恶意模型下的安全多方计算

Maliciously Secure Multiparty Computation

杨鹏

2022/06

摘要

安全多方计算（Secure Multiparty Computation，MPC）可以解决多个互不信任的参与方共同计算一个功能函数但不泄露各方隐私的问题。安全多方计算按照安全模型中定义的敌手类型大致可以分为半诚实模型下的安全多方计算和恶意模型下的安全多方计算两种。在半诚实安全多方计算中，敌手是半诚实的（Semi-honest），被其控制的参与方会试图获得诚实参与方输入但必须按照协议指令执行协议；在恶意安全多方计算中，敌手是恶意的（Malicious），被其控制的参与方可以任意偏离协议指令，以破坏协议的安全性。在更多的现实场景中，半诚实敌手的假设低估了敌手的能力，恶意敌手模型更能刻画敌手的性质。本文针对安全多方计算的恶意敌手模型进行研究，介绍和对比了恶意模型下的安全多方计算协议的几种构造范式。

1 引言

自从姚期智院士在 1982 年引入安全多方计算 [1] 的概念以来，已经有大量研究致力于构造恶意模型下的安全多方计算协议。在半诚实模型下，攻击者遵循安全多方计算协议的规则，试图通过检查计算过程中的交互记录，以得到更多超出允许范围的信息。与在半诚实模型下的攻击者不同，恶意模型下的攻击者可以任意偏离协议的规则，以破坏安全多方计算通用协议的安全性。因此，恶意敌手比半诚实敌手的行为更为复杂且更难以应对，这使得构造恶意安全多方计算协议比构造半诚实安全多方计算协议的难度更大，也更具有挑战性。为了构造安全多方计算通用协议以抵抗恶意攻击者，现有的研究工作大体上可以分为以下几种，表 1 总结了和对比了这几种抵抗恶意敌手的协议。

值得注意的是，表中所预处理模型是指基于秘密共享的安全多方计算，并采用预处理模型，利用 Beaver 乘法三元组进行运算。

文章的结构如下：第 2 节介绍了安全多方计算的一些基本知识，第 3 节详细介绍了上表提到的几种恶意安全多方计算协议，另外还介绍了其他范式的恶意安全多方计算协议，第 4 节总结了上述协议。

表 1: 恶意模型下安全多方计算协议

技术	参与方数量	协议轮数	低层协议
GMW 编译器	多方	常数轮	混淆电路协议
Cut-and-Choose 技术	两方	和使用的半诚实协议一致	任何半诚实安全多方计算协议
可认证秘密共享	多方	和电路深度成正比	基于预处理模型的安全多方计算协议
可认证混淆电路	多方	常数轮	BMR 协议
同态承诺	多方	和电路深度成正比	基于预处理模型的安全多方计算协议

2 预备知识

在本节中，将介绍一些预备知识，包括不经意传输基本概念（见 2.1 节），混淆电路协议（见 2.2 节）和秘密共享（见 2.3 节）。

2.1 不经意传输

不经意传输（Oblivious transfer，OT）是一种基本的加密原语，通常用作安全多方计算中的基本模块。下面，我们会介绍经典的 1-out-of-2 OT 协议，以及相关 OT 协议（Correlated OT，COT）。进一步地，我们还会介绍 OT 扩展（OT Extension）技术，其通过允许发送方和接收方以 λ 个基础 OT（例如 1-out-of-2 OT，COT）和 $O(m)$ 快速对称密钥操作为代价来实现 m 个 OT，从而最小化此成本，其中 λ 是安全参数，通常有 $\lambda \ll m$。

图 1 描述了 1-out-of-2 OT 和 COT，在这里要求发送方 S 不知道 b 的值，接收方 R 不知道 x_{1-b} 的值，$\perp$ 表示空字符串。

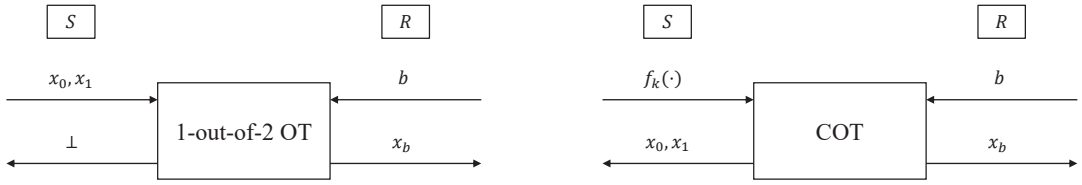


图 1: 1-out-of-2 OT 和 COT

经典的 OT 协议是我们下面介绍的 1-out-of-2 OT 协议，在该协议中，发送方 S 有两个输入 x_0 和 x_1，而接收方 R 有一个选择比特 b，并且想在不向 S 透露 b 和 R 不学习到任何其他信息的情况下获得 x_b。具体地，1-out-of-2 OT 可以通过协议 1 来实现，其是基于公钥密码学的。假设发送方 S 的输入为 l 比特的 x_0 和 x_1，接收方 R 输入为比特 b。我们用记号 $(\perp; x_b) \leftarrow (\mathsf{OT}(x_0, x_1; b))$ 表示实现此协议。

Protocol 1 Public key-based OT$(x_0, x_1; b)$

1: R 生成公钥和私钥 $(sk, pk) \leftarrow \mathsf{Gen}(1^n)$，然后随机选择一个公钥 $pk' \in_R \mathcal{PK}$，$\mathcal{PK}$ 是公钥空间。若 $b = 0$，则将 (pk, pk') 发送给 S；若 $b = 1$，则将 (pk', pk) 发送给 S。

2: S 将收到的公钥记为 (pk_0, pk_1)，然后将密文 $c_0 = \mathsf{Enc}_{pk_0}(x_0), c_1 = \mathsf{Enc}_{pk_1}(x_1)$ 发送给 R。

3: R 收到 c_0, c_1，然后用私钥 sk 解密密文 c_b，得到相应明文 x_b。

1-out-of-2 OT 协议的实现涉及到昂贵的公钥操作，因此 OT 扩展（OT Extension）技术被提出，其基本思想是用 λ 个基础 OT（例如 1-out-of-2 OT，和后面讲述的 COT）和 $O(m)$ 快速对称密钥操作为代价来实现 m 个 OT，通常 $\lambda \ll m$。半诚实模型下的 OT 扩展的实现如协议 2 所示。

Protocol 2 OT-Extension()

输入: S 的输入为 m 对 n 比特长的字符串 $(x_1^0, x_1^1), ..., (x_m^0, x_m^1)$；$R$ 的输入为 m 个选择比特 $\mathbf{r} = (r_1, ..., r_m)$；公共输入为对称密码安全参数 λ 和基础 OT 的数量 l，且 $l = \lambda$；以及基础 OT，伪随机数生成器 $G : \{0,1\}^\lambda \to \{0,1\}^m$，函数 $H : [m] \times \{0,1\}^l \to \{0,1\}^n$。

输出: S 没有输出，R 的输出为 $(x_1, ..., x_m) = (x_j^{r_1}, ..., x_j^{r_m})$。

1: （初始化 OT 阶段）

2: S 初始化一个随机向量 $\mathbf{s} = (s_1, .., s_l) \in \{0,1\}^l$；$R$ 选择 l 对种子 $(\mathbf{k}_i^0, \mathbf{k}_i^1) \in \{0,1\}^\lambda$。

3: **for** $i = 1$ to l **do**

4: 双方运行 1 个基础 OT，R 作为发送方，输入 $(\mathbf{k}_i^0, \mathbf{k}_i^1)$；$S$ 作为接收方，输入 s_i，得到 $\mathbf{k}_i^{s_i}$。

5: **end for**

6: R 对 $1 \le i \le l$ 计算 $\mathbf{t}^i = G(\mathbf{k}_i^0)$，其中 $\mathbf{t}^i$ 为 m 维列向量。将所有 $\mathbf{t}^i$ 组成 $m \times l$ 的矩阵 T，将矩阵 T 的第 i 列记为 $\mathbf{t}^i$，其中 $1 \le i \le l$；第 j 行记为 $\mathbf{t}_j$，其中 $1 \le j \le m$。

7: （OT 扩展阶段）

8: R 计算 $\mathbf{u}^i = \mathbf{t}^i \oplus G(\mathbf{k}_i^1) \oplus \mathbf{r}$，并将 $\mathbf{u}^1, ..., \mathbf{u}^l$ 发送给 S。

9: 对 $1 \le i \le l$，S 计算 $\mathbf{q}^i = (s_i \cdot \mathbf{u}^i) \oplus G(\mathbf{k}_i^{s_i})$。注意到 $\mathbf{t}^i = G(\mathbf{k}_i^0), \mathbf{u}^i = \mathbf{t}^i \oplus G(\mathbf{k}_i^1) \oplus \mathbf{r}$，当 $s_i = 0$ 时，$\mathbf{q}^i = G(\mathbf{k}_i^0) = \mathbf{t}^i$，当 $s_i = 1$ 时，$\mathbf{q}^i = \mathbf{t}^i \oplus \mathbf{r}$。总之 $\mathbf{q}^i = (s_i \cdot \mathbf{r}) \oplus \mathbf{t}^i$，其中 $\mathbf{q}^i$ 为 m 维列向量。

10: S 将所有 $\mathbf{q}^i$ 组成 $m \times l$ 的矩阵 Q，将矩阵 Q 的第 i 列记为 $\mathbf{q}^i$，其中 $1 \le i \le l$；第 j 行记为 $\mathbf{q}_j$，其中 $1 \le j \le m$。可以验证 $\mathbf{q}^i = (s_i \cdot \mathbf{r}) \oplus \mathbf{t}^i, \mathbf{q}_j = (r_j \cdot \mathbf{s}) \oplus \mathbf{t}_j$。

11: **for** $j = 1$ to m **do**

12: S 计算 $y_j^0 = x_j^0 \oplus H(j, \mathbf{q}_j), y_j^1 = x_j^1 \oplus H(j, \mathbf{q}_j \oplus \mathbf{s})$，并将 (y_j^0, y_j^1) 发送给 S。

13: R 计算得到 $x_j = y_j^{r_j} \oplus H(j, \mathbf{t}_j)$。

14: **end for**

可以验证，最终 R 得到的 $(x_1, ..., x_j) = (x_j^{r_1}, x_j^{r_m})$，这是用 $r_1, ..., r_m$ 作为选择比特从 S 的 m 对 n 比特长的字符串 (x_j^0, x_j^1)（其中 $1 \le j \le m$）选择得到的对应的结果，这一结果本来需要用 m 个 OT，而现在只用了 l 个 OT。

尽管如此，我们还是想进一步降低通信量，算法 2 中的第 11 步到第 14 步，一轮交互的通信量为 $2n + \lambda$，计算量为 3 个哈希函数，针对于此我们想进一步减少通信量。我们注意到前述的 COT（见图 1）有更少的通信量，使用 COT 的一轮交互的通信量为 $n + \lambda$，计算量为 3 个哈希函数。

具体地，我们只需要针对算法 2 的第 11 步到第 14 步进行下面的修改，修改后的 OT 扩展见协议 3 。

Protocol 3 COT-Extension()

输入: S 的输入为函数 $f_{k_1}() \ldots f_{k_m}()$；$R$ 的输入为 m 个选择比特 $\mathbf{r} = (r_1, ..., r_m)$；公共输入为对称密码安全参数 λ 和基础 OT 的数量 l，且 $l = \lambda$；以及基础 OT，伪随机数生成器 $G : \{0,1\}^\lambda \to \{0,1\}^m$，函数 $H : [m] \times \{0,1\}^l \to \{0,1\}^n$。

输出: S 输出 $(x_1^0, x_1^1), ..., (x_m^0, x_m^1)$，$R$ 的输出 $(x_1, ..., x_m) = (x_j^{r_1}, ..., x_j^{r_m})$。

1: ...

