通用 MPC 协议构造

基于秘密分享的 MPC 协议：GMW、BGW 协议

张聪
zhangcong@iie.ac.cn

中国科学院信息工程研究所国家重点实验室

2023 年 4 月 15 日

介绍

由于最初的混淆电路只能适用于两方布尔电路的情况 (将混淆电路扩展到多方和算术电路均不是很平凡的结果)，Goldreich, Micali 和 Wigderson[GMW87] 提出了 GMW 协议，是第一个支持多方计算布尔电路的 MPC 协议，且可以很容易地扩展到算数电路上。Ben-Or, Goldwasser 和 Wigderson[BGW88] 提出了 BGW 协议，是第一个实现无条件安全的 MPC 协议。

GMW 和 BGW 协议均使用秘密分享进行构造，不同的是，GMW 使用加法秘密分享，BGW 使用 Shamir 秘密分享。随后，Cramer, Damgård 和 Maurer[CDM00] 提出了使用一般的线性秘密分享 (Linear Secret Sharing Scheme,LSSS) 构造 MPC 的方法。一个直接的应用是用重复秘密分享 (Replicated Secret Sharing,RSS)[CDI05] 构造 MPC，使用 RSS 构造的 MPC 协议在参与方人数很少的时候效率很高。

不同框架对比

协议框架	GC	GMW	BGW	BMR
参与方数量	2	n	n	n
敌手门限	1	$t < n$	$t < n/2$	$t < n$
协议轮数	常数	与电路深度线性相关	与电路深度线性相关	常数
通信量	大	小	小	大
安全性	计算安全	计算安全	完美安全	计算安全
电路类型	布尔电路	布尔/算术电路	算术电路	布尔电路

表 1: 不同 MPC 框架对比

秘密分享方案

定义 ((t,n) 门限线性秘密分享方案)

域 $\mathbb{F}$ 上的 (t,n) 门限线性秘密分享方案是指两个算法 $(share, reconstruct)$:

- $Share(x)$：输入秘密 $x \in \mathbb{F}$，输出秘密分片 $[x] = (x_1, \ldots, x_n)$.
- $Reconstruct(\{x_{i_j}\}_{j\in[t+1]})$：输入任意 $t+1$ 个分片，输出秘密 x.

正确性：$\forall x \in \mathbb{F}, Pr[(x_1, \ldots, x_n) \leftarrow Share(x) : Reconstruct(\{x_{i_j}\}_{j\in[t+1]}) = x] = 1$

安全性：$\forall x, x' \in \mathbb{F}, \forall S \subseteq [n], |S| \leq t,$

$$\{(x_i)_{i\in S} : (x_1, \ldots, x_n) \leftarrow Share(x)\} \equiv \{(x'_i)_{i\in S} : (x'_1, \ldots, x'_n) \leftarrow Share(x')\}$$

线性组合：对系数 $c_1, \ldots, c_l, c \in \mathbb{F}$，$y = \sum_{i\in[l]} c_i x_i + c$ 的秘密分享 $[y] = \sum c_i[x_i] + c$.

LSSS

加法秘密分享

加法秘密分享 (Additive Secret Sharing) 是最简单的一种秘密分享，只能实现门限 $t = n - 1$:

- $Share(x)$: $x_i \leftarrow \mathbb{F}, i \in [n-1], x_n := x - \sum_{i \in [n-1]} x_i$，输出 $[x] := (x_1, \ldots, x_n)$.
- $Reconstruct(x_1, \ldots, x_n)$: 输出 $x := \sum_{i \in [n]} x_i$.

LSSS

Shamir 秘密分享

Shamir 秘密分享 (Shamir Secret Sharing) 可以实现任意门限 t，构造如下:

- $Share(x)$：令 $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$ 是 n 个不重复的非零元素 (e.g. $\alpha_i = i$)，随机采样一个 t 次多项式 $f(X)$ 使得 $f(0) = x$，令 $x_i := f(\alpha_i)$，输出 $\langle x \rangle_t := (x_1, \ldots, x_n)$.
- $Reconstruct(\{x_i\}_{i \in [t+1]})$：用拉格朗日插值公式，输出 $x := \sum_{i \in [t+1]} \delta_i(0) x_i$，其中 $\delta_i(X) = \Pi_{j \in [t+1], j \neq i} (X - \alpha_j)/(\alpha_i - \alpha_j)$.

注：拉格朗日插值公式:

$$f(X) = \sum_{i=1}^{n} x_i \delta_i(X) = \sum_{i=1}^{n} x_i \left(\prod_{j \neq i}^{1 \leq j \leq n} \frac{X - \alpha_j}{\alpha_i - \alpha_j} \right)$$

有

$$\delta_i(X) = \begin{cases} 1 & X = \alpha_i \\ 0 & X = \alpha_j, j \neq i \end{cases}$$

LSSS

重复秘密分享

重复秘密分享 (Replicated Secret Sharing, RSS) 可以实现任意门限 t，构造如下:

- $Share(x)$：对每个 $A \subseteq [n], |A| = t, x_A \leftarrow \mathbb{F}$,s.t. $\sum_{A\subseteq[n],|A|=t} x_A = x$，令 $x_i := \{x_A | i \notin T\}$，输出 $[x] := (x_1, \ldots, x_n)$.
- $Reconstruct(\{x_i\}_{i\in[t+1]})$：对所有 $A \subseteq [n], |A| = t$，输出 $x := \sum_A x_A$.

注：1. 每个参与方持有 C_{n-1}^t 个分片；2. 也有文献会取 A 为上述补集，即 $T \subseteq [n], |A| = n - t$，令 $x_i := \{x_A | i \in T\}$。此构造和上述构造相同，只是下标表示不同，因为 $C_{n-1}^t = C_{n-1}^{n-1-t}$.

