Detection of Dishonest Entities

Detection of Dishonest Entities

Shinsuke Tamura, Yusuke Ohashi, Shuji Tanigichi and Tatsuro Yanase
Faculty of Engineering, University of Fukui
Fukui, Japan
tamura@u-fukui.ac.jp

***Abstract*—This paper discusses mechanisms to identify dishonest users of services provided by a server in environments where identities of honest users must be kept as their secrets. An anonymous token based mechanism enables the server to identify dishonest users when dishonest events are detected while the users are receiving services, and a homomorhic anonymous token based one enables that even dishonest events can be detected only after the server completed their services and the users had left from the server. A linear equation based encryption algorithm that is used for implementing the above methods is also enhanced.**

***Keywords*—anonymous tokens, homomorphic anonymous tokens, dishonest event, dishonest entities, linear equation based encryption algorithm**

I. INTRODUCTION

Let entity S be a server that provides anonymous clients with its services. In this setting, although S can authenticate anonymous clients and collect fees for its providing services from the clients by using mechanisms for anonymous authentication [2][3] and credit card systems [4], it cannot protect itself from dishonest behaviors of clients once it had authenticated them as authorized ones. If S's service is to exhibit art objects to visitors for example, visitors that had been anonymously authenticated successfully by showing their tickets may break exhibits or steal them; however, S cannot impute its damages to anyone because visitors are anonymous. S may be able to identify liable visitors if it inquires all possible visitors of their alibis, but this solution is apparently not acceptable because privacies of even honest visitors are revealed.

This paper proposes 2 mechanisms to enable server S to identify dishonest clients in the above environments without revealing identities of honest clients. The 1st mechanism is based on anonymous tokens and enables S to identify dishonest clients when dishonest events are detected during they are receiving services, and the 2nd mechanism is based on homomorphic anonymous tokens and enables S to identify dishonest clients even dishonest events can be detected only after S had completed its services. In addition to these mechanisms, a linear equation based encryption function [4] is enhanced as the base for implementing the 2nd method.

In the remainder, it is assumed that an appropriate anonymous authentication mechanism is available. Therefore before proposing the mechanisms, firstly anonymous authentication mechanisms are discussed.

II. MECHANISMS FOR ANONYMOUS AUTHENTICATION

Anonymous authentication mechanisms enable a server to authenticate authorized clients without knowing their identities; they must satisfy the following requirements, i.e.

1) only authorized clients are successfully authenticated,
2) no one except a client itself can know the identity of the client that is being authenticated,
3) no one except a client itself can know that its past authentication requests are made by the same client,
4) qualifications of clients must be invalidated when they secede from the system managed by the server that provides the services, and
5) the server can handle clients that lose or forget their secrets necessary for being authenticated.

A mechanism based on anonymous tokens discussed in Sec. II-A satisfies the 1st, 2nd, and the 3rd requirements, and a one based on ID lists discussed in Sec. II-B satisfies the all of the above requirements.

A. Anonymous token based autentication

Server S can authenticate authorized clients by giving them tokens that are numbers with signatures of S, i.e. only authorized clients can show tokens with S's signatures. Anonymous tokens are also numbers with the signatures of S, however different from usual tokens S does not know the numbers on which it had signed. Therefore, clients can maintain their identities secret from S. Conditions that ensure anonymous tokens to work well are,

1) tokens are unique, and

2) a client can use each of its token only once.

If clients C_p and C_q can obtain the same token T, server S cannot determine that T is used in authorized ways by C_p or C_q, or T is given to unauthorized entities by them. Also, if C_p can use a single same token multiple times, it can give its tokens to other entities without any penalty. When the above conditions are ensured, anonymous tokens bring a substantial advantage to anonymous authentication mechanisms. Namely, different from the conventional password based authentication, in which non-qualified clients can be successfully authenticated as authorized ones by acquiring passwords from qualified clients, in a mechanism based on anonymous tokens, clients cannot disclose their tokens to other entities without losing their qualifications.

When client C_p discloses its token T to other client C_q, although C_q can be authenticated as an authorized one, C_p cannot use T anymore.

