

# 安全多方计算

——可证明安全视角

张秉晟 殷泽原 任奎 编著



v1.2.0, 2025 年 7 月



# 前言

随着大数据时代的到来，数据的价值在各个领域被越来越多地发掘出来。然而，对大规模数据的使用给隐私保护和信息安全也带来了前所未有的挑战。安全多方计算 (secure Multi-Party Computation, MPC) 是一种在保护原始数据的同时进行计算的技术，已经成为当前面对数据安全挑战的重要解决方案之一。

在现代密码学中，可证明安全是其核心基石。可证明安全基于形式化的定义、精确的假设和严格的安全证明。过去，密码方案的设计往往基于直觉，也就是“看起来是安全的”。而现代密码方案的设计和分析更加系统化，它要求设计者为给出的构造提供严格的安全证明——这样的证明实际上是一种归约：如果某个敌手能攻破某个方案，那么就可以基于其构造出一个新的敌手，新的敌手可以攻破底层的假设。有了这个证明，只要底层假设是可靠的，那么我们就可以确信：设计的方案在某种安全定义下一定是安全的。这种方法比基于直觉的设计大大增加了可靠性。

然而，当前 MPC/隐私计算领域的工程师们，大多对协议设计更感兴趣，对安全性证明的研究往往不够深入。许多研究 MPC 的硕士生，对可证明安全也只有一个模糊的概念。在市面上也找不到一本足够好的中文书，能把 MPC 的可证明安全性讲解得足够清楚。

基于这样的背景，我们希望写一本安全多方计算教材，它将介绍 MPC 经典协议，深入浅出地讲解可证明安全的模型，并给出安全性证明。期望可以帮助本领域的研究生更快速地入门，也把可证明安全的知识分享给 MPC/隐私计算领域的工程师同行。

## 欢迎交流

安全多方计算的发展日新月异，涵盖内容众多。而笔者所知有限，可能有不妥之处。非常欢迎读者将阅读过程中遇到的问题反馈给我们！无论是指出错误还是改进建议，或者是希望再版时增加的内容，都可以与我们交流。期待能与读者一起完成安全多方计算的知识学习与迭代。

邮箱：bingsheng@zju.edu.cn, im.yinzeyuan@qq.com

## 致谢

感谢陈俊杰、冯宇扬、季珂宇、刘健、卢天培、卢益彪、彭湃、孙嘉蔚、田磊原、王海涛、王熙璟、吴泽成、张文、张洵、张钰、张跃耀、赵庆澳、周哲磊在本书写作过程中给予的帮助。

本书的写作过程参考了教材<sup>[1-2]</sup>以及大量文献，对它们的作者表示诚挚的感谢。



# 目录

前言	iii
<b>1 引言</b>	<b>1</b>
1.1 安全多方计算 . . . . .	1
1.1.1 问题描述 . . . . .	1
1.1.2 安全加法与投票 . . . . .	2
1.1.3 安全乘法与配对 . . . . .	3
1.1.4 如果参与方不遵守指令? . . . . .	5
1.1.5 通用解决方案 . . . . .	6
1.2 内容概览 . . . . .	6
<b>2 基础知识</b>	<b>7</b>
2.1 现代密码学与可证明安全 . . . . .	7
2.1.1 形式化的定义 . . . . .	7
2.1.2 精确的假设 . . . . .	9
2.1.3 严格的安全性证明 . . . . .	10
2.2 基本术语和符号 . . . . .	11
2.3 基础原语 . . . . .	12
2.3.1 门限秘密分享 . . . . .	12
2.3.2 哈希函数与随机谕示机 . . . . .	13
2.3.3 伪随机数生成器 . . . . .	13
2.3.4 对称加密 . . . . .	14
2.3.5 (信息论一次性) 消息认证码 . . . . .	14
2.3.6 承诺 . . . . .	15
2.3.7 茫然传输 . . . . .	16
<b>3 一个安全多方计算协议的例子</b>	<b>17</b>
3.1 Shamir 秘密分享 . . . . .	17
3.1.1 拉格朗日插值法 . . . . .	17
3.1.2 计算示例 . . . . .	18
3.2 半诚实安全的电路计算协议 . . . . .	20

3.2.1 算术电路 . . . . .	20
3.2.2 协议描述 . . . . .	20
3.2.3 安全性分析 . . . . .	21
3.2.4 计算示例 . . . . .	24
<b>4 安全模型</b>	<b>27</b>
4.1 隐私性与恶意安全性 . . . . .	27
4.1.1 定义隐私性 . . . . .	27
4.1.2 定义恶意安全性 . . . . .	29
4.1.3 同时获得隐私性和恶意安全性 . . . . .	30
4.1.4 通用可组合框架概述 . . . . .	31
4.2 形式化模型——通用可组合框架 . . . . .	33
4.2.1 真实协议及其运行 . . . . .	33
4.2.1.1 真实协议的运行机制 . . . . .	35
4.2.1.2 敌手与诚实方的通信 . . . . .	35
4.2.1.3 平凡敌手 . . . . .	36
4.2.1.4 运行时间考虑 . . . . .	36
4.2.2 理想协议及其运行 . . . . .	36
4.2.2.1 理想协议的运行机制 . . . . .	37
4.2.2.2 运行时间考虑 . . . . .	37
4.2.3 安全函数计算的理想功能 . . . . .	38
4.2.3.1 概率功能 . . . . .	38
4.2.4 UC-安全实现 . . . . .	39
4.2.5 UC-安全实现的效果 . . . . .	41
4.2.5.1 并发组合的安全性 . . . . .	41
4.2.5.2 子协议组合的安全性 . . . . .	43
4.2.5.3 安全实现的传递性 . . . . .	45
4.2.6 定义半诚实安全性 . . . . .	46
4.2.6.1 定义受限框架 . . . . .	46
4.2.6.2 安全定义及其效果 . . . . .	47
4.2.7 不同类型的攻陷方式 . . . . .	47
4.2.7.1 适应性攻陷 . . . . .	47
4.2.7.2 可移动攻陷与主动安全 . . . . .	48
<b>5 茫然传输和茫然传输扩展</b>	<b>49</b>
5.1 关于 OT 的一些结论 . . . . .	49
5.1.1 不存在信息论安全的两方 OT 协议 . . . . .	49
5.1.2 OT 变种 . . . . .	51
5.1.3 一些其他结论 . . . . .	52
5.2 基于 DDH 假设的 OT 协议（半诚实安全） . . . . .	52

5.2.1 协议描述 . . . . .	52
5.2.2 安全性的直观说明 . . . . .	52
5.2.3 安全性证明 . . . . .	53
5.3 三方 OT 协议（恶意安全） . . . . .	55
5.3.1 协议描述 . . . . .	55
5.3.2 安全性的直观说明 . . . . .	56
5.3.3 安全性证明 . . . . .	57
5.4 IKNP OT 扩展协议 . . . . .	61
5.4.1 协议描述 . . . . .	62
5.4.2 安全性的直观说明 . . . . .	63
5.4.3 安全性证明 . . . . .	64
5.5 拓展阅读 . . . . .	65
<b>6 基于线性秘密分享的协议</b>	<b>67</b>
6.1 BGW 协议（半诚实安全） . . . . .	67
6.1.1 协议描述 . . . . .	67
6.1.2 安全性证明 . . . . .	67
6.2 GMW 协议 . . . . .	71
6.2.1 两方布尔电路 GMW 协议 . . . . .	71
6.2.1.1 协议描述 . . . . .	71
6.2.2 多方布尔电路 GMW 协议 . . . . .	72
6.2.2.1 协议描述 . . . . .	72
6.2.2.2 安全性证明 . . . . .	75
6.2.3 多方算术电路 GMW 协议 . . . . .	76
6.2.3.1 协议描述 . . . . .	76
6.2.3.2 安全性证明 . . . . .	77
<b>7 通过 Beaver 三元组实现 MPC</b>	<b>79</b>
7.1 基于 Beaver 三元组的 MPC 协议 . . . . .	79
7.1.1 Beaver 三元组 . . . . .	79
7.1.2 协议描述 . . . . .	80
7.1.3 安全性证明 . . . . .	80
7.2 Beaver 三元组的生成 . . . . .	82
7.2.1 基于可信第三方的方式 . . . . .	82
7.2.2 分布式生成方式 . . . . .	83
<b>8 基于混淆电路的协议</b>	<b>85</b>
8.1 姚氏混淆电路协议 . . . . .	85
8.1.1 直观思想 . . . . .	85
8.1.2 协议描述 . . . . .	88
8.1.3 安全性分析 . . . . .	92

8.1.4 独立模型 . . . . .	93
8.1.5 安全性证明 . . . . .	97
8.1.6 混淆电路的优化 . . . . .	102
8.2 BMR 协议 . . . . .	107
8.2.1 直观思想 . . . . .	108
8.2.2 协议描述 . . . . .	110
8.2.3 安全性分析 . . . . .	112
<b>9 恶意安全性</b>	<b>115</b>
9.1 GMW 编译器 . . . . .	115
9.2 切分选择 . . . . .	120
9.2.1 直观思想 . . . . .	120
9.2.2 LP11 协议 . . . . .	123
9.2.2.1 切分选择 OT . . . . .	123
9.2.2.2 输入一致性零知识证明 . . . . .	131
9.2.2.3 协议描述 . . . . .	131
9.2.3 安全性证明 . . . . .	132
9.3 BGW 协议（恶意安全） . . . . .	136
9.3.1 直观思想 . . . . .	136
9.3.2 准备工作——Shamir 秘密分享的性质与引理证明 . . . . .	142
9.3.3 BGW 可验证秘密分享 . . . . .	145
9.3.4 乘法协议 . . . . .	148
9.3.4.1 定义并行可验证秘密分享理想功能 $\mathcal{F}_{\text{VSS}}^n$ . . . . .	149
9.3.4.2 实现矩阵乘法理想功能 $\mathcal{F}_{\text{mat}}^A$ . . . . .	150
9.3.4.3 实现子份额秘密分享理想功能 $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$ . . . . .	154
9.3.4.4 实现多项式求值理想功能 $\mathcal{F}_{\text{eval}}$ . . . . .	158
9.3.4.5 实现乘积子份额秘密分享理想功能 $\mathcal{F}_{\text{VSS}}^{\text{mult}}$ . . . . .	163
9.3.4.6 实现乘法理想功能 $\mathcal{F}_{\text{mult}}$ . . . . .	170
9.3.5 $\{\mathcal{F}_{\text{VSS}}, \mathcal{F}_{\text{mult}}\}$ -混合模型下的安全计算协议 . . . . .	176
9.4 BDOZ 协议和 SPDZ 协议 . . . . .	179
9.4.1 BDOZ 协议 . . . . .	179
9.4.1.1 直观思想 . . . . .	179
9.4.1.2 协议描述 . . . . .	182
9.4.1.3 安全性证明 . . . . .	185
9.4.1.4 预处理阶段的实现 . . . . .	187
9.4.2 SPDZ 协议 . . . . .	192
9.4.2.1 直观思想 . . . . .	192
9.4.2.2 协议描述 . . . . .	196
9.4.2.3 安全性证明 . . . . .	201
9.4.2.4 * 预处理阶段的实现 . . . . .	203

目录

ix

后记

211



# Chapter 1

## 引言

### 1.1 安全多方计算

安全多方计算 (secure Multi-Party Computation, MPC) 研究的是这样一个计算任务：这个任务中有多个参与方，他们各自拥有一些隐私数据，并且想要计算一个关于这些隐私数据的函数。每个参与方都希望得到最终的结果，同时又希望自己的隐私数据不被其他参与方知道。

一个直接的方案是：寻找所有参与方都信任的第三方  $T$ ，将隐私数据发送给  $T$ ， $T$  执行所需的计算，然后公布结果，并诚实地删除计算过程中的所有隐私数据。然而，这个方案有两个明显的不足。首先，所有参与方都必须完全信任  $T$  能够确保正确性和隐私性。其次，即使  $T$  的确是诚实的，这个方案还是创造了一个危险的单点故障：一旦第三方  $T$  被攻破，所有的隐私数据都会被泄露。

那么，有没有可能在不依赖单一可信第三方的情况下完成这个任务呢？乍一看，这似乎不太可能，因为最终的结果取决于所有参与方提供的隐私数据。不过，正如本书后续内容所示，无论是在理论上还是实际应用中，都存在可行的解决方案。

#### 1.1.1 问题描述

让我们把这个任务描述得更精确一些。设参与方的数量为  $n$ ，将参与方记为  $P_1, \dots, P_n$ ，每个参与方  $P_i$  拥有隐私输入  $x_i$ 。所有参与方对要计算的函数  $f$  具有共识，即，他们都希望计算  $y = f(x_1, \dots, x_n)$ 。参与方希望计算过程满足以下两个性质：

- **正确性：**计算输出的结果  $y$  是正确的。
- **隐私性：**除了计算结果  $y$ ，没有其他信息被泄露。

满足正确性和隐私性的计算被称为安全多方计算。这里，我们先假设所有参与方都希望获得同样的公共输出；另一种更通用的情况是每个参与方可以获得不同的私有输出。为了简单起见，我们先考虑“单一公共输出”的情况。

第一价格密封拍卖 (first-price sealed-bid auction) 就是一个典型的实际案例。在这种拍卖方式中，所有竞拍者提交密封的出价，出价最高者以他提交的价格获得拍卖品。在这个场景下，隐私输入  $x_i$  就是竞拍者  $P_i$  提交的出价（假设  $x_i$  各不相同），定义函数  $f(x_1, \dots, x_n) = (z, j)$ ，其中  $x_j = z$  且

$x_j > x_i, 1 \leq i \leq n, i \neq j$ . 也就是说, 函数  $f$  输出最高价以及出价人的身份, 但是不泄漏其他竞拍者的出价。

接下来, 我们将用一个简单的协议作为示例, 展示如何在不依赖单一可信第三方的情况下安全地计算任何函数。我们先考虑一种比较简单的情况: 所有参与方都遵循协议。之后, 我们会讨论参与方违背协议时的处理方案。我们假设所有参与方之间可以安全地互相通信, 即,  $P_i$  可以向  $P_j$  发送消息  $m$ , 使得没有任何其他参与方可以看到  $m$ , 且  $P_j$  知道  $m$  来自于  $P_i$ .<sup>1</sup>

### 1.1.2 安全加法与投票

设有  $n$  个参与方  $P_1, \dots, P_n$  想要对某个决议进行投票表决。为此, 他们定义计票函数  $f(x_1, \dots, x_n) = \sum_{i=1}^n x_i$ , 其中  $x_i \in \{0, 1\}$ ,  $x_i = 0$  表示“反对”,  $x_i = 1$  表示“赞成”。如果可以安全地计算函数  $f$ , 那么他们就可以得知赞成票的数量, 且没有其他信息被泄露 (特别是不能泄漏特定参与方的具体投票情况)。下面, 我们将设计一个协议来实现安全投票。为了与下一小节保持一致, 我们考虑  $n = 3$  的情况。读者可以自行思考如何对任意的  $n$  设计安全投票协议。

**秘密分享 (secret sharing)**。实现这一目标的核心工具是秘密分享。这个词看上去似乎是有点自相矛盾: 如果一个东西是秘密, 那么可以与别人分享呢? 秘密分享的关键在于, 它允许秘密的持有者 (例如  $P_1$ ) 向所有参与方“分享”某个秘密  $x$ , 使得当所有参与方聚集在一起时可以恢复出  $x$ , 但是除  $P_1$  以外的任何单一参与方都不能独自获得关于  $x$  的任何信息。

我们使用的秘密分享方案如下。首先, 选择一个素数  $p$ , 定义  $\mathbb{Z}_p = \{0, 1, \dots, p - 1\}$ 。秘密  $x$  是  $\mathbb{Z}_p$  中的一个数。为了分享秘密  $x$ , 秘密持有者 (假设为  $P_1$ ) 在  $\mathbb{Z}_p$  中均匀随机地选择  $r_1, r_2$ , 并计算  $r_3 = x - r_1 - r_2 \bmod p$ . 换句话说,  $P_1$  在  $\mathbb{Z}_p$  中均匀随机地选择  $r_1, r_2, r_3$ , 满足  $r_1 + r_2 + r_3 = x \bmod p$ . 然后,  $P_1$  将  $r_1, r_3$  秘密地发送给  $P_2$ , 将  $r_1, r_2$  秘密地发送给  $P_3$ ,  $P_1$  自己保留  $r_2, r_3$ . 这里的  $r_j, j = 1, 2, 3$  被称为秘密  $x$  的份额 (share)。

让我们看看这个方案是否满足秘密分享的基本属性。首先, 当  $P_1, P_2, P_3$  聚集在一起时, 他们可以通过计算  $x = r_1 + r_2 + r_3 \bmod p$  来恢复出  $x$  (正确性)。另外, 对于  $P_1$  以外的任何单一参与方, 例如  $P_2$ , 他知道  $r_1$  和  $r_3$ , 但是不知道  $r_2$ , 因此他无法计算出  $x$  (隐私性)。对于  $P_3$  而言, 他也一样无法计算出  $x$ 。

我们可以用一种更严谨的方式来论证隐私性。从  $P_2$  的视角来看, 如果任取  $x_0 \in \mathbb{Z}_p$ , 存在  $x = x_0$  的可能吗? 答案是肯定的。如果  $x = x_0$ , 那么  $r_2 = x_0 - r_1 - r_3 \bmod p$ . 由于  $r_2$  是从  $\mathbb{Z}_p$  中均匀随机选择的, 它所有的取值概率都是均等的。因此, 对于任意的  $x_0 \neq x_1$ ,  $x = x_0$  的概率与  $x = x_1$  的概率都是相等的。由此我们得出结论,  $P_2$  的秘密份额没有泄漏关于  $x$  的任何信息。从  $P_3$  的视角来看, 也完全同理。

注意, 这里的隐私性是信息论意义上的。即使  $P_2$  拥有无限的计算能力, 他也无法获得关于  $x$  的任何信息。在这个秘密分享方案中, 每个参与方持有秘密的两个份额, 我们称之为复制秘密分享 (replicated secret sharing)。现在, 我们基于它构造安全加法协议。

#### 安全加法协议

安全加法协议中,  $P_1, P_2$  和  $P_3$  使用上述方式对他们的输入  $x_1, x_2$  和  $x_3$  秘密分享。不难发现, 参与方在本地将秘密份额相加并公布结果, 即可计算  $x_1, x_2$  和  $x_3$  的加和。协议如图 1.1 所示。

---

<sup>1</sup>在实际应用中, 这通常可以通过公钥基础设施 (Public Key Infrastructure, PKI) 来实现。

### 安全加法协议

**输入：**参与方是  $P_1, P_2, P_3$ .  $P_i$  的输入是  $x_i \in \mathbb{Z}_p$ , 其中  $p$  是事先约定的素数。

**输出：**每个参与方得到的输出是  $y = \sum_{i=1}^n x_i \bmod p$ .

**协议：**

1. 每个参与方  $P_i$  计算他的输入  $x_i$  的份额, 即,  $P_i$  在  $\mathbb{Z}_p$  中均匀随机地选择  $r_{i,1}, r_{i,2}$ , 并设置  $r_{i,3} = x_i - r_{i,1} - r_{i,2} \bmod p$ .
2. 每个参与方  $P_i$  秘密地发送  $r_{i,2}, r_{i,3}$  给  $P_1$ ,  $r_{i,1}, r_{i,3}$  给  $P_2$ ,  $r_{i,1}, r_{i,2}$  给  $P_3$ . 这里涉及到发送给“自己”, 例如,  $P_1$  现在保存着  $(r_{1,2}, r_{1,3}), (r_{2,2}, r_{2,3})$  和  $(r_{3,2}, r_{3,3})$ .
3. 每个参与方  $P_i$  将三个对应的秘密份额相加, 即, 对于  $\ell \neq i$ ,  $P_i$  计算  $s_\ell = r_{1,\ell} + r_{2,\ell} + r_{3,\ell} \bmod p$  并公布  $s_\ell$ . 每个参与方计算并公布两个值。
4. 每个参与方计算结果  $y = s_1 + s_2 + s_3 \bmod p$ .

图 1.1: 安全加法协议

我们首先检查一下结果是否是正确的。我们有

$$y = \sum_{\ell=1}^3 s_\ell \bmod p = \sum_{\ell=1}^3 \sum_{i=1}^3 r_{i,\ell} \bmod p = \sum_{i=1}^3 \sum_{\ell=1}^3 r_{i,\ell} \bmod p = \sum_{i=1}^3 x_i \bmod p$$

然后, 我们需要论证隐私性, 也就是除了计算结果  $y$  之外, 没有其他信息被泄露。以  $P_1$  为例, 我们刚才已经论证过, 这里使用的秘密分享方案不会泄露  $x_2, x_3$  的任何信息。 $P_1$  自己可以计算出  $s_2$  和  $s_3$ , 因此  $P_1$  在协议中唯一看到的新信息就是  $s_1$ . 那么  $s_1$  会不会泄漏  $x_2, x_3$  的信息呢? 这是不会的。因为协议的输出  $y = s_1 + s_2 + s_3 \bmod p$ , 这是  $P_1$  本应该从协议中得到的信息。而如果  $P_1$  知道了  $y$ , 他就可以反推出  $s_1 = y - s_2 - s_3 \bmod p$ . 所以  $s_1$  是可以根据协议的输出  $y$  计算出来的, 因此  $s_1$  不可能泄露  $y$  以外的任何信息。对于  $P_2$  和  $P_3$  而言, 也完全同理。

这种证明方式被称为基于模拟的证明。我们希望论证: 给定参与方应该知道的信息, 可以高效地计算(模拟)出他在协议中看到的所有信息。如果可以实现这样的模拟, 那么说明: 除了参与方应该得到的输出, 协议没有泄漏任何额外信息。这种证明方式将在本书后面的章节中多次出现。

注意,  $P_1$  的确可以计算出一些信息: 他可以计算  $y - x_1 = x_2 + x_3$ , 即, 可以计算另外两人的投票总和。读者可能会感到困惑, 认为这是协议的漏洞。然而, 基于最终结果  $y$  和他自己的输入  $x_1$ ,  $P_1$  确实可以计算其他人的投票总和, 这是无论协议如何设计都无法阻止的。因此, 这不是协议的漏洞, 协议能达到的最好效果就是让参与方只知道自己的输入和最终结果。

#### 1.1.3 安全乘法与配对

要想安全地计算任何函数, 仅有安全加法是不够的。下面我们介绍如何基于这种秘密分享方案构造安全乘法协议。

### 安全乘法协议

假设有两个数  $a, b \in \mathbb{Z}_p$ , 它们已经使用上述的方案进行了秘密分享, 即,  $a = a_1 + a_2 + a_3 \bmod p$ ,  $b = b_1 + b_2 + b_3 \bmod p$ . 我们有

$$ab = a_1b_1 + a_1b_2 + a_1b_3 + a_2b_1 + a_2b_2 + a_2b_3 + a_3b_1 + a_3b_2 + a_3b_3 \quad (1.1)$$

不难发现, 基于上述秘密分享方案, 式 (1.1) 中的每一项都可以由某个参与方在本地计算出来。例如,  $P_1$  拥有  $a_2, b_2, a_3, b_3$ , 那么他可以计算  $a_2b_2, a_2b_3, a_3b_3, a_3b_2$ . 然后, 参与方使用安全加法协议将这些项相加, 就可以实现安全乘法了。协议如图 1.2 所示。

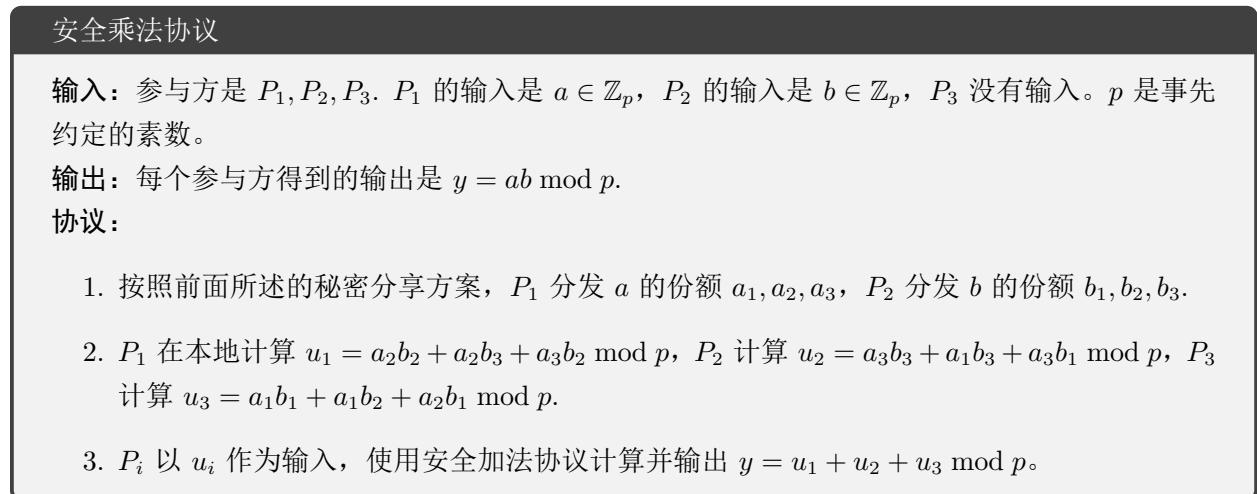


图 1.2: 安全乘法协议

让我们验证一下安全乘法协议的正确性和隐私性。首先, 由式 (1.1) 可知这样计算出的  $y$  是正确的。其次, 协议中除了本地计算之外, 只调用了安全加法协议。我们在前面论证过, 安全加法协议没有泄漏除了  $y$  以外的任何信息, 因此, 安全乘法协议也没有泄漏额外信息, 满足隐私性。

安全乘法的一个实际应用是实现用户的配对。例如, 在某个恋爱交友网站上, Alice 和 Bob 希望了解对方对自己是否感兴趣, 但又不想面对被对方拒绝的风险。我们可以让 Alice 选择  $a \in \{0, 1\}$ , 其中  $a = 1$  表示“感兴趣”,  $a = 0$  表示“不感兴趣”; Bob 也以同样的方式选择  $b$ . 然后, 他们可以让恋爱交友网站作为  $P_3$ , 三方执行安全乘法协议, 输出  $y = ab$ . 显然, 只有当 Alice 和 Bob 都选择“感兴趣”时, 输出结果才会为 1. 并且如果 Alice 选择了  $a = 0$ , 那么她将不会知道 Bob 的输入, 也就避免了 Bob 可能被拒绝的尴尬场面。

注意, 这里的恋爱交友网站  $P_3$  虽然协助了协议的完成, 但是他并不是单一可信第三方。具体而言,  $P_3$  只能知道 Alice 和 Bob 的配对是否成功, 而不能知道其他任何信息。特别是, 当配对不成功时,  $P_3$  无法得知是 Alice 对 Bob 不感兴趣, 还是 Bob 对 Alice 不感兴趣, 或是双方都选了不感兴趣。

一个自然的问题是: 如果没有  $P_3$  的协助, Alice 和 Bob 能否两个人自己实现安全的乘法和配对呢? 答案是肯定的。但是在这种场景下, 协议无法做到信息论意义上的安全, 必须假设某些问题在计算上是困难的。在本书后面的章节, 我们将介绍这样的协议。

### 1.1.4 如果参与方不遵守指令？

在上面所讲的协议中，我们假设参与方总是遵循协议规定执行。然而，现实情况往往并没有那么美好——参与方可能不遵守指令，目的是获得更多的信息，或者让协议输出错误的结果。我们将不遵守指令的行为分为两类：首先，参与方可以将自己的输入替换为另一个；其次，在协议执行过程中，参与方可以违背协议的规定。

#### 输入替换

以刚才的恋爱交友网站配对为例。如果 Alice 想了解 Bob 的输入，那么她可以始终选择  $a = 1$ （表示“感兴趣”）。此时，只有当 Bob 也选择  $b = 1$  时，输出才会为 1；如果输出为 0，说明 Bob 选择了  $b = 0$ 。这样，Alice 就可以破坏 Bob 的隐私性。

稍加思考就会发现，我们不可能通过协议的设计来解决这个问题。无论协议如何设计，恶意的参与方总是可以随心所欲地替换自己的输入，然后用这个新的输入执行协议。对于安全多方计算来说，这是它与生俱来的特性。

回到刚才的配对场景，如果 Bob 担心 Alice 会通过始终选择  $a = 1$  来破坏 Bob 的隐私性，那么他唯一的方法就是退出！如果他选择参与，那么他必须假设其他参与方会合理地选择自己的输入。这个假设成立的原因可能是：如果 Alice 实际上对 Bob 不感兴趣，那么选择  $a = 1$  将可能导致配对成功，这对 Alice 来说是时间的浪费！此类关于个体行为动机的研究属于“博弈论”的范畴，不在本书的讨论范围之内了。

#### 违背协议

前面讲过，无论协议如何设计，参与方总是可以替换自己的输入，这是我们无法阻止的行为。而另一种情况是，在协议执行过程中，参与方可能会违背协议的规定，为了使协议输出错误的结果或者得到更多的信息。例如，在一次拍卖中，参与方可能试图用低价成为拍卖的赢家。我们当然不希望这样的情况发生。但与前一种情况（输入替换）相比，这是可以通过协议设计来解决的。

让我们再研究一下安全加法协议。安全加法协议的第一步，是参与方将自己的输入秘密分享。以  $P_1$  为例，协议要求他均匀随机选取  $r_{1,1}, r_{1,2}, r_{1,3}$ ，使得  $x_1 = r_{1,1} + r_{1,2} + r_{1,3} \bmod p$ ，然后发送  $r_{1,1}, r_{1,3}$  给  $P_2$ ，发送  $r_{1,1}, r_{1,2}$  给  $P_3$ 。此时， $P_1$  可以违背协议的规定，将  $r_{1,1}, r_{1,3}$  发送给  $P_2$ ，并将  $r'_{1,1}, r_{1,2}$  发送给  $P_3$ ，其中  $r'_{1,1} \neq r_{1,1}$ 。此时， $P_1$  的输入  $x_1$  没有被良好定义，因而协议会输出什么结果、泄漏哪些信息，都是未知的。

幸运的是，有一个简单的方法可以检测到这种行为：当  $P_2$  和  $P_3$  从  $P_1$  那里接收到他们的秘密份额时， $P_2$  将它的  $r_{1,1}$  值发送给  $P_3$ ， $P_3$  将它自己的  $r_{1,1}$  值发送给  $P_2$ 。然后他们检查这个值是否相同。类似地， $P_1$  和  $P_3$  可以检查  $P_2$  发送的份额是否一致， $P_1$  和  $P_2$  可以检查  $P_3$  发送的份额是否一致。通过这一步检查，我们可以保证每个参与方的输入都被良好定义了。

在秘密分享阶段之后，各方在本地将秘密份额相加并公布。我们仍以  $P_1$  为例，协议要求他计算  $s_2$  和  $s_3$  并公布。 $P_1$  可能通过公布  $s'_2 \neq s_2$  来违背协议，这有可能导致协议输出错误的结果  $y = s_1 + s'_2 + s_3 \bmod p$ 。然而， $P_3$  也需要计算  $s_1$  和  $s_2$  并将其公布。因此， $P_1$  和  $P_3$  都应该公布  $s_2$  的值，参与方可以通过观察  $P_1$  和  $P_3$  是否公开了相同的  $s_2$ ，来检查是否存在这种违背协议的行为。同样地，参与方检查两个版本的  $s_1$  和两个版本的  $s_3$  是否相同来避免违背协议的行为。

在秘密分享阶段和输出阶段增加了一致性检查后，新的安全加法协议满足如下特性：当任何单一参与方违背协议时，另外两个参与方始终能够检测到这一点。这种特性被称为恶意安全性：任何除了“输入替换”以外的违背协议的行为都将被检测到。我们将在后面的章节中详细讨论它的形式化定义以及实现恶意安全性的各种方法。

### 1.1.5 通用解决方案

现在，我们已经设计了在  $\mathbb{Z}_p$  域中进行一次安全加法或乘法的协议。那么，如果我们希望对输入进行多次计算呢？不难发现，在安全加法协议的最后一步，如果参与方不公布  $s_\ell, \ell = 1, 2, 3$ ，那么  $P_1$  持有  $s_2, s_3$ ,  $P_2$  持有  $s_1, s_3$ ,  $P_3$  持有  $s_1, s_2$ . 这恰恰构成了计算结果  $y$  的秘密分享！对于安全乘法协议，因为它的最后一步调用了安全加法协议，所以各参与方也持有结果  $y$  的秘密份额。也就是说，如果希望对输入进行多次计算（例如，先做加法再做乘法），只需使用上一步的计算的秘密份额，调用安全加法/乘法协议中的步骤，即可安全地计算下一步。

众所周知，在  $\mathbb{Z}_p$  域中的加法和乘法可以模拟任何计算<sup>2</sup>。因此，我们已经得到了一个可以安全计算任何函数的协议！

让我们庆祝一下我们的初步进展吧！尽管这是在很强的假设下实现的：我们假设参与方会遵循协议的规定（安全乘法协议不能检测违背协议的行为）；我们只考虑了三个参与方的情况；我们假设不会有两个参与方串通；我们假设参与方可以安全地通信，等等。对于这些尚未解决的问题，我们将在后面的章节逐步探索他们的解决方案。

## 1.2 内容概览

本书接下来的内容组织结构如下。第2章介绍基础知识，包括可证明安全基础、符号和术语以及几个密码学基础原语。在第3章，我们会介绍一个通用安全多方计算协议，并尝试更严谨地分析其安全性。第4章，我们将在探讨隐私性与恶意安全性的过程中，引出形式化模型——通用可组合框架。接下来，对每个协议我们都会给出形式化的安全定义，并严格证明协议的安全性。我们首先在第5章介绍茫然传输协议（包括半诚实安全和恶意安全）。然后，我们从半诚实安全的协议开始——第6章介绍基于线性秘密分享的协议，第7章介绍基于 Beaver 三元组实现的协议，第8章介绍基于混淆电路的协议。最后，我们在第9章介绍恶意安全的协议。

请读者注意，这些章节并非相互独立，而是紧密联系的。例如，第4章的通用可组合框架是基于对第3章协议的讨论而引出的；而第5章之后的协议的安全性证明，又基于形式化的通用可组合框架；最后，第9章的恶意安全的协议，也与半诚实的协议有着对应关系，它们在半诚实协议的基础上，使用不同的技术实现了恶意安全性。总之，我们的章节设置假设读者拥有了前述章节的前置知识，因此，强烈建议读者按照章节顺序依次阅读每一章。

---

<sup>2</sup>任何计算都可以转化为布尔电路的形式，模素数加法和乘法可以模拟与门、或门、非门。

# Chapter 2

## 基础知识

本章将介绍阅读本书所需的基础知识。我们首先介绍现代密码学的核心——可证明安全。可证明安全的三个关键组成部分是定义、假设、证明。我们将通过一个公钥加密方案的例子带领读者了解可证明安全的理念。然后，我们介绍本书使用的符号和术语，并定义一些密码学基础原语。

### 2.1 现代密码学与可证明安全

从历史角度来看，密码学更像是一门艺术而非科学。早期的密码方案设计往往依赖设计者的直觉经验，其优劣评判标准主要取决于方案的复杂程度和构思巧妙性。安全性分析则采用“攻击-修补”的被动模式：通过穷举可能的攻击方式来检验方案，发现漏洞后再进行针对性修补。这种模式下，那些遭遇过重大攻击的方案自然会被认定为不安全，但反过来，要证明某个方案确实安全可靠，却始终缺乏系统性的判定标准。

Katz 和 Lindell 在 *Introduction to Modern Cryptography* 一书<sup>[3]</sup>中将现代密码学的主要不同总结为形式化的定义、精确的假设和严格的安全性证明。下面我们更详细地解释这三个原则。

#### 2.1.1 形式化的定义

一个常见的误区是：人们对于想实现的安全性有直观的认识，因此不需要花费时间做形式化的定义。下面，我们将以加密为例阐释定义的重要性。安全定义分为两个部分：安全保证和威胁模型。安全保证定义了方案应该满足什么样的安全属性，或者从敌手的角度来说，什么构成了成功的攻击；威胁模型定义了敌手拥有的能力，即敌手可以看到哪些信息、执行哪些操作。

对于一个加密方案来说，应该有怎样的安全保证呢？

你的第一反应可能是：敌手不能从密文中恢复出明文。这是一个直观的想法。然而，如果某个加密方案泄漏了 50% 的明文，而另外 50% 无法破译，那么按照这个定义它是安全的，但是这样的方案显然与实际的安全需求相悖。

我们把这个漏洞补上：敌手应该不能从密文中恢复出明文的任何一个字符。这比刚才的定义考虑得更全面。但是，以员工薪资的加密数据库为例，按照这个定义，敌手不能从密文中恢复出任何字符，但是他也许可以判断加密的薪资是否大于 20 万。那么这样的加密方案依然会泄露员工信息。

正确的定义是：敌手不能从密文中获得关于明文的任何信息。注意，该定义的关键在于它没有要求有关明文的信息是“有意义”的内容，而且仅要求不泄露任何信息。这一点很重要，它保证了安全的加密方案适用于所有需要保密性的潜在场景。

确定了安全保证之后，还需要明确威胁模型，也就是敌手拥有的能力。需要特别强调的是，威胁模型只规定了敌手的能力，而不对其采用的策略施加任何限制。这是因为：我们永远无法预判敌手可能采用的所有攻击手段。历史多次证明，预设敌手的攻击策略，注定会设计出失败的方案。

对于加密方案，有以下几种合理的威胁模型。我们按照敌手能力增强的顺序依次介绍：

- 仅密文攻击 (*ciphertext-only attack*)：这是最基本的攻击，敌手只能看到（多个）密文。
- 已知明文攻击 (*known-plaintext attack*)：敌手能够看到某个密钥生成的一个或多个明文/密文对。
- 选择明文攻击 (*chosen-plaintext attack*)：敌手能够看到某个密钥生成的明文/密文对，且敌手可以选择加密的明文。
- 选择密文攻击 (*chosen-ciphertext attack*)：敌手除了能选择加密的明文，还可以获得他选定的密文解密后的消息。敌手的目标仍然是获得使用相同密钥生成的其他密文所对应的明文信息，但是他不允许直接获得这些密文的解密。

尽管这些威胁模型中敌手的能力是增强的，但这并不代表哪一种威胁模型更好。我们应该根据实际场景来选择合适的威胁模型。

前两种模型比较直观：在网络通信中，我们通常假设敌手可以监听网络流量；对于已知明文攻击，一个简单的例子是，当两方开始通信时，他们的第一条消息可能总是“你好”。后两种模型乍一看似乎比较奇怪，敌手竟然可以选择加密的明文甚至解密指定的密文！不过，他们在现实世界中也有实际例子。在第二次世界大战期间，英国知道德国人发现水雷时会对这些位置进行加密并发送回总部；因此他们在特定地点布设水雷，然后这些加密消息被 Bletchley Park 的密码分析员用来破解德国的加密方案，这就是选择明文攻击。选择密文攻击的例子是，当敌手冒充客户端向服务器发送加密消息时，服务器会尝试解密，因而敌手可以从服务器的反应中推断出一些信息。例如，发送的密文是否被解密为格式正确的明文。

好，我们已经讨论了加密方案应该有的安全保证和威胁模型。为了精确地描述它们，密码学中我们通过游戏 (Game) 来准确地定义安全性。在一个游戏中，有两个角色——Challenger<sup>1</sup>和敌手 (Adversary)，他们按照游戏的规定进行一系列交互，最后，我们用敌手获胜的概率来定义安全性。

为了与下文中的例子相一致，这里给出公钥加密方案在选择明文攻击下<sup>2</sup>的不可区分游戏 (INDistinguishability under Chosen-Plaintext Attack, IND-CPA)。公钥加密与对称加密的不同之处在于：公钥加密方案的消息接收方首先生成一对公私钥 ( $\text{pk}, \text{sk}$ ) 并将  $\text{pk}$  公开，任何人都可以使用  $\text{pk}$  加密消息，但是只有拥有  $\text{sk}$  才可以将消息解密。更形式化地说，一个公钥加密方案  $\Pi$  包含 3 个算法 ( $\text{Gen}, \text{Enc}, \text{Dec}$ )，其中  $(\text{pk}, \text{sk}) \leftarrow \text{Gen}(1^\kappa)$  生成一对公私钥<sup>3</sup>； $c \leftarrow \text{Enc}_{\text{pk}}(m)$  输入公钥  $\text{pk}$  和消息  $m$ ，输出密文  $c$ ； $m/\perp \leftarrow \text{Dec}_{\text{sk}}(c)$  输入私钥  $\text{sk}$  和密文  $c$ ，输出消息  $m$  或特殊符号  $\perp$  表示解密失败。并且，算法满足  $\text{Dec}_{\text{sk}}(\text{Enc}_{\text{pk}}(m)) = m$ .

