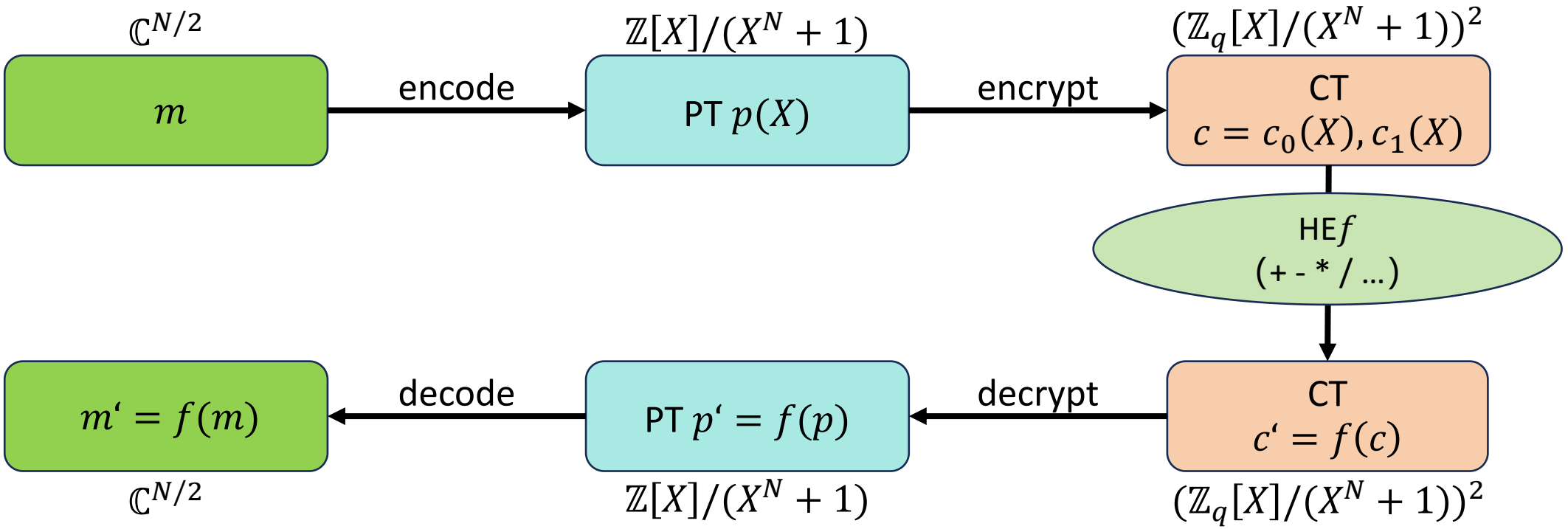
Implementation of Somewhat Fully Homomorphic Encryption

