Ring-Learning With Errors & HElib

Homomorphic Encryption

- Additive Homomorphic Encryption

$$\mathrm{Dec}_{sk}(\mathrm{Enc}_{pk}(a) \oplus \mathrm{Enc}_{pk}(b)) = a + b$$

- Multiplicative Homomorphic Encryption

$$\mathrm{Dec}_{sk}(\mathrm{Enc}_{pk}(a) \otimes \mathrm{Enc}_{pk}(b)) = a \times b$$

- Fully Homomorphic Encryption

Learning With Errors

- LWE-Assumption

$$a \leftarrow \mathbb{Z}_q^n \;\; s \leftarrow \mathbb{Z}_q^n \;\; e \leftarrow \mathbb{Z}_q^n \qquad r_1, r_2 \leftarrow \mathbb{Z}_q^n$$

$$(a, \langle a, s \rangle + e) \approx (r_1, r_2)$$

- Ring-LWE is just using a polynomial ring

$$\mathbb{Z}_q^n \rightarrow \mathbb{Z}_q[x]/\Phi_m(x)$$

$\Phi_m(x)$: cyclotomic polynomial

Ring-LWE

- Parameter settings

  $m \in \mathbb{Z}^+$ defines $\Phi_m(x)$

  p: a prime number defines $\mathbb{Z}_p[x]$

  σ defines a discrete Gaussian dist. χ

- Message Space

  $R_p := \mathbb{Z}_p[x]/\Phi_m(x)$

  e.g. $m = 8, p = 11; R_{11} := \mathbb{Z}_{11}[x]/(x^4 + 1)$

- Ciphertext Space:

  $R_q; q \gg p$

Basic Scheme Operations

- KeyGeneration

$$s \leftarrow \chi \quad a_0 \leftarrow R_q; e \leftarrow \chi$$

$$a_1 := sa_0 + ep$$

$$\text{secret key, sk} = s$$

$$\text{public key, pk} = (a_0, a_1)$$

Basic Scheme Operations

- Encryption $m \in R_p \ \mathrm{pk} := (a_0, a_1)$

$$u, f, g \leftarrow \chi$$

$$\mathrm{ctx} := (c_0 = a_1 u + gp + m, c_1 = a_0 u + fp)$$

- Decryption $\mathrm{ctx} = (c_0, c_1, \cdots c_k) \ \mathrm{sk} = s$

$$m = \sum_{i=0}^{k} c_i s^i \mod p$$

Homomorphic Operations

- Addition $\mathrm{ctx}_1 = (c_0, c_1, \cdots, c_k), \mathrm{ctx}_2 = (c_0', c_1', \cdots, c_k')$

$$\mathrm{ADD} = (c_0 + c_0', c_1 + c_1', \cdots, c_k + c_k')$$

- Multiplication $\mathrm{ctx}_1 = (c_0, c_1), \mathrm{ctx}_2 = (c_0', c_1')$

$$\mathrm{MUL} = (c_0 c_0', c_0 c_1' + c_1 c_0', c_1 c_1')$$

The size of ciphertext increases!

HElib

- Purely written in C++
- Implements the BGV-type encryption scheme
- Supports other optimazations such as: reLinearazation, bootstapping, packing
- Supports multithread this March

BGV-type Scheme

- The BGV-type scheme is a *leveled* homomorphic encryption scheme
- We define a parameter *L, called levels*
- *and define a sequence* $q_1 > q_2 > \cdots > q_L$
- *The ciphertext-space changes level by level*

$$R_{q_i} \Rightarrow R_{q_{i+1}}$$

- *The noise inside ciphetexts can reduce by* $\frac{q_{i+1}}{q_i}$
- *This operation called Modulo-switch*

Sample codes: Setup

```
FHEcontext context(m, p, r);
buildModChain(context, L);
FHESecKey sk(context);
sk.GenSecKey(64);
addSome1DMatrices(sk);
const FHEPubKey &pk = sk;
```

