ZERO-KNOWLEDGE PROOF (TZKP) PROTOCOL

T. C. Lam, Cheng-Chung Tan, Yu-Jen Chang, and Jyh-Charn Liu¹

Department of Computer Science

Texas A&M University

brianlam@tamu.edu, jonastan@cs.tamu.edu, stevenchang@tamu.edu, liu@cs.tamu.edu

Technical Report 2006-9-1

ABSTRACT

Incorporating time to the security management system is an important step to streamline its integration with other resource management functions in the real-time distributed systems. In addition to protecting its security properties, minimizing operational overheads of security management is also a major design issue. In this paper, we propose a timed zero knowledge proof (TZKP) protocol to support the session based secure access control of timed resources for anonymous consumers.

On the basis of Eng-Okamoto's general disposable authentication (GDA) model, we show that the service provider (SP) can authorize the timed resource access by adding session to a control variable in GDA. The consumer who receives timed access authorizations (TAAs) from the SP may choose to transfer them to others. In classical ZKP protocols, each transfer instance requires spending of one consumer token, which makes this desirable feature costly. To minimize the transfer overhead, we propose the multi-source reusability (MSR) condition that allows a consumer to reuse its token for multiple transactions with protected anonymity when no double spending of TAA occurs.

TZKP not only reduces the amount of tokens that need to be withdrawn by consumers, but also eliminates the need to keep track of the tokens once their marked sessions are expired. On the basis of the proven security properties of GDA in each session, together with the introduction of session time and the reduced cost transfers, we show that TZKP protocol can be efficiently used for secure access of the shared computing resources for anonymous consumers. Experiments run on DETER testbed show that TZKP has small to moderate run time, and can be easily expanded for large scale applications.

Key words: Time, authenticity, anonymity, zero knowledge proof, quota control, disposable authentication.

1. INTRODUCTION

Authenticity, anonymity, and accountability are three basic security management functions. While some authentication schemes use time/clock as one control parameter, *e.g.*, time

synchronous authentication [1], many security management paradigms do not consider time explicitly. Granted that the integrity of logical clock can be guaranteed by *digital time-stamping* [2], it only helps improving the authenticity for accessing timed resource, *e.g.*, bandwidth, processor cycles, but helps little with the anonymity of accessing principals. Knowing that the *zero knowledge proof* (ZKP) protocol is commonly used for anonymous authentication, it is useful to include time management to ZKP, so that it can support secure access control of timed resources for the anonymous consumers.

Conceptually, ZKP consists of a set of crypto primitives for a *prover* to prove to a *verifier* the possession of a *token withdrawn* from the *central authority* (CA), without letting know the private part of the token. The private and public parts of the token are both generated by the CA with the prover, and are unforgeable. For anonymous authentication, the private part of a token contains the *identity* of the token owner. After receiving the public part of the prover's token, the verifier requires the prover to produce a *response*, using the private part of the prover's token and the *challenge* that the verifier randomly generated. The verifier can validate the response by using publicly known information if a "yes" answer is produced from the verifier function. A token is said to be *spent* after the verifier validates the response. As such, the prover can stay anonymous to the verifier and the CA for this transaction, *i.e.*, "zero knowledge" of prover's identity. Traditionally, *double spending of token* is said to occur if a token is spent more than once. The identity of the prover will be involuntarily deciphered from the collections of messages produced from the *spent tokens*, or called the *credentials*.

In this paper we propose a *timed zero knowledge proof* (TZKP) protocol aiming for session-based access control of shared computing resources, such as bandwidth, processor cycles, *etc.*, in an open environment, where the *consumers* may be anonymous to the publicly known *service provider* (SP). By a simple manipulation of a control variable in the *general disposable authentication* (GDA) model² of Eng-Okamoto [3], *sessions*, or *logical clocks*, can be embedded into its cryptographic constructs, so that the resource access authorizations issued for distinct sessions can be managed

¹ Correspondence author

independently. Spending records and access authorizations that have passed the current session can be safely discarded, knowing that valid authorizations can only be produced by the SP. In contrast, in traditional designs, spending records need to be maintained indefinitely to catch consumers from using aged tokens to redeem services.

In TZKP the session time and some public information of the first consumer’s token are signed by the SP for the first consumer as a *timed access authorization* (TAA) of the resources at the scheduled time. The consumer needs to use TAA and other token information to redeem the service at scheduled time. A TAA becomes void after it passes the scheduled time slot. Another important issue is the freedom in transferring a TAA between consumers, using *cascaded credentials*. That is, in addition to the original TAA issued by the SP, a consumer must be able to use its own token to receive and pass on its TAA to another consumer. Different token data are used for receiving and passing the TAA in the transfer operation.

Cascading of credentials is a must for transferability of TAA [4], and thus, there is no room to reduce their storage. However, the withdrawal cost for transferring a TAA can be drastically reduced by using the *multi-source reusability* (MSR) condition proposed in this paper. Through MSR, the consumer can anonymously transfer multiple TAAs without withdrawing a new token, provided that these transfers do not constitute *double transferring/spending of TAA*. Double spending of TAA (also known as *quota violation*) is said to occur if the TAAs passed on are more than those received.

In summary, TZKP preserves the security properties of GDA within a session, at significantly reduced withdrawal costs and storage costs. It guarantees *transaction integrity*, *token authenticity*, *consumer anonymity* and *accountability for double spending of TAAs*, session by session. Adversary cannot breach identity privacy of token owner (consumer) by using the credentials across different sessions.

We demonstrate its applicability to secure computing resource management on DETER testbed [5]. Experiment results show that running time for withdrawal and service redemption are within range of seconds, making it highly practical for securing access control of large scale Internet resources.

The rest of this paper is organized as follows. We will introduce the secure resource management architecture of TZKP in Section 2. Then, we will explain the protocols for session-based management and MSR condition of TZKP in Section 3. Security and complexity analysis of the protocols will be given in Sections 4 and 5. Experimental results will be given in Section 6. Related work and possible extensions will be discussed in Sections 7 and 8. Finally, we conclude in Section 9.

² The GDA model in this paper refers to the general construct of DA in the second part of [3], but not the specific construct in first part of [3] or the specific construct in [10].

2. RESOURCE MANAGEMENT ARCHITECTURE

TZKP protocol aims at supporting anonymous, accountable, and secure access of shared computing resources. Following a generic system architecture for shared resource access, see Fig. 1, there are three types of principals in the system: the CA responsible for issuance of tokens to consumers, the SP responsible for issuance of TAAs to consumers and render services to consumers at their authorized session time. The consumers must first withdraw tokens from the CA in order to request TAAs from the SP, transfer TAAs, and redeem services from the SP. The consumer needs to use its token together with TAA in both transfer and service redemption.

