Scientific Research Publishing

Communications and Network, 2019, 11, 1-10
http://www.scirp.org/journal/cn
ISSN Online: 1947-3826
ISSN Print: 1949-2421

Acceleration of Homomorphic Arithmetic Processing Based on the ElGamal Cryptosystem

Takuma Jogan[1], Tomofumi Matsuzawa[2], Masayuki Takeda[2]

[1]Department of Information Sciences, Graduate School of Tokyo University of Science, Tokyo, Japan
[2]Department of Information Sciences, Tokyo University of Science, Tokyo, Japan
Email: 6317619@ed.tus.ac.jp, t-matsu@is.noda.tus.ac.jp, takeda@rs.tus.ac.jp

How to cite this paper: Jogan, T., Matsuzawa, T. and Takeda, M. (2019) Acceleration of Homomorphic Arithmetic Processing Based on the ElGamal Cryptosystem. *Communications and Network*, **11**, 1-10.
https://doi.org/10.4236/cn.2019.111001

Received: December 21, 2018
Accepted: January 21, 2019
Published: January 24, 2019

Abstract

In recent years, opportunities for using cloud services as computing resources have increased and there is a concern that private information may be leaked when processes data. The data processing while maintaining confidentiality is called secret computation. Cryptosystems can add and multiply plaintext through the manipulation of ciphertexts of homomorphic cryptosystems, but most of them have restrictions on the number of multiplications that can be performed. Among the different types of cryptosystems, fully homomorphic encryption can perform arbitrary homomorphic addition and multiplication, but it takes a long time to eliminate the limitation on the number of homomorphic operations and to carry out homomorphic multiplication. Therefore, in this paper, we propose an arithmetic processing method that can perform an arbitrary number of homomorphic addition and multiplication operations based on ElGamal cryptosystem. The results of experiments comparing with the proposed method with HElib in which the BGV scheme of fully homomorphic encryption is implemented showed that, although the processing time for homomorphic addition per ciphertext increased by about 35%, the processing time for homomorphic multiplication was reduced to about 1.8%, and the processing time to calculate the statistic (variance) had approximately a 15% reduction.

Keywords

ElGamal Cryptosystem, Homomorphic Cryptosystem, Delegating Computation

1. Introduction

With the recent development of cloud services, there has been a growing trend of outsourcing computational tasks. This gives rise to the important security is-

sue of protecting privacy, since personal information is being transferred. To solve the problem, homomorphic cryptosystems capable of computing plaintext by the manipulation of ciphertexts have attracted attention.

Homomorphic cryptosystems include additive homomorphic encryption that can perform only homomorphic additions such as Paillier encryption and lifted-ElGamal encryption, and multiplicative homomorphic encryption that can perform only homomorphic multiplications such as RSA encryption and ElGamal encryption [1] [2] [3]. In addition, homomorphic cryptosystems that can perform both homomorphic addition and homomorphic multiplication are called fully homomorphic encryption (FHE) [4]. FHE has high convenience, but there is a problem that its processing speed is very slow.

Therefore, in this paper, we propose a system capable of both homomorphic addition and homomorphic multiplication based on the ElGamal cryptosystem by unifying the random number part normally included in ElGamal ciphertext with all ciphertexts. However, this situation raises concerns about a decline in security compared to the ordinary ElGamal cryptosystem. Hence, in the state other than homomorphic computation, it is in the form of ordinary ElGamal ciphertext. To accomplish this, we replace the value of r included in ElGamal ciphertext into constants or random numbers.

The rest of the paper is organized as follows. Homomorphic Cryptosystem is introduced in Section 2. In Section 3, ElGamal Cryptosystem is introduced. In Section 4, Fully Homomorphic Encryption is introduced. We propose an arithmetic processing method that can perform an arbitrary number of homomorphic addition and multiplication operations based on ElGamal cryptosystem in Section 5. Section 6 shows results of two experiments. Sections 7 - 9 draw discussion, future work, and conclusions.

2. Homomorphic Cryptosystem

A homomorphic cryptosystem can perform the addition and multiplication of plaintext by the manipulation of ciphertexts. When ciphertext $Enc(m_1)$, $Enc(m_2)$ for plaintext m_1, m_2 are given, $Enc(m_1 \circ m_2)$ can be obtained without plaintext or a secret key, where $\circ$ is a binary operator such as addition or multiplication.

