

Hypercube based Divisibility Management for Integrated Credit Dispense and Computing Resource Allocation

T. C. LAM, Cheng-Chung TAN, and Jyh-Charn LIU

Department of Computer Science, Texas A&M University

College Station, TX 77843-3112

Technical Report 2006-9-2

Abstract — E-cash algorithms offer strong protection on principal anonymity, transaction authenticity and quota enforcement of credit dispense but as of now there is no study on how to integrate them with the resource allocation schemes. In this paper, we propose a hypercube based tracking system for both credit dispense and resource allocation of computing assets using E-cash algorithms. Hypercube is chosen as the common references of the two major functions, so that the study results can be tailored for a wide range of data structures that could be used for both purposes. Based on the N -divisibility framework of *disposable authentication*, we develop the analysis techniques to assess the security properties using hypercube to track credit spending and for subcube allocation for two different schemes. The first scheme is highly secure yet has high computing costs. On the other hand, the second scheme can achieve the same security goals at much lower computing costs by relaxing anonymity protection rules but adding a simple *anonymity hazard* checking routine before a subcube can be spent/allocated. Anonymity hazard refers to the condition that the unique secret (commonly called the *identity*) assigned to a token can be deciphered from messages generated from transactions even when no *double spending* occurs. Simulations results show that well known subcube allocation schemes *binary code* and *binary gray code* are free of anonymity hazards because of their restrictive allocation rules. Anonymity hazards can occur to an unrestricted subcube allocation scheme if the anonymity hazard checking routine is not devised. Our study suggests that by decoupling of credit management and resource management, highly secure and efficient computing resource management schemes can be developed based on one integrated framework.

Index Terms — Disposable Authentication, E-cash, Hypercube, N -divisible Token, Resource Allocation

I. INTRODUCTION

ANONYMITY, authenticity, and accountability (AAA) are three critical security needs for digital communications in the cyberspace. In a hostile networking environment, a principal may want to verify the authenticity of another anonymous principal, *i.e.*, the possession of a legitimate *token*

This work was supported in part by the NSF CNS-5030210 and DUE-0516825.

T. C. Lam is with the Department of Computer Science, Texas A&M University, TX 77843 USA (telephone: 979-847-8609, e-mail: brianlam@tamu.edu).

Cheng-Chung Tan is with the Department of Computer Science, Texas A&M University, TX 77843 USA (telephone: 979-847-8609, e-mail: jonastan@cs.tamu.edu).

Jyh-Charn Liu is with the Department of Computer Science, Texas A&M University, TX 77843 USA (telephone: 979-847-8609, e-mail: liu@cs.tamu.edu).

issued by the *central authority* (CA), prior to commitment of a transaction. The principal being authenticated may want to remain anonymous from others for conduction of the transaction. The CA wants to identify principals who misuse the granted tokens in order to protect the public interests. How to balance the strengths among the three types of privileges and rights to meet their security and performance requirements is important for the design of critical applications. The elegant mathematical constructs of *electronic cash* (E-cash) [1][2] that offers high strength protection on anonymity, authenticity, and accountability are excellent references for this kind of study.

E-cash represents a large family of cryptographic algorithms designed to support secure and off-line transactions between principals. In its original form a principal registered to the *bank* can *withdraw* an E-cash token to be *spent* in one transaction. This transaction cannot be associated with the principal's true *identity* (a unique *secret* assigned to the token by the bank) until the principal uses the token again, *i.e.*, *double spending*, to execute another transaction. When double spending occurs, a *double spending identification* (DSI) system built in the e-cash crypto system can decipher the identity of the offender using spent tokens which are also called the *credentials*.

Although E-cash did not achieve the ambitious goal to replace paper based currency mainly due to non-technical reasons [3], its crypto constructs based on the *blind signature* [4], *zero knowledge proof* (ZKP) [5], and *secret sharing* [6] provide important knowledge base for the design of critical Internet applications which have similar security needs. These applications include, but are not limited to, Internet voting [7][8], population survey [9], secure collaboration [10], role based information systems [11], and sharing of critical computing resources [12], *etc.* These applications all call for strong protection of principals' identities, token authenticity, and

regulated disposal of privileges, but the resources being managed in these and many other applications have very different properties from currency asset which is the primary focus of most E-cash work.

To make E-cash solutions more applicable to a broad range of Internet applications, the underlying properties of resources need to be taken into account more carefully in the E-cash design, and vice versa. Among the myriad of properties that would need to be considered in system resource management, in this paper we focus on the *divisibility* management of system resource and user credits. Divisibility refers to the abilities for the stakeholders to divide their assets, i.e., *credits* for users and *service resources* for service providers, into portions in order to render services. Divisibility is at center of investigation in the dispense of electronic credits, i.e., divisible E-cash. It is also the main focus on optimal allocation of computing resources. But there is no discussion on the relationships between the two types of designs in the literature.

An *N-divisible* E-cash token [1] allows a principal to engage in multiple transactions anonymously using the same token until the total amount of spending reaches the quota limit N . When using *N*-divisible tokens to access computing resources the tracking scheme of spending records should be made compatible with the resource allocation protocols, so that both the performance goals (resource utilization rate, resource fragmentation, *etc.*) and the security goals can be harmonized.

In the *coupon-based* schemes [13]-[15], each credential represents one unit of the spent asset. The spending patterns are monitored simply by counting the number of credentials derived from a token. In a more sophisticated approach the binary tree is used to keep track of spending patterns [16]-[18] of an *N*-divisible token, where a node located at the i^{th} level denotes 2^i units of the asset. In the second part of [16], Eng-Okamoto developed a *general disposable*

authentication (DA) model that can transform a non-divisible token into its N -divisible counterpart for a large class of E-cash schemes [19]-[22]. It is important to note that the work reported in this paper does not consider the first part of [16] which is a specific DA scheme based on Schnorr's identification [21], nor the one presented in [5], which is a specific DA scheme prior to its development for divisibility.

In this paper, we investigate how to design hypercube based N -divisible tokens, on the basis of the cryptographic framework of DA, for computing resource allocation. Instead of using tree as in [16], hypercube is chosen for our study because it is a widely used data structure for allocation of computing resources [23]-[25]. Being a superset of numerous data structures, *e.g.*, tree, mesh, star, *etc.*, some interesting insights on the interplay between the crypto constructs and resource allocation rules are summarized as follows.

- (1) A *hypercube dependency graph* is devised to track the spending patterns of a hypercube based N -divisible token, using both the top-down and bottom-up dependency analysis techniques. Only the latter approach was used for the tree based solution in DA.
- (2) In addition to the *same-node* and *route-node* double spending violations, which were classified in [16], a new type of *shared-node* double spending violations is identified for hypercube dependency graphs. Tracking of this new violation type makes the analysis much more complicated than the tree based solutions.
- (3) Unrestricted subcube allocation schemes could lead to *anonymity hazards*, *i.e.*, leakage of identity information even with no double spending offenses, unless a costly solution is considered. We found that anonymity hazards can be detected and avoided at small cost of *fragmentation* ($< 2\%$ in our simulation) when there is no restriction on the

subcube allocation scheme. Because of the more restrictive rules in subcube allocations, two commonly used subcube allocation schemes are found to be immune from anonymity hazards.

