ANONYMOUS, AUTHENTIC, AND ACCOUNTABLE RESOURCE MANAGEMENT BASED ON THE E-CASH PARADIGM

A Dissertation

by

TAK CHEUNG LAM

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

May 2008

Major Subject: Computer Science

ANONYMOUS, AUTHENTIC, AND ACCOUNTABLE RESOURCE MANAGEMENT BASED ON THE E-CASH PARADIGM

A Dissertation

by

TAK CHEUNG LAM

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Approved by:

Chair of Committee,	Jyh-Charn (Steve) Liu
Committee Members,	Sing-Hoi Sze
	Anxiao (Andrew) Jiang
	Bojan Popov
Head of Department,	Valerie E. Taylor

May 2008

Major Subject: Computer Science

ABSTRACT

Anonymous, Authentic, and Accountable Resource Management based on the E-cash Paradigm. (May 2008)

Tak Cheung Lam, B. S., The Chinese University of Hong Kong;

M. S., The Chinese University of Hong Kong

Chair of Advisory Committee: Dr. Jyh-Charn (Steve) Liu

The prevalence of digital information management in an open network has driven the need to maintain balance between *anonymity*, *authenticity* and *accountability* (AAA). Anonymity allows a principal to hide its identity from strangers before trust relationship is established. Authenticity ensures the correct identity is engaged in the transaction even though it is hidden. Accountability uncovers the hidden identity when misbehavior of the principal is detected. The objective of this research is to develop an AAA management framework for secure resource allocations. Most existing resource management schemes are designed to manage one or two of the AAA attributes. How to provide high strength protection to all attributes is an extremely challenging undertaking. Our study shows that the *electronic cash* (*E-cash*) paradigm provides some important knowledge bases for this purpose. Based on Chaum-Pederson's *general transferable E-cash model*, we propose a *timed-zero-knowledge proof* (TZKP) *protocol*, which greatly reduces storage spaces and communication overheads for resource transfers, without compromising anonymity and accountability. Based on Eng-Okamoto's *general divisible E-cash model*, we propose a

hypercube-based divisibility framework, which provides a sophisticated and flexible way to partition a chunk of resources, with different trade-offs in anonymity protection and computational costs, when it is integrated with different sub-cube allocation schemes. Based on the E-cash based resource management framework, we propose a privacy preserving *service oriented architecture* (SOA), which allows the service providers and consumers to exchange services without leaking their sensitive data. Simulation results show that the secure resource management framework is highly practical for mission-critical applications in large scale distributed information systems.

DEDICATION

To *My Wife* and *Parents*

ACKNOWLEDGEMENTS

I would like to sincerely thank my advisor, Dr. Jyh-Charn Liu, for his support, guidance, encouragement, and trust. He is more a mentor than a mere advisor to me. He set a high standard for me that really helped me become mature, both personally and professionally. I would like to thank Dr. Sing-Hoi Sze, Dr. Bojan Popov, and Dr. Anxiao Jiang for spending their valuable time and effort in serving on my committee. They gave me very helpful suggestions on improving the quality of this dissertation. I would like to thank Dr. Scott Pike also for his advice on my preliminary stage of research.

I would like to thank my labmates in the Real-time Distributed Systems Lab at TAMU. I am greatly in debt to Cheng-Chung Tan and Pu Duan for their help in various projects. I would like to thank Yong Xiong, Hong Lu, Ming Zhang, Chunhua Tang, Xu Yang, Yiping Shen, Shengya Lin, Steven Chang, and Huajun Ying for their generosity in sharing many ideas with me.

Last but not least, I appreciate my beloved wife, my parents and my sisters. Without their love, patience, and support, I would not have finished my Ph.D study.

NOMENCLATURE

AAA	Anonymity, Authenticity, Accountability
BC	Binary Code
BRGC	Binary Reflected Gray Code
CA	Central Authority
DHT	Distributed Hashing Table
DSI	Double Spending Identification
DVS	Delegation Key, Verification Key, Secret Share
GDA	General Disposable Authentication
GDM	General Divisibility Model
GTM	General Transferability Model
MLBF	Multi-Layer Bloom Filter
KDM	Key Dependency Map
MSR	Multi-Source Reusability
P2P	Peer-to-Peer
RC	Random Code
SH	Secret Handshake
SOA	Service Oriented Architecture
TAA	Timed Access Authorization
TZKP	Timed Zero-Knowledge Proof
XML	Extensible Markup Language

TABLE OF CONTENTS

Page

Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

I. INTRODUCTION

The prevalence of digital information management in an open network has driven a new level of expectation on security and privacy protection. One of the key issues is to maintain the fragile balance between *anonymity*, *authenticity*, *and accountability* (AAA). In a hostile environment, a principal may want to remain its identity anonymous when it communicates with strangers. In the mean, the principal also wants to guarantee that the stranger is a valid user with an authentic identity to take part in the communications. An authentic identity needs to be unveiled to account for misbehaviors from the ill-minded adversary in the group. Balance of the AAA strengths calls for a holistic design strategy in large-scale, distributed information systems. One must carefully adjust the strengths between the three management arms to yield the highest productivity for mission-critical applications.

The objective of this research is to develop secure resource allocations in an open network based on the AAA management criteria. Different from tangible resources which can be transferred physically, allocation of digital resources is essentially an exchange of authentication messages which claim the ownership of the resources. AAA management allows an anonymous principal to prove the ownership of its resources, and at the same time, it allows identification of the principal who transfers the ownership which has been transferred to others, via replay of authentication messages. One solution

---

This dissertation follows the style of *IEEE Transactions on Networking*.

approach is to relay all messages via a trusted server but then the trusted server becomes a single point of failure because any malfunctions of the trusted server will cease all system activities immediately. We will show that with proper enhancements on the *electronic cash* (*E-cash*) framework, one can manage the AAA attributes for resource allocations with minimum interventions from the trusted server.

E-cash was originally designed to support secure, anonymous transactions and culprit identifications for digital currency duplications. Distinguished from other digital payment systems, E-cash requires no centralized supervisions in the transaction phase. The elegant cryptographic constructs of E-cash provide important knowledge bases for the designs of secure resource allocations. Based on the E-cash framework, a principal can transfer the *ownership* of its resource (or money) to another principal. During the transfer, the transferring principal needs to show the proof of ownership. At the same time, the proof must not leak its user identity to the receiving principal unless a *double-transfer violation* occurs. A double-transfer violation occurs when a principal duplicates the proof of ownership and transfers them to different principals. The design of E-cash ensures that the user identity of the double-transfer violator can be deciphered from the proofs after-the-fact. In this dissertation, we investigated two major topics related to E-cash based resource allocation: *transferability management* and *divisibility management*.

Transferability is a fundamental need for secure resource allocations but it is also proven a costly operation under the *general transferability model* (*GTM*) [1]. We proposed a *timed zero-knowledge proof* (*TZKP*) *protocol* which drastically reduces the storage size and the communication overheads from $O(n)$ to constant, where n is the total

number of transfers occurred in the system. We will show that, by proper incorporation of *session time* into GTM, a principal can regularly discard the outdated log files, without losing track of potential double-transfer offenders. In contrast, original GTM requires the log files to be kept forever in order to guarantee accountability of double-transfer offenders. We will also show that, by distinguishing the *multi-source condition* from the *same-source condition*, and including them as a deciphering trigger in GTM, a principal can reuse a single *token* to exercise unlimited number of legitimate transfers, without leaking its user identity. In contrast, original GTM requires the principal to withdraw a new (dummy) token for each transfer in order to guarantee anonymity for the transferring principal. Since withdrawal is typically the most expensive operation in E-cash systems, the proposed reusable token significantly reduces the system overheads for E-cash based resource allocations.

Divisibility is useful to manage a chunk of resources using minimum number of tokens. A principal can use a single token to generate the ownership proofs for different divisions from the resource chuck, instead of using one token for each atomic unit of the chunk. We proposed a *hypercube-based divisibility framework*, which expands the number of possible divisibility configurations from $O(2^{m+1})$ to $O(3^m)$, in contrast to the tree-based framework of the *general divisibility model* (*GDM*) [2], where m is the number of bits to represent an atomic unit of the resource chunk. We analyzed the cryptographic constraints to maintain the AAA balance in the hypercube-based divisibility framework, and derived a scheme which yields better performance by a relaxed anonymity constraint. Simulation results show that my scheme assures

anonymity at a small extra cost of *fragmentation* (< 2%) under an unrestrictive sub-cube allocation scheme. Moreover, such a small cost can further be eliminated (0%) when restrictive sub-cube allocation schemes, such as *binary code* (*BC*) and *binary reflected gray code* (*BRGC*), are used to distribute resources [3]. It suggests that with proper adjustments to both the divisible tokens and resource allocation rules, highly secure and efficient resource management schemes can be developed based on one integrated framework.

Based on the above secure resource management solutions, we proposed a privacy-preserving framework for the reservation-based service oriented architecture (SOA). In the proposed system, a registered *service consumer* can reserve services (resources) from the *service provider*. The service consumer can choose to transfer the *service reservation* to another consumer. The service provider provides service to anyone who can present a valid service reservation. Although the E-cash paradigm paved a solid foundation for the protection of user identity in the above scenario, there is a missing link is on the privacy protection of inquiry and service contents during service discovery. To bridge this gap, we interfaced the E-cash based resource management framework with a *peer-to-peer* (*P2P*) system, *CHORD* [4], and extended its lookup protocol with the *multi-layer Bloom filter* (*MLBF*) [5], so that the service consumer can query other peer nodes for wanted services while those peer nodes do not have knowledge on both the inquiry and service contents. Such a privacy protection is important for collaborations between competing parties in a hostile environment. In addition, we proposed a *secret handshake* (*SH*) protocol to control the release of

sensitive information to anonymous party based on different qualifications, trust levels, capabilities, rights, or privileges represented by the group. The proposed SH protocol allows two anonymous principals to use their SH tokens to authenticate each other whether or not they belong to the same group. If they belong to the identical group, they are able to establish a common secret key for further transaction actions. Otherwise, they are not able to recognize the group identity of one another. The proposed protocol allows reuse of the SH token in different transactions without leaking information of the user identity. In contrast, the existing work which uses 2 pairing operations requires the withdrawal of a new SH token for each transaction in order to guarantee anonymity [6]. And the existing work [7] which supports reuse of token requires 6 pairing operation [8]. Our protocol provides a light-weight group authentication mechanism to filter strangers from different groups before proceeding to the more sophisticated authentications for the transaction phase in resource allocations. Preliminary experimental results show that the proposed secure resource management framework is highly practical for mission-critical applications in large scale distributed information systems.

The following sections of this dissertation are organized as follows. Section II introduces the background of the E-cash paradigm and its unified framework for secure resource allocations. Sections III and IV explain the transferability management and the divisibility management for secure resource allocations, respectively. Section V presents the reservation-based SOA in a P2P environment. Section VI provides a summary of the research work.

II. E-CASH PARADIGM

Electronic cash, or E-cash, was first proposed by Chaum in [9] to support secure anonymous transactions. Distinguished from other electronic payment systems, E-cash does not require online supervisions from the trusted third party during the transaction phase. Therefore, it is usually regarded as the *off-line payment model*. Although E-cash has not been broadly deployed to replace the paper-based currency, mainly due to non-technical concerns [10], its elegant cryptographic constructs for off-line authentications and privacy protection have offered important knowledge bases for the development of large-scale critical applications at different clearance levels. Examples include, but not limited to, electronic voting [11], population survey [12], collaborations with competing parties [13], role based information sharing [14], and mobile agent tracking system [15]. In this section we will introduce the E-cash paradigm and based on it to develop a secure resource management framework.

A. E-cash Basic Operations

The basic operations in the E-cash paradigm are depicted in Fig. 1. In its original form, a principal needs to *register* an account from the bank. Then, the *bank* will assign an *identity* (*account number*) to the principal and maintain the records of the principal. A registered principal can *withdraw* some *tokens* from the bank. Each token has a unique *serial number* signed by the bank. The serial number is associated with the encrypted

identity of the principal and some secrets for proving the ownership of the token. The principal (*payer*) can *spend* the token to another principal (*payee*). The *spent token* is an encrypted form of the token called the *credential*. The payee verifies the credential and deposits the credential to the bank. The credentials received by the payees or deposited to the bank cannot be associated with the payer's identity unless the payer spends the same token again to another transaction. This is called the *double spending* offense. The *double spending identification* (*DSI*) system can decipher the identity of double spending offender from the involved credentials. From the above description, the E-cash paradigm demonstrated a harmonic AAA balance between three types of principals: the payer can stay anonymous; the payee can assure the transaction made by the payer is authentic; the bank can assure that double spending offense is accountable. This simple AAA model is the foundation of our secure resource management framework in subsequent discussions.

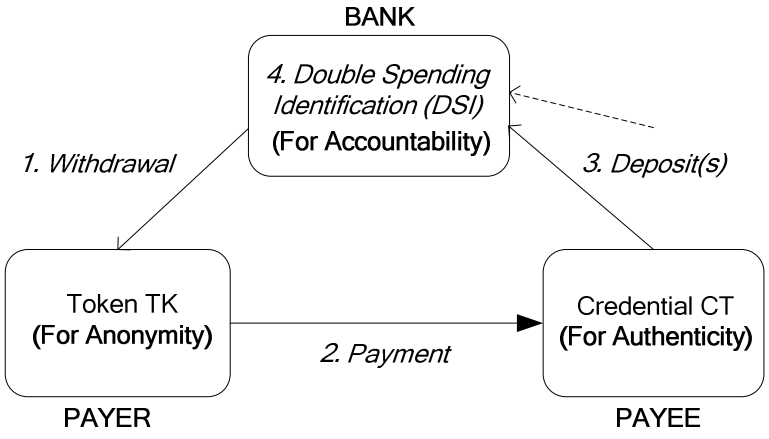


Fig. 1: AAA relationship in the E-cash paradigm.

B. E-cash High Level Construct

E-cash involves a myriad of cryptographic tools which interlocked relationship is carefully exercised to satisfy the AAA requirement simultaneously. In this section, we will introduce three cryptographic primitives commonly used by E-cash constructs: *blind signature schemes* [9, 16-21], *zero-knowledge proof* (*ZKP*) *protocols* [22-27], and *secret sharing schemes* [28-35]. A high level construct is described in Fig. 2.

1. Blind signature schemes

Blind signature schemes [9] allow the bank to sign a message without knowing the exact message contents. In E-cash, the signed message refers to the serial number of the token. Therefore, the bank is not able to associate the serial number from deposited credential with the principal's identity from the withdrawal record. A simple example of blind signature scheme is described as follows. Let (e, n) and (d, p, q) be the RSA public keys and private keys of the bank respectively. The principal wants to receive the bank's signature on message M without letting the bank know the plaintext of M.

BLIND: The principal randomly generates an integer r and send $r^e \cdot M \bmod n$ to the bank.

SIGN: The bank sends the signature $(r^e \cdot M \bmod n)^d \bmod n = r \cdot M^d \bmod n$ to the principal.

UNBLIND: The principal retrieves the signature by $((r \cdot M^d \bmod n)/r) \bmod n = M^d \bmod n$.

VERIFY: The bank's signature on M is correct if $(M^d \bmod n)^e \bmod n = M$.

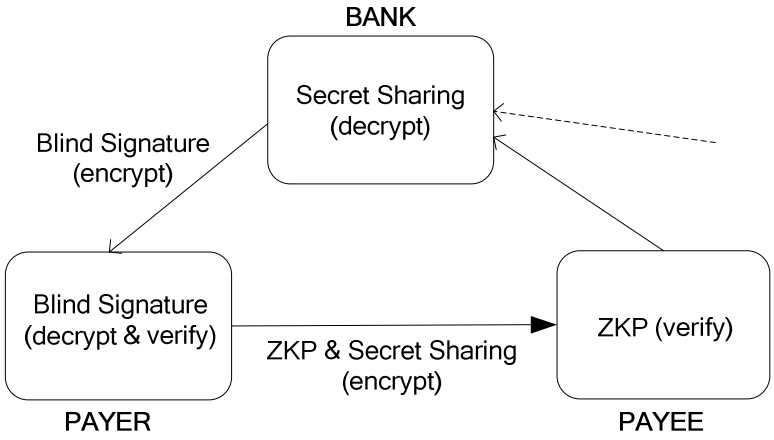


Fig. 2: High level constructs of E-cash system.

In the above simple example, the bank has no knowledge on the signed message *M*, because it does not know the random number *r*. In E-cash systems, the bank does not know the signed serial number of the token but it also needs to guarantee that the signed serial number is associated with the encrypted identity of the principal. It requires more advanced schemes such as *randomized blind signature* [36]. Advanced blind signature schemes induce severe overheads for the withdrawal operations in most E-cash schemes. In this dissertation, instead of designing efficient blind signatures for E-cash, our scheme minimizes the number of withdrawal operations needed to reduce the system overheads.

2. Zero-knowledge proof (ZKP) protocol

Zero-knowledge proof (ZKP) protocols [22] allow a principle (*prover*) to prove to another principal (*verifier*) that the presented message is associated with some secrets

under certain constraints but not allowing the verifier to know the secrets. In E-cash, the presented message refers to the serial number of the token, which is associated with the encrypted identity of the principal and some secrets for the principal to prove the token's ownership. It guarantees that the identity of the principal cannot be deciphered from the credential. A simple example of ZKP protocol is described as follows. Let (e, n) and (d, p, q) be the RSA public keys and private keys of the payer. The payer presents the public keys to the payee and wants to prove that it knows the private counterparts.

WITNESS: The payer publishes (e, n) to the payee.

CHALLENGE: The payee randomly generates a message M and sends it to the payer.

RESPONSE: The payer signs M and returns the signature $M^d \bmod n$ to the payee.

VERIFY: It is proven that the payer knows (d, p, q) if $(M^d \bmod n)^e \bmod n = M$.

In the above simple example, adversaries cannot pretend to know the private part by replaying the *response* because the *challenge M* is unpredictable. This *challenge-and-response* approach of ZKP is broadly used in E-cash systems. First, the payer presents the serial number of token to the payee. Then, the payee sends back a random challenge to the payer. The payer needs to sign the challenge message by the secrets of the token to give the response. Different from the simple example, the payee can verify the response message using the public parameters from the bank, instead of the individual public keys of the payer. It guarantees that no individual public keys can be linked to two payments of the same payer. Another difference is that typical ZKP protocols aim to protect secrets unconditionally while ZKP protocols used in E-cash systems need to decipher the secrets

(identity) on double-spending offense. Therefore, ZKP protocols used in E-cash systems are usually integrated with secret sharing schemes.

3. Secret sharing scheme

Secret sharing schemes [28] allows a secret message to be encrypted in multiple *secret shares*. A (k, n) secret sharing scheme guarantees that any k out of n secret shares can be used to reconstruct the secret message while fewer than k secret shares cannot. In E-cash systems, the secret message refers to the identity of the payer. The payer needs to produce one secret share to the payee. If the payer doubly spends the token and produces multiple secret shares, its identity can be involuntarily deciphered from the secret shares when their credentials are deposited to the bank. A simple example of a secret sharing is described as follows. Let (a_0, …, a_{k-1}) be the secret message to be encrypted in the (k, n) secret sharing scheme:

$$f(x) = a_0 + a_1 \cdot x + a_2 \cdot x^2 + \ldots + a_{k-1} \cdot x^{k-1} \tag{2.1}$$

SECRET SHARING: Compute the i^{th} secret share (x_i, $f(x_i)$), for some x_i, $i = 0, \ldots, n-1$.

SECRET RECOVERY: Use any k secret shares to construct k polynomials by (2.1). The secret message (a_0, …, a_{k-1}) can be recovered by interpolations of the k polynomials.

In this above simple example, the secret message is selected by the payer. On the other hand, E-cash systems have to enforce the payer to use its secret identity to produce the secret shares so that double spending offenses are accountable. Thus, secret sharing schemes in E-cash systems are usually integrated with ZKP, as we have mentioned in the

previous section. As we will see in subsequent discussions, the integrated constructs of secret sharing scheme and ZKP protocol form strong bonding among the AAA attributes for secure resource management.

C. E-cash Design Space

Numerous E-cash branches were proposed in the past two decades with different emphasis on *linkability*, *traceability*, *transferability*, *divisibility*, and *token revocation*, as depicted in Fig. 3. In this section, we will give a brief introduction to various branches and discuss their relationship. These branches inspired a very rich design space for mission-critical applications with similar AAA needs.

1. Linkability

Two credentials are *linkable* to a principal if the principal can determine whether or not both the credentials are produced by the same payer while the principal does not necessarily knowing the payer's identity. Two credentials involved in a double spending need to be linkable in order to enable DSI operations. Thus, linkability is usually used to describe the credentials without involved in a double spending offense.

Linkability can further be classified into *multi-token linking*, *same-token linking*, and *sub-token linking*. Multi-token linking is the case when the linkable credentials are produced by different tokens. It is useful to defense against misbehaviors such as *money*

laundering [37]. Same-token linking refers to the case when the linkable credentials are produced by the same token. A special case known as *forward linkability* occurs when a credential revisits the same payer after a series of transfers which is proven inevitable in the off-line model [1]. Sub-token linking refers to the case when the linkable credentials represent sub-values of a *divisible token*. Sub-token linking appears in most divisible E-cash schemes [2, 38-40] with the exceptions of [41, 42].

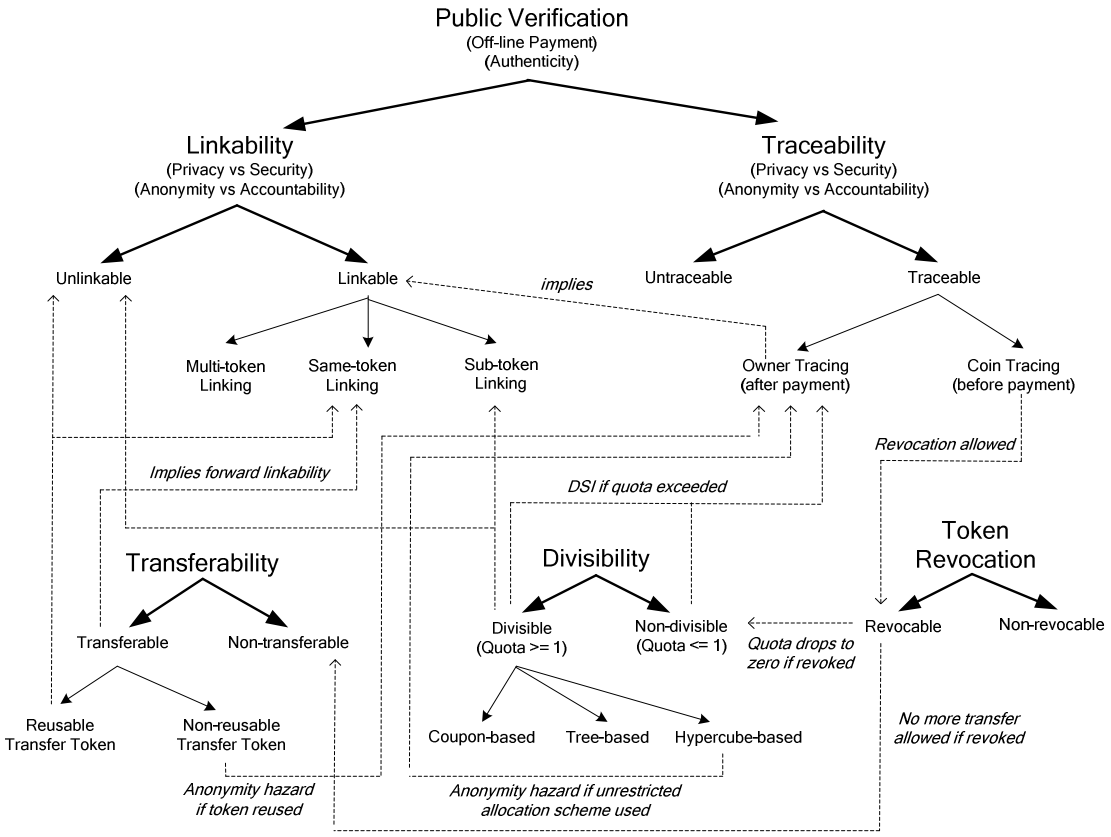


Fig. 3: E-cash design space.

2. Traceability

A credential is *traceable* to a principal if the principal can associate the credential with the withdrawal records of the token which generated this credential. Similar to the linkability, traceability is usually used to describe the credentials without involved in a double spending offense.

Traceability can further be categorized into *coin tracing* and *owner tracing*. Coin tracing refers to the case when a principal other than the payer can tell the serial number of the credential before the token is spent. It is necessary to create a token revocation list [46]. Owner tracing refers to the case when a principal other than the payer can tell the payer's identity after the token is spent. It directly implies that the credential is linkable to the principal. Owner tracing is usually enabled by a trusted third party other than the bank to catch the misbehaviors in addition to double spending. Typically, the bank needs to use some public parameters of the trusted third party to create the token, and the bank needs the trusted third party to use it private keys to decipher the identity of the offender from the credential.

3. Transferability

An E-cash scheme is *transferable* if the received money, privileges, resources, or assets represented by a credential can further be spent to other principals without having to deposit it to the bank first. A number of transferable schemes are proposed [1, 43, 44].

A non-transferable scheme can be transformed into a transferable scheme by the general technique proposed in [1]. By using this technique, the payer needs to spend a (dummy*) transfer token* in each transfer, and the same transfer token cannot be reused in the next transfer, or otherwise, the payer's identity is traceable from the credentials produced by the same transfer token. As a result, the payer needs to withdraw many transfer tokens in order to execute multiple transfers, causing unnecessarily high system overheads. In this dissertation, we will explain how to mitigate these withdrawal overheads by constructing a reusable transfer token and discuss the impacts on the traceability and linkability.

4. Divisibility

An E-cash scheme is *divisible* if a token can be spent for multiple times in such a way that each credential produced from it represents a subdivision of money, privileges, resources, or assets. Each subdivision is weighted by a *value*. Double spending occurs if the total values spent exceed the permitted *quota*. Typically, divisible E-cash schemes are either *coupon-based* [37, 39, 41, 45] or *tree-based* [2, 38, 40, 42]. In coupon-based schemes, the values of subdivisions are uniformly distributed. In contrast, in tree-based schemes, a tree node at level i of the tree represents $1/2^i$ of the allowed quota value. In this dissertation, we developed a *hypercube-based* divisible scheme which supports more subdivision configurations than tree-based schemes. We will show that owner tracing is possible, although unlikely, in the hypercube-based scheme when an unrestrictive sub-cube allocation scheme is used. We will also show that the chance of owner tracing can

be dropped to zero if the hypercube-based scheme is integrated with some other well-known sub-cube allocation schemes.

5. Token revocation

When a token is revoked, the permitted quota of the token will drop to zero, and no more transfer made by the token is allowed. Revocable schemes usually require coin-tracing ability to black list the revoked serial numbers [46, 47]. In this dissertation, We will discuss the use of *session time* to invalidate an expired token for E-cash schemes without coin-tracing ability.

D. E-cash Based Resource Management Framework

An E-cash based resource management framework is shown in Fig. 4. It contains three major types of principals, the *central authority* (CA), *resource owner*, and *resource consumer*, and five major operations, *withdrawal*, *allocation*, *transfer*, *consumption*, and *DSI*. The CA and resource owner are assumed well-known to the resource consumers.

In Fig. 4, the resource owner possesses a chunk of resources represented by the hypercube $G_4 = \times\times\times\times$, where $\times$ denotes the "*don't care*" bit. The resource owner wants to allocate the access rights for parts of its resources, represented by the sub-cubes, to the resource consumers. In order to participate to resource allocations/transfers, the resource consumers U_0, U_1, U_0' and U_1' first withdraw some tokens from the CA. On the requests

of U_0 and U_0', the resource owner allocates sub-cubes 0×××× and 1×××× to U_0 and U_0', respectively. U_0 transfers sub-cubes 0×0× and 0×10 to U_1 and U_1', respectively, and lets itself consume the sub-cube 0×11 from the resource owner. Finally, U_1 and U_1' consume sub-cubes 0×0× and 0×10 from the resource owner, respectively.

So far none of the hypercube nodes is transferred or consumed more than once. Thus, the resource access provided by 0×××× = {0×0×, 0×10, 0×11} is completed at the resource owner's site after U_1 and U_1' finish their consumptions. Ideally, the credentials produced by these consumers should be untraceable and unlinkable. However, a double spending occurs when U_0' consumes the sub-cube 11×× from the resource owner while it transfers the sub-cube 1×1× to U_1' because the sub-cube 111× = {1110, 1111} is used twice. The DSI system must assure identification of U_0'. Optionally, token revocation can be used to prevent further violations of this resource consumer.

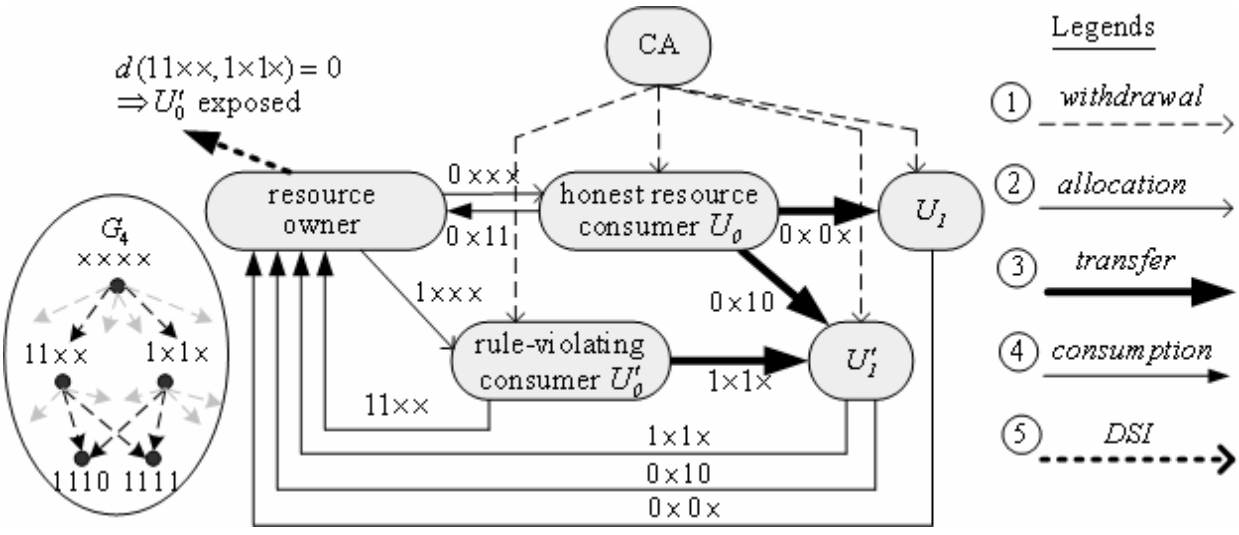


Fig. 4: E-cash based resource management framework.

