TFHE

October 2, 2024

1 Introduction

TFHE is a Fully Homomorphic Encryption (FHE) scheme. It is an encryption scheme that allows you to perform computations over encrypted data.

1.1 Integer ring modulo $\mathbb{Z}_q$

An integer ring modulo q (also called integers modulo a prime number), denoted by $\mathbb{Z}_q$, is a mathematical structure known as a cyclic group or finite field in abstract algebra. Specifically, it is the set of residue classes of integers modulo q. This means it's the set of all possible remainders when integers are divided by q. For example, if q equals 2, then $\mathbb{Z}_q$ (which is $\mathbb{Z}_2$ in this case) includes only 0 and 1 because any integer either leaves a remainder of 0 or 1 when divided by 2. In the field of cryptography, we typically work with finite fields. The exciting thing about working in a finite field is that the polynomial coefficients "wrap around" when multiplied. So, regardless of how much polynomial arithmetic we perform, the polynomial coefficients can still be bounded in a fixed range, which is very attractive from an implementation perspective. If q equals 17, then $\mathbb{Z}_q$ (which is $\mathbb{Z}_{17}$ in this case) includes only 0 to 16 because any integer leaves a remainder of 0 to 16 when divided by 17. Hence the vector multiplication for vectors $g = [1, 2, 3, 4]$, and $h = [5, 6, 7, 8]$ with modulo a prime number $q = 17$ is $[5, 16, 0, 9, 10, 1, 15]$

1.2 Congruence

The symbol $\equiv$ denotes congruence and is different from equality. While equality means that items are the same, congruence means items have the same remainder when divided by mod. In a ring $\mathbb{Z}_{7681}$ and $n = 4$, the 4-th root of unity, which satisfy the condition $\omega^4 \equiv 1 \bmod 7681$ are $3383, 4298, 7680$.

2 TFHE Ciphertexts

TFHE mainly uses three ciphertext types: LWE, RLWE, and RGSW. All of them have different properties which will be useful in the homomorphic operations

3 GLWE

General LWE, or GLWE includes both of LWE, RLWE

3.1 LWE

LWE encryption supports encrypting small-bit-width integers by placing those bits in the most significant bits of a machine word—for simplicity, say it's a 4-bit integer in the top-most bits of a

32-bit integer with all the other bits initialized to zero, and call that whole encoded plaintext. The secret key is a random binary vector of some fixed length chosen to achieve a specific security. Then, sample a random length vector of 32-bit integers to encrypt, take a dot product with the secret key, add the message, and add some random noise. The encrypted value is both the random samples chosen and the noise-masked result. LWE decryption then reverses this process: re-compute the dot product and subtract it from the output of the encryption. But at the end, you must apply a rounding step to remove the noise added to the ciphertext during encryption. Because the message is in the highest-order bits of the message (say, bits 28-31) and because the noise added was not very large, rounding to the nearest multiple removes it.

3.2 RLWE

In RLWE, the scalar multiplications and additions from LWE are upgraded to polynomial multiplications and additions. We pick a polynomial degree as the maximum degree (say 1024), the coefficients are always integers modulo some chosen modulus q, and finally, we pick a polynomial, usually $x^n + 1$, and represent the result of every operation as a remainder when divided by that polynomial. One or more small integer messages are encoded into a polynomial to encrypt. The secret key is a list of random polynomials with binary coefficients, and the samples are random polynomials with uniformly random mod q coefficients. Then, you take a dot product, add the message, and add a similar "noise polynomial" to mask the result. The main advantage of using RLWE over LWE is that you can pack many messages into a single polynomial and the homomorphic operations you apply to all the messages.

