Analysis of Partially and Fully Homomorphic Encryption

Liam Morris
lcm1115@rit.edu
Department of Computer Science,
Rochester Institute of Technology, Rochester, New York

May 10, 2013

1 Introduction

Homomorphic encryption is a form of encryption that allows for some computations to be performed on the ciphertext without decrypting the ciphertext. The result of the operations is returned as an encrypted result, which when decrypted is the same as if some operation was performed on the plaintext. Some applications for such a system are the implementation of secure voting systems and cloud computing. There are many form of partialy homomorphic cryptosystems that allow for some specific operations to be performed (namely addition and multiplication), but due to some very major drawbacks of fully homomorphic encryption, fully homomorphic encryption is not very practical. Some examples of such drawbacks are processing time and implementation complexity.

Many cryptosystems with homomorphic properties have been around for quite awhile. For example, RSA, Paillier, and ElGamal are partially homomorphic. It was thought that a fully homomorphic cryptosystem was possible, although no one had yet implemented it. The concept of a fully homomorphic cryptosystem was considered the "holy grail of cryptography" for many people. In 2009 the first fully homomorphic cryptosystem was developed by Craig Gentry. Rather than using simple modular arithmetic like most other cryptosystems, Gentry's cryptosystem is lattice-based. While this was very exciting for the field of cryptography, just because such a system is possible does not mean it is practical. The lattice-based cryptosystem presented by Gentry requires a very large ciphertext relative to the plaintext. The fact that it is lattice-based also makes implementation very complex and makes and operations run very slowly on the ciphertext. Additionally, even tuning the cryptosystem such that it exhibits better performance drastically reduces security of the system and even prevents homomorphism in some circumstances [4].

2 Homomorphic Encryption

2.1 General Definition

homomorphism - a transformation of one set into another that preserves in the second set the relations between elements of the first

homomorphic encryption - an operation performed on a set of ciphertexts such that decrypting the result of the operation is the same as the result of some operation performed on the plaintexts

2.2 Technical Definition

Definition 1. Consider a cryptosystem C that has an encryption function ε, plaintext x_n, ciphertext c_n such that $\varepsilon(x_n) = c_n$ and some operation $\triangle$.

C is considered *additively homomorphic* iff:
$\exists\triangle$: $\varepsilon(x_1) \triangle \varepsilon(x_2) = \varepsilon(x_1 + x_2)$

C is considered *multiplicatively homomorphic* iff:
$\exists\triangle$: $\varepsilon(x_1) \triangle \varepsilon(x_2) = \varepsilon(x_1 x_2)$

Note: This definition can be applied to any other operation $\square$ (where $\varepsilon(x_1) \triangle \varepsilon(x_2) = \varepsilon(x_1 \square x_2)$, but for the purposes of this paper only addition and multiplication will be considered.

3 Partially Homomorphic Cryptosystems

A cryptosystem is considered *partially homomorphic* if it exhibits either additive or multiplicative homomorphism, but not both [1]. Some examples of partially homomorphic cryptosystems are:

RSA - multiplicative homomorphism

ElGamal multiplicative homomorphism

Paillier additive homomorphism

3.1 RSA

RSA exhibits multiplicative homomorphism. By multiplying two (or more) RSA ciphertexts together, the decrypted result is equivalent to the multiplication of the two (or more) plaintext values.

Example 1. Consider an RSA key pair (d, e) and modulus n. Recall that the encryption procedure for a message m is

$$c \equiv m^e \bmod n$$

and the decryption procedure is

$$m \equiv c^d \bmod n$$

Given two plaintext messages, x_1 and x_2, the corresponding ciphertext is x_1^e and x_2^e, respectively.

Multiplying the ciphertext together yields $(x_1x_2)^e$. When decrypted, $((x_1x_2)^e)^d \equiv x_1x_2 \bmod n$.

3.2 ElGamal

ElGamal exhibits multiplicative homomorphism. By multiplying each component of multiple ciphertexts with their corresponding respective components, the decrypted result is equivalent to the multiplication of the plaintext values.

Example 2. Consider an ElGamal public key (α, β, p) with private key a.

Recall encryption of a plaintext message x with nonce k to be $\varepsilon(x,k) = (y_1,y_2)$ where

$$y_1 \equiv \alpha^k \bmod p$$
$$y_2 \equiv x\beta^k \bmod p$$

Given two plaintext messages x_1 and x_2 with nonces k_1 and k_2, the corresponding ciphertexts are:

$$\begin{aligned} \varepsilon(x_1,\, k_1) &= (y_1,y_2) \\ &= (\alpha^{k_1} \bmod p,\, x\beta^{k_1} \bmod p) \\ \varepsilon(x_2,\, k_2) &= (y_3,y_4) \\ &= (\alpha^{k_2} \bmod p,\, x\beta^{k_2} \bmod p) \end{aligned}$$

