Improved Secure Computation over Real Numbers and Its Application to Reliability Engineering

Takumi Iseki* Masahiro Hayashi**
Major of Informatics, Graduate School of Integrative Science and Engineering, Tokyo City University
Tokyo, Japan
Email: *g1981804@tcu.ac.jp **mhaya@tcu.ac.jp

***Abstract*—Secure computation over real numbers has recently been proposed for outsourcing computations while maintaining high security. This scheme has had difficulty in practice because it causes an unreasonably large computational complexity in decryption. While previous research resolved this difficulty, problems remain because the encryption is based on a kind of Caesar cipher known to be weak in terms of security. This paper proposes an improvement on the previous research. The key idea is replacing the differential operator used in the encryption and decryption process with another one so that encryption can be altered to use additional secret real numbers. This addition realizes stronger security because hackers must find not only keys but also these secret real numbers.**

***Index Terms*—Secure computation, Full homomorphic encryption, Reliability engineering, Cyber security**

I. Introduction

While computing is a useful tool in science and engineering (including network management) computations sometimes become very difficult especially when they become very large. In such cases, they can be outsourced to subcontractors or to powerful supercomputers through cloud network systems.

However, outsourcing has a security problem. That is, when we outsource a computation, we must give important data, such as on the customers, the reliability of equipment, etc., to the outsource side.

Secure computation is a solution to this problem, and fully homomorphic encryption (FHE) is a key technology [1]-[5]. This sort of encryption is realized by a mapping Φ having the following three properties, where x and y are numbers:

$$\Phi(x+y) = \Phi(x) + \Phi(y), \Phi(x \times y) = \Phi(x) \times \Phi(y),$$
$$\Phi^{-1} \text{ is obtained by security keys.}$$

Gentry [1] proposed the FHE scheme and Refs. [2]-[3] made improvements to it. On the other hand, Gai et al. [4] claimed that their computation becomes prohibitively large when real numbers are used. They proposed a new scheme, which this paper calls 'secure computation over real numbers' as a solution to the problem. (Gai et al. uses the phrase 'full homomorphic encryption over real numbers', whereas we will use the more easily understandable 'secure computation'.)

Gai's solution, avoids Gentry's problem of high computational complexity when using real numbers.

However, our previous study [5] claimed that Gai's approach requires the same number of operations (additions and multiplications) between the decryption on the outsourcing side and the computation on the subcontractor side in the worst case. In such case, it is obviously faster and more secure to execute a computation without outsourcing it. This is problematic.

Our previous study [5] solved this problem by focusing on computations involving polynomials whose input variables are n real numbers. The method of Ref. [5] outsources the computation of a polynomial after encrypting the inputs by adding a real number H (key). If we repeat the outsourcing n times while changing the value of H, we can decrypt the output of the polynomial by multiplying a special matrix with a vector consisting of encrypted computational results. Ref. [5] demonstrated its effectiveness in the reliability engineering field.

However, this approach is still problematic, because the encryptions of the inputs are executed by adding the same value H to n inputs in each outsourcing. This implies that the proposed encryption is a kind of Caesar cipher which is known to be weak.

This paper proposes to eliminate this weakness by altering the differential operator used in the scheme of Ref. [5] with another operator. This alteration in effect changes 'adding H' to 'adding $H \times \lambda_i$'. Here, λ_i is a secret real number.

This improvement realizes a secret computation that is surely stronger without a serious increase in computational complexity of encryption and decryption.

II. Previous Research

This section explains the most recent research on secret computation over real numbers in Ref. [5]

A. Preparations

Let f be a polynomial whose input variables are $x_1, x_2, \cdots, x_n$. The order of $f = f(x_1, x_2, \cdots, x_n)$ is the order of g, where g is a polynomial assuming that $x_1 = x_2 = \cdots = x_n$ in f. For example, the order of $f = (1-x_1)^2 x_2^2$ is four because $g = (1-x)^2x^2 + x^2 = 2x^2 - 2x^3 + x^4$.

