基于BGW协议的安全多方计算框架

电子科技大学毕业答辩

答辩人 ：刘冯润

指导老师：陈大江

实习企业：上海期智研究院

企业导师：郁昱

目录

研究背景

上海期智研究院是上海市科学技术委员会所属事业单位，由图灵奖得主、中科院院士姚期智先生牵头组建。

安全多方计算

- [82]姚氏百万富翁问题：两个百万富翁想比较到底谁更富有，但都不想让别人知道自己有多少钱，在没有可信第三方的情况下如何进行？ 提出了安全多方计算的概念
- [86]姚期智提出姚氏混淆电路(Garbled Circuits) 安全两方计算协议
- [86] Goldreich, Micali和Wigderson提出基于布尔电路的GMW协议 安全多方计算协议
- [87] Ben-Or, Goldwasser和Widerson提出基于算术电路的BGW协议 安全多方计算协议

关键词：安全多方计算；BGW协议；秘密共享

安全模型：模拟敌手行为，对安全协议发起挑战

参与方类型

- 诚实参与方：严格按照协议流程执行
- 半诚实参与方：在协议执行过程记录信息，尝试推导其他隐私信息
- 恶意参与方：按照自己的意愿执行，从而破坏其他参与方的结果

} 腐败方(corrupted)：被敌手控制的参与方。

安全设置：诚实方多数（Honest-Majority)

- 半诚实模型：腐败方都是半诚实参与方
- 恶意模型(with abort)：腐败方中包含恶意参与方

隐私安全性

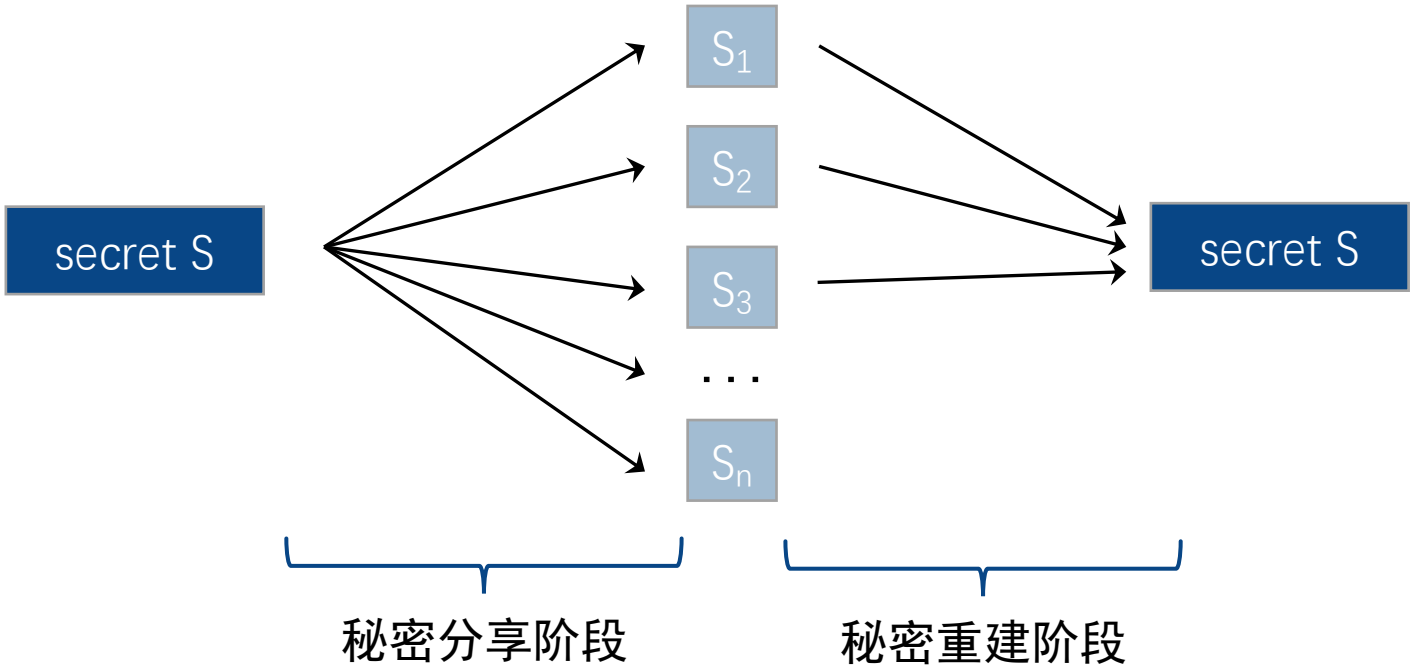
计算结果正确性

↓ 安全性更高

秘密共享

Shamir共享机制(n, t)：

- 秘密分享：通过**t次随机多项式$f(\cdot)$**将秘密S划分为n个子秘密，发送给n个参与者。
- 秘密重建：任意$k \geq t+1$个参与者都可以通过**拉格朗日插值函数**恢复出秘密S。



安全性：
任意t个或小于t个子秘密都不能泄漏秘密S的任何信息

秘密共享

$[S]_t$：分享
通过Shamir分享机制分享的子秘密集合
t：分享的度

Shamir共享机制(n, t)：

- 秘密分享：通过**t次随机多项式$f(\cdot)$**将秘密S划分为n个子秘密，发送给n个参与者。
- 秘密重建：任意$k \geq t+1$个参与者都可以通过**拉格朗日插值函数**恢复出秘密S。

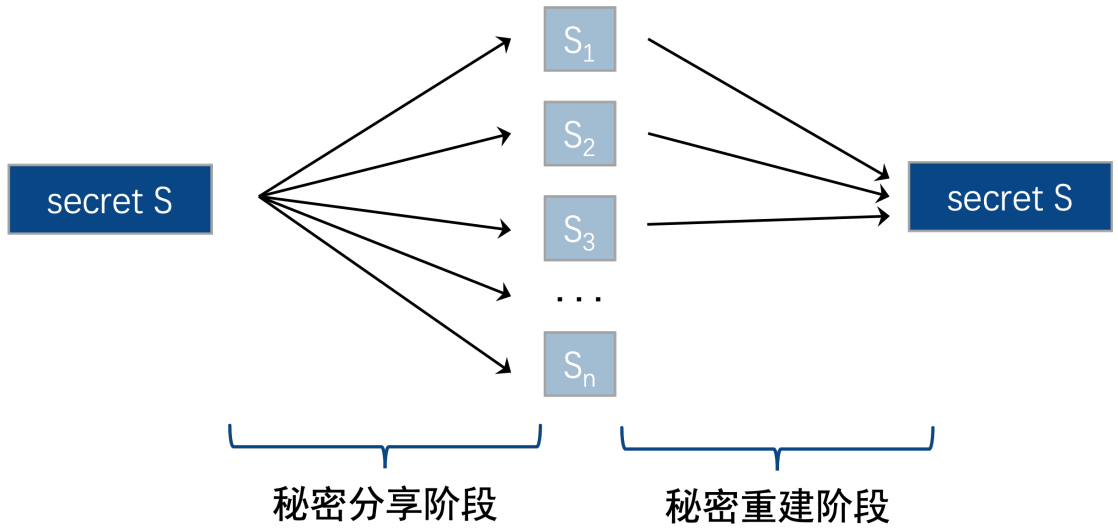
分享的性质：$[x]_t, [y]_t$

- 线性同态：$[\alpha x + \beta y]_t = \alpha[x]_t + \beta[y]_t$
- 乘法结果：$[x]_t \cdot [y]_t = [xy]_{2t}$

