Homomorphic Encryption via Ring Learning With Errors (RLWE)

May 12th, 2024

1 Introduction

Homomorphic encryption is an important cryptographic primitive that is necessary to support modern cryptographic protocols, such as anonymous voting. Some constructions of such a scheme are complicated and unintuitive. However, recent constructions relying on the Ring Learning With Errors (RLWE) problem provide reasonably concise and palatable constructions by relying on algebraic ring structures and computation therein. [1]

2 Background

Let $\mathcal{K}_\mathcal{S}, \mathcal{K}_\mathcal{P}, \mathcal{M}, \mathcal{C}$ represent the secret key space, public key space, message space, and ciphertext spaces of an encryption scheme, respectively.

We recall from class that a *group* is a mathematical structure on a set G that gives us closure, associativity, identity, and inverses for some operation $(\cdot) : G \times G \to G$.

A *ring* is a group with an additional operation $+$ (addition) that gives us closure, associativity, identity, and inverses for the operation $+$, and also satisfies the distributive property with respect to the original operation $(\cdot)$ (multiplication).

For this project, we will be concerned with rings of the form $R_n = \mathbb{Z}_n/(x^m + 1)$, which are rings of polynomials with coefficients in $\mathbb{Z}_n$ modulo some polynomial $x^m + 1$. The exact structure and meaning of "modulo" in the context of rings is outside the scope of the project, but you can think of this as a ring of polynomials where the coefficients are taken modulo n and the polynomial is taken modulo $x^m + 1$.

For instance, if we take two elements $a = 2x^2 + 3x + 1$ and $b = 4x^2 + 2x + 3$ in $R_5 = \mathbb{Z}_5/(x^3 + 1)$, then the sum of these elements is $a + b = 1x^2 + 0x + 4 = 4x^2 + 4$, and the product of these elements is $a \cdot b = 3x^4 + 4x^3 + 4x^2 + 2x + 2 = 3x^2 + 4x + 2$ (with a special modulo reduction operation that causes them to behave slightly differently than lattices and other structures). [2]

3 Homomorphic Encryption

When we say that an encryption scheme is *homomorphic*, what we really mean is that we are able to perform some binary operation $S_C : (C, C) \to C$ on encrypted ciphertexts that corresponds to some operation $S_M : (M, M) \to M$ on the original messages [1].

An intuitive example could be a scheme where adding two ciphertexts $c_1 + c_2$ results in a ciphertext that is a valid encryption of the *sum* of the original messages m_1, m_2, such that only the sum $m_1 + m_2$

[1] Disclaimer: I was first exposed to this scheme and the RLWE problem in CSCI1515 (Applied Cryptography). However, the presentation was incredibly brief and didn't give any mathematical background on the construction, and we later used a library implementation as a black-box to construct other protocols. The content in the project is completely disjoint from that exposure, including both the mathematical background and the custom implementation. [3]

[2] Understanding this really does require additional background that I don't have time to discuss here. The important thing is that we can easily (and programatically) *add* and *multiply* elements in these rings.

is revealed when the quantity $c_1 + c_2$ is decrypted. Importantly, this cannot reveal *anything about the original ciphertexts themselves* (other than what is discernable from the operation itself).

Definition (Homomorphic Encryption). An encryption scheme $E = (\text{Enc}_E, \text{Dec}_E)$ consisting of encryption and decryption functions is *homomorphic* for an operator $S = (S_M, S_C)$ if for any two messages $m_1, m_2 \in \mathcal{M}$, public key $k_p \in \mathcal{K}_\mathcal{P}$, and secret key $k_s \in \mathcal{K}_\mathcal{S}$, the following is satisfied:

$$\text{Dec}_E\left(k_s, S_C(\text{Enc}_E(k_p, m_1), \text{Enc}_E(k_p, m_2))\right) = S_M(m_1, m_2) \tag{1}$$

An important note is that the operations S_C and S_M are not necessarily (and are generally not) the same. For instance, it is possible that $S_C(c_1, c_2) = c_1 \cdot c_2$ (multiplying the ciphertexts) whereas the effect on the messages is addition: $S_M(m_1, m_2) = m_1 + m_2$ (adding the underlying messages).

The variant that we will discuss later is a *somewhat homomorphic encryption (SHE)* scheme, which allows for a fixed number of operations to be computed on a given ciphertext. These schemes are less expressive than FHE schemes, but are generally easier to implement, and many FHE schemes are actually built on top of SHE schemes by taking additional steps to ensure that the computation error never exceeds some critical value [1].