<sup>1</sup> Challenger 在这里的意思是参与这个游戏的人。若翻译成挑战者则不太贴切，因此不作翻译。

<sup>2</sup> 在实际应用中，选择明文攻击是较为常用的威胁模型。

<sup>3</sup> 这里的  $\kappa$  是安全参数，在第2.2节会详细介绍。读者可以暂时把  $\kappa$  理解成描述协议安全性的参数。 $1^\kappa$  是安全参数的一元表示，即  $\kappa$  个 1，这是为了确保  $\text{Gen}$  的运行时间是  $O(\kappa)$ 。

公钥加密方案在选择明文攻击下的不可区分游戏定义如下：

公钥加密方案 IND-CPA 游戏  $\text{PubK}_{\mathcal{A}, \Pi}^{\text{CPA}}(\kappa)$ :

1. Challenger 运行  $\text{Gen}(1^\kappa)$  得到  $(\text{pk}, \text{sk})$ , 并将  $\text{pk}$  发给敌手  $\mathcal{A}$ .
2. 敌手  $\mathcal{A}$  发送相同长度的  $m_0, m_1$  给 Challenger.
3. Challenger 均匀随机地选择一个比特  $b \leftarrow_{\$} \{0, 1\}$ , 计算  $c \leftarrow \text{Enc}_{\text{pk}}(m_b)$ , 然后把密文  $c$  发给  $\mathcal{A}$ .
4.  $\mathcal{A}$  输出比特  $b'$ . 如果  $b' = b$ , 游戏的输出为 1, 表示  $\mathcal{A}$  成功; 否则, 输出 0.

注意, 由于是公钥加密方案, 敌手  $\mathcal{A}$  拥有公钥  $\text{pk}$ , 因此他可以对任何消息自己进行加密, 而无需为其提供加密消息的接口  $\text{Enc}_{\text{pk}}(\cdot)$ .

现在, 我们可以给出公钥加密方案在选择明文攻击下具有不可区分性的形式化定义。

**定义 2.1.1.** 公钥加密方案  $\Pi = (\text{Gen}, \text{Enc}, \text{Dec})$  如果对任意概率多项式时间的敌手  $\mathcal{A}$ , 都存在一个可忽略函数  $\text{negl}$ <sup>4</sup>, 使得

$$\Pr[\text{PubK}_{\mathcal{A}, \Pi}^{\text{CPA}}(\kappa) = 1] \leq \frac{1}{2} + \text{negl}(\kappa)$$

那么, 我们说公钥加密方案  $\Pi$  在选择明文攻击下具有不可区分性 (*IND-CPA secure*)。

### 2.1.2 精确的假设

大多数现代密码方案都不是无条件安全的, 他们的安全性证明必须依赖于一些未经证明的假设<sup>5</sup>。对此, 读者可能会产生疑问: 既然假设是不可避免的, 那何必基于某假设来证明安全性? 为什么不直接假设方案本身是安全的? 这种思路看似直接, 实则存在问题。首先, 一个被研究多年未被攻破的假设要比新的、任意的假设更可靠; 其次, 我们喜欢关于一个“干净”的数学问题的难度的假设, 而不是一个复杂方案满足了复杂安全性定义的假设, 简单的假设更利于理解和研究。

密码学中有许多被广泛采用的假设, 例如大整数分解的困难性假设、离散对数假设、DDH(Decisional Diffie-Hellman) 假设等。这里, 我们给出 DDH 假设的精确定义, 作为一个例子。

考虑一个阶为  $q$  的循环群  $\mathbb{G}$ , 它的生成元为  $g$ . 给定  $g^x, g^y$ , 其中  $x, y \leftarrow_{\$} \mathbb{Z}_q$  是均匀随机选择的, DDH 假设说的是,  $g^{xy}$  与  $\mathbb{G}$  中的一个均匀随机元素在计算上是不可区分的。

**定义 2.1.2.** 如果对任意概率多项式时间的敌手  $\mathcal{A}$ , 都存在一个可忽略函数  $\text{negl}$ , 使得

$$|\Pr[\mathcal{A}(\mathbb{G}, q, g, g^x, g^y, g^z) = 1] - \Pr[\mathcal{A}(\mathbb{G}, q, g, g^x, g^y, g^{xy}) = 1]| \leq \text{negl}(\kappa)$$

其中, 群参数  $\mathbb{G}, q, g$  是由群的生成算法  $\mathcal{G}(1^\kappa)$  生成的,  $x, y, z \leftarrow_{\$} \mathbb{Z}_q$  是均匀随机选择的。那么, 我们说 DDH 问题在群  $\mathbb{G}$  中是困难的。

这里,  $\mathbb{G}, q, g$  是群的参数, 敌手  $\mathcal{A}$  得到的输入是  $(g^x, g^y, g^z)$  或者  $(g^x, g^y, g^{xy})$ . 在这两种情况下,  $\mathcal{A}$  输出 1 的概率几乎相等, 也就是说  $\mathcal{A}$  在接收到  $(g^x, g^y, g^z)$  或者  $(g^x, g^y, g^{xy})$  时不会有不同的表现, 这代表  $\mathcal{A}$  无法区分  $(g^x, g^y, g^z)$  和  $(g^x, g^y, g^{xy})$ . 像  $(g^x, g^y, g^{xy})$  这样形式的三元组被称为 DDH 三元组;  $(g^x, g^y, g^z)$  被称为随机三元组。

<sup>4</sup> 可忽略函数的定义在第2.2节会详细介绍, 直观地说, 可忽略函数趋近于 0 的速度比任何正多项式的倒数都更快。

<sup>5</sup> 绝大多数密码方案需要假设  $P \neq NP$ .

### 2.1.3 严格的安全性证明

基于既定的安全定义和数学假设，我们可以构建严格的安全性证明。这种证明本质上是一种“归约”，即，假设存在一个能够攻破该密码方案的敌手  $\mathcal{A}$ ，那么就可以基于  $\mathcal{A}$  构造出一个能够攻破底层假设的敌手  $\mathcal{B}$ 。由此可得：除非所依赖的底层假设本身不成立，否则该加密方案必然满足既定的安全定义。注意，这里我们不假定敌手  $\mathcal{A}$  的攻击策略，无论其采用什么策略，只要  $\mathcal{A}$  能攻破该密码方案，那么就可以构造敌手  $\mathcal{B}$  攻破底层假设。历史多次证明，假定敌手的攻击策略必然会导致有缺陷的密码方案。

下面，我们以一个公钥加密方案——ElGamal 为例，我们将严格地证明它在 DDH 假设下满足 IND-CPA 安全性。

**ElGamal 加密方案。**考虑一个阶为  $q$  的循环群  $\mathbb{G}$ ，它的生成元为  $g$ 。ElGamal 加密方案包含三个算法 (Gen, Enc, Dec)。生成算法 Gen 随机选择  $\text{sk} \leftarrow_{\$} \mathbb{Z}_q$ ，然后计算  $\text{pk} = g^{\text{sk}}$ ，输出  $(\text{pk}, \text{sk})$ 。加密算法 Enc 的输入是公钥  $\text{pk}$  和消息  $m \in \mathbb{G}$ ，输出密文  $c = (c_1, c_2) = (g^r, m \cdot \text{pk}^r)$ ，其中  $r \leftarrow_{\$} \mathbb{Z}_q$  是均匀随机选择的。解密算法 Dec 的输入是私钥  $\text{sk}$  和密文  $c = (c_1, c_2)$ ，输出  $\hat{m} = c_2 / c_1^{\text{sk}}$ （这里的除号 / 表示乘以某个群元素的逆元）。

我们首先检查一下正确性，即  $\text{Dec}_{\text{sk}}(\text{Enc}_{\text{pk}}(m)) = m$ 。设  $c = (c_1, c_2) = (g^r, m \cdot \text{pk}^r)$  是正确生成的密文。我们有

$$\hat{m} = \frac{c_2}{c_1^{\text{sk}}} = \frac{m \cdot \text{pk}^r}{(g^r)^{\text{sk}}} = \frac{m \cdot \text{pk}^r}{(g^{\text{sk}})^r} = \frac{m \cdot \text{pk}^r}{\text{pk}^r} = m$$

接着，我们来直观感受一下为什么 ElGamal 加密方案在 DDH 假设下是 IND-CPA 安全的。DDH 假设说的是，DDH 三元组  $(g^x, g^y, g^{xy})$  与随机三元组  $(g^x, g^y, g^z)$  在计算上是不可区分的。可以看到， $(\text{pk} = g^{\text{sk}}, c_1 = g^r, \text{pk}^r)$  恰恰构成了 DDH 三元组，因此  $\text{pk}^r$  与随机群元素是不可区分的，那么  $m \cdot \text{pk}^r$  也与随机群元素不可区分。反过来，我们从归约的角度来讲，如果存在敌手能够攻破 IND-CPA 游戏，也就是说他可以区分  $\text{Enc}_{\text{pk}}(m_0)$  和  $\text{Enc}_{\text{pk}}(m_1)$ 。我们假设 Challenger 加密的是  $m_0$ ，那么敌手  $\mathcal{A}$  得到的是  $(c_1, c_2) = (g^r, m_0 \cdot \text{pk}^r)$ 。如果敌手能从  $(c_1, c_2)$  中获得信息，那就说明在他看来  $m_0 \cdot \text{pk}^r$  和随机群元素并不是不可区分的（随机群元素不可能包含任何信息）。至此，我们要构造的归约就呼之欲出了。

**定理 2.1.** 假设 DDH 问题在群  $\mathbb{G}$  中是困难的，那么 ElGamal 加密方案是 IND-CPA 安全的。

证明. 将 ElGamal 加密方案记作 II。我们只需证明：假设存在敌手  $\mathcal{A}$  可以攻破 ElGamal 加密方案的 IND-CPA 安全性，即

$$\Pr[\text{PubK}_{\mathcal{A}, \text{II}}^{\text{CPA}}(\kappa) = 1] = \frac{1}{2} + \epsilon(\kappa)$$

其中， $\epsilon(\kappa)$  不是可忽略函数。那么，可以基于  $\mathcal{A}$  构造攻破 DDH 假设的敌手（算法） $\mathcal{B}$ 。

算法  $\mathcal{B}$  的构造如下：

算法  $\mathcal{B}$ :

算法的输入是  $(\mathbb{G}, q, g, h_1, h_2, h_3)$ 。

- 设置  $\text{pk} = h_1$ ，将  $\text{pk}$  发给  $\mathcal{A}$ ，然后得到  $\mathcal{A}$  发送的两条消息  $m_0, m_1 \in \mathbb{G}$ 。
- 选择均匀随机比特  $b'$ ，设置  $c_1 = h_2$ ,  $c_2 = m_{b'} \cdot h_3$ .
- 将密文  $(c_1, c_2)$  发给  $\mathcal{A}$  并获得  $\mathcal{A}$  的输出  $b$ 。若  $b' = b$ ，输出 1；否则，输出 0。

下面我们来分析  $\mathcal{B}$  输出 1 的概率，考虑  $(h_1, h_2, h_3)$  是随机三元组和 DDH 三元组两种情况。

**情况 1：** $(h_1, h_2, h_3)$  是随机三元组。即  $(h_1, h_2, h_3) = (g^x, g^y, g^z)$ . 此时， $(c_1, c_2) = (h_2, m_b \cdot h_3)$  是完全随机的，不可能包含  $m_b$  的信息。因此， $\mathcal{A}$  的输出  $b = b'$  的概率（即  $\mathcal{B}$  输出 1 的概率）等于  $\frac{1}{2}$ .

**情况 2：** $(h_1, h_2, h_3)$  是 DDH 三元组。即  $(h_1, h_2, h_3) = (g^x, g^y, g^{xy})$ . 此时， $\mathcal{A}$  收到的  $\text{pk} = h_1 = g^x$ ,  $(c_1, c_2) = (g^y, m_b \cdot g^{xy})$ .  $\mathcal{A}$  看到的所有消息与游戏  $\text{PubK}_{\mathcal{A}, \Pi}^{\text{CPA}}(\kappa)$  中是完全一样的。所以此时  $\mathcal{A}$  的输出  $b = b'$  的概率（即  $\mathcal{B}$  输出 1 的概率）等于  $\Pr[\text{PubK}_{\mathcal{A}, \Pi}^{\text{CPA}}(\kappa) = 1]$ .

综上所述，我们有

$$\begin{aligned} & |\Pr[\mathcal{B}(\mathbb{G}, q, g, g^x, g^y, g^z) = 1] - \Pr[\mathcal{B}(\mathbb{G}, q, g, g^x, g^y, g^{xy}) = 1]| \\ &= \left| \frac{1}{2} - \Pr[\text{PubK}_{\mathcal{A}, \Pi}^{\text{CPA}}(\kappa) = 1] \right| \\ &= \left| \frac{1}{2} - \left( \frac{1}{2} + \epsilon(\kappa) \right) \right| \\ &= \epsilon(\kappa) \end{aligned}$$

它不是可忽略函数。因此  $\mathcal{B}$  攻破了 DDH 假设。证毕。

□

希望读者能从这个例子中领会到可证明安全的理念。在本书后面的章节中，基于归约的证明技术将多次出现。

## 2.2 基本术语和符号

我们假设读者有高等数学、概率论与数理统计、离散数学、算法复杂度的基础知识，对于其中的基本概念，本书不再重复介绍。本书将安全多方计算 (secure Multi-Party Computation) 缩写为 MPC，将概率多项式时间 (Probabilistic Polynomial Time) 缩写为 PPT. 我们用  $[n]$  表示集合  $\{1, 2, \dots, n\}$ ，用  $\leftarrow_{\$}$  表示均匀随机选取。 $|X|$  表示集合  $X$  的大小。

**可忽略函数。** 可忽略函数  $\text{negl}(\cdot)$  是一个比任何正多项式的倒数趋近于 0 的速度更快的函数，即：对任意的正多项式  $\text{poly}(\cdot)$ ，存在整数  $N > 0$ ，使得对所有  $x > N$ ，都有  $\text{negl}(x) < \frac{1}{\text{poly}(x)}$ .

**安全参数。** 安全参数是密码学协议中影响参与方运行时间以及敌手成功概率的一个参数，它定义了协议的安全性等级，通常用字母  $\kappa$  或  $\lambda$  或  $n$  表示。在参与方运行一个协议之前，他们会选定安全参数的值（例如，可以将对称加密方案的密钥长度视作安全参数）。协议中，参与方的运行时间通常是关于安全参数的多项式；如果一个敌手的运行时间是关于安全参数的多项式，我们认为他是高效的敌手（这体现了安全参数影响参与方和敌手的运行时间）。如果敌手攻破协议的概率是关于安全参数的一个可忽略函数，那么这个协议是安全的（这体现了安全参数影响敌手的成功概率）。

**统计距离。** 设  $X_1$  和  $X_2$  是定义在同一概率空间上的两个随机变量，并且它们具有相同的取值范围  $D$ . 我们称

$$\delta(X_1, X_2) = \frac{1}{2} \sum_{d \in D} |\Pr[X_1 = d] - \Pr[X_2 = d]|$$

为  $X_1$  和  $X_2$  的统计距离<sup>6</sup>（又称总变差距离）。直观地说，两个随机变量的统计距离描述了它们在每个可能的取值上的概率之差的总和。

**不可区分性。**设  $X_1$  和  $X_2$  是关于安全参数  $\kappa$  的两个概率分布，或者等价地说， $X_1$  和  $X_2$  是以安全参数  $\kappa$  作为输入的两个算法。如果对于任意的敌手  $\mathcal{A}$ ，都存在可忽略函数  $\text{negl}(\cdot)$ ，使得

$$|\Pr[\mathcal{A}(X_1(\kappa)) = 1] - \Pr[\mathcal{A}(X_2(\kappa)) = 1]| \leq \text{negl}(\kappa) \quad (2.1)$$

那么我们称  $X_1$  和  $X_2$  是不可区分的，记作  $X_1 \approx X_2$ 。如果考虑的是 PPT 的敌手，那么此定义描述的是计算不可区分性，记作  $X_1 \xrightarrow{\text{comp}} X_2$ ；如果考虑的是拥有无限计算能力的敌手，那么此定义描述的是统计不可区分性，记作  $X_1 \xrightarrow{\text{stat}} X_2$ ，此时，式 (2.1) 的上界是  $X_1$  和  $X_2$  的统计距离；如果对所有的  $\kappa$ ，都有统计距离  $\delta(X_1(\kappa), X_2(\kappa)) = 0$ ，那么  $X_1$  和  $X_2$  是完美不可区分的，记作  $X_1 \xrightarrow{\text{perf}} X_2$ 。

计算安全性表示协议面对 PPT 敌手时的安全性；信息论安全性表示协议面对拥有无限计算能力的敌手时的安全性。

## 2.3 基础原语

在这一小节，我们将介绍这本书用到的一些密码学基础原语，这些基础原语将会在后续章节的安全多方计算协议中被经常使用。

### 2.3.1 门限秘密分享

秘密分享是一种非常重要的密码学基础原语，这里我们主要讨论  $(t, n)$ -门限秘密分享 (threshold secret sharing)，即一个秘密  $s$  被分成  $n$  个秘密份额之后，其中任意  $t$  个秘密份额不会泄露任何关于秘密  $s$  的信息，而拥有任意  $t + 1$  个秘密份额便能恢复出秘密  $s$ 。我们给出  $(t, n)$ -门限秘密分享的定义<sup>[4]</sup>。

**定义 2.3.1.** 设  $\mathcal{D}$  是秘密所在的域， $\mathcal{D}_1$  是秘密份额所在的域。 $(t, n)$ -门限秘密分享方案包含以下两个算法 (Share, Rec)：

- $(s_1, \dots, s_n) \leftarrow \text{Share}(s)$ : 这是分享算法，它以秘密  $s \in \mathcal{D}$  作为输入，最后输出  $n$  个秘密份额  $s_1, \dots, s_n \in \mathcal{D}_1$ .
- $s = \text{Rec}(s_{i_1}, \dots, s_{i_k})$ : 这是恢复算法，它以任意  $k \geq t + 1$  个秘密份额  $s_{i_1}, \dots, s_{i_k} \in \mathcal{D}_1$  作为输入，最终输出秘密  $s \in \mathcal{D}$ .

$(t, n)$ -门限秘密分享方案需要满足如下的性质：

- 正确性：令  $(s_1, \dots, s_n) \leftarrow \text{Share}(s)$ ，只要获得至少  $t + 1$  个秘密份额便能恢复出秘密  $s$ ，即

$$\Pr[\forall k \geq t + 1, \text{Rec}(s_{i_1}, \dots, s_{i_k}) = s] = 1 .$$

- (完美) 隐私性：任意不超过  $t$  个秘密份额都不会泄露任何有关于秘密  $s$  的信息。即，对任意集合  $U \in [n]$ ，如果  $|U| \leq t$ ，那么  $\{s_j\}_{j \in U}$  的分布与  $s$  无关。

<sup>6</sup>注意，如果范围  $\mathcal{D}$  是可数无穷的，那么只有在和收敛的情况下，这个定义才有意义。不过这个条件显然是成立的，因为级数  $\frac{1}{2} \sum_{d \in D} |\Pr[X_1 = d] - \Pr[X_2 = d]|$  是单调递增且上界为 1 的。

### 2.3.2 哈希函数与随机谕示机

哈希函数 (hash functions) 可以将较大的域映射到较小的范围。密码学的哈希函数  $H$  还有以下性质：

- 抗原像性 (*preimage-resistance*)：给定哈希的输出值  $y$ ，找到它的原像在计算上是不可行的。即，给定哈希输出值  $y$  且不知道对应的输入值，找到  $x'$  使得  $H(x') = y$  在计算上不可行。
- 抗第二原像性 (*2nd-preimage resistance*)：给定输入  $x$ ，找到另一输入使得它们的输出相同，在计算上是不可行的。即，给定  $x$ ，找到第二原像  $x' \neq x$  使得  $H(x) = H(x')$  在计算上不可行。
- 抗碰撞性 (*collision resistance*)：找到两个不同的输入使它们的哈希输出相同，在计算上是不可行的。即，找到  $x \neq x'$  使得  $H(x) = H(x')$  在计算上不可行。

随机谕示机 (Random Oracle, RO)<sup>[5]</sup>是一个理想化模型，它将哈希函数的输出理想化为完全随机。在随机谕示机模型下，所有参与方都被允许使用一个公开函数  $H : \{0, 1\}^* \rightarrow \{0, 1\}^\kappa$ ，其中  $\kappa$  为安全参数。当参与方调用函数  $H$  并以  $x \in \{0, 1\}^*$  作为输入时，函数  $H$  会查询它的所有历史调用记录，如果  $H(x)$  从来没有被调用过，那么  $H$  会挑选一个随机数  $r_x \leftarrow_{\$} \{0, 1\}^\kappa$ ，记录  $\langle x, r_x \rangle$  并返回  $r_x$  作为函数输出；如果  $H(x)$  曾经被调用过，那么  $H$  便会返回曾经记录下的  $r_x$  作为输出。

随机谕示机模型是一个输出纯随机的理想函数。在实际应用中，人们往往使用哈希函数来实例化随机谕示机模型，然而，现实中的哈希函数（例如 SHA256）的输出并不是纯随机的。因此，有可能构造出的密码协议在随机谕示机模型下可证明安全，但是当随机谕示机模型被实例化为哈希函数时又不安全<sup>[6]</sup>。尽管随机谕示机模型在实例化时可能存在缺陷，人们仍然广泛接受它并将它作为一个重要的启发式模型。

### 2.3.3 伪随机数生成器

伪随机数生成器 (PseudoRandom Generator, PRG) 是一个高效的确定性算法。它的输入是一个较短的均匀随机串（称为种子 seed），输出一个更长的“看起来随机的”（伪随机的）串。换句话说，伪随机数生成器通过少量的真随机数生成了大量的伪随机数，而生成的伪随机数和真随机数在计算上是不可区分的。

伪随机数生成器的形式化定义如下：

**定义 2.3.2.** 令  $\mathcal{G}$  是一个确定性多项式时间算法，对于任意的  $\kappa$  和任意的输入  $s \in \{0, 1\}^\kappa$ ，它的输出  $\mathcal{G}(s)$  是长度为  $\ell(\kappa)$  的串。如果以下两个条件成立：

1. (扩展性。) 对于每个  $\kappa$  都有  $\ell(\kappa) > \kappa$ .
2. (伪随机性。) 对于任意 PPT 的算法  $\mathcal{A}$ ，存在可忽略函数  $\text{negl}$  满足

$$|\Pr[\mathcal{A}(\mathcal{G}(s)) = 1] - \Pr[\mathcal{A}(r)] = 1| \leq \text{negl}(\kappa)$$

其中  $s \leftarrow_{\$} \{0, 1\}^\kappa, r \leftarrow_{\$} \{0, 1\}^{\ell(\kappa)}$ .

那么，我们称  $\mathcal{G}$  是一个伪随机数生成器。

### 2.3.4 对称加密

在对称加密方案 (symmetric encryption schemes) 中，加密密钥与解密密钥是相同的。常用的对称加密算法有 DES、AES 等。在上文第2.1.1节，我们已经初步讨论其安全性定义，并给出了公钥加密方案的 IND-CPA 安全性定义。现在，我们给出对称加密方案的形式化定义：

**定义 2.3.3.** 我们假设  $\mathcal{K}$  是密钥所在的空间， $\mathcal{M}$  是明文所在的空间， $\mathcal{E}$  是密文所在的空间。对称加密方案包含以下三个算法 ( $\text{Gen}, \text{Enc}, \text{Dec}$ )：

- $k \leftarrow \text{Gen}(1^\kappa)$ : 这是密钥生成算法，它以安全参数  $\kappa$  作为输入，输出密钥  $k \in \mathcal{K}$ .
- $c \leftarrow \text{Enc}_k(m)$ : 这是加密算法，它以密钥  $k \in \mathcal{K}$  和明文  $m \in \mathcal{M}$  作为输入，输出密文  $c \in \mathcal{E}$ .
- $m \leftarrow \text{Dec}_k(c)$ : 这是解密算法，它以密钥  $k \in \mathcal{K}$  和密文  $c \in \mathcal{E}$  作为输入，输出明文  $m \in \mathcal{M} \cup \{\perp\}$ ，其中  $\perp$  表示解密错误。

对于任意的密钥  $k \leftarrow \text{Gen}(1^\kappa)$  和明文  $m \in \mathcal{M}$ ，必须满足  $\text{Dec}_k(\text{Enc}_k(m)) = m$ .

**定义 2.3.4.** 对称加密的选择明文攻击安全性 (*IND-CPA secure*): 一个对称加密方案  $\Pi = (\text{Gen}, \text{Enc}, \text{Dec})$  如果对于任意的 PPT 敌手  $\mathcal{A}$ ，有

$$\Pr \left[ \begin{array}{l} k \leftarrow \text{Gen}(1^\kappa); (m_0, m_1) \leftarrow \mathcal{A}^{\text{Enc}_k(\cdot)}(1^\kappa); \\ b \leftarrow \{0, 1\}; c \leftarrow \text{Enc}_k(m_b); b' \leftarrow \mathcal{A}^{\text{Enc}_k(\cdot)}(1^\kappa, c, m_0, m_1) \end{array} : b = b' \right] \leq \frac{1}{2} + \text{negl}(\kappa),$$

其中  $\text{Enc}_k(\cdot)$  是一个接口，当输入为  $m$  时，它会返回  $c \leftarrow \text{Enc}_k(m)$  作为输出， $\mathcal{A}^{\text{Enc}_k(\cdot)}$  表示敌手  $\mathcal{A}$  可以任意地调用这个接口。

那么，我们称加密方案  $\Pi$  具有选择明文攻击安全性 (*IND-CPA secure*)。

### 2.3.5 (信息论一次性) 消息认证码

加密方案为消息传输提供了隐私性 (secrecy)，但不保证完整性 (integrity)。换句话说，加密方案虽然使敌手无法获得关于明文的任何信息，但不能阻止敌手在信道中注入消息或修改被传输的消息。消息认证码 (Message Authentication Codes, MACs) 提供了这层保障。在本书中，我们主要关注信息论一次性消息认证码 (information-theoretic one-time MACs)。直观来说，消息认证码方案使用密钥为消息生成一个标签，而不知道密钥的敌手无法为消息生成合法的标签。消息认证码的定义与信息论一次性消息认证码的安全性定义如下。

**定义 2.3.5.** 我们假设  $\mathcal{K}$  是密钥所在的空间， $\mathcal{M}$  是消息所在的空间， $\mathcal{T}$  是标签所在的空间。消息认证码方案包含以下三个算法 ( $\text{Gen}, \text{Mac}, \text{Ver}$ )：

- $k \leftarrow \text{Gen}(1^\kappa)$ : 这是密钥生成算法，它以安全参数  $\kappa$  作为输入，输出密钥  $k \in \mathcal{K}$ .
- $t \leftarrow \text{Mac}_k(m)$ : 这是标签生成算法，它以密钥  $k \in \mathcal{K}$  和消息  $m \in \mathcal{M}$  作为输入，输出标签  $t \in \mathcal{T}$ .
- $0/1 \leftarrow \text{Ver}_k(m, t)$ : 这是验证算法，它以密钥  $k \in \mathcal{K}$ 、消息  $m \in \mathcal{M}$  和标签  $t \in \mathcal{T}$  作为输入，输出  $0/1$  表示不合法或合法。

对于任意的密钥  $k \leftarrow \text{Gen}(1^\kappa)$  和消息  $m \in \mathcal{M}$ , 必须满足  $\text{Ver}_k(m, \text{Mac}_k(m)) = 1$ .

**定义 2.3.6.** 信息论一次性消息认证码安全性: 一个消息认证码方案  $\Pi = (\text{Gen}, \text{Mac}, \text{Ver})$  如果对于任意(拥有无限计算能力)的敌手  $\mathcal{A}$ , 有

$$\Pr \left[ \begin{array}{l} k \leftarrow \text{Gen}(1^\kappa); m' \leftarrow \mathcal{A}(1^\kappa); \\ t' \leftarrow \text{Mac}_k(m'); (m, t) \leftarrow \mathcal{A}(1^\kappa, m', t') \end{array} : \text{Ver}_k(m, t) = 1 \wedge m \neq m' \right] \leq \text{negl}(\kappa),$$

那么, 我们称消息认证码方案  $\Pi$  是信息论安全一次性消息认证码。

一种信息论一次性消息认证码的构造如下。令消息空间  $\mathcal{M} = \mathbb{Z}_p$ , 标签空间  $\mathcal{T} = \mathbb{Z}_p$ . 密钥  $(a, b)$  是  $\mathbb{Z}_p$  上的一对随机元素, 即密钥空间  $\mathcal{K} = \mathbb{Z}_p \times \mathbb{Z}_p$ . 标签生成算法定义如下:

$$\text{Mac}_{a,b}(m) = a \cdot m + b \pmod{p}$$

对于消息标签对  $(m, t)$ , 验证算法检查  $a \cdot m + b \pmod{p} \stackrel{?}{=} t$ , 若验证通过则输出 1, 否则输出 0.

### 2.3.6 承诺

承诺方案 (commitments) 中有两个参与方: 承诺者和接收者。承诺者可以利用承诺方案来对他所持有的消息  $m$  生成承诺  $c$ , 并将这个承诺  $c$  发送给接收方。接收方在收到承诺  $c$  之后并不能获得有关  $m$  的任意信息, 这就是承诺方案的隐藏性。之后, 承诺者可以发送  $m$  和打开信息  $d$  给接收方, 接收方根据  $d$  来验证  $c$  是不是对  $m$  的承诺。承诺方无法把之前所发送的承诺  $c$  打开为另一个消息  $m' \neq m$ , 这就是承诺方案的绑定性。

直观上, 承诺方案就类似于把秘密放入了一个不透明的信封, 其他人看不见信封中的内容。承诺者可以在之后将信封打开, 展示之前放入的秘密。我们给出承诺方案的定义:

**定义 2.3.7.** 我们假设  $\mathcal{K}$  是承诺密钥所在的空间,  $\mathcal{M}$  是消息所在的空间,  $\mathcal{C}$  是承诺所在的空间,  $\mathcal{D}$  是打开信息所在的空间。承诺方案包含以下三个算法  $(\text{Gen}, \text{Com}, \text{Ver})$ :

- $\text{ck} \leftarrow \text{Gen}(1^\kappa)$ : 这是承诺密钥生成算法, 它以安全参数  $\kappa$  作为输入, 最终输出承诺密钥  $\text{ck} \in \mathcal{K}$ .
- $(c, d) \leftarrow \text{Com}_{\text{ck}}(m)$ : 这是承诺算法, 它以承诺密钥  $\text{ck} \in \mathcal{K}$ , 消息  $m \in \mathcal{M}$  作为输入, 最终输出承诺  $c \in \mathcal{C}$  和打开消息  $d \in \mathcal{D}$ .
- $b = \text{Ver}_{\text{ck}}(c, m, d)$ : 这是承诺验证算法, 它以承诺密钥  $\text{ck} \in \mathcal{K}$ , 承诺  $c \in \mathcal{C}$ , 消息  $m \in \mathcal{M}$  和打开消息  $d \in \mathcal{D}$  作为输入, 最终输出一个比特  $b \in \{0, 1\}$  来表示通过验证 ( $b = 1$ ) 或者未能通过验证 ( $b = 0$ ).

承诺方案需要满足如下性质:

- 正确性: 对于任意的消息  $m \in \mathcal{M}$ , 我们有

$$\Pr[\text{ck} \leftarrow \text{Gen}(1^\kappa); (c, d) \leftarrow \text{Com}_{\text{ck}}(m) : \text{Ver}_{\text{ck}}(c, m, d) = 1] = 1.$$

- (计算上的) 隐藏性: 对于任意 PPT 的敌手  $\mathcal{A}$ , 我们有

$$\Pr \left[ \begin{array}{l} \text{ck} \leftarrow \text{Gen}(1^\kappa); (m_0, m_1) \leftarrow \mathcal{A}(\text{ck}); \\ b \leftarrow \{0, 1\}; (c, d) \leftarrow \text{Com}_{\text{ck}}(m_b); b' \leftarrow \mathcal{A}(c, \text{ck}, m_0, m_1) \end{array} : b = b' \right] \leq \frac{1}{2} + \text{negl}(\kappa).$$

- (计算上的) 绑定性: 对于任意 PPT 的敌手  $\mathcal{A}$ , 我们有

$$\Pr \left[ \begin{array}{l} \text{ck} \leftarrow \text{Gen}(1^\kappa); \\ (c, m_0, m_1, d_0, d_1) \leftarrow \mathcal{A}(\text{ck}) \end{array} : \begin{array}{l} \text{Ver}_{\text{ck}}(c, m_0, d_0) = \text{Ver}_{\text{ck}}(c, m_1, d_1) = 1 \\ \wedge m_0 \neq m_1 \end{array} \right] \leq \text{negl}(\kappa).$$

在现实场景下, 常用的承诺方案有基于哈希的承诺、Pedersen 承诺等。基于哈希的承诺无需承诺密钥, 承诺定义为  $c = H(m||r)$ , 其中  $H$  是抗碰撞的哈希函数,  $m$  是承诺的消息,  $r$  是随机数。当需要打开承诺时, 承诺者公布  $m, r$  (即方案的打开消息为  $d = r$ )。验证算法验证  $c \stackrel{?}{=} H(m||r)$ . Pedersen 承诺的承诺密钥  $\text{ck} = h \in \mathbb{G}$  是一个群元素, 其离散对数是未知的。假设群  $\mathbb{G}$  的生成元为  $g$ , Pedersen 承诺定义为  $c = g^m h^r$ , 其中  $m$  是承诺的消息,  $r$  是随机数。当需要打开承诺时, 承诺者公布  $m, r$  (即方案的打开消息为  $d = r$ )。验证算法验证  $c \stackrel{?}{=} g^m h^r$ .

### 2.3.7 茫然传输

茫然传输 (Oblivious Transfer, OT)<sup>7</sup>是一个两方协议, 包含两个参与方: 发送方和接收方。发送方持有两个消息  $x_0$  和  $x_1$ , 接收方持有一个选择比特  $b \in \{0, 1\}$ , 在协议的最后, 接收方可以获得消息  $x_b$ 。茫然传输协议有如下的隐私性保障: (1) 接收方无法得知发送方所持有的另一条消息  $x_{1-b}$ ; (2) 发送方无法得知接收方所持有的选择比特  $b$ .

在密码学中, 我们常常使用理想功能 (ideal functionality) 来定义原语。一个理想功能刻画了协议所希望实现的效果, 它类似于一个可信第三方的角色: 它接受所有参与方的输入, 执行正确的计算, 并将输出私密地发给参与方。一个功能描述了协议所期望达到的正确性和隐私性。

我们把茫然传输的理想功能记作  $\mathcal{F}_{\text{OT}}$ , 其具体定义如图 2.1 所示。由图 2.1 可知,  $\mathcal{F}_{\text{OT}}$  保障了接收方无法获取  $x_{1-b}$ , 同时  $\mathcal{F}_{\text{OT}}$  也保障了发送方不知道接收方收到了哪条消息, 因而  $\mathcal{F}_{\text{OT}}$  很好地展现了茫然传输的安全性。

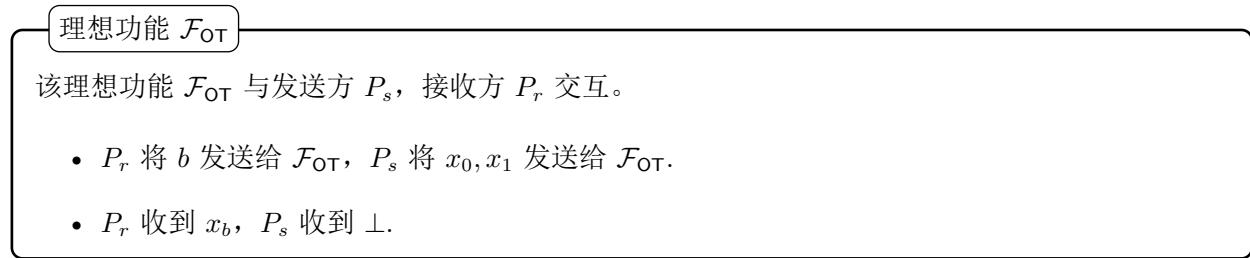


图 2.1: 茫然传输的理想功能  $\mathcal{F}_{\text{OT}}$

<sup>7</sup>也被翻译为“不经意传输”, 我们认为“茫然传输”更贴切。

# Chapter 3

## 一个安全多方计算协议的例子

本章将介绍一个具有完美隐私性的通用安全多方计算协议：假设有任意数量的  $n$  个参与方，当不超过  $t < n/2$  个参与方一起汇集他们的信息时，他们无法获得除了协议输出以外的任何额外信息。我们将在第6章严格证明：即使参与方的计算能力是无限的，他们也无法获得任何额外信息。

与第1章相同，我们仍然需要假设所有参与者都遵守协议，这种安全性被称为半诚实安全 (semi-honest security) 或被动安全 (passive security)。我们假设任意两个参与方都可以使用安全信道通信。

### 3.1 Shamir 秘密分享

本章介绍的安全多方计算协议基于 Shamir 秘密分享方案<sup>[7]</sup>。Shamir 秘密分享是一种基于有限域  $\mathbb{F}$  上的多项式的秘密分享，要求  $|\mathbb{F}| > n$ . 我们假设  $\mathbb{F} = \mathbb{Z}_p$ , 其中  $p$  是一个大素数且  $n < p$ .