BGW协议：乘法降阶

$$[xy]_{2t} \rightarrow [xy]_t$$

- 本系统：使用**随机分享对**$([r]_t, [r]_{2t})$



需求分析

安全多方计算协议模块

- 半诚实模型：
  - 需求：如何实现**乘法降阶**$[xy]_{2t} \rightarrow [xy]_t$？
  - 技术：**随机分享对**$([r]_t, [r]_{2t})$
- 恶意模型(with abort)：
  - 需求：在输出阶段之前，如何**高效验证计算结果的正确性**？
  - 技术：**批量验证技术**

网络通信模块*

- 需求：实现n个参与方的任意通信
- 技术：TCP连接通信网络

有限域操作模块*

- 需求：高效实现有限域中元素的操作
- 技术：使用梅森素数下的有限域

安全多方计算协议模块：12个基础协议

Shamir秘密共享：

- 协议1：秘密分享协议(Secret Sharing)
- 协议2：秘密重建协议(Secret Open)

半诚实模型：乘法降阶

- 协议3：生成随机分享对(DoubleRandom)
- 协议4：生成随机分享(Random)
- 协议5：乘法协议(Multipliation)
- 协议6：向量乘法协议(Extend-Multipliation)

恶意模型：高效验证乘法元组

- 协议7：投掷硬币协议(Coin)
- 协议8：压缩乘法元组协议(Compress)
- 协议9：压缩向量乘法元组协议(Extend-Compress)
- 协议10：去线形化协议(De-Linearization)
- 协议11：降维协议(Dim-Reduction)
- 协议12：随机化验证协议(Randomization)

半诚实模型：生成随机分享$[r]_t$

$[r]_t$：由所有参与方共同随机生成的分享

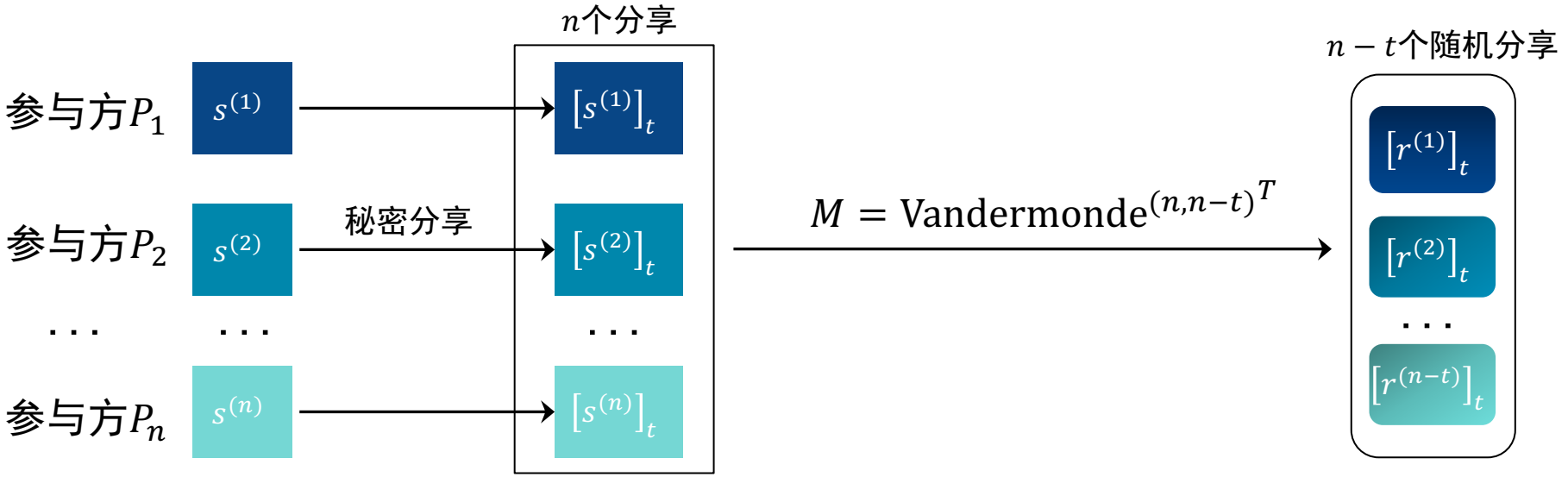
1. n个参与方分别取样一个**随机数**$s^{(i)}$
2. 依次执行**秘密分享**协议，发送分享$[s^{(i)}]_t$
3. 通过**范德蒙德矩阵**，生成$n-t$个随机分享

协议4: (Random) 生成$(n-t)$对随机秘密

1. 每一个参与方P_i随机取样一个秘密$s^{(i)}$，生成分享$[s^{(i)}]_t$，发送给其他参与者。
2. 每一个参与方计算：（其中$M = \mathrm{Van}^{(n,n-t)^T}$）

$$([r_1]_t, [r_2]_t, \cdots, [r_{n-t}]_t) = M([s^{(1)}]_t, [s^{(2)}]_t, \cdots, [s^{(n)}]_t)$$

输出：$[r_1]_t, [r_2]_t, \cdots, [r_{n-t}]_t$



协议4：生成随机分享(Random)