We define $\kappa_H(h)$ as follows, where d is any differential operator, and $d^i(h) = d(d^{i-1}(h))$ with $d^0(h) = h$, and h is suitably differentiable.

$$\kappa_H(h) = \sum_{i=0}^{\infty} \frac{H^i}{i!} d^i(h) \tag{1}$$

By we replacing d with a specific differential operator D defined below, we write $\kappa_H(f)$ as $\varphi_H(f)$.

$$D(f) = \sum_{i=1}^{n} \frac{\partial f}{\partial x_i}$$

$$\varphi_H(f) = \sum_{i=0}^{m} \frac{H^i}{i!} D^i(f) \tag{2}$$

Here, D is a differential operator and f is a polynomial of order m. TheFore, it is easily determined that if $i > m$, then $D_i(f) = 0$. That's why ∞ in the definition of $\kappa_H(f)$ is replaced with m in $\varphi_H(f)$.

B. Encryption and decryption

The following property was proved in Ref. [5] using the results of Ref. [6].

PROPERTY 1.

'For any polynomials $f_1 \,\&\, f_2, \varphi_H(f_1 + f_2) = \varphi_H(f_1) + \varphi_H(f_2)$, $\varphi_H(f_1 \times f_2) = \varphi_H(f_1) \times \varphi_H(f_2)$ are true.'

This implies that $\varphi_H(f)$ satisfies requirements (I) & (II) in the Introduction, and its output can be computed in a finite time. Therefore, fully homomorphic encryption will be realized if we can find a method of decryption that satisfies (III). Ref. [5] gave a solution for this decryption.

PROPERTY 2.

'Let F be an $(m+1)\times(m+1)$ matrix, in which each element (a, b) is $\frac{H_a^{b-1}}{(b-1)!}$. Furthermore, let G be an $m+1$ column vector whose $a-th$ element is $f(x_1 + H_a, x_2 + H_a, \cdots, x_n + H_a)$. The first element of the column vector obtained by $F^{-1}G$ equals f, where F^{-1} is the inverse matrix of F.'

Ref. [5] proved Property 2 using Property 1. The following is the procedure for executing encryption and decryption if we want to compute the output of f with inputs $x_1, x_2, \cdots, x_n$.

STEP 1. Prepare $m+1$ real numbers $H_1, H_2, \cdots, H_{m+1}$ as secret keys

STEP 2. Outsource the r. h. s. of the following computations.

$$G_1 = f(x_1 + H_1, x_2 + H_1, \cdots, x_n + H_1)$$

$$G_2 = f(x_1 + H_2, x_2 + H_2, \cdots, x_n + H_2)$$

$$\vdots$$

$$G_{m+1} = f(x_1 + H_{m+1}, x_2 + H_{m+1}, \cdots, x_n + H_{m+1})$$

STEP 3. Compute $F^{-1}G$. The first element of this vector shows the value of f, where F and G are as below.

$$F = \begin{bmatrix} \frac{H_1^0}{0!} & \frac{H_1^1}{1!} & \frac{H_1^2}{2!} & \cdots & \frac{H_1^m}{m!} \\ \frac{H_2^0}{0!} & \frac{H_2^1}{1!} & \frac{H_2^2}{2!} & \cdots & \frac{H_2^m}{m!} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{H_{m+1}^0}{0!} & \frac{H_{m+1}^1}{1!} & \frac{H_{m+1}^2}{2!} & \cdots & \frac{H_{m+1}^m}{m!} \end{bmatrix}, G = \begin{bmatrix} G_1 \\ G_2 \\ \vdots \\ G_{m+1} \end{bmatrix}$$

C. Motivative example

Let us show a simple example of executing the procedure in the previous subsection. Let f be $f = x_1 + x_2 x_3$ with $x_1 = 0.1, x_2 = 0.5, x_3 = 0.3$.

STEP 1. We prepare $H_1 = 1.0, H_2 = 2.0, H_3 = 3.0$, and $H_4 = 4.0$ as secret keys.