Huang Chi

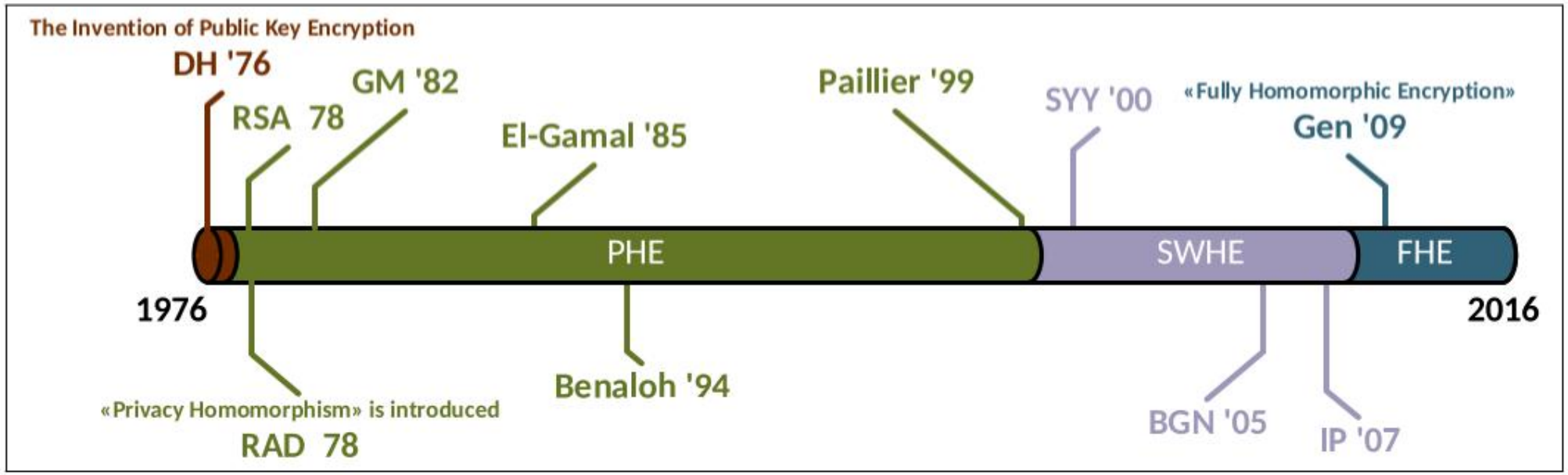

$\mathbb{C}^{N/2}$
m
encode
$\mathbb{Z}[X]/(X^N+1)$
PT $p(X)$
encrypt
$(\mathbb{Z}_q[X]/(X^N+1))^2$
CT
$c = c_0(X), c_1(X)$
HEf
(+ - * / ...)
CT
$c' = f(c)$
decrypt
PT $p' = f(p)$
decode
$m' = f(m)$
$\mathbb{C}^{N/2}$
$\mathbb{Z}[X]/(X^N+1)$
$(\mathbb{Z}_q[X]/(X^N+1))^2$

PHE

SWHE

LFHE

TFHE/FHEW



Papers/Resources

- FV12 Somewhat Practical Fully Homomorphic Encryption
- (2012/099): Homomorphic Evaluation of the AES Circuit
- (2021/204)： Revisiting Homomorphic Encryption Schemes for Finite Fields
- OpenFHE: github: openfheorg/openfhe-development

Related:

- CKKS17： Homomorphic Encryption for Arithmetic of Approximate Numbers
- MP21： Bootstrapping in FHEW-like Cryptosystems

Source Code:

- : MyCPlusPlus - LeetCode Playground (https://leetcode.cn/playground/rbCj7dfq/)

力扣 学习 题库 竞赛 讨论 求职 商店 0 Plus 会员

执行代码 MyCPlus... 3 / 5 扩容至无限 保存 C++ 输出:

```
#include <bits/stdc++.h>
using namespace std;

typedef long long int64;
typedef vector<int64> polyvec;

// ## 测试/输出方法
void print_vector(const polyvec &a, const string &s) {
      cout << s << " [";
      for (const int64 &x: a)
```

Preliminaries

Implementations

Future Works

Part 1

Preliminaries

1.1 Polynomial Rings

- Let $\mathbb{Z}$ be the ring of integers
- and $\mathbb{Z}_q$ be the integer ring modulo q
- $\mathbb{Z}$[x] to denote the polynomial ring in the variable x with integer coefficients
- $\mathbb{Z}_q$[x] to denote the polynomial ring with coefficients in $\mathbb{Z}_q$
- $R_q = \mathbb{Z}_q[x]/(x^n + 1)$, usually take $n = 2^k$

Example 1: Consider the polynomial ring $\mathbb{Z}_2[x]/(x^2 + 1)$. This polynomial ring has four elements: $\{0, 1, x, x + 1\}$. We can now multiply two polynomials, e.g., $x \cdot (x + 1) = x^2 + x \equiv x + 1 \pmod{q, x^2 + 1}$ as $x^2 \equiv -1 \pmod{q, x^2 + 1}$.

1.1 Presentations of Polynomial Rings

- Coefficient Representation
- Value Presentatation

1.2 Polynomial Multiplication (Schoolbook)

- Given two n-coefficient polynomials a, b:

  $a = \sum_{i=0}^{n-1} a_i x^i \quad b = \sum_{i=0}^{n-1} b_i x^i$

- The multiplication is defined as:

$$a \cdot b = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} a_i \cdot b_j \cdot x^{i+j}.$$

In case the polynomial ring is $\mathbb{Z}[x]/(x^n - 1)$, the multiplication becomes

$$a \cdot b \equiv \sum_{i=0}^{n-1} \sum_{j=i+1}^{n-1} a_i \cdot b_j \cdot x^{i+j} + \sum_{j=1}^{n-1} \sum_{i=n-j}^{n-1} a_i \cdot b_j \cdot x^{i+j-n} \pmod{x^n - 1}.$$

Similarly, for $\mathbb{Z}[x]/(x^n + 1)$ the multiplication is defined as

$$a \cdot b \equiv \sum_{i=0}^{n-1} \sum_{j=i+1}^{n-1} a_i \cdot b_j \cdot x^{i+j} - \sum_{j=1}^{n-1} \sum_{i=n-j}^{n-1} a_i \cdot b_j \cdot x^{i+j-n} \pmod{x^n + 1}.$$

