Homomorphic Encryption

Anna Helana Maliakal
Student Number: 17324430

***Abstract*—Every day, information that needs protection is constantly being encrypted both when it is being stored and when it is in transmit. Today, many encryption algorithms are available to secure this information and most of them are virtually unbreakable because of the processing power required to break these systems. However, these algorithms still require you to decrypt the data before it can be processed in any manner, making it vulnerable to anyone with mal-intent. We explore at how homomorphic encryption tackles this issue.**

***Index Terms*—homomorphic, encryption, BFV, RSA, FHE, PHE, CKKS**

I. Introduction

In algebra, homomorphism refers to a map that preserves the structure between two algebraic structures of the same form. Homomorphic Encryption (HE) is an encryption system that allows users to manipulate and perform computations on encrypted data. The encryption and decryption of this system can be compared to a homomorphism between plaintext and ciphertext. No parties can view the results of the computation without the secret key. There is no need to decrypt the data while performing computations. This ensures the utmost privacy and security of the information. The most common types of HE includes:

- Partially homomorphic encryption (PHE): Only permits certain operations to be performed on encrypted values such as multiplication or addition. In order to apply both operations on a plaintext/message, it must be encrypted individually by two different PHE systems. This can be a very expensive process. RSA is an example of a multiplicative homomorphic encryption scheme and Pallier is an example of an additive multiplicative homomorphic encryption scheme. We will be discussing the process of the RSA algorithm.
- Somewhat homomorphic encryption (SHE): Support limited computation to a particular threshold of complexity, these computations can only be carried out a certain number of times.
- Fully homomorphic encryption (FHE): Strongest form of HE. This concept is capable of handling any efficiently computable functions any number of times.

This paper will look closely at FHE and how it came to be while exploring an implementation of PHE. For each implementation, we will look at how it works, the security of the system and applications of the system.

II. History

The investigation into this design of encryption began in 1978 with Ronald L. Rivest and Len Adleman [4], who also were involved in the development of RSA encryption. After no major progress, the topic was raised again by Feigenbaum and Merritt who attempted homomorphic encryption algebraically. However, the systems being developed at this time, were only partially homomorphic cryptosystems.

A. First generation FHE

It was not until 2009, when Craig Gentry, through his seminal paper, that any substance was added to the topic. Making use of lattice-based cryptography [6] , Gentry illustrated the first feasible design for a fully homomorphic encryption scheme. This scheme supported basic arithmetic operations on ciphertexts, such as addition and multiplication. This proved useful when constructing circuits that performed random computation. However, with each operation, the error vector (used to encode the plaintext) grows and eventually this causes a decryption error. In addition to this, the updated implementation of Gentry's system – the Gentry-Halevi implementation – recorded a time of 30 minutes per basic bit operation. A second fully homomorphic encryption scheme was proposed in 2010 by Gentry, Marten van Dijk, Shai Halevi and Vinod Vaikuntanathan that was similar to Gentry's original construction but did not need ideal lattices. This presented a much simpler approach but proved to make no difference regarding efficiency or the operations.

B. Second generation FHE

The homomorphic schemes in use today originated from techniques developed from 2011 by Gentry, Vaikuntanathan, Zvika Brakerski and others. These second-generation schemes still observe the fundamentals of Gentry's original architecture – creating a somewhat homomorphic cryptosystem and then evolving it to fully homomorphic encryption using bootstrapping. These techniques allowed for somewhat more efficient and fully homomorphic systems including the Brakerski/Fan-Vercauteren scheme (which builds on Brakerski's scale-invariant cryptosystem) and Cheon-Kim-Kim-Song (CKKS) scheme (which uses a new rescaling method for dealing with the magnitude of plaintext). The additional optimizations of the systems of the second-generation allowed for much slower growth of the error vector during the computations. Another notable characteristic of these schemes is that they are satisfactory in their efficiency for many applications without invoking bootstrapping.

