Ring-Learning With Errors & HElib

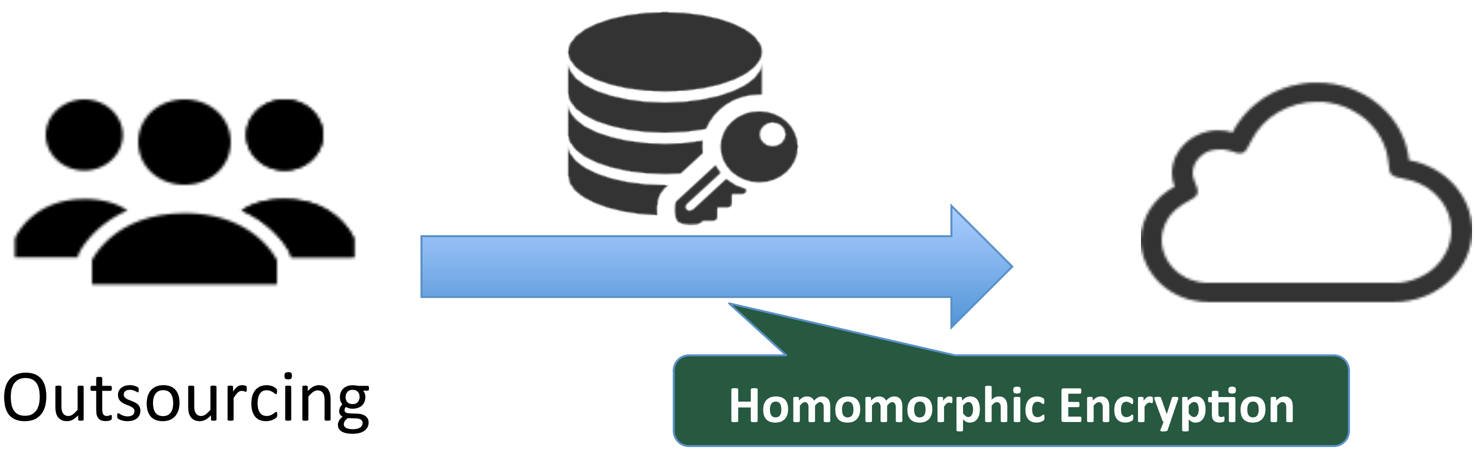
Wenjie Lu(陸 文杰）
University of Tsukuba
riku@mdl.cs.tsukuba.ac.jp

Outline

1. Homomorphic Encryption & Fully Homomorphic Encryption (FHE)
2. Learning With Errors (LWE) & Ring-LWE
3. Ring-LWE based FHE operations: *KeyGen* etc.
4. HElib
   1. BGV's leveled FHE scheme
   2. Optimizations e.g. modulo-switch in HElib
   3. Example codes
5. Two kind of packing methods & example codes
6. Some other routines in HElib & example codes
7. An application on epidemiology study
8. Misc. includes noise-estimation & parameters decision in HElib

Privacy Preserving Computing

Secure Multi-parties Computing



Secure Outsourcing

Homomorphic Encryption

- Additive Hom. Encryption, e.g. Paillier

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

- Multiplicative Hom. Encryption, e.g. ElGamal

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

- Fully Homomorphic Encryption

  Satisfies both additive and multiplicative

Fully homomorphic encryption

- Breakthrough by Gentry in 2009
- Main idea:
  1. First build a somewhat homomorphic encryption
  2. Then apply *bootstrapping* to achieve FHE
- Common facts of the current FHE schemes
  1. Noise grows with operations
  2. Multiplication yields the most noise
  3. Decryption will fail with too large noise

[A fully homomorphic encryption scheme Gentry 2009]

Ring-learning With Errors based FHE

- RLWE schemes: The most efficient schemes for now.
- Different kinds of RLWE based FHE

1. BGV's leveled scheme (implemented by HElib)
2. Brakerski, Scale-invariant scheme
3. ……

[(Leveled) fully homomorphic encryption without bootstrapping] Brakerski, Gentry, Vaikuntanathan 2012

[**Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP**] Brakerski,2012

Notations

- $\mathbb{Z}_q$: Integer modulo q
- $\mathbb{Z}_q^n$: Vectors consists of n integer modulo q
- $\boldsymbol{a} \leftarrow \mathbb{Z}_q^n$: uniformly sample $\boldsymbol{a}$ from $\mathbb{Z}_q^n$
- $\mathbb{Z}_q[x] := \{\sum_{i=0}^{*} \alpha_i x^i | \alpha_i \in \mathbb{Z}_q\}$: set of polynomials
- F(x) a polynomial, $\mathbb{Z}_q[x]/F(x)$: a quotient set

$$\{\boldsymbol{a} \mod (q, F(x)) | \boldsymbol{a} \in \mathbb{Z}_q[x]\}$$

- Cyclotomic polynomial: $\Phi_m(x) = \prod_{\substack{1 \le k \le m \\ \gcd(k,m)=1}} (x - e^{2i\pi\frac{k}{m}})$

Notations

$$5x^3 + 2x^2 + 4x + 1 \mod (x^2+1, 7)$$

$$\mathbf{5x^3 + 2x^2 + 4x + 1 = (5x+2)(x^2+1) + (-x-1)}$$

$$\mathbf{5x^3 + 2x^2 + 4x + 1 \equiv -x - 1 \mod x^2 + 1}$$

$$\mathbf{-x - 1 \equiv 6x + 6 \mod 7}$$

$i=0$

- F(x) a polynomial, $\mathbb{Z}_q[x]/F(x)$: a quotient set

$$\{\boldsymbol{a} \mod (q, F(x)) | \boldsymbol{a} \in \mathbb{Z}_q[x]\}$$

- Cyclotomic polynomial: $\Phi_m(x) = \prod_{\substack{1 \le k \le m \\ \gcd(k,m)=1}} (x - e^{2i\pi \frac{k}{m}})$