3.3 LWE and RLWE from GLWE

When we instantiate GLWE with $k = n$ and $N = 1$, we get LWE. Observe that R_q is actually Z_q when $N = 1$. We use small letters for (modular) integers (i.e., b, m, e ...).

$$b = \sum_{i=0}^{k-1} a_i.s_i + \triangle m + e \in \mathbb{Z}_q$$

When we instantiate GLWE with $k = 1$, we get RLWE. Here, we use capital letters for polynomials.

$$B = A.S + \triangle M + E \in R_q$$

3.4 Secret key

To generate any ciphertext, we first need a secret key. With GLWE ciphertexts, the secret key is a list of random polynomials from R:

$$\overrightarrow{S} = (S_0, S_1, ...S_{k-1}) \in R^k$$

The coefficients of the elements can be sampled from a uniform binary distribution, a uniform ternary distribution, a Gaussian distribution, or a uniform distribution. Please note that we can find parameters to achieve the desired security level for these secret keys.

3.4.1 Example

Let's choose N (degree or dimension) = 4 and k =2. Let's sample the secret key with a uniform binary distribution of a degree N — 1 polynomial. In this example, the secret keys are [0,1,1,0] and [1,0,1,1].

$$\overrightarrow{S} = ([0,1,1,0],[1,0,1,1]) \in R^2$$
$$\overrightarrow{S} = (0 + x + x^2 + 0x^3, 1 + 0x + x^2 + x^3) \in R^2$$
$$\overrightarrow{S} = (x + x^2, 1 + x^2 + x^3) \in R^2$$

3.4.2 Example TFHEpp lvl1param

For TFHEpp lvl1param, the value for $N = 1024$, $k = 1$. The key value maximum is 1, and the minimum is -1. For example:

$$\overrightarrow{S} = ([0,1,0,-1,-1.....1]) \in R^1$$
$$\overrightarrow{S} = (x - x^3 - x^5 - x^6 .... + x^{1023}) \in R$$

3.5 Message

The message is a polynomial of degree smaller than N with coefficients whose maximum value depends on the value p.

$$M \in R^p$$

3.5.1 Example

Let's choose N (degree) = 4 and p (plain modulus) = 4. The coefficient values of the message are stored in 2 bits (p=4 = 2^2). The possible coefficient value in binary format is 11, 10, 00, and 01. The possible coefficient value in a signed integer is -2, -1, 0 and 1. The possible coefficient value in an unsigned integer is 3, 2, 0 and 1. Let Message be [-2 1, 0 -1] in this example.

3.5.2 TFHEpp Example

In the TFHEpp library, each message is binary (i.e., 1 or 0 only). For 16-bit integers, we will have 16 messages, one for each bit.

3.6 Mask

To encrypt the message, we must sample a uniformly random mask with coefficients whose maximum value depends on q (modulus).

$$\overrightarrow{A} = (A_0, A_1, ...A_{k-1}) \in R_q^k$$

3.6.1 GLWE Example

Let's choose N (degree) = 4, k =2 and q = 64 (modulus). The coefficient values of the mask are stored in 6 bits (q=64 = 2^6). The possible coefficient value in binary format is 111111, 111110 ..., 100000, 000000, 000001, ... 011111. The possible coefficient value in a signed integer is -31, -30, ...-1, 0, 1, .. 32. The possible coefficient value in an unsigned integer is 64, 63, ... 33, 0, 1, ..32. Let Mask be [17, -2, -24, 9], [-14, 0, -1, 21] in this example.

$$\overrightarrow{A} = ([17, -2, -24, 9], [-14, 0, -1, 21]) \in R_{64}^2$$
$$\overrightarrow{A} = (17 - 2x - 24x^2 + 9x^3, -14 - x^2 + 21x^3) \in R_{64}^2$$

3.6.2 RLWE Example

Let's choose N (degree) = 4, k =1 (for RLWE) and q = 64 (modulus). The coefficient values of the mask are stored in 6 bits (q=64 = 2^6). The possible coefficient value in binary format is 111111, 111110 ..., 100000, 000000, 000001, ... 011111. The possible coefficient value in a signed integer is -31, -30, ...-1, 0, 1, .. 32. The possible coefficient value in an unsigned integer is 64, 63, ... 33, 0, 1, ..32. Let Mask be [17, -2, -24, 9] in this example.

$$\overrightarrow{A} = ([17, -2, -24, 9]) \in R_{64}^1$$
$$\overrightarrow{A} = (17 - 2x - 24x^2 + 9x^3) \in R_{64}$$

3.6.3 TFHEpp code

In file tlwe.hpp function tlweSymEncrypt(), it adds mask using std::uniform_int_distribution() C++ utility.