C. Third generation FHE

In 2013, a new technique was proposed for fully homomorphic encryption which avoided an expensive step of "relinearization" in homomorphic multiplication. This system (known as the GSW system after Gentry, Amit Sahai and Brent Waters) was observed to have an even slower rate of growth on the error vectors for certain types of circuits. This proved to be more efficient and provided stronger security. Based on these observations, others were able to explore a very efficient bootstrapping technique. These were then improved to construct efficient ring variants of the GSW system: FHEW (2014) and TFHE (2016). FHEW explained how it was possible to reduce the bootstrapping time by refreshing the ciphertext after every computation. It then presented a new technique, used in the TFHE scheme, to evaluate Boolean functions with a proportional noise overhead and bootstrap the full using only a linear noise overhead. Today, the most popular schemes include:

- **TFHE:** used to perform computations on either manually created circuits or with the output of automated circuit generation tools [3]
- **BFV(Brakerski/Fan-Vercauteren):** performs exact arithmetic on vectors of numbers (only integers).
- **CKKS (Cheon-Kim-Kim-Song):** performs approximate arithmetic on vectors (can be performed on complex numbers with limited precision)

We will look at BFV and CKKS later on in V and VI, respectively.

III. Cryptography Primitives

If there exists an encryption function E, $E(m1)$ denotes the cipher texts created after the encryption of the message, $m1$. E is homomorphic about $*$ on messages M, if there exists a function on ciphertexts, !, and decrypting $E(m1)!E(m2)$ is equal to $m1 * m2$. A scheme is known as additively homomorphic if $(M, *)$ is an additive semigroup. Otherwise, if $(M, *)$ is multiplicative semigroup, then it is known as multiplicatively homomorphic. A homomorphic encryption scheme on M is a quadruple (KeyGen, Enc, Dec, Eval) of probabilistic - polynomial time algorithms, satisfying the following functionalities:

- **KeyGen – Key generation** Since this is a public-key based system, the algorithm that generates the key take in a security argument as the parameter and outputs the keys (PK, SK) – an encryption/decryption key pair, where PK denotes the public key and SK denotes the secret key, which only the data owner holds. It also outputs EK, an evaluation key needed for manipulating the data.
- **Enc – Encryption** The algorithm that performs the encryption takes plaintext, or the message, and converts it to ciphertext. The function takes π and key_1 as arguments, where π is the plaintext in bit or integer form, and key_1 is either the PK – for a public cryptosystem or K – for symmetric system.
- **Dec – Decryption:** The algorithm performing decryption reverts ciphertext to plaintext. The function takes c and key_2 as parameters, where c is the ciphertext and key_2 is either the PK – for a public cryptosystem or K – for symmetric system.
- **Eval – Evaluation:** The evaluation algorithm outputs the result of a computation x on the ciphertexts involved. The function takes x, the ciphertexts and EK as parameters. The use of the evaluation key, EK, is optional.

Encryption schemes can be probabilistic or deterministic. A probabilistic scheme takes a random number q as an additional input for the algorithm E and a deterministic scheme does not. A probabilistic scheme, conditioned on the random number q, can have several ciphertexts for one message. The decryption scheme remains deterministic, meaning that given a ciphertext it maps to only one message The methods described below are used in key generation by HE schemes.

A. Learning with Errors (LWE)

With LWE, a random matrix A, a secret matrix s, and an error matrix e is used. We calculate a vector B by multiplying A and s and then adding e as demonstrated below:

$$B = A * s + e \tag{1}$$

The values of A and B can be used to form our public key and the value of s is used to form the secret key. The addition of the error matrix e, makes it more difficult to break a scheme.

B. Ring Learning with Errors (RLWE)

RLWE is similar to LWE, however instead of matrices, we now use coefficients of polynomials. Just like described above, a random polynomial A, a secret polynomial s, an error polynomial e, and the highest degree in the polynomial n, are used to get the generate keys. We create a ring by:

$$Y = x^{n+1} + 1 \tag{2}$$

$$BA_1 = A/Y \tag{3}$$

$$BA_2 = BA_1 * s \tag{4}$$

$$BA_3 = BA_2/Y \tag{5}$$

$$BA_4 = BA_3 + e \tag{6}$$

$$B = BA_4/Y \tag{7}$$

IV. RSA

PHE algorithms, like mentioned above, can only execute either multiplication or addition at a time. The Rivest-Shamir-Adleman is a multiplicative homomorphic encryption scheme. The RSA cryptosystem is one of the world's first public-key encryption scheme (also referred to as asymmetric encryption).

