

# **A Hypercube-based Divisibility Management Framework for e-cash Algorithms**

T. C. Lam and Jyh-Charn Liu  
Department of Computer Science  
Texas A & M University  
College Station, TX 77843-3112  
brianlam@tamu.edu, liu@cs.tamu.edu

**Technical Report 2004-10-4**

**Oct 12, 2004**

## **Abstract**

In this paper, we propose a divisibility management framework for e-cash algorithms. The proposed framework is based on the dependency graph of a hypercube, which is constructed from all subcubes of a hypercube, to keep track of partitioned spending of the resource pool represented by the hypercube. On the basis of Eng and Okamoto's recursive key generation equation, we develop an analysis framework to determine the necessary and sufficient conditions to meet the security requirements of authentication, privacy and double spending identification (DSI). We analyze and solve the shared-node DSI problem, in addition to the traditional same-node and route-node DSI problems. Shared-node double spending can occur when a subcube can be divided in different combinations for spending. Such freedom is fundamentally important to distributed resource management, but it is not allowed in the traditional tree-based divisible e-cash schemes. Another critical issue associated with the DSI guarantee is protection of privacy. We show that if the resource pool is allowed to be spent in arbitrary partitions, the user privacy can be violated when shared-node DSI is supported. This problem is solved by using our analysis technique to detect and prevent privacy violation before a particular resource partition can be spent. And any spent token is guaranteed to include a valid secret share of DSI for deciphering of the user identity when any type of double spending occurs.

## 1. Introduction

*Electronic resource trading* [1] is transfer of the asset ownership using electronic information as proof of the transaction, where the proof needs to be unforgeable and undeniable. In many applications, the payer (*user*) also wants to keep its identity private from the *merchant* and the *bank*. When the user uses a *token* to make a trade with the merchant, the merchant wants to make sure that the token has not been spent somewhere else, *i.e.*, *double spending*, and it is authentic, *i.e.*, issued by the bank. If the user does commit a double spending violation, the bank needs to be able to determine the true identity  $U$  of the user, based on the *spent tokens* generated from earlier transactions.

An “ideal” e-cash system [2] is distinguished from other electronic payment systems [3]-[7] by six properties: *off-line payment*, *security* (*anti-forgery* and *double spending identification*), *privacy* (*anonymity*, *untraceability* and *unlinkability*), *independence*, *transferability* and *divisibility*. Knowing that it is impractical to satisfy all these properties unconditionally, the first three e-cash properties have been the primary focus of extensive work [9]-[17], since the first off-line untraceable e-cash was introduced in [8]. Numerous e-cash algorithms have been developed for internet shopping and similar applications. For example, [11]-[13] rely on tamper resistant devices to ensure security. Traceable schemes [14]-[15] allow revokable anonymity to recover accidental loss of money or to track offenders of blackmailing or similar misbehaviors. On-line [18][19], non-transferable [20][21] or non-divisible [21][22] systems have also been investigated. E-cash algorithms have also been applied to electronic voting [23][24], mobile IP anonymity [25], mobile agent security [26]-[28] and confidential auditing [29]. The notion of *resource binding* in [30] allows application specific attributes [31] to be cryptographically associated with the token and the spent token, leaving rooms to extend e-cash algorithms to secure resource management policy [1].

Transferability is one of the key issues in e-cash design [2][9]-[11][22], where a spent token can be spent by the new owner again without causing problems. An e-coin chaining technique is proposed in [9] to incorporate transferability into off-line and non-transferable e-cash systems. It was further refined in [27] for *single-term* e-cash systems [16][21][22][32][33]. The single-term e-cash system introduced in [21] mitigates the communication overhead of *cut-and-choose* protocols [2][4]. Implementation of this transferable technique based on [21] was presented in [22][26].

In divisible e-cash [2][16][17][32]-[34], a token can be spent in multiple transactions provided that the total amount of spending is within its face-value. An  $n$ -spendable e-cash scheme [32] allows a token to be spent  $n$  times. Tree-based divisible e-cash [2][16][17][33][34] is widely adopted for divisibility management because it allows a face-value to be

split into different sizes for spending. Each tree node can be split into or merge from its child nodes for spending at different sub-values of the total face-value. A generalized tree-based algorithm is proposed in [16] to incorporate divisibility into systems that are off-line, non-divisible and single-term. Implementation of this technique based on [21] was briefly presented in [16] and further elaborated in [30]. [17][33][34] were proposed to improve the computational complexity in withdrawal and payment protocols, and to reduce the size of the spent token. [34] eliminated the linkability among different divisions of the same token.

In this paper, we propose a hypercube-based divisible e-cash management framework. Hypercube is a superset of many broadly used topologies, and is well suited for fault-tolerant resource routing [35]-[37], broadcasting [37]-[39] and processor allocation [40][41]. To keep track of spending of a set of resource represented by a hypercube  $Q_n$ , the hypercube is expanding into its *dependency graph*  $G_n$ , which represents the hierarchical relationship between subcubes. Using the recursive key generation formula in Eng and Okamoto [15] in a top-down fashion, we analyze the security tradeoff for authentication, privacy and *double spending identification* (DSI). We show that one cannot unconditionally protect privacy when the user is allowed to spend subcubes in an arbitrary fashion. Using our analysis technique, one can perform a *privacy leak test* before spending of subcube, so that the subcube can be spent only if privacy leak will not occur.

The rest of this paper is organized as follows. Section 2 introduces the baseline architecture for the e-cash algorithm design. Section 3 explains the key generation on withdrawal of a token. Section 4 defines the key and secret share selection requirements for key verification and identity validation in the payment phase. Section 5 further extends these requirements for DSI and privacy after deposit of spent tokens to the bank. Section 6 discusses the tradeoff issues between DSI and privacy, and proposes a privacy leak test for the user to determine whether or not to spend a subcube. Section 7 concludes the paper.

## 2. System Architecture

An  $n$ -dimensional hypercube  $Q_n$  and each of its subcubes can be uniquely represented by an  $n$ -bit *index*

$$P = P_{(1)}P_{(2)} \cdots P_{(n)}, \quad (1)$$

where  $p_{(j)} \in \{0, 1, \times\}$  and the symbol “ $\times$ ” represents a “don’t care” bit. An  $i$ -dimensional subcube  $iQ_n$  has  $i$  of its  $n$  bits marked as “ $\times$ ”, and the remaining bits either 0 or 1. The *dimension* of  $p$ , denoted by  $\text{dim}(p)$ , is the number of “ $\times$ ” in  $p$ . Any two hypercube nodes of the same dimension are connected when their Hamming distance [42] is one. The *hypercube distance* is the shortest distance between subcubes  $p$  and  $q$  in  $Q_n$ , denoted by

$$D(p, q) = \sum_{j=1}^n s_j, \begin{cases} s_j = 1, \text{ if } p_{(j)} \neq q_{(j)} \text{ and } p_{(j)}, q_{(j)} \neq \times \\ s_j = 0, \text{ otherwise} \end{cases} \quad (2)$$

$D(p, q)=0$  implies that subcubes  $p$  and  $q$  overlap at certain hypercube node and thus double spending occurs if both of them are spent. Each subcube represents a subset of resource that can be spent/purchased. To keep track of the spending patterns, we need to use the dependency graph of  $Q_n$ ,  $G_n$ , to represent the hierarchical relationship between a hypercube and its subcubes. Every subcube in  $Q_n$  is represented by a node in  $G_n$ .

**Definition 1:** A dependency graph  $G = (Y, E)$  is a directed graph, on which an edge  $(p, q) \in E$  iff

$$\dim(p) = \dim(q) + 1 \text{ and } d(p, q) = 1, \quad (3)$$

for  $p, q \in Y = \{0, 1, \times\}^n$  and  $d(p, q)$  is the Hamming distance between  $p$  and  $q$  over the alphabets  $\{0, 1, \times\}$ . The root of  $G_n$  represents  $Q_n$ , and each leaf node in  $G_n$  is a node in  $Q_n$ . In the rest of discussion, the subscript  $n$  will be omitted from  $Q_n$  and  $G_n$ , unless it is explicitly defined. Let  $p$  and  $q$  represent two nodes on  $G$  that have been spent. If both  $p$  and  $q$  can reach a same leaf node, double spending occurs to the leaf node. The dependency graph  $G_2$  of  $Q_2$  is depicted in Figure 1.