LSSS

重复秘密分享

例 $(n=5, t=2)$：要分享秘密 x，先写成加法形式，

$$x = x_{12} + x_{13} + x_{14} + x_{15} + x_{23} + x_{24} + x_{25} + x_{34} + x_{35} + x_{45}$$

秘密分享如下：

$$\begin{aligned}
P_1 &: x_{23}, x_{24}, x_{25}, x_{34}, x_{35}, x_{45} \\
P_2 &: x_{13}, x_{14}, x_{15}, x_{34}, x_{35}, x_{45} \\
P_3 &: x_{12}, x_{14}, x_{15}, x_{24}, x_{25}, x_{45} \\
P_4 &: x_{12}, x_{13}, x_{15}, x_{23}, x_{25}, x_{35} \\
P_5 &: x_{12}, x_{13}, x_{14}, x_{23}, x_{24}, x_{34}
\end{aligned}$$

可以看到，每个 P_i 拿到的是下标不含 i 的那些项，此时如果想要恢复秘密，需要至少 3 个人合作，因为任意两个人 P_i, P_j 拥有的项一定不包括 x_{ij}，所以无法恢复秘密。

LSSS
重复秘密分享

例 $(n=3, t=1)$：要分享秘密 x，先写成加法形式，

$$x = x_1 + x_2 + x_3$$

秘密分享如下：

$$\begin{aligned} P_1 &: x_2, x_3 \\ P_2 &: x_1, x_3 \\ P_3 &: x_1, x_2 \end{aligned}$$

任意两方可恢复秘密。有的文献也会令 P_1 持有 x_1, x_2，P_2 持有 x_2, x_3，P_3 持有 x_3, x_1。

LSSS

分享转化

当各方拥有 RSS 时，可以通过本地直接转换成 Shamir 秘密分享。设 (n,t)-RSS 方案为

$$x = \sum_{A \subset [n]:|A|=t} x_A$$

想要将 x 转换为 (n,t)−Shamir 分享，考虑对每个 $A \subset [n], |A| = t$ 构造 t 次多项式 f_A，满足

1. $f_A(0) = 1$,
2. $f_A(i) = 0$ for $i \in A$

LSSS
分享转化

此时每一方 P_j 计算自己的分享为

$$x_j = \sum_{A\subset[n]:|A|=t, j\notin A} x_A \cdot f_A(j)$$

这些 $\{x_j\}_{j\in[n]}$ 构成了 $(n,t)-$Shamir 分享。这是因为，考虑 t 次多项式

$$f = \sum_{A\subset[n]:|A|=t} x_A \cdot f_A$$

则 $f(j) = x_j$ 且 $f(0) = \sum_A x_A = x$

LSSS

分享转化

上述方法只能将一个 RSS 转换成一个 Shamir 秘密分享。一个观察是，当秘密 x 是随机的时候，所有的 RSS 分片 x_A 是独立随机的。此时可以将 x_A 作为伪随机函数 $\psi(\cdot)$ 的密钥，只要各方对一个常数 a 达成共识，则可以直接用 $\psi_{x_A}(a)$ 代替 x_A 构造 Shamir 分享，即：

$$x_j = \sum_{A\subset[n]:|A|=t,j\notin A} \psi_{x_A}(a)\cdot f_A(j)$$

这样，只要各方有了一个 RSS，就可以本地计算无穷个随机 Shamir 秘密分享。需要注意的是这里的秘密分享其实和真正的 Shamir 分享不太一样，真正的 Shamir 分享的秘密分片是真随机的，这里的秘密分片是伪随机的 (即，伪随机秘密分享，PRSS)，这会导致用这种方法得到的 MPC 不再是信息论安全，而是计算安全的了。

1. 背景介绍
2. **GMW 协议**
3. BGW 协议
4. 预处理技术
5. 参考文献

GMW 协议

GMW 协议使用加法秘密分享 $[x] = x_1 + \cdots + x_n$。为了安全计算电路 C，协议分为以下三步：

1. 输入分享
2. 电路求值
   - 加法门：本地计算
   - 乘法门：交互计算
3. 输出重构

GMW 协议

定义乘法理想功能 $\mathcal{F}_{\text{mult}}$:

1. 从各方收到输入 $[x]=(x_1,\ldots,x_n),[y]=(y_1,\ldots,y_n)$
2. 计算 $z:=xy=(x_1+\cdots+x_n)(y_1+\cdots+y_n)$
3. 随机分享 $z=z_1+\cdots+z_n$，令 z_i 作为 P_i 的输出

注：简单来说，就是 $\mathcal{F}_{\text{mult}}([x],[y])\rightarrow[z]$

GMW 协议

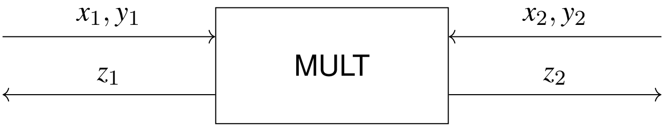
设有 n 个参与方 $P_1,\dots,P_n$，其中 P_i 输入 $\vec{x_i} \in \mathbb{F}^{l_i}$，计算 $C(\vec{x_1},\dots,\vec{x_n})$。描述 GMW 协议如下：

1. 输入分享：对每条输入线 w，设对应线上的值是参与方 P_i 的输入 x_w，P_i 进行加法分享 $[x_w] = (x_w^1,\dots,x_w^n)$，将 x_w^j 发给 $P_j, j \neq i$.
2. 电路求值：各方按照电路拓扑顺序求值，对每个电路门 (a,b,c,G)，设对应输入线 a,b 上的分享是 $[x],[y]$:
   - 如果 $G = ADD$：各方本地计算 $[z] := [x] + [y]$
   - 如果 $G = MULT$：各方输入 $[x],[y]$ 调用 $\mathcal{F}_{\text{mult}}$，得 $[z] = [xy]$
3. 输出重构：对 P_i 的输出线 w 上的值 y_w，各方把自己的分片 $y_w^j, j \neq i$ 发给 P_i，P_i 计算 $y_w := \sum_{i\in[n]} y_w^i$

GMW 乘法协议

构造 GMW 协议的问题就规约到构造乘法协议上了。先考虑两方布尔情况：

$P_1(x_1, y_1 \in \mathbb{F}_2)$ $\qquad$ $P_2(x_2, y_2 \in \mathbb{F}_2)$



$$z_1 \oplus z_2 = (x_1 \oplus x_2) \wedge (y_1 \oplus y_2)$$

GMW 乘法协议

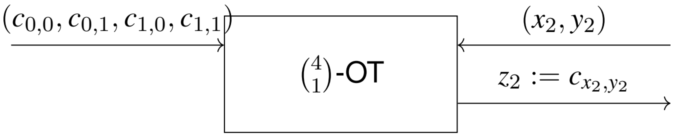
思路 1

由于 $z_2 = (x_1 \oplus x_2) \wedge (y_1 \oplus y_2) \oplus z_1$，让 P_1 随机选择 z_1，然后遍历 x_2, y_2 的所有可能，用 4 选 1OT 让 P_2 选择正确的值：