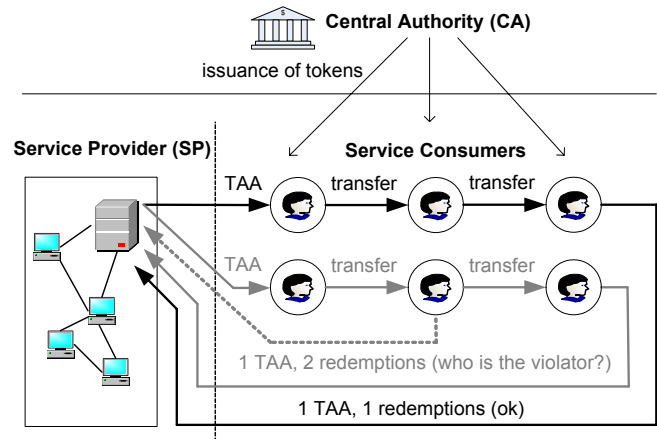


Fig. 1: Service Model and Adversary Behaviors.

A consumer initiates a service request by presenting the public part of its token, and the requested session time to access resources. The SP grants the request by generating a TAA, which consists of the signature of the bundled session time and the public data of the consumer’s token. Then, the consumer can use the TAA to redeem the services directly, or transfers it to another consumer, *i.e.*, transfer of a TAA from the *grantor* to the *grantee*. Transferability of TAA is highly desirable because it allows creation of a hierarchical distribution architecture of the resource access privileges, which is consistent with the current practice in large scale experimental facilities, such as DETER.

To access DETER, a faculty (the *group root* of DETER) first requests facility access, and then the students (the *local roots* or the *users* of DETER) can request the facility access through the faculty. In DETER, a user has the privileges to log into machines to do experiments. A local root has the privilege to create or destroy experiments in its project and the root privileges on machines that run the experiments. A group root has the privileges of local root and the privileges to approve new group members and modify information for users within the group.

By using TZKP, a TAA becomes void once it passes the scheduled time. The SP can safely discard spending records that have passed the current session time without affecting

its abilities to detect double spending of TAA for any future sessions. Except for very minor operational differences, the same authentication process is applicable to both the service redemption and the TAA transfer. As such, unless explicit clarification is necessary, we only discuss TAA transfer in remaining discussions.

We assume that the SP has a free-run system clock, and the time period each session can range between minutes to hours for the shared resources, because the computers often need to be reconfigured for various consumers. A consumer must use a valid token to transfer a TAA to another. Or, it can redeem services from the SP by using its token together with a TAA. The identity of a consumer will be deciphered when double spending/transferring of TAA occurs.

TZKP is designed for consumers to manage their access credits securely, while staying anonymous during operation. From the resource management viewpoints, it is easy for the SP to add TZKP to existing resource sharing rules, such as first come first serve (FCFS), because it imposes virtually no restriction on how the resources are reserved. The SP simply stops issuing TAA when the reserved resources in a session time reach a target level. And redemption of the services will be made to any principal who presents a valid token, together with a valid TAA. From the crypto analysis viewpoints, two main concerns need to be addressed. First, a consumer may attempt to doubly spend TAA in a session. Second, the SP may attempt to decipher consumer identities by collecting an infinite number of redeemed session tokens even though no double spending of TAA in any session. We will show later that neither of the two offenses can occur to TZKP, and security properties of GDA are preserved within each session.

3. TIMED ZKP (TZKP)

TZKP consists of two main crypto modifications to the well known GDA model. The first modification allows the SP to issue a TAA based on a signed session time bundled with some public information of a requester's token. The second modification allows a consumer to transfer numerous TAAs it received to other consumers by using the same token it possesses and still guarantee its anonymity.

3.1 General Disposable Authentication (GDA) Model [3]

The GDA framework proposed by Eng-Okamoto [3] is a versatile security control model which is compatible with many ZKP protocols designed for *electronic cash* [6]-[10]. The essence of GDA is summarized in Table 1, following the conventions defined in [3].

Table 1. Facts of General Disposable Authentication (GDA).

Goals (given m)	Data Required	Remarks
verify $m = f(x)$	$X, (E, Y)$	$X = F(x, r) = F'(m, r)$, $Y = D(m, r, x, E), m = f(x)$ iff $G(m, X, E, Y) = \text{"yes"}$

decipher x	R1: $(E, Y), r$	$r = \text{symmetric key to encrypt/decrypt } x$
	R2: $(E, Y), (E', Y')$	$(2, k) \text{ secret shares created by } (x, r)$

In GDA, the prover U can prove to the verifier V that it possesses x , which contains the identity of U, that satisfies $m = f(x)$ without letting V know x . U has been registered to CA when it joins the system. To perform the proof, U has to withdraw a token from the CA:

$$TK = (W, K), \quad (1)$$

$$\text{where } W = (b\text{-sign}_{CA}(m, X), m, X), \quad (2)$$

$$\text{and } K = (b\text{-sign}'_{CA}(x, r), x, r). \quad (3)$$

W and K respectively denote the public and private data of TK. m is a unique message produced by U and CA together during the withdrawal. Both $b\text{-sign}_{CA}(\cdot)$ and $b\text{-sign}'_{CA}(\cdot)$ are *blind signatures* [11] of the CA. m is some message given to verifier during the proof. $f(\cdot)$ is a publicly known one-way function. r is a message randomly selected by U, whose public counterpart is denoted by

$$X = F(x, r) = F'(m, r). \quad (4)$$

$F(\cdot)$ and $F'(\cdot)$ are publicly known one-way functions. For a given TK, the proof is done via a *three-move* (3v) protocol as follows. U first sends W to V, and V replies a randomly generated challenge message E. U must generate a response message

$$Y = D(m, x, r, E), \quad (5)$$

where $D(\cdot)$ is known as the *prover function*. The messages resulted from the above protocol are collectively called the credential

$$CT = (W, E, Y), \quad (6)$$

and $m = f(x)$ can be verified if the signature in (2) is valid and

$$G(m, X, E, Y) = \text{"yes"}, \quad (7)$$

where $G(\cdot)$ is known as the *verifier function*.

It is noted that x is decipherable if and only if either of the following conditions holds:

R1: (E, Y) and r are available, for Y produced from (x, r) .

R2: (E, Y) and (E', Y') are available, where both Y and Y' are produced from the same (x, r) , and $E \neq E'$.

The deciphering condition R1 is based on the fact that r is the symmetric key that encrypts/decrypts x to/from (E, Y) , while R2 is based on the $(2, k)$ *secret sharing schemes* [12], where k is an integer greater than two.

Given the above facts, anonymity control of consumers in the computing resource access can be implemented by a simple *time-stamping method* as below: U first withdraws a token TK from CA. Then, U sends m to the SP as a request to schedule a session for service redemption. SP authorizes a session t to U by issuing a signed TAA message

$$Z = (\text{sign}_{SP}(t, m), t), \quad (8)$$

where $\text{sign}_{SP}(\cdot)$ denotes the digital signature signed by SP. Later, when U wants to redeem the services, it adds Z to the first move of 3v protocol, and executes the second and the third moves as usual. In addition to the verifications of the original 3v protocol, SP also needs to verify $\text{sign}_{SP}(t, m)$ at

the session t . The main weakness of this scheme is that TK cannot be reused for different sessions. The proposed TZKP is to allow reuse of token with protected anonymity for rule-abiding consumers.

