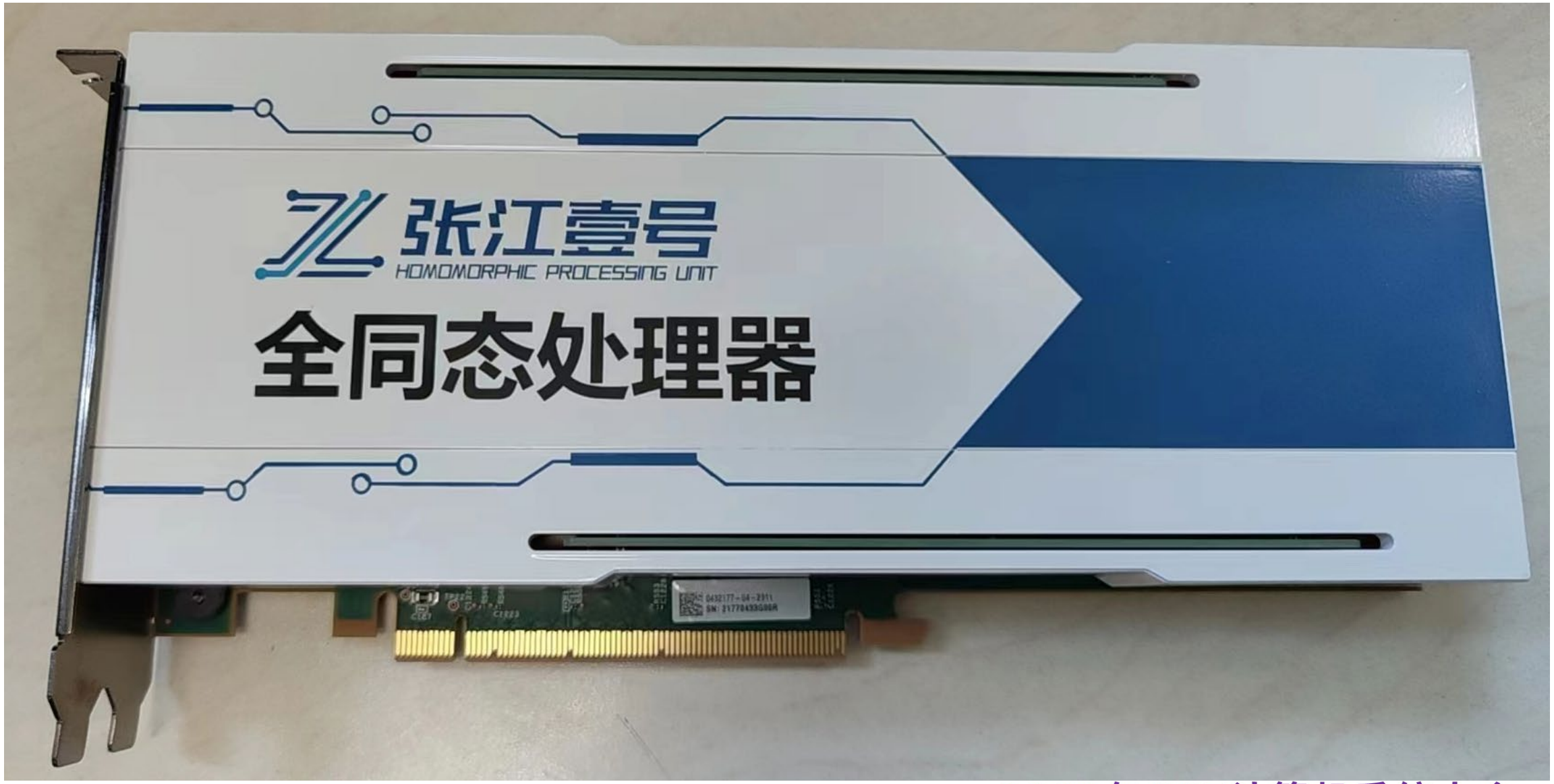
全同态处理器和算子库的设计探索

路 航

Poseidon

张江壹号
HOMOMORPHIC PROCESSING UNIT


张江壹号
HOMOMORPHIC PROCESSING UNIT
全同态处理器

全卡性能——CKKS

应用/平台	CPU （s）	张江壹号 （s）	提升
Degree = 2048			
密态查询PIR	53.748	1.486	**36.1x**
医疗隐私计算	1.987	0.093	**21.4x**
自举	51.372	3.070	**16.9x**
Degree = 4096			
密态查询PIR	146.403	5.719	**25.9x**
医疗隐私计算	4.001	0.165	**24.2x**
自举	110.866	6.536	**16.9x**
Degree = 8192			
密态查询PIR	512.426	26.773	**20.7x**
医疗隐私计算	8.237	0.355	**23.2x**
自举	234.469	13.895	**16.9x**

全卡性能——CKKS

应用/平台	CPU （s）	张江壹号 （s）	提升
Degree = 16384			
密态查询PIR	1751.006	93.864	**18.6x**
医疗隐私计算	16.749	0.689	**24.3x**
自举	577.843	34.773	**16.6x**
Degree = 32768			
密态查询PIR	5669.146	281.891	**20.1x**
医疗隐私计算	34.848	1.766883	**19.7x**
自举	1384.611	83.134	**16.6x**

全卡性能——BFV

应用/平台	CPU （s）	张江壹号 （s）	提升
Degree = 4096			
密态查询 BasePIR	0.1898	0.0081	**16.1x**
Degree = 8192			
密态查询 BasePIR	0.80467	0.0136	**23.4x**
Degree =16384			
密态查询 BasePIR	3.3973	0.0346	**59.1x**
Degree =32768			
密态查询 BasePIR	13.2914	0.1080	**98.2x**

Poseidon全同态计算库

Poseidon

Poseidon全同态计算库——核心数据结构 CKKS

CKKS	类型	名称
1	内存地址管理类	MemoryPool
2	**CKKS加密方案的参数类**	**CKKSParametersLiteralDefault**
3	上下文信息管理类	PoseidonContext
4	生成伪随机数的类	Blake2xbPRNGFactory
5	明文类	Plaintext
6	密文类	Ciphertext
7	公钥类	PublicKey
8	重现性化密钥类	RelinKeys
9	伽罗瓦密钥类	GaloisKeys
10	CKKS加密方案的编解码类	CKKSEncoder
11	表示明文矩阵的类	MatrixPlain
12	密钥生成类	KeyGenerator
13	加密类	Encryptor
14	解密类	Decryptor
15	运算库类	Evaluator

参数配置 CKKS

CKKSParametersLiteralDefault(DegreeType degreeType);	
参数	**DegreeType**：枚举类型，表示多项式的次数。

PoseidonContext(const ParametersLiteral& paramLiteral);	
参数	**ParametersLiteral**：参数申明类。
成员函数	**crt_context()**：获取 CRTContext 的指针。
描述	记录全局参数配置：原根，升降模参数

```
// Setting parameters for the CKKS encryption scheme  为CKKS加密方案设置参数。
CKKSParametersLiteralDefault ckks_param_literal(degree_4096);

//用户可通过ParametersLiteral自定义参数
 ParametersLiteral ckks_param_literal(CKKS, 15（degree），14（slot数量），logQ, logP, 35（缩放因子），192（汉明重量）);


// Instantiate the CKKS 上下文 参数对象。
PoseidonContext context(ckks_param_literal);
```

编解码 CKKS

CKKSEncoder encoder(const PoseidonContext& context);	
描述	CKKSEncoder是一个用于CKKS算法下对明文进行编解码的类。
参数	context：上下文类。
成员函数	int encode(vector<complex<double>> vec, Plaintext &plain, const mpf_class scaling_factor) ;:编码运算 。 int decode(const Plaintext &plain, vector<complex<double>>& vec) ;:解码运算。

```
//创建编解码对象
CKKSEncoder ckks_encoder(context);
```

```
//将 message 编码成对应的明文
ckks_encoder.encode(message,plainA,context.scaling_factor());
```

```
//将明文解密成 messagge
ckks_encoder.decode(plainRes,vec_result);
```

Encryptor(const PoseidonContext &context, const PublicKey &public_key, const SecretKey &secret_key); Decryptor(const PoseidonContext &context, const SecretKey &secret_key);	
描述	Encryptor是一个执行CKKS加密操作的类。
参数	context：const PoseidonContext & 类型，表示CKKS加密方案的上下文。 public_key： const PublicKey & 类型，表示加密使用到的公钥。 secret_key：const SecretKey & 类型，表示加密使用到的私钥。
成员函数	void encrypt(const Plaintext &plain, Ciphertext &destination) const; 对密文进行加密。