2: **for** $j = 1$ to m **do**

3: S 计算 $x_j^0 = H(j, \mathbf{q}_j), x_j^1 = f_{k_j}(x_j^0)$，并将 $y_j = x_j^1 \oplus H(j, \mathbf{q}_j \oplus \mathbf{s})$ 发送给 R。

4: R 根据 r_j 计算：若 $r_j = 0$，计算 $x_j = H(j, \mathbf{t}_j)$；否则，计算 $x_j = y_j \oplus H(j, \mathbf{t}_j)$。

5: **end for**

2.2 混淆电路协议

混淆电路协议（Yao's Garbled Circuit Protocol）是姚期智于 1986 年提出的半诚实安全两方计算协议 [2]。在通信轮数上，混淆电路协议只需要执行常数轮交互，因此协议的通信延迟很小，但混淆电路协议的通信量很大，因此通信复杂度并不是最优的。但尽管如此，很多安全多方计算协议（如 BMR 协议）都是在混淆电路协议的基础上进行构造的[1]。

2.2.1 混淆电路协议的基本思路

混淆电路协议的动机是十分自然的。我们考虑两个参与方的情况，P_0 持有自己的输入 $x \in X$，P_1 持有自己的输入 $y \in Y$，两个参与方希望共同计算功能函数 $f(x, y)$，其中 X, Y 分别表示 P_0, P_1 的输入域。

本质上，功能函数 f 是一个映射，它可以用查找表（look-up table）的形式表示。我们用 $X \cdot Y$ 行的查找表 T 来表示功能函数 f，其中每行的条目 $T_{x,y} = \langle f(x, y) \rangle$，$f(x, y)$ 的输出可以通过简单地从相应行中检索 $T_{x,y}$ 而得到。因此上述安全两方计算的任务实际上就变成了求相应行的查找表的值，具体来说可以通过如下方式进行：

P_0 为每一个可能的输入 x 和 y 随机指定一个强密钥来加密 T，即对于每一个 $x \in X$ 和 $y \in Y$，P_0 将选择 $k_x \in_R \{0,1\}^\kappa$ 和 $k_y \in_R \{0,1\}^\kappa$。P_0 同时使用两个密钥 k_x 和 k_y 加密查找表 T 中相应行的条目 $T_{x,y}$，并将加密后（且经过随机置换）的查找表 $\langle \mathsf{Enc}_{k_x,k_y}(T_{x,y}) \rangle$ 发送给 P_1。

下面的任务就是 P_1 解密与参与方输入相关联的行数据项 $T_{x,y}$，并且只能解密该数据项而不能解密其他数据项。具体的实现方式就是让 P_0 向 P_1 发送密钥 k_x 和 k_y。由于 P_0 已知自己的输入 x，因此只需要将密钥 k_x 直接发送给 P_1 即可。但 P_0 不知道 P_1 的输入 y，因此 P_0 和 P_1 需要执行一个 $|Y|$ 选 1 的 OT 协议，P_0 将 k_y 发送给 P_1。一旦收到 k_x 和 k_y，就可以使用这些密钥解密 $T_{x,y}$，得到输出 $f(x, y)$。在这里，非常关键的是，P_1 不应该也不能得到其他信息，因为 P_1 只有一对密钥，只能解密查找表中的一行条目。值得注意的是，单独使用 k_x 或 k_y 都不允许部分解密密文，甚至不能单独使用 k_x 或 k_y 判断某个密文是否是用 k_x 或 k_y 加密得到的。

[1] 文献 [2] 只讲述了混淆电路协议的思想，并没有讲述混淆电路协议具体流程，甚至没有给该协议具体命名，我们下面讲述的混淆电路协议更像是一个集大成者，包含诸多论文的思想。

但也出现了另外一个问题，即 P_1 怎么知道应该解密查找表 T 的哪一行。这一行依赖于两个参与方的输入，因此该信息本身也是敏感的。解决这个问题的最简单的方法是在 T 的加密条目中加入一些附加信息。例如，P_0 可以在 T 的每一行字符串末尾填充 δ 个 0。如果解密了错误的行，则解密结果的末尾仅有很低的概率（$p = 2^{-\delta}$）包含 δ 个 0，这样 P_1 就可以知道哪些解密结果是错误的。但这样的方案是低效的，因为不仅使得加密条目的长度增加，而且平均要解密查找表 T 中至少一半的条目。

Beaver 等人 [3] 在 1990 年提出了一种更好的解决方案称为标识置换（Point-and-Permute），该方案的基本思想是将密钥的一部分（第一个密钥的后 $\lceil \log|X| \rceil$ 比特和第二个密钥的后 $\lceil \log|Y| \rceil$ 比特）作为查找表 T 的置换标识，标识密钥应该用于加密哪行密文，根据置换标识对加密后的查找表进行置换。为了避免查找表的各行在分配过程中产生冲突，P_0 必须保证置换标识不会在 k_x 的密钥空间和 k_y 的密钥空间中产生冲突，可以通过多种方式实现这一点。严格来说，密钥长度必须要达到相应的安全等级。因此，参与方并不会直接把密钥的一部分作为置换标识，而是将置换标识附加在密钥之后，使密钥满足所需的长度需求。

从上面的讨论就可以看到，当 $|X|$ 和 $|Y|$ 很大时，查找表的大小是不可接受的，因为查找表的大小与 f 的定义域大小呈线性关系。我们很自然地想到了布尔电路表示，对于单个布尔电路门这样的小型函数，其输入只能为 $\{0,1\}$，此时查找表的大小为 4。用查找表表示此类小型函数是比较高效的，因此我们希望将前述的功能函数转换为布尔电路，然后基于布尔电路计算该函数。

与前面的描述一样，P_0 生成密钥并加密查找表，而 P_1 在不知道密钥和明文之间关系的条件下解密查找表。但在这种情况下，我们不能向 P_1 披露中间门的明文输出，我们可以让中间门的输出也是与明文一一对应的密钥。由于 P_2 不知道密文与明文之间的关联关系，因此这样可以达到隐藏中间门明文输出的目的。

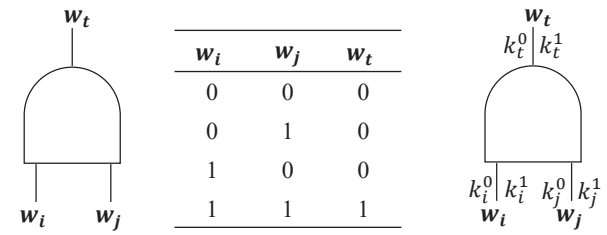
2.2.2 混淆电路协议的执行过程

假设我们已经将功能函数 f 表示为布尔电路，现在要计算该布尔电路。在混淆电路协议中，有两个参与方共同计算一个布尔电路，一个称为电路生成方（Circuit Generator），一个称为电路求值方（Circuit Evaluator），我们分别用 P_0 和 P_1 表示。

对于布尔电路 $\mathcal{C}$ 的每条输入线（Input Wire）和输出线（Output Wire）w_i，P_0 需要为其指定两个密钥 k_i^0 和 k_i^1，两个密钥分别与输入线/输出线的两个可能的明文值（0/1）相关联。我们将这些密钥称为线标签（Wire Label），将这些输入线/输出线的明文值称为线路值（Wire Value）。在对电路求值的过程中，根据计算过程中的输入，每条输入/输出线都将与一个特定的明文值和相应的线标签相关联，我们将此明文值称为激活值（Active Value），将于激活值对应的线标签称为激活标签（Active Label）。在整个协议执行过程中，电路求值方只知道激活标签，而不知道激活标签对应的激活值和非激活标签。

以与门电路为例，图 2 中的（a）和（b）是与门的图示及其真值表。该与门电路其有输入线 w_i, w_j 和输出线 w_t，P_0 首先为输入线/输出线指定密钥，如（c）所示，然后用输入线的密钥（线标签）加密输出线的密钥（线标签），加密的规则是用 k_i^x 和 k_j^y 加密 k_t^z，其中 $z = x \ \mathsf{AND} \ y$，如图（d）所示。最后，取出加密后的结果，并置换得到查找表，如（e）所示。

通常我们会把上述过程得到的查找表称为混淆表（Garbled Table），生成混淆表后，P_0 将所有混淆



w_i	w_j	w_t
0	0	0
0	1	0
1	0	0
1	1	1

(a) 与门的图示　(b) 与门的真值表　(c) 指定密钥

w_i	w_j	w_t
k_i^0	k_j^0	$E_{k_i^0}(E_{k_j^0}(k_t^0))$
k_i^0	k_j^1	$E_{k_i^0}(E_{k_j^1}(k_t^0))$
k_i^1	k_j^0	$E_{k_i^1}(E_{k_j^0}(k_t^0))$
k_i^1	k_j^1	$E_{k_i^1}(E_{k_j^1}(k_t^1))$

(d) 构建查找表

$E_{k_i^1}(E_{k_j^1}(k_t^1))$
$E_{k_i^0}(E_{k_j^0}(k_t^0))$
$E_{k_i^1}(E_{k_j^0}(k_t^0))$
$E_{k_i^0}(E_{k_j^1}(k_t^0))$

(e) 查找表

图 2: 混淆表生成

表发送给 P_1。此外，P_0 会将所有输入线的激活值对应的激活标签发送给 P_1。对于电路中属于 P_0 的输入线，P_0 直接将对应的激活标签发送给 P_1，而对于电路中属于 P_1 的输入线，P_0 和 P_1 通过运行一个 1-out-of-2 协议将对应的激活标签发送给 P_1。

在收到混淆表和输入线标签后，P_1 开始对电路求值。如上所述，P_1 必须能够正确解密每个混淆表中所需解密的数据行，通过前述的标识置换（Point-and-Permute）技术可以实现这一点。在混淆表只包含 4 行密文的情况下，标识置换技术非常简单和高效，因为此时每个输入的标识置换只有 1 比特长（$\lceil \log|X| \rceil = \lceil \log|Y| \rceil = 1$），故混淆表中的每行条目总共需要 2 比特长的置换标识。最终，P_1 完成混淆电路的求值，并得到电路输出线标签，并将得到的密钥发送给 P_0 进行解密，即可计算得到最终的结果。

不过，我们希望 P_1 可以不用将得到的密钥发送给 P_0 进行解密，以减少一轮的通信。具体方法是让 P_0 在发送混淆电路的同时，发送输出线的解码表（即最终输出线标签和真实值的对应关系表），此时得到输出线标签的 P_1 就可以在解码表中直接找到线标签对应的真实值。

协议 4 展示了混淆电路协议中混淆表的生成过程，而协议 5 展示了混淆电路协议的完整流程。

Protocol 4 GC-Generation

参数: 实现功能函数 f 的布尔电路 $\mathcal{C}$，安全参数 κ。

1: 生成线标签：对于布尔电路 $\mathcal{C}$ 的每条输入线/输出线，P_0 随机选择线标签：$w_i^b = (k_i^b, p_i^b)$，其中有密钥 $k_i^b \in_R \{0,1\}^\kappa$ 和置换标识位 $p_i^b \in_R \{0,1\}$，且 $p_i^b = 1 - p_i^{1-b}$。