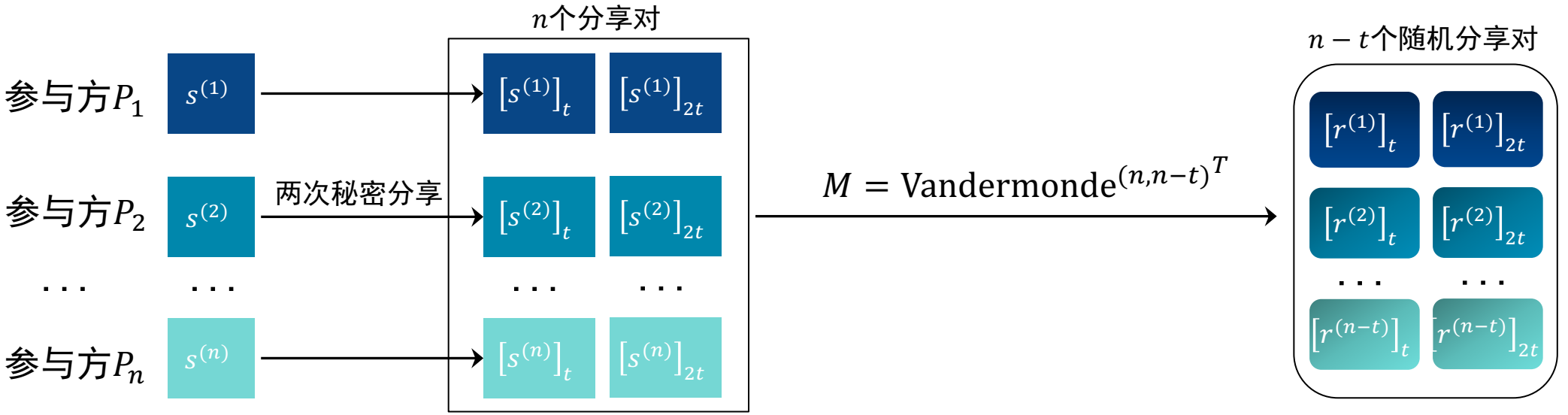
半诚实模型：生成随机分享对$([r]_t, [r]_{2t})$

$([r]_t, [r]_{2t})$：秘密值相同，但度不同

1. n个参与方分别取样一个随机数$s^{(i)}$
2. 依次执行**两次秘密分享**协议，发送**分享对**$([s^{(i)}]_t, [s^{(i)}]_{2t})$
3. 通过范德蒙德矩阵，生成$n-t$个**随机分享对**

协议3: (DoubleRandom) 生成$(n-t)$对随机秘密的分享

1. 每一个参与方P_i随机取样一个秘密$s^{(i)}$，生成一对分享$([s^{(i)}]_t, [s^{(i)}]_{2t})$，并分享给其他参与者。
2. 每一个参与方计算：（其中$M = \mathrm{Van}^{(n,n-t)^{\mathrm{T}}}$）
$$([r_1]_t, [r_2]_t, \ldots, [r_{n-t}]_t) = M([s^{(1)}]_t, [s^{(2)}]_t, \ldots, [s^{(n)}]_t)$$
$$([r_1]_{2t}, [r_2]_{2t}, \ldots, [r_{n-t}]_{2t}) = M([s^{(1)}]_{2t}, [s^{(2)}]_{2t}, \ldots, [s^{(n)}]_{2t})$$
输出：$([r^{(1)}]_t, [r^{(1)}]_{2t}), ([r^{(2)}]_t, [r^{(2)}]_{2t}), \cdots, ([r^{(n-t)}]_t, [r^{(n-t)}]_{2t})$



协议3：生成随机分享对(DoubleRandom)

半诚实模型：乘法运算

乘法降阶：使用随机分享对$([r]_t, [r]_{2t})$

- 乘法协议：$[xy]_{2t} \rightarrow [xy]_t$
- 向量内积协议：$[x \odot y]_{2t} \rightarrow [x \odot y]_t$

$[x]_t \times [y]_t = [xy]_{2t}$

协议5：乘法协议(Multiplication)

半诚实模型：乘法运算

乘法降阶：使用随机分享对$([r]_t, [r]_{2t})$

- 乘法协议：$[xy]_{2t} \rightarrow [xy]_t$
- 向量内积协议：$[\boldsymbol{x} \odot \boldsymbol{y}]_{2t} \rightarrow [\boldsymbol{x} \odot \boldsymbol{y}]_t$

协议5: (Multiplication) 乘法

1. 全体参与方选出一个特殊的参与方P_{king}作为主要计算方，用来接受（或发送给）各个参与方的分享。用$[x]_t, [y]_t$来表示乘法的输入分享。
2. 全体参与方执行DoubleRandom协议，生成一对随机秘密分享$([r]_t, [r]_{2t})$.
3. 全体参与方本地计算出$[x \cdot y + r]_{2t}$的值：
$$[x \cdot y + r]_{2t} = [x]_t \cdot [y]_t + [r]_{2t}$$
4. P_{king}收集所有参与方的分享$[x \cdot y + r]_{2t}$，恢复出秘密$x \cdot y + r$，然后执行SecretShare协议生成分享$[x \cdot y + r]_t$，发送给其他参与方。
5. 全体参与方本地计算：
$$[x \cdot y]_t = [x \cdot y + r]_t - [r]_t$$
输出：$[x \cdot y]_t$

随机分享对

$[x]_t \times [y]_t + [r]_{2t} = [xy + r]_{2t}$ → 秘密重建 → $xy + r$

$xy + r \quad - [r]_t = [xy]_t$

协议5：乘法协议(Multiplication)

安全性：
$xy + r$不会泄漏关于乘法结果xy的任何信息！

半诚实模型：向量内积运算

乘法降阶：使用随机分享对$([r]_t, [r]_{2t})$

- 乘法协议：$[xy]_{2t} \rightarrow [xy]_t$
- 向量内积协议：$[\boldsymbol{x}\odot\boldsymbol{y}]_{2t} \rightarrow [\boldsymbol{x}\odot\boldsymbol{y}]_t$

协议6: (Extend-Multiplication) 扩展乘法

1. 全体参与方选出一个特殊的参与方P_{king}作为主要计算方，用来接受（或发送给）各个参与方的分享。用$[\boldsymbol{x}]_t, [\boldsymbol{y}]_t$来表示乘法的输入向量分享。
2. 全体参与方执行DoubleRandom协议，生成一对随机秘密分享$([r]_t, [r]_{2t})$。
3. 全体参与方本地计算出$[\boldsymbol{x}\odot\boldsymbol{y}+r]_{2t}$的值：
$$[\boldsymbol{x}\odot\boldsymbol{y}+r]_{2t} = [\boldsymbol{x}]_t \odot [\boldsymbol{y}]_t + [r]_{2t}$$
4. P_{king}收集所有参与方的分享$[\boldsymbol{x}\odot\boldsymbol{y}+r]_{2t}$，恢复出秘密$\boldsymbol{x}\odot\boldsymbol{y}+r$，然后执行SecretShare协议生成分享$[\boldsymbol{x}\odot\boldsymbol{y}+r]_t$，发送给其他参与方。
5. 全体参与方本地计算：
$$[\boldsymbol{x}\odot\boldsymbol{y}]_t = [\boldsymbol{x}\odot\boldsymbol{y}+r]_{2t} - [r]_t$$
输出：$[\boldsymbol{x}\odot\boldsymbol{y}]_t$