The E-cash paradigm provided solid knowledge bases for the designs of secure resource management in a hostile peer-to-peer environment. However, directly applying E-cash may not be the most efficient mean to secure distributed systems, because it was originally designed for monetary applications only. In this dissertation we will adjust the E-cash algorithm to secure resource allocations with improved efficiency and flexibility.

III. TRANSFERABILITY MANAGEMENT

Transferability is important for resource management to switch the ownership for digital resources, assets or privileges from one principal to another principal. The E-cash paradigm provides elegant cryptographic constructs for transferability management but it induces serious system overheads under Chaum-Pedersen's general transferability model (GTM) [1, 48]. To make E-cash applicable to secure resource management, we proposed timed zero knowledge proof (TZKP) protocol for session-based access control of shared resources in an open environment. The main idea is to manipulate the anonymity control variables in Eng-Okamoto's general disposable authentication (GDA) model[1] [2] so that *session time* and *source of transfer* can be embedded into GDA as one of the decipher conditions. As a result, resource access authorizations assigned for different sessions (or transferred from distinct sources) can be managed independently by a single reusable token without compromising the anonymity requirement. At the same time, the credentials which have passed the current session can be safely discarded without weakening the accountability requirement. Our scheme maintains the AAA balance with a reduced number of tokens withdrawn and a reduced number of credentials stored. They are both reduced from $O(n)$ to constant where n is the number of transfer operations.

[1] The GDA model in this dissertation refers to the general construct of disposable authentication in the second part of [2], but not the specific construct in first part of [2] or the specific construct in [49].

One of the most critical concepts in our scheme is the *timed access authorization* (*TAA*). TAA is granted by the service provider to the service consumer when the service consumer requests a reservation of services (resources). It contains the service provider's signature on three messages: (i) the scheduled time for service redemption, (ii) the public part of the consumer's token, and (iii) the description of services. The consumer needs to present a valid TAA and the public part of its token in order to redeem services from the service provider at scheduled session. Coupling of session time and token in TAA allows the service provider to save storage by discarding the credentials associated with expired TAAs. In contrast, traditional schemes need to maintain credentials indefinitely to catch the service consumers from using aged tokens to redeem services. However, an improper use of TAA may lead to anonymity breaching of service consumers. We will show that our scheme can prevent adversaries from misusing expired credentials intentionally stored at the service provider. Our scheme guarantees the accountability for the service provider with reduced system overheads but not sacrificing the anonymity of honest service consumers.

Another critical concern is about the TAA transfers among service consumers. In additional to the original TAA issued from the service provider, a service consumer must be able to use its own token to receive and pass on the TAA to another service consumer. Different parts of the token is used for receiving and passing on the TAA. The TAA and the credentials produced by all service consumers who transferred this TAA are together called the *cascaded credential*. The cumulative size of the cascaded credential after each transfer is proven inevitable [1] thus there is no room to reduce bandwidth and storage in

this direction. On the other hand, the cost of token withdrawal per transfer operation can be significantly reduced by using the *multi-source reusability* (*MSR*) condition proposed in this section. Based on the MSR condition, the service consumer can transfer a number of TAAs anonymously without withdrawing a new token provided that these transfers do not constitute a double-transfer of TAA. A double-transfer happens when the number of TAAs passed on is more than received by the service consumer. The concept of double-transfer is mixed together with double-spending (reuse) of token in GTM. The main idea of MSR is to distinguish these two concepts and reflect their difference in the anonymity control variables of GDA. As a result, anonymity of rule-abiding service consumers can be guaranteed with much fewer withdrawal operations owing to the reusability of tokens. At the same time, we will show that accountability for double-transfer offense is assured in our scheme even under attacks from a series of colluders.

The rest of this section is organized as follows. Section A explains the protocol details for session-based management and the MSR condition. Sections B and C present security and complexity analysis. Section D delivers simulation results to demonstrate the applicability. Run time for token withdrawal and service redemption are within the range of seconds, making it highly practical to the secure access control of large scale Internet resources. Section E introduces the related work. Section F provides a summary of this section.

A. Timed Zero-Knowledge Proof (TZKP) Protocol

TZKP is designed for AAA management of shared resources at reduced system overheads comparing with the GDA model. TZKP consists of two major cryptographic modifications to the GDA model. The first one allows the service provider to issue TAA based on a signed session time bundled with the public data of the requester's token. The second one allows the service consumer to transfer numerous TAAs it received to other consumers by using the same token it possesses, and still assures the anonymity. In this section, we will introduce the system architecture for AAA resource management and then explain the details for the two modifications on the GDA model.

1. System overview

The system architecture depicted in Fig. 5 is similar to the one we have discussed in Fig. 4. However, in this section we emphasize on the transferability management while divisibility will be discussed in the next section. Following the generic architecture for shared resource access, there are three types of principals: the central authority (CA), the service provider, and the service consumer. The CA is responsible for issuing tokens to service consumers, while the service provider is responsible for issuing TAAs to service consumers, and rendering services to service consumers at authorized session time. The service consumer can choose to redeem services by using its TAA, or transfer it to other service consumers.

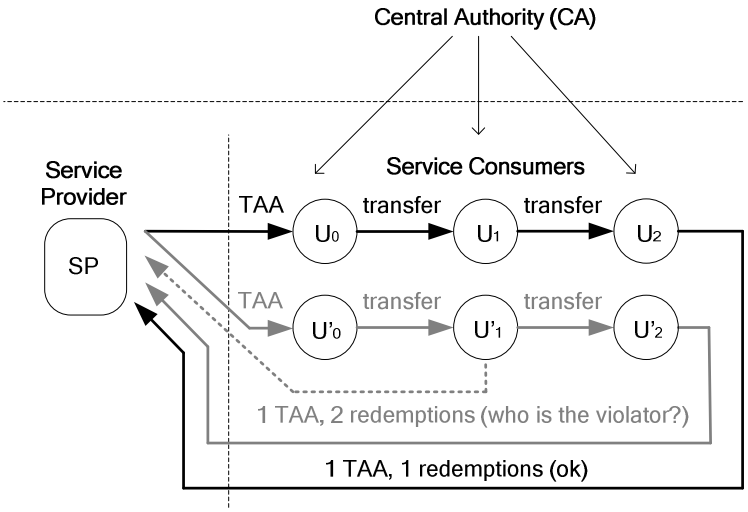


Fig. 5: Service model and adversary behaviors.

A consumer needs to first withdraw a token from the CA before it can request a TAA from the service provider, transfer TAAs to other consumers, and redeem services from the service provider. The consumer needs to use its token and TAA for transfer and service redemption. Having a token withdrawn from the CA, the consumer can initiate a service request by presenting the public part of its token, and the requested session time to access resources. The service provider grants the request by generating a TAA, which includes the signature on 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 the TAA to another consumer, *i.e.*, transfer of a TAA from the *grantor* to the *grantee*. Transferability of TAA is highly desirable because it allows the creations of hierarchical distribution architecture for resource access privileges. It is consistent with the current practice in the large scale experimental facilities, such as DETER[54]. By using TZKP, TAA becomes void once it passes the scheduled time. The service provider can safely

discard the credentials which have passed the current session without affecting its ability to detect double-transfer for future sessions. Except for minor operational differences the same authentication process is applicable to both service redemption and TAA transfer. Informally, we can consider service redemption as a TAA transferred back to the service provider. Unless explicit clarification is necessary, we only discuss TAA transfer in the remaining discussions.

TZKP is designed for rule-abiding consumers to manage access credits securely, while staying anonymous during operations. From the resource management viewpoints, it is easy to add TZKP to the existing resource sharing rules because it imposes virtually no restriction on how resources are reserved. The service provider simply stops issuing TAA when the reserved resources in a session time reach a target level. And redemption of services will be made to any principal who presents a valid credential, together with a valid TAA. For simplicity, we assume that the service provider has a free-run system clock. The time period of a session can range between minutes to hours for the shared resources because the computers often need to be reconfigured for various consumers. From the cryptographic analysis viewpoints, which are the focus of this paper, two main concerns need to be addressed. First, consumers may attempt to doubly transfer TAA in a session. Second, the service provider may attempt to decipher the consumers' identities by collecting an infinite number of redeemed session credentials even though no double-transfer of TAA in any session. we will show that neither of the two offenses can occur to TZKP, and the security properties of GDA are preserved within each session at reduced system costs.

2. General disposable authentication (GDA) model

The GDA model proposed by Eng-Okamoto in [2] is a versatile security control model which is compatible with various ZKP protocols designed for E-cash [36, 49-52]. In this section, we will explain the GDA basics needed to develop the TZKP protocol. The essence of GDA is summarized in Table 1, following the conventions defined in [2].

Table 1: General disposable authentication (GDA).

AAA Requirements	Information Needed	Remarks
verify $m = f(x)$	VU1: X, (E, Y)	$X = F(x, r) = F'(m, r)$, $Y = D(m, r, x, E)$, and $m = f(x)$ iff $G(m, X, E, Y)$ = "yes"
decipher x	DU1: (E, Y), r	r = symmetric key to encrypt/decrypt x
	DU2: (E, Y), (E', Y')	$(2, k)$ secret shares created by (x, r)
Keep x secret	Neither DU1 nor DU2 available	r = symmetric key to encrypt/decrypt x

In the GDA model, a prover U can prove that it possesses x, which satisfies $m = f(x)$ for certain constraint $f(\cdot)$, without letting the verifier V know x, where x contains the registered identity of U. To perform the proof, U has to withdraw a token from the CA:

$$TK = (W, K), \tag{3.1}$$

where

$$W = (b\text{-}sign_{CA}(m, X), m, X), \tag{3.2}$$

and

$$K = (b\text{-}sign'_{CA}(x, r), x, r). \tag{3.3}$$

W and K, represent the public and private parts of TK, respectively. m is a unique message co-produced by U and the CA. m will be given to the verifier during the proof. $b\text{-}sign_{CA}(\cdot)$ and $b\text{-}sign'_{CA}(\cdot)$ are *blind signatures* [9] of the CA. $f(\cdot)$ is a one-way function. r is a message randomly selected by U. The public counterpart of r is denoted by

$$X = F(x, r) = F'(m, r). \tag{3.4}$$

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

$$Y = D(m, x, r, E), \tag{3.5}$$

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

$$CT = (W, E, Y), \tag{3.6}$$

and $m = f(x)$ can be verified if the signature in (3.2) is valid, and (3.7) is satisfied:

$$G(m, X, E, Y) = \text{“yes”}, \tag{3.7}$$

where $G(\cdot)$ is the *verifier function*. x is decipherable if and only if either of the following conditions holds:

DU1: (E, Y) and r are available, for Y produced from (x, r).

DU2: (E, Y) and (E', Y') are available where Y and Y' are produced from the same (x, r) and $E \neq E$'.

The deciphering condition DU1 is based on the fact that r is a symmetric key that encrypts/decrypts x to/from (E, Y), and DU2 is based on the (2, k) secret sharing scheme [28], where k is an integer greater then two. DU2 is an important condition for the TZKP protocol design. Given the above facts, anonymity control of service consumers can be implemented by a simple *time-stamping method*: U first withdraws a token TK from the CA. From TK, U sends m to the service provider as the request to schedule a session for service redemption. The service provider grants a session time t to U by a TAA message:

$$TAA = (sign_{SP}(t, m), t), \tag{3.8}$$

where $sign_{SP}(\cdot)$ denotes the digital signature signed by the service provider. If the service provider provides various types of services, the type of services will also be included in the signature, which is not shown in (3.8) for simplicity. Later, when U wants to redeem the services, it adds TAA to the first move of protocol, and executes the second and the third moves as usual. Besides verifications in the original protocol, the service provider also needs to verify $sign_{SP}(t, m)$ at session t. The main weakness of this scheme is that TK cannot be reused for different sessions. The proposed TZKP allows reuse of a token with protected anonymity for rule-abiding consumers so that the number of withdrawals of new tokens can be drastically reduced.

3. TAA generation: session time authorization

TAA contains a service provider's signature on the public part of the requesting consumer and the session time t authorized for the consumer to redeem the services. The consumer who is granted the TAA is eligible to use its token to redeem the services once within the period described by t. Requesting service redemption more than once within the period t will lead to double-spending of the token in GDA. The consumer's identity will be involuntarily deciphered from the involved credentials. The credentials stored at the service provider can be discarded after t because the redemption of the same *TAA* requested after t will be rejected by the service provider. In contrast, without the notion of time in GDA (or $t = \infty$), the service provider will need to store credentials indefinitely to assure identification of possible service consumer who tries to redeem the same TAA again in the future.

Now, we expand the scenario to consider a service consumer who is granted two TAAs bundled with sessions, t_1 and t_2, respectively. In GDA, the consumer will need to use two tokens to receive these two TAAs. Otherwise, when the consumer passes these TAAs to others, the same token will be used to create the credentials which by definition is a double-spending of token. If the service provider intentionally stores the expired credential after t_1 ($t_1 < t_2$), then the service provider is able to decipher the consumer's identity after collecting another credential at t_2. The anonymity of service consumer is compromised because it never commits a double-transfer violation. To assure anonymity

of rule-abiding service consumers using GDA, a new token must be withdrawn from the CA for each TAA being granted, making unnecessarily high system overheads.

One of the design goals of TZKP, in addition to the reduced credential storage at the service provider, is to allow proper reuse of token for redeeming services at different sessions so that the number of withdrawals can be greatly reduced. The challenging issue is how to do it without sacrificing the anonymity of rule-abiding service consumers and accountability on double-transfer of TAA. TZKP considers the signed session time t as an additional decipherability control variable so that the deciphering condition on the identity of service consumer depends not only on whether the token is reused (doubly spent) but also whether the reuse refers to the same TAA (doubly transferred).

Recalled the GDA described in the previous section, X and its private counterpart (x, r) are control variables co-produced by the CA and the consumer in the withdrawal protocol. x, which contains the consumer's identity, can be deciphered if a token is spent twice. Since X is determined at withdrawal, if a token is spent twice, the same X must be used to produce the two credentials, or otherwise the verification in (3.7) will fail. To make the token reusable for multiple sessions, TZKP takes X out of the token from the withdrawal protocol. The consumer must produce a new X value (using the same x but different r) together with the service provider for each requested session t so that each jointly produced value of X can be used one time only in the requested session, without causing deciphering of x. By producing different values of X for different sessions, the consumer does not need to withdraw new tokens. On the other hand, since t and X are bundled together by the service provider's signature, redeeming services within t more

than once implies that (t, X) is used more than once to produce credentials. As a result, the consumer's identity can be deciphered from the two credentials, just like the original GDA model. The new definition of the modified token **TK** = (**W**, **K**) in TZKP is:

$$\mathbf{W} = (b\text{-}sign_{CA}(m), m), \tag{3.9}$$

and

$$\mathbf{K} = (b\text{-}sign'_{CA}(x), x), \tag{3.10}$$

where the modified parameter is denoted by bold face. In contrast to (3.2) and (3.3), X and r are separated[2] from **W** and **K**, respectively. Now U is free to select different values of r and X after withdrawal of **TK**. Each time when U requests a session t from the service provider, U presents a unique X value and the message m from its token in the request. The service provider authorizes session t by signing a TAA message

$$\mathbf{TAA} = (sign_{SP}(t, m, X), t). \tag{3.11}$$

Note the difference of (3.11) and (3.8) that X is included in **TAA** but not in *TAA*. When U redeems the services, it sends (**TAA**, **W**, X) in the first round of the three-move protocol and then executes the second and the third moves as usual. X is now an element of the modified credential

$$\mathbf{CT} = (\mathbf{W}, X, \mathbf{E}, Y). \tag{3.12}$$

[2] 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 [36] is plugged in to GDA in [2].

The modification from **E** to E will be discussed in (3.21) later. The verifications on **CT** are identical to those in the time-stamping approach introduced in section 3.2.1, except that correctness of (t, m, X) is checked by

$$sign_{SP}(t, m, X) \text{ and } b\text{-}sign_{CA}(m), \tag{3.13}$$

instead of

$$sign_{SP}(t, m) \text{ and } b\text{-}sign_{CA}(m, X). \tag{3.14}$$

Anonymity of U is guaranteed provided that **CT** and **CT'** are produced by the same token **TK** in distinct sessions. Although **W** and **W'** have identical m, **TAA** and **TAA'** have distinct values of (t, X) and (t', X'). The same value of m implies that **CT** and **CT'** are produced by the same token but the distinct values of (t, X) and (t', X') imply that the reuse of the token does not constitute a double-transfer of TAA because they are produced 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 values of (x, r) and (x, r'). Based on DU2, x cannot be deciphered from **CT** and **CT'**.

It is straightforward to show that x can be deciphered when U doubly transfers a TAA in the same session. Therefore, it will not be discussed further.

4. TAA transfer: multi-source reusability (MSR)

For transfer of TAA, one could apply the GTM model to the GDA model, which suggests that a cascaded credential contains:

(i) A TAA message signed by the service provider,

(ii) The credentials produced by all consumers in the previous transfers of this TAA, and

(iii) The credential produced in the current transfer.

From (3.6) and (3.8), the cascaded credential in GDA is

$$KC_{\mathrm{i}} = (TAA, CT_1, CT_2, \ldots, CT_{\mathrm{i}}) \tag{3.15}$$

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, U_i sends KC_{i-1} and W_{i} in the first round of the three-move protocol. In the second round, $U_{\mathrm{i+1}}$ produces the challenge message as the hash value of its token public data, instead of a random number:

$$E_i = H(W_{i+1}), \tag{3.16}$$

where $H(\cdot)$ denotes 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 *TAA* and each credential in KC_i, and

(ii) verify the linkage between each adjacent credential pair in KC_i by (3.16).

This step assures 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 in each transfer because any reuse of token in GDA may compromise the anonymity of the grantor. We raise similar questions as in the earlier discussions:

- When does a reuse (double-spending) of token constitute a double-transfer of TAA?
- Can the consumer's identity be deciphered from credentials produced by the reuse of a token when no TAA is doubly transferred?

To answer these questions we propose the MSR condition which decides whether or not a reuse of token constitutes a double-transfer of TAA. Knowing their differences, we modify (3.16) and the cascaded credential format in (3.15) so that withdrawal of new token can be eliminated when each received TAA is transferred once only. Anonymity of the consumer needs to be protected in this case because the total amount of privileges carried by the received TAAs does not increase, *i.e.*, no double-transfer. Based on MSR, deciphering condition on the consumer's identity is not only determined by the session time, as discussed in the precious section, but also determined by where the TAA comes from, *i.e.*, the *source* of the TAA. The modified cascaded credential becomes:

$$\mathbf{KC_i} = (\mathbf{TAA}, \mathbf{CT}_1, \mathbf{CT}_2, \ldots, \mathbf{CT}_i). \qquad (3.17)$$

Despite the similarity between (3.15) and (3.17), one must note that X_1 is signed in **TAA**, but not in *TAA*. X_j is contained in W_j of CT_j, but not in $\mathbf{W}_j$ of $\mathbf{CT}_j$. Given two cascaded credentials resulted from the double-transfers of TAA, the identity of consumer who made double-transfers can be deciphered based on DU2, using the two credentials positioned right after their longest common prefix. For the scenario depicted in Fig. 6, the identity of the double-transfer offender U_C can be deciphered from $\mathbf{CT}_C$ and $\mathbf{CT'}_C$.

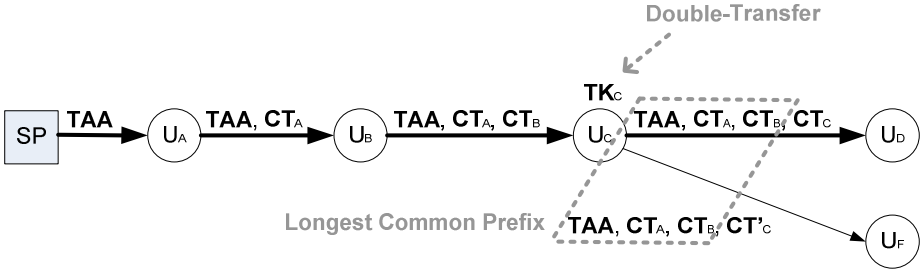


Fig. 6: Same-source transfers – a double transfer.

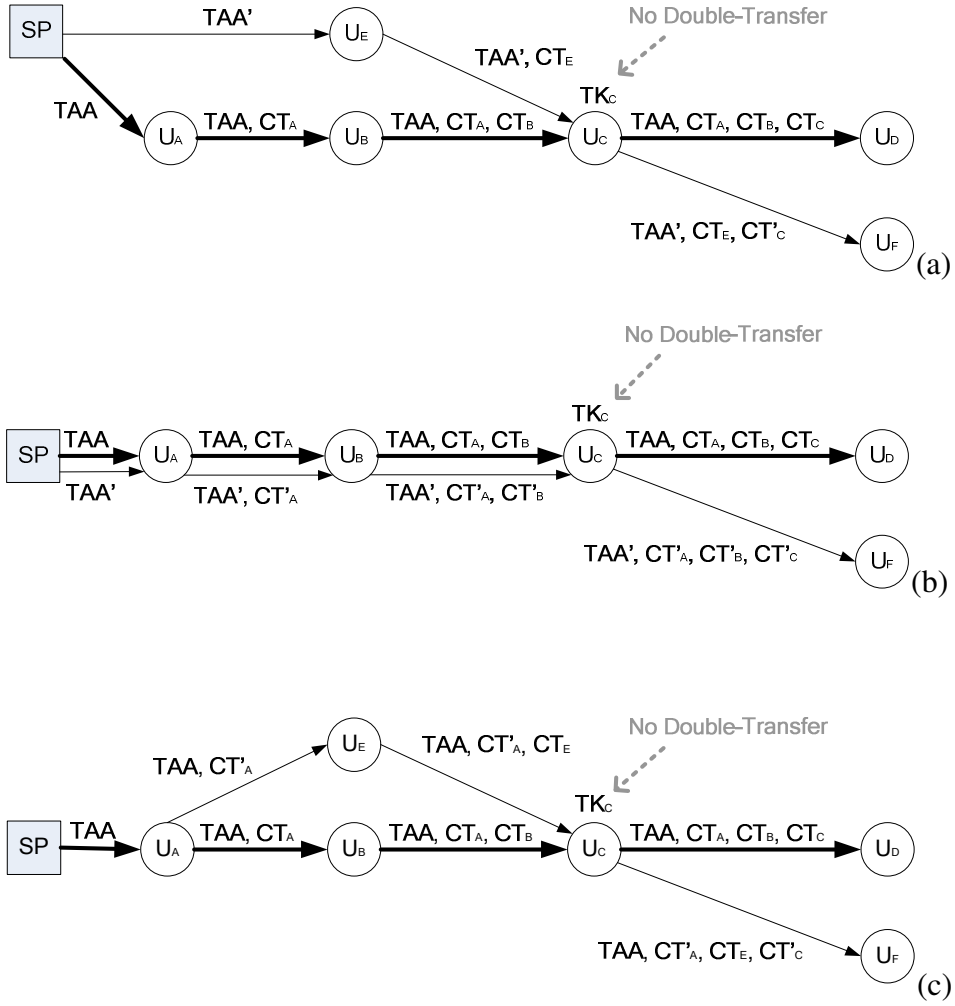


Fig. 7: Multi-source transfers – not a double transfer.

Fig. 7a to Fig. 7c depict several scenarios that do not constitute a double-transfer. In Fig. 7a, 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. 7b, 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. 7b, where U_C receives two TAAs from the same grantor but they are meant for distinct session times (**TAA** ≠ **TAA**'). Same-source transfers are also different from same-TAA transfer, as depicted in Fig. 7c, where U_C may not realize that it is transferring the same (copy of) TAA. U_C does not constitute a double-transfer offense in this case but U_A does. The cryptographic constructs should allow deciphering of the identity of U_A but not U_C. We achieve this, we define the notions of same-source and multi-source transfers as follows:

Definition 3.1: Two TAAs transferred from U_C, to U_D and U_F, are from the *same-source* if, and only if

$$\mathbf{KC}_{\mathrm{D}} = (\mathbf{TAA}, \mathbf{CT}_1, \ldots, \mathbf{CT}_{\mathrm{i}}, \mathbf{CT}_{\mathrm{C}}), \tag{3.18}$$

and

$$\mathbf{KC}_{\mathrm{F}} = (\mathbf{TAA'}, \mathbf{CT'}_1, .., \mathbf{CT'}_{\mathrm{j}}, \mathbf{CT'}_{\mathrm{C}}) \tag{3.19}$$

have the common prefix

$$(\mathbf{TAA}, \mathbf{CT}_1, \ldots, \mathbf{CT}_{\mathrm{i}}) = (\mathbf{TAA'}, \mathbf{CT'}_1, \ldots, \mathbf{CT'}_{\mathrm{i}}), \tag{3.20}$$

for $i \leq j$ without loss of generality. Otherwise, they are from the *multi-source*.

In summary, same-source transfer pattern constitutes a double-transfer violation while multi-source transfer pattern does not. A consumer should be allowed to reuse its token to make multi-source transfers with protected anonymity. Based on the analysis, we revise the three-move ZKP protocol so that the same-source transfers will guarantee identification of the double-transfer violator but the multi-source transfers will assure the consumer staying anonymous.

We attack this problem based on similar technique similar 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 the service provider when the TAA is transferred from U_i to U_{i+1}. To overcome this, X_i is bundled with the grantor's challenge message $\mathbf{E}_{i-1}$ as follows:

$$\mathbf{E}_i = H(Y_{i-1}, \mathbf{W}_{i+1}, X_{i+1}). \tag{3.21}$$

where $Y_0 = 0$ by default. 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 challenge message when U_i receives TAAs from different sources. When U_i passes on its TAAs, it can use different values of r_i to produce its responses. The anonymity of U_i is protected because x cannot be deciphered by DU2. On the contrary, if U_i offenses in multiple (same-source) transfers, it will be forced to use the same value of r_i (and hence X_i) when it passes on the TAAs. Otherwise, verifications in (3.21) will fail. The identity of U_i can be deciphered based on DU2, just like the GDA model.

Note that, in additional to X_{i+1}, Y_{i-1} is also added to the hash inputs of (3.21). 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.

B. Security Analysis

The major equation changes from GDA to TZKP are (3.8) to (3.11) and (3.16) to (3.21). The security analysis in this section will explain how such changes can guarantee

accountability to double-transfer violators, anonymity of rule-abiding service consumers, and authenticity of cascaded credentials under collusion attacks of malicious consumers.

1. Accountability

First, we will show that identification of double-transfer violators is guaranteed in GDA. Then, we will show how it can be guaranteed in TZKP via a different way to engage X_i. Suppose KC_i is doubly transferred by U_{i+1} in GDA. (m_j, X_j), for $j = 1, ..., i$, cannot be forged because $b\text{-}sign_{CA}(m_j, X_j)$ is a secure signature. (t, m_1) cannot be forged because it is signed by the service provider in (3.8). Based on this, and $b\text{-}sign_{CA}(m_1, X_1)$, (m_1, X_1) is guaranteed intact with the TAA. Let (m_j, X_j) be intact. We want to show that (m_{j+1}, X_{j+1}) is also intact. In other words, if (m_{j+1}, X_{j+1}) is used to receive a TAA, which KC_i guarantees that (m_j, X_j) is intact, then U_{j+1} is not able to compute another (m'_{j+1}, X'_{j+1}) to transfer this TAA without failing any tests. The proof contains three parts. First, given $CT_j = (W_j, E_j, Y_j)$, U_{j+1} is not able to compute another E'_j such that (W_j, E'_j, Y'_j) passes the test in (3.7). It is because U_{j+1} has no knowledge on (x_j, r_j) to produce Y'_j by (3.5). Second, U_{j+1} is not able to compute another m'_{j+1} to produce E_j, which passes the test in (3.16). It is because $H(\cdot)$ is collision resistant. Third, U_{j+1} is not able to compute another X'_{j+1} which produces $b\text{-}sign_{CA}(m_j, X_j)$. It is because the signature is secure. Based on the above arguments, and by mathematical induction, we conclude that U_{i+1} must use the same (m_{i+1}, X_{i+1}), and so the same (x_{i+1}, r_{i+1}), in both double-transfer instants. As a result, x_{i+1} can be involuntarily deciphered based on DU2.

Next, we will apply similar arguments to TZKP. Suppose $\mathbf{KC}_i$ is doubly transferred by U_{i+1}. Instead of using $b\text{-}sign_{CA}(m_j, X_j)$ to guarantee the integrity of (m_j, X_j) as in GDA, the integrity of (m_1, X_1) with the TAA is guaranteed by the service provider's signature in (3.11) instead. Let (m_j, X_j) be intact. We want to show that (m_{j+1}, X_{j+1}) is also intact. The proof has two parts. First, given that $\mathbf{CT}_j = (\mathbf{W}_j, X_j, \mathbf{E}_j, Y_j)$, U_{j+1} is not able to compute another $\mathbf{E'}_j$ such that $(\mathbf{W}_j, X_j, \mathbf{E'}_j, Y'_j)$ passes the test in (3.7). It is because U_{j+1} has no knowledge on (x_j, r_j) to produce Y'_j by (3.5). Second, U_{j+1} is not able to compute another (m'_{j+1}, X'_{j+1}) to produce $\mathbf{E}_j$ that passes the test in (3.21). It is because $H(\cdot)$ is collision-resistant. Since the association between m_{j+1} and X_{j+1} has been verified in this step, there is no third step in this proof. Based on the similar arguments as in the proof for GDA, U_{i+1} must use the same (m_{i+1}, X_{i+1}), and so the same (x_{i+1}, r_{i+1}), in both double-transfer instants. Based on DU2, x_{i+1} can be involuntarily deciphered.

2. Anonymity

Next, we will discuss how X_{i+1} in (3.21) can allow reuse of token for multi-source transfers with protected anonymity. Suppose that U_{i+1} uses the same token to receive two TAAs from multi-source, which cascaded credentials are $\mathbf{KC}_i$ and $\mathbf{KC'}_i$. Since the same token is used, the same m_{i+1} value will also be used. But different values of $\mathbf{E}_i$ and $\mathbf{E'}_i$ can be used 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}) to receive one TAA, and (m_{i+1}, r'_{i+1}) to receive the other one. When U_{i+1} passes on the two TAAs with $\mathbf{KC}_i$ and $\mathbf{KC'}_i$, different r_{i+1} and r'_{i+1}

are used to produce $\mathbf{CT}_{i+1}$ and $\mathbf{CT'}_{i+1}$, respectively, so based on DU2, x_{i+1} cannot be deciphered.