RSA encrypts messages with a public key, which can be shared. However, due to its mathematical properties, the message can only be decrypted with a private key, which must be kept secret. This differs from symmetric encryption where the encryption and decryption algorithm both use the same private key. Complicated messages would involve the use of much larger numbers.

A. Key Generation

1) Choose two relatively large prime numbers, p and q. Multiplying both these values give the modulus, which we call n.
2) Calculate a value PHI which is $(p-1)*(q-1)$
3) Select a value for the encryption key, PK, so that it is relative prime to PHI
4) Let d equate to the decryption key/private key. We calculate d through the formula:

$$(d*e)mod(n) = 1. \tag{8}$$

B. Encryption/Decryption

Let M equate to the message and E equate to the encrypted message

To encrypt use the formula, $E = M^e mod(n)$

To decrypt use the formula, $M = E^d mod(n)$

C. HE Multiply

As mentioned above, RSA is a multiplicative homomorphic encryption scheme, allowing us to perform the operation of multiplication. To multiply two values V_1 and V_2:

1) Encrypt the two values individually creating:

$$C1 = V_1^e mod(n) \tag{9}$$

$$C2 = V_2^e mod(n) \tag{10}$$

2) Multiply both these values:

$$C1*C2 = V_1^e * V_2^e mod(n) = (V_1*V_2)^e mod(n) \tag{11}$$

3) Decrypt using:

$$C1*C2^d mod(n) \tag{12}$$

D. HE Division

RSA can also perform division on two values V_1 and V_2:

1) Encrypt the two values creating:

$$C1 = V_1^e mod(n) \tag{13}$$

$$C2 = V_2^e mod(n) \tag{14}$$

2) Divide both these values:

$$C1/C2 = V_1^e/V_2^e mod(n) = (V_1*V_2^{-1})^e mod(n) \tag{15}$$

3) Decrypt using:

$$C1/C2^d mod(n) \tag{16}$$

E. RSA Padding

The structure of an encrypted message can allow attackers to detect patterns and compromise the encryption system. Padding allows random information to be inserted into the message to hide formatting clues. Since RSA does not feature an obvious formatting system, padding schemes such as OAEP are used to add extra data into the encrypted message. This prevents partial decryption of the messages.

F. Security

Unlike symmetric-key algorithms like AES, RSA algorithms can be broken through integer factorization. Therefore, in order to attain the same amount of security, RSA keys need to be sufficiently long. The largest key size factored to date is 240 decimal digits long or 795 bits. However, with the notable growth of computing power, we can presume that an attempt of the same intensity in the future, could factor a significantly larger RSA key. The length of the key should be determined by the security level required by a user. However, it is recommended that a key should have a minimum size of 2048 bits.

Another concern that arises is the fact that some RSA executions use random number generators that are quite weak. Attackers can easily factor keys that are not sufficiently random and can compromise the system. To avoid this problem, a pseudo-random number generator that is cryptographically secure can be used.

In addition to this, if the selected prime numbers, p and q, are too close to each other, the RSA keys can be easily discovered. It is recommended that p–q should be greater than $2n^{1/4}$ to make it more difficult for attackers to find these values. It is also advised that PHI should be a relatively large value, otherwise n can be factored quickly by Pollard's $p-1$ algorithm.

Side channel attacks also present a threat. These attacks may measure execution time or analyse the amount of power used to find the private key. A timing attack is a type of side channel attack that measures the amount of time it takes to decrypt a number of unique messages. This information can then make it possible to work out the private key used in the algorithm. This attack can be avoided by adding a random value which removes the correlation

G. Application

RSA is not usually used for encrypting files as it is comparatively slow. Therefore, files are usually encrypted symmetrically and only the key is encrypted by RSA. This is known as hybrid encryption. RSA is also often used in email, messaging applications, VPNs and web browsers – any system that require sending data in a secured manner.