$P_1(x_1, y_1 \in \mathbb{F}_2)$ $\qquad\qquad$ $P_2(x_2, y_2 \in \mathbb{F}_2)$

$$z_1 \leftarrow \mathbb{F}_2$$

$$c_{i,j} := (x_1 \oplus i) \wedge (y_1 \oplus j) \oplus z_1, i, j \in \{0, 1\}$$



$$z_1 \oplus z_2 = (x_1 \oplus x_2) \wedge (y_1 \oplus y_2)$$

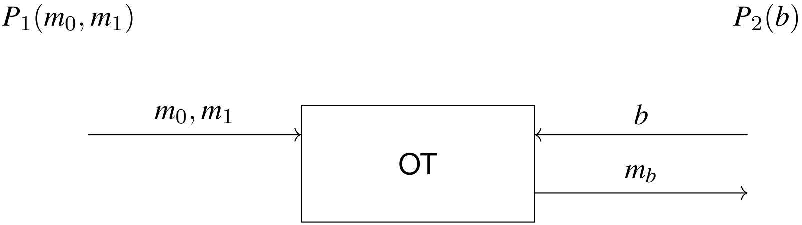
扩展到 $\mathbb{F}_p$: 用 p^2 选 1OT。

GMW 乘法协议

思路 2

展开 $(x_1 \oplus x_2) \wedge (y_1 \oplus y_2) = x_1y_1 \oplus x_1y_2 \oplus x_2y_1 \oplus x_2y_2$，这里 x_1y_1, x_2y_2 可以分别由 P_1 和 P_2 本地计算，下面的问题是如何计算交叉项 x_1y_2 和 x_2y_1 的加法分享。

回忆 OT 功能：



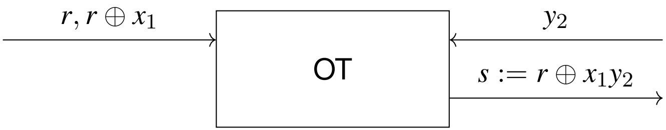
$$m_b = m_0 \oplus b(m_0 \oplus m_1)$$
$$m_b \oplus m_0 = b(m_0 \oplus m_1)$$

GMW 乘法协议

思路 2

$P_1(x_1)$ $\qquad\qquad$ $P_2(y_2)$

$r \leftarrow \mathbb{F}_2$



$$r \oplus s = x_1 y_2$$

GMW 乘法协议

$P_1(x_1, y_1)$ $P_2(x_2, y_2)$

$r_1 \leftarrow \mathbb{F}_2$ $r_2 \leftarrow \mathbb{F}_2$

$r_1, r_1 \oplus x_1$ → OT ← y_2

OT → $s_2 = r_1 \oplus x_1 y_2$

y_1 → OT ← $r_2, r_2 \oplus x_2$

$s_1 := r_2 \oplus x_2 y_1$ ← OT

$z_1 := x_1 y_1 \oplus s_1 \oplus r_1$ $z_2 := x_2 y_2 \oplus s_2 \oplus r_2$

$$z_1 \oplus z_2 = (x_1 \oplus x_2) \wedge (y_1 \oplus y_2)$$

扩展到 $\mathbb{F}_p$: 用 OLE 代替 OT

GMW 乘法协议

思路 3

用同态加密可以直接计算乘法分享:

$$
\begin{array}{ccc}
P_1(x_1, y_1\{pk_1, pk_2, sk_1\}) & & P_2(x_2, y_2, \{pk_1, pk_2, sk_2\}) \\
r_1 \leftarrow \mathbb{F}_2 & & r_2 \leftarrow \mathbb{F}_2 \\
c_1 := \mathrm{Enc}(pk_1, x_1) & & c_2 := \mathrm{Enc}(pk_2, x_2) \\
& \xrightarrow{\quad c_1 \quad} & \\
& \xleftarrow{\quad c_2 \quad} & \\
c_2' := y_1 c_2 + r_1 & & c_1' := y_2 c_1 + r_2 \\
& \xrightarrow{\quad c_2' \quad} & \\
& \xleftarrow{\quad c_1' \quad} & \\
s_1 := \mathrm{Dec}(sk_1, c_1') = x_1 y_2 + r_2 & & s_2 := \mathrm{Dec}(sk_2, c_2') = x_2 y_1 + r_1 \\
z_1 := x_1 y_1 - r_1 + s_1 & & z_2 := x_2 y_2 - r_2 + s_2
\end{array}
$$

多方 GMW 乘法协议

多方情况的 GMW 可以转化为两方情况。在多方情况下，P_i 有 x_i, y_i，满足 $\sum_{i\in[n]} x_i = x, \sum_{i\in[n]} y_i = y$，$P_i$ 想要得 z_i 满足 $\sum_{i\in[n]} z_i = (\sum_{i\in[n]} x_i)(\sum_{i\in[n]} y_i)$。有如下观察：

$$
\begin{aligned}
(\sum_{i\in[n]} x_i)(\sum_{i\in[n]} y_i) &= \sum_{i\in[n]} x_i y_i + \sum_{1\le i<j\le n} (x_i y_j + x_j y_i) \\
&= (1-(n-1)) \cdot \sum_{i\in[n]} x_i y_i + \sum_{1\le i<j\le n} (x_i + x_j)\cdot(y_i + y_j) \\
&= (2-n) \cdot \sum_{i\in[n]} x_i y_i + \sum_{1\le i<j\le n} (x_i + x_j)\cdot(y_i + y_j)
\end{aligned}
$$

可以看到，$(2-n)x_i y_i$ 可以由 P_i 本地计算，后面的项恰好对应于 P_i, P_j 的两方乘法协议。因此调用 C_n^2 次两方乘法协议即可。

BGW 协议

BGW 框架使用 Shamir 秘密分享 $\langle x\rangle_t = (x_1, \ldots, x_n)$，这里设 f 是 t 次多项式，$f(i) = x_i, f(0) = x$。和 GMW 协议相同，为了安全计算电路 C，协议分为以下三步：