Decryptor(const PoseidonContext &context, const SecretKey &secret_key);	
描述	Decryptor是一个执行CKKS解密操作的类。
参数	context：const PoseidonContext & 类型，表示CKKS加密方案的上下文。 secret_key：const SecretKey & 类型，表示加密使用到的私钥。
成员函数	void decrypt(const Ciphertext &encrypted, Plaintext &destination): 对密文进行解密。

加解密 CKKS

```
//创建加解密对象
Encryptor enc(context,public_key,kgen.secret_key());
Decryptor dec(context,kgen.secret_key());
```

```
//对明文加密
enc.encrypt(plainA,cipherA);
```

```
//对密文解密
dec.decrypt(cipherRes,plainRes);
```

生成不同算法的评估函数——CKKS or BFV

EvaluatorFactory::DefaultFactory(const PoseidonContext &context)->create(context);	
描述	用于创建多项式评估类。
参数	context：上下文类。
返回值	ckks_eva: Evaluator类型，可调用密态操作API。

```
//创建CKKS Evaluator
auto ckks_eva = EnvaluatorFactory::DefaultFactory()->create(context);
```

Poseidon全同态计算库——核心API CKKS

CKKS	API	名称
1	密文与密文加法	Evaluator::add_ciph
2	密文与明文加法	Evaluator::add_plain
3	密文与密文减法	Evaluator::sub_ciph
4	密文与密文乘法	Evaluator::multiply
5	密文明文乘法	Evaluator::multiply_plain
6	重缩放	Evaluator::rescale
7	密文旋转	Evaluator::rotate
8	求共轭	Evaluator::conjugate
9	**多项式评估**	**Evaluator::evaluatePolyVector**
10	乘矩阵	Evaluator::multiplyByDiagMatrixBSGS
11	系数放入明文时隙	Evaluator::coeff_to_slot
12	明文时隙放入系数	Evaluator::slot_to_coeff
13	**自举**	**Evaluator::bootstrap**

Poseidon全同态计算库——核心数据结构 BFV

BFV	类型	名称
1	内存地址管理类	MemoryPool
2	**BFV加密方案的参数类**	**BFVParametersLiteralDefault**
3	上下文信息管理类	PoseidonContext
4	生成伪随机数的类	Blake2xbPRNGFactory
5	明文类	Plaintext
6	密文类	Ciphertext
7	公钥类	PublicKey
8	重现性化密钥类	RelinKeys
9	伽罗瓦密钥类	GaloisKeys
10	BFV加密方案的编解码类	BFVEncoder
11	表示明文矩阵的类	MatrixPlain
12	密钥生成类	KeyGenerator
13	加密类	Encryptor
14	解密类	Decryptor
15	**运算库类**	**EvaluatorFactory**

Poseidon全同态计算库——核心API BFV

BFV	API	名称
1	密文与密文加法	Evaluator::add_ciph
2	密文与明文加法	Evaluator::add_plain
3	密文与密文减法	Evaluator::sub
4	密文与密文乘法	Evaluator::multiply
5	密文明文乘法	Evaluator::multiply_plain
6	重缩放	Evaluator::rescale
7	密文行旋	Evaluator::rotate_row
8	密文列旋转	Evaluator::rotate_col

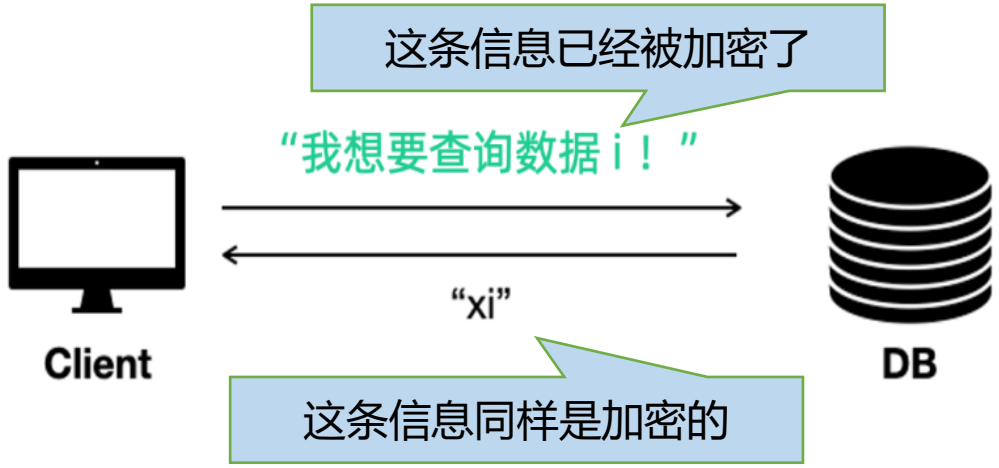
旋转 BFV

Void rotate_row (Ciphertext &encrypted,int rot_step, constGaloisKeys &galois_keys, Ciphertext &destination);	
参数	**encrypted**：Ciphertext对象引用，表示一个密文 **rot_step**：uint32_t类型，表示旋转的步长 **galois_keys**：GaloisKeys对象的常量引用，表示用于行旋转的加密密钥 **destination**：Ciphertext对象的引用，用于存储旋转后的密文
描述	函数用于对一个密文进行列旋转操作。

Void rotate_col (Ciphertext &encrypted, constGaloisKeys &galois_keys, Ciphertext &destination);	
参数	**encrypted**：Ciphertext对象引用，表示一个密文 **rot_step**：uint32_t类型，表示旋转的步长 **galois_keys**：GaloisKeys对象的常量引用，表示用于行旋转的加密密钥 **destination**：Ciphertext对象的引用，用于存储旋转后的密文
描述	函数用于对一个密文进行列旋转操作。

PIR应用简介

隐私信息检索（Private Information Retrieval），也称匿踪查询，是安全多方计算中非常实用的一门技术与应用，可以用来保护用户的查询隐私，进而也可以保护用户的查询结果。其目标是保证用户向数据源方提交查询请求时，在查询信息不被感知与泄漏的前提下完成查询。即对于数据源方来说，只知道有查询到来，但是不知道真正的查询条件、也就不知道对方查了什么。