The rest of this paper is organized as follows. In section II, we will discuss the hypercube based divisibility management, including the system architecture, design of tokens and credentials, and the dependency among encryption/decryption keys and verification keys, under different spending patterns. In section III, we will derive and analyze the crypto constraints needed to assure the AAA security requirements. Two crypto constructs are compared in section IV to illustrate the tradeoff between security strength and performance cost when different subcube allocation rules are used. Simulation results will be delivered in section V, with discussion and conclusion in sections VI and VII.

II. HYPERCUBE BASED DIVISIBILITY MANAGEMENT

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

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

$d(p, q) = 0$ 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: 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 two vertices of G , and $h(p,$

q) is the Hamming distance [26] between p and q over the alphabets $\{0, x, 1\}$. p is the parent of q if $(p, q) \in E$. Similar nomenclatures follow the convention of tree.

G_n presents the dependency relationship between all permissible subcube configurations of Q_n . To avoid ambiguity in the subsequent discussions, we use “node” to describe an atomic node in Q_n and “vertex” to describe a node in G_n , respectively. Fig. 1 depicts the dependency graph of Q_2 , where each vertex represents a subcube that can be derived from Q_2 . A vertex on G_n is marked when a corresponding subcube is allocated by its corresponding token. A double spending violation occurs if any leaf vertex is allocated twice (in two subcube allocations.)

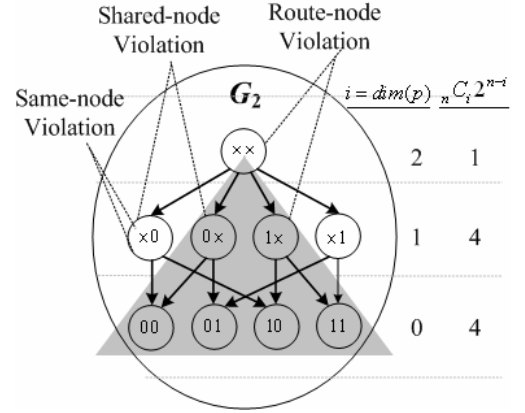


Fig. 1. Dependency Graph of G_2 .

Note that the shadowed area highlights a binary tree embedded into G_2 , implying that the hypercube based N -divisible token is required to handle more complicated security conditions than its tree based counterpart. From the example in Fig. 1 it shows that three types of double spending violations, *same-node violation*, *route-node violation*, and *shared-node violation*, can occur to 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 or descendant vertex, are spent. A shared-node violation occurs when two vertices, with no ancestor or 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 that needs to be addressed.

Now, we outline some basic issues in the resource allocation schemes. Minimizing subcube fragmentation and locating the largest set of useable subcubes is the focus of many subcube

allocation schemes. Compact representations of Boolean expressions are important to achieve the performance goals, but it may cause unintended double spending. For instance, $(0\times,\times 0)$ represents three nodes $(00,01,10)$, and this is considered a legal allocation when they are allocated to one user. Nevertheless, spending both $(0\times)$ and $(\times 0)$ terms in E-cash constitutes a double spending because node (00) is involved in two transactions. No double spending would occur if the set of nodes are allocated using one of the following three Boolean expressions: $(0\times,10)$, $(\times 0,01)$, or $(00,01,10)$, where redundant Boolean terms in subcubes are eliminated to prevent incorrect marking of 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 DA framework, we design the hypercube based divisible tokens, and derive the conditions to satisfy the AAA security constraints. Comparing with tree based solutions, applying hypercube to the DA 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 multiple-parent structure. This situation is even more complicated for hypercube due to its highly connected construct. For the DSI subsystem to identify offenders of all types of double spending violations, more information that can guarantee deciphering of principal's identity under such conditions needs to be included in the credentials. However, doing so may lead to *anonymity hazard*, which refers to the situation when the principal's identity can be deciphered from multiple instances of subcube allocations even when no double spending occurs¹.

¹ An example on this case will be given in Table IV.

Granted, one could develop a more sophisticated mathematical system to eliminate anonymity hazards but a much more practical solution approach is based

TABLE I
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

on subcube allocation schemes because anonymity hazards can be recognized via simple rules. To gain realistic understanding for such a tradeoff analysis, three subcube allocation schemes, *random code* (RC), *binary code* (BC), and *binary reflected gray code* (BRGC) [23], are analyzed and simulated for our study. Later we will show that anonymity hazards occur to RC, which is rarely used in the real world, but not to the widely adopted subcube allocation schemes, BC and BRGC.

In RC, there is no restriction on which available subcube to use for the subcube 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 I). A subcube request is matched with the first subcube configuration on the list in a linear search order. In either case, to search for an available subcube 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 further details of BC and BRGC, please refer to [23].

BC, BRGC or other similar non-exhaustive subcube allocation schemes would utilize a fraction of available subcubes, as the example depicted in Fig. 2a and 2b, where only the highlighted vertices can be used in BC and BRGC, respectively. As shown in Fig. 2a, the vertices that can be used in BC are the vertices that can be used in the tree-based schemes. As shown in Fig. 2b, BRGC has more flexible spending patterns than BC.

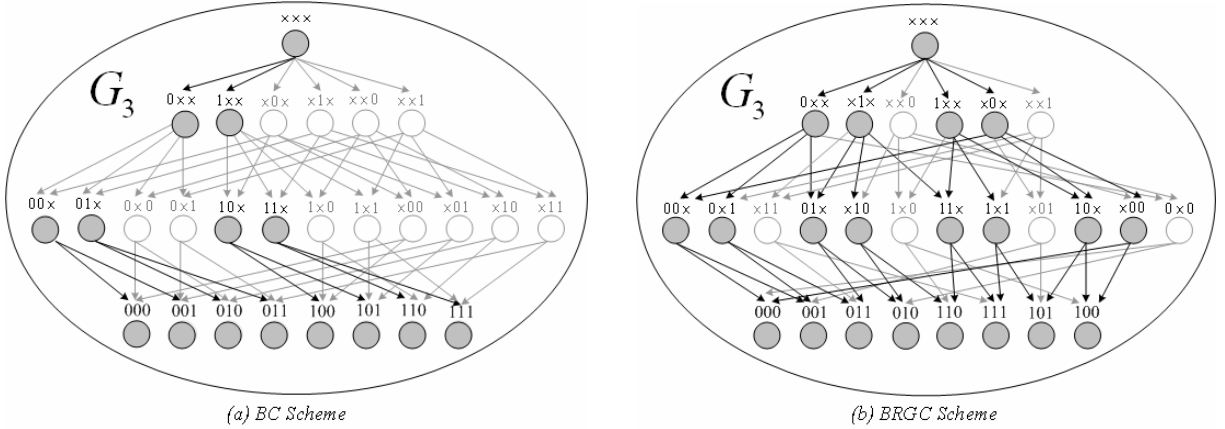


Fig. 2. Usable Vertices under (a) BC, and (b) BRGC Schemes.