3.1 Example: Diffie-Hellman

One illustrative example of homomorphic encryption is actually the Diffie-Hellman encryption scheme. If there are two encrypted messages $c_1 = (g^{r_1}, h^{r_1} m_1)$, and $c_2 = (g^{r_2}, h^{r_2} m_2)$, then we see that we get:

$$\begin{aligned} S_C(c_1, c_2) &= c_1 \cdot c_2 && (2)\\ &= (g^{r_1} \cdot g^{r_2}, h^{r_1} m_1 \cdot h^{r_2} m_2) && (3)\\ &= (g^{r_1 + r_2}, h^{r_1 + r_2} m_1 \cdot m_2) && (4) \end{aligned}$$

Note that we're basically just doing pointwise multiplication within the tuple ciphertext. Importantly, since r_1, r_2 are randomly chosen from $\mathbb{Z}_q$, the quantity $r_1 + r_2$ is also completely random. This means that this is the same as an encryption of the quantity $m_1 \cdot m_2$, the product of the original messages.

4 Ring Learning With Errors (RLWE)

The Ring Learning With Errors (RLWE) problem is a variant of the Learning With Errors (LWE) problem that is used to construct homomorphic encryption schemes over rings R [1]. This scheme (the BFV scheme in particular) requires the following parameters and structures [1]:[3]

1. Integers t, q, m to define the rings $R_t = \mathbb{Z}_t/(x^m + 1)$ and $R_q = \mathbb{Z}_q/(x^m + 1)$. For this scheme, the message space is $\mathcal{M} = R_t$ and the ciphertext space is $\mathcal{C} = R_q \times R_q$.

2. An error distribution (or function) χ generating elements $e \in R_q$ that are *small* in some sense. This is used to mask the encrypted values and ensure that the original message can be retrieved with the secret key. This error leaves a lot of room for variability, but for the sake of this project we will assuming that it is a Gaussian distribution with $\mu = 0$ and some standard deviation σ.

[3]Some of the structures have been simplified for this project for clarity and ease of implementation.

3. A large ratio $\Delta = \lfloor \frac{q}{t} \rfloor$, which requires that $q \gg t$. This is necessary to isolate the "message" part of the decrypted ciphertext from the "error" part in the presence of the secret key.

The RLWE assumption is then defined as follows (with the parameters above) [1]:

RWLE Assumption. Consider the ring $R_q = \mathbb{Z}_q/(x^m + 1)$ and an error distribution χ. Then, the following two distributions are indistinguishable:

1. $A = (\mathbf{a} \cdot \mathbf{s} + \mathbf{e}, \mathbf{a})$, where $\mathbf{a} \leftarrow R_q, s \leftarrow R_q, e \leftarrow \chi$.
2. $B = (\mathbf{u}, \mathbf{v})$, where $\mathbf{u}, \mathbf{v} \leftarrow R_q$ are completely random.

The intuition behind this assumption is that the error terms $\mathbf{e}$ are small enough that they can be masked by the secret key $\mathbf{s}$, and the public key $\mathbf{a}$ is random enough that the error terms are indistinguishable from random elements.

Note that the RLWE problem may seem to be a completely new construct from what we have discussed in class, but it turns out that there exists a quantum algorithm that reduces RLWE to the SVP in lattices that we discussed in class [1] [2].

4.1 BFV Operations

4.1.1 Key Generation

The key generation algorithm for the BFV scheme is as follows [1]:

1. Generate a secret key $\mathbf{s} \leftarrow \chi$, an error term $\mathbf{e} \leftarrow \chi$, and a random public element $\mathbf{a} \leftarrow R_q$.
2. Compute the public key as the tuple $pk = (-(\mathbf{a} \cdot \mathbf{s} + \mathbf{e}), \mathbf{a})$, which can be interpreted as a single element in $R_q \times R_q$.

This means that the public and private key spaces are $\mathcal{K}_\mathcal{P} = R_q \times R_q$ and $\mathcal{K}_\mathcal{S} = R_q$, respectively.

