Fully Homomorphic Encryption with Applications to Privacy-Preserving Machine Learning

Citation

Permanent link
https://nrs.harvard.edu/URN-3:HUL.INSTREPOS:37376404

Terms of Use
This article was downloaded from Harvard University’s DASH repository, and is made available under the terms and conditions applicable to Other Posted Material, as set forth at http://nrs.harvard.edu/urn-3:HUL.InstRepos:dash.current.terms-of-use#LAA

Share Your Story
The Harvard community has made this article openly available. Please share how this access benefits you. Submit a story.

Accessibility
Fully Homomorphic Encryption with Applications to Privacy-Preserving Machine Learning

A Thesis
Presented By

Michael C. Gul

To the Departments of
Mathematics and Computer Science
In Partial Fulfillment of the Degree of
Bachelor of Arts with Honors

Harvard College
Cambridge, Massachusetts
March 20 2023

Advisor: Professor Boaz Barak
Abstract

There are two dominant trends today that appear to be mutually exclusive: on the one hand, machine learning services that provide accurate predictions based on personal data have become widespread, but on the other hand, data privacy has become a paramount concern for many people. Furthermore, the use of machine learning is heavily restricted in situations like healthcare where it would be most impactful due to regulations preventing the sharing of private data. Fully Homomorphic Encryption (FHE) is the magic bullet that lets us “have our cake and eat it too,” allowing users to send their data to a remote machine learning provider and receive accurate predictions while mathematically guaranteeing complete data security. In this thesis, I give an exposition of Fully Homomorphic Encryption from first principles. I present the mathematical foundations of FHE, notably the Learning With Errors problem (and its Ring variant) used to prove FHE schemes secure, and then I describe two popular FHE schemes. Finally, I survey how FHE is currently used for machine learning, with a particular focus on settings where FHE unlocks opportunities that are otherwise infeasible due to privacy concerns. The primary contribution of this work is the completeness of its exposition, which takes a reader with no cryptography background to the forefront of current research in this revolutionary technology.
Contents

1 Introduction 5
  1.1 Overview ............................................................. 6
    1.1.1 Notation ...................................................... 7
  1.2 Cryptography Preliminaries ....................................... 7
    1.2.1 Public Key Cryptography ..................................... 12

2 Lattice-based Encryption and Learning With Errors 13
  2.1 Lattice Problems .................................................. 15
    2.1.1 Decision LWE ................................................ 17
  2.2 Encryption from Lattice Problems .................................. 18
  2.3 Ring LWE .......................................................... 20
    2.3.1 Algebraic Number Theory Background ........................ 21
    2.3.2 Ideal Lattices ............................................... 23
    2.3.3 The RLWE problem ........................................... 23
    2.3.4 Encryption from RLWE ....................................... 25
    2.3.5 Efficiency Advantages of RLWE .............................. 27

3 Fully Homomorphic Encryption 28
  3.1 Basic Definitions ................................................ 28
  3.2 A Brief History of FHE .......................................... 30
  3.3 FHE Example: GSW ............................................... 32
    3.3.1 GSW Bit Operations ....................................... 32
    3.3.2 The GSW Scheme ........................................... 33
    3.3.3 Circuits and Bootstrapping ................................ 38
  3.4 Advanced FHE: CKKS ............................................. 41
    3.4.1 CKKS Encoding .............................................. 41
    3.4.2 CKKS Encryption ........................................... 43
    3.4.3 CKKS Homomorphic Operations ............................... 44

4 Machine Learning Applications 48
  4.0.1 Why is Fully Homomorphic ML Useful? ....................... 49
  4.1 Machine Learning Background ..................................... 50
    4.1.1 Training Neural Networks .................................. 52
  4.2 How FHE is used in ML ........................................... 54
    4.2.1 A Brief History of Fully Homomorphic ML ................. 55
4.3 Federated Learning and Multikey FHE ................................. 56
  4.3.1 Multikey FHE Background .......................................... 56
  4.3.2 Federated Learning .................................................... 57
  4.3.3 A Fully Homomorphic Federated Learning Protocol ............ 57

5 Conclusion ................................................................. 62
Acknowledgements

Writing a thesis was incredibly fulfilling, and I would like to thank everyone who helped me along the way. First and foremost, I would like to thank my thesis advisor Professor Boaz Barak for all of his help and guidance at every step of this process, from helping me decide on a topic to reading over my drafts. Professor Barak also introduced me to cryptography, and FHE in particular, in the amazing CS227. I would also like to thank my shadow advisor in the math department, Professor Cliff Taubes, for his helpful advice and comments. Both advisors were incredibly generous with their time, and I owe them a debt of gratitude. I would like to thank Professor Salil Vadhan as well for generously serving as my thesis reader.

Thank you to Abigail for all of your support and love throughout this process, and for putting up with me when I was stressed and busy. Thank you to my parents and Emily for being so supportive throughout college and for being excited about my thesis, even if my dad is still disappointed that I haven’t turned it into a successful startup. And, thanks to my friends for providing much-needed distractions.
Chapter 1

Introduction

Working with encrypted data can at most store or retrieve the data for the user; any more complicated operations seem to require that the data be decrypted before being operated on.

On Data Banks and Privacy Homomorphisms

Suppose that Alice wants to send a letter to her friend Bob, and she wants to ensure that the contents of the letter remain totally secret. However, there is a significant obstacle to this task: Bob lives in China and does not speak English, but Alice does not know Mandarin, and neither wants to consult a translator in order to preserve the secrecy of the letter’s contents. A mutual friend, Charlie, offers a solution. Charlie takes Alice’s letter, and produces a translated version without opening the envelope and seeing the contents, thus preserving the secrecy of Alice’s message.

Such a story seems ridiculous; clearly, in order to translate the letter, Charlie must read the contents, which would violate the letter’s secrecy. Conversely, if Alice were to take measures to ensure the letter’s contents remain secret, then it seems impossible to be able to translate the letter despite such measures. However, as surprising and counterintuitive as it seems, fully homomorphic encryption (FHE) can solve this problem, allowing Alice to have her cake and eat it too: we can evaluate all kinds of functions on encrypted data, while simultaneously being able to prove mathematically that our secrets remain beyond the reach of any adversary. A useful analogy is to think of a lockbox with gloves built into its wall. We lock the box via encryption, manipulate the contents however we wish with the gloves, and then unlock it to reveal the finished contents.

The development of efficient fully homomorphic encryption has immense societal benefits, since it unlocks the power of machine learning for fields where privacy requirements otherwise prevent the use of private data. The medical field stands to benefit tremendously from FHE, since machine learning can help doctors detect disease earlier and thus potentially save lives, but restrictions like HIPAA in the US and the GDPR in the EU limit the sharing of
patient data. Moreover, even when the data can be legally obtained, studies have shown that patients are reluctant to participate in research that involves genetic data due to privacy concerns [FGMJ19], hampering researchers’ abilities. As we’ll see, FHE allows doctors to obtain accurate classification of tumors based on genetic data while complying with data privacy regulations [HPC+22]. It also enables hospitals to collaborate on training machine learning models, giving them more accurate predictions without revealing their data to one another [WCK+22]. Moreover, the benefits of FHE extend beyond healthcare. For example, in the financial setting, it can be used by banks to better detect fraud without revealing information about innocent users’ finances.

In this work, we offer an exposition of fully homomorphic encryption, with a particular eye towards machine learning applications. Informally, a homomorphism is a function between two sets that preserves the structure of a specific operation, so a function \( f \) is homomorphic with respect to addition if \( f(x + y) = f(x) + f(y) \). Thus, a fully homomorphic encryption scheme allows us to evaluate arbitrary functions on encrypted data. For example, if \( E \) is our encryption function and \( x \) and \( y \) are two integers, there is a way to obtain an encryption of \( xy \) given \( E(x) \) and \( E(y) \) using a homomorphic multiplication function. We can apply FHE to more complex functions, such as that which translates from English to Spanish, allowing us to transform \( E(\text{“hello”}) \) into \( E(\text{“hola”}) \) without losing any security. Fully homomorphic encryption was first realized by Craig Gentry in 2009 ([Gen09]), and since then many new improvements have made it more and more efficient, bringing this groundbreaking technology ever-closer to widespread use.

1.1 Overview

The main contribution of this work lies in the completeness of the exposition, which builds up from first principles to cutting edge FHE schemes and their applications. This thesis is meant to make this fascinating and useful technology accessible to a mathematically mature student with no background in cryptography, number theory, and/or machine learning. To that end, we will spend the rest of this chapter reviewing the basic vocabulary of cryptography, defining our notions of security that we will use when formally analyzing FHE schemes. Next, in Chapter 2, we present the mathematics that underpins most modern cryptography, including all common FHE schemes. We will first describe the learning with errors (LWE) problem, summarize the reductions that establish its intractability, and present an encryption scheme based on it. Then, after reviewing some algebraic number theory, we will see Ring LWE, a more efficient version of LWE. In Chapter 3, having seen the necessary mathematics to construct FHE, we will review the history of FHE, describing two notable schemes in full detail: GSW, invented by Gentry, Sahai, and Waters in [GSW13], and CKKS, invented by Cheon, Kim, Kim and Song in [CKKS17]. Finally, we conclude in Chapter 4 by looking at how FHE is used in machine learning. We will both look at how people have dealt with the challenges of realizing efficient fully homomorphic ML, as well as some use cases where FHE enables us to use ML in ways that were previously infeasible.
1.1.1 Notation

Throughout this work, we will use the standard mathematical notation for common sets: \( \mathbb{N} \) for the natural numbers, \( \mathbb{Z} \) for the integers, \( \mathbb{R} \) for the real numbers, and \( \mathbb{C} \) for the complex numbers. Superscript denotes the cartesian product, so \( \mathbb{Z}^3 \) is \( \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z} \), \( X^{m \times n} \) refers to the set of \( m \times n \) matrices over \( X \), and \( \mathbb{Z}_n \) refers to the integers modulo \( n \). We will denote sampling \( x \) from a distribution \( D \) by \( x \leftarrow D \), and we denote sampling \( x \) uniformly from the set \( S \) by \( x \leftarrow S \) (it will be clear from context which is the case). We say a function \( f \) is \( O(g) \) if \( f \) is asymptotically bounded by \( g \), i.e. if there exists \( C, x_0 \in \mathbb{R} \) such that \( |f(x)| \leq C \cdot g(x) \) for all \( x \) greater than \( x_0 \).

We write \( x = \text{poly}(n) \) when \( x \) is a polynomial function of \( n \), but the exact polynomial isn’t important. The inner product of two vectors, \( \langle x, y \rangle \), is \( \sum x_i y_i \). When taking the norm of a vector, we’ll use the notations \( \|v\|_1 = |v_1| + \ldots + |v_n| \), \( \|v\|_2 = \sqrt{v_1^2 + \ldots v_n^2} \), and \( \|v\|_\infty = \max |v_1|, \ldots, |v_n| \). \( \{0, 1\}^* \) refers to the set of binary strings of arbitrary length.

1.2 Cryptography Preliminaries

In order to prove that an encryption scheme is secure, we must first rigorously define a notion of security. We will mostly be following the definitions from [Bar21], but occasionally using [BS20].

**Definition 1.2.1.** Let \( \ell, C \) be two functions \( \mathbb{N} \to \mathbb{N} \). Then, the pair of polynomial-runtime functions \( E : (\{0, 1\}^* \times \{0, 1\}^*) \to \{0, 1\}^* \), \( D : (\{0, 1\}^* \times \{0, 1\}^*) \to \{0, 1\}^* \) that input and output bitstrings form a valid encryption scheme with plaintext length function \( \ell \) and ciphertext length function \( C \) if for every \( n \in \mathbb{N}, k \in \{0, 1\}^n \), and \( m \in \{0, 1\}^{\ell(n)} \), the following two conditions hold:

\[
E(k, m) = C(n) \quad D(k, E(k, m)) = m
\]

\( E \) is the encryption function and \( D \) is the decryption function, and the bitstrings \( m, k, \) and \( c \) are referred to as the message/plaintext, the key, and the ciphertext respectively. Thus, our first condition states that if we restrict \( E \) to keys of length \( n \), its domain is strings of length \( \ell(n) \) and its image consists of strings of length \( C(n) \). The second condition ensures validity, i.e. that decrypting an encrypted message will always recover the original message. The key \( k \) is usually written as a subscript, such as \( E_k \) and \( D_k \).
While this definition suffices to show validity, we have not yet shown how an encryption scheme can be secure. After all, the completely insecure scheme $E_k(m) = m, D_k(c) = c$ technically satisfies these requirements, but it is not exactly keeping the message secret. Intuitively, we want our encryption scheme to be designed such that it is impossible to recover the message $m$ given the ciphertext $c = E_k(m)$. In other words, the only possible way for to obtain $m$ is by random guessing, and therefore any attempt to recover given $E_k(m)$ must succeed with a probability no higher than a random guess. We formalize this intuition as follows:

**Definition 1.2.2.** A valid scheme $(E, D)$ is **perfectly secret** if for all sets of plaintexts $M \subset \{0, 1\}^n$ and for any adversary Eve, if $m \leftarrow M$ and $k \leftarrow \{0, 1\}^n$, Eve guesses $m$ when given $E_k(m)$ with probability at most $1/|M|$.

This definition is sometimes presented slightly differently, such as in [BS20]:

**Definition 1.2.3.** $(E, D)$ is **perfectly secret** if for all pairs of plaintexts $m_0, m_1 \in \{0, 1\}^n$ and for any adversary Eve, if $b \leftarrow \{0, 1\}$ and $k \leftarrow \{0, 1\}^n$ are chosen at random, then Eve guesses $b$ when given $E_k(m_b)$ with probability at most $1/2$.

We claim that these definitions are equivalent. It’s trivial that the first implies the second (take $M = \{m_0, m_1\}$), so we focus on the other direction:

**Proof.** Suppose that $(E, D)$ is perfectly secret according to the second definition, and assume for the sake of contradiction that there exists a set $M$ and an adversary Eve that can guess a given $m$ given $E_k(m)$ with probability greater than $1/|M|$. Fix $m_0 = 0^n$. Then, our assumption is equivalent to the fact that

$$\Pr[Eve(E_k(m)) = m] > \frac{1}{|M|}$$

Where $m$ is a random element of $M$, and the probability is taken over the choices of $m$ from $M$ and $k$ from $\{0, 1\}^n$. Note that $Eve(E_k(m)) = m$ is shorthand for “Eve outputs $m$ after seeing the ciphertext $E_k(m)$.” On the other hand, observe that the string $m' = Eve(E_k(m_0))$ is independent of our choice of $m$, and thus the probability over the choice of $m \in M$ that $m' = m$ is just $1/|M|$. Now, let $I$ be the Bernoulli random variable that is 1 if and only if $Eve(E_k(m)) = m$, and let $I'$ be the Bernoulli random variable that is 1 if and only if $E_k(m_0) = m$. We can thus rephrase the above as

$$\mathbb{E}[I] > \frac{1}{|M|} \quad \text{and} \quad \mathbb{E}[I'] \leq \frac{1}{|M|}$$

Where the expectation is being taken over the choice of $m$. By linearity of expectation, it follows that $\mathbb{E}[I - I'] > 0$. Now, recall the probabilistic fact that if the expectation of an event is nonzero, then there must be some element in the sample space such that the event is true with nonzero probability. In our case, we were taking the expectation over the choice of $m$ from $M$, and hence there exists a message $m_1 \in M$ such that

$$\Pr_{k \leftarrow \{0, 1\}^n}[Eve(E_k(m_1)) = m_1] - \Pr_{k \leftarrow \{0, 1\}^n}[Eve(E_k(m_0)) = m_1] > 0$$
We can use this fact to crack the perfect secrecy of \((E, D)\). Let the adversary \(Eve'\) feed a ciphertext \(c\) to \(Eve\). If \(Eve(c) = m_1\), \(Eve'\) outputs \(b = 1\), and if \(Eve(c) = m_0\), \(Eve'\) flips a coin and outputs either 0 or 1 with 50% probability each. Therefore, when given \(c = E_k(m_b)\) for randomly chosen \(b \leftarrow \{0, 1\}\), the probability that \(Eve'\) guesses \(b\) is
\[
\Pr[Ev'\left(E_k(m_b)\right) = b] = \frac{1}{2} \left( \Pr[Ev'\left(E_k(m_0)\right) = 0] + \Pr[Ev'\left(E_k(m_1)\right) = 1] \right)
\]
\[
= \frac{1}{2} \left( 1 + \Pr[Ev'\left(E_k(m_1)\right) = 1] - \Pr[Ev'\left(E_k(m_0)\right) = 1] \right)
\]
\[
= \frac{1}{2} + \frac{1}{2} \left( \Pr[Ev'\left(E_k(m_1)\right) = 1] - \Pr[Ev'\left(E_k(m_0)\right) = 1] \right) \tag{1.1}
\]
Since the term in the parentheses is nonzero, \(Eve'\) guesses \(b\) with probability greater than 1/2, violating Definition 1.2.3. This gives us our contradiction, so we conclude that the two definitions of perfect security are equivalent. \(\square\)

The canonical example of a perfectly secure encryption scheme is the “one time pad.” For key \(k \in \{0, 1\}^n\) and message \(m \in \{0, 1\}^n\), this scheme is given by
\[
E_k(m) = k \oplus m \quad D_k(c) = k \oplus c
\]
Where \(\oplus\) is the bitwise XOR operation, which satisfies \(0 \oplus 0 = 1 \oplus 1 = 0\) and \(1 \oplus 0 = 0 \oplus 1 = 1\). The validity of this scheme follows directly from properties of \(\oplus\), since
\[
D_k(E_k(m)) = k \oplus (k \oplus m) = (k \oplus k) \oplus m = 0^n \oplus m = m \tag{1.2}
\]
Moreover, the scheme is perfectly secure. The key \(k\) is chosen uniformly from \(\{0, 1\}^n\), and the map \(m \mapsto k \oplus m\) is a bijection, so \(E_k(m)\) is a uniform random variable over \(\{0, 1\}^n\) from the perspective of an adversary that doesn’t know \(k\). Since it is impossible to distinguish between two uniform distributions, it is impossible for an adversary to distinguish between \(E_k(m)\) and \(E_k(m')\) for any pair of messages \(m, m'\), giving us perfect security.

Our dual definitions of perfect security do a great job capturing our intuition about what a secure cryptosystem “should” do, and more importantly, they have enabled us to rigorously prove that an encryption scheme is secure. However, perfect security is quite a strong definition. Suppose that we constructed a scheme which is not perfectly secure, but even if every computer in the world were marshalled together and given millions of years to try to crack the scheme, any advantage obtained would be negligible. Such a scheme would still be incredibly useful, since for all intents and purposes, it might as well be perfectly secure. We can formalize this idea of “might as well be perfectly secure” by rigorously defining our terms.

**Definition 1.2.4.** A function \(f : \mathbb{N} \to [0, \infty)\) is negligible if for every polynomial \(p\), there exists \(N > 0\) such that \(n > N\) implies that \(f(n) < \frac{1}{p(n)^c}\).\(^1\)

Negligible functions are useful for two reasons. First of all, they give us a formal tool to describe something as being small enough that we don’t need to worry about it. A negligible

\(^1\)To someone familiar with big-O notation, this is equivalent to \(f\) being \(O(1/n^c)\) for all \(c \in \mathbb{N}\).

9
probability of success means that our scheme is secure for all intents and purposes. The function \( f(n) = 1/n \) is not negligible, since it is still a significant value for many values of \( n \), but \( g(n) = 2^{-n} \) is negligible, since it approaches zero incredibly rapidly. Second, negligible functions obey useful composition laws. If \( f \) and \( g \) are negligible, then \( f + g \) is negligible too, and if \( p \) is a polynomial, then the product \( p(n)f(n) \) is negligible [Bar21]. Since we typically don’t care what the particular negligible function is, we will often denote an arbitrary negligible function as \( \text{negl}(n) \). Now that we’re equipped with a definition of “small enough that we don’t need to worry about it,” we can present a more practical security definition:

**Definition 1.2.5.** Let \((E, D)\) be an encryption scheme with key length \( n \) and message length \( \ell = \ell(n) \), and let Eve be an adversary that receives a ciphertext and outputs a bit. The **computational security game** for Eve is as follows:

1. Two plaintexts \( m_0, m_1 \) are randomly chosen from \( \{0, 1\}^\ell \), and \( k \) and \( b \) are randomly and uniformly sampled from \( \{0, 1\}^n \) and \( \{0, 1\} \).

2. Eve receives \( c = E_k(m_b) \), and outputs a guess \( b' \).

3. Eve wins the game if \( b' = b \).

**Definition 1.2.6.** \((E, D)\) is **computationally secure** if every polynomial time Eve wins the game with probability at most \( \frac{1}{2} + \text{negl}(n) \).

Computational security is closely related to the notion of indistinguishability:

**Definition 1.2.7.** Let \( \{X_n\}_{n \in \mathbb{N}} \) and \( \{Y_n\}_{n \in \mathbb{N}} \) be families of distribution, with a function \( m : \mathbb{N} \to \mathbb{N} \) such that \( X_n \) and \( Y_n \) are distributions over \( \{0, 1\}^{m(n)} \) for all \( n \in \mathbb{N} \). Then, \( \{X_n\}_{n \in \mathbb{N}} \) and \( \{Y_n\}_{n \in \mathbb{N}} \) are **computationally indistinguishable** — often denoted \( \{X_n\}_{n \in \mathbb{N}} \approx \{Y_n\}_{n \in \mathbb{N}} \) — if for every polynomial \( p \) and for sufficiently large \( n \in \mathbb{N} \),

\[
\left| \Pr[A(X_n) = 1] - \Pr[A(Y_n) = 1] \right| \leq \frac{1}{p(n)}
\]

For every algorithm \( A \) that runs in at most \( p(n) \) steps. In other words, no efficient algorithm can distinguish between the two distributions with non-negligible advantage.

Often, \( n \) is clear from context (for encryption schemes \( n \) is usually the key length), so we just say that \( X \) and \( Y \) are computationally indistinguishable, or \( X \approx Y \). Therefore, our above condition is equivalent to the fact that for every polynomial-time algorithm \( A \),

\[
\left| \Pr[A(X) = 1] - \Pr[A(Y) = 1] \right| \leq \text{negl}(n)
\]

We can thus redefine computational security as the distributions \( E_k(m_0) \) and \( E_k(m_1) \) being computationally indistinguishable for all pairs of messages. The following fact about indistinguishability will be useful:

**Lemma 1.2.1.** If \( X \approx Y \) and \( Y \approx Z \) then \( X \approx Z \).
Proof. We will prove the contrapositive: suppose there exists an efficient adversary $A$ that distinguishes $X$ and $Z$ with probability $\epsilon$ for non-negligible $\epsilon$. To simplify the notation, let $p_X = \Pr[A(X) = 1]$, and so on for $Y$ and $Z$. Then,
\[
\epsilon = |p_X - p_Z| = |p_X - p_Y + p_Y - p_Z| 
\leq |p_X - p_Y| + |p_Y - p_Z|
\]
By the triangle inequality. It is impossible for the sum of two negligible values to be non-negligible, so either $|p_X - p_Y|$ or $|p_Y - p_Z|$ is non-negligible. In the former case $X \not\approx Y$ and in the latter case $Y \not\approx Z$, as desired.