1. 输入分享
2. 电路求值
   - 加法门：本地计算
   - 乘法门：交互计算
3. 输出重构

BGW 协议

定义乘法理想功能 $\mathcal{F}_{\text{mult}}$:

1. 从各方收到输入 $\langle x \rangle_t = (x_1, \ldots, x_n), \langle y \rangle_t = (y_1, \ldots, y_n)$
2. 计算 $z := xy$
3. 计算 Shamir 分享 $\langle z \rangle_t := (z_1, \ldots, z_n)$，令 z_i 作为 P_i 的输出

注：简单来说，就是 $\mathcal{F}_{\text{mult}}(\langle x \rangle_t, \langle y \rangle_t) \to \langle z \rangle_t$

BGW 协议

设有 n 个参与方 $P_1,\ldots,P_n$，其中 P_i 输入 $\vec{x_i}\in\mathbb{F}^{l_i}$，计算 $C(\vec{x_1},\ldots,\vec{x_n})$。描述 BGW 协议如下：

1. 输入分享：对每条输入线 w，设对应线上的值是参与方 P_i 的输入 x_w，P_i 进行 Shamir 分享 $\langle x_w\rangle_t=(x_w^1,\ldots,x_w^n)$，将 x_w^j 发给 $P_j, j\neq i$.
2. 电路求值：各方按照电路拓扑顺序求值，对每个电路门 (a,b,c,G)，设对应输入线 a,b 上的分享是 $\langle x\rangle_t,\langle y\rangle_t$:
   - 如果 $G=ADD$：各方本地计算 $\langle z\rangle_t:=\langle x\rangle_t+\langle y\rangle_t$
   - 如果 $G=MULT$：各方输入 $\langle x\rangle_t,\langle y\rangle_t$ 调用 $\mathcal{F}_{\mathrm{mult}}$，得 $\langle z\rangle_t=\langle xy\rangle_t$
3. 输出重构：对 P_i 的输出线 w 上的值 y_w，各方把自己的分片 $y_w^j, j\neq i$ 发给 P_i，P_i 计算 $y_w:=\sum_{i\in[n]}\delta_i(0)y_w^i$

BGW 乘法协议

设分享 x 的多项式是 f，分享 y 的多项式是 g，即 $x_i = f(i), x = f(0), y_i = g(i), y = g(0)$。Shamir 分享的一个好处是，各方可以本地直接把自己的分享相乘，即 P_i 令 $z_i := x_i y_i$。可以发现 $z_i = f(i)g(i)$，不妨设 $h = fg$ 是 $2t$ 次多项式，则显然 $h(0) = f(0)g(0) = xy$。因此这里 z_i 是 xy 的 $2t$ 次分享。

这里有两个问题:

1. z_i 其实不是标准的 Shamir 分享：h 是可约多项式
2. h 是一个 $2t$ 次分享，需要把它降低成 t 次分享

BGW 乘法协议

解决办法：

1. 令各方分享一个 $2t$ 次的 0，所有人把各方的分享和 z_i 加起来
2. 将 h 截断，去掉 t 次以上的项。设 $h(x)=\sum_{i\in[0,2t]}h_ix^i$，令 $\hat{h}=\sum_{i\in[0,t]}h_ix^i$ 表示 h 去掉 t 次以上项的截断多项式，则 $h(0)=\hat{h}(0)=xy$

问题：

在各方有 $h(1),\ldots,h(n)$ 的情况下，如何得到 $\hat{h}(1),\ldots,\hat{h}(n)$?

定理

设 $t<n/2$，存在矩阵 $A\in\mathbb{F}^{n\times n}$ 使得 $(\hat{h}(1),\ldots,\hat{h}(n))^T=A\cdot(h(1),\ldots,h(n))^T$。

BGW 乘法协议

证明：

令 $\vec{h} = (h_0, h_1, \ldots, h_t, \ldots, h_{2t}, 0, \ldots, 0) \in \mathbb{F}^n$，令 V 表示 n 阶范德蒙矩阵，

即 $V = \begin{bmatrix} 1 & 1 & \ldots & 1^{n-1} \\ 1 & 2 & \ldots & 2^{n-1} \\ \vdots & \vdots & & \vdots \\ 1 & n & \ldots & n^{n-1} \end{bmatrix}$

则 $V \cdot \vec{h}^T = (h(1), \ldots, h(n))^T$。由于 V 是可逆的，有 $\vec{h}^T = V^{-1} \cdot (h(1), \ldots, h(n))^T$

同样地，令 $\vec{\hat{h}} = (h_0, h_1, \ldots, h_t, 0, \ldots, 0)$，有 $V \cdot \vec{\hat{h}}^T = (\hat{h}(1), \ldots, \hat{h}(n))^T$

令 $T = \{1, \ldots, t\}$，且令 P_T 表示对角线前 t 个位置为 1，剩下位置为 0 的 n 阶矩阵，

即 $P_T(i,j) = 1$ 当且仅当 $i = j \in T$。则 $P_T \cdot \vec{h}^T = \vec{\hat{h}}^T$。于是

$(\hat{h}(1), \ldots, \hat{h}(n))^T = V \cdot \vec{\hat{h}}^T = V \cdot P_T \cdot \vec{h}^T = V \cdot P_T \cdot V^{-1} \cdot (h(1), \ldots, h(n))^T$

即 $A = V \cdot P_T \cdot V^{-1}$。证完。

BGW 乘法协议

有了 $(\hat{h}(1),\ldots,\hat{h}(n))^T = A\cdot(h(1),\ldots,h(n))^T$ 关系之后，可以发现，$2t$ 次分享和 t 次分享只相差一些线性运算，而线性运算可以在把输入分享之后，直接本地算。乘法就这样做完了。协议描述如下：

1. P_i 计算 $z_i'' := x_i y_i$。
2. P_i 随机选一个 $2t$ 次多项式 $q_i(x)$ 满足 $q_i(0)=0$，将 $s_{i,j}=q_i(j)$ 发给 $P_j, j\neq i$。
3. P_i 收到 $s_{1,i},\ldots,s_{n,i}$，令 $z_i' := z_i'' + \sum_{j\in[n]} s_{j,i}$。
4. 各方用 z_i' 作为输入，执行 BGW 协议，计算 $A\cdot(z_1',\ldots,z_n') = (z_1,\ldots,z_n)$，$P_i$ 输出 z_i。则 z_i 就是 xy 的 t 次秘密分享。