3. ElGamal Cryptosystem

The ElGamal cryptosystem is a public key cryptosystem based on the premise that a discrete logarithm problem of a group with a large order is difficult. ElGamal cryptosystem consists of three components: the key generator, the encryption algorithm, and the decryption algorithm.

Key generation

Generate a cyclic group G of order q which is a large prime number. Select a generator g of G and a random integer x from $\{0,\cdots,q-1\}$. Compute h as follows.

$$h \equiv g^x \pmod{q}$$

The public key is (G, q, g, h) and the secret key is x.

Encryption

To encrypt a message $m \in G$, we randomly select $r \in \{0, \cdots, q-1\}$ and compute $c_1, c_2 \in G^2$ as follows.

$$c_1 \equiv g^r \pmod{q}$$

$$c_2 \equiv mg^{xr} \pmod{q}$$

The ciphertext is (c_1, c_2).

Decryption

To decrypt a ciphertext $(c_1, c_2) \in G^2$, we compute $m \in G$ as follows.

$$\frac{c_2}{c_1^x} = \frac{mg^{xr}}{g^{xr}} = m \pmod{q}$$

The plaintext is m.

4. Previous Rsearch

Fully Homomorphic Encryption

FHE is capable of arbitrary operations such as the addition and multiplication of plaintext by the manipulation of ciphertexts. When plaintext is encrypted, it adds constant noise according to security parameters. This noise increases with each homomorphic operation, and if the noise becomes too large, it becomes impossible to decrypt the ciphertext into the original plaintext. In particular, when homomorphic multiplication is performed, noise increases greatly. Therefore, developers created somewhat homomorphic encryption (SHE), which restricts the number of homomorphic multiplications. Then, in 2009, Gentry proposed Bootstrap as a method to reduce ciphertext noise in SHE. This makes it possible to take restrictions on SHE and implement FHE. However, Bootstrap is not practical from the viewpoint of processing speed because it greatly increases the number of calculations.

Since then, studies such as a method called packing for encrypting plural plaintexts into one ciphertext and a scheme for reducing noise of ciphertext without using Bootstrap are progressing [5] [6]. These have greatly improved the performance, but there is still a problem with processing speed.

Bootstrap

Bootstrap reduces noise accumulated in ciphertext by homomorphic operation. FHE makes the decipherability difficult to realize by adding noise to the ciphertext as the basis for security. This noise increases with each iteration of homomorphic operation, and if it exceeds a certain threshold value it can not decode correctly.

Bootstrap encrypts ciphertexts in which noise is stored again and performs decryption processing using the encrypted secret key. As a result of this decoding, a new ciphertext is accumulated in which only the noise required for de-

coding is stored. With this approach, Gentry realized the configuration of FHE.

5. Proposed Method

5.1. Overview

In this paper, we propose a method capable of both homomorphic addition and homomorphic multiplication based on the ElGamal cryptosystem. In the proposed method, the random number part normally included in ElGamal ciphertext is unified with all ciphertexts. This allows for both homomorphic addition and homomorphic multiplication. However, since this situation raises concerns about a decline in security compared to the ordinary ElGamal cryptosystem, in the state other than homomorphic computation, it is in the form of ordinary ElGamal ciphertext. Hereafter, the form of the ciphertext at the time of homomorphic operation is called "an arithmetic form", and the form of the ciphertext in other case is called "a stored form".

5.2. System Configuration

We propose a delegating computation model in which encrypted data are transmitted from the user to the cloud, and the cloud performs arithmetic processing on those encrypted data.

It is assumed that the cloud includes a calculation server and a transformation server. The calculation server performs arithmetic operations such as statistical processing in the encrypted state, and the transformation server replaces the value of r included in ElGamal ciphertext into constants or random numbers (Figure 1). Also, it is assumed that the user and the transformation server can safely share the secret key.

Figure 1 shows the system's processing flow. First, the user transmits ElGamal ciphertexts that are stored form, to the calculation server, and the calculation server stores the data. Upon receiving the calculation request from the user, the calculation server exchanges data with the transformation server to convert the stored form of the ElGamal ciphertexts into the arithmetic form of the ElGamal ciphertexts. Then, the calculation server performs processing according to the calculation request by homomorphic operations, and obtains the calculation results of the encrypted state. Then, the calculation server exchanges data with the transformation server to converts the arithmetic form of the ElGamal ciphertexts into the stored form of the Elgamal ciphertexts. Finally, the stored form of the encrypted operation result is transmitted to the user, and the user decrypts it using a secret key to obtain the calculation result. If multiple processing contents are included in the calculation request, the same procedure is repeated (Figure 2).