A. System Architecture for Security Management

There are three types of principals in the system: the CA, the *resource owner*, and the *resource consumer*. The CA is a trusted third party with the sole responsibility of issuing identities and tokens. All consumers are required to register with the CA to receive their identities through some secure means. Then, the five major protocols among the principals are depicted in Fig. 3.

(i) *withdrawal* of token(s) from the CA to a resource consumer, (ii) *allocation* of a subcube (their access rights) from the resource owner to a consumer, using the owner's digital signature and the consumer's token, (iii) *transfer* of the access rights from one consumer to another, using both their tokens, (iv) *consumption* of the resources from the owner by the consumer, using the consumer's token, and (v) DSI by the owner, using the credentials produced by the consumers' tokens in (iii) and (iv).

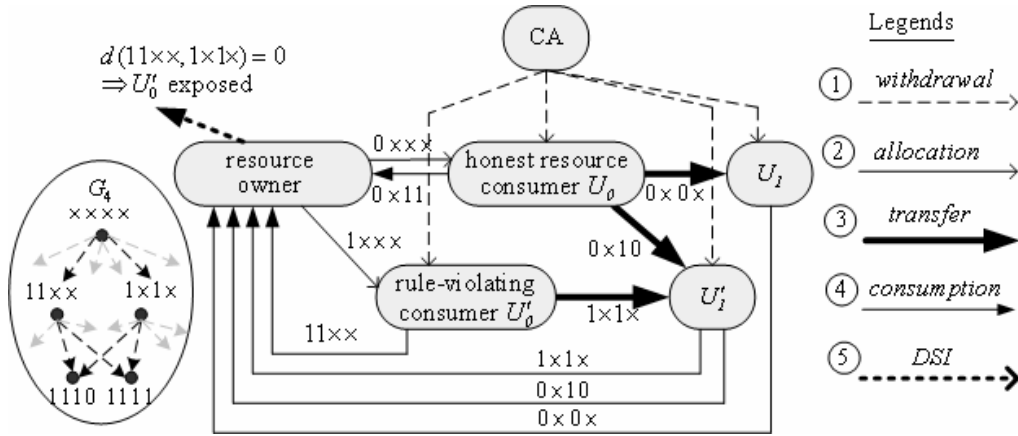


Fig. 3. System Architecture.

A resource consumer can stay anonymous in transfer/consumption protocol, whereas the CA and the resource owner are assumed to be well-known to the public. ZKP is used for both transfer and consumption protocols to assure that the authentic access rights are presented by an anonymous principal. The major difference between the transfer and consumption protocols is that, in the ZKP based transfer, both the *prover* (who possesses the token) and the *verifier* (who verifies the token) are resource consumers, but in the consumption protocol, the verifier is the resource owner. Another technical difference is that the challenge message in the ZKP based transfer is produced using a part of the verifier's token, so as to guarantee the accountability on the next transfer/consumption by this verifier [27]. On the contrary, a randomly generated number is sufficient to produce the challenge in the consumption protocol, because no further transfer/consumption is expected after this point.

In the example depicted in Fig. 3, consumers U_0 , U_1 , U_0' , and U_1' first withdraw some tokens from the CA to prepare for subsequent transactions. On the requests of U_0 and U_0' , the owner allocates subcubes $0\times\times\times$ and $1\times\times\times$ to U_0 and U_0' , respectively. U_0 yields $0\times0\times$ and 0×10 to U_1 and U_1' , and lets itself access 0×11 from the owner. Then U_1 and U_1' consume from the owner the subcubes transferred by U_0 . So far, none of the hypercube nodes is used for transfer or

consumption more than once. Therefore, service provided by $0\times\times = \{0\times0\times, 0\times10, 0\times11\}$ is completed at the owner site after U_1 and U_1' finish their consumptions. However, a double spending violation occurs when U_0' consumes $11\times\times$ from the owner, and transfers $1\times1\times$ to U_1' , because $111\times = \{1110\ 1111\}$ is used twice. DSI system must assure identification of U_0' .

In each ZKP instance of transfer (or consumption), a credential is produced by the prover and the verifier, which indicates the subcube p being transferred from the prover and p is a relative address that needs to be interpreted in a series of credentials (*cascaded credential*.) For the example in Fig. 3, the 0×10 received by the resource owner is comprised of three parts: the owner's signature on $p = 0\times\times$ (from the allocation between the resource owner and U_0), the credential marked $p = \times10$ (from the transfer between U_0 and U_1'), and the credential marked $p = \times$ (from the consumption between U_1' and the resource owner).

In subsequent discussions, we will focus on the divisibility management for transfer protocol. To keep track of the subcube divisibility using the dependency graph, in the first design step we adopt the secret information sharing technique in [16], where each vertex is associated with a *secret share*, and a *key pair* that can verify/decipher the prover's identity from the secret share. To help understanding our scheme, we summarize in Table II some important facts of DA and the relationship among the secret shares and the key pairs, where D , F , F' and G represent some polynomial time functions. For details, please refer to [16].

In the non-divisible version of DA, a *three-move* ZKP protocol is used, which allows a principal U_A to prove to another principal

TABLE II
DISPOSABLE AUTHENTICATION

AAA Requirement	Information Needed	Remarks
Verify $m = f(x)$ without knowing x	VU1: $X, (E, Y)$ available	$X = F(x, r) = F'(m, r), Y = D(r, x, E) m = f(x)$ iff $G(m, X, E, Y) = true$
Decipher x	DU1: $r, (E, Y)$ available, OR DU2: $(E, Y), (E', Y')$ available	r is a symmetric key (2, k) secret sharing
Keep x secret	Neither DU1 nor DU2 available	

U_B that U_A knows some secret x that satisfies $m = f(x)$, without letting U_B know the plaintext of x , where f is a publicly known one-way function, x is a private knowledge of U_A that can be mapped to the identity of U_A , and m is the public counterpart of x , that will need to be exposed to U_B during the transfer to produce a valid credential. Let r be the *delegation key* randomly selected by U_A , and $X = F(x, r) = F'(m, r)$ the corresponding *verification key*. U_A starts the ZKP protocol by sending (m, X) to U_B . Then, U_B replies a random *challenge* E . U_A answers the challenge by a *response* $Y = D(x, r, E)$. r serves as the symmetric key that allows x to be encrypted/decrypted to/from the secret share (E, Y) . (x, r) must be kept private by U_A to guarantee anonymity of U_A . Without knowing x , U_B can verify that $m = f(x)$ by checking that $G(m, X, E, Y) = \text{“true”}$, given the following data:

VU1: a secret share (E, Y) , and the verification key X which delegation key r creates (E, Y) .

x is decipherable, if, and only if, either of the following two is available:

DU1: a secret share (E, Y) , and the delegation key r that creates (E, Y) .

DU2: two secret shares $(E, Y), (E', Y')$, where $E \neq E'$, and Y, Y' are created by the same (x, r) pair.

x can be deciphered by DU1 because by definition r is the symmetric key (deciphering key) to do so. x can be deciphered by DU2 because in DA there exists a *knowledge extractor* [16], whose construct is based on a $(2, k)$ secret sharing scheme, where k is an integer not less than two. Using the verification and deciphering rules above, we design the hypercube based N -divisible tokens and their credentials.

