Lattice-Based Cryptography in Homomorphic Encryption

Leyan Pan, Dr. Vincent Mooney

VIP Secure Hardware Fall 2019

Dec. 11, 2019

Contents

1 Introduction

1.1 Secure (Multiparty) Computation

In the 1980s, Andrew Yao [11] proposed the concept of secure computation, which allowed multiple parties to jointly compute a result based on their own private data. Other parties have no information about the private data beside the result of the jointly evaluated function. A classic example in Yao's paper is the millionaire's problem, where two millionaires would like to know who is richer without allowing each other the know their own wealth. Such protocols can be extended to relate to many real-life scenarios such as bidding and anonymous voting. Secure Computation guarantees privacy among multiple participating entities without a trusted third party collecing all the private data.

1.2 Homomorphic Encryption

In 1978, Ronald L. Rivest, Len Adleman, and Michael L. Dertouzos published *On Data Banks and Privacy Homomorphisms* [9]. The paper suggested that computations could be performed in the encrypted domain and noted that this would greatly reduce the number of opportunities for adversaries to get hold of sensitive information. This concept would later get fleshed out by the developments of many computer scientists and cryptographers, ultimately being collectively referred to as homomorphic encryption schemes. [8] Although encryption schemes which supports limited types computations have existed as early as 1978 (the RSA crypto-system [**?**]), an encryption scheme that supports arbitrary computation on cipher-texts is proposed in 2009 [5] and implemented in 2010 [6]. Very few commercial or practical usage of homomorphic encryptions is explored due to its serious inefficiency and limitations till this day.

1.3 Purpose of Document

The purpose of this document is to present work done by the Secure Hardware Fall 2019 VIP team's Homomorphic Encryption sub-team. The authors further investigated the application of homomorphic encryption on the Obtention Scenario proposed by Vignesh Raman [8] and implemented a simulated scheme on a personal computer. The authors also researched about the algorithms underlying Fully Homomorphic encryption schemes and eventually learned about lattice-based encryption, which is essential to understand Gentry's construction of the first Fully Homomorphic encryption scheme. The document is also written in the hope that future members of the Homomorphic Encryption sub-team would build on this document and research further into the subject.

2 Terminology

Encryption The process of encoding a message or information in such a way that only authorized parties can access it.

Symmetric Encryption A kind of encryption that involves only one secret key to cipher and decipher information. The main disadvantage of the symmetric key encryption is that all parties involved have to exchange the key used to encrypt the data before they can decrypt it which can be obtained by an adversary.

Asymmetric Encryption/Public Key Encryption A kind of encryption that uses two keys – one to encrypt a plaintext (public key) and the other to decrypt a ciphertext (private key).

Cryptosystem/Encryption Scheme A suite of cryptographic algorithms needed to implement a particular security service. Typically, it comprises at least 3 algorithms – one for key generation, one for encryption and one for decryption.

Probabilistic Encryption Probabilistic encryption is the use of randomness in an encryption algorithm, so that when encrypting the same message several times it will, in general, yield different ciphertexts.

Ciphertext Data that was the encrypted value of some encryption scheme

Plaintext Opposing term to ciphertext which refers to data that was not encrypted

Homomorphic Encryption An encryption scheme that allows computations to be performed on the ciphertexts without gaining any information about their plaintext

Lattice Mathematical structure that represents linear integer combinations of a vectors of an integer basis of n-dimensional space. More information in later sections.

Bootstrapping Homomorphically decrypting a doubly encrypted ciphertext in order to reduce the noise parameter. More information in later sections.

KDM Security A encryption scheme is KDM secure if providing the encrypted information of the secret key does not comprehend security.

Modular Arithmetic A integer arithmetic system where all numbers are represented as their remainder when divided by a common integer. All numbers with the same remainder under then common positive integer is viewed as equivalent to each other under this modular arithmetic system. The notion "under modulo m" or "mod m" represents a modular arithmetic system with the common positive integer m.

Modular Multiplicative Inverse The modular multiplicative inverse of x under modulo m is an positive integer x^{-1} less than m that satisfies $x \times x^{-1} = 1 \mod m$

3 Homomorphic Encryption Overview

Homomorphic Encryption refers to special (usually asymmetric) encryption schemes that allows one to compute certain functions over encrypted data without the decryption key – i.e., given encryptions $E(m_1), \ldots, E(m_t)$ of $m_1, \ldots, m_t$, one can efficiently compute a compact ciphertext that encrypts $f(m_1, ..., m_t)$ for function f. [5]

Fully homomorphic encryption has numerous applications. For example, it enables private queries to a search engine – the user submits an encrypted query and the search engine computes a

succinct encrypted answer without ever looking at the (plaintext) query. It also enables searching on encrypted data – a user stores encrypted files on a remote file server and can later have the server retrieve only files that (when decrypted) satisfy some boolean constraint, even though the server cannot decrypt the files on its own. More broadly, fully homomorphic encryption improves the efficiency of secure multiparty computation. [5]

3.1 Traditional Public Key Encryption

For an encryption scheme to be a public key encryption scheme, it must support the following three operations:

KeyGen: Generate public key pk and secret key sk with randomness. There must be enough possible public and secret keys to prevent an adversary from attempting a brute force attack by trying all possible secret keys. It should also be impossible to compute the secret key while knowing only the public key in a reasonable amount of time.
The private key is usually kept private by authorized individuals and the public key is known to everyone involved in the communication.

Encrypt(pk, m): Given public key pk and plaintext m in the plaintext space, generate ciphertext ψ which represents the encrypted value of m. In probabilistic encryption, ψ can be different each time with the same pk and m. It must be satisfied that m must not be computable when knowing only ψ and pk in a reasonable amount of time.

Decrypt(sk, ψ): Given secret key sk and ciphertext ψ, generate plaintext m. The algorithm must satisfy that, for any public and secret key pair pk and sk in the encryption scheme, *Decrypt*(sk, *Encrypt*(pk, m)) $= m$ for all possible plaintext m.