Client C can obtain its anonymous token from server S based on the blind signature scheme [1],[2]. Firstly, C picks its n-th token T(C, n) from the token table prepared by S anonymously. Here, the token table maintains available tokens while publicly disclosing them through e.g. a bulletin board (BB) which is readable by anyone at any time, and when T(C, n) is picked by C, S signs on it by its signing key *a*. Then because only tokens with signatures generated by key *a* are valid and S does not sign on T(C, n) repeatedly, tokens become unique. Here, signatures generated by key *a* are different from those generated by the blind signature scheme based on signing key *b*. Signing key *a* is only for making tokens unique, and even unauthorized clients can obtain signatures generated by *a*, therefore notation T(C, n) is used for representing also T(C, n) with this signature. Also because even unauthorized entities can pick tokens in the token table, S must fill the token table with tokens more than they are actually required. However, S does not need to worry about dishonest events caused by tokens picked by unauthorized entities; tokens are valid only when they have S's signatures generated by key *b*.

To obtain the signature generated by signing key *b*, client C encrypts T(C, n) by its secret key *c* to E(*c*, T(C, n)) and shows it with its (n-1)-th signed token S(*b*, T(C, n-1)) to S. Then, S signs on E(*c*, T(C, n)) by its signing key *b* to generate S(*b*, E(*c*, T(C, n))), when S(*b*, T(C, n-1)) is not used repeatedly and has the correct signature of S, and finally, C decrypts S(*b*, E(*c*, T(C, n))) to S(*b*, T(C, n)). Here, S(*b*, T(C, n)) is the signature of S on C's n-th token T(C, n), i.e. no one except S can generate S(*b*, T(C, n)) and S can confirm that C is the authorized entity by checking the consistency of S(*b*, T(C, n)), however, S cannot identify C from it because S had signed on its encrypted form. Also, because S checks duplicated uses of S(*b*, T(C, n-1)), C can use S(*b*, T(C, n)) only once. As the exception, C obtains the signature on its initial token E(*c*, T(C, 1)) by showing its authenticity while disclosing its identity, and it is assumed that signing and encryption functions S(*b*, *x*) and E(*c*, *x*) are commutative of course. Fig.1 shows the anonymous token based authentication procedure.

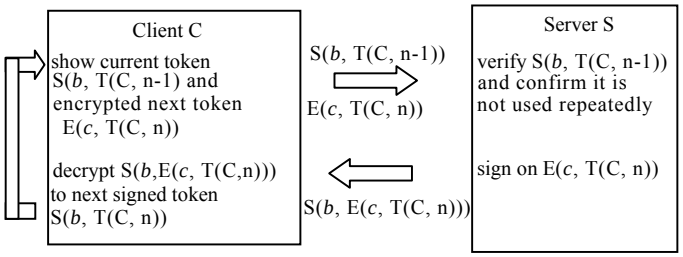

Figure 1. Anonymous token based authentication

A serious drawback of anonymous token based authentication mechanisms is that server S does not have any information about links between clients and their tokens; therefore problems arise when clients lose their qualifications, or forget or pretend to forget their tokens. Namely, firstly client C that loses its qualification still can use its token, and secondly C that obtains its new token while pretending to forget its token can use its old token. About the former problem, S can collect tokens from clients that secedes from the system while carrying out the client deregistration procedures in which clients return their unused signed tokens, however when clients secede from the system without carrying out the procedures, they still have effective tokens, and moreover they can obtain new tokens forever by refreshing them. Theoretically the both problems can be solved by invalidating all tokens that S had signed on. Here, S can find all signed tokens easily by examining the token tables. However this solution is apparently impractical because even irrelevant clients are asked to obtain their new tokens. Expiration times attached to tokens may mitigate the problem, but still clients can use invalid tokens until their expiration times, and also the clients can obtain new tokens by showing old tokens. The worse thing is that expiration times may suggest identities of token owners.

B. ID-List based Authentication

An ID-List based authentication mechanism [3] removes drawbacks of the anonymous token based one. Authentication processes proceed as shown in Fig. 2. Firstly, client C_h generates *D*, a list of IDs of randomly selecting clients that includes d_h, the ID of C_h, and sends *D* to server S as its authentication request. After receiving the request with *D*, S generates its random bit string R, and for each d_j in *D*, it finds password p_j in its database that corresponds to d_j and encrypts each p_j to $\underline{p}_j = E(k, p_j)$ by its public encryption key *k* to calculate $q_j = (\underline{p}_j \oplus R)$. Here, although entities other than C_j know public key *k* they do not know its decryption key k^{-1}, therefore they cannot know p_j, password of other client C_j, from $E(k, p_j)$. After calculating q_j, S develops *P* a password list consists of $\{d_j, q_j\}$ pairs to send it to C_h with $\underline{R} = E(k, R)$. Then, C_h that receives *P*, calculates $\underline{p}_h' = E(k, p_h)$ and $R' = (q_h \oplus \underline{p}_h')$ while finding pair $\{d_h, q_h\}$ corresponding to its ID from *P*. Here, if C_h is the authorized client and knows its password p_h, it can calculate $\underline{p}_h'$ and $\underline{R}'$ so that they coincide with $\underline{p}_h$ and R, respectively; therefore S can determine that C_h knows p_h when it receives R as the value of R'. On the other hand, S cannot identify C_h provided that S assigns same random bit string R to all clients in the ID list, because all clients in *D* know their password and can return R as the value of R'.