1.3 FFT / NTT

- Fast Fourier Transform (FFT)
  - Resource: https://www.bilibili.com/video/BV1ee411c7B7
  - Why
  - How
- Number-Theoretic Transform, NTT
  - Just FFT, but $\mathbb{C} \rightarrow \mathbb{Z}_q$
  - More restrictions

1.3 NTT: Primitive root of unity

- An element $a \in \mathbb{Z}_q$ is a k-th (with $k \in \mathbb{N}$) root of unity if $a^k \equiv 1 \bmod q$.
- It is a **primitive** root of unity, if there is no $k' < k$, s.t. $a^{k'} \equiv 1 \bmod q$.
- A primitive k-th root of unity exists only if $k|(q-1)$
  (or more generally, it only exists if k divides the order of $\mathbb{Z}_q^*$.)
- ω_k denotes a primitive k-th root of unity

Example 5: 1 and $-1 \equiv q-1 \bmod q$ are 2-nd roots of unity. However, only -1 is a primitive 2-nd root of unity.
For $\mathbb{Z}_7$, there are six 6-th roots of unity ($\{1,2,3,4,5,6\}$), but only $\{3,5\}$ are primitive 6-th roots of unity.

1.3 Number-Theoretic Transform (NTT)

Given two polynomials a, b, one computes their product as

$$c = a \cdot b = \mathrm{NTT}^{-1}(\mathrm{NTT}(a) \circ \mathrm{NTT}(b))$$

1.3 Compute NTT and NTT^{-1}

When computing an NTT, one is evaluating a polynomial at powers of a primitive root of unity ω_k: $\omega_k^0, \omega_k^1, \ldots, \omega_k^{n-1}$.

The NTT [Nus82, Sec. 8.1] is defined as

$$\hat{a} = \sum_{i=0}^{n-1} \hat{a}_i x^i \quad \text{with} \quad \hat{a}_i = \sum_{j=0}^{n-1} a_j \omega_k^{i \cdot j}.$$

Transforming $\hat{a}$ to the normal domain is achieved by computing NTT^{-1} as

$$a = \sum_{i=0}^{n-1} a_i x^i \quad \text{with} \quad a_i = n^{-1} \sum_{j=0}^{n-1} \hat{a}_j \omega_k^{-i \cdot j}.$$

1.3 NTT

- Pros
  - Efficient
- Cons (Limitations)
  - For given n, only certain primes can be used
    - E.g. n=4, only 17, 41, 73, 89, 97 has primitive roots ($8 \mid p - 1$)
  - Calculation complexity is high when *p* is big

1.4 Chinese Remainder Theorem, CRT

- **CRT Core Idea:**
  - For a composite modulus $M = m_1 * m_2 * ... * m_k$ where all m_i are pairwise co-prime, the ring Z_M is isomorphic to the direct product ring $Z\_m_1 \times Z\_m_2 \times ... \times Z\_m_k$.
  - A large integer a mod M can be uniquely represented by a tuple of smaller residues (a mod m_1, a mod m_2, ..., a mod m_k).
- **Residue Number System (RNS):**
  - A practical application of CRT that decomposes large-integer arithmetic into multiple parallel small-modulus computations.
  - **Advantages:** Eliminates costly carry propagation, enables massive parallelism, and is highly amenable to hardware acceleration.

1.5 Ring Learning With Errors (RLWE)

- **RLWE Problem Definition:**
  - Let $\Phi(x)$ be an irreducible polynomial (typically $x^N + 1$, where N is a power of two), defining the quotient ring $R_q = Z_q[x] / (\Phi(x))$.
  - **Secret s(x):** Sampled from a "small" error distribution (e.g., discrete Gaussian).
  - **Challenge:** Distinguish between:
    - **RLWE samples:** $(a_i(x), b_i(x) = a_i(x) * s(x) + e_i(x))$, where $a_i(x)$ is uniform in R_q and $e_i(x)$ is a small error.
    - **Uniform random samples:** $(u_i(x), v_i(x))$, uniform in $R_q \times R_q$.
- **Security Basis:** The hardness of the RLWE problem is based on worst-case hardness assumptions of lattice problems (e.g., Shortest Vector Problem), making it a leading candidate for post-quantum security .
- **Application in BFV:** The entire security of the BFV scheme is formally reduced to the hardness of the RLWE problem. Key and ciphertext structures are derived directly from this assumption.