$[x]_t \odot [y]_t = [x\odot y]_{2t}$

随机分享对

$[x]_t \odot [y]_t + [r]_{2t} = [x\odot y + r]_{2t}$ → 秘密重建 → $x\odot y + r$

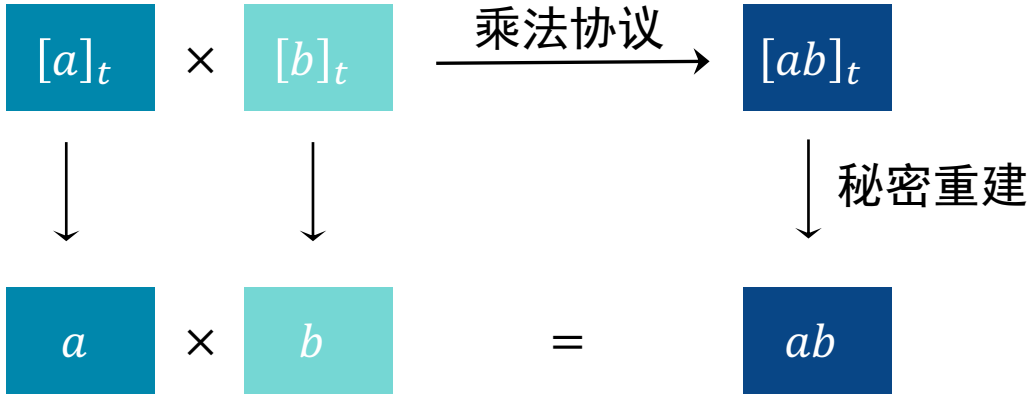
$x\odot y + r - [r]_t = [x\odot y]_t$

协议6：向量内积协议(Extend-Multiplication)

恶意模型：验证乘法结果的正确性

验证乘法元组$[a]_t \times [b]_t \rightarrow [ab]_t$

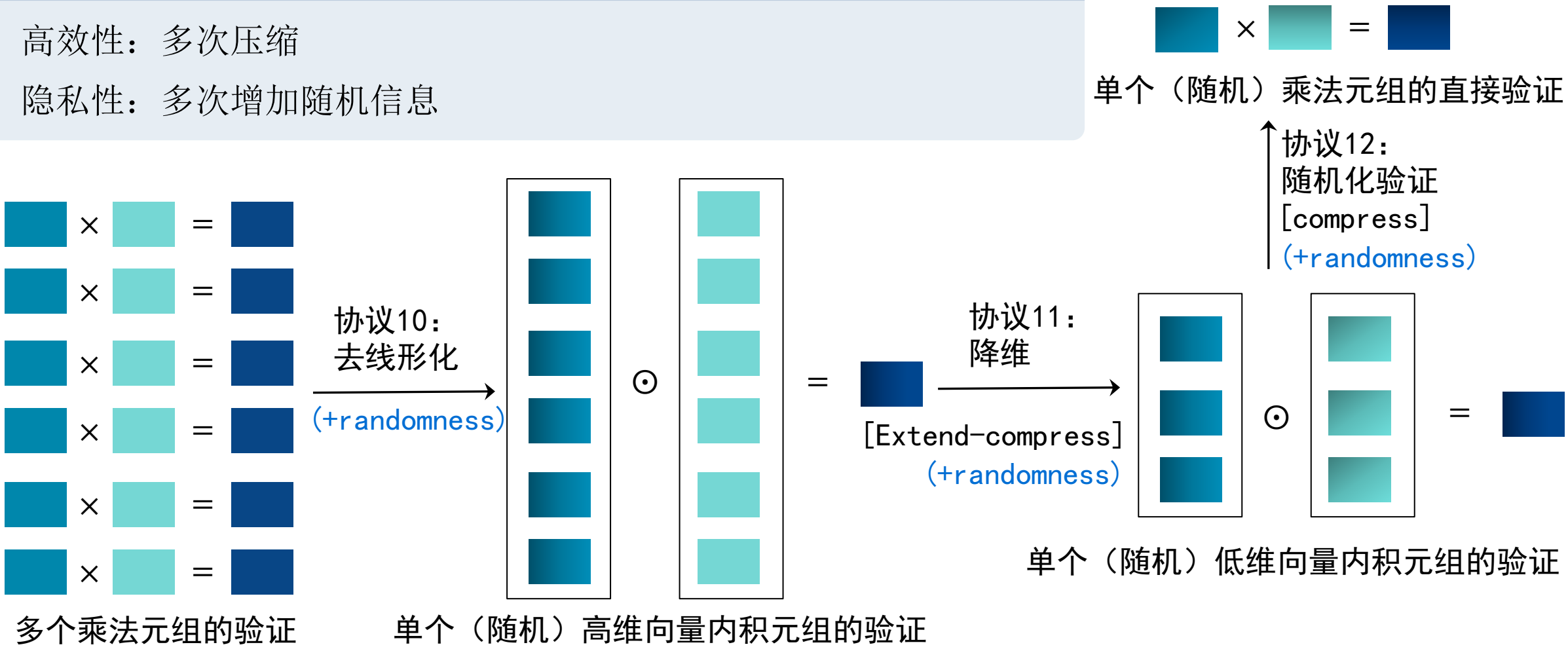
- 泄漏隐私信息
- 效率低下



恶意模型：验证乘法结果的正确性

把多个乘法元组的验证压缩为单个随机元组的直接验证

- 高效性：多次压缩
- 隐私性：多次增加随机信息



框架实现

系统模块

- 有限域操作模块
- 网络通信模块
- 安全多方计算协议模块
- …

```
❯ tree -L 1
.
├── Math                #有限域操作模块
├── Networking          #网络模块
├── Protocols           #安全多方计算协议模块
├── Tools
├── Test                #安全多方计算程序
├── CONFIG
├── Makefile
├── HOSTS.example       #网络地址设置文件
└── Player-Data         #参与方输入输出数据
```

C++语言实现，Makefile实现自动化编译

框架可以支持在诚实方多数情况下：
任意多个参与方计算任意函数

代码1：安全多方计算程序结构

```
1   // Usage: ./test_mpc.x ID
2   int main(int argc, char** argv)
3   {
4       // 1. 设置网络地址
5       Names player_name = Names(player_no, portnum_base, filename);
6       // 2. 建立TCP通信网络
7       PlainPlayer P(player_name, "");
8
9       // 3. 设置安全参数(t, n)等
10      init_share(&P, threshold, P.num_players());
11      init_input_file(input_file);
12
13      // 4. 算术电路（任意计算函数）
14      Share a, b, c;
15      ShareVector A, B, C;
16      // a.输入阶段
17      // b.计算阶段
18      // c.验证阶段（恶意模型）
19      // c.输出阶段
20      return 0;
21  }
```

安全多方计算协议模块：Share类

Class RandomShare: Share	
queue_random	存储**随机分享**的队列
random()	协议4：生成n-t个随机分享

Class DoubleRandomShare: DoubleShare	
queue_double_random	存储**随机分享对**的队列
double_random()	协议3：生成n-t个随机分享对