Next, we propose a new mechanism called *key dependency map* (KDM) to keep track of the correlation of the randomized key pairs among different vertices. When a subcube Q_x is transferred, a set of vertices, representing this subcube together with other subcubes, say Q_y , that

could be subject to double spending offenses, will be selected to have their secret shares and/or keys released in the credential. This way, if Q_y is indeed spent in the future, DSI can identify the offending patterns. The constraints to select these 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 III.

B. Hypercube Based Token and Credential

Following the DA framework, the token resulted from withdrawal is of the following format:

$$T = (m, x), \quad (2)$$

Through a *blind signature* scheme [4], 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 the token T and the hypercube dependency graph (that contains the spending patterns) should be integrated into the credentials to enforce DSI, while still protecting the anonymity of the token owner. Following the high level constructs of DA, the format of the credential produced by hypercube based divisible tokens, in the transfer of a subcube p , is

$$C = (p, m, DK(u(p)), VK(v(p)), VK(L), SS(s(p))). \quad (3)$$

$DK(V)$ represents the outputs produced from the publicly known function DK , which takes a set of vertices V in G as inputs, and gives the delegation keys of V as outputs. Similarly, $VK(V)$ and $SS(V)$ represent the outputs produced 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 later, when p is used in a transfer, other subcubes may also need to be considered in generation of DVS values.

How to control the generation and exposure of DVS values to enforce DSI, and to protect anonymity of the rule-abiding users, is the focal issue in the hypercube based N -divisibility management designs. Depending on the subcube usage patterns, some DVS values need to be

put as a part of the credential, *i.e.*, exposed them to public, but others should be kept by the token owner to avoid compromise of anonymity, *i.e.*, x becomes decipherable.

Exposure of the DVS values is controlled by the constructs of $u(p)$, $v(p)$, $s(p)$, and L in (2), and their interdependency relationships. $u(p)$, $v(p)$, and $s(p)$, collectively called the *exposure functions*, are publicly known functions which produce a collection 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 IV.

In subsequent discussions, we use the term “directly exposed” to describe the keys included in the credentials, *i.e.*, $DK(u(p))$, $VK(v(p))$ and $VK(L)$ in (3), without using KDM. In contrast, we use the term “indirectly exposed” to describe the keys derived from KDM, using the keys directly exposed from the credentials as the KDM inputs.

Separating the notations of vertex sets from their corresponding keys or secret shares is for the convenience of subsequent analysis that requires set intersection, union, and negation operations. For example, to check if the vertex p has both its delegation key and secret share exposed in the credential is equivalent to check if $u(p) \cap s(p) \neq \phi$. Otherwise, we cannot perform checking using the expression $DK(u(p)) \cap SS(s(p)) \neq \phi$ because the left-hand side and the right-hand side of the intersection operator are from different domains.

Similar to the classical tree based schemes, hypercube based divisible schemes need to satisfy:

(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 there are no such p and q that $d(p, q) = 0$, then x cannot be deciphered from their credentials.*

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

C. Key Dependency Map (KDM)

Recalled that each vertex in G_n is associated with a delegation/verification key pair. KDM is a function which keeps track of the one-way mapping relationship among the key pairs of different vertices. When p is spent (assuming $p \in s(p)$), the keys of vertices directly exposed in a credential (defined by $u(p)$ and $v(p)$) will be used as the input of KDM, so that the delegation keys of some other vertices that are *susceptible* to double spending violation in the future, *i.e.*, $d(p, q) = 0$, can be indirectly exposed from the KDM output. 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 delegation keys of *non-susceptible* vertices to prevent anonymity hazards. In addition, in order to supervise all susceptible future spending, only some susceptible vertices (with respect to p) need to have their delegation keys exposed because double spending condition is reflexive, *i.e.*, $d(p, q) = 0 \Leftrightarrow d(q, p) = 0$, it does not matter whether $dim(p) > dim(q)$ or $dim(q) > dim(p)$. For example, in the top-down KDM, from which the keys of a 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 all susceptible vertices at dimensions lower than $dim(p)$. Similarly, in the bottom-up KDM we only need to expose the delegation keys of all susceptible

vertices for dimensions higher than $dim(p)$. The exposure of keys for these susceptible vertices are illustrated in the shaded areas in Fig. 4(a) and 4(b). Let p be a vertex in G_n that represents the subcube being used to track a transfer transaction. H is a publicly known collision-free one-way hash function. The symbol “ \parallel ” denotes a concatenation operator between two ternary strings. $j = (n - dim(p))$ is the number of parents of p and (p_1, \dots, p_j) are the parents of p listed in ascending order. Ordering of vertices can be done by substituting the value of “ \times ” by “2”, and then order them as ternary numbers, *e.g.*, the ternary number of 0×1 is $7 = 0\cdot 3^2 + 2\cdot 3^1 + 1\cdot 3^0$, and for $\times 10$ it is $21 = 2\cdot 3^2 + 1\cdot 3^1 + 0\cdot 3^0$. The building block for top-down KDM is given by the following equations:

$$r_p = H(X_{p_1} \parallel \dots \parallel X_{p_j} \parallel p) \quad (4)$$

$$= H(F'(m, r_{p_1}) \parallel \dots \parallel F'(m, r_{p_j}) \parallel p), \quad (5)$$

Note that while (4) follows the basic structure of the tree-based (*t-value*, *r-value*) key generation function $r_p = H(X_{left-child(p)} \parallel X_{right-child(p)})$ in [16], which takes the verification keys of two children (*vs.* j parents) as inputs, in the hypercube-based design an additional p is appended at the end of the hash input in (4) to distinguish the keys for children of the root vertex. The

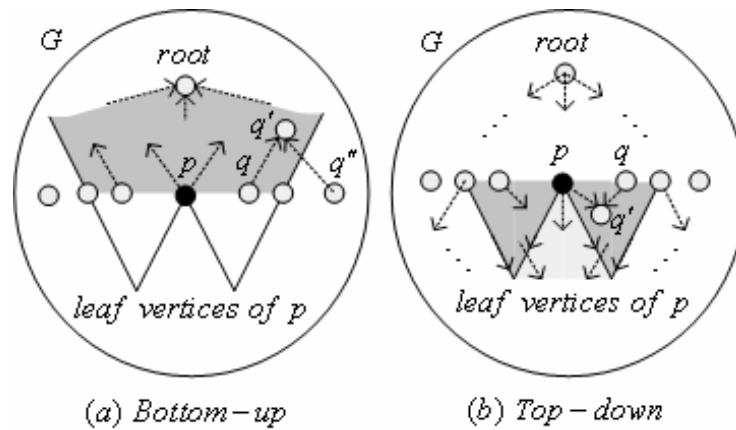


Fig. 4. Key Dependency Map (KDM).

subtle change (of adding p in (4)) required to meet the security goals makes the bottom-up analysis approach used in [16] not efficient for our hypercube based design. The shared-node violation depicted in Fig. 4 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 , respectively, in the bottom-up KDM scheme. q' must be susceptible to shared-node violation because the leaf vertices shared by p and q will also be shared by p and q' . In contrast, this is not necessarily true for q'' . Because we cannot expose the keys of q' (a susceptible vertex) by the bottom-up KDM without exposing the delegation key of q'' (a non-susceptible vertex), the bottom-up approach is unsuitable for our design.

