HOMOMORPHIC ENCRYPTION

ISABELLE HONG

1. Introduction

An encryption scheme is *homomorphic* if it is possible to perform operations on the ciphertext without altering the plaintext message or having access to the key. A scheme ε must be able to generate a public key, encrypt plaintext, and decrypt ciphertext. These algorithms are represented by $\text{KeyGen}_\varepsilon$, $\text{Encrypt}_\varepsilon$, and $\text{Decrypt}_\varepsilon$, respectively. It also has the $\text{Evaluate}_\varepsilon$ algorithm, which is able to output a ciphertext ψ given the public key pk, a circuit C from a fixed set of circuits C_ε, and some set of ciphertexts $\Psi = \psi_1, ..., \psi_t$. In this context, a circuit takes input values, which are then processed through gates that each compute a function. The definition of homomorphic encryption relies on correctness.

Definition 1 ([Gen09b]). A scheme ε is correct for circuits in C_ε if, for any key pair (sk, pk), any circuit $C \in C_\varepsilon$, any plaintexts $\pi_1, ..., \pi_t$, and any ciphertexts $\Psi = \psi_1, ..., \psi_t$ where $\psi \leftarrow \text{Encrypt}_\varepsilon(pk, \pi_i)$, it is true that if

$$\psi \leftarrow \text{Evaluate}_\varepsilon(\text{pk}, C, \Psi), \text{then}$$

$$C(\pi_1, ..., \pi_t) = \text{Decrypt}_\varepsilon(\text{sk}, \psi).$$

Now, we can formally define homomorphic encryption.

Definition 2 ([Gen09b]). ε is homomorphic for circuits in C_ε if ε is correct for C_ε and $(\text{Decrypt}_\varepsilon)$ can be expressed as a circuit D_ε.

Homomorphic encryption is useful in instances where one party has some private data that must be somehow manipulated by a second party, without direct access to the data itself.

Think, for example, of a situation in which you wish to bake a batch of gingersnaps using your great-great-great-grandmother's secret recipe. You happen to be extremely lazy, so you do not want to bake the cookies on your own, but you cannot reveal the secret ingredients to anyone else. You lure a friend to your house with the promise of freshly baked goods, but instead of feeding them, you lay out all of your baking supplies and ingredients and turn off the lights. You put on your special night vision goggles and guide your friend through the darkness to mix the dough, mold it into circular shapes, and bake the cookies.

Even though you're the one giving the directions, you aren't the one who is actually making the cookies. Your friend is the one doing the work and has no idea how to recreate your recipe without your help, even though the cookies are delectable enough to justify many more batches. They have no idea what your secret ingredient is because the kitchen was too dark for them to have seen anything.

Much like homomorphic encryption, the results of performing operations on the encrypted data, or the ingredients in the dark, are the same as if you had just baked your own gingersnaps. Such a system may be useful in the medical field, where medical information must be kept private, and in finance, where a corporation may need to protect its data [ABC$^+$15].

In this paper, we will explore various fully homomorphic encryption schemes and explain their possible limitations.

2. Somewhat Homomorphic Encryption Schemes

If a cryptosystem is somewhat homomorphic, then it has a homomorphic property. Many familiar cryptosystems are somewhat homomorphic.

For instance, RSA is homomorphic with respect to multiplication because the process of encryption encrypts the product of the plaintexts. For the ciphertexts $c(m_1) = m_1^e \pmod{n}$ and $c(m_2) = m_2^e \pmod{n}$, it is true that

$$\begin{aligned} c(m_1) \cdot c(m_2) &= m_1^e \cdot m_2^e \pmod{n} \\ &= (m_1 m_2)^e \pmod{n} \\ &= c(m_1 \cdot m_2). \end{aligned}$$

From this example, we can see that the product of the encrypted messages is equal to the encrypted product of the messages.

Similarly, the ElGamal encryption scheme, which is based on the Diffie-Helman key exchange, is also multiplicatively homomorphic. For a prime p, a generator $g \in \mathbb{F}_p^\times$, ciphertexts $c(m_1) = (g^{b_1}, m_1 \cdot g^{a_1 b_1}) \pmod{p}$ and $c(m_2) = (g^{b_2}, m_2 \cdot g^{a_2 b_2}) \pmod{p}$, where a and b are between 0 and $p-2$,

$$\begin{aligned} c(m_1) \cdot c(m_2) &= (g^{b_1}, m_1 \cdot g^{a_1 b_1}) \cdot (g^{b_2}, m_2 \cdot g^{a_2 b_2}) \\ &= (g^{b_1} \cdot g^{b_2}, m_1 \cdot g^{a_1 b_1} \cdot m_2 \cdot g^{a_2 b_2}) \\ &= (g^{b_1 b_2}, m_1 m_2 \cdot g^{a_1 b_1 + a_2 b_2}) \\ &= c(m_1 \cdot m_2). \end{aligned}$$

These earlier encryption schemes formed the basis for the development of fully homomorphic encryption schemes.

3. Fully Homomorphic Encryption Schemes

Definition 3. If a scheme is fully homomorphic, then it must be homomorphic for all circuits.