随机分享$[r]_t$

Class DoubleShare: Share	
sharings	一对分享值

随机分享对$([r]_t, [r]_{2t})$

Class Share	
sharing	分享值
secret	秘密值
degree	分享的度t

基类
表示一般分享$[a]_t$

把计算函数表示为算术电路：

- 输入阶段：input_from_...(P_i)
- 计算阶段：+, −, ×, ...
- 验证阶段（恶意模型）
- 输出阶段：reveal...(P_i)

安全多方计算协议模块：ShareVector类

把计算函数表示为算术电路：

- 输入阶段：input_from_...(P_i)
- 计算阶段：+, −,×, ...
- 验证阶段（恶意模型）
- 输出阶段：reveal...(P_i)

通过vector模版扩展到向量类

分享对

Class DoubleShare: Share	
sharings	一对分享值

分享对向量

DoubleShareVector:
vector<DoubleShare>

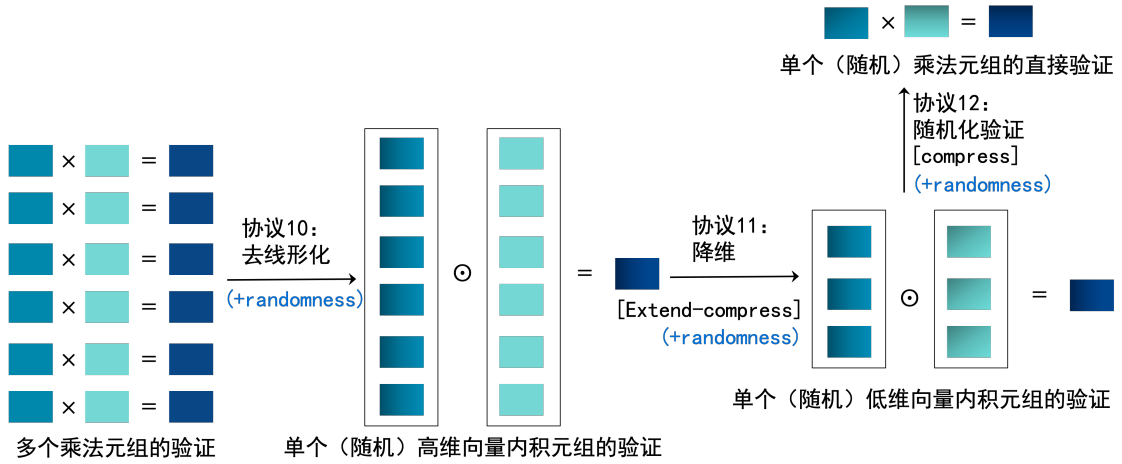
分享

Class Share	
sharing	分享值
secret	秘密值
degree	分享的度t

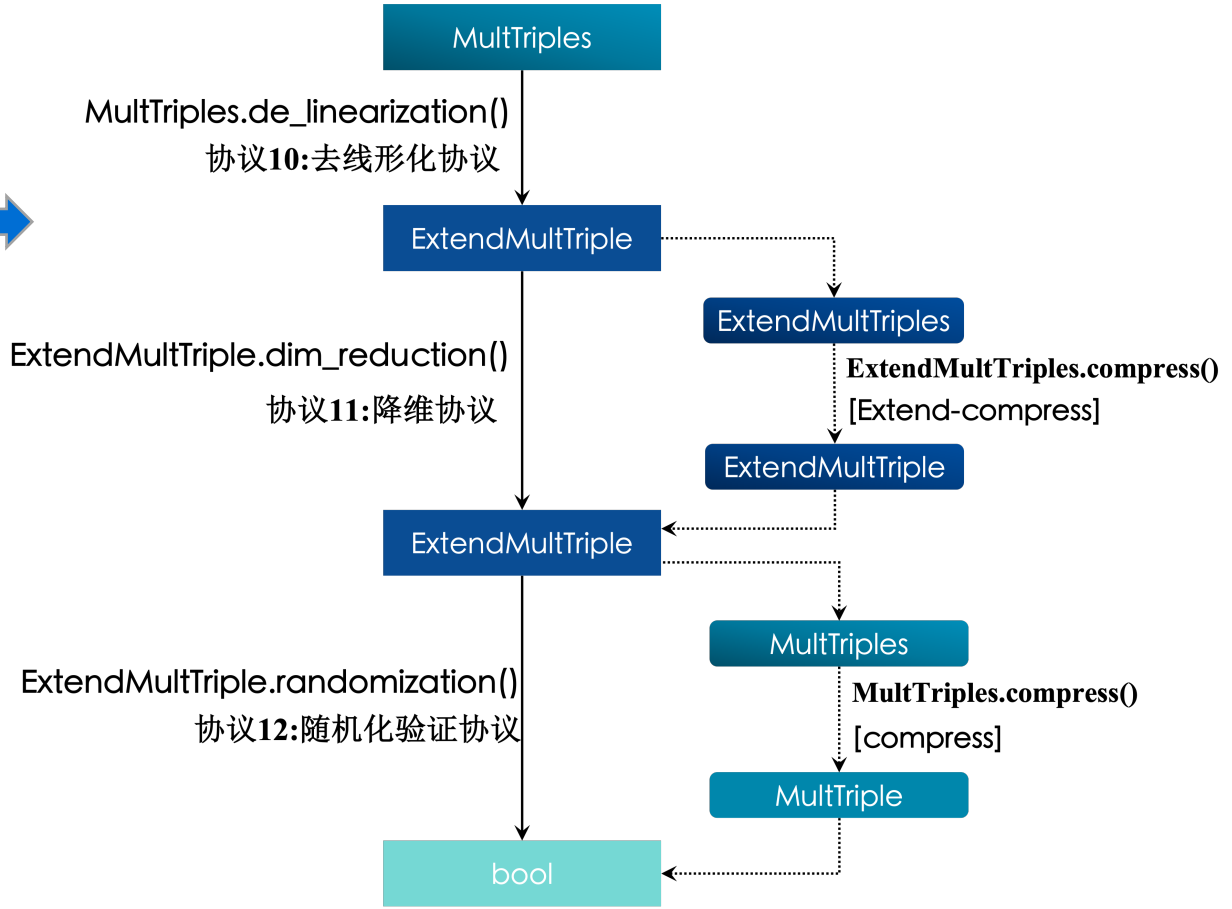
分享向量

ShareVector:
vector<Share>

验证阶段：MultTriple类



Class	成员	验证关系
MultTriple	单个乘法元组	$a \cdot b = c$
MultTriples	多个乘法元组	$a_i \cdot b_i = c_i$
ExtendMultTriple	单个向量内积元组	$a \odot b = c$
ExtendMultTriples	多个向量内积元组	$a_i \odot b_i = c_i$