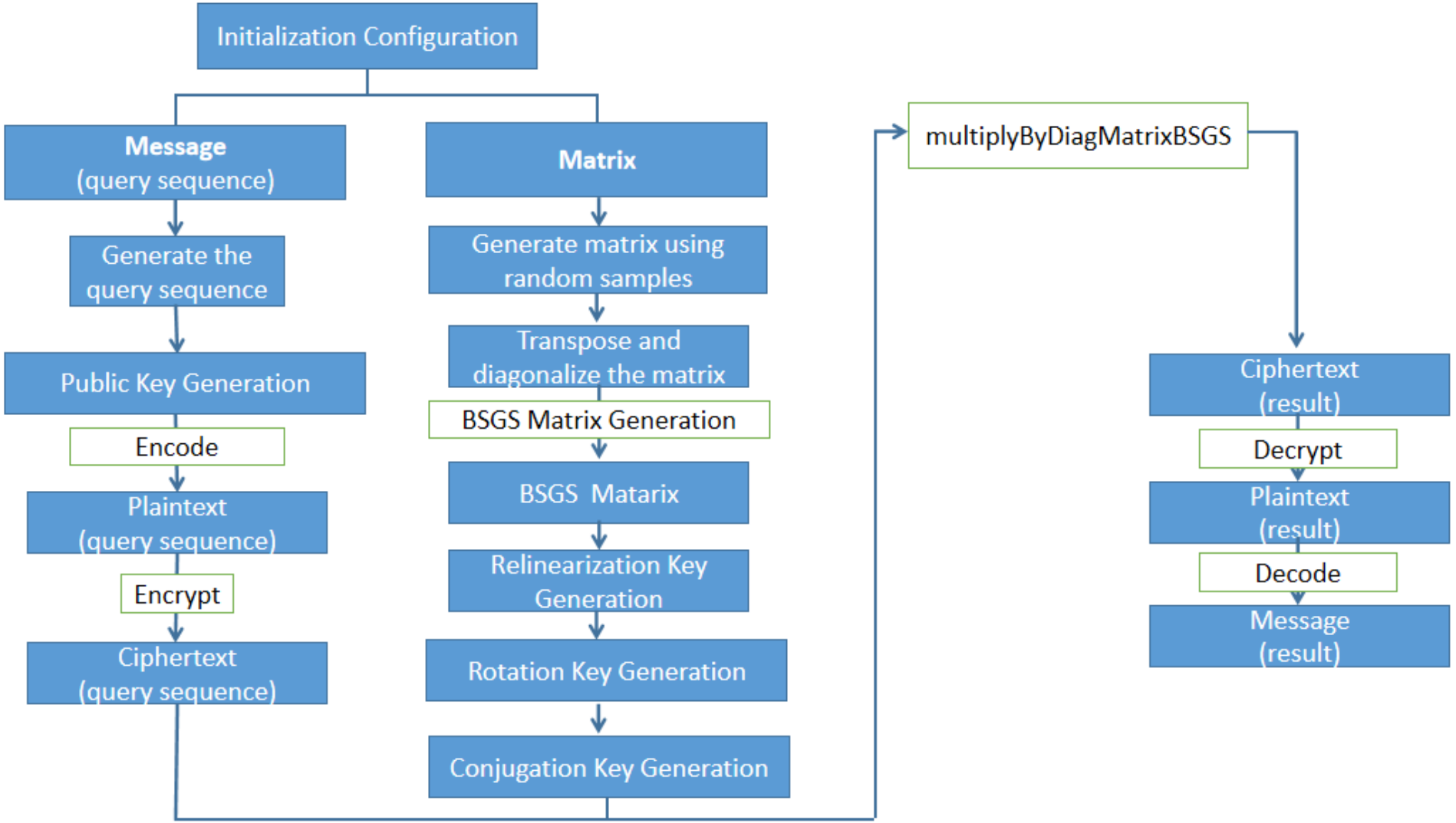
查询流程

1.数据方在数据库中存储的数据格式为<key, value>的格式；

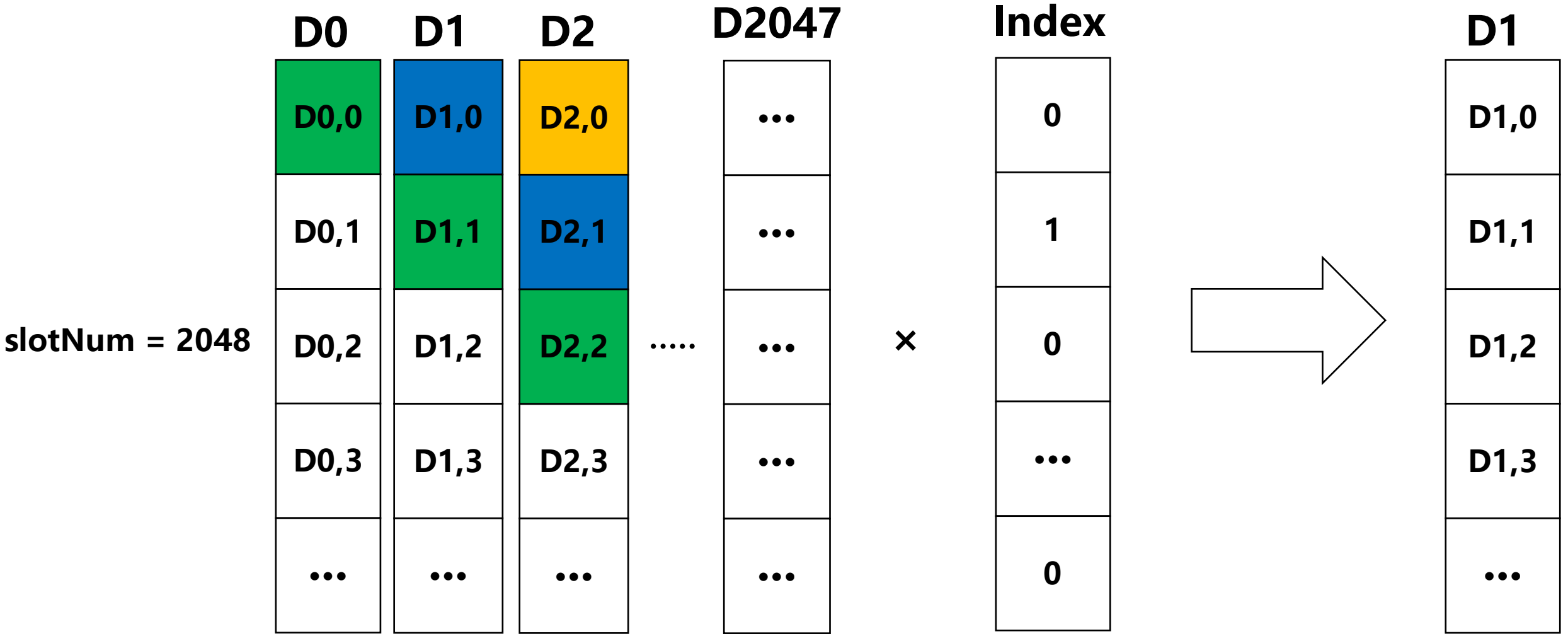
2.查询方使用其要查询的key，进行加密后去数据方查询对应的value；

3.在查询过程中数据方无法获知查询方的key具体是多少，也并不清楚最终发送了哪条value给了查询方。

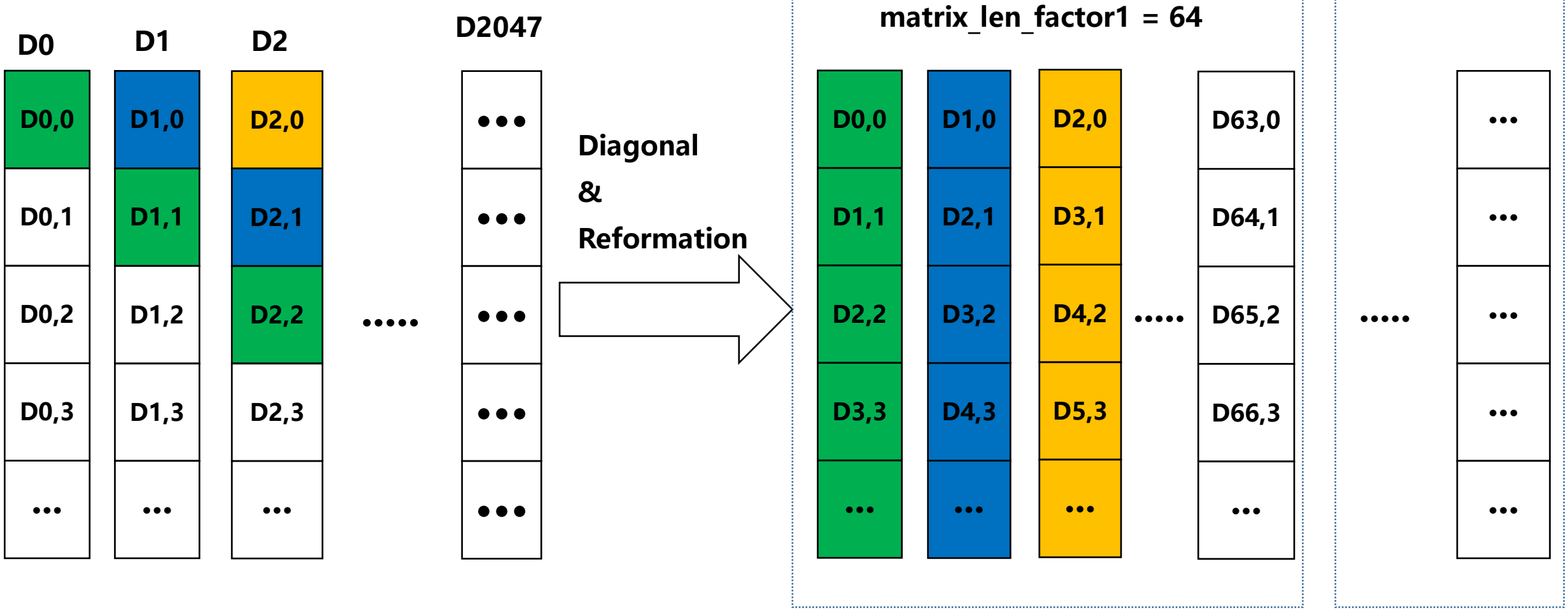
PIR应用流程图


Initialization Configuration
Message
(query sequence)
Generate the
query sequence
Public Key Generation
Encode
Plaintext
(query sequence)
Encrypt
Ciphertext
(query sequence)
Matrix
Generate matrix using
random samples
Transpose and
diagonalize the matrix
BSGS Matrix Generation
BSGS Matarix
Relinearization Key
Generation
Rotation Key Generation
Conjugation Key Generation
multiplyByDiagMatrixBSGS
Ciphertext
(result)
Decrypt
Plaintext
(result)
Decode
Message
(result)