H. Future Trends

Today, RSA is considered safe to use, despite the possible attacks. If the recommended guidelines are followed, the encrypted data can be considered secured. However, with the development of quantum computers, it could mean that in the future, RSA might not be a viable option. Quantum computers, because of their intense computing power, could quickly break RSA. Only after taking this into consideration and determining the security level needed by the data we want to protect, should we use RSA.

V. BFV

BFV refers to the Brakerski/Fan-Vercauteren scheme that buildings on Brakerski's scale invariant cryptosystem. This scheme is based on the Ring Learning with Errors (RLWE) (a computational problem that makes use of polynomial rings) and so is designed to withhold attacks via quantum computing.

They make use of relinearization, which is the method used to convert quadratic equations to linear equations. This calculates homomorphic multiplications over algebraic structures that could only calculate homomorphic additions. It can also reduce the size of the ciphertext after multiplication.

A. ArgGen $(\lambda, PT, K, B) \rightarrow Arguments$

This algorithm instantiates arguments used in the algorithms below.

λ = the security level desired from the scheme. For example, 256-bit, $\lambda = 256$

PT = denotes one of two types of parametrized plaintext spaces: modular integer (MI) or extension fields/rings (EX). MIs are parametrized by the modulus p of the numbers to be encrypted (p should be greater than 1), indicated as Z_p meaning that each value of the message space is between the range of 0 and $p-1$, and all computations on the values are performed modulo p. EXs, in addition to being parameterized by p, are also parameterized by a polynomial $f(x)$ over Z_p, denoted as $Z[x]/(p, f(x))$. All the values in the message should be a polynomial that is a degree smaller than $f(x)$ with co-efficient from the range $(0, p-1)$.

K = the number of vectors to be encrypted

B = The argument used to control the complexity of the programs to perform on encrypted messages. This value is proportional to how expensive the algorithm is going to be and the size of the keys involved.

In BFV, the arguments include the ciphertext modulus argument q, a ring $R = Z[x]/f(x)$ along with its corresponding rings, a key distribution $D1$ and an error distribution $D2$ over R, a bit-decomposition modulus T, $L = log_T q$ and $W = |q/p|$

B. KeyGen (Arguments) $\rightarrow PK, SK, EK$

The secret key, SK, needed for decryption, is composed of a random value from the distribution D_1. The public key, PK, is calculated by:

$$PK = (-(as + e), a) \tag{17}$$

where s is a value from the ring, R, a is a value from R/qR and e is an error value from D_2. The evaluation key, EK, is calculated by

$$EK_i = (-(a_i s + e_i) + T^i s^2, a_1) \tag{18}$$

where a, e and s are defined as above and i ranges from a value between 0 and L. Therefore, $EK = (EK_1 .... EK_L)$

C. Enc $PK, M \rightarrow C$

Before encrypting the message M, it is mapped onto an element in the ring R/pR.

$$C = (pk_0 u + e_1 + WM, pk_1 u + e_2) \tag{19}$$

where pk_0 and pk_1 is obtained by parsing PK, u is sampled from D_1, and e_1 and e_2 are sampled from D_2.

D. Dec $SK, C \rightarrow M$

The message can be decrypted by manipulating the formula:

$$C(s) = WM + e \tag{20}$$

for some error, e. W is divided by $C(s)$, with each integer being rounded to its nearest value, and using the coefficient modulo p to reduce each value.

E. Addition $EK, C1, C2 \rightarrow C3$

Each ciphertext is parsed as

$$C_i = (c_{i,0}, c_{i,1}) \tag{21}$$

To add two ciphertexts, use the formula

$$C3 = (c_{1,0} + c_{2,0}, c_{1,1} + c_{2,1}) \tag{22}$$

F. Multiplication $EK, C1, C2 \rightarrow C3$

First, calculate:

$$C3' = (c_{1,0} * c_{2,0}, c_{1,0} * c_{2,1} + c_{1,1} * c_{2,0}, c_{1,1} * c_{2,1}) \tag{23}$$

Then,

$$C3 = round((pq)C3')mod(q) \tag{24}$$

G. Security

BFV is still under risk of timing attacks mentioned in IV-F. To prevent any leak of information an upper bound limit, T, can be set for each operation. When a sample is being obtained, it has to be performed in less than T time. If an operation takes less than T to complete, the program should wait for T–t, where t is the time it took for the operation to end. This way, all computations, if completed, complete on the same time, not allowing for any information to be obtained from this.