Part 

Implementations

2.1 RLWE base scheme

- KeyGen
  - Generate a, s, e, output pk = (b=as+e, a), sk=s
- Encrypt
  - Scaling: $\Delta = Q/t$
  - Generate u, e_1, e_2, ct=(bu+ e_1 +Δm, au+ e_2)
- Decrypt
  - De-scaling: $1/\Delta = t/Q$
  - m=(ct[0] + ct[1] * s) / Δ
- DCRT / NTT applied in above methods

2.2 Eval_Add, Eval_Sub

- ct_add = ct1 + ct2
- ct_sub = ct1 - ct2

2.2 Eval_Mult

- ctx = (ct1[0], ct1[1]) * (ct2[0], ct2[1]) / Δ

  = (ctx[0], ctx[1], ctx[2])

- Decrypt:
  - $\Delta m = ctx[0] + ctx[1] * s + ctx[2] * s^2$

2.2 Eval_Mult Prove

- ct1=(B1+ Δm_1, A1), ct2=(B2+ Δm_2, A2), when B=As
- ctx[0] = B1B2 + Δ(B1 m_2 +B2 m_1) +Δ^2 m_1 m_2
- ctx[1] = A1B2 + A1Δm_2 + A2B1 + A2Δm_1
- ctx[2] = A1A2

Result:

- Δ^2 m_1 m_2= ctx[0] + ctx[1] * s + ctx[2] * s^2

2.2 Eval Mult

- Problem: cannot use DCRT
  - $\texttt{FV.SH.Mul}(\texttt{ct}_1, \texttt{ct}_2, \texttt{rlk})$: compute

$$\mathbf{c}_0 = \left[\left\lfloor \frac{t \cdot (\texttt{ct}_1[0] \cdot \texttt{ct}_2[0])}{q} \right\rceil\right]_q$$

$$\mathbf{c}_1 = \left[\left\lfloor \frac{t \cdot (\texttt{ct}_1[0] \cdot \texttt{ct}_2[1] + \texttt{ct}_1[1] \cdot \texttt{ct}_2[0])}{q} \right\rceil\right]_q$$

$$\mathbf{c}_2 = \left[\left\lfloor \frac{t \cdot (\texttt{ct}_1[1] \cdot \texttt{ct}_2[1])}{q} \right\rceil\right]_q$$

- Need HPS/HPSPOVERQ/BEHZ etc..

2.3 Relinearization

- ctx = (ctx[0], ctx[1], ctx[2]) -> (ctr[0], ctr[1])
- Compute rlk:

$$\texttt{rlk} := \left(\left[-(\mathbf{a}_0 \cdot \mathbf{s} + \mathbf{e}_0) + \mathbf{s}^2\right]_q, \mathbf{a}_0\right). \text{ Note that } \texttt{rlk}[0] + \texttt{rlk}[1] \cdot \mathbf{s} = \mathbf{s}^2 + \mathbf{e}_0.$$

- Problem?

> The problem however is that since $\mathbf{c}_2$ is a random element in R_q, the noise $\mathbf{e}_0$ will be magnified too much resulting in a bad approximation of $\mathbf{c}_2 \cdot \mathbf{s}^2$ and thus causing a huge error $\mathbf{r}$.

2.3 Relinearization: version1 vs version2

- `FV.SH.Relin Version 1`: write $\mathbf{c}_2$ in base T, i.e. write $\mathbf{c}_2 = \sum_{i=0}^{\ell} \mathbf{c}_2^{(i)} T^i$ with $\mathbf{c}_2^{(i)} \in R_T$ and set

$$\mathbf{c}'_0 = \left[\mathbf{c}_0 + \sum_{i=0}^{\ell} \mathtt{rlk}[i][0] \cdot \mathbf{c}_2^{(i)} \right]_q \quad \text{and} \quad \mathbf{c}'_1 = \left[\mathbf{c}_1 + \sum_{i=0}^{\ell} \mathtt{rlk}[i][1] \cdot \mathbf{c}_2^{(i)} \right]_q .$$

  Return $(\mathbf{c}'_0, \mathbf{c}'_1)$.
- `FV.SH.Relin Version 2`: compute

$$(\mathbf{c}_{2,0}, \mathbf{c}_{2,1}) = \left(\left[\left\lfloor \frac{\mathbf{c}_2 \cdot \mathtt{rlk}[0]}{p} \right\rceil \right]_q , \left[\left\lfloor \frac{\mathbf{c}_2 \cdot \mathtt{rlk}[1]}{p} \right\rceil \right]_q \right) ,$$

  and return $\left([\mathbf{c}_0 + \mathbf{c}_{2,0}]_q , [\mathbf{c}_1 + \mathbf{c}_{2,1}]_q \right)$.

Part 3

Future Works

3 Future Works

- Relinearization
- LeveledSHE
- Bootstraping (TFHE/FHEW)
- Theme switching (RLWE -> LWE)

Thank You