Multiplying the two ciphertexts together yields:

$$\begin{aligned} (y_1,y_2)\cdot(y_3,y_4) &= (y_1\cdot y_3, y_2\cdot y_4) \\ &= (\alpha^{k_1}\cdot\alpha^{k_2}, x_1\beta^{k_1}\cdot x_2\beta^{k_2}) \\ &= (\alpha^{k_1+k_2}, x_1x_2\beta^{k_1+k_2}) \end{aligned}$$

Decrypting the resulting ciphertext yields:

$$d(\alpha^{k_1+k_2}, x_1x_2\beta^{k_1+k_2}) = x_1x_2$$

3.3 Paillier

Paillier exhibits additive homomorphism. By multiplying each component of multiple ciphertexts with their corresponding respective components, the decrypted result is equivalent to the addition of the plaintext values.

Definition 2. The Paillier cryptosystem consists of the following values: Two large primes p and q and $n = pq$. We define $\lambda(n) = \text{lcm}(p - 1, q - 1)$. We choose some value g where $g \in Z^*_{n^2}$ and $L(g^\lambda \bmod n^2)^{-1} \bmod n$ (known as μ) exists. The public key is (n,g) and the private key is (λ,μ) [3].

Function 1. $\mathrm{L}(u) = \frac{u-1}{n}$ [3]

Encryption :

plaintext $\mathsf{m} < \mathsf{n}$

select a random $\mathsf{r} < \mathsf{n}$

ciphertext $\mathsf{c} = \mathsf{g}^m \cdot \mathsf{r}^n \bmod \mathsf{n}^2$

Decryption :

ciphertext $\mathsf{c} < \mathsf{n}^2$

plaintext $\mathsf{m} = \dfrac{\mathrm{L}(\mathsf{c}^\lambda \bmod \mathsf{n}^2)}{\mathrm{L}(\mathsf{g}^\lambda \bmod \mathsf{n}^2)} \bmod \mathsf{n}$

Source: "Public-Key Cryptosystems Based on Composite Degree Residuosity Classes"

Example 3. Consider two plaintext message x_1 and x_2 with corresponding ciphertexts:

$$\varepsilon(x_1,r_1) = g^{x_1}r_1^n \bmod n^2$$
$$\varepsilon(x_2,r_2) = g^{x_2}r_2^n \bmod n^2$$

Multiplying the two ciphertexts together yields:

$$\begin{aligned} \varepsilon(x_1,r_1)\cdot\varepsilon(x_2,r_2) &= g^{x_1}r_1^n\cdot g^{x_2}r_2^n \bmod n^2 \\ &= g^{x_1+x_2}(r_1r_2)^n \bmod n^2 \end{aligned}$$

Decrypting the resulting ciphertext yields:

$$d(\varepsilon(g^{x_1+x_2}(r_1r_2)^n) = x_1 + x_2 \bmod n$$

Example 4. Consider an implementation of a secure voting system using Paillier encryption. Let the total number of votes be represented by a single 4-bit string. Let the first two bits represent the number of votes for candidate A, and the last two bits represent the number of votes for candidate B. When adding a vote for candidate A, add 0100 to the string. When adding a vote for candidate B, add 0001 to the string.

For example, suppose the total number of votes is 1011_2. In this case, the first two bits (10) represent 2 votes for candidate A, while the last two bits (11) represent 3 votes for candidate B.

Consider a Paillier system where $p = 5$, $q = 7$, $n = 35$, $\lambda = 12$, $g = 164$, $\mu = 23$.

Let the following table represent 5 total votes being computed.

Person	Vote	x	r	c
1	A	0100 (4)	6	416
2	B	0001 (1)	31	764
3	B	0001 (1)	17	127
4	A	0100 (4)	22	613
5	A	0100 (4)	11	1191

In this case, the set of ciphertext is:

$$\{127, 416, 613, 764, 1191\} \quad .$$

Multiplying the ciphertexts together yields:

$$127 \cdot 416 \cdot 613 \cdot 764 \cdot 1191 \equiv 509 \bmod 1225$$

Decrypting the resulting ciphertext yields:

$$d(509) = 14 = 1110_2$$

Taking the first two bits (11) indicates that candidate A received 3 votes, and taking the second two bits(10) indicates that candidate B received 2 votes. Comparing these results to the table we can see that this is indeed true.

4 Fully Homomorphic Cryptosystems

A cryptosystem is considered *fully homomorphic* if it exhibits both additive and multiplicative homomorphism [1].

The first (and currently only) such system is a lattice-based cryptosystem developed by Craig Gentry in 2009.

The Gentry scheme relies on a complex mesh of ideal lattices for representing the keys and the ciphertext.

The private key of the system consists of a randomly generated matrix V and an additional matrix W such that:

$$V \times W \equiv c \bmod f(x)$$

where c is a constant [2].