验证阶段的执行过程

框架优化及扩展

系统模块

- 有限域操作模块
  - [优化]选择**梅森素数**：利用梅森素数的性质，提高有限域操作效率
- 安全多方计算协议模块
  - [优化]**向量化**：分离计算阶段和通信阶段，批量处理元素的通信阶段，打包在一个TCP包
  - [优化]**预处理**：在预处理阶段生成随机分享对，以备在线阶段使用
  - [优化] **多线程优化**：创建一个新的线程生成随机分享对
  - [扩展]**非线形运算**：实现分享的大小比较、相等性判断、比特分解、指数运算等

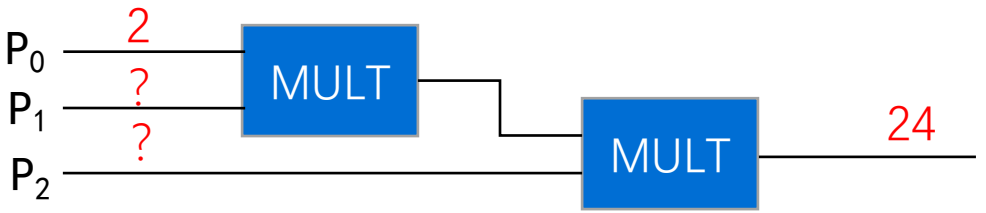
测试框架功能

安全设置

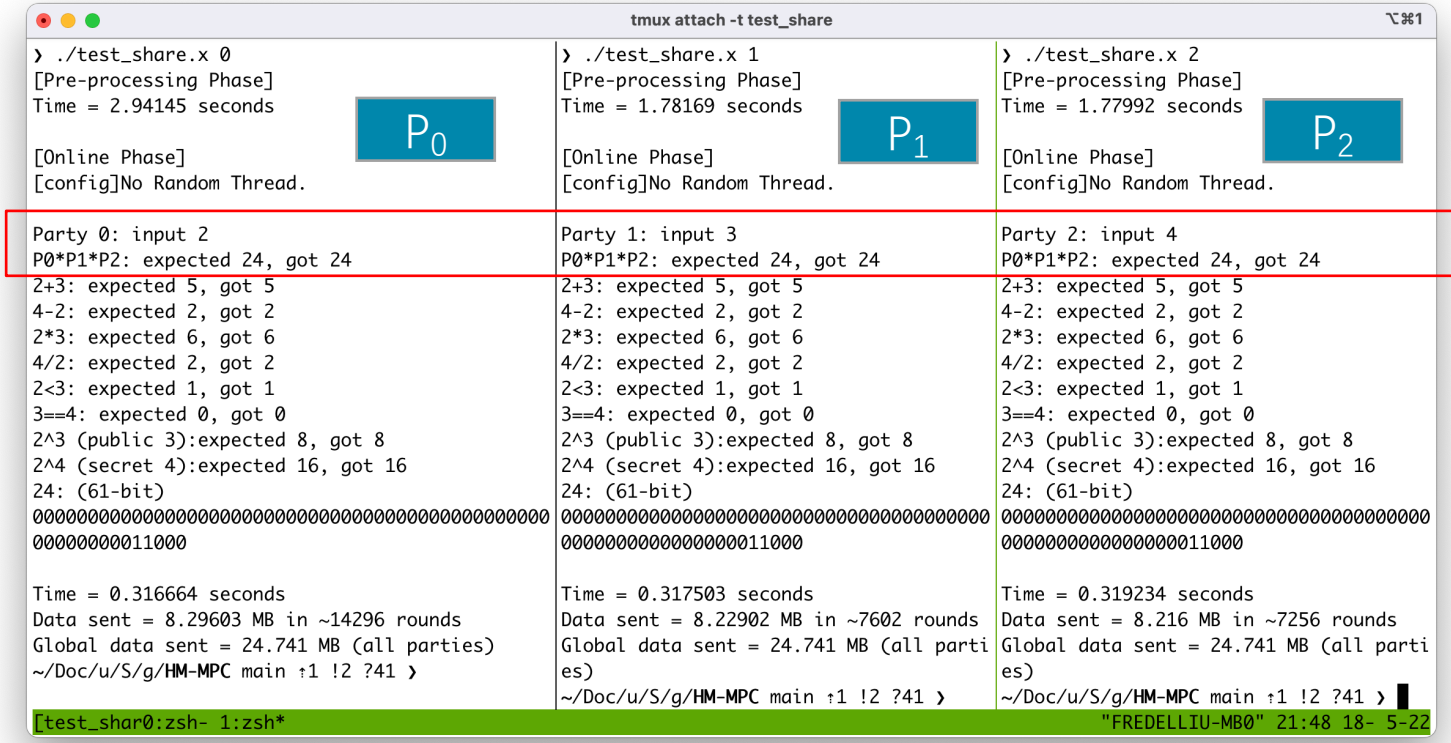
- 半诚实模型
- $n = 3, t = 1$

./test_share.x ID

- 输入阶段：输入隐私值
- 算术阶段
  - $+, -, \times, /$
  - $<, ==$
  - 指数运算
  - 比特分解
- 输出阶段：揭露计算结果