Notations

- $\mathbb{Z}_q$: Integer modulo q
- $\mathbb{Z}_q^n$: Vectors consists of n integer modulo q
- $\boldsymbol{a} \leftarrow \mathbb{Z}_q^n$: uniformly sample $\boldsymbol{a}$ from $\mathbb{Z}_q^n$
- $\mathbb{Z}_q[x] := \{\sum_{i=0}^{*} \alpha_i x^i | \alpha_i \in \mathbb{Z}_q\}$: set of polynomials
- F(x) a polynomial, $\mathbb{Z}_q[x]/F(x)$: a quotient set

$$\{\boldsymbol{a} \mod (q, F(x)) | \boldsymbol{a} \in \mathbb{Z}_q[x]\}$$

- Cyclotomic polynomial: $\Phi_m(x) = \prod_{\substack{1 \le k \le m \\ \gcd(k,m)=1}} (x - e^{2i\pi \frac{k}{m}})$

Notations

- $\mathbb{Z}_q$: Integer modulo q
- $\mathbb{Z}_q^n$: Vectors consists of n integer modulo q
- $\boldsymbol{a} \leftarrow \mathbb{Z}$ ... $\mathbb{Z}^n$

Cyclotomic polynomial: Some “prime” polynomial

$$m = 2^{d+1} \Leftrightarrow \Phi_m(x) = x^{2^d} + 1$$

$$\deg(\Phi_m(x)) = \phi(m)$$

- $\mathbb{Z}_q[x$ ...
- F(x) ... set

$$\{\boldsymbol{a} \mod (q, F(x ... \in \mathbb{Z}_q[x]\}$$

- Cyclotomic polynomial: $\Phi_m(x) = \prod_{\substack{1 \le k \le m \\ \gcd(k,m)=1}} (x - e^{2i\pi\frac{k}{m}})$

Learning With Errors

- LWE-Assumption

$$\boldsymbol{a} \leftarrow \mathbb{Z}_q^n \quad \boldsymbol{s} \leftarrow \mathbb{Z}_q^n \quad \boldsymbol{e} \leftarrow \chi^n \quad \boldsymbol{r}_1, \boldsymbol{r}_2 \leftarrow \mathbb{Z}_q^n$$

$$(\boldsymbol{a}, \langle \boldsymbol{a}, \boldsymbol{s} \rangle + \boldsymbol{e}) \approx^{\mathrm{c}} (\boldsymbol{r}_1, \boldsymbol{r}_2)$$

- Ring-LWE: use a polynomial ring instead

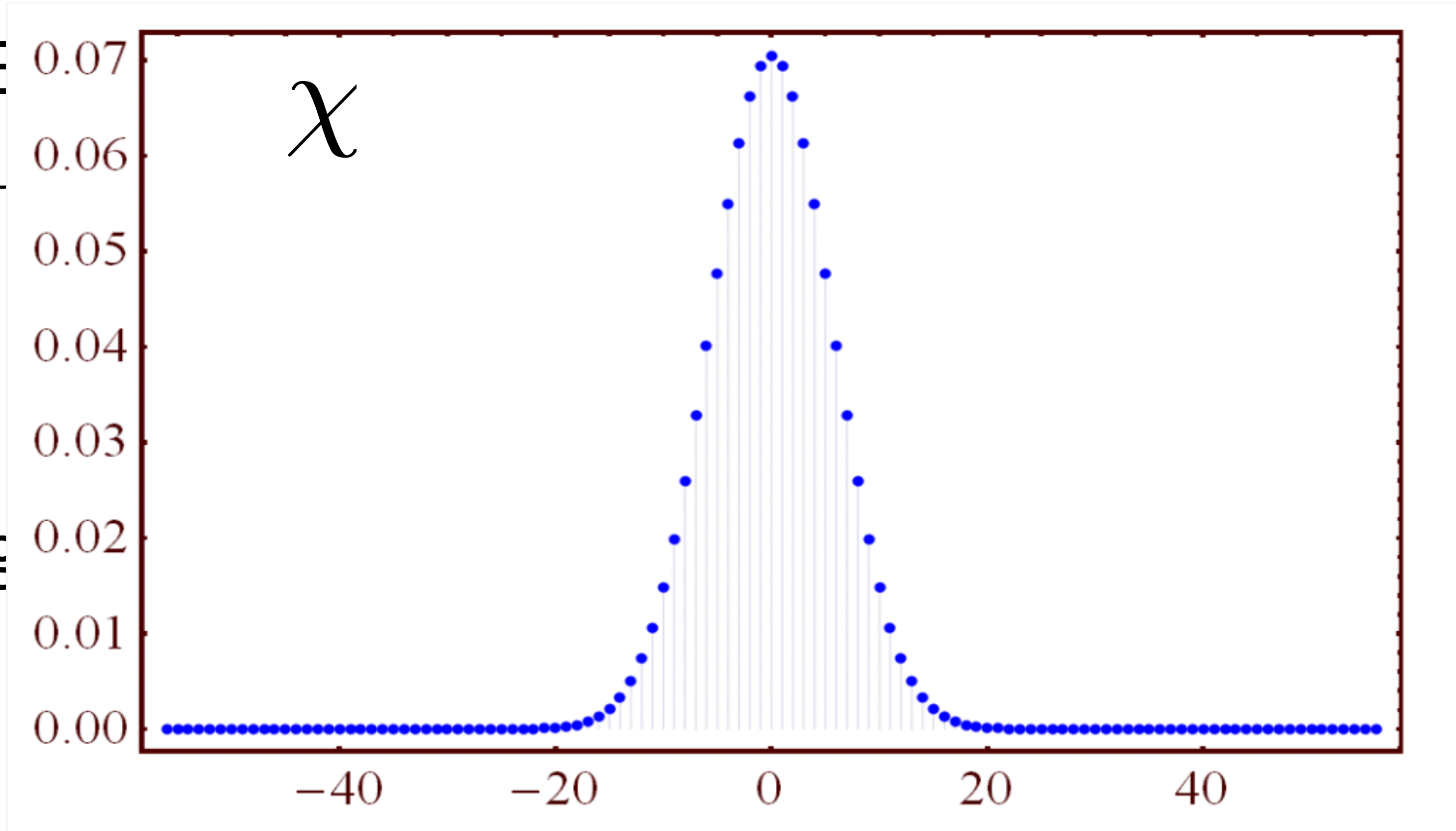
$$\mathbb{Z}_q^n \Rightarrow \mathbb{Z}_q[x]/\Phi_m(x) \quad n = \phi(m)$$