3.2 TAA Generation: Session Time Authorization

TZKP considers the signed session time as one additional decipherability control variable, so that a token can be used for service redemption only once in each session. SP must be prevented from deciphering the identity of any consumer who does not redeem the service twice in a session.

In GDA, X (and its private counterpart x , r) is a control variable jointly generated by the CA and the consumer in the token withdrawal protocol. x can be deciphered when a token is spent twice, or the same X is used to produce two credentials. To make the token reusable in multiple sessions for TZKP, we take X out of the token in withdrawal phase. The consumer must produce a new X value (by the same x , different r) together with SP for each requested session, so that each jointly produced value of X can be used only once in the requested session without causing deciphering of its identity. By producing different values of X for different sessions, the consumer need not withdraw new tokens.

In TZKP, the SP issues to the consumer U a signed time stamp t of the requested session, and requires U to bundle (t, X) and its signature with the token for the consumer to be able to redeem the services or to transfer the TAA. If a consumer uses the jointly produced (t, X) value more than once in a session, the consumer identity can be deciphered, just like in the original GDA model.

The new definition of the modified token $\mathcal{TK} = (\mathcal{W}, \mathcal{K})$ in TZKP is:

$$\mathcal{W} = (\text{b-sign}_{CA}(m), m), \quad (9)$$

$$\text{and } \mathcal{K} = (\text{b-sign}'_{CA}(x), x), \quad (10)$$

where the modified parameters are denoted by the different font. In contrast to (2) and (3), X is separated from \mathcal{W} , and r is separated from \mathcal{K} . After these separations³, U is free to select different values of r (and X) after withdrawal of \mathcal{TK} . Each time when U requests a session t from the SP, U presents a unique X value and the message m of its token in the request. The SP authorizes session t by signing a TAA message

$$\mathcal{Z} = (\text{sign}_{SP}(t, m, X), t). \quad (11)$$

Note the difference between (8) and (11) that X is included in \mathcal{Z} but not in Z . When U redeems the service, it sends $(\mathcal{W}, X, \mathcal{Z})$ in the first round of the 3v protocol and then executes the second and third moves as usual. X is now an element of the modified credential

$$\mathcal{CS} = (\mathcal{W}, X, E, Y). \quad (12)$$

³ One technique for such separations is to set r to be some publicly known constant value in withdrawal, and let the consumer choose its own r value during transfer. For example, $r = 1$ when [9] is plugged in to GDA in [3].

The verifications are identical to those in the time-stamping approach introduced in section 3.1, except that correctness of (t, m, X) is checked by

$$\text{sign}_{SP}(t, m, X) \text{ and } \text{b-sign}_{CA}(m), \quad (13)$$

instead of

$$\text{sign}_{SP}(t, m) \text{ and } \text{b-sign}_{CA}(m, X). \quad (14)$$

Anonymity of U is guaranteed for \mathcal{CS} and \mathcal{CS}' produced by the same token \mathcal{TK} in different sessions. Even though \mathcal{W} and \mathcal{W}' have identical m , but \mathcal{Z} and \mathcal{Z}' have distinct values for (t, X) and (t', X') . The same value of m implies that \mathcal{CS} and \mathcal{CS}' are produced from the same token, but the distinct values for (t, X) and (t', X') imply that reuse of the token does not constitute a double spending of TAA because they are generated from different sessions. Since X and X' are produced from (x, r) and (x, r') respectively, it implies that (E, Y) and (E', Y') are produced from distinct (x, r) and (x, r') . Based on R2, x cannot be deciphered from \mathcal{CS} and \mathcal{CS}' .

It is easy to show that x can be deciphered if U doubly spends the TAA in a session. Thus, it will not be discussed further.

3.3. TAA Transfer: Multi-source Reusability (MSR)

For transfer of a TAA, one could directly apply the *general transferability model* in [4] to GDA, which suggests that a cascaded credential contains:

- (i) A TAA message signed by the SP,
- (ii) The credentials produced by all consumers in previous transfers of this TAA, and
- (iii) The credential produced in the current transfer.

From (6) and (8), the cascaded credential in GDA is

$$KC_i = (Z, CT_1, CT_2, \dots, CT_i), \quad (15)$$

where CT_j is the credential produced from U_j to U_{j+1} , and U_i and U_{i+1} are the grantor and grantee in the current transfer, respectively. As a general procedure to add transferability [4], U_i needs to send KC_{i-1} and W_i in the first round of the 3v protocol. Then U_{i+1} produces the challenge message as the hash value of its token public data, instead of a random number:

$$E_i = H(W_{i+1}), \quad (16)$$

where $H(\cdot)$ is a collision-resistant one-way hash function. In the third round of the protocol, U_i sends the response to U_{i+1} as usual. Then, U_{i+1} needs to

- (i) verify Z and each credential in KC_i , and
- (ii) verify the linkage between each adjacent credential pair in KC_i by (16).

This step guarantees that all credentials on the cascaded credential can be verified for the said transfer. However, the weakness of this general approach is that the grantor needs to consume its token when it uses the challenge to produce the response - any reuse of token in GDA may compromise the anonymity of the grantor. We raise similar questions as in earlier discussions:

- When does the reuse of a token constitute a TAA double transfer?

- Can the consumer's identity be deciphered from reused tokens when no TAA is doubly transferred?

MSR refers to the condition under which the consumer can transfer its TAA to another without costing its token quota. The proposed MSR condition is to modify (16) and the cascaded credential format of GDA in (15), so that the withdrawal of new token can be eliminated when a received TAA is transferred to only one consumer, and thus the total amount of privileges carried by the received TAA does not increase. The cascaded credential under the MSR condition in TZKP is:

$$\mathcal{K}\mathcal{C}_i = (\mathcal{Z}, \mathcal{E}\mathcal{T}_1, \mathcal{E}\mathcal{T}_2, \dots, \mathcal{E}\mathcal{T}_i). \quad (17)$$

Despite the similarity between (15) and (17), one must note that X_1 is signed in \mathcal{Z} , but not in Z . X_j is contained in W_j of CT_j , but not in w_j of $\mathcal{E}\mathcal{T}_j$.

In Fig. 2 and 3, we use the node U_C to analyze different scenarios. Given two cascaded credentials resulted from the double transfers of a TAA, the identity of the consumer who makes the double transfers can be deciphered by R2, using the two credentials positioned right after their longest common prefix, e.g., U_C is identified from $\mathcal{E}\mathcal{T}_C$ and $\mathcal{E}\mathcal{T}'_C$ in Fig. 2.

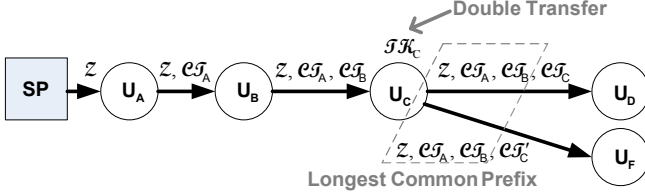


Fig. 2: Same-source Transfers – A Double Transfer.