STEP 2. We have

$$\begin{aligned} G_1 &= f(x_1 + H_1, x_2 + H_1, x_3 + H_1) \\ &= (0.1 + 1.0) + (0.5 + 1.0) \times (0.3 + 1.0) = 3.05 \\ G_2 &= (0.1 + 2.0) + (0.5 + 2.0) \times (0.3 + 2.0) = 7.85 \\ G_3 &= (0.1 + 3.0) + (0.5 + 3.0) \times (0.3 + 3.0) = 14.65 \\ G_4 &= (0.1 + 4.0) + (0.5 + 4.0) \times (0.3 + 4.0) = 23.45. \\ G &= \begin{bmatrix} 3.05 \\ 7.85 \\ 14.65 \\ 23.45 \end{bmatrix} \end{aligned}$$

STEP 3. F is obtained as follows.

$$F = \begin{bmatrix} \frac{1^0}{0!} & \frac{1^1}{1!} & \frac{1^2}{2!} & \frac{1^3}{3!} \\ \frac{2^0}{0!} & \frac{2^1}{1!} & \frac{2^2}{2!} & \frac{2^3}{3!} \\ \frac{3^0}{0!} & \frac{3^1}{1!} & \frac{3^2}{2!} & \frac{3^3}{3!} \\ \frac{4^0}{0!} & \frac{4^1}{1!} & \frac{4^2}{2!} & \frac{4^3}{3!} \end{bmatrix} = \begin{bmatrix} 1 & 1 & \frac{1}{2} & \frac{1}{6} \\ 1 & 2 & 2 & \frac{4}{3} \\ 1 & 3 & \frac{9}{2} & \frac{9}{2} \\ 1 & 4 & 8 & \frac{32}{3} \end{bmatrix}$$

Accordingly,

$$F^{-1}G = \begin{bmatrix} 1 & 1 & \frac{1}{2} & \frac{1}{6} \\ 1 & 2 & 2 & \frac{4}{3} \\ 1 & 3 & \frac{9}{2} & \frac{9}{2} \\ 1 & 4 & 8 & \frac{32}{3} \end{bmatrix}^{-1} \begin{bmatrix} 3.05 \\ 7.85 \\ 14.65 \\ 23.45 \end{bmatrix} = \begin{bmatrix} 0.250 \\ 1.80 \\ 2.00 \\ 23.5 \end{bmatrix}$$

D. Problem of our previous work

The decrypted value of f is 0.250.

Ref. [5] demonstrated the usefulness of the procedure of Steps 1-3 for secret computation especially in reliability engineering [7][8].

However, this method still has problems because it makes available the following data to the subcontractor in outsourcing.

$$x_1 + H_1, x_2 + H_1, \cdots, x_n + H_1$$

$$x_1 + H_2, x_2 + H_2, \cdots, x_n + H_2$$

$$\vdots$$

$$x_1 + H_{m+1}, x_2 + H_{m+1}, \cdots, x_n + H_{m+1}$$

The openly available data cause the following problems.

Problem 1.

If one key H_j of $H_1, H_2, \cdots, H_n$ is known to the subcontractor, then any input x_i can be computed from openly available data $x_i + H_j$ by $(x_i + H_j) - H_j$.

Problem 2.

Even if all of $H_1, H_2, \cdots, H_n$ are kept secret, the following important information is not concealed:

$\langle 1 \rangle$ Information about which is bigger between any pair of x_i and x_j, as a hacker can find $x_i < x_j$ if $(x_i + H_k) < (x_j + H_k)$

$\langle 2 \rangle$ Information about $x_i - x_j$, as $x_i - x_j$ is computed by $(x_i + H_k) - (x_i + H_k)$

III. Proposal

This section proposes an improved method to solve the problems described at the end of the previous section.

A. Basic idea

Our proposal makes only a small change to the differential operator used in the scheme of Ref. [5].