首先，参与方选定  $n$  个各不相同且均不为 0 的值  $(\alpha_1, \dots, \alpha_n)$ . 要分享某个值  $s \in \mathbb{F}$ , 算法选择阶数最多为  $t$  的随机多项式  $q_s(x) \in \mathbb{F}[x]$ ，使得  $q_s(0) = s$ , 然后私下向参与方  $P_j$  发送秘密份额  $s_j = q_s(\alpha_j)$ . 根据拉格朗日插值法可以证明，这种方法产生的秘密份额满足两个基本性质：(1) 任何  $t+1$  组或更多的秘密份额可以重建秘密  $s$ ; (2) 任何  $t$  组或更少的秘密份额不包含关于  $s$  的信息. 下面，我们进行详细说明。

#### 3.1.1 拉格朗日插值法

直观地说，我们可以通过  $\ell+1$  个横坐标不同的点确定一个唯一的至多  $\ell$  次的多项式，而拉格朗日插值法给出了该多项式的构造方式。设  $q(x)$  是  $\mathbb{F}$  上至多  $\ell$  次的多项式， $(\alpha_1, \dots, \alpha_{\ell+1})$  对所有  $i \neq j$  满足  $\alpha_i \neq \alpha_j$ ，则：

$$q(x) = \sum_{i=1}^{\ell+1} q(\alpha_i) \delta_i(x)$$

其中， $\delta_i(x)$  是  $\ell$  次多项式，满足  $\delta_i(\alpha_i) = 1$  且  $\delta_i(\alpha_j) = 0, \forall j \neq i$ . 换言之， $\delta_i(x)$  的定义如下

$$\delta_i(x) = \prod_{1 \leq j \leq \ell+1, j \neq i} \frac{x - \alpha_j}{\alpha_i - \alpha_j}$$

我们来看一下这个公式为什么成立。首先，每个  $\delta_i(x)$  是  $\ell$  个单项式的乘积，是  $\ell$  次多项式，所以  $q(x) = \sum_{i=1}^{\ell+1} q(\alpha_i)\delta_i(x)$  是  $\ell$  次的多项式相加，它最多为  $\ell$  次。其次，注意到  $\delta_i(x)$  满足  $\delta_i(\alpha_i) = 1$  且  $\delta_i(\alpha_j) = 0, \forall j \neq i$ ，故  $q(x)$  在  $\alpha_i$  处的值为  $q(\alpha_i)$ 。最后，我们还需要说明  $q(x)$  是唯一的。这是因为当  $x = \alpha_i, 1 \leq i \leq \ell + 1$  时， $q(x) - \sum_{i=1}^{\ell+1} q(\alpha_i)\delta_i(x)$  的值为 0，只有零多项式满足值为 0 的位置比它的次数更多，因此  $q(x) - \sum_{i=1}^{\ell+1} q(\alpha_i)\delta_i(x)$  一定是零多项式，即  $q(x) = \sum_{i=1}^{\ell+1} q(\alpha_i)\delta_i(x)$ 。

至此我们证明了 Shamir 秘密分享的第一条性质：任何  $t + 1$  组或更多的秘密份额可以重建秘密  $s$ 。因为参与方可以通过拉格朗日插值法计算  $q_s(x)$  并得到  $q_s(0) = s$ 。对于拉格朗日插值法构造出的多项式  $q(x)$ ，令  $x = 0$ ，我们有

$$q(0) = \sum_{i=1}^{\ell+1} q(\alpha_i) \cdot \delta_i(0)$$

对于所有阶数至多为  $\ell$  的多项式  $q(x)$  成立。令  $r_i = \delta_i(0)$ ，我们得到了一组向量  $\vec{r} = (r_1, \dots, r_{\ell+1})$ ，可以将所有阶数至多为  $\ell$  的多项式  $q(x)$  在 0 处的取值表示为  $q(\alpha_i)$  的线性组合：

$$q(0) = \sum_{i=1}^{\ell+1} q(\alpha_i) \cdot r_i$$

$\vec{r} = (r_1, \dots, r_{\ell+1})$  被称为重组向量 (recombination vector)。需要注意的是， $\delta_i(x)$  不依赖于  $q(x)$ ，所以  $\delta_i(0)$  也不依赖于  $q(x)$ 。重组向量  $\vec{r}$  适用于所有多项式  $q(x)$ ，是每个参与方都可以计算的公共信息。

下面，我们证明 Shamir 秘密分享的第二个性质：任意  $t$  个或更少的秘密份额不包含关于  $s$  的信息。我们证明如下命题：对于任意秘密  $s \in \mathbb{F}$  和一组  $\vec{\alpha}$  的值  $(\alpha_1, \dots, \alpha_t)$  ( $\alpha_i$  各不相同且均不为 0)，均匀随机采样一个阶数  $\leq t$  且满足  $q(0) = s$  的函数  $q(x)$ ，Shamir 秘密分享的  $t$  个秘密份额  $(q(\alpha_i))_{i \in [t]}$  服从  $\mathbb{F}^t$  上的均匀分布。由于  $\mathbb{F}^t$  上的均匀分布显然与  $s$  无关，这也就直接证明了这些秘密份额不包含关于  $s$  的信息。

我们的证明方法是构造一个从  $\mathbb{F}^t$  到  $\mathbb{F}^t$  的可逆映射，因为从  $\mathbb{F}^t$  到  $\mathbb{F}^t$  的任何可逆映射都将  $\mathbb{F}^t$  上的均匀分布映射到  $\mathbb{F}^t$  上的均匀分布。首先，Shamir 秘密分享需要采样随机多项式  $q(x)$  满足  $q(0) = s$ ，这等价于从  $\mathbb{F}^t$  中均匀随机采样  $\vec{b} = (b_1, \dots, b_t)$ ，并令  $q_b(x) = s + \sum_{j=1}^t b_j x^j$ 。对于固定的  $s$  和  $\vec{\alpha} = (\alpha_1, \dots, \alpha_t)$ ，定义如下的映射，它将  $\vec{b} = (b_1, \dots, b_t)$  映射到  $(q_b(\alpha_i))_{i \in [t]}$ 。显然这是一个  $\mathbb{F}^t$  到  $\mathbb{F}^t$  的映射。接下来我们说明它是可逆的。给定一组  $(y_i)_{i \in [t]} \in \mathbb{F}^t$  满足  $q_b(\alpha_i) = y_i$ ，由于我们还知道  $q_b(0) = s$ ，所以我们知道了  $q_b(x)$  上  $t + 1$  个点的坐标，可以通过拉格朗日插值法来计算  $q_b(x)$  和  $\vec{b} \in \mathbb{F}^t$ 。因此，该映射是可逆的。由此可知， $t$  个秘密份额  $(q(\alpha_i))_{i \in [t]}$  服从  $\mathbb{F}^t$  上的均匀分布。

### 3.1.2 计算示例

我们再用一个具体的示例来展示 Shamir 秘密分享的计算过程。假设有 5 个参与方  $P_1, \dots, P_5$ ，敌手控制了  $t = 2$  个被攻陷参与方。参与方在  $\mathbb{F} = \mathbb{Z}_{11}$  中计算，且被分享的秘密是  $s = 6$ 。令  $\vec{\alpha} = (\alpha_1, \dots, \alpha_5) = (1, 2, 3, 4, 5)$ 。算法均匀随机地选取  $b_1, b_2 \leftarrow \mathbb{F}$ ，假设选取到了  $b_1 = 7$  和  $b_2 = 1$ ，然后我们定义

$$h(x) = s + b_1 x + b_2 x^2 = 6 + 7x + x^2 \tag{3.1}$$

并计算  $s_1 = h(1) = 6 + 7 + 1 \bmod 11 = 3$ ,  $s_2 = h(2) = 24 \bmod 11 = 2$ ,  $s_3 = h(3) = 3$ ,  $s_4 = h(4) = 6$ ,  $s_5 = h(5) = 0$ . 因此, 秘密份额为

$$(3, 2, 3, 6, 0)$$

然后,  $s_i$  被安全地发送给  $P_i$ . 现在假设参与方得到了秘密份额  $s_3$ 、 $s_4$ 、 $s_5$ . 由于  $3 > 2$ , 这些参与方可以使用拉格朗日插值法来重建秘密。

首先计算

$$\delta_3(x) = \prod_{j=4,5} \frac{x - \alpha_j}{\alpha_3 - \alpha_j} = \frac{(x-4)(x-5)}{(3-4)(3-5)} = (x^2 - 9x + 20)((3-4)(3-5))^{-1} \pmod{11}$$

由于  $(3-4)(3-5) = 2$  且  $2 \cdot 6 \bmod 11 = 1$ , 因此  $((3-4)(3-5))^{-1} \bmod 11 = 6$ . 因此,

$$\delta_3(x) = (x^2 - 9x + 20) \cdot 6 = (x^2 + 2x + 9) 6 = 6x^2 + 12x + 54 = 6x^2 + x + 10 \pmod{11}$$

同理, 计算

$$\delta_4(x) = 10x^2 + 8x + 7$$

$$\delta_5(x) = 6x^2 + 2x + 6$$

现在, 对于任何的  $s_3, s_4, s_5$ , 如果我们令

$$h(x) = s_3 \cdot \delta_3(x) + s_4 \cdot \delta_4(x) + s_5 \cdot \delta_5(x)$$

那么有  $h(3) = s_3 \cdot 1 + s_4 \cdot 0 + s_5 \cdot 0 = s_3$ ,  $h(4) = s_3 \cdot 0 + s_4 \cdot 1 + s_5 \cdot 0 = s_4$ ,  $h(5) = s_3 \cdot 0 + s_4 \cdot 0 + s_5 \cdot 1 = s_5$ . 这说明  $h(x)$  就是我们希望重建的多项式。

将  $\delta_3(x), \delta_4(x), \delta_5(x)$  代入, 得到

$$\begin{aligned} h(x) &= s_3 \delta_3(x) + s_4 \delta_4(x) + s_5 \delta_5(x) \\ &= (6s_3 + 10s_4 + 6s_5)x^2 + (s_3 + 8s_4 + 2s_5)x + (10s_3 + 7s_4 + 6s_5) \end{aligned}$$

假设  $h(x)$  的形式为  $h(x) = s + b_1x + b_2x^2$ , 有

$$s = 10s_3 + 7s_4 + 6s_5 \bmod 11$$

$$b_1 = s_3 + 8s_4 + 2s_5 \bmod 11$$

$$b_2 = 6s_3 + 10s_4 + 6s_5 \bmod 11$$

这就是从秘密份额  $s_3 = h(3)$ ,  $s_4 = h(4)$  和  $s_5 = h(5)$  计算  $h(x)$  的通式。

最后, 我们将  $s_3 = 3$ ,  $s_4 = 6$  和  $s_5 = 0$  代入上述公式, 得到

$$s = 10 \cdot 3 + 7 \cdot 6 + 6 \cdot 0 \bmod 11 = 72 \bmod 11 = 6$$

$$b_1 = 3 + 8 \cdot 6 + 2 \cdot 0 \bmod 11 = 51 \bmod 11 = 7$$

$$b_2 = 6 \cdot 3 + 10 \cdot 6 + 6 \cdot 0 \bmod 11 = 78 \bmod 11 = 1$$

计算结果与秘密分享时选择的多项式 (3.1) 一致。

如果我们只想要得到秘密值  $s$  而不是整个多项式，那么只需计算  $s = 10s_3 + 7s_4 + 6s_5 \bmod 11$ . 这里的向量  $\vec{r} = (10, 7, 6)$  就是当  $h(x)$  阶数最大为 2 时，从  $h(3), h(4)$  和  $h(5)$  计算  $h(0)$  的重组向量。

## 3.2 半诚实安全的电路计算协议

现在，我们基于 Shamir 秘密分享来构造安全多方计算协议。

### 3.2.1 算术电路

我们用算术电路来表示有限域  $\mathbb{F}$  上的任何函数。假设每个参与方恰好有一个属于有限域  $\mathbb{F}$  的输入和输出。也就是说，函数  $f : \mathbb{F}^n \rightarrow \mathbb{F}^n, (x_1, \dots, x_n) \mapsto (y_1, \dots, y_n)$ ，即  $(y_1, \dots, y_n) = f(x_1, \dots, x_n)$ .

形式化地说，这样的电路是一个有向无环图，其中的节点称为门 (gate)，边称为导线 (wire)，每个门最多有两个输入导线。首先，每个参与方  $P_i$  有一个记为  $i$  的输入门， $P_i$  为该门提供输入值  $x_i$ . 电路共有  $n$  个输入门，它们没有输入导线但可以有任意数量的输出导线，输入值被复制到它的所有输出导线上。电路有一些内部的加法门和乘法门，它们有两个输入导线和任意数量的输出导线；这些门将它们的两个输入相加或相乘后放到输出导线上。还有一种常数乘法门，它们有一个输入导线和任意数量的输出导线，这类门带有  $\mathbb{F}$  中的一个常数  $c$ ，表示执行乘以  $c$  的运算。最后，对于每个  $P_i$ ，有一个记为  $i$  的输出门，它有一个输入导线但没有输出导线。最终分配给这个门的输入导线的值将会是  $y_i$ .

电路的计算过程如下：假设输入值已经被确定（也就是将  $x_i$  输入标记为  $i$  的输入门）。假设门已经以某种（任意的）方式编号。我们取第一个所有输入导线的值都已经确定的门，计算其输出值，并将该值放在该门的输出导线上。重复此过程，直到所有导线都被赋了值。当这个过程结束时，输出值也就确定了。我们将这个过程中遍历门的顺序称为计算顺序。

注意，考虑算术电路是没有一般性损失的：任何可计算的函数都可以用“与”门和“非”门表示为多项式大小的布尔电路。它们可以通过  $\mathbb{F}$  中的操作来模拟：布尔值“真”和“假”的编码是 1 和 0，比特  $b$  的“非”运算是  $1 - b$ ，比特  $b$  和  $b'$  的“与”运算是  $b \cdot b'$ .

### 3.2.2 协议描述

我们把某个参与方的子集记作  $C$ ，它的大小最多为  $t$ ，这些参与方将在协议结束后聚在一起，尽可能地从他们所见的数据中推导额外信息。我们把  $C$  中的参与方称为被攻陷的 (corrupted)，把不在  $C$  中的参与方称为诚实的 (honest)。由于我们要计算的函数是在  $\mathbb{F}$  上的算术电路，因此，协议需要对输入值（以及中间结果）进行  $\mathbb{F}$  中的一系列加法和乘法运算，并且不泄漏除了最终结果以外的任何信息。

为了协议描述的简洁性，我们定义一些方便的符号。

**定义 3.2.1.** 给定  $\vec{\alpha} = (\alpha_1, \dots, \alpha_n)$ . 设  $a \in \mathbb{F}$  且  $f$  是一个阶数最多为  $t$  的多项式，满足  $f(0) = a$ . 我们定义  $[a; f]_t$  如下：

$$[a; f]_t = (f(\alpha_1), \dots, f(\alpha_n))$$

即，它表示由  $f$  计算出来的秘密  $a$  的秘密份额集。根据上下文，我们有时会省略阶数或多项式，写作  $[a; f]$  或  $[a]$ .

注意，符号  $[a; f]$  一方面描述了一个对象，即秘密份额集  $(f(1), \dots, f(n))$ . 另一方面，它也是一个声明：它声明了  $f(0) = a$ . 符号  $[a; f]_t$  描述了同一个对象  $(f(1), \dots, f(n))$ ，但进一步地声明了  $\deg(f) \leq t$ .

对于  $c \in \mathbb{F}$ ，我们定义如下的逐项加法、常数乘法和逐项乘法：

$$\begin{aligned}[a; f]_t + [b; g]_t &= (f(\alpha_1) + g(\alpha_1), \dots, f(\alpha_n) + g(\alpha_n)) \\ c[a; f]_t &= (cf(\alpha_1), \dots, cf(\alpha_n)) \\ [a; f]_t * [b; g]_t &= (f(\alpha_1)g(\alpha_1), \dots, f(\alpha_n)g(\alpha_n))\end{aligned}$$

容易观察到： $f(\alpha_i) + g(\alpha_i) = (f + g)(\alpha_i)$  和  $f(\alpha_i)g(\alpha_i) = (fg)(\alpha_i)$ . 因此，我们有如下的引理：

**引理 3.1.** 对于任意  $a, b, \alpha \in \mathbb{F}$  和任意在  $\mathbb{F}$  上阶数最多为  $t$  的多项式  $f, g$ ，满足  $f(0) = a$  和  $g(0) = b$ ，以下等式成立：

$$\begin{aligned}[a; f]_t + [b; g]_t &= [a + b; f + g]_t \\ c[a; f]_t &= [ca; cf]_t \\ [a; f]_t * [b; g]_t &= [ab; fg]_{2t}\end{aligned}$$

注意，我们这里定义的符号不仅描述了秘密份额集，还声明了多项式的阶数。例如，等式  $[a; f]_t * [b; g]_t = [ab; fg]_{2t}$  表明对象  $(f(1), \dots, f(n)) * (g(1), \dots, g(n))$  与  $((fg)(1), \dots, (fg)(n))$  相同，它还表明  $(fg)(0) = ab$  且  $\deg(fg) \leq 2t$ .

该引理说明了一个重要的结论：通过对份额进行计算，我们可以隐式地对秘密进行相应的计算。例如，第一个等式表明，如果秘密  $a, b$  已被分享，每个参与方在本地将他的  $a$  和  $b$  的份额相加，那么就获得了秘密  $a + b$  的秘密份额。下面，我们再定义一些参与方的行为：

**定义 3.2.2.** 给定  $\vec{\alpha} = (\alpha_1, \dots, \alpha_n)$ . 参与方  $P_i$  分发  $[a; f_a]_t$ ，指的是他选择了一个阶数  $\leq t$  的均匀随机多项式  $f_a(x)$ ，且满足  $f_a(0) = a$ ，然后对于  $j = 1, \dots, n$  将秘密份额  $f_a(\alpha_j)$  发送给  $P_j$ . 每当参与方基于多项式  $f_a$  获得了值  $a$  的份额时，我们说参与方持有  $[a; f_a]_t$ .

假设参与方持有  $[a; f_a]_t, [b; f_b]_t$ . 那么，当我们说参与方计算  $[a; f_a]_t + [b; f_b]_t$  时，这指的是每个参与方  $P_i$  计算  $f_a(\alpha_i) + f_b(\alpha_i)$ ，根据引理 3.1，这意味着现在参与方持有  $[a + b; f_a + f_b]_t$ . 我们以类似的方式定义参与方计算  $c[a; f]_t$  和  $[a; f]_t * [b; g]_t$ .

基于上述定义的工具和术语，我们可以给出一个安全地计算算术电路的协议，我们称之为“半诚实安全的电路计算协议”，如图 3.1 所示。

### 3.2.3 安全性分析

我们来分析一下协议的安全性。我们要说明协议满足以下两个性质：

- 完美的正确性：参与方接收到正确输出的概率为 1.
- 完美的隐私性：任意大小最多为  $t < n/2$  的被攻陷方子集  $C$ ，无论他们的计算能力如何，执行协议后不会得到除  $\{x_j, y_j\}_{P_j \in C}$  之外的任何信息。

为了精确地描述隐私性，我们再次使用第 1 章中提到的“基于模拟的证明范式”：要证明某个参与方只得到了信息  $X$ ，我们将证明他看到的一切信息都可以在仅知道  $X$  的情况下高效地模拟。

### 半诚实安全的电路计算协议

协议分为三个阶段：输入分享阶段、计算阶段和输出重建阶段。

**输入分享阶段。**给定  $\vec{\alpha} = (\alpha_1, \dots, \alpha_n)$ . 每个持有输入  $x_i \in \mathbb{F}$  的参与方  $P_i$  分发  $[x_i; f_{x_i}]_t$ . 然后，按照先前定义的计算顺序逐个处理电路中的门。在输入分享阶段结束后，需要确保所有输入门已被处理过。

**计算阶段。**重复以下步骤，直到所有门都被处理（然后进入下一个阶段）。考虑计算顺序中尚未处理的第一个门。根据门的类型，执行以下操作之一：

- 加法门。参与方持有  $[a; f_a]_t, [b; f_b]_t$  作为门的两个输入  $a, b$ . 参与方计算  $[a; f_a]_t + [b; f_b]_t = [a + b; f_a + f_b]_t$ .
- 带有常数  $c$  的常数乘法门。参与方持有  $[a; f_a]_t$  作为门的输入  $a$ . 参与方计算  $c[a; f_a]_t = [ca; cf_a]_t$ .
- 乘法门。参与方持有  $[a; f_a]_t, [b; f_b]_t$  作为门的两个输入  $a, b$ .
  1. 参与方计算  $[a; f_a]_t * [b; f_b]_t = [ab; f_a f_b]_{2t}$ .
  2. 定义  $h \stackrel{\text{def}}{=} f_a f_b$ . 那么  $h(0) = f_a(0)f_b(0) = ab$ , 并且各方持有  $[ab; h]_{2t}$ ; 即,  $P_i$  持有  $h(\alpha_i)$ . 每个  $P_i$  分发  $[h(\alpha_i); f_i]_t$ .
  3. 注意到  $\deg(h) = 2t \leq n - 1$ . 设  $\vec{r}$  为第3.1节定义的重组向量，即向量  $\vec{r} = (r_1, \dots, r_n)$  对于任何阶数  $\leq n - 1$  的多项式  $h$ , 都有  $h(0) = \sum_{i=1}^n r_i h(\alpha_i)$ . 参与方计算

$$\begin{aligned}\sum_{i=1}^n r_i [h(\alpha_i); f_i]_t &= \left[ \sum_{i=1}^n r_i h(\alpha_i); \sum_{i=1}^n r_i f_i \right]_t \\ &= \left[ h(0); \sum_{i=1}^n r_i f_i \right]_t = \left[ ab; \sum_{i=1}^n r_i f_i \right]_t\end{aligned}$$

**输出重建阶段。**此时，包括输出门在内的所有门都已处理。对于每个标记为  $i$  的输出门，执行以下操作：参与方持有  $[y_i; f_{y_i}]_t$ , 其中  $y_i$  是分配给输出门的值。每个  $P_j$  安全地将  $f_{y_i}(\alpha_j)$  发送给  $P_i$ , 后者使用拉格朗日插值法从  $f_{y_i}(\alpha_1), \dots, f_{y_i}(\alpha_{t+1})$  或任何其他  $t+1$  个点计算出  $y_i = f_{y_i}(0)$ .

图 3.1: 半诚实安全的电路计算协议

定义视图  $\text{view}_j$  为  $P_j$  在协议执行过程中看到的所有值。更准确地说，视图  $\text{view}_j$  是一个向量，它的第一个元素是  $x_j$ , 然后每当  $P_j$  做出随机选择时，我们将其添加到视图  $\text{view}_j$  中，每当  $P_j$  收到一条消息时，我们也将这条消息添加到  $\text{view}_j$  中。注意，视图  $\text{view}_j$  是一个随机变量，因为它依赖于  $P_j$  和其他参与方做出的随机选择。

协议满足隐私性意味着，存在一个高效的算法——模拟器  $\mathcal{S}$ ，它以  $\{x_j, y_j\}_{P_j \in C}$  作为输入，生成一个输出，其分布与  $C$  在协议期间看到的所有信息完全相同；也就是说，我们希望

$$\mathcal{S} \left( \{x_j, y_j\}_{P_j \in C} \right) \stackrel{\text{perf}}{=} \{\text{view}_j\}_{P_j \in C}$$

由于我们要证明协议具有完美的隐私性，所以要求这两个分布完全相同。

请读者注意，尽管第4章给出的通用可组合安全模型也是基于模拟的范式，但它与这里的定义有很大不同。因为如果被攻陷方可以违背协议，那么仅有正确性和隐私性是不足以保证协议的安全的。

下面我们依次证明正确性和隐私性。

**正确性。**根据引理 3.1 的前两个公式可知，当加法门和常数乘法门被处理过后，参与方持有  $[y; f_y]_t$ ，其中  $y$  是输出导线的正确值。根据引理 3.1 的第三个公式和重组向量的定义可知，当乘法门被处理过后，参与方也持有  $[y; f_y]_t$ ， $y$  是输出导线的正确值。最后，根据拉格朗日插值法的正确性可知，输出重建阶段恢复的是正确的  $y_i, i \in [n]$ 。

**隐私性。**隐私性要证明  $t$  个被攻陷方看到的值不会使他们获得除了输入和输出以外的任何信息。我们发现，被攻陷方在协议中仅从诚实方接收两种类型的消息。

- **类型 1:** 在输入分享阶段和乘法门中，他们接收诚实方计算的秘密份额集  $[x_i; f_{x_i}]_t$  和  $[h(\alpha_i); f_i]_t$  中自己的份额。
- **类型 2:** 在输出重建阶段中，对于每个被攻陷方的输出  $y$ ，他们接收  $[y; f_y]_t$  中的所有份额。

我们要说明这两类消息不会泄漏额外信息，直观上来说，这是因为它们具有以下两个特性：

- **特性 1:** 诚实方在进行输入分享和计算乘法门时发送给被攻陷方的所有值，都是使用随机多项式  $f_{x_i}$  和  $f_i$  分享的，它们的阶数最多为  $t$ 。因为最多有  $t$  个被攻陷方，所以被攻陷方集合最多接收  $t$  个份额。在第3.1.1节我们论证过，阶数最多为  $t$  的随机多项式的  $t$  个份额只是均匀随机值。
- **特性 2:** 诚实方在输出重建阶段中发送给被攻陷方的值，给定被攻陷方已知的值和输出  $y$ ，他们自己就能计算出来。这是因为在输出重建阶段之前，他们已经知道每个被攻陷方  $P_i$  自己的份额  $f_y(\alpha_i)$ 。这给了他们  $f_y(x)$  上的  $t$  个点。如果又得到了结果  $y$ ，他们额外知道了  $f_y(0) = y$ ，就可以使用拉格朗日插值法计算  $f_y(x)$ 。由  $f_y(x)$  他们可以计算出所有诚实方  $P_j$  的  $f_y(\alpha_j)$ ，而  $f_y(\alpha_j)$  正是  $P_j$  在输出重建阶段中发送的  $y$  的份额。

细心的读者会注意到，这种直观上的分析实际上也接近于一个形式化的证明。回想一下，我们的目标是构建一个模拟器  $\mathcal{S}$ ，给定被攻陷方的输入和输出，输出一个与协议中被攻陷方视图具有相同分布的值。从上面的直观分析中可以看出  $\mathcal{S}$  应该如何工作：

1. 首先，它已知被攻陷方的输入，它遵循协议执行，就好像自己是被攻陷方那样。
2. 在输入分享阶段和计算乘法门时，它通过采样均匀随机值来模拟诚实方发送给被攻陷方的份额。根据特性 1，这样产生的分布是正确的。
3. 在输出重建阶段，每个输出  $y$  会被发给被攻陷方， $\mathcal{S}$  用它“反过来”计算诚实方在  $[y; f_y]$  中的份额，使其与被攻陷方的  $t$  个模拟份额以及  $f_y(0) = y$  相一致，然后用它们来模拟诚实方发送给被攻陷方的用来重建输出的份额。根据特性 2，这样产生的分布是正确的。

至此，我们通过一个通用安全多方计算协议的例子展示了“基于模拟的证明范式”。之后，本书的第4章将进一步讨论安全模型的形式化定义，在第6章给出上述协议的形式化安全证明。

### 3.2.4 计算示例

本章的最后，我们通过计算示例来更直观地展示：在输出重建阶段，将一个输出的所有份额发给被攻陷方不会影响安全性。这意味着秘密份额不包含“结果是如何计算出来”的信息：例如， $c = a + b$  被重建且结果为 8，那么  $c$  的  $n$  个秘密份额将能够同时与  $a = 4, b = 4$  和  $a = 3, b = 5$  这两种情况都保持一致，否则，协议就泄漏了额外信息。接下来我们通过例子详细说明。

假设在协议执行过程中，变量  $a$  和  $b$  已被计算，下一步需要计算变量  $c = a + b$  并输出给  $P_1$ 。令  $\vec{\alpha} = (\alpha_1, \alpha_2, \alpha_3) = (1, 2, 3)$ 。假设  $n = 3$  且  $t = 1$ 。即，有三个参与方  $P_1, P_2, P_3$ ，其中一个可以是被攻陷的。不失一般性地，假设  $P_1$  是被攻陷的。因为  $t = 1$ ，我们使用的多项式形式为  $f(x) = b_0 + b_1x$ ，是一条直线。

我们假设  $a = 4$  通过多项式  $a(x) = 4 + x$  进行秘密分享， $b = 4$  通过多项式  $b(x) = 4 + 2x$  进行秘密分享。那么计算过程如下表所示：

变量	值	$P_1$	$P_2$	$P_3$
$a$	4	<b>5</b>	6	7
$b$	4	<b>6</b>	8	10
$c = a + b$	8	<b>11</b>	<b>14</b>	<b>17</b>

根据多项式  $a(x) = 4 + x$  得  $a(1) = 5$ ,  $a(2) = 6$ ,  $a(3) = 7$ ，根据  $b(x) = 4 + 2x$  得  $b(1) = 6$ ,  $b(2) = 8$ ,  $b(3) = 10$ 。当各方计算变量  $c = a + b$  时，他们通过本地相加计算出份额 11、14 和 17。在表格中，所有  $P_1$  看到的秘密份额都加粗显示。

$P_1$  应该只能知道  $c = 8$ ，而不知道关于  $a$  和  $b$  的任何信息（除了  $a + b = 8$  以外）。这确实是成立的，因为我们可以把另一个假设  $a = 3, b = 5$  填入表中。如果  $a = 3, b = 5$ ,  $P_1$  将有以下视图 ( $P_1$  不知道  $P_2$  和  $P_3$  的份额)：

变量	值	$P_1$	$P_2$	$P_3$
$a$	3	<b>5</b>	?	?
$b$	5	<b>6</b>	?	?
$c = a + b$	8	<b>11</b>	<b>14</b>	<b>17</b>

如果  $a(0) = 3$  且  $a(1) = 5$ ，那么就有  $a(x) = 3 + 2x$ ，从而  $a(2) = 7$  和  $a(3) = 9$ 。同理，如果  $b(0) = 5$  且  $b(1) = 6$ ，那么就有  $b(x) = 5 + x$ ，从而  $b(2) = 7, b(3) = 8$ 。 $P_1$  将这些值填入表格得到：

变量	值	$P_1$	$P_2$	$P_3$
$a$	3	<b>5</b>	7	9
$b$	5	<b>6</b>	7	8
$c = a + b$	8	<b>11</b>	<b>14</b>	<b>17</b>

这个假设与  $P_1$  所看到的内容是一致的，因为  $7 + 7 = 14$  且  $9 + 8 = 17$ 。因此， $a = 3, b = 5$  与  $a = 4, b = 4$  同样可能。

我们再研究一个相乘的例子。假设  $a = 2, b = 4$ ，用于分享它们的多项式是  $a(x) = 2 + x$  和  $b(x) = 4 - x$ 。此时，计算过程如下：

变量	值	$P_1$	$P_2$	$P_3$
$a$	2	<b>3</b>	4	5
$b$	4	<b>3</b>	2	1
$d = ab$	8	<b>9</b>	8	5
$d_1$		<b>9</b>	<b>7</b>	<b>5</b>
$d_2$		8	<b>8</b>	8
$d_3$		5	<b>6</b>	7
$c = 3d_1 - 3d_2 + d_3$	8	<b>3</b>	<b>-2</b>	<b>-7</b>

我们先看表格的上半部分, 注意到  $d = ab = 8$  的份额不在一条直线上, 因为  $d(x) = a(x)b(x)$  不是一条线, 而是一个二次多项式。实际上,  $d(x) = (2+x)(4-x) = 8+2x-x^2$ , 这与  $d(1) = 9, d(2) = 8, d(3) = 5$  相一致。

当参与方计算了本地乘积  $d_i$  之后, 协议的下一步是让每个参与方分发  $d_i$ . 在表格的下半部分,  $P_1$  使用了多项式  $d_1(x) = 9 - 2x$ ,  $P_2$  使用了多项式  $d_2(x) = 8 + 0x$ ,  $P_3$  使用了多项式  $d_3(x) = 5 + x$ . 然后, 各方通过重组向量  $(3, -3, 1)$  在本地合并它们的份额, 得到表格中的份额: 例如,  $P_1$  计算出  $3 = 3 \cdot 7 + (-3) \cdot 8 + 1 \cdot 6$ . 最后, 参与方再基于  $3, -2, -7$  重建出最终的结果  $c = 8$ .

与加法时类似, 可以假设另一种情况  $a = 1, b = 8$ , 然后填出一张一致的表格, 如下表所示:

变量	值	$P_1$	$P_2$	$P_3$
$a$	1	<b>3</b>	5	7
$b$	8	<b>3</b>	-2	-7
$d = ab$	8	<b>9</b>	-10	-49
$d_1$		<b>9</b>	<b>7</b>	<b>5</b>
$d_2$		-10	<b>8</b>	26
$d_3$		-49	<b>6</b>	61
$c = 3d_1 - 3d_2 + d_3$	8	<b>3</b>	<b>-2</b>	<b>-7</b>

由这两个例子我们可以更直观地了解到, 输出重建阶段只重建了最终的结果, 而没有泄漏关于“结果是如何计算出来的”信息。



# Chapter 4

## 安全模型

上一章介绍的协议具有完美的正确性和完美的隐私性，但它要求所有参与方都遵循协议的规定，这类敌手被称为半诚实的敌手 (semi-honest adversary)，这类协议被称为半诚实安全/被动安全的 (semi-honest secure/passively secure) 协议。在第9章中，我们还将介绍一些协议，即使这些协议的参与方未遵循协议，也能保持正确性和隐私性，这类敌手被称为恶意敌手 (malicious adversary)，这类协议被称为是恶意安全/主动安全的 (maliciously secure/actively secure) 协议。在证明这些协议的恶意安全性之前，我们需要一个形式化的定义来说明恶意安全性具体指什么。

本章，我们将介绍一个密码学的安全模型——通用可组合框架 (Universal Composability framework, UC framework)<sup>[8]</sup>。这个模型中，被证明为安全的协议无论在什么运行环境下使用，都仍然能保持安全；也就是说，它可以与任何协议“通用地”组合使用。这一优秀的性质确保了模块化设计协议的安全性，因而受到密码学家的广泛青睐。

在介绍通用可组合框架的细节之前，我们首先延续上一章中对隐私性的讨论，进一步探讨如何以类似的方式定义恶意安全性。然后，我们将说明为什么需要通过一个共同的定义来定义这两个属性，并引出通用可组合框架的基本思想。

### 4.1 隐私性与恶意安全性

#### 4.1.1 定义隐私性

让我们回顾一下，第3章中是怎样非形式化地定义隐私性的。我们首先需要选取一组被攻陷方  $C \subset \{P_1, \dots, P_n\}$ ，它们的数量最多为  $t$  ( $|C| \leq t$ )。然后选取  $(x_1, \dots, x_n)$ ，并以此为输入运行协议。令  $\text{view}_j$  是协议运行中参与方  $P_j$  的视图， $\{\text{view}_j\}_{P_j \in C}$  的值正好构成了在协议运行过程中泄露给被攻陷方  $C$  的信息（在安全性分析中，我们假设所有被攻陷方会互相串通）。隐私性的定义是，泄露的信息不会让被攻陷方得到任何他们不应该得到的信息。

显然，被攻陷方必然会得到他们自己的输入和输出，因此我们称  $\{x_j, y_j\}_{P_j \in C}$  为允许值。而  $\{\text{view}_j\}_{P_j \in C}$  的值是被攻陷方在协议运行过程中看到的所有信息，我们称其为泄漏值。

隐私性的非形式化的定义如下：

如果一个协议总是满足泄漏值不包含比允许值更多的信息，那么它具有隐私性。

这个说法仍有些模糊：如何定义一个值  $V_1$  不包含比另一个值  $V_2$  更多的信息呢？试想，如果  $V_1$  可以仅从  $V_2$  计算出来，那么显然  $V_1$  不包含比  $V_2$  更多的信息。然而，这样定义太宽泛了，特别是当我们考虑密码学的安全性时。例如，假设  $V_2 = n$  是公开的，其中  $n = pq$  是两个大素数  $p, q$  的乘积，定义  $V_1 = (p, q)$ 。由于整数的素因数是唯一的，从  $V_2 = n$  计算出  $V_1 = (p, q)$  当然是可能的。然而，如果分解大整数是一个困难的问题（使用 RSA 公钥加密时，我们希望如此），那么可以公开  $n$  并期望  $p, q$  仍然保密。

从这个角度看， $V_2 = (p, q)$  包含的信息比  $V_1 = n$  更多。因此，从密码学的角度，如果  $V_1$  可以从  $V_2$  高效地计算出来，我们就说  $V_1$  不包含比  $V_2$  更多的信息。

沿着这条思路，我们得到了一个更加明确的隐私性定义：

如果一个协议总是满足泄露值可以从允许值高效地计算出来，那么它具有隐私性。

为了形式化地说明泄露值可以从允许值“高效地计算”，我们需要给出一个多项式时间的算法，它可以允许值作为输入，输出泄露值。这个算法被称为模拟器 (simulator)。这里，请读者注意，视图  $\text{view}_j$  是由协议中的参与方生成的，他们可以做出随机选择，所以  $\{\text{view}_j\}_{P_j \in C}$  实际上不是一个固定值，而是一个随机变量。模拟器也将使用内部随机性输出一个随机变量。至此，我们给出隐私性的形式化定义：

如果存在一个高效的模拟器  $\mathcal{S}$ ，满足模拟值  $\mathcal{S}(\{x_j, y_j\}_{P_j \in C})$  和泄露值  $\{\text{view}_j\}_{P_j \in C}$  具有不可区分的分布，那么我们说这个协议具有隐私性。

这里我们需要暂停一下，探讨一个微妙的细节：“分布”具体指的是什么？我们将通过一个例子来展示，这里的“分布”需要考虑它与诚实方输出的联合分布。

考虑一个“很愚蠢”的功能  $\mathcal{F}$ ，它有两个参与方  $P_1$  和  $P_2$ ， $\mathcal{F}$  使  $P_1$  获得一个随机比特， $P_2$  什么也没有获得。显然，我们可以构造一个协议： $P_1$  选择一个随机比特并输出， $P_2$  什么也不做。这是一个实现  $\mathcal{F}$  的安全的协议。

现在我们定义一个不安全的协议： $P_1$  选择一个随机比特并输出，并把它发送给  $P_2$ 。显然，这个协议中  $P_2$  获得了他不应该获得的信息，是不安全的。然而，如果我们不考虑被攻陷方的视图与诚实方的输出的联合分布，我们会惊讶地发现，（当  $P_1$  是诚实方， $P_2$  被攻陷时） $P_2$  的视图竟然是可模拟的！模拟器  $\mathcal{S}$  只需要生成一个随机比特，它与  $P_1$  选择的随机比特具有完全相同的分布！也就是说，一个显然不安全的协议，如果只考虑被攻陷方的视图能否被模拟，竟然可以被证明是安全的！

这里的问题在于：我们必须考虑被攻陷方的视图与诚实方输出的联合分布。回到这个例子，模拟值、泄漏值、 $P_1$  的输出都是一个随机比特，但是它们还满足“泄漏值等于  $P_1$  的输出”这个关系。如果仅仅满足模拟值与泄漏值分布相同，并不能保证“模拟值也等于  $P_1$  的输出”。抽象地来说，这样的模拟并不能充分体现泄漏值所泄露的信息。因此，我们需要考虑被攻陷方的视图与诚实方输出的联合分布。回到上面的例子，当我们考虑联合分布时，模拟器输出的随机比特与  $P_1$  输出的随机比特有  $1/2$  的概率不相同，也就是有  $1/2$  的概率会模拟失败。这就符合协议不安全的直觉了。

幸运的是，在接下来介绍的通用可组合安全框架中，这一点被很好地刻画了。这里，我们可以告诉读者以下结论：如果要实现的功能是有随机性的，例如上面例子中的  $\mathcal{F}$ ，那么我们必须考虑被攻陷方的视图与诚实方输出的联合分布；如果要实现的功能是确定性的 (deterministic)，即输出完全由输入所确定，不包含随机性，此时可以仅考虑被攻陷方自己的视图<sup>1</sup>。

<sup>1</sup>直观地说，当实现的是确定性功能时，诚实方的输出是确定的值，因此如果模拟值与泄漏值有不可区分的分布，那么当考虑它们与诚实方输出的联合分布时，仍然不可区分；而实现概率功能时，诚实方的输出是一个随机变量，此时如果模拟值与泄漏值有不可区分的分布，并不能保证它们与诚实方输出的联合分布也不可区分。

### 4.1.2 定义恶意安全性

接下来，我们探讨如何定义恶意安全性 (malicious security)。恶意安全性考虑的是敌手的攻击会对协议产生怎样的影响，即协议的输出可能因为敌手的恶意行为而有所不同。也就是说，这种攻击是为了造成影响而不仅仅是为了获取信息。因此，在定义中我们采用与隐私性相同的思路，但是以“影响”而不是“信息”作为基本的度量单位。

在考虑恶意安全性时，被攻陷方是拜占庭式的 (Byzantine)，这意味着我们对它们的行为没有任何假设。具体来说，假设敌手控制了被攻陷方，每当需要他们按照协议规定的方式计算并发送某些消息时，敌手可以指示他们发送任何其他消息。而协议设计者需要确保无论敌手的行为如何，协议都能正确工作。

在进入下一步的讨论之前，我们先看两个启发性的例子。

**例 1.** 考虑图 3.1 中的半诚实安全的电路计算协议，在三方和  $t = 1$  的设定下运行。参与方计算函数  $f(x_1, x_2, x_3) = (x_1 x_2 x_3, x_2, x_3)$ 。也就是说， $P_1$  要得到  $x_1 x_2 x_3$ ，而其他参与方只能得到他们自己的输入。假设  $P_1$  被攻陷， $P_1$  将其输入  $x_1$  替换为  $x'_1 = 1$ ；也就是说，在输入分享阶段，它做了 1 的秘密分享。然后它诚实地运行协议的其余部分。最终  $P_1$  将得到  $y_1 = x'_1 x_2 x_3 = 1 \cdot x_2 x_3 = x_2 x_3$ .  $\square$

这个例子说明， $P_1$  可以通过这种方式始终得到  $x_2 x_3$  的值。请读者思考，这是不是一个安全问题？如果是，原因是什么？

**例 2.** 还是考虑图 3.1 中的半诚实安全的电路计算协议，在三方和  $t = 1$  的设定下运行。参与方计算函数  $f(x_1, x_2, x_3) = (x_1 x_2 x_3, x_2, x_3)$ 。假设我们按照这样的顺序安排电路中的门，首先计算  $[u]_t = [x_1 x_2]_t$ ，然后计算  $[y_1]_t = [ux_3]_t$ 。最后，在输出阶段，参与方将他们在  $[y_1]_t$  中的所有份额发送给  $P_1$ <sup>2</sup>。我们仍然假设  $P_1$  被攻陷。现在，假设  $P_1$  在输入共享阶段使用输入  $x'_1 = 0$  并遵循协议，然后，在执行乘法协议计算  $[u]_t = [x'_1 x_2]_t$  时，各参与方在本地计算多项式  $h$ ，其中  $h(0) = x'_1 x_2$ 。然而， $P_1$  没有按照协议规定的方式分发  $[h(\alpha_1)]_t$ ，而是分发了  $[r_1^{-1} + h(\alpha_1)]_t$ ，其中  $r_1$  是重组向量中的第一个元素。这意味着结果变成了

$$\begin{aligned}[u]_t &= r_1[r_1^{-1} + h(\alpha_1)]_t + \sum_{i=2}^n r_i[h(\alpha_i)]_t \\ &= [r_1 r_1^{-1} + r_1 h(\alpha_1) + \sum_{i=2}^n r_i h(\alpha_i)]_t \\ &= [1 + \sum_{i=1}^n r_i h(\alpha_i)]_t \\ &= [1 + x'_1 x_2]_t \\ &= [1]_t\end{aligned}$$

由于  $x'_1 = 0$ ，所以最后一个等号成立。然后， $P_1$  在乘法协议中诚实地计算  $[y_1]_t = [ux_3]_t$ ，并遵循输出重建阶段的协议。最终， $P_1$  将得到  $y_1 = ux_3 = 1 \cdot x_3 = x_3$ .  $\square$

