Homomorphic Encryption

Saadiq, Rushil and Krishna
Euler Circle

December 9, 2019

Abstract

This paper introduces Homomorphic Encryption Schemes as devices for enabling remotely performed computations on encrypted data. We first lay down some groundwork to ensure the reader understands the mathematical idea of a homomorphism and the cryptographical ideas of encryption and decryption. We then use examples to provide motivation for the Homomorphic Encryption Schemes. This paper also presents a few examples of Homomorphic Encryption Schemes, and proposes how these could be improved. Finally, the paper will discuss the modern development of Fully Homomorphic Encryption and Bootstrapping methods. Finally, we address the practicality of Homomorphic Encryption and Fully Homomorphic Encryption, and how such schemes will evolve in the future.

1 Homomorphic Encryption Functions

A Homomorphic Encryption (HE) scheme involves four essential operations: the *KeyGen* operation, the classical *Enc* and *Dec* operations, and the *Eval* operation. The first three are operations involved in classical encryption schemes, but *Eval* is specific to Homomorphic Encryption schemes. To perform encryption, a public key *pk* is used, while a secret key *sk* is used for decryption.

Definition 1. The *KeyGen* operation returns a key that is used to decrypt the ciphertext into plaintext. (A good example of a KeyGen operation is the Diffie-Hellman Key Exchange, which is a crucial stepping stone for cryptosystems like RSA.)

Definition 2. *Enc*(*pk*, c) is the encryption function, that encrypts the plaintext m and turns it into a ciphertext c using some public key *pk*.

Definition 3. *Dec*(*sk*, c) is the decryption function that decrypts some ciphertext c into its plaintext m using some secret key *sk*.

Before we discuss the *Eval* function, we must first describe what it means for an encryption scheme to be *homomorphic*. Homomorphic encryption borrows the idea of a homomorphism from abstract algebra in that a computer doing some operations (evaluating circuits) on ciphertexts can be translated into plaintexts with operations done on them (using the decryption function), though a homomorphism doesn't necessarily need to be involved.

Definition 4. A *Group Homomorphism* from $(G_1, *)$ to $(G_2, \cdot)$ is a function f such that

$$\forall m, n \in G := f(m \cdot n) = f(m) \cdot f(n).$$

We shall name the character who is attempting to retrieve results from her data *Alice.* Alice is storing her encrypted data in a remote location (say, the Cloud). *Bob*, on the other hand, oversees and assists actions that are being made on the data in the remote location (in this sense, Bob is the one who runs the Cloud). In most homomorphic encryption schemes, Alice will be sending a query to Bob, who will perform Alice's operation on the *encrypted* data, and send the *encrypted* results back to Alice. Alice can then perform her decryption function on Bob's message to retrieve her result.

Definition 5. *Eval* is the homomorphic equivalent of the function f that Alice wishes to compute on her data. To better understand the *Eval* operation, we consider the following scenario. Suppose that Alice has a set of encrypted data that she stores on the Cloud. She wishes to perform some function on that set of data (perhaps a "search" query), but as she does not have it readily available, she must ask Bob, who runs the Cloud server, to compute the function for her. However, Bob only has access to Alice's *encrypted* data, which may not cooperate well with Alice's original search query. So, Alice must supply the Cloud server with a *homomorphism* of her search function, namely the *Eval* function, which will cooperate well with encrypted data. The Cloud server then applies the *Eval* function, and sends Alice the encrypted results of her query, who then decrypts the message to retrieve her search results. The beauty of homomorphic encryption schemes is that the third-party Cloud server computed operations on Alice's data while blindfolded! In a sense, homomorphic encryption allows Alice to remotely perform operations on encrypted data without having to disclose her results to the third party.

For a concrete example of the *Eval* function, let's say our set of messages is $(Z_{p-1}, +)$. Assume we know a generator g in $(\mathbb{F}_p \setminus \{0\}, *)$ and we define our encryption function as $E(n) = g^n$ to take advantage of the group homomorphism $E(r+s) = E(r) * E(s)$. Then if we wanted to add two of our plaintexts m_1 and m_2, we could send $E(m_1)$ and $E(m_2)$ to Bob and ask him to compute $E(m_1) * E(m_2)$. Assume we knew how to compute discrete logs for the sake of this example; then we could take the discrete log of Bob's result with base g, and we obtain the result $m_1 + m_2$. Then in this scheme, our *Dec* function would be the discrete log and the *Eval* function would be multiplication. Our *KeyGen* function could then give us the prime p.