BFV is IND-CPA secure. This means that it provides indistinguishability under chosen-plaintext attack (CPA). A CPA performs cryptanalysis, which allows the attacker to retrieve information about the system and decrease the security provided. This ensures that given PK, the attacker only has a slight advantage over random guessing to breaking the system.

However, BFV is not IND-CCA secure. This means that, if the attacker gains access to a decryption oracle (used to decrypt random ciphertexts), the system can be compromised. This shows that there is room for many security pitfalls. In a scenario where the data-owner accidentally leaks partial information, the Cloud will be given the opportunity to break the CPA model and gather more information.

H. Application

To date, BFV has been used to develop machine learning algorithms that intend to improve homomorphic algorithms. However, the outcome of this was very similar to Microsoft's SEAL's implementation of BFV, which removed the need for relinearization. However, the implementation found in SEAL was used in recognising data from encrypted pictures using neural networks. BFV has also been used for precise arithmetic on vectors of numbers.

There are limited uses for BFV as high-secured calculations, which feature significantly large keys, are very expensive to calculate. Also, it does not satisfy IND-CCA, and this raises many concerns.

VI. CKKS

Unlike BFV, which mainly supports discrete arithmetic, the Cheon-Kin-Kim-Song (CKKS) scheme [2], is used to take advantage of approximate computation. This fairly new scheme, first developed in 2017, is another example of FHE that presents a more efficient method of approximate computation on HE. The main idea here is that an encrypted message C has some error value, e, inserted to guarantee security and e is small enough to avoid destruction of the original message.

With other HE schemes, there is a problem of the message bit-size increasing exponentially to the depth of a circuit without rounding. A new technique called rescaling was introduced to tackle this issue. This would require the message to be encoded to plaintext before being encrypted to ciphertext.

A. Encoding/Decoding

CKKS encodes and decodes by taking advantage of a ring isomorphism

$$\phi : \mathbb{R}[X]/(X^n + 1) \rightarrow \mathbb{C}^{n/2} \tag{25}$$

To encode a message, the above isomorphism, along with a scaling factor and a coefficient-wise rounding function is used and the message is output as a polynomial.

To decode a polynomial message, compute:

$$\vec{z} = \Delta^{-1} \cdot \phi(m(X)) \in \mathbb{C}^{n/2} \tag{26}$$

The scaling factor Δ is used here to control the error consequent of the rounding process.

For the algorithms, let $R_q := R/qR$ be the quotient ring of R modulo q, where q is a positive integer and let X_s, X_R, X_e be distributions over R that calculate polynomials with small coefficients. An initial modulus Q, the ring dimension n and the above distributions are calculated before $KeyGen(\lambda)$.

B. KeyGen (λ) $\rightarrow PK, SK, EK$

1) Sample a polynomial s fromX_s
2) Sample a randomly from R_Q and $e1$, $e2$ from X_e
3) SK is calculated from the s, and PK and EK is calculated using the above arguments selected.

C. Enc(m)$\rightarrow$ c

Given an encoded message as a polynomial m, this function will output a ciphertext c. This ciphertext will satisfy

$$c = (c_0 = r * b + e_0 + m, c_1 = r * a + e_1), \tag{27}$$

where r is sampled from X_R.

D. Dec(c)$\rightarrow m'$

Given a ciphertext c, output a polynomial.

$$m' \leftarrow \langle c, SK\rangle mod(q) \tag{28}$$

When decoded an approximation of the original message m is received. The approximation error can be calculated given the samples from the distributions X_s, X_R, X_e.

E. Add(c1,c2) $\rightarrow c3$

Given two ciphertexts $c1$ and $c2$, output $c1 + c2$ as $c3$.

F. Multiply(c1,c2)$\rightarrow$ c3

1) Parse each ciphertext so that $c1 = (c_{1,1}, c_{1,2})$ and $c2 = (c_{2,1}, c_{2,2})$.
2) Compute
$$d_0, d_1, d_2 = (c_{1,1} * c_{2,1} + c_{1,2} * c_{2,2}) mod(q) \tag{29}$$
3)
$$c3 = (d_0, d_1) + \lfloor P^{-1} \rceil * d_2 * EK(mod(q)), \tag{30}$$
where P is an integer based on the security parameter λ

G. Security

Like BFV, CKKS, is only IND-CPA secure, meaning that if data is leaked by the data owner the encryption system could be compromised.