In practice, computational security does not adequately capture the behavior of most adversaries. For example, what if our adversary can choose messages that it has encrypted? Perhaps the adversary can gain an advantage by carefully choosing the messages to be encrypted in a way that gives it insight into how the encryption scheme works. This concern is not theoretical: in World War 2, the US Navy suspected that they had cracked the Japanese codeword AF as meaning “Midway,” so they sent messages about Midway island being low on water and monitored the Japanese lines for AF, which indeed appeared ([KL14]). This intelligence helped the US win the battle of Midway, a major turning point in World War 2. The following definition would have been quite helpful for the Japanese:

**Definition 1.2.8** ([Bar21, KL14]). An encryption scheme $(E, D)$ is **Chosen Plaintext Attack Secure**, or **CPA Secure**, if every polynomial time adversary $Eve$ wins the following game with probability at most $\frac{1}{2} + \text{negl}(n)$
1. $k$ is sampled randomly from $\{0, 1\}^n$
2. $Eve$ gets $1^n$ as input
3. $Eve$ has access to an oracle for $E_k$, meaning that for polynomial amount of rounds, $Eve$ chooses messages $m_i$ and receives $E_k(m_i)$
4. $Eve$ chooses $m_0, m_1$, and receives $c^* = E_k(m_b)$ for randomly chosen $b$.
5. $Eve$ gets to interact with the oracle $E_k(\cdot)$ for another polynomial number of rounds.
6. $Eve$ outputs a bit $b'$ and wins if and only if $b' = b$.

While it’s possible to create definitions that are even stronger than CPA Security, it will be sufficient for all of our purposes. In addition to its strong security guarantees, CPA Security is useful in that it is secure under concatenation. Suppose that $(E, D)$ is a CPA Secure scheme with 1-bit messages. Then, we can obtain a scheme $(E', D')$ with $n$-bit messages by simply concatenating:

$E'_k(x_1 \ldots x_n) = E_k(x_1) || \ldots || E_k(x_n)$
$D'_k(c_1 \ldots c_n) = D_k(c_1) || \ldots || D_k(c_n)$

We omit the proof that $(E', D')$ is CPA Secure, but it follows directly from the definition; intuitively, it is impossible to gain non-negligible advantage in breaking $E'$ without being able to distinguish between ciphertexts of $E$ ([KL14]). This fact is extremely helpful when designing CPA secure encryption schemes, since we only need to create a scheme that works for a single bit in order to obtain a scheme that works for bitstrings of arbitrary length.
1.2.1 Public Key Cryptography

The definitions that we have seen so far are all private key, meaning that if Alice and Bob want to exchange secret messages, they must first agree on a key $k$ that they each have access to. But what if this agreement isn’t possible? What if I want to send my encrypted credit card information to Amazon, but Jeff Bezos can’t find time to meet up with me to determine a secret key? Intuitively, this should be impossible. In order to communicate privately, there must be some information that only the two parties involved know, and “Public Key Cryptography” sounds like an oxymoron. But yet, as unbelievable as it seems, public key encryption does exist.

Definition 1.2.9. A Public Key Encryption Scheme consists of 3 algorithms $G, E, D$:

- $G$ is the key generation algorithm, and it probabilistically outputs a pair $(e, d)$ on input $1^n$.
- $E$ and $D$ are the encryption and decryption algorithms, and they work just as in private key, but now $E$ uses $e$ as its key and $D$ uses $d$.
- For validity, we require that for all $m$, $D_d(E_e(m)) = m$ with probability $1 - \text{negl}(n)$ over the choice of $(e, d)$ and the internal randomness of $E$ and $D$.

Observe that they key difference here is in key generation: instead of having one key $k$ which is kept private, there are now two keys. $e$ is known as the public key, and $d$ is known as the private (or secret) key; in other texts they are sometimes written as $(pk, sk)$. We can modify our security definitions for this new setting:

Definition 1.2.10 (Public Key CPA Security). A public key encryption scheme $(G, E, D)$ is CPA Secure, if every polynomial time adversary $Eve$ wins the following game with probability at most $\frac{1}{2} + \text{negl}(n)$

1. $(e, d) \leftarrow G(1^n)$, and the public key $e$ is given to $Eve$.
2. $Eve$ chooses two messages $m_0, m_1$, and receives $c^* = E_k(m_b)$ for randomly chosen $b$.
3. $Eve$ outputs a bit $b'$ and wins if and only if $b' = b$.

This definition is slightly different from private CPA security due to the new setting. Since $e$ is public and $E$ runs in polynomial time, $Eve$ doesn’t need an oracle to $E$ — she can just compute $E_e(m)$ herself for any message that she wants.
Chapter 2

Lattice-based Encryption and Learning With Errors

Cryptography is replacing trust with mathematics

Boaz Barak

In this chapter, we will learn the mathematics that takes the place of trust in fully homomorphic encryption. The broader family of encryption schemes that we will consider is known as lattice-based encryption, since its hardness is based on problems on lattices that are conjectured (and widely believed) to be impossible to solve efficiently. After briefly looking at the Learning with Errors problem ([Reg05]), we will define lattices and some computational problems associated with them. Then, we will review Regev’s reduction that establish the hardness of Learning with Errors, and we will see an encryption scheme whose security relies on that hardness. Finally, after taking a detour to recall some basic algebraic number theory, we will look at the Ring Learning with Errors problem, a more efficient variant of Learning with Errors, and its associated reductions and encryption schemes. Though we will prove many of the results discussed, some of them use tools that are well beyond the scope of this work, and as such those proofs will be summarized, with references given to the complete proof in case the reader desires the full details.

We begin by defining a few terms and notations.

Definition 2.0.1. The finite field $\mathbb{Z}_q$, where $q$ is a prime number, is the set $\{0, 1, \ldots, q-1\}$, with the operations of addition and multiplication modulo $q$.

Definition 2.0.2. A vector $v$ in $\mathbb{Z}_q^n$ has $n$ elements in $\mathbb{Z}_q$, i.e. $v = (v_1, \ldots, v_n)$ for $v_i \in \mathbb{Z}_q$. Similarly, $\mathbb{Z}_q^{m \times n}$ consists of the $m \times n$ matrices with entries in $\mathbb{Z}_q$.

Definition 2.0.3. The inner product of two vectors $x, y \in \mathbb{Z}_q^n$, $\langle x, y \rangle$, is equal to $x_1y_1 + x_2y_2 + \ldots + x_ny_n$.

Many popular lattice-based schemes are based on Learning With Errors, a problem first posed by Oded Regev in 2005. Informally, Learning With Errors (LWE) relies on solving modular linear equations that have a slight error term introduced.
For example, consider the following system of equations modulo 131, with error terms $\epsilon_i$:

\[
\begin{align*}
12x_1 + 12x_2 + 12x_3 &= 48 + \epsilon_1 \pmod{131} \\
36x_1 + 24x_2 + 24x_3 &= 12 + \epsilon_2 \pmod{131} \\
12x_1 + 12x_2 &= 24 + \epsilon_3 \pmod{131}
\end{align*}
\]

(2.1)

If there is no error, then this problem is quite straightforward to solve; it could easily be a homework problem for an introductory linear algebra student. The Gaussian Elimination algorithm will efficiently give a solution to any such system of linear equations. In this case, it is straightforward to recover $x = (124, 9, 2)$.

But what happens if the error terms are not zero? Gaussian Elimination fails entirely in such a scenario, since it is incredibly ill-prepared to handle errors in the equation. Gaussian Elimination works by adding scalar products of equations to other equations, but this step magnifies the error! Indeed, we can consider what would happen here with a smaller example and concretely small error terms:

\[
\begin{align*}
12x_1 + 12x_2 + 12x_3 &= 48 - 2 \pmod{131} \\
36x_1 + 24x_2 + 24x_3 &= 12 + 1 \pmod{131} \\
12x_1 + 12x_2 &= 24 - 1 \pmod{131}
\end{align*}
\]

(2.2)

The first step in Gaussian Elimination might be to subtract twice the first row from the second:

\[
\begin{align*}
12x_1 + 12x_2 + 12x_3 &= 46 \pmod{131} \\
12x_1 &= 13 - 92 \pmod{131} \\
12x_1 + 12x_2 &= 23 \pmod{131}
\end{align*}
\]

(2.3)

Then, we subtract the third row from the first, and then the second row from the third:

\[
\begin{align*}
12x_3 &= 46 - 23 \pmod{131} \\
12x_1 &= 52 \pmod{131} \\
12x_1 + 12x_2 &= 23 \pmod{131} \\
12x_3 &= 23 \pmod{131} \\
12x_1 &= 52 \pmod{131} \\
12x_2 &= 102 \pmod{131}
\end{align*}
\]

(2.4)

(2.5)

Since 131 is prime, we can divide by 12 modulo 131 to obtain $x_1 = 48$, $x_2 = 74$, and $x_3 = 122$. This is quite different than the actual answer, showing how these small errors totally thwarted us. We can formalize this problem as follows:

**Definition 2.0.4** (*LWE*, [Reg05]). Let $q$ be a prime integer whose size is polynomial in $n$, and let $\chi$ be a probability distribution on $\mathbb{Z}_q$. For some $s \in \mathbb{Z}_q^n$, suppose that we have a series of equations of the form

$$\langle s, a_i \rangle + e_i = b_i \pmod{q}$$

Where the $a_i$ are chosen uniformly and independently from $\mathbb{Z}_q^n$ and $e_i$ are sampled independently from $\chi$. The $a_i$ and $b_i$ are public, and $s$ and the $e_i$ are secret. Then, the **learning with errors problem** LWE$_{q,\chi}$ is the problem of recovering $s$ given $\langle s, a_i \rangle, b_i + e_i$. 

14
Our example highlights the fact that Gaussian elimination is not able to solve LWE, but it is not immediately clear that no efficient algorithm can out-smart the added error. In order to formalize this idea, we rely on some problems related to lattices.

## 2.1 Lattice Problems

**Definition 2.1.1** ([Pei16]). A lattice is a subset $L \subset \mathbb{R}^n$ that satisfies the following two requirements:

1. $L$ is an additive subgroup, meaning that $0 \in L$, $L$ is closed under addition ($x, y \in L \implies x + y \in L$), and inverses are contained in $L$ ($x \in L \implies -x \in L$).
2. $L$ is discrete, so for any $x \in L$, there exists a neighborhood $U \subset \mathbb{R}^n$ containing $x$ such that $y \notin U$ for all other $y \in L$.

Informally, an $n$-dimensional lattice is a subset of $\mathbb{R}^n$ that behaves like the vector space of integers, $\mathbb{Z}^n$. By definition, $\mathbb{Z}^n$ is a lattice, as is something like $c\mathbb{Z}^n$ for a constant $c$, and $L = \{2x + y : x, y \in \mathbb{Z}\} \subset \mathbb{R}^2$.

**Definition 2.1.2.** A set of linearly independent vectors $B = \{b_1, \ldots, b_k\}$ is a basis for $L$ if $L$ is the set of integer linear combinations of the vectors in $L$, i.e.,

$$L = \left\{ \sum_{i \leq k} x_i b_i : x_i \in \mathbb{Z} \right\}$$

**Definition 2.1.3.** The shortest vector in $L$ is denoted as $\lambda_1(L)$, and it is equal to $\min_{v \in L \neq 0} \|v\|$. More generally, $\lambda_i(L)$ is the smallest $k$ such that $L$ has $i$ linearly independent vectors of norm $\leq k$.

**Definition 2.1.4.** The dual of $L$, or $L^*$, is the set of $y \in \mathbb{R}^n$ such that $\langle x, y \rangle \in \mathbb{Z}$ for all $x \in L$. It can be shown that $L^*$ is always a valid lattice. For example, the dual of $c\mathbb{Z}^n$ is $\frac{1}{c}\mathbb{Z}^n$.

Many of the problems we will see will use a version of the Gaussian distribution adapted to the lattice setting:

**Definition 2.1.5.** The discrete Gaussian distribution on $L$ with standard deviation $\sigma$, sometimes denoted $D_{L,\sigma}$, has support $L$ and outputs $x$ with probability proportional to $e^{-\|x/2\sigma\|^2}$.

There are a number of computational problems related to various aspects of lattices. Here, we will consider three particular problems which are relevant to Learning With Errors. All three involve approximating certain properties of a lattice to some approximation function $\gamma$ that depends on the dimension.

**Definition 2.1.6** ([Pei16]). Given a basis $B$ of an $n$-dimensional lattice $L$, the **Decisional Approximate Shortest Vector Problem** with approximation function $\gamma : \mathbb{N} \to [1, \infty)$ (GapSVP$_{\gamma}$) is the decision problem of whether $\lambda_1(L) \leq 1$ or $\lambda_1(L) > \gamma(n)$. 

15
Definition 2.1.7 ([Pei16]). Given a basis $B$ with full rank\(^1\) of an $n$-dimensional lattice $L$, the goal of the \textbf{Shortest Independent Vector Problem} (SIVP\(_\gamma\)) is to output a set $S \subset L$ of $n$ linearly independent vectors such that $\|s\| \leq \gamma(n)\lambda_n(L)$ for every $s \in S$.

Definition 2.1.8. Let $L$ be a lattice, and let $d < \lambda_1(L)/(2\gamma(n))$. Given a target point $t$ that is guaranteed to lie within distance $d$ of some $x \in L$, the goal of the \textbf{BDD\(_\gamma\)} problem, or \textbf{Bounded Distance Decoding}, is to output $x$.

Due to the approximation $\gamma$, each problem really represents a class of problems, so choosing a smaller $\gamma$ will make the problem harder to solve. Crucially, these problems are conjectured (and widely believed) to be be impossible to solve efficiently:

\textbf{Conjecture 2.1.1.} There are no polynomial time algorithms that can solve BDD\(_\gamma\), SIVP\(_\gamma\), or GapSVP\(_\gamma\) for any $\gamma(n) = \text{poly}(n)$.

All extant algorithms either take exponential time for a polynomial approximation $\gamma$ or polynomial time for exponential $\gamma$ [Pei16]. Throughout the rest of this work, we will be assuming that this conjecture is true. Moreover, this hardness is believed to include quantum algorithms, which are algorithms that use quantum computation for speedups. Unfortunately, quantum computation is beyond the scope of this work, so we won’t be delving into these algorithms. What’s important for our purposes is that other encryption schemes which are conjectured to be hard in the non-quantum setting, such as RSA, are in fact susceptible to quantum attacks. Even though quantum technology is currently extremely limited, the fact that lattice problems are “quantum-proof” means that they will remain secure even if (or when) efficient large-scale quantum computers are constructed.

We can use the hardness of these lattice problems to show that LWE is also hard. In particular, we will use the technique of a reduction: if we can show that an efficient algorithm to solve LWE can be transformed into an efficient algorithm to solve GapSVP\(_\gamma\) for a $\gamma$ such that GapSVP\(_\gamma\) is conjectured to be unsolvable, then the existence of such an algorithm would contradict the conjectured hardness of GapSVP\(_\gamma\). Therefore, we can safely conclude that no efficient algorithm — even one that relies on quantum computation — exists for LWE.

\textbf{Theorem 2.1.1} (Hardness of LWE, [Reg05]). \textit{Let $\Psi_\alpha$ be the discrete Gaussian distribution on $\mathbb{Z}_q$ with standard deviation $\alpha q$ and rounded to the nearest integer. And, let $m$ be polynomial in $n$, $\alpha \in (0,1)$ and $q \leq 2^{\text{poly}(n)}$ such that $\alpha q > 2\sqrt{n}$. Then, if there exists a polynomial-time algorithm to solve LWE\(_{q,\Psi_\alpha}\), there exists a polynomial-time quantum algorithm to solve GapSVP\(_\gamma\) and SIVP\(_\gamma\) on $n$-dimensional lattices for $\gamma(n) = O(n/\alpha)$}.\(^2\)

Though this reduction is quantum, Peikert later demonstrated a classical reduction with different parameters [Pei09]. We omit the full proof of Regev’s reduction but we will summarize it briefly. Given an oracle to LWE\(_{q,\Psi_\alpha}\), and given access to a polynomial number of samples to the discrete Gaussian distribution on $L^*$ with standard deviation $r$ (for a small $r$), if we have a point $x$ which is known to be within $\alpha q/r\sqrt{2}$ of some point in $L$, we can output the closest unique lattice point. This problem is just Bounded Distance Decoding with distance

---

\(^1\)Meaning that $B$ contains $n$ linearly independent vectors, i.e. $B$ spans $L$

\(^2\)The notation $f = \tilde{O}(g)$ means that $f$ is $O(g \log^k(x))$ for some $k$, since often we can safely disregard such logarithmic terms.
Next, Regev shows that if we can solve BDD with distance $d$ in polynomial time, we can use a quantum algorithm to efficiently sample from the discrete Gaussian distribution on $L^*$ with standard deviation $\sqrt{n}/d$. This problem is known as Discrete Gaussian Sampling, and it forms the core of this reduction. Then, we keep iterating this construction, and when $\alpha p > 2\sqrt{n}$, samples from this distribution can be used to solve $\text{GapSVP}_\gamma$ [Reg10]. Since $\text{GapSVP}_\gamma$ is conjectured to be unsolvable by a polynomial-time algorithm for $\gamma(n) = \tilde{O}(n/\alpha)$, LWE must similarly be impossible to solve efficiently.

### 2.1.1 Decision LWE

The LWE problem is a search problem, since the objective is to find the secret vector $s$ given the LWE samples, but it turns out that there is a decision version that is hard as well.

**Definition 2.1.9.** The Decision LWE Problem, $\text{DLWE}_{q,\chi}$, is the problem of distinguishing between the uniform distribution over $\mathbb{Z}_q^{m\times n} \times \mathbb{Z}_q^m$, and $\{(A, As + e)\}$, where $m = \text{poly}(n)$, $A$ is a uniformly sampled $m \times n$ matrix over $\mathbb{Z}_q$, $s \leftarrow \mathbb{Z}_q^n$ and $e_i \leftarrow \chi$, with $\chi$ being an error distribution over $\mathbb{Z}_q$.

Note that we say algorithm $A$ distinguishes between two distributions $X$ and $Y$ when it has non-negligible advantage in its output on one versus the other. Formally, $A$ distinguishes between $X$ and $Y$ if

$$\left| \Pr[A(X) = 1] - \Pr[A(Y) = 1] \right| \geq \epsilon$$

For some non-negligible $\epsilon$, which we call the advantage. As the name suggests, the idea of an algorithm that can distinguish between two distributions is closely linked to indistinguishability (Definition 1.2.7): two distributions are indistinguishable if and only if there is no algorithm that can distinguish between them.

**Theorem 2.1.2** (Search-to-Decision for LWE). Let $q$ be a prime and $\chi$ be a distribution over $\mathbb{Z}_q$. If there exists a polynomial time algorithm that can solve $\text{DLWE}_{q,\chi}$, then there exists a polynomial time algorithm that can solve $\text{LWE}_{q,\chi}$.

**Proof.** Assume that there exists an algorithm $A$ to distinguish between this distribution and the uniform distribution with advantage $\epsilon = 1/\text{poly}(n)$. We can use this algorithm to create a new algorithm, $B_{i,a}$, which will output 1 if $s_i = a$ for some $i \in [n]$ and $a \in \mathbb{Z}_q$; it follows that $qn$ invocations of this algorithm will give us $s$.

$B_{i,a}$ takes in a pair $(A, y = As + e)$. Let $r$ be a random element of $\mathbb{Z}_q^m$, and let $R_i$ be the matrix whose $i$th column is $r$ and every other column is 0. Consider the modified pair $(A + R_i, y + ar)$. Observe that $(A + R_i)s = As + s_ir$, so if $s_i = a$ then $y + ar = As + e + ar = (A + R_i)s + e$, so $(A + R_i, y + ar)$ takes the LWE distribution. Conversely, if we tried the same modification when $s_i \neq a$, we get the uniform distribution: the matrix $A + R_i$ is uniformly distributed, and since $s_i \neq a$, $y + ar$ is equal to $(A + R_i)s + e + s_ir - ar$. But $(s_i - a)r$ is just a uniformly distributed vector in $\mathbb{Z}_q^m$ because $q$ is prime, and hence $y' = y + ar$ is uniformly distributed as well.

We thus have an efficient construction that converts $(A, y)$ into an $(A', y')$, where $(A', y')$ is uniform if and only if $s_i \neq a$. Now, recall that $A$ has advantage $\epsilon$ in deciding between the two distributions. If $A$ needs $m$ equations to achieve this advantage, $B_{i,a}$ creates $100mn/\epsilon^2$. 

17
equations to call $A$ $100n/\epsilon^2$ times. Then, if $p$ is the probability that $A$ outputs 1 given the uniform distribution, $B_{s,a}$ will output 1 if and only if $A$ outputs 1 at least $(p + \frac{\epsilon}{2})(100n/\epsilon^2)$ times. Since each round is independent, the Chernoff bound tells us that when $s_i = a$ and the samples are from the LWE distribution, if $X_i$ is the event that $A$ outputs 1 on the $i$’th iteration, then

$$\Pr \left[ \sum_{i=0}^{100n/\epsilon^2} X_i - (p + \epsilon)(100n/\epsilon^2) > \frac{\epsilon}{2}(100n/\epsilon^2) \right] \leq 2^{e^{-2(\epsilon/2)^2(100n/\epsilon^2)}} = 2^{-100n}$$

In other words, the probability that $A_i$ outputs 1 less than $(p + \frac{\epsilon}{2})(100n/\epsilon^2)$ — and equivalently that $B_{a,i} = 0$ — is negligible. An identical calculation shows that when $s_i \neq a$, $B_{a,i} = 1$ with negligible probability. $B_{s,a}$ runs in polynomial time, so we can run it $nq = poly(n)$ times to recover $s$ with negligible probability of failure, breaking the hardness of LWE.

### 2.2 Encryption from Lattice Problems

There are many encryption schemes based on LWE, including some of the fully homomorphic schemes that we will see later. Here, we will present a simpler scheme, which is a great introduction to how LWE-based schemes work.

**Definition 2.2.1.** [LWE Encryption. [Reg10, Bar21]] The scheme has a few parameters: the number of equations $m$, the prime modulus $q$, and the noise parameter $\alpha$. In our case, we will let $q$ be a prime between $n^2$ and $2n^2$, $m = \lceil 1.1n \log q \rceil$, and $\alpha = 1/(\sqrt{n} \log^2 n)$. Then, the Learning With Errors Encryption Scheme is as follows:

- **Key Generation:** The private key is a uniformly and randomly sampled $s \leftarrow \mathbb{Z}_q^n$. The public key is $m$ pairs $(a_i, \langle a_i, s \rangle + e_i)$, where $a_i$ is a random vector in $\mathbb{Z}_q^n$ and $e_i \in \mathbb{Z}_q$ is sampled from the discrete Gaussian $\Psi_\alpha$. We can represent this public key as $(A, y)$, where $A$ is the $m \times n$ matrix whose rows are the $a_i$ and $y$ is an $m$–dimensional vector in $\mathbb{Z}_q^m$ whose $i$’th entry is $\langle a_i, s \rangle + e_i$.