R_{p^r}

levels

Add extra information for reLinearization

Sample codes: Enc/Dec/Mult

```
Ctxt ctxt(pk);
ZZX plain = to_ZZX(10);
pk.Encrypt(ctxt, plain); // ctxt = Enc(10)
ctxt.mulByConstant(to_ZZX(1));
ctxt.addConstant(to_ZZX(20)); //ctxt = Enc(20)
//using reLinearaztion
ctxt.multiplyBy(ctxt); //ctxt = Enc(400)
//not using reLinearaztion
ctxt *= ctxt; // ctxt = Enc(160000)
sk.Decrypt(ctxt, plain); // plain = 160000 mod p^r
```

Different kinds of packing

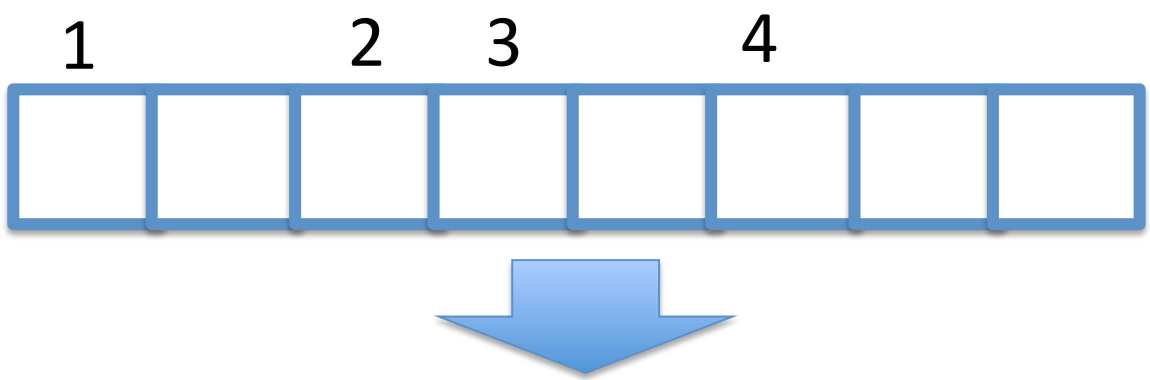
- Pack into coefficients
- Pack into subfields (so-called CRT-based packing)

I. Pack into coefficients

- Example Message Space $p = 13, m = 16$

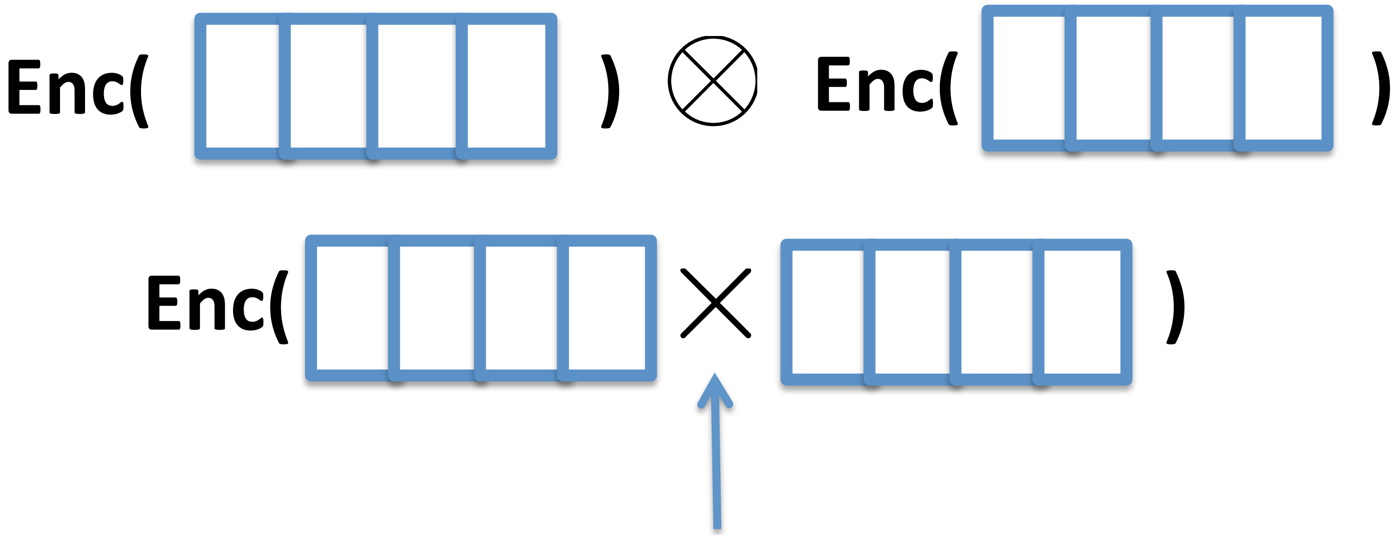
$$R_{13} := \mathbb{Z}_{13}[x]/(x^8 + 1)$$