2: 混淆电路生成：对于布尔电路 $\mathcal{C}$ 的每一个门 G_i，按照拓扑顺序执行下述操作：

(a) 假设 G_i 是一个实现函数 $g_i : w_c = g_i(w_a, w_b)$ 的 2-输入布尔门，其中输入线标签为 $w_a^0 = (k_a^0, p_a^0), w_a^1 = (k_a^1, p_a^1), w_b^0 = (k_b^0, p_b^0), w_b^1 = (k_b^1, p_b^1)$，输出线标签为 $w_c^0 = (k_c^0, p_c^0), w_c^1 = (k_c^1, p_c^1)$。

(b) 构造 G_i 的混淆表，G_i 的输入值为 $v_a, v_b \in \{0,1\}$，输入值共有 2^2 中可能的组合。对于每一种组合，设置混淆表的条目为 $e_{v_a,v_b} = H(k_a^{v_a} || k_b^{v_b} || i) \oplus w_c^{g_i(v_a,v_b)}$，然后根据输入线标签上的置换标识位对混淆表中的所有条目 e 进行排位，将条目 e_{v_a,v_b} 放置在位置 $\langle p_a^{v_a}, p_b^{v_b} \rangle$ 上。

3: 输出解码表：对于每一条最终的输出线 w_i（假设此输出线是门 G_j 的输出线），假设其对应的线标签为 $w_i^0 = (k_i^0, p_i^0), w_i^1 = (k_i^1, p_i^1)$，要为两个可能的真实值 $v \in \{0,1\}$ 创建解码表。设置混淆表的条目为 $e_v = H(k_i^v || \text{“out”} \ || j) \oplus v$，然后根据线标签的置换标识位对解码表中的条目 e 排序，将条目 e_v 放置在位置 p_i^v 上（因为 $p_i^1 = p_i^0 \oplus 1$，所以放置的位置不会发生冲突）。

从协议 4 可以看出，我们用 $e_{v_a,v_b} = H(k_a^{v_a}||k_b^{v_b}||i) \oplus w_c^{g_i(v_a,v_b)}$ 实现了前述的加密过程。另外，注意到 $e_v = H(k_i^v||\text{“out”}\ ||j) \oplus v$ 的异或操作，这里只需要使用 H 输出的最低位生成 e_v。

Protocol 5 Garbled-Circuit

参数: 实现功能函数 f 的布尔电路 $\mathcal{C}$，安全参数 κ，以及参与方 P_0 的输入 $x \in \{0,1\}^n$ 和 P_1 的输入 $y \in \{0,1\}^n$。

1: P_0 作为电路生成方，按照协议 4 生成混淆表，然后将获得的混淆电路 $\hat{\mathcal{C}}$（混淆表和解码表）全部发送给 P_1。

2: 对于 P_0 需要提供输入值的输入线，P_0 向 P_1 发送其输入值对应的激活标签。对于 P_1 需要提供输入值的输入线，P_0 和 P_1 需要运行一个 1-out-of-2 OT 协议：P_0 作为发送方，输入该输入线的两个标签；而 P_1 作为接收方，输入自己的输入值，获得对应的激活标签。

3: P_1 从输入线的激活标签开始，按照拓扑顺序逐门对收到的混淆电路 $\hat{\mathcal{C}}$ 进行求值。对于混淆表为 $T = (e_{0,0}, ..., e_{1,1})$，输入线的激活标签为 $w_a = (k_a, p_a)$ 和 $w_b = (k_b, p_b)$ 的门 G_i，P_1 计算输出线的激活标签 $w_c = (k_c, p_c)$：$w_c = H(k_a||k_b||i) \oplus e_{p_a,p_b}$

4: 然后 P_1 应用解码表得到输出：当混淆电路 $\hat{\mathcal{C}}$ 中的所有门完成求值后，P_1 将第 2 个密钥设置为 “out”，解码最终的输出门，得到明文计算结果。P_1 将得到的计算结果发送给 P_0，双方均将计算结果作为协议的输出。

在这里，一定要注意的是，我们解码表对应的输出线一定是最终计算结果对应的输出线，而不是中间结果对应的输出线。

2.2.3 混淆电路技术的优化：FreeXOR 技术

执行混淆电路协议的主要开销是传输混淆表所需要的网络带宽开销，以及混淆电路生成和求值过程中所需的计算开销，在典型场景中（局域网/广域网条件下，使用智能手机或笔记本等具有中等计算资源的设备执行协议），混淆电路协议的主要开销是网络带宽开销。协议 5 已经使用了许多优化技术，例如 Point-and-Permute 技术等，近些年许多论文也介绍了很多优化技术，这里主要讲述 FreeXOR 技术。

FreeXOR 技术是由 Kolesnikov 和 Schneider[4] 在 2008 年提出的混淆电路优化技术。FreeXOR 技术是将 Gate Evaluation Secret Sharing（GESS）方案的 XOR 门生成技术引入到混淆电路中，通过调整混淆电路秘密值的生成过程来解决直接引入此技术引发的正确性问题。Gate Evaluation Secret Sharing（GESS）方案是 Kolesnikov 提出的信息论安全的混淆电路方案，具体细节请参见 [5, 6]。

FreeXOR 要求生成混淆电路所有输入线/输出线标签时，都要使用相同的偏置 Δ，这样就可以将 GESS 方案的 XOR 门构造方法引入到混淆电路中。具体来说，对于混淆电路 $\hat{\mathcal{C}}$ 的每一条线 w_i 的每一对线标签 w_i^0, w_i^1，我们要求 $w_i^0 \oplus w_i^1 = \Delta$，其中 $\Delta \in_R \{0,1\}^\kappa$ 是预先随机选取的。这样，我们就让线标签有了相关性，正是因为这一点我们就可以让 XOR 门正确地计算输出线的标签。

协议 6 展示了 FreeXOR 技术优化的混淆电路生成协议。

Protocol 6 Garbled-Circuit-FreeXOR

参数: 实现功能函数 f 的布尔电路 $\mathcal{C}$，安全参数 κ，以及哈希函数 $H : \{0,1\}^* \to \{0,1\}^{\kappa+1}$。

1: 随机选择全局密钥偏置 $\Delta \in_R \{0,1\}^\kappa$。

2: 对于 $\mathcal{C}$ 中的每一条输入线 w_i，随机选择 0 所对应的输入线标签 $w_i^0 = (k_i^0, p_i^0) \in_R \{0,1\}^{\kappa+1}$，将另一条输入线标签设置为 $w_i^1 = (k_i^1, p_i^1) = (k_i^0 \oplus \Delta, p_i^1 \oplus 1)$

3: 按照拓扑顺序对 $\mathcal{C}$ 的每个门执行下述步骤：

(a) 如果 G_i 是一个输入线标签为 $w_a^0 = (k_a^0, p_a^0), w_a^1 = (k_a^1, p_a^1), w_b^0 = (k_b^0, p_b^0), w_b^1 = (k_b^1, p_b^1)$ 的 XOR 门 $w_c = \mathsf{XOR}(w_a, w_b)$：

(i) 将 0 所对应的输出线标签设置为 $w_c^0 = (k_a^0 \oplus k_b^0, p_a^0 \oplus p_b^0)$。

(ii) 将 1 所对应的输出线标签设置为 $w_c^1 = (k_a^0 \oplus k_b^0 \oplus \Delta, p_a^0 \oplus p_b^0 \oplus 1)$。

(b) 如果 G_i 是一个输入线标签为 $w_a^0 = (k_a^0, p_a^0), w_a^1 = (k_a^1, p_a^1), w_b^0 = (k_b^0, p_b^0), w_b^1 = (k_b^1, p_b^1)$ 的 2-输入门 $w_c = g_i(w_a, w_b)$：

(i) 随机选择 0 所对应的输出线标签 $w_c^0 = (k_c^0, p_c^0) \in_R \{0,1\}^{\kappa}$。

(ii) 将 1 所对应的输出线标签设置为 $w_c^1 = (k_c^1, p_c^1) = (k_c^0 \oplus \Delta, p_c^0 \oplus 1)$。

(iii) 创建 G_i 的混淆表。G_i 的输入值为 $v_a, v_b \in \{0,1\}$ 共有 2^2 中可能的组合。对于每一种组合，设置混淆表的条目为 $e_{v_a,v_b} = H(k_a^{v_a} || k_b^{v_b} || i) \oplus w_c^{g_i(v_a,v_b)}$。然后根据输入线标签上的置换标识位对混淆表中的所有条目 e 进行排位，将条目 e_{v_a,v_b} 放置在位置 $\langle p_a^{v_a}, p_b^{v_b} \rangle$ 上。

4: 按照协议 4 的方法计算输出解码表。

FreeXOR 混淆电路协议的执行过程和标准姚氏混淆电路协议 5 完全一样，唯一的区别就是在第 3 步中，P_1 不需要为 XOR 门进行任何加密和解密处理，对于输入线标签为 $w_a = (k_a, p_a)$ 和 $w_b = (k_b, p_b)$ 的 XOR 门 G_i，可以直接计算 $(k_a \oplus k_b, p_a \oplus p_b)$ 得到输出线标签。可以看到，计算 XOR 门无需更多的计算，因此计算 XOR 门是 “Free” 的。

2.3 基于秘密共享的安全多方计算

2.3.1 秘密共享基本概念

秘密共享（Secret Sharing，SS）是安全多方计算的基础模块之一，基于秘密共享的安全多方计算是解决安全多方计算问题的基本范式之一。

秘密共享主要有两个阶段，即秘密分发阶段和秘密重构阶段：假设我们有一个秘密值 s

1. 秘密分发阶段：秘密管理员将秘密 s 分发给秘密参与方 $P_1, P_2, ..., P_n$，每个秘密参与方手中会持有秘密的一部分，我们称之为秘密份额，记作 $\langle s \rangle$，秘密参与方 P_i 手中的秘密份额为 $\langle s \rangle_i$，其中 $i \in \{1, ..., n\}$；

2. 秘密重构阶段：所有秘密参与方将手中的秘密份额 $\langle s \rangle_1, \langle s \rangle_2, ..., \langle s \rangle_n$ 发送给秘密管理员，秘密管理员基于所有的秘密份额重构得到秘密 s

值得注意的是，其中的秘密管理员也可以是秘密参与方，也即秘密参与方将自己的秘密分发给自己和其他参与方，这一种形式的秘密共享是后面经常使用到的。

关于秘密共享的密码原语有很多，如门限秘密共享、加性秘密共享、复制秘密共享、同态秘密共享、函数秘密共享等，本文中讲述的秘密共享是加性秘密共享（Additive Secret Sharing）。接下来，我们以加性秘密共享为例，形式化地定义秘密共享的秘密分发操作和秘密重构操作。

在加性秘密共享方案中，其常常定义在一个环上，秘密值是环上的一个元素，环上有模算术加法和模算术乘法运算。设有环 $\mathbb{Z}_{2^l}$，秘密值 $s \in \mathbb{Z}_{2^l}$，其长度为 l 比特。特别地，我们假设 $n = 2$，也即只有两个秘密参与方 P_0 和 P_1。加性秘密共享方案的秘密分发和重构操作有：

1. 秘密分发（$\mathsf{Shr}(s)$）：秘密管理员随机选择 $r \in_R \mathbb{Z}_{2^l}$，然后将 $\langle s \rangle_0 = r$ 发送给 P_0，将 $\langle s \rangle_1 = (s - r) \bmod 2^l$ 发送给 P_1；

2. 秘密重构（$\mathsf{Rec}(s)$）：秘密参与方 P_0 将秘密份额 $\langle s \rangle_0$ 发送给秘密管理员，秘密参与方 P_1 将秘密份额 $\langle s \rangle_1$ 发送给秘密管理员，秘密管理员计算 $s = (\langle s \rangle_0 + \langle s \rangle_1) \bmod 2^l$，重构得到秘密 s。