- **Encryption:** To encrypt a single bit $b$, randomly and uniformly choose $w \subset \{0, 1\}^m$. Then,

$$E_{A,y}(b) = (w^T A, \langle w, y \rangle + b \lfloor \frac{q}{2} \rfloor)$$

- **Decryption:** $D_s(a, \sigma) = 0$ if and only if $|\langle a, s \rangle - \sigma| < q/4$.

Note that unlike previous schemes we’ve seen, a ciphertext for this scheme is a tuple $(a, \sigma) \in \mathbb{Z}_q \times \mathbb{Z}_q^n$, where $\sigma$ contains the encrypted bit, and $a$ contains extra information that we’ll need for decryption. Informally, observe that the scheme sums a random subset of the equations $\langle a_i, s \rangle + e_i$ by taking the product $\langle w, y \rangle = \langle w, As + e \rangle$ for random $w$, and then we add this to $b \lfloor \frac{q}{2} \rfloor$ to hide the bit $b$. We can show formally that decryption succeeds.

**Theorem 2.2.1.** The scheme from 2.2.1 is valid with high\(^3\) probability.

\(^3\)“High” here means $1 - negl(n)$, so in other words, decryption fails with negligible probability. Such a probability is low enough that we can treat this scheme as always being correct.
Proof. Observe that, by definition of this scheme, \( D_s(E_{A,y}(b)) \) is 0 if and only if
\[
|\langle a, s \rangle - \sigma| = \left| \langle w^T A, s \rangle - \langle w, y \rangle - b \left\lfloor \frac{q}{2} \right\rfloor \right| = \left| \langle w^T A, s \rangle - \langle w, As + e \rangle - b \left\lfloor \frac{q}{2} \right\rfloor \right| \tag{2.6}
\]
Is less than \( q/4 \). But, by properties of the inner product, this is equal to
\[
\left| \langle w, As \rangle - \langle w, As \rangle - \langle w, e \rangle - b \left\lfloor \frac{q}{2} \right\rfloor \right| = \left| \langle w, e \rangle + b \left\lfloor \frac{q}{2} \right\rfloor \right| \tag{2.7}
\]
But since \( \langle w, e \rangle \) is just the sum of some subset of the elements of \( e \), each of which is normally distributed with standard deviation \( \alpha q \), and thus the standard deviation of the sum is at most
\[
\sqrt{m \alpha q} = \sqrt{1.1n\log q} \frac{1}{\sqrt{n\log^2 n}} q < \frac{q\sqrt{n\log n}}{\log^2 n} < \frac{q}{\log n}
\]
The probability that a Gaussian with this standard deviation is at least \( q/4 \) is negligible, so equivalently, the probability that decryption fails is negligible, and this scheme is valid. \( \square \)

We can show that this scheme is CPA Secure according to Definition 1.2.10.

**Theorem 2.2.2 ([Reg05]).** The LWE Encryption scheme described above is CPA Secure

**Proof.** We proceed via reduction to the Decision-LWE. Assume for the sake of contradiction that the scheme is insecure, so there exists an efficient adversary Eve that wins the CPA game with non-negligible advantage. In other words, given the public key \((A, y)\) and the encryption of \( b \), Eve can guess \( b \) with non-negligible advantage (there are only two possible messages for Eve to choose in the CPA game, 0 and 1). Without loss of generality, let \( \Pr[Eve(E_{A,y}(1)) = 1] = \frac{1}{2} + \epsilon \) for a non-negligible \( \epsilon \).

We can use such an Eve to solve Decision LWE. Suppose instead of giving Eve the \((A, y)\) from the key generation algorithm, we just gave her a randomly and uniformly sampled \( A' \leftarrow \mathbb{Z}_{q}^{m\times n} \) and \( y' \leftarrow \mathbb{Z}_{q}^{n} \), as well as \( E_{A',y'}(b) \) for a random \( b \leftarrow \{0,1\} \). We require the following Lemma to proceed:

**Lemma 2.2.1 (Leftover Hash Lemma, [ILL89]).** Fix \( \epsilon > 0 \). Let \( \mathcal{H} \) be a family of universal hash functions \( \mathcal{W} \rightarrow \mathcal{V} \), meaning that for all \( x \neq x' \), \( \Pr_{h \leftarrow \mathcal{H}}[h(x) = h(x')] \) is at most \( \frac{1}{|\mathcal{V}|} \). And, let \( W \) be a random variable over \( \mathcal{W} \) such that \( H_{\infty}(W)^4 \geq \log |\mathcal{V}| + 2 \log\frac{1}{\epsilon} - 2 \). Then, the statistical difference \(^5\) between \((H(W), H)\) and \((V, H)\) is less than \( \epsilon \), where \( H \) is uniformly distributed over \( \mathcal{H} \) and \( V \) is uniformly distributed over \( \mathcal{V} \).

For a proof of the Leftover Hash Lemma, see chapter 11 in [Bar21]. Intuitively, the Lemma says that as long as no particular event from \( W \) is too likely to happen, its image under a hash function is extremely close to uniform.

To apply the lemma, we consider the family \( \mathcal{H} = \{h_A\} \) parameterized by matrices over \( \mathbb{Z}_{q}^{m\times (n+1)} \), with \( h_A : \mathbb{Z}_{q}^{m} \rightarrow \mathbb{Z}_{q}^{n+1} \) given by \( h_A(w) = w^T A \). Observe that this family satisfies

\(^4H_{\infty}\) is the min-entropy function on random variables. For a random variable \( X \) over a set \( S \), if \( p(x) \) is the probability that \( X = x \), and \( p_{\max} \) is \( \max_{x \in S} p(x) \), then \( H_{\infty} = -\log(p_{\max}) \).

\(^5\)The statistical difference between random variables \( X \) and \( Y \) over a set \( S \), denoted \( \Delta(X,Y) \), is
\[
\frac{1}{2} \sum_{x \in S} |\Pr[X = x] - \Pr[Y = x]|\]
the criterion for a universal hash family, since the probability over a random choice of $A$ that $h_A(x) = h_A(x')$ is the probability that $x^T A = x'^T A$, or equivalently the probability that $(x - x')^T A = 0$. For a randomly chosen matrix, this probability is $q^{-(n+1)} \frac{1}{\log q} = \frac{1}{\log q}$, so $H$ is a universal hash family. We can observe that a uniformly distributed $W$ over $\{0,1\}^n$ and $O(1)$ and

Plugging into the equation from the lemma confirms that if $\epsilon$ satisfies $-\log(2^m) = m$, since every $w$ has equal probability of being chosen. Plugging into the equation from the lemma confirms that if $\epsilon = 2^{-n}$, then

$$H_\infty(W) = m \geq \log |Z_q^{n+1}| + 2 \log(2^n) - 2 \geq (n + 1) \log(q) + 2n - 2$$

Which holds because $m = 1.1n \log q$. The Lemma then implies that if $V$ is uniformly distributed over $Z_q^{n+1}$, the statistical distance between $(W^T A, A)$ and $(V, A)$ is less than $2^{-n}$. Observe that computational indistinguishability follows directly from negligible statistical distance. In the case of our experiment, the encryption that $\text{Eve}$ receives is equal to $(w^T A', \langle w, y' \rangle + b \lfloor \frac{q}{2} \rfloor)$. We can view $A'\|y'$ as being a $m \times (n + 1)$ matrix, in which case the Leftover Hash Lemma tells us that $w^T (A'\|y')$ is statistically indistinguishable from the uniform distribution; adding $b \lfloor \frac{q}{2} \rfloor$ to the last coordinate of this vector does not change that.

We have shown that in our experiment of randomly choosing $A', y'$ in lieu of key generation, the resulting distributions of $E_{A', y'}(0)$ and $E_{A', y'}(1)$ are indistinguishable from the uniform distribution (so by Theorem 1.2.1, from each other as well). Now, we can use $\text{Eve}$ to distinguish between the the LWE distribution and $(A', y')$: given access to some pair $(A, y)$, a new adversary $\text{Eve}'$ can simply compute $E_{A, y}(1)$ and feed it to $\text{Eve}$, outputting whatever $\text{Eve}$ outputs. If $(A, y)$ is really $(A', y')$, then the fact that this distribution is indistinguishable from uniform means that $\Pr[\text{Eve}(E_{A', y'}(1)) = 1] \leq \frac{1}{2} + \text{negl}(n)$. Conversely, if $(A, y)$ comes from the LWE distribution, then by our assumption $\Pr[\text{Eve}(E_{A, y}(1)) = 1] = \frac{1}{2} + \epsilon$ for a non-negligible $\epsilon$. Hence $\text{Eve}'$ outputs 1 with probability $\frac{1}{2} + \epsilon$ on the LWE distribution and $\frac{1}{2} + \text{negl}(n)$ on the uniform distribution, contradicting Theorem 2.1.2 and ultimately breaking the hardness of Learning with Errors, which is a contradiction.

In summary, we have the following chain of reductions from the CPA security of LWE Encryption to $\text{GapSVP}_\gamma$:

<table>
<thead>
<tr>
<th>Theorem</th>
<th>Theorem</th>
<th>Theorem</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{GapSVP}_\gamma$</td>
<td>2.1.1</td>
<td>$\text{Search-LWE}$</td>
</tr>
</tbody>
</table>

2.3 Ring LWE

The LWE-based encryption scheme is great in theory, but in practice it is simply too inefficient. The public key is an $m \times n$ matrix with entries in $\mathbb{Z}_q$, so it has size $O(mn \log q)$ bits, and each bit of the message requires $O(n \log q)$ bits of ciphertext, meaning that an $n$-bit string would require $O(n^2 \log q)$ bits. With such a large overhead, this scheme would not be feasible for most real-world uses of encryption. The issue is that each LWE equation $\langle a_i, s \rangle + e_i$ is only a single element of $\mathbb{Z}_q$, but we need to multiply $n$-dimensional vectors in
order to achieve this. In an ideal world, we would have a scheme where each equation gives
us an element of \( \mathbb{Z}_q^n \), so we gain much more security for each \( n \)-dimensional multiplication. Unfortunately, such a scheme does not exist in our current setting of \( \mathbb{Z}_q^n \). The crucial insight of Ring LWE (or RLWE) is that adding some additional algebraic structure allows us to define a product that gives us this more efficient multiplication. Before we get to RLWE, we review some elementary algebraic number theory.

### 2.3.1 Algebraic Number Theory Background

**Definition 2.3.1 ([Art11]).** A **ring** is a set \( R \) equipped with binary operations \( + \) and \( \times \) such that

1. \( (R, +) \) is an abelian group, meaning that \( R \) is closed under \( + \), every element has an additive inverse, addition is associative, and there exists an additive identity \( 0 \).
2. \( \times \) is commutative and associative, and there exists a multiplicative identity \( 1 \).
3. \( + \) and \( \times \) distribute, so \( a \times (b + c) = a \times b + a \times c \)

Other texts define rings slightly differently — in particular, not everyone requires commutativity or the existence of a multiplicative identity — but this definition is most useful for our purposes. The canonical example of a ring is the set of integers \( \mathbb{Z} \) equipped with the usual operations. The Gaussian integers, \( \mathbb{Z}[i] = \{a + bi : a, b \in \mathbb{Z}\} \), also form a ring, with the operations inherited from \( \mathbb{C} \). When a ring has division (i.e. every element has a multiplicative inverse), it is known as a **field**. For example, the ring \( \mathbb{Z} \) is not a field because the element 2 has no integer multiplicative inverse, but \( \mathbb{Q} \) is a field, since \( a/b \) has inverse \( b/a \). Crucially, there are many ways to form rings from existing rings, two of which will be useful for our RLWE purposes: polynomial rings and quotient rings.

**Definition 2.3.2.** Given a ring \( R \), the **polynomial ring** \( R[x] \) consists of all polynomials

\[
p(x) = a_n x^n + \cdots + a_1 x + a_0
\]

Where \( n \) is finite, and all of the \( a_i \) are elements of \( R \). Addition in \( R[x] \) is done component-wise, and multiplication is done according to the distribution law in \( R \).

Thus, the set \( \mathbb{Z}[x] \) is the polynomial ring containing all polynomials in \( x \) with integer coefficients. In order to define quotient rings, we need to define a few more terms:

**Definition 2.3.3.** A nonempty subset \( I \subset R \) is an **ideal** if

1. For every \( x, y \in I \), \( x + y \in I \).
2. If \( s \in I, r \in R \), then \( rs \in I \).

**Definition 2.3.4.** The **ideal generated by** \( x \), denoted \( (x) \) or \( xR \), is the set \( \{xr : r \in R\} \).

Observe that \( (x) \) is indeed a valid ideal of \( R \). \( (x) \) is nonempty, since it contains \( x \), and if \( a, b \in (x) \), then \( a = rx \) and \( b = sx \) for some \( r, s \in R \). Hence, \( a + b = (r+s)x \), so \( a+b \in (x) \). Moreover, if \( s \in (x) \) and \( r \) is an element of \( R \), then \( s = xy \) for some \( y \in R \), and thus \( rs = (ry)x \), so \( rs \in (x) \).
**Definition 2.3.5.** If \( I \) is an ideal of \( R \), the quotient ring \( R/I \) consists of equivalence classes under the relation \( x \sim y \) if \( x - y \in I \). Equivalently, it consists of additive cosets of \( I \).

For example, if \( (5) \subset \mathbb{Z} \) is the ideal generated by the number 5, which consists of all integer multiples of 5, the quotient ring \( \mathbb{Z}/(5) \) is the set \( \{0, 1, 2, 3, 4\} \). We will be studying a particular type of quotient ring, which emerges from the study of polynomials over the rationals.

**Definition 2.3.6** ([Art11]). A number field \( K = \mathbb{Q} (\zeta) \) is a field that is obtained by adjoining\(^6\) an element \( \zeta \in \mathbb{C} \) to the rationals. If \( f \) is the polynomial with rational coefficients such that \( \zeta \) is a root of \( f \), \( f \) is monic (its first coefficient is 1), and \( f \) is irreducible (meaning that it has no nonconstant factors), then we call \( f \) the minimal polynomial of \( \zeta \).

Note that there is an isomorphism between \( K \) and the quotient ring \( \mathbb{Q}[x]/(f) \), where \( (f) \) is the ideal generated by \( f \). If \( n \) is the degree of \( f \), we can consider \( K \) as a vector space over \( \mathbb{Q} \) with basis \( \{1, \zeta, \zeta^2, \ldots, \zeta^{n-1}\} \). For a concrete example, we can consider the number field \( K = \mathbb{Q}(i) \), where \( i \)'s minimal polynomial is \( f(x) = x^2 + 1 \). Elements of this field are of the form \( a + bi \), where \( a \) and \( b \) are rational. For our purposes, we will be focusing on special subsets of number fields:

**Definition 2.3.7.** An algebraic integer \( \alpha \in K \) is an element whose minimal polynomial has integer coefficients. The set of all algebraic integers forms a ring[LP10], which is known as the ring of integers in \( K \) and is denoted \( \mathcal{O}_K \).

We will be looking at two specific ideals of this ring: an integral ideal \( I \subseteq \mathcal{O}_K \) is an ideal in the ring of integers, and a fractional ideal \( I \subseteq K \) is an ideal in \( K \) such that \( dI \) is an ideal in \( \mathcal{O}_K \), where \( d \) is an element of \( \mathcal{O}_K \). A useful fact about fractional ideals in \( \mathcal{O}_K \) is that they are \( \mathbb{Z} \)-modules of rank \( n \), meaning that every \( I \subseteq \mathcal{O}_K \) is precisely the set of integer linear combinations of some basis \( \{b_1, \ldots, b_n\} \subseteq \mathcal{O}_K \). Moreover, we can perform arithmetic on ideals, with \( I + J \) defined as \( \{x + y : x \in I, y \in J\} \) and the product \( IJ \) defined as the set of all finite sums of the form \( x_1y_1 + \cdots + x_my_m \) with \( x_i \in I, y_i \in J \).

A particularly useful class of number fields is obtained from the cyclotomic polynomials:

**Definition 2.3.8.** The \( n \)'th cyclotomic polynomial, \( \Phi_n(x) \in \mathbb{Z}[x] \), is the polynomial whose roots are the primitive \( n \)'th roots of unity, i.e. \( \{e^{2\pi i k/n}\} \) for all \( k \leq n \) coprime to \( n \). This polynomial has degree \( \varphi(n) \), where \( \varphi \) is Euler's totient function, which counts the number of \( k \leq n \) that are coprime to \( n \).

For example, for a prime \( p \), the \( p \)'th cyclotomic polynomial is \( x^{p-1} + x^{p-2} + \cdots + x + 1 \). And, if \( n \) is a power of 2, then \( \Phi_n(x) = x^{n/2} + 1 \). Observe that \( \Phi_n \) is always irreducible and monic, and therefore it is the minimal polynomial of any root of unity \( \zeta_n \) (\( \zeta_n \) is the primitive \( n \)'th root of unity, so \( (\zeta_n^i)^n = 1 \) for all \( i \)). Therefore, the number field \( \mathbb{Q}(\zeta_n) \) is isomorphic to the quotient \( \mathbb{Q}[x]/(\Phi_n(x)) \). We call this field the \( n \)'th cyclotomic field. An easy example of a cyclotomic field is \( \mathbb{Q}(i) \cong \mathbb{Q}[x]/(x^2 + 1) \), since \( x^2 + 1 \) is the fourth cyclotomic polynomial. One useful fact about cyclotomic fields is that if \( K = \mathbb{Q}(\zeta_n) \), i.e. the \( n \)'th cyclotomic field \( \mathbb{Q}[x]/(\Phi_n(x)) \), then its ring of integers is \( \mathbb{Z}[x]/(\Phi_n(x)) \) (see Theorem 7.2.6 in [Ash]).

---

\(^6\)This means that \( K \) is the smallest subfield of \( \mathbb{C} \) containing both \( \mathbb{Q} \) and \( \zeta \). Alternatively, \( K \) is equal to the image of \( \mathbb{Q}[x] \) under the map \( x \mapsto \zeta \), i.e. all polynomials with rational coefficients evaluated at \( x = \zeta \).
2.3.2 Ideal Lattices

The reason why LWE can be adopted to the abstract ring setting is that ideals of the ring of integers in an algebraic number field are closely related to lattices. Consider the set \( H = \mathbb{R}^{s_1} \times \mathbb{C}^{2s_2} \subset \mathbb{C}^n \), where \( s_1 + 2s_2 = n \); this space is isomorphic to \( \mathbb{R}^n \) as an inner product space. We give \( H \) the norm induced on it from \( \mathbb{C}^n \), and for the \( \ell_2 \) norm, this coincides with the norm on \( \mathbb{R}^n \). For more details regarding this space, see [LPR10]. This space is incredibly useful, since it allows us to embed a number field in \( \mathbb{C}^n \), giving it geometric properties. An arbitrary algebraic number field \( K = \mathbb{Q}(\zeta) \) has \( n \) embeddings \( \sigma_i : K \to \mathbb{C} \), where each maps \( \zeta \) to a different root of its minimal polynomial \( f \). If \( \sigma \) corresponds to a real root of \( f \) it is called a real embedding, and similarly for complex roots. Then, if \( s_1 \) is the number of real embeddings and \( 2s_2 \) is the number of complex embeddings (consisting of \( s_1 \) pairs of conjugates), then the canonical embedding of \( K \) into \( H \) is defined as

\[
\sigma(x) = (\sigma_1(x), \ldots, \sigma_n(x))
\]

The embedding allows us to apply geometric notions from \( H \) to elements of our number field; for example, we define \( \|x\|_p = \|\sigma(x)\|_p \). In particular, we note that if \( I \) is an integral or fractional ideal of \( \mathcal{O}_K \), then the embedded \( \sigma(I) \) actually forms a lattice (recall from Definition 2.1.1). As we stated above, a fractional ideal \( I \) is a \( \mathbb{Z} \)-module with a basis \( \{b_1, \ldots, b_n\} \subset \mathcal{O}_K \), and thus \( \sigma(I) \) is a lattice with basis \( \{\sigma(b_1), \ldots, \sigma(b_n)\} \). For an example of an ideal lattice, consider the ring of integers of the fourth cyclotomic field, \( \mathcal{O}_K = \mathbb{Z}[x]/(x^2 + 1) \). This ring has two embeddings, one mapping \( x \mapsto i \) and one mapping \( x \mapsto -i \). Therefore, the canonical embedding maps \( \mathcal{O}_K \), and any ideal thereof, into \( \mathbb{C}^2 \), with basis elements \( \sigma(1) = (1, 1) \) and \( \sigma(x) = (i, -i) \). A more detailed example of the canonical embedding is used later in Section 3.4.1.

We can use the canonical embedding to rephrase the lattice problems discussed in section 2.1 in terms of ideal lattices. The slightly different setting does require some modifications — for example, \text{GapSVP}\_\gamma (2.1.6) can be efficiently solved in the ideal setting [Pei16], so it is no longer useful for reductions — but we can still use other lattice problems.

**Definition 2.3.9 ([LPR10])**. Let \( K \) be a number field, and let \( \gamma \geq 1 \). Given a fractional ideal \( I \) of \( \mathcal{O}_K \), \( K\text{-SVP}_\gamma \) problem is to find \( x \in I \) such that \( \|x\| \leq \gamma \lambda_1(I) \), where, as with lattices, \( \lambda_1(I) \) denotes the length of the shortest vector in the canonical embedding of \( I \). The \( K\text{-SIVP}_\gamma \) problem is to find \( n \) linearly independent elements of \( I \) whose norms are less than \( \lambda_n(I) \).

**Definition 2.3.10 ([LPR10])**. Let \( K \) be a number field, let \( I \) be a fractional ideal in \( \mathcal{O}_K \), and let \( d < \frac{1}{2} \lambda_1(I) \). Given \( y = x + e \) with \( x \in I \) and \( \|e\| \leq d \), the \( K\text{-BDD}_{I,d} \) problem, i.e. Bounded Distance Decoding, is to recover \( x \).

These problems are conjectured to be hard to solve in the ring setting as well, and therefore they provide a suitable basis on which to base cryptography.

2.3.3 The RLWE problem

Having gained the necessary ring theory background, we can define the Ring version of the Learning With Errors problem. Our parameters will be a number field \( K \), an integer
modulus \( q \geq 2 \), and an error distribution \( \chi \). Let \( R \) be the ring of integers \( \mathcal{O}_K \). In practice, we will usually take \( K \) to be a power-of-two cyclotomic field, so \( K = \mathbb{Q}[x]/(x^n + 1) \) and \( \mathcal{O}_K = \mathbb{Z}[x]/(x^n + 1) \) where \( n \) is a power of \( 2 \). We will mostly be working with \( R_q \), which is the analogue to \( \mathbb{Z}_q \), i.e. \( \mathbb{Z}_q[x]/(x^n + 1) \).