安全性：
P0可以得到三个参与方输入的乘法结果，但不会知道P1或P2的隐私输入



测试有限域模块操作效率

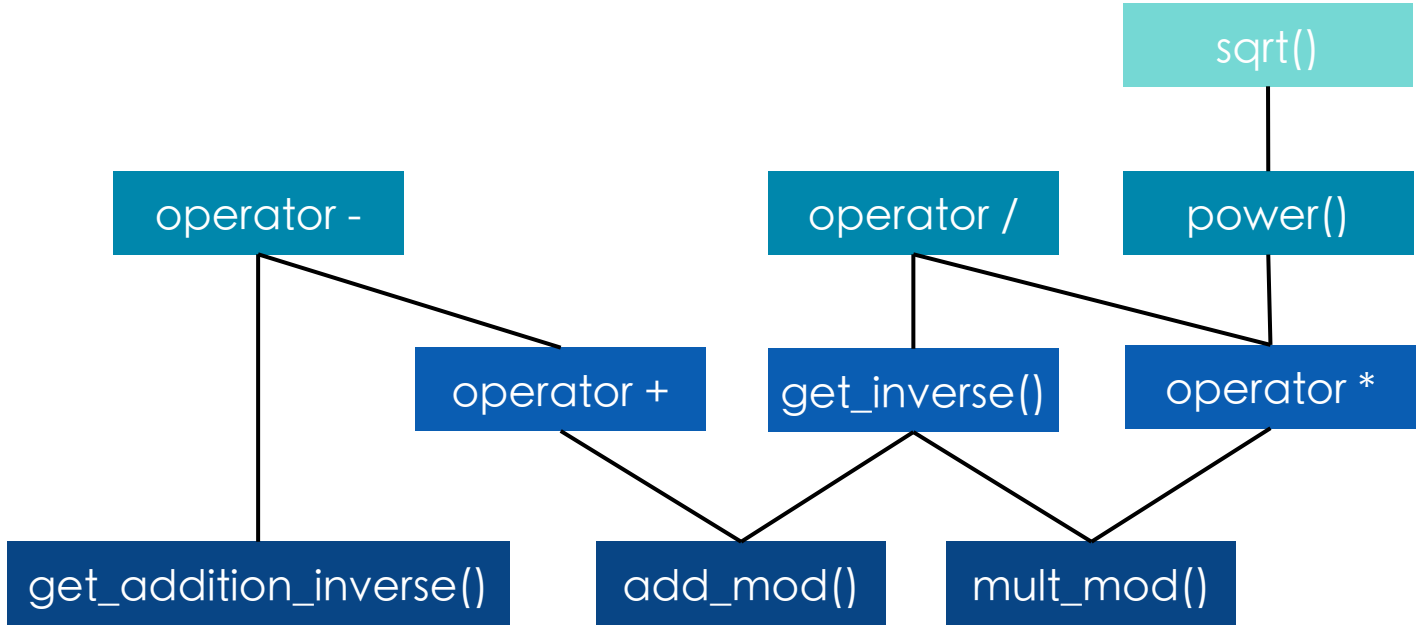
./test_gfp_speed.x

- 本框架：梅森素数$p = 2^{61} - 1$
- GNU多重精度运算库（GMP）

表 6-2 有限域操作的速度

操作	本框架（秒）	GMP（秒）	倍数
1e7 次加法操作	0.017158	0.529471	30.86
1e8 次加法操作	0.16741	4.61898	27.59
1e7 次乘法操作	0.036247	0.540848	14.92
1e8 次乘法操作	0.345394	4.98227	14.42

结论：加法操作快～28倍，乘法操作快～14倍。



利用梅森素数的性质，
优化底层的加法模操作和乘法模操作，
进而优化有限域上其他操作。

7方一百万个乘法操作的执行效率（P_0在线时间）

设置：

- 半诚实模型：285.5s
  - [+]向量化优化
  - [+]预处理
  - [+]多线程优化
- 恶意模型
  - [+]验证阶段

评估效率

- 执行时间
- 通信轮数
- 通信量

} 参与方P_0

- 总通信量

} 所有参与方

程序：测试一百万个乘法的执行效率

在线时间：285.5s

$2 \cdot 10^6$个隐私数据的输入；10^6个乘法操作； 10^6 个乘法结果的输出。

表 6-3 向量化优化对通信效率的影响

			3 方 $(n=3,t=1)$	5 方 $(n=5,t=2)$	7 方 $(n=7,t=3)$
向量化优化（批量大小=10000）	P_0	时间	3.51 s	6.07 s	8.65 s
		轮数	1200	1844	2400
		通信量	48 MB	85.76 MB	120 MB
	总通信量		136 MB	284.8 MB	432 MB
无优化	P_0	时间	160.89 s	224.47 s	285.49 s
		轮数	1.00*1e7	1.57*1e7	2.10*1e7
		通信量	48 MB	85.33 MB	120 MB
	总通信量		136 MB	282.67 MB	432 MB

结论：通信量不变，但能大幅降低通信时间和通信轮数。

7方一百万个乘法操作的执行效率（P_0在线时间）

设置：

- 半诚实模型：285.5s
  - [+]向量化优化：8.63s
  - [+]预处理
  - [+]多线程优化
- 恶意模型
  - [+]验证阶段

评估效率

- 执行时间
- 通信轮数
- 通信量

} 参与方P_0

- 总通信量

} 所有参与方

程序：测试一百万个乘法的执行效率$(n = 7, t = 3)$ **在线时间：8.63s**

$2 \cdot 10^6$个隐私数据的输入；10^6个乘法操作； 10^6 个乘法结果的输出。

表 6-4 批量大小对向量化优化的影响

批量大小（BATCH_SIZE）	P_0			总通信量
	时间	轮数	通信量	
10^2	11.94 s	240000	120 MB	432 MB
10^3	9.51 s	24000		
10^4	**8.63 s**	**2400**		
10^5	13.52 s	252	124.8 MB	465.6 MB
$2 \cdot 10^5$	62.40 s	138	134.4 MB	532.8 MB

结论：批量大小影响向量化优化的效果。
批量太大，TCP报文会在网络层被切片为多个IP包。
批量太小，一个向量将会被切分到多个TCP包中传输。

7方一百万个乘法操作的执行效率（P_0在线时间）

设置：

- 半诚实模型：285.5s
  - [+]向量化优化：8.63s
  - [+]预处理：3.42s
  - [+]多线程优化
- 恶意模型
  - [+]验证阶段

评估效率

- 执行时间
- 通信轮数
- 通信量

} 参与方P_0

- 总通信量

} 所有参与方

程序：测试一百万个乘法的执行效率$(n = 7, t = 3)$ **在线时间：3.42s**

$2 \cdot 10^6$个隐私数据的输入；10^6个乘法操作； 10^6 个乘法结果的输出。

表 6-5 预处理和多线程优化对通信效率的影响

设置		P_0			总通信量
		预处理时间	在线时间	通信量	
无预处理	/	0	8.45 s	120 MB	432 MB
	✓	0	9.17 s	120.96 MB	438.72 MB
R=5	/	0.83 s	6.60 s	120 MB	432 MB
	✓	1.86 s	7.30 s	120.96 MB	438.72 MB
R=10	/	1.77 s	5.93 s	120 MB	432 MB
	✓	2.22 s	6.47 s	120.96 MB	438.72 MB
R=25	/	**4.19 s**	**3.42 s**	**120 MB**	**432 MB**
	✓	4.15 s	3.90 s	120.96 MB	438.72 MB
R=50	/	8.87 s	3.54 s	144 MB	600 MB
R=70	/	12.54 s	3.42 s	163.2 MB	734.4 MB

结论：预处理生成适量的随机分享对能有效减少参与方的在线时间。

7方一百万个乘法操作的执行效率（P_0验证时间）

设置：

- 半诚实模型：285.5s
  - [+]向量化优化：8.63s
  - [+]预处理：3.42s
  - [+]多线程优化
- 恶意模型
  - [+]验证阶段：3.42+3.77