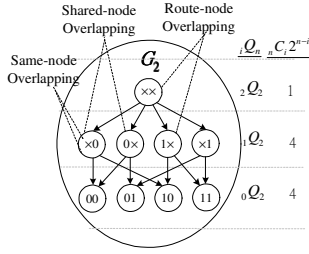


Figure 1. Dependency graph  $G_2$ .

We adopt a typical off-line, non-transferable e-cash environment whose stakeholders are the bank, the user and the merchant. Figure 2 (a) and (b) depict the interactions among the three parties in the non-divisible and the divisible settings. Three major protocols are involved between the three parties: *withdrawal* of tokens from the bank by the user, *payment* made by the user to the merchant and DSI by the bank using the spent tokens *deposited* from the merchant. A non-divisible token can be used only once in a transaction, without having  $U$  deciphered by the bank from the spent tokens. In divisible e-cash systems, stakeholders use  $G$  as a *resource spending space* to keep track of the resource spending. For design of the divisibility management framework, we focus mainly on the payment and DSI related issues. The withdrawal protocol will only be highlighted as needed.

For the user to make trades, it needs to first register its plaintext *user identity*  $U$  to the bank. In withdrawal of a token, the bank cannot associate  $U$  with other token parameters to enforce untraceability of e-cash.  $U$  is verified but kept private from the merchant during normal operations.

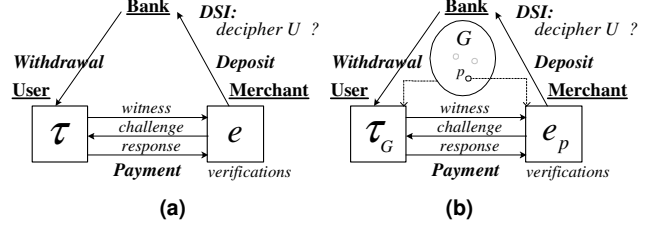


Figure 2. Off-line, non-transferable e-cash. (a) non-divisible, and (b) divisible.

Each node  $p$  in  $G$  is assigned by the user a *native key*  $k_p$  and a *verification key*  $K_p$  using a publicly known *key dependency map* (KDM), where  $\{N_G\} = \{k_p : \forall p \in G\}$  and  $\{V_G\} = \{K_p : \forall p \in G\}$ .

Through the withdrawal protocol, a 6-tuple token  $\tau_G$  is granted to the user by the bank:

$$\tau_G = (T, \{V_L\}, \sigma_{(T, V_L)}, \sigma_{(T, U)}, U, \{N_G\}), \quad (4)$$

where  $T$  is a unique *token identifier*,  $\{V_L\}$  the collection of leaf level verification keys (generated by the user), and  $\sigma_{(T, V_L)}$  and  $\sigma_{(T, U)}$  respectively are the *blind signatures* [20][21][43][44] of  $(T, \{V_L\})$  and  $(T, U)$  from the bank.

In each transaction, the user can spend a portion of resources, denoted by a subcube in  $Q$  or a node  $p$  in  $G$ , and the transaction is recorded by a 6-tuple spent token  $e_p$  jointly generated by the user and the merchant:

$$e_p = (T, \{V_L\}, \sigma_{(T, V_L)}, \{N_p\}, \{V_p\}, \{S_p\}). \quad (5)$$

By comparing (4) and (5), one can see that to spend  $p$ ,  $(T, \{V_L\}, \sigma_{(T, V_L)})$  is provided to the merchant without any change, but  $(\sigma_{(T, U)}, U, \{N_G\})$  is kept private from the merchant. The user provides a proper set of native and verification keys  $(\{N_p\}, \{V_p\})$  to the merchant, and collaboratively they produce  $\{S_p\}$ , where  $\{V_p\} \subseteq \{V_G\}$  and  $\{N_p\} \subseteq \{N_G\}$ .  $\{V_L\}$ , together with its signature  $\sigma_{(T, V_L)}$ , is used by merchant as a reference to determine if the keys of other non-leaf nodes provided by the user are valid. It takes two steps in the *zero-knowledge proof* (ZKP) algorithm,  $\mathcal{Z}(\cdot)$ , [45]-[48] for the merchant to verify that a spent token is being created from a token issued by the bank. First the merchant needs to verify that  $(\{N_p\}, \{V_p\})$  is valid, so that subsequent transactions steps can continue. Then, the merchant needs to test if  $U$  is encrypted into the spent token, bonded with its *challenge* message  $x$ . A valid spent token can be collaboratively generated by user and merchant only if  $\{S_p\}$  also includes valid *secret shares* of a *secret sharing* scheme,  $\mathcal{S}(\cdot)$ , [49][52] for the DSI system. This way, each and every spent token can guarantee that when double spending occurs, the collection of spent tokens can be used to decipher  $U$ .

$(\{N_p\}, \{V_p\})$  represents the collection of keys that are related to the current transaction for node  $p$ . It does not necessarily represent the native and verification keys of  $p$ . In

giving  $(\{N_p\}, \{V_p\})$  to the merchant, the user needs to make sure that  $U$  cannot be deciphered from the collection of spent tokens generated from completed transactions. We note that because  $p$  represents a subcube, we must consider all possible subcubes that are related to double spending and privacy violations of  $p$ . After the merchant receives  $(\{N_p\}, \{V_p\})$ , it is verified by using KDM and  $\{V_L\}$ , which was signed by the bank.

KDM uses a *one-way hash function*  $\mathcal{H}(\cdot)$  as its building block, and can be used in the top-down or bottom-up fashion. During a payment for node  $p$  (in  $G$ ), the user uses  $k_p$  to encrypt  $U$  into a secret share  $(x, R_p)$  (for DSI) as a part of the spent token. Given a KDM, the following critical design issues surrounding KDM will be addressed in the rest of the paper:

- 1) Key generation before payment.
- 2) Integrity checking of native/verification keys during payment.
- 3) DSI enforcement after payment.
- 4) Privacy protection

While our scheme is applicable to both top-down and bottom-up tracing approaches, we will only consider the top-down approach to allow focused discussions.

### 3. Key Generation before Payment

At the beginning of the withdrawal phase, the user generates  $(\{N_p\}, \{V_p\})$  using KDM, which consists one-way functions,  $\mathcal{F}(\cdot)$  and  $\mathcal{H}(\cdot)$  [43][44], where  $\mathcal{F}(\cdot)$  must meet the ZKP requirements [44][45], but  $\mathcal{H}(\cdot)$  just needs to be a regular hash function, *e.g.*, MD5, SHA-1, RC4 [43][44], *etc.* Starting as the seed of native key generation process, the root level native key,  $k_{root}$ , is a random number assigned by the user. Using  $k_{root}$  as the input, its verification key,  $K_{root}$ , is generated by  $\mathcal{F}(\cdot)$ . For any non-root node  $p$ , its native key,  $k_p$ , is generated by the verification keys of all its parent nodes using  $\mathcal{H}(\cdot)$ , and the verification key of  $p$ ,  $K_p$ , is generated by  $\mathcal{F}(\cdot)$  using  $k_p$  as the input. After the keys are generated, the user asks the bank to blindly sign  $\{V_L\}$ ,  $T$  and  $U$  with the signatures  $\sigma_{(T, V_L)} = \text{sign}(T, \{V_L\})$  and  $\sigma_{(T, U)} = \text{sign}(T, U)$ , through the withdrawal protocol, where  $T$  is a unique *token identifier* blindly assigned by the bank to a token.  $\sigma_{(T, U)}$  is not passed to the merchant, so that the merchant cannot use the parameters received from the user to create another valid spent token in future transactions, even if the merchant happens to know  $U$  by whatever reasons.

We now discuss the details on the building blocks of KDM. At a non-root node  $p$ , its native key is derived based on the following equation:

$$k_p = \mathcal{H}(K_{p_1} \parallel \dots \parallel K_{p_s} \parallel p) \quad (6)$$

$$= \mathcal{H}(\mathcal{F}(T, k_{p_1}) \parallel \dots \parallel \mathcal{F}(T, k_{p_s}) \parallel p), \quad (7)$$

where “ $\parallel$ ” denotes concatenation of binary strings and  $p_1, p_2, \dots, p_s$  are the parent nodes of  $p$ .  $\mathcal{H}(\cdot)$  is derived from the (*t-value*, *r-value*) generation function of Eng and Okamoto [16], *i.e.*,  $k_p = \mathcal{H}(K_{left-child} \parallel K_{right-child})$ , for the bottom-up approach. The node identifier  $p$  is added to (6) in order to generate  $n$  unique native keys for the  $n$  child nodes from the root node. The computational flow of KDM starting from level  $i$  of  $G$  is depicted by Figure 3, where  $A$  and  $B$  *respectively* represent a collection of native keys and verification keys in KDM. The routine recursively invokes itself until when the leaf level of  $G$  is reached, *i.e.*,  $i = 0$ , and then the processing terminates at Line 4 of the last execution instance. At the end of the execution, KDM produces another set of native keys and verification keys,  $\{A_{out}\}$  and  $\{B_{out}\}$ .