2 Some Helpful Definitions

This section will lay some groundwork for the sections that follow.

Definition 6. A *Circuit* in the context of cryptography is essentially the same as the typical logic circuit, in that it is a function built with some configuration of *Logic Gates*, which represent Boolean operators (NAND, XOR, AND, NOT). In this sense, a Circuit is a construction built from Boolean functions that are composed in a specific way in order to produce a certain binary output given one or more binary inputs.

Definition 7. A *Single-Hop Homomorphic Encryption Scheme* (SHHE scheme) is one that allows encrypted data to work well with one evaluation, but not necessarily with two or more. This means that a Single-Hop Homomorphic Scheme would not allow Alice to perform *two* operations (say a sort and then another operation on that sorted data) on her data. In other words, a SHHE scheme does not allow the user to re-apply the homomorphic *Eval* function to already-evaluated ciphertexts.

Definition 8. A *Multi-Hop Homomorphic Encryption Scheme* is one that allows for *multiple* iterations of homomorphic evaluation functions, and in this sense, is better than a SHHE scheme.

The conversion of Single-Hop HE Schemes to Multi-Hop HE schemes is accomplished with *Bootstrapping* methods, which are discussed in further detail later in the paper.

3 Attributes of Homomorphic Encryption

Definition 9. (Correct Decryption). A HE scheme *correctly decrypts* if, for all plaintexts π in the plaintext space $\mathcal{P}$,

$$\Pr[\mathrm{Dec}(sk, \mathrm{Enc}(pk, \pi)) = \pi] = 1,$$

where Pr is the probability function. In other words, we can guarantee that, given a ciphertext encrypting plaintext π with public key pk, decrypting it with the secret key sk corresponding with the public key pk will always give the correct result. This may seem like an obvious and useless definition at first sight, but this becomes something important to keep in mind when we consider the addition of a noise term when encrypting to deal with the issue of malleability.

Definition 10. (Correct Evaluation). A HE scheme *correctly evaluates* all circuits $\mathcal{C}$, if for all $C \in \mathcal{C}$, all $c_i \in \mathcal{X}$ (the ciphertext space) that encrypt m_i in the plaintext space, and a negligible function ε, we have

$$\Pr[\mathrm{Dec}(\mathrm{Eval}(evk, C, c_1, ..., c_k)) = C(m_1, ..., m_k)] = 1 - \varepsilon.$$

Once again, circuits can just be thought of as a sequence of operations to be performed on data. The evaluation key *evk* of a circuit C is the homomorphic equivalent of the circuit, that produces the predicted result on the ciphertexts such that when we decrypt the evaluated ciphertexts we get the circuit C evaluated on the plaintexts. Correct Evaluation, like Correct Decryption, is a seemingly frivolous definition that becomes important when considering the intentional sources of error resulting from evaluating circuits in a secure manner.

4 Homomorphic Encryption Schemes

Definition 11. Multiplicative Homomorphic Encryption Scheme

Let $\mathbb{F}^* = \mathbb{F}_q \setminus \{0\}$ and $\mathbb{Z}^*_{q-1} = \{k \in \mathbb{Z}_{q-1} | \gcd(k, q-1) = 1\}$, where q is the power of a prime. For a positive integer n, let η be a primitive element of $\mathbb{F}^*$, where a primitive element is a generator of the multiplicative group of the field. Then, $\beta = \eta^{\frac{q^n-1}{q-1}}$ is a primitive element of $\mathbb{F}_q$.

We now demonstrate how to generate the key for a multiplicative homomorphic encryption scheme.

Choose a positive integer d such that $d|(q^n-1)/(q-1)$ and $\gcd(d,q-1)=1$, and choose $l \in \mathbb{Z}^*_{q-1}$. The tuple (d,l) is the secret key.

Encryption: Let $\alpha = \eta^{\frac{q^n-1}{d}}$, which is a primitive d-th root of unity over $\mathbb{F}_q$. In order to encrypt a plaintext $m \in {\mathbb{F}_q}^*$, one chooses a random $r \in 0,1,....,d-1$ and computes the ciphertext as

$$c = \gamma^{log_\beta m}\alpha^r,$$

where $\gamma = \eta^{l(q^{n-1}/d(q-1))}$, the discrete logarithm $\log_\beta(m) = a$ if $\beta^a = m$.

