Information Security and Computer Fraud, 2014, Vol. 2, No. 2, 21-27
Available online at http://pubs.sciepub.com/iscf/2/2/1

DOI:10.12691/iscf-2-2-1

A Scheme for Delegating Program Executions without Disclosing Secret Values

Shinsuke Tamura*, Shuji Taniguchi

School of Engineering, University of Fukui, Fukui, Japan
*Corresponding author: tamura@dance.plala.or.jp

Received August 05, 2014; Revised August 15, 2014; Accepted August 19, 2014

Abstract A scheme that enables entities to delegate accomplishments of their secret tasks to others under complicated conditions is proposed. The proposed scheme exploits multidimensional array based encryption functions, and different from schemes that exploit public key based fully homomorphic encryption functions, it can handle real numbers in the same way as integers. Also, it enables entities to define complicated calculation algorithms as computer programs that are easy to develop and efficient to execute. In addition, together with data encryption, data redundancy and test data insertion principles it disables relevant entities to accomplish tasks dishonestly.

Keywords: *privacy, multidimensional array based encryption functions, order preserving encryption functions, tamper resistant programs, anonymous signature*

Cite This Article: Shinsuke Tamura, and Shuji Taniguchi, "A Scheme for Delegating Program Executions without Disclosing Secret Values." *Information Security and Computer Fraud*, vol. 2, no. 2 (2014): 21-27. doi: 10.12691/iscf-2-2-1.

1. Introduction

Advances in information and communication technologies had been making every kind of activities in our society highly convenient and efficient. For example, cloud computing systems enable companies to outsource even their core tasks and to quickly launch new businesses without maintaining their own permanent computation resources that are expensive and require long development time. However, these companies must be aware that they are facing serious threats at the same time. Namely, cloud computing resources can easily gather secrets of companies that outsource their task accomplishments. Therefore, people cannot reap full benefits from advancing information and communication technologies, e.g. companies in the above still must accomplish their important tasks by themselves to make their competitiveness sustainable.

Theoretically, fully homomorphic encryption functions can remove the above inconveniences. Namely, to ask cloud computing server S to calculate function $f(X_1, X_2, ---, X_N)$ of values $X_1, X_2, ---, X_N$, client P encrypts $X_1, X_2, ---, X_N$ to $E(k, X_1), E(k, X_2), ---, E(k, X_N)$ by encryption key k of fully homomorphic encryption function $E(k, x)$, and S calculates $f(E(k, X_1), E(k, X_2), ---, E(k, X_N))$. Then because $E(k, x)$ is fully homomorphic, P can reconstruct $f(X_1, X_2, ---, X_N)$ by decrypting $f(E(k, X_1), E(k, X_2), ---, E(k, X_N))$, on the other hand, S that does not know decryption key k^{-1} cannot know any of $X_1, X_2, ---, X_N$ or $f(X_1, X_2, ---, X_N)$.

However, although long time had passed since schemes for delegating function calculations to others without disclosing relevant data were proposed [2,3,4], many schemes including most recent fully homomorphic encryption function based ones [7,8] are still impractical; they require cumbersome operations for handling real numbers that appear in most business and engineering applications. Also, client P must define calculation algorithms as complicated logic circuits instead of computer programs because server S cannot determine whether conditions about jumps and loops in the programs are satisfied or not when relevant values are encrypted.

To enable entities to delegate their secret task executions under complicated conditions, this paper proposes an MA-based scheme that exploits multidimensional array based (MA-based) encryption functions. MA-based encryption function $E(k, x)$ is both additive and multiplicative, i.e. relations $E(k, x)+E(k, y) = E(k, x+y)$ and $E(k, x)E(k, y) = E(k, xy)$ hold, and it can handle real numbers as same as integers. In addition, MA-based encryption functions can be made order preserving, in other words, entities can determine whether relation $x \leq y$ holds or not even from encryption forms $E(k, x)$ and $E(k, y)$. Therefore, by the proposed scheme server S can easily calculate various functions of integers and real numbers, e.g. polynomials and maximums/minimums, etc. without knowing their values, and at the same time, client P can communicate complicated calculation algorithms to server S by computer programs that are much easier to develop and much more efficient to execute when compared with logic circuits.

About the integrity of calculation results, several schemes to protect programs from dishonest executions had been proposed, e.g. in one of them client P asks multiple servers to carry out same calculations and detects

dishonesties when these servers respond differently, and in another scheme P makes programs complicated so that other entities cannot analyze the programs to modify them consistently [5]. However in the former scheme, P must ask the calculations to many servers if possibilities in which different servers conspire are considered, and in the latter scheme still other entities can easily replace programs with ones that they had developed by themselves even if the original programs are difficult to analyze. Although schemes based on fully homomorphic encryption functions are also proposed to maintain the integrity of calculation results [8], they are not practical enough as same as the base schemes for delegating function calculations. Therefore in this paper, data encryption, data redundancy and test data insertion principles are combined with the MA-based scheme. Namely, client P provides the data for server S while encrypting them by MA-based encryption functions, duplicating same data and inserting test data about which calculation results are known in advance. Here, MA-based encryption functions are probabilistic, i.e. even same data are encrypted to different values, therefore S cannot identify duplicated data or test data among ones given from P. This means P can detect dishonest calculations of S by the facts that S returns different values or wrong values for same data or test data respectively.

2. Overall Behaviors of the Scheme

Figure 1 shows the configuration of a system that exploits the proposed scheme. It consists of clients and servers, and client P shows values of its data items while encrypting them by its secret encryption key to ask servers to accomplish its secret tasks. On the other hand, server S maintains programs that correspond to algorithms for accomplishing the tasks, where these programs are developed by P itself or other entities including servers.