- image that 8 boxes and each can put in a less than 13 positive integer.



$$1 + 2x^2 + 3x^3 + 4x^5$$

I. Pack into coefficients



Just the multiplication between polynomials!

- So we need to design how to encode our data into a useful polynomial form

Example: Encoding for scalar product

- Given two vectors of integers

$$\boldsymbol{v} = [v_0, v_1, v_2] \ \boldsymbol{u} = [u_0, u_1, u_2]$$

- If we make two polynomials like

$$V(x) = v_0 + v_1 x + v_2 x^2 \quad U(x) = u_0 + u_1 x + u_2 x^2$$

- The mult. of $V(x)U(x)$ wouldn't give scalar product
- If we change a little bit $\tilde{U}(x) = u_2 + u_1 x + u_0 x^2$

  the 3-th term of $V(x)\tilde{U}(x)$ is the scalar product

between u and v.

```
long v[4] = {1, 2, 3, 4};
long u[4] = {1, 2, 3, 4};
ZZX V, U;
V.setLength(4); U.setLength(4);
for (int i = 0; i < 4; i++) {
    setCoeff(V, i, v[i]);
    setCoeff(U, 3 - i, u[i]);
}
//V = 1 + 2x + 3x^2 + 4x^3
//U = 4 + 3x + 2x^2 + x^3
Ctxt encV(pk), encU(pk);
pk.Encrypt(encV, V);
pk.Encrypt(encU, U);
//encV *= encU
encV.multiplyBy(encU);
ZZX result;
sk.Decrypt(result, encV);
std::cout << result[3]; // 30 \mod p^r
```

Sample codes: Pack into Coeff.

II. Pack into subfields

- Not put into each coefficients directly
- Utilize the Chinese Reminder Theorem

Let's consider the CRT in the integer field

A number p can be factorized into *prime factors*

$$p = \prod_{i=1}^{\ell} p_i$$

We have the isomorphism from CRT

$$\mathbb{Z}_p \cong \mathbb{Z}_{p_1} \bigotimes \cdots \bigotimes \mathbb{Z}_{p_\ell}$$

$\text{where } \bigotimes \text{ is Cartesian product}$

II. Pack into subfields

- Polynomial-CRT

The cyclotomic polynomial can be factorized into distinct ℓ *irreducible* polynomials

$$\Phi_m(x) = \prod_{i=1}^{\ell} F_i(x) \mod p$$

For each irreducible polynomial $d := \deg(F_i(x)) = \dfrac{\phi(m)}{\ell}$

$$\begin{aligned}
&\mathbb{Z}_p[x]/\Phi_m(x) \\
\cong &\mathbb{Z}_p[x]/F_1(x) \otimes \cdots \otimes \mathbb{Z}_p[x]/F_\ell(x) \\
\cong &\underbrace{\mathbb{F}_{p^d} \otimes \cdots \otimes \mathbb{F}_{p^d}}_{\ell-\text{copies}}
\end{aligned}$$

called *slots*

Example

- m = 8, p = 17

$$\Phi_8(x) = x^4 + 1 = (x-2)(x-2^3)(x-2^5)(x-2^7) \mod 17$$

$$d := \deg(F_i(x)) = 1$$

- So each slot can hold d = 1 number mod 17.