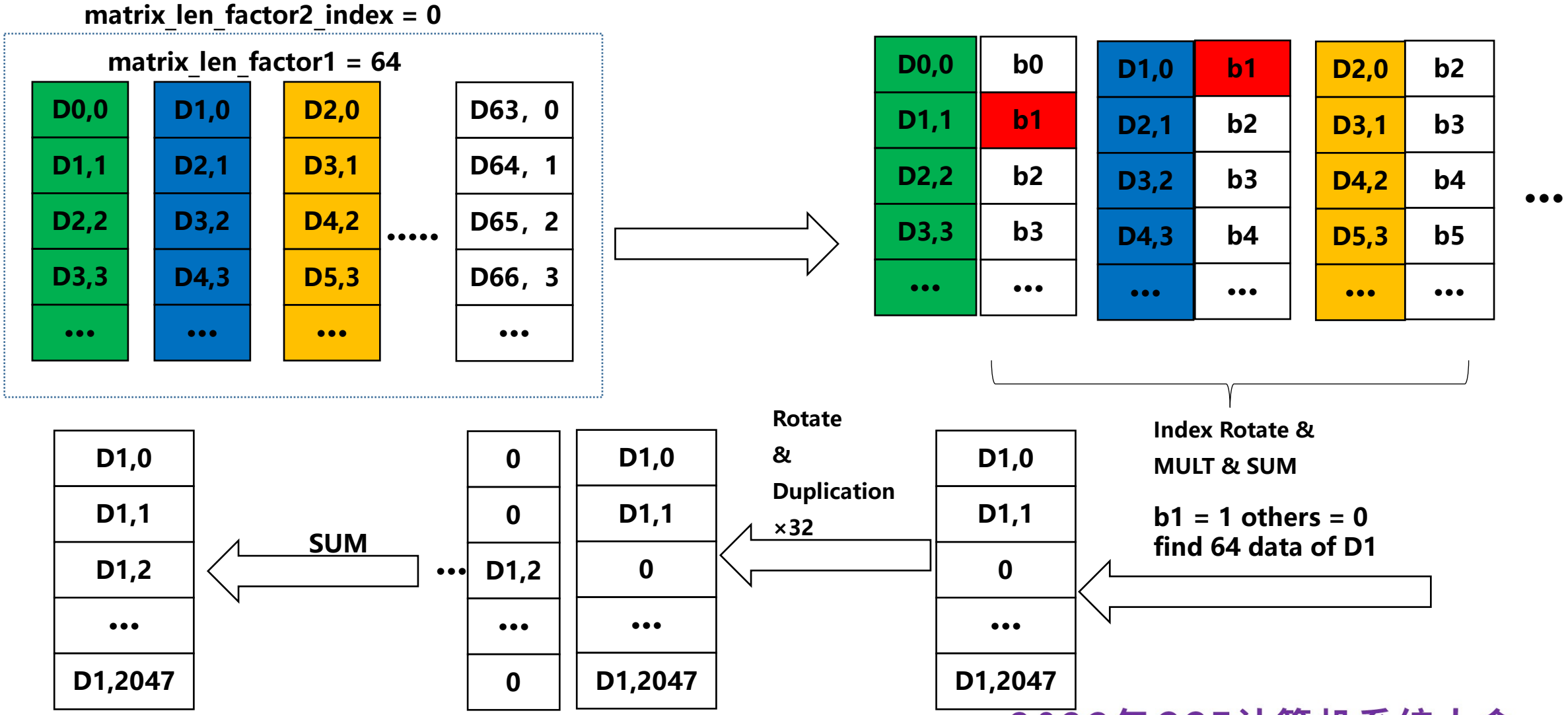
PIR矩阵构造方法


D0
D1
D2
D2047
Index
D1
slotNum = 2048
D0,0
D1,0
D2,0
D0,1
D1,1
D2,1
D0,2
D1,2
D2,2
D0,3
D1,3
D2,3
0
1
0
0
D1,0
D1,1
D1,2
D1,3

PIR矩阵构造方法


matrix_len_factor2 = 32
matrix_len_factor1 = 64
D0
D1
D2
D2047
D0,0
D0,1
D0,2
D0,3
D1,0
D1,1
D1,2
D1,3
D2,0
D2,1
D2,2
D2,3
Diagonal
&
Reformation
D3,1
D3,2
D3,3
D4,2
D4,3
D5,3
D63,0
D64,1
D65,2
D66,3

PIR矩阵构造方法


matrix_len_factor2_index = 0
matrix_len_factor1 = 64
D0,0
D1,1
D2,2
D3,3
D1,0
D2,1
D3,2
D4,3
D2,0
D3,1
D4,2
D5,3
D63，0
D64，1
D65，2
D66，3
b0
b1
b2
b3
b4
b5
Index Rotate &
MULT & SUM
b1 = 1 others = 0
find 64 data of D1
Rotate
&
Duplication
×32
SUM
D1,2047

PIR中对Poseidon库API的使用——multiplyByDiagMatrixBSGS

multiplyByDiagMatrixBSGS(Ciphertext &ciph, MatrixPlain& plain_mat,Ciphertext &result,const GaloisKeys &rot_keys);	
参数	**ciph**：密文对象。 **plain_mat**：矩阵向量明文数组的指针。 **result**：返回密文结果。 **rot_keys**：密文旋转执行switch_key所需的密钥。
描述	该函数实现密文和矩阵明文乘。