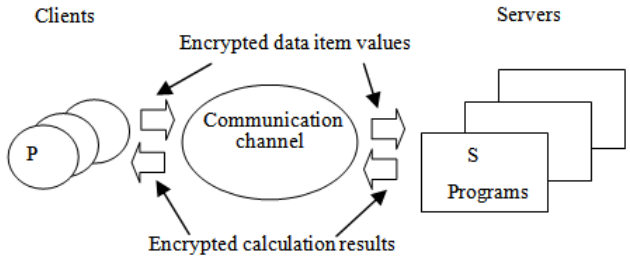

Figure 1. Configuration of a system that exploits an MA-based scheme

Important things are encryption functions are MA-based that are both additive and multiplicative and as will be discussed later MA-based encryption functions can be made order preserving. Therefore, when the programs calculate functions of encrypted values given by P, P can obtain plain function values by decrypting the calculation results, provided that the functions are simple ones, e.g. polynomials and maximums/minimums. Also, S can evaluate conditional statements in the programs without decrypting values encrypted by P. Then, P can ask S to calculate functions of its secret data with minimum interactions between S even if calculation conditions are complicated. Here, P can also conceal its identities if anonymous communication channels [1,6,9] are used, and S can collect fees from P even if P is anonymous by using anonymous credit card systems for example [10].

But without additional mechanisms servers and clients can behave maliciously, e.g. servers may calculate functions incorrectly or replace the programs with wrong ones, and on the other hand, clients may intentionally provide servers with wrong data. To protect clients and servers from malicious behaviors of others P configures its encryption data as shown in Fig. 2. In the figure, $E(k, x; p_x)$ is an MA-based encryption function, p_x is a random factor to make $E(k, x)$ probabilistic, k is a secret key of P, and P encrypts value X_j of its j-th data item to $E(k, X_j; p_j)$. But it does not inform S of $E(k, X_j; p_j)$ individually, instead, P gives S a sequence of encrypted values $\{E(k, X_1; p_1), E(k, X_2; p_2), ---, E(k, X_j; p_j), ---, E(k, X_j; p_{j*}), ---, E(k, T_h; p_{Th}), ---, E(k, X_N; p_N)\}$. Here, $E(k, X_j; p_j)$ and $E(k, X_j; p_{j*})$ take different values despite that they are encryption forms of same X_j because $E(k, x; p_x)$ is probabilistic, and P knows the calculation result corresponding to test data T_h in advance. Then, S cannot identify same data item or test data item values in the input sequence, and as a result, P can detect S's dishonesties as facts that S calculates different values or incorrect values for same data items or test data items respectively.

About dishonesties of client P, $\{S(d_1 \| d_2, T_P^{R+1} K_w C_w^R)^W, h(E(k, x; p))^R\}$ in Figure 2 is an anonymous signature of client P on encryption form $E(k, x; p)$, i.e. only P can generate it consistently but no one except P can identify the entity that had generated it. Then, server S can convince anyone that it had certainly received encryption form $E(k, x; p)$ from a legitimate entity, at the same time provided that it is honest, P can conceal its identity from S.

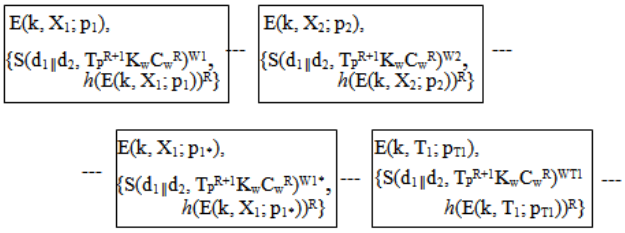

Figure 2. A data sequence client P provides for server S

3. MA-based Encryption Functions

Because fully homomorphic encryption functions are asymmetric public key based [7,8], they are not convenient enough especially for handling real numbers that appear frequently in many business and engineering applications. But it must be noted that the entity which encrypts and decrypts data item values secrets of client P in Figure 1 is P itself, therefore encryption function $E(k, x; p_x)$ in the figure does not need to be public key based. Then, implementation of both additive and multiplicative encryption function $E(k, x; p_x)$ that can handle real numbers as same as integers becomes easy. It is also straightforward to make $E(k, x; p_x)$ order preserving,

In the following notation $E(k, x)$ is used instead of $E(k, x; p_x)$ when confusions can be avoided.

3.1. 1-dimensional Representation

Multidimensional array based (MA-based) encryption function $E(k, x)$ that is both additive and multiplicative

can be constructed as below [10]. Let E(k, x) be a function that transforms real number x to vector {x(1), x(2), ---, x(z)} so that relation a_1x(1)+a_2x(2)+ --- +a_zx(z) = x holds for a set of real number coefficients {a_1, a_2, --- a_z} that are secrets of client P. Then, vector E(k, x) is an encryption form of x; although P that knows coefficients {a_1, --- a_z} can easily calculate x from {x(1), ---, x(z)}, it is not straightforward to know x from them without knowing the coefficients. Here, it must be noted that some of coefficients a_{h1}, a_{h2}, --- a_{hm} may have value 0, and this means x(h_1), x(h_2), ---, x(h_m) are dummy elements that can have any values. Also, E(k, x) = E(k, x; p_x) is probabilistic, i.e. 2-vectors {x(1), ---, x(z)} and {y(1), ---, y(z)} are decrypted to same real number x even if {x(1), ---, x(z)} ≠ {y(1), ---, y(z)}, provided that relations a_1x(1)+ --- +a_zx(z) = x and a_1y(1)+ --- +a_zy(z) = x hold.

About secureness of encryption function E(k, x), apparently it is weak against plain text attacks. But client P can protect its secrets from others because data item values are encrypted and decrypted by P itself, i.e. P never discloses secret values in their plain forms. In detail, for entities that do not know values of coefficients {a_1, a_2, --- a_z} or plain and encryption form pairs, every real number has equal possibility to be a decryption form of vector {x(1), ---, x(z)}.

3.2. Multidimensional Representation

E(k, x) in the above is apparently additive, i.e. relation E(k, x)+E(k, y) = E(k, x+y) holds. E(k, x) can be made both additive and multiplicative when it is extended from vector {x(1), ---, x(z)} to n-dimensional array {x(i_1, i_2, ---, i_n)} as follow.