The public key, B, is the Hermite normal form of V, which has the form:

B can be represented by just the integers r and d.

Definition 3. To encrypt some bit b, one must first generate a random noise vector, $\vec{u}$, with values 0 (with probability q) and ± 1 (with each having probability (1 - q) / 2).

$$\text{Let } \vec{a} = 2\vec{u} + b \cdot \vec{e}_1$$
$$\text{Let } \vec{c} = \vec{a} \bmod \text{B}$$

The result of this computation ($\vec{c}$) is the ciphertext [2].

Definition 4. To decrypt some ciphertext $\vec{c}$, one must compute:

$$\vec{a} = \vec{c} \bmod \text{V}$$
$$b = a_0 \bmod 2$$

The recovered bit b is the original plaintext [2].

5 Benefits

Homomorphic encryption has many benefits and applications. One such benefit is that of enhanced privacy. Privacy is one of the goals of cryptography in general, but homomorphic encryption can provide even further privacy than typical encryption schemes. Consider applications in the banking world. Suppose that a customer of a bank has the total value of their accounts encrypted using their private key and that is what is stored on the bank's servers. Without decrypting the customer's account values, things such as interest and transfers could theoretically be computed without ever needing to view the customer's specific dollar amount attached to their accounts. This privacy also can be applied to voting systems. Much like the Paillier example provided earlier, secure voting systems could be implemented such that votes are encrypted and remain unknown until all computations are completed and the results are decrypted. While the previously mentioned example is a very small toy example, it is quite feasible to have a larger system implemented for the purposes of legitimate voting ballots. Cloud computing could see benefits of privacy as well. Similarly to the banking transactions, a user could store their encrypted data in the cloud somewhere. The host could perform computations and return the encrypted result to the user, and the user can decrypt this ciphertext to see various statistics, metrics, or whatever else they're looking for from their data, all while keeping this data hidden from the host.

One great application for homomorphic encryption is that of private information retrieval. Gentry mentions that a user could theoretically encrypt a search query, retrieve encrypted results of the query, then decrypt those results [1]. This process would obscure any data from the search engine while still returning meaningful information to the user. While the search engine is the most obvious of such uses, any information retrieval process could implement homomorphic encryption to protect a user's privacy. One of the biggest benefits to this application is that if a user lives in an area

where privacy is considered a luxury, sensitive data can still be retrieved without ever revealing even the nature of the data.

6 Drawbacks

While the benefits to homomorphic encryption are great, they do not come without considerable drawbacks. One of the biggest drawbacks is the complexity of the systems. In partially homomorphic cryptosystems, there is not much overhead involved in performing the computations, at least for those presented. However, fully homomorphic encryption requires a lattice-based cryptosystem that is significantly more complex. Implementation of such a cryptosystem even for basic operations requires significantly more complicated computations and massive ciphertext sizes. When using recommended security parameters, ciphertexts produced are on the order of 128MB and a public key of 128PB [5]. Ciphertexts and public keys of this size are simply not practical. Even when minimizing security parameters to the point of homomorphism no longer being possible, the key size is still on the order of several GB, with encryption of a single bit still requiring up to 30 minutes [4].

Another potential drawback of homomorphic cryptosystems is that in some cases, they are vulnerable to malware. Consider the case of a secure voting system that implements additively homomorphic encryption. If one of the voting booths were to be infected with malware, it is within the realm of possibility that the malware could manipulate votes before they are submitted. Similar flaws could be applied to nearly any other system that exhibits homomorphism, such as a homomorphic cryptosystem used for banking in which transactions are modified to withdraw or deposit a different amount than the user intended.

7 Conclusions

Homomorphic cryptosystems allow for the same level of privacy as any other cryptosystem, while also allowing for operations to be performed on the data without the need to see the actual data. If a computationally efficient fully homomorphic cryptosystem were to be developed, the implications are phenomenal. Complete privacy between client and server would be possible without any decreased functionality. Such systems could be applied to nearly anything that requires computation, such as voting, banking, cloud computing, and many others.

While the theoretical applications of homomorphic encryption are extremely beneficial and exciting, it is still the case that these systems have not yet matured to the point of being practical. However, these systems are extremely young and there is still a great amount of research that needs to be done in this area. Further installments in this area should prove interesting, and eventually it may be possible that such systems be widely adopted.

References

[1] Gentry, C. *Fully homomorphic encryption using ideal lattices*. 2009.

[2] Gentry, C., Halevi, S. *Implementing Gentry's Fully-Homomorphic Encryption Scheme*. 2011.

[3] Paillier, P. *Public-Key Cryptosystems Based on Composite Degree Residuosity Classes*. 1999.

[4] Smart, N., Vercauteren, F. *Fully Homomorphic Encryption with Relatively Small Key and Ciphertext Sizes*. 2009.

[5] Snook, M. *Integer-Based Fully Homomorphic Encryption*. 2011.