Fig. 3a to 3c depicts several scenarios that do not constitute quota violations. In Fig. 3a, U_C needs to use its token to transfer TAAs from two distinct sources, but such a transfer pattern does not inflate the access privileges of TAAs, and thus the identity of U_C should be protected. In Fig. 3b, the two TAAs transferred from the same origin are meant for distinct sessions and the identity of U_C should be protected.

Note that same-source transfers are different from same-grantor transfers, as depicted in Fig. 3b, where U_C receives two TAAs from the same grantor but they are meant for distinct session times ($Z \neq Z'$). A similar argument can be made for Fig. 3c. It is worth noting that in Fig. 3c although U_C does not commit double transfers, U_A does, because the same Z value is in its two TAAs.

Definition: Two TAAs transferred from U_C , to U_D and U_F , are from the *same-source* if, and only if

$$\mathcal{K}\mathcal{C}_D = (\mathcal{Z}, \mathcal{E}\mathcal{T}_1, \dots, \mathcal{E}\mathcal{T}_i, \mathcal{E}\mathcal{T}_C), \quad (18)$$

$$\text{and } \mathcal{K}\mathcal{C}_F = (\mathcal{Z}', \mathcal{E}\mathcal{T}'_1, \dots, \mathcal{E}\mathcal{T}'_j, \mathcal{E}\mathcal{T}'_C) \quad (19)$$

have the common prefix

$$(\mathcal{Z}, \mathcal{E}\mathcal{T}_1, \dots, \mathcal{E}\mathcal{T}_i) = (\mathcal{Z}', \mathcal{E}\mathcal{T}'_1, \dots, \mathcal{E}\mathcal{T}'_j). \quad (20)$$

Otherwise, they are from the *multi-source*.

In summary, the same-source transfer pattern constitutes a double transfer violation but the multi-source transfers are not. A consumer should be allowed to use its token to make multi-source transfers with its anonymity protected. Based

on the analysis, we revise the 3v protocol so that the same-source transfers will assure identification of quota violator, but multi-source transfers will assure the consumer staying anonymous.

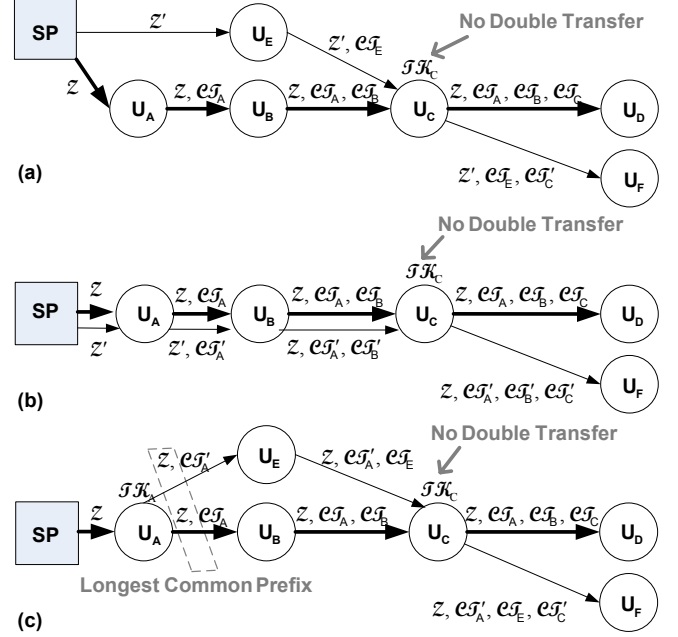


Fig. 3: Multi-source Transfers – Not a Double Transfer.

We attack this problem based on the technique similar to the one we discussed in last subsection via adjustments of X in GDA. A major difference between MSR enforcement and TAA issuance is that X_i cannot be signed by SP when a TAA is transferred from U_i to U_{i+1} . To overcome this, we propose to bundle X_i with grantor's challenge message \mathcal{E}_{i-1} as follows:

$$\mathcal{E}_i = H(Y_{i-1}, \mathbf{w}_{i+1}, X_{i+1}). \quad (21)$$

where Y_0 is defined as 0. Now, for U_i to engage in multiple (multi-source) transfers with protected anonymity, U_i needs to choose different values of r_i (and hence X_i) to produce its challenges when U_i receives TAAs from different sources. When U_i passes on the TAAs, U_i can use different values of r_i to produce its responses. Therefore, the anonymity of U_i is protected because x cannot be deciphered by R2.

Note that, in addition to X_{i+1} , Y_{i-1} is also added to the hash input of (21). We will show next that including Y_{i-1} in this way is crucial to prevent the collusion attack between consumers upon forgery of cascaded credential.

4. SECURITY ANALYSIS

Recall that the major equation changes from GDA to TZKP are from (8) to (11), and (16) to (21). The security analysis in this section explains how such changes can guarantee the identification of quota violator, and anonymity protection of rule-abiding consumers, and how to prevent forgery of the cascaded credentials under the collusion attack of malicious consumers.

First, we show that identification of quota violators can be guaranteed, by comparing the ways on how the value of X_i is engaged in GDA and TZKP. In GDA, X_i is included in CA's signature $b\text{-sign}_{CA}(m_i, X_i)$ in (2) when the token is withdrawn. In TZKP, the value of X_1 is included in SP's signature $\text{sign}_{SP}(t_1, m_1, X_1)$ in (11), and X_i ($i > 1$) is included in the challenge \mathcal{E}_{i-1} in (21), after withdrawal of the token. Assuming that the signature is secure, it is infeasible to find two different pairs of (m_i, X_i) which can produce the same signature. Also, it is infeasible to find two different pairs of (m_{i+1}, X_{i+1}) mapping to the same challenge \mathcal{E}_i because $H(\cdot)$ is collision-resistant. Suppose U_{i+1} violates the quota rule by transferring $\mathcal{K}\mathcal{C}_i$ twice. U_{i+1} is not able to alter the values of $\text{sign}_{SP}(t_1, m_1, X_1)$ and \mathcal{E}_i in $\mathcal{K}\mathcal{C}_i$. Otherwise, the verification on $\mathcal{E}\mathcal{F}_i$ would fail. Thus, U_{i+1} has to use (m_{i+1}, X_{i+1}) in both transfers, or otherwise, the linkage verification between $\mathcal{E}\mathcal{F}_i$ and $\mathcal{E}\mathcal{F}_{i+1}$ would fail. Furthermore, by (4), U_{i+1} has to use (m_{i+1}, r_{i+1}) in both transfers, or otherwise, the verifications on $\mathcal{E}\mathcal{F}_{i+1}$ or $\mathcal{E}\mathcal{F}'_{i+1}$ would fail. Based on R2, the identity of U_{i+1} can be deciphered from $\mathcal{E}\mathcal{F}_{i+1}$ or $\mathcal{E}\mathcal{F}'_{i+1}$.