However, S can identify clients when it encrypts passwords of individual clients by different keys. For example, when S encrypts p_h by key R_h unique to C_h, S can easily identify C_h when it receives R_h from C_h as the value of R'. Therefore a mechanism to force S to encrypt passwords of all clients in the ID list by the same key is necessary, and this is implemented by C_h's calculation of $\underline{R}' = E(k, R')$. Namely, when $\underline{R}'$ does not coincide with $\underline{R}$, C_h can determine that S had encrypted passwords of different clients in the ID list by different keys, but when they coincide, it can believe that S had used the same key R to all clients. Because S does not know the exact client that is requesting the authentication, even when it encrypts p_h, the password of C_h, by encryption key R_h unique to C_h, it cannot send $E(k, R_h)$ as the value of $\underline{R}$ to C_h. When S uses different encryption keys for different clients, it must take risks to send $\underline{R}$ that does not coincide with $E(k, R_h)$.

Different from the anonymous token based mechanism, ID-List based one can easily handle clients that lose their

qualifications or forget their secrets, because server S checks the qualifications of clients by their passwords that are linked directly to their IDs. When C_h loses its qualification or forgets its password, the only thing that S must do is to delete d_h, ID of C_h, form the database or replace p_h, password of C_h, in the database with newly declared one.

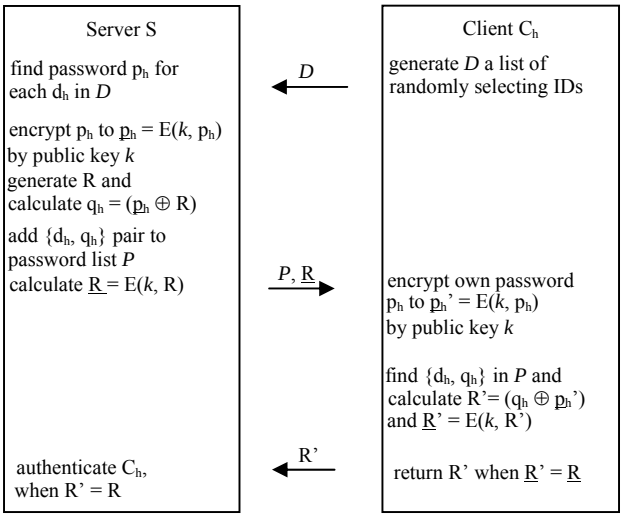

Figure 2. ID-List based authentication

Possible threats to the ID-List based mechanism that must be considered are 1) password disclosure from password list, and 2) identifications of frequent visiting clients. About the 1st threat, different from usual authentication systems, in this case, dishonest client C_j can try to estimate passwords of other client C_h infinite times without being noticed by server S, e.g. C_j includes d_h, an ID of C_h, in the ID list when it requests authentication, assumes possible passwords of C_h such as its birth date, telephone number, etc. and encrypts them by using publicly known key k to compare the result with the value in the password list. An ID-List based mechanism protects clients from this kind of password disclosures by exploiting function $G(g_h, p_h)$ that transforms passwords so that different passwords may have the same transformed forms, where p_h is the password of C_h and g_h is a secret bit string shared by S and C_h and attached to d_h at C_h's membership registration time. Namely, in Fig. 2, password p_h is firstly transformed to $G(g_h, p_h)$, then it is encrypted to $E(k, G(g_h, p_h))$ finally to be XORed by S's secret random bit string R. Therefore, S generates the password list while calculating $q_h = E(k, G(g_h, p_h)) \oplus R$. Then, client C_j that knows its own password p_j tries to steal p_h in the following way. Firstly, it calculates $E(k, G(g_j, p_j))$ to know $R = \{E(k, G(g_j, p_j)) \oplus q_j\}$, and extract $E(k, G(g_h, p_h))$ by calculating $q_h \oplus R$, then generates $E(k, G(g_{h1}, p_{h1}))$ while assuming g_{h1} and p_{h1} as the values of g_h and p_h, to compare the result $E(k, G(g_{h1}, p_{h1}))$ with $E(k, G(g_h, p_h))$. However, because multiple pairs, e.g. $\{g_{h1}, p_{h1}\}$ and $\{g_{h2}, p_{h2}\}$, may satisfy relations $E(k, G(g_h, p_h)) = E(k, G(g_{h1}, p_{h1}))$ and $E(k, G(g_h, p_h)) = E(k, G(g_{h2}, p_{h2}))$, C_j cannot determine if p_{h1} or p_{h2} is the correct password of C_h or not even $E(k, G(g_{h1}, p_{h1}))$ and $E(k, G(g_{h2}, p_{h2}))$ coincide with $E(k, G(g_h, p_h))$; and C_j's authentication request at other server S' may be rejected, and when S and S' are communicating, S can notify C_h that someone is trying to use its ID and password to suggest C_h to change its password. Here, to protect password p_h securely, $N(p_h)$ the number of passwords that are transformed into the same form $G(g_h, p_h)$ must be large, i.e. the probability that entities successfully use their steeling p_h is $1/N(p_h)$.