3.2 Homomorphic Encryption

In addition to the three algorithms that must be provided in a public key encryption scheme, a homomorphic encryption scheme have an additional required algorithm *Evaluate*:

Evaluate($pk, C, \psi_1, \ldots, \psi_n$): Given a function C which is a function with plaintext arguments $m_1, \ldots, m_n$, public key pk, and encrypted values $\psi_1, \ldots, \psi_n$ of $m_1, \ldots, m_n$, calculate a ciphertext value ψ that can be decrypted into $C(m_1, \ldots, m_n)$.

No information about the plaintext values of $m_1, \ldots, m_n$, or $C(m_1, \ldots, m_n)$ must be revealed to the entity performing the computation. Note that only selected functions C needs to be supported by the *Evaluate* function in an encryption scheme to be called a homomorphic encryption scheme.

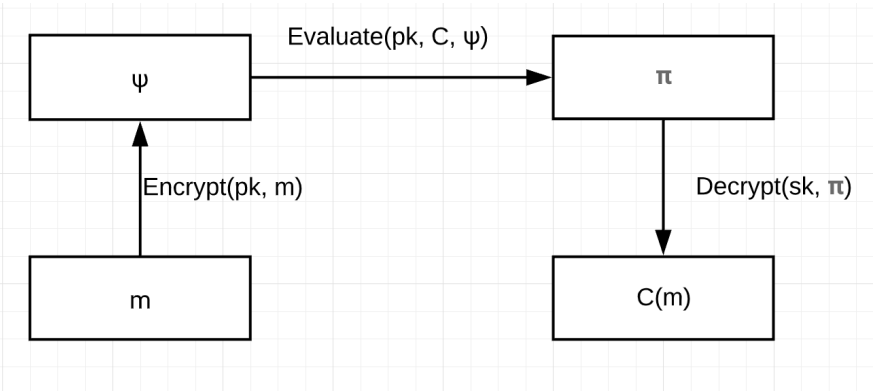


Figure 1: Illustration of Homomorphic Evaluation

3.3 Types of Homomorphic Encryption

As mentioned earlier, an homomorphic encryption scheme does not have to be able to evaluate all possible functions on any argument. However, based on the generality of operations supported, several types of homomorphic encryptions are distinguished among each other.

3.3.1 Partially Homomorphic Encryption (PHE)

Partially Homomorphic encryption schemes are only able to evaluate a single type of operation (usually one of addition or multiplication) but can perform infinite such evaluations correctly.
An example of an PHE is the famous RSA encryption scheme [9]. Suppose an RSA encryption scheme have public key $\{n, e\}$ and secret key $\{n, d\}$. Two plaintexts m_1 and m_2 would yield ciphertexts c_1 and c_2 which satisfies:

$$c_1 = m_1^e \mod n$$
$$c_2 = m_2^e \mod n$$

Applying modular multiplication on these ciphertexts yields:

$$c_1 c_2 = m_1^e m_2^e = (m_1 m_2)^e \mod n$$

which is the encrypted value of the product of the two original plaintext. This operation can be done infinitely many times between any two ciphertexts in the same encryption scheme correctly. However, modular addition is not supported for homomorphic evaluation in the RSA encryption scheme. Therefore, the RSA encryption scheme is a multiplicatively homomorphic encryption scheme.

Note that in the context of homomorphic encryption, multiplication and addition usually refers to modular multiplication and addition. Under modular 2, addition and multiplication were reduced to bit operations *xor* and *and*. This can be further reduced into a single bit operation *nor* which can be used to implement all other operations. A lot of PHEs are constructed throughout history, with the most influential of those listed in Table 1. Many of these PHE schemes are not actually designed intentionally for homomorphic computations during their construction, and achieves homomorphic encryption properties by coincidence.

	Homomorphic Operation	
Scheme	**Add**	**Mult**
RSA [Rivest et al. 1978b]		✓
GM [Goldwasser and Micali 1982]	✓	
El-Gamal [ElGamal 1985][4]		✓
Benaloh [Benaloh 1994]	✓	
NS [Naccache and Stern 1998]	✓	
OU [Okamoto and Uchiyama 1998]	✓	
Paillier [Paillier 1999]	✓	
DJ [Damgård and Jurik 2001]	✓	
KTX [Kawachi et al. 2007]	✓	
Galbraith [Galbraith 2002]	✓	

Table 1: Overview of Past PHE implementations [1]

3.3.2 Fully Homomorphic Encryption (FHE)

Despite the vast amount of PHE schemes, constructing an encryption scheme that supports arbitrary number of evaluations of both (modular) addition and multiplication proved to be a significantly harder task. Craig Gentry provided the first plausible theoretical construction of a fully homomorphic encryption scheme in 2009 [5] and was implemented in 2010 [6]. However, the initial implementation takes 30 minutes per basic bit operation (of *xor*, *and* etc.) [6], which was too inefficient for any practical usage. Various improvements and implementation of FHE have been created since then. One of the most efficient FHE implementation to date is TFHE [3] and performs homomorphic bit operations using roughly 23 milliseconds according to our tests. Despite significant improvement, FHE is still inefficient for wide commercial usage and would need further improvements in either hardware or algorithm for them to be widely usable.

3.3.3 Somewhat Fully Homomorphic Encryption (S/FHE)

A homomorphic encryption scheme that can perform both addition and multiplication homomorphically but can only perform such homomorphic operations a limited number of time is called Somewhat Fully homomorphic Encryption. Exceeding the number of allowed homomorphic operations may result in incorrect results.This is usually due to the existence of *noise parameters* in S/FHE ciphertext

S/FHE is much more widespread than FHE and is usually much more efficient than an FHE scheme. To give a sense of how S/FHE schemes work and how noise parameters affect decryption correctness, we would present a very simple S/FHE encryption scheme that can guarantee at least one *xor* and *and* can be performed while guaranteeing correctness.