Next, we discuss how X_{i+1} in (21) could allow reuse of a token for multi-source transfers with protected anonymity. Suppose that U_{i+1} uses the same token to transfer two TAAs from multi-source, denoted by $\mathcal{K}\mathcal{C}_i$ and $\mathcal{K}\mathcal{C}'_i$, respectively. Despite the same token (the same m_{i+1}) used, \mathcal{E}_i and \mathcal{E}'_i still have distinct values because U_{i+1} can select different values of X_{i+1} and X'_{i+1} to produce them, *i.e.*, using (m_{i+1}, r_{i+1}) in the first transfer, and (m_{i+1}, r'_{i+1}) in the second one. Based on R2, U_{i+1} cannot be identified.

Finally we explain why Y_{i-1} is needed in (21) to prevent forgery. So far, we consider that X_i is randomly selected by U_i , and this is bundled with \mathcal{E}_{i-1} of U_{i-1} , "before" it is used to produce $\mathcal{E}\mathcal{F}_i$ during the transfer between U_i and U_{i+1} . Based on this, the security level to forgery is equivalent to that of [3][4] because they are identical except that integrity of X_i is protected by different means: signatures of the CA or SP; hash value of X_i . However, when collisions are considered, $\mathcal{E}\mathcal{F}_i$ and $\mathcal{K}\mathcal{C}_{i+1}$ can possibly be forged by careful assignment of X_i before this is bundled with \mathcal{E}_i . Followings we describe how such a forgery attempt is possible without Y_{i-1} in (21), and then prove how Y_{i-1} can prevent this from happening.

In this forgery attack, U_i and U_{i+1} are colluders. First, U_{i+1} sends to U_i the challenge message $\mathcal{E}_i = H(\mathbf{w}_{i+1}, X_{i+1})$. U_i arbitrarily selects its Y_i . Given the inverse function of $G(\cdot)$, U_i derives an X_i which satisfies $G(m_i, X_i, \mathcal{E}_i, Y_i) = \text{"yes"}$, where m_i is from a valid token of U_i . When U_{i-1} transfers the authorization to U_i , U_i sends to U_{i-1} the challenge message $\mathcal{E}_{i-1} = H(\mathbf{w}_i, X_i)$ in the second move of the 3v protocol. U_{i-1} replies by Y_{i-1} as usual. Now, U_i has all data available to forge a credential $\mathcal{E}\mathcal{F}_i = (\mathbf{w}_i, X_i, \mathcal{E}_i, Y_i)$, and so the cascaded credential $\mathcal{K}\mathcal{C}_i$ to U_{i+1} . U_{i+1} can now transfer $\mathcal{K}\mathcal{C}_i$ to U_{i+2} without any anomaly detected. Since Y_i is selected by U_i arbitrarily, without encrypted data of its identity included, any violations done by U_i will not be identified.

To prove that Y_{i-1} in (21) can prevent forgery under the collusion attack, we introduce a dependency graph analysis, depicted in Fig. 4a to 4c.

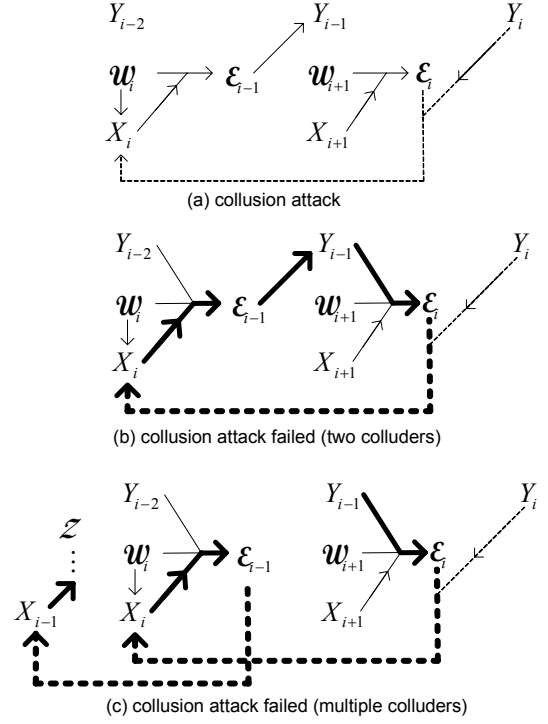


Figure 4: Dependency Graph Analysis.

In the dependency graphs, the arrow pointing from P to Q implies that creation of Q requires the prior knowledge of P. The solid line denotes the dependency when 3v protocol is executed in rule-abiding way. The dotted line denotes the dependency when this is executed based on forgery attempt. Derived from the dependency graph, we use the dependency formula,

$$(P_0, P_1, \dots) \rightarrow (Q_0, Q_1, \dots), \quad (22)$$

to denote that the creations of all parameters in Q_0, Q_1, \dots , require prior knowledge of some parameters in P_0, P_1, \dots , where P_i and Q_i are simply the data in the 3v protocol, or themselves the dependency formulas. Straightforwardly, the dependencies are transitive, *i.e.*,

$$(P \rightarrow V \text{ and } V \rightarrow Q) \text{ implies } P \rightarrow Q. \quad (23)$$

Fig. 4a depicts the case when Y_{i-1} is removed from (21). $\mathcal{E}\mathcal{F}_i = (\mathbf{w}_i, X_i, \mathcal{E}_i, Y_i)$ can be forged by creating parameters in the following sequence:

$$(((\mathbf{w}_{i+1} \rightarrow X_{i+1}) \rightarrow \mathcal{E}_i), Y_i, \mathbf{w}_i) \rightarrow X_i \quad (24)$$

From (24), all parameters to forge $\mathcal{C}\mathcal{T}_i$ are available to U_i , if U_{i+1} gives U_i the prior knowledge of \mathbf{w}_{i+1} and X_{i+1} . Fig. 4b and 4c depict the cases when Y_{i-1} is included in (21). In Fig. 4b, we only consider collusion of U_i and U_{i+1} , where the sequence of parameter creations is as follows:

$$((((Y_{i-1}, (\mathbf{w}_{i+1} \rightarrow X_{i+1})) \rightarrow \mathcal{E}_i), Y_i, \mathbf{w}_i) \rightarrow X_i, \mathbf{w}_i, Y_{i-2}) \rightarrow (\mathcal{E}_{i-1} \rightarrow Y_{i-1}) \quad (25)$$

From (25), a dependency loop (in bold lines of Fig. 4b) is formed, which means that the creation of Y_{i-1} requires the

prior knowledge of Y_{i-1} , which is a contradiction. Adversary has nowhere to initiate the malicious action, so the forgery attempt fails. Fig. 4c considers a series of colluders. The dependencies trace all the way back until some X_j , either $j > 1$ or $j = 1$. For $j > 1$, it means that U_{j-1} is not a colluder, who computes Y_{j-1} from \mathcal{E}_{i-1} , and then closes the loop. For $j = 1$, no dependency loop is formed, but X_1 is signed by SP in \mathcal{Z} , so the forgery attempt fails again.