Firstly, 1-dimensional encryption form E_1(k, x) = {x(1), x(2), ---, x(z)} can be extended to 2-dimensional array E_2(k, x) = <{x(1, 1), x(1, 2), ---, x(1, z)}, {x(2, 1), x(2, 2), ---, x(2, z)}, ---, {x(z, 1), x(z, 2), ---, x(z, z)}> by defining values {x(j, 1), x(j, 2), ---, x(j, z)} so that relation x(j) = a_1x(j, 1)+a_2x(j, 2)+ --- +a_zx(j, z) holds for each element x(j) (in addition to E(k, x) notation E_n(k, x) is also used for representing an n-dimensional encryption form of x). Namely, E_2(k, x) is decrypted to a_1{a_1x(1, 1)+a_2x(1, 2)+ --- + a_zx(1, z)} + a_2{a_1x(2, 1)+a_2x(2, 2)+ --- +a_zx(2, z)} + --- + a_z{a_1x(z, 1)+a_2x(z, 2)+ --- +a_zx(z, z)} = a_1x(1)+a_2x(2)+ --- +a_zx(z) = x. Where x(j, h) can have any value when coefficient a_h is 0, i.e. x(j, h) is a dummy element as in the previous subsection. In the same way n-dimensional extension of E(k, x) can be defined as an n-dimensional array E_n(k, x) = {x(i_1, i_2, ---, i_n)}, i.e. it is decrypted to x according to equation (1).

$$x = \sum_{i_1=1}^{i_1=z}\left\{\sum_{i_2=1}^{i_2=z}\left\{\cdots\sum_{i_n=1}^{i_n=z}\left\{a_{i1}a_{i2}\cdots a_{in}\mathrm{x}\left(i_1,i_2,\cdots,_n\right)\right\}\cdots\right\}\right\} \quad (1)$$

Now, addition and multiplication can be defined in a set of encryption forms. Firstly, product of E_r(k, x) = {x(i_1, ---, i_r)} and E_n(k, y) = {y(q_1, ---, q_n)} is (r+n)-dimensional array {z(i_1, ---, i_r, q_1, ---, q_n)} = {x(i_1, ---, i_r)y(q_1, ---, q_n)}. For example, product of E_1(k, x) = {x(1), ---, x(z)} and E_1(k, y) = {y(1), ---, y(z)} is 2-dimensional array <{x(1)y(1), x(1)y(2), ---, x(1)y(z)}, {x(2)y(1), x(2)y(2), ---, x(2)y(z)}, ---, {x(z)y(1), x(z)y(2), ---, x(z)y(z)}>, where it is apparent that product {x(i)y(q)} is decrypted to xy. About addition, r-dimensional encryption form E_r(k, x) = {x(i_1, ---, i_r)} can be transformed to an n-dimensional one (n > r) by multiplying (n-r)-dimensional encryption form E_{n-r}(k, 1), i.e. product E_r(k, x)E_{n-r}(k, 1) = {x$_*$(i_1, ---, i_n)} is an n-dimensional array and it is decrypted to x1 = x. Then, because {x$_*$(i_1, ---, i_n)+y(i_1, ---, i_n)} is decrypted to x+y, MA-based encryption functions become both additive and multiplicative provided that addition of r-dimensional and n-dimensional encryption forms E_r(k, x) = {x(i_1, ---, i_r)} and E_n(k, y) = {y(i_1, ---, i_n)} is defined as {x$_*$(i_1, ---, i_n)+y(i_1, ---, i_n)}. Here, it is apparent that x and y in the above can have real number values as well as integer values. Also, because disclosed plain and encryption form pair is only {1, E_{n-r}(k, 1)}, still plain text attacks are difficult enough.

While exploiting the above additive and multiplicative property client P can ask server S to calculate polynomial functions of {X_1, X_2, ---, X_N} without disclosing values{X_1, X_2, ---, X_N}. For example, $f(X_1, X_2, X_3) = X_1^2X_2 + X_2^3X_3$ can be calculated as follows. Firstly while using secret encryption key k, P encrypts X_1, X_2 and X_3 to E_1(k, X_1), E_1(k, X_2) and E_1(k, X_3) as 1-dimensional arrays and informs S of them together with encryption form E_1(k, 1) that it prepared in advance. After that S calculates E_1(k, X_1)$^2E_1$(k, X_2) = E_3(k, $X_1^2X_2$) and E_1(k, X_2)$^3E_1$(k, X_3) = E_4(k, $X_2^3X_3$) as 3-dimensional and 4-dimensional arrays respectively, and transforms E_3(k, $X_1^2X_2$) to 4-dimensional array E_3(k, $X_1^2X_2$)E_1(k, 1) = E_4(k, $X_1^2X_2$) to add it to E_4(k, $X_2^3X_3$). Then, P can calculate $X_1^2X_2 + X_2^3X_3$ by decrypting E_4(k, $X_1^2X_2$)+E_4(k, $X_2^3X_3$) by its secret decryption key k^{-1}.

3.3. Preserving Orders

Provided that order is defined between x and y and between their encryption forms E(k, x) and E(k, y), encryption function E(k, x) is called order preserving when relation E(k, x) ≤ E(k, y) implies $x \le y$. MA-based encryption function E(k, x) can be made order preserving as follows.

For 1-dimensional encryption forms, client P defines secret positive constant real number λ and element position p in encryption forms (P may reveal position p to others), and to encrypt values x and y assigns values λx and λy to elements x(p) and y(p) in vectors E_1(k, x) = {x(1), ---, x(z)} and E_1(k, y) = {y(1), ---, y(z)} respectively. Where, other elements {x(1), ---, x(p-1), x(p+1), ---, x(z)} and {y(1), ---, y(p-1), y(p+1), ---, y(z)} are defined so that a_1x(1)+ --- +a_{p-1}x(p-1)+a_{p+1}x(p+1)+ --- +a_zx(z) = (1-$a_p\lambda$)x and a_1y(1)+ --- +a_{p-1}y(p-1)+a_{p+1}y(p+1)+ --- +a_zy(z) = (1-$a_p\lambda$)y hold when coefficient $a_p \neq 0$ of course. Then, provided that the order between encryption forms E_1(k, x) and E_1(k, y) is defined in a way that relation x(p) ≤ y(p) implies E_1(k, x) ≤ E_1(k, y), encryption function E_1(k, x) becomes order preserving, and as a consequence, if client P informs server S of element position p in vectors, S can determine whether $x \le y$ or not by comparing x(p) and y(p). But S cannot know x or y from x(p) or y(p) because λ is known only to P. Here, P can assign different values to p and λ for different encryptions if required, and in the reminder, x(p) and y(p), i.e. the p-th elements of E_1(k, x) and E_1(k, y), are called order-indicators of E_1(k, x) and E_1(k, y).