3. Authenticity

Finally, we explain why Y_{i-1} is needed in (3.21) to prevent credential forgery. So far X_i is considered a random value selected by U_i when the TAA is transferred from U_i to U_{i+1}. And X_i is bundled with $\mathbf{E}_{i-1}$ of U_{i-1}, "before" it is used to produce $\mathbf{CT}_i$. As we have just shown, the ability in tracing double-transferring violators is equivalent to that of [1, 2] because they are identical except that integrity of X_i is protected by different means: (3.8) to (3.11) and (3.16) to (3.21). Nevertheless, when collusion is considered, $\mathbf{CT}_i$ and $\mathbf{KC}_{i+1}$ can possibly be forged by a careful assignment of X_i before it is bundled with $\mathbf{E}_i$. Followings we describe how such a forgery attempt is possible without Y_{i-1} in (3.21), 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 the challenge $\mathbf{E}_i = H(\mathbf{W}_{i+1}, X_{i+1})$ to U_i. Then, U_i arbitrarily selects Y_i. Given the inverse function of $G(\cdot)$, U_i derives an X_i which satisfies $G(m_i, X_i, \mathbf{E}_i, Y_i)$ = "yes", where m_i is from a valid token of U_i. When U_{i-1} transfers the TAA to U_i, U_i sends to U_{i-1} the challenge message $\mathbf{E}_{i-1} = H(\mathbf{W}_i, X_i)$ in the second round of the three-move protocol. Then, U_{i-1} replies by Y_{i-1} as usual. Now, U_i has all data available to forge a credential $\mathbf{CT}_i = (\mathbf{W}_i, X_i, \mathbf{E}_i, Y_i)$, and so the cascaded credential $\mathbf{KC}_i$ to U_{i+1}. U_{i+1} can transfer $\mathbf{KC}_i$ to U_{i+2} without anomaly detected.

Since Y_i is selected by U_i arbitrarily without any encrypted data of its identity included, any violations done by U_i will not be identified.

To prove that Y_{i-1} in (3.21) can prevent such a credential forgery under collusion attacks, we introduce a prior-knowledge graph analysis as depicted in Fig. 8a - Fig. 8c.

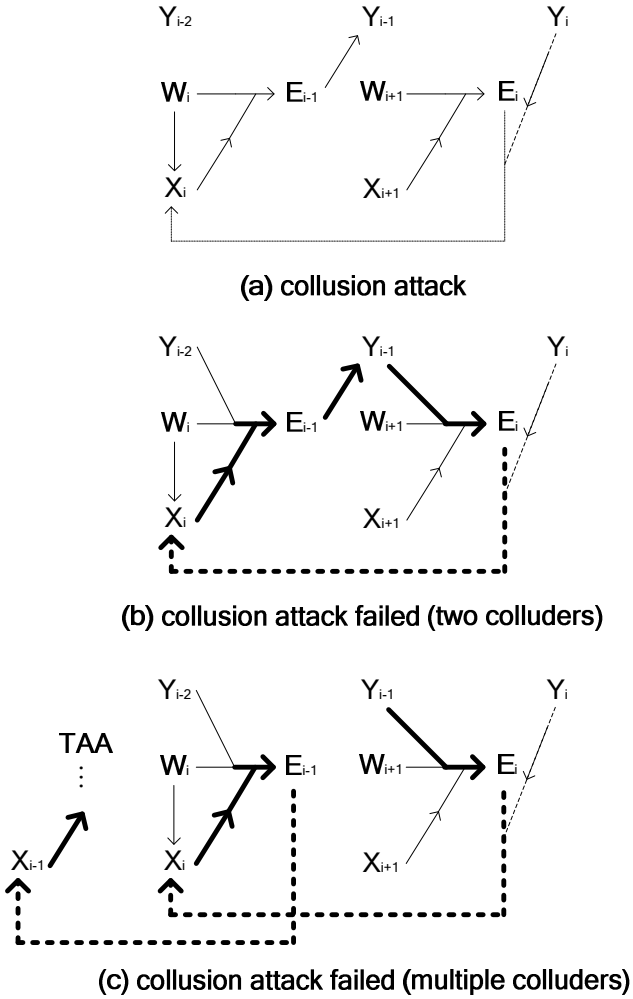


Fig. 8: Prior-knowledge graph analysis for TZKP.

The arrow pointing from P to Q implies that the creation of Q requires the prior knowledge of P. The solid line denotes the dependency when the three-move protocol is executed in a 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,

$$(P_0, P_1, \ldots) \rightarrow (Q_0, Q_1, \ldots), \tag{3.22}$$

to denote that the creations of all parameters in Q_0, Q_1, …, require the prior knowledge of some parameters in P_0, P_1,…, where P_i and Q_i are simply data in the three-move ZKP protocol or themselves the dependency formulas. The dependencies are transitive, *i.e.*,

$$(P \rightarrow V \text{ and } V \rightarrow Q) \text{ implies } P \rightarrow Q. \tag{3.23}$$

Fig. 8a depicts the case when Y_{i-1} is removed from (3.21). $\mathbf{CT_i} = (\mathbf{W}_i, X_i\ \mathbf{E}_i, Y_i)$ can be forged by creating parameters in the following sequence:

$$(((\mathbf{W}_{i+1} \rightarrow X_{i+1}) \rightarrow \mathbf{E}_i), Y_i, \mathbf{W}_i) \rightarrow X_i. \tag{3.24}$$

From (3.24), all parameters to forge $\mathbf{CT}_i$ 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. 8b and Fig. 8c depict the cases when Y_{i-1} is included in (3.21). In Fig. 8b, we only consider the collusion of U_i and U_{i+1}, where the sequence of parameter creations is as follows:

$$(((((Y_{i-1},(\mathbf{W}_{i+1} \rightarrow X_{i+1})) \rightarrow \mathbf{E}_i), Y_i, \mathbf{W}_i) \rightarrow X_i), \mathbf{W}_i, Y_{i-2}) \rightarrow (\mathbf{E}_{i-1} \rightarrow Y_{i-1}). \tag{3.25}$$

From (3.25), a dependency loop (in bold lines of Fig. 8b) is formed, which means that the creation of Y_{i-1} requires the prior knowledge of Y_{i-1}, which has a contradiction. The adversary has nowhere to initiate the malicious action, so the forgery attempt fails. Fig. 8c 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 $\mathbf{E}_{i-1}$, and then closes the loop. For $j = 1$, no dependency loop is formed, but X_1 is signed by the service provider in **TAA**, so the forgery attempt fails again.

C. Complexity Analysis

The time and message size are measured based on three operations: withdrawal of tokens, transfer of cascaded credentials and detection of double transfers as described below:

T_w = the total computation time for withdrawals

S_w = the total communication message size for withdrawals

T_f = the total computation time for transfers

S_f = the total communication message size for transfers

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 service provider grants p TTAs to a consumer. Each is legally transferred through the same set of q consumers before it is redeemed from the service provider. Table 2 is the summary of the complexity analysis. In GDA, since the reuse of a token is prohibited, the total number of withdrawals is the total number of transfers in the system, we have

$$T_w = O(p \cdot q) \text{ and } S_w = O(p \cdot q). \tag{3.26}$$

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.*,

$$T_w = O(q) \text{ and } S_w = O(q). \tag{3.27}$$

Table 2: Complexity analysis for TZKP.

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

For transfers of cascaded credentials the message size increases by one credential after each transfer in GDA and TZKP. The growing size also increases the computation time for the cascaded credentials. It is our desire to eliminate the cumulative overheads, but this has been proven inevitable [1]. Intuitive reason for this is that, every anonymous consumer along a series of transfers can be potentially a double-transfer offender. When the authorization is transferred, the consumer has to contribute part of its identity to the authorization data before it is circulated back for central inspection for double-transfer.

Therefore, regardless of the token, credential and protocol designs, the total transfers in the system have the following complexity:

$$T_{\mathrm{f}} = O(p\cdot(1 + 2 + \ldots + q)) = O(p\cdot q^2) \tag{3.28}$$

and

$$S_f = O(p\cdot q^2). \tag{3.29}$$

In spite of the same complexity formula used in (3.28), it has subtly different implications to GDA and TZKP. In GDA, q is unbounded because the authorization will never expire, and it can be transferred indefinitely before service redemption. In contrast, in TZKP, we have

$$q < T_e/((1 + 2 + \ldots + q)\cdot T_{ct}) < T_e/(q\cdot T_{ct}) \tag{3.30}$$

$$\Rightarrow q < (T_e/T_{ct})^{1/2}, \tag{3.31}$$

where T_e denotes the period starting from the authorization is granted from the service provider 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:

$$T_w = O(1),\ \ S_{\mathrm{w}} = O(1), \tag{3.32}$$

and

$$T_f = O(p),\ \ S_f = O(p). \tag{3.33}$$

To analyze the complexity for double-transfer detections, 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. To guarantee identification of double-transfer offenders in GDA, the credentials cannot be discarded once they are received. Therefore, we have

$$S_d = O(p \cdot q), \tag{3.34}$$

and

$$T_d = O(\log p \cdot q). \tag{3.35}$$

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

$$S_d = O(q), \tag{3.36}$$

and

$$T_d = \mathrm{O}(\log q). \tag{3.37}$$

Again, if (T_e/T_{ct}) is a constant, then it can be further optimized to:

$$S_d = O(1) \tag{3.38}$$

and

$$T_d = O(1). \tag{3.39}$$

D. Experimental Results

We implemented TZKP protocol on top of the software architecture of *CREAT* (Cybersecurity Remote Education Access Tool), which binary version for the Windows operating system is available for download [53]. By plugging Ferguson's e-coin scheme

[36] into GDA, with the modifications made to implement TAA and MSR in TZKP, we implemented the CA, the service provider, and the service consumer modules, and tested them on the DETER testbed [54]. As the system architecture depicted in Fig. 5, the CA node issues tokens to the consumer nodes. The service provider node grants TAAs to the consumer nodes. The consumer nodes request tokens from the CA, request TAAs from the service provider, redeem services from the service provider at scheduled time, and transfer the TAAs to other consumers. Key generation and other modular arithmetic are computed by the big integer library [55]. SHA-1 is used for one-way computations. A 1024-bit RSA scheme is implemented for signature and other usages.

To measure the run-times and the message sizes of withdrawals and transfers, we repeat each experiment by 1000 runs and take the average values. The CA node 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. 1024-bit RSA is considered the standard key strength for contemporary technologies. 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 the administrative boundary. The limitation is further affected by the duration of session. As such, one can expect a small number of transfers in each session.

Table 3: Runtime and message size of withdrawal.

Metrics	Values
Total runtime (sec)	4.0
Computation time (sec)	3.5
Transmission time (sec)	0.5
Token size (KB)	1.76

Table 4: Runtime of the i-th transfer.

i	1	5	10	15
Time (sec)	1.7	7.2	12.9	19.1

Table 5: Message size of the i-th transfer.

i	1	5	10	15
Size (K bytes)	2.17	9.87	19.5	29.1

Table 6: Upper bounds on credential storage size.

T_e	60	120	240	480
Size (K bytes)	< 19.53	< 26.04	< 39.06	< 54.25

One can use the 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 storage overheads for credentials, identification of double-transfer offender in traditional ZKP protocol also significantly slows down with the number of spent tokens.

Other statistics on the runtime and message size of TZKP are depicted in Fig. 9 and Fig. 10. 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.

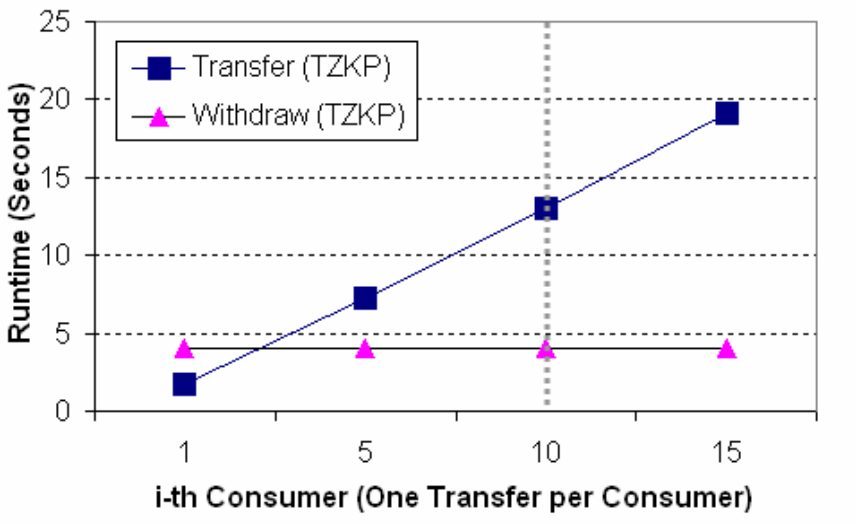


Fig. 9: Runtimes of withdrawal and transfer in TZKP.

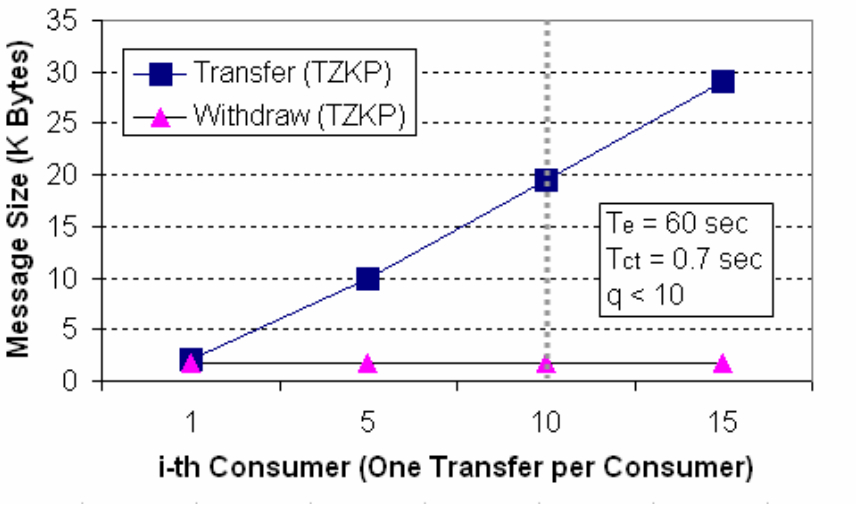


Fig. 10: Message sizes of withdrawal and transfer in TZKP.

E. Related Work

ZKP protocol was 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* [56] considers time management in ZKP. It ensures the protections of "proof" and "zero-knowledge" when multiple ZKP instances are executed sequentially or in parallel. To guarantee such protections, concurrent ZKP requires both the prover and the verified to contribute some random numbers in each instance of the ZKP protocol and verify the consistency of the exchanged data, which is similar to the technique we adopted in TZKP. On the other hand, the hidden knowledge in concurrent

ZKP is protected unconditionally. It does not revoke the hidden knowledge such as the user identity on double-transfer violation events.

E-cash systems [1] consider anonymity protection of the rule-abiding users and identification of double-transfer violators. Some E-cash systems support transferability and divisibility which are useful features for secure resource management. Nevertheless, E-cash does not have proper time management. As a result, each token can only be used one-time and all credentials have to be stored indefinitely, leading to severe withdrawal and storage overheads.

Uncloneable group identification [57] introduces the notion of time management by associating session time with the random numbers engaged in the ZKP protocol. This technique is similar to the one we used in TZKP. And it also guarantees the revocation of anonymity on double spending events within the same session. However, uncloneable group identification does not consider transfers of authorization further from the receiver to another user. To do an authorization transfer, the user has to redeem the authorization from the central authority first. In contrast, TZKP can keep transferring the authorization from one another while the central authority remains offline.

Proxy signature [58-63] is useful for transfers of authorizations from one to another without the online participation from the central authority. However, it usually requires the individual public key from every intermediate consumer (proxy signer) for signature verification. Verification of individual public keys could be expensive and may induce linkability of identity information if each user is associated with one public key

only. In contrast, TZKP only requires the users to verify the public key from the central authority.

Group signature [64-69] is useful for verifying group membership without knowing the individual identity. However, the signature is usually not transferable, and the anonymity revocation is unconditionally controlled by the CA. In contrast TZKP allows the peer nodes to transfer and verify the credentials without the prior knowledge on the identity of each other, while it can still decipher the identities of double-transfer violators.

F. Summary

In this section, we proposed a timed ZKP (TZKP) protocol on the basis of GDA to support session-based access of computing resources for anonymous consumers. In the classical GDM, a consumer can transfer its access authorization to another consumer without notifying the service provider but each transfer instance requires a spending of one consumer token, which contains the encrypted identity of the consumer, making this desirable feature costly. To minimize the overhead to withdraw new tokens, we propose the multi-source reusability (MSR) condition which allows a consumer to reuse its token for multiple transactions with protected anonymity unless a double-transfer of access authorization occurs. Furthermore, we propose the notion of timed access authorization (TAA) so that the service provider can eliminate the need to store and keep track of the spent tokens for double-transfer violation once their marked sessions are expired. TZKP

protocol prevents the service provider from compromising the anonymity of an honest consumer via misuse of expired spent tokens intentionally stored at the service provider, while drastically reducing the system overhead by allowing proper reuse of a token in different timed sessions. Implementation of TZKP is evaluated on the DETER testbed. The running time for computers with modest resources was found to be quite reasonable.

The linkability of credentials deserves further investigations in the future. In the current construct, the credentials created by the reuse of token are linkable because they identical **W**. There is a tradeoff for the service consumer to choose fewer overheads with more reuse of token or less linkability by withdrawing more tokens. A possible solution to achieve unlinkability of credentials and reusability of token at the same time could be plugging unlinkable E-cash schemes such as [70] into our secure resource management framework. But the problem is that in the unlinkable scheme, **W** is not available to be an input of $H(\cdot)$ to produce the challenge message **E**. One candidate to replace **W** is the data used to detect double-spending in the unlinkable scheme because what we need is just to make sure that some common value (linkable) to be used on double-transfer offense and they could be different (unlinkable) on the rule-abiding case. Careful mapping from the unlinkable scheme in [70] to the secure resource management framework and its security analysis are needed in the future research.

IV. DIVISIBILITY MANAGEMENT

Divisibility refers to the ability for stakeholders to divide the assets into portions to render services. It is at the center of investigation for dispensing of electronic credits in E-cash. It is also the main focus on optimal allocation of digital resources. However, there is no discussion on the relationship between the two types of designs in literatures. In this section, we will focus on the divisibility management of system resources and user credits. In E-cash, an *N-divisible* token [38] allows a service consumer to engage several transactions anonymously using the same token until the total spending amount reaches the quota limit *N*. When using the *N*-divisible token to access computing resources, the tracking scheme for spending records should be compatible with the resource allocation protocols so that both the performance goals and the security goals can be harmonized.

Existing *N*-divisible tokens are either *coupon-based* or *tree-based*. In the coupon-based schemes [37, 39, 41, 45, 71], each credential produced by the token represents one unit of spent assets. The spending patterns are simply monitored by counting the number of credentials produced from the token. In a more sophisticated approach, a binary tree is used to keep track of spending patterns [2, 38, 40, 42], where a node located at the level *i* denotes $1/2^i$ units of the total asset. In the second part of [2], Eng-Okamoto developed a *general divisibility model* (*GDM*) which can transform the non-divisible token into its *N*-divisible counterpart for a large class of E-cash schemes [36,50-52] based on the *general disposable authentication* (*GDA*) model [2]. The work reported in this section does not

consider the first part of [2], which is a specific disposable authentication scheme based on Schnorr's identification [52]. It also does not consider the scheme presented in [49], which is a specific scheme prior to its development for divisibility.

In this section, we investigate how to design the hypercube based *N*-divisible token for computing resource allocations based on the GDA framework. Instead of using a tree as in [2], hypercube is chosen in this study because it is a widely used data structure for allocation of computing resources [3, 72, 73]. Being a superset of various data structures such as tree, mesh, and star, it spots some interesting insights on the interplay between the cryptographic constructs and the resource allocation rules. First, hypercube is a more flexible data structure that expands the possible divisibility configurations from $O(2^{n+1})$ to $O(3^n)$, comparing with the binary tree, where n is the bit length to represent an atomic unit of assets. However, the expanded divisibility configurations also create the new type of *shared-node* double-transfer violation pattern, in addition to the traditional *same-node* and *route-node* violation patterns. Tracking of the new violation type makes the analysis much more complicated than the tree based solutions. Therefore, we devised a *hypercube dependency graph* to track the spending patterns using both the top-down and bottom-up dependency analysis techniques. In contrast, only the latter approach is used for the tree based solution in GDA. From the analysis, we found that unrestricted sub-cube allocation schemes might cause leaking of identity, even without double-transfer offenses, unless a costly solution is considered. Such a leaking of identity information is called *anonymity hazard.* We found that anonymity hazard can be detected and avoided at small extra cost of *fragmentation* (<

2% in our simulation), without any restrictions on the sub-cube allocation schemes. Furthermore, two commonly used sub-cube allocation schemes are found to be immune (0%) from anonymity hazard because of their more restrictive allocation rules.

The rest of this section is organized as follows. Section A explains the details of the hypercube-based divisibility management framework. Section B analyzes the cryptographic constraints needed to assure the AAA requirements. Section C compares two cryptographic constructs to demonstrate the tradeoffs between security strength and performance cost when different sub-cube allocation rules are used. Section D presents the simulation results. Section E gives a summary of this section.

A. Hypercube Based Divisibility Management

An n-dimensional hypercube Q_n has 2^n nodes, and each node is connected to n neighbors. Each sub-cube is uniquely represented by an n-bit ternary string $p = p_{(1)} p_{(2)} \ldots p_{(n)}$, where $p_{(i)} \in \{0, \times, 1\}$, and "×" denotes a "don't care" bit. The number of don't care bit(s) in p is also the *dimension* of the sub-cube p, $dim(p)$. The shortest distance between sub-cubes p and q can be measured by their *hypercube distance:*

$$d(p,q) = \sum_{j=1}^{n} s_j \begin{cases} s_j = 1, \text{ if } p_{(i)} \neq q_{(j)} \text{ and } p_{(j)}, q_{(j)} \neq \times \\ s_j = 0, \text{ otherwise} \end{cases} \tag{4.1}$$

$d(p, q) = 0$ implies that p and q share some common node(s), and thus, committing both p and q will lead to double spending violation on their shared node(s).

Definition 4.1: A *dependency graph* $G = (V, E)$ is a directed graph, on which an edge $(p, q) \in E$ if, and only if, $dim(p) = dim(q) + 1$ and $h(p, q) = 1$, where $p, q \in V$ are vertices of G, and $h(p, q)$ is the Hamming distance [74] between p and q over the alphabets $\{0,\times,1\}$. p is the parent of q if $(p, q) \in E$. Similar nomenclatures follow the convention of tree.

G_n represents the dependency between all permissible sub-cube configurations of Q_n. To avoid ambiguity in subsequent discussions, we use "node" to describe an atomic node in Q_n and "vertex" to describe a node in G_n. Fig. 11 depicts the dependency graph of Q_2, where each vertex represents a sub-cube that can be derived from Q_2. A vertex on G_n is marked when a corresponding sub-cube is allocated by its corresponding token. A double spending violation occurs if any leaf vertex is allocated twice in two sub-cube allocations.

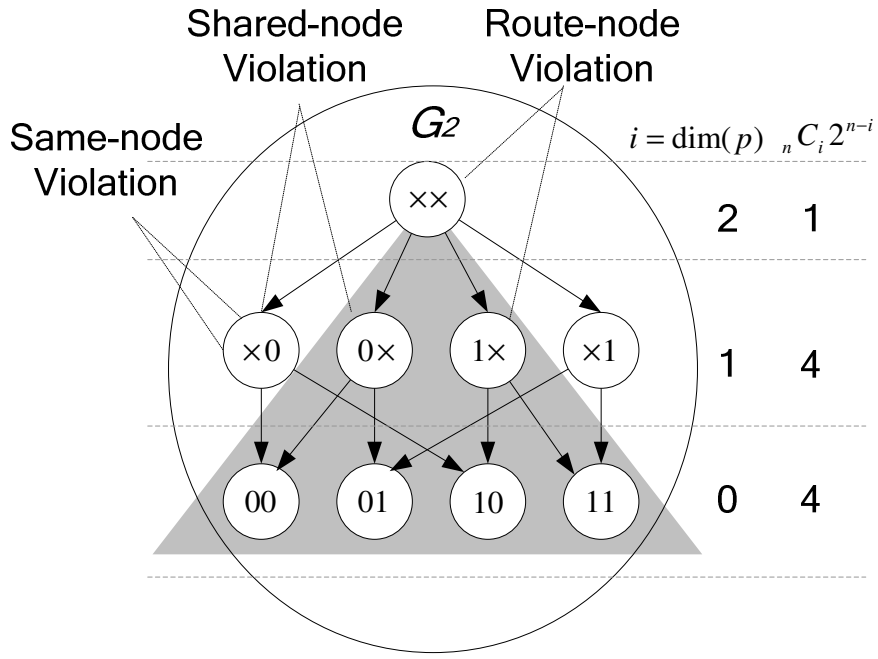


Fig. 11: Dependency graph of a 2-dimensional hypercube, G_2.

Note that the shadowed area highlights a binary tree embedded into G_2, implying that hypercube based *N*-divisible token is required to handle more complicated security conditions than its tree based counterpart. From the example in Fig. 11, it shows that three types of double spending violations, same-node violation, route-node violation, and shared-node violation, can occur to the hypercube based system. A same-node violation occurs if a vertex is spent twice. A route-node violation occurs if both a vertex and its ancestor/descendant vertex are spent. A shared-node violation occurs when two vertices, with no ancestor-descendant relationship but sharing one or more leaf vertices, are spent. The same-node and route-node violations were studied in tree based schemes but shared-node violation is a new type of violation identified in this work to be addressed.

Now, we outline some basic issues in resource allocations. Minimizing sub-cube fragmentation and locating the largest set of useable sub-cubes is the focus of many sub-cube allocation schemes. Compact representations of Boolean expressions are important to achieve performance goals but it may cause unintended double spending. For instance (0×, ×0) represents three nodes (00, 01, 10) and this is considered a legal allocation when they are allocated to one user. Nevertheless, spending both (0×) and (×0) terms in E-cash constitutes a double spending offense because node (00) is involved in two transactions. No double spending occurs if the set of nodes are allocated using one of the following three Boolean expressions: (0×, 10), (×0, 01), or (00, 01, 10), where redundant Boolean terms in sub-cubes are eliminated to prevent incorrect marking of the double spending patterns. Using node-by-node expression can avoid the anonymity protection issues but it requires the highest computation overhead, and so not considered further.

Following the GDA framework, we design the hypercube based divisible tokens, and derive the conditions to satisfy the AAA security constraints. Comparing with tree based solutions, applying the hypercube to the GDA framework requires more advanced analysis techniques because we need to track the new shared-node violation type also. Shared-node violation can occur in data structure whose dependency graph contains a multiple-child and multiple-parent structure. This situation is even more complicated for hypercube because of its highly connected constructs. For the DSI subsystem to identify offenders of all types of double spending violations, more information that can guarantee deciphering of the principal's identity under such conditions needs to be included in the credentials. However, doing so may lead to anonymity hazard which is situation when the principal's identity can be deciphered from multiple instances of sub-cube allocations even when no double spending occurs. An example will be given in Table 10.

Granted one could develop more sophisticated mathematical systems to eliminate anonymity hazards but a much more practical solution approach is based on the sub-cube allocation schemes because anonymity hazards can be recognized via simple rules. To gain a more realistic understanding for such a tradeoff analysis, three sub-cube allocation schemes, *random code* (*RC*), *binary code* (*BC*), and *binary reflected gray code* (*BRGC*) [3] are analyzed and simulated in this study. We will show that anonymity hazards appear in the rarely used RC but not to the widely adopted sub-cube allocation schemes, BC and BRGC.

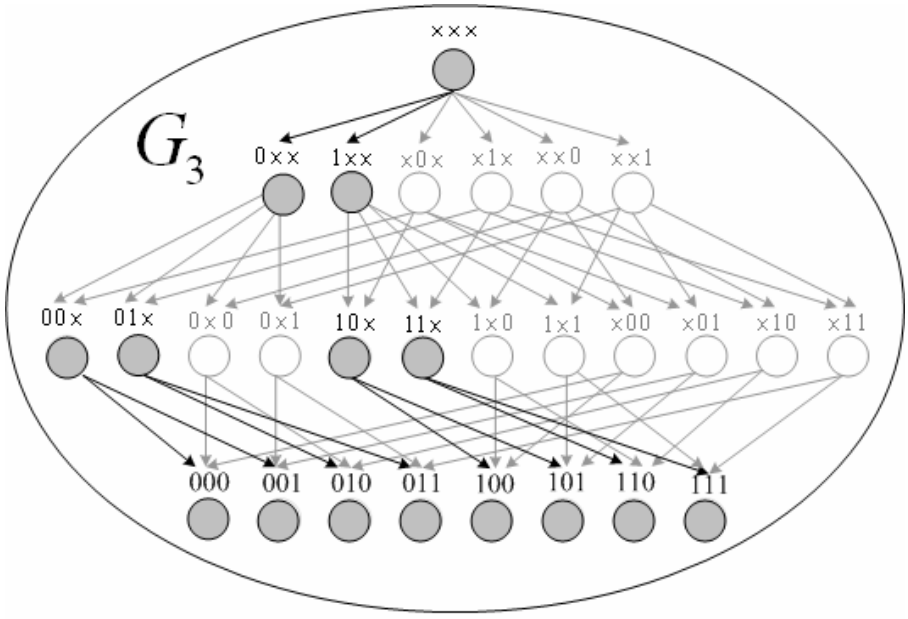
In RC, there is no restriction on which available sub-cube to use for the sub-cube request. In BC and BRGC, the hypercube topology is reduced to a linear list based on

the binary code and binary gray code order, respectively (see an example in Table 7). A sub-cube request is matched with the first sub-cube configuration on the list in a linear search order. In either case, to search for an available sub-cube Q_k is equivalent to find 2^k consecutive free nodes from node i to node $(i + 2^k - 1)$ in their corresponding allocation list. Their main difference is that, in BC, i needs to be a multiple of 2^k, while in BRGC, i only needs to be a multiple of 2^{k-1}. For details of BC and BRGC, please refer to [3][3].