这个例子说明， $P_1$  可以通过这种方式始终得到  $x_3$  的值。请读者思考，这是不是一个安全问题？如果是，原因是什么？

<sup>2</sup>  $P_2$  和  $P_3$  可以直接输出他们的输入，虽然协议会对他们的输入进行秘密分享然后再重建。

笼统地说，如果敌手无法通过影响协议来获得任何好处，那么这个协议就是恶意安全的。然而，需要注意的是，如果敌手能够接管某个参与方  $P_i$ ，那么有一种影响是不可避免的，就是“输入替换”。例如，如果敌手控制了参与方  $P_1$ ，那么它当然可以强迫  $P_1$  在协议中使用另一个值  $x'_1$  作为输入，而不是  $x_1$ 。于是，协议将计算  $f(x'_1, x_2, \dots, x_n)$  而不是  $f(x_1, x_2, \dots, x_n)$ 。这与诚实的  $P_1$  使用输入  $x'_1$  运行协议的情况完全一致，这显然是无法避免的。在函数计算协议中，唯一允许的影响就是输入替换。

为了精确地定义什么是允许的影响，在安全多方计算中，我们通常会定义一个解决问题的理想方式。然后，我们可以说，在这个理想的世界中，任何可能的影响都是允许的影响。我们可以想象一个完全可信的硬件盒子，允许每个参与方  $P_i$  安全地向盒子输入  $x_i$ ，而其他参与方对  $x_i$  一无所知。然后盒子计算  $(y_1, \dots, y_n) = f(x_1, \dots, x_n)$  并秘密地将  $y_i$  输出给  $P_i$ 。随后，盒子便会爆炸，其内部状态永远丢失。很容易看出，在这个理想世界中，被攻陷方所拥有的唯一权力就是把他们的输入集合起来，然后向盒子发送替代输入；也就是说，他们只能进行输入替换。

在实际运行协议时，敌手的产生影响称为实际影响。在具有安全信道的网络上运行安全函数计算协议时，敌手的实际影响是他可以代表一个或多个被攻陷方向诚实方发送任意消息。

现在，我们采用与定义隐私性时类似的思路。如果无论敌手如何使用他的实际影响，效果始终对应于允许的影响，那么我们就说协议是恶意安全的。这表明无论实际影响有何效果，我们都可以使用允许的影响获得相同的效果。这引出了我们定义恶意安全性的第一次尝试：

如果一个协议满足实际影响的效果可以通过允许的影响获得，那么它是恶意安全的。

回想一下，在定义隐私性时，我们要求协议的泄露值可以从允许值高效地计算出来。为了使恶意安全性的定义更加精确，我们将重用这种方法。也就是说，恶意安全性要求对于每个攻击协议的敌手，都可以高效地计算出一种具有相同效果的允许影响。计算这种允许影响的算法被称为模拟器 (simulator)。

如果对于每个攻击协议的敌手，都存在一个模拟器  $S$  可以高效地计算出具有相同效果的允许影响，那么该协议是恶意安全的。

对于安全函数计算协议，这个定义意味着，对于每次攻击导致了输出  $(y'_1, \dots, y'_n)$ ，都能够高效地计算出被攻陷方的输入替换，使其输入函数计算的理想盒子后，能给出相同的输出  $(y'_1, \dots, y'_n)$ 。

回到前面讲的两个例子。在第一个例子中， $P_1$  对协议的影响是始终得到  $x_2x_3$ 。这不是安全问题，因为  $P_1$  可以通过将  $x_1 = 1$  作为输入来获得这个结果，所以这是允许的影响。在第二个例子中， $P_1$  对协议的影响是始终得到  $x_3$ 。这是一个安全问题，因为没有输入替换可以让  $P_1$  始终得到  $x_3$ 。（为了更直观地理解，可以考虑  $P_2$  输入为 0 的情况，此时，无论  $P_1$  的输入如何选择，结果都将是 0，因而  $P_1$  不可能得到  $x_3$  的任何信息。然而，在这个攻击中，不管  $P_2$  的输入值是多少， $P_1$  都能得到  $x_3$ 。）

### 4.1.3 同时获得隐私性和恶意安全性

刚刚讲的后一种攻击乍一看是对正确性的攻击： $P_1$  违背了协议使得协议输出了错误的结果。然而，它实际上也是对隐私性的攻击：错误的结果对  $P_1$  泄露了  $P_3$  的输入，这本应该是保密的。正确性和隐私性以这种方式相互联系，我们不应该孤立地考虑它们。

那么，安全性完整的正式定义应该是什么呢？应该存在一个单一的模拟器，同时保证隐私性和恶意安全性。这意味着模拟器接收敌手试图影响真实协议的行为，并高效地将其转化为允许的影响（输入替换）；同时，它接收允许值，并高效地向敌手模拟泄露值。如果可以构造这样的模拟器，它本质上表明敌手在攻击中可能获得的任何东西，都可以通过定义上允许的方式获得。

#### 4.1.4 通用可组合框架概述

现在，我们要将刚才讨论的关于隐私性和恶意安全性的直观想法变成形式化的定义。通用可组合框架<sup>[8]</sup>所采用的方法是将参与方视为计算实体。

在通用可组合框架中，每个参与方是一个交互式图灵机 (interactive Turing machine, ITM)。交互式图灵机在图灵机的基础上增加了一组输入纸带和输出纸带，它们是 ITM 之间相互通信的方式。每个 ITM 还拥有唯一的身份 (identity)。在假设存在安全信道的场景下，当某个 ITM  $P_1$  在它的输出纸带上输出消息  $m$  和接收方  $P_2$ ，并执行“发送”命令，那么 ITM  $P_2$  将会在它的输入纸带上收到消息  $m$  和发送方  $P_1$ 。

直观上，ITM 类似于具有开放端口的计算机，在这些端口上它可以接收和发送消息。一个技术区别在于：ITM 还有一个长度为 1 比特的激活纸带 (activation tape)，只有当收到消息时，激活纸带被置为 1，它才会执行操作、改变状态。在协议开始时，一个称为敌手  $\mathcal{A}$  的特殊的 ITM 首先被激活，它执行一系列操作后，发送消息给下一个 ITM 使其被激活。当一个 ITM  $P_1$  被激活时，它读取在输入纸带上的消息，然后可能会执行一些计算、改变状态，最后它发送消息给下一个 ITM，结束自己的激活。如果  $P_1$  在激活结束时没有发送消息来激活下一个 ITM，那么敌手  $\mathcal{A}$  会再次被激活。因此，一个协议的运行过程就是许多 ITM 依次被激活，运行协议。注意，在每一个时刻，只有一个 ITM 在运行（处于激活状态），这就像一个激活令牌在 ITM 之间传递。

#### 理想功能

正如上文讨论恶意安全性时所讲，为了规定协议应该如何工作，需要定义一个实现协议的理想方式  $\mathcal{F}$ 。 $\mathcal{F}$  总是具有正确的输入输出行为，只泄露允许的值，并且只受到允许的影响。我们称  $\mathcal{F}$  为理想功能 (ideal functionality)，它承担了类似于可信第三方的角色，其唯一任务是具有正确的输入输出行为，以便我们将协议  $\Pi$  与  $\mathcal{F}$  进行比较。

#### 理想敌手（模拟器）

当我们尝试将协议  $\Pi$  和理想功能  $\mathcal{F}$  进行比较时，我们立即遇到了以下问题：根据理想功能  $\mathcal{F}$  的定义，它只允许非常有限的信息泄露和敌手影响。但是，协议  $\Pi$  包括  $n$  个参与方  $P_1, \dots, P_n$ ，这些参与方需要使用信道  $\mathcal{C}$  连接。所有的参与方、信道  $\mathcal{C}$  都可能会泄露信息、受到敌手影响。例如，当敌手攻陷 (corrupt) 参与方  $P_i$  时，它就获得了  $P_i$  的所有信息；敌手还可以通过影响信道  $\mathcal{C}$  来阻碍一些消息的发送和接收。但对于理想功能  $\mathcal{F}$ ，不存在参与方  $P_i$ ，也不存在信道  $\mathcal{C}$  的外部接口。

因此，我们不能期望  $\Pi$  和  $\mathcal{F}$  在行为上等价，因为从外部看来，它们的接口完全不同。参与方和信道  $\mathcal{C}$  的外部接口似乎会允许比  $\mathcal{F}$  更多的泄露和影响。

为了解决这个问题，我们引入理想敌手（又叫“模拟器”）。模拟器  $\mathcal{S}$  是一个连接到理想功能  $\mathcal{F}$  的交互式图灵机。也就是说，它看到  $\mathcal{F}$  泄露的信息并对  $\mathcal{F}$  施加允许的影响。同时，它需要模拟  $\Pi$  中所有参与方以及信道  $\mathcal{C}$  的所有外部接口。在理想世界中，模拟器  $\mathcal{S}$  在其内部运行真实敌手  $\mathcal{A}$ ，如此一来，在真实敌手  $\mathcal{A}$  的视角中，模拟器  $\mathcal{S}$  和协议  $\Pi$  至少具有了相同的外部接口。现在，我们可以合理地要求： $\Pi$  和  $\mathcal{S}^{\mathcal{F}}$  应当在行为上等价，这里  $\mathcal{S}^{\mathcal{F}}$  表示  $\mathcal{S}$  可以与  $\mathcal{F}$  互相通信。由于  $\mathcal{S}$  需要模拟所有参与方和外部资源（例如信道  $\mathcal{C}$ ），所以  $\mathcal{S}$  被称为模拟器。

请注意， $\Pi$  和  $\mathcal{S}^{\mathcal{F}}$  能否在行为上等价在很大程度上取决于模拟器  $\mathcal{S}$ 。模拟器  $\mathcal{S}$  的目的是使理想世界与真实协议对任何区分器看起来相同。也就是说，它将敌手对协议的影响转换为对理想功能的允许

影响；同时，它看到理想功能的泄露值，然后必须在协议中模拟对应的泄露值。并且，它的模拟方式必须使得真实敌手  $\mathcal{A}$  无法区分协议 II 和模拟器  $\mathcal{S}$ 。这恰恰符合我们之前对隐私性和恶意安全性的非正式定义。注意，这里我们还没有正式讨论如何模拟敌手攻陷参与方。我们将在给出完整定义时回答这个问题。

### 不可区分性

至此，我们已经了解了通用可组合框架的基本思想和概况。但是，这里还有一个问题：在设计通用可组合框架时，我们希望协议在任意组合时都是安全的；当运行一个协议时，如何模拟协议可能面对的所有外部情况呢？通用可组合框架引入了一个新的实体——环境 (environment)  $\mathcal{Z}$ 。环境  $\mathcal{Z}$  是一个特殊的交互式图灵机，它的基本功能是为协议的参与方提供输入，另外，它可以与敌手  $\mathcal{A}$  实时通信，通过向敌手  $\mathcal{A}$  发送指令来决定攻击协议的方式。

#### 通用可组合模型与独立 (stand-alone) 模型

读者可能在阅读本书之前了解过独立模型，为避免混淆，这里简要解释两者的区别。如果读者没有了解过独立模型<sup>[9-10]</sup>，可以跳过这一部分，不影响后文阅读。

在通用可组合框架被提出之前，协议的安全性分析通常是在一种叫做“独立的”(stand-alone) 模型下进行的。在这种模型下，没有引入环境  $\mathcal{Z}$ （或者环境  $\mathcal{Z}$  不允许与协议实时交互），并且协议的会话是独立运行的。然而，这样的安全性模型只能确保在“没有两个协议会话同时运行”的情况下是安全的。很多例子表明，在“独立的”模型下证明安全的协议，当同时运行两个协议会话时，敌手可以发起两个会话的联合 (coordinated) 攻击。

在通用可组合框架中，环境  $\mathcal{Z}$  的引入，模拟了协议可能面对的（可能是同时发生的）任意的外部环境，因而通用可组合框架下证明安全的协议，可以在任何情况下与任何协议同时组合。然而，从技术的角度看，允许环境  $\mathcal{Z}$  与敌手  $\mathcal{A}$  实时通信的代价是：模拟器  $\mathcal{S}$  不可以将  $\mathcal{A}$  “倒带” (rewind)<sup>a</sup>。在通用可组合框架下，模拟器必须提供“直线的”(straight-line) 模拟，这也对协议提出了更高的安全性要求。

<sup>a</sup>这里说的“倒带”是安全性证明中一种重要的技术，可以直观理解为： $\mathcal{S}$  保存了  $\mathcal{A}$  在某个时刻的“快照”，在协议运行一段时间后， $\mathcal{S}$  重新将这个“快照”导入  $\mathcal{A}$ ，从而让  $\mathcal{A}$  回到了协议的之前某一步，然后  $\mathcal{S}$  可能在继续运行时给  $\mathcal{A}$  发送不同的消息。

我们前面讲到，在协议运行的每一个时刻，只有一个交互式图灵机被激活。在引入环境  $\mathcal{Z}$  后，我们现在可以给出协议运行的完整流程：协议运行开始时， $\mathcal{Z}$  被激活， $\mathcal{Z}$  可以向参与方发送一些输入，或向敌手  $\mathcal{A}$  发送一些指令，然后敌手  $\mathcal{A}$  指定下一个被激活的参与方；当参与方被激活时，它会遵循协议，执行协议规定的程序；当敌手  $\mathcal{A}$  被激活时，它可以攻陷一些参与方，获取它们的信息，并代替被攻陷方执行操作；执行完成后，环境  $\mathcal{Z}$  又被激活，它可以再次向参与方发送输入，或向敌手  $\mathcal{A}$  发送指令；如此不断继续，环境  $\mathcal{Z}$  控制着协议运行的所有过程；……；协议运行结束时，参与方会将协议的输出发送给环境  $\mathcal{Z}$ ；当环境  $\mathcal{Z}$  完成所有的操作后，它会输出一比特的随机变量  $b \in \{0, 1\}$ 。

上面描述的是真实世界中协议的运行过程。在理想世界中，参与方不运行协议，环境  $\mathcal{Z}$  发送给参与方的输入都被直接转发给理想功能  $\mathcal{F}$ ， $\mathcal{F}$  发送给参与方的输出都直接转发给环境  $\mathcal{Z}$ ；此时，环境  $\mathcal{Z}$  的攻击指令发送给理想敌手（模拟器） $\mathcal{S}$ ，再由  $\mathcal{S}$  转发给敌手  $\mathcal{A}$ ；敌手  $\mathcal{A}$  攻陷的也不是真正的协议参与方，而是模拟器  $\mathcal{S}$  模拟的参与方。换句话说，理想世界中  $\mathcal{A}$  看到的所有东西都是  $\mathcal{S}$  模拟出来的。协议运行结束时，环境  $\mathcal{Z}$  同样会输出一比特的随机变量  $b \in \{0, 1\}$ 。

安全性的定义如下。用  $\text{EXEC}_{\Pi, \mathcal{A}, \mathcal{Z}}$  表示环境  $\mathcal{Z}$  与协议  $\Pi$  和敌手  $\mathcal{A}$  交互后输出的随机变量； $\text{EXEC}_{\mathcal{F}, \mathcal{S}^{\mathcal{A}}, \mathcal{Z}}$  表示环境  $\mathcal{Z}$  与理想功能  $\mathcal{F}$  和模拟器  $\mathcal{S}$  交互后输出的随机变量。如果  $\text{EXEC}_{\Pi, \mathcal{A}, \mathcal{Z}}$  和  $\text{EXEC}_{\mathcal{F}, \mathcal{S}^{\mathcal{A}}, \mathcal{Z}}$  有不可区分的分布，即  $\mathcal{Z}$  总是无法区分自己在理想世界还是真实世界，那么我们就说协议  $\Pi$  安全地实现了理想功能  $\mathcal{F}$ 。这表示协议  $\Pi$  和理想功能  $\mathcal{F}$  一样安全。

注意，这里的环境  $\mathcal{Z}$  实际上扮演了理想世界和真实世界的区分器，并且环境  $\mathcal{Z}$  可以任意地选择协议的输入和攻击的方式，这就模拟了协议所面对的所有可能的外部环境。如果  $\mathcal{Z}$  总是无法区分理想世界和真实世界，这就意味着在任何情况下，敌手  $\mathcal{A}$  对协议的攻击都被模拟器  $\mathcal{S}$  转化为了对理想功能  $\mathcal{F}$  的攻击，同时，协议泄露的信息也都被模拟器  $\mathcal{S}$  从允许的泄露信息中高效计算出来。那么，协议  $\Pi$  至少和  $\mathcal{F}$  一样安全。这里我们不能说协议  $\Pi$  就是安全的，而是  $\Pi$  至少和  $\mathcal{F}$  一样安全，因为  $\mathcal{F}$  实际上定义了协议满足的安全性（包括泄漏的信息和允许的影响）。

### 通用可组合定理

通用可组合模型广受欢迎的一个重要原因是通用可组合定理。通用可组合定理表明，如果  $\Pi$  是用于完成某个任务的安全协议（由理想功能  $\mathcal{F}$  定义），那么在任何情境下，都可以安全地将协议  $\Pi$  用来完成该任务。更具体地说，如果某个协议在使用  $\mathcal{F}$  作为外部资源时是安全的，那么将  $\mathcal{F}$  替换为协议  $\Pi$ ，该协议仍然是安全的。这允许我们将某个复杂协议  $\Pi_{\text{CMPLX}}$  的一部分抽象成较为简单的子协议  $\Pi$ ，然后证明两个简化了的命题：(1) 当使用理想功能  $\mathcal{F}$  作为资源时， $\Pi_{\text{CMPLX}}$  是安全的；(2) 协议  $\Pi$  安全地实现了  $\mathcal{F}$ 。然后，根据通用可组合定理可知，当  $\Pi_{\text{CMPLX}}$  使用  $\Pi$  作为子协议而不是使用  $\mathcal{F}$  时，它也是安全的。

## 4.2 形式化模型——通用可组合框架

上一节，我们讨论了通用可组合框架的设计思想，本节介绍通用可组合框架的具体细节。

### 4.2.1 真实协议及其运行

考虑一个有  $n$  个参与方的协议的运行过程： $n$  个参与方  $P_1, \dots, P_n$  在运行某个协议  $\Pi$ 。我们假设每个参与方  $P_i$  都清楚自己的唯一索引  $i$ 。此外，存在一个敌手  $\mathcal{A}$ 。我们先考虑恶意敌手的情况。稍后，在第4.2.6节中，我们将讨论半诚实敌手的情况。对于被攻陷的参与方 (corrupted party)，我们将其行为纳入  $\mathcal{A}$  的行为逻辑之中。这模拟了被攻陷方可能与敌手串通的情况，因此可以将被攻陷方的行为视为  $\mathcal{A}$  的行为来简化模型。

我们假设所有参与方通过一个异步通信网络  $\mathcal{C}$  进行通信。通常假设  $\mathcal{C}$  能够提供安全的点对点信道，每个点对点信道都提供了消息隐私性和完整性。在本节内容中，我们不深入讨论它的实现细节，而是抽象化地将  $\mathcal{C}$  视为提供物理层面安全的点对点信道。尽管  $\mathcal{C}$  提供了消息隐私性和完整性，敌手仍然可以获得一些参与方使用信道的信息，并对其施加一定的控制：

- 当任何参与方通过信道发送消息时，敌手会知道这一事实，并得知消息的长度、发送方和预定的接收方。这反映了即使在加密通信中，加密消息的长度仍可能被泄露，以及在现实网络环境中，通常可以观察到消息的发送方与接收方。
- 敌手拥有决定消息是否以及何时被传递的能力。因为我们将  $\mathcal{C}$  视为异步通信网络，其中消息的传递时间和顺序未被保证。我们允许敌手调度消息传递的顺序。

- 敌手无法获得关于消息的其他信息，也不能修改消息内容或发送方，并且不能将消息转发给非预定的接收方。

在我们的模型中，还包括了一个关键组成部分——环境，用符号  $\mathcal{Z}$  表示。环境的角色极其重要：它不仅为诚实的参与方提供输入并收集其输出，还负责在协议运行前确定哪些参与方是诚实的，哪些已被攻陷。此外，环境在协议运行的过程中可以与敌手  $\mathcal{A}$  进行交互和协作。这样的设计模拟了真实世界中，系统外部的因素可能会影响协议参与方的行为和状态。

图 4.1展示了真实协议运行的基本结构，具体如下：

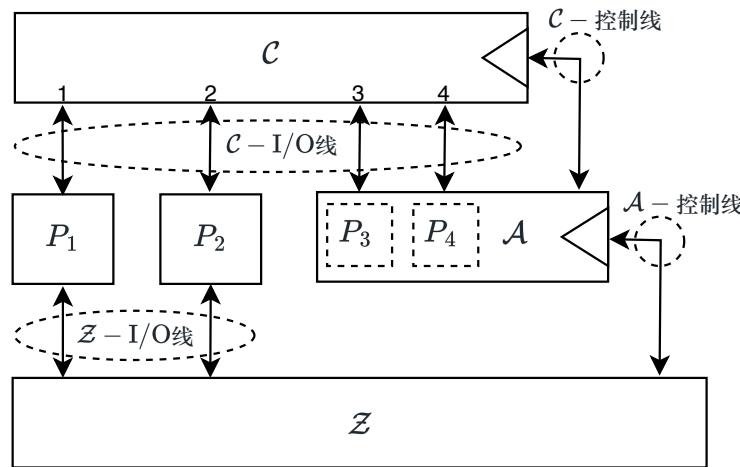


图 4.1: 真实协议运行，图中包括异步的安全信道  $\mathcal{C}$ ，敌手  $\mathcal{A}$ ，环境  $\mathcal{Z}$ ，诚实方  $P_1, P_2$ ，被攻陷方  $P_3, P_4$

- 在图 4.1 中，共有 4 个参与方， $P_1, \dots, P_4$ ，其中我们假设参与方  $P_3$  和  $P_4$  是被攻陷的，并且他们的行为现在由敌手  $\mathcal{A}$  控制。
- 通信网络  $\mathcal{C}$  有 4 条线路，分别连接到 4 个参与方。由于参与方  $P_3$  和  $P_4$  被纳入到  $\mathcal{A}$  中，因此，原本连接  $P_3$  和  $P_4$  的通信线路现在直接与  $\mathcal{A}$  相连。这种安排允许  $\mathcal{A}$  向  $P_1$  发送消息，仿佛这些消息是从  $P_3$  发出的（仅需通过标记为 3 的线路即可）。同理， $\mathcal{A}$  也可以接收  $P_1$  发往  $P_3$  的消息。我们将这些线路称为  $\mathcal{C} - \text{I/O}$  线。
- 在  $\mathcal{A}$  和  $\mathcal{C}$  之间还有一条“控制线”，我们称之为  $\mathcal{C}$ -控制线。这条线路允许  $\mathcal{A}$  接收到“泄露消息”，这些消息向  $\mathcal{A}$  泄露了关于诚实方发送消息的信息。通过这条线路， $\mathcal{A}$  还能够发送“控制消息”给  $\mathcal{C}$ ，以指示  $\mathcal{C}$  何时将消息传递给诚实方。
- 对于环境  $\mathcal{Z}$ ，它与诚实方  $P_1$  和  $P_2$  连接。通过这些连接，环境能够向诚实方发送协议输入并接收输出，我们将这些连接称为  $\mathcal{Z} - \text{I/O}$  线。环境和  $\mathcal{A}$  之间还有一条控制线，即  $\mathcal{A}$ -控制线，允许二者直接通信。通过这条线路， $\mathcal{Z}$  可以发送“控制消息”给  $\mathcal{A}$ ，为其提供下一步行动的指导，同时  $\mathcal{Z}$  也可能接收来自  $\mathcal{A}$  的“泄露消息”，这些消息包含了协议所泄露的信息。

### 4.2.1.1 真实协议的运行机制

下面，我们介绍真实协议的运行机制。在真实协议的运行过程中，多台机器依次执行一系列操作，每台机器代表一个实体。通用可组合模型规定，任何时刻只有一台机器可以执行操作。这个限制虽然看似限制了并发性，但实际上它能够模拟多个机器的并发运行。因为我们允许敌手  $\mathcal{A}$  控制消息传递的顺序，因而他可以精确地控制各机器交替执行计算的过程来模拟并发运行时任何可能的顺序。

在运行一定时间后，当前运行的机器将生成的消息发送给第二台机器。此时，原机器暂停运行，而第二台机器继续运行。这种交替执行的模式在整个协议过程中持续进行。初始阶段， $\mathcal{Z}$  开始运行并确定一组被攻陷方，这是所有参与方  $P_1, \dots, P_n$  的一个子集。 $\mathcal{Z}$  将这些信息记录在一个特殊的纸带上，该纸带对  $\mathcal{A}$  和  $\mathcal{C}$  是可见的，但对诚实方不可见，也就是说诚实方并不知道哪些参与方是被攻陷的。当  $\mathcal{Z}$  结束运行时，协议运行也随之结束。 $\mathcal{Z}$  在运行结束之前将一个比特写入一条特殊的输出纸带。

我们讨论一下机器间所有可能的消息类型：

- $\mathcal{Z}$  可以沿着  $\mathcal{Z}-\text{I/O}$  线向任何诚实方  $P_i$  传递协议输入。
- $\mathcal{Z}$  可以沿着  $\mathcal{A}$ -控制线向  $\mathcal{A}$  传递控制消息。
- 诚实方  $P_i$  或代表被攻陷方  $P_i$  的  $\mathcal{A}$  可以沿着对应的  $\mathcal{C}-\text{I/O}$  线向  $\mathcal{C}$  传递发送消息请求。请求格式为  $(\text{SEND}, (j_1, m_1), (j_2, m_2), \dots)$ ，其中  $j_k$  是消息的预期接收方， $m_k$  是消息内容。
- $\mathcal{A}$  可以沿着  $\mathcal{C}$ -控制线发送传递消息的指令。指令格式为  $(\text{DELIVER}, k)$ ，其中  $k$  是待传递消息的索引，具体细节将在下面描述。
- $\mathcal{A}$  可以沿着  $\mathcal{A}$ -控制线向  $\mathcal{Z}$  传递泄露消息。
- 诚实方  $P_i$  可以沿着  $\mathcal{Z}-\text{I/O}$  线向  $\mathcal{Z}$  传递协议输出。
- 当通信网络  $\mathcal{C}$  通过  $\mathcal{C}-\text{I/O}$  线接收到参与方  $P_i$  ( $P_i$  可能是诚实的也可能是被攻陷的) 的发送消息请求  $(\text{SEND}, (j_1, m_1), (j_2, m_2), \dots)$  后，它会将  $(i, j_1, m_1), (i, j_2, m_2), \dots$  的元组添加到内部列表  $buf$  中（初始为空），并沿着  $\mathcal{C}$ -控制线向  $\mathcal{A}$  传递泄露消息  $(\text{SENT}, i, (j_1, \text{len}(m_1)), (j_2, \text{len}(m_2)), \dots)$ 。
- 在沿着  $\mathcal{C}$ -控制线从  $\mathcal{A}$  接收到传递消息请求  $(\text{DELIVER}, k)$  后，通信网络  $\mathcal{C}$  执行以下操作：如果  $buf$  中的第  $k$  个元组是  $(i, j, m)$ ，它会沿着对应于  $P_j$  的  $\mathcal{C}-\text{I/O}$  线传递消息  $(\text{RECEIVE}, i, m)$ ，这会将消息  $m$  传递给  $P_j$ （如果  $P_j$  是诚实的）或  $\mathcal{A}$ （如果  $P_j$  是被攻陷的）；如果没有第  $k$  个元组， $\mathcal{C}$  会沿着  $\mathcal{C}$ -控制线向  $\mathcal{A}$  报告错误。

### 4.2.1.2 敌手与诚实方的通信

目前我们介绍的模型中，敌手  $\mathcal{A}$  无法直接与诚实方  $P_i$  交互：所有此类交互都是通过安全网络  $\mathcal{C}$  间接进行的。我们也可以允许敌手与诚实方直接交互：它是一种模型的变体，在这种变体中，模型没有内置的安全通信网络  $\mathcal{C}$ ，所有的消息都发送给敌手，这模拟了不安全的通信网络。参与方需要使用其他的方式（例如，公钥基础设施）来构建安全的通信信道。

### 4.2.1.3 平凡敌手

为了叙述的简洁性，我们定义一种特殊的敌手  $\mathcal{A}_{\text{triv}}$ ，称为“平凡敌手”。平凡敌手仅充当了环境和其他机器之间的“路由器”：它接收来自  $\mathcal{Z}$  的控制消息，并忠实地遵循这些消息的指示，将特定消息沿指定的路线发送到特定机器。此外， $\mathcal{A}_{\text{triv}}$  将接收到的其他机器的消息都转发给  $\mathcal{Z}$ ，同时附带着该消息来源于哪个机器的信息。

### 4.2.1.4 运行时间考虑

在讨论密码协议的安全性时，我们常常考虑的是高效的敌手，即敌手的运行时间是多项式的。这里，我们严格定义在通用可组合框架下，什么是高效的协议、敌手和环境。

- 如果环境  $\mathcal{Z}$  的运行时间是关于安全参数的多项式（这个多项式可能依赖于  $\mathcal{Z}$ ），那么我们说环境  $\mathcal{Z}$  是良好的。
- 如果在任何良好的环境  $\mathcal{Z}$  下，诚实方  $P_i$  和敌手  $\mathcal{A}$  的总运行时间（以压倒性的概率）是关于安全参数和  $\mathcal{Z}$  发送的消息长度的多项式（这个多项式依赖于  $\Pi$  和  $\mathcal{A}$ ，但不依赖于  $\mathcal{Z}$ ），那么我们说敌手  $\mathcal{A}$  与协议  $\Pi$  是兼容的。

有了这两个定义，如果  $\mathcal{Z}$  是良好的且  $\mathcal{A}$  与  $\Pi$  是兼容的，那么所有机器的运行时间总和也仍然（以压倒性的概率）是关于安全参数的多项式（这个多项式依赖于  $\Pi, \mathcal{A}, \mathcal{Z}$ ）。因此，我们给出如下定义。

- 如果平凡敌手与协议  $\Pi$  是兼容的，那么我们称  $\Pi$  是高效的。

这个定义表明，一个高效的协议中所有机器的运行时间，是关于环境  $\mathcal{Z}$  的输入长度的多项式。该定义有足够的灵活性，它允许协议处理来自环境的无限数量的请求。

## 4.2.2 理想协议及其运行

接下来，我们将介绍理想协议的运行过程。与真实协议的运行过程相比，主要的变化包括：

- 理想世界中有一个不同的敌手  $\mathcal{S}$ ，我们称之为模拟器。
- 理想世界引入了理想功能  $\mathcal{F}$  代替通信网络。理想功能是根据协议要解决的问题而定义的。例如，针对安全函数计算问题的理想功能接收所有参与方（无论是诚实的还是被攻陷的）的输入，进行函数计算，然后把结果输出给所有参与方。然而，对于不同的任务和协议，我们需要设计不同的理想功能。
- 诚实方本身在协议中不起主动作用：他们将从环境  $\mathcal{Z}$  接收到的任何输入直接传递给  $\mathcal{F}$ ，并将从  $\mathcal{F}$  收到的任何输出直接传递给  $\mathcal{Z}$ 。

图 4.2 说明了理想协议运行的基本拓扑结构。在理想协议中， $\mathcal{F}-\text{I/O}$  线替代了  $\mathcal{C}-\text{I/O}$  线：对于诚实方，这些线路直接与环境相连；而对于被攻陷方，则连接到模拟器  $\mathcal{S}$ 。 $\mathcal{F}$ -控制线连接模拟器  $\mathcal{S}$  与理想功能  $\mathcal{F}$ ，替代了通信网络的  $\mathcal{C}$ -控制线。 $\mathcal{S}$ -控制线连接环境  $\mathcal{Z}$  与模拟器  $\mathcal{S}$ ，替代了  $\mathcal{A}$ -控制线。

对比图 4.1 和图 4.2，可以发现，无论是真实协议还是理想协议，它们向环境  $\mathcal{Z}$  提供的接口是完全相同的。

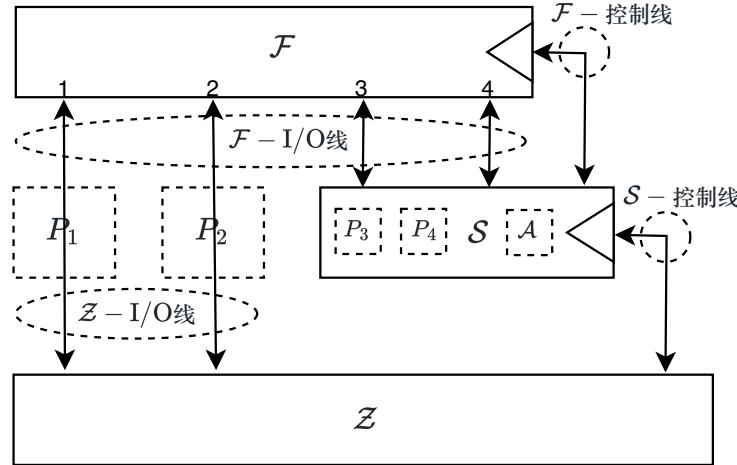


图 4.2: 理想协议运行, 图中包括理想功能  $\mathcal{F}$ , 模拟器  $\mathcal{S}$ , 环境  $\mathcal{Z}$ , 诚实方  $P_1, P_2$ , 被攻陷方  $P_3, P_4$

#### 4.2.2.1 理想协议的运行机制

下面, 我们介绍理想协议的运行机制, 它与真实协议的运行过程相似。协议开始时, 环境  $\mathcal{Z}$  开始运行并确定参与方  $P_1, \dots, P_n$  中哪些是被攻陷的。相关信息会被记录在一条特殊的纸带上, 这个纸带对模拟器  $\mathcal{S}$  和理想功能  $\mathcal{F}$  是可见的。然后, 协议随着各个机器的交替运行而进行。当  $\mathcal{Z}$  停止运行时, 协议的运行也随之结束。在运行结束之前,  $\mathcal{Z}$  将一个比特写入一条特殊的输出纸带。

同样, 我们讨论一下机器之间可能传递的所有消息类型:

- $\mathcal{Z}$  可以通过  $\mathcal{Z}$ -I/O 线路向任何诚实方  $P_i$  传递协议输入, 这些输入会直接转发给理想功能  $\mathcal{F}$ .
- $\mathcal{Z}$  能够通过  $\mathcal{S}$ -控制线向模拟器  $\mathcal{S}$  传递控制消息。
- 模拟器  $\mathcal{S}$  可通过被攻陷方对应的  $\mathcal{F}$ -I/O 线向理想功能  $\mathcal{F}$  传递协议输入, 或者通过  $\mathcal{F}$ -控制线传递控制消息给  $\mathcal{F}$ .
- 模拟器  $\mathcal{S}$  可以通过  $\mathcal{S}$ -控制线向环境  $\mathcal{Z}$  传递泄漏信息。
- $\mathcal{F}$  可以执行以下任一操作:
  - 通过与诚实方相连接的  $\mathcal{F}$ -I/O 线向环境  $\mathcal{Z}$  传递协议输出;
  - 通过与被攻陷方相连接的  $\mathcal{F}$ -I/O 线向模拟器  $\mathcal{S}$  传递协议输出;
  - 通过  $\mathcal{F}$ -控制线向模拟器  $\mathcal{S}$  传递泄漏信息。

#### 4.2.2.2 运行时间考虑

与真实协议类似, 我们要求限制理想功能、模拟器和环境是高效的。定义如下:

**定义 4.2.1.** 如果对于每一个良好的环境  $\mathcal{Z}$ , 理想功能  $\mathcal{F}$  和模拟器  $\mathcal{S}$  的总运行时间 (以压倒性的概率) 是关于安全参数和  $\mathcal{Z}$  发送的消息长度的多项式 (这个多项式依赖于  $\mathcal{F}$  和  $\mathcal{S}$ , 但不依赖于  $\mathcal{Z}$ ), 那么我们称  $\mathcal{S}$  与  $\mathcal{F}$  是兼容的。

有了这个定义，如果  $\mathcal{Z}$  是良好的，且模拟器  $\mathcal{S}$  与理想功能  $\mathcal{F}$  兼容，那么所有相关机器的总运行时间将（以压倒性概率）是关于安全参数的多项式（这个多项式依赖于  $\mathcal{F}, \mathcal{S}, \mathcal{Z}$ ）。

### 4.2.3 安全函数计算的理想功能

接下来，我们来探讨一下安全函数计算问题 (secure function evaluation, sfe) 的理想功能  $\mathcal{F}_{\text{sfe}}$  的具体定义。宏观地看， $\mathcal{F}_{\text{sfe}}$  接收所有参与方（包括诚实方和被攻陷方）的协议输入，并将计算后的结果输出给相应的参与方。 $\mathcal{F}_{\text{sfe}}$  扮演的是一个“可信第三方”的角色，它负责正确计算所有输出，并确保除了将结果传递给被攻陷方外，不泄露任何额外信息给敌手。需要注意的是， $\mathcal{F}_{\text{sfe}}$  允许敌手控制诚实方的输出接收时机和条件。并且， $\mathcal{F}_{\text{sfe}}$  并不保证所有诚实方都能接收到输出（因此，有可能被攻陷方已经获得了输出，而诚实方没有获得输出）。

$\mathcal{F}_{\text{sfe}}$  更形式化的描述如下。首先，假设待计算的函数是公开确定的。一个函数通常可以用布尔电路或算术电路来表示。我们用输入导线和输出导线来表示函数的输入和输出。假设每条输入导线都标有：

- 一个标识该输入导线的  $inputID$ .
- 一个索引，指示哪个参与方需要为该导线提供输入值。

同样地，每条输出导线都标有：

- 一个标识该输出导线的  $outputID$ .
- 接收该输出的参与方的索引列表。

理想功能  $\mathcal{F}_{\text{sfe}}$  的工作方式如下：

- 理想功能  $\mathcal{F}_{\text{sfe}}$  可以通过  $\mathcal{F}_{\text{sfe}} - \text{I/O}$  线接收协议输入，无论是来自诚实方  $P_i$  还是来自代表被攻陷方  $P_i$  的敌手  $\mathcal{S}$ 。协议输入是格式为  $(\text{INPUT}, inputID, x)$  的元组。前提是，参与方  $P_i$  必须是指定的为该输入导线提供输入值的参与方，并且必须还没有提供输入值。 $\mathcal{F}_{\text{sfe}}$  记录元组  $(\text{INPUT}, inputID, x)$ ，并通过  $\mathcal{F}_{\text{sfe}}$ -控制线向  $\mathcal{S}$  发送泄露消息  $(\text{INPUT}, inputID)$ 。
- 理想功能  $\mathcal{F}_{\text{sfe}}$  可以接收  $\mathcal{S}$  通过  $\mathcal{F}_{\text{sfe}}$ -控制线发送的控制消息，格式为  $(\text{OUTPUT}, outputID, i)$ 。前提条件是所有参与方（诚实的与被攻陷的）都已提供输入，且  $i$  在该输出值指定的接收方列表中。然后， $\mathcal{F}_{\text{sfe}}$  计算适当的输出值  $y$ ，并通过参与方  $P_i$  对应的  $\mathcal{F}_{\text{sfe}} - \text{I/O}$  线发送元组  $(\text{OUTPUT}, outputID, y)$ ，如果  $P_i$  是诚实的，则直接发给  $P_i$ ；若是被攻陷的，则发给  $\mathcal{S}$ 。

注：当理想功能收到或发送消息时，它必须知道对应的消息发送方或消息接收方。因此，为了表达的简洁性，我们通常会采用如下句式来描述一个理想功能：当收到来自  $P_i$  的消息  $(\text{INPUT}, x)$  时，执行……（描述收到消息）；当……时，将  $(\text{OUTPUT}, y)$  发送给  $P_i$ （描述发送消息）。本书后文中定义理想功能时，也会采用这样的表达方式。

#### 4.2.3.1 概率功能

目前为止，我们讨论的是为确定性函数设计的理想功能  $\mathcal{F}_{\text{sfe}}$ 。我们也可以将  $\mathcal{F}_{\text{sfe}}$  的概念扩展，使其支持概率性函数。概率性函数的不同之处在于，除了常规输入之外，它们还需要额外的随机输入，这些随机输入由理想功能  $\mathcal{F}_{\text{sfe}}$  自行生成。

#### 4.2.4 UC-安全实现

现在，我们正式给出协议  $\Pi$  在通用可组合框架下安全地实现理想功能  $\mathcal{F}$  的定义，称为“UC-安全实现”。这里的 UC 是 Universally Composable 的缩写，指通用可组合。

- 我们用  $\text{EXEC}_{\Pi, \mathcal{A}, \mathcal{Z}}$  表示环境  $\mathcal{Z}$  在真实世界中与协议  $\Pi$ 、敌手  $\mathcal{A}$  交互后输出的随机变量。
- 我们用  $\text{EXEC}_{\mathcal{F}, \mathcal{S}, \mathcal{Z}}$  表示环境  $\mathcal{Z}$  在理想世界中与理想功能  $\mathcal{F}$ 、模拟器  $\mathcal{S}$  交互后输出的随机变量。