In a case where n-dimensional encryption forms E_n(k, x) = {x(i_1, ---, i_n)} and E_n(k, y) = {y(i_1, ---, i_n)} are compared, firstly it must be noted that both E_n(k, x) and E_n(k, y) are

generated as products of the lower dimensional encryption forms, therefore if all 1-dimensional encryption forms that constitute the products are constructed as in the above, $(p, p, ---, p)_n$-th elements of $E_n(k, x)$ and $E_n(k, y)$ have values $\lambda^n x$ and $\lambda^n y$ respectively, where $(p, p, ---, p)_n$ represents element positions in n-dimensional arrays. Then, server S can decide whether $x \leq y$ or not by comparing $(p, p, ---, p)_n$-th elements of $E_n(k, x)$ and $E_n(k, y)$ without decrypting them, i.e. $E_n(k, x)$ and $E_n(k, y)$ preserve the order between x and y, and $(p, p, ---, p)_n$-th elements of encryption forms constitute order indicators.

But in a case when n-dimensional and r-dimensional encryption forms $E_n(k, x) = \{x(i_1, ---, i_n)\}$ and $E_r(k, u) = \{u(i_1, ---, i_r)\}$ are compared (n > r), because $\lambda^n \neq \lambda^r$, their order indicators $x(p, ---, p)_n = \lambda^n x$ and $u(p, ---, p)_r = \lambda^r u$ may not preserve the order. To compare them, $E_n(k, x)$ and $E_r(k, u)$ must be transformed to same dimensional encryption forms. Although additional considerations are required to protect the scheme from plain text attacks as will be discussed later, this can be achieved by multiplying $E_n(k, x)$ and $E_r(k, u)$ by appropriate dimensional encryption forms which are prepared by client P in advance. For example, provided that $E_{m1}(k, v_1)$, $E_{m2}(k, 1/v_1)$, $E_{m3}(k, v_2)$, $E_{m4}(k, v_3)$ and $E_{m5}(k, 1/v_2v_3)$ are m_1, m_2, m_3, m_4 and m_5 dimensional encryption forms and relation $n+m_1+m_2 = r+m_3+m_4+m_5 = m$ holds, $E_n(k, x)E_{m1}(k, v_1)E_{m2}(k, 1/v_1) = E_m(k, x)$ and $E_r(k, u)E_{m3}(k, v_2)E_{m4}(k, v_3)E_{m5}(k, 1/v_2v_3) = E_m(k, u)$ are same dimensional encryption forms of x and u, and their $(p, ---, p)_m$-th elements have values $(\lambda^n x)(\lambda^{m1}v_1)(\lambda^{m2}/v_1) = \lambda^m x$ and $(\lambda^r u)(\lambda^{m3}v_2)(\lambda^{m4}v_3)(\lambda^{m5}/v_2v_3) = \lambda^m u$, respectively. This means S that know element position p can determine whether $x \leq y$ or not without decrypting either $E_n(k, x)$ or $E_r(k, u)$.

Because $E(k, x)$ is a MA-based encryption function, it is also additive and multiplicative. In addition, provided that $E_n(k, x) = \{x(i_1, ---, i_n)\}$ and $E_n(k, y) = \{y(i_1, ---, i_n)\}$ are n-dimensional encryption forms of x and y, when values $\lambda^n x$ and $\lambda^n y$ are assigned to their $(p, ---, p)_n$-th elements, the $(p, ---, p)_n$-th element of $E_n(k, x)+E_n(k, y) = E_n(k, x+y)$ has value $\lambda^n(x+y)$, and this means the order is preserved even when encryption forms are added. In the same way, provided that $(p, ---, p)_n$-th and $(p, ---, p)_r$-th elements of n-dimensional and r-dimensional encryption forms $E_n(k, x) = \{x(i_1, ---, i_n)\}$ and $E_r(k, y) = \{y(i_1, ---, i_r)\}$ have values $\lambda^n x$ and $\lambda^r y$ respectively, the $(p, ---, p)_{n+r}$-th element of $E_n(k, x)E_r(k, y) = E_{n+r}(k, xy)$ has value $\lambda^{n+r}xy$, i.e. the order is preserved also when encryption forms are multiplied.

A problem is that order indicators include clues to know decrypted values of encryption forms. Namely, entities can know values λx and λy from $E(k, x)$ and $E(k, y)$ if they know the position of order indicators which makes plain text attacks easier although it is not straightforward to calculate x and y from λx and λy. A fortunate thing is theoretically anyone cannot calculate x from $E(k, x)$ without knowing decryption key k^{-1} even if it gathers numbers of encryption form and order indicator pairs.