在这里，由于秘密份额满足 $(\langle s \rangle_0 + \langle s \rangle_0) \bmod 2^l = s$，因此才称为加性秘密共享（为了简化表述，我们省去模算术运算）。

2.3.2 Beaver 乘法三元组

构造安全多方计算协议的一个常见范式就是将安全多方计算协议划分为预处理阶段（参与方输入未知时）和在线阶段（参与方选择好输入时）。预处理阶段会为各个参与方生成一些相互之间具有一定关联性的随机量，然后参与方可以于在线阶段“消耗”这些随机量。目前，基于秘密共享的安全多方计算大多也采用这种范式。从下一节可以看到，基于秘密共享的安全多方计算实现基本运算（比如加法和乘法运算）时主要的计算代价在于乘法运算（加法运算可以在本地进行），基于上述秘密份额进行乘法运算需要进行多轮交互，一个直观的想法就是能不能将这些交互（部分地）转移到预处理阶段。Beaver 在 1991 年提出了一种非常聪明的方法，可以将乘法运算的大部分通信量都转移到预处理阶段 [7]。

Beaver 乘法三元组指的是秘密份额三元组 $\langle u \rangle, \langle v \rangle, \langle z \rangle$，其中 u 和 v 是从某个适当的环/域选择出来的随机数，而 $z = uv$。在这里，我们不介绍如何生成乘法三元组，只介绍如何利用该乘法三元组辅助我们的乘法运算。

我们考虑下面的场景：假设秘密值 a 和 b 已经通过算术秘密共享的方式秘密分发给 P_0 和 P_1，我们希望 P_0 和 P_1 计算得到 $a \cdot b$ 的秘密份额 $\langle a \cdot b \rangle$，然后这些秘密份额就可以重构出 $a \cdot b$。假设 P_0 和 P_1 已经共享了 $\langle u \rangle, \langle v \rangle, \langle z \rangle$，其中 u, v 是在 $\mathbb{Z}_{2^l}$ 上均匀分布的随机值，并且 $z = uv$。此时 P_0 有 $\langle a \rangle_0, \langle b \rangle_0, \langle u \rangle_0, \langle v \rangle_0, \langle z \rangle_0$，$P_1$ 有 $\langle a \rangle_1, \langle b \rangle_1, \langle u \rangle_1, \langle v \rangle_1, \langle z \rangle_1$。接下来，$P_0$ 和 P_1 进行下面的计算：

1. P_0 计算 $\langle e \rangle_0 = \langle a \rangle_0 - \langle u \rangle_0, \langle f \rangle_0 = \langle b \rangle_0 - \langle v \rangle_0$，并发送给 P_1；P_1 计算 $\langle e \rangle_1 = \langle a \rangle_1 - \langle u \rangle_1, \langle f \rangle_1 = \langle b \rangle_1 - \langle v \rangle_1$，并发送给 P_0。然后 P_0 和 P_1 分别在本地重构得到 $e = a - u, f = b - v$。

2. 观察到 $c = a \cdot b = (a - u + u) \cdot (b - v + v) = (e + u) \cdot (f + v) = e \cdot f + e \cdot v + u \cdot f + u \cdot v$，而 P_0 和 P_1 已知 e, f，分别有秘密份额 $\langle u \rangle, \langle v \rangle, \langle z \rangle$，因此有 $\langle c \rangle = e \cdot f + e \cdot \langle v \rangle + \langle u \rangle \cdot f + \langle z \rangle$，我们可以令 $\langle c \rangle_0 = e \cdot \langle v \rangle_0 + \langle u \rangle_0 \cdot f + \langle z \rangle_0, \langle c \rangle_1 = e \cdot f + e \cdot \langle v \rangle_1 + \langle u \rangle_1 \cdot f + \langle z \rangle_1$，满足 $\langle c \rangle_0 + \langle c \rangle_1 = c = a \cdot b$。

尽管我们是用加性秘密共享来描述 Beaver 乘法三元组，而且仅考虑了两个参与方，但实际上只需秘密共享方案满足下面的特性就可以使用 Beaver 乘法三元组 [8]：

1. 加同态性 (Additive homomorphism)：给定秘密份额 $\langle a\rangle, \langle b\rangle$ 和公开值 c，那么参与方可以本地计算 $\langle a+b\rangle, \langle a+c\rangle$ 和 $\langle c\cdot a\rangle$。
2. 可打开性 (Opening)：给定秘密份额 $\langle a\rangle$，参与方可以选择向所有其他参与方披露 a。
3. 隐私性 (Privacy)：无论哪种攻击者，都无法从 $\langle a\rangle$ 中得到与 a 相关的任何信息。
4. 乘法三元组生成 (Beaver triples)：各个参与方可以为每一个乘法运算生成满足 $z=uv$ 的随机乘法三元组 $\langle u\rangle, \langle v\rangle, \langle z\rangle$。
5. 随机输入工具 (random input gadgets)：对于参与方 P_i 而言，各个参与方都可以得到一个随机秘密份额 $\langle r\rangle$，秘密份额 $\langle r\rangle$ 对于除 P_i 的所有参与方来说是随机的，只有 P_i 知道背后的秘密 r。在协议执行过程中，当 P_i 选择好自己的输入 a 后，P_i 可以向所有其他参与方公开 $\delta=a-r$（但这不会泄露关于 a 的任何信息），参与方可以利用加同态性在本地计算 P_i 输入 a 的秘密份额 $\langle a\rangle=\langle r\rangle+\delta$

只要秘密共享方案满足上述所有性质，Beaver 乘法三元组方法就是安全的。进一步地，只要该秘密共享方案在恶意攻击下仍然满足可重构性和隐私性，那么 Beaver 乘法三元组也可以抵御恶意敌手的攻击，该恶意敌手无法伪造出未披露的 a。

2.3.3 秘密份额上的加法和乘法运算

下面我们介绍加性秘密共享的加法和乘法运算，我们考虑下面的场景：假设秘密值 a 和 b 已经通过算术秘密共享的方式秘密分发给 P_0 和 P_1，我们希望 P_0 和 P_1 计算得到 $a+b$ 的秘密份额 $\langle a+b\rangle$，和 $a\cdot b$ 的秘密份额 $\langle a\cdot b\rangle$，然后这些秘密份额就可以重构出 $a+b$ 和 $a\cdot b$。

对于加法运算，P_0 和 P_1 只需要在本地对秘密份额相加即可。对于乘法运算，常用的方法是利用乘法三元组进行计算。假设 P_0 和 P_1 已经共享了 $\langle u\rangle, \langle v\rangle, \langle z\rangle$，其中 u, v 是在 $\mathbb{Z}_{2^l}$ 上均匀分布的随机值，并且 $z=uv$。此时 P_0 有 $\langle a\rangle_0, \langle b\rangle_0, \langle u\rangle_0, \langle v\rangle_0, \langle z\rangle_0$，$P_1$ 有 $\langle a\rangle_1, \langle b\rangle_1, \langle u\rangle_1, \langle v\rangle_1, \langle z\rangle_1$。下面是具体的计算过程：

- 秘密份额上的加法运算（$\mathsf{ADD}(\cdot,\cdot)$）：$P_0$ 计算 $\langle c\rangle_0=\langle a\rangle_0+\langle b\rangle_0$；$P_1$ 计算 $\langle c\rangle_1=\langle a\rangle_1+\langle b\rangle_1$ 即可。
- 秘密份额上的乘法运算（$\mathsf{Mul}(\cdot,\cdot)$）：$P_0$ 计算 $\langle e\rangle_0=\langle a\rangle_0-\langle u\rangle_0, \langle f\rangle_0=\langle b\rangle_0-\langle v\rangle_0$，并发送给 P_1；P_1 计算 $\langle e\rangle_1=\langle a\rangle_1-\langle u\rangle_1, \langle f\rangle_1=\langle b\rangle_1-\langle v\rangle_1$，并发送给 P_0。然后 P_0 和 P_1 分别在本地计算 $\langle e\rangle=\langle a\rangle_i-\langle u\rangle_i, \langle f\rangle_i=\langle b\rangle_i-\langle v\rangle_i$，其中 $i\in\{0,1\}$，此时 P_0 有 $\langle e\rangle_0, \langle e\rangle_1, \langle f\rangle_0, \langle f\rangle_1$，$P_1$ 也有有 $\langle e\rangle_0, \langle e\rangle_1, \langle f\rangle_0, \langle f\rangle_1$。然后 P_0 和 P_1 运行 $\mathrm{Rec}(\langle e\rangle_0, \langle e\rangle_1)$ 和 $\mathrm{Rec}(\langle f\rangle_0, \langle f\rangle_1)$ 使得双方均获得 e 和 f。最后 P_i 计算 $\langle c\rangle_i=i\cdot e\cdot f+e\cdot\langle v\rangle_i+f\cdot\langle u\rangle_i+\langle z\rangle_i$ 即可得到结果的秘密份额。

图 3是秘密份额上的加法运算和乘法运算。要验证上述运算的正确性，只需验证 $\langle c\rangle_0+\langle c\rangle_1=a+b$ 是否成立（加法），$\langle c\rangle_0+\langle c\rangle_1=a\cdot b$ 是否成立（乘法）。

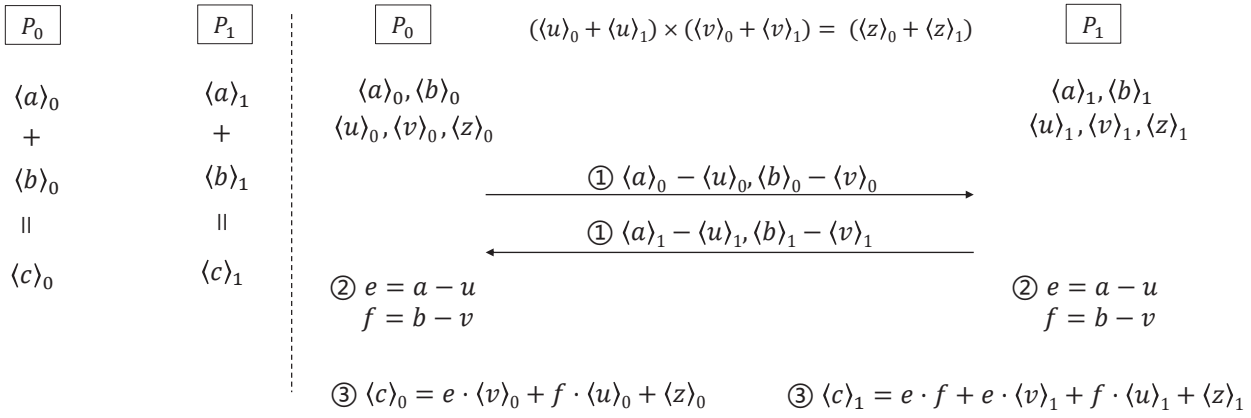


图 3: 秘密份额上的加法和乘法运算

3 恶意安全多方计算

下面我们开始详细介绍恶意安全多方计算的几种基本范式：3.1 节介绍了基于 GMW 编译器的恶意安全多方计算，3.2节介绍了基于 Cut-and-Choose 技术的恶意安全多方计算，3.3节介绍了基于消息认证码技术的恶意安全多方计算，3.4节介绍了基于可认证混淆电路技术的恶意安全多方计算技术，3.5节介绍了基于同态承诺技术的恶意安全多方计算，3.6节介绍了恶意安全多方计算的其他范式。