我们使用  $X \approx Y$  来表示两个随机变量  $X$  和  $Y$  是不可区分的（参见第 2.2 节），当  $X$  和  $Y$  是 0/1 随机变量时，这等价于说  $|\Pr[X = 1] - \Pr[Y = 1]|$  是可忽略的。

我们先给出协议对于特定敌手的安全性的定义：

**定义 4.2.2.** 设  $\mathcal{A}$  是与协议  $\Pi$  兼容的敌手。如果存在与  $\mathcal{F}$  兼容的模拟器  $\mathcal{S}$ （可能依赖于  $\mathcal{A}$ ），使得对于每个良好的环境  $\mathcal{Z}$ ，都有

$$\text{EXEC}_{\Pi, \mathcal{A}, \mathcal{Z}} \approx \text{EXEC}_{\mathcal{F}, \mathcal{S}, \mathcal{Z}}.$$

那么我们说  $\Pi$  对于敌手  $\mathcal{A}$  安全地实现了理想功能  $\mathcal{F}$ 。

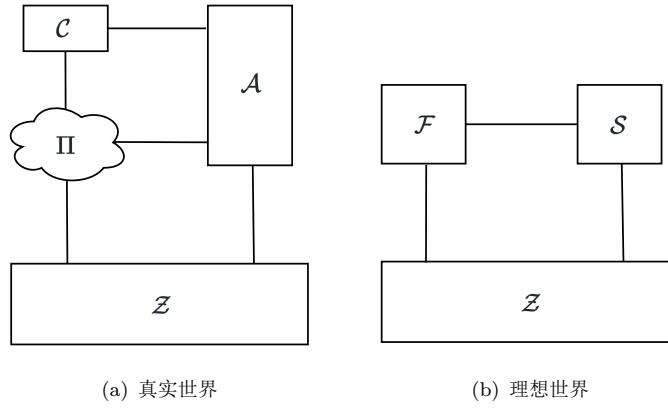


图 4.3: 定义 4.2.2 的示意图

图 4.3 给出了定义 4.2.2 的示意图。图 4.3(a) 是真实世界，其中用  $\Pi$  标记的云朵表示运行协议  $\Pi$  的机器  $P_1, P_2, \dots, P_n$ 。这些机器与环境  $\mathcal{Z}$ 、通信网络  $\mathcal{C}$  以及敌手  $\mathcal{A}$  直接交互。同时，敌手  $\mathcal{A}$  也能够直接与环境以及通信网络进行交互。

图 4.3(b) 是理想世界。在此情况下，环境  $\mathcal{Z}$  不再与诚实机器直接交互，而是与理想功能  $\mathcal{F}$  进行交互；同时，环境与敌手  $\mathcal{A}$  的直接交互也被替换为与模拟器  $\mathcal{S}$  的交互。此外，模拟器和理想功能也可以直接交互。根据定义 4.2.2，任何环境  $\mathcal{Z}$  都不能高效地区分真实世界和理想世界。

下面，我们将定义扩展到对于所有可能的敌手的安全性，它定义了一个安全协议所应具备的属性：

**定义 4.2.3.** 如果对于每个与  $\Pi$  兼容的敌手  $\mathcal{A}$ ， $\Pi$  都对于该敌手  $\mathcal{A}$  安全地实现了理想功能  $\mathcal{F}$ ，那么我们说协议  $\Pi$  UC-安全实现了理想功能  $\mathcal{F}$ 。

这个定义表示：协议  $\Pi$  安全地实现了理想功能  $\mathcal{F}$ ，当且仅当对于任何兼容的敌手  $\mathcal{A}$ ，都存在一个与  $\mathcal{F}$  兼容的模拟器  $\mathcal{S}$ （该模拟器可能依赖于  $\mathcal{A}$ ），使得对于每个良好的环境  $\mathcal{Z}$ ，都有  $\text{EXEC}_{\Pi, \mathcal{A}, \mathcal{Z}} \approx \text{EXEC}_{\mathcal{F}, \mathcal{A}, \mathcal{Z}}$ 。

这个安全定义的直观含义是：无论敌手  $\mathcal{A}$  如何尝试攻击真实协议  $\Pi$ ，造成的损害都不会超过另一敌手  $\mathcal{S}$ （该敌手可能依赖于  $\mathcal{A}$ ）在理想模型中攻击理想功能所能造成的损害。例如，在安全函数计算的场景中，可以看出理想功能  $\mathcal{F}_{\text{sfe}}$  确保了隐私性、正确性和输入独立性。如果某个协议  $\Pi_{\text{sfe}}$  UC-安全实现了  $\mathcal{F}_{\text{sfe}}$ ，那么它也一定满足隐私性、正确性和输入独立性。

在进行协议的安全性证明时，有一个非常实用的定理：

**定理 4.1.**（平凡敌手的完备性。）设  $\Pi$  是一个高效的协议。如果  $\Pi$  对于平凡敌手安全地实现了理想功能  $\mathcal{F}$ ，则  $\Pi$  UC-安全实现了  $\mathcal{F}$ 。

证明概要：假设  $\Pi$  安全地实现了对于平凡敌手的  $\mathcal{F}$ ，则存在一个与  $\mathcal{F}$  兼容的模拟器  $\mathcal{S}_{\text{triv}}$ ，使得对于每个良好的环境  $\mathcal{Z}$ ，都有  $\text{EXEC}_{\Pi, \mathcal{A}_{\text{triv}}, \mathcal{Z}} \approx \text{EXEC}_{\mathcal{F}, \mathcal{S}_{\text{triv}}, \mathcal{Z}}$ 。

现在考虑与  $\Pi$  兼容的任意敌手  $\mathcal{A}$ 。我们的目标是证明  $\Pi$  对于  $\mathcal{A}$  安全地实现了  $\mathcal{F}$ 。考虑一个良好的环境  $\mathcal{Z}$ ，它通过  $\mathcal{A}$ -控制线与  $\mathcal{A}$  交互。我们可以构造一个新的环境  $\mathcal{A}|\mathcal{Z}$ ，将  $\mathcal{A}$  嵌入到  $\mathcal{Z}$  中。环境  $\mathcal{A}|\mathcal{Z}$  通过  $\mathcal{A}_{\text{triv}}$ -控制线与平凡敌手交互，运行  $\mathcal{A}$  和  $\mathcal{Z}$  的代码（这里必须确保  $\mathcal{A}|\mathcal{Z}$  是良好的）。那么  $\text{EXEC}_{\Pi, \mathcal{A}, \mathcal{Z}} \approx \text{EXEC}_{\Pi, \mathcal{A}_{\text{triv}}, \mathcal{A}|\mathcal{Z}}$ ，因为它们运行的代码是完全一样的。如图 4.4(a)和图 4.4(b)所示。

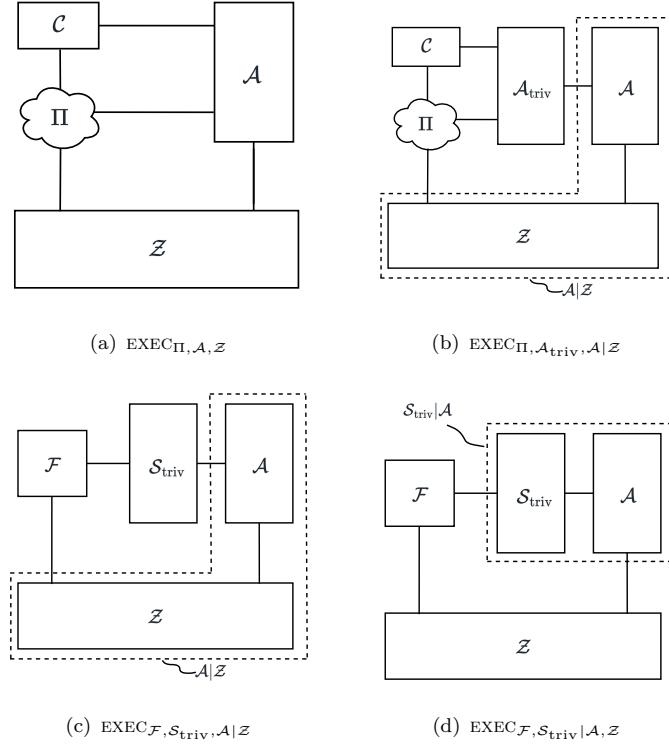


图 4.4: 定理 4.1 的证明图

又因为  $\Pi$  对于平凡敌手  $\mathcal{A}_{\text{triv}}$  安全地实现了  $\mathcal{F}$ ，我们有

$$\text{EXEC}_{\Pi, \mathcal{A}_{\text{triv}}, \mathcal{A}|\mathcal{Z}} \approx \text{EXEC}_{\mathcal{F}, \mathcal{S}_{\text{triv}}, \mathcal{A}|\mathcal{Z}}$$

最后，我们可以构造一个新的模拟器  $\mathcal{S}_{\text{triv}}|\mathcal{A}$ ，将  $\mathcal{A}$  嵌入到  $\mathcal{S}_{\text{triv}}$  中。模拟器  $\mathcal{S}_{\text{triv}}|\mathcal{A}$  通过  $\mathcal{A}$ -控制线与环境交互，运行  $\mathcal{S}_{\text{triv}}$  和  $\mathcal{A}$  的代码。可以看出，在这两种情况下（图 4.4(c)和图 4.4(d)），环境  $\mathcal{Z}$  的视图是完全一样的，因此  $\text{EXEC}_{\mathcal{F}, \mathcal{S}_{\text{triv}}, \mathcal{A}|\mathcal{Z}} \approx \text{EXEC}_{\mathcal{F}, \mathcal{S}_{\text{triv}}|\mathcal{A}, \mathcal{Z}}$ （并且  $\mathcal{S}_{\text{triv}}|\mathcal{A}$  与  $\mathcal{F}$  兼容）。

综上所述，我们有

$$\text{EXEC}_{\Pi, \mathcal{A}, \mathcal{Z}} \approx \text{EXEC}_{\Pi, \mathcal{A}_{\text{triv}}, \mathcal{A} | \mathcal{Z}} \approx \text{EXEC}_{\mathcal{F}, \mathcal{S}_{\text{triv}}, \mathcal{A} | \mathcal{Z}} \approx \text{EXEC}_{\mathcal{F}, \mathcal{S}_{\text{triv}} | \mathcal{A}, \mathcal{Z}}$$

这表明  $\text{EXEC}_{\Pi, \mathcal{A}, \mathcal{Z}} \approx \text{EXEC}_{\mathcal{F}, \mathcal{S}_{\text{triv}} | \mathcal{A}, \mathcal{Z}}$ . 也就是说，对于任何敌手  $\mathcal{A}$ ，存在模拟器  $\mathcal{S} = \mathcal{S}_{\text{triv}} | \mathcal{A}$  使得环境  $\mathcal{Z}$  无法区分真实世界和理想世界。由此，我们得出结论，协议  $\Pi$  对于每个敌手  $\mathcal{A}$  都安全地实现了理想功能  $\mathcal{F}$ ，即  $\Pi$  UC-安全实现了  $\mathcal{F}$ .  $\square$

这个定理表明，为了证明  $\Pi$  安全地实现了  $\mathcal{F}$ ，我们不需要为每一个可能的敌手分别构造模拟器，只需要为平凡敌手构造一个模拟器即可。

**注释 4.2.1.** (黑盒归约) 定理 4.1 的证明表明，在定义 4.2.3 中，对于每个敌手  $\mathcal{A}$  的模拟器  $\mathcal{S}$  实际上可以通过将  $\mathcal{A}$  视作为黑盒来构造。具体来说，我们可以设置  $\mathcal{S} = \mathcal{S}_{\text{triv}} | \mathcal{A}$  (见定理 4.1 的证明)。这个模拟器  $\mathcal{S}$  本质上是“ $\mathcal{A}$  的一个基本包装”： $\mathcal{S}$  在内部运行  $\mathcal{A}$ ，转发  $\mathcal{A}$  与  $\mathcal{Z}$  的消息，并向  $\mathcal{A}$  模拟协议中看到的消息。  $\square$

## 4.2.5 UC-安全实现的效果

UC-安全实现是一个非常强的安全概念，本节将通过几个定理来阐述它的重要性质。这里，我们只介绍概念性的证明思路，但不进行形式化的完整证明，因为那将过于抽象和晦涩。

### 4.2.5.1 并发组合的安全性

UC-安全实现的一个显著优势是，它确保了协议即便在多个实例并发运行的情况下也能保持安全性。为了准确描述这一结果，我们需要对真实协议运行和理想协议运行的概念进行扩展，以涵盖多个协议实例并发运行的情况。这一扩展的基本要点包括：

- 每个真实协议实例都配备有一个独立的通信网络。
- 每个理想协议实例都对应有一个独立的理想功能实体。

参见图 4.5 和图 4.6，它们将图 4.1 和图 4.2 扩展到两个协议实例并发运行的情形。在这些示例中，四个参与方  $P_1, \dots, P_4$  正在运行同一协议  $\Pi$  的两个独立实例。无论是在真实运行还是理想运行中，参与方  $P_1$  和  $P_2$  均为诚实方，而  $P_3$  和  $P_4$  都是被攻陷方。

值得注意的是，由于图 4.5 中出现的两个通信网络是独立的，如果第一个实例中的  $P_1$  向第一实例中的  $P_2$  发送消息，敌手无法将该消息转发给第二个实例中的  $P_2$ . 同理，在图 4.6 展示的理想运行场景中，由于两个理想功能是独立的，模拟器  $\mathcal{S}$  无法在这两个理想功能之间制造任何交互行为。

为了模拟这一点，我们为每个协议实例分配一个唯一的“会话 ID”(session ID,  $\text{sid}$ )，并将其包括在每一条协议消息中。以安全函数计算功能  $\mathcal{F}_{\text{sfe}}$  为例，在并发运行的情形下，它接收的输入格式为  $(\text{INPUT}, \text{sid}, \text{inputID}, x)$ ，输出格式为  $(\text{OUTPUT}, \text{sid}, \text{outputID}, y)$ . 在研究如何模拟多个协议或同一协议的多个实例的运行时，引入“会话 ID”的概念至关重要。所有参与方都可以通过它来识别消息属于哪个协议实例。在真实世界运行中，每个机器  $P_i$  不仅要用其索引  $i$  初始化，还要用其会话 ID 进行标识。同样，在理想世界运行中，每个理想功能也需要根据其会话 ID 进行初始化。

我们可以证明以下定理：

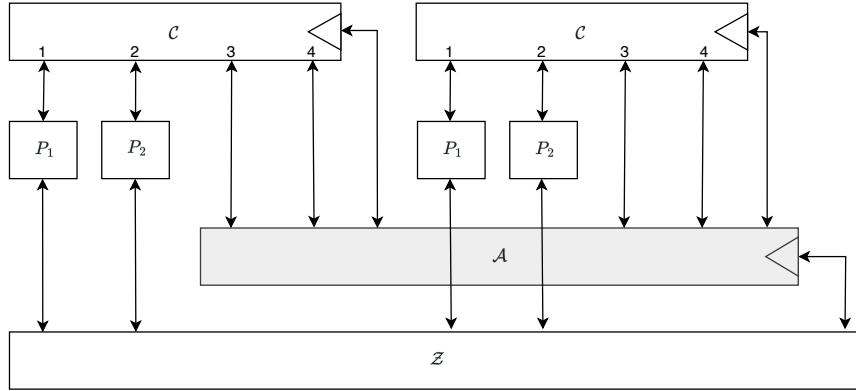


图 4.5: 两个真实协议实例并发运行

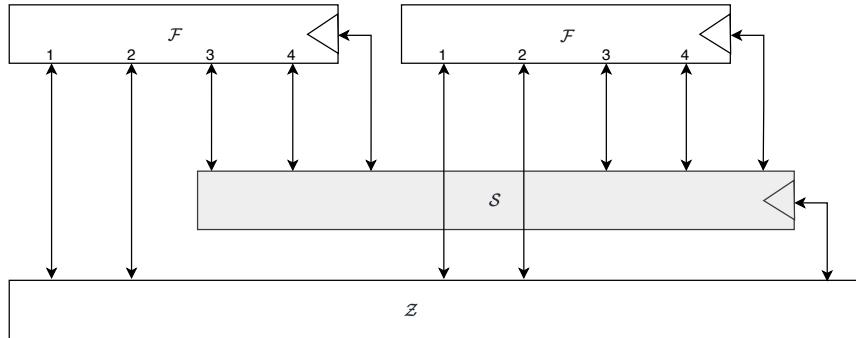


图 4.6: 两个理想协议实例并发运行

**定理 4.2.** (并发组合定理) 如果  $\Pi$  在单实例设置中是一个高效的协议, 那么它在多实例设置中仍然是高效的。此外, 如果  $\Pi$  在单实例设置中  $UC$ -安全实现了  $\mathcal{F}$ , 那么它在多实例设置中也  $UC$ -安全实现了  $\mathcal{F}$ .

**证明概要:** 这里我们不深入讨论有关运行时间的问题, 它可以通过一些简单(但稍显枯燥)的分析得以证明。我们将重点放在安全性上。

假设单实例安全性成立, 即存在一个与  $\mathcal{F}$  兼容的模拟器  $\mathcal{S}_{\text{triv}}$ , 使得对于每个与平凡敌手  $\mathcal{A}_{\text{triv}}$  进行交互的良好的单实例环境  $\mathcal{Z}$ , 都有  $\text{EXEC}_{\Pi, \mathcal{A}_{\text{triv}}, \mathcal{Z}} \approx \text{EXEC}_{\mathcal{F}, \mathcal{S}_{\text{triv}}, \mathcal{Z}}$ .

证明过程本质上是一个混合论证 (hybrid argument)。为简单起见, 我们仅用两个协议实例来说明证明思路。根据定理 4.1 的推广, 我们可以假设多实例真实世界中的敌手是平凡敌手, 它只是在环境和其他机器之间传递消息, 如图 4.7(a)所示。未标记的方框代表简单的路由机, 底部的路由机会根据消息中的会话 ID, 将来自  $\mathcal{Z}$  的消息转发到上方的未标记方框。

基于单实例安全性, 我们可以将第一个真实世界协议实例替换为其对应的理想世界协议实例, 如图 4.7(b)所示。为了形式化这一步骤, 我们定义了一个单实例环境  $\mathcal{Z}_1$ , 它封装了第二个协议实例 (如图中的虚线框所示)。单实例安全性保证  $\text{EXEC}_{\Pi, \mathcal{A}_{\text{triv}}, \mathcal{Z}_1} \approx \text{EXEC}_{\mathcal{F}, \mathcal{S}_{\text{triv}}, \mathcal{Z}_1}$ , 这表明多实例环境  $\mathcal{Z}$  无法区分图 4.7(a)中的真实世界与图 4.7(b)中的混合世界。

再次利用单实例安全性, 我们同样可以将第二个真实世界协议实例替换为其对应的理想世界协议实例。这如图 4.7(c)所示。为了形式化地证明这一步, 我们定义了另一个单实例环境  $\mathcal{Z}_2$ , 它封装了第

一个协议实例 (如图中虚线框所示)。单实例安全性意味着  $\text{EXEC}_{\Pi, \mathcal{A}_{\text{triv}}, \mathcal{Z}_2} \approx \text{EXEC}_{\mathcal{F}, \mathcal{S}_{\text{triv}}, \mathcal{Z}_2}$ , 这意味着多实例环境  $\mathcal{Z}$  无法区分图 4.7(b)中的混合世界和图 4.7(c)中的理想世界。

最后, 我们定义理想世界中的多实例模拟器, 它由简单的消息路由机制和两个单实例模拟器  $\mathcal{S}_{\text{triv}}$  的副本组成。结合上述分析, 环境  $\mathcal{Z}$  无法区分自己在图 4.7(a)的真实世界还是在图 4.7(c)的理想世界。定理得证。  $\square$

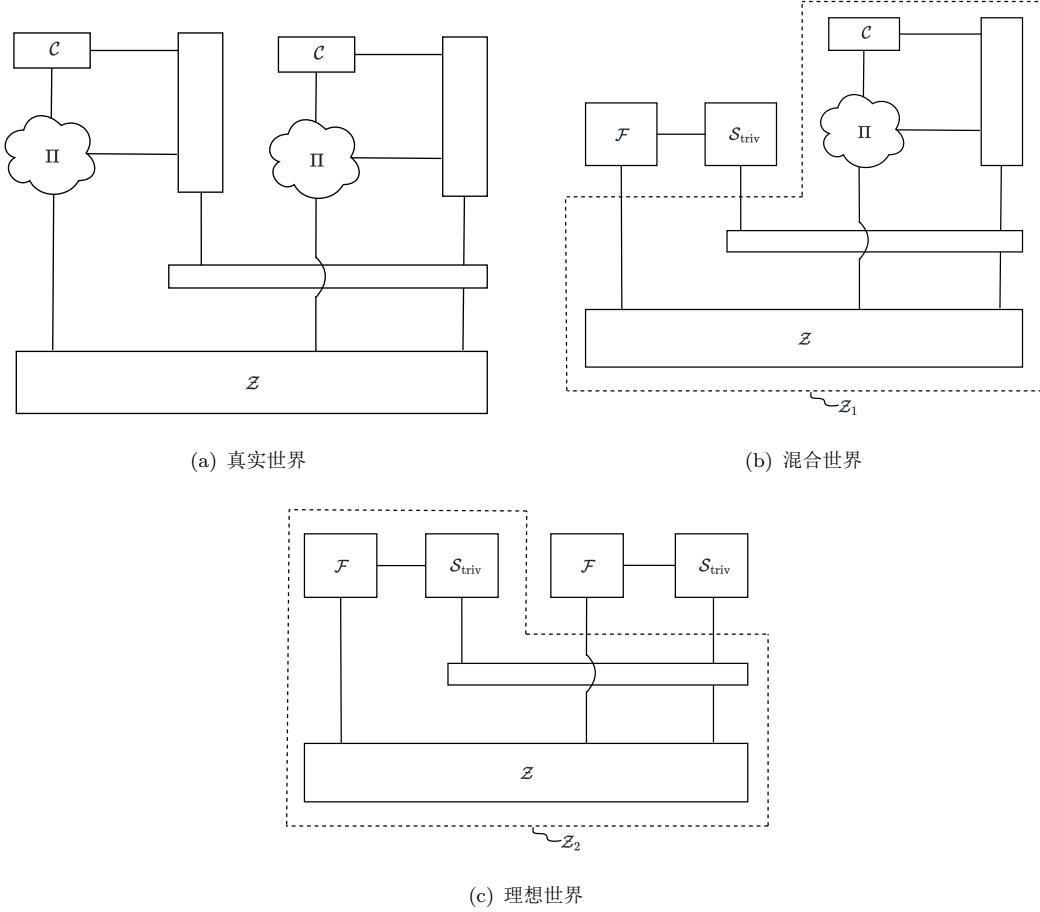


图 4.7: 定理 4.2 的证明图

#### 4.2.5.2 子协议组合的安全性

安全实现的另一个效果是, 我们可以在更大的协议中使用安全协议作为子协议, 而不会影响整体协议的安全性。同样地, 我们在这里仅简要介绍定理的证明思路, 而不进行形式化的完整证明。假设我们设计了一个协议  $\Pi$ , 我们想证明  $\Pi$  安全地实现了某个理想功能  $\mathcal{F}$ . 进一步假设,  $\Pi$  中使用了另一个协议  $\Pi^*$  作为子协议, 而  $\Pi^*$  已被证明安全地实现了另一个理想功能  $\mathcal{F}^*$ .

我们采用模块化的证明方法。首先考虑一个混合协议  $\Pi_{\mathcal{F}^*}$ , 在该协议中, 子协议  $\Pi^*$  被其相应的理想功能  $\mathcal{F}^*$  所替代。通常,  $\Pi_{\mathcal{F}^*}$  比  $\Pi$  本身要简单得多, 也更容易分析, 因为它移除了  $\Pi^*$  的所有实现细节。那么, 我们有如下结论: 如果混合协议  $\Pi_{\mathcal{F}^*}$  安全地实现了  $\mathcal{F}$ , 那么这意味着  $\Pi$  安全地实现了  $\mathcal{F}$ .

参考图 4.8可以获得更直观的理解。在图 4.8(a)中, 协议  $\Pi$  使用协议  $\Pi^*$  作为子协议。为了简洁,

这里假设只有两个参与方，并且未画出敌手和环境。标记为  $P_1$  和  $P_2$  的机器运行了  $\Pi$  的“顶部”部分（即不包括子协议  $\Pi^*$  的部分），它们通过安全通信网络  $\mathcal{C}$  作为对等方进行通信。标记为  $P_1^*$  和  $P_2^*$  的机器运行子协议  $\Pi^*$ ，它们通过另外的通信网络  $\mathcal{C}^*$  作为对等方进行通信。机器  $P_1$  和  $P_1^*$  都属于第一个参与方，可以将它们视为在同一台计算机上运行的两个不同程序，因此  $P_1$  可以直接向  $P_1^*$  提供输入，并从  $P_1^*$  接收输出。同理，机器  $P_2$  和  $P_2^*$  都属于第二个参与方。

图 4.8(b)呈现了被称为混合协议的  $\Pi_{\mathcal{F}^*}$ 。之所以称其为混合协议，是因为它既有真实世界的元素 ( $P_1$ 、 $P_2$  和  $\mathcal{C}$ )，也有理想世界的元素 (理想功能  $\mathcal{F}^*$ )。在这个混合协议中，输入不是由  $P_1$  传递给  $P_1^*$ ，而是直接传递给理想功能  $\mathcal{F}^*$ 。类似地， $P_1$  不是从  $P_1^*$  接收输出，而是直接从理想功能  $\mathcal{F}^*$  接收输出。

这种混合协议的运行模型在本质上与真实世界协议运行非常相似，不同之处在于现在同时存在通信网络  $\mathcal{C}$  和理想功能  $\mathcal{F}^*$ 。

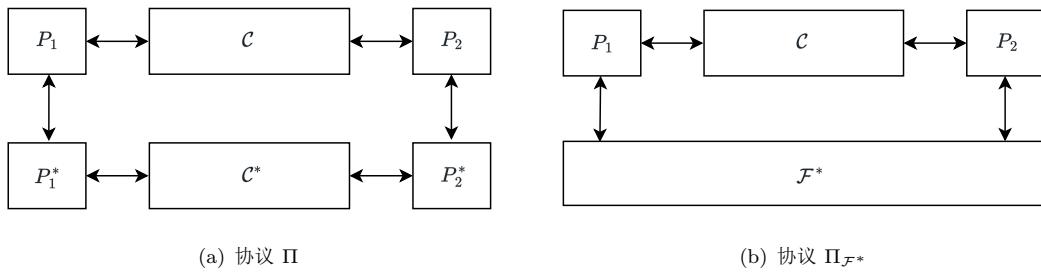


图 4.8: 混合协议

可以证明以下定理：

**定理 4.3.** (子协议组合定理) 如果以下条件均成立：

- $\Pi^*$  是一个高效的协议，它安全地实现了理想功能  $\mathcal{F}^*$ .
- $\Pi$  是一个高效的协议，它使用  $\Pi^*$  作为子协议。
- 混合协议  $\Pi_{\mathcal{F}^*}$  (其中理想功能  $\mathcal{F}^*$  取代了子协议  $\Pi^*$ ) UC-安全实现了  $\mathcal{F}$ .

那么  $\Pi$  UC-安全实现了  $\mathcal{F}$ .

**证明概要：**首先，由于  $\Pi^*$  安全地实现了  $\mathcal{F}^*$ ，因此存在与  $\mathcal{F}^*$  兼容的模拟器  $S_{\text{triv}}^*$ ，使得对于每个良好的环境  $\mathcal{Z}$ ，我们有  $\text{EXEC}_{\Pi^*, \mathcal{A}_{\text{triv}}^*, \mathcal{Z}} \approx \text{EXEC}_{\mathcal{F}^*, S_{\text{triv}}^*, \mathcal{Z}}$ 。这里， $\mathcal{A}_{\text{triv}}^*$  是与  $\Pi^*$  交互的平凡敌手。

其次，由于  $\Pi_{\mathcal{F}^*}$  安全地实现了  $\mathcal{F}$ ，因此对于任何与  $\Pi_{\mathcal{F}^*}$  兼容的敌手  $\mathcal{A}$ ，都存在一个与  $\mathcal{F}$  兼容的模拟器  $\mathcal{S}$ ，使得对于任意一个良好的环境  $\mathcal{Z}$ ，都有  $\text{EXEC}_{\Pi_{\mathcal{F}^*}, \mathcal{A}, \mathcal{Z}} \approx \text{EXEC}_{\mathcal{F}, \mathcal{S}, \mathcal{Z}}$ 。

接下来的证明本质上是一个混合论证。根据定理 4.1 的推广，我们可以假设在协议  $\Pi$  的真实世界运行中面临的敌手是平凡敌手，如图 4.9(a)所示。图中，被标记为  $\Pi^*$  的云朵表示运行子协议  $\Pi^*$  的诚实机器，被标记为  $\Pi_{\text{top}}$  的云朵代表运行  $\Pi$  的“顶部”(即不包括子协议  $\Pi^*$  的部分)的诚实机器。未标记的方框代表了一个简单的路由机制，被标记为  $\mathcal{A}_{\text{triv}}^*$  的方框是与子协议  $\Pi^*$  交互的平凡敌手。未标记的方框会根据消息中的会话 ID，将来自  $\mathcal{Z}$  的消息路由至  $\Pi_{\text{top}}$  或  $\mathcal{A}_{\text{triv}}^*$ 。

由于  $\Pi^*$  安全地实现了  $\mathcal{F}^*$ ，我们可以用  $\mathcal{F}^*$  替换  $\Pi^*$ ，如图 4.9(b)所示。为了形式化地证明这一步，我们定义了一个环境  $\mathcal{Z}_{\text{top}}$ ，它封装了  $\Pi_{\text{top}}$  (如图中标记为  $\mathcal{Z}_{\text{top}}$  的虚线框所示)。 $\Pi^*$  能够安全地实

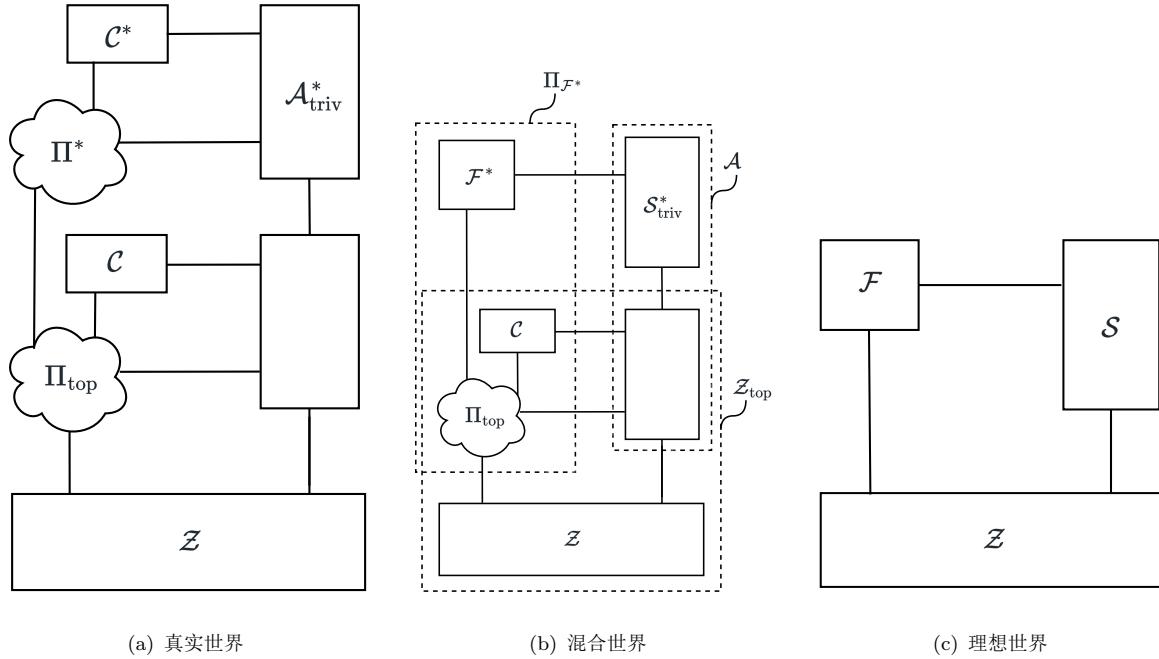


图 4.9: 定理 4.3 的证明图

现  $\mathcal{F}^*$  这一事实意味着  $\text{EXEC}_{\Pi^*, \mathcal{A}_{\text{triv}}^*, \mathcal{Z}_{\text{top}}} \approx \text{EXEC}_{\mathcal{F}^*, \mathcal{S}_{\text{triv}}^*, \mathcal{Z}_{\text{top}}}$ ，这意味着环境  $\mathcal{Z}$  无法区分图 4.9(a)中的真实世界和图 4.9(b)中的混合世界。

由于  $\Pi_{\mathcal{F}^*}$  安全地实现了  $\mathcal{F}$ ，我们可以用  $\mathcal{F}$  替换  $\Pi_{\mathcal{F}^*}$ ，如图 4.9(c)所示。为了形式化地证明这一步，我们定义了一个敌手  $\mathcal{A}$ ，它包括路由器和  $\mathcal{S}_{\text{triv}}^*$ （如图中标记为  $\mathcal{A}$  的虚线框所示）。 $\Pi_{\mathcal{F}^*}$  能够安全地实现  $\mathcal{F}$  意味着  $\text{EXEC}_{\Pi_{\mathcal{F}^*}, \mathcal{A}, \mathcal{Z}} \approx \text{EXEC}_{\mathcal{F}, \mathcal{S}, \mathcal{Z}}$ ，表明环境  $\mathcal{Z}$  无法区分图 4.9(b)中的混合世界与图 4.9(c)中的理想世界。

综上所述，环境  $\mathcal{Z}$  无法区分图 4.9(a)中的真实世界与图 4.9(c)中的理想世界，即  $\Pi$  UC-安全实现了  $\mathcal{F}$ 。定理得证。□

子协议组合定理在更一般的场景下依然成立。例如，协议  $\Pi$  的一个实例可能会调用子协议  $\Pi^*$  的多个实例，这些子协议实例可能是并发运行的。与定理 4.3的证明中使用的方法类似，我们可以将每个子协议  $\Pi^*$  的实例替换为理想功能  $\mathcal{F}^*$  的一个实例，从而得到相应的混合协议  $\Pi_{\mathcal{F}^*}$ 。定理 4.3在这种更一般的场景中同样适用。

**注释 4.2.2.**（适用于所有环境的模拟器的作用）在定理 4.2和定理 4.3的证明中使用的技术体现了在定义协议安全性时为什么要将敌手和环境分开。定义中要求：必须有一个适用于所有环境的模拟器。这就允许我们在证明组合定理时，将系统的某些部分构建为环境，同时将系统的其他部分替换为它们的理想功能。□

#### 4.2.5.3 安全实现的传递性

在第4.2.5.1节中，我们介绍了混合协议的概念，这种协议是由运行诚实方代码的普通机器和理想功能的组合而成。真实协议是一种特殊的混合协议，它仅使用了表示安全通信网络的理想功能  $\mathcal{C}$ 。理想协议也是一种特殊的混合协议，其中机器只在环境和理想功能之间传递输入和输出。因此，包括真实世界

和理想世界中的所有协议都可以被看作混合协议。

UC-安全实现的概念也可以扩展至任意混合协议。对于两个混合协议  $\Pi$  和  $\Pi'$ , 如果对于每一个与  $\Pi$  兼容的敌手  $\mathcal{A}$ , 都存在一个依赖于  $\mathcal{A}$  且与  $\Pi'$  兼容的敌手  $\mathcal{A}'$ , 使得对于任何行为良好的环境  $\mathcal{Z}$ , 都有  $\text{EXEC}_{\Pi,\mathcal{A},\mathcal{Z}} \approx \text{EXEC}_{\Pi',\mathcal{A}',\mathcal{Z}}$ , 则我们说  $\Pi$  UC-安全实现了  $\Pi'$ . 在这种情况下, 我们没有必要在术语和符号上区分支手和模拟器。

注意到, 平凡敌手的完备性定理 (定理 4.1)、并发组合定理 (定理 4.2) 以及子协议组合定理 (定理 4.3) 都可以很容易地推广到任意混合协议。我们还可以证明一个很有用的传递性质:

**定理 4.4.** 假设  $\Pi, \Pi', \Pi''$  是高效的混合协议。如果  $\Pi$  UC-安全实现了  $\Pi'$  且  $\Pi'$  UC-安全实现了  $\Pi''$ , 则  $\Pi$  UC-安全实现了  $\Pi''$ .