这里第二第三步是重随机化，第四步是次数约减。

BGW 乘法协议

其实还有另一种角度看待这个乘法协议。如何在次数约减这一步更直观地理解矩阵 A 呢？在 Shamir 分享中，要重构秘密，需要用到拉格朗日插值公式，即过 $(1, x_1), \ldots, (n, x_n)$ 的多项式 $f(x)$ 可以写为：

$$f(x) = \sum_{i=1}^{n} x_i \Big(\prod_{j \neq i}^{1 \le j \le n} \frac{x - j}{i - j} \Big)$$

有

$$x = f(0) = \sum_{i=1}^{n} x_i \Big(\prod_{j \neq i}^{1 \le j \le n} \frac{-j}{i - j} \Big), \text{ 令 } \lambda_i = \prod_{j \neq i}^{1 \le j \le n} \frac{-j}{i - j}$$

BGW 乘法协议

则 $x = \sum_{i\in[n]} \lambda_i x_i$，因此 Shamir 分享的线性组合可以直接恢复秘密。那么当各方得到 xy 的 $2t$ 次分享之后，各方可以直接把这个分享再用 t 次 Shamir 分享分享出去，然后各方用 λ_i 把收到的分享线性组合，就能得到 xy 的 t 次分享。这样看还有一个好处，就是因为各方使用了新的 t 次分享，此时各方得到的分享一定是随机的，所以重随机步骤可以不要了。重新描述协议如下：

1. P_i 计算 $z_i' := x_i y_i$。
2. P_i 把 z_i' 用 Shamir 分享出去，即随机选取 t 次多项式 $u_i(x)$，满足 $u_i(0) = z_i'$，将 $v_{i,j} = u_i(j)$ 发给 $P_j, j \neq i$。
3. P_i 收到 $v_{1,i}, \ldots, v_{n,i}$，令 $z_i := \lambda_1 v_{1,i} + \cdots + \lambda_n v_{n,i}$。由于 $\{v_{i,j}\}_{j\in[n]}$ 是 z_i' 的 t 次 Shamir 分享，$\{z_i\}_{i\in[n]}$ 就是 $\sum_{i\in[n]} \lambda_i z_i' = z = xy$ 的 t 次分享。这里 λ_i 是 $2t$ 次插值多项式的拉格朗日系数。

RSS 乘法协议

由于重复秘密分享 RSS 各方拥有的分享大小为指数 (C_{n-1}^{t})，因此不适用于 n 比较大的情况，一般具体考虑 $n=3,4,5$ 的时候会使用这种秘密分享 [Ara+16; Boy+19]，它的好处就是乘法的具体通信量很低，只需发送一个元素。这里以 $n=3$ 为例。

先回顾一下 $(3,1)$-RSS 方案，要分享 x，令 $x=x_1+x_2+x_3$，则 P_1 拿 x_2,x_3，P_2 拿 x_1,x_3，P_3 拿 x_1,x_2（即 P_i 拿 x_{i-1} 和 x_{i+1}，下标模 3）。要计算乘法

$$
\begin{aligned}
xy &= (x_1+x_2+x_3)(y_1+y_2+y_3) \\
&= x_1y_1+x_2y_2+x_3y_3+x_1y_2+x_1y_3+x_2y_1+x_2y_3+x_3y_1+x_3y_2 \\
&= \sum_{i\in[3]}(x_{i+1}y_{i+1}+x_{i+1}y_{i-1}+x_{i-1}y_{i+1}) := \sum_{i\in[3]} z_{i+1}
\end{aligned}
$$

RSS 乘法协议

可以看到，P_i 可以直接本地直接算出 xy 的加法分享 z_{i+1}。为了保持 RSS 的形式，P_i 把 z_{i+1} 发给 P_{i-1}，此时 P_i 就有了 z_{i+1}, z_{i-1} 满足是 xy 的一个 RSS。但是这里还有一个问题，就是 z_{i+1} 并不是随机的，它是由输入的一些乘积加起来得到的，也会出现像 BGW 中类似的问题，因此也需要重新随机化一下，即，让各方生成一个 0 的分享 $\alpha_1+\alpha_2+\alpha_3=0$。然后把这个分享加上去，
$z_{i+1} := x_{i+1}y_{i+1}+x_{i+1}y_{i-1}+x_{i-1}y_{i+1}+\alpha_i$，即可。
下面描述如何生成这样的 0 分享。只需要 P_i 随机选一个元素 ρ_i 发给 P_{i+1}，然后 P_i 令 $\alpha_i := \rho_i - \rho_{i-1}$。容易验证，$\alpha_1+\alpha_2+\alpha_3=0$。为了大量生成这样的随机对，可以利用伪随机秘密分享 PRSS 类似的想法，把 ρ 作为 PRF 的密钥，即令
$\alpha_i := F_{\rho_{i-1}}(id) - F_{\rho_i}(id)$，这样就能本地计算无限多个 0 分享。代价是由无条件安全变成了计算安全。

RSS 乘法协议

下面描述 3 方 RSS 乘法协议:

1. P_i 计算 $z_{i+1} := x_{i+1}y_{i+1} + x_{i+1}y_{i-1} + x_{i-1}y_{i+1} + \alpha_i$。发给 P_{i-1}。
2. P_i 从 P_{i+1} 收到 z_{i-1}，令分享为 (z_{i-1}, z_{i+1})。

可以看到，在 setup 生成 0 分享之后，每方只需发送一个元素即可计算乘法。

预处理技术

如果在得到输入之前可以预先计算一些相关随机对，各方在得到输入之后就可以很快地计算乘法。目前这方面主要有两种方法：Beaver triple 和 Double sharing。其中 Beaver triple 可以同时适用 GMW 和 BGW 协议，而 Double sharing 只适用于 BGW 协议，即 honest majority。