1. **KDM** ( $\{A\}, \{B\}, i$ )     {
2.  $\{A_{out}\} = \{A\}; \{B_{out}\} = \{B\};$
3. **if** ( $i = n$ ) {  $\{B_{out}\} = \{B_{out}\} \cup \{K = \mathcal{F}(T, k): \forall k \in \{A_{out}\}\}$  }
4. **if** ( $i = 0$ ) { **return**  $\{\{A_{out}\}, \{B_{out}\}\};$  }
5. **for** (every node  $q$  at dimension  $(i - 1)$ )     {
6.     **if** ( $\forall K_{q_1} \dots K_{q_s} \in \{B_{out}\}$ , where  $q_1, \dots, q_s$  are the parents of  $q$ ) {
7.          $\{A_{out}\} = \{A_{out}\} \cup \{k_q = \mathcal{H}(K_{q_1} \parallel \dots \parallel K_{q_s} \parallel q)\};$
8.          $\{B_{out}\} = \{B_{out}\} \cup \{K_q = \mathcal{F}(T, k_q)\};$      }
9. **KDM** ( $\{A_{out}\}, \{B_{out}\}, i - 1$ ); }

Figure 3. Pseudo code of KDM.

KDM is publicly known to stakeholders. By controlling what is available to  $\{A\}$  and  $\{B\}$ , the user can control what the merchant or the bank can know about the produced keys. To generate  $\{N_G\}$  and  $\{V_G\}$  in Line 4, the user invokes KDM by using inputs  $(\{A\}, \{B\}, i) = (\{k_{root}\}, \text{null}, n)$  in Line 1. A high level view of the top-down key generation process in  $G_2$  is given in Figure 4.

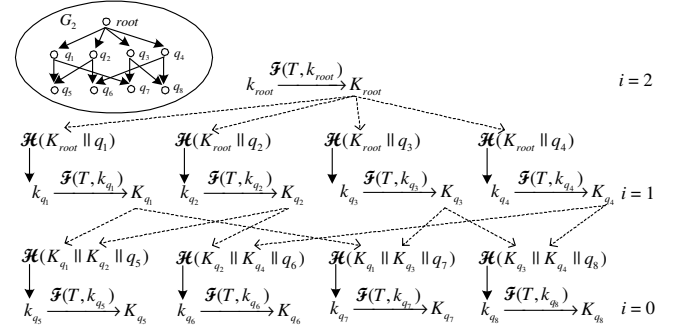


Figure 4. High level view of KDM.

For the root node at dimension  $i = 2$ ,  $k_{root}$  is randomly generated by the user to compute  $K_{root} = \mathcal{F}(T, k_{root})$ .  $K_{root}$  is used to compute keys at  $i = 1$ , *i.e.*,  $k_{q_j} = \mathcal{H}(K_{root} \parallel q_j)$  and  $K_{q_j} = \mathcal{F}(T, k_{q_j})$ , for  $j = 1, 2, 3, 4$ . The verification keys are then used to compute the keys at  $i = 0$ , *i.e.*,  $k_{q_j} = \mathcal{H}(K_{q_a} \parallel K_{q_b} \parallel q_j)$  and  $K_{q_j} = \mathcal{F}(T, k_{q_j})$ , for  $j = 5, 6, 7, 8$ , where  $q_a$  and  $q_b$  denote the parents of  $q_j$ . For example,  $k_{q_5} = \mathcal{H}(K_{q_1} \parallel K_{q_2} \parallel q_5)$ . Then both keys for all nodes have been produced, with

$\{N_G\} = \{A_{out}\} = \{k_{root}, k_{q_1}, \dots, k_{q_8}\}$  and  $\{V_G\} = \{B_{out}\} = \{K_{root}, K_{q_1}, \dots, K_{q_8}\}$ . The leaf level verification keys are extracted from  $\{V_G\}$  to form  $\{V_L\} = \{K_{q_5}, \dots, K_{q_8}\}$ , which will be signed by bank at the end of the withdrawal protocol. The KDM input/output relationships for key generation are summarized in the first column of Table 1. The same example will be used to illustrate verification of keys and DSI.

**Table 1. Input/output of KDM.**

	Key Generation	Key Verification	DSI
<b>Inputs</b> $((A), \{B\})$	$(\{k_{root}\}, null)$	$\{N_p\} = \{k_{q_1}, k_{q_4}\}, \{V_p\} = \{k_{q_2}\}$ $\{N_{p'}\} = \{k_{q_2}, k_{q_3}\}, \{V_{p'}\} = \{k_{q_1}\}$	
		$(\{N_p\}, \{V_p\})$	$(\{N_p\} \cup \{N_{p'}\}, \{V_p\} \cup \{V_{p'}\})$
<b>Outputs</b> $((A_{out}), \{B_{out}\})$	$\{N_G\} = \{k_{root}, k_{q_1}, \dots, k_{q_8}\}$ $\{V_G\} = \{K_{root}, K_{q_1}, \dots, K_{q_8}\}$	$\{A_p\} = \{k_{q_1}, k_{q_4}, k_{q_5}, k_{q_6}\}$ $\{B_p\} = \{K_{q_1}, K_{q_2}, K_{q_4}, K_{q_5}, K_{q_6}\}$	$\{A_{p,p'}\} = \{k_{q_1}, \dots, k_{q_8}\}$ $\{B_{p,p'}\} = \{K_{q_1}, \dots, K_{q_8}\}$
<b>Outputs of interest</b>	$\{N_G\}$ and $\{V_L\} = \{V_G\} @ \text{leaf level} = \{K_{q_5}, \dots, K_{q_8}\}$	$\{V_{L_p}\}$ $= \{V_L\} \cap \{B_p\}$ $= \{K_{q_5}, \dots, K_{q_8}\}$	$\{A_{p,p'}\}$

#### 4. $(\{N_p\}, \{V_p\})$ Verification and $U$ Validation in Payment

The objective of the payment protocol is to generate a spent token  $e_p = (T, \{V_L\}, \sigma_{(T, V_L)}, \{N_p\}, \{V_p\}, \{S_p\})$  by the user and the merchant based on three message exchanges, *i.e.*, witness, challenge, and response. After receiving the witness message,  $(T, \{V_L\}, \sigma_{(T, V_L)})$ , the merchant verifies whether or not the signature  $\sigma_{(T, V_L)}$  matches its plaintext  $(T, \{V_L\})$ . Knowing that  $(T, \{V_L\})$  is correct, the merchant randomly generates a challenge message  $x$  and sends it back to the user. Then the user prepares and sends back to merchant a response message,  $(\{N_p\}, \{V_p\}, \{S_p\})$ . The merchant first verifies  $(\{N_p\}, \{V_p\})$ , and then  $\{S_p\}$ .

For key verification,  $(\{N_p\}, \{V_p\})$  needs to be defined based on requirements (12) and (13) to be described in the next subsection. Given  $(\{N_p\}, \{V_p\})$ , the merchant invokes KDM, and compares the results on  $\{B_{out}\}$  of KDM with  $\{V_L\}$ . If all entries in  $\{B_{out}\}$  match  $\{V_L\}$ , then  $(\{N_p\}, \{V_p\})$  is considered valid, and the transaction can proceed. Otherwise, the transaction is aborted. The merchant always invokes KDM from level  $n$ , using  $(A, B, i) = (\{N_p\}, \{V_p\}, n)$  as inputs, because KDM can skip any unmatched parent-child relationship, and continue the matching and computation until it reaches the leaf level.