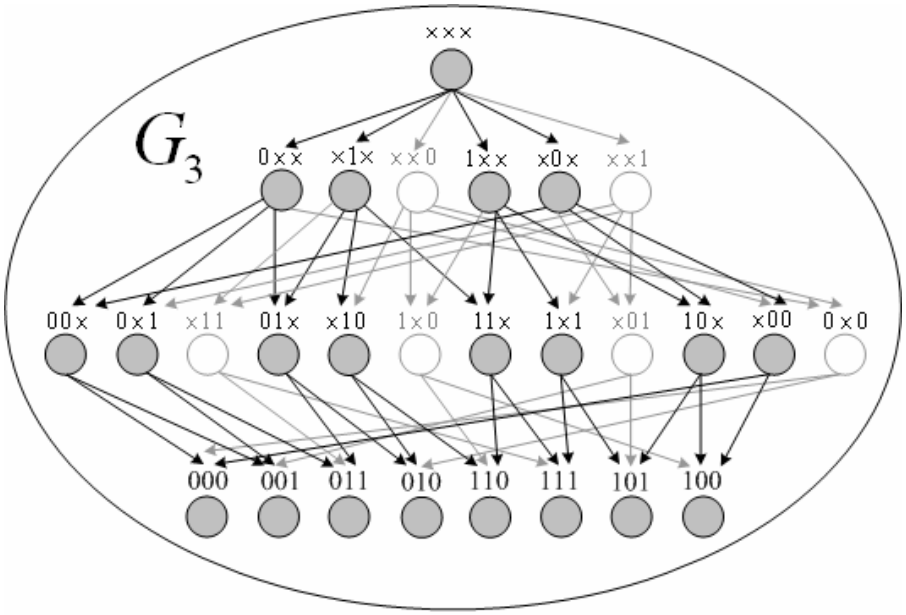
BC, BRGC, or other similar non-exhaustive sub-cube allocation schemes would utilize a fraction of available sub-cubes as the example depicted in Fig. 12a and Fig. 12b, where only highlighted vertices can be used in BC and BRGC, respectively. As shown in Fig. 12a, the vertices that can be used in BC are the vertices that can be used in the tree-based schemes. As shown in Fig. 12b, BRGC has more spending patterns than BC.

Table 7: Allocation list for Q_3 in BC and BRGC schemes.

i	0	1	2	3	4	5	6	7
BC	000	001	010	011	100	101	110	111
BRGC	000	001	011	010	110	111	101	100



(a) BC Scheme



(b) BRGC Scheme

Fig. 12: Usable vertices under (a) BC scheme (b) BRGC scheme.

1. System overview

The hypercube based divisibility management framework is depicted in Fig. 13, which is identical to the resource management framework depicted in Fig. 4. We repeat the figure here to highlight details related to divisibility management only. For other details, please refer to section II and section III.

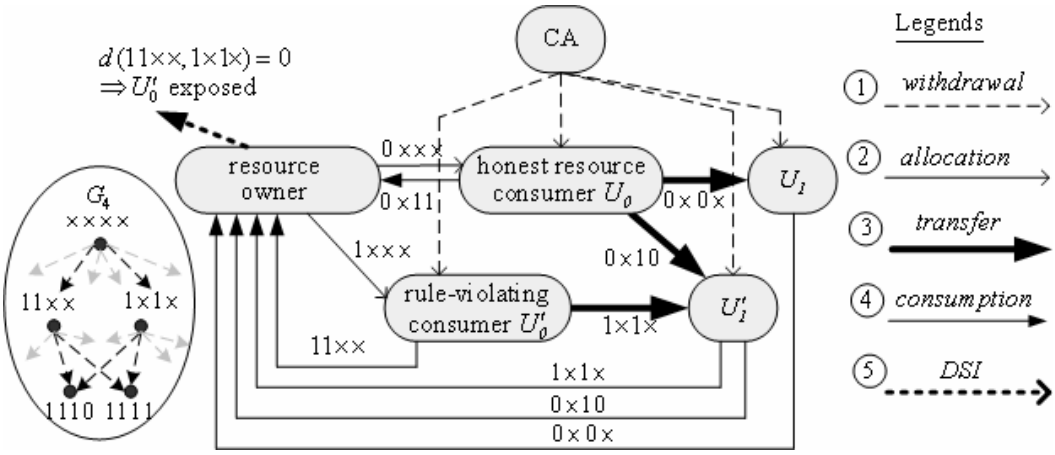


Fig. 13: System architecture for hypercube-based resource management.

In each transfer, the credential produced by the service consumers indicates the ownership of a sub-cube p, where p is the relative address to be interpreted in a series of credentials called *cascaded credential.* For the example in Fig. 13, 0×10 received by U'_1 comprises of three parts: (i) resource owner's signature on $p_1 = 0\times\times\times$ from the allocation between the resource owner and U_0, (ii) credential marked $p_2 = \times 10$ from the transfer of

U_0 and U_1', and (iii) credential marked $p_3 = \times$ from the consumption between U_1' and the resource owner. To keep track of spending patterns, the service consumer needs to select certain data derived from the token to be exposed in the credential. The selection of data is based on a dependency graph. Each vertex in the dependency graph is associated with three data: secret share (E, Y), delegation key r, and verification key X. The definitions and notations of these data follow the GDA model as described in Table 1 in section III. We repeat the table here in Table 8 for convenience of discussions.

Table 8: General disposable authentication (GDA).

AAA Requirements	Information Needed	Remarks
verify $m = f(x)$	VU1: X, (E, Y)	$X = F(x, r) = F'(m, r)$, $Y = D(m, r, x, E)$, and $m = f(x)$ iff $G(m, X, E, Y)$ = "yes"
decipher x	DU1: (E, Y), r	r = symmetric key to encrypt/decrypt x
	DU2: (E, Y), (E', Y')	(2, k) secret shares created by (x, r)
Keep x secret	Neither DU1 nor DU2 available	r = symmetric key to encrypt/decrypt x

Using the three types of data associated with the dependency graph, we proposed a *key dependency map* (*KDM*) to keep track of the correlation of the randomized key pairs between different vertices. When the sub-cube Q_x is transferred, a set of vertices will be selected to have their secret shares or keys released in the credential. The set includes the vertex which represents Q_x, and other vertices for sub-cubes, say Q_y, that is subject to the double spending offenses. This way, if Q_y is indeed spent in the future, DSI can identify the offending patterns. The constraints to select the secret shares and keys can be derived in a top-down or bottom-up fashion of the KDM, as it will become clear in section B.

2. Hypercube based token and credential

Following the GDA framework, the token is of the following format:

$$TK = (m, x), \tag{4.2}$$

Through the blind signature scheme [9], the CA is not able to trace the identity of the consumer by associating m or x with the withdrawal records. The information carried by a token TK and the hypercube dependency graph (that contains the spending patterns) should be integrated into the credential CT to enforce DSI, while protecting anonymity of the token owner. Following the high level constructs of GDA, the format of credential produced by hypercube based divisible tokens, in the transfer of a sub-cube p, is

$$CT = (p, m, DK(u(p)), VK(v(p)), VK(L), SS(s(p))). \tag{4.3}$$

DK(*V*) denotes the outputs produced from the publicly known function *DK* which takes a set of vertices *V* in *G* as inputs and produces the delegation keys of *V* as outputs. Similarly *VK*(*V*) and *SS*(*V*) denote the outputs from the publicly known functions *VK* and *SS*, respectively, which produce the verification keys and secret shares of *V* as outputs. *DK*(*V*), *VK*(*V*), and *SS*(*V*) are collectively called the *DVS* values. As it will become clear, when *p* is used in a transfer, other sub-cubes also need to be considered in the generation of DVS values. How to control the generations and exposures of DVS values to enforce DSI and protect the anonymity of rule-abiding users is the focal issue in the hypercube based divisibility management designs. Depending on the sub-cube usage patterns, some DVS values need to be put as a part of the credential while others should be kept by the token owner to avoid compromising the anonymity.

Exposure of DVS values is controlled by the constructs of *u*(*p*), *v*(*p*), *s*(*p*), and *L* in (4.2), and their interdependency relationships. *u*(*p*), *v*(*p*), and *s*(*p*), collectively called *exposure functions*, are publicly known functions that produce a set of vertices according to the input vertex *p*. Selected prior to a transaction, *L* is a set of reference vertices to be used for integrity check of credentials. Detailed constructs of the exposure functions and *L* are given in section 3. In subsequent discussions, we will use the term "directly exposed" to describe the keys included in the credentials, that is, *DK*(*u*(*p*)), *VK*(*v*(*p*)) and *VK*(*L*) in (4.3), without using KDM. We will use the term "indirectly exposed" to describe the keys derived from KDM, using the keys directly exposed from the credentials as the inputs.

Separating notations of vertex sets from their corresponding keys or secret shares is convenient for subsequent analysis which requires set intersection, union and negation operations. For example, to check whether or not the vertex p has both its delegation key and secret share exposed in the credential is equivalent to check whether or not $u(p) \cap s(p) \neq \phi$. Such a checking cannot be performed by the expression $DK(u(p)) \cap SS(s(p)) \neq \phi$ because the left-hand side and right-hand side of intersection are from different domains. Similar to the classical tree based schemes, a hypercube based divisible scheme needs to satisfy the following AAA requirements:

(A1) Authenticity: for any p being used for transfer/consumption, the condition $m = f(x)$ can be verified from the credentials of p without unveiling the plaintext of x.

(A2) Accountability: for any p and q, $d(p, q) = 0$, being used for transfer/consumption, x can be deciphered from the credentials of p and q.

(A3) Anonymity: for all vertices being used for transfer/consumption, if it does not exist p and q such that $d(p, q) = 0$, then x cannot be deciphered from their credentials.

Next, we will show how to satisfy the three properties using KDM to create details of the proposed divisible token, and proof of its correctness.

3. Key dependency map (KDM)

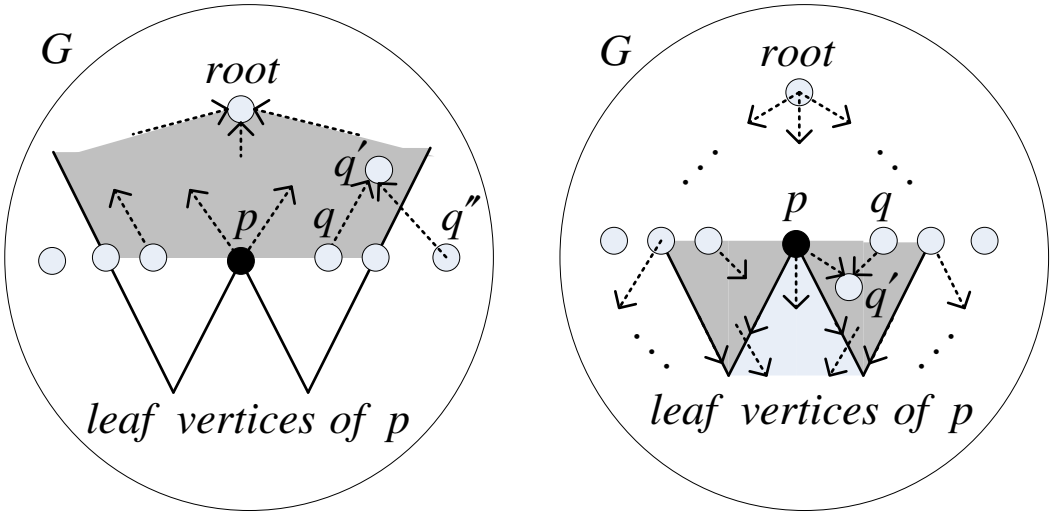
Recall that each vertex in G_n is associated with a delegation/verification key pair. KDM is a function that keeps track of the one-way mapping relationship among the key pairs of different vertices. When the sub-cube p is spent (assuming $p \in s(p)$), the keys of

the vertices directly exposed in a credential (defined by $u(p)$ and $v(p)$) will be used as the inputs of KDM, so that the delegation keys of some other vertices that are *susceptible* to double spending violation in the future can be indirectly exposed from the KDM outputs. In this way, when q is spent later, the secret shares exposed from q (because $q \in s(q)$) in the spending of q, together with its delegation key exposed from spending of p, can be used to decipher the identity of the double spending offender using DU1. Furthermore, if a vertex is spent twice its secret shares will be exposed twice on two different challenge messages and the double spending offender can be identified using DU2.

KDM should avoid exposing the delegation keys of *non-susceptible* vertices to prevent anonymity hazards. In order to supervise susceptible future spending, only some susceptible vertices (with respect to p) need to have the delegation keys exposed because $d(p, q) = 0 \Leftrightarrow d(q, p) = 0$. It does not matter whether $dim(p) > dim(q)$ or $dim(q) > dim(p)$. For example in the top-down KDM, the keys of the vertex p can only be derived from the keys of other vertices at dimensions higher than $dim(p)$. We only need to expose the delegation keys of susceptible vertices at dimensions lower than $dim(p)$. Similarly, in the bottom-up KDM, we only need to expose the delegation keys of susceptible vertices for dimensions higher than $dim(p)$. Exposure of keys for susceptible vertices are illustrated in the shaded areas in Fig. 14a and Fig. 14b.

Let p be a vertex in G_n that represents the sub-cube being used to track a transfer transaction. H is a publicly known collision-free one-way hash function. The symbol "||" is a concatenation operator between two ternary strings. $j = (n - dim(p))$ is the number of parents of p and $(p_1, \ldots, p_j)$ are the parents of p listed in ascending order. The ordering of

vertices can be computed by substituting "×" by "2", and order them as ternary numbers. For example, the ternary number of "0×1" is $7 = 0 \cdot 3^2 + 2 \cdot 3^1 + 1 \cdot 3^0$, and for "×10" is $21 = 2 \cdot 3^2 + 1 \cdot 3^1 + 0 \cdot 3^0$.



(a) Bottom-up (b) Top-down

Fig. 14: Key dependency map (KDM).

The building block for top-down KDM is given by the following equations:

$$r_p = H(X_{p_1} \| ... \| X_{p_j} \| p) \tag{4.4}$$

$$= H(F'(m, r_{p_1}) \| ... \| F'(m, r_{p_j}) \| p). \tag{4.5}$$

(4.4) follows the tree-based equation, $r_p = H(X_{left\text{-}child(p)} \| X_{right\text{-}child(p)})$, proposed in [2], with the following differences: (i) The hypercube-based design takes the verification keys from j parents (vs. 2 children) as inputs; (ii) An additional p is appended at the end of hash input in (4.4) to distinguish the keys for the children of the root vertex; (iii) Top-

down approach is used in contrast to the bottom-up approach used in [2]. The bottom-up is not efficient for the hypercube based design. The shared-node violation shown in Fig. 14 demonstrates this point. Let p be the vertex being used for transfer, and q be a vertex susceptible to shared-node violation with p, where $dim(p) = dim(q)$. Let q' and q" be the parent and the sibling of q. q' must be susceptible to a shared-node violation, because the leaf vertices shared by p and q will also be shared by p and q', but this is not necessarily true for q". Since we cannot expose the keys for q' (a susceptible vertex) in the bottom-up KDM without exposing the delegation key for q" (a non-susceptible vertex), it shows that the bottom-up approach is unsuitable for the hypercube-based design.

Next, we consider the top-down KDM approach. Let q' be the child of q and is susceptible to route-node/shared-node violation with p. This implies that all parents of q' are also susceptible, because the leaf vertices shared by p and q' will also be shared by p and the parents of q'. To expose the delegation key of q' (a susceptible vertex) by a top-down KDM, we only need to expose the keys for the parents of q'. The parents of q' are also susceptible. Therefore, the top-down KDM is much more efficient than the bottom-up approach. In the rest of discussions, only the top-down KDM is considered, unless explicitly specified otherwise.

The pseudocode of KDM, based on (4.4) and (4.5), is depicted in Fig. 15, where DK_{in} and VK_{in} are respectively the input collections of delegation keys and verification keys. Starting from the i^{th} dimension, the routine recursively invokes itself until the leaf level is reached ($i = 0$). The process terminates at Line 05 of the last execution instance.

At the end of the execution, KDM produces a collection of delegation and verification keys denoted by DK_{out}, VK_{out}, respectively.

```
KDM(Dk_in, VK_in, i)
01:     set DK_out = DK_in;   set VK_out = VK_in;
02:     if i = n then
03:         set VK_out = VK_out È {X = F'(m, r) | "r Î DK_out};
04:     else if i = 0 then
05:         return (DK_out, VK_out);
06:     endif:
07:     for every vertex q at dimension (i - 1) do
08:         if " X_q1, X_q2, ..., X_qj, Î VK_out , then // q_1, q_2, ..., q_j are parents of q
09:             set DK_out = DK_out È {r_q = H(X_q1 || X_q2 ||..|| X_qj || q)};
10:             set VK_out = VK_out È {X_q = F'(m, r_q)};
11:         endif:
12:     endfor:
13:     KDM(Dk_out, VK_out, i-1);
```

Fig. 15: Pseudocode for key dependency map (KDM).

By controlling which keys are available in (DK_{in}, VK_{in}), the consumer (prover) can manage what the other consumer (verifier) can know about the produced keys in the transfer. For example, given r_{root} (randomly generated by the prover prior to the ZKP protocol), the set of delegation keys for all vertices in G, denoted by DK_G, and the corresponding set of verification keys, denoted by VK_G, can be produced by KDM, when KDM is invoked by (DK_{in}, VK_{in}, i) = ({r_{root}}, *null*, n). The consumer will directly expose a selected subset of keys from DK_G and VK_G during transfer. A high level view of the above key generation process for G_2 is depicted in Fig. 16. The keys of the root are used to compute the keys of q_1 and q_2, and then q_5. Generation of keys for the rest of vertices can be done in a similar fashion. Despite the similarity between Fig. 16 and the figure in

[2], the high connectivity in hypercube dependency graph makes the security analysis on A1, A2, and A3 much more complicated than its tree based counterpart.

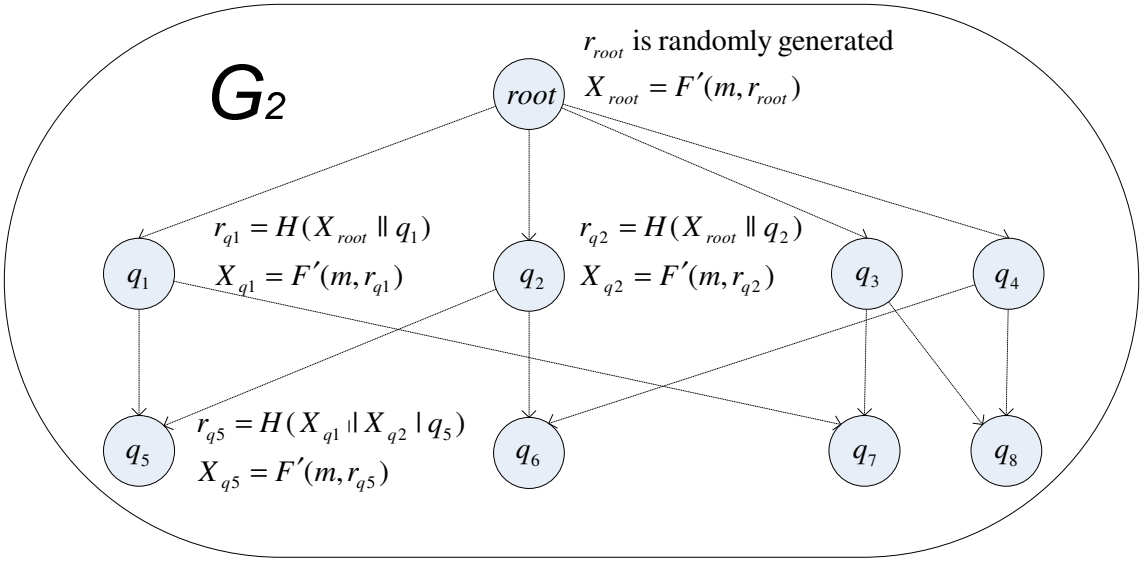


Fig. 16: High level view of key dependency map (KDM).

C. Cryptographic Constraint Analysis

In this section, we analyze the cryptographic constrains for $u(p)$, $v(p)$, $s(p)$ and L, so that the exposure of keys and secret shares in the hypercube based credentials could satisfy A1, A2, and A3, based on VU1, DU1, DU2, and KDM. The main concern of the analysis is to evaluate the conditions for the exposure functions that would be subject to anonymity hazards and the techniques to detect and prevent them from occurring in the runtime. Recall that $u(p)$, $v(p)$, $s(p)$, and L control the secret shares and keys directly exposed from the credential of p in a particular spending instance. For the analysis of

indirect exposures from multiple spending instances of p_1, p_2, …, p_j, it is convenient to represent them in terms of their KDM input/output. In particular, we denote

$$(DK_{KDM|p_1,\ldots,p_j}, VK_{KDM|p_1,\ldots,p_j}) \tag{4.6}$$

as the final KDM output when it is invoked by the initial input,

$$(DK(u(p_1)\cup\cdots\cup u(p_j)),\quad VK(v(p_1)\cup\cdots\cup v(p_j)\cup L),\ n)). \tag{4.7}$$

Note that the KDM input in (4.7) can always start the KDM execution from level n, regardless the dimensions of $p_1,\ldots,p_j$. This is because that any unmatched parent-child relationship that does not contribute to the key generation will be skipped by the loop at Line 07 in Fig. 15, and then it will continue the matching by the recursive call of KDM at Line 13 until it reaches the leaf level. The KDM output in (4.6) contains all delegation and verification keys in G_n that can be derived from the credentials produced from the transfers of p_1, …, p_j. Let DK^{-1} and VK^{-1} be the inverse functions of DK and VK. The vertex sets for delegation keys and verification keys in (4.6) are respectively denoted by

$$u_{KDM}(\{p_1,\ldots,p_j\}) = DK^{-1}(DK_{KDM|p_1,\ldots,p_j}), \tag{4.8}$$

and

$$v_{KDM}(\{p_1,\ldots,p_j\}) = VK^{-1}(VK_{KDM|p_1,\ldots,p_j}). \tag{4.9}$$

Similar notations are described as follows: (i) replace the subscript KDM in (4.6), (4.8) and (4.9) by $KDM\backslash L$ if L is removed from (4.7); (ii) replace the subscript KDM in (4.6) by $KDM\backslash L,q$ if both L and q are removed from (4.7) for $q \in (u(p_1)\cup\cdots\cup u(p_j)) \cup (v(p_1)\cup\cdots\cup v(p_j))$. Some KDM results related to A1, A2, A3, and the key generation are summarized in Table 9.

Table 9: Input/output of key dependency map (KDM).

Key Generation	A1	A2	A3
(DK_{in}, VK_{in})	$(\{r_{root}\}, null)$	$(DK(u(p)), VK(v(p)))$	$(DK(u(p_1) \cup ... \cup u(p_j)),$ $VK(v(p_1) \cup ... \cup v(p_j) \cup L))$
(DK_{out}, VK_{out})	(DK_G, VK_G)	$(DK_{KDM \setminus L \mid p}, VK_{KDM \setminus L \mid p})$	$(DK_{KDM \mid p_1,...,p_j}, VK_{KDM \mid p_1,...,p_j})$
Outputs of interest	$DK_G, VK_G \subseteq VK_G$	$VK(\ell(p)) = VK(L) \cap VK_{KDM \setminus L \mid p}$	$DK_{KDM \mid p_1,...,p_j}$

1. Authenticity

The cryptographic constraints for A1 can be analyzed using the set diagram in Fig. 17, which provides an excellent visual aid on the feasibility conditions for different constraints to co-exist simultaneously. These conditions are derived from VU1, which requires the integrity check on the secret share (E, Y), and its corresponding verification key X. Therefore, it requires $s(p)$ to be a non-empty set, so that at least one secret share is exposed to prove the condition $m = f(x)$. Moreover, it is required that

$$v_{\mathrm{KDM}}(\{p\}) \supseteq s(p), \tag{4.10}$$

so that every secret share exposed in a credential has the corresponding verification key available for the integrity check. The integrity check for verification keys is much more complicated than the GDA (in Table II), which only needs to examine one verification key. In contrast, we need to check all verification keys in $VK(v_{\mathrm{KDM}}(\{p\}))$.

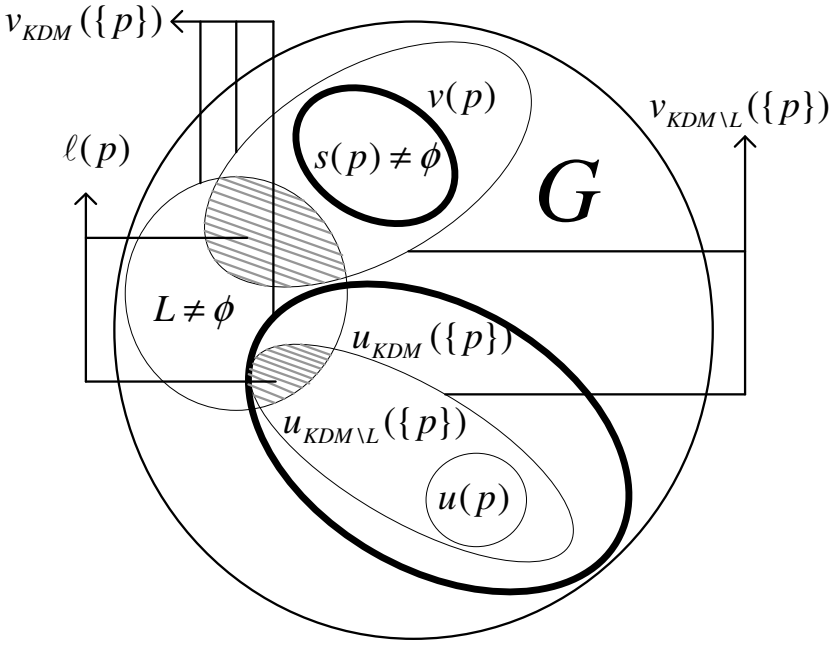


Fig. 17: Cryptographic constraints for authenticity (A1).

For simplicity, here we first assume that $VK(L)$ is authentic and we will explain how to assure the authenticity shortly. To enable integrity check of $VK(v_{\mathrm{KDM}}(\{p\}))$ based on $VK(L)$, L needs to satisfy two constraints:

$$\ell(p) = L \cap v_{KDM\setminus L}(\{p\}) \neq \phi\,, \tag{4.11}$$

and

$$\ell(p) \supset L \cap v_{KDM\setminus L,q}(\{p\}), \forall q \in u(p) \cup v(p)\,, \tag{4.12}$$

(4.11) assures that some verification keys in $VK(L)$ are compared against those indirectly exposed by the KDM input $(DK(u(p)), VK(v(p)), n)$. Since KDM is constructed by one-way collision-free hash function, any forgery to its inputs $DK(u(p))$ and $VK(v(p))$ will lead to mismatched outputs for comparisons.

Moreover, (4.12) guarantees that no key in $DK(u(p))$ and $VK(v(p))$ is redundant for comparisons. Otherwise, if removal of vertex $q \in u(p) \cup v(p)$ can still give the same result as in (4.11), it implies that one might fail to validate the key of q. Knowing that L is independent of p, $VK(L)$ should be determined before sub-cube spending and remain unchanged. Otherwise, if a consumer can use two different L's, then he will be able to use different r's to produce secret shares in two spending instances of p. However, DU2 requires both secret shares to be produced by the same (x, r) pair so that x can be deciphered on DSI. Therefore, the integrity check of $VK(L)$ is to make sure that the same collection of keys is used in different spending instances. To do this, two approaches have been addressed in TZKP described in section III: (i) $VK(L)$ has to be signed by the CA when the token is withdrawn, and the signature is included as part of m in the token T. (ii) $VK(L)$ has to be included as part of the challenge E in the previous ZKP instance for receiving payment. For further details and comparisons of these two approaches, please refer to section III.

2. Accountability

The cryptographic constraints for A2 and A3 are depicted in the set diagram of Fig. 18. Different from the constraints for A1, this time we ought to manage the secret shares and the keys for all combinations among 3^n vertices in G_n. Such a management scheme needs to keep track of all secret shares and keys exposed on the credential and indirectly exposed by the KDM.

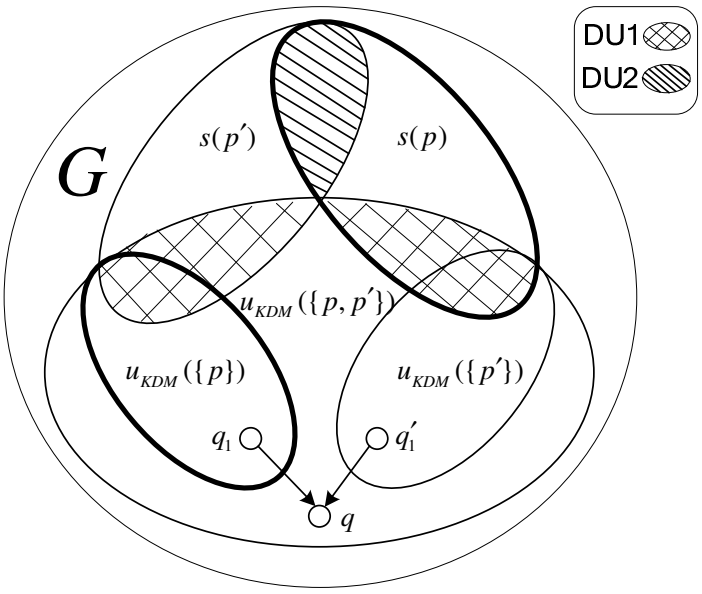


Fig. 18: Cryptographic constraints for anonymity and accountability (A2, A3).

Consider two vertices p and p' that were involved in double spending violations, *i.e.*, $d(p, p') = 0$. DU1 and DU2 dictate that one of the following constraints holds:

$$\text{DU1}: \; u_{KDM}(\{p, p'\}) \cap (s(p) \cup s(p')) \neq \phi, \tag{4.13}$$

or

$$\text{DU2}: \; s(p) \cap s(p') \neq \phi. \tag{4.14}$$

By (4.13) it implies that there exists at least one vertex whose delegation key is indirectly exposed by KDM when p and p' are spent. If its secret share is exposed the identity of the double spending offender will be deciphered based on DU1. By (4.14) it implies that at least one vertex $q \in s(p) \cap s(p')$ has its secret share exposed twice when both p and p' are spent. As discussed in A1, the secret share of q are guaranteed to be produced by the same delegation key so A2 is assured based on DU2.

3. Anonymity

The cryptographic constraints for A3, which are complements of (4.13) and (4.14) are given by:

$$\overline{\text{DU1}}: u_{KDM}(\{p_1,...,p_t\}) \cap \left(\bigcup_{j=1}^{t} s(p_j) \right) = \phi, \tag{4.15}$$

and

$$\overline{\text{DU2}}: \bigcap_{j=1}^{t} s(p_j) = \phi, \tag{4.16}$$

where p_1, …, p_t are vertices without double spending violations. (4.15) and (4.16) can be interpreted by using counter arguments of (4.13) and (4.14).