评估效率

- 执行时间
- 通信轮数
- 通信量

} 参与方P_0

- 总通信量

} 所有参与方

程序：测试一百万个乘法的执行效率$(n = 7, t = 3)$ 在线时间：3.42s

$2 \cdot 10^6$个隐私数据的输入；10^6个乘法操作； 10^6 个乘法结果的验证。

表 6-6 压缩参数 k 对验证时间的影响 验证时间：3.77s

压缩参数	P_0					总通信量
	预处理时间	验证时间	在线时间	轮数	通信量	
k = 10	4.36 s	11.08 s	14.54 s	1.20*1e6	80.16 MB	393.13 MB
k = 30	5.39 s	4.52 s	7.90 s	4.03*1e5	75.04 MB	373.29 MB
k = 50	**5.55 s**	**3.77 s**	**7.36 s**	**2.43*1e5**	**74.40 MB**	**372.01 MB**
k = 70	5.31 s	4.23 s	7.63 s	1.75*1e5	74.13 MB	371.47 MB
k = 100	5.43 s	5.06 s	8.56 s	1.24*1e5	73.93 MB	371.06 MB
k = 150	5.60 s	6.46 s	9.79 s	8.57*1e4	73.77 MB	370.75 MB

结论：压缩参数会影响验证阶段的效率。

7方诚实方多数情况下：
～3秒完成一百万次乘法运算；～3秒完成一百万个乘法元组的验证。

研究结果

基于BGW协议的安全多方计算协议框架的设计和实现

- 理论：设计了12个底层安全多方计算协议
  - 实现了**诚实方多数**情况下**半诚实模型的隐私安全性**和**恶意模型下的结果正确性**
- 实践：实现了安全多方计算协议框架
  - 框架能**将计算函数表示为算术电路，算术电路是图灵完备的，能够对隐私输入做任意计算**
  - 框架把操作扩展到向量运算上，使用**向量化技术**提高向量操作的效率
  - 框架使用**预处理和多线程技术**提高参与方在线阶段的执行效率
  - 框架支持**简单的非线形运算**
- 成果：在7个参与方（诚实方多数）的安全设置下
  框架能够在～3秒左右完成一百万次乘法运算，在～3秒左右完成一百万个乘法元组的验证

关键词：安全多方计算；BGW协议；秘密共享

Thanks for Listening.

网络通信模块

建立TCP连接，存储套接字

- 获得参与方的网络地址（域名和监听端口）
  - 未知：通过协调服务器获得
  - 已知：通过本地文件获得
- 用最少的TCP连接数建立全双工的通信网络：$\sum n + (n-1) + \cdots + 2 + 1 = \frac{(n+1)\cdot n}{2}$个连接

通过协调服务器获得各个参与方的网络地址

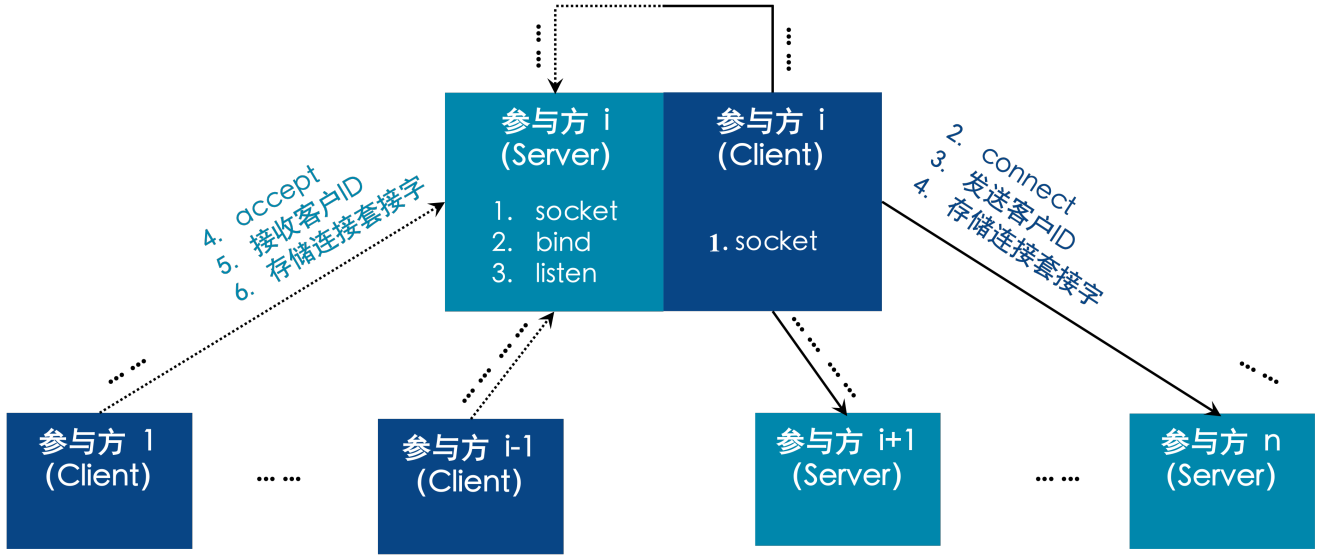
网络地址文件（HOSTS.example）
1 127.0.0.1: 5001
2 127.0.0.1: 5002
3 127.0.0.1: 5003

通过本地文件获得各个参与方的网络地址

网络通信模块

建立TCP连接，存储套接字

- 获得参与方的网络地址（域名和监听端口）
  - 未知：通过协调服务器获得
  - 已知：通过本地文件获得
- 用最少的TCP连接数建立全双工的通信网络：$\sum n + (n-1) + \cdots + 2 + 1 = \frac{(n+1)\cdot n}{2}$个连接



每个参与方既是服务器，也是客户端

参与方P_i
服务器：接收来自$P_1 - P_i$客户端的TCP连接请求
客户端：向$P_i - P_n$的服务器发起TCP连接

测试协议的执行效率

设置：

- 半诚实模型
  - [+]向量化优化
  - [+]预处理
  - [+]多线程优化
- 恶意模型
  - [+]验证阶段

评估效率

- 执行时间
- 通信轮数
- 通信量

} 参与方P_0

- 总通信量

} 所有参与方

程序：测试一百万个乘法的执行效率$(n = 7, t = 3)$ **在线时间：3. 42s**

$2 \cdot 10^6$个隐私数据的输入；10^6个乘法操作； 10^6 个乘法结果的输出。

表 6-5 预处理和多线程优化对通信效率的影响

设置		P_0			总通信量
		预处理时间	在线时间	通信量	
无预处理	/	0	8.45 s	120 MB	432 MB
	✓	0	9.17 s	120.96 MB	438.72 MB
R=5	/	0.83 s	6.60 s	120 MB	432 MB
	✓	1.86 s	7.30 s	120.96 MB	438.72 MB
R=10	/	1.77 s	5.93 s	120 MB	432 MB
	✓	2.22 s	6.47 s	120.96 MB	438.72 MB
R=25	/	**4.19 s**	**3.42 s**	**120 MB**	**432 MB**
	✓	4.15 s	3.90 s	120.96 MB	438.72 MB
R=50	/	8.87 s	3.54 s	144 MB	600 MB
R=70	/	12.54 s	3.42 s	163.2 MB	734.4 MB

结论1：预处理生成适量的随机分享对能有效减少参与方的在线时间。

结论2：在通信时间远大于计算时间时，多线程优化对效率影响不大。