**Definition 2.3.11.** [Pei16] Let \( s \) be an element of \( R_q \). The **Ring-LWE** distribution \( A_{s,\chi} \) over \( R^2_q \) is \( A_{s,\chi} = (a, sa + e \mod q) \) for a randomly and uniformly chosen \( a \leftarrow R_q \) and for \( e \leftarrow \chi \).

To make RLWE a little less abstract, we can consider a concrete example. Consider the ring \( R = \mathbb{Z}[x]/(x^4 + 1) \), which is the ring of integers of the eight cyclotomic field, with \( q = 37 \). \( a \) could be \( 22x^2 + 16x + 2 \), \( s \) might be \( 5x^3 + 36x + 12 \), and our error distribution is always chosen to generate “small” values whose magnitude is significantly smaller than our modulus, say \( e = 2x \). We can compute \( as \) by multiplying according to the rules in this quotient ring:

\[
as = (22x^2 + 16x + 2)(5x^3 + 36x + 12) \\
= 110x^5 + 80x^4 + 10x^3 - 22x^3 - 16x^2 - 2x + 288x^2 + 192x + 24
\]  
(2.9)

We’ll use the facts that \( x^4 + 1 \) is the identity in this ring (since it is a member of the ideal \((x^4 + 1)\)), so \( x^4 \equiv -1 \) and \( x^5 \equiv -x \). Then, we simply combine terms and reduce modulo \( q \):

\[
as = -110x - 80 + 10x^3 - 22x^3 - 16x^2 - 2x + 288x^2 + 192x + 24 \\
= 25x^3 + 13x^2 + 6x + 16
\]  
(2.10)

And therefore our sample from \( A_{s,\chi} \) is

\[
(a, as + e) = (22x^2 + 16x + 2, 25x^3 + 13x^2 + 8x + 16)
\]

Looking at these polynomials, it seems like it would be difficult to recover \( s \) without knowing \( e \) — just as in the LWE case, adding the error means that we can no longer efficiently undo the multiplication. We can indeed formalize this intuition by defining search and decision problems based on this distribution, just as we did with normal LWE.

**Definition 2.3.12.** The **RLWE problem**, or RLWE\(_{q,\chi,m}\), is the problem of recovering \( s \) given \( m \) samples from \( A_{s,\chi} \).

**Definition 2.3.13.** Given \( m \) samples \( (a, b) \in R^2_q \) that either come from \( A_{s,\chi} \) for a randomly chosen \( s \) or the uniform distribution over \( R^2_q \), the **Decision-RLWE problem**, or DRLWE\(_{q,\chi,m}\), is the problem of determining which option is true with non-negligible advantage.

It should be noted that this problem is slightly different from what is given in the original Lyubashevsky-Peikert-Regev 2010 paper. In their scheme, the RLWE distribution is over \( R'_q \), which is the dual of \( R_q \), a notion closely related to dual lattices (recall Definition 2.1.4). However, the definition we are using here is easier to understand and does not require even more number theory background, and in any event they have been shown to be equivalent[Pei16]. It should also be noted the problems are sometimes given in “Normal
form,” where $s$ is sampled from $\chi$ instead of uniformly, but these are equivalent up to a small difference in $m$. The hardness of RLWE is based on the conjectured hardness of the ideal lattice problems discussed above. The reduction is quite similar to the LWE one, but the full version given in [LPR10] is extremely technical, and it relies on machinery well beyond the scope of this work, so our treatment will be slightly informal.

**Theorem 2.3.1** (Hardness of RLWE, Informal ([LPR10])). Let $K$ be the cyclotomic field of degree $n$. For any $m = \text{poly}(n)$, let $\chi$ be an error distribution with error rate $\alpha < 1$, and let $q$ be a prime that is 1 modulo $n$. Then, if there exists an algorithm to solve $\text{DRLWE}_{q,\chi,m}$ for approximation factor $\gamma(n) = \text{poly}(n)/\alpha$, there exists a quantum algorithm to solve $\text{SVP}_\gamma$.

Just as in Theorem 2.1.1, we establish the hardness using a reduction, i.e. by showing that an efficient solution to $\text{DRLWE}_{q,\chi,m}$ gives an efficient solution to $\text{SVP}_\gamma$. Taking the contrapositive, the conjectured lack of an efficient solution, or hardness, of $\text{SVP}_\gamma$ on ideal lattices implies the hardness of decision RLWE. There are two primary steps: a reduction from $\text{SIVP}$ to RLWE, and then a search-to-decision reduction, both of which are similar to their analogues for LWE. The reduction from Search RLWE to SVP uses the Discrete Gaussian Sampling problem as an intermediary, which is the problem of sampling from the discrete Gaussian distribution on a given ideal with a given standard deviation. The quantum reduction that [Reg05] uses for LWE works here too, and then we can take advantage of the fact that a sample from the discrete Gaussian is guaranteed to have short length to solve SIVP.

The next phase in the reduction is from RLWE to DRLWE. For LWE, we created a transformation that turned a pair $(A, As + e)$ into either the uniform distribution or the LWE distribution depending on the $i$’th entry in $s$, allowing us to efficiently recover each entry in $s$ using an algorithm for Decision-LWE. This trick doesn’t quite work in the ring setting, so we use the intermediate problem of recovering $s$ modulo a specific type of prime ideal given the Ring LWE distribution. We can then use the structure of the ring to perform this trick on $s$ modulo these ideals, giving a reduction to a modified version of the decision problem. Finally, a hybrid argument is used to reduce to DLWE.

### 2.3.4 Encryption from RLWE

In later chapters, we will see many encryption schemes whose security is based on the RLWE problem. Here, we present a simpler system designed to highlight the advantages that RLWE has over LWE.

**Definition 2.3.14.** Let $R = \mathbb{Z}[x]/(x^n + 1)$ for some power of two $n$, let $q$ be a prime modulus that is 1 modulo $n$, and let $\chi$ be our error distribution. The **RLWE-Encryption** Scheme is as follows:

- **Key Generation**: Sample $a \leftarrow R_q$ uniformly, as well as $s$ and $e$ independently from $\chi$. Output $s$ as the secret key and $(a, as + e)$ as the public key.

---

7The actual distribution isn’t so important — [LPR10] use a discrete Gaussian — but the error rate being less than $\alpha$ means that each entry in any $e \leftarrow \chi$ is less than $\alpha$
• **Encryption:** To encrypt a message \( m \in \{0, 1\}^n \), we view it as a polynomial over \( R_q \), with the coefficient of \( x^i \) being 1 if \( m_i = 1 \) and 0 otherwise. Then,

\[
E_{(a, b)}(m) = (ar + e, br + e' + \lceil \frac{q}{2} \rceil m)
\]

Where \( e, e', r \leftarrow \chi \) are randomly and independently sampled. Note that all addition is done modulo \( q \).

• **Decryption:** \( m^* = D_{s,(a,b)}(u, v) = v - us \). To convert \( m^* \) back to a string of bits, round every coefficient \( c_i \) to 0 if \( |c_i| < \frac{q}{4} \) (i.e. if \( c_i \) is closer to 0 than to \( q/2 \)), and 1 otherwise.

**Lemma 2.3.1.** This scheme is valid, meaning that decryption fails with negligible probability.

**Proof.** Observe that for any \( m \in \{0, 1\}^n \),

\[
D_{s,(a,b)}(E_{(a,b)}(m)) = D_{s,(a,b)}(ar + e, br + e' + \lceil \frac{q}{2} \rceil m)
\]

\[
= br + e' + \lfloor \frac{q}{2} \rfloor m - s(ar + e)
\]

\[
= (as + e)r + e' - sar - se + \lfloor \frac{q}{2} \rfloor m
\]

\[
= (er + e' - se) + \lfloor \frac{q}{2} \rfloor m
\]

(2.11)

The decoding then successfully recovers \( m \) if and only if \( |(er + e' - se)| < q/4 \). Everything in the parentheses is drawn from our henceforth unspecified error distribution, so if the distribution is such that \( (er + e' - se) \) is less than \( q/4 \) with overwhelmingly high probability, we have validity as desired. In practice, the Gaussian distribution similar to that used in LWE can be used in RLWE to achieve this property. \( \Box \)

This scheme is also CPA secure. We omit the full proof of CPA security, since it is essentially identical to the proof of security for the LWE Encryption Scheme. To summarize, we first observe that the “normal form” of the RLWE problem, which is when \( s \) is sampled from \( \chi \) instead of uniformly, is equivalent to the standard version, and thus the public key \( (a, as + e) \) is indistinguishable from the uniform distribution on \( R_q^2 \). Let \( A \) be an adversary that has access to a distribution \( D \) over \( R_q \times R_q \), and assume that \( Eve \) wins the CPA game for this scheme with messages \( 0^n \) and \( 1^n \) with advantage \( \epsilon \). Without loss of generality, let

\[
\Pr[Eve(E_{(a,b)}(0^n))] > \Pr[Eve(E_{(a,b)}(1^n))] + \epsilon
\]

As before, we consider the case where the key \( (a, b) \) is sampled uniformly instead of from the RLWE distribution. In this case, the encryption of 0 is \( E_{(a,b)}(0) = (u, v) = (ar + e, br + e') \), so the pairs \( (a, u) = (a, ar + e) \) and \( (b, v) = (b, br + e') \) are instances of the RLWE distribution as well and thus indistinguishable from uniform by the hardness of DRLWE. Thus, we can let our adversary \( A \) sample \( (a, b) \) from \( D \), encrypt \( 0^n \) and \( 1^n \) with it, feed it to \( Eve \), and output 1 iff \( Eve \) outputs 1, since when \( D \) is uniform the joint distribution \( (a, b, u, v) \) is indistinguishable from uniform, so \( Eve \) has negligible advantage, and when \( D \) is the RLWE distribution, \( Eve \) is getting real encryptions and thus has advantage \( \epsilon \). But this means that \( A \) distinguishes the uniform and RLWE encryptions with non-negligible advantage \( \epsilon \), breaking the hardness of DRLWE and contradicting Theorem 2.3.1. Taking the contrapositive, our scheme must be CPA secure.
2.3.5 Efficiency Advantages of RLWE

Why did we need to bring in all of this ring theory, when linear algebra and basic lattice theory can provide us with secure, quantum-resistant encryption schemes? It turns out that RLWE is significantly more efficient than normal LWE. We can look at the two encryption schemes described above as a case study. To encrypt an $n$-bit message with LWE, we use $m$ $n$-dimensional vectors in $\mathbb{Z}_q$ for each bit of the message, so the total key size is $O(mn^2 \log(q))$. For RLWE, though, we can use an $n$-dimensional vector for every bit of the message, giving us a key size of $O(n \log(q))$. Moreover, the actual encryption and decryption procedures are more efficient in RLWE due to better algorithms for polynomial multiplication in rings. Key generation and encryption in our RLWE scheme require 1 and 2 polynomial multiplications respectively, each of which can be done in $O(n \log n)$ time using the Fast Fourier Transform algorithm [LPR13]. Moreover, the big-O is not hiding any large constant factors here, and there indeed exist efficient implementations of polynomial multiplication in cyclotomic rings. Another algorithm called the Number Theoretic Transform allows for even more optimized polynomial multiplication (see [RVM+13] for details). On the other hand, LWE requires taking a dot product of $n$-dimensional vectors and multiplying an $m \times n$ matrix by a vector for every bit, so encryption runs in time $O(mn^2)$. There exist “compact” varieties of LWE analogous to our RLWE scheme that allow us to drop the $m$ factor, but even still RLWE gives a substantial speedup. As we move on to fully homomorphic encryption, we will see that the biggest barrier to its widespread use is its inefficiency, highlighting how important the speedups of RLWE are.
Chapter 3

Fully Homomorphic Encryption

In a presentation to my fellow Ph.D. admits four years ago, Dan highlighted fully homomorphic encryption as an interesting open problem and guaranteed an immediate diploma to anyone who solved it. Perhaps I took him too literally.

Craig Gentry

In this chapter, we will explore Fully Homomorphic Encryption. We will first survey the brief history of FHE, summarizing the different schemes that have been created since FHE was first introduced by Gentry in [Gen09]. We will focus on the bootstrapping theorem, Gentry’s breakthrough that allowed him to construct FHE. Next, we will explore the GSW scheme from [GSW13], whose relative simplicity makes it an instructive example for how FHE works. Finally, we will review the approximate FHE scheme from [CKKS17], a much more efficient scheme which is frequently used in fully homomorphic machine learning.

3.1 Basic Definitions

Before we define fully homomorphic encryption, we will look at the more general notion of partially homomorphic encryption. Throughout this section, we will assume that all encryption schemes are CPA-secure.

First, we informally review the basic definitions of Boolean circuits, which will be necessary for our discussion of homomorphic encryption. A boolean circuit is a series of boolean logic gates such as AND, OR, and NOT, that are connected to one another and are used to compute a function based on inputs. We say that a circuit $C$ computes the function $f$ if $C(x) = f(x)$ for all inputs $x$. For example, the function $ALLONE : \{0,1\}^3 \rightarrow \{0,1\}$, which outputs whether or not all of its inputs are 1, can be computed by the circuit $AND(AND(x_0, x_1), AND(x_2, x_3))$, which consists of three AND gates. It turns out that all boolean functions can be computed by circuits consisting only of NAND gates (i.e. the composition of NOT and AND), a fact which will be quite useful later on — if we can con-
struct a way to homomorphically evaluate the NAND of two ciphertexts, and we show that we can perform an arbitrary amount of NANDS, then we have obtained fully homomorphic encryption. We can also use addition and subtraction to when our inputs are integers, since $NAND(b_1, b_2) = 1 - b_1 \cdot b_2$, so in some cases we will use these as our basis for so-called arithmetic circuits. Finally, the depth of a boolean circuit is the length of the longest path (measured by the number of gates) from any input bit to the output. More details about boolean circuits can be found in [Bar22].

The notion of a homomorphism originates in abstract algebra, where it refers to a function that preserves the structure of a certain operation. For example, a function $f$ is a homomorphism with respect to addition if $f(x + y) = f(x) + f(y)$ for all $x, y$. A homomorphic encryption scheme has the same idea, with $f$ being our encryption scheme $E$, so an encryption scheme that is homomorphic with respect to addition would satisfy the property that $E(x + y) = E(x) + E(y)$. It turns out that this requirement is slightly too strong, and it is much more useful to instead say that there exists a different function $ENCPLUS$ such that $ENCPLUS(E(x), E(y))$ is a valid encryption of $x + y$; this definition preserves the core idea that one can operate on encrypted data without decrypting it.

Definition 3.1.1. [Bar21] Let $\mathcal{F} = \cup_{\ell \in \mathbb{N}} \mathcal{F}_\ell$ be a family of functions such that each $f \in \mathcal{F}$ maps $\{0, 1\}^\ell \rightarrow \{0, 1\}$, i.e. $\mathcal{F}$ is a family of boolean functions that each have a fixed input length. An encryption scheme $(G, E, D)$ is $\mathcal{F}$-partially homomorphic if there exists a polynomial time algorithm $EVAL$ such that for all $(e, d) \leftarrow G(1^n)$, $\ell = poly(n)$, $x_1, \ldots, x_\ell \in \{0, 1\}$, and $f \in \mathcal{F}_\ell$ which can be described in $|f| < poly(\ell)$ bits, the following two conditions hold:

1. **Compactness:** $c = EVAL_e(f, E_e(x_1), \ldots, E_e(x_\ell))$ has length at most $n$.

2. **Correctness:** $c$ decrypts to $f(x_1, \ldots, x_\ell)$, i.e. $D_d(c) = f(x_1, \ldots, x_\ell)$

In English, a scheme is partially homomorphic with respect to a family of functions if for all $f$ and for all inputs $x$, we can use $EVAL$ to generate an encryption of $f(x)$ given only an encryption of $x$ and the public key. For fully homomorphic encryption, we expand this requirement to include *every* function

Definition 3.1.2. An encryption scheme $(G, E, D)$ is fully homomorphic if it is $\mathcal{F}$-partially-homomorphic, where $\mathcal{F}$ is the set of all functions

There is an intermediary definition which turns out to be extremely useful. “Leveled” homomorphic encryption uses a collections of encryption schemes, each of which is partially-homomorphic for the family of functions that can be computed by boolean circuits up to a specific depth. Thus, each individual encryption scheme is limited in its power — and therefore easier to construct and more efficient to use — whereas the entire collection still gives us the power to homomorphically evaluate any function that we realistically want to compute as long as we know the depth in advance.

Definition 3.1.3. [Gen09] A family of encryption schemes $\{(G^d, E^d, D)\}_{d \in \mathbb{N}}$ is levelled fully homomorphic if for every $d \in \mathbb{N}$, $(G^d, E^d, D)$ is partially homomorphic with respect to the family of all functions computable by a circuit of depth at most $d$. In other words, there
exists an algorithm $EVAL^d$ whose runtime is polynomial in $d$, $n$, and the size of the circuit, that for all $f$ computable by $C$ of depth $\leq d$ and for all $E_e(x)$ can output a ciphertext $c$ with length at most $n$ that decrypts to $f(x)$.

As we’ll see, FHE relies on something called bootstrapping, which is extremely inefficient, whereas leveled FHE schemes avoid this step. Thus, when one knows the maximum depth of their evaluation circuits, it is much more efficient to use a leveled FHE scheme.

### 3.2 A Brief History of FHE

Fully homomorphic encryption was first proposed in 1978 by Rivest, Adleman, and Dertouzos [RAD78]. They recognized the utility of such a “privacy homomorphism,” but for many decades it remained a fantasy, until Craig Gentry published his groundbreaking scheme in (2009 [Gen09]). While we won’t look too deeply into Gentry’s construction, which is quite complicated, we will describe bootstrapping, his key insight which made FHE possible. Bootstrapping provides a way for a partially homomorphic scheme that satisfies specific requirements to be converted into a fully homomorphic (or a levelled FHE) scheme, and it turns out that those requirements can be easily achieved by LWE and RLWE-based encryption schemes.

To understand bootstrapping, it is important to note that the limiting factor for FHE is noise. All LWE-based encryption schemes use some amount of randomness to hide the message, but performing operations on these ciphertexts will simply amplify the noise until the ciphertext can no longer be decrypted. However, if we had some way to “refresh” the noise in a ciphertext and then perform a single homomorphic operation, this problem could be avoided. We could refresh, and then evaluate a NAND gate. Then, we can refresh again, and do another NAND. And another. In fact, we can keep repeating this procedure to do an arbitrary amount of NANDs to our ciphertexts. As discussed above, any boolean function can be computed with a circuit composed of NANDs, so achieving this capability gives us a fully homomorphic encryption scheme.

Such a refresh procedure seems like it’s too good to be true, but it turns out that it’s feasible. To get there, we must ask ourselves a strange question: what happens if a partially homomorphic encryption scheme $(G, E, D, EVAL)$ can homomorphically evaluate its own decryption function, $D$. This question seems strange because there are quite a number of functions which would be useful to homomorphically evaluate, but the decryption function does not seem particularly useful for anything other than turning a double encryption $E_e(E_e(m))$ into the single encryption $E_e(m)$, a task for which it is hard to envision any uses. However, Gentry realized that if you doubly-encrypt a ciphertext, homomorphically decrypting it will perform this exact refreshing procedure. Suppose that $(e, d) \leftarrow G(1^n)$, and that $c = E_e(m)$ for some message $m$. Then, let $\overline{d}$ be the encryption of the secret key, i.e. $E_e(d)$, and let $\overline{c}$ be the encryption of $c$ — a double encryption of $m$ — or $E_e(c)$. If $EVAL$ can homomorphically evaluate $D$, then $c^* = EVAL(D, \overline{d}, \overline{c})$ will be by definition be an encryption of $m$, since $D_d(c^*) = D_d(D_d(\overline{c})) = m$. Now, suppose that $D$ can be computed by a circuit of depth $d$, and $EVAL$ can homomorphically evaluate circuits of depth $d + 1$. Then, the circuit $C(d, x, y) = NAND(D_d(x), D_d(y))$ has depth $d + 1$, so if $c_1, c_2$ are the encryptions of two
bits $m_1, m_2 \in \{0, 1\}$, then we can doubly encrypt these into $\overline{c_1}$ and $\overline{c_2}$ and homomorphically compute
\[ c' = EV AL(C, \overline{d}, c_1, c_2) \]
Observe that $c'$ is an encryption of $NAND(m_1, m_2)$ by definition of $EVAL$, since
\[ D_d(c') = C(c_1, c_2) = NAND(D_d(c_1), D_d(c_2)) = NAND(m_1, m_2) \]
In other words, homomorphically evaluating $C$ is precisely the refresh-then-NAND procedure that we wished for earlier, since the new ciphertext $c'$ has a “normal” amount of noise such that the procedure can be repeated again.

This idea can be formalized as follows:

**Definition 3.2.1.** A $F$-homomorphic encryption scheme $(G, E, D, EV AL)$ is **bootstrap-pable** if all circuits of depth $d + 1$ are in $F$, where $d$ is the depth of the circuit computing $D$.

**Theorem 3.2.1 (Bootstrapping Theorem).** If $(G, E, D, EV AL)$ is bootstrappable, then it can be converted into a levelled FHE scheme. If it is CPA circular secure — meaning that it retains CPA security even when an adversary knows $E_e(d)$ — then it can be converted into a FHE scheme.

The second implication is more relevant for our purposes; circular security is difficult to prove, but is believed ([Bar21, Gen14]) to hold for popular schemes.

Gentry’s initial construction was extremely inefficient, but subsequent work has made significant improvements. The second generation of FHE schemes includes the BGV\(^1\) scheme of Brakerski, Gentry, and Vaikuntanathan ([BGV12]), which uses a “modulus switching” technique to keep noise down, as well as the BFV scheme of Brakerski, Fan, and Vercauteren ([FV12]). Both of these schemes use RLWE. Another notable second-generation scheme is the LWE-based GSW scheme from Gentry, Sahai, and Waters ([GSW13]), which we’ll discuss in depth shortly. After these schemes, the state of the art for FHE schemes split into two branches. One branch, which focuses on optimizing leveled FHE, contains the CKKS scheme, proposed by Cheon, Kim, Kim and Song in [CKKS17], which we will discuss more in depth later on. CKKS is “approximate,” meaning that it trades off a tiny bit of accuracy for massive efficiency gains. It is primarily leveled FHE, but a bootstrapped FHE version was later proposed in [CHK^+18]. The second branch focuses on speeding up bootstrapping, and it is based off of Ducas and Micciancio’s FHEW, the Fastest Homomorphic Encryption in the West. They propose a new algorithm for $EVALNAND$ that generates a lot less noise and thus requires bootstrapping less frequently, and they use a much faster bootstrapping method. Together, these optimizations sped up bootstrapping — which was always the slowest part of existing schemes — by an order of magnitude. The Toroidal FHE scheme, or TFHE [CGGI18], by Chillotti, Gama, Georgieva, and Izabachène, further optimizes the FHEW, bringing bootstrapping times down into the milliseconds (compared to half a second for FHEW and hours for Gentry’s original construction). TFHE and CKKS are the current state of the art for FHE and leveled FHE respectively.