C. Cryptographic Constructs and Sub-cube Allocation Schemes

Constructs of the exposure functions, ($u(p)$, $v(p)$, $s(p)$), and L determine the AAA properties. When they are used with sub-cube allocation schemes together, with the performance overheads taken into consideration, we found that there exists an interesting and important tradeoff between the security strength and performance cost. To examine the balance between these factors, we propose two schemes for exposure functions and *L,* in conjunction with their sub-cube allocation rules.

An important tradeoff issue is noted here for the design of hypercube based *N*-divisible token. The first option (scheme I) is not using DU1 to track double spending offenses, but this leads to high computation costs due to secret share generation and

verification for DU2. The second option (scheme II) is using DU1, but with possible anonymity hazards because exhaustive combinations of vertices $\{p_1, p_2, \ldots, p_t\}$ in (4.21) are not tracked. As such, a simple checking routine needs to be added to the sub-cube allocation scheme to prevent their occurrences. Since simulation results show that sub-cube fragmentation related to anonymity hazards is less than 2%, it is highly effective to use sub-cube allocation rules to avoid anonymity hazards, rather than eliminating anonymity hazards unconditionally.

1. Scheme I

The exposure functions and L in Scheme I are defined as follows:

$$s(p) = \{\textit{leaf vertices of } p\}, \tag{4.17}$$

$$v(p) = \{\textit{leaf vertices of } p\}, \tag{4.18}$$

$$u(p) = \phi, \tag{4.19}$$

$$L = \{\textit{all leaf vertices in } G\}. \tag{4.20}$$

It is relatively straightforward to see that (4.17) – (4.20) satisfy all constraints in (4.10) – (4.16) but the number of secret shares that need to be exposed is equal to the number of nodes being spent. Note that generation and verification of secret shares have the highest computing costs in transfer protocol (assuming the computing $H(\cdot)$ is fast), making it a high computing overhead design.

Through the following arguments we assert that it is difficult, if not impossible, to find a more efficient alternative to Scheme I that can guarantee A3. First, we note that

it is relatively easy to satisfy A1 and A2 because they only consider one or two vertices each time. The analysis of A3 is complex because $u_{\mathrm{KDM}}(\cdot)$ in (4.15) is non-linear:

$$u_{KDM}(\{p_1,..,p_t\}) \supseteq \bigcup_{j=1}^{t} u_{KDM}(\{p_j\}). \tag{4.21}$$

The inequality in (4.21) is caused by the multi-parent, multi-child structure in G_n. It implies that delegation keys produced individually from $p_1,\ldots, p_t$, might miss some keys from those produced jointly. Losing track of these delegation keys will lead to anonymity hazard based on DU1. We illustrate this by Fig. 18, from which q has two parents, $q_1 \in u_{KDM}(\{p\})$ and $q_1' \in u_{KDM}(\{p'\})$, where $d(p, p') \neq 0$, $d(q, p) \neq 0$ and $d(q, p') \neq 0$. Spending p and p' should not expose the delegation key of q, because q is not a susceptible vertex. $q \notin u_{KDM}(\{p\})$ because $q_1' \notin v_{KDM}(\{p\})$. Similarly, $q \notin u_{KDM}(\{p'\})$ because $q_1 \notin v_{KDM}(\{p'\})$. However, we have $q \in u_{KDM}(\{p, p'\})$ because both q_1 and q_1' are available when credentials of p and p' are jointly considered. The delegation key of q (exposed by credentials of p and p') and its secret share (exposed by credential of q) create an anonymity hazard based on DU1. This kind of anonymity hazard does not occur to Scheme I, because the equality in (4.21) is assured by (4.19). On the other hand, Scheme I can only use DU2 for DSI, which requires a large number of secret shares to maintain A1 and A2.

2. Scheme II

The exposure functions and L in scheme II are defined as follows:

$$s(p) = \{p\}, \tag{4.22}$$

$$v(p) = \{p\}, \tag{4.23}$$

$$u(p) = \{\forall q \mid d(p, q) = 0, p \neq q, dim(q) = dim(p)\}, \tag{4.24}$$

$$L = \{all\ leaf\ vertices\ in\ G\}. \tag{4.25}$$

In contrast to scheme I, this scheme reduces the computation cost in the transfer protocol by selecting (4.22) as a minimal set that contains p alone. By using the minimal set of secret shares, we can guarantee the identification of same-node DSI violator, based on DU2, and the rest of analysis on distinct vertices will focus on DU1 and its complement.

By (4.22) – (4.23) and the constructs of KDM, we have $v_{KDM}(\{p\}) \supseteq v(p) = s(p)$, and hence, (4.10) is satisfied. To show that (4.22) – (4.25) also satisfy (4.11) – (4.12) for A1 and (4.13) – (4.14) for A2, we need to study their KDM outputs. (4.23) represents all vertices susceptible to shared-node violation with p at dimension $dim(p)$. Together with (4.23), which contains p only, $u(p) \cup v(p)$ represents all vertices susceptible to shared-node and same-node violations at $dim(p)$. We show that

$$v_{KDM \setminus L}(\{p\}) = \{\forall q \mid d(p, q) = 0, dim(q) \leq dim(p)\}, \tag{4.26}$$

by considering the following *vertex marking scheme*:

(i) Initially all vertices unmarked;

(ii) Mark all descendants of p;

(iii) Mark the unmarked parent(s) of the marked vertices if the unmarked parents are at the dimension not greater than $dim(p)$;

(iv) Repeat step (iii) until no more vertices can be marked.

The resulting collection of marked vertices are susceptible vertices at dimensions lower than or equal to $dim(p)$ as described in (4.26) because they share some leaf vertices of p. Those marked vertices at $dim(p)$, *i.e.*, $u(p) \cup v(p)$ can use their keys to compute the keys of other marked vertices by KDM, because the ancestor searching process in step (iii) and (iv) guarantees that any marked vertex at dimension lower than $dim(p)$ has the delegation/verification keys of its parents available from other marked vertices to produce its keys using (4.4). Fig. 19 depicts how the vertex marking scheme works in G_3.

Suppose $p = 0\times\times$ is a vertex used for transfer. To evaluate $v_{KDM}(\{0\times\times\})$, we first mark all descendants of $0\times\times$, *i.e.*, $\{00\times, 01\times, 0\times0, 0\times1, 000, 001, 010, 011\}$, and then we mark all unmarked ancestors of these vertices at levels $\leq dim(p)$, *i.e.*, $\{\times00, \times01, \times10, \times11, 0\times\times, \times0\times, \times1\times, \times\times0, \times\times1\}$. We can see that $u(p) = \{\times0\times, \times1\times, \times\times0, \times\times1\}$ and $v(p) = \{0\times\times\}$ have formed all the dependencies that are required to compute the verification keys of all marked vertices. The marked nodes are susceptible to double spending with p at dimensions $\leq dim(p)$. Given $v_{KDM\setminus L}(\cdot)$ defined in (4.26), (4.11) holds because each vertex p has at least one leaf vertex q at some dimension not greater than $dim(p)$, that is susceptible to double spending (equal dimension if $p = q$.) Furthermore, (4.12) also holds because removing any vertex q from $u(p) \cup v(p)$ will prohibit the key computation of

some leaf vertex shared by p and q. Since (4.22)-(4.25) satisfy (4.10)-(4.12), Scheme II satisfies all cryptographic constraints for A1.

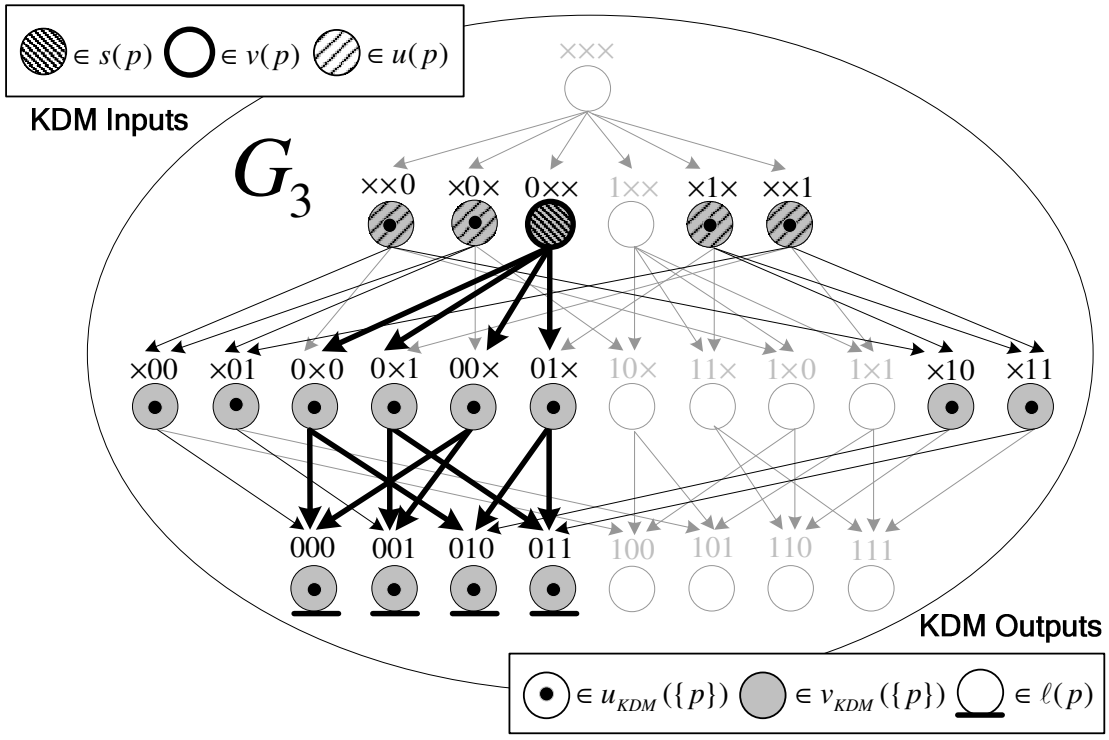


Fig. 19: Vertex marking scheme for susceptible vertices with respect to $p = 0\times\times$.

To show that (4.22)-(4.25) also satisfy the crypto constraints for A2, we need to show that they satisfy either (4.13) or (4.14) for any vertices p and p', where $d(p, p') = 0$. For the case of $p = p$', it is straightforward that (4.14) is satisfied. For $p \neq p$', $dim(p) \geq dim(p')$, we first show that

$$u_{KDM}(\{p\}) = \{\forall q \mid d(p, q) = 0, p \neq q, dim(q) \leq dim(p)\}. \qquad (4.27)$$

(4.26) and (4.27) are depicted in Fig. 19, by the highlighted vertices, and the set of vertices with a dot marked inside, respectively. (4.26) and (4.27) are almost identical except that $\{p\}$ is excluded from (4.27). In (4.27), the vertices at dimension $dim(p)$ are contributed by (4.24), while those at dimensions lower than $dim(p)$ are justified by the vertex marking scheme we described before. Based on (4.27), we have

$$u_{KDM}(\{p\}) \cap s(p') = p'. \tag{4.28}$$

In addition, based on (4.21) or $u_{KDM}(\{p, p'\}) \supseteq u_{KDM}(\{p\})$, we conclude that (4.13) is satisfied for A2.

Table 10 depicts a spending configuration that can lead to anonymity hazards in G_3, where $p_1 = 1\times0$, $p_2 = 01\times$, $p_3 = \times01$, and $p_4 = 000$ are four vertices for transfers[3]. Even though $d(p_i, p_j) \neq 0$ for any distinct pair, by using their credentials, the delegation key of 000 can be produced by (4.4), *i.e.*, $r_{000} = H(X_{00\times} || X_{0\times0} || X_{\times00} || 000)$, because $\times00 \in u(p_1)$, $0\times0 \in u(p_2)$ and $00\times \in u(p_3)$. Using r_{000}, and the secret share of 000 exposed from $s(p_4) = \{000\}$, the identity of this rule-abiding consumer can be deciphered by DU1.

[3] We will show in Appendix that the minimum number of vertices to cause anonymity hazard in Scheme II is four.

Table 10: Anonymity hazard scenario in G_3.

Spent vertex	Exposure functions
$p_1 = 1\times0$	$u(p_1) = \{10\times, 11\times, \mathbf{\times00}, \times10\}$
$p_2 = 01\times$	$u(p_2) = \{\times10, \times11, \mathbf{0\times0}, 0\times1\}$
$p_3 = \times01$	$u(p_3) = \{0\times1, 1\times1, \mathbf{00\times}, 10\times\}$
$p_4 = 000$	$s(p_4) = \{\mathbf{000}\}$

Scheme II is more efficient than Scheme I. However, it does not guarantee A3 unconditionally. It is designed to work with a sub-cube allocation scheme to detect and avoid anonymity hazards before a sub-cube is spent or allocated. We propose a simple *anonymity hazard test* (AHT) algorithm to test whether or not a vertex p of the requested sub-cube size is subject to anonymity hazard with respect to previously spent vertices, so that only hazard-free sub-cubes will be spent and allocated. When all available vertices at the requested sub-cube size are subject to anonymity hazard, the request will need to be divided into smaller requests, each of which will need to be served separately. This type of fragmentation condition is caused by anonymity hazard, as the example depicted in Table 11.

Table 11: Fragmentation scenario when Q_1 is requested.

Spent vertex	Exposure functions
$p_1 = 1\times0$	$u(p_1) = \{10\times, 11\times, \mathbf{\times00}, \times10\}$
$p_2 = 01\times$	$u(p_2) = \{\times10, \times11, \mathbf{0\times0}, 0\times1\}$
$p_3 = \times01$	$u(p_3) = \{0\times1, 1\times1, \mathbf{00\times}, 10\times\}$
$p_4 = 111$	$s(p_4) = \{\mathbf{111}\}$

In this example, $p_1 = 1\times0$, $p_2 = 01\times$, $p_3 = 000$, and $p_4 = 111$ are spent. If the next sub-cube requested is a Q_1 then the only available vertex of this size is $\times01$. As shown in Table 10, an anonymity hazard will result from spending of $\times01$, after p_1, p_2, and p_3 have been spent but the anonymity hazard is eliminated when $\times01$ is divided into two smaller sub-units {001, 101} for spending. The sub-cube fragmentation increases computation and communication overheads. Fortunately, simulation results show that fragmentation caused by anonymity hazard in Scheme II is negligible even when arbitrary sub-cube allocation scheme is considered. The simulation further shows that no anonymity hazard is detected when Scheme II is integrated with two popular sub-cube allocation schemes, BC or BRGC.

D. Experimental Results

The objective of this simulation is to evaluate the occurrences of anonymity hazards and their fragmentation effects for BC, BRGC and RC. Before giving details of the simulation program, we first explain the AHT algorithm as shown in Fig. 20.

In AHT, each vertex in *G* is represented by one of the four colors: white, black, gray and red. Each vertex is initialized to white before any spending instance. A black vertex implies that its verification key has been exposed from past spending instance(s). A red vertex implies that both its verification key and the secret share of this vertex have been exposed from past spending instance(s). A gray vertex implies that its delegation and verification key are not exposed from past spending instance(s), but will be exposed if the current spending instance succeeds.

```
AHT(p, G)
01:     store the color of p;
02:     if all parents of p are not white then
03:         return TRUE;
04:     endif:
05:     turn all white vertices in u(p) È v(p) to gray;
06:     set i = dim(p) – 1;
07:     for all vertex q at dimension i, i ³ 0; i-- do
08:         if all parents of q are not white then
09:             if q is white then
10:                 turn q to gray;
11:             else if q is red then
12:                 turn all gray vertices in G to white;
13:                 resume color of p in G;
14:                 return TRUE;
15:             endif:
16:         endif:
17:     endfor:
18:     turn p to red;
19      turn all gray vertices to black;
20:     return FALSE;
```

Fig. 20: Pseudocode of anonymity hazard test (AHT).

The algorithm returns TRUE when an anonymity hazard is detected at Lines 03 and 14 for the pending sub-cube spending. Line 03 is the case when the verification keys or the delegation keys of all parents of p have been exposed from the past spending instance(s). Thus, all parents are not white in Line 02. By (4.4) and (4.5), the delegation key of p can be derived from these delegation keys or verification keys. Furthermore, by (4.22), the secret share of p will be exposed if the current spending instance succeeds. If both the delegation key and secret share of p are available, the identity can be deciphered based on DU1. Line 14 handles the case when the algorithm attempts to turn a red vertex q to gray. The algorithm attempts to change a vertex q to gray color when all parents of q are not white (Line 08), which means the delegation key of q can be derived by (4.22) if p is spent. Since q is originally red, its secret share was exposed from previous spending instance. Given both the delegation key and the secret share of q available, the identity can be deciphered based on DU1.

In Line 05, the originally uncolored vertices in $u(p) \cup v(p)$ are colored in gray by definitions of $u(p)$ and $v(p)$, but colored vertices remain unchanged so that only gray vertices need to be rolled back to the white color if p is found to be subject to anonymity hazard (Line 14.) The loop from Lines 07 to 17 is to update the colors of vertices due to Line 05. For the top-down KDM analysis, Line 05 computes the delegation keys and verification keys for vertices at the dimensions lower than $dim(p)$ in this loop. Let q be a susceptible vertex in the loop. Line 08 checks if all parents of q are not white; and if they are all not white, it means that the delegation key of q can be derived by (4.22) if p is indeed spent. As a result, q needs to be marked gray if it is not colored before (Lines 09

to 10). However, if q has been marked in red (Line 11), then anonymity hazard will occur if p is spent. p needs to roll back to its original color (black or white) and all vertices colored in gray during this test need to be rolled back to the white color (Lines 12 and 13) and an anonymity hazard condition needs to be reported (Line 14). If no anonymity hazard is found after checking all q in the loop, then p can be spent without causing any problem. Before returning FALSE (Line 20), p and all vertices colored in gray need to mark their colors red and black (Lines 18 and 19), respectively.

The pseudo code of the simulation is given in Fig. 21 to measure occurrences of anonymity hazard, and their fragmentation effects. The program starts by initializing all leaf vertices in G_n as "not spent" in Line 01. Then it keeps generating sub-cube requests of different sizes for spending, and finally it terminates at Line 25 when all leaf vertices are marked "spent".

Lines 02 to 03 randomly and uniformly generate a vertex $p\_tmp = 0, 1, \ldots, 3^n-1$. The ternary representation of p_tmp is used to determine $i = dim(p\_tmp)$. In this way, the probability in producing requests of extremely large/small sub-cubes is minimized. For example, in G_3 the probability to request the entire hypercube (the root vertex) is $1/3^3 = 1/27$, while that for a sub-cube at dimensions 2, 1, and 0 from G_3 are 6/27, 12/27, and 8/27, respectively. A request can be served only if one sub-cube of the same size can be found from the unspent sub-cube pool based on the sub-cube allocation rule, such as BC, BRGC, RC, and no anonymity hazard is detected. Otherwise, the request is discarded.

```
01:     leaf_vertex[0…(2^n)-1] = “not spent”;
02:     randomly generate p_tmp from 0 to (3^n)-1; // ternary numbers
03:     set i = dim(p_tmp);
04:     if (2^i) > number of leaf vertices marked “not spent” then
05:             goto 02;
06:     endif:
07:     use the current allocation scheme to select an available sub-cube p at dimension i;
08:     if no such p exist then
09:             report fragmentation (NOT caused by anonymity hazard)
10:             goto 02:
11:     endif:
12:     do anonymity hazard test (AHT) on p;
13:     if AHT passes then
14:             mark all leaf_vertex[x] of p as “spent”;
15:     else
16:             report anonymity hazard;
17:             use the current allocation scheme to find another available sub-cube p at dimension i;
18:             if no such p exists then
19:                     report fragmentation (caused by anonymity hazard)
20:             else
21:                     goto 12;
22:             endif:
23:     endif:
24:     if all leaf_vertex[ ] are marked “spent” then
25:             terminate the program;
26:     else    goto 02; // next subcube request
27:     endif:
```

Fig. 21: Pseudocode of the simulation program for fragmentation.

The simulation results for G_4 to G_{10} are depicted in Fig. 22 and Fig. 23 based on the average results of 1,000 runs of simulation instances. The *anonymity hazard ratio* is measured by the number of AHT executions which return TRUE to the total number of AHT executions in Line 12. The *fragmentation ratio* is the proportion of fragmentations (caused by anonymity hazards) versus the total number of sub-cube requests in Line 07. For RC, the anonymity hazard ratio increases in a linear fashion with the hypercube size from G_4 to G_{10}. On the other hand, the fragmentation ratio decreases with the hypercube size and is consistently lower than 2%. In contrast, anonymity hazard did not occur to BC or BRGC in all simulation runs, and thus AHT is not needed for these two schemes.

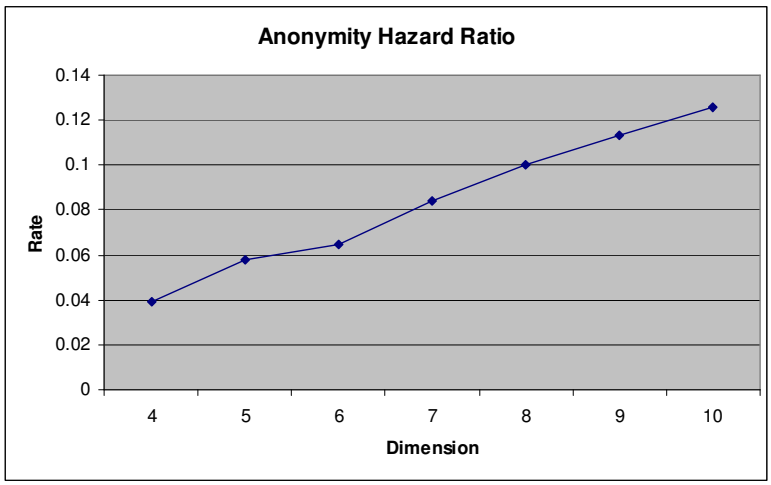


Fig. 22: Anonymity hazard ratio.

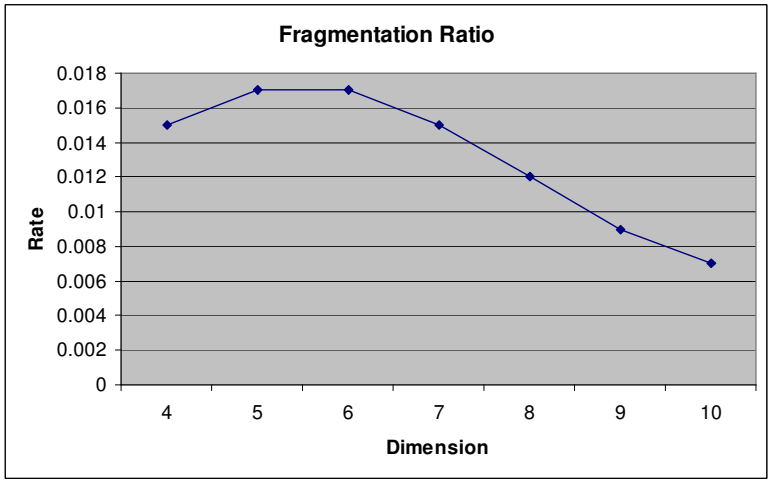


Fig. 23: Fragmentation ratio (caused by anonymity hazards).

E. Summary

In this section, we investigated the relationship between *N*-divisible tokens, sub-cube allocation schemes, and their integration. We demonstrated that a holistic security

management system can be created by tailoring the *N*-divisible token framework of the disposable authentication with different sub-cube allocation schemes. We developed the analysis techniques to guarantee the AAA properties of hypercube based *N*-divisible tokens. As expected, the most secure solution requires the highest computing cost. As an alternative, we also show that one can achieve the same security management goals at much lower computing costs by relaxing the anonymity protection rules and adding an anonymity hazard checking routine before the sub-cube can be allocated. The anonymity hazard checking routine is simple and reliable. Furthermore, simulation results show that existing sub-cube allocation schemes binary code and binary gray code are immune from anonymity hazards because of their restrictive allocation rules. Our study suggests that with proper adjustment to both the *N*-divisible tokens and the resource allocation rules, highly secure and efficient computing resource management schemes can be developed based on one integrated framework.

V. PRIVACY PRESERVING SERVICE ORIENTED ARCHITECTURE (SOA)

Service oriented architecture (SOA) is a software paradigm which links business and computation resources on demand to achieve desired results for service consumers. It promotes the reuse of computer resources at macro level (services) rather than micro level (objects). The abstraction and reusability on the loosely coupled and interoperable services help business respond timely and cost-effectively to changing market conditions [75]. In this section, we will introduce a privacy preserving SOA framework to support *service reservation* in a *peer-to-peer* (P2P) network.

Service reservation is useful for SOA to allow service providers more predictable workload. It also allows service consumers to be more certain on availability of services. For example, a service consumer wants to guarantee that all required service components are reserved before the service components are used to construct new composite, higher-level services. Nevertheless, service reservation may also cause a waste of resources if a service consumer does not need the reserved services anymore. For example, the service consumer reserves two similar services but only uses the better one based on the current market condition. Therefore, a more advanced design should allow the service consumer to transfer its service reservation to another service consumer, so that any consumer who can present a valid service reservation can redeem the services from the service provider.

Clearly, the E-cash based resource management framework proposed in previous sections paves a solid foundation for the reservation and transfer operations. However, the E-cash paradigm has a missing link in service discovery to get service providers and

service consumers know each other before they can start the service transactions. In this section, we aim to bridge the gap by studying the service discovery in a P2P network.

Service discovery in SOA is typically achieved by *service brokering* [76], where the service providers and service consumers need to register to the *service broker*(*s*), and then the service broker serves as a directory to introduce the service consumer to service provider, and then the service consumer and service provider can communicate directly for transactions. Existing service discovery protocols such as UDDI [77], SSDS [78], and Splendor [79], provide directory service through a collection of dedicated trusted servers which locations are well-known. On the other hand, service discovery in P2P networks require every peer node in the network to be the directory of other peer nodes, and their locations may not be known in advance. Without using well-known trusted servers, peer nodes are reluctant to release their sensitive information, such as work requests, service capabilities, terms, and availability schedules. Leakage of the inquiry or service offering may trigger rumors or market volatility. However, without releasing enough information, they can hardly locate potential collaborators for business transactions.

To solve this dilemma, our main idea is to allow users to progressively release sensitive information for matching and verification, while not leaking the information to unintended recipients upon failure of matching or verification. It is similar to the idea of qualification verification which mutually verifies such information as the possessions of goods, skills, and finance resources, while blocking speculators from accessing privilege information exchanged between users. In particular, we focus on the privacy protection

of service contents, group identities, and user identities through the *multi-layer Bloom filter* (MLBF) and *secret handshake* (SH) protocol. Both cryptographic tools require no centralized supervision during regular operations so they are useful for the P2P network.

In the proposed framework, users are interconnected by CHORD [80]. CHORD is a ring-based P2P network that supports efficient posting and inquiries of services with *distributed hashing tables* (*DHTs*) [81]. Nevertheless, the key space of CHORD is flat so it does not inherently support service lookup of structural data such as XML. XML is the de facto standard data format in SOA. Numerous works are proposed for the XML-based service discovery in CHORD [82, 83] but they require all or part of the XML plaintext to be kept in the directory nodes, leaving vulnerability for leaking sensitive information. To provide better privacy protection, we hash the service contents by MLBF to generate the location information on CHORD. Publications of hashed results protect the exact natures of services while still allowing potential service consumers who can specify the related service to locate matched service providers. At the same time, contents of inquires from consumers are also protected because they are also hashed by MLBF.

Given that all service contents are published in their hashed forms, MLBF offers reasonable protection from outsiders. But it does not protect the participants from insider adversary who are determined to make broad surveys or scans of the business activities. Furthermore, there is no way for matched participants to make risk-free, fair exchange of information, unless they belong to the same group that implies a similar trusted level or capabilities. As such, we only use MLBF to distribute the natures of available services. After two users are matched, they need to execute an SH protocol to test whether or not

they belong to the same group. If they are within the same group, they can execute the transaction using the TZKP protocol to perform such functions as service reservations, service redemptions, and transfers of reservations. Otherwise, if they belong to different groups, the SH protocol guarantees that their group identities cannot be deciphered from one another using their authentication messages.

Different from the original SH protocol [6], our scheme supports *reusable tokens* which allow two users from the same group to authenticate each other without exposing their user identities. In contrast, reusing a token in [6] will link authentication messages of different transactions to the same user identity. Our protocol does not need to assign a set of pseudonyms to the user, while the original SH requires the allocation of one-time pseudonyms for each user. Our scheme only requires 2 pairing operations. On the other hand, the scheme in [7] uses 6 pairing operations for the similar functions. Experimental results show that our scheme takes 65 milliseconds for each SH operation. It provides a light-weight authentication solution for screening of unknown users in secure resource allocations.

The rest of this section will be organized as follows. Section A gives a system overview. Section B explains the basic operations needed in the system. Section C elaborates the protocol details of the system. Section D delivers the experiment results. Section E gives a summary of this section.

A. System Overview

In this section, we explain the system architecture of our privacy preserving SOA framework. The system comprises of three major phases: *system initialization*, *service discovery*, and *service transaction*. As shown in Fig. 24, the *central authority* (CA) only participates in the system initialization phase. The peer nodes can operate autonomously on the logical ring (CHORD) to look up target nodes in the service discovery phase, and then communicate directly (point-to-point) with target nodes in the transaction phase.

In system initialization, the CA prepares the public-private parameters required by the system. After that, the users can register from the CA. During registration, the CA assigns a unique user identity and a group identity by issuing some tokens to the user. The users will need to use these tokens to authenticate each other as a legitimate service provider or service consumer during the service discovery and service transaction phases.

In service discovery, users search on the CHORD ring for other users who have matched groups, service requests, and service descriptions. The user who joined the ring can add the MLBF representation of its service descriptions onto the ring. Two types of users can add their descriptions. The first type is the service provider who offers services. The second type is the service consumer who reserved services and wants to transfer it to another consumer. Both types of users want potential consumers to search their service descriptions on the ring. The consumer searches the service descriptions by preparing an MLBF representation of its service request and looks up on the ring. The lookup returns the locations of providers who provide potentially matched services and the locations of

consumers who transfer potentially matched service reservations. The consumer contacts some of these locations to further match their groups by the SH protocol. The consumer proceeds to the service transaction phase if they belong to the same group.