To the plus side, because of this scheme's use of rescaling, it is able to preserve the integrity of the message after an approximate calculation. It has also proved to be the most preferred solution for the computation of large integers due to the reduction in size of the encrypted text.

H. Application

The main usage of this scheme is for more efficient (less costly) approximate arithmetic of calculated values, without the loss of precision of information. CKKS has also been involved in privacy preserving machine learning [5]. However, since this is a very young scheme, new applications are only being developed using this scheme as it is still being explored for improvements.

VII. Applications for FHE

Though we have looked at the above implementations of FHE and the ways these schemes have been applied, one might wonder the application of this encryption in our day-to-day life. Researchers of this topic have outlined the following applications:

- **Protecting Data Stored in the Cloud:** HE allows data to be securely stored in the cloud, while giving the opportunity for computations and searches to be performed on ciphered-text and retaining the precision of the data. [1]
- **Enabling Privacy in Data Analytics Industry:** User data is very valuable in today's world for market research and developing technologies. However, this is not exactly

protecting a customer's privacy of their data. HE can allow companies to perform data analytics while ensuring that a customer's data is not at risk. This will prove use for medical data analysis, personalised advertising and image recognition discussed in V-H.

- **Improving Election/Vote Systems:** HE can be used to tally votes while keeping their values private and secured from manipulation.

These applications are just an insight into the many ways it could improve a person's daily life and develop society.

VIII. Future Trends

While HE has been successful in scenarios, it still requires major improvements to be used in the above examples and regularly used in our daily lives. Researchers are now focusing on two main limitations with HE:

- Inability to handle multiple users: If there are many users in the one homomorphically encrypted system, then the provider of this system would create a separate database for each user and in a system with many users this system would not be practical
- Inability to handle large computations: Although the efficiency of FHE schemes has improved since 2009, it is still much slower than running the same calculations in a non-encrypted environment. This puts users into a position of having to decide between securing their data and achieving results after a much longer period or fast results and unsecured data.

IX. Conclusion

Since Rivest and Adleman's first look into the discussion of homomorphic encryption and Gentry's first theoretical development of the subject, FHE has grown from a concept to a reality. However, while FHE seems as an ideal solution for privacy concerns, it is still very much in the making. With the limitations it presents, FHE is not at all practical to be used in systems that we currently use every day. Given that quantum computing is not easily attainable, we can rest assured that our data can be heavily protected by PHE schemes such as RSA. However, with the rate of development of quantum computing, more investments should be placed in tackling these limitations and hence, providing a more secured society.

References

[1] Tebaa, M., El Hajii, S., "Secure cloud computing through homomorphic encryption", International Journal of Advancements in Computing Technology(IJACT), vol. 5, pp. 33–37, December 2013 .

[2] Cheon, Jung Hee; Kim, Andrey; Kim, Miran; Song, Yongsoo (2017). "Homomorphic encryption for arithmetic of approximate numbers". Takagi T., Peyrin T. (eds) Advances in Cryptology – ASIACRYPT 2017. ASIACRYPT 2017. Springer, Cham. pp. 409–437.

[3] Ilaria Chillotti; Nicolas Gama; Mariya Georgieva; Malika Izabachene. "Faster Fully Homomorphic Encryption: Bootstrapping in less than 0.1 Seconds". Retrieved 31 December 2016

[4] Rivest, R.L; Shamir,A; Adleman L, "A method for obtaining digital signature and public key cryptosystems", Communications of the ACM, vol. 21, February 1978.

[5] Phong, Le Trieu; Aono, Yoshinori; Hayashi, Tayuka; Wang, Lihua; Moriai, Shiho; "Privacy-preserving deep learning via additively homomorphic encryption", ATIS 2017

[6] Regev, Oded; "Lattice-based cryptography", CRYPTO 2006, LNCS 4117, pp. 131-141