[On lattices, learning with errors, random. Regev 2005]

Learning With Errors

- LWE

  $\boldsymbol{a} \leftarrow$



- Ring

[On lattices, learning with errors, random. Regev 2005]

Learning With Errors

- LWE-Assumption

$$\boldsymbol{a} \leftarrow \mathbb{Z}_q^n \quad \boldsymbol{s} \leftarrow \mathbb{Z}_q^n \quad \boldsymbol{e} \leftarrow \chi^n \quad \boldsymbol{r}_1, \boldsymbol{r}_2 \leftarrow \mathbb{Z}_q^n$$

$$(\boldsymbol{a}, \langle \boldsymbol{a}, \boldsymbol{s} \rangle + \boldsymbol{e}) \approx^{\mathrm{c}} (\boldsymbol{r}_1, \boldsymbol{r}_2)$$

- Ring-LWE: use a polynomial ring instead

$$\mathbb{Z}_q^n \Rightarrow \mathbb{Z}_q[x]/\Phi_m(x) \quad n = \phi(m)$$

[On lattices, learning with errors, random. Regev 2005]

Add. & Mul. On $\mathbb{Z}_q[x]/(x^n+1)$

(i.e. n is power of 2)

For example: $q = 17, n = 4$

$$\boldsymbol{a} := 15 + 2x + 4x^2 + 7x^3 \in \mathbb{Z}_{17}[x]/(x^4+1)$$

$$\boldsymbol{b} := 8 + 9x + 3x^2 + 4x^3 \in \mathbb{Z}_{17}[x]/(x^4+1)$$

$$\boldsymbol{a} + \boldsymbol{b} = 6 + 11x + 7x^2 + 11x^3 \mod (17, x^4+1)$$

★ $x^4 \equiv -1 \mod (x^4+1)$

$$\begin{aligned} \boldsymbol{a} \cdot \boldsymbol{b} &= 120 + 151x + 95x^2 + 158x^3 + 83x^4 + 37x^5 + 28x^6 \\ &\equiv 37 + 114x + 67x^2 + 158x^3 \mod (x^4+1) \\ &\equiv 3 + 12x + 16x^2 + 5x^3 \mod (17, x^4+1) \end{aligned}$$

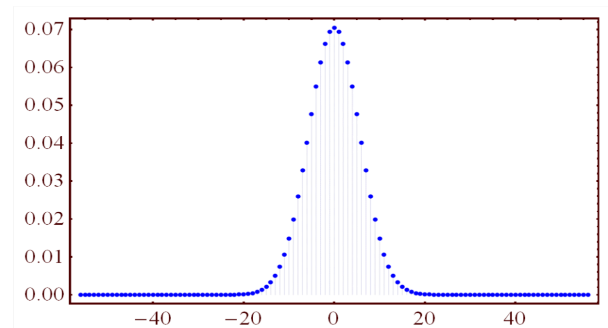
Paremeters in RLWE-based scheme

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

$$p : \text{prime number }, \text{integer } r$$

$$\mathbb{Z}_{p^r}[x], \text{ polynomial ring}$$

σ The stand deviation of the discrete Gaussian distribution (default 3.2)



$$\text{ciphertext space parameter:}$$

$$q = q(m, p, r, \sigma, \kappa)$$

Security parameter

Message Space & Ciphertext Space

- Message Space: polynomial quotient ring

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

Coefficients modulo p

Polynomial modulo $\Phi_m(x)$

- Ciphertex Space

$$R_q := \mathbb{Z}_q[x]/\Phi_m(x); q \gg p$$

Coefficients modulo q

Polynomial modulo $\Phi_m(x)$

Basic Encryption Scheme Operations

- KeyGeneration

$$\boldsymbol{s} \leftarrow \chi^n, \boldsymbol{a}_1 \leftarrow R_q, \boldsymbol{e} \leftarrow \chi^n \text{ i.e. } n = \phi(m)$$

$$\boldsymbol{a}_0 := -(\boldsymbol{a}_1 \cdot \boldsymbol{s} + \boldsymbol{e} \cdot p)$$

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

$$\text{public key, pk} := (\boldsymbol{a}_0, \boldsymbol{a}_1)$$

[**Can Homomorphic Encryption be Practical? K. Lauter et al. 2011**]

Basic Encryption Scheme Operations

- Encryption $\text{message } \boldsymbol{M} \in R_p \quad \text{pk} \ := (\boldsymbol{a}_0, \boldsymbol{a}_1)$

$$\boldsymbol{u}, \boldsymbol{f}, \boldsymbol{g} \leftarrow \chi^n$$

$$\text{ctx} \ := (\boldsymbol{c}_0 := \boldsymbol{a}_0\boldsymbol{u} + \boldsymbol{g}p + \boldsymbol{M}, \boldsymbol{c}_1 := \boldsymbol{a}_1\boldsymbol{u} + \boldsymbol{f}p)$$

additions, multiplications over polynomial ring.