The 2nd problem is a one that d_h, ID of frequently visiting client C_h, appears many times in ID lists accompanying individual authentication requests, and S can identify C_h as a frequent visiting client, although it cannot identify clients that requesting individual authentications. Aliases are the solution to mitigate this problem. Namely, C_h can decrease the appearance of d_h in the ID lists even it requests authentications frequently by using different alias ID and password pairs at its individual visits to S. However, different from non-anonymous systems, in anonymous authentication systems, server S must invalidate all aliases of client C_h without knowing links between C_h and these aliases, when C_h loses its qualification, e.g. when C_h secedes from the system. Also, links between aliases of same clients must be hidden from any entity except the clients themselves; when these links are known to S, S can easily know existences of frequently visiting clients. These mechanisms can be implemented by using implicit transaction links [4].

III. MECHANISMS FOR IDENTIFYING DISHONEST ENTITIES

Anonymous authentication mechanisms discussed in the previous sections enable S to authenticate authorized clients without identifying clients themselves, however clients can behave dishonestly without any penalty once they are authenticated successfully. This section discusses mechanisms to identify clients that behave dishonestly after they were authenticated.

A. Anonymous Token

Anonymous tokens used in the anonymous token based authentication mechanism can be directly used to identify dishonest clients provided that S can detect dishonest events while the clients are receiving services. Namely in Fig. 1, when S does not give the n-th token to C until it can confirm the honest behavior of C, C can obtain its n-th token only if it is honest. Therefore although C can receive services from S at its (n-1)-visit to S by using token T(C, n-1), after that it cannot receive any service from S. Moreover, when S inquires clients of their last tokens, C can show only its used token because it had shown T(C, n-1) already at its (n-1)-th visit and did not receive its new token. On the other hand, honest clients always possess their unused tokens because they show their tokens in exchange for their new tokens. Then S can identify dishonest client C as the client that cannot show its unused token. However, S cannot know any privacy of honest clients from unused tokens that they show at the requests of S because they did not receive any service by using these tokens.

However, S cannot identify dishonest clients when dishonest events are detected after it had completed its services, i.e. S cannot decide whether C is honest or

dishonest at a time when C leaves from S. Therefore S must sign on C's next token, and C can show its unused token when S inquires of its unused token.

B. Homomorphic Anonymous Tokens

When tokens in the previous subsection are homomorphic, S can identify dishonest clients, even if it can detect dishonest events only after the clients leave from S. A homomorphic anonymous token is a one that is encrypted by a token holder while using homomorphic encryption function. Namely, token holder C encrypts its token S(b, T(C, n)) signed by server S to E(c, S(b, T(C, n))) by its encryption key c, where, E(c, x) is a homomorphic encryption function and b is a signing key of S. As same as anonymous tokens, S(b, T(C, n)) must be constructed so that no one except C can identify C from it. However, different from the anonymous token scheme, in the homomorphic anonymous token scheme, it is not necessary for C to conceal its tokens at times when it obtains signatures on them. C can obtain its tokens also while hiding its identity without concealing them. In the followings, encryption function E(c, x) is assumed to satisfy E(c, x) + E(c, y) = E(c, x+y), i.e. E(c, x) is homomorphic under addition.