3.3.4 Somewhat Fully Homomorphic Encryption (S/FHE) Scheme Example [5]

We would present a symmetric Somewhat Fully Homomorphic Encryption scheme to illustrate S/FHE and the *noise parameter*. Note that most homomorphic encryption schemes are asymmetric and this simple encryption scheme is only for illustration purposes without practical usage and is not representative of any actual S/FHE scheme.

Plaintext Space: $m \in \{0, 1\}$, meaning that the scheme can only encrypt boolean values.

Keygen: Generate k = random large odd positive integer. The key is k.

Encrypt(k, m): Generate random integer $-\frac{k}{4} < x < \frac{k}{4}$, random large positive integer p. Let $\psi = m + 2x + pk$. ψ is is the ciphertext of m.

Decrypt(k, ψ): Compute $m = ((\psi \mod k) \mod 2)$. m is the decrypted plaintext of

The correctness of the above encryption scheme relies on the fact that when $|2x| < k$, $\psi \mod k = (m + 2x + pk \mod k) = m + 2x$ and $(m + 2x \mod 2) = m$. In this scheme, the value $2x$ is called the *noise parameter* because the value of the noise parameter must be within a certain range for the decryption circuit to work correctly. Now we would introduce the homomorphic bit operations *and* and *xor* in this simple S/FHE scheme. The homomorphic *xor* operation is by simply adding the two ciphertexts together:

$$\begin{aligned} Evaluate(xor, \psi_1, \psi_2) &= \psi_1 + \psi_2 \\ &= m_1 + 2x_1 + p_1 k + m_2 + 2x_2 + p_2 k \\ &= (m_1 + m_2) + 2(x_1 + x_2) + (p_1 + p_2)k \end{aligned}$$

During this evaluation, the noise parameter (the portion in the ciphertext corresponding to $2x$) of the resulting ciphertext is $2(x_1 + x_2)$. This means that the resulting ciphertext would only decrypt into the correct value if the new noise parameter, $2(x_1 + x_2)$, is still smaller than k. When both ciphertext parameters are "fresh" ciphertexts (i.e. the ciphertext generated by an *Encrypt* algorithm), then $-\frac{k}{4} < x_1, x_2 < \frac{k}{4}$ and therefore $|2(x_1 + x_2)| < k$ is always satisfied. However, if the ciphertext parameters are already the result of some other homomorphic *xor* evaluations, then the new noise parameter may exceed its constraint and result in incorrect decrypted values. A similar phenomenon occurs when performing the homomorphic *and* operation by multiplying the two ciphertexts.

$$\begin{aligned} Evaluate(and, \psi_1, \psi_2) &= \psi_1 \psi_2 \\ &= m_1 m_2 + 2(m_1 x_2 + m_2 x_1 + 2x_1 x_2) + (p_1 p_2 k)k \end{aligned}$$

where the new ciphertexts' noise parameter becomes $2(m_1 x_2 + m_2 x_1 + 2x_1 x_2)$. With large x_1 and x_2 values, the new noise parameter can easily exceed its constraint and result in incorrect homomorphic evaluation results.

In the general sense, the noise parameter is a part of the ciphertext that is constrained within a certain threshold for correct decryption whose value changes with homomorphic evaluations. The number of allowed evaluations differs based on the homomorphic encryption scheme (1 in our simple scenario) but is always finite.

3.4 Previous Work: The Obtention Scenario [8]

In *Security for Public Infrastructure using Homomorphic Encryption*, Vignesh Raman and Dr. Vincent Mooney proposed a potential real life scenario where homomorphic encryption may be applied.

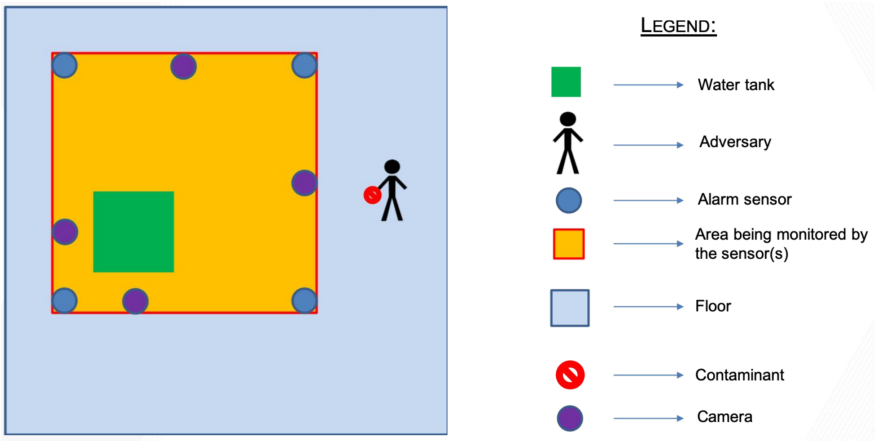


Figure 2. The Obtention Scenario [8]

In the proposed scenario, an adversary is attempting to pollute the water in a public water tank protected by cameras and alarm sensors. The cameras collect images and sends them to the alarm sensor while the alarm sensors perform The adversary is able to obtain one of the alarm sensors and gain information about the circuit in the alarm sensors. The adversary are not aware of the presence of cameras. In the paper, the authors would like to apply homomorphic encryption to the alarm sensors so that even if the adversary fully analyze the alarm sensors they would nor gain information about the detection technique near the water tank.

The authors proposed to apply homomorphic encryption to the File Size Comparison algorithm [2], which involves JPEG compression of consecutive images from a video stream and computing the file size differences in order to detect motion. A mathematical formulation of the algorithm is $|a - b| > d$, where a and b are JPEG file sizes of two consecutive images and d is a predefined threshold that tunes the sensitivity of the alarm sensor.

We further formulated the process flow of the detection system as `Figure 3`. Within the figure, red block denote portions of the alarm system that we would focus on in out implementation. Specifically, we would focus on key generation, encryption, homomorphic evaluation of the comparison operation in the File Size Comparison method, and decryption.