3.1 基于 GMW 编译器的恶意安全多方计算

在本节，我们将阐述如何将半诚实模型下的安全多方计算协议（original protocol，下称原始协议）转换为相应的恶意模型下的安全多方计算协议（compiled protocol，下称编译协议），这一过程我们称之为“编译”。特别地，在这里我们讨论两个参与方的情况，其可以很自然地拓展到多方的情况 [9]。

我们主要的思想是利用零知识证明迫使恶意参与方只能以与协议一致的方式行动。具体来说，编译协议按如下步骤进行：

1. 首先，在进行原始协议之前，各方利用一个承诺方案对其输入进行承诺。此外，各方也需要利用具有零知识性的知识的证明（Zero-Knowledge Proof-Of-Knowledge，ZKPOK）来证明他知道自己的输入，也就是说，他能够正确地对其承诺值进行解承诺。

2. 其次，所有参与方共同为每一个参与方生成一个随机比特序列，使得只有该参与方知道这个随机比特序列，其他参与方获得该随机比特序列的承诺。这些序列将被用作原始协议的随机输入（random input）。

3. 接下来，考虑诚实大多数的模型被编译，此时各方通过可验证的秘密共享协议（Verifiable Secret Sharing Protocol）分发自己的输入和随机输入。这是为了保证一旦有参与方中止协议执行，那么所有参与方能够一起重构他的秘密，并可以继续执行该协议。

4. 最后，在每一步，各方都使用零知识来扩充由原协议确定的将要发送的消息，该零知识断言消息确实被正确计算。注意到，由原协议确定的将要发送的消息是发送方自己输入、随机输入和前序消息[2]的函数，而且发送方自己的输入由第 1 步的承诺确定，随机输入由第 2 步的承诺确定，前序消息则是公开的，因为我们假设的信道是广播信道。因此，下一条消息可以认为是公开字符串（承诺及公开的前序消息）的函数。此外，将要发送的消息能被正确计算的断言是一个 NP 断言（NP-assertion），发送方知道相应的的 NP 证据（NP-witness），所以发送方能够向其他参与方证明他将要发送的消息的确是根据原始协议计算的。

形式化地，考虑将一个半诚实模型下安全两方计算协议转换为恶意模型下的安全两方计算协议，具体转换流程详见协议 7 。假设有半诚实模型下的安全两方计算 $\pi = (\pi_0, \pi_1)$，其中 $\pi_b(x, r, T)$ 表示参与方 P_b 用输入 x，随机带 r 和前序消息集合 T 生成的下一条消息，其中 $b = 0, 1$ 分别代表两个参与方 P_0, P_1。在原协议中，P_0 和 P_1 分别有自己的输入 x_0 和 x_1，编译后的协议记作 π^*。

Protocol 7 GMW-Compiler

1: 对于参与方 P_0 和 P_1 有：P_b 选择随机值 r_b 并生成 (x_b, r_b) 的承诺 c_b，其中打开承诺 c_b 所需要用到的披露值为 δ_b。

2: 对于参与方 P_0 和 P_1 有：P_b 选择并发送一个随机的 r'_{1-b}（即协议执行过程中，对方随机带上的一个秘密份额）。

3: 参与方 P_0 和 P_1 依次执行下列协议，直至协议终止：

(a) 令 T 表示 π 生成的前序消息集合（初始时为空）。P_b 计算并发送 π 的下一条消息 $t = \pi_b(x_b, r_b \oplus r'_b, T)$。如果 π_b 要求终止协议，P_b 也终止协议（无论 π_b 的执行结果是否包含相应的输出）。

(b) P_b 作为零知识证明的证明方，以私有输入 x_b, r_b, δ_b 和公开电路 $\mathcal{C}[\pi_b, c_b, r'_b, T, t]$ 完成零知识证明。其中公开电路 $\mathcal{C}[\pi_b, c_b, r'_b, T, t]$ 定义为：$\mathcal{C}[\pi_b, c_b, r'_b, T, t](x, r, \delta)$：当且仅当 δ 是与 (x, r) 的承诺 c 有效披露值，且 $t = \pi(x, r \oplus r', T)$。如果零知识证明验证失败，那么另一参与方 P_{1-b} 则中止。

基于 GMW 编译器的恶意安全多方计算的一个缺点就是使用了零知识证明，因为零知识证明涉及大量昂贵的公钥操作，因此其通信量和计算量非常大。

3.2 基于 Cut-and-Choose 技术的恶意安全多方计算

Cut-and-Choose 技术 [10] 是将 Yao 的安全两方计算协议从半诚实安全提升为恶意安全的重要技术之一。该技术的主要思想是在协议中使得一个参与方对某条消息构造足够多的副本，而另一参与方随机选择并检查该消息所有副本中的一个子集，如果所选子集中的副本都通过验证，则另一参与方相信剩余副本是大概率正确的，并在后续协议执行过程中使用剩余副本完成电路求值的计算任务。

我们考虑两方的设置，这种设置很容易拓展到多方。我们从半诚实的混淆电路协议开始，其中有两个参与方 P_1（也被称为发送方、电路生成方）和参与方 P_2（也被称为接收方、电路评估方）。基于半诚实的混淆电路协议（下称原协议）构造恶意的混淆电路协议（下称），需要考虑以下问题：

[2]前序消息是指在此刻以前各参与方交互时发送的消息的集合。

- 恶意的 P_1 必须被迫使正确生成混淆电路，以使得该电路能够正确计算功能函数。在这里，一种被称为 Cut-and-Choose 的技术被使用，Cut-and-Choose 技术的主要思想如下：

  1. P_1 必须生成混淆电路的许多副本，并发送给 P_2；然后 P_2 让 P_1 随机地去 “打开” 其中的一半
  2. P_1 打开后，和 P_2 一起检查打开的电路是否正确，使得 P_2 相信剩下未打开的混淆电路是正确的
  3. 之后参与方在原协议中计算剩余未打开的混淆电路，然后输出多数输出值（其中的输出值有正确的，有错误的，但多数输出值（majority output-value）都是正确的）

- Cut-and-Choose 技术能够使得 P_1 正确生成混淆电路，但会引入新的问题：P_1 和 P_2 必须在每个电路中使用相同的输入，这被称之为一致性检查（consistency checks），否则 P_1 和 P_2 可以从最终的输出中学习到额外知识。

- 证明安全性时出现的另一个问题是模拟器必须能够欺骗 P_2 并为其提供不正确的电路（即使 P_2 运行 Cut-and-Choose 测试）。这很容易便可解决，例如通过 coin-tossing 协议选择要打开的电路。

- 最后的一个问题是，P_1 可能会为 P_2 在 OT 协议中的一些可能选择提供特定的输入，并根据 P_2 是否中止协议来学习 P_2 的输入。

针对以上问题，我们首先大致地讲述基于 Cut-and-Choose 技术的恶意安全两方计算协议的流程，然后针对一些重要的环节进行详细地讲述。协议 8 展示了我们大致的协议流程，假设参与方 P_1 和 P_2 分别有输入 x 和 y，然后期望计算功能函数 $f(x, y)$：

Protocol 8 (high-levle overview) Malicious GarbledCircuit

1: 参与方确定计算功能函数 f 的电路，然后他们用一个门电路来代替 P_2 的每一个输入线。该门电路是一个异或门，其输入线为 P_2 的 s 条新的输入线，其输出是这些输入线的异或。因此，P_2 输入线的数量增加了 s 倍。
2: P_1 对 s 个计算 f 的不同混淆电路进行承诺，其中 s 是一个统计安全参数。P_1 也对该电路输入线对应的混淆值进行承诺。这些承诺值以一种特殊的方式进行构造以实现一致性检查。
3: 对于 P_1 的每个输入比特，参与方 P_1 和 P_2 运行一个 1-out-of-2 OT 协议，然后 P_2 获得其输入对应输入线的混淆值。
4: P_1 将第 2 步的所有承诺发送给 P_2。
5: P_1 和 P_2 运行一个 coin-tossing 协议以选择一个随机串，该随机串定义了哪些混淆电路的承诺将会被打开。
6: P_1 打开混淆电路和承诺的输入值，P_2 验证这些打开的电路的正确性，并进行这些解承诺输入值的一致性检查。
7: P_1 将未打开电路中 P_1 的输入线对应的混淆值发送给 P_2，P_2 也对这些值进行一致性检查。
8: 若所有的一致性检查都通过了，那么 P_2 就评估剩下未打开电路，并输出多数值。

3.3 基于可认证秘密共享的恶意安全多方计算

回顾一下，在 2.3 节中介绍了基于秘密共享的安全多方计算协议，还利用了 Beaver 乘法三元组来提高乘法运算的效率。其中讲到，如果 Beaver 乘法三元组和秘密共享方案满足以下性质 [8]：

1. 秘密份额满足加同态性，即给定秘密 x, y 的秘密份额和公共值 c，那么参与方可以本地计算 $x+y, x+c$ 和 $c \cdot x$ 的秘密份额。
2. 秘密份额在（恶意）攻击者的攻击下也可以成功隐藏对应的秘密值。
3. 即使存在恶意攻击者，也可以正确打开秘密份额中隐藏的秘密值。

那么，我们认为该安全多方计算协议满足恶意安全要求。本节主要介绍两类基于可认证秘密共享的恶意安全多方计算：BDOZ 和 SPDZ。

3.3.1 BDOZ 可认证秘密共享

BDOZ（Bendlin-Damgård-Orlandi-Zakarias）[11] 是 Bendlin 等人在 2011 年提出的技术，其主要思想是在秘密共享方案中引入了信息论安全的消息认证码（information-theoretic MACs）以抵御恶意敌手的攻击。

令 $\mathbb{F}$ 是一个域，满足 $|\mathbb{F}| \geq 2^{\kappa}$，其中 κ 是安全参数。将 $K, \Delta \in \mathbb{F}$ 看成密钥，定义 $\mathsf{MAC}_{K,\Delta}(x) = K + \Delta \cdot x$。$\mathsf{MAC}_{K,\Delta}(x)$ 是一个信息论安全的一次 MAC（information-theoretic one-time MAC）[3]，而且当敌手获得一个 x 对应的 $\mathsf{MAC}_{K,\Delta}(x)$ 不能产生另一个有效的消息认证码 $\mathsf{MAC}_{K,\Delta}(x')$，其中 $x \neq x'$。事实上，如果一个敌手能够计算这样的消息认证码，那么它可以按如下步骤计算 Δ：

$$\begin{aligned}(x-x')^{-1}(\mathsf{MAC}_{K,\Delta}(x') - \mathsf{MAC}_{K,\Delta}(x)) &= (x-x')^{-1}(K + \Delta x - K + \Delta x') \\ &= (x-x')^{-1}(\Delta(x-x')) \\ &= \Delta\end{aligned}$$

但敌手只有 $\mathsf{MAC}_{K,\Delta}(x)$ 将不能获得 Δ，因此敌手通过计算伪造一个有效的 MAC 的概率的上界为 $1/|\mathbb{F}| = 1/2^{\kappa}$，即猜对随机从域 $\mathbb{F}$ 中选择的 Δ 的概率。事实上，即使当一个诚实方有很多 MAC 密钥（密钥中有相同的 Δ 值，但 K 值是独立随机的）时，该 MAC 也是安全的。因此，我们将 Δ 称为全局 MAC 密钥，而将 K 称为本地 MAC 密钥。

BDOZ 技术就是利用这些信息论安全的 MAC 来对各方的秘密份额进行认证，在这里，我们考虑两方的情况。各方 P_i 先生成全局 MAC 密钥 Δ_i，然后 P_0 持有 x_0, m_0 和 K_0，而 P_1 持有 x_1, m_1 和 K_1，其满足：