Now we consider the top-down KDM scheme. 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 top-down KDM, we only need to expose the keys of the parents of q' , which are also susceptible. Therefore, top-down KDM is much more efficient than the bottom-up approach. In the rest of our discussions, only the top-down KDM is considered, unless explicitly specified otherwise.

The pseudocode of KDM, based on (4) and (5), is depicted in Fig. 5, 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 leaf level is reached, *i.e.*, $i = 0$. 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.

By controlling what 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

```

KDM(DKin, VKin, i)
01:  set DKout = DKin;  set VKout = VKin;
02:  if i = n then
03:      set VKout = VKout ∪ {X = F'(m, r) | ∀ r ∈ DKout};
04:  else if i = 0 then
05:      return (DKout, VKout);
06:  endif:
07:  for every vertex q at dimension (i - 1) do
08:      if ∃ Xq1, Xq2, ..., Xqj ∈ VKout, then ∃ q1, q2, ..., qj are parents of q
09:          set DKout = DKout ∪ {rq = H(Xq1 || Xq2 || ... || Xqj || q)};
10:          set VKout = VKout ∪ {Xq = F'(m, rq)};
11:      endif:
12:  endfor:
13:  KDM(DKout, VKout, i-1);

```

Fig. 5. Pseudocode for Key Dependency Map (KDM).

transfer. For example, given r_{root} (randomly generated by the prover before 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 it is invoked by $(DK_{in}, VK_{in}, i) = (\{r_{root}\}, null, n)$. The consumer (prover) 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.6. 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. In spite of the similarity between Fig. 6 and the figure in [16], the high connectivity in hypercube dependency graph makes the security analysis on A1, A2, and A3 much more complicated than its tree based

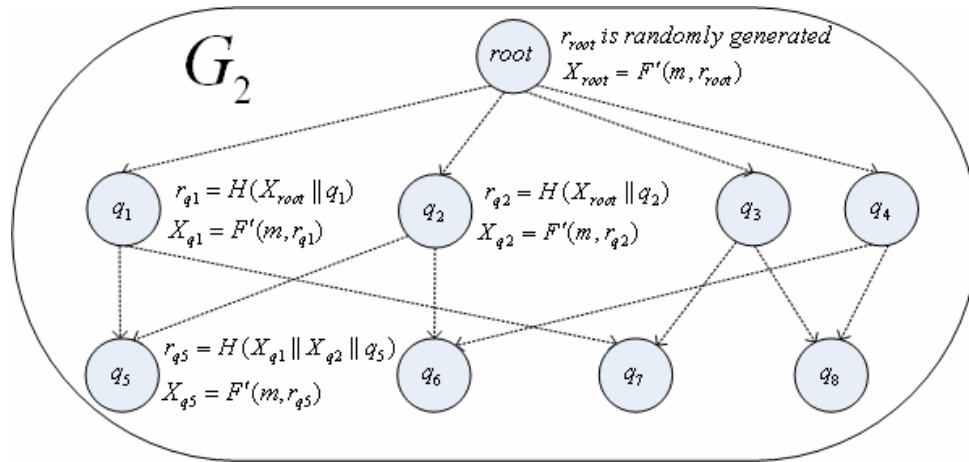


Fig. 6. High Level View of KDM.

counterpart.

III. CRYPTO CONSTRAINT ANALYSIS

In this section, we analyze the crypto 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 determine the conditions of exposure functions that would be subject to anonymity hazards and 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 analysis of indirect exposures from multiple spending instances of p_1, p_2, \dots, p_j , it is more convenient to represent them in terms of their KDM input/output. In particular, we denote

$$(DK_{KDM|p_1, \dots, p_j}, VK_{KDM|p_1, \dots, p_j}) \quad (6)$$

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

$$(DK(u(p_1) \cup \dots \cup u(p_j)), VK(v(p_1) \cup \dots \cup v(p_j) \cup L), n). \quad (7)$$

Note that the KDM input in (7) can always start the KDM execution from level n , regardless the dimensions of p_1, \dots, 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. 5, 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 (6) contains all delegation/verification keys in G_n that can be derived from the credentials produced from the transfers of p_1, \dots, p_j . Let DK^{-1} and VK^{-1} be the inverse functions of DK and VK , respectively. The vertex sets for the delegation keys and the verification keys in (6) are respectively denoted by

$$u_{KDM}(\{p_1, \dots, p_j\}) = DK^{-1}(DK_{KDM|p_1, \dots, p_j}), \text{ and} \quad (8)$$

$$v_{KDM}(\{p_1, \dots, p_j\}) = VK^{-1}(VK_{KDM|p_1, \dots, p_j}). \quad (9)$$

Similar notations are described as follows: we replace the subscript KDM in (6), (8) and (9) by $KDML$ if L is removed from (7), and we replace this subscript by $KDML, q$ if both L and q are removed from (7), for $q \in (u(p_1) \cup \dots \cup u(p_j)) \cup (v(p_1) \cup \dots \cup v(p_j))$. Some KDM results related to A1, A2, A3, and the key generation are summarized in Table III.

A. Crypto Constraints for Authenticity (A1)

The crypto constraints for A1 can be analyzed using the set diagram in Fig. 7, 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

$$v_{KDM}(\{p\}) \supseteq s(p), \quad (10)$$

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 non-divisible version of DA (in Table II), which only needs to examine one

TABLE III
INPUT/OUTPUT OF KDM

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

verification key. In contrast, we need to check all verification keys in $VK(v_{KDM}(\{p\}))$.

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 $VK(v_{KDM}(\{p\}))$ based on $VK(L)$, L needs to satisfy two constraints:

$$\ell(p) = L \cap v_{KDM \setminus L}(\{p\}) \neq \phi, \text{ and} \quad (11)$$

$$\ell(p) \supset L \cap v_{KDM \setminus L, q}(\{p\}), \forall q \in u(p) \cup v(p), \quad (12)$$

(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, (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 (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 subcube 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). Thus, the integrity check of $VK(L)$ is to make sure that the same collection of keys is used in different spending instances. To do

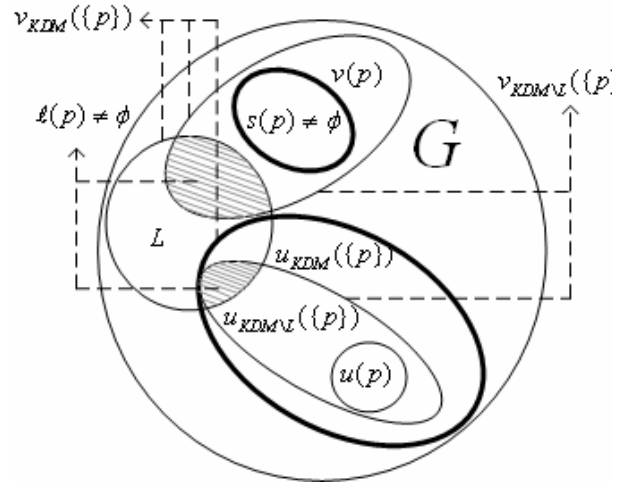


Fig. 7. Crypto Constraints for A1

this, two approaches are addressed in [12]: (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 [12].