S identifies dishonest client C as follows. Firstly, at anonymous client C's n-th visit to S, S asks C to show its n-th token S(b, T(C, n)) and the encrypted (n+1)-th token (i.e. homomorphic anonymous token E(c, S(b, T(C, n+1)))), that are not used before, where C obtains S(b, T(C, n)) and S(b, T(C, n+1)) from S through the other independent process and encrypts S(b, T(C, n+1)) to E(c, S(b, T(C, n+1))) by its secret encryption key c in advance. Then, C receives services from S and S memorizes triple {S_{ID}, S(b, T(C, n)), E(c, S(b, T(C, n+1)))} as the service record, where S_{ID} is the identifier of the service that S had provided. When dishonest events are detected in the service corresponding to S_{ID}, S asks all clients to decrypt E(c, S(b, T(C, n+1))), and C is identified as a dishonest one, because C that knows decryption key c^{-1} can decrypt E(c, S(b, T(C, n+1))) to S(b, T(C, n+1)), the consistent signature of S, on the other hand, the decryption result of honest client D is E(d^{-1}, E(c, S(b, T(C, n+1)))) that does not have consistent meanings. The important thing is that E(d^{-1}, E(c, S(b, T(C, n+1)))) does not include any information about the service that D had received, i.e. a homomorphic anonymous token does not reveal any privacy of honest client D.

Here, although dishonest C tries not honestly to decrypt E(c, S(b, T(C, n+1))) of course, homomorphic property of encryption function E(c, x) disables that. Because E(c, x) is additive, when S asks C to decrypt $\underline{test}$ = w_1E(c, T_{c1}) + w_2E(c, T_{c2}) + --- + w_mE(c, T_{cm}) + w_0E(c, S(b, T(C, n+1))), the decryption result must coincide with $test$ = w_1T_{c1} + w_2T_{c2} + --- + w_mT_{cm} + w_0S(b, T(C, n+1))*, if C had answered S(b, T(C, n+1))* honestly as the decrypted form of E(c, S(b, T(C, n+1))). Here, {E(c, T_{c1}), E(c, T_{c2}), ---, E(c, T_{cm})} are encrypted forms of test bit strings T_{c1}, T_{c2}, --- T_{cm} that are registered by C in advance, and w_1, w_2, ---, w_m, w_0 are random bit strings secret from C. Then, S can decide that C decrypts E(c, S(b, T(C, n+1))) dishonestly to S(b, T(C, n+1))* when the decrypted form of $\underline{test}$ does not coincide with $test$. Namely, when E(c^{-1}, E(c, S(b, T(C, n+1)))) ≠ S(b, T(C, n+1))*, C that does not know w_1, w_2, ---, w_m, w_0 cannot decrypt $\underline{test}$ to $test$. On the other hand, when S(b, T(C, n+1))* is the correct decrypted form of E(c, S(b, T(C, n+1))), C can calculate $test$ by simply decrypting $\underline{test}$ without knowing any of w_1, w_2, ---, w_m, w_0. However, the above scheme does not work correctly when C had dishonestly encrypted S(b, T(C, n+1)) from the beginning. It must be noted that S cannot use test bit strings to confirm the correct encryption of S(b, T(C, n+1)), because S must know the encrypted test bit strings {E(c, T_{c1}), E(c, T_{c2}), ---, E(c, T_{cm})} that differ from those of other clients to verify correctness of encryptions, i.e. to enable S to find the test bit strings corresponding to C, C must inform S of its identity. ITLs (Implicit Transaction Links) [4] enable S to force C also to encrypt S(b, T(C, n+1)) honestly without identifying C.

Visit counter	Serveice ID	Current token	Next token
n	S_{ID}(n)	v_nS(b, T(C, n))	v_nE(c, S(b, T(C, n+1)))

Figure 3. Used token record UTR(C, n)

To disable C to encrypt S(b, T(C, n+1)) dishonestly, S generates used token record UTR(C, n) consists of 4 items, i.e. visit counter, service identifier, current token, and next token, as shown in Fig. 3, and encrypts it to E(k_S, UTR(C, n)) by its secret encryption key k_S to be maintained by C. Here, E(k_S, x) is an additive (homomorphic) encryption function as same as E(c, x). The visit counter represents the number of visits that C had made before including the current one, service identifier S_{ID}(n) represents the service that C had received at its n-th visit to S, and current token S(b, T(C, n)) and encrypted next token E(c, S(b, T(C, n+1))) are multiplied by the token concealers v_n and v_{n+1} that are secret of S so that C cannot modify or forge pair {v_nS(b, T(C, n)), v_{n+1}E(c, S(b, T(C, n+1)))} consistently. It must be noted that, the value of the visit counter must be initialized when it exceeds the relatively small value defined in advance, to disable S to extract frequently visiting entities.