3.5 Implementation of computations in the Obtention Scenario with the TFHE Library [4]

The Fast Fully Homomorphic Encryption over the Torus (TFHE) is an online library that implements the scheme from the paper "Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds" [3]. It is a C/C++ library that is intended to evaluate an arbitrary boolean circuit composed of binary gates, over encrypted data, without revealing any information of the data.

The TFHE library provides API for encryption and decryption of a single bit (or boolean value) and homomorphic evaluations of boolean operations including AND, OR, NOT and MUX. MUX takes three parameters a, b, c and outputs b if $a = 1$ or c if $a = 0$. $MUX(a, b, c)$ is often represented as $a?b : c$.

However, in order to implement the threshold comparison in the Obtention Scenario, we need to perform calculation on integers instead of boolean values. Fortunately, homomorphic boolean operations can be used to implement homomorphicinteger adders and comparators (operating on integers of a specific bitsize). Hence, we transformed the threshold comparison in the following way:

$$|a - b| > d$$

$$(a - b) > 0 \ ? \ (a - b) > d : (b - a) > d$$

$$(a > b) \ ? \ a > (b + d) : b > (a + d)$$

The resulting formula is composed of only comparison, addition and MUX operations, which can all be implemented using homomorphic bit operations in straightforward means. The result of a homomorphic comparison is a single encrypted bit (1 for $true$, 0 for $false$) and therefore the result of threshold value comparison is also an encrypted bit.

We implemented the homomorphic threshold comparison operation on a personal computer with 2.3 GHz Intel Core i5 CPU and encrypted integer bit sizes of 16 bits during the evaluation (meaning that file sizes are all represented as encrypted 16-bit integers). The recorded results are

as follows:
Key generation: 1.06 seconds
Key Size: 82.1MB for both public and secret key
Encryption (of three 16-bit integers, a, b and d): 0.8 milliseconds
Encrypted Input Size: 98KB
Homomorphic Threshold Comparison: 3.6 seconds
Decryption (of a single result bit): 0.013 milliseconds

Judging from the performance of our implementation, the TFHE library is much more efficient than the Gentry-Halevi implementation of Gentry's Original Scheme [6], which costs 30 minute per basic homomorphic bit operation. However, the TFHE library is still considerably slower than plaintext operations. With the efficiency of the current implementation, the TFHE library is still not feasible for use in the alarm sensor in the Obtention Scenario due to two consecutive images in video streams being less than a second apart. In general, fully homomorphic encryption still need significant improvement in performance to be practical in real-life scenarios.

We also conclude that homomorphic encryption cannot be used to conceal the detection mechanism if the adversary have knowledge about the homomorphic encryption scheme used in alarm sensor. Homomorphic operations have the same algorithm for the same homomorphic operations, and therefore the adversary can theoretically reverse engineer each homomorphic operation performed in the alarm sensor and convert them into their corresponding plaintext operations to obtain the detection mechanic. However, if the adversary does not know which encryption scheme is used in the alarm system, then they would not gain any knowledge about the corresponding plaintext operations performed in the alarm sensor.

Despite implementing an working example of an homomorphic threshold comparison, we did not have any knowledge about the underlying scheme in the TFHE library and therefore cannot guarantee its security. Therefore, we decided to investigate further on the algorithms underlying the Homomorphic Encryption Schemes by researching Gentry's original proposal of and FHE scheme [5].

3.6 Gentry's approach of Fully Homomorphic Encryption [5]

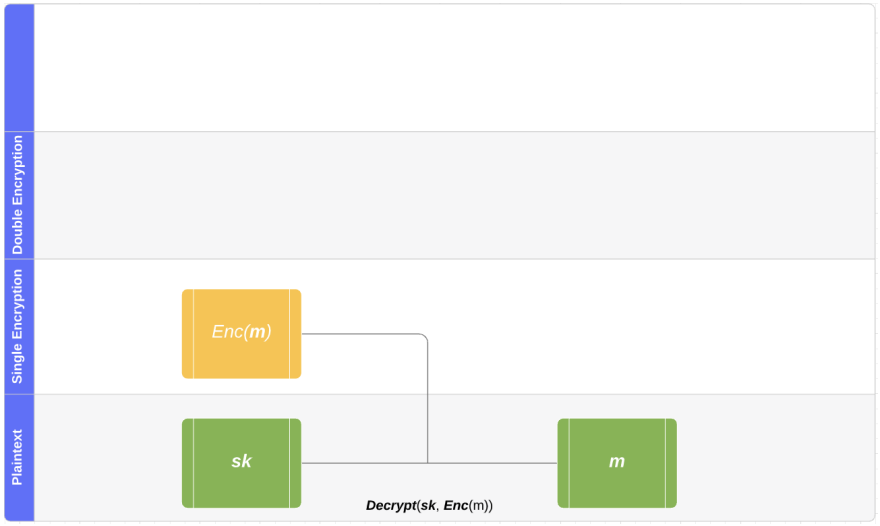
Craig Gentry, using lattice-based cryptography, described the first plausible construction for a fully homomorphic encryption scheme in 2009. The Gentry-Halevi implementation of Gentry's original scheme [6] reported timing of about 30 minutes per basic (Homomorphic) bit operation.

The core idea behind Gentry's construction of Fully Homomorphic Encryption is Bootstrapping, which allows the transformations of certain S/F Homomorphic Encryption schemes into a Fully Homomorphic Scheme. Despite various improvements in FHE by multiple researchers after Gentry's first proposal, including TFHE [4], bootstrapping is still the core technique behind almost all Fully Homomorphic encryption schemes. In this paper we would briefly introduce the core idea of bootstrapping.