4.1.2 Encryption

The encryption scheme is as follows [1]:

1. Given a public key $\mathbf{pk} = (\mathbf{pk}_0, \mathbf{pk}_1)$ and a message $\mathbf{m} \in R_t$, generate random error terms $\mathbf{u}, \mathbf{e}_1, \mathbf{e}_2 \leftarrow \chi$.
2. Compute $\mathbf{c}_0 = pk_0 \cdot \mathbf{u} + \mathbf{e}_1 + m \cdot \Delta$ and $\mathbf{c}_1 = pk_1 \cdot \mathbf{u} + \mathbf{e}_2$.
3. Return the encrypted ciphertext $\mathbf{c} = (\mathbf{c}_0, \mathbf{c}_1) \in R_q \times R_q = \mathcal{C}$.

4.1.3 Decryption

The decryption process for a ciphertext $\mathbf{c}$ and secret key $\mathbf{s}$ is then as follows [1]:

1. Compute $\mathbf{d} = \mathbf{c}_0 + \mathbf{c}_1 \cdot \mathbf{s}$.
2. Round all of the values in $\mathbf{d}$ to the nearest multiple of Δ, and return the result as the decrypted message $\mathbf{m}' \in R_t$.

From this construction itself it is not immediately clear why this decryption process works, but we can see:

Theorem (BFV Decryption Correctness). The BFV decryption scheme is correct on ciphertexts encrypted with the BFV encryption scheme.

Proof. We see that before the final rounding process, the decrypted values are:

$$\begin{aligned}
\mathbf{d} &= \mathbf{c}_0 + \mathbf{c}_1 \cdot \mathbf{s} && (5)\\
&= pk_0 \cdot \mathbf{u} + \mathbf{e}_1 + \mathbf{m} \cdot \Delta + (pk_1 \cdot \mathbf{u} + \mathbf{e}_2) \cdot \mathbf{s} && (6)\\
&= (-(\mathbf{a} \cdot \mathbf{s} + \mathbf{e})) \cdot \mathbf{u} + \mathbf{e}_1 + \mathbf{m} \cdot \Delta + (\mathbf{a} \cdot \mathbf{u} + \mathbf{e}_2) \cdot \mathbf{s} && (7)\\
&= -\mathbf{a} \cdot \mathbf{s} \cdot \mathbf{u} - \mathbf{e} \cdot \mathbf{u} + \mathbf{e}_1 + \mathbf{m} \cdot \Delta + \mathbf{s} \cdot \mathbf{a} \cdot \mathbf{u} + \mathbf{e}_2 \cdot \mathbf{s} && (8)\\
&= \mathbf{m} \cdot \Delta - \mathbf{e} \cdot \mathbf{u} + \mathbf{e}_1 + \mathbf{e}_2 \cdot \mathbf{s} && (9)
\end{aligned}$$

So, in order for the rounded values of $\mathbf{d}$ to be equal to $\mathbf{m}$, we need to show that the error terms $\mathbf{e} \cdot \mathbf{u} + \mathbf{e}_1 + \mathbf{e}_2 \cdot \mathbf{s}$ are small enough that the rounding process doesn't change the value of $\mathbf{m}$.

Roughly speaking, since we know that χ is a Gaussian distribution with standard deviation σ, we can say that the sum of the error terms on the right have a collective distribution with standard deviation at most $\sqrt{3}\sigma$, since the sum of independent Gaussian variables has a standard deviation that is the square root of the sum of the squares of the individual standard deviations.

Therefore, so long as we choose σ such that the probability that *any of the polynomial coefficients of the error terms* exceed $\frac{\Delta}{2}$ is negligible, we can be confident that the rounding process will not change the value of $\mathbf{m}$.

Therefore, as long as such a σ is chosen, we conclude that the decryption protocol is correct. □

The security of the scheme relies on the opposite constraint on σ: it must be large enough that the error terms are indistinguishable from random noise, but small enough that the rounding process doesn't change the value of the message. One thing that helps with this is that we are free to choose t, q to be as large as we want (generally), so we can vary Δ to make this tradeoff feasible.