Decryption: For $c \in \mathbb{F}^*_{q^n}$, one computes

$$m' = c^{d \cdot l^{-1}},$$

where l^{-1} is the inverse of l in $\mathbb{Z}^*_{q-1}$.

(Proof) We can prove that this scheme does indeed work by showing that the decryption of the encrypted plaintext results in the original plaintext. To decrypt the ciphertext $c = \gamma^{log_\beta m}\alpha^r$, we look at

$$m' = c^{d*l^{-1}} = \gamma^{log_\beta m^{d*l^{-1}}}(\alpha^r)^{d*l^{-1}} = (\gamma^d)^{l^{-1}*log_\beta m}(\alpha^d)^{r*l^{-1}} = \beta^{l*l^{-1}*\log_\beta m} = m.$$

This uses the fact that $\gamma^d = \beta^l$ and $\alpha^d = 1$. Then, to prove the multiplicative part of this scheme, we can look to the fact that the decryption function $m' = c^{d*l^{-1}}$ is a power function. For instance, let the ciphertexts of m_1 and m_2 be c_1 and c_2 respectively. This means that the decryption of $c_1 * c_2$, for instance, results in

$$(c_1 * c_2)^{d*l^{-1}} = c_1^{d*l^{-1}} * c_2^{d*l^{-1}} = m_1 * m_2.$$

Definition 12. Additive Homomorphic Encryption Scheme

Let q be a prime power and n a positive integer. Also let

$$F(x) = \sum_{i=0}^{n-1} \delta_i x^{q^i} - \alpha$$

be a q^n-ary affine function, where $\alpha \in \mathbb{F}_{q^n}$ and $\delta_i \in \mathbb{F}_{q^n}$, $i = 0,....,n-1$. Then, an element $\beta \in \mathbb{F}_{q^n}$ is a root of $F(x)$ if and only if $F(\beta) = \alpha$.

Key Generation Choose $\alpha \in \mathbb{F}^*$ as the secret key. Define a q^n-ary affine function $F(x) = {Tr_1}^n(\alpha x)$

Encryption To encrypt a plaintext $m \in \mathbb{F}_q$, one randomly chooses a root $c \in \mathbb{F}_{q^n}$ of the affine q-polynomial $F(x) - m$. Then, c is the ciphertext.

Decryption For $c \in \mathbb{F}_{q^n}$, one computes $m' = F(c)$.

(Proof) To prove that the above scheme is indeed additive, we have to utilize the fact that the trace function is linear. For example, decrypting $c_1 + c_2$ gives $F(c_1+c_2) = {Tr_1}^n(\alpha(c_1+c_2)) = {Tr_1}^n(\alpha c_1) + {Tr_2}^n(\alpha c_2) = F(c_1) + F(c_2) = m_1 + m_2$.

5 Noise Reduction in Homomorphic Encryption

One issue of homomorphic encryption is that of malleability; because such encryption schemes require changes in data to be predictable so computation can be outsourced, the person doing those computations can potentially alter the encrypted data and predict what the result will be. In certain situations where important data is being passed, like bank transactions, an attacker with knowledge of the kinds of functions that would be use to encrypt the data could potentially alter the data in ways they can predict, like changing the transaction amount. To safeguard against this, we add a noise term when encrypting data in homomorphic encryption schemes. For this to happen, the decryption function must be able to "see through" the noise.

However, a problem arises when Bob operates on the encrypted data. The noise parameter can grow in size beyond the range from which the decryption function can decrypt it. We want Bob to be able to operate on the encrypted data as much as possible: in other words, we want a Multi-Hop Encryption scheme. To solve this, we employ the technique of Bootstrapping.

5.1 Bootstrapping

Definition 13. A *Somewhat Homomorphic* Encryption scheme is an extension of a *Single-Hop* Homomorphic Encryption scheme in that it can evaluate multiple circuits on data, but only a finite amount.

Definition 14. A *Fully Homomorphic* Encryption scheme is a *Multi-Hop* encryption scheme that can evaluate any amount of circuits and still be able to correctly decrypt.

The first fully homomorphic encryption scheme was found by Craig Gentry using the concept of ideal lattices. The actual scheme is very long and complicated, so we restrict our attention to the very novel idea of bootstrapping that Gentry uses in the context of his encryption scheme.