\(^1\)As will become apparent, FHE schemes are always known by acronyms, either for the names of the authors or the title of the paper.
3.3 FHE Example: GSW

In order to better understand fully homomorphic encryption, we will do a deeper dive into the scheme invented by Gentry, Sahai, and Waters in [GSW13] (henceforth the “GSW scheme,” or just GSW). The relatively simple nature of GSW makes it much easier to understand than other FHE schemes. Our exposition will largely follow [GSW13] and [Gen14], and we add a full proof that it can be bootstrapped (similar to [Bar21]), modifying the scheme to allow for an easily-understandable bootstrapping proof. We also add examples to make the scheme more accessible.

A common thought experiment (see [Gen14, Bar21]) that helps understand the GSW scheme is to consider how we would construct FHE in an ideal world without Gaussian elimination (analogous to our thought experiment with LWE in Chapter 2). In this world, inverting a matrix is computationally intractable, so we can encrypt a message $m$ with a matrix $C$ such that $Cs = ms$, where $s$ is the secret key. Decrypting a message is straightforward when $s$ is known. This encryption scheme lends itself well to FHE, since matrix arithmetic translates itself directly to homomorphic arithmetic. For any messages $m_1, m_2$, with ciphertexts $C_1, C_2$, $(C_1 + C_2)s = (m_1 + m_2)s$, and $C_1C_2s = C_1(m_2s) = m_1m_2s$.

Unfortunately, Gaussian elimination does exist, and this scheme is not secure. But, just as we saw before, adding a little bit of noise will thwart Gaussian elimination, making this scheme secure. Unlike the standard LWE setting, however, we can’t just add noise willy-nilly, since too much noise will cause the homomorphic operations to fail. The insight of GSW is to carefully tune the amount of noise to preserve both security and our homomorphic operations.

3.3.1 GSW Bit Operations

In order to keep noise down, GSW uses special transformations to turn vectors in $\mathbb{Z}_q^k$ into vectors with 0/1 entries. Let $u, v$ be vectors in $\mathbb{Z}_q^k$, $\ell = \lceil \log_2(q) \rceil + 1$, and let $N = \ell k$.

- $\text{BitDecomp}(u)$ takes in a vector $u \in \mathbb{Z}_q^k$ and outputs the $N$-dimensional vector
  \[(a_{1,0}, \ldots, a_{1,\ell-1}, \ldots, a_{k,0}, \ldots, a_{k,\ell-1}) \in \{0, 1\}^N\]
  Which is the concatenation of the $k$ bit vectors (each of length $\ell$) that are the binary representations of the elements in $u$, ordered least-to-most significant.\(^2\)

- $\text{BitDecomp}^{-1}$ is the inverse of this operation — it takes an $N$-dimensional vector, splits it into $k$ blocks of length $\ell$, and converts it into the corresponding number in $\mathbb{Z}_q$. To do so, for each block $a_{i,0}, \ldots, a_{i,\ell-1}$ we compute $\sum 2^j a_{i,j}$ modulo $q$ and output the concatenation of all $k$ of these, which is in $\mathbb{Z}_q^k$. This operation is well defined for any vector in $\mathbb{Z}_q^N$ even if its entries are not binary.

- $\text{Flatten}(u) = \text{BitDecomp}(\text{BitDecomp}^{-1}(u))$. Flatten takes in $N$-dimensional vectors over $\mathbb{Z}_q$ and outputs $N$-dimensional vectors of bits.

---

\(^2\)The least significant bit of a binary representation is the lowest bit, and the most significant bit is the highest. For example, $6 = 0 \cdot 2^0 + 1 \cdot 2^1 + 1 \cdot 2^2$, so the binary representation of 6 is 110 in most-to-least order and 011 in least-to-most order.
• \textbf{Powersof2} multiplies each coordinate of \( u \in \mathbb{Z}_q^n \) by all powers of 2 up to \( \ell - 1 \), i.e. \( \text{Powersof2}(u) = (u_1, 2u_1, \ldots, 2^{\ell-1}u_1, \ldots, u_k, \ldots, 2^{\ell-1}u_k) \in \mathbb{Z}_q^N \).

Note that \( \text{BitDecomp}(\text{BitDecomp}^{-1}(u)) \) is not simply the identity function because \( \text{BitDecomp}^{-1} \) is only the inverse when restricted to inputs in \( \{0,1\}^N \). We can see why this is the case through a counterexample. Let \( q = 3 \), and \( k = 3 \), so \( \ell = \lceil \log_2(q) \rceil + 1 = 2 \) and \( N = 6 \). Then, if \( u = (0 2 1 1 2 0) \) is a vector in \( \mathbb{Z}_q^N \), we have

\[
\text{BitDecomp}^{-1}(u) = ((2^0 \cdot 0 + 2^1 \cdot 2) \ (2^0 \cdot 1 + 2^1 \cdot 1) \ (2^0 \cdot 2 + 2^1 \cdot 0)) = (1 0 2)
\]

Note that all of our arithmetic is modulo \( q = 3 \). We can then apply \( \text{BitDecomp} \) to the result

\[
\text{BitDecomp}(\text{BitDecomp}^{-1}(u)) = \text{BitDecomp}((2 0 2)) = (100001)
\]

Since 10 and 01 are the binary representation of 1 and 2 respectively when ordered least-to-most significant. Observe that if \( u \) only had coordinates in 0/1, then \( \text{Flatten} \) would simply be the identity function.

We can note a few interesting properties of these functions. First, for any pair of vectors \( u, v \), their inner product \( \langle u, v \rangle = \langle \text{BitDecomp}(u), \text{Powersof2}(v) \rangle \). And, for any \( N \)-dimensional vector \( a \) and \( k \)-dimensional \( b \),

\[
\langle a, \text{Powersof2}(b) \rangle = \langle \text{Flatten}(a), \text{Powersof2}(b) \rangle = \langle \text{BitDecomp}^{-1}(a), b \rangle
\]

These identities follow directly from the definitions, since the various powers of two all cancel out. We can also extend these functions to matrices, defining \( \text{Flatten}(A) \) to be the matrix obtained by flattening every row of \( A \), and so on and so forth.

\section{The GSW Scheme}

We can now present the GSW scheme. In the original paper, the scheme uses ciphertexts in \( \mathbb{Z}_q \), but we will modify this slightly (in a similar fashion to [Gen14]) to accept ciphertexts in \( \{0,1\} \) for simplicity and for consistency with the other schemes discussed:

\textbf{Definition 3.3.1} (GSWENC, [GSW13]). The parameters are a modulus \( q \), dimension \( n \), error distribution \( \chi \), and \( m = O(n \log q) \), all chosen so that LWE holds. Let \( \ell = \lceil \log q \rceil + 1 \) and \( N = (n + 1)\ell \).

• \textbf{Key Generation}: Sample a random \( t \leftarrow \mathbb{Z}_q^n \), and let \( d = (1, -t_1, \ldots, -t_n) \in \mathbb{Z}_q^{n+1} \). Then, let \( B \leftarrow \mathbb{Z}_q^{m \times n} \) be sampled uniformly, \( e \) be the \( m \)-dimensional vector where each component is sampled from \( \chi \), and let \( b = B \cdot t + e \). Let \( A = b \parallel B \), i.e. the matrix in \( \mathbb{Z}_q^{m \times (n+1)} \) whose first column is \( b \) and 2nd through \( n+1 \)'th columns are \( B \). Output \( (e = A, d) \).

• \textbf{Encryption}: To encrypt a bit \( b \), sample a random \( R \leftarrow \{0,1\}^{N \times m} \). Then,

\[
E_A(b) = \text{Flatten} \left( b \cdot I_N + \text{BitDecomp}(RA) \right)
\]

Where \( I_N \) is the \( N \times N \) identity matrix.
**Decryption:** Let $v = \text{Powersof2}(d)$, and since $d$'s first coefficient is 1, the first $\ell$ entries in $v$ are $1, 2, \ldots, 2^{\ell-1}$. Let $i$ be the numbers such that $q/4 < v_i = 2^i \leq q/2$. To decrypt a ciphertext $C$, let $C_i$ be its $i$'th row (indexing from 0), and compute $x = \langle C_i, v \rangle$. Output 0 if $|x| < v_i/4$ and 1 otherwise.\(^3\)

**NAND:**

$$\text{EVALNAND}(C_1, C_2) = \text{Flatten}(I_N - C_1C_2)$$

Each step of this encryption process accomplished one of two distinct tasks. The first task is to “hide” our bit $b$ behind enough randomness such that the value of $b$ cannot be determined, which is essentially the definition of a secure encryption scheme. We do so by selecting a random LWE instance $A$, taking random sums of its rows via multiplication by $R$, and then adding $b$ to this, a process which is largely identical to how we encrypted a bit in the LWE Encryption scheme (see Definition 2.2.1) from the previous chapter. The second task is to ensure that the ciphertext isn’t too noisy so that we can perform homomorphic operations. Applying the BitDecomp function to $RA$ gave us a matrix with entries in $\{0, 1\}$, and then Flattening at the end ensures that we’re left with a 0/1 matrix after adding $b \cdot I_N$. Having small entries means that errors won’t be magnified too much when we add and multiply matrices.

An important property of this scheme that we will formalize later in our analysis is that for any ciphertext $C$ that encrypts $b$, $Cv = bv + \text{small}$, where small denotes a small error vector. In linear-algebraic terms, $v$ is an “approximate eigenvector,” and our message $b$ is the “approximate eigenvalue;” GSW refer to this type of encryption schemes as the “approximate eigenvalue method.” Observe that this corresponds to our thought experiment from before, where we wished to encrypt $b$ as a matrix $C$ such that $Cv = bv$, but with added noise for security. This property is incredibly useful, since if $b_1$ and $b_2$ are the approximate eigenvalues for $C_1$ and $C_2$ respectively, then $(b_1 + b_2)$ is an approximate eigenvalue for $(C_1 + C_2)$ and $b_1b_2$ is an approximate eigenvalue for $C_1C_2$. This fact allows us to construct homomorphic addition and multiplication respectively.

We can look at an example to better understand how the GSW Scheme works. We’ll choose the insecure parameters of $q = 31$, $n = 1$, and $m = 3$ for simplicity’s sake, so $\ell = 5$ and $N = 10$. For key generation, we first sample a random $t = (13) \in \mathbb{Z}_{31}$ so $d = (118)$, since $-13 \equiv 18 \pmod{31}$. Then, we sample

$$B = \begin{bmatrix} 22 \\ 8 \\ 11 \end{bmatrix} \leftarrow \mathbb{Z}_q^{m \times n} = \mathbb{Z}_{31}^{3 \times 1} \quad e = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \in \mathbb{Z}_{31}^3$$

The error vector needs to have a low magnitude relative to $q$, and given the low values of the parameters used in this example, we need to use such a sparse vector — in a real-world application with secure parameters (i.e. a much higher $q$), $e$ would not be mostly zeroes. In any event, we then compute

$$A = (B \cdot t + e)\|B = \begin{bmatrix} 22 \\ 8 \\ 11 \end{bmatrix} (13) + \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \| \begin{bmatrix} 22 \\ 8 \\ 11 \end{bmatrix} = \begin{bmatrix} 722 \\ 128 \\ 1911 \end{bmatrix}$$

\(^3\)Note that in $\mathbb{Z}_q$, $|x|$ is $\min(x, q-x)$
And then the public key is $A$ and the secret key is $d$. To encrypt, we sample a random $R$ from $\{0, 1\}^{10 \times 3}$, and then compute $RA$:

$$RA = \begin{bmatrix}
0 & 1 & 0 \\
1 & 0 & 1 \\
1 & 1 & 0 \\
0 & 1 & 0 \\
1 & 0 & 1 \\
1 & 1 & 1 \\
0 & 0 & 1 \\
0 & 1 & 0 \\
0 & 0 & 0 \\
1 & 0 & 0
\end{bmatrix} \begin{bmatrix}
7 & 22 \\
12 & 8 \\
19 & 11
\end{bmatrix} = \begin{bmatrix}
12 & 8 \\
26 & 2 \\
19 & 30 \\
12 & 8 \\
26 & 2 \\
7 & 10 \\
19 & 11 \\
12 & 8 \\
0 & 0 \\
7 & 22
\end{bmatrix} \in \mathbb{Z}_{31}^{10 \times 10}$$

We then BitDecomp this matrix, which, as discussed above, means applying BitDecomp to each row:

$$\text{BitDecomp}(RA) = \text{BitDecomp} \begin{bmatrix}
12 & 8 \\
26 & 2 \\
19 & 30 \\
12 & 8 \\
26 & 2 \\
7 & 10 \\
19 & 11 \\
12 & 8 \\
0 & 0 \\
7 & 22
\end{bmatrix} = \begin{bmatrix}
00110 & 00010 \\
01011 & 01000 \\
11001 & 01111 \\
00110 & 00010 \\
01011 & 01000 \\
11100 & 01010 \\
11001 & 11010 \\
00110 & 00010 \\
00000 & 00000 \\
11000 & 01101
\end{bmatrix} \in \mathbb{Z}_{31}^{10 \times 10}$$

Finally, we add $b \cdot I_{10}$ and then flatten. If we’re encrypting $b = 1$, we obtain

$$\text{Flatten} \begin{bmatrix}
0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1
\end{bmatrix} + \begin{bmatrix}
1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 2 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \\
0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 & 2 & 0 & 1 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\
1 & 1 & 0 & 0 & 1 & 1 & 2 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 2
\end{bmatrix} \in \mathbb{Z}_{31}^{10 \times 10}$$
Recalling the definition of Flatten, we apply $\text{BitDecomp}^{-1}$ and then $\text{BitDecomp}$ to obtain 

$$C = \text{BitDecomp} \left( \begin{bmatrix} 13 & 8 \\ 28 & 2 \\ 23 & 30 \\ 20 & 8 \\ 11 & 2 \\ 7 & 11 \\ 19 & 13 \\ 12 & 12 \\ 0 & 8 \\ 7 & 7 \end{bmatrix} \right) = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \end{bmatrix} \in \{0, 1\}^{10 \times 10}$$

The numbers that changed by $\text{BitDecomp}^{-1}$ are bolded. Notice that flattening does not change the rows whose entries are 0/1.

We can use this example ciphertext to see how decryption works. To decrypt, we let 

$$v = \text{Powersof2}(d) = \text{Powersof2}(118) = (1 2 4 8 16 18 5 10 20 9) \in \mathbb{Z}_{31}^{10}$$

Since $31/4 < 8 = 2^3 < 31/2$, we'll use $i = 3$. We then compute the inner product of $v$ with the fourth row of $C$ (since we're indexing from 0):

$$\langle C_3, v \rangle = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 2 \\ 4 \\ 8 \\ 16 \\ 18 \\ 5 \\ 10 \\ 20 \\ 9 \end{bmatrix} = 9 \mod 31$$

Thus $|\langle C_3, v \rangle| = 9$, which is greater than $v_i/4 = 2$, so we recover $b = 1$ as desired. We can also verify the approximate eigenvector property:

$$Cv = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 2 \\ 2 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = bv + \begin{bmatrix} 1 \\ 2 \\ 9 \\ 9 \\ 9 \end{bmatrix}$$
Observe that $Cv$ is in fact approximately equal to $bv$, with the difference being a vector of quite small magnitude.

Having seen how the GSW scheme works, we will now analyze it more formally. We’ll prove the usual correctness and security, as well as the correctness of $\text{EVALNAND}$; as discussed above, $\text{EVALNAND}$ will allow us to construct a homomorphic circuit for any function.

**Theorem 3.3.1.** The GSW Scheme is correct, meaning that decryption succeeds with probability $1 - \text{negl}(n)$.

**Proof.** By definition of $\text{Powersof2}$ and $d$ we know that

$$v = \text{Powersof2}(d) = \text{Powersof2}((1, -t_1, \ldots, -t_n)) = (1, \ldots, 2^\ell - t_1, \ldots, 2^\ell - t_n)$$

And hence for any ciphertext $C = E_A(b)$,

$$Cv = \left(\text{Flatten}\left(b \cdot I_N + \text{BitDecomp}(RA)\right)\right)\text{Powersof2}(v) = bv + RAe$$

By the identities discussed above. But since $d = (1, -t_1, \ldots, -t_n) \in \mathbb{Z}_q^{n+1}$ and $A = (Bt + e)\|B$, we have $Ad = (Bt + e) - Bt = e$, so $Cv = bv + Re$. Note that $R$ has values in $\{0, 1\}$, and the entries of $e$ have small magnitude, so $Re$ is our small vector mentioned above. Then, $x = \langle C_i, v \rangle$ is equal to $bv_i + \langle R_i, e \rangle$. Since $R_i$ takes values in $\{0, 1\}$, the magnitude of $\langle R_i, e \rangle$ is at most $\|e\|_1$; since the entries in $e$ are being chosen from a $\chi$ that satisfies the requirements of LWE, we can choose our parameters such that $\|e\|_1 \geq q/16$ with negligible probability. Equivalently, $\langle R_i, e \rangle$ is less than $q/16$ with similarly high probability.

In this case, the triangle inequality tells us that

$$bv_i - \frac{q}{16} < |x| = |bv_i + \langle R_i, e \rangle| < bv_i + \frac{q}{16}$$

If $b = 0$, then $|x| < q/16 < v_i/4$, since $v_i > q/4$, and we output 0, and if $b = 1$, then $|x| > v_i - q/16 > 3v_i/4$ and we output 1. Note that in order for $|x|$ to not be less than $v_i/4$ or greater than $3v_i/4$, $\|e\|_1$ would need to be greater than $q/16$, but this happens with negligible probability, so there is no issue with our decryption function simply outputting 1 in this case. Therefore, decryption succeeds in all cases with extremely high probability. $\square$

A key fact emerges from this proof, which will be essential in our analysis of the homomorphic operations:

**Corollary 3.3.1.** If a ciphertext $C$ is such that $Cv = bv + e$, then decryption succeeds as long as $\|e\|_1 < q/16$.

Armed with this corollary in our arsenal, we can prove that $\text{EVALNAND}$ in fact works.

**Theorem 3.3.2.** If $C_1$ is an encryption of $b_1$ and $C_2$ is an encryption of $b_2$ then $\text{EVALNAND}(C_1, C_2)$ decrypts to $\text{NAND}(b_1, b_2)$.
Proof. Let $C_{\text{NAND}} = \text{EVALNAND}(C_1, C_2)$. Recall that to decrypt $C_3$, we let $c$ be its first row, and we compute $\langle c, v \rangle$. Since $C_{\text{NAND}}$ is obtained by Flattening $I_N - C_1 C_2$, its product with $v = \text{Powersof2}(d)$ is equal to $(I_N - C_1 C_2)v = v - C_1 C_2 v$. Recalling the approximate eigenvector property of the encryption scheme, we note that $C_2 v = b_2 v + e'_2$, where $e'_2$ is equal to $R' e_2$ for $R' \in \{0, 1\}^{N \times m}$ and $e_2$ is our original error vector. We can then apply this property again along with linearity to see that

$$C_1(C_2 v) = C_1(b_2 v + C_1 e'_2) = b_1 b_2 v + b_2 e'_1 + C_1 e'_2$$

And hence

$$(C_{\text{NAND}}) v = (1 - b_1 b_2) v + b_2 e'_1 + C_1 e'_2.$$

But $b_2$ is a bit, and $C_1$ is obtained by Flattening, so all of its entries are bits as well. Hence, as long as we choose our parameters (particularly $\chi$) such that $(b_2 e'_1 + C_1 e'_2) < q/16$ is small with overwhelming probability, then an identical argument to our correctness proof shows that this decrypts to $(1 - b_1 b_2)$, which is precisely $\text{NAND}(b_1, b_2)$ as desired.

While we will mostly be focused on homomorphic NAND operation, the GSW scheme supports a few others, and one of the nice things about the scheme is how natural these are. When we use the above scheme to encrypt integers in $\mathbb{Z}_q$ instead of bits, we homomorphically add $C_1$ and $C_2$ by simply computing $\text{Flatten}(C_1 + C_2)$, and we multiply by computing $\text{Flatten}(C_1 C_2)$.

Finally, in our excitement to prove facts about homomorphisms, we cannot forget about the whole point of cryptography: security.

**Theorem 3.3.3.** The GSW Scheme is CPA secure.

**Proof.** Let $C_b = E_A(b) = \text{Flatten}(b \cdot I_N + \text{BitDecomp}(RA))$. Observe that each row of $RA$ is precisely an encryption of 0 under the LWE encryption scheme from Definition 2.2.1, since $R$ is a random matrix over $\{0, 1\}$ and $A = (Bt + e) \parallel B$ for a random matrix $B \leftarrow \mathbb{Z}_q^{m \times n}$. By the security of this scheme (Theorem 2.2.2), RA is indistinguishable from the uniform distribution, and hence

$$\text{BitDecomp}^{-1}(C_b) = b \text{BitDecomp}^{-1}(I_N) + RA$$

Is indistinguishable from uniform as well. Then, if $U$ is a uniform random variable, applying $\text{BitDecomp}$ to both sides tells us that $\text{Flatten}(C_b)$ is indistinguishable from $\text{BitDecomp}(U)$. But $\text{Flatten}(C_b)$ is just $C_b$, so $C_b$ is indistinguishable from $\text{BitDecomp}(U)$. This fact doesn’t depend on $b$, so $C_0$ and $C_1$ are both indistinguishable from $\text{BitDecomp}(U)$, and we conclude by the transitivity of indistinguishability (Lemma 1.2.1) that encryptions of 0 and 1 are indistinguishable, giving us CPA security.

### 3.3.3 Circuits and Bootstrapping

We’ve established that we can successfully evaluate homomorphic NANDs, but what happens when we combine them into circuits? Suppose that our ciphertexts have $B$–bounded
error, meaning that each entry in the error vector $e$ is greater than $B$ with negligible probability, so we can assume that $\|e\|_\infty \leq B$. Then, we saw in our proof of Theorem 3.3.2 that $\text{NAND}(C_1, C_2) v$ has an error term of $b_2 e'_1 + C_1 e'_2$. When decrypting, we will only be considering a single row $\langle \text{NAND}(C_1, C_2), v \rangle$, and we note that each entry of $b_2 e'_1$ has an error of $B$ because $b_2$ is either 0 or 1. Furthermore, each entry of $C_1 e'_2$ is computed by summing up to $N$ entries of $e'_2$, and hence we can bound the total error as being at most $B(N + 1)$.

With this fact in mind, we can increase the depth of our circuit. If we had three ciphertexts, $C_1, C_2,$ and $C_3$, and we tried computing the depth 2 circuit $\text{NAND}(C_1, \text{NAND}(C_2, C_3))$, then the inner NAND would have an error of $B(N + 1)$, and then the outer NAND would multiply this by another factor of $(N + 1)$. We can observe that this holds for every NAND gate that we add, so a depth $L$ circuit will have an error of $B(N + 1)^L$. Thus, applying our corollary from above, we need our $\chi$ to be chosen such that $B(N + 1)^L < q/16$ in order to homomorphically evaluate depth-$L$ circuits. In other words, the GSW scheme is levelled fully homomorphic as per Definition 3.1.3, since we can just treat $L$ as a parameter and adjust the other parameters accordingly to preserve security for any $L$.