证明概要: 首先, 协议  $\Pi$  能够安全地实现协议  $\Pi'$ , 这意味着对于每个与  $\Pi$  兼容的敌手  $\mathcal{A}$ , 都存在一个与  $\Pi'$  兼容的敌手  $\mathcal{A}'$ , 使得  $\text{EXEC}_{\Pi,\mathcal{A},\mathcal{Z}} \approx \text{EXEC}_{\Pi',\mathcal{A}',\mathcal{Z}}$ . 其次,  $\Pi'$  安全实现了  $\Pi''$ , 这意味着对于每个与  $\Pi'$  兼容的敌手  $\mathcal{A}'$ , 都存在一个与  $\Pi''$  兼容的敌手  $\mathcal{A}''$ , 使得  $\text{EXEC}_{\Pi',\mathcal{A}',\mathcal{Z}} \approx \text{EXEC}_{\Pi'',\mathcal{A}'',\mathcal{Z}}$ . 由此可得, 对于每个与  $\Pi$  兼容的敌手  $\mathcal{A}$ , 都存在一个与  $\Pi''$  兼容的敌手  $\mathcal{A}'$ , 使得  $\text{EXEC}_{\Pi,\mathcal{A},\mathcal{Z}} \approx \text{EXEC}_{\Pi'',\mathcal{A}'',\mathcal{Z}}$ .  $\square$

#### 4.2.6 定义半诚实安全性

在定义半诚实安全性时, 需要对真实世界中的敌手 (以及理想世界中的模拟器) 的行为增加一些限制, 我们称这个修改后的框架为受限框架。幸运的是, 我们仍然可以在原有的框架内定义受限的敌手 (或模拟器) 的概念, 而无需从头开始定义一个新的受限框架。

##### 4.2.6.1 定义受限框架

**受限框架的真实世界。**首先, 我们讨论受限框架的真实世界。与无限制框架不同, 在受限框架中, 被攻陷方不是简单地被并入敌手, 而是使用独立于敌手的实体来表示。此外, 就像诚实方一样, 被攻陷方直接从环境接收其输入, 并将其输出直接传递给环境。它们运行的是协议的“诚实但泄露”的版本, 即它们完全按照协议运行, 但会向敌手报告以下信息:

- 从通信网络接收到的所有消息。
- 它们生成的所有随机比特串 (假设所有随机性最终都源自这些比特串)。

这些报告是具有一些特殊规则的特殊消息。具体来说, 在收到报告后, 合法的敌手必须做出以下行动之一:

- 立即将控制权交还给报告的发送方。
- 向环境发送一个或多个报告序列, 然后将控制权交还给报告的发送方。

此外, 合法的环境必须遵守以下规则: 在收到敌手的报告时, 它必须立即将控制权交还给敌手, 或者停止运行。

上述规则本质上说明, 在受限框架中, 敌手 (以及环境) 只能够在不中断正常协议流程的前提下, 持续监控被攻陷方的内部状态。

受限框架中，通信网络  $\mathcal{C}$  的行为与无限制框架中的行为一致，但敌手与  $\mathcal{C}$  的交互仅限于  $\mathcal{C}$ -控制线，而不能通过  $\mathcal{C}$  - I/O 线。只有诚实方和被攻陷方可以通过  $\mathcal{C}$  - I/O 线与  $\mathcal{C}$  通信。不过，敌手仍然可以决定是否以及何时将消息传递给任何参与方（诚实方或被攻陷方）；但它既不能知道或更改在两个诚实方之间发送的消息内容，也不能更改从被攻陷方发送到任何其他方的任何消息的内容。当然，敌手能够知道发往被攻陷方的消息内容以及被攻陷方发出的消息内容。

相同地，我们可以定义平凡敌手  $\mathcal{A}_{\text{triv}}$ ，它只按照环境的指示行动。不过，由于在这个框架中敌手无法控制被攻陷方，当被攻陷方向  $\mathcal{A}_{\text{triv}}$  报告信息时， $\mathcal{A}_{\text{triv}}$  仅能将这些信息直接发送给环境。

**受限框架的理想世界。**在受限框架的理想世界中，所有  $\mathcal{F}$  - I/O 线直接将环境  $\mathcal{Z}$  连接到理想功能  $\mathcal{F}$ 。这意味着模拟器  $\mathcal{S}$  不能更改被攻陷方的输入或输出。然而，与真实世界相对应，理想世界的被攻陷方是“诚实但泄露信息”的。这意味着当理想世界的被攻陷方从环境  $\mathcal{Z}$  收到输入或从理想功能  $\mathcal{F}$  收到输出时，这些消息也会发送给模拟器  $\mathcal{S}$ 。

#### 4.2.6.2 安全定义及其效果

定义 4.2.2 和定义 4.2.3 都可以直接应用到受限框架：为了与无限制框架相区分，对于定义 4.2.2，我们说  $\Pi$  对于半诚实的敌手  $\mathcal{A}$  安全实现了  $\mathcal{F}$ ；对于定义 4.2.3，我们说  $\Pi$  对于半诚实敌手 UC-安全实现了  $\mathcal{F}$ 。

很容易发现，平凡敌手的完备性定理（定理 4.1）、并发组合定理（定理 4.2）、子协议组合定理（定理 4.3）以及传递性定理（定理 4.4）也能在受限框架中以完全相同的方式证明，本质上只是将“UC-安全实现”替换为“对于半诚实敌手 UC-安全实现”。

#### 4.2.7 不同类型的攻陷方式

在前面的定义中，环境  $\mathcal{Z}$  需要在协议开始之前就确定一组被攻陷方，这种攻陷方式称为“静态攻陷”(static corruption)。有时，我们需要考虑具有更强能力的敌手，他可以在协议运行过程中决定被攻陷的参与方；甚至，他可以中途取消对一些参与方的攻陷，转而攻陷其他参与方。这两种攻陷方式分别称为“适应性攻陷”(adaptive corruption) 和“可移动攻陷”(mobile corruption)。下面，我们分别介绍这两种攻陷方式。

##### 4.2.7.1 适应性攻陷

在适应性攻陷模型下，环境  $\mathcal{Z}$  无需在协议开始前确定被攻陷方，它可以在协议运行过程中进行攻陷。具体而言，以真实世界的参与方  $P_i$  为例，当它被攻陷前，它作为诚实方执行协议操作；在某个时刻，环境  $\mathcal{Z}$  可以指示敌手  $\mathcal{A}$  向  $P_i$  发送一条 (CORRUPT) 指令，此后  $P_i$  由  $\mathcal{A}$  所控制， $\mathcal{A}$  可以看见  $P_i$  收到的所有消息且  $P_i$  的行为纳入  $\mathcal{A}$  的行为逻辑之中。同样地，这个攻陷信息会被记录在一个特殊的纸带上，该纸带对  $\mathcal{A}$  和通信网络  $\mathcal{C}$  可见。在理想世界中，此时模拟器  $\mathcal{S}$  也可以对应地攻陷理想世界的  $P_i$ ，此后  $\mathcal{S}$  可以获得  $P_i$  的输出并代表  $P_i$  提供输入。

需要注意的是，当  $P_i$  被攻陷时， $\mathcal{A}$  是否能得到  $P_i$  的历史状态取决于模型的设置。在有“安全擦除”(secure erasure) 的设置下， $P_i$  可以在被攻陷时“安全擦除”历史数据，从而避免敌手  $\mathcal{A}$  得到其历史信息。在没有“安全擦除”的设置下， $\mathcal{A}$  在攻陷  $P_i$  时将看到其完整的历史状态。

#### 4.2.7.2 可移动攻陷与主动安全

上面所讲的“静态攻陷”和“适应性攻陷”都是“永久的”攻陷，也就是说，当敌手攻陷某个参与方之后，该参与方就一直处于被攻陷状态直到协议结束。还有一种攻陷方式，称为“可移动攻陷”(mobile corruption)，它允许参与方从被攻陷状态恢复并重新获得安全性。能够抵御“可移动攻陷”的安全性通常被称为主动安全(proactive security)。

在这个模型下，某个被攻陷方  $P_i$  可以收到来自敌手  $\mathcal{A}$  的一条 (RECOVER) 指令。此后， $P_i$  停止向  $\mathcal{A}$  泄漏他接收到的消息，也停止遵循  $\mathcal{A}$  的指令。同样，这个恢复信息会被记录在一个特殊的纸带上，该纸带对  $\mathcal{A}$  和通信网络  $\mathcal{C}$  可见。在理想世界中，此时模拟器  $\mathcal{S}$  也相应地停止对理想世界中  $P_i$  的控制，此后  $\mathcal{S}$  不再可以获得  $P_i$  的输出并代表  $P_i$  提供输入。

注意，一个具有“主动安全”的协议，必须具有某种恢复机制。因为当某个被攻陷参与方  $P_i$  收到 (RECOVER) 指令时，它的内部状态为空，也就是说， $P_i$  对此时协议处于哪个阶段等信息一无所知。因此，这个时候  $P_i$  必须通过协议的某种恢复机制得到关于协议的信息，重建自己的内部状态，然后作为诚实方继续运行协议。

# Chapter 5

## 茫然传输和茫然传输扩展

上一章，我们详细介绍了安全多方计算的形式化模型——通用可组合框架。接下来，我们将在通用可组合框架下给出协议的安全性定义和安全证明。我们首先从茫然传输协议 (Oblivious Transfer, OT) 开始。OT 本身是一个非常重要的密码学原语，在过去几十年里受到了研究者的深入研究，至今仍是一个重要的研究方向。我们首先介绍关于 OT 的一些已知结论。然后，我们介绍两个基础的 OT 构造，它们分别可以抵御半诚实敌手和恶意敌手；这两个构造比较简单，读者可以通过它们的安全性证明来进一步理解通用可组合安全框架。最后，我们介绍 IKNP 提出的茫然传输扩展协议<sup>[11]</sup>。

### 5.1 关于 OT 的一些结论

我们先简单回顾一下 OT 的定义（见第2.3.7节）。OT 有两个参与方：发送方和接收方，发送方的输入是  $x_0, x_1$ ，接收方的输入是  $b$ 。协议结束后，接收方得到  $x_b$ ，发送方和接收方都不能得到任何额外信息。

本节介绍一些关于 OT 的已知结论。我们首先证明：不存在信息论安全的两方 OT 协议。接着，我们将介绍 OT 的一些变种，并给出一些相关的结论。

#### 5.1.1 不存在信息论安全的两方 OT 协议

我们分两步证明这个结论：首先，我们通过两方 OT 协议构造一个两方 AND 协议；然后，我们证明不存在信息论安全的两方 AND 协议。因此，不存在信息论安全的两方 OT 协议。

**定理 5.1.** 不存在信息论安全的两方 OT 协议。

证明. 首先，我们通过两方 OT 协议构造一个两方 AND 协议。

在两方 OT 协议中，发送方  $P_s$  的输入为  $x_0, x_1$ ，接收方  $P_r$  的输入为  $b \in \{0, 1\}$ 。接收方  $P_r$  得到的输出为  $x_b$ ，发送方不得到输出。注意到，当  $x_0, x_1 \in \{0, 1\}$  时，有  $x_b = (1 \oplus b)x_0 \oplus bx_1$ 。

在两方 AND 协议中，设两个参与方的输入分别为  $a$  和  $b$ ，他们希望计算  $a \wedge b$ ，而不泄漏其他信息。我们可以用如下方式构造两方 AND 协议：两个参与方执行 OT 协议，令发送方  $P_s$  的输入为  $(x_0 = 0, x_1 = a)$ ，接收方的输入为  $b$ 。根据  $x_b = (1 \oplus b)x_0 \oplus bx_1$ ，协议的输出为  $ab$ 。当发送方被攻陷时，根据 OT 协议中接收方的隐私性，发送方不知道  $b$  的值。当接收方被攻陷时，如果接收方的输入

$b = 0$ , 根据 OT 协议中发送方的隐私性, 那么接收方不知道  $x_1$  即  $a$  的值。因此这是一个安全的两方 AND 协议。

下面, 我们证明不存在信息论安全的两方 AND 协议。

**引理 1.** 不存在信息论安全的两方 AND 协议。

证明. 我们先证明较为简单的情形, 即不存在两方 AND 协议满足完美隐私性和完美正确性。然后我们可以把证明过程推广到隐私性和正确性有可忽略的失败概率的情形。

假设存在两方 AND 协议  $\Pi_{2\text{AND}}$  满足完美隐私性和完美正确性。设  $\Pi_{2\text{AND}}$  有  $s$  轮通信, 其中  $s \geq 1$ . 将参与方计作  $P_0$  和  $P_1$ , 他们的输入计作  $b_0$  和  $b_1$ .  $\Pi_{2\text{AND}}$  需要满足完美的正确性, 即协议总是输出  $y = b_0 \wedge b_1$ ;  $\Pi_{2\text{AND}}$  还需要满足完美的隐私性, 也就是说, 当  $P_i$  被攻陷时, 他的视图  $\text{view}_i$  只依赖于  $b_i$  和  $y$ , 而与另一方的输入无关。

我们用  $\mathcal{T}(c, d)$  表示当  $b_0 = c, b_1 = d$  时,  $\Pi_{2\text{AND}}$  执行过程中可能产生的消息集合。即, 如果存在随机数  $r_0, r_1 \in \{0, 1\}^*$ , 使得消息集合  $\mathcal{T} = (m_{01}, m_{11}, \dots, m_{0s}, m_{1s}, y)$  是参与方  $P_0$  和  $P_1$  以  $c$  和  $d$  以及随机数  $r_0$  和  $r_1$  作为输入执行  $\Pi_{2\text{AND}}$  所产生的消息集合, 那么  $\mathcal{T} \in \mathcal{T}(c, d)$ . 接下来我们证明三个断言。

**断言 1.**  $\mathcal{T}(0, 0) = \mathcal{T}(0, 1)$ .

证明. 由于我们考虑的是信息论安全, 敌手具有无限的计算能力, 他可以遍历随机空间来计算  $\mathcal{T}(0, 0)$  和  $\mathcal{T}(0, 1)$ . 假设存在  $\mathcal{T}'$  满足  $\mathcal{T}' \in \mathcal{T}(0, 0)$  且  $\mathcal{T}' \notin \mathcal{T}(0, 1)$ . 那么当两个参与方以  $b_0 = 0, b_1 = 0$  执行协议时,  $P_0$  就有可能看到消息集合  $\mathcal{T}'$ . 如果  $P_0$  是被攻陷的, 他就可以判断  $\mathcal{T}'$  不是  $\mathcal{T}(0, 1)$  中的元素, 从而断定  $b_1 = 0$ . 这就违背了  $\Pi_{2\text{AND}}$  的完美隐私性。故该断言成立。  $\square$

**断言 2.**  $\mathcal{T}(0, 0) = \mathcal{T}(1, 0)$ .

证明. 这与断言 1 完全同理。  $\square$

**断言 3.**  $\mathcal{T}(0, 1) \cap \mathcal{T}(1, 0) \subset \mathcal{T}(1, 1)$ .

证明. 这个断言的证明基于  $\mathcal{T}$  的定义。令  $\mathcal{T} = (m_{01}, m_{11}, \dots, m_{0s}, m_{1s}, y)$ . 如果  $\mathcal{T} \in \mathcal{T}(0, 1) \cap \mathcal{T}(1, 0)$ , 首先, 根据  $\mathcal{T} \in \mathcal{T}(0, 1)$  可知, 存在随机数  $r_1$  使得  $P_1$  以  $b_1 = 1$  和  $r_1$  作为输入运行协议且收到的消息是  $\{m_{0i}\}_{i \in [s]}$  时, 会得到  $\mathcal{T}$ ; 又由  $\mathcal{T} \in \mathcal{T}(1, 0)$  可知, 存在随机数  $r_0$  使得  $P_0$  以  $b_0 = 1$  和  $r_0$  作为输入运行协议且收到的消息是  $\{m_{1i}\}_{i \in [s]}$  时, 会得到  $\mathcal{T}$ . 那么, 当  $P_0$  和  $P_1$  以  $b_0 = 1, b_1 = 1$  以及  $r_0, r_1$  作为输入运行协议时, 也会得到  $\mathcal{T}$ , 即  $\mathcal{T} \in \mathcal{T}(1, 1)$ . 因此  $\mathcal{T}(0, 1) \cap \mathcal{T}(1, 0) \subset \mathcal{T}(1, 1)$ .  $\square$

基于前两个断言, 我们有

$$\mathcal{T}(0, 1) \cap \mathcal{T}(1, 0) = \mathcal{T}(0, 0)$$

而根据断言 3,  $\mathcal{T}(0, 1) \cap \mathcal{T}(1, 0) \subset \mathcal{T}(1, 1)$ , 因此  $\mathcal{T}(0, 0) \subset \mathcal{T}(1, 1)$ . 而当输入为  $b_0 = 0, b_1 = 0$  时, 输出  $y = 0$ , 当输入为  $b_0 = 1, b_1 = 1$  时, 输出  $y = 1$ . 因此  $\mathcal{T}(0, 0)$  不可能包含于  $\mathcal{T}(1, 1)$ . 矛盾。故不存在两方 AND 协议满足完美隐私性和完美正确性。

如果  $\Pi_{2\text{AND}}$  的隐私性和正确性有可忽略的失败概率, 那么在断言 1 和断言 2 中,  $\mathcal{T}(0, 0)$  和  $\mathcal{T}(0, 1)$  以及  $\mathcal{T}(0, 0)$  和  $\mathcal{T}(1, 0)$  都有可忽略的统计距离。断言 3 仍然成立。依然可以推出矛盾。因此, 不存在信息论安全的两方 AND 协议。

$\square$

综上，我们可以通过两方 OT 协议构造两方 AND 协议。而引理 1 证明了不存在信息论安全的两方 AND 协议。因此，定理得证。

□

注意，这里我们证明的是，不存在对发送方和接收方均满足信息论安全的 OT 协议。但是，构造一个当接收方被攻陷时满足信息论安全，当发送方被攻陷时满足计算安全的 OT 协议，或者当发送方被攻陷时满足信息论安全，当接收方被攻陷时满足计算安全的 OT 协议，都是已经被实现的。

### 5.1.2 OT 变种

本节，我们介绍几个有名的 OT 变种，以及它们之间的关系。

**Rabin OT<sup>[12]</sup>**. 在 Rabin 的 OT 变种中，发送方  $P_s$  只有一个隐私输入  $x$ ，接收方有  $1/2$  概率接收到  $x$ ，有  $1/2$  概率接收到  $\perp$ 。发送方不知道接收方的输出。

**随机 OT(random OT)**. 在随机 OT 中，发送方和接收方都没有任何输入。协议会给发送方输出  $(x_0, x_1)$  并给接收方输出  $(b, x_b)$ ，其中  $x_0, x_1$  是随机均匀选取的消息， $b$  是随机均匀选取的比特。

**OT 变种的等价性**. 值得注意的是，OT（第2.3.7节所定义的常见形式）、Rabin OT 和随机 OT 都是等价的。我们下面简要说明这一点。

我们首先说明 OT 和随机 OT 是等价的，即通过 OT 可以构造随机 OT，通过随机 OT 也可以构造 OT. 前者非常简单。发送方均匀随机选取  $x_0, x_1$ ，接收方均匀随机选取  $b \leftarrow_{\$} \{0, 1\}$ ，然后以它们作为输入执行 OT 协议，接收方得到  $x_b$ 。最后，发送方的输出是  $(x_0, x_1)$ ，接收方的输出是  $(b, x_b)$ 。容易证明这是一个安全的随机 OT 协议。

通过随机 OT 构造 OT 的方法如下。发送方和接收方首先执行随机 OT 协议，发送方获得  $(x_0, x_1)$ ，接收方获得  $(b, x_b)$ 。我们将发送方在 OT 协议中的输入记作  $(m_0, m_1)$ ，接收方在 OT 协议中的输入记作  $c$ 。现在接收方需要获得  $m_c$ 。这里的  $x_0, x_1$  都是均匀随机的，可以作为一次一密的密钥，我们的思路是用  $x_b$  加密  $m_c$ ，用  $x_{b \oplus 1}$  加密  $m_{c \oplus 1}$  发送给接收方。

然而发送方  $P_s$  不知道  $b$  与  $c$  的关系：如果  $b = c$ ，那么应该用  $x_0$  加密  $m_0$ ，用  $x_1$  加密  $m_1$ ；如果  $b = c \oplus 1$ ，那么应该用  $x_1$  加密  $m_0$ ，用  $x_0$  加密  $m_1$ 。为此，接收方  $P_r$  发送  $d = b \oplus c$  给  $P_s$ （告知  $b$  与  $c$  是否相等）。根据  $d$  的值， $P_s$  接下来计算  $(y_0, y_1) = (x_d \oplus m_0, x_{d \oplus 1} \oplus m_1)$  并发送给  $P_r$ 。最后，接收方  $P_r$  输出  $m'_c = y_c \oplus x_b$ 。

我们简单分析一下这个协议的正确性和隐私性。首先，正确性： $m'_c = y_c \oplus x_b = (x_{d \oplus c} \oplus m_c) \oplus x_b = (x_b \oplus m_c) \oplus x_b = m_c$ 。而  $x_{b \oplus 1}$  作为一次一密的密钥加密了  $m_{c \oplus 1}$ ，保护了发送方的隐私； $b$  作为一次一密的密钥加密了选择比特  $c$ ，保护了接收方的隐私。

所以，我们通过随机 OT 构造了 OT 协议。故随机 OT 和 OT 是等价的。

关于 Rabin OT，Crépeau<sup>[13]</sup>证明了它与 OT 也是等价的。因此，OT、Rabin OT、随机 OT 都是等价的。

**1-out-of-N-OT**. 在这种 OT 中， $P_s$  有  $N$  个隐私输入  $x_0, \dots, x_{N-1}$ ， $P_r$  有一个隐私索引  $b \in \{0, \dots, N-1\}$ 。 $P_r$  的输出是  $x_b$ 。发送方  $P_s$  不知道  $b$ ，接收方  $P_r$  不知道  $x_b$  以外的其他隐私输入。

**k-out-of-N-OT**. 在这种 OT 中， $P_s$  有  $N$  个隐私输入  $x_0, \dots, x_{N-1}$ ， $P_r$  有  $k$  个隐私索引  $i_0, \dots, i_{k-1} \in \{0, \dots, N-1\}$ 。 $P_r$  的输出是  $x_{i_0}, \dots, x_{i_k}$ 。发送方  $P_s$  不知道隐私索引  $i_0, \dots, i_{k-1}$ ，接收方  $P_r$  不知道  $x_{i_0}, \dots, x_{i_k}$  以外的其他隐私输入。

Naor 和 Pinkas<sup>[14]</sup>证明了：可以通过 1-out-of-2-OT 构造 1-out-of- $N$ -OT 和  $k$ -out-of- $N$ -OT。例如 GMW 协议中（见第6.2节），需要调用 1-out-of-4-OT 来计算乘法门。有了这个结论，我们就可以聚焦于构造 1-out-of-2-OT 了。

### 5.1.3 一些其他结论

本节，我们列出关于 OT 的一些其他重要结论以及相关文章，可以作为读者拓展阅读的材料。

- OT 是完备的，即基于 OT 可以安全地计算任何函数<sup>[15-16]</sup>。
- OT 是对称的，即基于 OT 可以构造 OT'，其中 OT' 的发送方是 OT 的接收方，OT' 的接收方是 OT 的发送方<sup>[17]</sup>。
- 不能以黑盒的方式基于公钥加密 (Public Key Encryption, PKE) 构造 OT<sup>[18]</sup>。
- OT 可以基于增强陷门置换 (enhanced trapdoor permutation)<sup>[19-20]</sup>、DDH 假设、RSA 假设<sup>[19]</sup>、格密码<sup>[21]</sup>构造。

## 5.2 基于 DDH 假设的 OT 协议（半诚实安全）

本小节，我们介绍一个在半诚实敌手模型下，基于 DDH 假设的 OT 协议。

### 5.2.1 协议描述

这个协议的基本思想很简单。接收方  $P_r$  生成两个公钥  $\text{pk}_0, \text{pk}_1$ ，且  $P_r$  知道  $\text{pk}_b$  对应的私钥，不知道  $\text{pk}_{1-b}$  对应的私钥（具体的做法是， $P_r$  生成一个公私钥对  $(\text{pk}_b, \text{sk}_b)$ ，然后在群上随机选择一个元素作为  $\text{pk}_{1-b}$ ）。然后，接收方  $P_r$  将  $\text{pk}_0, \text{pk}_1$  发给发送方  $P_s$ 。随后  $P_s$  用  $\text{pk}_0, \text{pk}_1$  通过 ElGamal 加密算法分别加密  $x_0, x_1$ ，并将密文发送给  $P_r$ 。因为接收方只知道一个私钥，因此只能解密一个密文。而发送方只能看到两个公钥，无法知道接收方拥有的是哪一个私钥。

该协议的完整描述如图 5.1 所示。

### 5.2.2 安全性的直观说明

我们首先在半诚实敌手模型下给出一个安全性的直观说明。

**正确性。**在半诚实敌手模型下，参与方均遵循协议。我们有

$$\frac{c_b}{v^{\text{sk}}} = \frac{c_b}{(g^r)^{\text{sk}}} = \frac{c_b}{(g^{\text{sk}})^r} = \frac{c_b}{\text{pk}_b^r} = \frac{x_b \cdot \text{pk}_b^r}{\text{pk}_b^r} = x_b$$

因此，接收方  $P_r$  的输出是  $x_b$ 。

**隐私性。**首先，我们说明  $P_s$  无法得到关于  $P_r$  的输入  $b$  的任何信息。在协议中， $(\text{pk}_0, \text{pk}_1)$  是两个随机的群元素，它的分布与  $b$  的值无关。因此， $P_s$  无法得到关于  $P_r$  的输入  $b$  的任何信息。

然后，我们说明  $P_r$  可以得到  $x_b$ ，但是不能得到关于  $x_{1-b}$  的任何信息。这需要假设 DDH 问题在群  $\mathbb{G}$  上是困难的。在协议中， $P_r$  知道的信息包括自己的输入  $b$ ，私钥  $\text{sk}$ ，公钥  $\text{pk}_0, \text{pk}_1$ ，协议的输出  $x_b$ ，以

### 基于 DDH 假设的 OT 协议

设  $\mathbb{G}$  是一个阶为  $q$  的群 ( $q$  为素数), 生成元是  $g$ . 设发送方  $P_s$  的输入  $x_0, x_1$  是群  $\mathbb{G}$  中的元素 (或者可以编码成群  $\mathbb{G}$  中的元素)。

- 输入: 发送方  $P_s$  的输入是  $x_0, x_1 \in \mathbb{G}$ ; 接收方  $P_r$  的输入是  $b \in \{0, 1\}$ .
- 协议:

- 接收方  $P_r$  计算

$$\text{sk} \leftarrow_{\$} \mathbb{Z}_q, \quad \text{pk}_b = g^{\text{sk}}, \quad \text{pk}_{1-b} \leftarrow_{\$} \mathbb{G}$$

并将  $(\text{pk}_0, \text{pk}_1)$  发给  $P_s$ .

- 当从  $P_r$  收到  $(\text{pk}_0, \text{pk}_1)$  后, 发送方  $P_s$  计算

$$r \leftarrow_{\$} \mathbb{Z}_q, \quad v = g^r, \quad c_0 = x_0 \cdot \text{pk}_0^r, \quad c_1 = x_1 \cdot \text{pk}_1^r$$

并将  $(v, c_0, c_1)$  发给  $P_r$ .

- 当从  $P_s$  收到  $(v, c_0, c_1)$  后, 接收方  $P_r$  输出  $c_b/v^{\text{sk}}$ .

图 5.1: 基于 DDH 假设的 OT 协议

及从  $P_s$  收到的  $(v, c_0, c_1)$ . 其中,  $v$  是随机群元素, 不会泄漏任何关于  $x_{1-b}$  的信息;  $c_b = x_b v^{\text{sk}}$  可以由  $P_r$  的已知信息推算出来, 也不会泄漏  $x_{1-b}$ . 因此我们只需要论证  $c_{1-b}$  没有泄漏信息。基于 DDH 假设, 三元组  $(g^r, \text{pk}_{1-b}, \text{pk}_{1-b}^r)$  与  $(g^r, \text{pk}_{1-b}, e)$  是计算上不可区分的, 其中  $e \in \mathbb{G}$  是一个随机群元素。由于  $c_{1-b} = x_{1-b} \text{pk}_{1-b}^r$ , 区分  $(v = g^r, \text{pk}_{1-b}, c_{1-b})$  和  $(v = g^r, \text{pk}_{1-b}, e)$  等价于区分  $(v = g^r, \text{pk}_{1-b}, c_{1-b}/x_{1-b})$  和  $(v = g^r, \text{pk}_{1-b}, e/x_{1-b})$ . 前者是 DDH 三元组, 而  $e/x_{1-b}$  仍是随机群元素, 所以在计算上它们是不可区分的。因此,  $P_r$  不能得到关于  $x_{1-b}$  的任何信息。

### 5.2.3 安全性证明

现在, 我们给出形式化的安全性证明。我们先给出理想功能  $\mathcal{F}_{\text{OT}}$  的定义, 如图 5.2 所示。

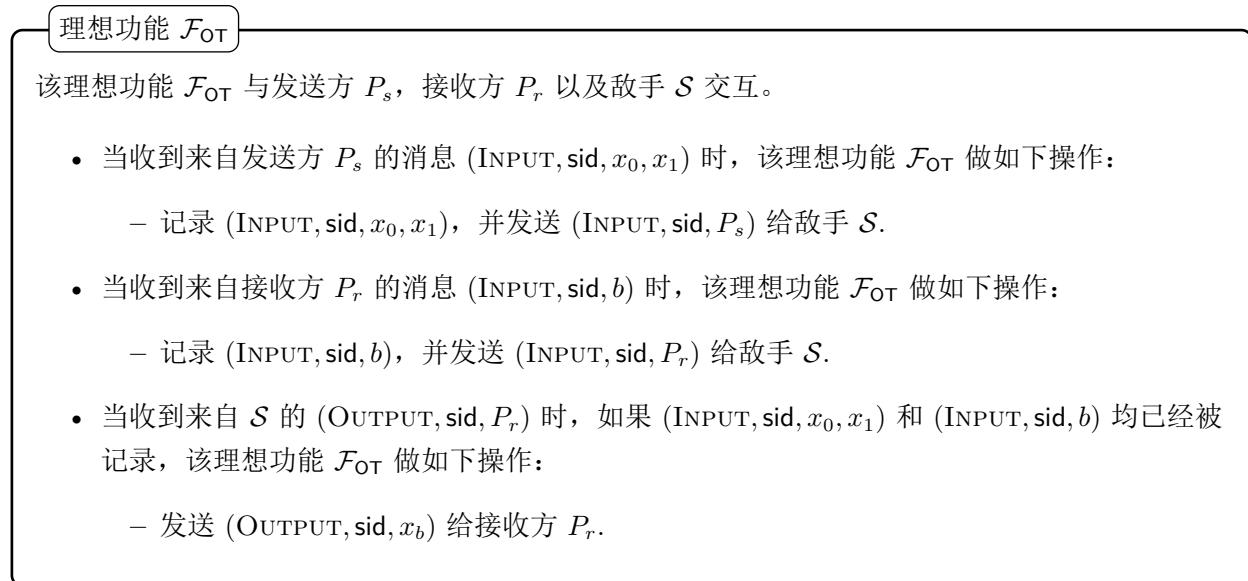
**定理 5.2.** 假设 DDH 问题在群  $\mathbb{G}$  上是困难的。图 5.1 展示的协议 II 对于静态半诚实对手 UC-安全实现了理想功能  $\mathcal{F}_{\text{OT}}$  (图 5.2)。

证明. 要证明该定理, 我们需要构造模拟器  $\mathcal{S}$ , 使得对所有 PPT 的环境  $\mathcal{Z}$ , 都有

$$\text{EXEC}_{\Pi, \mathcal{A}, \mathcal{Z}} \approx \text{EXEC}_{\mathcal{F}_{\text{OT}}, \mathcal{S}, \mathcal{Z}}$$

其中  $\mathcal{A}$  是平凡敌手, 只执行  $\mathcal{Z}$  的指令 (基于定理 4.1 平凡敌手的完备性)。

**模拟器  $\mathcal{S}$ .** 首先, 模拟器  $\mathcal{S}$  在内部运行  $\mathcal{A}$ , 转发  $\mathcal{Z}$  与  $\mathcal{A}$  之间的消息。根据参与方被攻陷的不同情况, 模拟器  $\mathcal{S}$  的行为会对应地有所不同。在两方均被攻陷或两方均诚实的情况下,  $\mathcal{S}$  就不需要做什么了。因此, 我们重点考虑一个参与方被攻陷的情况。

图 5.2: 茫然传输的理想功能  $\mathcal{F}_{\text{OT}}$ 

**情况 1:** 发送方  $P_s$  被攻陷。在这种情况下,  $\mathcal{S}$  的工作方式如下。

- 当  $\mathcal{Z}$  发送输入  $(\text{INPUT}, \text{sid}, x_0, x_1)$  给  $P_s$  时, 它会被直接转发给  $\mathcal{F}_{\text{OT}}$ , 同时,  $\mathcal{S}$  也会收到  $(\text{INPUT}, \text{sid}, x_0, x_1)$ . 此后,  $\mathcal{S}$  模拟  $P_s$  遵循协议运行, 并将  $P_s$  的内部状态报告给  $\mathcal{Z}$ .
- 在某个时刻,  $\mathcal{Z}$  会给  $P_r$  发送输入  $(\text{INPUT}, \text{sid}, b)$ , 这会让真实世界的  $P_s$  收到消息  $(\text{pk}_0, \text{pk}_1)$ . 在理想世界中, 此时  $\mathcal{S}$  会从  $\mathcal{F}_{\text{OT}}$  收到  $(\text{INPUT}, \text{sid}, P_r)$  (告知  $\mathcal{S}$  “ $P_r$  已提供输入”).  $\mathcal{S}$  并不能知道  $P_r$  的输入  $b$ , 但是  $\mathcal{S}$  需要“编造”出消息  $(\text{pk}_0, \text{pk}_1)$ . 此时,  $\mathcal{S}$  随机选取  $\text{pk}_0 \leftarrow_{\$} \mathbb{G}, \text{pk}_1 \leftarrow_{\$} \mathbb{G}$ , 并把  $(\text{pk}_0, \text{pk}_1)$  报告给  $\mathcal{Z}$ .
- 当  $P_s$  向  $P_r$  发送  $(v, c_0, c_1)$  后,  $\mathcal{S}$  向  $\mathcal{F}_{\text{OT}}$  发送  $(\text{OUTPUT}, \text{sid}, P_r)$ .

以上就是  $P_s$  被攻陷时  $\mathcal{S}$  的工作方式。下面, 我们需要论证理想世界和真实世界的不可区分性。

**不可区分性。**我们观察到, 理想世界与真实世界唯一的不同, 就是  $(\text{pk}_0, \text{pk}_1)$  的生成方式。然而, 无论  $P_r$  的输入  $b$  是 0 还是 1,  $(\text{pk}_0, \text{pk}_1)$  都是两个群上的随机元素。因此, 理想世界和真实世界是完美不可区分的。

**情况 2:** 接收方  $P_r$  被攻陷。在这种情况下,  $\mathcal{S}$  的工作方式如下。

- 当  $\mathcal{Z}$  发送输入  $(\text{INPUT}, \text{sid}, b)$  给  $P_r$  时, 它会被直接转发给  $\mathcal{F}_{\text{OT}}$ , 同时,  $\mathcal{S}$  也将收到  $(\text{INPUT}, \text{sid}, b)$ . 此后,  $\mathcal{S}$  模拟  $P_r$  遵循协议运行, 即计算  $\text{sk} \leftarrow_{\$} \mathbb{Z}_q, \text{pk}_b = g^{\text{sk}}, \text{pk}_{1-b} \leftarrow_{\$} \mathbb{G}$ , 将  $(\text{pk}_0, \text{pk}_1)$  发给  $P_s$ .  $\mathcal{S}$  将  $P_r$  的内部状态报告给  $\mathcal{Z}$ .
- 在某个时刻,  $\mathcal{Z}$  会给  $P_s$  发送输入  $(\text{INPUT}, \text{sid}, x_0, x_1)$ , 如果  $P_r$  已经发送了第一条消息  $(\text{pk}_0, \text{pk}_1)$  给  $P_s$ , 那么这会让真实世界的  $P_r$  收到消息  $(v, c_0, c_1)$ . 在理想世界中, 这对应于  $\mathcal{S}$  从  $\mathcal{F}_{\text{OT}}$  收到  $(\text{INPUT}, \text{sid}, P_s)$  (告知  $\mathcal{S}$  “ $P_s$  已提供输入”).  $\mathcal{S}$  并不能知道  $P_s$  的输入  $x_0, x_1$ , 但是  $\mathcal{S}$  需要“编造”出消息  $(\text{pk}_0, \text{pk}_1)$  给  $P_s$ .

造”出消息  $(v, c_0, c_1)$ . 此时,  $\mathcal{S}$  发送  $(\text{OUTPUT}, \text{sid}, P_r)$  给  $\mathcal{F}_{\text{OT}}$  使被攻陷的接收方  $P_r$  得到其在理想世界中的输出  $x_b$ . 然后,  $\mathcal{S}$  计算  $v \leftarrow_{\$} \mathbb{G}, c_b = x_b v^{\text{sk}}, c_{1-b} \leftarrow_{\$} \mathbb{G}$ , 并将  $(v, c_0, c_1)$  报告给  $\mathcal{Z}$ .

以上就是  $P_r$  被攻陷时  $\mathcal{S}$  的工作方式。下面, 我们需要论证理想世界和真实世界的不可区分性。

**不可区分性。**观察到, 理想世界与真实世界唯一的不同, 就是  $(v, c_0, c_1)$  的生成方式。然而, 不论在理想世界还是真实世界中,  $v$  都是随机群元素,  $c_b$  都被  $P_r$  已知的信息决定。在真实世界中,  $c_{1-b} = x_{1-b} \cdot \text{pk}_{1-b}^r$ , 在理想世界中,  $c_{1-b}$  是随机群元素, 正如我们在第5.2.2小节中的论证, 基于 DDH 假设, 它们计算上是不可区分的。因此, 理想世界和真实世界是计算上不可区分的。

□

**思考题:** 之前我们提到, 不能以黑盒的方式基于公钥加密 (PKE) 构造 OT. 但是这里我们似乎只使用了 ElGamal 公钥加密方案就构造了 OT 协议。这是否矛盾了呢?

**答案:** 上面的构造中暗含了一个假设: 接收方可以在不知道私钥的情况下选取一个公钥。而公钥加密并不保证这个假设成立。因此不矛盾。

## 5.3 三方 OT 协议（恶意安全）

上一节介绍的协议是对半诚实敌手安全的 OT 协议, 这个协议在面对恶意敌手时显然是不安全的。如果接收方  $P_r$  是恶意的, 他可以生成两个公私钥对, 然后将这两个公钥发给  $P_s$ . 这样,  $P_r$  就可以解密两个密文, 打破发送方的隐私性。

一个抵抗该攻击的简单想法是: 在协议的第一步, 发送方  $P_s$  随机选取  $d \leftarrow_{\$} \mathbb{G}$  并发送给  $P_r$ , 接收方的公钥生成方式改为  $\text{sk} \leftarrow_{\$} \mathbb{Z}_q, \text{pk}_b = g^{\text{sk}}, \text{pk}_{1-b} \leftarrow d/\text{pk}_b$ . 当  $P_s$  收到  $\text{pk}_0, \text{pk}_1$  后, 检查条件  $\text{pk}_0 \cdot \text{pk}_1 = d$  是否满足, 若不满足则中止协议。这样,  $P_r$  不可能同时知道  $\text{pk}_0$  和  $\text{pk}_1$  对应的私钥, 因为这等价于求解  $d$  的离散对数。

然而, 仅仅作上述修改还不足以构造出一个恶意安全的 OT 协议: “敌手不知道公钥对应的私钥”并不能证明“敌手不知道关于消息的任何信息”。本节, 我们将给出一个三方 OT 协议, 并证明: 在只有一个参与方被攻陷的情况下, 该协议是恶意安全的。

### 5.3.1 协议描述

在该协议中, 发送方  $P_s$  的输入是  $x_0, x_1$ , 接收方  $P_r$  的输入是  $b \in \{0, 1\}$ , 还有一个额外的“协助者”  $P_h$ , 他没有任何输入。协议结束时, 接收方  $P_r$  获得  $x_b$ , 而发送方  $P_s$  和“协助者”  $P_h$  不获得输出。

我们使用抗碰撞哈希函数  $H : \mathcal{M} \times \mathcal{R} \rightarrow \mathcal{C}$  作为非交互承诺方案。其中,  $\mathcal{M}$  是消息空间,  $\mathcal{R}$  是(足够大的)随机元素的空间,  $\mathcal{C}$  是输出的空间(也就是承诺的空间)。这里随机元素的作用是隐藏输入消息, 即对任意选定的  $x_0, x_1 \in \mathcal{M}$  和随机选择的  $r \in \mathcal{R}$ ,  $H(x_0, r)$  和  $H(x_1, r)$  是计算上不可区分的。

该协议的完整描述如图 5.3 所示。

**协议的直观描述。**直观上来说, 该协议的思想如下:

- 协助者生成两个均匀随机的消息  $p_0, p_1$ , 作为“一次一密”的密钥来加密发送方的消息。然后, 协助者生成对  $p_0, p_1$  的承诺, 将两个承诺的打开都发给发送方; 对于接收方, 协助者向他发送  $p_\beta$  的打开和  $p_{\beta \oplus 1}$  的承诺, 这里  $\beta$  是协助者均匀随机选择的比特。

### 三方 OT 协议

- 准备步骤: 抗碰撞哈希函数  $H : \mathcal{M} \times \mathcal{R} \rightarrow \mathcal{C}$ .
- 输入:  $P_s$  的输入是  $x_0, x_1 \in \mathcal{M}$ ;  $P_r$  的输入是  $b \in \{0, 1\}$ .
- 协议:
  - $P_h$  计算

$$p_0, p_1 \leftarrow_{\$} \mathcal{M}, \quad r_0, r_1 \leftarrow_{\$} \mathcal{R}, \quad \beta \leftarrow_{\$} \{0, 1\}, \quad c_{\beta \oplus 1} = H(p_{\beta \oplus 1}, r_{\beta \oplus 1})$$

并把  $(\beta, p_\beta, r_\beta, c_{\beta \oplus 1})$  发给  $P_r$ , 把  $(p_0, r_0, p_1, r_1)$  发给  $P_s$ .

- $P_r$  从  $P_h$  处收到  $(\beta, p_\beta, r_\beta, c_{\beta \oplus 1})$  后, 计算

$$\tau = b \oplus \beta, \quad c_\beta = H(p_\beta, r_\beta)$$

并把  $(\tau, c_0, c_1)$  发给  $P_s$ .

- $P_s$  从  $P_h$  处收到  $(p_0, r_0, p_1, r_1)$ , 从  $P_r$  处收到  $(\tau, c_0, c_1)$  后, 首先检查

$$c_0 \stackrel{?}{=} H(p_0, r_0), \quad c_1 \stackrel{?}{=} H(p_1, r_1)$$

如果检查不通过, 则中止协议 (同时通知  $P_r$  中止协议)。

否则,  $P_s$  计算

$$e_0 = x_0 \oplus p_\tau, \quad e_1 = x_1 \oplus p_{\tau \oplus 1}$$

并把  $(e_0, e_1)$  发给  $P_r$ .

- $P_r$  从  $P_s$  处收到  $(e_0, e_1)$  后, 输出  $e_b \oplus p_\beta$ .

图 5.3: 三方 OT 协议

- 接着, 接收方将  $\beta$  作为“一次一密”的密钥加密他的选择比特  $b$  得到  $\tau$ , 然后将  $\tau$  和对  $p_0$  和  $p_1$  的两个承诺发给发送方。
- 发送方首先检查: 从协助者处收到的承诺的打开与从接收方处收到的承诺是否一致。这一步防止了恶意的协助者给发送方和接收方发送不一致的信息。如果检查通过, 发送方用  $p_\tau$  加密  $x_0$  得到  $e_0$ , 用  $p_{\tau \oplus 1}$  加密  $x_1$  得到  $e_1$ , 并将  $e_0, e_1$  发给接收方。
- 最后, 接收方用  $p_\beta$  解密  $e_b$  得到  $x_b$ .