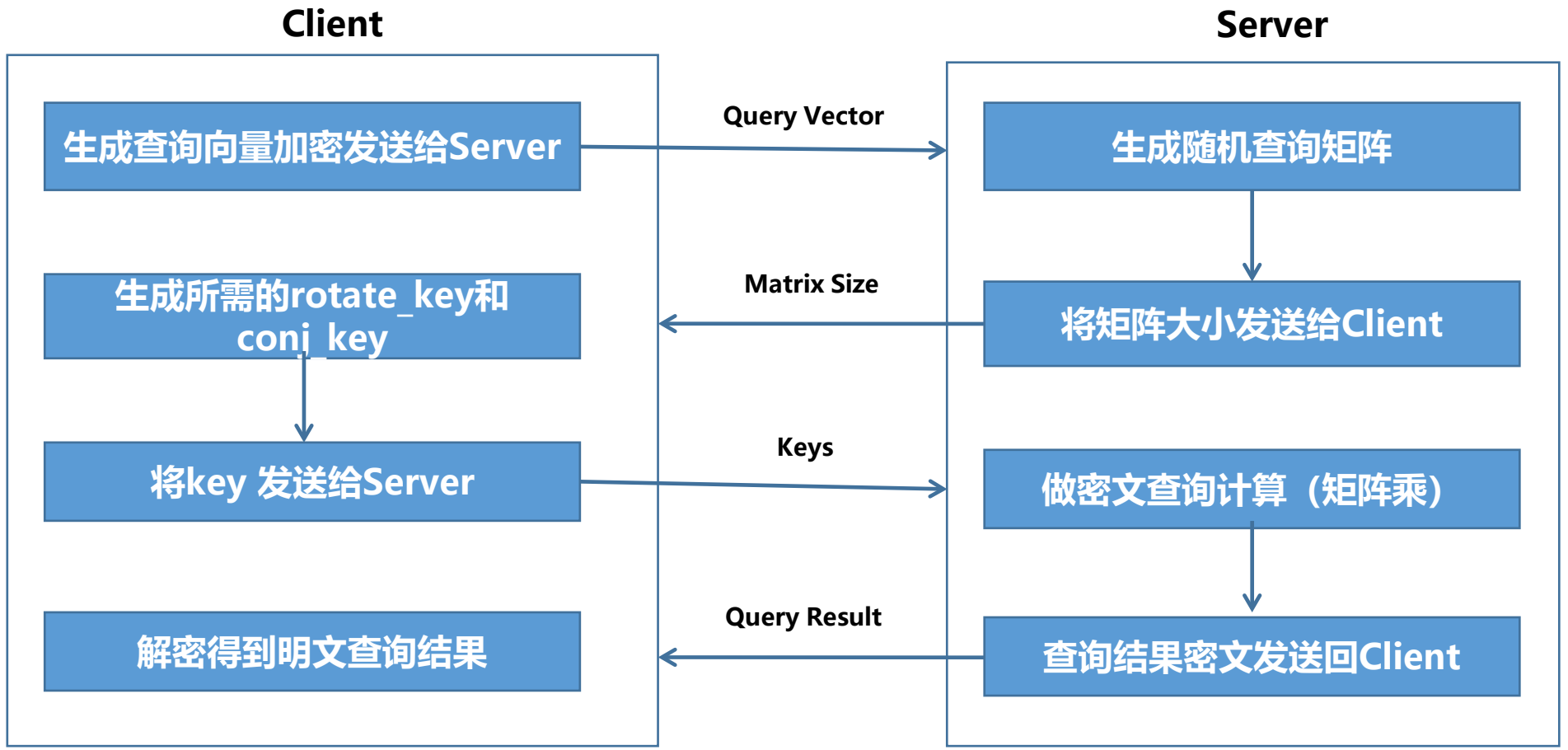
基于BFV的PIR应用

8组待查数据<key,value>，输入查询key值，查找出value值。

```
//密钥生成
KeyGenerator kgen(context);
kgen.create_public_key(public_key);
Encryptor enc(context,public_key,kgen.secret_key());
Decryptor dec(context,kgen.secret_key());

//对索引和查询数据编码
BFVEncoder encoder(context);
encoder.encode(data1,plainA);
encoder.encode(data2,plainB);
encoder.encode(data3,plainC);
encoder.encode(data4,plainD);
encoder.encode(data5,plainE);
encoder.encode(data6,plainF);
encoder.encode(data7,plainG);
encoder.encode(data8,plainH);

encoder.encode(index1,plainIndexA);
encoder.encode(index2,plainIndexB);
encoder.encode(index3,plainIndexC);
encoder.encode(index4,plainIndexD);
encoder.encode(index5,plainIndexE);
encoder.encode(index6,plainIndexF);
encoder.encode(index7,plainIndexG);
encoder.encode(index8,plainIndexH);

//将所有索引加密
enc.encrypt(plainIndexA,cipherIndexA);
enc.encrypt(plainIndexB,cipherIndexB);
enc.encrypt(plainIndexC,cipherIndexC);
enc.encrypt(plainIndexD,cipherIndexD);
enc.encrypt(plainIndexE,cipherIndexE);
enc.encrypt(plainIndexF,cipherIndexF);
enc.encrypt(plainIndexG,cipherIndexG);
enc.encrypt(plainIndexH,cipherIndexH);
```

```
//申明临时密文
Ciphertext ciph1,ciph2,ciph3,ciph4,ciph5,ciph6,ciph7,ciph8;

//将索引和查询数据进行明文乘操作
eva->multiply_plain(cipherIndexA,plainA,ciph1);
eva->multiply_plain(cipherIndexB,plainB,ciph2);
eva->multiply_plain(cipherIndexC,plainC,ciph3);
eva->multiply_plain(cipherIndexD,plainD,ciph4);
eva->multiply_plain(cipherIndexE,plainE,ciph5);
eva->multiply_plain(cipherIndexF,plainF,ciph6);
eva->multiply_plain(cipherIndexG,plainG,ciph7);
eva->multiply_plain(cipherIndexH,plainH,ciph8);

//对查询结果进行累加求和
eva->add(ciph1,ciph2,ciph2);
eva->add(ciph3,ciph4,ciph4);
eva->add(ciph5,ciph6,ciph6);
eva->add(ciph7,ciph8,ciph8);
eva->add(ciph2,ciph4,ciph2);
eva->add(ciph6,ciph8,ciph6);
eva->add(ciph2,ciph6,ciph2);

vector<uint32_t> res_data;
//将结果从硬件读回
eva->read(ciph2);

//解密
dec.decrypt(ciph2,plainRes);

cout << "============= decode ===============" << endl;
//解码以及结果打印
encoder.decode(plainRes,res_data);
for(int i = 0; i < 10; i++){
    printf("%c\n",(poseidon_byte)res_data[i]);
}
```

与PrimiHub集成完成PIR



医疗隐私计算——预测心血管疾病FHS

Framingham Heart Study（FHS）是1948年由美国国立卫生研究院资助的持续时间最长的心血管流行病学研究,旨在提高对美国冠心病流行病学的了解。FHS表明心血管疾病与年龄、血压、胆固醇、身高、体重等多种因素的关系紧密，因此可用这些健康相关的参数对心血管疾病的概率进行预测。

$x = 0.072 \cdot Age + 0.013 \cdot SBP - 0.029 \cdot DBP + 0.008 \cdot CHL - 0.053 \cdot height + 0.021 \cdot weight$