Here, pair {v_nS(b, T(C, n)), v_{n+1}E(c, S(b, T(C, n+1)))} is an ITL at C's n-th visit, and S inquires sums of encrypted used token records from individual clients periodically, e.g. at times when S collect fees for its providing services. Therefore, S can calculate the sums of current and encrypted next tokens of C by decrypting the sum of encrypted records that C had reported by exploiting the additive property of E(k_S, x), and because encryption function E(c, x) is also additive, S can calculate the sum of next tokens by asking C to decrypt the sum of encrypted next tokens. Then, based on the fact that the sums of current and next tokens are equal provided that the difference between the initial and the last tokens is compensated, S can detect dishonest token encryptions of C. Namely, although C can dishonestly encrypt S(b, T(C, n+1)) to $\underline{X}$ at its n-th visit to S, at its (n+1)-th visit, C must show token S(b, T(C, n+1)) accompanied by the consistent signature that is not encrypted to $\underline{X}$, and as the consequence, S generates C's (n+1)-th used token record UTR(n+1) as UTR(n+1) = {n+1, S_{ID}(n+1), v_{n+1}S(b, T(C, n+1)), v_{n+2}E(c, S(b, T(C, n+2)))}, however S had generated UTR(n) as UTR(n) = {n, S_{ID}(n), v_nS(b, T(C, n)), $v_{n+1}\underline{X}$} at C's n-th visit. Then, S detects inconsistency between S(b, T(C, n+1)) and $\underline{X}$ at a time when it collect fees from C for

its providing services, i.e. the sum of current tokens and that of next tokens do not coincide.

As a conclusion, client C is identified as a dishonest one through one of the following reasons, i.e. firstly C decrypts E(c, S(b, T(C, n+1))) to consistent S(b, T(C, n+1)) at a time when S detects dishonest events, or secondly, at a time when S calculates the sum of used token records of each client, C cannot decrypt the sum of encrypted next tokens so that it coincides with the sum of current tokens.

In the above, ITLs disable C to report the sum of its service records dishonestly, e.g. while modifying, forging, or deleting its records, based on the relation that the sums of current and next tokens must be same [4]. At its n-th visit to S, C cannot report n, the number of its past visits to S, dishonestly either.

IV. LINEAR EQUATION BASED ENCRYPTION FUNCTION

A. Basic Mechanism

A linear equation based encryption function E(Q, M) encrypts bit string M by representing M as a H-dimensional integer vector $\{M\} = \{m_1, m_2, ---, m_H\}$ and linearly combining its elements by using linearly independent (H x H) secret coefficient matrix Q = $\{q_{ij}\}$ to generate H-dimensional vector $\{M_*\} = \{m_{*1}, m_{*2}, ---, m_{*H}\}$ as shown in (1).

$$\begin{aligned} m_{*1} &= q_{11}m_1 + q_{12}m_2 + ---- + q_{1H}m_H \\ m_{*2} &= q_{21}m_1 + q_{22}m_2 + ---- + q_{2H}m_H \\ &---- \\ m_{*H} &= q_{H1}m_1 + q_{H2}m_2 + ---- + q_{HH}m_H \end{aligned} \quad (1)$$

Then, an entity that does not know Q cannot calculate M from $\{M_*\}$; however when Q is disclosed, anyone can calculate M from $\{M_*\}$ by solving (1). Therefore, $\{M_*\}$ is the encrypted form of M, and matrices Q and Q^{-1} are considered as encryption and decryption keys, respectively. Encryption function E(Q, M) is homomorphic under addition, i.e. when M_1 and M_2 are encrypted into E(Q, M_1) and E(Q, M_2), then sE(Q, M_1) + tE(Q, M_2) is decrypted into sM_1 + tM_2, where s and t are arbitrary integers. However this property introduces a serious drawback that it is weak against plain text attacks. When H mutually independent H-dimensional vector $\{A_{1*}\}$, $\{A_{2*}\}$, ---, $\{A_{H*}\}$ are given as encryption results of known bit strings A_1, A_2, ---, A_H, because arbitrarily given H-dimensional vector x_* is represented as $x_* = g_1A_{1*} + g_2A_{2*} + ---- + g_HA_{H*}$, x_* can be easily decrypted into $x = g_1A_1 + g_2A_2 + ---- + g_HA_H$ even coefficient matrix Q is not known.