### 5.3.2 安全性的直观说明

在给出形式化的安全性证明之前, 我们先从直观上分析该协议的正确性、隐私性和输入独立性。输入独立性指的是: 所有参与方的输入必须是独立于其他参与方的输入的 (半诚实情况下敌手不能更改协议输入, 故无需分析输入独立性)。协议假设最多只有一个参与方被攻陷, 因此我们分别对“协助者”

被攻陷”、“接收方被攻陷”和“发送方被攻陷”三种情况进行分析。

**协助者被攻陷：**由于哈希函数  $H$  的抗碰撞性，当被攻陷的协助者给发送方和接收方发送不一致的消息时，协议会中止。而如果协助者发送了一致的消息，接收方就能得到正确的消息  $x_b$ 。因此正确性成立。由于发送方和接收方之间的通信是通过安全信道进行的，协助者无法得到任何关于  $x_0, x_1$  和  $b$  的信息。因此隐私性成立。因为协助者没有输入，输入独立性平凡地成立。

**接收方被攻陷：**当被攻陷的接收方将  $\tau$  发给发送方时，它的选择比特  $b$  就由关系  $b = \tau \oplus \beta$  所决定了，其中  $\beta$  是诚实的协助者选择的比特。此时，接收方没有收到任何与发送方的输入  $x_0, x_1$  有关的消息，因此输入独立性成立。接下来分析当接收方从发送方处收到  $e_0, e_1$  时的情况。此时，接收方可以由  $e_b$  解密得到  $x_b$ 。根据承诺方案的隐藏性，接收方不能从  $c_{\beta \oplus 1}$  中得到任何关于  $p_{\beta \oplus 1}$  的信息。而  $p_{\beta \oplus 1}$  作为“一次一密”的密钥加密了  $x_{b \oplus 1}$  得到  $e_{b \oplus 1}$ ，因此  $e_{b \oplus 1}$  没有泄漏任何关于  $x_{b \oplus 1}$  的信息，隐私性成立。最后，由于诚实的协助者和发送方没有输出，正确性平凡地成立。

**发送方被攻陷：**当发送方将  $e_0, e_1$  发给接收方时，他的输入  $x_0, x_1$  由关系  $x_0 = e_0 \oplus p_\tau$  和  $x_1 = e_1 \oplus p_{\tau \oplus 1}$  决定。此时，接收方的输入比特  $b$  被  $\beta$  完美地保护了，因此发送方不知道关于  $b$  的任何信息，输入独立性成立。在之后的协议执行过程中，发送方没有获得任何与  $b$  有关的消息，因此隐私性也成立。最后，接收方可以正确地解密  $e_b$  得到  $x_b$ ，因此正确性成立。

### 5.3.3 安全性证明

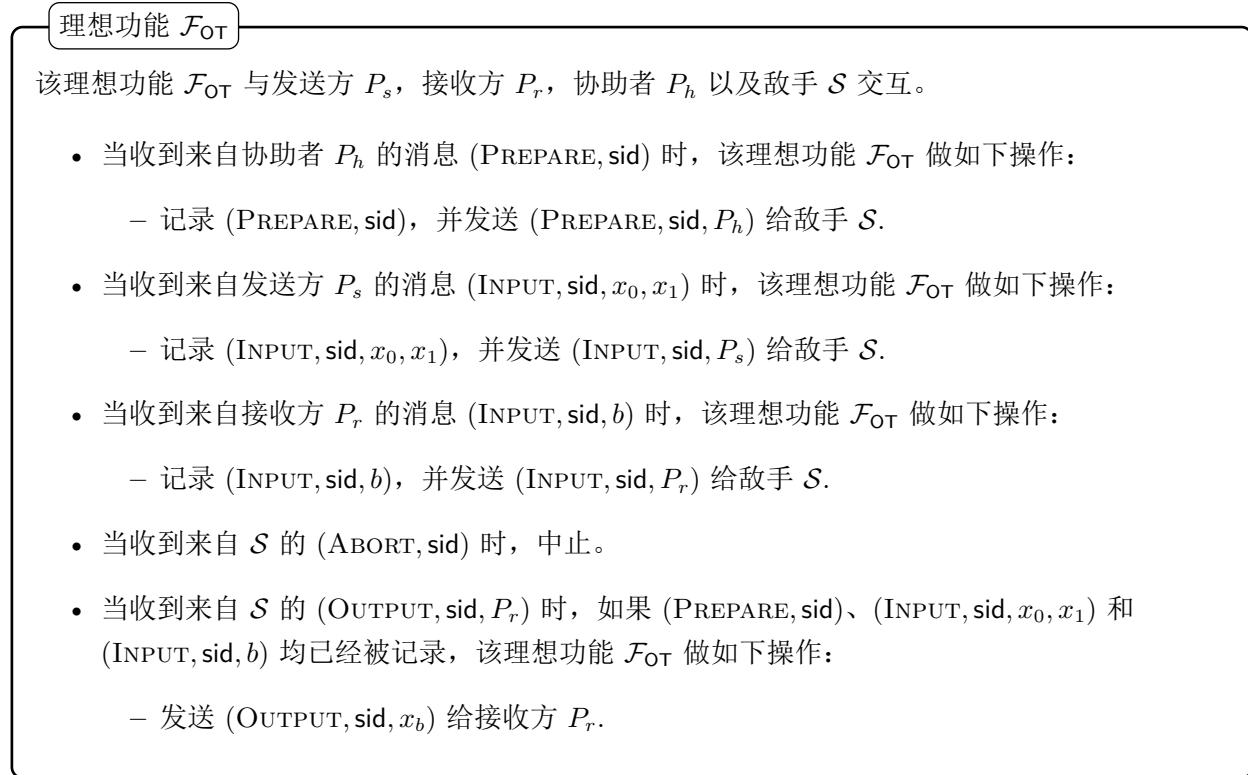
由于这里加入了一个角色——协助者，并且考虑的是恶意安全，我们需要对理想功能  $\mathcal{F}_{OT}$  稍作修改：当协助者执行协议时，发送一个通知消息给  $\mathcal{S}$ ； $\mathcal{S}$  可以向  $\mathcal{F}_{OT}$  发送一个中止指令，使其结束运行，不产生输出。修改后的理想功能  $\mathcal{F}_{OT}$  如图 5.4 所示。

**定理 5.3.** 假设参与方之间存在点对点安全信道；假设哈希函数  $H$  是抗碰撞的，且满足隐藏性；假设最多只有一个参与方被攻陷。图 5.3 展示的协议 II 对于静态恶意对手 UC-安全实现了理想功能  $\mathcal{F}_{OT}$ （图 5.4）。

**真实世界。**首先，让我们回顾一下真实世界的细节。环境  $\mathcal{Z}$  在最开始决定哪些参与方是被攻陷的，之后，它向诚实方提供输入并从诚实方接收输出。（在这个协议中，只有  $P_s$  和  $P_r$  可以接收输入，只有  $P_r$  可以产生输出。）通信网络由  $\mathcal{Z}$  控制（通过平凡敌手  $\mathcal{A}$  向  $\mathcal{C}$ -控制线发送指令），因此它可以决定何时将协议消息发送给诚实方。注意，我们可以认为被攻陷方并不真实存在，它们发送给诚实方的任何消息实际上是由  $\mathcal{Z}$  发送的（通过  $\mathcal{A}$  间接地实现），而任何诚实方发送给被攻陷方的消息实际上是发送给  $\mathcal{Z}$  的（通过  $\mathcal{A}$  间接地实现）。尽管如此，为了描述的明确性，在下面的证明中我们仍然会说“从被攻陷方发送给诚实方的消息”和“从诚实方发送给被攻陷方的消息”。

**理想世界。**理想世界中，有一个模拟器  $\mathcal{S}$ 。与真实世界一样， $\mathcal{Z}$  在最开始决定哪些方是被攻陷的，之后，它向诚实方提供输入并从诚实方接收输出。诚实方的输入直接转发给理想功能，被攻陷方的输入由  $\mathcal{S}$  提供给理想功能。当理想功能收到了所有参与方的输入之后， $\mathcal{S}$  可以选择何时让理想功能生成输出：诚实方的输出直接转发给  $\mathcal{Z}$ ，而被攻陷方的输出则交给  $\mathcal{S}$ 。

**证明的总体方法。**为了证明协议 II 的安全性，我们需要设计  $\mathcal{S}$ ，使得所有 PPT 的良好的  $\mathcal{Z}$  都无法区分真实世界和理想世界。直观地说，这意味着  $\mathcal{S}$  在理想世界中与  $\mathcal{Z}$  的交互应该能使  $\mathcal{Z}$  “认为”它仍然在真实世界中。为此， $\mathcal{S}$  需要与  $\mathcal{Z}$  交换消息（通过  $\mathcal{S}$ -控制线），这些消息看起来就像真实世界中的通信网络上的消息。因此，即使在理想世界中，我们也可以“从被攻陷方发送给诚实方的消息”（其

图 5.4: 茫然传输的理想功能  $\mathcal{F}_{\text{OT}}$ 

内容实际上是由  $\mathcal{Z}$  发送给  $\mathcal{S}$  的) 以及“从诚实方发送给被攻陷方的消息”(其内容实际上是由  $\mathcal{S}$  生成并发送给  $\mathcal{Z}$  的)。对于“诚实方之间发送的消息”, 其内容实际上从未生成:  $\mathcal{S}$  只需要与  $\mathcal{Z}$  交换消息, 所以  $\mathcal{S}$  无需模拟诚实方之间的消息。

每当真实世界中的诚实方向真实世界中的被攻陷方发送协议消息时,  $\mathcal{S}$  必须以某种方式“编造”出一个协议消息发送给该被攻陷方(实际上发送给  $\mathcal{Z}$ )。此外, 根据真实世界的被攻陷方发送给真实世界的诚实方的消息,  $\mathcal{S}$  必须从这些消息中提取出被攻陷方的有效输入, 并将其提交给理想功能。注意,  $\mathcal{S}$  需要在不知道诚实方的输入输出的情况下完成这些模拟(但它最终可以知道被攻陷方的输出)。并且, 这些模拟必须使得  $\mathcal{Z}$  无法高效地区分理想世界和真实世界( $\mathcal{Z}$  知道所有诚实方的输入和输出)。

基于上述讨论我们可以发现, 恶意敌手模型和半诚实敌手模型的一个重要技术区别是: 在恶意敌手模型中, 模拟器必须从被攻陷方的协议消息中“提取”有效输入; 而在半诚实模型中, 模拟器可以直接得到这些输入。

下面, 我们给出形式化的证明。

证明. 根据平凡敌手的完备性, 我们需要构造模拟器  $\mathcal{S}$ , 使得对所有 PPT 的环境  $\mathcal{Z}$ , 都有

$$\text{EXEC}_{\Pi, \mathcal{A}, \mathcal{Z}} \approx \text{EXEC}_{\mathcal{F}_{\text{OT}}, \mathcal{S}, \mathcal{Z}}$$

其中  $\mathcal{A}$  是平凡敌手, 只执行  $\mathcal{Z}$  的指令。

模拟器  $\mathcal{S}$ . 首先, 模拟器  $\mathcal{S}$  在内部运行  $\mathcal{A}$ , 转发  $\mathcal{Z}$  与  $\mathcal{A}$  之间的消息。根据参与方被攻陷的不同情况, 模拟器  $\mathcal{S}$  的行为会对应地有所不同。在三方均诚实的情况下,  $\mathcal{S}$  就不需要做什么了。因此, 我们重点

考虑一个参与方被攻陷的情况。

**情况 1：协助者  $P_h$  被攻陷。**在这种情况下， $\mathcal{S}$  的工作方式如下。

- 当  $\mathcal{S}$  获得了  $P_h$  发给  $P_r$  的消息  $(\beta, p_\beta, r_\beta, c_{\beta \oplus 1})$  以及  $P_h$  发给  $P_s$  的消息  $(p_0, r_0, p_1, r_1)$  ( $\mathcal{S}$  可以收到所有“被攻陷方发给诚实方的消息”时， $\mathcal{S}$  在理想世界中代表  $P_h$  将  $(\text{PREPARE}, \text{sid})$  发给  $\mathcal{F}_{\text{OT}}$ )。
- 在某个时刻， $\mathcal{Z}$  会给  $P_r$  发送输入  $(\text{INPUT}, \text{sid}, b)$ ，这会让  $\mathcal{S}$  从  $\mathcal{F}_{\text{OT}}$  收到  $(\text{INPUT}, \text{sid}, P_r)$  (告知  $\mathcal{S}$  “ $P_r$  已提供输入”)。此时， $\mathcal{S}$  已经获得了  $P_h$  发给  $P_r$  的消息  $(\beta, p_\beta, r_\beta, c_{\beta \oplus 1})$  以及  $P_h$  发给  $P_s$  的消息  $(p_0, r_0, p_1, r_1)$ 。 $\mathcal{S}$  检查这两条消息是一致的：即，发给  $P_r$  的  $p_\beta, r_\beta$  与发给  $P_s$  的值是一致的，以及  $c_{\beta \oplus 1}$  确实是发给  $P_s$  的  $p_{\beta \oplus 1}, r_{\beta \oplus 1}$  的哈希。如果检查不通过， $\mathcal{S}$  将发送  $(\text{ABORT}, \text{sid})$  给  $\mathcal{F}_{\text{OT}}$  (模拟  $P_s$  中止协议)，也就是  $\mathcal{S}$  将不会让  $\mathcal{F}_{\text{OT}}$  为  $P_r$  生成输出。
- 如果检查通过， $\mathcal{S}$  将模拟  $P_s$  不中止协议。在之后的某个时刻， $\mathcal{Z}$  会给  $P_s$  发送输入  $(\text{INPUT}, \text{sid}, x_0, x_1)$ ，这会让  $\mathcal{S}$  从  $\mathcal{F}_{\text{OT}}$  收到  $(\text{INPUT}, \text{sid}, P_s)$  (告知  $\mathcal{S}$  “ $P_s$  已提供输入”)。此时， $\mathcal{S}$  发送  $(\text{OUTPUT}, \text{sid}, P_r)$  给  $\mathcal{F}_{\text{OT}}$ ，让  $\mathcal{F}_{\text{OT}}$  为  $P_r$  生成输出。

以上就是  $P_h$  被攻陷时  $\mathcal{S}$  的工作方式。下面，我们需要论证理想世界和真实世界的不可区分性。  
不可区分性。令 Game 0 是真实世界的执行，我们定义 Game 1 如下：Game 1 与 Game 0 相同，除了当  $P_h$  向  $P_s$  和  $P_r$  发送了不一致的信息时， $P_s$  会中止协议。(在真实世界中，此时  $P_s$  不可能中止协议，因为他不知道  $P_h$  发给  $P_r$  的消息；但我们这里可以这样定义 Game 1，它本质上是一个思想实验)。可以观察到，除非  $P_s$  和  $P_r$  收到了不同的  $(p_\beta, r_\beta)$  但有相同的哈希，Game 0 与 Game 1 将完全相同。因此，环境  $\mathcal{Z}$  能区分 Game 0 和 Game 1 的概率就是找到哈希碰撞的概率，而基于抗碰撞哈希的假设，这个概率是可忽略的。

最后，我们观察到 Game 1 的执行与理想世界的执行是完全一样的。因此，理想世界和真实世界不可区分。

**情况 2：接收方  $P_r$  被攻陷。**在这种情况下， $\mathcal{S}$  的工作方式如下。

- 在某个时刻， $\mathcal{Z}$  会给  $P_h$  发送  $(\text{PREPARE}, \text{sid})$ ，这条消息会让真实世界中  $P_r$  从  $P_h$  处收到  $(\beta, p_\beta, r_\beta, c_{\beta \oplus 1})$ 。此时， $\mathcal{S}$  从  $\mathcal{F}_{\text{OT}}$  收到  $(\text{PREPARE}, \text{sid}, P_h)$ ， $\mathcal{S}$  需要“编造”出一条模拟的消息发给  $P_r$ 。 $\mathcal{S}$  均匀随机选取

$$\beta \leftarrow_{\$} \{0, 1\}, \quad p_0, p_1 \leftarrow_{\$} \mathcal{M}, \quad r_0, r_1 \leftarrow_{\$} \mathcal{R}$$

并计算

$$c_0 = H(p_0, r_0), \quad c_1 = H(p_1, r_1) \tag{5.1}$$

然后将  $(\beta, p_\beta, r_\beta, c_{\beta \oplus 1})$  发给  $P_r$ ，正如真实世界中  $P_h$  所做的那样。

- 之后的某个时刻，当  $\mathcal{S}$  获得  $P_r$  发给  $P_s$  的消息  $(\tau, \hat{c}_0, \hat{c}_1)$  时，首先， $\mathcal{S}$  检查  $\hat{c}_0 \stackrel{?}{=} c_0$  和  $\hat{c}_1 \stackrel{?}{=} c_1$ ，这里的  $c_0, c_1$  是  $\mathcal{S}$  在上一步通过式 (5.1) 所计算出来的。

如果检查不通过， $\mathcal{S}$  将发送  $(\text{ABORT}, \text{sid})$  给  $\mathcal{F}_{\text{OT}}$  (模拟  $P_s$  中止协议)。

否则， $\mathcal{S}$  通过计算  $b = \tau \oplus \beta$  来“提取” $P_r$  的输入  $b$ ，并在理想世界中代表  $P_r$  将  $(\text{INPUT}, \text{sid}, b)$  发给  $\mathcal{F}_{\text{OT}}$ 。

- 假设协议没有中止，之后的某个时刻， $\mathcal{Z}$  会将  $P_s$  发送输入  $(\text{INPUT}, \text{sid}, x_0, x_1)$ ，这条消息会让真实世界的  $P_r$  从  $P_s$  处收到  $(e_0, e_1)$ 。

在理想世界中， $\mathcal{S}$  会从  $\mathcal{F}_{\text{OT}}$  收到  $(\text{INPUT}, \text{sid}, P_s)$ （告知  $\mathcal{S}$  “ $P_s$  已提供输入”）。 $\mathcal{S}$  需要“编造”一条消息发给  $P_r$ 。注意，在这个时候  $\mathcal{F}_{\text{OT}}$  已经收到了  $P_s$  的输入  $x_0, x_1$ ，而  $\mathcal{S}$  并不知道这个输入。但是， $\mathcal{S}$  可以发送  $(\text{OUTPUT}, \text{sid}, P_r)$  给  $\mathcal{F}_{\text{OT}}$  使其生成  $P_r$  的输出  $(\text{OUTPUT}, \text{sid}, x_b)$ ，这条消息会直接发给  $\mathcal{S}$ 。当得到  $x_b$  后， $\mathcal{S}$  计算

$$e_b = x_b \oplus p_\beta, \quad e_{b \oplus 1} \leftarrow_{\$} \mathcal{M}$$

并将  $(e_0, e_1)$  作为模拟的消息发给  $P_r$ 。

以上就是  $P_r$  被攻陷时  $\mathcal{S}$  的工作方式。下面，我们来论证理想世界和真实世界的不可区分性。不可区分性。令 Game 0 是真实世界的执行。那么， $P_s$  和  $P_h$  将通过式 (5.1) 的方式计算  $c_0, c_1$ ，并且

- $P_h$  会将  $c_{\beta \oplus 1}$  发给  $P_r$ ；
- 当  $P_s$  从  $P_r$  处收到  $(\tau, \hat{c}_0, \hat{c}_1)$  时，他会检查  $\hat{c}_0 = c_0$  和  $\hat{c}_1 = c_1$ 。

现在，定义 Game 1 如下：Game 1 与 Game 0 相同，除了  $P_s$  和  $P_h$  计算  $c_0, c_1$  的方式变为

$$c_\beta = H(p_\beta, r_\beta), \quad c_{\beta \oplus 1} = H(p, r_{\beta \oplus 1})$$

其中  $p \leftarrow_{\$} \mathcal{M}$  是均匀随机选择的消息。也就是说，对于  $c_{\beta \oplus 1}$ ，将  $p_{\beta \oplus 1}$  替换为随机的  $p$ 。注意，在这个游戏 (Game) 中， $P_s$  仍然按照协议规定使用  $p_0, p_1$  来计算  $e_0, e_1$ ，即  $e_0 \leftarrow x_0 \oplus p_\tau$ ,  $e_1 \leftarrow x_1 \oplus p_{\tau \oplus 1}$ （在真实世界中， $P_s$  不可能这么做，因为他不知道  $\beta$ ；但我们这里可以这样定义 Game 1，它是一个思想实验）。于是，我们可以发现  $\mathcal{Z}$  能区分 Game 0 和 Game 1 的概率就是区分  $H(p_{\beta \oplus 1}, r_{\beta \oplus 1})$  和  $H(p, r_{\beta \oplus 1})$  的概率。基于哈希函数的隐藏性假设，这个概率是可忽略的。

注意到  $p_{\beta \oplus 1}$  在 Game 1 中唯一用到的地方就是作为“一次一密”的密钥加密  $x_{b \oplus 1}$  得到  $e_{b \oplus 1}$ ，其中  $b = \tau \oplus \beta$ 。因此，定义 Game 2 如下：Game 2 与 Game 1 相同，除了  $P_s$  在最后一步直接均匀随机选取

$$e_{b \oplus 1} \leftarrow_{\$} \mathcal{M}$$

作为最后一步发给  $P_r$  的消息（在 Game 1 中  $e_{b \oplus 1} = x_{b \oplus 1} \oplus p_{\beta \oplus 1}$ ）。

由于在 Game 1 中将  $c_{\beta \oplus 1}$  修改为了  $c_{\beta \oplus 1} \leftarrow H(p, r_{\beta \oplus 1})$ ，故  $P_r$  不可能从  $c_{\beta \oplus 1}$  中获得任何关于  $p_{\beta \oplus 1}$  的信息。那么根据“一次一密”的完美隐私性，Game 1 与 Game 2 对于  $\mathcal{Z}$  完美不可区分。最后，Game 2 与理想世界的执行是完全相同的。因此，理想世界和真实世界不可区分。

**情况 3：**发送方  $P_s$  被攻陷。在这种情况下， $\mathcal{S}$  的工作方式如下。

- 在某个时刻， $\mathcal{Z}$  会给  $P_h$  发送  $(\text{PREPARE}, \text{sid})$ ，这条消息会让真实世界中  $P_s$  从  $P_h$  处收到  $(p_0, r_0, p_1, r_1)$ . 在理想世界中，这对应于  $\mathcal{S}$  从  $\mathcal{F}_{\text{OT}}$  收到  $(\text{PREPARE}, \text{sid}, P_h)$ .  $\mathcal{S}$  选取

$$p_0, p_1 \leftarrow_{\$} \mathcal{M}, \quad r_0, r_1 \leftarrow_{\$} \mathcal{R}$$

并将  $(p_0, r_0, p_1, r_1)$  发给  $P_s$ ，正如真实世界中  $P_h$  所做的那样。

- 之后的某个时刻， $\mathcal{Z}$  会给  $P_r$  发送输入  $(\text{INPUT}, \text{sid}, b)$ ，这条消息会让真实世界中的  $P_s$  从  $P_r$  处收到  $(\tau, c_0, c_1)$ .

在理想世界中， $\mathcal{S}$  会从  $\mathcal{F}_{\text{OT}}$  收到  $(\text{INPUT}, \text{sid}, P_r)$  (告知  $\mathcal{S}$  “ $P_r$ ” 已提供输入)。此时， $\mathcal{S}$  计算

$$c_0 = H(p_0, r_0), \quad c_1 = H(p_1, r_1), \quad \tau \leftarrow_{\$} \{0, 1\}$$

并模拟  $P_r$  将  $(\tau, c_0, c_1)$  发给  $P_s$ . 也就是说， $\mathcal{S}$  诚实地计算  $c_0, c_1$ ，但  $\tau$  是随机选取的。

- 之后的某个时刻，当  $P_r$  从  $P_s$  处收到  $(e_0, e_1)$  时， $\mathcal{S}$  计算

$$x_0 = e_0 \oplus p_\tau, \quad x_1 = e_1 \oplus p_{\tau \oplus 1}$$

来“提取” $P_s$  的输入  $x_0, x_1$ ，并在理想世界中代表  $P_s$  将  $(\text{INPUT}, \text{sid}, x_0, x_1)$  发给  $\mathcal{F}_{\text{OT}}$ . 此时， $\mathcal{F}_{\text{OT}}$  获得了来自  $P_s$  和  $P_r$  的输入，于是  $\mathcal{S}$  发送  $(\text{OUTPUT}, \text{sid}, P_r)$  给  $\mathcal{F}_{\text{OT}}$  使其生成  $P_r$  的输出，这个输出将直接转发给  $\mathcal{Z}$ .

以上就是  $P_s$  被攻陷时  $\mathcal{S}$  的工作方式。下面，我们来论证理想世界和真实世界的不可区分性。

**不可区分性。**首先，根据  $x_0, x_1$  的提取方式可知，在理想世界和真实世界中， $P_r$  的输出都是  $x_b$ . 然后，理想世界和真实世界唯一的不同，是  $\tau$  的生成方式。在理想世界中， $\tau \leftarrow_{\$} \{0, 1\}$ ; 在真实世界中， $\tau = b \oplus \beta$ , 而  $\beta \leftarrow_{\$} \{0, 1\}$  是  $P_h$  均匀随机选取的。因此，理想世界和真实世界中  $\tau$  的分布完全相同，它们完美不可区分。

□

## 5.4 IKNP OT 扩展协议

上面我们分别介绍了半诚实安全和恶意安全的 OT 协议，在一个 MPC 协议中，往往需要大量的 OT 协议。由于两方 OT 无法基于单向函数 (one way function) 实现<sup>[22]</sup>，而必须使用开销较大的公钥密码部件，因此，通常的做法是，首先选用合适的基 OT (base OT) 协议 (例如文献<sup>[21,23-25]</sup>的协议)，然后使用 OT 扩展技术高效地实现多个 OT. 下面，我们将介绍 Ishai, Kilian, Nissim 和 Petrank<sup>[11]</sup> (IKNP) 提出的 OT 扩展协议。

OT 扩展的思想与混合加密 (hybrid encryption) 类似。当我们希望用公钥加密算法加密一条很长的消息时，直接对其加密是很低效的；然而，我们可以用公钥加密算法加密一个对称加密算法的密钥  $k$ ，然后用对称加密算法使用密钥  $k$  加密消息。这样的方法只需黑盒使用对称加密，就可以大大扩展公钥加密的效率。类似地，在 OT 扩展中，我们希望：将参与双方少量的 ( $k$  个) OT，通过高效的对称密码组件，扩展到大量的 ( $m$  个) OT.

符号表示。OT 扩展的目标是实现  $m$  个长度为  $\ell$  比特的 OT，记作  $\text{OT}_\ell^m$ ，它实现了如图 5.5 所示的理想功能：

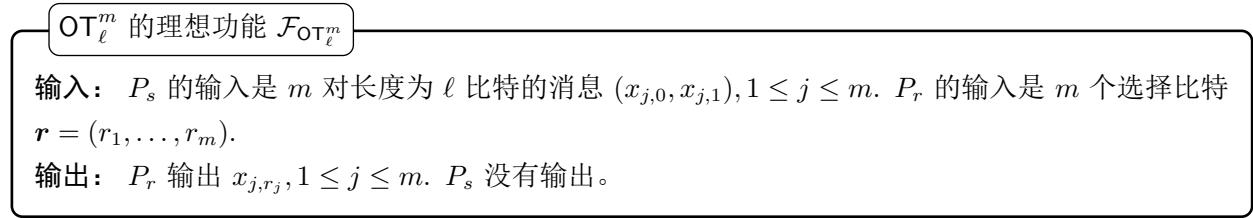


图 5.5:  $\text{OT}_\ell^m$  的理想功能  $\mathcal{F}_{\text{OT}_\ell^m}$

参与双方  $P_s$  和  $P_r$  本身可以执行  $k$  个 OT ( $k < m$ )。我们不妨假设这  $k$  个 OT 传输的是长度为  $m$  的消息<sup>1</sup>，即我们希望基于  $\text{OT}_m^k$  构造  $\text{OT}_\ell^m$ .

在下面的叙述中，我们将用大写字母表示矩阵，用加粗的小写字母表示向量。对于矩阵  $M$ ，我们用  $\mathbf{m}_j$  表示它的第  $j$  行，用  $\mathbf{m}^i$  表示它的第  $i$  列，用  $\mathbf{m}_j[i]$  表示它第  $j$  行的第  $i$  个元素。符号  $b \cdot \mathbf{v}$  表示比特  $b$  与向量  $\mathbf{v}$  相乘，当  $b = 0$  时，它的值为  $\mathbf{0}$ ；当  $b = 1$  时，它的值为  $\mathbf{v}$ .

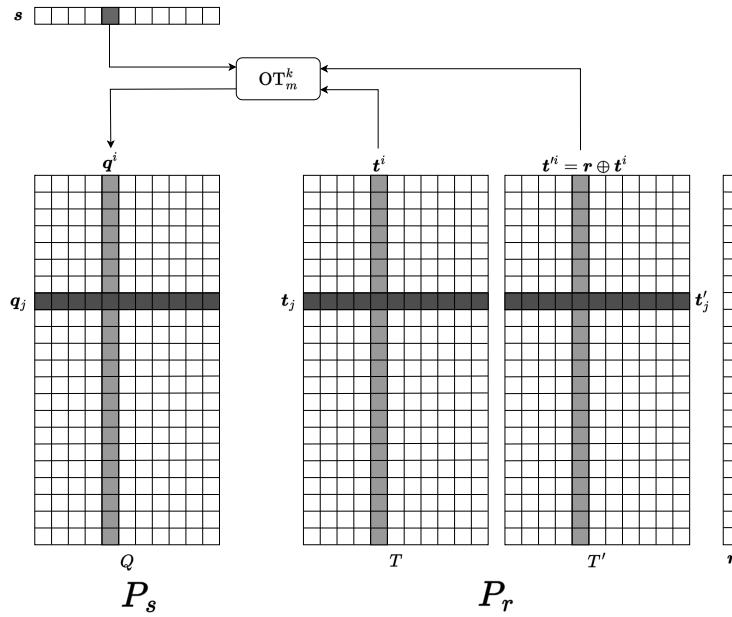


图 5.6: IKNP 协议示意图

### 5.4.1 协议描述

IKNP 协议是在随机谕示机模型下（见第 2.3.2 节）实现的。协议的初始设置是一个随机谕示机  $H : [m] \times \{0, 1\}^k \rightarrow \{0, 1\}^\ell$ . 协议的参与双方可以访问  $\text{OT}_m^k$  的理想功能  $\mathcal{F}_{\text{OT}_m^k}$ .  $k$  是基 OT 的数量，实际上

<sup>1</sup>如果 OT 传输的消息长度小于  $m$ ，可以采用如下方法。将需要发送的消息记为  $(x_0, x_1)$ ,  $|x_0| = |x_1| = m$ . 首先以两条均匀随机的消息  $(k_0, k_1)$  为输入调用 OT，它们是伪随机数生成器（见第 2.3.3 节）的种子（密钥），然后发送方将  $(PRG(k_0) \oplus x_0, PRG(k_1) \oplus x_1)$  发给接收方，其中  $PRG$  是伪随机数生成器。接收方拥有一个密钥，只能解密一条消息。换句话说，通过伪随机数生成器，OT 是可以延展的。

是协议的一个安全参数。发送方  $P_s$  的输入是  $m$  对长度为  $\ell$  比特的消息  $(x_{j,0}, x_{j,1}), 1 \leq j \leq m$ . 接收方  $P_r$  的输入是  $m$  个选择比特  $\mathbf{r} = (r_1, \dots, r_m)$ .

图 5.6 展示了协议的运行过程。首先，接收方  $P_r$  生成一个  $m \times k$  比特的均匀随机矩阵  $T$ .  $P_r$  将矩阵  $T$  的每一列  $\mathbf{t}^i$  与选择比特  $\mathbf{r}$  进行异或计算，得到新矩阵  $T'$ ，即  $T'$  的每一列  $\mathbf{t}'^i = \mathbf{r} \oplus \mathbf{t}^i$ . 发送方  $P_s$  均匀随机地生成一个长度为  $k$  比特的向量  $\mathbf{s} \leftarrow_{\$} \{0, 1\}^k$ .

接着，双方调用  $k$  个 OT 的理想功能  $\mathcal{F}_{OT_m^k}$ . 注意，此时  $P_r$  扮演了发送方的角色，它将  $T$  和  $T'$  的每一列作为  $OT_m^k$  的输入，即输入为  $(\mathbf{t}^i, \mathbf{t}'^i), 1 \leq i \leq k$ .  $P_s$  扮演了接收方的角色，他的输入是上一步中生成的随机向量  $\mathbf{s}$ .  $P_s$  将  $\mathcal{F}_{OT_m^k}$  的输出按列组成  $m \times k$  比特的矩阵  $Q$ .

我们仔细观察矩阵  $Q$ . 因为 OT 是按列执行的，对于其每一列  $\mathbf{q}^i$ ，如果  $s_i = 0$ ，那么  $\mathbf{q}^i = \mathbf{t}^i$ ；如果  $s_i = 1$ ，那么  $\mathbf{q}^i = \mathbf{t}'^i = \mathbf{r} \oplus \mathbf{t}^i$ ，即  $\mathbf{q}^i = (s_i \cdot \mathbf{r}) \oplus \mathbf{t}^i$ .

再观察矩阵  $Q$  的行。对于每一行  $\mathbf{q}_j$ ，如果这一行对应的  $r_j = 0$ ，那么  $\mathbf{t}_j = \mathbf{t}'_j$ ，此时无论  $\mathbf{s}$  为何值，都有  $\mathbf{q}_j = \mathbf{t}_j$ ；如果  $r_j = 1$ ，那么有

$$\mathbf{q}_j[i] = \begin{cases} \mathbf{t}_j[i], & s_i = 0 \\ r_j \oplus \mathbf{t}_j[i], & s_i = 1 \end{cases} \quad (5.2)$$

式 (5.2) 可以写作  $\mathbf{q}_j = (r_j \cdot \mathbf{s}) \oplus \mathbf{t}_j$ . 当  $r_j = 0$  时， $\mathbf{q}_j = \mathbf{t}_j$  也满足  $\mathbf{q}_j = (r_j \cdot \mathbf{s}) \oplus \mathbf{t}_j$ . 综上，矩阵  $Q$  的每一行满足  $\mathbf{q}_j = (r_j \cdot \mathbf{s}) \oplus \mathbf{t}_j, 1 \leq j \leq m$ .

于是，我们得到了  $\mathbf{q}_j, \mathbf{t}_j, \mathbf{s}$  和  $r_j$  之间的关系。让我们纵览全局。原始 OT 按列执行，因而有  $\mathbf{q}^i = (s_i \cdot \mathbf{r}) \oplus \mathbf{t}^i$ ；而矩阵的行也满足  $\mathbf{q}_j = (r_j \cdot \mathbf{s}) \oplus \mathbf{t}_j, 1 \leq j \leq m$ . 这充分体现了 OT 的对称性！原始 OT 的选择比特是  $s_i$ ，输入消息是  $\mathbf{t}^i$  和  $\mathbf{r} \oplus \mathbf{t}^i$ ，输出消息是  $\mathbf{q}^i$ ；我们也可以把  $r_j$  看成选择比特，那么输入消息就是  $\mathbf{q}_j$  和  $\mathbf{q}_j \oplus \mathbf{s}$ ，输出消息是  $\mathbf{t}_j$ （当  $r_j = 0$  时， $\mathbf{t}_j = \mathbf{q}_j$ ；当  $r_j = 1$  时， $\mathbf{t}_j = \mathbf{q}_j \oplus \mathbf{s}$ ），同时，发送方和接收方的身份也进行了对调。于是，我们便把  $k$  个 OT 扩展为了  $m$  个 OT！

最后，发送方  $P_s$  通过随机谕示机对消息进行加密，对  $1 \leq j \leq m$ ，其计算  $y_{j,0} = x_{j,0} \oplus H(j, \mathbf{q}_j)$ ,  $y_{j,1} = x_{j,1} \oplus H(j, \mathbf{q}_j \oplus \mathbf{s})$ ，并将  $(y_{j,0}, y_{j,1})$  发给接收方  $P_r$ . 接收方  $P_r$  输出  $z_j = y_{j,r_j} \oplus H(j, \mathbf{t}_j), 1 \leq j \leq m$ . 图 5.7 是 IKNP 协议形式化的描述。

### 5.4.2 安全性的直观说明

我们首先在半诚实安全模型下直观分析一下 IKNP OT 扩展协议的安全性。

**发送方的隐私性。**发送方使用了  $H(j, \mathbf{q}_j)$  和  $H(j, \mathbf{q}_j \oplus \mathbf{s})$  分别加密  $x_{j,0}$  和  $x_{j,1}$ . 接收方知道  $(\mathbf{q}_j, \mathbf{q}_j \oplus \mathbf{s})$  其中的一个；由于  $\mathbf{s}$  是均匀随机选取的，因此接收方不知道另一个。当然，接收方可以试图“猜测”这个消息并请求随机谕示机，但是这个消息均匀分布于  $2^k$  的空间中，因此，当接收方只能对随机谕示机进行多项式次数的访问时，他成功的概率是可忽略的。

**接收方的隐私性。**在发送方的视角中，协议中看到的信息只有矩阵  $Q$ . 由于矩阵  $T$  是均匀随机选取的，因此  $T$  和  $T'$  都是均匀随机的。无论  $\mathbf{s}$  为何值，矩阵  $Q$  都是均匀随机的矩阵，因此不会泄露有关  $\mathbf{r}$  的信息。

对于恶意敌手模型的情况，我们作一个简单的分析。协议中，发送方选择了随机向量  $\mathbf{s}$ ，并把  $(y_{j,0}, y_{j,1})$  发给接收方；而无论发送方如何选择  $(y_{j,0}, y_{j,1})$  的值，它们都对应于输入为  $x_{j,0} = y_{j,0} \oplus H(j, \mathbf{q}_j)$ ,  $x_{j,1} = y_{j,1} \oplus H(j, \mathbf{q}_j \oplus \mathbf{s})$  时的消息，也就是等价于输入替换。因此，协议对恶意发送方是安全的。

**IKNP OT 扩展协议**

公共参数：安全参数  $k$ .

设置：随机谕示机  $H : [m] \times \{0, 1\}^k \rightarrow \{0, 1\}^\ell$ ;  $\text{OT}_m^k$  的理想功能  $\mathcal{F}_{\text{OT}_m^k}$ .

$P_s$  的输入： $m$  对长度为  $\ell$  比特的消息  $(x_{j,0}, x_{j,1}), 1 \leq j \leq m$ .

$P_r$  的输入： $m$  个选择比特  $\mathbf{r} = (r_1, \dots, r_m)$ .

1.  $P_s$  选择均匀随机向量  $\mathbf{s} \leftarrow \{0, 1\}^k$ ,  $P_r$  选择  $m \times k$  的均匀随机矩阵  $T$ .
2. 调用  $\mathcal{F}_{\text{OT}_m^k}$ ,  $P_s$  作为接收方, 其输入是  $\mathbf{s}$ ;  $P_r$  作为发送方, 其输入是  $\mathbf{t}^i, \mathbf{r} \oplus \mathbf{t}^i, 1 \leq i \leq k$ .
3. 将  $P_s$  收到的  $m \times k$  矩阵记为  $Q$  (注意, 矩阵  $Q$  满足  $\mathbf{q}^i = (\mathbf{s}_i \cdot \mathbf{r}) \oplus \mathbf{t}^i, \mathbf{q}_j = (r_j \cdot \mathbf{s}) \oplus \mathbf{t}_j$ )。对  $1 \leq j \leq m$ ,  $P_s$  计算  $y_{j,0} = x_{j,0} \oplus H(j, \mathbf{q}_j)$ ,  $y_{j,1} = x_{j,1} \oplus H(j, \mathbf{q}_j \oplus \mathbf{s})$ , 并发送  $(y_{j,0}, y_{j,1})$ .
4.  $P_r$  输出  $z_j = y_{j,r_j} \oplus H(j, \mathbf{t}_j), 1 \leq j \leq m$ .

图 5.7: IKNP OT 扩展协议

然而, 对于恶意的接收方, 我们通过一个例子来表明协议确实是不安全的。接收方不遵循协议计算矩阵  $T'$ , 而是让  $\mathbf{t}^i$  和  $\mathbf{t}'^i$  只有第  $i$  个位置不同。不失一般性地, 假设  $\mathbf{t}^i[i] = 0, \mathbf{t}'^i[i] = 1$ . 那么此时, 可以观察到, 发送方得到的矩阵  $Q$  的第  $j$  行  $\mathbf{q}_j$  的第  $j$  个位置就包含  $s_j$ , 而其他位置的值是接收方知道的 (因为这些位置  $T$  和  $T'$  是相同的)。于是, 如果接收方拥有关于消息空间的一些先验知识 (例如, 哪些是有意义的消息), 接收方分别取  $s_j = 0$  和  $s_j = 1$  请求两次  $H$ , 就可以确定  $s_j$  的值。当接收方恢复出向量  $\mathbf{s}$  后, 他可以解密两条消息, 破坏发送方的隐私性。

### 5.4.3 安全性证明

现在, 我们给出形式化的安全性证明。

**定理 5.4.** 设  $t$  是敌手请求随机谕示机  $H$  的次数的上界。图 5.7 展示的协议  $\Pi_{\text{IKNP}}$  对恶意的发送方和半诚实的接收方在  $\{H, \mathcal{F}_{\text{OT}_m^k}\}$ -混合世界中对于静态敌手  $UC$ -安全实现了理想功能  $\mathcal{F}_{\text{OT}_\ell^m}$  (图 5.5), 其中敌手的区分优势不超过  $(t + 1) \cdot 2^{-k}$ .