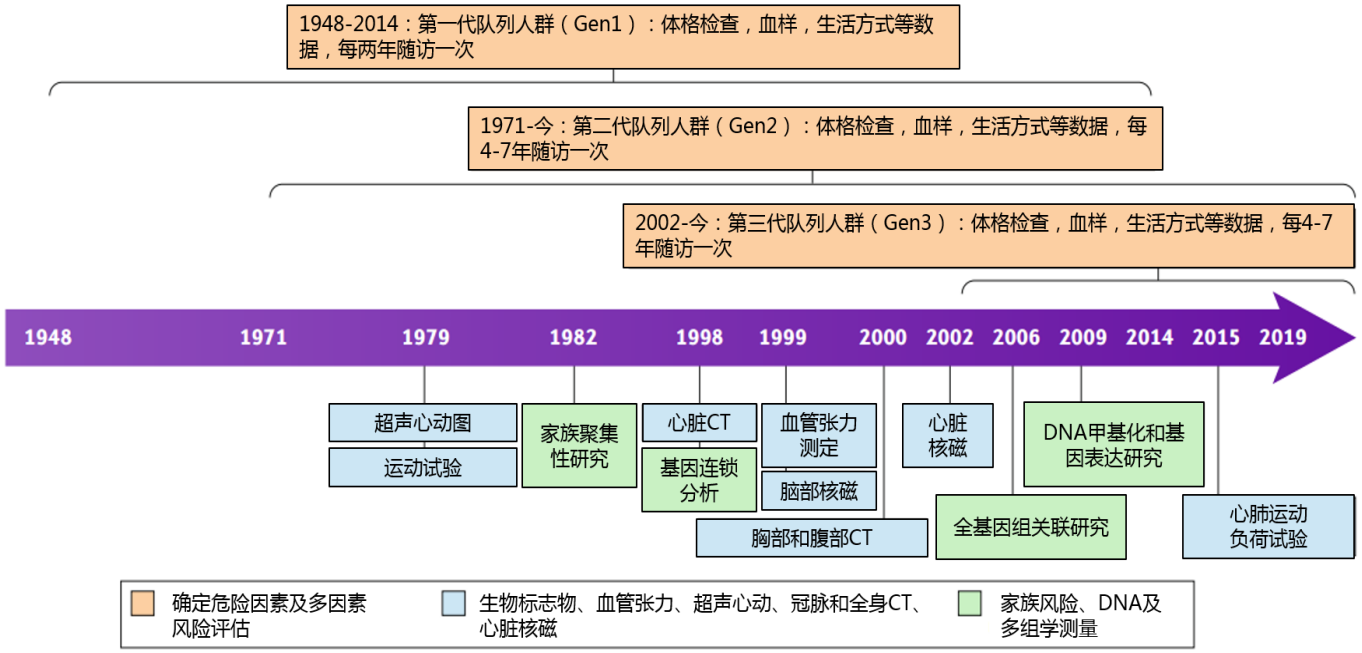
预测：$e^x/(1+e^x)$

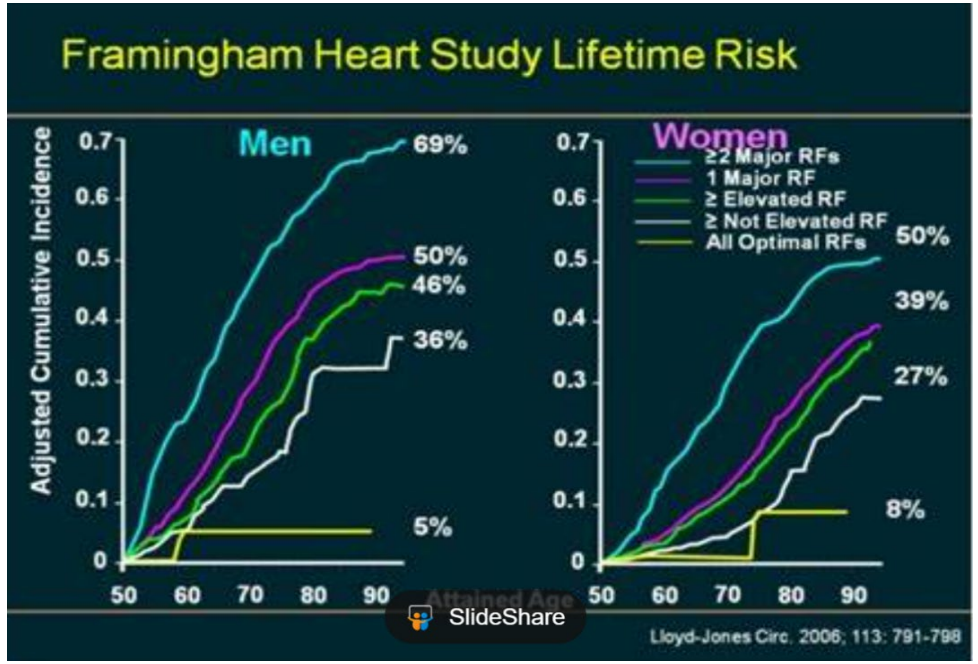
预测流程

1.用户输入自己的体征参数并进行加密

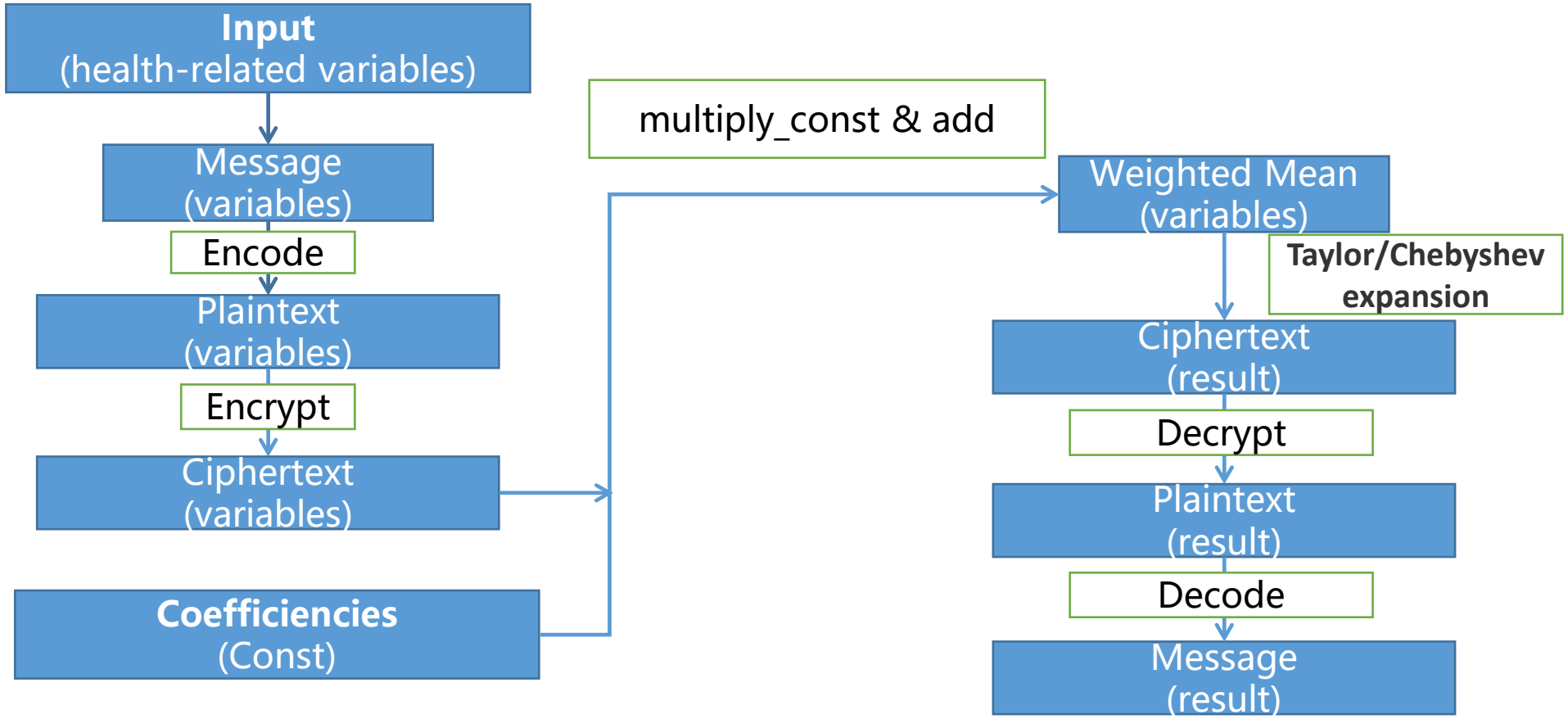
2.处理器对加密后的数据进行密态计算

3.用户对计算结果进行解密得到对自己患心血管疾病概率的预测





医疗隐私计算——预测心血管疾病FHS



对Poseidon库API的使用

void evaluatePolyVector(Ciphertext &ciph,Ciphertext &destination,const PolynomialVector &polys,mpf_class scalingfactor,const RelinKeys &relin_key,CKKSEncoder &encoder) = 0;

描述	Encryptor是一个执行CKKS加密操作的类。
参数	ciph：Ciphertext & 类型，密文操作数。 destination： Ciphertext & 类型，结果。 polys：const PolynomialVector & 类型，多项式组类型。 scalingfactor：mpf_class 类型，计算结束后目标放大倍数。 relin_key：const RelinKeys &类型，从线性化密钥。 encoder：CKKSEncoder ，编码类。

evaluatePolyVector

```
46
47    //输入健康医疗计算函数
48    double this_fun(double x)  {
49        return  exp(x) / (exp(x) + 1);
50    }
51
```

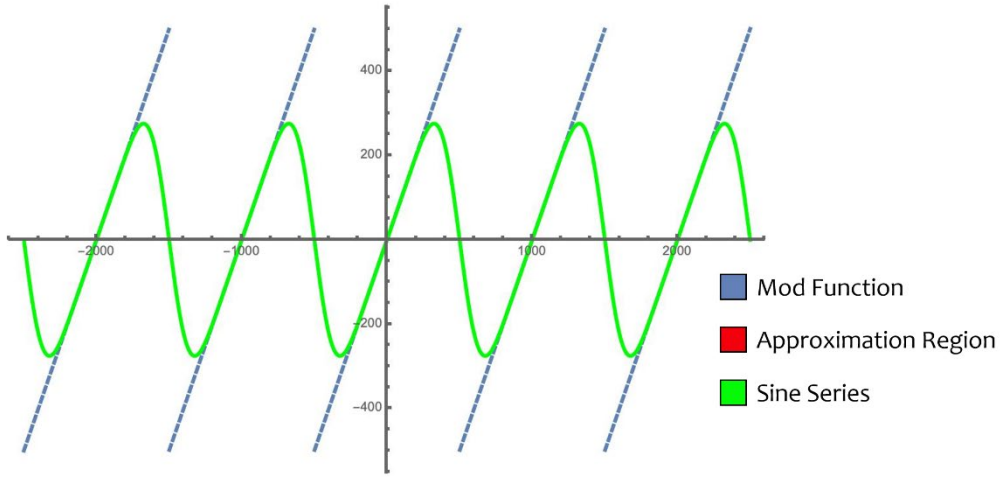
```
//输入切比雪夫上下限和阶数
auto a = -8.0;
auto b = 8.0;
auto deg = 64;

//使用切比雪夫拟合生成多项式，放入多向量组
auto approxF = util::Approximate(this_fun, a, b, deg);

//或者使用泰勒公式构造多项式
Polynomial  approxF(taylor_coeff,0,0,16,Monomial);
```

```
//slotIndex指定多项式组种，各多项式多项式评估的点
vector <Polynomial> poly_v{approxF};
PolynomialVector polys(poly_v,slotsIndex);
```

```
//计算e^x/(e^x+1)
ckks_eva->multiply_const(cipher_x,(2.0/(double)(b-a)),cipher_x);
ckks_eva->rescale(cipher_x);
ckks_eva->evaluatePolyVector(cipher_x,cipher_result,
polys,cipher_x.metaData()->getScalingFactor(),relinKeys,ckks_encoder);
```