B. Protecting Encryption Function from Plain text Attacks

The drawback of the linear equation based encryption finctions can be excluded by the following 3 ways, i.e.

1) by inserting secret random dummy elements at random positions in the encrypted vector,
2) by adding secret random terms to (1), and
3) by representing the value of each element of the encrypted vector as the sum of values randomly split into multiple elements.

The 1st method mixes vector $\{m_{*1}, m_{*2}---, m_{*H}\}$, which is calculated from M according to (1), with secret random vector $\{w_{*1}, w_{*2}, ---, w_{*T}\}$ while shuffling the elements of both vectors to generate a single (H+T)-dimensional vector; therefore the encryption result becomes to $\{M_*'\} = \{w_{*2}, w_{*1}, m_{*3}\ w_{*3}, m_{*1}---, w_{*5}\}$ for an example. As the consequence, to decrypt $\{M_*'\}$ into M, positions where $m_{*1}, m_{*2}---, m_{*H}$ are located in $\{M_*'\}$ must be determined. However, there are still linear relationships among encrypted forms of known bit strings when dummy elements are removed, and it is not so difficult to identify the positions where elements $\{m_{*1}, m_{*2}---, m_{*H}\}$ are allocated.

To make the linear relationships among encrypted forms difficult to be identified, the 2nd method adds G secret random bit strings r_1, ---, r_G to $\{m_1, m_2, ---, m_H\}$. Therefore, M is represented as (H+G)-dimensional vector $\underline{M} = \{m_1, m_2, ---, m_H, r_1, r_2, ---, r_G\}$, and key Q is extended to (H+G) x (H+G)-matrix $\underline{Q}$, i.e. M is encrypted to $\underline{M}_* = \{\underline{m}_{*1}, \underline{m}_{*2}, ---, \underline{m}_{*H+G}\}$ by using secret coefficient matrix $\underline{Q}$, and $\underline{M}_*$ is decrypted into M by solving the linear equation and deleting random bit strings r_1, r_2, ---, r_G. The important thing is that r_1, r_2, ---, r_G are kept as secrets of the entity that encrypts M and they are changed at every encryption. Therefore, although there are still linear relationships among encrypted forms of known bit strings, it is extremely difficult to use them for estimating the encryption key.

However, it must be noted that the 2nd method cannot protect encrypted forms without the 1st method. When (H+G) linearly independent encrypted forms of known bit strings $\{M_1^*, M_2^*, ---, M_{H+G}^*\}$ are fortunately obtained, any entity can decrypt arbitrary encrypted form M_* by decomposing it into a linear combination of $\{M_1^*, M_2^*, ---, M_{H+G}^*\}$. When M_* is decomposed into $a_1M_1^* + a_2M_2^*, --- + a_{H+G}M_{H+G}^*$, although it is not possible to identify random secret numbers r_1, r_2, ---, r_G that are used for calculating M_*, M itself can be reconstructed as $M = a_1M_1 + a_2M_2, --- + a_{H+G}M_{H+G}$, provided that M_j^* is the encrypted form of M_j for each j.

In the 3rd method, the value of each element m_{*j} of the encrypted vector is randomly split into a set of multiple elements $\{m_{*j1}, m_{*j2}---, m_{*jp}\}$ so that relation $m_{*j} = m_{*j1} + m_{*j2} + --- + m_{*jp}$ is satisfied, i.e. single element m_{*j} is represented as a set of elements $\{m_{*j1}, m_{*j2}---, m_{*jp}\}$. Therefore, linear relationships between encrypted forms of known bit strings totally disappear, and plain text attacks become impossible. Because linear relationships among encrypted forms are hidden, all possibilities that $\{m_{*1}, m_{*2}---, m_{*H}\}$ are allocated in an (H+T)-dimensional vector must be examined to calculate the coefficient matrix, i.e. ${}_{(H+T)}P_H$ number of possibilities must be examined. When H and T are set to 50, ${}_{(H+T)}P_H$ becomes ${}_{100}P_{50} > 2^{500}$, and this can be increased by adding more dummy elements. Also for individual possible arrangements, extracting linear relationships among encryption results of known bit strings is extremely difficult because of the 2nd and the 3rd methods. When the coefficient matrix is given, LU-decomposition method [5], for example, solves linear equations with sufficient performance in terms of both computation speed and accuracy even when the dimensions of coefficient matrices are more than 1,000.