An example on the  $(\{N_p\}, \{V_p\})$  verification is given in the second column of Table 1 based on the example given in Figure 4, where  $\{N_p\} = \{k_{q_1}, k_{q_4}\}$ ,  $\{V_p\} = \{K_{q_2}\}$ . The computation of KDM directly proceeds to  $i = 1$  because the keys of root are

not available.  $k_{q_1}, k_{q_4} \in \{N_p\}$  are used to compute  $K_{q_1} = \mathcal{F}(T, k_{q_1})$  and  $K_{q_4} = \mathcal{F}(T, k_{q_4})$ . Using  $(K_{q_1}, K_{q_2}, K_{q_4})$ , the keys that can be computed at  $i = 0$  are  $k_{q_5} = \mathcal{H}(K_{q_1} \parallel K_{q_2} \parallel q_5)$ ,  $k_{q_6} = \mathcal{H}(K_{q_2} \parallel K_{q_4} \parallel q_6)$ ,  $K_{q_5} = \mathcal{F}(T, k_{q_5})$  and  $K_{q_6} = \mathcal{F}(T, k_{q_6})$ . KDM returns the outputs  $\{A_p\} = \{A_{out}\} = \{k_{q_1}, k_{q_4}, k_{q_5}, k_{q_6}\}$ ,  $\{B_p\} = \{B_{out}\} = \{K_{q_1}, K_{q_2}, K_{q_4}, K_{q_5}, K_{q_6}\}$ .  $\{V_{L_p}\} = \{K_{q_5}, K_{q_6}\}$  is selected from  $\{B_p\}$  to compare against  $\{V_L\}$  as the final step of  $(\{N_p\}, \{V_p\})$  verification.

Next, we consider how to guarantee that  $\{S_p\}$  is a collection of valid secret shares for the DSI. For each element included in  $\{S_p\}$ , its ‘‘matching’’ element to solve the secret sharing scheme of DSI will be generated when double spending occurs. For simplicity of explanation, we let each element in  $\{S_p\}$  take the form of  $(x, R_q)$ , where  $q$  is a node that satisfies the requirements (14) and (15) with respect to the node  $p$  being traded. A simplified example of secret share, which will be introduced in Table 2 of the next section, is computed from the secret share generator,

$$S(q) : R_q = U \cdot x + k_q. \quad (8)$$

This way, when any hypercube node is spent twice, the DSI can decipher  $U$  from the collection of spent tokens. We also let  $\mathcal{S}(\cdot)$  be equal to the ZKP prover function  $\mathcal{Z}(\cdot)$ , which has the general form,

$$\mathcal{Z}(q) : R_q = \mathcal{Z}(T, U, \sigma_{(T, U)}, x, k_q). \quad (9)$$

We note that (8) for  $\mathcal{Z}(\cdot)$  only satisfies secret sharing properties, but not the ZKP properties. We will show in the appendix how to add ZKP properties to specific implementations, such as the one in Ferguson’s e-coin [21] that has been commonly used to demonstrate e-cash techniques [16][22][26][27][30].

The merchant can verify each element  $(x, R_q)$  in  $\{S_p\}$ , using the ZKP verifier function  $\bar{\mathcal{Z}}(\cdot)$ , which has the general form

$$\bar{\mathcal{Z}}(q) : \bar{\mathcal{Z}}(T, x, R_q, K_q) \stackrel{?}{=} TRUE. \quad (10)$$

Example of (10) corresponding to (8) is  $W_q^v = C^{R_q} B^v K_q$ , where  $v$  is the public key of the bank, and  $B, C, W_q$  are parameters related to  $T$  and  $R_q$ , whose details will be shown in the appendix.

#### 4.1 Selection Requirements of $(\{N_p\}, \{V_p\}, \{S_p\})$

In this subsection, we consider the selection criteria of  $(\{N_p\}, \{V_p\}, \{S_p\})$  that the user passes to the merchant for key verification and identity validation. For ease of presentation, we use  $\{Y_X\}$  to denote a collection of nodes in  $G$  that defines  $\{X\}$ , where  $\{X\}$  can be  $\{N_p\}, \{V_p\}, \{S_p\}$ , *etc.* That is,

$$\{S_p\} = \{(x, R_q) : q \in \{Y_{S_p}\}\}, \quad (11)$$

$$\{V_p\} = \{K_q : q \in \{Y_{V_p}\}\}, \text{ and} \quad (12)$$

$$\{N_p\} = \{k_q : q \in \{Y_{N_p}\}\}. \quad (13)$$

To guarantee that  $U$  can be validated, it is necessary to have

$$\{Y_{S_p}\} \neq \emptyset, \quad (14)$$

so that at least one secret share contains the encrypted  $U$ , and  $U$  can be decrypted when double spending occurs. Next, we need to consider the relationship between  $(\{V_p\}, \{S_p\})$ . To ensure that every secret share  $(x, R_q)$  generated by  $k_q, q \in \{Y_{S_p}\}$ , can be verified by  $K_q, q \in \{Y_{V_p}\}$ , it is necessary to have

$$\{Y_{V_p}\} \supseteq \{Y_{S_p}\}. \quad (15)$$

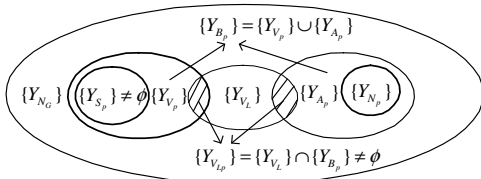
The third requirement is related to how to use  $\{V_L\}$ , which is the absolute system integrity reference, to guarantee the integrity of  $(\{N_p\}, \{V_p\})$ . This requirement can be satisfied by (16) and (17):

$$\{Y_{V_L}\} = \{Y_{V_L}\} \cap \{Y_{B_p}\} \neq \emptyset, \text{ and} \quad (16)$$

$$\{Y_{V_L}\} \text{ cannot be computed by KDM} \\ \text{if any node in } \{Y_{V_p}\} \cup \{Y_{N_p}\} \text{ is removed.} \quad (17)$$

That is, when a KDM instance is invoked by the merchant to verify  $(\{N_p\}, \{V_p\})$ , the choice of  $(\{N_p\}, \{V_p\})$  needs to guarantee that the output  $\{B_p\}$  contains some leaf level verification keys that can be compared against  $\{V_L\}$ . Furthermore,  $(\{N_p\}, \{V_p\})$  needs to guarantee that when any element in  $(\{N_p\}, \{V_p\})$  is missing, the computed verification keys of leaf nodes in  $\{B_p\}$  are different *i.e.*, no element in  $(\{N_p\}, \{V_p\})$  is redundant. Otherwise, removal of a node may still lead to a valid verification result, implying that one might fail to validate the removed node.

The relationship among  $\{Y_{N_G}\}, \{Y_{S_p}\}, \{Y_{N_p}\}, \{Y_{V_p}\}, \{Y_{A_p}\}, \{Y_{B_p}\}, \{Y_{V_L}\}, \{Y_{V_L}\}$  based on the selection criteria of  $(\{N_p\}, \{V_p\}, \{S_p\})$  discussed above is depicted in Figure 5.  $\{Y_{N_G}\}$  is the largest set which contains all nodes in  $G$ .  $\{Y_{V_p}\} \supseteq \{Y_{S_p}\} \neq \emptyset$  because of (14) and (15).  $\{Y_{N_p}\} \cap \{Y_{V_p}\} = \emptyset$  because the native keys in  $\{N_p\}$  can be used to compute the corresponding verification keys, so that no duplicated verification key in  $\{V_p\}$  is necessary.  $\{Y_{A_p}\} \supseteq \{Y_{N_p}\}$  because  $\{N_p\}$  is used to initialize  $\{A_p\}$  in Line 2 of KDM.  $\{Y_{B_p}\} = \{Y_{A_p}\} \cup \{Y_{V_p}\}$  because  $\{V_p\}$  is used to initialize  $\{B_p\}$  and the native keys in  $\{A_p\}$  are used to compute the corresponding verification keys and have them added to  $\{B_p\}$  in Line 3 and Line 8 of KDM.  $\{Y_{V_L}\} = \{Y_{V_L}\} \cap \{Y_{B_p}\} \neq \emptyset$  because of (16).



**Figure 5.**  $(\{N_p\}, \{V_p\})$  verification and  $U$  validation: relationship among  $\{Y_{N_G}\}, (\{Y_{N_p}\}, \{Y_{V_p}\}, \{Y_{S_p}\}), (\{Y_{A_p}\}, \{Y_{B_p}\})$  and  $(\{Y_{V_L}\}, \{Y_{V_L}\})$ .