3.7 Error

We must add a discrete Gaussian Error (small coefficients) to encrypt the message. $\chi_{\mu,\sigma}$ is a Gaussian probability distribution with mean μ and standard deviation σ

$$E \in R_q$$

3.7.1 Example

Let's add [-1,1,0,1] error.

$$E = ([-1, 1, 0, 1]) \in R_q$$
$$E = (x + x^2, 1 + x^3) \in R_q$$

3.7.2 TFHEpp code

In TFHEpp, the noise is added for each message using ModularGaussian() in the utils hpp file. i.e. Noise standard deviation for lvl1param is 2^{-25}. i.e. if the △ message is 536870912, add some noise and make it 536870870.

3.8 Body

The body of an encrypted message is:

$$B = \sum_{i=0}^{k-1} A_i.S_i + \triangle M + E \in R_q$$

where △ = q/p.

3.8.1 Example

Let's continue with previous examples for N (degree) = 4, p (plain modulus) = 4, k = 2, q = 64 (modulo) and △ = q/p = 16.

$$\begin{aligned} B &= \sum_{i=0}^{k-1} A_i.S_i + \triangle M + E \in R_q \\ &= \sum_{i=0}^{2} A_i.S_i + 16M + E \in R_q \\ &= A_0.S_0 + A_1.S_1 + 16M + E \in R_q \end{aligned}$$

When we compute R_q, we do polynomial operations modulo $x^N + 1$ and modulo q. To reduce modulo $x^N + 1$, you can observe that:

$x^N = x^N \equiv -1 \bmod x^N + 1$

So

$$\begin{aligned} A_0.S_0 &= (17 - 2x - 24x^2 + 9x^3).(x + x^2) \\ &= 17x + (17-2)x^2 + (-2-24)x^3 + (-24+9)x^4 + 9x^5 \\ &= 17x + 15x^2 - 26x^3 - 15x^4 + 9x^5 \\ &= 17x + 15x^2 - 26x^3 + (-15+9x)x^4 \\ &= 17x + 15x^2 - 26x^3 + (-15+9x)(-1) \\ &= 17x + 15x^2 - 26x^3 + 15 - 9x \\ &= 15 + 8x + 15x^2 - 26x^3 \in R_q \end{aligned}$$

In the same way:

$$\begin{aligned} A_1.S_1 &= -13 - 20x + 28x + 7x3 \in R_q \\ \triangle M &= -32 + 16x - 16x^3 \end{aligned}$$

Then:

$$\begin{aligned} B &= A_0.S_0 + A_1.S_1 + \triangle M + E \in R_q \\ &= -31 + 5x - 21x^2 + 30x^3 \in R_q \end{aligned}$$

3.9 Encryption

A GLWE ciphertext encrypting the message M under the secret key $\overrightarrow{S}$ is a tuple:

$$GLWE_{\overrightarrow{S},\sigma}(\triangle M) = (A_0, A_1, ...A_{k-1}, B) \subseteq R_q^{k+1}$$

3.9.1 Example

Let's continue with previous examples for N (degree) = 4, p (plain modulus) = 4, k = 2, q = 64 (modulo) and △ = q/p = 16.

$$\begin{aligned} GLWE_{\overrightarrow{S},\sigma}(\triangle M) &= (A_0, A_1, B) \subseteq R_{64}^3 \\ &= (17 - 2x - 24x^2 + 9x^3, -14 - x^2 + 21x^3, -31 + 5x - 21x^2 + 30x^3) \subseteq R_{64}^3 \end{aligned}$$

3.9.2 TFHEpp Example

In the TFHEpp library, each message is binary (i.e., 1 or 0). The following formula transforms $\triangle M$ the message in the function bootsSymEncrypt() tlwe hpp file.

$$message = message?\mu : -\mu;$$

For lvl1param, mu = 1 « 29 = 536870912 (0x2000000). So for message = 1, it is transformed to 536870912 else it is transformed to -536870912 (0xE0000000).