We can now show this scheme to be bootstrappable:

**Theorem 3.3.4.** The GSW Scheme is bootstrappable, i.e. it can evaluate all circuits of depth $d + 1$, where $d$ is the depth of the circuit computing its decryption function.

**Proof.** Recalling Definition 3.2.1, we need to show that the decryption function can be computed by the levelled version of this scheme, meaning that it can be computed by a circuit of depth $L$ such that $B(N + 1)^L < q/16$. We will prove bootstrapping use a slightly stronger version of LWE, following [Bar21], so assume that LWE holds for $q = 2^{n^{0.5}}$. While bootstrapping has been shown to hold for polynomial $q$ ([Bar21]), the proof for the larger $q$ is simpler and more instructive. Observe that if $L < n^{0.49}$ for large enough $n$, then the fact that $N = (n + 1)(\lfloor \log q \rfloor + 1)$ implies that

$$B(N + 1)^L = B((n + 1)(\lfloor \log q \rfloor + 1) + 1)^L < B(2n\sqrt{n})^{n^{0.49}}$$

For sufficiently large $n$, the higher exponent of $n^{0.5}$ compared to $n^{0.49}$ means that this term will be less than $q/16 = 2^{n^{0.5}}/16$, and therefore it suffices to show that $L < n^{0.49}$. To do so, we will show that the decryption circuit has a depth $\text{polylog}(n)$, which is less\(^4\) than $n^{0.49}$ for large enough $n$, which will imply the desired result.

Recall that to decrypt a ciphertext, we compute $x = \langle C_i, v \rangle$ and check if $|x| < v_i/4$. Note that figuring out the $i$ such that $v_i \in (q/4, q/2]$ is not part of the decryption circuit, since that can be computed in advance solely based on $q$. Our decryption circuit will need to take the secret key $d$ as input, and circuits can only take in boolean inputs, so we need to modify the way that we compute the decryption function. We’ll construct a circuit that takes in $d = \text{BitDecomp}(d) \in \{0, 1\}^N$. Observe that

$$\langle C_i, v \rangle = \langle \text{BitDecomp}^{-1}(C_i), d \rangle = \langle \text{Powersof2}(\text{BitDecomp}^{-1}(C_i)), \text{BitDecomp}(d) \rangle$$

By the properties of $\text{Powersof2}$ and $\text{BitDecomp}$ stated above and the fact that $v = \text{Powersof2}(d)$. Thus, we want a family of circuits $C$ such that for any $c \in \mathbb{Z}_q^N$ corresponding to $\text{Powersof2}$

\(^4\)A function $f$ is polylog($n$), or polylogarithmic, if $f(x) = p(\log(n))$ for some polynomial $p$. If $f$ is polylogarithmic, then for sufficiently large $n$, $f(n) < n^c$ for any $c > 0$. 39
Lemma 3.3.1 ([Bar21]). For every $c \in \mathbb{Z}_q^N$, there exists a function $f : \{0, 1\}^N \rightarrow \{0, 1\}$ satisfying the following three properties:

1. For all \( \hat{d} \in \{0, 1\}^N \), \( f(\hat{d}) = 0 \) if \( |\langle c, \hat{d} \rangle| < v_i/4 \).
2. For all \( \hat{d} \in \{0, 1\}^N \), \( f(\hat{d}) = 1 \) if \( |\langle c, \hat{d} \rangle| > 3v_i/4 \).
3. There exists a circuit of depth $O(\log(N)^3)$ that computes $f$.

The proof of the lemma will use some details about constructing circuits for arithmetic functions, so it is not so essential and can be safely skipped, but we will include it for completeness. Our general approach will be to use approximate arithmetic, rounding off the entries of $c$ so that we only need to use a small fraction of the digits of each, which will significantly reduce the size of our circuit.

Proof of Lemma 3.3.1. For a number $x \in \mathbb{Z}_q$, we let $\tilde{x}$ be the binary representation of $x$ rounded to the nearest multiple of $q/N^{10}$, which we accomplish by setting all bits except for the $\log(N^{10}) = 10\log(N)$ most significant bits to zero. It follows that $\tilde{x} \leq x \leq \tilde{x} + q/N^{10}$. We will let $\hat{c}$ denote the analogous approximate representation of $c$, where we round off each entry in this fashion.

Consider the function $f$ which computes $|\langle \hat{c}, \hat{d} \rangle| \mod \tilde{q}$ and outputs 0 if this value is less than $v_i/2$ and 1 otherwise. We claim that this $f$ satisfies all three of our desired conditions, and we’ll begin with (3). We will use without proof the fact that two $k$-digit numbers can be added by a circuit with $O(k^2)$ depth. Since $\hat{d}$ has values in $\{0, 1\}$, the dot product $\langle \hat{c}, \hat{d} \rangle$ is just the sum of up to $N$ numbers, each of which has $\log(N)$ digits, and thus any two of them can be added with a circuit of depth $O(\log(N)^2)$. To add $N$ of these numbers, we can use a binary tree approach: add $N/2$ pairs of numbers, then $N/4$ pairs of those sums, then $N/8$ pairs of those sums, and so on and so forth. The resulting tree of additions had depth $\log(N)$, so if each addition requires a circuit of depth $O(\log(N)^2)$, then there is a circuit of depth $O(\log(N)^3)$ computing $f$ as desired.

It remains to show that $f$ satisfies conditions (1) and (2). Before taking the modulus, \( x = |\langle \hat{c}, \hat{d} \rangle| \) is an integer between 0 and $qN$. Therefore, there must exist some $r < q, k < M$ such that $x = kq + r$, or alternatively, $x = k\tilde{q} + k(q - \tilde{q}) + r$. Hence, the difference between $x \mod q$ and $x \mod \tilde{q}$ is at most $N\tilde{q}/N^{10} = q/N^9$. Furthermore, for each entry $c_i$ in $c$, $|\hat{c}_i - c_i| \leq q/N^{10}$, and therefore $|\langle \hat{c}, \hat{d} \rangle - \langle c, \hat{d} \rangle|$ is at most $N\tilde{q}/N^{10} = q/N^9$. Combining these two facts, we note that if $|\langle c, \hat{d} \rangle | > 3v_i/4 \mod q$, then $|\langle \hat{c}, \hat{d} \rangle| \mod \tilde{q}$ will be greater than $3v_i/4 - 2q/N^9$. This value is greater than $v_i/2$, meaning that $f(\hat{d}) = 1$, giving us condition (2). Similarly, $|\langle c, \hat{d} \rangle | < v_i/4 \mod q$ implies that $|\langle \hat{c}, \hat{d} \rangle| \mod \tilde{q} < v_i/4 + 2q/N^9$, and this is less than $v_i/2$, so $f(\hat{d}) = 0$, which is condition (1). As discussed above, the case of $|\langle c, \hat{d} \rangle| \in [v_i/4, 3v_i/4]$ happens with negligible probability, so we don’t need to worry about it. We have therefore constructed an $f$ satisfying the three desired properties. \( \Box \)
This lemma gives us a way to compute the decryption circuit for any ciphertext using a circuit of depth $O(\log(N))^3$; as discussed above, if $C$ encrypts $0$ then $|\langle C_i, v \rangle| < v_i/4$ and if $C$ encrypts $1$ then $|\langle C_i, v \rangle| > 3v_i/4$, each with probability $1 - \operatorname{negl}(n)$, so once we apply \texttt{Powersof2} and \texttt{BitDecomp} to $C_i$ and \texttt{BitDecomp} to $d$, we can use the circuit from this lemma to decrypt. Since $N = O(n \log(q))$, and $\log(q) = \sqrt{n}$, these circuits have depth $O(\log(n^{3/2}))^3 = O(\log(n))^3$, i.e. polylogarithmic in $n$. As discussed at the beginning of the proof, the fact that these circuits have polylogarithmic depth means that they can be computed by the GSW levelled scheme. And, since we assume that the GSW scheme is CPA circular secure ([Bar21]), we can apply the Bootstrapping Theorem to give us a fully homomorphic scheme.

3.4 Advanced FHE: CKKS

While the GSW scheme is relatively easy to understand and thus offers a gentle introduction to fully homomorphic schemes, it is not particularly efficient. The CKKS scheme\(^5\), first introduced in [CKKS17], is a Ring-LWE based FHE scheme that is much more efficient than any predecessors. CKKS takes advantage of the fact that in many real-world applications, 100 percent accuracy is not needed. Thus, we trade a slight loss in accuracy for increased efficiency. Instead of treating noise as something that must be eliminated from ciphertexts during decrypting, the CKKS scheme is fine with a slight error in decryption as long as it’s below a certain threshold. In practice, this error does not make a significant difference. For machine learning, everything is already being approximated, so there is nothing wrong with losing a bit of accuracy, and in fact it’s actually more useful that the scheme works with real numbers instead of truncating to integers. CKKS is a levelled scheme, although a fully homomorphic version was published in a followup paper [CHK+18]. Our exposition of the CKKS scheme follows the original paper, but we elaborate on some proofs and provide examples in order to make it easier to understand.

3.4.1 CKKS Encoding

The CKKS scheme uses a special encoding scheme that allows it to pack many plaintexts into a single “message,” which is then encrypted. In addition to making encryption more efficient per ciphertext, packing allows us to parallelize homomorphic computations in a Single Instruction, Multiple Data (SIMD) fashion. Let $\mathcal{R}$ be the $M$th cyclotomic ring of degree $N = M/2$, i.e. $\mathcal{R} = \mathbb{Z}[x]/(x^N + 1)$. Like other RLWE based schemes, messages in the CKKS scheme are elements of $\mathcal{R}$, and our goal for this encoding scheme will be to transform a vector $z \in \mathbb{Z}[i]^{N/2}$ of $N/2$ Gaussian integers\(^6\) into a polynomial $m \in \mathcal{R}$ with small coefficients.

Recall the space $H$ discussed in section 2.3.2, and the notion of the canonical embedding, a map $\sigma$ that maps a number field (or in this case, the ring of integers of a number field)

\(^5\)Also known as HEAAN, after the title of the paper, “Homomorphic Encryption for Arithmetic of Approximate Numbers”

\(^6\)The Gaussian integers, mentioned in Chap. 2, are the set of all complex numbers $a + bi$ such that $a$ and $b$ are both integers.
into $H$. As discussed there, we can give elements in $\mathcal{R}$ a norm by considering their norm under the canonical embedding. Recall that the canonical embedding for a number field $K = \mathbb{Q}[x]/f(x)$ is given by $x \mapsto (\sigma_1(x), \ldots, \sigma_n(x))$, where each $\sigma_i$ maps $x$ to a root of $f$. For our purposes here, we will be most concerned with the $\ell_\infty$ norm, defining $\|x\|_\infty = \|\sigma(x)\|_\infty$.

Now, consider the group $\mathbb{Z}_M^\times$, which is the multiplicative group of all numbers $1, \ldots, M-1$ coprime to $M$, which we’ll represent instead using the numbers between $-(M/2)$ and $M/2$. In our case, $M$ is a power of two, so it consists of all the odd numbers from $-(M/2)-1$ to $M/2-1$, meaning that $|\mathbb{Z}_M^\times| = N$. Let $T$ be a cyclic subgroup of order $N/2$, in which case $\mathbb{Z}_M^\times/T$ is $\{-1, 1\}$; proving that such a group exists is out of the scope of this work, but it follows directly from the fact that 3 has order $N/2$ in $\mathbb{Z}_3^\times$, and hence 3 generates a subgroup of order $N/2$. We can then define a projection map $\pi : H \to \mathbb{C}^{N/2}$ given by $(z_j)_{j \in \mathbb{Z}_M^\times} \mapsto (z_j)_{j \in T}$.

We can also define an inverse projection map $\pi^{-1} : \mathbb{C}^{N/2} \to H$, where the $j$th coordinate of $\pi^{-1}(z)$ is $z_j$ if $j \in T$ and $\bar{z}_{-j}$ otherwise. This inverse is well-formed, since $H$ precisely consists of the vectors $(z_j)_{j \in \mathbb{Z}_M^\times}$ such that $z_j = \bar{z}_{-j}$. The canonical embedding, the projection map $\pi$, and their inverses give us the tools to encode $z \in \mathbb{Z}[i]^{N/2}$ as a message $m \in \mathcal{R}$.

**Definition 3.4.1 ([CKKS17]).** The **CKKS encoding scheme** has two algorithms, ($\text{ENCODE}$, $\text{DECODE}$). Both take in a scaling factor $\Delta$, which will be omitted as an input going forward.

- **$\text{ENCODE}(z, \Delta) = \sigma^{-1}(\lfloor \Delta \pi^{-1}(z) \rfloor) \in \mathcal{R}$**

- **$\text{DECODE}(m, \Delta)$:** Compute $\pi(\sigma(m/\Delta))$ and round it to the nearest element of $\mathbb{Z}[i]^{N/2}$.

In English, to decode a message, we divide by our scaling factor, applying the canonical embedding, and then applying our projection map and rounding; encoding is simply the inverse of this procedure, rounding to ensure that we get a valid element of $\mathcal{R}$. It follows trivially that the encoding scheme is valid, meaning that

$$\text{DECODE(ENCODE}(z)) = \left[ \pi \left( \sigma \left( \frac{\sigma^{-1}(\lfloor \Delta \pi^{-1}(z) \rfloor)}{\Delta} \right) \right) \right] = z$$

We can give a concrete example to make the encoding scheme clearer. Let $M = 8$, in which case $N = 4$, so we’re working in $\mathcal{R} = \mathbb{Z}[x]/(x^4 + 1)$. And, let our scaling factor $\Delta$ be 32. The group $\mathbb{Z}_8^\times$ is $\{-3, 1, 1, 3\}$, so we’ll take $T = \{1, 3\}$, which corresponds to to $\{\zeta_8, \zeta_8^3\}$, where $\zeta_8$ is the primitive 8th root of unity $e^{i\pi/4}$. We can pack $N/2 = 2$ Gaussian integers into a ciphertext, so we’ll let $z = (1 + i, 3 - 2i)$. To encode $z$, we first apply $\pi^{-1}$, obtaining $z' = (3 + 2i, 1 - i, 1 + i, 3 - 2i)$. Multiplying by 32 and applying $\sigma^{-1}$, we get

$$f(x) = 32 \left( \frac{\sqrt{2}}{4} x^3 + \frac{3}{2} x^2 - \frac{3\sqrt{2}}{4} x + 2 \right) \in \mathbb{R}[x] / (x^4 + 1)$$

Observe that $f$ is indeed $\sigma^{-1}(\Delta z')$, since

$$\sigma(f) = \left( f(\zeta_8^3), f(\zeta_8^{-1}), f(\zeta_8), f(\zeta_8^3) \right) = 32 \left( 3 + 2i, 1 - i, 1 + i, 3 - 2i \right)$$

\footnote{It should be noted that we’re indexing our vectors in $\mathbb{C}^N$ and $\mathbb{C}^{N/2}$ with $\mathbb{Z}_T^\times$ and $T$ respectively, instead of just $1, \ldots, N$ and $1, \ldots, N/2$. While this system seems strange, it lends itself well to the manipulations we are carrying out.}
We’re not quite done, since \( f \) is an element of \( \mathbb{R}[x]/(x^4) \), so we need to round it to the nearest polynomial with integral coordinates in order to obtain an element of \( \mathcal{R} \):

\[
m(x) = 11x^3 + 48x^2 - 34x + 64 \in \frac{\mathbb{Z}[x]}{(x^4 + 1)}
\]

To decode \( m \), we divide by 32, and apply \( \sigma \) by evaluating at \((\zeta_8^{-3}, \zeta_8^{-1}, \zeta_8, \zeta_8^3)\) to obtain

\[
\left(2.994 + 2.008i, 1.005 - 0.992i, 1.005 + 0.992i, 2.994 - 2.008i\right)
\]

Applying \( \pi \), we get \((1.005 + 0.992i, 2.994 - 2.008i)\), which we see is quite an accurate approximation, and when we round we recover \( z \).

### 3.4.2 CKKS Encryption

In order to describe the encryption scheme, we must define a few distributions which are used in the scheme. The \( N \)-dimensional discrete Gaussian \( DG(\sigma^2) \), where \( \sigma > 0 \), samples a vector in \( \mathbb{Z}^N \) where each component is independently sampled from the discrete Gaussian with variance \( \sigma^2 \). And, \( HWT(h) \) is the uniform distribution on the set of vectors in \( \{0, -1, 1\}^N \) with precisely \( h \) nonzero entries\(^8\). Finally, the distribution \( ZO(p) \) over \( \{0, -1, 1\}^N \) samples each component independently with probability \( p/2 \) each of -1 and 1 and probability \( 1 - p \) of being 0. Throughout this scheme, we will be treating vectors as polynomials, so an element of \( \mathbb{Z}^N \) sampled from \( DG(\sigma^2) \) should be understood as the coefficients of a degree \( N - 1 \) polynomial.

**Definition 3.4.2** ([CKKS17]). The parameters of the scheme are a power-of-two \( M \), an integer \( h \), an integer \( P \), and a real number \( \sigma \). Fix some integer \( p > 0 \) and a modulus \( q_0 \), and let \( q_\ell = p^\ell q_0 \) for all \( \ell \) from 1 to \( L \).

1. **Key Generation**: Sample \( s \leftarrow HWT(h) \), \( a \leftarrow \mathcal{R}_{q_L} \), and \( e \leftarrow DG(\sigma^2) \). Let \( b = -as + e \mod q_L \). Output \( sk = (1, s) \) as the secret key and \( pk = (b, a) \) as the public key. Then, sample \( a' \leftarrow \mathcal{R}_{Pq_L} \) and \( e' \leftarrow DG(\sigma^2) \) and let \( b' = -a's + e' + Ps^2 \); output \( evk = (b', a') \) as the evaluation key.

2. **Encryption**: Sample \( v \leftarrow ZO(0.5) \) and \( e_0, e_1 \leftarrow DG(\sigma^2) \). Output \( c = v \cdot pk + (m + e_0, e_1) \mod q_L \in \mathcal{R}_{q_L}^2 \)

3. **Decryption**: Given \( c = (b, a) \in \mathcal{R}_{q_L}^2 \), output \( \langle c, sk \rangle = b + as \mod q_\ell \in \mathcal{R}_{q_\ell} \)

It is also sometimes beneficial to view the Encoding/Decoding procedures as being built into the encryption and decryption algorithms respectively. Note that each ciphertext also carries some “tagged” information, so it really outputs \( (c, \ell, v, B) \), where \( 0 \leq \ell \leq L \) is the level, \( v \in \mathbb{R} \) is an upper bound on the norm of the message, and \( B \) is an upper bound on noise as measured by the norm of the error vector. A ciphertext \( c \) is a valid encryption of \( m \)

\(^8\)The number of nonzero entries in the string is formally known as its Hamming weight, which is where the name \( HWT \) comes from.
if these two error bounds are satisfied, meaning that \( \|m\|_\infty \leq v \), and \( \langle c, sk \rangle - m = e \mod q_\ell \) with \( \|e\|_\infty \leq B \). The CPA security of this scheme follows directly from the hardness of the RLWE problem, and the proof is analogous to that of the security of the RLWE Encryption Scheme (Definition 2.3.14): the public key is drawn from the RLWE distribution, but if its drawn from the uniform distribution then the ciphertext is uniform, so an adversary that can distinguish between encryptions of 0 and the uniform distribution can be used to distinguish between an oracle to the RLWE distribution and the uniform distribution.

In order to better understand CKKS encryption, we will encrypt the example from the previous section, \( m(x) = 11x^3 + 48x^2 - 34x + 64 \). We’ll use small parameters for simplicity, even though they will not actually give a secure encryption: \( M = 8, N = 4, h = 2, P = 12, \sigma = 0.5, p = 2, q_0 = 3, L = 3 \). Then, \( q_1 = 6,q_2 = 12,q_3 = 24 \). For key generation, we sample \( s = (01 - 10) \) from \( HW(2) \), \( a = 14x^3 + 9x^2 + 6x + 18 \in \mathcal{R}_{q_L} = \mathcal{R}_{24} \), and \( e = (-10000) \in \mathbb{Z}^4 \), i.e. \(-x^3 \in \mathcal{R} \), from \( DG(0.5^2) \). We then compute

\[
b = -as + e = -(14x^3 + 9x^2 + 6x + 18)(x^2 - x) + (-x^3) = x^3 - 12x^2 + 8x - 5 \in \mathcal{R}_{q_L} = \frac{\mathbb{Z}_{24}[x]}{(x^4 + 1)}
\]

And then \( sk = (1, x^2 - x) \) is the private key and \((b, a)\) is the public key.

To encrypt \( m \), we sample \( v = (10 - 10) \) from \( ZO(0.5) \) and \( e_0 = (0001), e_1 = (-10000) \) from \( DG(0.5^2) \). Our ciphertext is then \((c_0, c_1)\), where

\[
c_0 = v \cdot b + m + e_0 = (x^3 - x)(x^3 - 12x^2 + 8x - 5) + (11x^3 - 10x + 16) + 1 \mod 24
= 18x^3 - 9x^2 + 7x + 10
\]

\[
c_1 = v \cdot a + e_1 = (x^3 - x)(14x^3 + 9x^2 + 6x + 18) + (-x^3) \mod 24
= 8x^3 + 20x^2 - 3x + 8
\]

To decrypt \( c = (c_0, c_1) \), we compute

\[
\langle c, sk \rangle = c_0 + sc_1 = 18x^3 - 9x^2 + 7x + 10 + (x^2 - x)(8x^3 - 20x^2 - 3x + 8) \mod 24
= 11x^3 + 2x^2 - 9x + 14
= m(x) + (2x^2 + x - 2)
\]

As discussed above, this scheme is approximate. When we decrypt an encryption of \( m \), we don’t obtain \( m \), but we obtain something close to it; in this case, we got an element of \( \mathcal{R}_{q_L} \) that only differs from \( m \) by the small polynomial \( 2x^2 + x - 2 \). In real-world applications, the parameters will be chosen such that the error is much smaller relative to the message.

### 3.4.3 CKKS Homomorphic Operations

In addition to the usual addition and multiplication, the CKKS Scheme has an algorithm called “rescaling,” which brings a ciphertext from one modulus \( q_\ell \) to another \( q_\ell' \). This is analagous to the modulus raising techniques from other FHE schemes — the point is to transform a valid ciphertext into another valid ciphertext while reducing the amount of noise, which will give us more room for homomorphic operations. The original CKKS paper presents a number of homomorphic operations, but we will focus on the elementary arithmetic operations that allow us to construct arithmetic circuits:
• **Addition:** Given \(c_1, c_2 \in \mathcal{R}^2_{q\ell}\), \(EVALADD(c_1, c_2) = c_1 + c_2 \mod q\ell\), where addition is done component-wise.

• **Multiplication:** Given \(c_1 = (b_1, a_1), c_2 = (b_2, a_2)\), and the evaluation key \(evk\), let \(d_0 = b_1b_2, d_1 = a_1b_2 + a_2b_1\), and \(d_2 = a_1a_2\) (all mod \(q\ell\)). Output \(EVALMUL(c_1, c_2, evk) = (d_0, d_1) + [(d_2 \cdot evk)/P] \mod q\ell\).