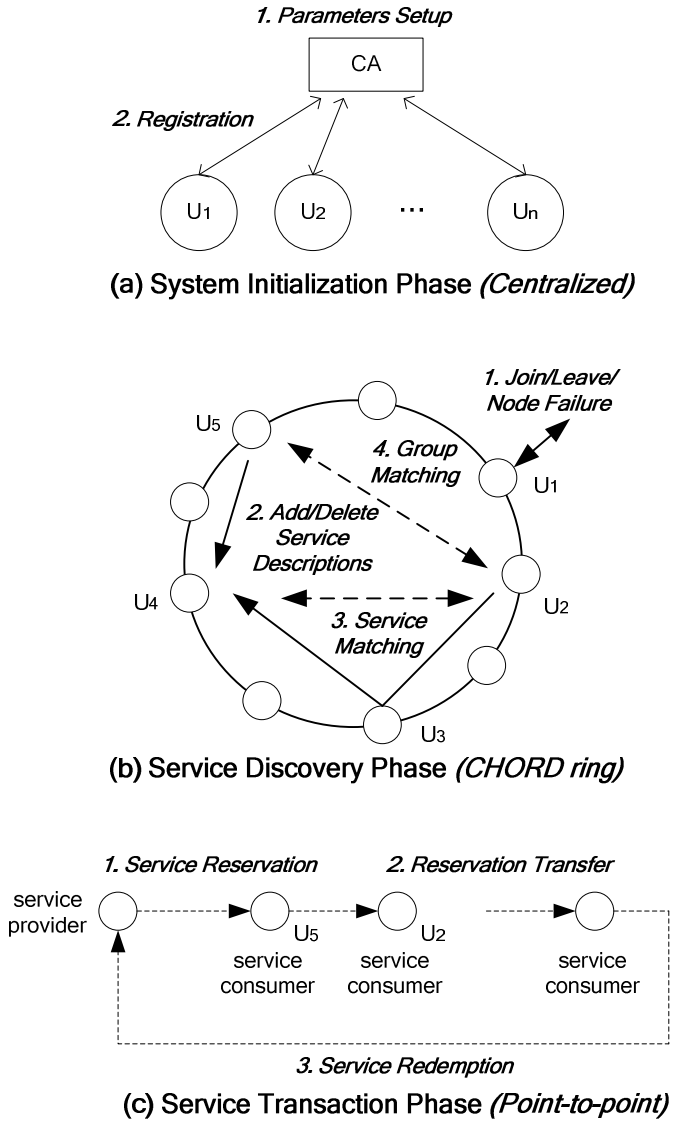


Fig. 24: System architecture for reservation based SOA.

In service transaction, the matched service providers and consumers in service discovery can use the TZKP protocol to take part in three protocols: service reservation, reservation transfer, and service redemption. In service reservation, the service provider issues a *reservation credential* (a.k.a. TAA in section III) to the consumer. It grants the reservation credential holder the authorization to redeem the said service. In reservation transfer, the service consumer can choose to give up the authorization and transfer it to another consumer. In service redemption, the service provider offers service to the user who can present the reservation credential, no matter who the user is.

Fig. 24b and Fig. 24c demonstrate an example for service discovery and service transaction. Node U_5 is a service consumer who reserved some services from the service provider. Now, U_5 wants to transfer its service reservation to others. It adds the service description to the CHORD ring. We assume that the service description is added to the node U_4 in this example. Another service consumer U_2 wants to search for services. It prepares a service request and looks up on the CHORD ring. U_2 finds that U_4 stores a potentially matched service description. U_4 returns the location of U_5 to U_2. U_2 contacts U_5 to check whether or not they belong to the same group. If they belong to the same group, then they are willing to exchange their service description and service request, to check whether or not they are matched. If they are matched, U_5 transfers the reservation to U_2, and removes the service description from U_4. Finally, U_2 can redeem the service, or transfer the reservation to other consumer.

B. Basic Operations

The building blocks include CHORD, MLBF, and the SH protocol. CHORD facilitates service lookup in a P2P network. MLBF protects the service descriptions and requests for service lookups. SH protocol protects group identities of service providers and service consumers who take part in the service lookup.

1. P2P network: CHORD

CHORD [80, 81] is a P2P network from which each node is ordered in a logical ring modulo 2^m. A node can add a key-value pair to another node and lookup the value from the node using the key. The lookup protocol of CHORD is efficient, which requires $O(\log n)$ hops only, where n is the number of nodes on the ring. We denote the location of node A on the ring as LC_A. The basic operations of CHORD are described as follows:

LOOKUP: Lookup refers to the mapping from the key to the node location. It maps key k to the first node location equals to or follows k. This node is called the *successor node* of k, denoted by *successor*(k). To speedup the lookup process, it uses a *finger table* up to m entries. The i-th entry at node n stores s = *successor*($n+2^{i-1}$). Node n can skip all nodes between its successor and the precedent of s if k is larger than the precedent of s.

ADD/RETRIEVE/DELETE VALUE: Lookup the node location by the key, and then add the value to the node, or retrieve or delete the value from the node.

JOIN/LEAVE/FAILURE OF NODE: Every node updates its finger table periodically to reflect changes caused by join, leave, or failure of nodes. For join operation, some values are migrated to the joined node from its precedent. For leave operation, all values are moved from the leaving node to its successor. To handle the simultaneous failures, each node keeps a *successor list* which stores the first *r* successors. If the immediate successor does not respond, the node substitutes it with the second entry in the successor list.

2. Multi-layer Bloom filter (MLBF)

MLBF [84] is originally designed for space-efficient content-based routing in the tree-based topology. It is an array of hash functions for heuristic membership testing for data in hierarchical structures such as XML. To understand MLBF, we first introduce the basic operations for baseline Bloom filter as follows:

INITIALIZATION: We prepare a vector v with m bits, initially all set to 0, then prepare k hash functions, h_1, h_2, …, h_k, each with range 1 to m. m the length of the Bloom filter.

ADD ELEMENT: To add element a to v, we set $v = v \vee BF(a)$, where $\vee$ is the binary OR operator, and $BF(a)$ is an m-bit string with its positions, $h_1(a)$, $h_2(a)$, …, $h_k(a)$, set to 1, and the rest of bits set to 0.

MATCH ELEMENT: To match element b with v we check the bits at the positions, $h_1(b)$, $h_2(b)$, …, $h_k(b)$, in v. If any of them is 0, then b was not added. Otherwise, we conjecture that b was added despite a certain probability that it was not. k and m are selected in such a way that the false positive rate is acceptable.

MLBF is constructed by multiple baseline Bloom Filters as described as follows:

INITIALIZATION: We prepare BF_0, …, BF_j and their vectors v_0, …, v_j. Then, we prepare a vector t with n bits initially all set to 0, where n is the total length of the Bloom filters.

ADD DOCUMENT: To add an XML document D to t, we add all element names at level i of D to v_i with BF_i, where root level = 1. In addition, we add all element names in D to v_0 with BF_0. We compute $MLBF(D) = v_0 \| \ldots \| v_j$, where "‖" is the concatenation operator. Then we set $t = t \vee MLBF(D)$ and reset all v_i to be 0.

MATCH REQUEST: To match XPath T = "$/a_1/\ldots/a_p$" with t, we compute u_i as follows:

$$u_0 = BF_0(a_1) \vee \ldots \vee BF_0(a_j), \text{ for } a_i = \text{an element name.} \tag{5.1}$$

$$u_i = \begin{cases} BF_i(a_i), & \text{for } a_i = \text{an element name} \\ 0, & \text{for } a_i = \text{a wildcard operator} \\ 0, & \text{for } p < i \le j \end{cases} \tag{5.2}$$

Let $t = t_1 \ldots t_n$ and $s = MLBFQ(T) = s_1 \ldots s_n = u_0 \| \ldots \| u_j$, where t_i, $s_i = \{0,1\}$. If $(t \wedge s) \otimes s = 0$, *i.e.,* $t_i = 1$ for all $s_i = 1$, then T potentially matches some documents added to t, where $\wedge$ is binary AND operator, and $\otimes$ is binary XOR operator. Otherwise, this is a mismatch. The false positive rate is decided by the BF sizes at each level and natures of D and T.

3. Secret handshake (SH) protocol

Among various group authentication schemes, *secret handshake protocol* [6, 7, 85-91] emphasizes on protection of group membership information. In its original form, secret handshake protocol allows a group member to verify the membership of another

member while non-members cannot determine or impersonate group membership from their authentication messages. The authentication does not require supervisions from the central authority (CA) but users need to receive some *SH tokens* from the CA to become a member to be able to take part in secret handshakes. The SH token is derived from the *user identity* and a *group secret*. The group secret is known by the central authority only. Secret handshake protocol differs from other group authentication schemes that it does not require group public key. It checks whether or not both users can compute a common value using their tokens. A common value implies that their tokens are derived from the same group secret, and hence, they are from the same group. In contrast, distinct values do not expose any information that can link to the group membership of the users.

The proposed SH protocol is derived from the pairing-based SH protocol in [6]. Pairing-based cryptography is based on bilinear maps over groups of large prime order [8]. In pairing-based cryptography, "groups" refers to groups in linear algebra which is different from groups in SH protocol. G_1 denotes an additive cyclic group of prime order q. G_2 denotes a multiplicative cyclic group of order q. G_1 and G_2 are selected in such a way that the *Discrete Logarithm Problem* (DLP) [8] is hard in both of them.

Definition 5.1: A pairing is a bilinear map e: $G_1 \times G_1 \rightarrow G_2$ if, for any $P, Q \in G_1$ and any $a, b \in Z^*_q$, we have $e(a \cdot P, b \cdot Q) = e(a \cdot P, Q)^b = e(P, b \cdot Q)^a = e(P, Q)^{a \cdot b}$ and $e(P,Q) = e(Q, P)$ for $\forall\ P, Q \in G_1$.

A typical choice of G_1 is a set of points on an elliptic curve. G_2 is a multiplicative cyclic group over integers. Our SH protocol uses *Tate parings* on supersingular elliptic

curves because their computations of bilinear maps are efficient [92], provided that the following problem is hard:

Definition 5.2 (Bilinear Diffie-Hellman (BDH) Assumption): Given P, $a \cdot P$, $b \cdot P$, $c \cdot P$ for random $a, b, c \in Z^*_q$ and $P \in G_1$, it is not possible to compute $e(P, P)^{a \cdot b \cdot c}$ with a non-negligible probability, *i.e.*, it is hard to compute $e(P, P)^{a \cdot b \cdot c}$.

Our protocol uses two hash functions H_1 and H_2. H_1 maps a string with arbitrary length to an element in G_1, *i.e.*, a point on a specific elliptic curve. H_2 maps a string with arbitrary length to a string with fixed length. "||" denotes a string concatenation operator.

The SH system comprises of a central authority (CA) and a collection of users. The CA is responsible for setting up system parameters and issuing tokens for users to prove the group membership. The CA sets up pairing parameters (q, G_1, G_2, e, H_1, H_2) during system initialization. It also prepares a series of *group secrets* [g_1, …, g_n] to represent different groups. The pairing parameters are published to the users whereas the group secrets are known by the CA only. When user A joins group g_A, A presents its user identity, ID_A, to the CA. Then the CA grants the group membership by issuing A a token, K_A, derived from g_A and ID_A. K_A is the secret for A to prove its group membership to other users, without exposing any information which can link to ID_A. A cannot forge a token to prove a group membership other than g_A.

The group membership knowledge that can be observed from the authentication messages is summarized in Table 12. As depicted in the table, A and B can authenticate one another anonymously using their tokens if they are from the same group, $g_A = g_B$, but they do not know the values of g_A and g_B. If A and B belong to different groups, or

any of them belong to no group, they know $g_A \neq g_B$ only. *A* and *B* obtain no information that can link to the values of g_A and g_B. Other users cannot perceive whether *A* and *B* belong to the same group or not. In this section, we proposed a reusable SH token so that users can use an SH token multiple times in different transactions, without exposing any information that can link to the user identity.

Table 12: Secret handshake protocol with reusable tokens.

	A and *B* ∈ some group	*A* or *B* ∉ any group	Users other than *A* and *B*
Succeeds	$g_A = g_B$	Always Fails	Uncertain:
Fails	$g_A \neq g_B$		$g_A = g_B$?

Our main idea is to let the user generate a secret random number in every secret handshake. The user needs to multiply the random number to an elliptic curve point that represents the user identity. The random number minimizes the correlation between the authentication messages even though they are produced by reuse of a token. The simple construct is more efficient than other reusable schemes with similar functions [6, 89]. Our protocol comprises of three phases: *INITIALIZATION*, *JOIN GROUP*, and *SECRET HANDSHAKE*, as detailed below:

INITIALIZATION: The CA determines the pairing parameters (q, G_1, G_2, e, H_1, H_2) and group secrets [g_1,…, g_n] given a security parameter 1^k, where q is a large prime and $g_i \in Z^*_q$. The CA publishes the pairing parameters while keeping the group secrets in private.

JOIN GROUP: User A requests the CA to join group $g_A \in [g_1,\ldots, g_n]$. The CA verifies A's user identity, ID_A, to decide whether A can join the group. The CA grants the group membership to A by issuing a token $g_A \cdot H_1(ID_A) \in G_1$. The token is a secret of A to prove its membership in group g_A to another user in the same group. A cannot deduce g_A from $g_A \cdot H_1(ID_A)$ and $H_1(ID_A)$ assuming that DLP is hard in G_1. It is important for preventing forgery of tokens.

SECRET HANDSHAKE: Users A and B use their tokens, $K_A = g_A \cdot H_1(ID_A)$ and $K_B = g_B \cdot H_1(ID_B)$, to generate authentication messages to one another. A randomly generates two non-zero integers, n_{A1} and s_{A1}. n_{A1} prevents replay attacks as in [85]. s_{A1} minimizes the correlations of authentication messages produced by the same token. Since using a token multiple times will not create messages that can link to the user identity, the token is reusable. B also randomly generates two non-zero integers, n_{B1} and s_{B1}, for the same purpose. Detailed interactions of our secret handshake protocol are described as follows:

(a) $A \rightarrow B$: n_{A1}, $W_{A1} = s_{A1} \cdot H_1(ID_A)$

(b) B: Compute $V_{B,A} = H_2(U_{B,A} \| n_{A1} \| n_{B1} \| 0)$, $U_{B,A} = e(W_{A1}, s_{B1} \cdot K_B)$

(c) $B \rightarrow A$: n_{B1}, $W_{B1} = s_{B1} \cdot H_1(ID_B)$, $V_{B,A}$

(d) A: Compute $V'_{B,A} = H_2(U'_{B,A} \| n_{A1} \| n_{B1} \| 0)$, $U'_{B,A} = e(W_{B1}, s_{A1} \cdot K_A)$

If $V_{B,A} = V'_{B,A}$, then A knows B belongs to the same group, *i.e.*, $g_A = g_B$. Otherwise, B belongs to a different group, *i.e.*, $g_A \neq g_B$ or B belongs to no group.

(e) $A \rightarrow B$: $V_{A,B} = H_2(U'_{B,A} \| n_{A1} \| n_{B1} \| 1)$

(f) B: Compute $V'_{A,B} = H_2(U_{B,A} \| n_{A1} \| n_{B1} \| 1)$

If $V_{A,B} = V'_{A,B}$, B knows that A belongs to the same group. Otherwise, A belongs to a different group, *i.e.*, $g_A \neq g_B$ or A belongs to no group.

The protocol succeeds when $V_{B,A} = V'_{B,A}$ and $V_{A,B} = V'_{A,B}$ in steps (d) and (f). Based on the BDH assumption, it succeeds if, and only if, $g_A = g_B$. Otherwise, if it fails, A and B only know $g_A \neq g_B$. Users other than A and B do not know whether $g_A = g_B$ or not, because they cannot compute $V'_{B,A}$ and $V'_{A,B}$ without K_A and K_B. A sketch of proof for $V_{B,A} = V'_{B,A}$ is shown in (5.3). The rest of proof for $V_{A,B} = V'_{A,B}$ can be derived similarly. A detailed security analysis of our SH protocol is given in Appendix B.

$$
\begin{aligned}
V_{B,A} &= H_2(U_{B,A} \| n_{A1} \| n_{B1} \| 0) \\
&= H_2(e(W_{A1}, s_{B1} \cdot K_B) \| n_{A1} \| n_{B1} \| 0) \\
&= H_2(e(s_{A1} \cdot H_1(ID_A), s_{B1} \cdot g_B \cdot H_1(ID_B)) \| n_{A1} \| n_{B1} \| 0) \\
&= H_2(e(s_{A1} \cdot g_B \cdot H_1(ID_A), s_{B1} \cdot H_1(ID_B)) \| n_{A1} \| n_{B1} \| 0) \\
&= H_2(e(s_{B1} \cdot H_1(ID_B), s_{A1} \cdot g_B \cdot H_1(ID_A)) \| n_{A1} \| n_{B1} \| 0) \\
&= H_2(e(W_{B1}, s_{A1} \cdot K_A) \| n_{A1} \| n_{B1} \| 0) \quad \text{// if, and only if, } g_A = g_B \\
&= H_2(U'_{B,A} \| n_{A1} \| n_{B1} \| 0) \\
&= V'_{B,A} \qquad (5.3)
\end{aligned}
$$

Our scheme simply adds a multiplication of the random number s to the elliptic curve point W on top of the original secret handshake protocol [6]. Our protocol requires 2 pairing operations in steps (b) and (d), while other reusable schemes [6, 89] require additional pairing operations or use a composite construct.

C. Protocol Details

In this section, we present the details of our privacy preserving SOA framework. It contains three phases: system initialization, service discovery, and service transaction. The protocol details in each phase are explained as follows.

1. System initialization phase

PARAMETER SETUP: The CA prepares the public and private parameters for CHORD, MLBF, SH, and TZKP. Then, it publishes the public parameters and the following data:

(i) $enc_{key}(msg)$, $dec_{key}(msg)$: symmetric encryption and decryption functions where *key* is a symmetric key and *msg* is the message to be encrypted or decrypted.

(ii) $sign_{pri}(msg)$, $ver_{pub}(msg)$: signature and verification functions where (*pri*, *pub*) is the private-public key pair for signing or verifying the message *msg*.

(iii) *mask*: a *t*-bit binary string with *k* ones in it, where *t* is the total length of MLBF and 2^k is the ring size. *mask* will be used to map the MLBF key to the CHORD key as we will discuss shortly.

REGISTRATION: The user requests the CA to issue a TZKP token and two SH tokens. The first SH token contains a system-wise common group secret which indicates that the user is a registered user. The second SH token contains a group secret that distinguishes different user groups. We use $K = SH1(A, B)$ and $K = SH2(A, B)$ to denote the executions

of SH protocol between nodes *A* and *B* using the first and the second tokens respectively. If the SH protocol fails, $K < 0$. Otherwise, *K* is the common secret between *A* and *B*.

2. Service discovery phase

ADD SERVICE DESCRIPTION: Users who joined the ring can add service descriptions. The first concern is to ensure that the user who added the service description is the only one who can remove it. An intuitive approach is to record the *LC* of the node who adds it and check the LC when the node removes it. This approach is good for a static setting but not when the nodes are highly dynamic. We will show how to use the common secret established by SH protocol to verify this authority even the location is changed.

The second concern is the mapping of service description, *D*, to the CHORD key. An intuitive approach is to map *MLBF*(*D*) as the CHORD key. Nevertheless, *MLBF*(*D*) typically needs thousands bits to achieve an acceptably low false positive rate, making the ring extremely large. Although the lookup complexity does not increase with the ring size but the number of nodes, the finger table size does. Moreover using an extremely large ring costs many big integer operations which degrades the performance. To reduce the ring size, we define a *sampling function*:

$$SAM(x, mask) = c_{i1}c_{i2}\ldots c_{ik}, \tag{5.4}$$

where i_k is the position which bit is "1" in *mask*, and c_{ik} is the bit at position i_k of *x*. For example, if $x = 10011$, and $mask = 11001$, then $i_1, i_2, i_3 = 1, 2, 5$, and the 1^{st}, 2^{nd}, 5^{th} bits of *x* will be extracted to form $SAM(x, mask) = 101$.

In addition to the finger table and successor list, our scheme needs a *service table* and a *directory table* in each node. The service table stores the information related to the services *this node* will provide or transfer. In contrast, the directory table stores the data related to the services which *other nodes* will provide or transfer. The protocol for node A to add a service description D is described as follows:

Protocol: Add Service Description

(i) A computes the CHORD key $SAM(MLBF(D), mask)$ to locate node B.

(ii) A and B compute $K = SH1(A, B)$. Terminate if $K < 0$.

(iii) A sends to B the following messages:

$$MLBF(D),\ pub,\ sign_{pri}(MLBF(D)) \tag{5.5}$$

These messages are needed by service reservations and transfers in the future.

(iv) A randomly generates R and sends $enc_R(K)$ to B. R is needed by A to remove the service description from the ring in the future.

The messages in (5.5) are came from A's reservation credential if A is the service consumer who wants to transfer the reservation. If A is a service provider, (*pri, pub*) is a private-public key pair randomly generated by A. (*pri, pub*) will be used to authenticate the service provider on redemption of services described by D. Different (*pri, pub*) key pairs are used for different service descriptions to guarantee unlinkability of the service provider. At the end of the protocol, the following entries are added to A's service table:

$$D,\ K,\ R,\ pub. \tag{5.6}$$

If *A* is a service provider, then it also stores *pri* in this entry of service table. If *A* is a service consumer, then it stores the TZKP cascaded credential that it used to receive the service reservation. The following entries are added to *B*'s *directory table*:

$$MLBF(D), LC_A, pub, sign_{pri}(MLBF(D)), K, enc_R(K) \quad (5.7)$$

B only knows $MLBF(D)$ but not *D*. Thus, it protects the contents of *D* from *B*, even *B* is responsible for matching *D* with the requests from other nodes in the future.

Fig. 25 depicts the scenario from which node 5 wants to add the XML document *D* to the ring. Node 5 adds the first level element a_0, to BF_1, the second level elements b_0 and b_1, to BF_2, and the third level elements c_0, c_1, d_0 and d_1 to BF_3. It adds all elements to BF_0. The output of each level is concatenated to become:

$$MLBF(D) = \mathbf{0}111\ 010\mathbf{1}\ 00\mathbf{11}\ 01\mathbf{11} \text{ (decimal: 30007)} \quad (5.8)$$

Then, node 5 samples the above result by the mask:

$$mask = \mathbf{1}000\ 000\mathbf{1}\ 00\mathbf{11}\ 00\mathbf{11} \quad (5.9)$$

As shown by the bolded bits above, the sampled result is **011111**, which decimal is 31.

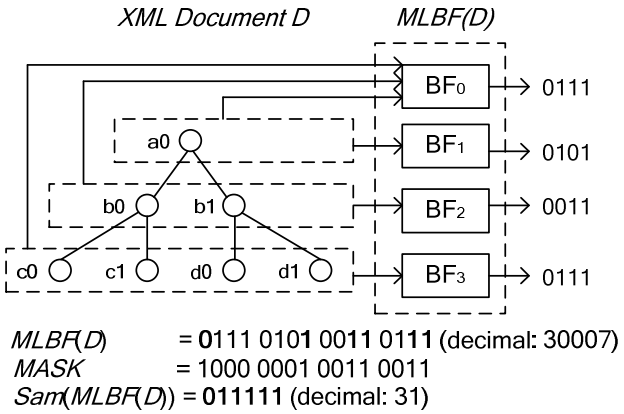


Fig. 25: Mapping XML document to CHORD key

Node 5 uses 31 as the key to locate the node to store 30007 as depicted in Fig. 26 and Table 13. First, node 5 checks that 31 is larger than its location 5. Thus, it looks up its finger table. Since 31 is in between 5 + 16 = 21 and 5 + 32 = 37, node 5 forwards (5, 30007, …) to *suc*(21) = 32. Then, node 32 checks that the sampled result of 30007 is 31, which is smaller than its location 32. It means node 32 is the node to store (5, 30007,…). Node 32 does SH protocol with node 5, exchanges the data as in step (iii) and (iv) of the protocol, and stores the data in its directory table.

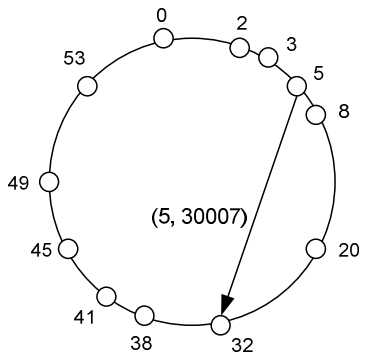


Fig. 26: Add service description.

Table 13: Scenario for adding service description.

Form	**To**	**Lookup Key**	**Finger Table**
5	5	31 > 5	suc(5+16) = 32 suc(5+32) = 38 (21 ≤ 31 ≤ 37)
5	32	31 ≤ 32	--

DELETE SERVICE DESCRIPTION: Only the user who added the service description can delete it. The authentication is done by checking the knowledge of R created in the add service description protocol. Therefore, even if A changes its location, as long as it has the knowledge of R, it can still prove the authority to remove the service description from the ring. The protocol for node A to remove D from the ring is described as follows

Protocol: Delete Service Description

(i) A computes the CHORD key $SAM(MLBF(D), mask)$ to locate node B.

(ii) A and B compute $L = SH1(A, B)$. Terminate if $L < 0$.

(iii) A sends $MLBF(D)$ to B.

(iv) B checks ($MLBF(D)$, …, K, …) from its directory table and send $E_L(K)$ to A.

(v) A decrypts $E_L(K)$ and checks if K matches the entry in service table. If it does, retrieve R and send $E_L(R)$ to B.

(vi) B decrypts $E_L(R)$ and checks whether $E_R(K)$ equals to the entry in its directory table. If it does, delete the entry from the directory table.

At the end of the protocol, the following entries are removed from B's directory table:

$$MLBF(D), LC_A, pub, sign_{pri}(MLBF(D)), K, enc_R(K) \quad (5.10)$$

SERVICE MATCHING: A registered user can post service request for service matching. The first concern is on the matching of multiple results. In CHORD, lookup is a one-to-one matching. On the contrary, XML-based query matches multiple service descriptions resided in different nodes on the ring. For example, the service request denoted by 0101 matches all service descriptions represented by 0101, 0111, 1101, and 1111. An intuitive approach is to rewrite the original request to all matched combinations. However, it will

generate massive number of requests when many zeros are in the original request. An alternative approach is to circulate the request around the ring and receive replies from the nodes which have a match in their service tables. Yet, it generates as many requests as the number of nodes. Thus, we extend CHORD's lookup protocol to reach multiple matched results. Our solution also circulates the request but we skip a large number of nodes by using finger tables. The request is circulating around the successors of matched key, *e.g.*, *suc*(0101), *suc*(0111), *suc*(1101), and *suc*(1111) instead of circulating around every node in the ring. The number of nodes visited can be reduced if there are common successors. For example, it is likely that *suc*(0101) = *suc*(0111) because 0101 and 0111 are near in the ring. In order to derive the potentially matched CHORD keys from the original request, we define the $\oplus$ operator as follows:

$$z = y \oplus x, \tag{5.11}$$

where the number of bits in x equals to the number of zeros in y, and z is obtained by replacing the zero bits in y by the corresponding bits in x. For example, $0101 \oplus 00 = 0101$, $0101 \oplus 01 = 0111$, $0101 \oplus 10 = 1101$, and $0101 \oplus 11 = 1111$.

The second concern is about the number of results returned. A consumer may receive an extremely large number of matched results if the request is too general, *e.g.*, 0000, which could crash the node unintentionally. A straightforward approach is to hardcode the protocol to return the first k results. But then the node may not be able to reach other matched results every time using the same request although the unreachable results may be more useful. Therefore, we take a probabilistic approach that different k results are returned each time. To avoid the request from circulating forever in the ring,

we also keep track of a hop count to cease further searching of results, even fewer than k results have been return. The protocol for node A to match a service request T on the ring is described as follows.

Protocol: Service Matching

(i) A uses $SAM(MLBFQ(T), mask)$ to locate node B.

(ii) A and B compute $K = SH1(A, B)$. Terminate if $K < 0$.

(iii) A sends the following to B:

(LC_A, $MLBFQ(T)$, *inc_count*, *result_count*, *hop_count*)

result_count = max number of results to be returned

hop_count = max number of nodes to be visited.

inc_count = number of matched key tested so far

inc_count is initially 0 from the original requestor A.

(iv) B checks its directory table. For each table entry, if $SAM(MLBF(D), mask)$ and $SAM(MLBFQ(T), mask)$ match and *result_count* > 0, then B sends the following entries to LC_A with probability = p:

$$MLBF(D), LC, pub, sign_{pri}(MLBF(D)), MLBFQ(T) \quad (5.12)$$

For each result sent, B decreases *result_count* by one.

Terminate if *hop_count* = 0, or *result_count* = 0, or *inc_count* cannot be further increased, *i.e.*, 11…1.

(v) B increases *inc_count* by one and then locates $suc(x)$ where $x = SAM(MLBFQ(T), mask) \oplus inc\_count$.

(vi) B and *suc(x)* compute $K = SH1(A, B)$. Terminate if $K < 0$.

(vii) *B* decreases *hop_count* by one and sends *suc*(*x*) the following messages:

$$(LC_A, MLBFQ(T), result\_count, hop\_\text{count}, inc\_count) \quad (5.13)$$

LC_A instead of LC_B is used in the sent message so that the results will be sent to the original requestor.

(viii) *B* and *suc(x)* repeat from (iv) as *A* and *B* did.

At the end of the protocol, A receives a collection of results which potentially match the wanted services.

Fig. 27 depicts the scenario from which node 2 matches the service request T = /a_0/b0/c0 on the ring. The element names a_0, b_0, c_0 are respectively added to BF_1, BF_2, and BF_3. All element names are added to BF_0. The output of each level is concatenated:

$$MLBFQ(T) = \mathbf{0}111\ 010\mathbf{1}\ 00\mathbf{01}\ 00\mathbf{11} \text{ (decimal: 29971)} \quad (5.14)$$

Then, node 2 samples the result in (5.14) by the mask in (5.15):

$$mask = \mathbf{1}000\ 000\mathbf{1}\ 00\mathbf{11}\ 00\mathbf{11} \quad (5.15)$$

As shown by the bolded bits above, the sampled result is **010111**. As depicted in Table 14, by substituting the 0 bits in 010111 by 00, 01, 10 and 11, there are four keys derived from the original request: 23, 31, 55, and 63 (in decimal). Instead of routing the request from node 2 to *suc*(23), 2 to *suc*(31), 2 to *suc*(55), and 2 to *suc*(63), we forward the request from 2 to *suc*(23), *suc*(23) to *suc*(31), *suc*(31) to *suc*(55), and *suc*(55) to *suc*(63). The detailed steps are presented in Fig. 28 and Table 15.