Note that the basis of Ref. [5] is Eq. (2) in Subsection A of Section II. This equation is

$$\varphi_H(f) = \sum_{i=0}^{m} \frac{H^i}{i!} D^i(f) \tag{2}$$

Here, the differential operator D is the one defined in Section II:

$$D(f) = \sum_{i=1}^{n} \frac{\partial f}{\partial x_i}$$

Even if we replace $\varphi_H(f)$ with $\varphi_{0H}(f)$, where $\varphi_{0H}(f)$ is obtained from $\varphi_H(f)$ by replacing D with the following D_0, Property 1 in subsection A of section II is true.

$$D_0(f) = \sum_{i=1}^{n} (\frac{\partial f}{\partial x_i} \times \lambda_i)$$

This is because the proof of Property 1 in Ref. [5] applies even when D is replaced with any other differential operator. Therefore, Property 2 is true even if we replace $\varphi_H(f)$ with $\varphi_{0H}(f)$. Our improvement's only difference from the scheme of Ref. [5] is caused by using a different differential operator in the encryptions of $x_1, x_2, \cdots, x_n$. In Ref. [5], $x_1, x_2, \cdots, x_n$ are encrypted with $x_1 + H_i, x_2 + H_i, \cdots, x_n + H_i$, because Ref. [5] uses $\varphi_{H_i(x_1)}$ for encryption. That is, when we encrypt $x_1, x_2, \cdots, x_n$ to $\varphi_{H_i}(x_1), \varphi_{H_i}(x_2), \cdots, \varphi_{H_i}(x_n), \varphi_{H_i}(x_i) = H_i^0 D^0(x_j) + H_i^1 D^1(x_j) = x_i + H_i$ because of the definition of $\varphi_H()$. However, if we use $\varphi_{0H_i}()$, then $x_1, x_2, \cdots, x_n$ are encrypted with $x_1 + H_i\lambda_1, x_2 + H_i\lambda_2, \cdots, x_n + H_i\lambda_n$, because $\varphi_{0H_i}(x_i) = H_i^0 D_0^0(x_j) + H_i^1 D_0^1(x_j) = x_j + H_i\lambda_i$ from the definitions of $\varphi_{0H_i}()$ and D_0.

B. Details of encryption and decryption

The following is the key property of our proposal.

Property 3.

Let G' be an $m+1$ column vector whose $a-th$ element is $f(x_1 + H_a\lambda_1, x_2 + H_a\lambda_2, \cdots, x_n + H_a\lambda_n)$. The first element of the column vector obtained by $F^{-1}G$' equals f, where F is the same matrix in Property 2 in subsection B of section III.

Property 3 is proved in the same manner as Property 2 in Ref. [5] simply by exchanging $D()$ with $D_0()$.

The encryption and decryption procedure is obtained from the steps in subsection B of section II by replacing G_i with G_i' in Steps 2 and 3, G with G' in Step 3, and replacing the equations in Step 2 with the following ones.

$$G_1\text{'} = f(x_1 + H_1\lambda_1, x_2 + H_1\lambda_2, \cdots, x_n + H_1\lambda_n)$$
$$G_2\text{'} = f(x_1 + H_2\lambda_1, x_2 + H_2\lambda_2, \cdots, x_n + H_2\lambda_n)$$
$$\vdots$$
$$G_{m+1}\text{'} = f(x_1 + H_{m+1}, x_2 + H_{m+1}, \cdots, x_n + H_{m+1}\lambda_n)$$

C. Motivative example

Let us show a simple example of executing the steps of the procedure in the previous subsection. Let f and the values of x_1, x_2, and x_3 be the same as in subsection C of section II.

STEP 1.

We prepare the same values for H_1, H_2, and H_3 as in subsection C of section II as secret keys, and $\lambda_1 = 4.0, \lambda_2 = 5.0, \lambda_3 = 6.0$.

STEP 2.