5. COMPLEXITY ANALYSIS

The runtime and message size are measured based on three operations:

(Withdrawals of Tokens)

T_w = the total computation time for withdrawals

S_w = the total communication message size for withdrawals

(Transfers of Cascaded Credentials)

T_f = the total computation time for transfers

S_f = the total communication message size for transfers

(Detections of Quota Violations)

T_d = the total search time from the database

S_d = the total storage message size at the database

The analysis is based on the scenario that the SP grants p TTAs to a consumer. Each is legally transferred through the same set of q consumers before it is redeemed from the SP. The complexity is summarized in Table 2.

Table 2. Complexity Analysis.

Metrics	GDA	TZKP	TZKP
			($T_e/T_{ct} = \text{constant}$)
T_w, S_w	$O(pq)$	$O(q)$	$O(1)$
T_f, S_f	$O(pq^2)$ (q unbounded)	$O(pq^2)$ ($q < (T_e/T_{ct})^{1/2}$)	$O(p)$
T_d	$O(\log pq)$	$O(\log q)$	$O(1)$
S_d	$O(pq)$	$O(q)$	$O(1)$

In GDA, since the reuse of token is prohibited, the total number of withdrawals is the total number of transfers in the system, we have

$$T_w = O(pq) \text{ and } S_w = O(pq). \quad (26)$$

In contrast, in TZKP, a consumer can reuse a token for any number of multi-source transfers. So, only one withdrawal is required for each consumer, *i.e.*,

$$T_w = O(q) \text{ and } S_w = O(q). \quad (27)$$

For the transfers of cascaded credentials, we notice that the message size increases by one credential after each transfer, in both GDA and TZKP. The growing size also increases computation time for the cascaded credential. This is our desire to avoid the cumulative overheads, but this has been proven inevitable [4]. Intuitive reason for this is that, every anonymous consumer along a series of transfers could be potentially a quota violator, and so, when the authorization is transferred, the consumer has to contribute part of its identity information to the authorization data, before it is circulated back for central inspection for quota violation. Therefore, regardless of the token, credential and protocol designs, the total transfers in the system have the following complexity:

$$T_f = O(p \cdot (1 + 2 + \dots + q)) = O(p \cdot q^2) \quad (28)$$

$$\text{and } S_f = O(pq^2) \quad (29)$$

In spite of the same complexity formula used in (28), it has subtly different implications to GDA and TZKP. In GDA, q is unbounded because the authorization never expires, and can be transferred indefinitely before service redemption. In contrast, in TZKP, we have

$$q < T_e / ((1 + 2 + \dots + q) \cdot T_{ct}) < T_e / (q \cdot T_{ct}) \quad (30)$$

$$\Rightarrow q < (T_e / T_{ct})^{1/2}, \quad (31)$$

where T_e denotes the period starting from the authorization granted from the SP, to the end of session, and T_{ct} denotes the time required to verify one credential. This is derived from the fact that after a bounded number of transfers, the total time on cascaded credential verification in all transfers will exceed the allowed time for service redemption, and so, q cannot grow indefinitely. If (T_e/T_{ct}) is a (small) constant, then our solution is further optimized to:

$$T_w = O(1), \quad S_w = O(1), \quad (32)$$

$$\text{and } T_f = O(p), \quad S_f = O(p). \quad (33)$$

To analyze the complexity for detections of quota violators, we consider that each credential received by the provider is sorted in its database, and matching an incoming credential with n credentials in the database is based on the $O(\log n)$ time algorithm. In GDA to guarantee identification of quota violators in far future, credentials cannot be discarded once they are received. Therefore, we have

$$S_d = O(pq), \quad (34)$$

$$\text{and } T_d = O(\log pq). \quad (35)$$

In contrast, in TZKP, the credentials are kept only for one session of duration, and then they can be discarded after the examination of quota violations. Suppose that the p TAAs are meant for p different sessions, we have

$$S_d = O(q), \quad (36)$$

$$\text{and } T_d = O(\log q). \quad (37)$$

Again, if (T_e/T_{ct}) is a constant, this is further optimized to:

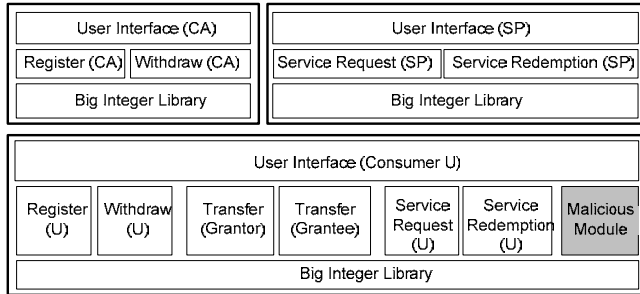
$$S_d = O(1) \quad (38)$$

$$\text{and } T_d = O(1). \quad (39)$$

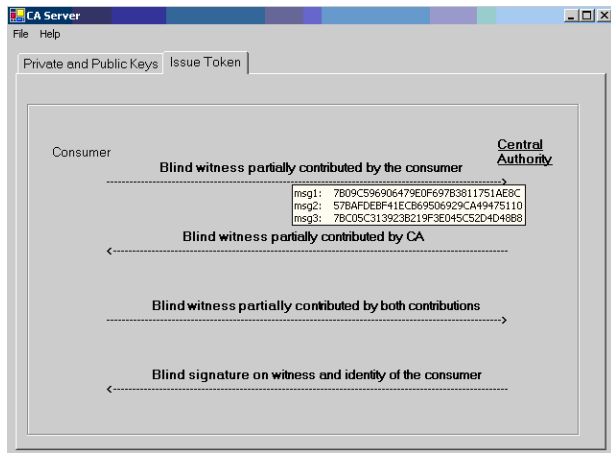
6. IMPLEMENTATION

We implemented TZKP on top of the software architecture of *CREAT* (Cybersecurity Remote Education Access Tool), whose binary version for the Windows operating system is available for download [14]. By plugging the Ferguson e-coin scheme [9] into GDA, with the modifications made to implement time-stamp and MSR in TZKP, we implemented the CA, SP and consumer modules, and tested them on the DETER testbed [5]. As the architecture depicted in Fig. 1, the CA node issues tokens to consumer nodes, the SP node grants TAAs, and consumers can request tokens from the CA, request TAAs from the SP, and also redeem services at scheduled time, or transfer the TAAs to other consumers. Key generation and other modular arithmetic are computed by using the big integer library [13]. SHA-1 is used for one-way computations. A 1024-bit RSA scheme is implemented for signature and other usages.

The architectures of three TZKP modules are illustrated in Fig. 5a, where a “malicious behavior” module is added to the consumer node, to simulate insider and outsider attacks. A screenshot of the token withdrawal at the CA node is also given in Figure 5b.



(a) Software Architecture of the Three TZKP Modules



(b) A Screen Shot of CA for Token Withdrawal

Fig. 5: Software Architecture of TZKP.