$$[8, 5, 16, 9] \Longleftrightarrow 1 + x + 7x^2 + 12x^3$$

$$[5, 5, 3, 7] \Longleftrightarrow 5 + 14x + 4x^2 + 3x^3$$

$$[13, 10, 19, 16] \mod 17 \Longleftrightarrow 6 + 15x + 11x^2 + 15x^3$$

$$[40, 25, 48, 63] \mod 17 \Longleftrightarrow 10 + x + 12x^3$$

Operations supported by HElib

- Component-wise (entry-wise) addition/mult.
- Rotation on each slots

➢ Shift; padding with 0s

➢ Running sums, total sums $[x_1, x_2, x_3, \cdots, x_n]$

$$[x_1, \sum_{i=1}^{2} x_i, \sum_{i=1}^{3} x_i, \cdots, \sum_{i=1}^{n} x_i]$$

$$[\sum_{i=1}^{n} x_i, \sum_{i=1}^{n} x_i, \sum_{i=1}^{n} x_i, \cdots, \sum_{i=1}^{n} x_i]$$

Codes for CRT-packing

Actually we can pack a vector of polynomials

```
std::vector<long> u = {1, 2, 3, 4};
std::vector<long> v = {4, 3, 2, 1};
ZZX F = context.alMod.GetFactorsOverZZ()[0];
EncryptedArray ea(context, F);
Ctxt encV(pk), encU(pk);
ZZX V, U;
ea.encode(V, v); ea.encode(U, u);
// V = ??, U = ??
pk.Encrypt(encV, V);
pk.Encrypt(encU, U);
/*
ea.encrypt(encV, pk, v); ea.encrypt(encU, pk, v);
*/
encV *= encU
ZZX result;
sk.Decrypt(result, encV); // result = ??
std::vector<long> decoded;
ea.decode(result, decoded); // decoded = [4, 6, 6, 4] //mod p^r
/*
ea.decrypt(decoded, sk, encV);
*/
```

Sample codes for other Helib routines

```
std::vector<long> u = {1, 2, 3, 4};
ZZX F = context.alMod.GetFactorsOverZZ()[0];
EncryptedArray ea(context, F);
Ctxt encU(pk);
ea.encrypt(encU, pk, v); //encU = Enc([1, 2, 3, 4])
ea.rotate(encU, 1); // encU = Enc([4, 1, 2, 3])
ea.rotate(encU, -2); // encU = Enc([2, 3, 4, 1])
ea.shift(encU, 1); // encU = Enc([0, 2, 3, 4])
runningSums(ea, encU) // encU = Enc([0, 2, 5, 9])
totalsSums(ea, encU) // encU = Enc([16, 16, 16, 16])
```

Rules of thumb

- 32-bits platform, open *–DNOT_HALF_PRIME* flag before building the Helib
- If p^r != 2, to add extra levels $2\lceil \frac{3r\log_2(p)}{\mathrm{FHE\_p2Size}} \rceil + 1$

To install HElib

- Fistly install NTL(Number Theory Library)

  http://www.shoup.net/ntl/
- Install GMP, m3 library.
- Install Helib

https://github.com/shaih/HElib

- To use multithread, need g++4.9(seems not works on Mac OS for now)

Reference

- The design document inside the HElib repo.
- *Fully Homomorphic SIMD operations. N.P.Smart, et.al*
- *Can homomorphic be practical? K. Lauter et. al*
- *Secure Pattern Matching using Somewhat homomorphic encryption. M. Yasusa et. al*
- *Fully Homomorphic Encryption without Boostrapping. Z. Brakerski et. al*
- *Fully Homomorphic Encryption with Polylog Overhead. C. Gentry et al.*