- Decryption $\text{ctx} := (\boldsymbol{c}_0, \boldsymbol{c}_1, \cdots, \boldsymbol{c}_k) \ \text{sk} := \boldsymbol{s}$

$$\boldsymbol{M} = \sum_{i=0}^{k} \boldsymbol{c}_i \boldsymbol{s}^i \mod (p, \Phi_m(x))$$

Homomorphic Operations

- Addition $\mathrm{ctx}_1 = (\boldsymbol{c}_0, \boldsymbol{c}_1, \cdots, \boldsymbol{c}_k)\ \mathrm{ctx}_2 = (\boldsymbol{c}'_0, \boldsymbol{c}'_1, \cdots, \boldsymbol{c}'_k)$

$$\mathrm{ADD} = (\boldsymbol{c}_0 + \boldsymbol{c}'_0, \boldsymbol{c}_1 + \boldsymbol{c}'_1, \cdots, \boldsymbol{c}_k + \boldsymbol{c}'_k)$$

- Multiplication $\mathrm{ctx}_1 = (\boldsymbol{c}_0, \boldsymbol{c}_1), \mathrm{ctx}_2 = (\boldsymbol{c}'_0, \boldsymbol{c}'_1)$

$$\mathrm{MUL} = (\boldsymbol{c}_0\boldsymbol{c}'_0, \boldsymbol{c}_0\boldsymbol{c}'_1 + \boldsymbol{c}_1\boldsymbol{c}'_0, \boldsymbol{c}_1\boldsymbol{c}'_1)$$

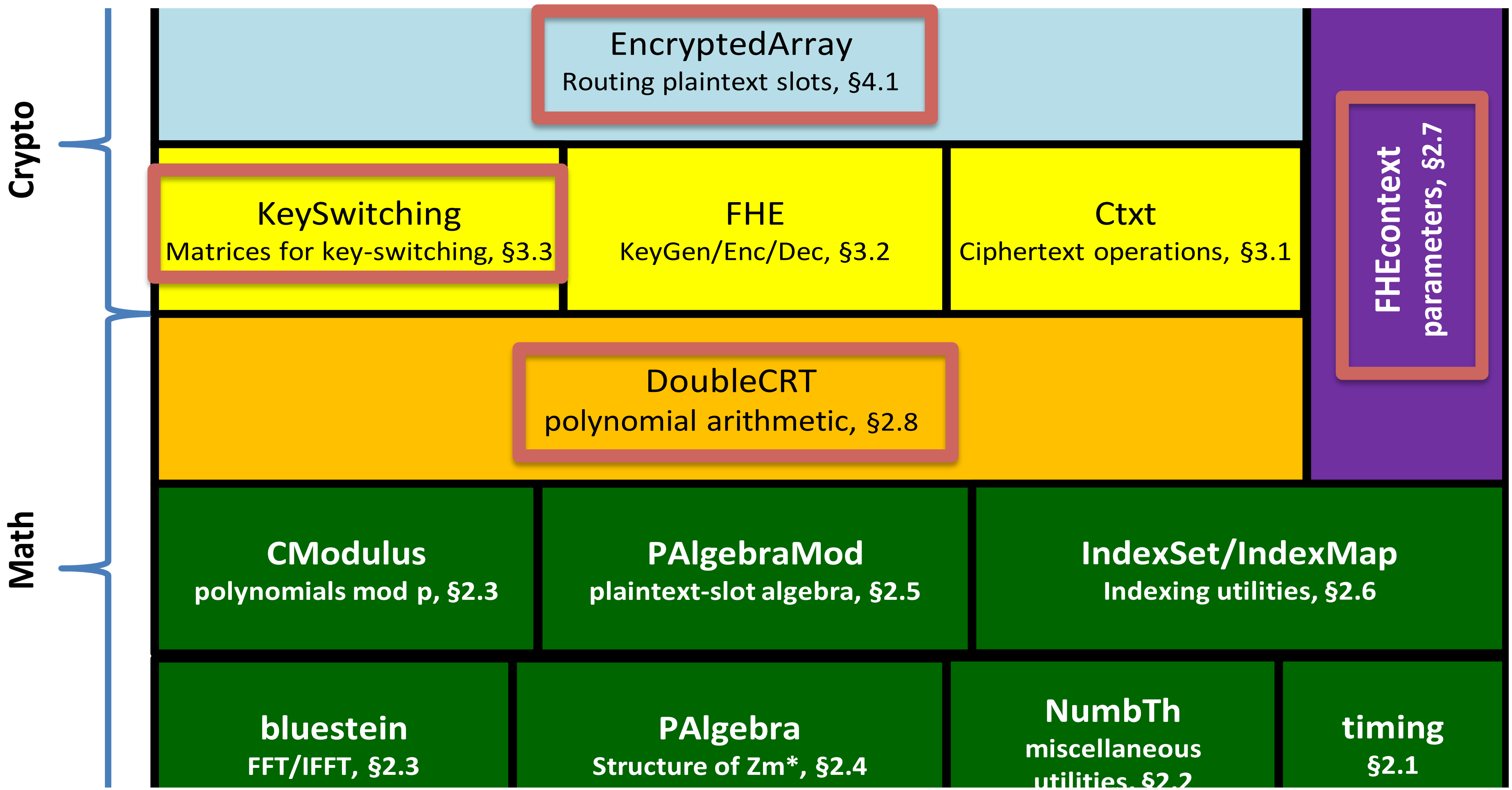
The size of ciphertext increases!