In detail, although any entity can obtain relations $\lambda\{a_1x_1(1)+ --- +a_zx_1(z)\} = \lambda x_1 = x_1(p)$, $\lambda\{a_1x_2(1)+ --- +a_zx_2(z)\} = \lambda x_2 = x_2(p)$, ---, $\lambda\{a_1x_w(1)+ --- +a_zx_w(z) = \lambda x_w = x_w(p)$ from encryption forms $E(k, x_1)$, $E(k, x_2)$, ---, $E(k, x_w)$ that include order indicators $x_1(p)$, $x_2(p)$, ---, $x_w(p)$, and the entity may be able to calculate $\{\lambda a_1, \lambda a_2, ---, \lambda a_z\}$ as $\{\delta_1, \delta_2, ---, \delta_z\}$ by solving simultaneous linear equations, it cannot know coefficients $\{a_1, a_2, ---, a_z\}$ or λ because any set of real numbers $\{a_{1*}, a_{2*}, ---, a_{z*}, \lambda_*\}$ that satisfies relations $\lambda_* a_{1*} = \delta_1$, $\lambda_* a_{2*} = \delta_2$, ---, $\lambda_* a_{z*} = \delta_z$ is consistent. But, values $\{\delta_1, \delta_2, ---, \delta_z\}$ enable entities to know ratios among coefficients which are good clues to decrypt encryption forms, e.g. ratios among $\{a_1, a_2, ---, a_z\}$ and 1-plain and encryption form pair enable any entity to decrypt all encryption forms. Therefore, client P must restrict values that are encrypted by order preserving encryption functions to selected ones as will be discussed in the next section. Also, P must conceal correspondences between order preserving encryption forms of same values in different dimensions. When n-dimensional and r-dimensional order preserving encryption forms $E_n(k, x)$ and $E_r(k, x)$ of same value x are available (n > r), it is easy to calculate secret real constant λ. Because order indicators of $E_n(k, x)$ and $E_r(k, x)$ take values $\lambda^n x$ and $\lambda^r x$ respectively, it is trivial to calculate λ from relation $\lambda^{n-r} = \lambda^n x/\lambda^r x$.

4. Describing Calculation Algorithms

As in the previous section, client P in Fig. 1 can successfully delegate calculations of simple functions (e.g. polynomials) of integers and real numbers $X_1, X_2, ---, X_N$ to server S without disclosing their values. P can also calculate maximums and minimums of its secret values if encryption function $E(k, x)$ is order preserving, and when encryption forms $E(k, 1/X_1)$, ---, $E(k, 1/X_N)$ are prepared in addition to $E(k, X_1)$, ---, $E(k, X_N)$, calculations of wider variety of functions become possible. Here, P protects encryption forms from plain text attacks by not disclosing plain and encryption form pairs except specific ones and by limiting encryption forms with order preserving features to selected ones.

But even if individual functions are simple, usually S is asked to calculate numbers of functions for many data in complicated and varying conditions, and P must inform S of these conditions with minimum interactions between S. Namely, if P and S need to frequently interact with each other to communicate calculation conditions of individual functions, benefits of the outsourcing totally vanish, as a result P may calculate functions by itself. To reduce interactions between P and S, in schemes based on public key fully homomorphic encryption functions, P communicates calculation algorithms by logic circuits, but logic circuits are complicated and cumbersome to develop and not efficient enough to execute. The proposed scheme makes calculation algorithms easy to develop and efficient to execute by describing them as usual computer programs instead of logic circuits, as a result, in the proposed scheme complicated calculation conditions are represented as combinations of conditional jumps and loops. Then, to execute the programs S must be able to determine whether conditions are satisfied or not based on encrypted values.

Order preserving MA-based encryption function $E(k, x)$ enables S to execute conditional statements based on encrypted values, if all values included in conditions are numerical ones. Namely, if P encrypts X and Y to $E(k, X)$ and $E(k, Y)$, from them S can not only calculate polynomial functions of X and Y but also can determine whether relation $X \leq Y$ holds or not without knowing X or Y. Here, even if computer programs are developed by P

itself and P does not disclose plain forms of its secret values, S can know positions of order indicators by analyzing the programs, i.e. conditional statements directly compare order indicators of encryption forms. Also, to add or compare n-dimensional and r-dimensional encryption forms $E_n(k, X)$ and $E_r(k, Y)$ ($n \neq r$), computer programs generate same dimensional arrays $E_m(k, X)$ and $E_m(k, Y)$, and this means S can obtain at least one different dimensional order preserving encryption form pair $\{E_n(k, X), E_m(k, X)\}$ or $\{E_r(k, Y), E_m(k, Y)\}$ which enables S to calculate X and Y, as discussed in the previous section. Therefore to protect its secrets from plain text attacks, P must restrict variables that are encrypted by order preserving encryption functions only to the ones that are included in conditional statements (in the remaining, variables that are encrypted by order preserving encryption functions are called order preserving variables). In addition, to disable S to decrypt encryption forms from pair $\{E_n(k, X), E_m(k, X)\}$ or $\{E_r(k, Y), E_m(k, Y)\}$, mechanisms to conceal correspondences between different dimensional order preserving encryption forms of same values are necessary.

About the mechanism to restrict order preserving variables to selected ones, fortunately, $E(k, x_1)+E(k, x_2)$ and $E(k, x_1)E(k, x_2)$, addition and multiplication of MA-based encryption forms, are decrypted to x_1+x_2 and x_1x_2 respectively regardless that $E(k, x_1)$ and $E(k, x_2)$ are order preserving or not. Therefore, by using single encryption key k, P can encrypt x_1 and x_2 so that relations $E(k, x_1)+E(k, x_2) = E(k, x_1+x_2)$ and $E(k, x_1)E(k, x_2) = E(k, x_1x_2)$ hold while making $E(k, x_1)$ order preserving and $E(k, x_2)$ not order preserving. Namely, P can restrict order preserving variables to the ones relating to conditional statements while maintaining additive and multiplicative features. In addition, P can define different values $\lambda_1, \lambda_2, ---, \lambda_M$ as constant λ of order indicators. This means except ones that are included in same conditional statements no one other than P can know order indicators corresponding to individual constants $\lambda_1, \lambda_2, ---, \lambda_M$. As a result, server S cannot calculate $\{\lambda_j a_1, \lambda_j a_2, ---, \lambda_j a_z\}$ for any j even if it obtains numbers of order preserving encryption forms, i.e. it cannot know ratios among coefficients $\{a_1, a_2, ---, a_z\}$. Mechanisms to remove appearances of different dimensional order preserving encryption forms of same values, in other words to conceal correspondences between different dimensional order preserving encryption forms of same values, can be implemented in 2 ways as below.