We assume the roles of the user, the calculation server, the transformation server, the constraints imposed, and the functions as follows.

User

- Know the plaintext

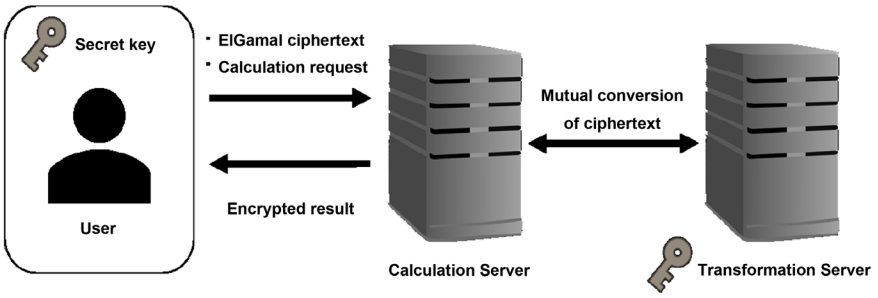

Figure 1. System configuration.

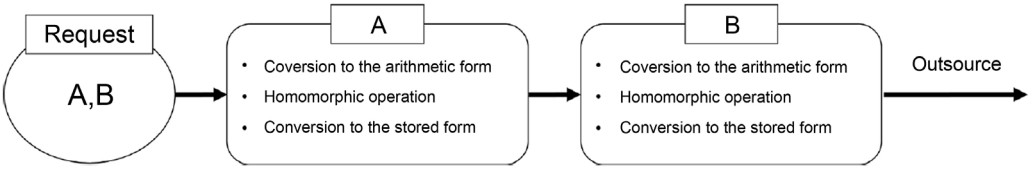

Figure 2. Process flow of the calculation request.

- Encrypt the plaintext and send the encrypted data to the calculation server
- Have a secret key

Calculation Server

- Can not get the plaintext
- When encrypted data are transmitted to the transformation server, they are multiplied by a random number
- Do not collaborate with the transformation server
- Do not have a secret key

Transformation Server

- Can not get the plaintext
- When plaintext are transmitted to the calculation server, it is multiplied by a random number
- Do not collaborate with the calculation server
- Have a secret key

5.3. Homomorphism

In the arithmetic form of ciphertext, we unify the value of r included in ElGamal ciphertext by all ciphertexts. As a result, the arithmetic form of the ciphertext satisfies both additive homomorphism and multiplicative homomorphism.

Given ciphertexts $c_1 = (c_{11}, c_{12}) = (g^r, m_1 g^{xr})$, $c_2 = (c_{21}, c_{22}) = (g^r, m_2 g^{xr})$ where $m_1, m_2 \in G$.

Additive Homomorphism

Compute ciphertext for $m_1 + m_2$ as follows.

$$c_{12} + c_{22} = m_1 g^{xr} + m_2 g^{xr} = (m_1 + m_2) g^{xr}$$

Then, using $c_{11} = c_{21} = g^r$, output $(g^r, (m_1 + m_2) g^{xr})$.

Multiplicative Homomorphism

Compute ciphertext for $m_1 * m_2$ as follows.

$$c_{12}c_{22} = m_1 g^{xr} * m_2 g^{xr} = (m_1 m_2) g^{2xr}$$

Then, compute $c_{11}c_{21} = g^{2r}$ and output $\left(g^{2r}, (m_1 m_2) g^{2xr}\right)$.

5.4. Conversion Processing of Ciphertext

We show the method of mutual conversion between the arithmetic and the stored forms of ciphertext.

Conversion from Stored to Arithmetic Form

Given $(c_{i1}, c_{i2}) = \left(g^{r_i}, m_i g^{xr_i}\right)$ where $i \in \mathbb{Z}_q, m_i \in G$, r_i is a random number:

1) The calculation server generates a random number $\alpha_i \in G$ and sends $(c_{i1}, \alpha_i c_{i2})$ to the transformation server.