Bootstrapping attempts to solve the problem brought about by the noise parameter in a S/FHE scheme. In theory, if the entity performing the evaluation can decrypt the ciphertext with large noise and encrypt the decrypted value as a new ciphertext (remember that ciphertexts produced by the $Encrypt(pk, m)$ function have low noise), then the entity can effectively lower the noise of a ciphertext. However, this is illegal because the entity performing homomorphic operations must not have the secret key. Bootstrapping solves this problem by performing the decryption

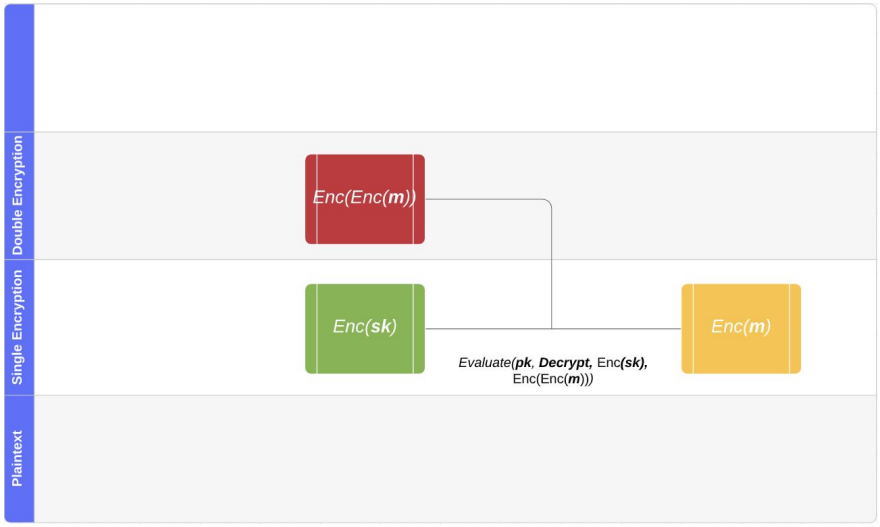
homomorphically so that a plaintext secret key would not be required.

Notice that the $Decrypt(sk, \psi)$ is also function with the secret key sk and ciphertext ψ as parameters, as shown in `Figure 3`.



`Figure 3: Decryption Function`

This implies that it is possible for the Decrypt function to be evaluated homomorphically, requiring an encrypted version of the secret key and the ciphertext, which is an doubly encrypted ciphertext, as illustrated in `Figure 4`.



`Figure 4: Homomorphically Decrypting a ciphertext`

If the decryption is not performed homomorphically, then the decrypted plaintext would not have any noise (it's not even a ciphertext therefore there's not a noise parameter). Therefore, if the decryption is performed homomorphically, then the noise parameter in the new ciphertext resulting from the homomorphic decryption would be independent from the original ciphertext with large noise parameter. The noise parameter in the new ciphertext is only generated by the homomorphic

decryption circuit. If noise resulting from the homomorphic decryption is relatively small, then we can effectively reduce the noise of a ciphertext.

We now can introduce the definition of **bootstrappable**: A somewhat fully homomorphic encryption scheme is **bootstrappable** if it can homomorphically evaluate its own decryption circuit (and one extra homomorphic operation) without exceeding the noise parameter limit and is KDM secure. In the terminology section we introduced a scheme to be KDM secure if revealing the encrypted value of the secret key does not compromise security. If a S/FHE is bootstrappble, then we can define the following algorithm:

Recrypt algorithm:
Input: High-noise ciphertext ψ, public key pk, encrypted secret key $Enc(sk)$

Let $\psi' = Encrypt(pk, \psi)$

$\psi_{new} = Evaluate(pk, Decrypt, Enc(sk), \psi')$

Output ψ_{new}

The algorithm effectively takes a high-noise ciphertext can create a relatively lower noise ciphertext encrypting the same data.

Now we have an algorithm which reduces ciphertext noise, we can straightforwardly construct a method which takes a bootstrappable Somewhat Fully Homomorphic Encryption Scheme and construct a Fully Homomorphic Encryption Scheme. The following algorithm allows a bootstrappable S/FHE to evaluate functions of arbitrary numbers of homomorphic computations.

1. Start with original ciphertexts and encryption of the secretkey, $Enc(sk)$
2. Perform homomorphic operations until it is impossible to perform further operations while guaranteeing correctness
3. Perform *Recrypt* algorithm on the ciphertext with heavy noise to get a ciphertext with relatively low noise
4. Perform further homomorphic operations until it is impossible to perform further operations while guaranteeing correctness
5. Repeat step 3 and 4 until the required function is fully evaluated.

However, for an S/F Homomorphic Encryption scheme to be bootstrappable, the scheme need to evaluate its own decryption circuit. The create such a scheme, it is straightforward to divide the restriction into two separate parts:

1. The scheme should have short decryption circuit (Low number of operations in the *Decrypt* function)
2. The scheme should have large evaluation strength (maximum number of homomorphic operations while guaranteeing correctness)

However, many traditional public key encryption have long decryption circuits especially when they rely on exponential calculations. RSA [9], specifically, rely on modulus exponential in their decryption circuit, making bootstrapping possible. Gentry claims that many lattice-based encryption algorithms, specifically ideal lattices, have short decryption circuits that can be further reduced with minor modifications. However, we were not familiar with any lattice-based encryption related materials and we were motivated to investigate into lattice theory and lattice-based encryption to get a better understanding of Gentry's constructed S/F Homomorphic Encryption Scheme.

4 Lattice-Based Cryptography

4.1 Motivation

Lattice-based encryption schemes refer to a type of encryption schemes that bases its security on hard lattice problems. With the (supposed) future arrival of functional quantum computers, traditional public key encryption schemes whose security is based on the hardness of prime factorization is no longer secure due to already known fast quantum algorithms. Shor's algorithm, for example, achieves polynomial time in solving the prime factorization problem using a quantum computer. However, there is still no feasible algorithm to date, even with the existence of quantum computers, that can compromise the security of lattice-based crypto-systems, making these crypto-systems one of the ideal candidates for post-quantum encryption standards.