C. *Protecting Encryption Function form Forgeries*

Although the above methods protect the encryption function from plain text attacks, it is still easy to generate consistent encrypted forms without knowing the key. By linearly combining encrypted forms of known mutually independent bit strings, encrypted forms of arbitrary bit strings can be generated, and entities can behave dishonestly while modifying or forging encrypted forms in unauthorized ways, as same as in cases where public key encryption functions are used. The following mechanism disables entities to modify or forge encrypted forms of bit strings in unauthorized ways.

The basic idea is to attach a check code (CC) and a check value (CV) to bit string M to be encrypted as shown in Fig. 4. Here, values of the CC and the CV are determined as the secret of entity P that encrypts M, and P changes their values randomly in every encryption so that C and V, the values of the CC and the CV, satisfy $V = f(C)$. Therefore, when E(Q, M), the encrypted form of M, is generated in unauthorized ways, P can detect that by checking if $V = f(C)$ is satisfied or not. For entities that do not know C and V, it is difficult to generate E(Q, M) while satisfying $V = f(C)$ even if they know function $f(x)$.

Here, to protect encrypted forms, $f(C)$ must be a nonlinear function. If it is a linear function, e.g. $V = sC + t$ (s and t are constant values), it is not so difficult to generate consistent encrypted forms of given bit strings without knowing the encryption key by lineally combining already known encrypted forms. On the other hand, it is desirable that $f(C)$ is linear to exploit advantages of the additive property. When it is linear, the consistency of a set of n encrypted forms can be confirmed by checking the sum of the encrypted forms in the set without checking individual encrypted forms, and more importantly without knowing individual data.

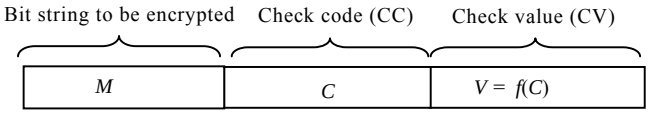

Figure 4. A check code and a check value

The CC and CV shown in Fig. 5 satisfy both of the above requirements. In the figure, CC is configured as a set of Z integers $\{CC_1, CC_2, ---, CC_Z\}$ with values 0 or 1, and P defines their values randomly so that only one of CC_j has 1 as its value. The value of CV is defined according to CC_j to which value 1 is assigned. Namely, a set of values V_1, V_2, ---, V_Z are corresponded to CC_1, CC_2, ---, CC_Z in advance, and the CV value is calculated as V_j when CC_j is 1. Then the relation between values of the CC and the CV is not linear anymore, and entities cannot generate consistent encrypted forms by combining those of already known bit strings, e.g. when an entity generates $pE(Q, M_1) + qE(Q, M_3)$, its $CC = \{CC_1, CC_2, ---, CC_Z\}$ may have multiple nonzero elements. On the other hand, P can convince itself that a set of given encrypted forms are consistent ones without checking values of the CC and the CV of individual forms. When the sum of the encrypted forms is decrypted, P can extract T_j, the sum of CC_j values for each j, and W, the sum of CV values, and all encrypted forms in the set are consistent when $W = T_1V_1 + T_2V_2 + --- + T_zV_z$ is satisfied.

By using the additive property, linear equation based encryption algorithms can be made also verifiable as shown in Sec. III-B, namely entity can confirm the correctness of encryptions while using test bit strings without knowing either of the encryption or the decryption keys.

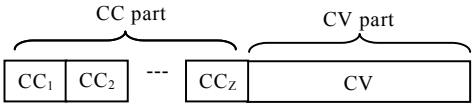

Figure 5. Configuration of CC and CV

REFERENCES

[1] D. Chaum, "Security without identification: Transaction systems to make gig brother obsolete," Communications of ACM, Vol.28, No.10, 1985, pp.1030-1044.

[2] R. Shigetomi, et.al., "Refreshable Tokens and Its Applications to Anonymous Loans," SCIS 2003, 2003.

[3] S. Tamura, et al., "Information sharing among untrustworthy entities," IEEJ Trans. EIS, Vol.125, No.11, 2005, pp.1767-1772.

[4] S. Tamura, et al., "A mechanism for anonymous credit card systems," IEEJ Trans. EIS, Vol. 127, No. 1, 2007, pp. 81-87.