- $x_0 + x_1 = x$，也即 x_0, x_1 是 x 的秘密份额。
- P_0 持有秘密份额 x_0 在 P_1 的 MAC 密钥 “加密” 得到的 MAC：$m_0 = \mathsf{MAC}_{K_1,\Delta_1}(x_0) = K_1 + \Delta_1 x_0$；而 P_1 持有秘密份额 x_1 在 P_0 的 MAC 密钥“加密”得到的 MAC：$m_1 = \mathsf{MAC}_{K_0,\Delta_0}(x_1) = K_0 + \Delta_0 x_1$。

[3]一次 MAC 是指拥有 MAC 密钥的参与方只能使用此 MAC 密钥生成一个 MAC。

可以证明，上述秘密分享方案满足本节开头所提到的安全性要求：

1. 加同态性：这里非常关键的点在于，当方案中的同一个参与方的 MAC 使用的都是同一个 Δ 时，那么该 MAC 也满足所要求的的加同态性。我们以参与方 P_0, P_1 计算 $x + x'$ 为例：

   (a) 一开始 P_0 持有 x 的秘密份额 $x_0, K_0, m_0 = \mathsf{MAC}_{K_1,\Delta_1}(x_0)$ 和 x' 的秘密份额 $x'_0, K'_0, m'_0 = \mathsf{MAC}_{K'_1,\Delta_1}(x'_0)$，$P_1$ 持有 x 的秘密份额 $x_1, K_1, m_1 = \mathsf{MAC}_{K_0,\Delta_0}(x_1)$ 和 x' 的秘密份额 $x'_1, K'_1, m'_1 = \mathsf{MAC}_{K'_0,\Delta_0}(x'_1)$。

   (b) 然后 P_0 和 P_1 分别在本地计算 $x_0 + x'_0, K_0 + K'_0, m_0 + m'_0$，$x_1 + x'_1, K_1 + K'_1, m_1 + m'_1$ 即可。可以验证，其分别是 $x + x'$ 的秘密份额，而且其中比较重要的一点是 $m_0 + m'_0 = \mathsf{MAC}_{K_1,\Delta_1}(x_0) + \mathsf{MAC}_{K'_1,\Delta_1}(x'_0) = \mathsf{MAC}_{K_1+K'_1,\Delta_1}(x_0 + x'_0)$。

2. 隐私性：各个参与方都无法得到秘密 x 的任何信息，因为各个参与方手中只有秘密 x 的秘密份额 $\langle x \rangle$，且如果不知道另一个参与方的 MAC 密钥（注意到，一个参与方的 MAC 密钥不会披露给其他参与方），那么 m 也不会泄露关于 x 的任何信息。

3. 安全可打开性：为了重构一个秘密值，只需 P_0 将自己的秘密份额 (x_0, m_0) 发送给 P_1，P_1 将自己的秘密份额 (x_1, m_1) 发送给 P_0，然后 P_0 和 P_0 分别在本地进行计算即可重构 $x = x_0 + x_1$。随后，P_0 可以用自己的 MAC 密钥验证 $m_1 = \mathsf{MAC}_{K_0,\Delta_0}(x_1)$ 是否成立，如果等式不成立，那么 P_0 中止协议；P_1 也用类似的方法验证 m_0。需要注意的是，如果打开的秘密值与真实秘密值不相等，则一次 MAC 一定无法通过验证。

图 4 展示了上述过程。

秘密份额	参与方P_0的秘密份额	参与方P_1的秘密份额
$\langle x \rangle$	x_0 K_0 $\mathsf{MAC}_{K_1,\Delta_1}(x_0)$	x_1 K_1 $\mathsf{MAC}_{K_0,\Delta_0}(x_1)$
$\langle x' \rangle$	x'_0 K'_0 $\mathsf{MAC}_{K'_1,\Delta_1}(x'_0)$	x'_1 K'_1 $\mathsf{MAC}_{K'_0,\Delta_0}(x'_1)$
$\langle x + x' \rangle$	$x_0 + x'_0$ $K_0 + K'_0$ $\mathsf{MAC}_{K_1+K'_1,\Delta_1}(x_0 + x'_0)$	$x_1 + x'_1$ $K_1 + K'_1$ $\mathsf{MAC}_{K_0+K'_0,\Delta_0}(x_1 + x'_1)$

图 4: 可认证秘密共享 BDOZ

很容易将 BDOZ 方法推广到 n 个参与方的场景（但方案的性能开销较大）。所有参与方均持有各自的全局 MAC 密钥，同时每个参与方分别持有 x 的加性秘密份额，且每个参与方的秘密份额需经过所有其他参与方 MAC 密钥的认证。

经过上面的分析，我们可以知道 BDOZ 可认证秘密共享方案满足使用 Beaver 乘法三元组所需要的安全性和同态性，可以利用 Beaver 乘法三元组来执行乘法运算，下面我们简单介绍一下如何生成 BDOZ 秘密共享方案的乘法三元组：

可以看到，即使相关参数（即可认证秘密份额 $\langle x\rangle$ 对应的秘密 x）被限制在域 $\mathbb{F}$ 的子域中，BDOZ 秘密共享方案也满足相应的性质。此时，可认证秘密份额 $\langle x\rangle$ 在对应子域中满足同态性。利用这一性质，将域 $\{0,1\}$ 看作 $\mathbb{F}=GF(2^\kappa)$ 中的一个子域，从而逐比特地构造 BDOZ 可认证秘密份额。值得注意的是，域 $\mathbb{F}$ 的阶必须是指数级的，这样才能保证方案的安全性（可认证性）。

逐比特构造 BDOZ 可认证秘密份额的最新方法是使用 Tiny-OT[12]，此方法在传统 OT 扩展协议的基础上构造了一种协议变体，用这一协议变体生成 BDOZ 中的可认证比特 $\langle x\rangle$。随后，此方法使用一系列协议将可认证比特逐个相乘，最终生成所需要的 Beaver 乘法三元组可认证秘密份额。

3.3.2 SPDZ 可认证秘密共享

可以看到，在 BODZ 秘密共享方案中，各方在本地持有的 $\langle x\rangle$ 中都包括由其他参与方生成的 MAC。也就是说，协议的存储复杂度随参与方数量的增加而线性增长。为解决这个问题，Damgård 等人在 2012 年提出了一个新的方案 SPDZ(Smart-Pastro-Damgård-Zakarias)[13]，将各个参与方的可认证秘密份额的存储复杂度降低到了常数大小。

SPDZ 的核心思想是设置一个全局 MAC 密钥 Δ，但任何参与方都无法得到这个全局 MAC 密钥 Δ。不过，参与方分别持有 Δ_i，可以将其看作全局 MAC 密钥 $\Delta=\sum_{i=0}^{n-1}\Delta_i$ 的秘密份额。同样地，我们考虑两个参与方 P_0 和 P_1 的情况，其中 P_0 持有 Δ_0，而 P_1 持有 Δ_1，全局 MAC 密钥 $\Delta=\Delta_0+\Delta_1$。在 SPDZ 可认证秘密份额 $\langle x\rangle$ 中，P_0 持有 (x_0,t_0)，P_1 持有 (x_1,t_1)，其中 $x_0+x_1=x$ 且 $t_0+t_1=\Delta\cdot x$。因此，实际上 P_0 和 P_1 分别拥有了 x 和 $\Delta\cdot x$ 的加性秘密份额，其中我们可以将 $\Delta\cdot x$ 看作是 x 的一种零次性信息论安全的 MAC（0-time information-theoretic MAC）[4]。

同样地，我们需要审视一下该 SPDZ 秘密共享方案是否满足使用 Beaver 乘法三元组的性质。显然地，该方案满足隐私性，x 的真实值不会被泄露出去。下面我们讨论一下安全可打开性和加法同态性：

1. 安全可打开性：注意到，不能让双方简单地公布他们的可认证秘密份额，因为这样会泄露 Δ 的值。在 SPDZ 方案中，在整个过程都保持 Δ 的机密性十分重要。为了打开 $\langle x\rangle$ 而不披露 Δ，需要执行以下三步：

   (a) 参与方仅公布 x_0,x_1，可以用这两个秘密份额重构 x，但此时 x 并未认证。

   (b) 注意到，如果 x 的值是正确地，那么有 $(\Delta_0x-t_0)+(\Delta_1x-t_1)=(\Delta_0+\Delta_1)x-(t_0+t_1)=\Delta x-\Delta x=0$。$P_0$ 可以本地计算得到 (Δ_0x-t_0)，P_1 也可以本地计算得到 (Δ_1x-t_1)。随后，P_0 对 (Δ_0x-t_0) 进行承诺，P_1 对 (Δ_1x-t_1) 进行承诺。

   (c) 最后，参与方打开承诺值，如果承诺值的和不等于 0，那么参与方中止协议。请注意，如果其中一个参与方不对 (Δ_ix-t_i) 作出承诺，仅公布 (Δ_ix-t_i) 的值，则另一个参与方可以选择适

[4]在 SPDZ 方案中，任何参与方都无法得到全局 MAC 密钥 Δ。也就是说，任何参与方都未使用此 MAC 密钥真正生成一个 MAC，因此将称作是一个零次性信息论安全的 MAC

当的 $(\Delta_{1-i}x - t_{1-i})$，使得求和结果为 0。SPDZ 方案通过承诺协议迫使参与方必须提前计算得到 $(\Delta_i x - t_i)$。

上述过程的安全性可以这样理解：当 P_0 对某个值 c 承诺时，它希望 P_1 也对 $-c$ 进行承诺。我们可以很容易地在理想世界中完成打开步骤中承诺协议的模拟过程，这意味着他们并不会泄露 Δ 的任何信息。可以证明，如果恶意参与方可以成功地将 $\langle x \rangle$ 打开为另一个不同的 x'，则此参与方可以猜测出 Δ，而由于敌手不知道 Δ 的任何信息，因此该事件发生的概率是可忽略的。

2. 加法同态性：在 SPDZ 可认证秘密份额 $\langle x \rangle$ 中，参与方分别持有 x 和 $\Delta \cdot x$ 的加性秘密份额。由于这两个秘密份额满足加同态性，因此 SPDZ 可认证秘密份额自然也满足加同态性。图 5 展示了同态常量加法，可以看到这里也会利用到 Δ 秘密份额的加同态性。

秘密份额	参与方P_0的秘密份额	参与方P_1的秘密份额	参与方P_0和P_1的秘密份额的和
$\langle x \rangle$	x_0 t_0	x_1 t_1	$x = x_0 + x_1$ $\Delta x = t_0 + t_1$
$\langle x + c \rangle$	$x_0 + c$ $t_0 + \Delta_0 c$	x_1 $t_1 + \Delta_1 c$	$x + c = x_0 + (x_1 + c)$ $\Delta(x + c) = \big((t_0 + \Delta_0 c) + (t_1 + \Delta_1 c)\big)$

图 5: 可认证秘密共享 SPDZ

从上面的分析可知，SPDZ 秘密共享方案满足使用 Beaver 乘法三元组的安全计算性质。下面我们再来讨论一下如何按照 SPDZ 可认证秘密份额的格式生成 Beaver 乘法三元组。SPDZ 的原始论文 [13] 提出了一个利用部分同态加密（somewhat homomorphic encryption）生成乘法三元组的方案，后续的工作 [14] 建议使用高效 OT 扩展生成 SPDZ 可认证秘密份额。