2) The transformation server decrypts the received ciphertexts:

$$\frac{\alpha_i c_{i2}}{c_{i1}^x} = \frac{\alpha_i m_i g^{xr_i}}{g^{xr_i}} = \alpha_i m_i$$

3) The transformation server generates ciphertexts from $\alpha_i m_i$ and random number $r \in G$ and send them to the calculation server. r is generated while encrypting $\alpha_1 m_1$, and the same r is used for encryption of $\alpha_i m_i (i = 2, 3, \cdots)$. Also, when multiple processing contents are included in the calculation request, a different r is used for each processing content:

$$(c'_{i1}, c'_{i2}) = \left(g^r, \alpha_i m_i g^{xr}\right)$$

4) The calculation server removes the random number α_i from the received ciphertexts and computes $\left(g^r, m_i g^{xr}\right)$.

Conversion from Arithmetic to Stored Form

Given $(c_{i1}, c_{i2}) = \left(g^r, m_i g^{xr}\right)$ where $i \in \mathbb{Z}_q, m_i \in G$, r is a constant number:

1) The calculation server generates a random number $\beta_i \in G$ and sends $(c_{i1}, \beta_i c_{i2})$ to the transformation server.

2) The transformation server decrypts the received ciphertexts:

$$\frac{\beta_i c_{i2}}{c_{i1}^x} = \frac{\beta_i m_i g^{xr}}{g^{xr}} = \beta_i m_i$$

3) The transformation server generates ciphertexts from $\beta_i m_i$ and random number $r_i \in G$ and sends them to the calculation server:

$$(c''_{i1}, c''_{i2}) = \left(g^{r_i}, \beta_i m_i g^{xr_i}\right)$$

4) The calculation server removes the random number β_i from the received ciphertexts and computes $\left(g^{r_i}, m_i g^{xr_i}\right)$.

6. Experiment

6.1. Overview

In this experiment, as the performance evaluation of the proposed method, statistical processing using homomorphic computation is performed and its processing time is measured. As a comparison target, HElib on which the BGV

scheme of FHE is implemented was used. HElib is an open source library published by IBM and available in C ++. Also, implementation of the proposed method was done in C.

Experiment 1

We measure the processing time of homomorphic addition and homomorphic multiplication. We also measured the processing time of mutual conversion of the ciphertext between stored and arithmetic forms.

Experiment 2

We computed the variance of 1000 - 10,000 data items and measured the processing time. We converted the stored form to arithmetic form, performed statistical processing, and converted the arithmetic form to a stored form. Then, we measured the time taken for this series of flows.

6.2. Experiment Environment

The experimental environment was as follows.

- OS: Ubuntu 18.04.1 LTS
- CPU: Intel(R) Core(TM) i7-4790 CPU @ 3.60 GHz
- Memory: 4 GB
- Compiler: gcc 7.3.0, g++ 7.3.0
- Library: NTL-11.0.0, GMP-5.0.4
- Security: 1024 bit

6.3. Dataset

As an experimental data set, we used the "Adult" labeled dataset provided by UCI. This data set contains 32,561 data items divided by 14 attributes such as age, gender, race, etc. In the experiment, we used the age attribute.

6.4. Results

Experiment 1

We measured the processing time for homomorphic addition and homomorphic multiplication (see Table 1). In homomorphic addition and homomorphic multiplication of HElib, it is not the processing time required for homomorphic operation between packed ciphertexts. It is the processing time of homomorphic operation per ciphertext calculated by dividing the processing time by the number of slots.

Experiment 2

We measured the processing time taken to calculate the variance by homomorphic operations. Table 2 and Figure 3 show the transition of the processing time when the number of data items changes from 1000 to 10,000 in increments of 1000.

7. Discussion

Experiment 1

Table 1. Processing time (homomorphic addition, homomorphic multiplication, and conversion).

Homomorphic Content	Proposed Method (μsec)	HElib (μsec)
Homomorphic Addition	0.4834	0.3579
Homomorphic Multiplication	1.0774	61.2303
Conversion to stored form	28.9767	×
Conversion to arithmetic form	73.7447	×

Table 2. Processing time (variance).