## 5. DSI after Payment and Privacy Preservation

After the payment is completed, the merchant deposits the spent token  $e_p = (T, \{V_L\}, \sigma_{(T, V_L)}, \{N_p\}, \{V_p\}, \{S_p\})$  to the bank. The bank determines whether or not two spent tokens represent overlapped resources  $p$  and  $q$ , by comparing their hypercube distance. The choices of  $(\{N_p\}, \{V_p\}, \{S_p\})$  not only have to satisfy the key verification and identity validation requirements, as discussed earlier, but also needs to meet the DSI and the privacy requirements, *i.e.*, how can a  $(2, m)$  secret sharing scheme decipher  $U$  from the spent tokens, if, and only if, double spending occurs. In a  $(t, m)$  secret sharing scheme, a plaintext is encoded into  $m$  secret shares. The plaintext can be recovered when any  $t$  of the  $m$  secret shares become available.

Three double spending scenarios need to be considered. When  $p = q$ , it is referred to as *same-node overlapping*. When  $p$  is an ancestor of  $q$ , *i.e.*, the subcube represented by  $q$  is a proper subset of the subcube represented by  $p$ , it is called *route-node overlapping*. When  $p$  and  $q$  share some but not all descendants, it is called *shared-node overlapping*. Same-node and route-node overlapping are well known in tree-based divisible e-cash, but the shared node overlapping is a new double spending scenario that needs to be considered. We replace the word “overlapping” with “double spending” when double spending occurs between  $p$  and  $q$ .

Next we discuss how to use a  $(2, m)$  secret sharing scheme to decipher  $U$ , which has been encrypted into the spent tokens in the payment phase. An example of  $(2, m)$  secret sharing scheme in [49] allows  $U$  to be deciphered under the two conditions,  $DU_1$  and  $DU_2$ , is summarized in Table 2.

**Table 2.** Conditions to decipher  $U$  and polynomial based secret sharing examples.

Conditions to decipher $U$	Variable values generated from doubly spent tokens	$(2, n)$ polynomial secret sharing
$DU_1$	Two secret shares: $(x, R_p), (x', R'_p)$	Solve $U$ and $k_p$ by $(x, R_p = U \cdot x + k_p)$ and $(x', R'_p = U \cdot x' + k_p)$
$DU_2$	A secret share + its native key: $(x, R_p), k_p$	Solve $U$ by $(x, R_p = U \cdot x + k_p)$ and $k_p$

$DU_1$  allows  $U$  and  $k_p$  to be solved from a pair of secret shares using polynomial interpolation.  $DU_2$  allows  $U$  to be solved from a secret share and its native key. Based on  $DU_2$ , one can conclude that  $\mathcal{F}(\cdot)$  must be a one-way function. Otherwise, if the verification key can also be used to compute its native key, this native key together with the secret share being validated can decipher  $U$  even when no double spending occurs.

In existing e-cash systems (divisible or non-divisible),  $DU_1$  is used when the same-node double spending occurs. In

tree-based divisible e-cash,  $DU_2$  is used when route-node double spending occurs. In our hypercube-based scheme, both  $DU_1$  and  $DU_2$  are used to handle the shared-node double spending scenario, which was not considered in the tree-based scheme. Knowing that the secret share and the native key of a node affect the states of multiple nodes in the hypercube, the choices of  $(\{N_p\}, \{V_p\}, \{S_p\})$  control the availability of secret shares and native keys for nodes in  $G$  to meet the DSI and privacy requirements.

## 5.1 Selection Requirements of $(\{N_p\}, \{V_p\}, \{S_p\})$

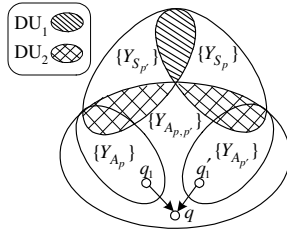
For the discussions of DSI and privacy, which concern about multiple nodes,  $p_1, \dots, p_m$ , we let  $\{A_{p_1, \dots, p_m}\}$  be the collection of native keys generated from KDM when  $(\{A\}, \{B\}, i) = (\cup_{i=1}^m \{N_{p_i}\}, \cup_{i=1}^m \{V_{p_i}\}, n)$  is used as inputs. It is important to note that, in general,  $\{A_{p_1, \dots, p_m}\} \neq \cup_{i=1}^m \{A_{p_i}\}$ .

Considering two nodes  $p$  and  $p'$  involved in a double spending incident,  $DU_1$  and  $DU_2$  dictate that one of the following two conditions must hold:

$$DU_1 : \{Y_p\} \cap \{Y_{p'}\} \neq \emptyset, \quad \text{or} \quad (18)$$

$$DU_2 : \{Y_{A_{p,p'}}\} \cap (\{Y_{S_p}\} \cup \{Y_{S_{p'}}\}) \neq \emptyset. \quad (19)$$

The decipherable regions of  $U$  by  $DU_1$  and  $DU_2$  are shown in Figure 6. (18) implies that some node in  $G$  was involved in generation of the spent tokens for  $p$  and  $p'$ , so that its secret share will be revealed twice in these two spent tokens for  $DU_1$  to decipher  $U$ . When (19) holds, it implies that there is at least one native key of  $q$ ,  $k_q \in \{A_{p,p'}\}$ , such that its secret share ( $x, R_q$ ) is available from  $\{S_p\} \cup \{S_{p'}\}$  for  $DU_2$  to decipher  $U$ .



**Figure 6. DSI and privacy: relationship among  $\{Y_{S_p}\}, \{Y_{S_{p'}}\}, \{Y_{A_p}\}, \{Y_{A_{p'}}\}$  and  $\{Y_{A_{p,p'}}\}$**

The privacy requirements, which are the complements of (18) and (19), are summarized as follows:

$$\overline{DU_1} : \cap_{i=1}^m \{Y_{S_{p_i}}\} = \emptyset, \quad \text{and} \quad (20)$$

$$\overline{DU_2} : \{Y_{A_{p_1, \dots, p_m}}\} \cap (\cup_{i=1}^m \{Y_{S_{p_i}}\}) = \emptyset, \quad (21)$$

where  $p_1 \dots p_m$  are nodes without double spending. (20) makes sure that no secret share of a node appears twice from the spent tokens so that  $DU_1$  cannot be satisfied to decipher  $U$ . (21) implies that the native key of node  $q$ ,  $k_q \in \{A_{p_1, \dots, p_m}\}$ , does not have its secret share included in  $\cup_{i=1}^m \{S_{p_i}\}$ , so that  $DU_2$

cannot be satisfied to decipher  $U$ . Figure 6 depicts the case of (21) when  $m = 1$ , *i.e.*,  $\{Y_{A_p}\} \cap \{Y_{S_p}\} = \emptyset$ . Figure 6 also depicts the  $(\{Y_{A_p}\}, \{Y_{A_{p'}}\}, \{Y_{A_{p,p'}}\})$  relationship when  $m = 2$ :

$$\{Y_{A_p}\} \cup \{Y_{A_{p'}}\} \subseteq \{Y_{A_{p,p'}}\}. \quad (22)$$

(22) implies that some native keys generated from the transactions  $p$  and  $p'$  separately by KDM are missing from those generated jointly from both transactions. Losing track of these missing native keys might lead to privacy violation based on  $DU_2$ , that we will discuss next section. Note that (22) is not a selection requirement for DSI/privacy, but the direct result from using KDM, as depicted by the following scenario (also shown in Figure 6).

Suppose node  $q$  has two parents,  $q_1 \in \{Y_{A_p}\}$ ,  $q'_1 \in \{Y_{A_{p'}}\}$ , but  $q_1 \notin \{Y_{A_{p'}}\}$ ,  $q'_1 \notin \{Y_{A_p}\}$ .  $q \notin \{Y_{A_p}\}$  because one parent  $q'_1$  is not available. Similarly,  $q \notin \{Y_{A_{p'}}\}$  because another parent  $q_1$  is not available. However,  $q \in \{Y_{A_{p,p'}}\}$  because both parents are available when both transactions of  $p$  and  $p'$  are considered.