To measure run-times and message sizes of withdrawals and transfers, we repeat each experiment for 1000 runs and take the average value. The CA and the consumer machines are both equipped with 2.0 GHz Intel Pentium-4 processors, but the CA has 768M RAM, while the consumer has 512M. The simulation results are depicted in Tables 3 to 6. Note that RSA-1024 is considered the standard key strength for contemporary technologies and RSA-2048 is tested only as a reference. Clearly, the runtime of TAA transfer increases linearly with the length of the cascaded credential, but in real world practice, a limit is commonly set on the number of transfers due to administrative boundary. The limitation is further affected by the duration of a session. As such, one can expect that a small number of transfers in each session.

Table 3: Runtime and Message Size of a Withdrawal.

	RSA-1024	RSA-2048
Total runtime (sec)	4.0	22.4
Computation time (sec)	3.5	20.8
Transmission time (sec)	0.5	1.6
Token size (KB)	1.76	3.5

Table 4. Runtime (sec) of the i-th Transfer.

i	RSA-1024	RSA-2048
1	1.7	8.8
5	7.2	37.5
10	12.9	73.6
15	19.1	109.8

Table 5. Message Size (K byte) of the i-th Transfer.

i	RSA-1024	RSA-2048
1	2.17	4.30
5	9.87	19.5
10	19.5	38.5
15	29.1	57.5

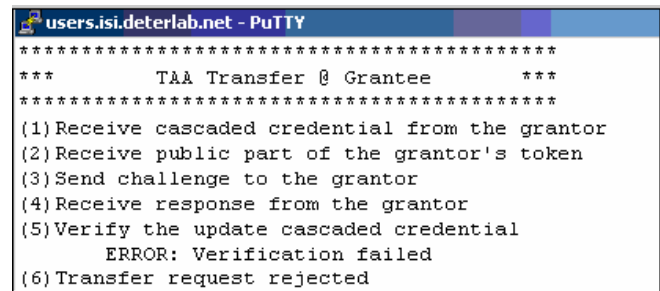
Table 6. Upper Bounds on Credential Storage Size (K Byte).

Te (sec)	RSA-1024	RSA-2048
60	< 19.53	< 12.90
120	< 26.04	< 21.50
240	< 39.06	< 30.10
480	< 54.25	< 43.00

Basic experiments on the registration, withdrawal, TAA request, transfer, and service redemption were first tested to evaluate the correctness of TZKP, under normal conditions, and under attacks by malicious outsiders and insiders. Fig. 6a shows that a consumer rejected a TAA being transferred using an invalid token because the verification step failed. Fig. 6b shows that the SP rejected the service redemption because the TAA presented by the consumer is expired. Fig. 6c shows that the SP deciphers the identity of the consumer who redeemed its TAA twice.

One can use traditional ZKP protocol to achieve similar security goals, but it faces the following major drawbacks: (1) there is no hierarchical distribution of access privileges, (2) each token can only be used for one-time access of the requested resource, and (3) indefinite storage of credentials. In addition to the storage overhead for credentials, double spending identification in ZKP traditional also significantly slows down with the number of spent tokens.

Other statistics on runtimes and message sizes of TZKP are depicted in Fig. 7 and 8. Experiments show that it takes about 4 seconds to withdraw a 1.7 KB token. The runtimes to transfer a cascaded credential (sized from 2KB to 30KB) increases from 2 to 20 seconds when its length grows from 1 to 15, which is an extreme condition to test the viability of the proposed scheme.



(a) A consumer rejects a TAA being transferred using an invalid token

```

users.isi.deterlab.net - PuTTY
*****
*** TAA Redemption @ Service Provider ***
*****
(1)Receive cascaded credential from the consumer
(2)Receive public part of the consumer's token
(3)Send challenge to the consumer
(4)Receive response from the consumer
(5)Verify the update cascaded credential
(6)Verify timed signature in the cascaded credential
    55B1A407A5305BEF1CAB6BCE8905D835CF5BCEF237D2A
(7)Check TAA time (60 secs per session)
    TAA issuance time:    1160585901
    TAA redemption time: 1160586020
    ERROR: TAA expired
(8)Redemption request rejected

```

(b) The SP rejects a redemption based on an expired TAA

```

users.isi.deterlab.net - PuTTY
*****
*** TAA Redemption @ Service Provider ***
*****
(1)Receive cascaded credential from the consumer
(2)Receive public part of the consumer's token
(3)Send challenge to the consumer
(4)Receive response from the consumer
(5)Verify the update cascaded credential
(6)Verify timed signature in the cascaded credential
    157DF6D54CF705D97450B1870486EDED7DCB0B602DC09
(7)Check TAA time (60 secs per session)
    TAA issuance time:    1160586298
    TAA redemption time: 1160586334
(8)Detect quota violation.
    ERROR: Double spending detected
    ID of the quota violator = 336
(9)Redemption request rejected

```

(c) The SP identifies a doubly spent TAA

Fig. 6: Screen Captures of TZKP Experiments on DETER.

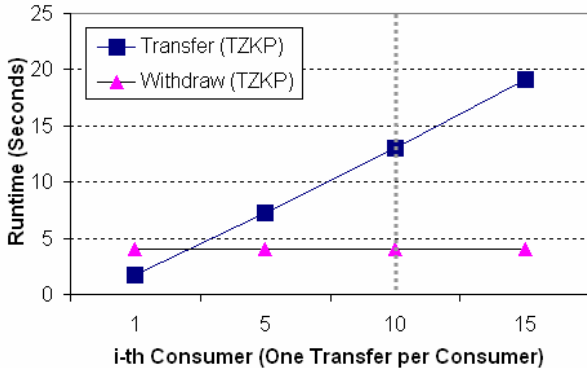


Fig. 7: Runtimes of Withdrawal and Transfer in TZKP.

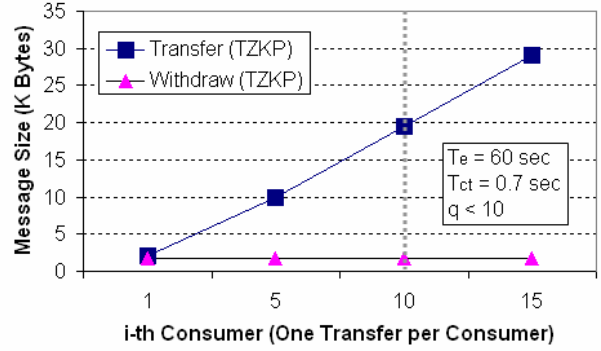


Fig. 8: Message Sizes of Withdrawal and Transfer in TZKP.