• **Rescaling:** To rescale \(c\) from \(\ell\) to \(\ell'\),

\[
RS_{\ell \rightarrow \ell'}(c) = \left\lfloor \frac{q\ell}{q_\ell} c \right\rfloor \mod q_{\ell'}
\]

We will now turn our attention to proving that this scheme behaves as we want it to. Our proofs follow those in [CKKS17], with some details filled in. In order to prove correctness, CKKS use heuristic methods to establish high-probability bounds on the magnitudes of the ring elements, since we generally don’t care about a negligible probability of error. Observe that for a polynomial \(p(x) \in \mathcal{R}\), its image under the canonical embedding is \(p(\zeta_M)\), where \(\zeta_M\) is the primitive \(M\)’th root of 1. But \(p(\zeta_M)\) is equal to the inner product of the coefficient vector of \(p\) and \((1, \zeta_M, \zeta_M^2, \ldots, \zeta_M^{N-1})\), which has norm \(\sqrt{N}\), and hence the variance of \(p(\zeta_M)\) is \(\sigma^2N\), where \(\sigma^2\) is the variance of the coefficients of \(p\). Moreover, since each coordinate of \(p\) is independent and identically distributed, \(p(\zeta_M)\) is the sum of i.i.d. random variables, so its distribution is near-Gaussian, meaning that if \(\sigma^2\) is the variance of each coefficient, the probability that \(\|p\|\) will be great than \(6\sigma\) is negligible. Similarly, if two of these near-Gaussian random variables \(p\) and \(q\) are multiplied, and their coefficients have variance \(\sigma_p^2\) and \(\sigma_q^2\), then the same argument shows that \(\|p \cdot q\| \leq 16\sigma_p\sigma_q\) with high probability. We can use this fact to prove correctness:

**Theorem 3.4.1** (Lemma 1 in [CKKS17]). Encryption noise is bounded by \(B_{\text{clean}} = 8\sqrt{2}\sigma N + 6\sigma\sqrt{N} + 16\sigma\sqrt{hN}\). If \(c = E_{pk}(ENCODE(z, \Delta))\), and \(\Delta > N + 2B_{\text{clean}}\), then \(DECODE(D_{sk}(c), \Delta) = z\).

**Proof.** To encrypt \(c\), we sampled \(v \leftarrow ZO(0.5)\) and \(e_0, e_1 \leftarrow DG(\sigma^2)\) and let \(c = v \cdot pk + (m + e_0, e_1) \mod q\ell = (b, a)\). We then have

\[
D_s(c_0, c_1) = c_1 + c_0 s = v \cdot e + e_0 + e_1 \cdot s
\]

And thus

\[
B_{\text{clean}} = \|D_s(c_0, c_1)\|_\infty = \|v \cdot e + e_0 + e_1 \cdot s\|_\infty \\
\leq \|v \cdot e\|_\infty + \|e_0\|_\infty + \|e_1 \cdot s\|_\infty
\]

By definition of \(ZO(0.5)\), each coefficient of \(v\) has variance \(0.5\), and coefficients of \(e\) have variance \(\sigma^2\), so by the above \(\|v \cdot e\|_\infty \leq 16\sqrt{0.5^2N}\) with high probability. Similarly applying our heuristics, we see that \(\|e_0\|_\infty \leq 6\sigma\sqrt{N}\) and that \(\|e_1 \cdot s\|_\infty \leq 16\sigma\sqrt{hN}\), giving us our bound for \(B_{\text{clean}}\). Then, observe that an encryption \(c\) of \(ENCODE(z, \Delta)\) with bound \(B_{\text{clean}}\) is also an encryption of \(\Delta \cdot \sigma^{-1}(\pi^{-1}(z))\) (i.e. encoding without rounding) with bound \(B_{\text{clean}} + N/2\). As long as \(B_{\text{clean}} + N/2 < \Delta/(2)\), then the rounding in the decoding will remove this error, and hence \(DECODE(D_{sk}(c), \Delta) = z\) as desired. \(\square\)
Next, we will show how the rescaling operation affects noise:

**Theorem 3.4.2** (Lemma 2 in [CKKS17]). If \((c, v, \ell, B)\) is an encryption of \(m\), and \(c' = RS_{\ell \to \ell'}(c)\), then
\[
(c', \ell', p^{\ell' - \ell}v, p^{\ell' - \ell}B + \sqrt{N/3(3 + 8\sqrt{h})})
\]

is a valid encryption of \(p^{\ell' - \ell}m\).

**Proof.** Recalling our validity condition from above, \(e = \langle c, sk \rangle - m \mod q_\ell\) has magnitude at most \(B\). Observe that \(c' = \lceil \frac{qe}{q_\ell} \rceil\), and hence
\[
\langle c', sk \rangle = \frac{qe}{q_\ell}(m + e) + e_{\text{scale}}
\]

Where \(t = (t_0, t_1) = c' - \frac{qe}{q_\ell}c\) and \(e_{\text{scale}} = \langle t, sk \rangle\). But, by the security of the scheme \(t\) is computationally indistinguishable from the uniform distribution, since otherwise we could distinguish \(c\) and \(c'\). Thus each coefficient of \(t_0\) and \(t_1\) is indistinguishable from the uniform distribution, so its variance is approximately the variance of the uniform distribution, \(1/12\).

Thus
\[
\|e_{\text{scale}}\|_\infty = \|\langle t, sk \rangle\|_\infty \leq \|t_0\|_\infty + \|t_1 \cdot s\|_\infty \leq 6\sqrt{N/12} + 16\sqrt{hN/12} = \sqrt{N/3(3 + 8\sqrt{h})}
\]

Since \(qe/q_\ell = p^{\ell' - \ell}\), we have
\[
\langle c', sk \rangle - p^{\ell' - \ell}m = + p^{\ell' - \ell}e + e_{\text{scale}} \mod q_\ell \leq p^{\ell' - \ell}B + \sqrt{N/3(3 + 8\sqrt{h})}
\]

As desired. \(\square\)

Finally, we can prove error bounds for the homomorphic operations described above.

**Theorem 3.4.3** (Lemma 3 in [CKKS17]). Let \((c_1, \ell, v_1, B_1)\) and \((c_2, \ell, v_2, B_2)\) be encryptions of \(m_1\) and \(m_2\) respectively. Then, if \(c_{\text{add}} = EVALADD(c_1, c_2)\), \((c_{\text{add}}, \ell, v_1 + v_2, B_1 + B_2)\) is a valid encryption of \(m_1 + m_2\).

**Proof.** The message error being bounded by \(v_1 + v_2\) follows directly from the triangle inequality, so we’ll focus on the noise. If \(c_1 = (b_1, a_1)\) and \(c_2 = (b_2, a_2)\), then by definition of the inner product,
\[
\langle c_1 + c_2, sk \rangle - (m_1 + m_2) = (\langle c_1, sk \rangle - m_1) + (\langle c_2, sk \rangle - m_2)
\]

The former is bounded by \(B_1\), and the latter is bounded by \(B_2\), so by the triangle inequality
\[
\|\langle c_1 + c_2, sk \rangle - (m_1 + m_2)\|_\infty \leq B_1 + B_2
\]
as desired. \(\square\)

**Theorem 3.4.4** (Lemma 3 in [CKKS17]). With the notation as before, if \(c_{\text{mul}} = EVALMUL(c_1, c_2, evk)\), then
\[
(c_{\text{mul}}, \ell, v_1v_2, v_1B_2 + v_2B_1 + B_1B_2 + B_{\text{mul}}(\ell))
\]
is a valid encryption of \(m_1m_2\), where
\[
B_{\text{mul}}(\ell) = \frac{q_\ell}{P}\left(\frac{8\sigma N}{\sqrt{3}}\right) + \sqrt{\frac{N}{3}}\left(3 + 8\sqrt{h}\right)
\]
46
Proof. Recall that in order to multiply \( c_1 = (b_1, a_1) \) and \( c_2 = (b_2, a_2) \), we let \( d_0 = b_1b_2, d_1 = a_1b_2 + a_2b_1 \), and \( d_2 = a_1a_2 \) (all mod \( qe \)). If each message \( m_i \) is encrypted with error \( e_i \), i.e. \( \langle c_i, sk \rangle = b_i + a_is = m_i + e_i \), then we have

\[
\langle (d_0, d_1, d_2), (1, s, s^2) \rangle = b_1b_2 + (a_1b_2 + a_2b_1)s + a_1a_2 s^2
\]

\[
= (a_1s + b_1)(a_2s + b_2)
\]

\[
= (m_1 + e_1)(m_2 + e_1)
\]

\[
= m_1m_2 + e_1m_2 + e_2m_1 + e_1e_2
\]

It follows that \( \langle (d_0, d_1), sk \rangle \) must have smaller error than this. Then, since \( c_{mul} = (d_0, d_1) + \lceil (d_2 \cdot evk) / P \rceil \), we must determine how much error is added when we add this term. Since \( evk = (-a's + e' + Ps^2, a') \) where \( a' \leftarrow \mathcal{R}_{qL} \), there is an additional error here of \( e'' = d_2e'/P \), as well as a rounding error, which we know from our proof of Theorem 3.4.2 is at most \( \sqrt{N/3(3 + 8\sqrt{h})} \). To estimate \( e'' \), we observe that \( d_2 \) is indistinguishable from the uniform distribution over \( \mathcal{R}_{qL} \), so by variance of the uniform distribution and by our distribution bounds discussed above,

\[
\|d_2e'\|_\infty \leq 16\left( \sqrt{\frac{Nq^2}{12}} \right) \sqrt{N\sigma^2} = 8N\sigma qL \left( \frac{1}{\sqrt{3}} \right)
\]

Combining all of these facts, we see that

\[
\|\langle c_{mul}, (1, s) \rangle - m_1m_2\|_\infty = \|\langle (d_0 + \lceil (d_2 \cdot evk) / P \rceil, d_1 + \lceil (d_2 \cdot evk) / P \rceil, (1, s) \rangle - m_1m_2\|_\infty
\]

\[
\leq \|e_1m_2 + e_2m_1 + e_1e_2 + e'' + \sqrt{N/3(3 + 8\sqrt{h})}\|
\]

But we know bounds for the \( m_i \) and \( e_i \), so we have

\[
\|\langle c_{mul}, (1, s) \rangle - m_1m_2\|_\infty \leq v_1v_2, v_1B_2 + v_2B_1 + B_1B_2 + \frac{qL}{P} \left( \frac{8\sigma N}{\sqrt{3}} \right) + \sqrt{\frac{N}{3}} \left( 3 + 8\sqrt{h} \right)
\]

Together, these three lemmas give us useful bounds on the ciphertext noise after carrying out homomorphic operations. Recall that CKKS is not required to be exactly right like other schemes are — instead, CKKS is approximate, meaning that there will always be a little bit of noise, so it’s up to the user how much. We see from these equations that one can pick values of \( P, \sigma, N \), and the moduli \( qL \) to determine whatever noise bound they wish.
Chapter 4

Machine Learning Applications

Anything that blocks that flow of information, such as strict regulations regarding the use of human genetic data, will inevitably hurt the effectiveness of both research and medical care.

David Korn, Harvard Medical School [oB98]

Having seen how fully homomorphic encryption has been realized, we can now look at some applications for machine learning. As machine learning has become incredibly widespread, many concerns have been raised about data privacy, but keeping data private makes it difficult to perform accurate analyses. This conundrum is best summarized by the above quote — it seems straightforward that anything that increases the privacy of genetic data would make data analysis harder. However, the privacy-preserving nature of FHE is perfectly suited to solve these concerns, since we can maintain total privacy without compromising on the accuracy of our analyses. It should be noted that our question is not “can we do fully homomorphic machine learning?” This question has an obvious answer: by Definition 3.1.2, a fully homomorphic encryption scheme can homomorphically evaluate any function, so as long as we can describe any machine learning problem in terms of a boolean function — which turns out to be fairly straightforward to do — we can solve it homomorphically. However, such an answer isn’t really useful. It doesn’t tell us how practical FHE-based machine learning schemes are (if at all), and it also doesn’t reveal any specific ways that FHE can be used to add value and solve specific problems with current ML schemes. In this chapter, we will address both of these topics. Indeed, we will see that fully homomorphic machine learning is becoming increasingly efficient. Moreover, by enabling machine learning analyses of sensitive data, especially medical data, the development of FHE-based private ML protocols has crucial societal ramifications.

The main setting we will be considering is a client-server model, where the client does not have enough computing resources to train machine learning models, and therefore they want to use a cloud-based ML service. However, the client needs to keep the data confidential for
one reason or another, and thus it cannot simply upload all of its data to the ML service provider. Using FHE, the client can simply encrypt the data, the server will do everything homomorphically, and then the client can decrypt the result, as depicted in Figure 4.1.

![Figure 4.1: A client interacting with an ML-service provider that evaluates the model $h$ on encrypted data](image)

We will assume that the server is Honest but Curious, which is the threat model commonly used in fully homomorphic ML protocols. Honest means that the server will follow the agreed-upon protocol, but it is Curious in that it will peek at any data it has access to.

### 4.0.1 Why is Fully Homomorphic ML Useful?

Before we begin our study of FHE in machine learning, we must ask a fundamental question: what value does FHE add to machine learning? Why is it worth the extra time to evaluate a network homomorphically?

The primary use case for FHE in machine learning is settings with strict data requirements. In finance, for example, it can be used by banks to better detect fraud without revealing information about innocent users’ finances. FHE has also been demonstrated to work for text classification, which enables analyses to be done on secure messages ([BHM+20]). The most notable use case, though, is medicine, since medical data is highly regulated by laws like HIPAA in the US and the GDPR in the EU. If a hospital wants to use a ML-as-a-service provider to help their doctors diagnose illness, they could very well find themselves unable to. With FHE, hospitals can simply encrypt their data and obtain predictions homomorphically, allowing patients to receive better treatment while preserving their privacy. Fully homomorphic networks have been used to classify tumors ([HPC+22]), predict heart attacks ([BLN14]), and remotely monitor blood pressure and ECG data ([PKA+14]), all without revealing an iota of information about the patient due to the ironclad guarantees provided by CPA security.

A related use case is with genetic data. Even in settings where there are no legal restrictions, people tend to be protective of their genetic data, and patients are less likely to participate in research that involves genetic data due to privacy concerns [FGMJ19]. FHE allows genetic research to be done without any of these concerns. For example, genome-wide association studies (GWAS) are used to analyze how genes produce different traits and
to predict outcomes based off of genomes. Recent work has performed accurate privacy-preserving GWAS using FHE [KHB+20], enabling these valuable studies to be done in a private manner.

Finally, FHE is useful in allowing parties that do not trust each other to collaborate. As we’ll see later in Section 4.3, many individuals can jointly train a machine learning model on their data without any individual sharing any data with the others. Additionally, one party can send encrypted data to another party, which will evaluate an encrypted neural network on the data homomorphically and send it back to be decrypted [KWN20]. In this case, the first party obtains accurate predictions, and the second party does not risk leaking information from their network. Overall, the use of FHE for private machine learning has tremendous potential.

4.1 Machine Learning Background

In this work, we will be focusing on the subset of machine learning known as supervised learning. The goal of supervised learning is to approximate an unknown function \( f : X \rightarrow Y \) based on a number of samples \((x_1, f(x_1)), \ldots, (x_n, f(x_n))\). In order to do so, a model is trained, meaning that it uses the samples \((x_1, f(x_1)), \ldots, (x_n, f(x_n))\) to come up with a model \( h \), with the aim being that \( h(x) = f(x) \) with extremely high probability over the choice of \( x \). Then, \( h \) can be used to evaluate \( f \) on new data, a step also known as inference.

We can generally subdivide machine learning tasks into two categories: regression, which is the problem of predicting the value of a real variable such as temperature or weight, and classification, which is predicting the value of a discrete value, such as labeling a tumor as “benign” or “malignant.” For example, consider the classification task of handwriting recognition: \( X \) is the set of all possible images of a certain size (possibly represented as a vector of RGB values), \( Y \) is the alphabet, and \( f : X \rightarrow Y \) is the function that maps each photo of a letter to the letter in the alphabet that it depicts. We could use a series of labeled photos to train a model \( h \), and then evaluate \( h \) on new photos in order to scan them.

There are a number of different methods that are used for machine learning, but in this work, we will primarily be focusing on Neural Networks, which are both extremely widespread and extremely powerful. In fact, neural networks are known to be universal approximators, meaning that a big enough neural network can approximate any function [Bis06]. For details on other ML methods, we refer the reader to an introductory machine learning book like [Deu] or [Bis06].

A neural network consists of a number of nodes organized into layers that are formed into a circuit. There is an input layer and an output layer, and any intermediary layers are known as hidden layers. Figure 4.2 depicts a simple neural network, with three inputs, one hidden layer, and a single output. The network computes a value by following the arrows. Each node receives values from the nodes in the previous layer, and it computes a linear combination of the vectors and a weight vector, which contains values in \([0, 1]\). Thus, in our example, the top node in the hidden layer would compute \( a = w_0 + x_1w_1 + x_2w_2 + x_3w_3 \), where \( w = (w_0, w_1, w_2, w_3) \) is the node’s weight vector, and \( w_0 \) is the bias. This value is referred to as the activation. Then, we apply a special function \( f \) known as the activation function to \( a \), and the node outputs \( f(a) \). This output is then input to the nodes in the
next layer. Note that every node in one layer need not be connected to every node in the next layer; in our example, the middle node in the hidden layer does not have $x_2$ as input, which is equivalent to $w_2$ being zero for that node. We can thus formalize the computation of this neural network — if $w_j = (w_{j,0}, \ldots, w_{j,3})$ is the weight vector for the $j$'th node from the top in the hidden layer, $w_y$ is $y_1$'s weight vector, and $f$ is our activation function, then

$$y_1 = f \left( w_{y,0} + \sum_{j=1}^{3} w_{y,j} f \left( w_{j,0} + \sum_{k=1}^{3} w_{j,k} x_k \right) \right)$$

In practice, we might want to use a different activation function for different layers, but the principle is still the same. Some popular choices of activation functions ([Deu, GBDL+16]) include:

- The hyperbolic tangent function $tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
- The sigmoid function $f(x) = 1/(1 + e^{-x})$
- Rectified Linear Units, or ReLU, $f(x) = \max(0, x)$
- The softmax function, which is a generalization of the sigmoid function to $D$-dimensional vectors. The $i$'th entry in the softmax is
  $$\text{softmax}_i(x) = \frac{e^{x_i}}{\sum_{0 \leq j \leq D} e^{x_j}}$$
- Max/mean pooling, which compute the maximum value and mean respectively of some subset of the preceding layer.
There are different types of neural networks, which use different types of layers and specialized cells. Many of the works we will discuss use convolutional neural networks, or CNNs, which achieve particularly good performance on image classification problems. CNNs add two types of network layers to our arsenal. Convolutional layers have one or multiple filters, which each sample a smaller section of the input according to its weights, “stepping” across the image to apply the filter to every section. For example, in Figure 4.3, the filter is $[1, 0, 1]$, and we apply it to every group of 3 neurons. In practice, CNNs are often used on 2-dimensional images, in which case the filters are matrices that sample smaller squares of pixels from the image. The second layer type is max and mean pooling layers, which compute the max and mean respectively of a subset of inputs. These layers serve to reduce dimensionality and make the model more efficient.

![Mean Pooling Diagram](Image)

**Figure 4.3:** An example of a 1-dimensional Convolutional Neural Network, with a convolutional layer and mean pooling. The Convolutional layer has filter $[1,0,1]$.

### 4.1.1 Training Neural Networks

One crucial question remains: how are neural networks trained? The performance of a network is evaluated against a **loss function** $L$, which measures how accurate a network’s predictions are. The goal of training is thus to minimize the loss function against a set of test data. One notable loss function is the least squares loss, $L(h) = \frac{1}{2} \sum_{i=0}^{n} \| h(x_i) - y_i \|^2$, where $(x_i, y_i)$ are samples. We can think of a loss function as taking all of the network’s parameters as inputs, and thus our goal in training a network is to minimize $L$ with respect to all of the weights and biases.

In an ideal world, there would be some closed-form way to compute the parameters that minimize $L$, but unfortunately such a method does not exist. Instead, the primary method used is known as **gradient descent**. A useful analogy for optimization is to think of the function’s outputs as being topography; thus, in our case, we are standing on a hill, and looking for the lowest point in the valley (i.e. the minimum). Gradient descent works by taking a sequence of small steps, each pointing downhill of our current location. It’s reasonable that this procedure will eventually lead us to the minimum, or at the very least
to some local minimum close to it. Mathematically, the gradient of a function, denoted by \( \nabla \), is a vector that points “uphill,” i.e. in the direction where the function is most increasing, and hence its negation is pointing directly downhill. Thus, given a parameter vector \( w \), the vector \( w' = w - \nabla L(w) \) represents the parameters that are a step downhill from our current parameters. Typically, a parameter \( \eta \), called the learning rate, is used to govern the size of the steps, so \( w' = w - \eta \nabla L(w) \) will be the equation used in practice. And, since it’s often impractical to compute \( \nabla L \) using all of the training data once, it’s common to instead use a randomly chosen subset of the data, even as small as one point [Deu]. This method is known as stochastic gradient descent, or SGD. Frequently, SGD will run for a fixed number of passes over the training data, or epochs.