One can extend (22) to consider  $m$  nodes,  $p_1, \dots, p_m$ , based on similar argument:

$$\cup_{i=1}^m \{Y_{A_{p_i}}\} \subseteq \{Y_{A_{p_1, \dots, p_m}}\} \quad (23)$$

Figure 4 and Table 1 further elaborate the above example, which takes the KDM inputs  $\{N_p\} = \{k_{q_1}, k_{q_4}\}$ ,  $\{N_{p'}\} = \{k_{q_2}, k_{q_3}\}$ ,  $\{V_p\} = \{K_{q_2}\}$  and  $\{V_{p'}\} = \{K_{q_1}\}$  to compute  $\{A_p\} = \{k_{q_1}, k_{q_4}, k_{q_5}, k_{q_6}\}$  and  $\{A_{p'}\} = \{k_{q_2}, k_{q_3}, k_{q_5}, k_{q_7}\}$ . Comparing with  $\{A_{p,p'}\}$ ,  $\{A_p\} \cup \{A_{p'}\}$  has one element,  $k_{q_8}$ , missing from  $\{A_{p,p'}\} = \{k_{q_1}, \dots, k_{q_8}\}$ , *i.e.*,  $q = q_8$  from the above example.

## 6. Tradeoff between DSI and Privacy: An Example

It is relatively easy to satisfy DSI, but it is much more difficult to preserve privacy because one must guarantee that the bank cannot decipher  $U$  from any collection of spent tokens when DSI is not violated. A primary concern is due to the inequality in (23). In tree-based scheme, this inequality holds, because no two nodes share a common child in tree. Any non-root tree node has one, and only one, parent to compute its native key. Therefore, the scenario depicted in Figure 6 for this inequality cannot happen, which requires  $q$  to have multiple parents. However, for the hypercube-based scheme, shared-nodes can exist between different spending instances. It is necessary for the hypercube-based scheme to track the available native keys in  $\{A_{p_1, \dots, p_m}\}$  to prevent privacy leak. One could modify KDM to guarantee this requirement unconditionally, but doing so will drastically complicate the structure of KDM.

We propose a privacy leak test (PLT) before a node can be spent. A node  $p$  in  $G$  can be spent only if the test is passed. When  $p$  cannot be spent directly, the user can divide  $p$  into its two subcubes to see if spending them would lead to

violation of privacy. A leaf node cannot be spent and become void if spending it violates privacy.

We use a *minimal secret share* (MSS) set to illustrate creation of  $(\{N_p\}, \{V_p\}, \{S_p\})$  that guarantees key verification, identity validation and DSI. Here,  $(\{N_p\}, \{V_p\}, \{S_p\})$  under the MSS setting are summarized as follows:

$$\{Y_{S_p}\} = \{p\}, \quad (24)$$

$$\{Y_{N_p}\} = \{q : D(p, q) = 0, p \neq q, \dim(p) = \dim(q)\}, \quad (25)$$

$$\{Y_{V_p}\} = \{p\}. \quad (26)$$

Among the above inter-dependent sets, we first choose (24) to be the minimal set, which is sufficient support DSI, based on  $DU_1$ , when same-node double spending occurs. Moreover, this choice is sufficient to prevent privacy leak, when only  $DU_1$  is considered. However, if  $DU_2$  is also a permissible DSI rule, then even (24) alone, which cannot be further reduced to serve  $DU_1$ , will still lead to privacy leak, as we will see the examples in Table 3 shortly. Following we illustrate a privacy leak scenario when (24) is not chosen to be minimal. Let  $p, p'$  be two spent nodes without double spending,  $q$  be an unspent node which has zero hypercube distance with  $p$  and  $p'$ , respectively. In a hypothetically defined DSI, if  $\{Y_{S_p}\}$  must include  $q$ , and so does  $\{Y_{S_{p'}}\}$ , then the bank will be able to decipher  $U$  using  $DU_1$  even when  $q$  has not been spent.

Given (24), next we discuss how (25) and (26) are chosen to support DSI for route/shared-node double spending. (25) is chosen to be all nodes (except  $p$ ) that have their resources overlapped with  $p$  at  $\dim(p)$ .  $p$  is not included in (25) to prevent privacy leak using (24) and  $DU_2$ .  $p$  is included in (26) as a supplement of (25) to give necessary KDM inputs for the key verification and DSI. As we will show shortly, (25) and (26) can be used to generate the native keys of all double spending suspicious nodes (except  $p$ ) at dimension  $\leq \dim(p)$ .

When any of the above suspicious nodes, denoted by  $p'$ , is spent, the secret shares of  $p'$  is generated by (24) in the current transaction. Together with the native key of  $p'$  generated by (25) and (26) in earlier transactions, the offender identity  $U$  can be deciphered based on  $DU_2$ . Note that the above scheme only needs to reveal the native keys of doubly spent nodes whose dimensions are less than or equal to  $\dim(p)$ . Otherwise, one can interchange the roles of  $p$  and  $p'$  from the above scenario for analysis.

To understand how (25) and (26) can be used by KDM to generate all native keys of the above suspicious nodes, we first mark all descendant nodes of  $p$ , *i.e.*, all route-nodes of  $p$  at dimension  $< \dim(p)$ , and then we mark all unmarked ancestors of the marked nodes at dimension  $\leq \dim(p)$ , which include  $p$  itself and the shared-nodes of  $p$ . Since (25) and (26) are the marked nodes at  $\dim(p)$ , so that the native keys of any marked nodes (except  $p$ ) at  $\dim(p)$  are available from (25). The native keys of other marked nodes can be computed by KDM because the ancestor searching process in the second

step ensures that any marked node at dimension  $< \dim(p)$  has the verification keys of its parents available from other marked nodes.

Figure 7 depicts an example on how to mark these double spending suspicious nodes in  $G_3$ . Suppose node  $p = "0 \times \times"$  is spent. By (24) and (26),  $\{Y_{S_{\text{doc}}}\} = \{Y_{V_{\text{doc}}}\} = \{0 \times \times\}$ . To evaluate  $\{Y_{N_{\text{doc}}}\}$ , we first mark all descendant of  $"0 \times \times"$ , *i.e.*,  $\{0 \times 0, 00 \times, 0 \times 1, 01 \times, 000, 001, 010, 011\}$ . Then we mark all unmarked ancestors of these nodes at dimensions lower or equal to  $\dim(p)$ , *i.e.*,  $\{\times 00, \times 01, \times 10, \times 11, \times \times 0, 0 \times \times, \times 0 \times, \times 1 \times, \times \times 1\}$ .  $\{Y_{N_{\text{doc}}}\}$  is all marked nodes (except  $p$ ) at  $\dim(p)$ , *i.e.*,  $\{\times \times 0, \times 0 \times, \times 1 \times, \times \times 1\}$ . We can observe from the figure that  $\{Y_{N_{\text{doc}}}\}$  and  $\{Y_{V_{\text{doc}}}\}$  have formed the dependencies that are necessary and sufficient to compute the keys of all marked nodes, *i.e.*, the double spending suspects.

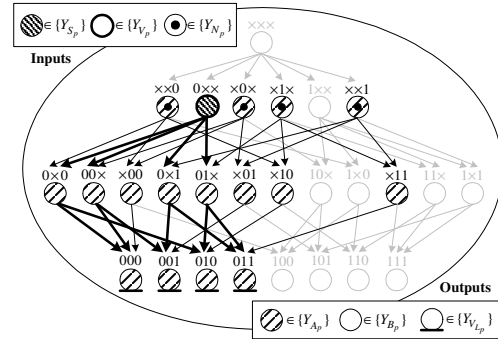


Figure 7. Example: marking double spending suspicious nodes in  $G_3$

Next, we show a privacy leak scenario. As shown in Table 3,  $p_1 = 1 \times 0$ ,  $p_2 = 01 \times$ ,  $p_3 = \times 01$  and  $p_4 = 000$  are four spent nodes whose resources are not overlapped. Using  $k_{\times 00} \in \{N_{p_1}\}$ ,  $k_{0 \times 0} \in \{N_{p_2}\}$  and  $k_{00 \times} \in \{N_{p_3}\}$ , we compute their verification keys by  $K_q = \mathcal{H}(T, k_q)$  for  $q \in \{\times 00, 0 \times 0, 00 \times\}$ . With these verification keys, the native key of  $p_4 = 000$  can be computed by  $k_{000} = \mathcal{H}(K_{\times 00} \parallel K_{0 \times 0} \parallel K_{00 \times} \parallel 000)$ . Knowing that  $(x, R_{000}) \in S_{p_4}$ ,  $U$  can be deciphered from  $(x, R_{000})$  using  $DU_2$ .

Table 3. Example: spent nodes with privacy leak.