We have

$$\begin{aligned} G_1 &= f(x_1 + H_1\lambda_1, x_2 + H_1\lambda_2, x_3 + H_1\lambda_3) \\ &= (0.1 + 1.0 \times 4.0) + (0.5 + 1.0 \times 5.0) \\ &\quad \times (0.3 + 1.0 \times 6.0) = 38.75 \\ G_2 &= (0.1 + 2.0 \times 4.0) + (0.5 + 2.0 \times 5.0) \\ &\quad \times (0.3 + 2.0 \times 6.0) = 137.25 \\ G_3 &= (0.1 + 3.0 \times 4.0) + (0.5 + 3.0 \times 5.0) \\ &\quad \times (0.3 + 3.0 \times 6.0) = 295.75 \\ G_4 &= (0.1 + 4.0 \times 4.0) + (0.5 + 4.0 \times 5.0) \\ &\quad \times (0.3 + 4.0 \times 6.0) = 514.25. \end{aligned}$$

$$\text{Accordingly, } G\text{'} = \begin{bmatrix} 38.75 \\ 137.25 \\ 295.75 \\ 514.25 \end{bmatrix}$$

STEP 3. F is the same as in the example of subsection C of section II. Accordingly, we obtain

$$F^{-1}G\text{'} = \begin{bmatrix} 1 & 1 & \frac{1}{2} & \frac{1}{6} \\ 1 & 2 & 2 & \frac{4}{3} \\ 1 & 3 & \frac{9}{2} & \frac{9}{2} \\ 1 & 4 & 8 & \frac{32}{3} \end{bmatrix}^{-1} \begin{bmatrix} 38.75 \\ 137.25 \\ 295.75 \\ 514.25 \end{bmatrix} = \begin{bmatrix} 0.250 \\ 8.50 \\ 60.0 \\ 0.00 \end{bmatrix}$$

This example gives 0.250 as the decrypted result.

IV. Computational complexity and security strengh

Ref. [5] indicated that the computational complexity of its encryption and decryption is determined by computing F^{-1} from F. We emphasize that this computational complexity does not increase as a result of our improvement, because our improvement only affects the data used in outsourcing from $x_i + H_i$ to $x_j + H_i\lambda_i$, which does not increase the computational complexity.

On the other hand, the security becomes stronger because we no longer face the problems described at the end of the previous section:

1. In the previous method, if a key H_j is known to the subcontractor, then x_i can be computed from openly available data x_i+H_j by $(x_i+H_j)-H_j$. However, even if H_i is known, in our improved method, x_i cannot be computed because the hacker must find the secret real number λ_i to compute x_i from H_j and $(x_i + H_j\lambda_i)$.

2. In the previous method, even if all of $H_1, H_2, \cdots, H_n$ are kept secret, the following important information is not concealed:

$\langle 1 \rangle$ Information about which is bigger between any pair of x_i and x_j, as a hacker can find $x_i < x_j$ if $(x_i + H_j) < (x_j + H_k)$

$\langle 2 \rangle$ Information about $x_i - x_j$, as $x_i - x_j$ is computed by $(x_i + H_k) - (x_j + H_k)$

Our improvement solves the problem of $\langle 1 \rangle$ because the hacker cannot find $x_i < x_j$ even if $(x_i + H_k\lambda_i) < (x_j + H_k\lambda_j)$ is found by appropriately selecting $\lambda_1, \lambda_1, \cdots, \lambda_n$. It also solves $\langle 2 \rangle$ because even if a hacker wants to find the value of $x_i - x_j$ from $(x_i + H_k\lambda_i) - (x_j + H_k\lambda_j) = (x_i - x_j) + H_k(\lambda_i - \lambda_j)$, he or she must find information about H_k, λ_i, and λ_j , which are secret.

Consequently, our method increases the security over that of the previous method without increasing the computational complexity.

V. Application to Reliability Engineering

A. Problem of system reliability

Ref. [5] indicated that the method described in section II is very useful for reliability engineering, because this field commonly uses long polynomials and important field for network management.

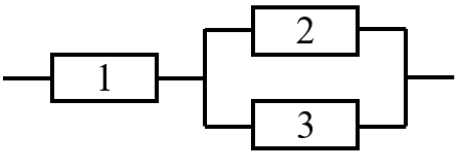

Fig. 1. Example of reliability block diagram.