3.4 基于可认证混淆电路的恶意安全多方计算

Wang 等人在 2017 年提出了一种应用于安全多方计算的可认证混淆电路技术（Authenticated Garbling）[15]，该技术结合了信息论安全的 MPC 协议（可认证秘密共享和乘法三元组）和计算安全的 MPC 协议（混淆电路和 BMR 电路生成）。为了简便起见，我们考虑两方的设置，但可以很容易地将大部分技术推广到多方的情况 [16]。

在介绍可认证混淆电路技术之前，我们回顾一下 3.3.1 节介绍的 BDOZ 可认证秘密共享，在两方设置下的 BDOZ 可认证秘密份额 $\langle x \rangle$ 有：

1. P_0 持有的可认证秘密份额为 $x_0, K_0, m_0 = \mathsf{MAC}_{K_1,\Delta_1}(x_0) = K_1 \oplus x_0\Delta_1$。
2. P_0 持有的可认证秘密份额为 $x_1, K_1, m_1 = \mathsf{MAC}_{K_0,\Delta_0}(x_1) = K_0 \oplus x_1\Delta_0$。

我们将 x, x_1, x_2 看作比特串，因此相应的运算都在域 $\mathbf{G}(2^\kappa)$ 中进行，我们用 $\oplus$ 表示域中加法运算。因此，可以观察到：

$$\underbrace{(K_0 \oplus x_0\Delta_0)}_{P_0\text{已知}} \oplus \underbrace{(K_0 \oplus x_1\Delta_0)}_{P_1\text{已知}} = (x_0 \oplus x_1)\Delta_0 = x\Delta_0$$

所以，BDOZ 可认证秘密份额 $\langle x \rangle$ 可以使得参与方持有 $x\Delta_0$ 的加性秘密份额，其中 Δ_0 是 P_0 的全局 MAC 密钥。

带着这个结论，我们考虑 2.2 节介绍的混淆电路协议，电路生成方 P_0 为电路中的每一条输入线/输出线 i 选择线标签 k_i^0, k_i^1。在这里，我们令 k_i^b 的上标 b 表示该线标签对应的公开 Point-and-Permute 的置换标识比特，而不再是线标签所对应的真实值。我们令 p_i 表示实际的置换标识比特，因此 $k_i^{p_i}$ 表示真实值为 0 对应的线标签，$k_i^{\bar{p_i}}$ 表示真实值为 1 对应的线标签。

以 AND 门为例，假设 AND 门的输入线为 a 和 b，输出线为 c。如果用新的符号表示方法来描述标准混淆电路的构造方式，可以得到下述的混淆表：

$$
\begin{aligned}
e_{0,0} &= H(k_a^0||k_b^0) \oplus k_c^{p_c \oplus p_a \cdot p_b} \\
e_{0,1} &= H(k_a^0||k_b^1) \oplus k_c^{p_c \oplus p_a \cdot \bar{p_b}} \\
e_{1,0} &= H(k_a^1||k_b^0) \oplus k_c^{p_c \oplus \bar{p_a} \cdot p_b} \\
e_{1,1} &= H(k_a^1||k_b^1) \oplus k_c^{p_c \oplus \bar{p_a} \cdot \bar{p_b}}
\end{aligned}
$$

我们来解释一下“新的混淆表”。首先，明白混淆表本质是用两个输入线标签加密输出线标签，针对第一行，标准混淆电路的混淆表（未应用 FreeXOR 技术，见 2.2.2 节）为 $e_{0,0} = H(k_a^0||k_b^0||i) \oplus w_c^0$，由于我们这里只考虑一个 AND 门，因此哈希函数中的 i 可以省略。当用新的符号表示方式来，$w_c^0 = (k_c^0, p_c^0)$ 可以直接用一个 $k_i^{p_i}$ 表示，针对于输出线，这里的 i 应为 c，而 $p_c \oplus p_a \cdot p_b$ 表示了结果 p_c 或 $\bar{p_c}$。

应用 2.2.3 节介绍的 FreeXOR 技术，则有 $k_i^1 = k_i^0 \oplus \Delta$，其中 Δ 是一个全局不变的参数。在这种情况下，我们可以将混淆表重新表示为：

$$
\begin{aligned}
e_{0,0} &= H(k_a^0||k_b^0) \oplus k_c^0 \oplus (p_c \oplus p_a \cdot p_b)\Delta \\
e_{0,1} &= H(k_a^0||k_b^1) \oplus k_c^0 \oplus (p_c \oplus p_a \cdot \bar{p_b})\Delta \\
e_{1,0} &= H(k_a^1||k_b^0) \oplus k_c^0 \oplus (p_c \oplus \bar{p_a} \cdot p_b)\Delta \\
e_{1,1} &= H(k_a^1||k_b^1) \oplus k_c^0 \oplus (p_c \oplus \bar{p_a} \cdot \bar{p_b})\Delta
\end{aligned}
$$

可认证混淆电路的主要思想是让两个参与方以分布式的形式构造混淆表，使得任意一个参与方都无法得到置换标识比特 p_i 的值。假设两个参与方只拥有形式为 $\langle p_a \rangle, \langle p_b \rangle, \langle p_a \cdot p_b \rangle, \langle p_c \rangle$ 的 BDOZ 可认证秘密份额，任意一个参与方都无法获得 p_i 的明文值。进一步地，假设 P_0 已经选择好混淆电路的线标签，且满足 $\Delta = \Delta_0$（即 Δ 等于 BDOZ 中 P_1 的全局 MAC 密钥）。这样一来，两个参与方分别持有 $p_a\Delta, p_b\Delta, (p_a \cdot p_b)\Delta, p_c\Delta$ 的加性秘密份额。然后，他们就可以利用加性秘密份额的加同态性在本地计算得到 $(p_c \oplus p_a \cdot p_b)\Delta, (p_c \oplus p_a \cdot \bar{p_b})\Delta, (p_c \oplus \bar{p_a} \cdot p_b)\Delta, (p_c \oplus \bar{p_a} \cdot \bar{p_b})\Delta$ 的秘密份额。

观察应用 FreeXOR 技术后的混淆表中的第一行，我们可以看到：

$$
e_{0,0} = \underbrace{H(k_a^0||k_b^0) \oplus k_c^0}_{P_0\text{已知}} \oplus \underbrace{(p_c \oplus p_a \cdot p_b)\Delta}_{\text{参与方拥有的加性秘密份额}}
$$

因此两个参与方只需要通过本地计算（只需要让 P_0 于本地在秘密份额中加上仅自己知道的一个已知量）就可以得到混淆表中 $e_{0,0}$ 及其他所有行的加性秘密份额。

总而言之，分布式生成混淆电路的工作原理是：为电路中的每条输入线/输出线随机生成满足 BDOZ 可认证秘密份额格式的随机置换标识比特 $\langle p_i \rangle$，为电路中的每个 AND 门生成 BDOZ 可认证秘密份额格式的 Beaver 乘法三元组 $\langle p_a \rangle, \langle p_b \rangle, \langle p_a \cdot p_b \rangle$。随后，两个参与方只需要通过本地计算就可以得到混淆电路的加性秘密份额，且混淆表以 p_i 作为置换标识比特。最后，P_0 将混淆电路的秘密份额发送给 P_1，P_1 打开秘密份额后对电路进行求值。

3.5 基于同态承诺技术的恶意安全多方计算

Frederiksen 等人在 2018 年提出了同态承诺技术（Homomorphic Commitments）来实现恶意安全多方计算 [17]。同态承诺技术的主要思想是让参与方先分别用同态承诺技术承诺他们各自的输入，然后利用已承诺的输入参与后续的安全多方计算，这样一来，一旦恶意参与方在协议执行过程中篡改了自己的输入，那么其他参与方将通过是否可以有效打开某些中间值或者最终输出的承诺，以发现恶意参与方的这种恶意行为。

为了简便起见，我们考虑两方的设置，而且我们仅考虑两方同态承诺的简化版本，只考虑标量值 $x \in \mathbb{F}$，而不是文献 [17] 中考虑的向量值 $x \in \mathbb{F}^m$。下面是功能函数 $\mathcal{F}_{HCOM}$：

功能函数 $\mathcal{F}_{HCOM}$

功能函数 $\mathcal{F}_{HCOM}$ 与两个参与方 P_0，P_1 以及攻击者 $\mathcal{A}$ 进行交互

Commit：一旦接收到来自 P_0 的消息 $(commit, id)$，发送消息 $(commit, id)$ 给 $\mathcal{A}$，其中 id 是标识符。如果 $\mathcal{A}$ 返回消息 $(no\text{-}corrupt)$，随机选择 $x_{id} \in \mathbb{F}$ 并存储 $raw[id] = x_{id}$。然后，发送 $(committed, (id, x_{id}))$ 给 P_0，并且发送 $(committed, id)$ 给 P_1 和 $\mathcal{A}$。如果 $\mathcal{A}$ 返回消息 $(corrupt, (id, \bar{x}_{id}))$，存储 $raw[id] = \bar{x}_{id}$。发送 $(committed, (id, \bar{x}_{id}))$ 给 P_0，并且发送 $(committed, id)$ 给 P_1 和 $\mathcal{A}$。

Input：一旦接收到来自 P_1 的消息 $(input, id, y)$，如果 $raw[id] \neq \perp$，存储 $raw[id] = \perp$ 和 $actual[id] = y$。发送 $(input, id)$ 给 P_1 和 $\mathcal{A}$。

Rand：一旦接收到来自 P_0 的消息 $(random, id)$，如果 $raw[id] = x_{id} \neq \perp$，存储 $raw[id] = \perp$ 和 $actual[id] = x_{id}$。发送 $(random, id)$ 给 P_1 和 $\mathcal{A}$。

Linear Combination：一旦接收到来自 P_1 的消息 $(linear, \{(id, \alpha_{id})\}_{id \in ID}), \beta, id')$，其中 $\alpha_{id}, \beta \in \mathbb{F}$，$ID$ 是所有 id 的集合，如果 $actual[id] = x_{id} \neq \perp$ 和 $raw[id'] = actual[id'] = \perp$，存储 $raw[id'] = \perp$ 和 $actual[id] = \sum_{id \in ID}(\alpha_{id} \cdot x_{id}) + \beta$。发送 $(linear, \{(id, \alpha_{id})\}_{id \in ID}), \beta, id')$ 给 P_1 和 $\mathcal{A}$。

Open：一旦接收到来自 P_1 的消息 $(open, id)$，如果 $actual[id] = x_{id} \neq \perp$ 发送 $(opened, x_{id})$ 给 P_1 和 $\mathcal{A}$。

对于命令 **Commit**，它使得 P_0 可以向 P_1 承诺某个随机值 $x \in \mathbb{F}$。通过调用该命令，P_0 将持有承诺值组合 $[x]_0 = \{x, \langle x \rangle_0\}$，$P_1$ 将持有承诺 $\langle x \rangle_0$，其中 $\langle x \rangle_0$ 是承诺值 x 的打开信息。为了使 P_1 可以按照它的需要承诺某个指定值 y，它可以先调用命令 **Commit** 生成某个随机值的承诺，然后调用命令 **Input** 使得 P_0 持有承诺值组合 $[y]_0 = \{y, \langle y \rangle_0\}$，$P_1$ 持有承诺 $\langle y \rangle_0$。如果 P_0 仅仅需要向 P_1 承诺某个随机值，

那么它可以直接调用命令 **Rand**。当 P_0 需要向 P_1 打开先前被承诺的值 x 时，它可以调用命令 **Open** 将打开信息 $\langle x\rangle_0$ 发送给 P_1。此时 P_1 根据它所拥有的信息就可以打开 P_0 先前向它承诺的值 x。对于命令 **Linear Combination**，它可以支持如下三个操作：