$p_1 = 1 \times 0 : \{Y_{N_{100}}\} = \{\underline{\times 00}, 11 \times\}$	$p_3 = \times 01 : \{Y_{N_{011}}\} = \{\underline{00 \times}, 1 \times 1\}$
$p_2 = 01 \times : \{Y_{N_{010}}\} = \{0 \times 0, \times 11\}$	$p_4 = 000 : \{Y_{S_{000}}\} = \{000\}$

## 6.1 Privacy Leak Test

The user can use PLT to test and prevent privacy leak by  $DU_2$  before node  $p$  is spent, assuming that spending  $p$  will not cause double spending. In this test, each node in  $G$  is initialized to white before any transactions, and it will become black, gray, red, or remain unchanged after each

transaction. Black color represents the node that has its verification key unveiled from previous transactions. Red color represents the node which also has its secret share unveiled from previous transactions. Gray color represents the node that has its native key or verification key unveiled from the current transaction. If PLT attempts to change a red node to gray, then it returns a positive result. Otherwise, a negative result is returned.

A node becomes gray when all its parents are not white, *i.e.*, the verification keys of its parents are available from current (gray parents) or previous (black or red parents) transactions to compute the native key of this node using KDM. If the secret share of this node is also available from the current or previous (red) transactions, then  $U$  can be deciphered using  $DU_2$ , *i.e.*, privacy leak. The pseudo code of PLT is depicted in Figure 8.

```

1. PLT( $p, G$ ) {
2.   if ( all parents of  $p$  are not white) { return TRUE; }
3.   turn all white nodes in  $\{Y_{N_p}\} \cup \{Y_{V_p}\}$  to gray;
4.   for (all node  $q$  at  $dim(i), i = dim(p) - 1; i \geq 0; i --$ )
5.     if (all parents of  $q$  are not white) {
6.       if ( $q$  is white) { turn  $q$  in gray ; }
7.       if ( $q$  is red ) {
8.         turn all gray nodes to white;
9.         resume color of  $p$ ;
10.        return TRUE; }
11.  turn all gray nodes to black and then turn  $p$  to red;
12.  return FALSE; }
```

**Figure 8. Pseudo code of PLT.**

Since  $p$  has to give its secret share in the current transaction, Line 2 only needs to test if all parents of  $p$  are not white, *i.e.*, the native key of  $p$  can be computed. Line 2 terminates its execution if the result is positive. Otherwise, Lines 3-10 continue to test if the transaction of  $p$  can give additional keys to compute the native key of node  $q$ , whose secret share has been unveiled from earlier transactions.

Line 3 updates the key information available from the transaction of  $p$ , by changing all white nodes in  $\{Y_{N_p}\} \cup \{Y_{V_p}\}$  to gray. Since the keys of these nodes can only be used to compute the native keys and verification keys of nodes at lower dimensions, Line 4 considers node  $q$  at dimension lower than that of  $p$  only. If all parents of  $q$  are not white in Line 5, then the native key and verification key of  $q$  can be computed. Line 6 records this new key information by changing  $q$  from white to gray, if it is not recorded as black or red before. The gray nodes, together with the red and black nodes colored from previous transactions, will be used to update the colors of lower dimension nodes in the current transaction.

However, if  $q$  is originally red, then the secret share of  $q$  has been unveiled before. Together with the native key of  $q$  just computed, we can conclude that privacy leak exists. Since  $p$  cannot be spent, Lines 8-9 resume the gray nodes to

white,  $p$  to its original color (black or white), and then returns positive result of PLT. If no positive result is given after checking all  $q$ , then  $p$  can be spent without privacy leak. Before returning the negative result in Line 12, Line 11 updates the key and the secret share information by changing all gray nodes to black and  $p$  to red. These update colors of nodes will be used by PLT in next transaction.

It is possible that all subcubes of  $p$  and all subcubes that contain  $p$  have positive results on PLT. In this case,  $p$  becomes void and cannot be utilized. For example, in Table 3, all ancestors  $p_4 = 000$ , *i.e.*,  $\{\times\times\times, 0\times\times, \times 0\times, \times\times 0, 00\times, 0\times 0, \times 00\}$  are double spending suspects of  $p_2 = 1\times 0$ ,  $p_2 = 01\times$ , or  $p_3 = \times 01$ , while  $p_4$  itself cannot be spent due to the privacy leak as we have shown before.

## 7. Conclusion

In this paper, we presented a hypercube-based divisibility management framework for e-cash algorithms. Based on the analysis of hypercube dependency graph, we determined the necessary and sufficient conditions in choosing proper sets of keys and secret shares, so that the spent tokens generated by these sets can satisfy the inter-locked security requirements, authentication, privacy and DSI, at the same time.

We proposed the solution to solve the shared-node DSI problem, in addition to the traditional same-node and route-node DSI problems. Shared-node double spending can occur when more freedom is granted to the user to partition resources in different spendable combinations. Such freedom is important to distributed resource management, such as the allocations of address spaces, CPU cycles, processor clusters, file spaces, *etc.*, but it cannot be addressed by the tree-based models. To effectively support the DSI system, our solution ensures authenticity of the spent tokens, which guarantees that a valid secret share is included for deciphering of the user identity of offender when any type of double spending occurs.

Privacy preservation is another critical issue associated with DSI guarantee. We show that when the resource pool is allowed to be spent in arbitrary partitions, privacy leak can occur if shared-node DSI must be in place. This problem is solved by using our analysis technique to detect and prevent privacy leak before a particular resource partition can be spent. A resource becomes void if it cannot be spent by any combinations. It is easy to create a protocol for the user to turn in unused/unusable portion of the token to the bank for *refund*. Due to the space limitations, we will complete this portion of analysis in follow up work.