B. Crypto Constraints for Anonymity and Accountability (A2 and A3)

The crypto constraints for A2 and A3 are depicted in the set diagram of Fig. 8. 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. 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:

$$\text{DU1: } u_{KDM}(\{p, p'\}) \cap (s(p) \cup s(p')) \neq \phi, \text{ or} \quad (13)$$

$$\text{DU2: } s(p) \cap s(p') \neq \phi. \quad (14)$$

By (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 (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

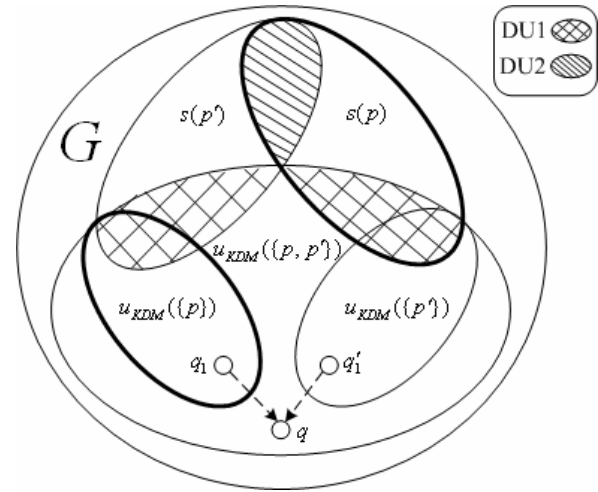


Fig. 8. Crypto Constraints for A2 and A3.

delegation key, so A2 is guaranteed based on DU2. The crypto constraints for A3, which are complements of (13) and (14), are given by:

$$\overline{\text{DU1}}: u_{KDM}(\{p_1, \dots, p_t\}) \cap \left(\bigcup_{j=1}^t s(p_j) \right) = \phi, \quad \text{and} \quad (15)$$

$$\overline{\text{DU2}}: \bigcap_{j=1}^t s(p_j) = \phi, \quad (16)$$

where p_1, \dots, p_t are vertices without double spending violations. (15) and (16) can be interpreted by using counter arguments of (13) and (14).

IV. CRYPTO CONSTRUCTS AND SUBCUBE ALLOCATION SCHEMES

Constructs of the exposure functions, $(u(p), v(p), s(p))$, and L determine the AAA properties. When they are used with subcube 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 subcube 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 $\{p_1, p_2, \dots, p_t\}$ in (21) are not tracked. As such, a simple checking routine needs to be added to the subcube allocation scheme to prevent their occurrences. Since simulation results show that subcube fragmentation related to anonymity hazards is less than 2%, it is highly effective to use subcube allocation rules to avoid anonymity hazards, rather than eliminating anonymity hazards unconditionally.

A. Scheme I

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

$$s(p) = \{\text{leaf vertices of } p\}, \quad (17)$$

$$v(p) = \{\text{leaf vertices of } p\}, \quad (18)$$

$$u(p) = \phi, \quad (19)$$

$$L = \{\text{all leaf vertices in } G\}. \quad (20)$$

It is relatively straightforward to see that (17) – (20) satisfy all constraints in (10) – (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 argument we assert that it is very 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 $u_{KDM}(\cdot)$ in (15) is a non-linear function, *i.e.*,

$$u_{KDM}(\{p_1, \dots, p_t\}) \supsetneq \bigcup_{j=1}^t u_{KDM}(\{p_j\}). \quad (21)$$

The inequality in (21) is caused by the multi-parent, multi-child structure in G_n . It implies that delegation keys produced individually from p_1, \dots, 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. 8, 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 (21) is assured by (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.

B. Scheme II

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

$$s(p) = \{p\}, \quad (22)$$

$$v(p) = \{p\}, \quad (23)$$

$$u(p) = \{\forall q \mid d(p, q) = 0, p \neq q, \dim(q) = \dim(p)\}, \quad (24)$$

$$L = \{\text{all leaf vertices in } G\}. \quad (25)$$

In contrast to scheme I, this scheme reduces the computation cost in the transfer protocol by selecting (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 (22)-(23) and the constructs of KDM, we have $v_{KDM}(\{p\}) \supseteq v(p) = s(p)$, and hence, (10) is satisfied. To show that (22)-(25) also satisfy (11)-(12) for A1 and (13)-(14) for A2, we need to study their KDM outputs. (24) represents all vertices susceptible to shared-node violation with p at dimension $\dim(p)$. Together with (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

$$v_{KDM,L}(\{p\}) = \{\forall q \mid d(p, q) = 0, \dim(q) \leq \dim(p)\}, \quad (26)$$

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 parent is 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 (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). Fig. 9 depicts how the vertex marking scheme works in G_3 .

Suppose $p = 0xx$ is a vertex used for transfer. To evaluate $v_{KDM}(\{0xx\})$, we first mark all descendants of $0xx$, *i.e.*, $\{00x, 01x, 0x0, 0x1, 000, 001, 010, 011\}$, and then we mark all unmarked ancestors of these vertices at levels $\leq \dim(p)$, *i.e.*, $\{x00, x01, x10, x11, 0xx, x0x, x1x, xx0, xx1\}$. We can see that $u(p) = \{x0x, x1x, xx0, xx1\}$ and $v(p) = \{0xx\}$ have formed all the dependencies

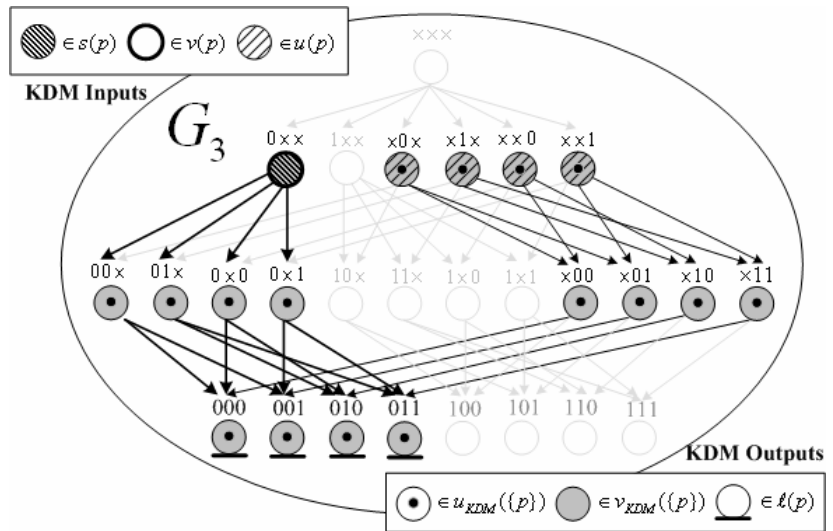


Fig. 9. Vertex Marking Scheme for Susceptible Vertices with Respect to $p = 0xx$.

that are required to compute the verification keys of all marked vertices, *i.e.*, nodes susceptible of double spending with p at dimensions $\leq \dim(p)$. Given $v_{KDM}(\cdot)$ defined in (26), (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, (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 (22)-(25) satisfy (10)-(12), Scheme II satisfies all crypto constraints for A1.