3.1. **Gentry's Scheme Based on Ideal Lattices.**

Gentry's scheme ([Gen09a], [Gen09b]), the first fully homomorphic cryptosystem, uses ideal lattices, rather than exponents, because the process of decryption is less complex. In addition, ideal lattices already have operations of addition and multiplication because of their ring structure. (A ring is an abelian group under addition with the properties of multiplicative associativity and distributivity.) Gentry constructed his system by starting with a bootstrappable scheme, then manipulating this property to obtain a fully homomorphic system.

The construction of his scheme is best left to Gentry himself; however, a few important ideas introduced by his construction have been tantamount to the development of other fully homomorphic systems, many of which also involve lattices and bootstrapping.

Definition 4 ([Gen09b])**.** Let C_ε be a set of circuits, where ε is homomorphic to the circuits of C_ε. We can define ε as bootstrappable with respect to a set of gates with plaintext inputs and outputs Γ if

$$D_\varepsilon(\Gamma) \subseteq C_\varepsilon,$$

where $D_\varepsilon(\Gamma)$ is a g-augmented decryption circuit in which D_ε inputs a secret key and ciphertext and a g-gate, $g \in \Gamma$ connects copies of D_ε.

Theorem 5 ([Gen09b])**.** *If ε is bootstrappable with respect to Γ, then the family of schemes $\varepsilon^{(d)}$ is leveled fully homomorphic.*

The bootstrappable property enables a scheme to transform from somewhat homomorphic to fully homomorphic, which is extremely useful in constructing schemes. Bootstrapping reduces the noise produced by encryption, and this property is necessary because decryption becomes impossible when there is too much noise.

However, somewhat homomorphic schemes are not bootstrappable by default when decryption involves the step of generating $n+1$ vectors from n vectors. Gentry introduces the process of "squashing the decryption circuit," which limits the parameters of the Decrypt function to make the scheme bootstrappable without altering the set of circuits.

3.2. **The BGV Scheme.**

A newer fully homomorphic cryptosystem is the Brakerski-Gentry-Vaikuntanathan (BGV) scheme ([BGV11]), which does not rely on bootstrapping and instead depends on the ring learning with errors (RLWE) problem. The authors define a general learning with errors (GLWE) problem to include both learning with errors and its variant, ring learning with errors.

Definition 6 ([BGV11])**.** Let $n = n(\lambda)$ be an integer dimension, $f(x) = x^d + 1$ where $d = d(\lambda)$, $q = q(\lambda) \geq 2$ be a prime integer, $R = \mathbb{Z}[x]/(f(x)))$, $R_q = R/qR$, and $\chi = \chi(\lambda)$ be a distribution over R for a security parameter λ. The $\mathrm{GLWE}_{n,f,q,\chi}$ problem is to differentiate between a distribution in which $(\mathbf{a}_i, b_i)$ is sampled uniformly from R_q^{n+1} and another distribution where $s \leftarrow R_q^n$ is drawn uniformly and $(\mathbf{a}_i, b_i) \in R_q^{n+1}$ is sampled by sampling $\mathbf{a}_i \leftarrow R_q^n$ uniformly, $e_i \leftarrow \chi$, and b_i is set equal to $\langle \mathbf{a}_i, \mathbf{s} \rangle + e_i$. This problem is assumed to be impossible.

The general learning with errors problem is critical for encryption in the BGV scheme, which is unfeasible to explain completely in this paper.

Additionally, the BGV scheme relies on its Refresh function: this algorithm expands an encrypted ciphertext and changes the moduli and keys. Like bootstrapping, the Refresh function decreases the noise produced by encryption. However, the steps of the Refresh function make the scheme more efficient, in terms of computation time, because noise can be reduced more simply.

4. Limitations

Fully homomorphic encryption does not account for multiple users. Because each user would need a separate public key to protect their data from a provider, it is difficult to extend FHE to apply to a system with many users.

FHE schemes also have large runtimes because they require so much computation. Although these schemes run in polynomial time, they are not yet efficient enough to have practical applications. It also does not allow for algorithms to be kept private because fully homomorphic encryption does not enable function encryption. This which could be an issue in some contexts, especially those concerning data of some corporations. See [ABC$^+$15] for more details.

Although fully homomorphic encryption may not yet be completely applicable to the real world, it is important to remember that the first major breakthroughs in this area of cryptography only occurred ten years ago, with Gentry's scheme. In the past decade, many more techniques and schemes have been developed, so it is probable that many more achievements are to come.

References

[ABC$^+$15] Frederik Armknecht, Colin Boyd, Christopher Carr, Kristian Gjøsteen, Angela Jäschke, Christian A. Reuter, and Martin Strand. A guide to fully homomorphic encryption. *IACR Cryptology ePrint Archive*, 2015:1192, 2015.

[BGV11] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. *Fully Homomorphic Encryption without Bootstrapping.* 2011.

[Gen09a] Craig Gentry. *A Fully Homomorphic Encryption Scheme.* 2009.

[Gen09b] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, 2009.