HElib

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

[https://github.com/shaih/Helib]

Architecture of HElib

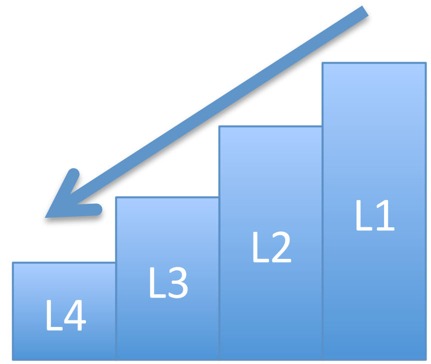


* Reference from the HElib design document

Leveled homomorphic encryption

The BGV-type scheme is a *leveled* homomorphic encryption scheme

- What is levels?
  - The ciphertext space is not fixed.



- Why *need* levels ?
  - Bootstrapping is too too heavy
  - To *somehow* reduce the noise inside ciphertexts
- When to change the level?
  - Majorly after ciphertexts multiplication

[(Leveled) fully homomorphic encryption without bootstrapping]
Brakerski, Gentry, Vaikuntanathan 2012

Parameters of the Leveled homomorphic encryption

1. *An positive integer L, called levels*
2. *A prime sequence* $q_1 > q_2 > \cdots > q_L$

- *The ciphertext-space changes level by level*

$$R_{q_1} := \mathbb{Z}_{q_1}/\Phi_m(x) \xrightarrow{\text{One multiplication}} R_{q_2} := \mathbb{Z}_{q_2}/\Phi_m(x)$$

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

[(Leveled) fully homomorphic encryption without bootstrapping]
Brakerski, Gentry, Vaikuntanathan 2012

How to decide the levels?

- Majorly depends on the evaluation function

$\mathrm{Enc}(a) \otimes \mathrm{Enc}(b) \otimes \mathrm{Enc}(c)$ Need *at least* 2-levels

$\mathrm{Enc}(a) \otimes \mathrm{Enc}(b) \oplus \mathrm{Enc}(c) \otimes \mathrm{Enc}(d)$ 1-level may also works

$\bigoplus_{i=1}^{10000} \mathrm{Enc}(a_i)$ 1-level may not works

reLinearization (Key switching)

$$\mathrm{MUL} = (\boldsymbol{c}_0 \boldsymbol{c'}_0, \boldsymbol{c}_0 \boldsymbol{c}'_1 + \boldsymbol{c}_1 \boldsymbol{c}'_0, \boldsymbol{c}_1 \boldsymbol{c}'_1)$$

The dimension of ciphertext increases!

- What is relinearization?

$$\mathrm{ctx} = (\boldsymbol{c}_0, \boldsymbol{c}_1, \boldsymbol{c}_2) \Rightarrow \mathrm{ctx}' = (\boldsymbol{c}'_0, \boldsymbol{c}'_1)$$

$$\mathrm{Dec}_{\mathrm{s}k}(\mathrm{ctx}) = \mathrm{Dec}_{\mathrm{s}k}(\mathrm{ctx}')$$

[Efficient fully homomorphic encryption from LWE Brakerski, Vaikuntanathan, 2011]

reLinearization (Key switching)

- Why *want* to reLinearize ?
  - To reduce the overhead in ciphertext multiplication

$$(\boldsymbol{c}_0, \boldsymbol{c}_1), (\boldsymbol{c}'_0, \boldsymbol{c}'_1) \ \Rightarrow \text{need 4 polynomial multiplications}$$

$$(\boldsymbol{c}_0, \boldsymbol{c}_1, \boldsymbol{c}_2, \boldsymbol{c}_3), (\boldsymbol{c}'_0, \boldsymbol{c}'_1) \Rightarrow \text{need 8 polynomial multiplications!}$$

- Need to add extra information into the public key
- Should always reLinearize ?
  - Depends on the multiplication depth

[Efficient fully homomorphic encryption from LWE Brakerski, Vaikuntanathan, 2011]

Sample codes: Setup

```
1.   FHEContext context(m, p, r);
2.   buildModChain(context, L);
3.   FHESecKey sk(context);
4.   sk.GenSecKey(64);
5.   addSome1DMatrices(sk);
6.   FHEPubKey pk = sk;
```

R_{p^r}

levels

To add extra information for reLinearization

Line 6: The FHESecKey class was designed to inherit from the FHEPubKey class

Sample codes: Enc/Dec/Mult

$\mathbb{Z}[x]$

```
1.  Ctxt ctxt(pk);
2.  ZZX plain = to_ZZX(10);
3.  pk.Encrypt(ctxt, plain); // Enc(10)
4.  ctxt.mulByConstant(to_ZZX(2)); // Enc(20)
5.  ctxt.addConstant(to_ZZX(10)); // Enc(30)
6.  // using reLineration
7.  ctxt.multiplyBy(ctxt); // Enc(900)
8.  // not using reLineration
9.  ctxt *= ctxt; // Enc(810000)
10. sk.Decrypt(plain, ctxt);        // plain = 810000 mod p^r
```

Line 2: Plaintext need to be a polynomial.

Line 7 & 9: To use or not use reLinerazation during homomorphic multiplication

Packing

What is packing ?

- To pack several messages into one ciphertext

Why use packing ?