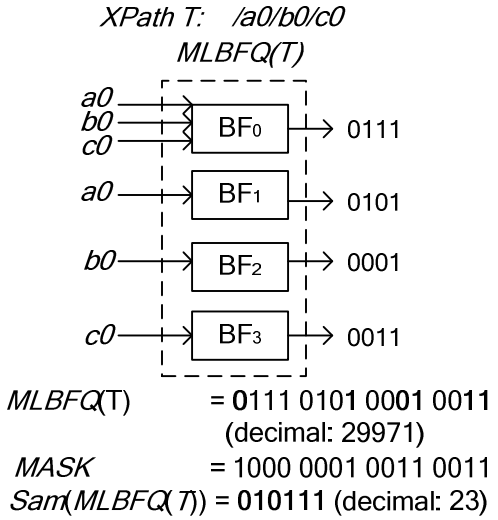


Fig. 27: Mapping from XPath to CHORD key.

Table 14: Computations for the next CHORD key.

Binary	**Decimal**	**⊕ Representation**
01**0**111	23	23 ⊕ 0
01**1**111	31	23 ⊕ 1
11**0**111	55	23 ⊕ 2
11**1**111	63	23 ⊕ 3

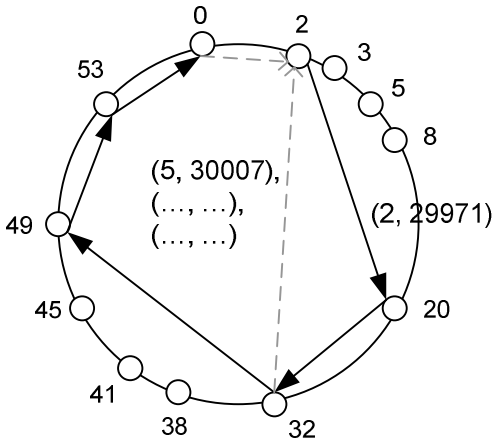


Fig. 28: Matching service request.

Table 15: Scenario for matching service request.

From	To	Lookup Key	Directory Table	Finger Table
2	2	23⊕ 0=23 > 2	--	suc(2+16) = 20 suc(2+32) = 20 (18≤23≤34)
2	20	23⊕ 0=23 > 20	--	suc(20+2) = 32 suc(20+4) = 32 (22≤23≤24)
20	32	23⊕ 0=23≤ 32 23⊕ 1=31≤ 32 23⊕ 2 =55> 32	21, **23**, **23**, 25, 30, **31**	suc(32+16)=49 suc(32+32)=2 (48≤55≤64)
32	49	23⊕ 2=55 > 49	--	suc(49+4)=53 suc(49+8)=0 (53≤55≤57)
49	53	23⊕ 2=55 > 53	--	suc(53+2)=0 suc(53+4)=0 (55≤55≤57)
53	0	23⊕ 2=55≤ 0+64 23⊕ 3=63≤ 0+64	54, 57, 59, **63**	--

First, node 2 checks that 23 is larger than its location. Thus, it looks up its finger table. Since 23 is in between 2 + 16 = 18 and 2 + 32 = 34, node 2 forwards (2, 29971, 0, …) to *suc*(18) = 20. Node 20 checks that the sampled result of 29971 is 23. 23 ⊕ 0 is larger than its location so it looks up its finger table. Since 23 is in between 20 + 2 = 22 and 20 + 4 = 24, it forwards (2, 29971, 0, …) to *suc*(22) = 32.

Node 32 checks that the sampled result of 29971 is 23. 23 ⊕ 0 is smaller than its location. Thus, it looks up its directory table and finds two entries match. The results are returned to node 2. Next, it updates *inc_count* from 0 to 1, and computes 23 ⊕ 1 = 31. It finds that *suc*(31) is node 32 itself thus it looks up its directory table and finds one entry match. This entry was added by node 5 in the previous example so node 32 returns the result (5, 30007, …) to node 2. Node 32 updates *inc_count* from 1 to 2, and computes 23 ⊕ 2 = 55. Since 55 is in between 32 + 16 = 48 and 32 + 32 = 64, node 32 forwards (2, 29971, 2,…) to *suc*(48) = 49.

Similarly, the request is forwarded from node 49 to node 53 and then to node 0. Node 0 checks that the sampled result of 29971 is 23. 23 ⊕ 2 = 55 is smaller than its location 0 + 64 (the addition of 64 is needed if the sending node's location is larger than the receiving node's location) so it looks up its directory table and finds no entry match. Node 0 updates *inc_count* from 2 to 3 and computes 23 ⊕ 3 = 63. It finds that *suc*(63) is node 0 itself. Thus, it looks up its directory table and finds one entry match. The result is returned to node 2. At this point, node 0 cannot further increments *inc_count*. Thus, the protocol is terminated.

GROUP MATCHING: Using the *LC*s resulted from service matching, user *A* can contact some of these locations and execute $L = SH2(A, B)$. If *B* is in the same group, *B* will send *D* to *A*. *A* does an exact matching of *D* and *T*. If they are matched (not only potentially matched), then *A* and *B* can continue to the transaction phase for reservation of services or transfer of reservation. *L* will be used as the symmetric key to build a secure channel for the transaction phase. The secure channel will not be explicitly mentioned in the rest of the discussions.

JOIN/LEAVE/FAILRE OF NODE: Identical to CHORD but the users located at the ends of the broken ring will need to execute $K = SH1(A,B)$ to verify their precedent and the successor as registered users. In additional to the key-value pair, they also need to move their service tables and directory tables to the precedent and the successor.

3. Service transaction phase

SERVICE RESERVATION: Identical to TZKP except that *pub* is not well-known public-key but received during the service matching phase. The reservation credential becomes

$$RS = (pub, sign_{pri}(MLBF(D)), sign_{pri}(W, X, T, MLBF(D)), T, MLBF(D)) \quad (5.16)$$

RESERVATION TRANSFER: Identical to TZKP except that the consumer who transfers the service reservation needs to remove its service description after the transfer.

SERVICE REDEMPTION: Identical to TZKP except that the consumer needs to verify the service provider is the one who originally posted the service description. It can be checked by sending a random string to the service provider. If the service provider can

correctly sign the random string by the private key *pri*, then it is the one who originally posted the service description.

D. Experimental Results

Matching of multiple results for service discovery in P2P networks is an open problem [93]. It is easy to see that our protocol takes $O(n)$ time to match all results in the worst case, where n is the number of nodes on the ring. Nevertheless this complexity is inevitable because the service consumer can always choose the request which returns all services from all nodes. Thus, average run time is a more interesting attribute to study.

In this experiment, we evaluate the run time for the service matching protocol to demonstrate its feasibility in a large-scale P2P environment. The run time is estimated by multiplying the average number of hops required to match the services by the average run time required to talk to a node one hop away and do the secret handshake. Note that the average run time can be influenced by different distributions of nodes on the ring, different natures of service descriptions and requests, and different networking quality. The simulation does not cover all situations but provides a good reference to understand the performance of the system in practice.

1. Average hop count

In this simulation, we evaluate the average number of hops that a request needs to route through for all potentially matched services. The average values are obtained by 1000 runs of the experiment on a 15-bit simulated CHORD ring. In each run, a certain number of nodes are randomly distributed on the ring and a *mask* is randomly generated. To demonstrate a more realistic XML workload, we select 45 XML documents from the *XML common business library* (xCBL) [94] and add them to the ring. Then, we derive 77 XPaths (shown in Appendix C) from the 45 XML documents as the service requests. We randomly select a node from the ring fire each request. The MLBF contains 8 layers. Each layer contains 300 hash functions. Each hash function is a variant of SHA-1 [95] to produce a 160-bit output. The 160-bit MLBF output is sampled by the *mask* into a 15-bit key for lookup on the CHORD ring. The hop counts for different number of nodes on the ring are shown in Table 16. It shows that for every 10 times increase nodes, the average hop counts increases by 3 folds, when the number of nodes increases from 10 to 10000. Such a relationship is shown in Fig. 29 by the log scales on both axes. In reality, we may need fewer hop counts because we are interested in *k* results only but not all of them.

Table 16: Average hop count for returning all results.

Number of Nodes	10	100	1000	10000
Number of Hops	5	17	49	141

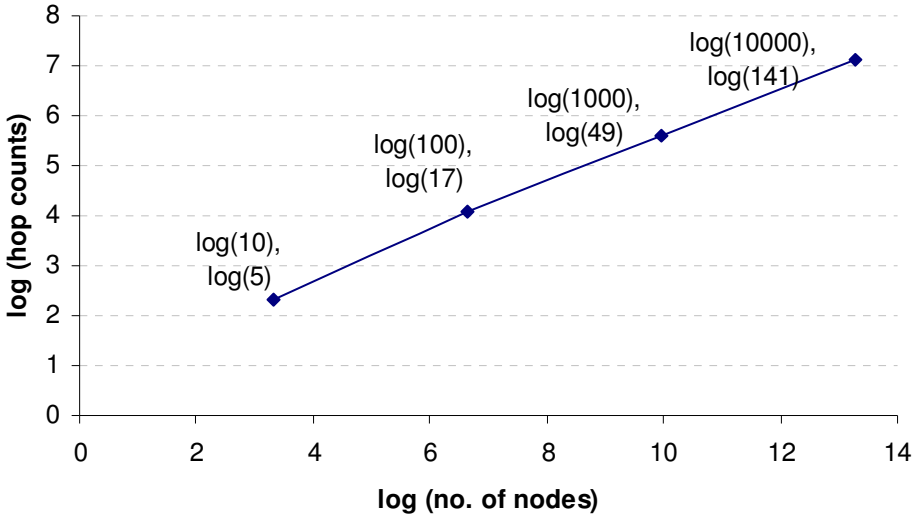


Fig. 29: Average hop count for returning all results.

2. Average runtime per hop

In this experiment, we evaluate the average run time needed by a node to talk with a node one hop away and do secret handshake. We implemented the proposed SH protocol in C++ with MIRACL library [96]. We run the protocol on an Intel Pentium-4 2-GHz processor with 256-Mbyte RAM under Windows XP environment. The bilinear map e is Tate pairing. G_1 is an additive group of points of a supersingular elliptic curve with prime order $q = 2^{159} + 2^{17} + 1$, and G_2 is a multiplicative group of the finite field $F^*_{q^2}$. The Tate pairing is computed based on the supersingular elliptic curve $y^2 = x^3 + x$. The pairing parameters chosen above are based on [92] which deliver a security level comparable to the 1024-bit RSA cryptography. We used the built-in hash function in the MIRACL for H_1. We used SHA-1 [95] for H_2. Table 17 and Table 18 summarize the

experimental results. We measure the average time and message size in 100 runs of secret handshakes between two group members. Main computations are 2 pairing operations, 2 elliptic curve point multiplications, H_1 and H_2. The pairing operations are the most costly operations, but they consume 65 milliseconds only. Our protocol is more efficient than the schemes in [7], which have similar functions but cost 6 pairing operations. The total message size is smaller than 350 bytes which is compact enough for most applications with reasonable bandwidth.

Table 17: Average runtime of our secret handshake protocols

	Pairing Computations	Point Multiplications	H_1, H_2	Total
Time (ms)	32.5×2	$<< 1$	$<< 1$	65

Table 18: Average message size of our secret handshake protocols

	n_{A1},n_{B1}	W_{A1},W_{B1}	$V_{A,B}$,$V_{B,A}$	Total
Size (bytes)	40 bytes	256 bytes	40 bytes	336 bytes

To evaluate the average run time for a node to reach another node one hop away, we simulate a 15-bit CHORD ring with 14 nodes on 100Mb Ethernet. It takes about 2.4 milliseconds reach from one node to another node. Therefore, we estimate that each hop count requires 65 + 2.4 = 67.4 milliseconds. Based on this estimation, we summarize the total run times in Table 19 for different scenarios from Table 16.

Table 19: Total Runtime for different number of nodes

Number of Nodes	10	100	1000	10000
Number of Hops	5	17	49	141
Avg. Run Time (s)	0.3	1.1	3.3	9.5

As shown above, our protocol returns all potentially matched services from 1000 nodes in fewer than 4 seconds, showing the practicality to be deployed in a large scale P2P environment.

E. Summary

In this section, we propose a management framework for the privacy-preserved service oriented architecture (SOA). Service providers and consumers first establish a trust relationship in the peer-to-peer (P2P) network CHORD, before they are willing to exchange sensitive data. The key challenge is to maintain the balance of security and privacy in a distributed and dynamic P2P environment without centralized supervisions during the regular operations. To achieve this, we propose to use multi-layer Bloom filters (MLBF) to match service requests/descriptions without unveiling their contents during the service discovery phase. We also propose to use secret handshake (SH) protocol to match group membership between service providers and consumers without unveiling their group identities on mismatched events. After service matching and group matching, the two users can execute the transaction by the timed zero-knowledge proof

(TZKP) protocol to perform such functions as service reservation, redemption, and transfers of reservations, without unveiling their user identities under normal situations. Integration of above cryptographic tools forms strong foundation of security and privacy protection for the next generation communication model. Preliminary experimental results show that our system is practical for a large P2P network.

VI. SUMMARY

The objective of this research work is to develop an anonymous, authentic, and accountable (AAA) management framework for secure resource allocations based on the E-cash paradigm. While most existing resource management schemes emphasize on one or two of the AAA attributes, E-cash provides solid knowledge bases to maintain fragile balance between them. Nevertheless, since E-cash was originally designed for monetary applications, directly applying E-cash to secure resource allocations may not be the most efficient and effective way. Therefore, we proposed several management solutions to tailor E-cash algorithms for secure resource management.

Transferability management is important for transferring of resource ownership from principal to another principal. E-cash algorithms allow anonymous transfer without centralized supervisions but the transfer operation is expensive under Chaum-Pederson's general transferability model (GTM). We proposed a timed zero-knowledge proof (TZKP) protocol which drastically reduces the storage and communication overheads needed in the traditional E-cash model. The key idea is manipulate the anonymity control variables in Eng-Okamoto's general disposable authentication (GDA) model so that session time and source of transfer can be embedded into the cryptographic construct as a deciphering condition of user identity. With proper adjustment on the deciphering condition, the user can reuse a token for multiple legitimate transfers without losing anonymity as in GTM. At the same time, the service provider can discard expired credentials without sacrificing the accountability on double-transfer violators.

Divisibility management allows a principal to organize a chunk of resources into different divisions with minimum number of tokens. Traditional divisibility management solutions use a binary tree to represent different subdivisions of the resources. We proposed a hypercube based divisibility framework which supports much more flexible divisibility configurations than Eng-Okamoto's general divisibility model (GDM). The flexibility in is traded from the overheads in tracking of a new type of double-transfer violation called shared-node violation. We analyzed the cryptographic constraints and found that it is very costly to guarantee all AAA constraints at the same time in the hypercube-based scheme. However, using a slightly relaxed anonymity constraint, and integrating the scheme with practically used resource allocation rules, we found that the overheads can be significantly reduced.

Based on the above AAA management solutions, we proposed a privacy-preserving service oriented architecture on the peer-to-peer (P2P) network. By TZKP protocol, user identities can be protected in transaction phase. To offer privacy protection in the service discovery phase, we extended CHORD's lookup protocol to enable XML query using the multi-layer Bloom filter (MLBF). The proposed solution allows service consumers to query peer nodes for wanted services while the peer nodes do not have the knowledge on both the query and service contents. In addition, we proposed a new secret handshake (SH) protocol to screen strangers based on their qualification, privileges, capabilities, or trust levels represented by their group, before the ownership of resources is transferred to the unknown collaborators. SH protocol allows two users to verify whether they belong to the same group while not leaking their

group identities upon failure of verifications. Our protocol allows the user to reuse the SH token for multiple secret handshake instances without linking the user identity. We showed that our SH protocol is more efficient than existing schemes with similar functions. Experimental results showed that E-cash based secure resource management framework is practical for AAA management in the large-scale distributed network.

REFERENCES

[1] D. Chaum and T. Pederson, "Transferred cash grows in size," *Advances in Cryptology - EUROCRYPT '92*, Balatonfüred, Hungary, May 1992, pp. 390 – 407.

[2] T. Eng and T. Okamoto, "Single-term divisible electronic coins," *Advances in Cryptology - EUROCRYPT '94*, Perugia, Italy, May 1994, pp. 311-323.

[3] M. Chen and K. G. Shin, "Subcube allocation and task migration in hypercube machines," *IEEE Transactions on Computers*, vol. 39, no. 9, pp. 1146-1155, September 1990.

[4] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for Internet applications," *IEEE/ACM Transactions on Networking*, vol. 11, pp. 17-32, February 2003.

[5] G. Koloniari and E. Pitoura, "Filters for XML-based service discovery in pervasive computing," *Computer Journal: Special Issue on Mobile and Pervasive Computing*, vol. 47, no. 4, pp. 461 – 474, 2004.

[6] D. Balfanz, G. Durfee, N. Shankar, D. Smetters, J. Staddon, and H. Wong, "Secret handshakes from pairing-based key agreements," *IEEE Symposium on Security and Privacy*, Berkeley, CA, May 2003, pp.180-196.

[7] G. Ateniese and M. Blanton "Secret handshakes with dynamic and fuzzy matching," *14th Annual Network and Distributed System Security Symposium,* San Diego, CA, February 2007.

[8] I. F. Blake, G. Seroussi, and N. P. Smart, *Elliptic curves in cryptography*, Cambridge University Press, 1999, New York.

[9] D. Chaum. "Blind signatures for untraceable payments," *Advances in Cryptology – CRYPTO'82*, Santa Barbara, CA, USA, August 1983, pp. 199-203.

[10] M. Froomkin, "The unintended consequences of e-cash," *Computers, Freedom and Privacy Conference*, Burlingame, CA, March 1997.

[11] R. Y. Chan, J. C. Wong, and A. C. Chan, "Anonymous electronic voting with non-transferable passes," *Proc. of the IFIP Tc11 15th Annual Working Conference on Information Security for Global Information Infrastructures*, Kluwer B.V., Deventer, The Netherlands, August 2000, pp. 331-340.

[12] K. B. Frikken and M. J. Atallah, "Privacy preserving electronic surveillance," *Proc. of the 2003 ACM Workshop on Privacy in Electronic Society*, *WPES '03*, New York, 2003, pp. 45- 52.

[13] Y. Shen, T. C. Lam, J-C, Liu, and W. Zhao, "On the confidential auditing of distributed computing systems," *24th International Conference of Distributed Computing Systems (ICDCS) 2004*, Washington, DC, March 2004, pp. 600-607.

[14] G. Ahn, , B. Mohan, and S. Hong, "Towards secure information sharing using role-based delegation," *Journal of Network and Computer Applications,* vol. 30, no. 1, pp. 42-59, Jan. 2007.

[15] T. C. Lam and V. K. Wei, A mobile agent clone detection system with itinerary privacy," *IEEE 11th Int'l Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE-2002)*, Pittsburgh, PA, June 2002, pp.68 - 73.

[16] D. Chaum, "Blind signature system," *Advances in Cryptology- CRYPTO'83*, Santa Barbara, CA, USA, August 1983, pp. 153.

[17] S. von Solms and D. Naccache, "On blind signatures and perfect crimes," *Computers & Security*, vol. 11, pp. 581-583, 1992.

[18] J. L. Camenisch, J.-M. Piveteau, and M.A. Stadler, "Blind signatures based on the discrete logarithm problem," *Advances in Cryptology - EUROCRYPT'94,* Perugia, Italy, May 1994, pp. 428 – 432.

[19] M. Stadler, J.-M. Piveteau, and J. Camenisch, "Fair blind signatures," *Advances in Cryptology - EUROCRYPT'95,* St. Malo, France, May 1995, pp. 209-219.

[20] D. Pointcheval, and J. Stern, "New blind signatures equivalent to factorization (extended abstract)," *Proc of the 4th ACM conference on Computer and Communications Security,* Zurich, Switzerland, April 1997, pp.92-99.

[21] F. Zhang and K. Kim, "Efficient ID-based blind signature and proxy signature from bilinear pairings," *Proc. of the 8th Australian Conference on Information Security and Privacy* (*ACISP'03*), Wollongong, Australia, July 2003, pp. 312-323.

[22] J-J Quisquater, L. C. Guillou, and T. A. Berson, "How to explain zero-knowledge protocols to your children," *Advances in Cryptology - CRYPTO '89*, Santa Barbara, CA, USA, August 1989, pp. 628-631.

[23] U. Feige, A. Fiat, and A. Shamir, "Zero knowledge proofs of identity," *Journal of Cryptology*, vol. 1, no. 2, 1988, pp. 77-94.

[24] O. Goldreich and Y. Oren, "Definitions and properties of zero-knowledge proof systems," *Journal of Cryptology*, vol. 7, no. 1, 1994, pp. 1-32.

[25] M. Bellare, M. Jackobsson, and M. Yung, "Round-optimal zero-knowledge arguments based on any one-way function," *Advances in Cryptology – EUROCRYPT'97*, Konstanz, Germany, May 1997, pp. 280-305.

[26] O. Goldreich and H. Krawczyk, "On the composite of zero knowledge proof systems," *SIAM Journal on Computing*, vol. 25, no. 1, 1996, pp. 162-192.

[27] Martin Tompa, "Zero knowledge interactive proofs of knowledge (a digest)," *Proc of the 2nd Conference on Theoretical Aspects of Reasoning about Knowledge*, Pacific Grove, CA, USA, March 1988, pp. 1 – 12.

[28] A. Shamir, "How to share a secret," *ACM Communications,* vol. 22, no. 11, pp. 612 – 613, November 1979.

[29] G. R. Blakley, "Safeguarding cryptographic keys," *Proc of American Federation of Information Processing Societies*, vol. 48, pp. 313-317, 1979.

[30] G. R. Blakley and G. A. Kabatianski, "Linear algebra approach to secret sharing schemes," *Error Control, Cryptology, and Speech Compression*, vol. 829, pp. 33-40, 1994.

[31] G. R. Blakley and G. A. Kabatianski, "On general perfect secret sharing schemes," *Advances in Cryptology - CRYPTO'95*, Santa Barbara, CA, August 1995, pp. 367-371.

[32] M. Carpentieri, "A perfect threshold secret sharing scheme to identify cheaters," *Designs, Codes and Cryptography*, vol. 5, pp. 183-188, 1995.

[33] C. Dwork, "On verification in secret sharing," *Advances in Cryptology - CRYPTO '91*, Santa Barbara, CA, August 1992, pp. 114-128.

[34] E. D. Karnin, J. W. Greene and M. E. Hellman, "On secret sharing systems," *IEEE Transactions on Information Theory*, vol. 29, pp. 35 – 41, 1983.

[35] M. Stadler, "Publicly verifiable secret sharing," *Advances in Cryptology - EUROCRYPT'96*, Zaragoza, Spain, May 1996, pp. 190-199.

[36] N. Ferguson, "Single term off-line coins," *Advances in Cryptology – EUROCRYPT'93*, Lofthus, Norway, May 1994, pp. 318-328.

[37] J. Camenisch, S. Hohenberger, and A. Lysyanskaya, "Balancing accountability and privacy using E-cash," *Security and Cryptography for Networks (SCN) 2006,* Maiori, Italy, September 2006, pp. 141-155.

[38] T. Okamoto and K. Ohta, "Universal electronic cash," *Advances in Cryptology - CRYPTO' 91*, Santa Barbra, CA, USA, August 1991, pp. 324 – 337.

[39] N. Ferguson, "Extensions of single-term off-line coins," *Advances in Cryptology - CRYPTO' 93*, Santa Barbra, CA, August 1993, pp. 292-301.

[40] A. Chan, Y. Frankel, and Y. Tsiounis, "Easy come – easy go divisible cash," *Advances in Cryptology - EUROCRYPT'98*, Helsinki, Finland, May 1998, pp. 561 – 575.

[41] T. Nakanishi, N. Haruna, and Y. Sugivama, "Unlinkable electronic coupon protocol with anonymity controls," *Proc.* of the *2nd International Workshop on Information Security*, Kuala Lumpur, Malaysia, November 1999, pp. 37 – 46.

[42] T. Nakarishi and Y. Sugiyama, "Unlinkable divisible electronic cash," *Proc. of the 3rd International Workshop on Information Security*, Sydney, Australia, December 2000, pp. 121 – 134.

[43] J. Byun, D. Lee, J. Lim, and C. Park, “Efficient transferable cash with group signatures,” *Proc. of the 4th International Conference on Information Security,* October 2001, pp. 462 – 474.

[44] J. Liu, S. Wong, and D. Wong, “A new transferable e-cash scheme,” *Proc. of the 2nd International Conference on Applied Cryptography and Network Security (ACNS) – Technical Track*, Yellow Mountain, China, June 2004, pp. 408 – 415.

[45] T. C. Lam, "A note on the n-spendable extension of Ferguson's single term off-line coins," *Cryptology e-Print Archive, Report 2005/439*, 2005.

[46] K. Wei, “More compact e-cash with efficient coin tracing.” *Cryptology e-Print Archive*, *Report 2005/439*, 2005.

[47] J. Camenisch and A. Lysyanskaya, “Dynamic accumulators and application to efficient revocation of anonymous credentials,” *Advances in Cryptology - CRYPTO’ 2002*, Santa Barbara, CA, USA, August 2002, pp. 61 – 71.

[48] T. C. Lam and V. K. Wei, "Mobile agent clone detection using general transferable e-cash", *International Conference on Information Security (InforSecu '02)*, Shanghai, China, June 2002.

[49] G. Tsudik and S. Xu, “Flexible framework for secret handshakes (multi-party anonymous and un-observable authentication),” *Cryptology e-Print Archive*, *Report 2005/034*, 2005.

[50] D. Chaum, A. Fiat, and M. Naor, “Untraceable electronic cash,” *Advances in Cryptology – CRYPTO’88*, Santa Barbara, CA, USA, August 1989, pp. 319-327.

[51] S. Brands, "Untraceable off-line cash in wallets with observers," *Advances in Cryptology – CRYPTO'93*, Santa Barbara, CA, August 1993, pp. 302-318.

[52] C. Schnorr, "Efficient signature generation by smart cards," *Journal of Cryptology*, vol. 4, no. 3, pp. 161-174, 1991.

[53] (2006) TZKP Demo. [Online]. Available: http://rtds.cs.tamu.edu/tzkp.php
Accessed: May 2006

[54] T. Benzel, R. Braden, D. Kim, A. Joseph, C. Neuman, R. Ostrenga, S. Schwab, and K. Sklower, "Design, deployment, and use of the DETER testbed," *Proc. of the DETER Community Workshop on Cyber Security Experimentation and Test*, Boston, MA, August 2007.

[55] (2002) C. Tan, C# Big Integer Class. [Online].
Available: http://www.codeproject.com/csharp/biginteger.asp?target=biginteger
Accessed: May 2006

[56] C. Dwork, M. Naor, and A. Sahal, "Concurrent zero-knowledge," In *Proc. of the 30th Annual Symposium on Theory of Computing*, Dallas, Texas, May 1998, pp. 409-418.

[57] I. Damgård, K. Dupont, and M. Perdersen, "Unclonable group identification," *Cryptology e-Print Archive*, *Report 2005/170*, 2005.

[58] Z. Tan and Z. Liu, "Provably secure delegation-by-certificate proxy signature schemes," In *Proc. of the 3rd International Conference on Information Security*, Shanghai, China, November 2004, pp. 38-43.

[59] M. Mambo, K. Usuda, and E. Okamoto. "Proxy signatures for delegating signing operation," *Proc. of the 3rd ACM Conference on Computer and Communications Security* (*CCS'96*), New Delhi, India, March 1996, pp. 48-57.

[60] Z. Shao, "Proxy signature schemes based on factoring," *Information Processing Letters*, vol. 85, pp. 137-143, 2003.

[61] H.-M. Sun and B.-T. Hsieh. "On the security of some proxy signature schemes," *Cryptology e-Print Archive*, *Report 2003/068*, 2003.

[62] Guilin Wang, Feng Bao, Jianying Zhou, and Robert H. Deng, "Security analysis of some proxy signatures," *Information Security and Cryptology*, Seoul, Korea, November 2003, pp. 305-319.

[63] Huaxiong Wang and Josef Pieprzyk, "Efficient one-time proxy signatures," *ASIACRYPT 2003*, Taipei, Taiwan, November 2003, pp. 507-522.

[64] J. Kiu, V. Wei, and D. Wong, "Linkable spontaneous anonymous group signature for ad hoc groups (extended abstract)." *Information Security and Privacy (ACISP),* Sydney, Australia, July 2004, 325-335.

[65] D. Chaum and E. van Heyst, "Group signatures," *Advances in Cryptology - EUROCRYPT'91,* Brighton, United Kingdom, April 1991, pp. 257-265.

[66] L. Chen and T. P. Pedersen, "New group signature schemes," *Advances in Cryptology - EUROCRYPT'94,* Perugua, Italy, May 1994, pp. 171-181.

[67] J. Camenisch, "Efficient and generalized group signatures," *Advances in Cryptology - EUROCRYPT'97,* Konstanz, Germany, May 1997, pp. 465-479.

[68] W-B. Lee and C-C. Chang, "Efficient group signature scheme based on the discrete logarithm," *IEE Proceedings Computer and Digital. Techniques*, vol. 145, no. 1, pp. 15-18, 1998.

[69] G. Ateniese and G. Tsudik, "Some open issues and new directions in group signature schemes," *Financial Cryptography' 99*, Anguilla, British West Indies, February 1999, pp. 196-211.

[70] J. Camenisch, S. Hohenberger, M. Kohlweiss, A. Lysyanskaya and M. Meyerovich, "How to win the clone wars: efficient periodic n-times anonymous authentication," *Cryptology e-Print Archive*, *Report 2006/454*, 2006.

[71] T. Okamoto and K. Ohta, "One-time zero-knowledge proof authentications and their applications to untraceable electronic cash," *IEICE Transactions on Fundamentals of Electronics*, *Communications and Computer Sciences*, vol. E81-A, no. 1, pp. 2 - 10, 1998.

[72] P-J Chuang and N-F Tzeng, Dynamic processor allocation in hypercube computer*s*, *ACM SIGARCH Computer Architecture News*, vol. 18, no. 3, pp.40-49, 1990.

[73] C-H Huang, J-Y Juang, "A partial compaction scheme for processor allocation in hypercube multiprocessors," *International Conference on Parallel Processing*, University Park, PA, August 1989, pp. 211-217.

[74] (1996) Hamming distance between ternary numbers. [Online].
Available: http://www.its.bldrdoc.gov/fs-1037/dir-017/_2529.htm
Accessed: May 2006

[75] T. Erl, *Service-Oriented Architecture: Concepts, Technology, and Design.* Prentice Hall, 2005, New York.