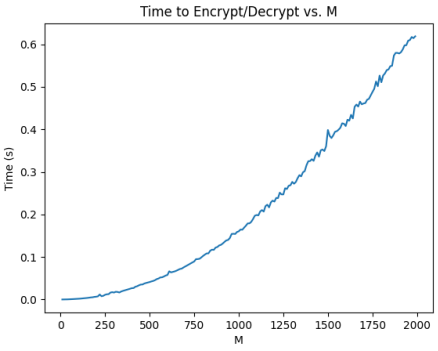
4.1.4 Homomorphic Addition

The BFV protocol also allows us to perform homomorphic addition on ciphertexts. Given two ciphertexts $\mathbf{c}_0 = (\mathbf{c}_{0,0}, \mathbf{c}_{0,1})$ and $\mathbf{c}_1 = (\mathbf{c}_{1,0}, \mathbf{c}_{1,1})$, we can compute the sum of the ciphertexts as follows [1]:

$$S_C(\mathbf{c}_0, \mathbf{c}_1) = (\mathbf{c}_{0,0} + \mathbf{c}_{1,0}, \mathbf{c}_{0,1} + \mathbf{c}_{1,1}) \tag{10}$$

Such that the function we compute on the message space of the ciphertexts is basic addition in the ring R_t: $S_M(m_1, m_2) = m_1 + m_2$. Using a similar argument to the decryption correctness proof, we can see that decryption of added ciphertexts will decrypt to $\mathbf{m} \cdot \Delta$ plus some error term, which

$$\begin{aligned}
\mathbf{d}' &= \mathbf{c}_0 + \mathbf{c}_1 \cdot \mathbf{s} && (11)\\
&= (\mathbf{c}_{0,0} + \mathbf{c}_{1,0}) + (\mathbf{c}_{0,1} + \mathbf{c}_{1,1}) \cdot \mathbf{s} && (12)\\
&= \dots && (13)\\
&= (\mathbf{m}_1 + \mathbf{m}_2) \cdot \Delta + (-\mathbf{e}_0 \cdot \mathbf{u}_0 + \mathbf{e}_{01} + \mathbf{e}_{02} \cdot \mathbf{s}) + (-\mathbf{e}_1 \cdot \mathbf{u}_1 + \mathbf{e}_{11} + \mathbf{e}_{12} \cdot \mathbf{s}) && (14)
\end{aligned}$$




```python
bfv = BFV(args.sigma, args.q, len(args.m), args.t)

# Encode the message into R_T
m = bfv.R_T([ord(c) for c in args.m])

# Generate the keypair
sk, pk = bfv.generate_keypair()

# Encrypt the message
c = bfv.encrypt(pk, m)

# Decrypt the message
m_prime = bfv.decrypt(sk, c)

# Decode the message
m_prime_str = "".join([chr(c) for c in m_prime.c])

print("Encoded Message:", m)
print("Encrypted Message:", c)
print("Decrypted Message:", m_prime)
print("Decoded Message:", m_prime_str)
```

Figure 1: Left: Encryption time as a function of message length m. Right: Code sample of using my custom BFV class for encryption and decryption.

Note that for successive addition operations, the error term will *increase linearly*, meaning that the error term will eventually exceed the critical value $\frac{\Delta}{2}$ and the decryption will fail (which is why this scheme is called *somewhat* homomorphic).

5 Implementation

In the code associated with this report, I implemented my own BVF encryption class called `BFV`, which allows for encryption, decryption, and homomorphic addition of ciphertexts. Figure 1 shows a code sample of how to use this class to encrypt and decrypt messages, and the file when run exposes a simple CLI to encrypt and decrypt ASCII messages.

Everything is implemented from scratch (including custom ring classes exposing convenient APIs for adding and multiplying elements in R_n for any n), and the only external library you need is `matplotlib`, but only to generate the figures present in this report (not for the actual encryption and decryption).

5.1 Performance

Since all of the operations are at most quadratic with regards to the message length m (except for multiplication of elements in R_q, which in my implementation is $O(m^2)$), we expect that the encryption and decryption times will be at most quadratic with respect to the message length. Results of some experiments can be seen in Figure 1, which do show the approximately quadratic runtime. I should also note that the entire process is really quite slow, but this is largely an artifact of using Python as opposed to a compiled language.

References

[1] Junfeng Fan and Frederik Vercauteren. *Somewhat Practical Fully Homomorphic Encryption*. Mar. 2012. URL: `https://eprint.iacr.org/2012/144.pdf`.

[2] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. "On Ideal Lattices and Learning with Errors over Rings". In: *Advances in Cryptology – EUROCRYPT 2010*. Ed. by Henri Gilbert. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 1–23. ISBN: 978-3-642-13190-5.

[3] Peihan Mao. *CSCI 1515: Applied Cryptography*. Apr. 2024. URL: `https://cs.brown.edu/courses/csci1515/`.