Beaver triple

Beaver triple[Bea91] 就是一个随机的乘法分享对 $[a], [b], [c]$(对 BGW 就是 $\langle a\rangle_t, \langle b\rangle_t, \langle c\rangle_t$)，其中 $c = ab$。当有了 Beaver triple 时，在线阶段的乘法就可以通过打开两次秘密完成，不需要额外的 OT 或者 HE。相当于把乘法放到 offline 做了。这主要基于一个观察:

$$xy = (x - a + a) \cdot (y - b + b) = (x - a)(y - b) + a(y - b) + b(x - a) + ab$$

在线阶段直接打开 $\rho = x - a, \sigma = y - b$，则 $[xy] = \rho\sigma + \sigma[a] + \rho[b] + [c]$。即 $[a], [b], [c]$ 的线性计算，可以直接在本地完成。协议描述如下:

1. P_i 计算 $x_i - a_i, y_i - b_i$ 并发给所有人。
2. 各方收到秘密后重构 $\rho := x - a, \sigma := y - b$。
3. P_i 计算 $z_i := \rho\sigma + \sigma a_i + \rho b_i + c_i$。

Beaver triple

关于 Beaver triple 的生成，其实和乘法协议是类似的，把输入换成了随机选取的即可。这里以两方布尔加法分享的 Beaver triple 生成为例，协议如下：

1. P_1 随机选 $a_1 \leftarrow \{0,1\}$ 作为输入，和 P_2 执行随机 OT，P_1 得到 $m^2_{a_1}$，P_2 得到 (m^2_0, m^2_1)。
2. P_2 随机选 $a_2 \leftarrow \{0,1\}$ 作为输入，和 P_1 执行随机 OT，P_2 得到 $m^1_{a_2}$，P_1 得到 (m^1_0, m^1_1)。
3. P_1 令 $b_1 := m^1_0 \oplus m^1_1$，$P_2$ 令 $b_2 := m^2_0 \oplus m^2_1$。
4. P_1 令 $c_1 := a_1 b_1 \oplus m^2_{a_1} \oplus x^1_0$，$P_2$ 令 $c_2 := a_2 b_2 \oplus m^1_{a_2} \oplus x^2_0$。

容易验证，$c_1 \oplus c_2 = (a_1 \oplus a_2)(b_1 \oplus b_2)$。类似的，也可以用 HE 做。

Beaver triple

关于 BGW 协议 Beaver triple 的生成 $\langle a\rangle_t, \langle b\rangle_t, \langle c\rangle_t$，思路是让各方先生成随机 t 分享 $\langle a\rangle_t, \langle b\rangle_t$，然后执行一次乘法协议算 $\langle c\rangle_t$ 即可。生成随机 t 分享目前主要有两种方法。一种基于伪随机秘密分享 (psudorandom secret sharing,PRSS)[CDI05]，该方法使各方能够生成伪随机元素的分享，而无需任何交互。主要使用的就是之前 RSS 转化成 Shamir 分享的思路。这种方法的问题是，每一方持有的密钥数量随着参与方的数量呈指数增长，这大大增加了生成分享的计算工作量。因此，只有当协议中的参与方数量较少时，这种方法才有效。具体协议描述如下：

1. (Setup) 对每个 $A \subset [n], |A| = t$，对不在集合 A 中最小的那一方 (即 $P_i, i = min\{j \in [n] | j \notin A\}$) 随机选择 k_A，并发给其他所有不在 A 中的参与方 (即 $P_j, j \notin A$)。最后 P_i 就会得到所有 $\{k_A | i \notin A\}$。
2. (Upon request) 各方对每个 $A \subset [n], |A| = t$ 构造 t 次多项式 f_A，满足 $f_A(0) = 1, f_A(i) = 0$ for $i \in A$。P_i 令 $r_i = \sum_{A\subset[n]:|A|=t, i\notin A} F_{k_A}(id) \cdot f_A(i)$。这里 id 是公开参数，每生成一次新的分享都要用一个新的 id。

Beaver triple

一种是基于范德蒙矩阵 [DN07]，该矩阵可用于从 n 个分享“提取随机性”到 $t+1$ 个新分享。思路是是让每一方向其他方分享一个随机元素。然后，在持有 n 个分享向量时，各方将该大小为 n 的向量与 $n-t$ 行 n 列的范德蒙矩阵相乘，得到 $n-t$ 个“新”随机分享。通过随机性提取性质，我们得到新的分享是 $\mathbb{F}$ 中随机元素的分享。由于 $t<\frac{n}{2}$，各方得到至少 $\frac{n}{2}+1$ 个分享。由于每方需要发送 $n-1$ 个元素，平均每个随机分享通信约为 2 个元素。设 $V_{n-t}=\begin{bmatrix} 1 & 1 & \dots & 1^{n-t-1} \\ 1 & 2 & \dots & 2^{n-t-1} \\ \vdots & \vdots & & \vdots \\ 1 & n & \dots & n^{n-t-1} \end{bmatrix}$ 是一个 $n\times(n-t)$ 的范德蒙矩阵，协议描述如下：

1. P_i 随机选一个 u_i 并用 t 次 Shamir 分享把 u_i 分享出去，即 $\langle u_i\rangle_t$。
2. 各方本地计算 $(\langle r_1\rangle_t,\dots,\langle r_{n-t}\rangle_t)^T := V_{n-t}^T\cdot(\langle u_1\rangle_t,\dots,\langle u_n\rangle_t)^T$。

Beaver triple

使用 Beaver triple 计算乘法门时，每个乘法门需要打开两个值，即平均每人通信 2 个元素，[BNO19] 观察发现通过改进分享形式，在函数依赖阶段 (function-dependent phase) 生成 Beaver triple，可以降低到每人通信 1 个元素。其主要思路是在线阶段令每条线 w 上的分享为 $(\Lambda_W, [\lambda_w])$，其中 $\Lambda_w = \lambda_w + x_w$。这里 Λ_w 是公开值。此时加法门依然可以本地计算，乘法门只需消耗一对 Beaver triple。

$$\begin{aligned}
\Lambda_c &= x_c + \lambda_c \\
&= x_a x_b + \lambda_c \\
&= (\Lambda_a - \lambda_a)(\Lambda_b - \lambda_b) + \lambda_c \\
&= \Lambda_a \Lambda_b - \Lambda_a \lambda_b - \Lambda_b \lambda_a + \lambda_{ab} + \lambda_c
\end{aligned}$$