## References

- [1] B. F. Cooper and H. Garcia-Molina, "Peer-to-peer Resource Trading in a Reliable Distributed System." *The 1st International Workshop on Peer-to-Peer Systems (IPTPS)* (2002).
- [2] T. Okamoto and K. Ohta, "Universal Electronic Cash." In *Advances in Cryptology – CRYPTO' 91* (1991) 324 – 337.
- [3] D. Chaum, "Online Cash Checks." In *EUROCRYPT' 89* (1989) 288 – 293.
- [4] D. Chaum, B. den Boer, E. van Heyst, S. Mjolsnes and A. Steenbeek, "Efficient offline electronic checks." In *EUROCRYPT 89* (1989) 294-301.
- [5] R. Rivest and A. Shamir, "PayWord and MicroMint: Two Simple Micropayment Schemes." In *Proceedings of the International Workshop on Security Protocols* (1996) 69 – 87.
- [6] S. Glassman, M. Manasse, M. Abadi, P. Gauthier and P. Sobalvarro, *The Millicent Protocol for Inexpensive Electronic Commerce*, World Wide Web Journal **1**(1) (1995) 89.
- [7] B. Schoenmakers, "Basic Security of the ecash<sup>TH</sup> Payment System." *Computer Security and Industrial Cryptography: State of Art and Evolution* (1997).
- [8] D. Chaum, A. Fiat, and M. Naor, "Untraceable electronic cash." In *Advances in Cryptology – CRYPTO'88* (1988) 319-327.
- [9] D. Chaum and T. Pedersen, "Transferable Cash Grows in Size." In *Advances in Cryptology –EUROCRYPT'93* (1993) 390 – 407.
- [10] I. Jeong, D. Lee, J. Lim, "Efficient Transferable Cash with Group Signatures." *ISC* (2001) 462-474.
- [11] C. Fremdt, H. Neumann, "Transferability in Coin Systems with Observer." *Communications and Multimedia Security Issues - CMS* (2001).
- [12] S. Brands, "Untraceable off-line cash in wallets with observers" In *Advances in Cryptology - CRYPTO '93* (1993) 302-318.
- [13] A. de Solages and J. Traore, "An Efficient Fair Offline Electronic Cash System with Extensions to Checks and Wallets with Observers." *Financial Cryptography* (1998) 275-295.
- [14] J. Camenisch, U. Maurer and M. Stadler, "Digital Payment Systems with Passive Anonymity - Revoking Trustees." In *Computer Security - ESORICS 96*, (1996) 33-43.
- [15] J. Liu, V. Wei, S. Wong, "Recoverable and Untraceable E-cash", In *Proceedings of IEEE Region 8 EUROCON 2001 International Conference on Trends in Communications*, (2001) 132-135.
- [16] T. Eng and T. Okamoto, "Single-Term Divisible Electronic Coins." In *Advances in Cryptology EUROCRYPT'94*, (1994) 311-323.
- [17] T. Okamoto, "An Efficient Divisible Electronic Cash Scheme." In *Proceedings of the 15th Annual International Cryptology Conference on Advances in Cryptology* (1995) 438-451.
- [18] D. Chaum, "Security without identification: Transaction Systems to Make Big Brother Obsolete." *Communications of the ACM* **28**(10) (1985) 1030 – 1044.
- [19] R. Anand and C. Veni Madhavan, "An Online, Transferable E-Cash Payment System." *Progress in Cryptology - INDOCRYPT 2000: First International Conference in Cryptology in India* (2000) 93.
- [20] Hans can Antwerpen. *Electronic cash*. Master's Thesis, CWI, 1990.
- [21] N. Ferguson, "Single Term Off-line Coins." In *Advances in Cryptology - EUROCRYPT' 93* (1993) 318-328.
- [22] J. Liu and S. Wong and D. Wong, "A New Transferable E-Cash Scheme" ACNS 2004
- [23] Y. Chan, C. Wong and C. Chan "Anonymous Electronic Voting with Non-Transferable Passes." In *Proceedings on the IFIP 16th Working Conference on Information Security* (2000) 331-340.
- [24] R. Radwin, "An Untraceable, Universally Verifiable Voting Scheme." *Seminar in Cryptology* (1995).
- [25] Y. Chan and J. Liu, "Achieving Non-Transferable and Revokable Anonymity in Mobile IP." In *Proceedings on the International Symposium on Information Theory and Its Applications* (2002).
- [26] T. Lam and V. Wei, "A mobile agent clone detection system with itinerary privacy." *IEEE 11th International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE-2002)* (2002) 68 - 73.
- [27] T. Lam and V. Wei, "Mobile agent clone detection using general transferable e-cash." *International Conference on Information Security (InforSecu '02)* (2002).
- [28] M. Jakobsson and A. Juels, "X-cash: Executable Digital Cash." In *Proceedings of Financial Cryptography '98* (1998) 16-27.
- [29] Y. Shen, T. Lam, J-C Liu and W Zhao, "On the confidential auditing of distributed computing systems." *The 24th International Conference of Distributed Computing Systems (ICDCS 2004)* (2004).
- [30] T. Lam and J-C Liu, *On the evidence based peer-to-peer resource management in distributed computing systems*, Technical Report 2003-7-2, Department of Computer Science, Texas A & M University (2003).
- [31] A. Goscinski and M. Bearman, "Resource Management in Large Distributed Systems." *ACM SIGOPS Operating Systems Review* **24**(4) (1990) 7 – 25.
- [32] N. Ferguson, "Extensions of Single-Term Coins." In *Advances in Cryptology—CRYPTO '93* (1993) 292-301.
- [33] M. Zhong, Y. Feng and Y. Yang, "Single-Term Divisible Electronic Cash Based on Bit Commitment." In *Proceedings of the Fifth IEEE Symposium on Computers and Communications (ISCC 2000)* (2000) 280.
- [34] A. Chan, Y. Frankel and Y. Tsiounis, "Easy come - easy go divisible cash." In *Advances in Cryptology - EUROCRYPT '98* (1998) 561-575.
- [35] S. Dandamudi, "A Performance Comparison of Routing Algorithms for Hierarchical Hypercube Multicomputer Networks." *ICPP 1* (1990) 281-285.
- [36] C. Raghavendra and M. Sridhar, "Optimal Routing of Bit-Permutates on Hypercube Machines." *ICPP 1* (1990) 286-290.
- [37] M. Peercy and P. Banerjee, "Distributed Algorithms for Shortest Path, Deadlock-Free Routing and Broadcasting in Arbitrarily Faulty Hypercubes." In *Proceedings of IEEE Fault-Tolerant Computing Symposium*. (1990) 218-225.
- [38] B. Chlebus, K. Diks and A. Pelc, "Optimal broadcasting in faulty hypercubes." In *Proceedings of 21st Annual International Symposium on Fault-Tolerant Computing* (1991) 266-273.

- [39] J-Y Tien, C-T Ho and W-Pa Yang, *Broadcasting on Incomplete Hypercubes*, IEEE Transactions on Computers **42**(11) (1993) 1393-1398
- [40] P-J Chuang and N-F Tzeng, "Dynamic processor allocation in hypercube computers." *ACM SIGARCH Computer Architecture News*, **18**(3) (1990) 40-49.
- [41] C-H Huang, J-Y Juang, "A Partial Compaction Scheme for Processor Allocation in Hypercube Multiprocessors." *ICPP* **1** (1989) 211-217.
- [42] Hamming Distance. URL: [http://www.its.bldrdoc.gov/fs-1037/dir-017/\\_2529.htm](http://www.its.bldrdoc.gov/fs-1037/dir-017/_2529.htm)
- [43] B. Schneier, *Applied Cryptography, Second Edition: Protocols, Algorithms, and Source Code in C*, John Wiley & Sons, Inc. (1996).
- [44] A. Menezes, P. van Oorschot and S. Vanstone, *Handbook of Applied Cryptography*, New York: CRC Press, 1997.
- [45] R. Ostrovsky, "One-way functions, Hard-on-Average Problems, and Statistical Zero-Knowledge Proofs." *Structures in Complexity Theory* **91** (1991).
- [46] M. Burmester, Y. Desmedt, F. Piper and M. Walker, "A General Zero-Knowledge Scheme." *Des. Codes Cryptography* **12**(1) (1997) 13-37.
- [47] T. Okamoto and K. Ohta, "Disposable zero-knowledge authentications and their applications to untraceable electronic cash." In *Advances in Cryptology - CRYPTO'89*, (1998) 481-496.
- [48] U. Feige, A. Fiat and Shamir, *Zero knowledge proofs of identity*, *Journal of Cryptology* **1**(2) (1998) 77 - 94.
- [49] Shamir, "How to Share a secret." *Communications ACM* **22**(11) (1979) 612 - 613.
- [50] G. Simmons, "How to (Really) Share a Secret." In *Proceedings of the 8th Annual International Cryptology Conference on Advances in Cryptology* (1988) 390-448.
- [51] G. Blakley, "Safeguarding Cryptographic Keys." In *Proceedings of AFIPS 1979 National Computer Conference* (1979) 313-317.
- [52] Josh. Benaloh, "Secret Sharing Homeomorphisms: Keeping Shares of Secret Secret (Extended Abstract)." In *Proceedings of CRYPTO' 86* (1986).

## Appendix: An implementation example using Ferguson's E-coin

### Public parameters in Ferguson's implementation:

- $n$  RSA modulus of the bank
- $v$  RSA public key of the bank, which is a large prime
- $p$  Large prime, where  $p - 1$  is a multiple of  $n$
- $g_a, g_b, g_c$  Elements of large orders in the multiplicative group of  $Z_n^*$
- $h_b, h_c$  Elements of orders  $n$  in the multiplicative group of  $Z_p^*$
- $f(\cdot)$  One-way function mapping from  $Z_n^*$  to  $Z_v^*$

### Symbols in our scheme versus Ferguson's implementation:

- $T = (a, b, c)$ , where  $a, b, c \in Z_n^*$
- $\sigma_{(T,U)} = (D, E)$ , where  $D = (C^U B)^{1/v}$ ,  $E = (CA)^{1/v}$ ,  
 $A = ag_a^{f(a)}$ ,  $B = bg_b^{f(h_b)}$ ,  $C = cg_c^{f(h_c)}$
- $\sigma_{(T,\{V_L\})} = \text{hash}(A, B, C, \{V_L\})^{1/v}$ .
- $\mathcal{F} : (T, k_q) \rightarrow (K_q)$ , where  $K_q = A^{k_q}$ .
- $\mathcal{Z} : (T, U, \sigma_{(T,U)}, x, k_q) \rightarrow (R_q, W_q)$ , where  
 $R_q = Ux + k_q$ ,  $W_q = D^x E^{k_q}$
- $\bar{\mathcal{Z}} : (T, x, (R_q, W), K_q) \rightarrow (TRUE, FALSE)$ ,  
 $TRUE$  if  $W_q^v = C^{R_q} B^x K_q$ ,  $FALSE$  otherwise.
- $\{N_p\}, \{V_p\}, \{S_p\}$ : based on (14) - (21).