Many homomorphic encryption scheme also lies in the realm of lattice-based encryption schemes. Gentry's first construction of FHE, for example, is based on ideal lattices, which is the primary motivation for this research into Lattice-based cryptography. Gentry claims that Lattice-based cryptography is suitable for the construction of a bootstrappable S/FHE scheme due to the fact that lattice-based encryption schemes can be design to have very simple decryption circuits, although later implementations of FHE proved that the property is not unique to lattice-based crypto-systems.

We decided to investigate Lattice-based cryptography to obtain a deeper understanding of Gentry's Homomorphic Encryption scheme. However, due to lack of time, we would only informally introduce lattices and some examples of (insecure) lattice-based encryption schemes.

4.2 Lattice

Lattices are mathematical structures with linear combinations of integer vectors. Formally, for n independent integer vectors $\{v_1, v_2, \ldots, v_n\}$ in $\mathbb{Z}^n$ (integer vectors with n components), a lattice in $\mathbb{R}^n$ is defined as $\mathbf{L} = \{\sum_{i=1}^{n} a_i v_i \mid a_i \in \mathbb{Z}\}$. The vectors $\{v_1, v_2, \ldots, v_n\}$ are the basis vectors of the lattice. A example of a lattice is given below.

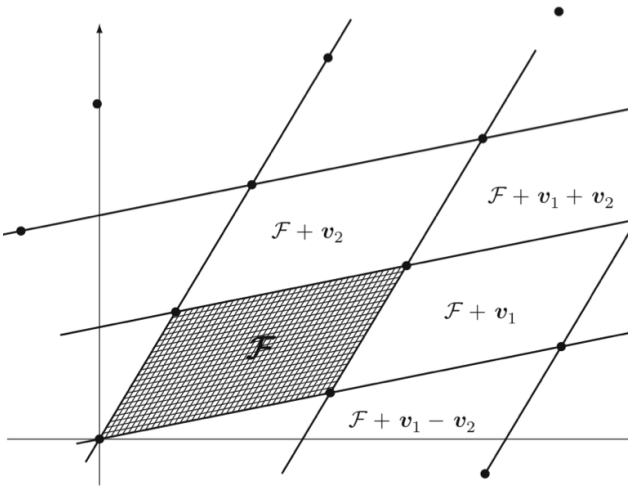


Figure 5. Example Lattice [7]

Intuitively, in a 2D coordinate space, a lattice is composed of infinite parallelograms adjacent to each other. Likewise, a 3D lattice is composed of infinite parallelepiped adjacent to each other. In the Figure, the shaded area is called the *Fundamental Region* of a Lattice and the two vectors surrounding the fundamental region connected to the origin are the two basis vectors. It is important to note that a single lattice can have more than one set of basis. If all the vectors in onebasis can be represented by integer linear combinations of vectors in another basis and vice versa, then the two basis would form the same lattice.

In this report, lattice basis would be represented by a matrix of its basis vectors. For example, a lattice with basis vectors $\begin{pmatrix}1 & 2\end{pmatrix}$ and $\begin{pmatrix}3 & 4\end{pmatrix}$ can be represented by the matrix $\begin{pmatrix}1 & 2\\ 3 & 4\end{pmatrix}$, where each row is one vector in the basis. Using this basis matrix notation, we can see that the basis $\begin{pmatrix}1 & 2\\ 3 & 4\end{pmatrix}$ and the basis $\begin{pmatrix}1 & 0\\ 2 & 2\end{pmatrix}$ actually constructs the same lattice because $a\begin{pmatrix}1 & 2\end{pmatrix} + b\begin{pmatrix}3 & 4\end{pmatrix} = (-a-b)\begin{pmatrix}1 & 0\end{pmatrix} + (a+2b)\begin{pmatrix}2 & 2\end{pmatrix}$ and $a\begin{pmatrix}1 & 0\end{pmatrix} + b\begin{pmatrix}2 & 2\end{pmatrix} = (-2a-b)\begin{pmatrix}1 & 2\end{pmatrix} + (a+b)\begin{pmatrix}3 & 4\end{pmatrix}$, and thus all the vectors in one lattice can be represented by integer linear combinations of vectors in another basis. By definition, these two basis are two different basis of a same lattice.

4.3 Lattice Problems

Lattice problems are computationally hard problems involving lattices. This report would primarily focus on the **Shortest Vector Problem (SVP)**. As the name suggests, this problem requires an algorithm that finds the shortest non-zero vector in the lattice given the basis of a lattice, measured in Euclidean distance. Modifications of the problem may also ask to find a vector whose length is below a certain threshold, but it does not reduce the computational hardness of the problem. In order to demonstrate why this apparently simple problem can be computationally hard, consider the following basis:

$$\begin{pmatrix}6 & 0 & 1\\ 1 & 3 & 1\\ 7 & 3 & -5\end{pmatrix}$$

Using this basis, it is straightforward (and correct) to judge that the shortest vector in the lattice is the vector $\begin{pmatrix}1 & 3 & 1\end{pmatrix}$ with Euclidean distance $\sqrt{1^2+3^2+1^2} = \sqrt{11}$. However, consider another basis for the same lattice:

$$\begin{pmatrix}449857 & 1731 & 72769\\ 224936 & 870 & 36380\\ 224927 & 861 & 36390\end{pmatrix}$$

Note that

$$74689\times\begin{pmatrix}449857 & 1731 & 72769\end{pmatrix} - 74880\times\begin{pmatrix}224936 & 870 & 36380\end{pmatrix} - 74496\times\begin{pmatrix}224927 & 861 & 36390\end{pmatrix} = \begin{pmatrix}1 & 3 & 1\end{pmatrix}$$

However, without knowing the shortest vector $\begin{pmatrix}1 & 3 & 1\end{pmatrix}$ beforehand, it is almost impossible to obtain the coefficients $74689, -74880, -74496$ using brute force. In higher dimensions, the number of coefficients would further increase, making such brute force attack even less feasible.

4.4 Example Lattice-based Encryption Scheme: Knapsack Encryption [7]

In this example, we would introduce a lattice-based encryption scheme that has its security based on the SVP. The decryption of the encryption scheme itself does not require lattices and we would introduce its relationship with lattices when proving its security.