From now on "we" refers to Bob, or the party that the computation is outsourced to. We start by assuming we have a noisy ciphertext c encrypting a message m using a secret key sk. By noisy, we mean that if we evaluate any more circuits on (adding more error/noise), we cannot guarantee Alice can correctly decrypt it and obtain the desired message m (refer to the definition of Correct Decryption). To accomplish this, we try to "refresh" the ciphertext and remove the noise.

We know that we have a decryption function *Dec(sk,c)* that decrypts any ciphertext c using the secret key sk associated with the public key pk used to encrypt the plaintext. Normally, we see this as a function of the ciphertext for some secret key. The trick is to view it instead as a function of the secret key. Let us call this new circuit D_c for the noisy ciphertext c we are trying to refresh. We now assume that this function in terms of the secret key is a low-depth ciruit (that is, it involves a relatively low number of operations).

Next, Alice gives us *Enc(pk',sk)*, an encryption of the secret key using some other public key pk' (to maintain security).

Finally, we compute

$$\text{Eval}(evk, D_c, \text{Enc}(pk', sk))$$

where evk is the evaluation key for the circuit D and the public key pk'. If we decrypt this, we see from our definition of correct evaluation that we should get

$$\text{Dec}(\text{Eval}(evk, D_c, \text{Enc}(pk', sk))) = D_c(sk)$$

But note that by our definition of D_c that this is simply $\text{Dec}(sk, c)$. So the decryption of this computation is simply the plaintext message that c is encrypting. Let us call this value c'. Then we have just shown that c' encrypts the same value as c; however, only two circuits have been evaluated to obtain c': the homomorphic equivalent of the decryption circuit as a function of sk and the encryption circuit from Alice's side to encrypt the secret key. Encryption should involve relatively simple operations, and since we assumed that the decryption circuit as a function of sk has low depth, its homomorphic equivalent should also have relatively few operations. This mean our new ciphertext c' has less noise than c, but encrypts the same value! So in essence, we have "refreshed" the ciphertext and continue evaluating circuits on it and refreshing it again as necessary, thus allowing us to infinitely compute on the ciphertexts Alice sends us. This notion is called "bootstrapping," and is captured formally in the Bootstrapping Theorem due to Craig Gentry, a central result in Fully Homomorphic Encryption.

Theorem 1. (Bootstrapping Theorem) Any Somewhat Homomorphic Encryption scheme can be turned into a Fully Homomorphic Encryption scheme if it can handle its own decryption circuit.

Here, "handle" simply means that the SWHE scheme is consistent with the following criteria:

1. The encryption scheme can evaluate its own decryption circuit (i.e. our assumption that there was an evaluation key *evk* for D_c was justified).

2. The circuit is of low depth, which was our other assumption.

In essence, the proof of the Bootstrapping Theorem follows from our description of bootstrapping.

6 Conclusion

As of now, homomorphic encryption is still mostly confined to theory. The issues of efficiency in practical implementation prove to outweigh the benefits of outsourcing computation, at least for now. The more recent advancements in homomorphic encryption like Gentry's construction of a fully homomorphic encryption scheme appear promising, but are also based on theoretical ideas that are very hard to implement. Even Gentry's bootstrapping theorem, though seemingly powerful, does not see much application; in fact, in practice, most Somewhat Homomorphic Encryption schemes prove to be unable to handle their own decryption circuits without severe and costly modifications, as if by law of nature homomorphic encryption cannot be allowed to have any efficient implementation. However, the potential that homomorphic encryption has, in the ability to allow encrypted computing without requiring the vast amount of power required to run multiple servers that current methods require, will make it continue to be an important area of research for computer scientists in the years to come.

References

[1] Craig Gentry , Dan Boneh, *A Fully Homomorphic Encryption Scheme*, Stanford University, 2009
Last accessed 8 December 2019

[2] Craig Gentry, *Computing Arbitrary Functions of Encrypted Data*, Stanford University
Last accessed 8 December 2019

[3] Craig Gentry, *Fully Homomorphic Encryption Using Ideal Lattices*, Symposium on the Theory of Computing (STOC), 2009
Last accessed 8 December 2019

[4] Frederik Armknecht , Colin Boyd , Christopher Carr , Kristian Gjøsteen , Angela Jaschke1 , Christian A. Reuter , and Martin Strand, *A Guide to Fully Homomorphic Encryption*
Last accessed 8 December 2019

[5] Jian Lu, Lusheng Chen, Sihem Mesnager *Partially Homomorphic Encryption Schemes Over Finite Fields*, 2016
Last accessed 8 December 2019