Here are the complete steps for encryption:

1. If the message is 1.
2. Transform the message to 536870912 (0x2000000) by shifting the bits of number 1 to the left 29 places
3. Add a small noise to the transformed message, i.e. make it make it 536870870 (0x1FFFFFF6)
4. Now add the mask to the message. message + $\sum_{i=0}^{1023} A_i.S_i$, where A_i is mask coefficients, and S_i is secret key coefficients. The masked message will be 1128353980 (0x434150b). It will be the last element of the cypher text. The first 1024 elements of the cypher text are mask coefficients.

3.10 Decryption

We can decrypt the ciphertext using the following equation:

$$B - \sum_{i=0}^{k-1} A_i.S_i = \triangle M + E$$
$$(\triangle M + E)/\triangle = M$$

Observe that the message M is in the MSB part (thanks to the multiplication by $\triangle$) while E is in the LSB part. If $|E| < \triangle/2$ (so if every coefficient of $|e_i| < \triangle/2$), then the second step of the decryption M returns as expected.

3.10.1 Example

Let's continue with previous examples.

$$\begin{aligned}
B &= -31 + 5x - 21x^2 + 30x^3 \\
A_0.S_0 &= 15 + 8x + 15x^2 - 26x^3 \\
A_1.S_1 &= -13 - 20x + 28x + 7x3 \\
\triangle M + E &= B - \sum_{i=0}^{1} A_i.S_i \\
\triangle M + E &= B - A_0.S_0 - A_1.S_1 \\
\triangle M + E &= -29 + 17x - 15x^2 \\
M &= (-29 + 17x - 15x^3)/\triangle \\
M &= (-29 + 17x - 15x^3)/16 \\
&= -2 + x - x^2
\end{aligned}$$

3.10.2 TFHEpp code

tlweSymPhase() function in tlwe.hpp removes the mask from the encrypted message. tlweSymDecrypt() checks if MSB bit is set or not. If the MSB bit is not set, it is 1, or else it is 0.

Here are the complete steps for decryption:

1. The masked message is 1128353980 (0x434150b) in our previous example
2. Now remove the mask from the masked message. message - $\sum_{i=0}^{1023} A_i.S_i$, where A_i is mask coefficients, and S_i is secret key coefficients. The message will be 536870870 (0x1FFFFFF6). tlweSymPhase()
3. Function tlweSymDecrypt() checks if the MSB bit is set. For 0x1FFFFFF6 the MSB bit is not set. From it, we deduce that the message is 1.

4 GLev

GLev is an intermediate ciphertext type between GLWE and GGSW ciphertexts, which can be very useful for better understanding GGSW ciphertexts. GLev can be seen as a generalization of the well-known Powers of two encryptions used in BGV. A GLev ciphertext contains redundancy: a list of GLWE ciphertexts encrypting the same message with different and exact scaling factors $\triangle$. Two parameters are necessary to define these special $\triangle$'s: a base β and many levels l $\in \mathbb{Z}$. β and q are the power of 2.

$$\triangle^i = \frac{q}{\beta^i}$$

The secret key is the same as for GLWE ciphertexts. To decrypt, it is sufficient to decrypt one of the GLWE ciphertexts with the corresponding scaling factor. The set of GLev encryptions of the same message, under the secret key $\overrightarrow{S}$, with Gaussian noise with standard deviation, with base β and level l, will be noted $GLev_{S,\sigma}^{\beta,l}(M)$

$$(GLWE_{\overrightarrow{S},\sigma}(\frac{q}{\beta^1}M) \times ... \times GLWE_{\overrightarrow{S},\sigma}(\frac{q}{\beta^l}M)) = GLev_{S,\sigma}^{\beta,l}(M) \subseteq R_q^{lk+1}$$

4.1 Lev and RLev from GLev

In the same way that we saw that GLWE was a generalization for both LWE and RLWE, we can observe that GLev can be specialized into Lev and RLev by following the same rules. When we instantiate GLev with $k = n$ and $N = 1$, we get Lev. When we instantiate GLev with $k = 1$, we get RLev.