In order to use gradient descent, we need to be able to compute the derivative of our loss function with respect to the parameters. We do so using a procedure called backpropagation. The main idea of backpropagation is the observation that a neural network is just a composition of a bunch of functions of the input values, namely the sums at the nodes and the activation functions. Thus, we can use the chain rule from calculus \(^1\) to unfold the compositions and compute the derivatives that compose the gradient. We’ll use the least squares loss \( L(w) = \frac{1}{2} \sum_{i=0}^n \| h(x_i) - y_i \|^2 \) as an example, and the input \( w \) is a vector containing all of the weights from all the nodes. Let \( a^l_j \) be the activation of the \( j \)’th node in the \( l \)’th layer, let \( w_{j,i}^l \) be the \( i \)’th weight at that node, and let \( z^l_j \) be the output of that layer, so \( z^l_j = f(a^l_j) \), where \( f \) is the activation function. Recall from earlier that the activation at each node is computed by the equation \( a^l_j = \sum_i w^l_{j,i} z^{l-1}_i = \sum_i w^l_{j,i} f(a^{l-1}_i) \)

Where \( N \) is the number of nodes in the previous layer. To compute the derivative of \( L \) with respect to a specific weight \( w^l_{j,i} \), we can first use the chain rule and observe that

\[
\frac{\partial L}{\partial w^l_{j,i}} = \frac{\partial L}{\partial a^l_j} \frac{\partial a^l_j}{\partial w^l_{j,i}} = \frac{\partial L}{\partial z^{l-1}_i}
\]

By the previous equation. Note that we will often use the notation \( \delta^l_j \) for \( \frac{\partial L}{\partial a^l_j} \), and since we compute \( z^{l-1}_i \) anyways during the forward pass, computing \( \delta^l_j \) for each \( l, j \) will give us the gradient. For an output node \( \hat{y}_j \), which doesn’t have an activation function, computing the gradient \( \partial L/\partial \hat{y}_j \) is simple, and thus

\[
\delta^l_j = \frac{\partial L}{\partial a^l_j} = \frac{\partial L}{\partial \hat{y}_j} = \hat{y}_j - y_j
\]

Since in this case, \( L(w) = \frac{1}{2}(\hat{y}_j - y_j)^2 \), where \( y_j \) is the actual value from the training data. For nodes in hidden layers, we can again apply the chain rule with respect to the activations from the next layer:

\[
\delta^l_j = \frac{\partial L}{\partial a^l_j} = \sum_{i=1}^N \frac{\partial L}{\partial a^{l+1}_i} \frac{\partial a^{l+1}_i}{\partial a^l_j}
\]

\(^1\) The chain rule states that the derivative of \( f \circ g \) at \( x \) is \( f'(g(x))g'(x) \)
Substituting and applying the chain rule to the calculation of activation at each node, we see that

\[ \delta_l^j = \sum_{i=1}^{N} \left( \delta_{l+1}^{i+1} \frac{\partial a_{l+1}^i}{\partial a_l^j} \right) = \sum_{i=1}^{N} \left( \delta_{l+1}^{i+1} \frac{\partial (\sum_m w_{j,m}^l f(a_m^l))}{\partial a_l^j} \right) = \sum_{i=1}^{N} \left( \delta_{l+1}^{i+1} f'(a_l^j) w_{j,i}^l \right) \]

We can then pull out the common factor of \( f'(a_l^j) \) to conclude that

\[ \delta_l^j = f'(a_l^j) \sum_{i=1}^{N} \delta_{l+1}^{i+1} w_{j,i}^l \]

In English, this means that in order to compute \( \delta \) with respect to a given weight, we need three things: the derivative of the activation function, the weights from the next layer, and the \( \delta \)s from the next layer. And, as discussed earlier, the gradient with respect to \( w_{j,i}^l \) is just \( \delta_l^j \cdot z_{l-1}^i \). If we proceed backwards from the output nodes, then at each node we will have already computed all of this information, and thus the computation is quite straightforward. In other words, the errors propagate backwards through the network, from output to input, which is what gives backpropagation its name. Note that the above equations are for computing the gradient at a single point; if we used a batch of points \( B \), then we would simply compute the gradient at each \( b \in B \) and then average them. We’ll use the notation \( \nabla L(w; B) \) to denote computing the loss of the parameters \( w \) on batch \( B \).

### 4.2 How FHE is used in ML

As mentioned above, it’s easy to show that ML can be done homomorphically, but doing so efficiently is a different story. In general, FHE is quite inefficient, although as discussed last chapter, efficiency has still improved significantly since Gentry’s initial scheme. Levelled FHE schemes are particularly useful here, since when we’re evaluating an ML model we will usually know its depth in advance.

Moreover, there are other efficiency challenges that are unique to the ML setting. Machine learning often requires preprocessing steps that use functions which are hard for FHE to compute. Preprocessing might be too computationally intensive for the client to carry out, but also too complex to be done homomorphically by the server. FHE schemes can only support ciphertexts of a limited size, whereas ML datasets can be massive. For example, one study ([HPC+22]) which classifies tumors based on millions of genetic datapoints is limited to ciphertexts of 65536 elements by the CKKS scheme. It therefore requires many ciphertexts, which is extremely inefficient. Preprocessing the data to reduce the number of entries can help with this issue, but as mentioned above is not always so trivial.

Finally, FHE does not compute all functions equally well. While it’s true that all arithmetic circuits can be computed using addition and multiplication, it’s sometimes quite inefficient to do so. It’s not just obscure functions that have this issue — division, for example, cannot be practically done by FHE schemes ([GLN12]), and thus has to be approximated. ML in particular faces challenge in this regard, since almost all of the common activation functions are not FHE-friendly. ReLU and max/min pooling use a comparison, and \( \tanh \), sigmoid, and softmax use exponentials. These real-valued functions cannot fully be represented...
by boolean circuits, but instead they can only be approximated to a fixed number of bits. We can construct a circuit that computes the first \( n \) bits of \( e^x \), and this circuit can be composed of just addition and multiplication, but there is a significant tradeoff here: the accuracy of the approximation is directly correlated with the degree of the polynomial, but higher-degree polynomials require a higher level for the FHE scheme, which needs larger parameters makes evaluation take longer. For example, \( \lim_{d \to \infty} \sqrt{x_1^d + \ldots + x_n^d} \) is \( \max(x_1, \ldots, x_n) \), so this polynomial can be used to approximate max pooling, as in [GBDL+16].

4.2.1 A Brief History of Fully Homomorphic ML

The history of fully homomorphic ML is defined by the different ways computer scientists have overcome these obstacles. As homomorphic encryption has become more and more efficient, ML scientists have gained more and more tools at their disposals. The first significant attempt at homomorphic machine learning was the ML Confidential paper [GLN12]. They didn’t use neural networks, which were too complex for the homomorphic encryption schemes of the time; instead, they trained a Linear Means Classifier and a Fisher Linear Discriminant Classifier — two simpler ML models — using the BFV levelled FHE scheme.

A major breakthrough in fully homomorphic ML was the Cryptonets paper, which demonstrated that it was feasible to evaluate a fully homomorphic neural network [GBDL+16]. They leverage a number of optimizations in order to do so. First of all, they don’t actually encrypt the network. In some FHE schemes, it is possible and more efficient to homomorphically multiply a ciphertext by non-encrypted plaintext than by another ciphertext, so this speeds up the weighted sum at each node of the network; computing \( \sum_{i \leq n} w_i x_i \) only requires \( n - 1 \) homomorphic additions instead of \( n \) multiplications and \( n - 1 \) additions. Instead of using a nonpolynomial activation function, the neural network in Cryptonets simply uses \( f(x) = x^2 \), which is a good enough activation function for many purposes, and it only needs a single homomorphic multiplication. Their results are quite impressive given the limitations inherent to using FHE — using a CNN trained on the MNIST dataset\(^2\) of handwritten digits with 98.95\% accuracy, they were able to classify over 50,000 images per hour. This neither as fast nor as accurate as a non-encrypted network; for comparison, the state of the art at the time Cryptonets was published had an accuracy of 99.79\% ([WZZ+13]). Nonetheless it is extremely impressive for an “inefficient” technology like FHE, since many optimizations used in state of the art models are difficult or impossible to implement in the FHE setting. [CdWM+17] improve on the Cryptonets result by using low-degree polynomials to approximate ReLU as their activation function, allowing them to use a deeper CNN to obtain a 99.3\% accuracy. The aptly-named Faster Cryptonets ([CBL+18]) is an even bigger improvement, both in terms of speed and accuracy.

As FHE efficiency has improved with newer schemes like CKKS and TFHE, recent papers have been able to push the state of the art even further. SHE ([LJ19]) uses the improved performance TFHE to homomorphically train a much deeper network than predecessors. Moreover, they actually compute ReLU and max pooling, which gives a significant accuracy improvement over the approximations used in previous works. They achieve a 99.77\%
accuracy on the MNIST handwriting dataset, which is close to the state of the art for non-encrypted networks, in half as much time as Cryptonets. A shallower network has 99.54% accuracy in under 4% of the time of Cryptonets. Furthermore, hardware optimizations can be combined with theoretical advances to deliver even faster homomorphic ML. CraterLake ([SFK+22]) is a hardware accelerator designed to optimize CKKS homomorphic operations, which speeds up computations by an order of magnitude compared to the usual CPU implementation.

4.3 Federated Learning and Multikey FHE

In this section, we will do a deeper dive into one of the most promising uses of fully homomorphic machine learning. Suppose that a group of hospitals all wanted to use machine learning to improve their predictions based on X-rays (as in [WCK+22]). They might want to use federated learning, a machine learning paradigm that allows a number of small clients to pool their resources to train one large ML model, coordinated by a central server. Federated learning allows these parties to achieve better accuracy than they would have on their own, and it in theory provides better privacy than outright sending data to the other parties. However, attacks ([MSCS18]) have been found that allow information recovery directly from the model, so one client can obtain another client’s information solely with the model parameters sent by the server. Due to the high privacy requirements associated with healthcare data, these attacks all but rule out federated learning for this case.

Once again, fully homomorphic encryption saves the day. In this section, we will explore how a variant of FHE known as multikey FHE allows a number of parties to encrypt their data under different keys, and then the central server can coordinate the federated learning homomorphically. By keeping all the data encrypted, we prevent any information from being learned, thus preserving all the privacy of the patients, while allowing the hospitals to use the improved predictions that result for lifesaving purposes.

4.3.1 Multikey FHE Background

Multikey FHE, or MKFHE, allows multiple participants to encrypt data with their own keys, and then functions can be homomorphically evaluated on that data. We can formalize this notion by modifying our definition of a FHE scheme:

**Definition 4.3.1 ([LATV13, PS16]).** A scheme \((SETUP, G, E, D, EVAL)\) is **multikey-fully-homomorphic** if for all \(N\), the following properties hold:

- \(SETUP\) outputs a “public parameter” \(pp\), which is shared by all parties. All the other algorithms implicitly have \(pp\) as input.
- Key Generation, Encryption, and Evaluation are unchanged.
- Decryption requires any secret keys \(d_1, \ldots, d_k\) that were involved in creating the ciphertexts.
- Encryption is CPA-secure.
• Correctness: Let $F$ be some boolean function with $N$ inputs, and suppose that there are $k$ parties. Let $\pi : [N] \rightarrow [k]$ “assign” an input to each party, and let $x = (x_1, \ldots, x_N) \in \{0, 1\}^N$. If we generate $k$ key pairs $(e_i, d_i) \leftarrow G(1^n)$ and encrypt $c_i = E_{e_{\pi(i)}}(x_i)$, then the scheme correctly computes $F$ if

$$D_{d_1, \ldots, d_k}(\text{EVAL}(F, c_1, \ldots, c_N)) = F(x_1, \ldots, x_N)$$

With probability $1 - \text{negl}$. The scheme is correct if this holds for all $F$ and for all $\pi$.

• Compactness: Just as in Definition 3.1.1, there must exist a polynomial $p$ such that $c = \text{EVAL}(F, c_1, \ldots, c_N)$ satisfies $|c| < p(n, k)$.

Most common MKFHE schemes are “multi-hop,” meaning that we can homomorphically evaluate a function on a ciphertext, and then use that result in another computation with a ciphertext that was encrypted under different keys. A multikey scheme where all parties have to be known in advance is called “single-hop.” Also, note that we will be working in the Common Reference String (CRS) model, where there is a string (the public parameter) that all parties will have access to.

Multikey FHE was first proposed in [LATV13], based on NTRU, a different lattice problem. Since then, multikey versions of popular FHE schemes have been proposed: [CM22] developed a single-hop LWE-based MKFHE scheme, and [PS16] and [BP16] simultaneously published simpler and more efficient multi-hop schemes that are based on the GSW scheme. More recently, [CDKS19] and [CCS19] have proposed multikey variants of CKKS and TFHE respectively.

### 4.3.2 Federated Learning

As mentioned above, federated learning allows a number of clients to train a neural network together, with the procedure moderated by a central server. There have been a few different federated learning protocols proposed, but we will be using the FederatedAveraging algorithm from [MMR+17]. If there are $K$ clients, then in each round the server selects a random subset of clients, and each one runs batched SGD on its training dataset for a few epochs. Then, the new parameters from each client are sent back to the server, which computes their average. This is approach is preferable to the naive method of simply having every client run SGD at every step, since it significantly reduces communication overhead without compromising on accuracy.

### 4.3.3 A Fully Homomorphic Federated Learning Protocol

We will present a modified version of the FHE-based secure federated learning protocol from [MNSL21]. In particular, the scheme we present computes the weighted average of the parameters from each client, instead of the average as in [MNSL21], making it more useful for settings when clients have different sized datasets. Also, just as in the original FederatedAveraging, we only have a subset of the clients send weights each round.

In order to implement homomorphic federated learning, the authors of [MNSL21] modify the multikey CKKS-based scheme from [CDKS19] to be better suited for the federated learning setting, and they term the new scheme xMK-CKKS. Observe that the FederatedAveraging
Algorithm 1 FederatedAveraging: There are \( K \) clients interacting with the server. The parameters are a learning rate \( \eta \), a number of epochs \( E \), a batch size \( B \) for the client-side training, and a fraction \( C \) of the clients that are used in each round. Client \( k \) has training data \( D_k \).

```plaintext
# Client \( k \) receives weights \( w \)
function CLIENTUpdate(\( k \), \( w \)):
    \( B \leftarrow D_k \) partitioned into batches of size \( B \)
    for Epoch in 1...\( E \) do
        # Run SGD on the batches
        for \( b \in B \) do
            \( w \leftarrow w - \eta \nabla L(w; b) \)
        send \( w \) to the server

Server’s Procedure:
\( w_0 \leftarrow \) initial parameter setting
for round \( t \) do
    \( S_t \leftarrow \) random subset of clients of size \( \max(CK, 1) \)
    # Compute the weighted average
    for client \( k \) in \( S_t \) do
        \( w_{t+1}^k \leftarrow \) CLIENTUpdate(\( k \), \( w_t \))
    \( n \leftarrow \sum_{k \in S_t} |D_k| \)
    \( w_{t+1} \leftarrow \frac{1}{n} \sum_{k \in S_t} |D_k| w_{t+1}^k \)
```

Protocol only really requires homomorphic addition. Thus, we don’t present a homomorphic multiplication algorithm (so technically, xMK-CKKS is MK-partially homomorphic), and we use a special decryption method that works on ciphertexts that are the sums of other ciphertexts.

Definition 4.3.2 ([MNSL21]). The parameters of the xMK-CKKS Scheme with \( N \) parties are a RLWE dimension \( n \), a ring \( \mathcal{R} \), a modulus \( q \), key distributions \( \kappa \) and \( \psi \), and error distributions \( \chi \) and \( \phi \).

- **Setup**: Sample a random \( a \leftarrow \mathcal{R}_q \) uniformly, and output this as the public parameter.

- **Key Generation**(\( G \)): Each device \( d_i \) generates its secret and public keys \( (s_i, b_i) \) using the key generation algorithm from the CKKS scheme \(^3\) (recall Defn 3.4.2): sample \( s_i \leftarrow \kappa \) and \( e^i \leftarrow \chi \), and let \( b_i = -a \cdot s_i + e^i \mod q \). Then, we output the aggregate public key, \( b_{aggr} = \sum_{i \leq N} b_i \).

- **Encryption**(\( E \)): For client \( i \) to encrypt \( m \), sample \( v \leftarrow \psi \) and \( e_0, e_1 \leftarrow \chi \), and output \( E_{b_{aggr}}(m) = (v \cdot b_{aggr} + m + e_0, v \cdot a + e_1) \). This is precisely the CKKS encryption of \( m \), but replacing the public key \( b \) with the aggregate public key.

\(^3\)In our concrete description of CKKS in the previous chapter, we were using \( HW_T(h) \) as \( \kappa \), \( DG(\sigma^2) \) as \( \chi \), and \( ZO(0.5) \) as \( \psi \). We’re using the more generalized form here to more closely follow the notation from [MNSL21].
• **Decryption of sums** ($\text{SUMDEC}$): Given a sum ciphertext $c_+ = (c_0, c_1)$ and the secret keys $s_1 \ldots s_N$, each client $i$ computes a “decryption share,” $D_i = s_i \cdot c_1 + e_i^*$, where $e_i^* \leftarrow \phi$. Then,

$$\text{SUMDEC}(c_+, D_1, \ldots, D_N) = c_0 + \sum_{i=1}^N D_i \mod q$$

• **Homomorphic Addition** ($\text{ADDEV AL}$): Addition is done componentwise in the natural way, so if $(c_i^0, c_i^1) = c_i$, then

$$\text{ADDEV AL}(c^1, \ldots, c^N) = \left( \sum_{i=1}^N c_i^0, \sum_{i=1}^N c_i^1 \right)$$

This scheme is slightly different than what is presented in [MNSL21], since their public key and public parameter are $d$-dimensional vectors over $R_q$, but the extra dimensions are not needed for these purposes. In any event, we can use the algorithms from this scheme in our secure federated learning protocol. The protocol is just a modified $\text{FederatedAveraging}$ that encrypts each client’s weights to avoid information leakage.

To prove that $\text{SecureFederatedAveraging}$ works and is secure, we need to be certain that xMK-CKKS works as desired.

**Lemma 4.3.1.** $\text{SUMDEC}$ is correct, meaning that if $c_+ = (c_0, c_1) = \text{ADDEV AL}(E(m_1), \ldots, E(m_N))$, then $\text{SUMDEC}(c_+, D_1, \ldots, D_N) = \sum_{i=1}^N m_i + e$ for a small error $e$, assuming that the decryption shares are computed correctly.

**Proof.** This proof is a straightforward computation. We use $e_i^0$ to denote the $e_0$ from client $i$’s encryption, and so on and so forth.

$$\text{SUMDEC}(c_+, D_1, \ldots, D_N) = c_0 + \sum_{i=1}^N D_i \mod q$$

$$= c_0 + \sum_{i=1}^N s_i \cdot c_1 + \sum_{i=1}^N e_i^*$$

$$= \sum_{i=1}^N (v^i \cdot b_{aggr} + m_i + e_i^0) + \sum_{i=1}^N s_i \cdot \left( \sum_{i=1}^N (v^i \cdot a + e_i^1) \right) + \sum_{i=1}^N e_i^*$$

Applying the definition of $b_{aggr}$, we obtain

$$\sum_{i=1}^N v^i \cdot \left( \sum_{i=1}^N -a \cdot s_i + e_i^1 \right) + \sum_{i=1}^N m_i + \sum_{i=1}^N e_i^0 + \sum_{i=1}^N s_i \cdot \left( \sum_{i=1}^N (v^i \cdot a + e_i^1) \right) + \sum_{i=1}^N e_i^*$$

Which allows us to cancel the $v^i \cdot a \cdot s_i$ terms, giving us

$$\sum_{i=1}^N m_i + \left( \sum_{i=1}^N v^i \cdot \left( \sum_{i=1}^N e_i^1 \right) \right) + \sum_{i=1}^N e_i^0 + \sum_{i=1}^N s_i \cdot \left( \sum_{i=1}^N e_i^1 \right) + \sum_{i=1}^N e_i^*$$
Algorithm 2 SecureFederatedAveraging

function ClientUpdate(k, w):
    $B \leftarrow D_k$ partitioned into batches of size $B$
    for Epoch in 1...$E$
        # Run SGD on the batches
        for $b \in B$
            $w \leftarrow w - \eta \nabla L(w; b)$
        $w_{\text{scaled}} \leftarrow D_kw$
        send $E(w_{\text{scaled}})$ to the server

Server’s Procedure:

$w_0 \leftarrow$ initial parameter setting
$a \leftarrow \text{SETUP}(1^n)$
Clients run $G(1^n)$, and the server computes $b_{\text{aggr}}$ and sends to clients
for round $t$
    $S_t \leftarrow$ random subset of clients of size $\max(CK, 2)$
    # Compute the weighted average
    for client $k$ in $S_t$
        $c^k \leftarrow \text{CLIENTUPDATE}(k, w_t)$
    $c_+ \leftarrow \text{EVALADD}(c^1, \ldots, c^N)$
    for client $k$ in $S_t$
        Client $k$ computes and sends the decryption share $D_k$
    $n \leftarrow \sum_{k \in S_t} |D_k|$
    $w_{t+1} \leftarrow \frac{1}{n} \text{SUMDEC}(c_+, D_1, \ldots, D_k)$

Notice that the parenthetical term is all sums of errors, so assuming that we chose our distributions $\kappa, \chi, \psi, \phi$ correctly (like we did in CKKS, see proof of Theorem 3.4.3), this term will be small with extremely high probability, as desired.

Just as with CKKS, this scheme is approximate, so a user should choose parameters according to the desired size of the decryption error. It should be noted that we can only guarantee correctness in this Honest-But-Curious setting, when all parties follow the protocol. If a group of clients decide to, say, send fake weights to mess with the federated learning, the current protocol would end up with an inaccurate model. Designing a scheme that is resilient to these attacks is unfortunately beyond the scope of this work.

This protocol is also secure, both in the sense that the server cannot learn the clients’ private information, but also the clients cannot learn about each others’ private information. There are two pieces of information that each client computes: the ciphertext and the decryption share. Any client’s ciphertext is

$$c = \left( v \cdot \left( \sum_{i=1}^{N} -a \cdot s_i + e^i \right) + m + e_0, v \cdot a + e_1 \right)$$
By the hardness of RLWE and security of CKKS, \( v \cdot (-a \cdot s_i + e^i) + m + e_0 \) is indistinguishable from uniform for any \( i \), and thus encryption is CPA secure. The decryption shares

\[
D_i = s_i \cdot \left( \sum_{i=1}^{N} (v^i \cdot a + e_i^i) \right) + e_i^*
\]

Are indistinguishable from uniform for the exact same reason, since each term in the sum is indistinguishable from uniform by the hardness of RLWE.

Our scheme differs from that of [MNSL21] in that we only use a subset of clients from each round and that the server computes a weighted average. Each of these introduces potential security concerns which can be mitigated. We require a minimum of 2 clients in \( S_i \) each round, since if there were only a single client, the server would reveal its weights. Moreover, if clients do not wish for the server to obtain the sizes of their datasets, they can each encrypt \( |D_k| \), and the server can compute and decrypt the sum.
Chapter 5

Conclusion

In this thesis, we gave a comprehensive overview of fully homomorphic encryption. We began by studying the mathematical background that underpins many modern cryptographic constructions. Then, we saw how FHE is actually constructed, and we looked at how it can be used to enable the use of machine learning in settings where it was previously restricted. While FHE has come a long way since Gentry’s initial construction, current schemes are still not so efficient, and there is still much work to do to make FHE practical. The good news is that computers are getting more and more powerful, and FHE schemes are getting more and more efficient, so it’s only a matter of time before FHE can become ubiquitous. Indeed, machine learning used to have the same issue, but the development of faster algorithms and more powerful computers have made machine learning widespread; it is not unreasonable to imagine a similar future for FHE.

As a joint concentrator in Math and Computer Science, I found FHE to be a particularly fitting topic with which to conclude my journey through these two fields. It is appealing to the theoretician as the “holy grail” of cryptography whose construction solved a decades-long open problem, but it is also extremely useful to the practitioner who needs accurate analyses under onerous privacy requirements. Additionally, it offers a fascinating example of how advanced abstract mathematics can have significant societal benefit. We began with algebraic number theoretic topics which at first glance don’t appear to have direct applications, such as integer lattices and ideals in cyclotomic fields, and we saw how these ideas can be used (with a few intermediate steps) to improve cancer treatments. It is my hope that this thesis will help others learn about this amazing field.
Bibliography


Jing Ma, Si-Ahmed Naas, Stephan Sigg, and Xixiang Lyu. Privacy-preserving federated learning based on multi-key homomorphic encryption, 2021.