1. To reduce the numbers of ciphertext
2. To amortize the computation time

Different kinds of packing

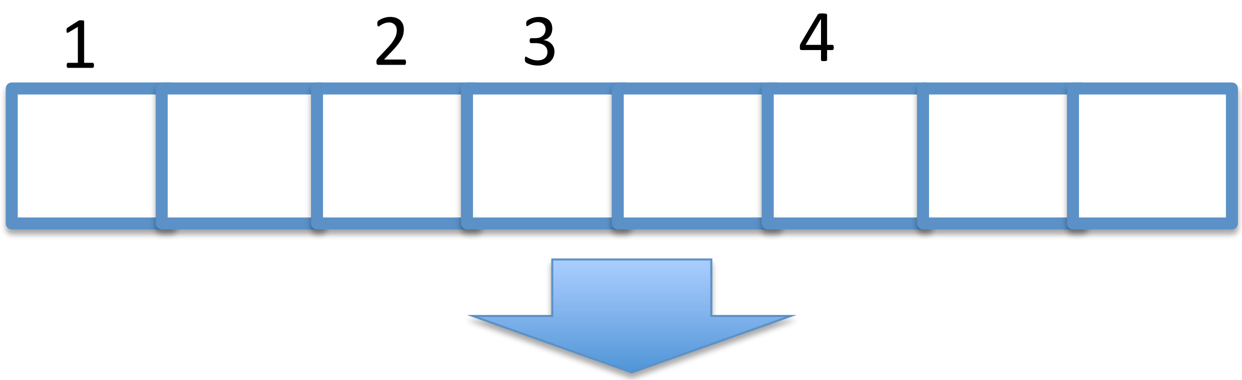
- Pack into coefficients
- Pack into subfields (so-called CRT-based packing)

I. Pack into coefficients

- Example Message Space $p = 13, m = 16, r = 2$

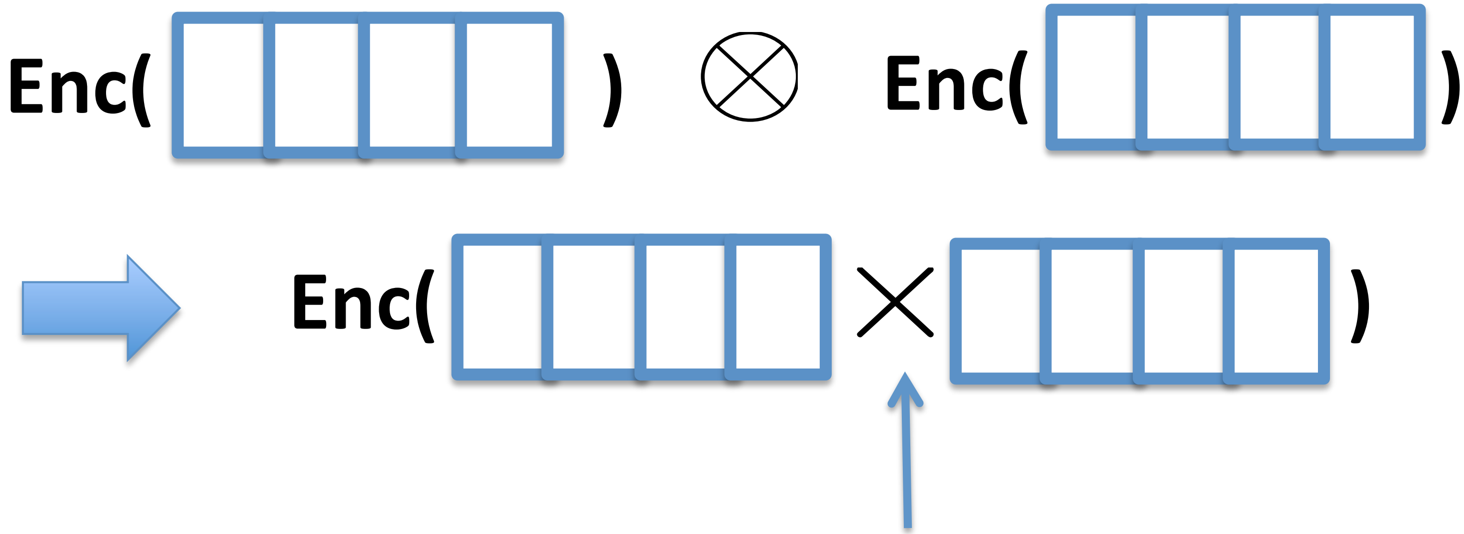
$$R_{13^2} := \mathbb{Z}_{13^2}[x]/(x^8+1)$$

- image that 8 boxes and each can put in a less than 13^2 positive integer.



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

I. Pack into coefficients



Just the multiplication between polynomials!
mod (13^2, x^8 + 1)

- <u>We need to design how to encode our data into a useful polynomial form</u>

Example: Encoding for *scalar product*

- Given $\boldsymbol{v} = [1, 2, 3], \boldsymbol{u} = [4, 5, 6]$
- If we make two polynomials such as

$$V(x) = 1 + 2x + 3x^2 \qquad U(x) = 4 + 5x + 6x^2$$

- But

$$V(x)U(x) = 4 + 13x + 28x^2 + 27x^3 + 18x^3$$

- Change a little bit $\hat{U}(x) = 6 + 5x + 4x^2$

$$V(x)\hat{U}(x) = 6 + 17x + 32x^2 + 23x^3 + 12x^4$$

$$32 = \langle \boldsymbol{v}, \boldsymbol{u} \rangle$$

Sample codes:
Pack into
Coefficients

```
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^3 + 1x^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);
cout << result[2];// 30 mod p^r
```

3rd coeff.