自举方案

主要包含四步：扩模（ModRaise），同态编码（CoeffToSlot），同态取模（EvalMod），同态解码（SlotToCoeff）。

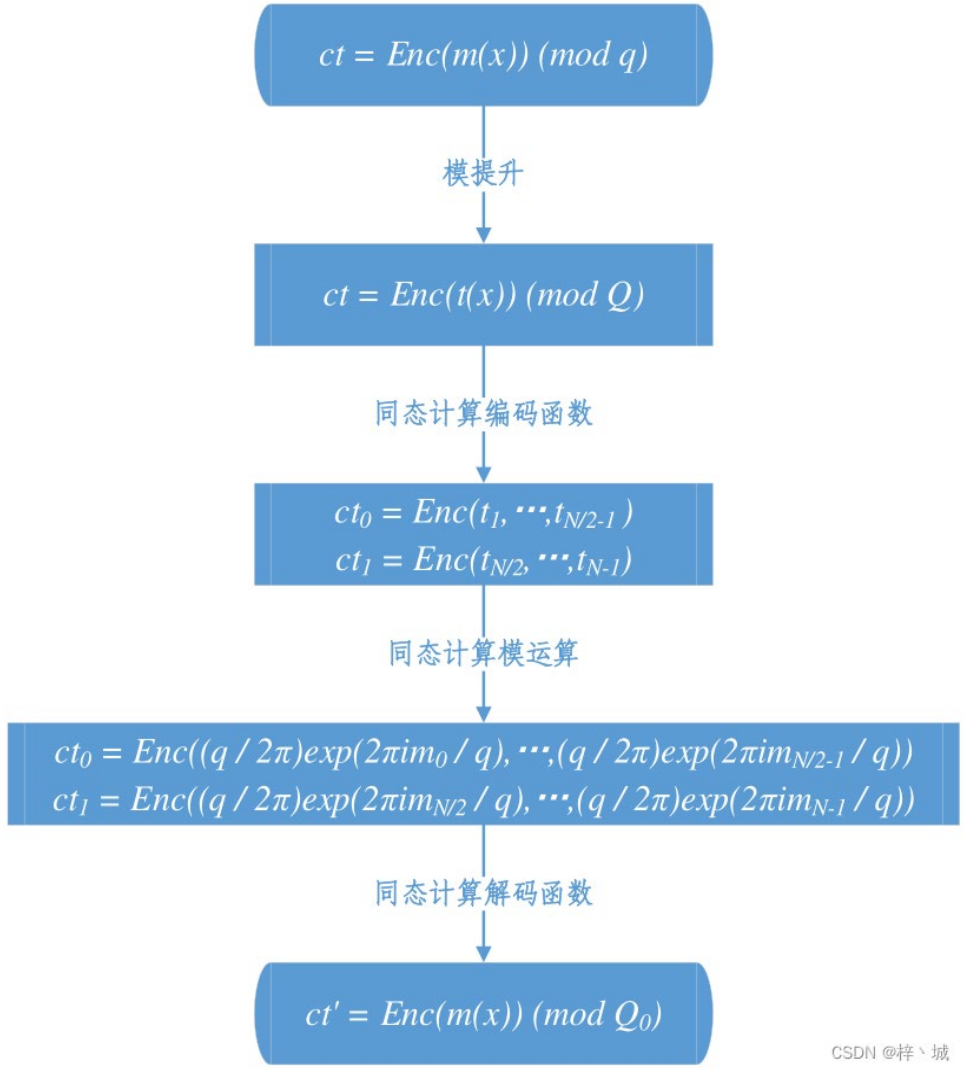


A value $t \in (-Kq, Kq)$ is given as an input of the decryption formula.

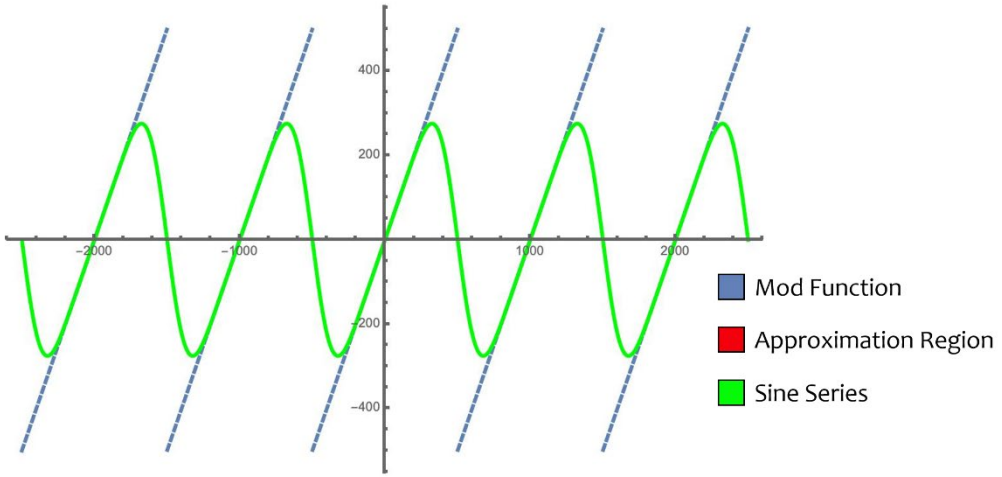
1. Consider the complex exponential function of $\exp\left(\frac{2\pi i t}{2^r \cdot q}\right)$ and compute its (scaled) Taylor expansion as

$$P_0(t) = \Delta \cdot \sum_{k=0}^{d_0} \frac{1}{k!} \left(\frac{2\pi i t}{2^r \cdot q}\right)^k$$

of degree $d_0 \geq 1$.
2. For $j = 0, 1, \ldots, r-1$, repeat the squaring $P_{j+1}(t) \leftarrow \Delta^{-1} \cdot (P_j(t))^2$.
3. Return $P_r(t)$.



自举方案（优化：切比雪夫）



Recall that in the bootstrapping procedure, we need to homomorphically evaluate

$$\frac{q}{2\pi}\sin\left(\frac{2\pi t}{q}\right).$$

with $t \in [-Kq, Kq]$. After a change of variables, we see that it suffices to evaluate

$$g(x) := \frac{1}{2\pi}\sin(2\pi K x)$$

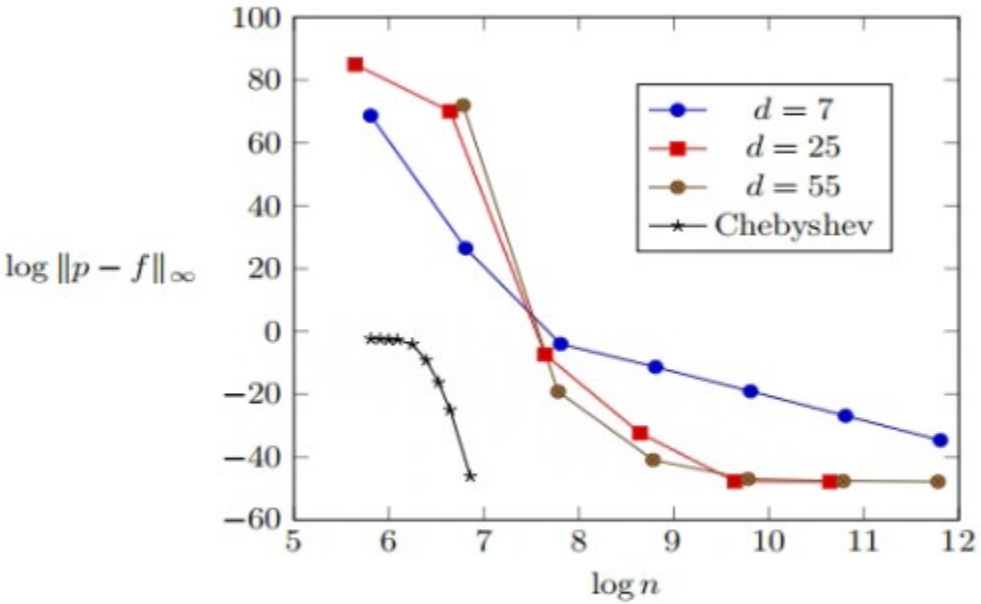


Fig. 2. Polynomial approximation errors to $\frac{1}{2\pi}\sin(2\pi K x)$ $(K = 12)$.