The 1st way is to prepare multiple encryption forms for each relevant value. For example, to add or compare $E(k, XY)$ and $E(k, XZ)$, and $E(k, XZ)$ and $E(k, Y^3)$, in addition to encryption forms $E_1(k, X, \lambda)$, $E_1(k, Y, \lambda)$ and $E_1(k, Z, \lambda)$ client P prepares $E_2(k, X, \mu^2)$, $E_1(k, Y, \mu)$ and $E_1(k, Z, \mu)$, where notation $E_n(k, x, \lambda^n)$ represents an n-dimensional encryption form of x with order indicator value $\lambda^n x$. Then, server S calculates $E_2(k, XY, \lambda^2) = E_1(k, X, \lambda)E_1(k, Y, \lambda)$ and $E_2(k, XZ, \lambda^2) = E_1(k, X, \lambda)E_1(k, Z, \lambda)$ to add or compare $E(k, XY)$ and $E(k, XZ)$. On the other hand to add or compare $E(k, XZ)$ and $E(k, Y^3)$, S calculates $E_3(k, XZ, \mu^3) = E_2(k, X, \mu^2)E_1(k, Z, \mu)$ and $E_3(k, Y^3, \mu^3) = E_1(k, Y, \mu)^3$. Namely, although S may know different encryption forms of same values $\{E_1(k, X, \lambda), E_2(k, X, \mu^2)\}$, $\{E_1(k, Y, \lambda)\ E_1(k, Y, \mu)\}$, $\{E_1(k, Z, \lambda), E_1(k, Z, \mu)\}$ and $\{E_2(k, XZ, \lambda^2), E_3(k, XZ, \mu^3)\}$, it cannot calculate X, Y, Z or XZ from them because order indicators of corresponding encryption forms are calculated based on different unknown constants λ and μ.

In the 2nd mechanism to add or compare different dimensional order preserving encryption forms $E_n(k, X)$ and $E_r(k, Y)$, P prepares set $\{E_{p(1)}(k, W_1), E_{p(2)}(k, W_2), ---, E_{p(Q)}(k, W_Q)\}$, encryption forms of secret numbers $W_1, W_2, ---, W_Q$ in advance, and instead of $E_n(k, X)$ and $E_r(k, Y)$ computer programs calculate $E_n(k, X_c)$ and $E_r(k, Y_c)$. After that they calculate $E_n(k, X_c)E_{p(n1)}(k, W_{n1})E_{p(2)}(k, W_{n2})---E_{p(nV)}(k, W_{nV}) = E_m(k, X)$ and $E_r(k, Y_c)E_{p(r1)}(k, W_{r1})E_{p(r2)}(k, W_{r2})---E_{p(rT)}(k, W_{rT}) = E_m(k, Y)$, and finally adds or compares $E_m(k, X)$ and $E_m(k, Y)$. Here, programs must choose encryption forms $\{E_{p(n1)}(k, W_{n1}), ---, E_{p(nV)}(k, W_{nV})\}$ and $\{E_{p(r1)}(k, W_{r1}), ---, E_{p(rT)}(k, W_{rT})\}$ from set $\{E_{p(1)}(k, W_1), ---, E_{p(Q)}(k, W_Q)\}$ so that relations $X_cW_{n1}---W_{nV} = X$, $Y_cW_{r1}---W_{rT} = Y$, $n+n_1+ --- +n_V = m$ and $r+r_1+ --- +r_T = m$ hold of course. Then, P can remove appearances of different dimensional encryption forms of X and Y, i.e. encryption form pairs $\{E_n(k, X), E_m(k, X)\}$ and $\{E_r(k, Y), E_m(k, Y)\}$. Here, provided that $E_1(k, x)$ has value λx as its order indicator, S can obtain pairs of order indicators $\{\lambda^n X_c, \lambda^m X_c W_{n1}W_{n2}---W_{nV}\}$ and $\{\lambda^r Y_c, \lambda^m Y_c W_{r1}W_{r2}---W_{rT}\}$, but S that does not know X, X_c, W_{n1}, ---, W_{nV}, Y, Y_c, W_{r1},---W_{rT} cannot calculate λ from them. But to avoid intense and sophisticated communications between P and S, values X_c and Y_c and encryption forms $\{E_{p(1)}(k, W_1), ---, E_{p(Q)}(k, W_Q)\}$ in the above must be generated and included in programs by P itself.

As above, client P can restrict variables to which order preserving encryption functions are applied to only ones that relate to conditional statements, and it can also protect encryption forms from plain text attacks even in environments where different dimensional order preserving encryption forms $E_r(k, X)$ and $E_m(k, Y)$ are added or compared.

5. Features of the Proposed Scheme

As discussed in previous sections, while exploiting secret key MA-based encryption functions, the proposed scheme successfully enables client P to delegate calculations of polynomials, maximums/minimums, etc. of values to server S while preserving secrets about the values. About the efficiency, although required computation volumes increase as numbers of elements in encryption forms increase, it is still practical when orders of polynomials are not too high. In detail, when x and y are encrypted to 1-dimensional arrays $\{x(1), x(2), ---, x(z)\}$ and $\{y(1), y(2), ---, y(z)\}$ for example, S must calculate $x(i)+y(i)$ for each $x(i)$ and $y(i)$ in $\{x(1), x(2), ---, x(z)\}$ and $\{y(1), y(2), ---, y(z)\}$ to calculate $x+y$, and as a worse thing, numbers of elements in encryption forms increase as dimensions of arrays increase. But if orders of polynomials are not high, numbers of elements in encryption forms can be restricted, e.g. when each value is encrypted to 30-elements 1-dimensional array and orders of the polynomials do not exceed 4, the number of elements in each encryption form is less than or equal to $(30)^4$ (< 1,000,000). Provided that T is the computation volume for calculating polynomials of plain values, this means the computation volume of the scheme does not exceed 10^6T which is comparable to volumes required for

handling large but not extremely large arrays (e.g. 1,000 x 1,000 2-dimensional or 100 x 100 x 100 3-dimensional arrays) appear in typical numerical simulations.