An example framework of this sort of engineering is explained as below.

Suppose we have a reliability block diagram (RBD) representing the system structure, such as the one illustrated in Fig. 1. If the probability of each component working (component reliability, denoted by x_i) is known, then the probability of the whole system working (system reliability, denoted by R) can be computed from RBD, where system is working if there exists a path between right side end and left side end. In Fig. 1, the system is working if Components 1 and 2 are working or Components 1 and 3 are working. Therefore, $R = x_1x_3 + x_1x_3 - x_1x_2x_3$. If R is not sufficiently near 1, then the system design plan is reconfigured by altering RBD and R is recomputed. The repetition of this process gives a reliable design plan at a reasonable cost.

B. Numerical experiment

We wrote software to compute R by using our improved method. The environment of this software was the same as in Ref. [5]. The side that receives the outsourced job computed R by using a factoring algorithm [8]. Below, we describe numerical results of computing R of the RBD in Fig. 2, where the component numbers are omitted.

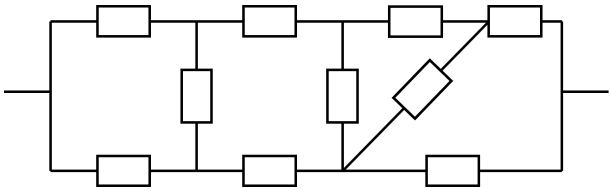

Fig. 2. Reliability block diagram for numerical experiment.

he secret keys are

$$H_1 = 0.396465, H_2 = 0.840485, H_3 = 0.353336,$$
$$H_4 = 0.446583, H_5 = 0.318693, H_6 = 0.886428,$$
$$H_7 = 0.015583, H_8 = 0.584090, H_9 = 0.159369,$$
$$H_{10} = 0.38716, \text{and } H_{11} = 0.691004.$$

The secret real numbers are

$$\lambda_1 = 0.058859, \lambda_2 = 0.899854, \lambda_3 = 0.163546,$$
$$\lambda_4 = 0.159072, \lambda_5 = 0.533065, \lambda_6 = 0.604144,$$
$$\lambda_7 = 0.582699, \lambda_8 = 0.269971, \lambda_9 = 0.390478, \text{and}$$
$$\lambda_{10} = 0.293401.$$

The result of the encryption is $R = 0.99969606129548460594$, where the runtime of the software is 0.016 seconds. If we compute R without encryption, then $R = 0.99969606129555621532$. For the same reliability block diagram and same values of x_i and H_i, the result computed using the method in Ref. [5] is 0.99969606129548460594. Thus, our approach has almost the same accuracy as the method of Ref. [5]

References

[1] C. Gentry, "Full homomorphic encryption using ideal lattices", 41th ACM symp. Theory of Computing, pp. 169-178, 2009.

[2] P. N. Smart and F. Vercauteren"Fully homomorphic encryption with relatively small key and ciphertext sizes", 10th PKC, 6050, pp. 420-443, 2010.

[3] C. Gentry and S. Halevi, "Implementing gentry's full-homomorphic encryption scheme", 11th EUROCRYPT, pp. 129-148, 2011.

[4] K. Gai, M. Qiu, Y. Li, and Y. X. Liu, "Advanced fully homomorphic encryption scheme over real numbers", 4th CSCloud, pp. 64-69, 2017.

[5] T. Iseki and M. Hayashi, "New secure computation over real numbers and its application to reliability engineering", International Workshop on Security, submitting, 2019.

[6] D. Kalman, "Combinatorial and functional identities in one-parameter matrices", The American Mathematical Monthly, 94(1), pp. 21-3, 1987.

[7] J. C. Colbourn, "The combinatorics of network reliability", Oxford University of Waterloo, New York, 1987.

[8] B. L. Page and E. J. Perry, "A practical implementation of factoring theorem for network reliability", IEEE Trans. Reliability, vol. 37, no, 3, pp. 259-267, 1988.