7. RELATED WORK

ZKP has been broadly investigated mostly in the contexts of cryptographic constructs, but there is rarely a cryptosystem which satisfies all requirements as in TZKP. For example, concurrent ZKP [15] considers time management in ZKP, aiming at protection of “proof” and “zero-knowledge” when multiple instances of ZKP are executed. It does not consider revocation of “zero-knowledge” on quota violator’s identity. E-cash systems [4] consider anonymity of the rule-abiding users and identification of the quota violators, but they do not have appropriate time management. The uncloneable group identification schemes [16] consider management of time, but the quota is not transferable. *Proxy signature* [17] is useful for transferring authorizations, but it requires the individual public key from every intermediate consumer (proxy signer) for signature verification. *Group signature* [18] is useful for verification of membership, without knowing the individual identity, but this is usually not transferable, and anonymity revocation is unconditionally controlled by the CA. On the contrary, TZKP allows peer nodes to transfer and verify the credentials, without prior knowledge on the identity of each another, and decipher the identity of quota violators.

8. DISCUSSIONS

ZKP, together with blind withdrawal and double spending identification protocols supports anonymous resource access with guaranteed quota control. The challenge-and-response authentication model provides strong protection of the transactions even when the consumers stay anonymous. Its key management system is simpler than the traditional PKI, because the SP and the consumers only need to know the public key of the CA and the SP to verify a transaction.

TZKP and the similar protocols are inherently immune from some of the most challenging threats, such as message replay, spoofing, *etc.* An eavesdropper is not able to reuse the collected messages because it does not have private part of the token to produce valid responses for new challenges. When a token is found to be stolen by the consumer, it just needs to request the CA to place its public token data \mathcal{W} on a black list to void the token. \mathcal{W} and its private counterpart

must be generated by the consumer and the CA together, and therefore, it is not forgeable. Consumers should refuse receiving TAAs from another consumer whose \mathcal{W} is on the black list. Similarly, the SP should refuse serving consumer whose \mathcal{W} is on the black list.

In spite of obvious benefits of ZKP in providing holistic and robust security management, there was virtually no experimental effort to test viability of ZKP for real world applications prior this paper. Our experiments show that TZKP can be easily supported by contemporary technology and the operational overheads can be effectively controlled. Although the e-cash paradigm, based on the concepts of blind withdrawal, ZKP and double spending identification, failed to replace real currency, mainly due to non-technical reasons [19], they can be effectively tailored for the critical applications with high requirements on privacy protection, quota control, and time management.

The TZKP protocol discussed so far only considers “one access per session” for each consumer request to the SP. Many more interesting issues related to the management of timed resources can and should be explored. We can easily extend the scheme to support “n accesses per session” by using the n-divisible tokens [20], or to grant n sessions in one request by modifying (11) to

$$\mathcal{Z} = (\text{sign}_{\text{SP}}((t_1, \dots, t_n), m, (X_1, \dots, X_n)), (t_1, \dots, t_n)). \quad (40)$$

Another interesting extension is the notion of attribute binding, where the SP can bind local attributes to the timed signature in (11), and a consumer can bind local attributes to the credential by adding them to the hash inputs of (21).

Multiple TAAs requested by using the same token of an anonymous consumer are linkable, because a same \mathcal{W} value assigned to the token must be used in different sessions to make the requests. We are currently investigating how to expand TZKP to provide unlinkable resource accesses.

A final note is related to the cost on the transferability of TAAs. Because the evaluation cost of a cascaded credential increases linearly with the number of nodes that it passes, it is necessary to limit the number of transfers for practical reasons, say 5 to 10. It also suggests that “fat and shallow” structure is more efficient than “thin and tall” structure for distribution of authorization privileges.

9. CONCLUSION

In this paper we propose a timed ZKP (TZKP) protocol on the basis of GDA to support the session-based access of computing resources. We have implemented and evaluated TZKP on the DETER testbed. The running time of TZKP for computers with modest resources was found to be quite reasonable. In its current form, our TZKP eliminates the overheads for token withdrawals and the storage of aged credentials. On the basis of these time-based primitives, we are developing secure distributed algorithms such as leader election, group formation, and exclusive group selection, etc., with promising preliminary results.

10. ACKNOWLEDGEMENT

This work is supported in part by an NSF CNS-5030210 and DUE-0516825.

11. REFERENCES

- [1] Information security Glossary web page, RSA security: <http://www.rsasecurity.com/glossary/default.asp?id=1054>
- [2] S. Haber and W. Stornette, *How to time-stamp a digital document*, Journal of Cryptology, vol. 3 (1999) 99 – 111.
- [3] T. Eng and T. Okamoto, “Single-term divisible electronic coins.” In *Advances in Cryptology - EUROCRYPT' 94* (1994) 311-323.
- [4] D. Chaum and T. Pedersen, “Transferable cash grows in size.” In *Advances in Cryptology - EUROCRYPT' 93* (1993) 390 – 407.
- [5] DETER Testbed: <http://www.isi.edu/deter/>
- [6] D. Chaum, A. Fiat, and M. Naor, “Untraceable electronic cash.” In *Advances in Cryptology - CRYPTO'88* (1989) 319-327.
- [7] S. Brands, “Untraceable off-line cash in wallets with observers.” In *Advances in Cryptology - CRYPTO'93*, (1994) 302-318.
- [8] C. Schnorr, *Efficient signature generation by smart cards*, Journal of Cryptology, vol. 4, no. 3 (1991) 161-174.
- [9] N. Ferguson, “Single term off-line coins,” In *Advances in Cryptology - EUROCRYPT'93* (1993) 318-328.
- [10] T. Okamoto and K. Ohta, “Disposable zero-knowledge authentication and their applications to untraceable electronic cash,” In *Advances in Cryptology - CRYPTO '89* (1990) 481-96.
- [11] D. Chaum, “Blind signatures for untraceable payments.” In *Advances in Cryptology - CRYPTO' 82* (1982) 199 – 203.
- [12] A. Shamir, “How to share a secret.” *Communications ACM* vol. 22, no. 11 (1979) 612 – 623.
- [13] C. Tan, C# BigInteger Class: <http://www.codeproject.com/csharp/biginteger.asp?target=biginteger>
- [14] TZKP Demo: <http://rtds.cs.tamu.edu/security.php>
- [15] C. Dwork, M. Naor, and A. Sahal, “Concurrent zero-knowledge.” In *Proceedings of the 30th Annual Symposium on Theory of Computing*, ACM (1998) 409-418.
- [16] I. Damgård, K. Dupont, and M. Pedersen, “Unclonable group identification.” Cryptology ePrint Archive: 2005/170
- [17] Z. Tan and Z. Liu, “Provably secure delegation-by-certificate proxy signature schemes.” In *Proceedings of 3rd International Conference on Information Security* (2004) 38-43.
- [18] J. Kiu, V. Wei, and D. Wong, “Linkable spontaneous anonymous group signature for ad hoc groups (extended abstract).” *Information Security and Privacy (ACISP)* (2004) 325-335.
- [19] M. Froomkin, “The unintended consequences of e-cash.” Computers, Freedom and Privacy Conference (1997).
- [20] T. C. Lam, C-C. Tan, and J-C. Liu, “Hypercube based N-divisible token and its integration with subcube allocation schemes.” Submitted to *IEEE Transactions on Secure and Dependable Computing*. Also available at Technical Report 2006-9-2, Department of Computer Science, Texas A&M University.