II. Pack into subfields

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

Consider the CRT in the integer field

A number p can be factorized into *prime factors*

$$p = \prod_{i=1}^{\ell} p_i \qquad \text{e.g. } 15 = 3 \times 5$$

We have the isomorphism from CRT

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

where $\otimes$ is Cartesian product

II. Pack into subfields

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

Consider the CRT in the integer field

A number p can

$$\prod^{\ell}$$

$0 \cong (0 \mod 3, 0 \mod 5)$	$1 \cong (1 \mod 3, 1 \mod 5)$	$2 \cong (2 \mod 3, 2 \mod 5)$
$3 \cong (0 \mod 3, 3 \mod 5)$	$4 \cong (1 \mod 3, 4 \mod 5)$	$5 \cong (2 \mod 3, 0 \mod 5)$
$6 \cong (0 \mod 3, 1 \mod 5)$	$7 \cong (1 \mod 3, 2 \mod 5)$	$8 \cong (2 \mod 3, 3 \mod 5)$
$9 \cong (0 \mod 3, 4 \mod 5)$	$10 \cong (1 \mod 3, 0 \mod 5)$	$11 \cong (2 \mod 3, 1 \mod 5)$
$12 \cong (0 \mod 3, 2 \mod 5)$	$13 \cong (1 \mod 3, 3 \mod 5)$	$14 \cong (2 \mod 3, 4 \mod 5)$

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

where $\otimes$ is Cartesian product

II. Pack into subfields

- Polynomial-CRT

The cyclotomic polynomial can be factorized into distinct ℓ *irreducible* polynomials
(Some “prime” polynomial)

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

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

Euler function

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

called *slots*

Example: Component-wise Operations

- 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 hold <u>degree 0 polynomial modulo 17</u>.

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

$$\begin{aligned} 1 + x + 7x^2 + 12x^3 &\equiv 8 \mod (17, x-2) \\ &\equiv 5 \mod (17, x-2^3) \\ &\equiv 16 \mod (17, x-2^5) \\ &\equiv 9 \mod (17, x-2^7) \end{aligned}$$

Example: Component-wise Operations

$[8, 5, 16, 9]$ + $[5, 5, 3, 7]$ = $[13, 10, 2, 16]$ mod 17

$1 + x + 7x^2 + 12x^3$ + $5 + 14x + 4x^2 + 3x^3$ = $6 + 15x + 11x^2 + 15x^3$

$[8, 5, 16, 9]$ x $[5, 5, 3, 7]$ = $[6, 8, 14, 12]$ mod 17

$1 + x + 7x^2 + 12x^3$ x $5 + 14x + 4x^2 + 3x^3$ = $10 + x + 12x^3$

Example: "Rotation" Operation

On field: $\mathbb{Z}_{17}[x]/(x^4+1)$

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

An automorphism

Replace $x \longrightarrow x^5$

$$1 + x + 7x^2 + 12x^3 \Longrightarrow 1 + x^5 + 7(x^5)^2 + 12(x^5)^3$$

$$\Big\downarrow \bmod (17, x^4+1)$$

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

2 left-rotated !!

Operations supported by HElib

- Component-wise (entry-wise) addition/mult.
- Rotation

  $$\mathrm{Enc}([1,2,3,4]) \lll 3 \;\Rightarrow \mathrm{Enc}([4,1,2,3])$$

  – Shift; padding with 0s

  $$\mathrm{Enc}([1,2,3,4]) \gg 2 \Rightarrow \mathrm{Enc}([0,0,1,2])$$

  – Running sums

  $$\mathrm{RS}(\mathrm{Enc}([1,2,3,4])) \Rightarrow \mathrm{Enc}([1,3,6,10])$$

  – total sums

  $$\mathrm{TS}(\mathrm{Enc}([1,2,3,4])) \Rightarrow \mathrm{Enc}([10,10,10,10])$$

Sample codes

Codes for CRT-packing

```
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), envU(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, u);
encV *= encU;
ZZX result;
sk.Decrypt(result, encV); // result = ??
std::vector<long> decoded;
ea.decode(result, decoded);//decoded = {4, 6, 6, 4};
//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 envU(pk);
ea.encrypt(encU, pk, u);//Enc([1, 2, 3, 4])
ea.rotate(encU, 1);//Enc([4, 1, 2, 3])
ea.rotate(encU, -2);//Enc([2, 3, 4, 1])
ea.shift(encU, 1);//Enc([0, 2, 3, 4])
runnigSums(ea, encU);//Enc([0, 2, 5, 9])
totalSums(ea, encU);//Enc([16, 16, 16, 16])
```

Noise Estimation

- *Ok to decrypt:* $\sqrt{\text{noiseVar}} \leq q_{\text{current}}/2$
- *Fresh ciphertext:* $\sigma^2(1+\phi(m)^2/2+\phi(m)(H+1))(p/2)^2$
- *Modulus-switch:* $\text{noiseVar}/(\frac{q_i}{q_{i+1}})^2 + \text{addedNoise}$
- *reLinearation:* $\text{noiseVar} + k \cdot \phi(m) \cdot \sigma^2 \cdot p^2 \cdot$
- *Ctxt-plain add.:* $\text{noiseVar} + (q_{current} \mod p)^2 \cdot \phi(m) \cdot (p/2)^2$
- *Ctxt-plain mult.:* $\text{noiseVar} \cdot \phi(m) \cdot (p/2)^2$

* Reference from the HElib design document

Noise Estimation

- *Ctxt-ctxt add.:* $\text{noiseVar} + \text{noiseVar}'$
- *Ctxt-ctxt mult.:* $\text{noiseVar} \cdot \text{noiseVar}' \cdot \binom{r_1 + r_2}{r_1}$
- *Rotation*: 1 ctxt-plain mult., 1 ctxt-plain add.
- *Shift: 2 ctxt-plain mult.*

* Reference from the HElib design document

Application χ^2-Test on epidemiology

observed

	a	A	a (expected)	A(expected)	Count
Case	o1	o2	e1=n3n1/n	e2=n4n1/n	n1
Control	o3	o4	e3=n3n2/n	e4=n4n2/n	n2
Count	n3	n4			n

$$\chi^2 = \sum_{k=1}^{4} \frac{(o_k - e_k)^2}{e_k}$$

Application χ^2-Test on epidemiology

Data:

[AA, Aa, aa, Aa, …..] **[case, control, case, control, ….]**

Encoding **Encoding**

x=[2 , 1, 0, 1, …..] **y=[1 , 0, 1, 0, …..]**

	a	A	Count
Case	o1	o2	n1
Control	o3	o4	n2
Count	n3	n4	n

$$o_2 = \langle \boldsymbol{x}, \boldsymbol{y} \rangle$$

$$n_1 = \langle \boldsymbol{y}, \mathbf{1} \rangle$$

$$n_4 = \langle \boldsymbol{x}, \mathbf{1} \rangle$$

Application χ^2-Test on epidemiology

Data:

[AA, Aa, aa, Aa, ...] **[case, control, case, control, ...]**

Encoding **Encoding**

x=[2 , 1, 0, ... **0, 1, 0, ...]**

How to efficiently compute scalar product?? Packing!

	a		
Case	o1		
Control	o3	o4	n2
Count	n3	n4	n

$$= \langle x, y \rangle$$

$$_1 = \langle y, 1 \rangle$$

$$n_4 = \langle x, 1 \rangle$$

Misc: Example to choose parameters

- Problem: To calculate chi-square test
- Main computation: scalar product of integer vectors
- Integer vectors: $\{0,1,2\}^N$; Data set size $N = 2000$

1. Plaintext space parameters: **p, r**

   $p^r > 4000; \text{ such as, p} = 2\text{, r} = 12\text{; or p} = 4001\text{, r} = 1$

2. Polynomial parameter: **m**

   $\phi(m)$ > 2000 to pack N integers

3. Levels parameter: **L**

   Pack into coefficient: Only need 1 multi. => L = 1

   Pack into subfields: Need 1 multi. & 1 running sums => L = 2 is better

4. Check m again by calling FindM

Misc: FindM() & the number of slots

- HElib providers a such function:

  FindM(k, L, p, m);

  It print out :

  *** Bound N=XXX, choosing m=YYY, phi(m)=ZZZ

  k-bits security <u>if phi(m) >= N</u>

- The number of slots is defined as:

$$\#\mathrm{slot} = \frac{\phi(m)}{\mathrm{order}(m, p)}$$

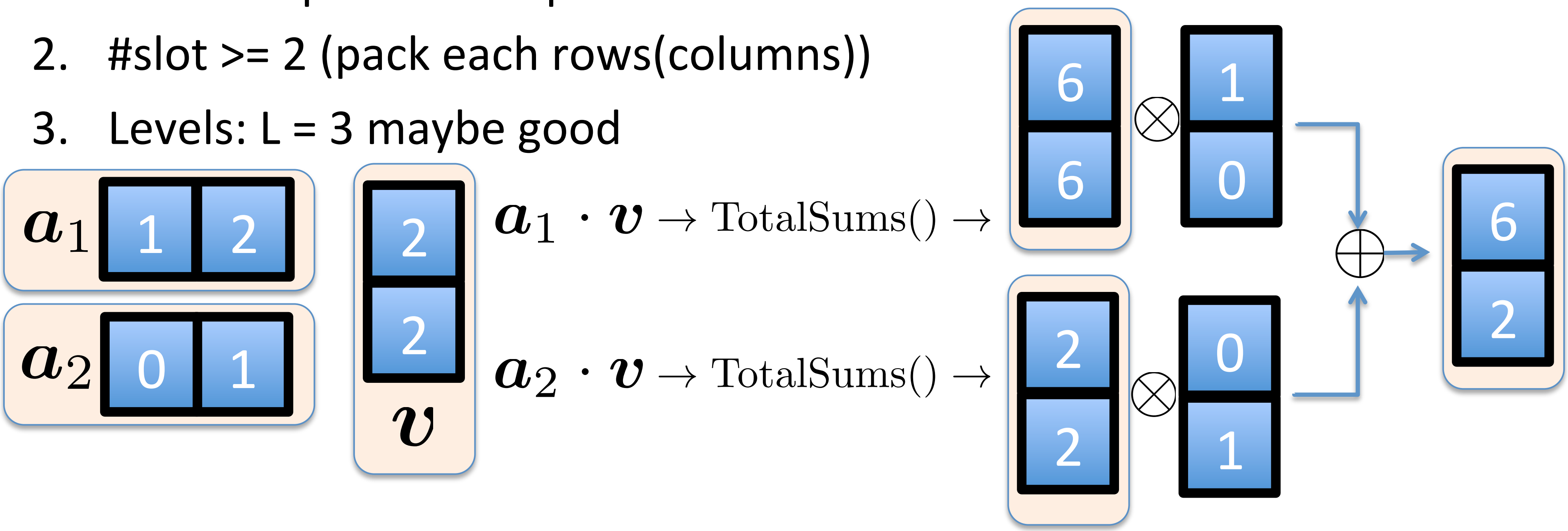
$$o := \mathrm{order}(m, p); \mathrm{then}\ p^o \equiv 1 \mod m$$

Misc: Example to choose parameters

- Problem: To compute the product of matrix and vector

$$\boldsymbol{A} \in \{0,1,2\}^{2\times 2} \qquad \boldsymbol{v} \in \{0,1,2\}^{2}$$

1. Plaintext parameter: p^r > 8
2. #slot >= 2 (pack each rows(columns))
3. Levels: L = 3 maybe good



4. m is decided by FindM() according to L, p, #slots

Misc.: Rules of thumb

- By default, only use half of the machine bits for levels
  1. 32-bits platform, open *–DNOT_HALF_PRIME* flag before building the HElib
  2. 64-bits platform: before call buildModChain()

$$\text{context.bitsPerLevel} \; = 14 + \frac{\log_2(m)}{2} + r\log_2(p);$$

- Plaintext parameter p^r != 2, better to add more levels

$$L \mathrel{+}= 2\lfloor \frac{3r\log_2(p)}{\text{FHE\_p2Size}} \rfloor + 1;$$

[https://github.com/shaih/HElib/issues/52]

Misc.: Install HElib

- 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)

References

- 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.*

Thank you!