About secureness of the scheme, although MA-based encryption functions are weak against plain text attacks, entities can preserve their secrets because the scheme does not disclose data values in their plain forms. Provided that client P can assign U different values to each coefficient a_i, server S must examine U^{30} different possibilities to decrypt the above 1-dimensional encryption form {x(1), X(2), ---, x(30)} to $a_1x(1)+a_2x(2)+$ --- $+a_zx(30) = x$, and U is large enough because each a_i is a real number.

When compared with schemes based on public key based encryption functions including most recent fully homomorphic encryption functions [7,8], the proposed scheme is more practical as below. Firstly, the proposed scheme can handle real numbers that frequently appear in business and engineering applications easily totally in the same way as integers. Secondly, because MA-based encryption functions can have order preserving features, clients can communicate calculation algorithms to servers by usual computer programs instead of logic circuits. As a consequence, developments of calculation algorithms become easy even under complicated conditions. In addition, server S can execute computer programs more efficiently than simulating behaviors of complicated logic circuits.

Although not so efficient, the proposed scheme enables client P to delegate also calculations of general logical functions of its secret values to server S. Namely, S can calculate encryption forms of NOT, AND, OR of logical values from values encrypted by P as follows, and general logical functions can be constructed by combining NOT, AND, OR components. In the following u and v are real numbers, and they are regarded as true or false when their values are 1 or 0.

- NOT component Construction of NOT component is straight forward, i.e. when encryption form E(k, u) is given, the NOT component simply calculates E(k, 1)-E(k, u), i.e. E(k, 1)-E(k, u) is decrypted to (1–u) that corresponds to NOT of u. Where, client P informs server S of E(k, 1) in advance.

- AND component E(k, u) AND E(k,v) can be calculated simply as E(k, u)E(k, v). Namely, E(k, u)E(k, v) = E(k, uv) is decrypted to uv which becomes 1 only when both u and v are 1.

- OR component OR component is constructed by using AND and NOT components based on relation {u OR v} = NOT{(NOT u) AND (NOT v)}, i.e. E(k, u) AND E(k, v) is calculated as E(k, 1)-{E(k, 1)-E(k, u)}{E(k, 1)-E(k, v)}.

But it must be noted that AND component is implemented as a product of 2-logical values, and this means that encryption forms of logical functions may include high dimensional arrays. As a consequence, P and S must handle large number of elements, which decreases the efficiency of calculations. About plain text attacks, client P can successfully protect its encryption forms because encryption and plain form pair that P must disclose is only {E(k, 1), 1}.

6. Protecting Calculations from Malicious Entities

6.1. Protecting Clients

Server S in the previous section may behave dishonestly, i.e. S may return wrong calculation results to client P. P can protect itself from dishonest servers by exploiting 3 principles, i.e. data encryption, data redundancy and test data insertion.

The following mechanisms assume that client P provides server S with a series of values {X_1, X_2, ---, X_N} and corresponding to these inputs S returns a series of calculation results {R_1, R_2, ---, R_N} to P. But according to the data encryption principle, P encrypts the values to {E(k, X_1), ---, E(k, X_N)} by probabilistic MA-based encryption function E(k, x) and S calculates encryption forms {E(k, R_1), ---, E(k, R_N)} to be decrypted by P as discussed already. Then to protect P from dishonest servers, the data redundancy principle randomly inserts encryption forms {E(k, X^*_{h1}), ---, E(k, X^*_{hQ})} that are decrypted to {X_{h1}, ---, X_{hQ}} in input series {E(k, X_1), ---, E(k, X_N)}. As a result, to behave dishonestly server S, which does not know the correspondence between encryption forms E(k, X_{hj}) and E(k, X^*_{hj}) in the input series, must take risks of returning E(k, R_{hj}) and E(k, R^*_{hj}) that are decrypted to different values as calculation results corresponding to E(k, X_{hj}) and E(k, X^*_{hj}).

However, redundant encryption forms cannot protect P when S modifies programs, i.e. although modified programs may generate wrong calculation results, they produce results that are decrypted to a same value for each pair E(k, X_{hj}) and E(k, X^*_{hj}). To disable S to dishonestly modify the programs, the test data insertion principle randomly inserts encrypted test data E(k, T_1), E(k, T_2), ---, E(k, T_Q) at secret positions in input series {E(k, X_1), ---, E(k, X_N)}. Namely, P provides S with series {E(k, X_1), ---, E(k, X_{n1-1}), E(k, T_1), E(k, X_{n1+1}), ---, E(k, X_{n2-1}), E(k, T_2), E(k, X_{n2+1}), ---, E(k, X_N)}. Where, P knows correct calculation result t_j of each test value T_j. Then if S modifies the programs, they produce a result that is different from t_j for test encryption form E(k, T_j). Of course S tries to use original programs for E(k, T_j), but it cannot identify E(k, T_j) in the input series because E(k, x) is probabilistic.

6.2. Protecting Servers

As server S disturbed client P's business by having returned wrong calculation results, P may disturb S's business by claiming that S is dishonest despite S is honest. Servers can protect themselves from dishonest clients by exploiting anonymous signatures.

To protect it from P's dishonesty, S receives program A_P and encrypted data item value E(k, X_P) from P with anonymous signatures {$S(d_1 \| d_2, (T_P^{R+1}K_wC_w^R)^W{}_{\text{mod } B})$, $h(A_P)^R{}_{\text{mod } B}$} and {$S(d_1 \| d_2, (T_P^{R+1}K_wC_w^R)^W{}_{\text{mod } B})$, h(E(k, $X_P)^R{}_{\text{mod } B}$)}, and accepts them when the signatures are consistent. Where, B is a publicly known appropriate sufficiently large integer, R, W and w are P's secret integers, T_P is an integer assigned to P, and K_w and C_w are integers P calculates as $K_w = k^w{}_{\text{mod } B}$ and $C_w = c^w{}_{\text{mod } B}$ based on publicly known constant integers k and c and secret integer w. Under these settings, provided that d_1 and d_2 are different signing keys of an authority (this authority does not need to be S), anonymous credential $S(d_1 \| d_2, T_P^{R+1}K_wC_w^R)$ represents signature pair {$S(d_1, T_P^{R+1}K_wC_w^R)$, $S(d_2, T_P^{R+1}K_wC_w^R)$} (to simplify notations $_{\text{mod } B}$ is omitted