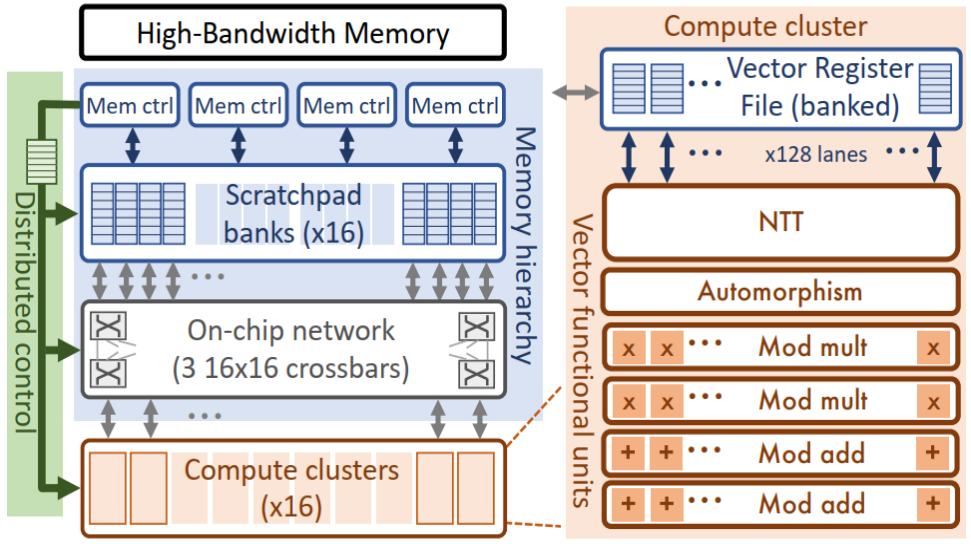
FHE加速器未来发展趋势

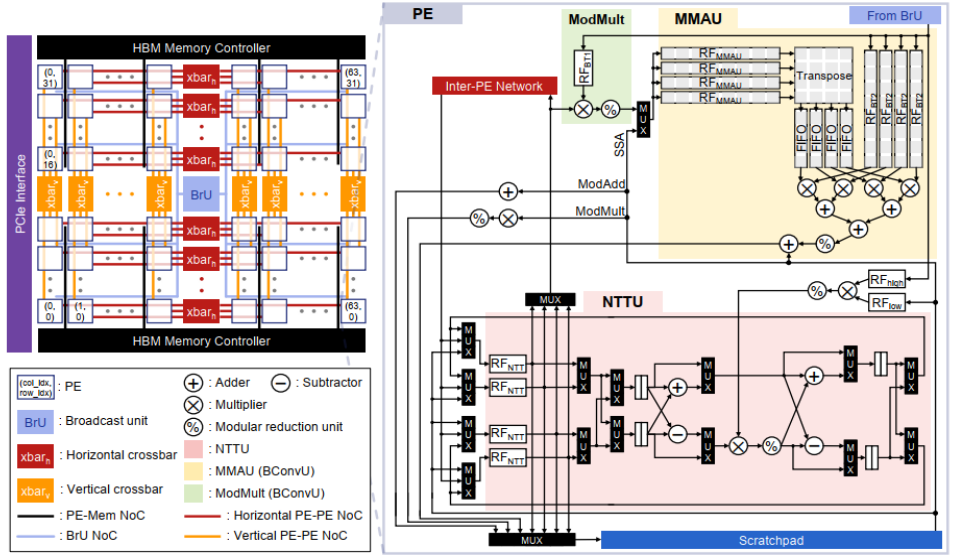
尚无实用的FHE ASIC

目前FHE相关加速工作主要是使用GPU或者专用加速硬件平台即FPGA和ASIC进行加速。

1. 基于GPU的FHE加速器能够较为直接和方便的设计和使用，但是其计算单元功能固定，不能很好地与FHE相关算子很好地匹配，因此实际的加速效果不够理想。目前已有的GPU加速器工作有over100x和tensorfhe。

2. 基于ASIC的FHE加速器能够通过分配超大片上缓存（高达512MB）以及计算资源来解决FHE应用的高带宽需求和高计算复杂度的问题，但这种芯片投入生产和使用困难度高，实际使用价值较低。目前已有ASIC加速器设计有：F1,Craterlake,BTS,ARK。

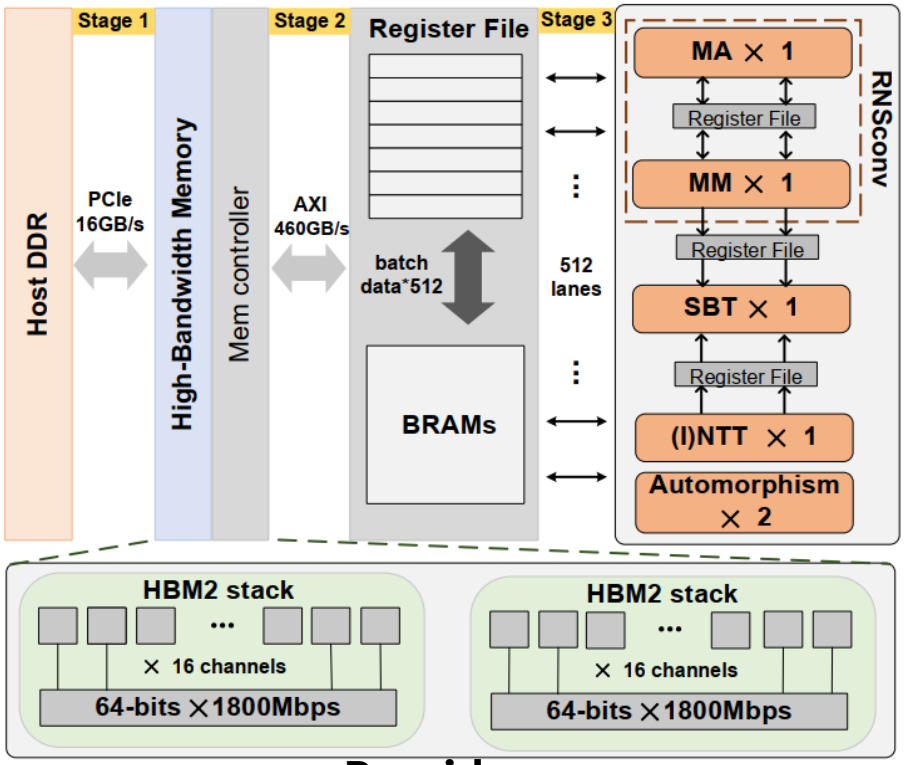




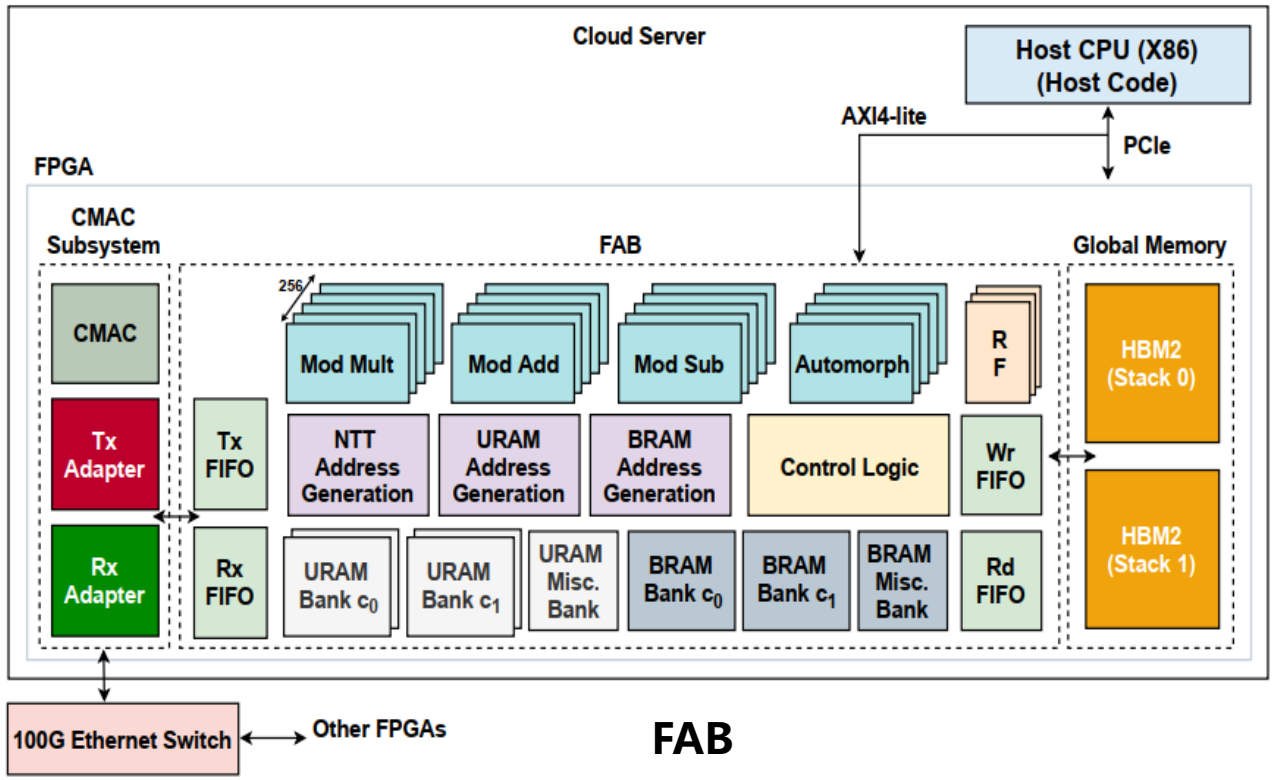
FHE加速器未来发展趋势

FPGA的资源有限，FHE要求访存理论带宽是~3TB/s

基于FPGA的FHE加速器兼具GPU和ASIC的优点，能够定制化设计计算电路以及能够较快的实际使用，但是其缺点是带宽和计算资源有限，即使使用目前最高端的FPGA器件也很难达到理想的加速需求。目前已有的FPGA加速器设计有：Poseidon和FAB。



Poseidon



FAB

谢 谢!

路 航

张江壹号
HOMOMORPHIC PROCESSING UNIT