Beaver triple

1. 离线阶段:
   - 对每条非加法门输出线 w 生成一个随机加法分享 $[\lambda_w]$，对每个加法门 (a,b,c,ADD)，令 $[\lambda_c] := [\lambda_a] + [\lambda_b]$
   - 对 P_i 的输入线 w，向 P_i 打开 $[\lambda_w]$
   - 对每个乘法门 (a,b,c,AND) 计算一对 Beaver triple $([\lambda_a],[\lambda_b],[\lambda_{ab}])$
2. 在线阶段:
   1. 输入分享：对每条输入线 w，设对应线上的值是参与方 P_i 的输入 x_w，且各方有随机分享 $[\lambda_w]$。P_i 计算 $\Lambda_w := x_w + \lambda_w$ 并将 Λ_w 发给所有参与方。此时每条输入线 w 上均有 $(\Lambda_w,[\lambda_w])$
   2. 电路求值：各方按照电路拓扑顺序求值，对每个电路门 (a,b,c,G)，设对应输入线 a,b 上的分享是 $(\Lambda_a,[\lambda_a]),(\Lambda_b,[\lambda_b])$:
      - 如果 $G = ADD$：各方本地计算 $\Lambda_c := \Lambda_a + \Lambda_b, [\lambda_c] := [\lambda_a] + [\lambda_b]$
      - 如果 $G = MULT$：各方计算 $[\Lambda_c] := \Lambda_a\Lambda_b - \Lambda_a[\lambda_b] - \Lambda_b[\lambda_a] + [\lambda_{ab}] + [\lambda_c]$ 并打开
   3. 输出重构：对 P_i 的输出线 w 上的分享 $(\Lambda_w,[\lambda_w])$，各方把自己的分片 $\lambda_w^j, j \neq i$ 发给 P_i，P_i 计算 $\lambda_w := \sum_{i\in[n]} \lambda_w^i, y_w := \Lambda_w - \lambda_w$

Double Sharing

Double sharing[DN07] 是指同一个秘密的 t 次和 $2t$ 次分享，即 $(\langle r\rangle_t, \langle r\rangle_{2t})$。可以看到这种形式是专门针对 honest majority 的。使用 double sharing 的好处是可以将乘法协议的通信复杂度降到 $O(n)$。注意到之前的协议，不管是 GMW 还是 BGW，还是用 Beaver triple，都需要每次乘法每个人向其他所有人发送消息，即总的通信复杂度为 $O(n^2)$。

Double Sharing

使用 double sharing 的主要思想是选取某一方作为 P_{king}，让大家先把自己的分享本地相乘，再减去 $2t$ 次随机分享 r，即 $\langle x\rangle_t \cdot \langle y\rangle_t - \langle r\rangle_{2t}$，发给 P_{king}。P_{king} 重构出 $xy - r$ 之后再发给各方，各方收到之后在加上 r 的 t 次分享，即 $xy - r + \langle r\rangle_t$ 这样就得到了 xy 的 t 次分享 $\langle xy\rangle_t$。设 $\langle r\rangle_t$ 和 $\langle r\rangle_{2t}$ 中各方的分享分别是 r_i^t 和 r_i^{2t}。使用 double sharing 的乘法协议如下：

1. P_i 计算 $x_iy_i - r_i^{2t}$ 并发给 P_{king}。
2. P_{king} 重构出 $xy - r$，发给所有人。
3. P_i 令 $z_i := xy - r + r_i^t$。

Double Sharing

double sharing 的生成和单个随机 BGW 分享 $\langle r \rangle_t$ 的生成类似，也是使用范德蒙矩阵进行提取。这里不能使用 PRSS 的方法是因为我们需要同时生成同一个数的 t 和 $2t$ 的分享，而 PRSS 中的门限和 RSS 的门限是一样的，这意味着要想生成 $2t$ 次分享，还要再进行一个 $2t$ 的 setup，但是即使 setup 了之后，由于每次运行都是生成新的伪随机数的分享，无法保证和 t 门限时候的值一致。这里描述用范德蒙矩阵生成 double sharing 的协议如下：

1. P_i 随机选一个 u_i 并同时用 t 次和 $2t$ 次 Shamir 分享把 u_i 分享出去，即 $\langle u_i \rangle_t, \langle u_i \rangle_{2t}$。
2. 各方本地计算
$(\langle r_1 \rangle_t, \ldots, \langle r_{t+1} \rangle_t)^T := V_{t+1}^T \cdot (\langle u_1 \rangle_t, \ldots, \langle u_n \rangle_t)^T$,
$(\langle r_1 \rangle_{2t}, \ldots, \langle r_{t+1} \rangle_{2t})^T := V_{t+1}^T \cdot (\langle u_1 \rangle_{2t}, \ldots, \langle u_n \rangle_{2t})^T$

Double Sharing

开销计算

使用 Double Sharing 计算乘法的通信开销为：

- 随机对生成：生成 t 对 double sharing 需要每个人发送 $2n$ 个元素，取 $t \approx n/2$，平均生成一对 double sharing 通信 4 个元素
- 乘法计算：每个乘法门通信 $2n$ 个元素，平均每个人通信 2 个元素

平均每个乘法门每人需要通信 6 个元素

Double Sharing

开销优化

[GSZ20] 主要观察：

- 乘法计算阶段，P_{king} 可以发送 $e = xr - r$ 的分享 $\langle e \rangle_t$，而不是 e 本身。其他人收到 $\langle e \rangle_t$ 后计算 $\langle z \rangle_t := \langle e \rangle_t + \langle r \rangle_t$
- 由于 e 不需要保密，可令其中 t 个分片为 0，节省通信

平均每个乘法门每人需要通信 5.5 个元素

Double Sharing
开销优化

[Goy+21] 主要观察：

- 乘法计算阶段，P_{king} 依然发送 $e = xr - r$ 的分享 $\langle e \rangle_t$，但发送的是随机 Shamir 分享。
- 让每个人轮流做 P_{king}：
  - 如果 P_{king} 诚实，则随机 Shamir 分享 $\langle e \rangle_t$ 可以保证 $\langle z \rangle_t := \langle e \rangle_t + \langle r \rangle_t$ 也是随机的 Shamir 分享，此时 double sharing $\langle r \rangle_t$ 不必是随机的 Shamir 分享
  - 如果 P_{king} 是敌手，此时需要 double sharing $\langle r \rangle_t$ 是随机的 Shamir 分享
- 此时 t 对 double sharing 可以用于计算 n 个乘法门