4.4.1 The Subset-sum Problem

The subset-sum problem is a traditional computationally hard problem that involves finding a subset of numbers that adds up to a specific value. Formally, given a set of n positive integers (for simplicity, assume they are sorted in ascending order) $\mathbf{r} = \{r_1, r_2, \ldots, r_n\}$ and positive integer N, find a subset of $\mathbf{r}$ whose sum is exactly N or claim that it is impossible to do so. The best algorithms can only solve this general problem in time complexity $O(2^{n/2})$ and is therefore infeasible for large n.

However, special cases of the problem can be easily solved in a short time. If the set $\mathbf{r}$ satisfy that each element in r is at least two times as large as the previous element, then $\mathbf{r}$ is called a *super-increasing* sequence. For this type of set, the subset-sum problem can be easily solved.

For example, consider the set $\mathbf{r} = \{3, 7, 18, 36, 80, 170, 352\}$ and $N = 475$. Due to the fact that $\mathbf{r}$ is a super-increasing sequence, we can solve this problem by comparing (a copy of) N to each element in $\mathbf{r}$ from the last one and subtracting the number if possible.

We create a new variable x initialized to $N = 475$. We use the following the process to solve this subset-sum problem by constructing a subset $\mathbf{s}$

$$x > r_7 = 352 \quad 352 \in \mathbf{s} \quad x = 475 - 352 = 123 \tag{1}$$
$$x < r_6 = 170 \quad 170 \notin \mathbf{s} \quad x = 123 \tag{2}$$
$$x > r_5 = 80 \quad 80 \in \mathbf{s} \quad x = 123 - 80 = 43 \tag{3}$$
$$x > r_4 = 36 \quad 36 \in \mathbf{s} \quad x = 43 - 36 = 7 \tag{4}$$
$$x < r_3 = 18 \quad 18 \notin \mathbf{s} \quad x = 7 \tag{5}$$
$$x = r_2 = 7 \quad 7 \in \mathbf{s} \quad x = 7 - 7 = 0 \tag{6}$$

and when $x = 0$ we stop and output the subset $\mathbf{s}$. However, if $\mathbf{r}$ is not superincreasing, then this algorithm would not work and a much slower algorithm would be needed to solve this problem.

4.4.2 Tiny Symmetric Encryption Scheme

In this section we would introduce how to use the subset-sum problem to represent information by introducing a symmetric encryption scheme that uses the subset-sum problem. All information stored on computers can be represented by a number of bits, and we can divide these bits into bit-strings of length n. We describe the scheme as follows:

$m \in \{0, 1\}^n$, i.e. the plaintext is a bit-string of length n.

Keygen: Generate random super-increasing sequence $\mathbf{r} = \{r_1, r_2, \ldots, r_n\}$, the key $k = \mathbf{r}$ is the key.

Encrypt(k, m): let $\psi = \sum_{i=1}^{n} m_i \mathbf{r}_i$, which basically sums up all elements in $\mathbf{r}$ whose corresponding m is 1. ψ is the encrypted value.

Decrypt(k, ψ): Run the process described in the **4.4.1** to get a subset $\mathbf{s}$ that sums up to ψ. Create bit-string m of length n where $m_i = 1$ if $r_i \in \mathbf{s}$ and $m_i = 0$ otherwise. The new bit-string is the decrypted value.

The correctness of the scheme is straight forward and will not be discussed in detail. However, for most applications we would like to construct a public-key encryption scheme instead of a symmetric encryption scheme. We would approach this goal by creating a nonsuper-increasing sequence as the public key so that solving the subset-sum problem using the public key is infeasible and use a super-increasing sequence as the private key for decryption.

4.4.3 Asymmetric Knapsack Encryption

In order to break the super-increasing property, we can multiply all elements in a super-increasing sequence by a constant and take its modulus by another very large integer. Consider the modification on the previous exmaple: $\mathbf{r} = \{3, 7, 18, 36, 80, 170, 352\}$, if we multiply all elements in $\mathbf{r}$ by constant $A = 29$ and take the modulus with respect to $B = 761$, we would get:

$$A\mathbf{r} = \{87, 203, 522, 1044, 2320, 4930, 10208\} \mod B \tag{7}$$

$$= \{87, 203, 522, 283, 37, 364, 315\} \mod B \tag{8}$$

Whereas $\mathbf{r}$ is a super-increasing sequence, the new set $A\mathbf{r} \mod B$ is not. Note that the subset-sum problem for a nonsuper-increasing sequence is computationally hard to solve. Now we could formulate a public key encryption scheme as follows:

Keygen: Generate random super-increasing sequence $\mathbf{r} = \{r_1, r_2, \ldots, r_n\}$. Generate random positive integer A and B, where B is larger than two times the largest term in $\mathbf{r}$ and is relatively prime to A. Compute $\mathbf{r}' = A\mathbf{r} \mod B$. The secret key sk is $\{\mathbf{r}, A, B\}$ and the public key pk is $\mathbf{r}'$.

Encrypt(pk, m): let $\psi = \sum_{i=1}^{n} m_i \mathbf{r}'_i$, which basically sums up all elements in $\mathbf{r}'$ whose corresponding m is 1. ψ is the encrypted value.

Decrypt(k, ψ): Calculate the multiplicative modular inverse $A^{-1} \mod B$, which satisfies that $AA^{-1} \mod B = 1$. Calculate $\psi' = A^{-1}\psi \mod B$. Run the process described in the **4.4.1** to get a subset $\mathbf{s}$ of $\mathbf{r}$ that sums up to ψ'. Create bit-string m of length n where $m_i = 1$ if $r_i \in \mathbf{s}$ and $m_i = 0$ otherwise. The new bit-string is the decrypted value.