To show that (22)-(25) also satisfy the crypto constraints for A2, we need to show that they satisfy either (13) or (14) for any vertices p and p' , where $d(p, p') = 0$. For the case of $p = p'$, it is straightforward that (14) is satisfied. For $p \neq p'$, $\dim(p) \geq \dim(p')$, we first show that

$$u_{KDM}(\{p\}) = \{\forall q \mid d(p, q) = 0, p \neq q, \dim(q) \leq \dim(p)\}. \quad (27)$$

(26) and (27) are depicted in Fig. 9, by the highlighted vertices, and the set of vertices with a dot marked inside, respectively. (26) and (27) are almost identical except that $\{p\}$ is excluded from (27). In (27), the vertices at dimension $\dim(p)$ are contributed by (24), while those at dimensions lower than $\dim(p)$ are justified by the vertex marking scheme we described before. Based on (27), we have

$$u_{KDM}(\{p\}) \cap s(p') = p'. \quad (28)$$

In addition, based on (21) or $u_{KDM}(\{p, p'\}) \supseteq u_{KDM}(\{p\})$,

we conclude that (13) is satisfied for A2.

Table IV depicts a spending configuration that can lead to anonymity hazards in G_3 , where $p_1 = 1 \times 0$, $p_2 = 01 \times$, p_3

TABLE IV
ANONYMITY HAZARD SCENARIO IN G_3

Spent vertex	Exposure functions
$p_1 = 1 \times 0$	$u(p_1) = \{10 \times, 11 \times, \mathbf{x00}, \times 10\}$
$p_2 = 01 \times$	$u(p_2) = \{\times 10, \times 11, \mathbf{0x0}, 0 \times 1\}$
$p_3 = \times 01$	$u(p_3) = \{0 \times 1, 1 \times 1, \mathbf{00x}, 10 \times\}$
$p_4 = 000$	$s(p_4) = \{\mathbf{000}\}$

$= \times 01$, and $p_4 = 000$ are four vertices for transfers². 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), *i.e.*, $r_{000} = H(X_{00\times} || X_{0\times 0} || X_{\times 00} || 000)$, because $\times 00 \in u(p_1)$, $0\times 0 \in u(p_2)$ 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.

Scheme II is more efficient than Scheme I but it does not guarantee A3 unconditionally. It is designed to work with a subcube allocation scheme to detect and avoid anonymity hazards before a subcube is spent/allocated. We propose a simple *anonymity hazard test* (AHT) algorithm to test whether or not a vertex p (of the requested subcube size) is subject to anonymity hazard with respect to the previously spent vertices, so that only hazard-free subcubes will be spent and allocated. When all available vertices at the requested subcube 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 V. In this example, $p_1 = 1\times 0$, $p_2 = 01\times$, $p_3 = 000$, and $p_4 = 111$ are spent. If the next subcube requested is a Q_1 then the only available vertex of this size is $\times 01$. As shown in Table IV, an anonymity hazard will result from spending of $\times 01$, after p_1 , p_2 , and p_3 have been spent but the anonymity hazard is eliminated when $\times 01$ is divided into two smaller units $\{001, 101\}$ for spending. Subcube

fragmentation increases the computation and communication overheads. Fortunately, our simulation results show that fragmentation caused by anonymity hazard in Scheme II is negligible, even when arbitrary

TABLE V
FRAGMENTATION SCENARIO IN G_3 WHEN Q_1 IS
REQUESTED

Spent vertex	Exposure functions
$p_1 = 1\times 0$	$u(p_1) = \{10\times, 11\times, \times 00, \times 10\}$
$p_2 = 01\times$	$u(p_2) = \{\times 10, \times 11, 0\times 0, 0\times 1\}$
$p_3 = 000$	$s(p_3) = \{000\}$
$p_4 = 111$	$s(p_4) = \{111\}$

² We will show in Appendix that the minimum number of vertices to cause anonymity hazard in Scheme I is four.

subcube allocation scheme is considered. Our extensive simulations further show that no anonymity hazard can be detected when Scheme II is integrated with two popular subcube allocation schemes, BC or BRGC.

V. SIMULATION 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, whose pseudocode is depicted in Fig. 10. 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 represents that its verification key has been exposed from past spending instance(s). A red vertex represents that both the verification key and the secret share of this vertex have been exposed from past spending instance(s). A gray vertex represents that its delegation/verification key is not exposed from past spending instance(s) but will be exposed if the current spending instance succeeds.

The algorithm returns TRUE when an anonymity hazard is detected at Lines 03 and 14 for the pending subcube 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) and (5), the delegation key of p can also be derived

```

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  $u(p) \cup v(p)$  to gray;
06:   set  $i = \dim(p) - 1$ ;
07:   for all vertex  $q$  at dimension  $i, i \geq 0; i--$  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. 10. Pseudocode of Anonymity Hazard Test (AHT) Algorithm.

from these delegation keys or verification keys. Furthermore, by (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 represents 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 (22) if p is spent. Since q is originally red, its secret share has been 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 delegation keys or 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 (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. 11 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 subcube 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 $p_tmp = 0, 1, \dots, 3^n - 1$ and use its ternary representation to determine $i = \dim(p_tmp)$. In this way, the probability in producing requests of extremely large/small subcubes 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 subcube 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 subcube of the same size can be found from the unspent subcube pool based on the subcube allocation rule 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
27:     goto 02; \\ next subcube request

```

Fig. 11. Pseudocode of the Simulation Program.

The simulation results for G_4 to G_{10} are depicted in Fig. 12 and 13 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, while the *fragmentation ratio* is the proportion of fragmentations (caused by anonymity hazards) versus the total number of subcube requests in Line 07. For RC, the anonymity hazard ratio increases in a nearly 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 therefore AHT is not needed for these two schemes.

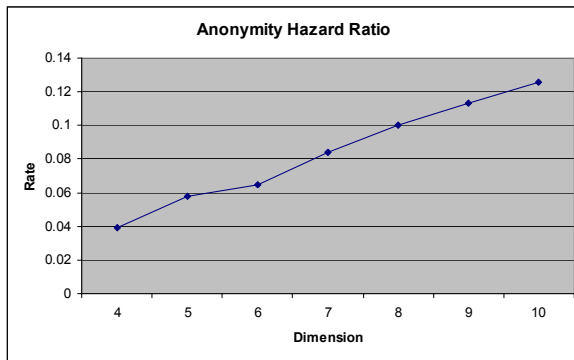


Fig. 12. Anonymity Hazard Ratio.