平均每个乘法门每人需要通信 4 个元素

1. 背景介绍
2. GMW 协议
3. BGW 协议
4. 预处理技术
5. 参考文献

主要参考文献

1. [LN17] Yehuda Lindell and Ariel Nof. A Framework for Constructing Fast MPC over Arithmetic Circuits with Malicious Adversaries and an Honest-Majority. CCS 2017.
2. [AL17] Gilad Asharov and Yehuda Lindell. A Full Proof of the BGW Protocol for Perfectly Secure Multiparty Computation. J. Cryptol. 30(1): 58-151 (2017)
3. [EKR18]David Evans, Vladimir Kolesnikov, and Mike Rosulek. “A Pragmatic Introduction to Secure Multi-Party Computation”. In: Foundations and Trends in Privacy and Security 2.2-3 (2018), pp. 70–246.
4. [FY22] Dengguo Feng, and Kang Yang. "Concretely efficient secure multi-party computation protocols: survey and more." Security and Safety 1 (2022): 2021001.

参考文献 I

[Ara+16] Toshinori Araki et al. "High-Throughput Semi-Honest Secure Three-Party Computation with an Honest Majority". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*. 2016, pp. 805–817. DOI: 10.1145/2976749.2978331. URL: https://doi.org/10.1145/2976749.2978331.

[AL17] Gilad Asharov and Yehuda Lindell. "A Full Proof of the BGW Protocol for Perfectly Secure Multiparty Computation". In: *J. Cryptology* 30.1 (2017), pp. 58–151. DOI: 10.1007/s00145-015-9214-4. URL: https://doi.org/10.1007/s00145-015-9214-4.

参考文献 II

[Bea91] Donald Beaver. “Efficient Multiparty Protocols Using Circuit Randomization”. In: *Advances in Cryptology - CRYPTO ’91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*. 1991, pp. 420–432. DOI: 10.1007/3-540-46766-1_34. URL: https://doi.org/10.1007/3-540-46766-1_34.

参考文献 III

[BNO19] Aner Ben-Efraim, Michael Nielsen, and Eran Omri. "Turbospeedz: Double Your Online SPDZ! Improving SPDZ Using Function Dependent Preprocessing". In: *Applied Cryptography and Network Security - 17th International Conference, ACNS 2019, Bogota, Colombia, June 5-7, 2019, Proceedings*. Ed. by Robert H. Deng et al. Vol. 11464. Lecture Notes in Computer Science. Springer, 2019, pp. 530–549. DOI: 10.1007/978-3-030-21568-2_26. URL: https://doi.org/10.1007/978-3-030-21568-2_26.

参考文献 IV

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. "Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation (Extended Abstract)". In: *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*. 1988, pp. 1–10. DOI: 10.1145/62212.62213. URL: https://doi.org/10.1145/62212.62213.

参考文献 V

[Boy+19] Elette Boyle et al. "Practical Fully Secure Three-Party Computation via Sublinear Distributed Zero-Knowledge Proofs". In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*. 2019, pp. 869–886. DOI: 10.1145/3319535.3363227. URL: https://doi.org/10.1145/3319535.3363227.

参考文献 VI

[CDI05] Ronald Cramer, Ivan Damgård, and Yuval Ishai. “Share Conversion, Pseudorandom Secret-Sharing and Applications to Secure Computation”. In: *Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005, Proceedings*. 2005, pp. 342–362. DOI: 10.1007/978-3-540-30576-7_19. URL: https://doi.org/10.1007/978-3-540-30576-7_19.

参考文献 VII

[CDM00] Ronald Cramer, Ivan Damgård, and Ueli M. Maurer. "General Secure Multi-party Computation from any Linear Secret-Sharing Scheme". In: *Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding*. 2000, pp. 316–334. DOI: 10.1007/3-540-45539-6_22. URL: https://doi.org/10.1007/3-540-45539-6_22.

参考文献 VIII

[DN07] Ivan Damgård and Jesper Buus Nielsen. "Scalable and Unconditionally Secure Multiparty Computation". In: *Advances in Cryptology - CRYPTO 2007, 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007, Proceedings*. 2007, pp. 572–590. DOI: 10.1007/978-3-540-74143-5_32. URL: https://doi.org/10.1007/978-3-540-74143-5_32.

[EKR18] David Evans, Vladimir Kolesnikov, and Mike Rosulek. "A Pragmatic Introduction to Secure Multi-Party Computation". In: *Foundations and Trends in Privacy and Security* 2.2-3 (2018), pp. 70–246. DOI: 10.1561/3300000019. URL: https://doi.org/10.1561/3300000019.

参考文献 IX

[FY22] Dengguo Feng and Kang Yang. "Concretely efficient secure multi-party computation protocols: survey and more". In: *Security and Safety* 1 (2022), p. 2021001.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. "How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority". In: *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*. 1987, pp. 218–229. DOI: 10.1145/28395.28420. URL: https://doi.org/10.1145/28395.28420.

参考文献 X

[GSZ20] Vipul Goyal, Yifan Song, and Chenzhi Zhu. “Guaranteed Output Delivery Comes Free in Honest Majority MPC”. In: *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part II*. 2020, pp. 618–646. DOI: 10.1007/978-3-030-56880-1_22. URL: https://doi.org/10.1007/978-3-030-56880-1_22.

参考文献 XI

[Goy+21] Vipul Goyal et al. “ATLAS: Efficient and Scalable MPC in the Honest Majority Setting”. In: *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part II*. 2021, pp. 244–274. DOI: 10.1007/978-3-030-84245-1_9. URL: https://doi.org/10.1007/978-3-030-84245-1_9.

参考文献 XII

[LN17] Yehuda Lindell and Ariel Nof. “A Framework for Constructing Fast MPC over Arithmetic Circuits with Malicious Adversaries and an Honest-Majority”. In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*. 2017, pp. 259–276. DOI: 10.1145/3133956.3133999. URL: https://doi.org/10.1145/3133956.3133999.

Thanks!