[76] (2004) M. Colan, Service-oriented architecture expands the vision of Web services. [Online].
Available: http://www-128.ibm.com/developerworks/webservices/library/ws-soaintro2/ Accessed: October 2007

[77] (2001) UDDI Technical White Paper. [Online].
Available: http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.pdf
Accessed: October 2007

[78] S. E. Czerwinsi, B. Y. Zhao, T. D. Hodes, A. D. Joseph, and R. H. Katz, "An architecture for a secure service discovery service," *International Conference on Mobile Computing and Networking*, Seattle, WA, August 1999, pp. 24 – 35.

[79] F. Zhu, M. Mutka, and L. Ni, "Splendor: a secure, private, and location-aware service discovery protocol supporting mobile services," *Proc. of the 1st IEEE International Conference on Pervasive Computing and Communications*, Fort Worth, TX, March 2003, pp. 235-242.

[80] I. Stoica. R. Morris, D. Liben-Nowell, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: a scalable peer-to-peer lookup service for Internet applications," *Proc. of ACM SIGCOMM'01*, San Diego, CA, August 2001, pp. 149-160.

[81] H. Balakrishnan, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Looking Up Data in P2P Systems," *Communications of the ACM*, vol. 46, no. 2, pp. 43-48 2003.

[82] P. Rao and B. Moon, psiX: hierarchical distributed index for efficiently locating XML data in peer-to-peer networks, *Technical Report 05-10*, Department of Computer Science, The University of Arizona, 2005, Tucson, AZ.

[83] G. Koloniari and E. Pitoura, "Peer-to-Peer Management of XML Data: Issues and Research Challenges." *SIGMOD Record*, vol. 34, no. 2, 2005.

[84] G. Koloniari and E. Pitoura, "Filters for XML-based service discovery in pervasive computing," *Computer Journal: Special Issue on Mobile and Pervasive Computing*, vol. 47, no. 4, pp. 461 – 474, 2004.

[85] Y. Zhang, W. Liu and W. Lou, "MASK: anonymous on-demand routing in mobile ad hoc networks," *IEEE Transactions on Wireless Communications*, vol. 5, no. 9, pp.2376-2385, 2006.

[86] G. Liang and S. Chawathe, "Anonymous routing: a cross-layer coupling between application and network layer," *Conference on Information Sciences and System*, Princeton University, NJ, March 2006, pp. 783 - 788.

[87] Y. Zhang, W. Liu, W. Lou and Y. Fang, "Anonymous handshakes in mobile ad hoc networks," *IEEE Military Communications Conference*, Monterey, CA, October 2004, pp. 1193 – 1199.

[88] S. Xu and M. Yung, "k-anonymous secret handshakes with reusable credentials," *ACM Conference on Computer and Communications Security*, Washington, D. C., October 2004, pp.158-167.

[89] G. Tsudik and S. Xu, "Flexible framework for secret handshakes (multi-party anonymous and un-observable authentication)," *Cryptology e-Print Archive*, *Report 2005/034*, 2005.

[90] S. Jarecki, J. Kim, and G. Tsudik, "Authentication for paranoids: multi-party secret handshakes," *International Conference on Applied Cryptography and Network Security*, Singapore, June 2006, pp. 325 – 339.

[91] D. Vergnaud, "RSA-based secret handshakes," *International Workshop on Coding and Cryptography*, Bergen, Norway, March 2005, pp. 252-274.

[92] G. Frey, M. Muller, and H. G. Ruck, "The Tate pairing and the discrete logarithm applied to elliptic curve cryptosystems." *IEEE Transactions on Information Theory*, vol. 45, no. 5, pp. 1717 – 1719, 1999.

[93] J. Li, B. T. Loo, J. Hellerstein, F. Kaashoek, D. Karger, and R. Morris, "On the feasibility of peer-to-peer web indexing and search." *Proc of International Workshop on Peer-to-Peer Systems*, Berkley, CA, Feb 2003, pp. 207 -215.

[94] (2003) xCBL (XML Common Business Library). [Online].
Available: http://www.xcbl.org/ Accessed: October 2007

[95] (2001) US Secure Hash Algorithm 1 (SHA-1). [Online].
Available: http://tools.ietf.org/html/rfc3174 Accessed: October 2007

[96] (2007) Shamus Software Ltd, MIRACL Library. [Online].

Available: http://www.shamus.ie/ Accessed: October 2007

APPENDIX A: ANONYMITY GUARANTEE WITH THREE VERTICES

Following is a sketch of proof to show that the minimum number of vertices that can constitute an anonymity hazard is four, regardless the hypercube size. Let p, q_1, q_2 are spent without causing double spending offense. We now show that anonymity hazard is impossible by using these three vertices. Without loss of generality, we assume that $n > dim(q_1) \geq dim(q_2) > dim(p)$ and the trivial case of vertices at the root level (n) is not considered because spending a root vertex will cause double spending with any other spent vertices. We also do not consider $dim(q_1) = dim(p)$ or $dim(q_2) = dim(p)$, because in a top-down approach only vertices at dimension higher than $dim(p)$ are useful to compute the delegation key of p. Since $d(p, q_1) > 0$ and $d(p, q_2) > 0$, there is at least one 0/1 bit difference between the (p, q_1) pair and between the (p, q_2) pair. We assume that the bit differences occur at the i^{th} bit of the (p, q_1) pair and the j^{th} bit of the (p, q_2) pair.

Fig. 30 depicts the case when $dim(q_2) < (n\text{-}1)$, so that every vertex at $dim(q_1)$ contains at least two bits which are 0 or 1. Let p_2 be an ancestor of p at $dim(q_2)$, whose i^{th} and j^{th} bits are identical to those of p. Consider the path from p_2 to p. The j^{th} bit assures that all vertices on this path are non-susceptible to q_2, implying that none of them are in $u_{KDM}(\{q_2\})$ which contains vertices susceptible to q_2. We extend this path to p_1, which is an ancestor of p at $dim(q_1)$ with its i^{th} bit identical to that in p. Similarly, the i^{th} bit ensures that all vertices on the new path (from p_1 to p) are not susceptible to q_1, implying that none of them are in $u_{KDM}(\{q_1\})$. Each vertex on this path has at least one parent who

is in neither $u_{KDM}(\{q_1\})$ nor $u_{KDM}(\{q_2\})$. Thus, the delegation key of p cannot be computed and anonymity hazard can never occur in this case.

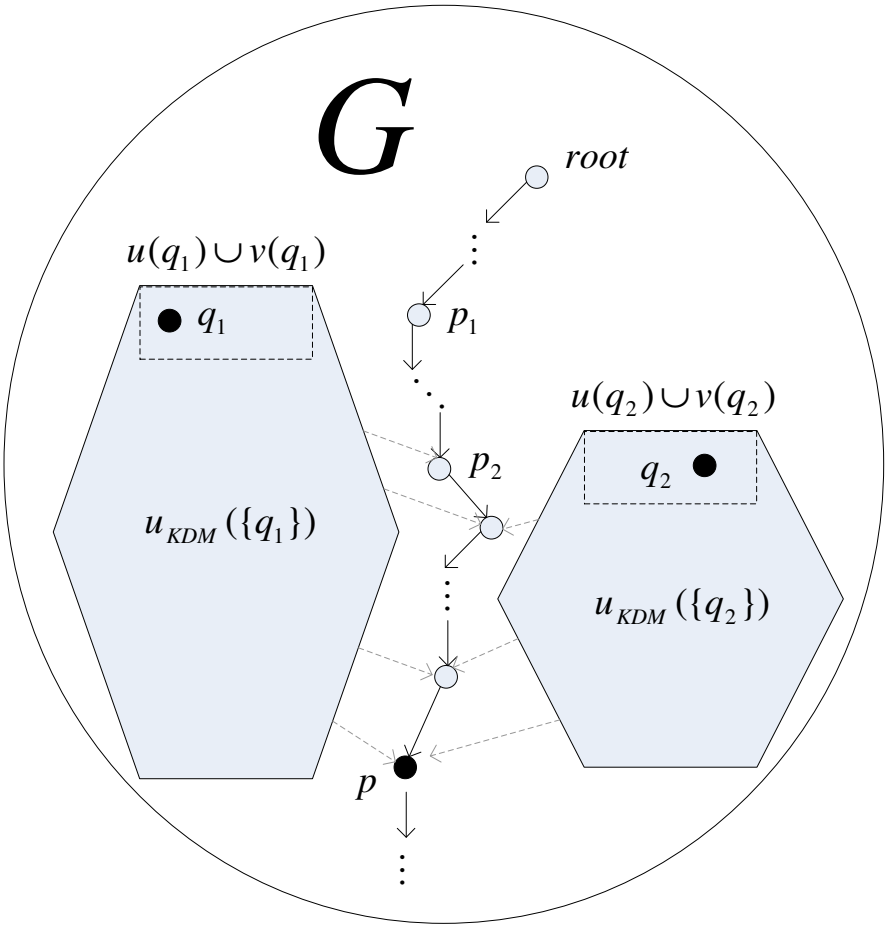


Fig. 30: Anonymity hazard impossible with 3 vertices.

Next, we consider the case when $dim(q_2) = (n\text{-}1)$. In this case, it also implies that $dim(q_1) = (n\text{-}1)$. The only possible combinations of bit pair at the i^{th} and the j^{th} positions of q_1 and q_2 are (0,×), (1,×), (×,0), and (×,1), because at dimension (n-1) every vertex

contains only one bit which is not ×. Furthermore, (×,×) is not allowed, otherwise they will cause a double spending offense with p. The only possible combinations of bit pair at the i^{th} and the j^{th} positions of p are (0,0), (0,1), (1,0), and (1,1). Since $d(q_1, q_2) > 0$, the combinations for q_1 and q_2 to co-exist could be {(0,×), (1,×)} or {(×,0), (×,1)}. In either case, we cannot find any bit pair from (0,0), (0,1), (1,0), and (1,1), such that both $d(p, q_1) > 0$ and $d(p, q_2) > 0$. Since this is impossible to construct a case for $dim(q_1) = dim(q_2) = (n\text{-}1)$, without causing double spending offense, this case is invalid.

APPENDIX B: SECURITY ANALYSIS FOR SECRET HANDSHAKE

In this section, we will show that our SH protocol satisfies the following security properties: *group member impersonation resistant*, *group member detection resistant*, and *unlinkability*. To facilitate the proof, we first define the negligible function as below:

Definition B.1: A function $\varepsilon(k)$ is negligible if for every positive polynomial $p(\cdot)$ and all sufficiently large k, $\varepsilon(k) < p(k)^{-1}$.

Group member impersonation happens when an adversary attempts to convince a valid group member that it is also a legal group member. Based on the hardness of BDH assumption, an adversary is unable to execute a successful impersonation in our protocol without compromising any valid group member or obtaining knowledge of group secret g. In other words, our protocol provides impersonation resistance that any polynomial-time adversary only has negligible probability of cheating as a group member without corrupting a member or knowing the group secret in the target group.

Group detection happens when an adversary attempts to learn whether a user is a valid member of a target group by interacting with this user. Based on the hardness of BDH assumption, an adversary cannot recognize the membership of a valid user in our protocol without compromising other group members or knowing the group secret g. In other words, our protocol provides group detection resistance that an adversary only has probability p to recognize a target user's group membership without corrupting any other member or knowing the group secret g, where p is at most negligibly larger than 1/2.

Unlinkable refer to the case when an eavesdropper cannot recognize whether or not two secret handshake instances are performed by the same user. An adversary only has probability p to decide whether or not two secret handshake instances are performed by the same user in our protocol, where p is at most negligibly larger than 1/2.

1. Group member impersonation resistance

Suppose there is an adversary B who aims at impersonating members of a certain group G^T. B may communicate with legitimate users in G^T, corrupt some valid users and obtain their secrets. B picks a target user u^T and wants to convince u^T that B is a member in G^T. *Group Member Impersonation Game* (*GMIG*) for a randomized polynomial-time adversary B is defined as follows:

(i) B communicates with users in G^T on its own choice. B may compromise certain user $U^C \subseteq U$ and obtain their secrets.

(ii) B selects a target user $u^T \not\subset U^C$, where $u^T \in G^T$.

(iii) B wants to convince u^T that $B \in G^T$.

B wins GMIG if B convinces u^T that B is a valid member in G^T, *i.e.*, B responds correctly to u^T in the SH protocol. To prove our scheme is group member impersonation resistant, we define the following probability:

$$GMIG_B = Pr[B \text{ wins GMIG}] \tag{B.1}$$

When B does not compromise any valid user $U^C \cap U$, the above probability becomes:

$$GMIG_{B\backslash(U^C \cap U)=\phi} = Pr[B \text{ wins GMIG} \mid (U^C \cap U) = \varnothing] \tag{B.2}$$

The proof needs to show that $GMIG_{B\setminus(U^C \cap U)=\phi}$ is negligible for any B in our scheme. The proof is based on the group member impersonation resistance property in [6]:

Theorem B.1 [6]**:** If BDH problem is hard to probabilistic polynomial time adversary B, then $AdvMIG_B^{U' \cap G^* = \phi}$ is negligible.

$AdvMIG_B^{U' \cap G^* = \phi}$ defined in [6] represents $GMIG_{B\setminus(U^C \cap U)=\phi}$ defined in our scheme.

Corollary B.1: If the SH protocol in [6] is group member impersonation resistant, then our SH protocol also holds the property.

Proof B.1: Suppose B is the adversary. B needs to produce W_{B1} and $V'_{B,A}$ such that it can convince A that $V_{B,A} = V'_{B,A}$. Since H_2 is collision resistant, it requires B to produce W_{B1} and $U_{B,A}$ such that $U_{B,A} = U'_{B,A}$. B does not know $s_{A1} \cdot K_A$ because K_A is kept secret by A, and s_{A1} cannot be computed from W_{A1} based on the BDH assumption. Suppose B is able to find $U_{B,A} = U'_{B,A} = e(W_{B1}, s_{A1} \cdot K_A)$ with the knowledge of W_{B1} only. It implies that B can also find $e(H(ID_B), g_A \cdot H(ID_A))$ with the knowledge of $H(ID_B)$ only. If B can do so, then B can win GMIG in [6], which contradicts to Theorem B.1. Therefore, our scheme is group member impersonation resistant:

Theorem B.2: If the BDH problem is hard to probabilistic polynomial time adversary B, then $GMIG_{B\setminus(U^C \cap U)=\phi}$ is negligible.

2. Group member detection resistance

Suppose there is an adversary B who aims at identifying members of a certain group G^T. B may communicate with legitimate users in G^T, compromise some valid users, and obtain their secrets. B picks a target user u^T and wants to decide whether $u^T \in G^T$. Suppose there is another random simulator r. If B aims at identifying members of G^T, it should distinguish between u^T and r such that B can determine the identity of u^T. *Group Member Detection Game* (*GMDG*) for a randomized, polynomial-time adversary B is defined as follows:

(i) B communicates with users of target group G^T based on its own choice. B may compromise certain user $U^C \subseteq U$ and obtain their secrets.

(ii) B selects a target user $u^T \not\subset U^C$, where $u^T \in G^T$.

(iii) A random bit $b \leftarrow \{0, 1\}$ is flipped.

(iv) There is another random simulator r.

(v) If $b = 0$, B interacts with u^T. If $b = 1$, B interacts with a random simulator r.

(vi) B outputs a guess b' for b.

B wins GMDG when $b' = b$. To prove our scheme is group member detection resistant, we define the following probability:

$$GMDG_B = Pr[B \text{ wins GMDG}] - 1/2 \qquad \text{(B.3)}$$

When B does not compromise any valid user $U^C \cap U$, the above probability becomes:

$$GMDG_{B|(U^C \cap U)=\phi} = Pr[B \text{ wins GMDG} \mid (U^C \cap U)=\varnothing] - 1/2 \qquad \text{(B.4)}$$

The proof needs to show that $GMDG_{B|(U^C \cap U)=\phi}$ is negligible for any B in our scheme. The proof is based on the group member detection resistance property in [6]:

Theorem B.3 [6]: If BDH problem is hard to probabilistic polynomial time adversary B, then $AdvMDG_B^{U' \cap G^* = \phi}$ is negligible.

$AdvMDG_B^{U' \cap G^* = \phi}$ defined in [6] represents $GMDG_{B|(U^C \cap U)=\phi}$ defined in our scheme.

Corollary B.3: If the SH proposed in [6] is group member detection resistance, then our SH protocol also holds the property.

Proof B.3: Suppose B is an adversary. Suppose B can win GMDG in our scheme without knowing s_A. Then, B should be able to win GMDG in our scheme when s_A is known also. When s_A is known, our scheme is identical to the scheme in [6], which implies that B can win GMDG in [6]. Nevertheless, it contradicts to Theorem B.3. Therefore, our scheme is group member detection resistant:

Theorem B.4: If BDH problem is hard to probabilistic polynomial time adversary B, then $GMDG_{B|(U^C \cap U)=\phi}$ is negligible.

3. Unlinkability

Suppose there is an adversary B who aims at telling whether two executions of secret handshake protocol correspond to a same user or not of a target group G^T. B may communicate with legitimate users of G^T, compromise some valid users and obtain their secrets. B picks a target user u^T. Suppose there are two different executions of secret

handshake. B attempts to tell whether the two executions correspond to the same target user u^T. *Identity Linking Game* (*ILG*) for a randomized, polynomial-time adversary B is defined as follows:

(i) B communicates with users of target group G^T based on its own choice. B may compromise certain user $U^C \subseteq U$ and obtain their secrets.

(ii) B selects a target user $u^T \not\subset U^C$, where $u^T \in G^T$.

(iii) A random bit $b \leftarrow \{0, 1\}$ is flipped.

(iv) There are two executions of secret handshake protocol.

(v) If $b = 0$, the two executions are not both performed by u^T. If $b = 1$, the two executions are both performed by u^T

(vi) B outputs a guess b' for b.

B wins ILG when $b' = b$. To prove that our scheme is unlinkable, we define the following probability:

$$ILG_B = Pr[B \text{ wins ILG}] - 1/2 \tag{B.5}$$

When B does not compromise any valid user $U^C \cap U$, the above property becomes:

$$ILG_{B|(U^C \cap U)=\phi} = Pr[B \text{ wins ILG} \mid (U^C \cap U) = \varnothing] - 1/2 \tag{B.6}$$

The proof needs to show that $ILG_{B|(U^C \cap U)=\phi}$ is negligible for any B in our scheme. The proof is based on the unlinkability property in [6]:

Theorem B.5 [6]: If BDH problem is hard to probabilistic polynomial time adversary B, then the SH protocol in [6] is unlinkable.

Corollary B.5: If SH protocol in [6] is unlinkable, then our SH protocol also holds the property.

Proof B.5: Our protocol generates elliptic curve point $s \cdot H_1(ID)$ as the pseudonym instead of assigned pseudonym "id" as in [6]. Based on the hardness of Elliptic Curve Discrete Logarithm Problem (ECDLP) [8], probabilistic polynomial time adversary has negligible probability to compute $H_1(ID)$ from $s \cdot H_1(ID)$ without knowing s. Therefore, the user can utilize one assigned pseudonym, ID, to generate a set of new pseudonyms as $s_i \cdot H_1(ID)$, where s_i are different random integers. The manipulated pseudonyms cannot be used to link to the identity of the user. Therefore, our protocol is unlinkable.

Theorem B.6: If BDH problem is hard to probabilistic polynomial time adversary B, then $ILG_{A|(U^C \cap U)=\varnothing}$ is negligible.

APPENDIX C: XML AND XPATH DATA SET

We derived 77 XPath expressions from 45 XML documents of xCBL to simulate the average hop count needed in a 15-bit CHORD ring. The 77 XPaths are shown below:

/AccountCheckRequest/AccountCheckRequestHeader/AccountCheckRequestIssueDate

/AccountCheckRequest/ListOfAccountCheckRequestDetail/AccountCheckRequestDetail/AccountCheckRequestBaseItemDetail/LineItemNum

/AdvanceShipmentNotice/ASNHeader/ASNOrderNumber/core:BuyerOrderNumber

/AdvanceShipmentNotice/ASNHeader/ASNParty/BuyerParty/core:PartyID/core:Ident

/ApplicationResponse/ApplicationResponseHeader/ApplicationResponseSender/core:PartyID/core:Ident

/ApplicationResponse/ApplicationResponseHeader/BusinessDocumentTypeCoded

/AvailabilityCheckRequest/AvailabilityCheckRequestHeader/SellerParty/core:PartyID/core:Ident

/AvailabilityCheckRequest/ListOfAvailabilityCheckRequestItemDetail/AvailabilityCheckRequestItemDetail/LineItemNum/core:BuyerLineItemNum

/AvailabilityCheckResult/AvailabilityCheckResultHeader/AvailabilityCheckResultID

/AvailabilityCheckResult/AvailabilityCheckResultHeader/BuyerParty/core:PartyID/core:Ident

/AvailabilityToPromise/AvailabilityToPromiseHeader/AvailabilityToPromisePurpose/AvailabilityToPromisePurposeCoded

/AvailabilityToPromise/AvailabilityToPromiseHeader/AvailabilityDeliveryOption/AvailabilityDeliveryOptionCoded

/AvailabilityToPromiseResponse/AvailabilityToPromiseResponseHeader/AvailabilityToPromiseRefernece/core:RefNum

/AvailabilityToPromiseResponse/AvailabilityToPromiseResponseHeader/InitiatingParty/core:PartyID/core:Ident

/ChangeOrder/ChangeOrderHeader/ChangeOrderNumber/BuyerChangeOrderNumber

/ChangeOrder/ChangeOrderHeader/SellerParty/core:PartyID/core:Ident

/ErrorResponse/CategoryCoded

/FXRateRequest/FXRateRequestHeader/Language/core:LanguageCoded

/FXRateResponse/FXRateResponseHeader/FXRateRequestID/core:RefNum

/FXRateResponse/ListOfFXRateResponseDetail/FXRateResponseDetail/ReferenceCurrency/core:Currenc
yCoded

/GetERPData/GetERPDataIssueDate

/GetERPData/ListOfKeyField/KeyField/KeyFieldName

/GetERPDataResponse/ReceiverParty/core:PartyID/core:Ident

/GetERPDataResponse/ErrorInfo/core:CompletionMsg/core:Language/core:LanguageCoded

/GetOrder/ListOfPOReferences/POReferences

/GoodsReceipt/GoodsReceiptHeader/GoodsReceiptParty/ShipFromParty/core:PartyID/core:Ident

/InventoryReport/ListOfInventoryReportDetail/InventoryReportDetail/TotalInventoryQuantity/core:UnitOfMea
surement/core:UOMCoded

/Invoice/InvoiceHeader/InvoiceLanguage/core:LanguageCoded/

/Invoice/InvoiceDetail/ListOfInvoiceItemDetail/InvoiceItemDetail/InvoiceBaseItemDetail/LineItemNum/core:B
uyerLineItemNum

/Invoice/InvoiceDetail/ListOfInvoiceItemDetail/InvoiceItemDetail/InvoiceBaseItemDetail/InvoicedQuantity/cor
e:UnitOfMeasurement/core:UOMCoded

/InvoiceResponse/InvoiceResponseHeader/InvoiceReference/core:RefNum

/InvoiceResponse/InvoiceResponseHeader/InvoiceParty/BuyerParty/core:PartyID/core:Ident

/Order/OrderHeader/OrderNumber/BuyerOrderNumber

/Order/OrderHeader/OrderParty/BuyerParty/core:PartyID/core:Ident

/OrderConfirmation/OrderConfirmationDetail/ListOfOrderConfirmationItemDetail/OrderConfirmationItemDet
ail/OrderConfirmationDetailReferences/PurchaseOrderReference/core:BuyerOrderNumber

/OrderConfirmation/OrderConfirmationDetail/ListOfOrderConfirmationItemDetail/ItemDetail/BaseItemDetail/
TotalQuantity/core:UnitOfMeasurement/core:UOMCoded

/OrderConfirmationResponse/OrderConfirmationResponseHeader/SellerOrderConfirmationReference/core:
RefNum

/OrderConfirmationResponse/OrderConfirmationResponseHeader/OrderConfirmationResponseParty/Buyer
Party/core:PartyID/core:Ident

/OrderRequest/OrderRequestHeader/OrderRequestCurrency/core:CurrencyCoded

/OrderResponse/OrderResponseHeader/OrderResponseIssueDate

/OrderResponse/OrderResponseHeader/OrderResponseNumber/BuyerOrderResponseNumber

/OrderStatusRequest/OrderStatusRequestHeader/BuyerParty/core:PartyID/core:Ident

/OrderStatusRequest/ListOfOrderStatusRequestDetail/OrderStatusRequestDetail/OrderStatusReference/BuyerReferenceNumber

/OrderStatusResult/OrderStatusResultHeader/BuyerParty/core:PartyID/core:Ident

/OrderStatusResult/ListOfOrderStatusResultDetail/OrderStatusResultDetail/OrderStatusResultReference/OrderStatus/core:StatusEvent/core:StatusEventCoded

/PaymentRequest/PaymentRequestHeader/PayerParty/core:PartyID/core:Ident/

/PaymentRequest/PaymentRequestHeader/FinancialServicesParty/core:PartyID/core:Ident

/PaymentRequest/ListOfPaymentRequestDetail/PaymentRequestDetail/FinancialInstitutionDetail/core:ReceivingFinancialInstitution/core:AccountDetail/core:AccountName1

/PaymentRequest/ListOfPaymentRequestDetail/PaymentRequestDetail/PaymentRequestParty/PayeeParty/core:PartyID/core:Ident

/PaymentRequestAcknowledgment/PaymentRequestAcknHeader/FinancialServicesParty/core:PartyID/core:Ident

/PaymentRequestAcknowledgment/ListOfPaymentRequestAcknDetail/PaymentRequestAcknDetail/PaymentDocumentID/core:RefNum

/PaymentStatusRequest/PaymentStatusRequestHeader/FinancialServicesParty/core:PartyID/core:Ident

/PaymentStatusRequest/ListOfPaymentStatusRequestDetail/PaymentStatusRequestDetail/PaymentRequestID/core:RefNum

/PaymentStatusResponse/PaymentStatusResponseHeader/PaymentStatusRequestID/core:RefNum

/PaymentStatusResponse/ListOfPaymentStatusResponseDetail/PaymentStatusResponseDetail/ListOfPaymentException/PaymentException/PaymentExceptionCoded

/PlanningSchedule/PlanningScheduleHeader/ScheduleParty/SellerParty/core:PartyID/core:Ident

/PlanningSchedule/ListOfLocationGroupedPlanningDetail/LocationGroupedPlanningDetail/ListOfLocationPlanningItemDetail/LocationPlanningItemDetail/BasePlanningDetail/LineItemNum/core:BuyerLineItemNum

/PlanningScheduleResponse/PlanningScheduleResponseHeader/BuyerParty/core:PartyID/core:Ident

/PlanningScheduleResponse/ListOfLocationGroupedPlanningResponse/LocationGroupedPlanningRespons
e/LocationGroupedPlanningDetail/ListOfLocationPlanningItemDetail/LocationPlanningItemDetail/BasePlann
ingDetail/LineItemNum/core:BuyerLineItemNum
/PlanningScheduleResponse/ListOfLocationGroupedPlanningResponse/LocationGroupedPlanningRespons
e/LocationGroupedPlanningDetail/ListOfLocationPlanningItemDetail/ListOfScheduleDetail/ScheduleDetail/S
cheduleQuantities/core:QuantityCoded/core:UnitOfMeasurement/core:UOMCoded
/PriceCheckRequest/PriceCheckRequestHeader/BuyerParty/core:PartyID/core:Ident
/PriceCheckRequest/ListOfPriceCheckRequestItemDetail/PriceCheckRequestItemDetail/LineItemNum/core
:BuyerLineItemNum
/PriceCheckResult/PriceCheckResultHeader/ShipToParty/core:PartyID/core:Ident
/PriceCheckResult/ListOfPriceCheckResultItemDetail/PriceCheckResultItemDetail/ResultPrice/core:UnitPri
ce/core:UnitPriceValue
/Quote/QuoteHeader/QuoteParty/BuyerParty/core:PartyID/core:Ident
/RemittanceAdvice/RemittanceAdviceHeader/PaymentCurrency/core:CurrencyCoded
/RemittanceAdvice/RemittanceAdviceDetail/ListOfSubsidiary/Subsidiary/ListOfInvoicingDetail/InvoicingDeta
il/InvoicingDetailReference/core:PrimaryReference/core:RefNum
/RequestForQuotation/RequestQuoteHeader/QuoteParty/BuyerParty/core:PartyID/core:Ident
/Requisition/RequisitionHeader/RequisitionParty/RequisitionerParty/core:PartyID/core:Ident
/ShippingSchedule/ShippingScheduleHeader/ScheduleParty/ShipToParty/core:PartyID/core:Ident
/ShippingSchedule/ListOfLocationGroupedShippingDetail/LocationGroupedShippingDetail/ListOfLocationS
hippingItemDetail/LocationShippingItemDetailLocationShippingItemDetail/BaseShippingDetail/TotalQuantity
/core:UnitOfMeasurement/core:UOMCoded
/ShippingSchedule/ListOfLocationGroupedShippingDetail/LocationGroupedShippingDetail/ListOfLocationS
hippingItemDetail/ListOfShipScheduleDetail/ScheduleQuantities/core:QuantityCoded/core:UnitOfMeasurem
ent/core:UOMCoded
/ShippingScheduleResponse/ShippingScheduleResponseHeader/ResponseType/core:ResponseTypeCode
d
/ShippingScheduleResponse/ListOfLocationGroupedShippingResponse/LocationGroupedShippingRespons
e/LocationGroupedShippingDetail/ListOfLocationShippingItemDetail/LocationShippingItemDetail/ListOfShip

ScheduleDetail/ShipScheduleDetail/ScheduleQuantities/core:QuantityCoded/core:UnitOfMeasurement/core

:UOMCoded

/TimeSeries/TimeSeriesHeader/Language/core:LanguageCoded

/TimeSeriesRequest/TimeSeriesRequestHeader/TimeSeriesParty

/TimeSeriesResponse/TimeSeriesResponseHeader/Language/core:LanguageCoded

VITA

Name:	Tak Cheung Lam
Address:	440 Dempsey Road, Unit 242, Milpitas, CA 95035
Email Address:	brianlam@tamu.edu
Education:	B.S., Information Engineering, The Chinese University of Hong Kong, 2000
	M.S., Information Engineering, The Chinese University of Hong Kong, 2002
	Ph.D., Computer Science, Texas A&M University, 2008