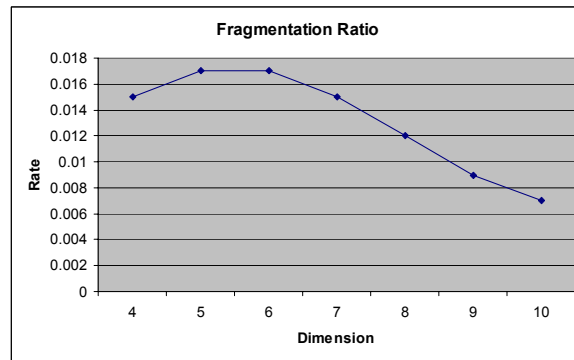


Fig. 13. Fragmentation Ratio (caused by Anonymity Hazards).

VI. CONCLUSION

In this paper, we investigated the relationship between N -divisible tokens and subcube 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 subcube allocation schemes. We developed the analysis techniques to guarantee the security properties of hypercube based N -divisible tokens. As expected, the most secure solution has 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 a subcube can be spent/allocated. The anonymity hazard checking routine is simple and reliable. Furthermore, simulation results show that existing subcube allocation schemes *binary code* and *binary gray code* are immune from anonymity hazard because of their restrictive allocation rules. Our study suggests that with proper adjustment to both the N -divisible tokens and resource allocation rules, highly secure and efficient computing resource management schemes can be developed based on one integrated framework.

APPENDIX

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.14 depicts the case when $\dim(q_2) < (n-1)$, 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

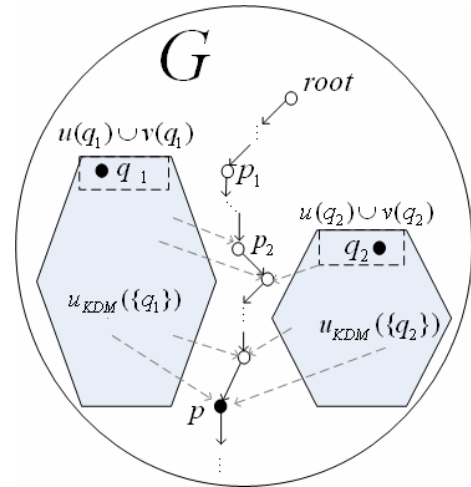


Fig. 14. Anonymity Hazard Impossible with 3 Vertices.

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.

Next, we consider the case when $\dim(q_2) = (n-1)$. In this case, it also implies that $\dim(q_1) = (n-1)$. The only possible combinations of bit pair at the i^{th} and the j^{th} positions of q_1 and q_2 are $(0, \times)$, $(1, \times)$, $(\times, 0)$, and $(\times, 1)$, because at dimension $(n-1)$ every vertex contains only one bit which is not \times . Furthermore, (\times, \times) 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 $\{(0, \times)$, $(1, \times)\}$ or $\{(\times, 0)$, $(\times, 1)\}$. 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 $\dim(q_1) = \dim(q_2) = (n-1)$, without causing double spending offense, this case is invalid.

REFERENCES

- [1] T. Okamoto and K. Ohta, "Universal electronic cash," In *Advances in Cryptology – CRYPTO' 91*, pp. 324 – 337, 1991.

- [2] H. Antwerpen, *Electronic cash*, Master's Thesis, CWI, 1990.
- [3] M. Froomkin, "The unintended consequences of e-cash," Computers, Freedom and Privacy Conference, 1997.
- [4] D. Chaum, "Blind signatures for untraceable payments," In *Advances in Cryptology – CRYPTO '82*, pp. 199 – 203, 1983.
- [5] T. Okamoto and K. Ohta, "Disposable zero-knowledge authentication and their applications to untraceable electronic cash," In *Advances in Cryptology -- CRYPTO '89*, pp. 481-96, 1990.
- [6] A. Shamir, *How to share a secret*, Communications of the ACM, vol. 22, no. 11, pp. 612-613, 1979.
- [7] 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*, pp. 331-340, 2000.
- [8] R. Radwin, "An untraceable, universally verifiable voting," *Seminar in Cryptology*, 1995.
- [9] K. B. Frikken and M. J. Atallah, "Privacy preserving electronic surveillance," In Proceedings of the *2003 ACM Workshop on Privacy in Electronic Society*, pp. 45- 52, 2003.
- [10] Y. Shen, T. C. Lam, J-C. Liu, and W. Zhao, "On the confidential auditing of distributed systems," In Proceedings of the *24th International Conference of Distributed Computing Systems*, pp. 600-607, 2004.
- [11] G-J. Ahn, B. Mohan, and S. Hong, Secure information sharing using role-based delegation, In Proceedings of *the International Conference on Information Technology: Coding and Computing*, vol. 2, pp. 810, 2004.
- [12] T. C. Lam, Cheng-Chung Tan, and Jyh-Charn Liu, "Timed Zero-Knowledge Proof (TZKP) Protocol," submitted to *IEEE Real-Time and Embedded Technology and Application Symposium 2007*. Also available at Technical Report 2006-9-1, Department of Computer Science, Texas A&M University.
- [13] T. Nakanishi, N. Haruna, and Y. Sugiyama, "Unlinkable electronic coupon protocol with anonymity controls," In Proceedings of *the 2nd International Workshop on Information Security*, pp. 37 – 46, 1999.
- [14] 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.
- [15] N. Ferguson, "Extensions of single-term off-line coins," In *Advances in Cryptology – CRYPTO '93*, pp. 292-301, 1993.
- [16] T. Eng, and T. Okamoto, "Single-term divisible electronic coins," In *Advances in Cryptology – EUROCRYPT'94*, pp/ 311-323, 1994.
- [17] T. Nakanishi and Y. Sugiyama, "Unlinkable divisible electronic cash," In Proceedings of *3rd International Workshop on Information Security*, pp. 121 – 134, 2000.
- [18] A. Chan, Y. Frankel, and Y. Tsiounis, "Easy come – easy go divisible cash," In *Advances in Cryptology – EUROCRYPT'98*, pp. 561 – 575, 1998.
- [19] D. Chaum, A. Fiat, and M. Naor, "Untraceable electronic cash," In *Advances in Cryptology – CRYPTO '88*, pp. 319–327, 1989.
- [20] S. Brands, "Untraceable off-line cash in wallets with observers," In *Advances in Cryptology – CRYPTO '93*, pp. 302-318, 1994.
- [21] C. Schnorr, *Efficient signature generation by smart cards*, Journal of Cryptology, 4(3):161-174, 1991.
- [22] N. Ferguson, "Single term off-line coins," In *Advances in Cryptology – EUROCRYPT'93*, pp. 318-328, 1993.
- [23] 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, 1990.
- [24] P-J Chuang and N-F Tzeng, *Dynamic processor allocation in hypercube computers*, ACM SIGARCH Computer Architecture News, vol. 18, no. 3, pp.40-49, 1990.
- [25] C-H Huang, J-Y Juang, "A partial compaction scheme for processor allocation in hypercube multiprocessors," *ICPP*, vol. 1, pp. 211-217, 1989.
- [26] Hamming distance between ternary numbers. URL: http://www.its.bldrdoc.gov/fs-1037/dir-017/_2529.htm
- [27] D. Chaum and T. Pedersen, "Transferred cash grows in size," In *Advances in Cryptology – EUROCRYPT'92*, pp. 390 – 407, 1993.