in the following). Namely, only P that knows integers R can prove the validity and the ownership of its showing credential $S(d_1 \| d_2, T_P^{R+1}K_wC_w^R)^W$ by successfully decomposing $(T_P^{R+1}K_wC_w^R)^W$ into $\{T_P^W, T_P^{RW}, K_w^W, C_w^{RW}\}$. On the other hand, no one except P can identify P from $S(d_1 \| d_2, T_P^{R+1}K_wC_w^R)^W$, because P never discloses R and it changes form $S(d_1 \| d_2, T_P^{R+1}K_wC_w^R)$ that had given from the authority to $S(d_1 \| d_2, T_P^{R+1}K_wC_w^R)^W$ by its secret integer W when it shows the credential [11].

About the anonymous signature, $h(x)$ is a hash function, i.e. for a given signature $\{S(d_1 \| d_2, T_P^{R+1}K_wC_w^R)^W, h(x)^R\}$ no one can generate y so that $h(y)$ becomes equal to $h(x)$. Then together with the fact that only P who knows R can calculate $h(x)^R$ from $h(x)$, S can prove that program A_P and encrypted data value $E(k, X_P)$ were certainly provided by a legitimate entity (i.e. P). But different from usual digital signatures, validities and ownerships of the above anonymous signatures must be proved by their owners themselves. Here, although P never discloses R to others, anonymous credential $S(d_1 \| d_2, T_P^{R+1}K_wC_w^R)^W$ forces P to honestly calculate $h(A_P)^R$ and $h(E(k, X_P))^R$ from $h(X_P)$ and $h(E(k, X_P))$ [11].

In detail, at a time when P claims A_P and $E(k, X_P)$ are not the ones that it had provided, S calculates hash values $h(A_P)$ and $h(E(k, X_P))$ from A_P and $E(k, X_P)$. Then, because no one can modify A_P or $E(k, X_P)$ while maintaining their hash values as $h(A_P)$ or $h(E(k, X_P))$, and only P can calculate $h(A_P)^R$ and $h(E(k, X_P))^R$ from $h(A_P)$ and $h(E(k, X_P))$, S can convince anyone that it certainly had received A_P and $E(k, X_P)$ from an entity that is claiming S is dishonest, when P calculates $h(A_P)^R$ and $h(E(k, X_P))^R$ from $h(A_P)$, $h(E(k, X_P))$ and credential $S(d_1 \| d_2, T_P^{R+1}K_wC_w^R)^W$ that includes P's secret R. Of course P may provide A_P and $E(k, X_P)$ as $\{S(d_1 \| d_2, (T_Q^{U+1}K_vC_v^U)^V), h(A_P)^U\}$ and $\{S(d_1 \| d_2, (T_P^{U+1}K_vC_v^U)^V), h(E(k, X_P)^U)\}$ while stealing or borrowing credential $S(d_1 \| d_2, T_P^{U+1}K_vC_v^U)$ from others to claim A_P and $E(k, X_P)$ are not correct, but S does not accept them if the credential is a stolen one because P does not know U. In a case where $S(d_1 \| d_2, T_P^{U+1}K_vC_v^U)$ is a borrowed one, P or an entity that had lent the credential to P must calculate $h(A_P)^U$ and $h(E(k, X_P)^U$ honestly. Also if P is dishonest, S can identify it even if P is anonymous. On the other hand, P can conceal its secret R from S if it is honest, in other words, honest P can preserve its anonymity.

7. Conclusion

While exploiting MA-based encryption functions, a scheme that enables entities to delegate calculations of simple functions including polynomials and maximums/ minimums of values to others without disclosing secrets about the values is proposed. Distinctive features of the scheme are it can handle real numbers in the same way as integers and it enables entities to define complicated calculation algorithms as computer programs. About the efficiency, the scheme is practical enough when orders of polynomials are not too high.

References

[1] Chaum, D., "Untraceable electronic mail, return address and digital pseudonyms," *Communications of the ACM*, 24 (2). 84-88. 1981.

[2] Yao, A. C., "How to generate and exchange secrets," *Proc. of the 27th IEEE Symposium on Foundations of Computer Science*, 162-167. 1986.

[3] Goldreich, O., Micali, M. and Wigderson, A., "How to play any mental game," *Proc. of 19th ACM Symposium on Theory of Computing*, 218-229. 1987.

[4] Naor, M., Pinkas, B. and Sumner, R., "Privacy preserving auctions and mechanism design," *1st ACM Conference on Electronic Commerse*, 129-139. 1999.

[5] Ogiso, T., Sakabe, Y., Soshi, M. and Miyaji, A., "Software obfuscation on a theoretical basis and its implementation," *IEICE Trans. Fundamentals*, E86-A, (1), 176-186. 2003.

[6] Golle, P. and Jakobsson, M., "Reusable anonymous return channels," *Proc. of the 2003 ACM Workshop on Privacy in the Electronic Society*, 94-100. 2003.

[7] Gentry, C. "Fully homomorphic encryption using ideal lattices," *Proc. of Symposium on theory of computing –STOC 2009*, 169-178. 2009.

[8] Chung, K., Kalai, Y. and Vadhan, S., "Improved Delegation of Computation Using Fully Homomorphic Encryption," *CRYPT 2010, LNCS 6223*, 483-501. 2010.

[9] Haddad, H, Tamura, S., Taniguchi, S. and Yanase, T., "Development of anonymous networks based on symmetric key encryptions," *Journal of Networks*, 6 (11), 1533-1542. 2011.

[10] Tamura, S., *Anonymous Security Systems and Applications: Requirements and Solutions*, Information Science Reference, 2012.

[11] Tamura, S. and Taniguchi, S., "Enhanced Anonymous Tag Based Credentials," *Information Security and Computer Fraud*, 2 (1), 10-20, 2014.