Number of Data	Proposed Method (sec)	HElib (sec)
1000	0.1071	5.7759
2000	0.2164	5.8716
3000	0.3234	5.8882
4000	0.4315	6.0660
5000	0.5336	6.3134
6000	0.6557	6.4208
7000	0.7514	6.6371
8000	0.8538	6.8953
9000	0.9630	7.0232
10,000	1.0736	7.3504

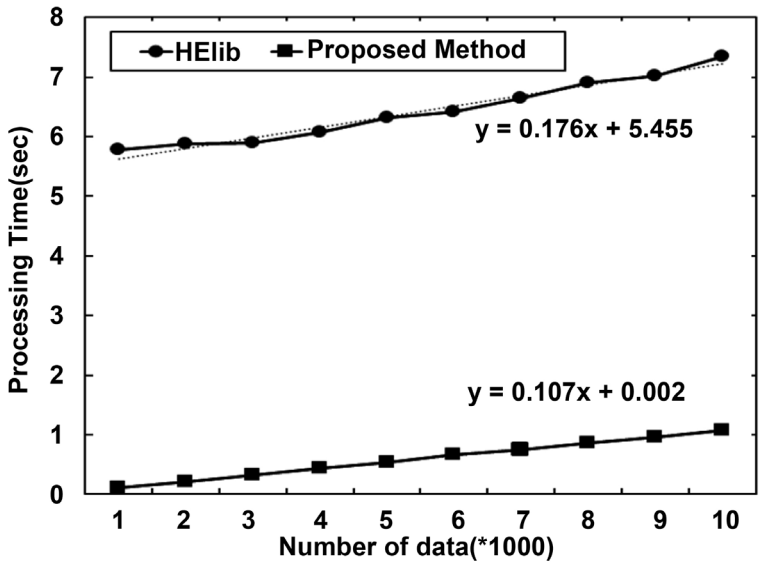

Figure 3. Transition of the processing time (variance).

We measured the processing time for homomorphic addition and homomorphic multiplication and conversion. In the processing time of homomorphic addition, the proposed method required about 135% processing time compared with HElib, but the homomorphic multiplication reduced the processing time to

about 1.8%.

Experiment 2

We measured variance was obtained using 1000 - 10,000 data items and we measured the processing time. In **Figure 2**, the respective approximate straight lines are obtained, where the value of the vertical axis is y and the value of the horizontal axis is x, HElib is represented by $y = 0.176x + 5.455$, and the proposed method is represented by $y = 0.107x + 0.002$.

8. Future Work

We need to improve the security of the proposed method in which we convert from a stored form to an arithmetic form before the homomorphic operation. In arithmetic form, the value of r included in the ElGamal ciphertext $\left(g^r, mg^{xr}\right)$ is unified in all ciphertexts. Therefore, when ElGamal ciphertexts $c_1 = (c_{11}, c_{12}) = \left(g^r, m_1 g^{xr}\right)$, $c_2 = (c_{21}, c_{22}) = \left(g^r, m_2 g^{xr}\right)$ where $m_1, m_2 \in G$, are given, they satisfy the following.

$$\frac{c_{22}}{c_{12}} = \frac{m_2}{m_1}$$

Since the ratio of the plaintext can be obtained from the ratio of the ciphertexts, if any plaintext is deprived in any way, all the plaintexts will leak out. However, when converting to a stored form again and converting it to an arithmetic form from it, the value of r is unified in all ciphertexts, but it can be changed to a value different from the value of r before conversion.

9. Conclusion

In this paper, we propose the acceleration of homomorphic arithmetic processing based on the ElGamal cryptosystem and present experiments, evaluation, and discussion. The results of experiments comparing the proposed method with HElib showed that, although the processing time for homomorphic addition per ciphertext increased by about 35%, the processing time for homomorphic multiplication was reduced to about 1.8%, and the processing time to calculate the statistic (variance) had approximately a 15% reduction.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Paillier, P. (1999) Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. *Proceedings of the EUROCRYPTO'*99, *Lecture Notes in Computer Science*, **1592**, 223-238. https://doi.org/10.1007/3-540-48910-X_16

[2] Rivest, R.L., Shamir, A. and Adleman, L. (1978) A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, **21**, 120-126. https://doi.org/10.1145/359340.359342

[3] Elgamal, T. (1985) A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. *IEEE Transactions on Information Theory*, **31**, 469-472. https://doi.org/10.1109/TIT.1985.1057074

[4] Gentry, C. (2009) A Fully Homomorphic Encryption Scheme. Doctoral Dissertation, Stanford University, Stanford.

[5] Smart, N.P. and Vercauteren, F. (2010) Fully Homomorphic Encryption with Relatively Small Key and Ciphertext Sizes. *PKC*, 420-443. https://doi.org/10.1007/978-3-642-13013-7_25

[6] Brakerski, Z., Gentry, C. and Vaikuntanathan, V. (2011) Fully Homomorphic Encryption without Bootstrapping. *IACR Cryptology ePrint Archive*, *Report* 2011/277.