Note that B is larger than the largest possible sum if a subset summation is performed using the secret key (the super-increasing sequence) and therefore $\psi' = A^{-1}\psi \mod B$ must be equal to $\sum_{i=1}^{n} m_i \mathbf{r}_i$ according to modulus arithmetic properties. We would demonstrate this scheme using the previous example:

$m = \{0, 1, 0, 1, 1, 0, 1\}$ which represents 45 in binary

$\mathbf{r} = \{3, 7, 18, 36, 80, 170, 352\}$

$A = 29$

$B = 761$

$A\mathbf{r} = \{87, 203, 522, 1044, 2320, 4930, 10208\}$

$pk = \mathbf{r}' = A\mathbf{r} = \{87, 203, 522, 283, 37, 364, 315\} \mod B$

$sk = \{\mathbf{r}, A, B\}$ Note: A and B must be relatively prime to each other and B must be larger than $2r_n$

Encrypt(pk, m): $\psi = 203 + 283 + 37 + 315 = 838$

Decrypt(sk, ψ): $A^{-1}\psi = 105 \times 838 = 475 \mod 761$, and then we can run the process described in **4.4.1**

Now we've described the entire encryption scheme without the concept of lattices, we would state its relationship to lattice-based cryptography.

4.4.4 Relationship to Lattices

Assume an adversary is attempting to break the crypto-system (decrypting the cipher-text using only the public key) by solving the subset-sum problem using the public key. In the previous example, this would be equivalent to solving the subset-sum problem with $N = 838$ and $\mathbf{r} = \{87, 203, 522, 283, 37, 364, 315\}$. We can construct the following lattice basis:

$$\begin{pmatrix} 2 & 0 & 0 & 0 & 0 & 0 & 0 & 87 \\ 0 & 2 & 0 & 0 & 0 & 0 & 0 & 203 \\ 0 & 0 & 2 & 0 & 0 & 0 & 0 & 522 \\ 0 & 0 & 0 & 2 & 0 & 0 & 0 & 283 \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 & 37 \\ 0 & 0 & 0 & 0 & 0 & 2 & 0 & 364 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & 315 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 838 \end{pmatrix}$$

In the above $n+1$ dimensional lattice basis, the rightmost column is the numbers in the public key and the cipher-text to be broken. The left columns is composed of an $n \times n$ matrix with diagonal values of 2 and all other values 0. The last row are all ones except for that last number.

Note that the plaintext $m = \{0, 1, 0, 1, 1, 0, 1\}$, which is the goal value of the adversary. If we add up the columns corresponding to $1s$ in m (The second, fourth, fifth and seventh row) and subtract the last row, we would get the vector $\begin{pmatrix} -1 & 1 & -1 & 1 & 1 & -1 & 1 & 0 \end{pmatrix}$. By definition, this vector is contained in the lattice formed by the lattice basis matrix. We can obtain the value of m by substituting -1 in the vector to 0 and removing the last 0 in the vector. Notice that no matter the plaintext, there is always a vector of such form in the lattice and will always have length $\sqrt{n}$ measured in Euclidean distance. It is also very rare for a vector in this same lattice to have length smaller than $\sqrt{n}$. Therefore, we can say that obtaining the plaintext is equivalent to obtaining the shortest vector in the above lattice, thus solving the SVP in the lattice.

Note that this is only an example scheme for illustrative purposes and an adversary may use other attacks on the scheme for better efficiency. But we can still claim that the security of the described scheme relies on the hardness on the SVP problem and is thus a lattice-based encryption scheme.

5 Conclusion and Future Work

In this report we gave an overview of the research of the Spring 2019 Secure Hardware VIP Team Homomorphic Encryption Sub-team. We conclude that current methods achieving fully homomorphic encryption is still not feasible for commercial or practical usage. In order to gain more understanding of the underlying schemes in fully homomorphic encryption, we briefly investigated

basic concepts of lattice-based cryptography. For further VIP Secure Hardware teams to investigate further on this topic, we suggest the following guideline for future work:

1. To construct a specialized encryption scheme that is capable of efficiently performing the operations in the Obtention Scenario. However, according to our current understanding, this would be really hard to achieve.

2. In order to understand the mathematically behind fully homomorphic encryption, "Fully Homomorphic Encryption over the Integers" [10] is another encryption scheme proposed a year later that implements FHE using bootstrapping without sophisticated mathematical knowledge about ideal lattices.

3. To investigate further in Lattice-based cryptography, it is necessary to learn more about lattice concepts in order to understand purely lattice-based encryption schemes. More knowledge about lattice concepts are necessary and giving an example of a secure and purely lattice-based encryption scheme would significantly deepen our current understanding.

References

[1] Abbas Acar, Hidayet Aksu, A. Selcuk Uluagac, and Mauro Conti. A survey on homomorphic encryption schemes: Theory and implementation, 2017.

[2] Abbas Cheddad, Joan Condell, Kevin Curran, and Paul Mc Kevitt. Review: Digital image steganography: Survey and analysis of current methods. *Signal Process.*, 90(3):727–752, March 2010.

[3] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. Cryptology ePrint Archive, Report 2016/870, 2016. `https://eprint.iacr.org/2016/870`.

[4] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast fully homomorphic encryption library, August 2016. https://tfhe.github.io/tfhe/.

[5] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009.

[6] Craig Gentry and Shai Halevi. Implementing gentry's fully-homomorphic encryption scheme. Cryptology ePrint Archive, Report 2010/520, 2010. `https://eprint.iacr.org/2010/520`.

[7] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. *An Introduction to Mathematical Cryptography*. Springer Science+Business Media, 2014.

[8] Vignesh Raman and Dr. Vincent Mooney. Security for public infrastructure security for public infrastructure using homomorphic encryption. *Georgia Institute of Technology*, 2019.

[9] R.L. Rivest, L. Adleman, and M.L. Dertouzos. On data banks and privacy homomorphisms. In *Foundations on Secure Computation, Academia Press*, pages 169–179, 1978.

[10] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, pages 24–43, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[11] A. C. Yao. Protocols for secure computations. In *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*, pages 160–164, Nov 1982.