(a) 加法：当 P_0 调用命令 $(linear, \{(id_0, 0), (id_1, 1)\}, 0, id')$ 以得到新的被承诺值 $[x_{id_0} + x_{id_1}]_0$ 时，P_0 本地计算 $\{(x_{id_0} + x_{id_1)}), (\langle x_{id_0}\rangle_0 + \langle x_{id_1}\rangle_0)\}$，且 P_1 本地计算 $([x_{id_0}]_0 + [x_{id_1}]_0)$，其中 $x_{id_0} + x_{id_1} = x_{id'}$，$\langle x_{id_0}\rangle_0 + \langle x_{id_1}\rangle_0 = \langle x_{id_0} + x_{id_1}\rangle_0 = \langle x_{id'}\rangle_0$ 且 $[x_{id_0}]_0 + [x_{id_1}]_0 = [x_{id_0} + x_{id_1}]_0 = [x_{id'}]_0$。

(b) 常数加法：当 P_0 调用命令 $(linear, \{(id_0, 0)\}, \beta, id')$ 以得到新的被承诺值 $[x_{id_0} + \beta]_0$ 时，P_0 本地计算 $\{(x_{id_0} + \beta, (\langle x_{id_0}\rangle_0 + \beta)\}$，且 P_1 本地计算 $([x_{id_0}]_0 + \beta)$，其中 $x_{id_0} + \beta = x_{id'}$，$\langle x_{id_0}\rangle_0 + \beta = \langle x_{id_0} + \beta\rangle_0 = \langle x_{id'}\rangle_0$。

(c) 常数乘法：当 P_0 调用命令 $(linear, \{(id_0, \alpha)\}, 0, id')$ 以得到新的被承诺值 $[\alpha \cdot x_{id_0}]_0$ 时，P_0 本地计算 $\{(\alpha \cdot x_{id_0}, (\alpha \cdot \langle x_{id_0}\rangle_0)\}$，且 P_1 本地计算 $([\alpha \cdot x_{id_0}]_0)$，其中 $\alpha \cdot x_{id_0} = \alpha \cdot x_{id'}$，$\alpha \cdot \langle x_{id_0}\rangle_0 = \langle \alpha \cdot x_{id_0}\rangle_0 = \langle x_{id'}\rangle_0$ 且 $\alpha \cdot [x_{id_0}]_0 = [\alpha \cdot x_{id_0}]_0 = [x_{id'}]_0$。

值得注意的是，基于所选择用来构建安全多方计算协议的同态承诺技术，此类协议不再依赖于随机预言机假设，也不局限于指定的数域。

3.6 其他范式的恶意安全多方计算

在这里，除了上述几种方案，我们还简单介绍一些其他范式的恶意安全多方计算。

Mohassel 等人 [18] 在 2015 年提出了只支持一个恶意参与方存在的恶意模型下的安全三方计算协议。该工作保障混淆电路正确生成的主要思想：首先，将三个参与方中的两方设置为电路生成方，另一个参与方设置为电路求值方。因为协议只允许一个恶意参与方存在，所以两个电路生成方中的一方总是诚实的。其次，让两个电路生成方基于同一随机种子分别生成对应的混淆电路，并将其发送给电路求值方。这样，电路求值方就可以通过对比所接收到的两个电路是否一致，来发现恶意参与方生成错误混淆电路的恶意行为。之后，为了支持更多数量参与方，Chandran 等人 [19] 利用分布式混淆方法生成混淆电路。该工作通过引入新的密码学原语 attested OT，设计了一个具体的承诺方案。并在此基础上，构建了只允许两个恶意参与方存在的恶意模型下的安全五方计算协议，该协议还可扩展到 n 个参与方且最多支持 $t \approx \sqrt{n}$ 个恶意参与方存在。

另外，一种将半诚实安全协议转换为恶意安全协议的黑盒方法称为 “MPC in the head” [20, 21]，该方法的主要思想是让实际参与方想象出一个由 “虚拟” 参与方执行的交互协议。实际参与方需要执行实现虚拟参与方行为的 MPC 协议，而不是执行实现某特定功能函数的 MPC 协议。然而，用于模拟虚拟参与方行为的 MPC 协议只需要满足较弱的安全性要求，虚拟参与方之间运行的协议只需要满足半诚实安全性要求。对于零知识证明这一特殊的 MPC 协议，通过 “MPC in the head” 方法构造出来的零知识证明协议 [22, 23, 24, 25] 是目前为止性能最优的。

4 总结

GMW 编译器可以将半诚实安全协议转换为恶意安全协议。然而，该转换方案得到的协议实际执行效率一般都比较低，原因在于 GMW 编译器没有把半诚实安全协议看作黑盒（black-box）使用。参与方必须通过零知识证明来确保其正确执行了半诚实安全协议的“下一条消息”函数。如果完成了这一证明，一般都需要把“下一条消息”函数用电路形式描述。Jarecki 和 Shmatikov[26] 提出了混淆电路协议的恶意安全变体协议。该协议的基本思想和 GMW 编译器有些相似，电路生成方需要证明每一个混淆表生成结果的正确性，但每一个门的证明都要用到多次公钥密码学操作。

在密码学的文献中，Cut-and-Choose 技术最开始提出的时候，只有方案没有证明。Mohassel 和 Franklin[27]、Kiraz 和 Schoenmakers[28] 于 2006 年分别在论文中提出了不同的 Cut-and-Choose 协议，但没有给出该技术的安全性证明，Lindell 和 Pinkas[10] 在 2007 年第一次提出了包含完整安全性证明的 Cut-and-Choose 协议，我们在 3.2 节讲述的 Cut-and-Choose 协议就来自 Lindell 和 Pinkas 的论文，他们提出的 Cut-and-Choose 协议可以抵御选择性中止攻击。不过针对 Cut-and-Choose 技术的输入一致性问题，3.2 节则介绍了 Shelat 和 Shen[29] 在 2011 年提出的解决方案。

我们还提出了实现可认证秘密共享的 BDOZ 技术和 SPDZ 技术。学者们还提出很多可认证秘密共享的构造方法 [30, 31]，Keller 等人 [14, 32] 讨论了基于 SPDZ 技术生成可认证秘密共享的多种高效变体方案。

从现有恶意模型下支持 n 个参与方的安全多方计算通用协议研究工作来看，基于 GMW 编译器的范式由于依赖高开销的零知识证明技术以抵抗恶意参与方的恶意行为，因而效率很低。而基于 Cut-and-Choose 技术的范式，由于需要生成、传输和验证足够数量的混淆电路以保证计算的隐私性和正确性，因而此类协议的计算和通信开销较大，且应用的范围有局限性。而基于消息认证码技术的范式，凭借消息认证码技术的加法/异或同态性质，实现了的加法门/异或门的快速本地计算。但由于各个参与方对每个乘法门都需要一次交互以获得相应的计算信息，这使得其计算和通信开销与电路深度成线性相关。而基于同态承诺技术的范式，旨在构造摆脱随机预言机假设的安全多方计算协议，以实现更高级别的安全性保障。

参考文献

[1] Andrew C Yao. Protocols for secure computations. In *23rd annual symposium on foundations of computer science (sfcs 1982)*, pages 160–164. IEEE, 1982.

[2] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, pages 162–167. IEEE, 1986.

[3] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 503–513, 1990.

[4] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free xor gates and applications. In *International Colloquium on Automata, Languages, and Programming*, pages 486–498. Springer, 2008.

[5] Vladimir Kolesnikov. Gate evaluation secret sharing and secure one-round two-party computation. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 136–155. Springer, 2005.

[6] Vladimir Kolesnikov. *Secure two-party computation and communication*, volume 68. 2006.

[7] Donald Beaver. Efficient multiparty protocols using circuit randomization. In *Annual International Cryptology Conference*, pages 420–432. Springer, 1991.

[8] David Evans, Vladimir Kolesnikov, Mike Rosulek, et al. A pragmatic introduction to secure multi-party computation. *Foundations and Trends® in Privacy and Security*, 2(2-3):70–246, 2018.

[9] Oded Goldreich. *Foundations of Cryptography, Volume 2*. Cambridge university press Cambridge, 2004.

[10] Yehuda Lindell and Benny Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 52–78. Springer, 2007.

[11] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 169–188. Springer, 2011.

[12] Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In *Annual Cryptology Conference*, pages 681–700. Springer, 2012.

[13] Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Annual Cryptology Conference*, pages 643–662. Springer, 2012.

[14] Marcel Keller, Emmanuela Orsini, and Peter Scholl. Mascot: faster malicious arithmetic secure computation with oblivious transfer. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 830–842, 2016.

[15] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Authenticated garbling and efficient maliciously secure two-party computation. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, pages 21–37, 2017.

[16] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Global-scale secure multiparty computation. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 39–56, 2017.

[17] Tore K Frederiksen, Benny Pinkas, and Avishay Yanai. Committed mpc - maliciously secure multiparty computation from homomorphic commitments. In *IACR International Workshop on Public Key Cryptography*, pages 587–619. Springer, 2018.

[18] Payman Mohassel, Mike Rosulek, and Ye Zhang. Fast and secure three-party computation: The garbled circuit approach. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 591–602, 2015.

[19] Nishanth Chandran, Juan A Garay, Payman Mohassel, and Satyanarayana Vusirikala. Efficient, constant-round and actively secure mpc: beyond the three-party case. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 277–294, 2017.

[20] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 21–30, 2007.

[21] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer–efficiently. In *Annual international cryptology conference*, pages 572–591. Springer, 2008.

[22] Irene Giacomelli, Jesper Madsen, and Claudio Orlandi. {ZKBoo}: Faster {Zero-Knowledge} for boolean circuits. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 1069–1083, 2016.

[23] Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, and Greg Zaverucha. Post-quantum zero-knowledge and signatures from symmetric-key primitives. In *Proceedings of the 2017 acm sigsac conference on computer and communications security*, pages 1825–1842, 2017.

[24] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In *Proceedings of the 2017 acm sigsac conference on computer and communications security*, pages 2087–2104, 2017.

[25] Jonathan Katz, Vladimir Kolesnikov, and Xiao Wang. Improved non-interactive zero knowledge with applications to post-quantum signatures. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 525–537, 2018.

[26] Stanisław Jarecki and Vitaly Shmatikov. Efficient two-party secure computation on committed inputs. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 97–114. Springer, 2007.

[27] Payman Mohassel and Matthew Franklin. Efficiency tradeoffs for malicious two-party computation. In *International Workshop on Public Key Cryptography*, pages 458–473. Springer, 2006.

[28] Mehmet Kiraz and Berry Schoenmakers. A protocol issue for the malicious case of yao' s garbled circuit construction. In *27th Symposium on Information Theory in the Benelux*, volume 39, 2006.

[29] Chih-hao Shen et al. Two-output secure computation with malicious adversaries. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 386–405. Springer, 2011.

[30] Ivan Damgård and Sarah Zakarias. Constant-overhead secure computation of boolean circuits using preprocessing. In *Theory of Cryptography Conference*, pages 621–641. Springer, 2013.

[31] Ivan Damgård, Jesper Buus Nielsen, Michael Nielsen, and Samuel Ranellucci. The tinytable protocol for 2-party secure computation, or: Gate-scrambling revisited. In *Annual International Cryptology Conference*, pages 167–187. Springer, 2017.

[32] Marcel Keller, Valerio Pastro, and Dragos Rotaru. Overdrive: making spdz great again. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 158–189. Springer, 2018.