**证明.** 要证明该定理, 我们需要构造模拟器  $\mathcal{S}$ , 使得对所有 PPT 的环境  $\mathcal{Z}$ , 都有

$$\text{EXEC}_{\Pi_{\text{IKNP}}, \mathcal{A}, \mathcal{Z}}^{H, \mathcal{F}_{\text{OT}_m^k}} \approx \text{EXEC}_{\mathcal{F}_{\text{OT}_\ell^m}, \mathcal{S}, \mathcal{Z}}$$

其中  $\mathcal{A}$  是平凡敌手。

**模拟器  $\mathcal{S}$ .** 首先, 模拟器  $\mathcal{S}$  在内部运行  $\mathcal{A}$ , 转发  $\mathcal{Z}$  与  $\mathcal{A}$  之间的消息。并且,  $\mathcal{S}$  需要模拟  $H$  和  $\mathcal{F}_{\text{OT}_m^k}$ 。我们分别考虑发送方和接收方被攻陷的情况。

**情况 1: 发送方  $P_s$  被攻陷。** 在这种情况下, 敌手是恶意的, 可以不遵循协议。 $\mathcal{S}$  的工作方式如下。

- 在协议第 2 步,  $\mathcal{S}$  记录  $P_s$  对  $\mathcal{F}_{\text{OT}_m^k}$  的输入  $\mathbf{s}$ , 并选择一个  $m \times k$  的均匀随机矩阵  $Q$  作为  $\mathcal{F}_{\text{OT}_m^k}$  的输出 ( $\mathcal{S}$  不知道  $P_r$  的输入)。

- 在协议第 3 步, 当  $P_s$  发送  $(y_{j,0}, y_{j,1}), 1 \leq j \leq m$  给  $P_r$  时,  $\mathcal{S}$  “提取”出  $x_{j,0} = y_{j,0} \oplus H(j, \mathbf{q}_j)$ ,  $x_{j,1} = y_{j,1} \oplus H(j, \mathbf{q}_j \oplus \mathbf{s})$ ,  $1 \leq j \leq m$ . 并在理想世界中代表  $P_s$  将  $(x_{j,0}, x_{j,1}), 1 \leq j \leq m$  输入给  $\mathcal{F}_{\text{OT}}^m_\ell$ .

**不可区分性。**首先, 根据前面的推导, 矩阵  $Q$  满足  $\mathbf{q}_j = (r_j \cdot \mathbf{s}) \oplus \mathbf{t}_j, 1 \leq j \leq m$ , 因而协议的正确性成立。根据  $\mathcal{S}$  计算  $(x_{j,0}, x_{j,1})$  的方式,  $P_r$  在理想世界和真实世界中的输出是一致的。其次, 在敌手的视角中, 无论在真实世界还是理想世界,  $Q$  都是均匀随机的矩阵。因此, 理想世界和真实世界完美不可区分。

**情况 2:** 接收方  $P_r$  被攻陷。在这种情况下, 敌手是半诚实的, 因此  $\mathcal{S}$  知道  $P_r$  的协议输入  $\mathbf{r}$ .  $\mathcal{S}$  的工作方式如下。

- 在协议第 2 步,  $\mathcal{S}$  模拟  $P_r$  遵循协议均匀随机地选择矩阵  $T$  并以  $\mathbf{t}^i, \mathbf{r} \oplus \mathbf{t}^i, 1 \leq i \leq k$  为输入调用  $\mathcal{F}_{\text{OT}}^k_m$ .  $\mathcal{S}$  模拟  $P_s$  均匀随机选择  $\mathbf{s}$ , 并诚实地计算  $Q$ .
- 当  $\mathcal{S}$  在理想世界中收到协议的输出  $z_j, 1 \leq j \leq m$  时,  $\mathcal{S}$  计算  $y_{j,r_j} = z_j \oplus H(j, \mathbf{q}_j \oplus (r_j \cdot \mathbf{s}))$ ,  $y_{j,1-r_j} = 0^\ell \oplus H(j, \mathbf{q}_j \oplus ((1 - r_j) \cdot \mathbf{s}))$ , 并将  $(y_{j,0}, y_{j,1}), 1 \leq j \leq m$  发给  $P_r$ . (也就是说,  $\mathcal{S}$  不知道  $P_s$  的输入: 它的做法是, 对于  $P_r$  应该收到的消息,  $\mathcal{S}$  诚实地计算; 对于  $P_r$  不知道的那条消息,  $\mathcal{S}$  将其替换为  $0^\ell$ )

**不可区分性。**首先, 当  $\mathbf{s} \neq 0$  时, 敌手在理想世界与真实世界的  $(y_{j,0}, y_{j,1}), 1 \leq j \leq m$  具有完全相同的分布。而  $\mathbf{s} = 0$  发生的概率是  $2^{-k}$ , 是关于  $k$  的可忽略函数。然而, 需要注意的是, 这里的敌手是可以请求  $H$  的。如果敌手恰好请求了  $H(j, \mathbf{t}_j \oplus \mathbf{s})$ , 那么敌手就可以区分理想世界和真实世界 (因为敌手拥有消息的一些先验知识, 如果请求了  $H(j, \mathbf{t}_j \oplus \mathbf{s})$  就可以解密两条消息, 从而得到  $\mathbf{s}$ )。我们将这些 (坏) 情况记为事件  $B$ , 即:  $\mathbf{s} = 0$  或敌手请求了  $H(j, \mathbf{t}_j \oplus \mathbf{s})$ . 下面计算事件  $B$  发生的概率。由于  $\mathbf{s}$  是均匀随机选择的, 因此  $\mathbf{t}_j \oplus \mathbf{s}$  均匀分布于  $2^k$  大小的空间中, 对于任意  $j$ , 敌手恰好请求到  $H(j, \mathbf{t}_j \oplus \mathbf{s})$  的概率为  $2^{-k}$ . 因为敌手请求  $H$  次数的上界为  $t$ , 我们有

$$\Pr[B] \leq (t + 1) \cdot 2^{-k}$$

即敌手区分理想世界和真实世界的优势不超过  $(t + 1) \cdot 2^{-k}$ . 定理得证。 □

## 5.5 拓展阅读

OT 最初是以 Rabin OT 的形式提出的<sup>[12]</sup>, 自提出以来就受到密码学界的广泛研究。对于 OT 的构造, 较为著名的有 PVW 的构造<sup>[21]</sup>, Chou 和 Orlandi 的构造<sup>[23]</sup>, Barreto 等人的构造<sup>[25]</sup>等。

在实际应用中, 由于 OT 必须基于公钥密码操作来实现<sup>[22]</sup>, 通常的做法是将一些基 OT(base OT) 通过 OT 扩展的方式高效地扩展到大量 OT. Beaver 通过非黑盒地使用单向函数提出了第一个 OT 扩展的方案<sup>[26]</sup>; 然而, Beaver 的方案是比较低效的。后来, 研究者提出了很多高效的 OT 扩展方案, 其中, 半诚实安全的方案有<sup>[11,27]</sup>等; 恶意安全的方案有<sup>[28-30]</sup>等。

为了进一步提高基 OT 的效率, Masny 和 Rindal 在 2019 年提出了 Endemic OT 的概念<sup>[31]</sup>, 其考虑了一种更弱的安全性定义, 提高了运行效率, 并且与 OT 扩展技术所兼容, 因而得到了广泛的关注和使用。

最后, libOTe<sup>[32]</sup>是一个著名的 OT 库。



# Chapter 6

## 基于线性秘密分享的协议

从本章起，我们开始介绍经典的 MPC 协议。我们从半诚实安全的协议开始，第6章介绍基于线性秘密分享的 MPC 协议，第7章介绍通过 Beaver 三元组实现的 MPC 协议；第8章介绍基于混淆电路的 MPC 协议；最后，在第9章，我们介绍恶意安全的 MPC 协议。

### 6.1 BGW 协议（半诚实安全）

#### 6.1.1 协议描述

让我们回到第3章所介绍的协议，它就是 Ben-Or, Goldwasser, Wigderson<sup>[33]</sup>(BGW) 所提出的著名协议的半诚实安全版本。我们再次展示整个协议，如图 6.2所示。其中， $t < n/2$ .

#### 6.1.2 安全性证明

在第3章中，我们已经做了一个很接近于形式化证明的安全性分析。假设被攻陷方的数量为  $t$ . 安全性分析基于两个观察：(1) 在输入分享阶段和计算乘法门时，被攻陷方收到的  $t$  个份额只是均匀随机值。(2) 在输出重建阶段，被攻陷方收到的诚实方份额可以根据已有的份额和输出值计算出来。现在，我们给出通用可组合模型下的形式化证明。

我们采用第4.2.3节所定义的安全函数计算理想功能  $\mathcal{F}_{\text{sfe}}$ ，如图 6.1所示。

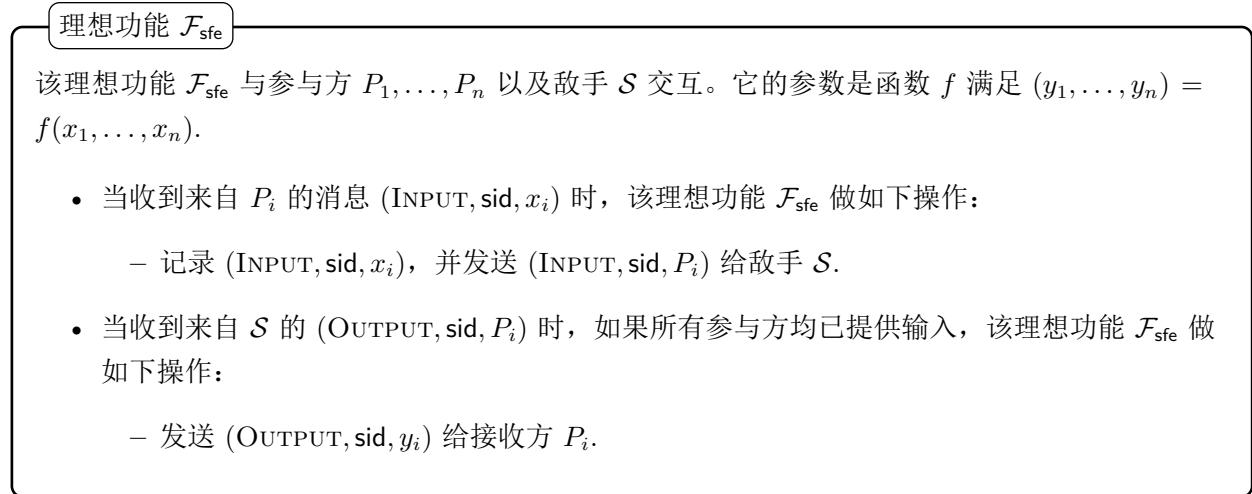
为了使证明更加简洁，我们假设电路在每个输出之前都紧跟着一个乘法门。如果电路对于输出  $y_i$  不满足这样的结构，我们可以增加一个新的乘法门计算  $y_i x'_i$  并输出，其中  $x'_i = 1$ . 这样的修改最多增加常数个乘法门。

注意，即使没有这个假设，协议仍然是安全的，但会让证明更加复杂。我们将在后面讨论这一点。

**定理 6.1.** 假设参与方之间存在点对点安全信道。假设被攻陷方数量不超过  $t$ ,  $t < n/2$ . 图 6.2中的协议  $\Pi_{\text{BGW}}$  对于静态半诚实敌手  $UC$ -安全实现了理想功能  $\mathcal{F}_{\text{sfe}}$  (图 6.1)。

证明. 要证明该定理，我们需要构造模拟器  $\mathcal{S}$ ，使得对所有环境  $\mathcal{Z}$ ，都有

$$\text{EXEC}_{\Pi_{\text{BGW}}, \mathcal{A}, \mathcal{Z}} \approx \text{EXEC}_{\mathcal{F}_{\text{sfe}}, \mathcal{S}, \mathcal{Z}}$$

图 6.1: 安全函数计算理想功能  $\mathcal{F}_{\text{sfe}}$ 

其中  $\mathcal{A}$  是平凡敌手。因为 BGW 协议是信息论安全的，所以这里不需要限制环境  $\mathcal{Z}$  是 PPT 的。

**模拟器  $\mathcal{S}$ .** 模拟器  $\mathcal{S}$  在内部运行  $\mathcal{A}$ ，转发  $\mathcal{Z}$  与  $\mathcal{A}$  之间的消息。令  $C$  为被攻陷方的集合。 $\mathcal{S}$  的工作方式如下。

- 当从  $\mathcal{F}_{\text{sfe}}$  收到  $(\text{INPUT}, \text{sid}, P_i)$  时，如果  $P_i \in C$ ，那么  $P_i$  是被攻陷的， $\mathcal{S}$  可以得到  $P_i$  的输入  $x_i$ 。 $\mathcal{S}$  以  $x_i$  为输入模拟  $P_i$  遵循协议执行输入分享。
- 当从  $\mathcal{F}_{\text{sfe}}$  收到  $(\text{INPUT}, \text{sid}, P_j)$  时，如果  $P_j \notin C$ ，那么  $P_j$  是诚实的。 $\mathcal{S}$  设置  $P_j$  的输入为  $x_j = 0$ ，然后模拟  $P_j$  遵循协议执行输入分享，将秘密份额发送给被攻陷方。
- 对于加法门和常数乘法门， $\mathcal{S}$  模拟被攻陷方遵循协议执行。
- 对于每个乘法门， $\mathcal{S}$  模拟被攻陷方遵循协议执行；对于每个诚实方， $\mathcal{S}$  模拟其分发  $[0]_t$  给被攻陷方。
- 在输出重建阶段重建  $P_i$  的输出时， $\mathcal{S}$  发送  $(\text{OUTPUT}, \text{sid}, P_i)$  给  $\mathcal{F}_{\text{sfe}}$ 。如果  $P_i$  是被攻陷的， $\mathcal{S}$  将得到  $P_i$  的输出  $y_i$ 。 $\mathcal{S}$  构造  $t$  阶多项式  $f_{y_i}$ ，满足  $f_{y_i}(0) = y_i$  且与  $P_i$  已知的  $|C|$  个秘密份额相一致。具体而言，如果  $|C| = t$ ， $\mathcal{S}$  通过拉格朗日插值构造  $f_{y_i}$ ；如果  $|C| < t$ ， $\mathcal{S}$  均匀随机选取  $t - |C|$  个诚实方的份额，然后进行拉格朗日插值构造  $f_{y_i}$ 。最后，对于  $P_j \notin C$ ， $\mathcal{S}$  将  $f_{y_i}(\alpha_j)$  作为  $P_j$  的份额发送给  $P_i$ 。

**不可区分性。** 我们通过两个引理来证明环境  $\mathcal{Z}$  在真实世界和理想世界的视图是不可区分的。

对于第一个引理，我们定义一个函数  $\text{Strip}$ .  $\text{Strip}(\text{view}_i)$  以被攻陷方  $P_i \in C$  的视图作为输入，移除输出重建阶段来自诚实方的份额后直接输出。即  $\text{Strip}(\text{view}_i)$  包括  $P_i$  的输入  $x_i$ ， $P_i$  做的随机选择， $P_i$  在输入分享阶段和计算乘法门时收到的所有份额，以及  $P_i$  在输出重建阶段收到的来自被攻陷方的份额。

**引理 1.** 令  $C \subset \{P_1, \dots, P_n\}$  是被攻陷方的集合， $|C| \leq t$ . 考虑两个输入向量  $\vec{x}^{(0)} = (x_1^{(0)}, \dots, x_n^{(0)})$  和  $\vec{x}^{(1)} = (x_1^{(1)}, \dots, x_n^{(1)})$ ，对于  $P_i \in C$  满足  $x_i^{(0)} = x_i^{(1)}$ ，即两个向量中被攻陷方的输入相同。令  $\text{view}_i^{(b)}$

BGW 协议  $\Pi_{\text{BGW}}$ 

协议分为三个阶段：输入分享阶段、计算阶段和输出重建阶段。参数  $t, n$  满足  $t < n/2$ . 参与方已经把要计算的函数转化成了算术电路。协议描述采用第3章定义的术语和符号。

**输入分享阶段。**给定  $\vec{\alpha} = (\alpha_1, \dots, \alpha_n)$ . 每个持有输入  $x_i \in \mathbb{F}$  的参与方  $P_i$  分发  $[x_i; f_{x_i}]_t$ . 然后，按照先前定义的计算顺序逐个处理电路中的门。在输入分享阶段结束后，需要确保所有输入门已被处理过。

**计算阶段。**重复以下步骤，直到所有门都被处理（然后进入下一个阶段）。考虑计算顺序中尚未处理的第一个门。根据门的类型，执行以下操作之一：

- 加法门。参与方持有  $[a; f_a]_t, [b; f_b]_t$  作为门的两个输入  $a, b$ . 参与方计算  $[a; f_a]_t + [b; f_b]_t = [a + b; f_a + f_b]_t$ .
- 带有常数  $c$  的常数乘法门。参与方持有  $[a; f_a]_t$  作为门的输入  $a$ . 参与方计算  $c[a; f_a]_t = [ca; cf_a]_t$ .
- 乘法门。参与方持有  $[a; f_a]_t, [b; f_b]_t$  作为门的两个输入  $a, b$ .
  1. 参与方计算  $[a; f_a]_t * [b; f_b]_t = [ab; f_a f_b]_{2t}$ .
  2. 定义  $h \stackrel{\text{def}}{=} f_a f_b$ . 那么  $h(0) = f_a(0) f_b(0) = ab$ , 并且各方持有  $[ab; h]_{2t}$ ; 即,  $P_i$  持有  $h(\alpha_i)$ . 每个  $P_i$  分发  $[h(\alpha_i); f_i]_t$ .
  3. 注意到  $\deg(h) = 2t \leq n - 1$ . 设  $\vec{r}$  为第3.1节定义的重组向量，即向量  $\vec{r} = (r_1, \dots, r_n)$  对于任何阶数  $\leq n - 1$  的多项式  $h$ , 都有  $h(0) = \sum_{i=1}^n r_i h(\alpha_i)$ . 参与方计算

$$\begin{aligned}\sum_{i=1}^n r_i [h(\alpha_i); f_i]_t &= \left[ \sum_{i=1}^n r_i h(\alpha_i); \sum_{i=1}^n r_i f_i \right]_t \\ &= \left[ h(0); \sum_{i=1}^n r_i f_i \right]_t = \left[ ab; \sum_{i=1}^n r_i f_i \right]_t\end{aligned}$$

**输出重建阶段。**此时，包括输出门在内的所有门都已处理。对于每个标记为  $i$  的输出门，执行以下操作：参与方持有  $[y_i; f_{y_i}]_t$ , 其中  $y_i$  是分配给输出门的值。每个  $P_j$  安全地将  $f_{y_i}(\alpha_j)$  发送给  $P_i$ , 后者使用拉格朗日插值法从  $f_{y_i}(\alpha_1), \dots, f_{y_i}(\alpha_{t+1})$  或任何其他  $t+1$  个点计算出  $y_i = f_{y_i}(0)$ .

图 6.2: BGW 协议  $\Pi_{\text{BGW}}$

是  $P_i$  运行  $\Pi_{\text{BGW}}$  后得到的视图，其中  $b = 0, 1$ . 那么，我们有

$$\{\text{Strip}(\text{view}_i^{(0)})\}_{P_i \in C} \stackrel{\text{perf}}{=} \{\text{Strip}(\text{view}_i^{(1)})\}_{P_i \in C}$$

证明. 对于  $b = 0$  或  $1$ , 我们分析协议的每个阶段所产生的分布是否相同。

- 输入分享阶段。对于  $P_i \in C$ , 有  $x_i^{(0)} = x_i^{(1)}$ , 显然会产生相同的分布。对于  $P_j \notin C$ , 有可能  $x_j^{(0)} \neq x_j^{(1)}$ , 但是在第3.1节论证过  $t$  个或更少的份额只是均匀随机值，与  $x_j^{(b)}$  无关。因此，无论  $b = 0$  或  $1$ , 产生的分布均相同。

- 加法门和常数乘法门。加法门和常数乘法门无需交互，对  $\text{view}_j^{(b)}$  的分布没有影响。
- 乘法门。由输入分享阶段的分析可知，对于被攻陷方，他们乘法门的输入是均匀随机值，在计算乘法门时收到的不超过  $t$  个份额也是均匀随机值。因此，被攻陷方计算并发送的值对于  $b = 0$  或  $1$  有相同的分布。
- 输出重建阶段。由于  $\text{Strip}$  移除了输出重建阶段来自诚实方的份额，因此输出重建阶段对  $\text{Strip}(\text{view}_i^{(b)})$  的分布没有影响。

引理得证。  $\square$

对于第二个引理，我们定义一个函数  $\text{Dress}$ 。给定向量  $\vec{y}_C = \{y_i\}_{P_i \in C}$ ， $\text{Dress}_{\vec{y}_C}$  的输入是  $\{\text{Strip}(\text{view}_i)\}_{P_i \in C}$ ，然后为其填补上被  $\text{Strip}$  移除的输出重建阶段来自诚实方的份额。具体而言， $\text{Dress}$  的做法就如模拟器  $\mathcal{S}$  那样：对于每个  $P_i \in C$ ，构造  $t$  阶多项式  $f_{y_i}$ ，满足  $f_{y_i}(0) = y_i$  且与  $P_i$  已知的  $|C|$  个秘密份额相一致。如果  $|C| = t$ ，通过拉格朗日插值构造  $f_{y_i}$ ；如果  $|C| < t$ ，均匀随机选取  $t - |C|$  个诚实方的份额，然后进行拉格朗日插值构造  $f_{y_i}$ ，最后，将  $f_{y_i}(\alpha_j)$  作为诚实方  $P_j \notin C$  的份额填补上。

**引理 2.** 令  $C \subset \{P_1, \dots, P_n\}$  是被攻陷方的集合， $|C| \leq t$ 。令  $\vec{x}$  为输入向量， $(y_1, \dots, y_n) = f(\vec{x})$ ， $\vec{y}_C = \{y_i\}_{P_i \in C}$ 。令  $\text{view}_i$  是参与方以  $\vec{x}$  为输入运行  $\Pi_{\text{BGW}}$  后  $P_i$  的视图。那么，我们有

$$\text{Dress}_{\vec{y}_C}(\{\text{Strip}(\text{view}_i)\}_{P_i \in C}) \stackrel{\text{perf}}{=} \{\text{view}_i\}_{P_i \in C}$$

证明。设  $\text{view}_i$  中的输出值为  $y'_i$ 。根据 BGW 协议的正确性可知， $y'_i = y_i$ ，其中  $(y_1, \dots, y_n) = f(\vec{x})$ 。因此， $\text{view}_i$  的输出重建阶段得到了  $y_i$  的  $n$  个秘密份额  $[y_i; f_{y_i}]_t$ 。如果  $|C| = t$ ，根据拉格朗日插值的正确性， $\text{Dress}_{\vec{y}_C}$  恰恰正确恢复了被  $\text{Strip}$  移除的份额。如果  $|C| < t$ ， $\text{Dress}_{\vec{y}_C}$  首先均匀随机选取了  $t - |C|$  个诚实方的份额：这  $t - |C|$  个份额与真实运行得到的份额都是均匀随机值，有相同的分布。然后，因为  $f_{y_i}(0) = y_i$ ，由拉格朗日插值唯一确定了剩余的诚实方份额。综上可知， $\text{Dress}_{\vec{y}_C}(\{\text{Strip}(\text{view}_i)\}_{P_i \in C}) \stackrel{\text{perf}}{=} \{\text{view}_i\}_{P_i \in C}$ 。  $\square$

有了这两个引理，我们可以论证  $\mathcal{Z}$  在真实世界和理想世界的视图是完美不可区分的。令  $\vec{x} = (x_1, \dots, x_n)$  为输入向量，令  $\vec{x}^{(0)} = (x_1^{(0)}, \dots, x_n^{(0)})$  对  $P_i \in C$  有  $x_i^{(0)} = x_i$ ，对  $P_j \notin C$  有  $x_j^{(0)} = 0$ 。那么，根据  $\mathcal{S}$  的构造，理想世界的视图为

$$\text{Dress}_{\vec{y}_C}(\{\text{Strip}(\text{view}_i^{(0)})\}_{P_i \in C})$$

令真实世界的视图为  $\{\text{view}_i\}_{P_i \in C}$ ，根据引理 1 可知

$$\{\text{Strip}(\text{view}_i^{(0)})\}_{P_i \in C} \stackrel{\text{perf}}{=} \{\text{Strip}(\text{view}_i)\}_{P_i \in C}$$

因为算法不能增加两个随机变量的统计距离<sup>1</sup>，我们有

$$\text{Dress}_{\vec{y}_C}(\{\text{Strip}(\text{view}_i^{(0)})\}_{P_i \in C}) \stackrel{\text{perf}}{=} \text{Dress}_{\vec{y}_C}(\{\text{Strip}(\text{view}_i)\}_{P_i \in C})$$

根据引理 2

$$\text{Dress}_{\vec{y}_C}(\{\text{Strip}(\text{view}_i)\}_{P_i \in C}) \stackrel{\text{perf}}{=} \{\text{view}_i\}_{P_i \in C}$$

---

<sup>1</sup> 即  $\delta(\mathcal{A}(X_1), \mathcal{A}(X_2)) \leq \delta(X_1, X_2)$ ，其中  $X_1, X_2$  是随机变量， $\delta$  表示统计距离， $\mathcal{A}$  是任何算法。它的证明留作读者的练习。

故

$$\text{Dress}_{\vec{y}_C}(\{\text{Strip}(\text{view}_i^{(0)})\}_{P_i \in C}) \stackrel{\text{perf}}{=} \{\text{view}_i\}_{P_i \in C}$$

即真实世界和理想世界的视图有完全相同的分布。证毕。

□

为什么假设输出门之前都有乘法门？以上的证明假设了输出门之前都有乘法门，如果没有这样的假设，协议仍然是安全的，但是在证明时有一些微妙的差别需要注意。例如，当  $|C| < t$  时，如果协议运行过程中计算了  $[a; f_a]_t, [b; f_b]_t$  和  $[a + b; f_{a+b}]_t$ ，而在输出重建阶段，被攻陷方又得到了这些值，那么敌手就可以通过检查  $f_a + f_b = f_{a+b}$  来试图区分真实世界和理想世界。因此  $\mathcal{S}$  的构造需要更加小心，它做的随机选择必须满足这样的一致性要求。但是，如果我们假设输出门之前都有乘法门，由于乘法门的计算过程加入了新的随机性，因此输出重建阶段的多项式就总是均匀随机的了。

## 6.2 GMW 协议

上一节介绍的 BGW 协议是基于 Shamir 秘密分享的，第1章的协议是基于复制秘密分享的。本节，我们介绍 GMW 协议<sup>[20]</sup>。它由 Goldreich, Micali, 和 Wigderson 所提出，是一个基于加法秘密分享的通用 MPC 协议，它允许最多  $n - 1$  个被攻陷方（ $n$  为参与方的数量），且适用于布尔电路和算术电路。我们首先考虑两方布尔电路，然后将其拓展到多方布尔电路；最后，我们介绍多方算术电路的 GMW 协议。

### 6.2.1 两方布尔电路 GMW 协议

#### 6.2.1.1 协议描述

我们首先介绍两方的布尔版本 GMW 协议。假设参与方  $P_1$  和  $P_2$  的输入分别为  $x_1$  和  $x_2$ ，他们想要计算  $(y_1, y_2) = f(x_1, x_2)$ ，并且已经将  $f$  转化为了布尔电路  $\mathcal{C}$ 。协议流程如下。

**加法秘密分享。**对于输入  $x_1$  的每个比特  $x_{1,k} \in \{0, 1\}$ ， $P_1$  选择随机比特  $r_{1,k}^1, r_{1,k}^2$ ，满足  $r_{1,k}^1 \oplus r_{1,k}^2 = x_{1,k}$ 。具体而言， $P_1$  选择均匀随机比特  $r_{1,k}^1$  并令  $r_{1,k}^2 = r_{1,k}^1 \oplus x_{1,k}$ 。然后将  $r_{1,k}^2$  发给  $P_2$ 。类似地， $P_2$  的输入  $x_2$  的每一个比特也以同样的方式秘密分享。

**逐门计算。**完成输入分享后， $P_1$  和  $P_2$  对电路  $\mathcal{C}$  进行逐门计算。考虑输入导线为  $w_i, w_j$ ，输出导线为  $w_k$  的门  $G$ ，每一根输入导线  $w_x$  ( $x \in \{i, j\}$ ) 的值都被秘密分享为  $s_x^1$  和  $s_x^2$ ，满足  $s_x^1 \oplus s_x^2 = w_x$ 。 $P_1$  持有  $w_i$  和  $w_j$  上的  $s_i^1$  和  $s_j^1$ ， $P_2$  持有  $s_i^2$  和  $s_j^2$ ，协议需要使得门  $G$  计算完成后， $P_1$  和  $P_2$  分别持有  $w_k$  的秘密份额  $s_k^1$  和  $s_k^2$  满足  $s_k^1 \oplus s_k^2 = w_k$ 。不失一般性地，我们假设  $\mathcal{C}$  由非门、异或门、与门组成<sup>2</sup>。下面我们依次介绍各个门的计算方式。

**非门 (NOT)。**非门只有一个输入  $w_i$ ，输出  $w_k$  为  $w_i$  的翻转。假设  $P_1$  持有  $s_i^1$ ， $P_2$  持有  $s_i^2$ ，满足  $w_i = s_i^1 \oplus s_i^2$ ，那么，

$$w_k = \overline{w_i} = \overline{s_i^1 \oplus s_i^2} = s_i^1 \oplus s_i^2 \oplus 1 = (s_i^1 \oplus 1) \oplus s_i^2$$

我们可以让  $P_1$  设置  $s_k^1 = s_i^1 \oplus 1$ ， $P_2$  设置  $s_k^2 = s_i^2$ ，即  $P_1$  将自己的输入导线的秘密份额翻转，而  $P_2$  直接将输入导线的秘密份额作为输出导线的份额。这样，参与方就获得了  $w_k$  的秘密份额。

<sup>2</sup>在离散数学中，非门、异或门、与门是完备的。

异或门 (XOR)。异或门的计算也不需要双方交互。假设异或门的输入为  $w_i, w_j$ , 输出为  $w_k = w_i \oplus w_j$ .  $P_1$  持有秘密份额  $s_i^1, s_j^1$ ,  $P_2$  持有  $s_i^2, s_j^2$ , 满足  $w_i = s_i^1 \oplus s_i^2$ ,  $w_j = s_j^1 \oplus s_j^2$ . 我们有

$$w_k = w_i \oplus w_j = (s_i^1 \oplus s_i^2) \oplus (s_j^1 \oplus s_j^2) = (s_i^1 \oplus s_j^1) \oplus (s_i^2 \oplus s_j^2)$$

我们可以让  $P_1$  设置  $s_k^1 = s_i^1 \oplus s_j^1$ ,  $P_2$  设置  $s_k^2 = s_i^2 \oplus s_j^2$ , 也就是双方分别将自己的秘密份额在本地计算异或, 即可获得  $w_k$  的秘密份额。

与门 (AND)。与门的计算需要交互,  $P_1$  和  $P_2$  需要调用 1-out-of-4 OT. 让我们看一下与门的情况。同样, 假设与门的输入为  $w_i, w_j$ , 输出为  $w_k = w_i \wedge w_j$ .  $P_1$  持有秘密份额  $s_i^1, s_j^1$ ,  $P_2$  持有  $s_i^2, s_j^2$ , 满足  $w_i = s_i^1 \oplus s_i^2$ ,  $w_j = s_j^1 \oplus s_j^2$ . 那么,

$$w_k = w_i \wedge w_j = (s_i^1 \oplus s_i^2) \wedge (s_j^1 \oplus s_j^2)$$

从  $P_1$  的角度来看, 其持有的  $s_i^1, s_j^1$  是已知的, 而  $P_2$  持有的两个份额  $s_i^2, s_j^2$  是未知的, 共有四种可能的情况。 $P_1$  不能直接获得  $P_2$  的秘密份额, 因为那样会违背隐私性, 但是  $P_1$  可以为  $P_2$  持有份额的每种可能情况准备相应的  $w_k$  的秘密份额, 并和  $P_2$  运行 1-out-of-4 OT 协议, 让  $P_2$  根据其持有的份额来选择对应的  $w_k$  的份额。

具体来说, 令  $S(\alpha, \beta) = (s_i^1 \oplus \alpha) \wedge (s_j^1 \oplus \beta)$  是与门的输出, 其中  $\alpha, \beta \in \{0, 1\}$  代表  $P_2$  持有的秘密份额  $s_i^2, s_j^2$ ,  $P_1$  均匀随机选择掩码  $r \leftarrow_{\$} \{0, 1\}$ , 并计算一张包含四个 OT 输入的表:

$$T = \begin{pmatrix} r \oplus S(0, 0) \\ r \oplus S(0, 1) \\ r \oplus S(1, 0) \\ r \oplus S(1, 1) \end{pmatrix}$$

之后  $P_1$  和  $P_2$  运行 1-out-of-4 OT 协议, 其中  $P_1$  作为发送方以表  $T$  作为输入,  $P_2$  作为接收方以  $s_i^2, s_j^2$  作为选择比特, 得到  $r \oplus S(s_i^2, s_j^2)$ . 然后,  $P_1$  和  $P_2$  分别将  $r$  和  $r \oplus S(s_i^2, s_j^2)$  作为与门输出导线的秘密份额。根据 OT 协议的隐私性,  $P_1$  不知道  $P_2$  的选择比特,  $P_2$  也不知道  $P_1$  的其余三个输入, 因此与门的输入不会被泄露。

**输出重建。**输出重建的过程非常简单。设  $P_1$  的输出为  $y_1$ , 对于  $y_1$  的每个比特  $y_{1,k} \in \{0, 1\}$ ,  $P_2$  将自己的秘密份额  $s_{1,k}^2$  发给  $P_1$ , 然后  $P_1$  计算  $y_{1,k} = s_{1,k}^1 \oplus s_{1,k}^2$ . 对于  $P_2$  的每个输出比特, 也执行同样的操作即可。

## 6.2.2 多方布尔电路 GMW 协议

### 6.2.2.1 协议描述

上述针对两方布尔电路的协议很容易拓展到多方的场景。假设  $n$  个参与方  $P_1, P_2, \dots, P_n$  想要计算  $(y_1, \dots, y_n) = f(x_1, \dots, x_n)$ , 并将函数  $f$  转换为了布尔电路  $C$ . 协议的流程仍然包括加法秘密分享、逐门计算和输出重建。

**加法秘密分享。**对于输入  $x_i$  的每个比特  $x_{i,k} \in \{0, 1\}$ ,  $P_i$  选择随机比特  $r_{i,k}^1, \dots, r_{i,k}^n$  满足  $\bigoplus_{\ell=1}^n r_{i,k}^\ell = x_{i,k}$ , 然后将  $r_{i,k}^\ell$  发给  $P_\ell$  作为秘密份额。每个参与方都以这样的方式对自己的输入进行秘密分享。可以看

出，这样的方式使得任意  $n - 1$  个参与方无法获得关于秘密的任何信息，根据第2.3.1节的定义，它是  $(n - 1, n)$ -门限秘密分享。

**逐门计算。**考虑具有输入导线  $w_i, w_j$  以及输出导线  $w_k$  的门  $G$ ，每一根输入导线  $w_x (x \in \{i, j\})$  的值都被秘密分享为  $s_x^1, \dots, s_x^n$ ，满足  $s_x^1 \oplus \dots \oplus s_x^n = w_x$ .  $P_\ell$  持有  $w_i$  和  $w_j$  上的  $s_i^\ell$  和  $s_j^\ell$ ，协议需要使得门  $G$  计算完成后， $P_\ell$  持有  $w_k$  的秘密份额  $s_k^\ell$ . 不失一般性地，我们仍假设  $\mathcal{C}$  由非门、异或门、与门组成。  
**非门 (NOT)。** 非门只有一个输入  $w_i$ ，输出  $w_k$  为  $w_i$  的翻转。假设  $P_\ell$  持有  $s_i^\ell$ ，满足  $w_i = s_i^1 \oplus \dots \oplus s_i^n$ ，那么，

$$w_k = \overline{w_i} = \overline{s_i^1 \oplus \dots \oplus s_i^n} = s_i^1 \oplus \dots \oplus s_i^n \oplus 1 = (s_i^1 \oplus 1) \oplus s_i^2 \dots \oplus s_i^n$$

我们可以让  $P_1$  设置  $s_k^1 = s_i^1 \oplus 1$ ，其他参与方  $P_\ell, \ell \neq 1$  设置  $s_k^\ell = s_i^\ell$ ，即  $P_1$  将自己的输入导线的秘密份额翻转，而其他参与方将输入导线的秘密份额作为输出导线的份额。

**异或门 (XOR)。** 假设异或门的输入为  $w_i, w_j$ ，输出为  $w_k = w_i \oplus w_j$ .  $P_\ell$  持有  $s_i^\ell, s_j^\ell$ ，满足  $w_i = s_i^1 \oplus \dots \oplus s_i^n, w_j = s_j^1 \oplus \dots \oplus s_j^n$ . 我们有

$$w_k = w_i \oplus w_j = (s_i^1 \oplus \dots \oplus s_i^n) \oplus (s_j^1 \oplus \dots \oplus s_j^n) = (s_i^1 \oplus s_j^1) \oplus \dots \oplus (s_i^n \oplus s_j^n)$$

我们可以让  $P_\ell$  设置  $s_k^\ell = s_i^\ell \oplus s_j^\ell$ ，也就是参与方将自己的秘密份额在本地计算异或，即可获得  $w_k$  的秘密份额。

**与门 (AND)。** 与门的计算最为复杂。同样，假设与门的输入为  $w_i, w_j$ ，输出为  $w_k = w_i \wedge w_j$ .  $P_\ell$  持有  $s_i^\ell, s_j^\ell$ ，满足  $w_i = s_i^1 \oplus \dots \oplus s_i^n, w_j = s_j^1 \oplus \dots \oplus s_j^n$ . 那么，因为与运算对异或运算满足分配律<sup>3</sup>，我们有

$$w_k = w_i \wedge w_j = (s_i^1 \oplus \dots \oplus s_i^n) \wedge (s_j^1 \oplus \dots \oplus s_j^n) = \left( \bigoplus_{\ell=1}^n s_i^\ell \wedge s_j^\ell \right) \oplus \left( \bigoplus_{\ell_1 \neq \ell_2} s_i^{\ell_1} \wedge s_j^{\ell_2} \right) \quad (6.1)$$

观察上式，项  $s_i^\ell \wedge s_j^\ell$  可以由  $P_\ell$  在本地计算。对于  $s_i^{\ell_1} \wedge s_j^{\ell_2}$ ，它的计算需要  $P_{\ell_1}$  和  $P_{\ell_2}$  的交互，显然， $P_{\ell_1}$  和  $P_{\ell_2}$  可以通过调用两方 GMW 协议来计算。或者，他们采用两方 GMW 协议中的方法，调用 1-out-of-4 OT 来计算  $(s_i^{\ell_1} \wedge s_j^{\ell_2}) \oplus (s_j^{\ell_1} \wedge s_i^{\ell_2})$ . 具体而言，令  $S(\alpha, \beta) = (s_i^{\ell_1} \wedge \alpha) \oplus (s_j^{\ell_1} \wedge \beta)$ ，其中  $\alpha, \beta \in \{0, 1\}$  代表  $P_{\ell_2}$  所持有的秘密份额  $s_j^{\ell_2}, s_i^{\ell_2}$ ， $P_{\ell_1}$  均匀随机选择掩码  $r \leftarrow \$_{0,1}$  并计算一张包含四个 OT 输入的表：

$$T = \begin{pmatrix} r \oplus S(0, 0) \\ r \oplus S(0, 1) \\ r \oplus S(1, 0) \\ r \oplus S(1, 1) \end{pmatrix}$$

之后  $P_{\ell_1}$  和  $P_{\ell_2}$  运行 1-out-of-4 OT， $P_{\ell_1}$  以表  $T$  为输入作为发送方， $P_{\ell_2}$  作为接收方以  $s_j^{\ell_2}, s_i^{\ell_2}$  作为选择比特，得到  $r \oplus S(s_j^{\ell_2}, s_i^{\ell_2})$ . 然后， $P_{\ell_1}$  设置  $r_{\ell_1, \ell_2} = r$ ， $P_{\ell_2}$  设置  $r_{\ell_2, \ell_1} = r \oplus S(s_j^{\ell_2}, s_i^{\ell_2})$ . 最后，每个参与方  $P_\ell$  将  $s_i^\ell \wedge s_j^\ell$  与 OT 过程中所得到的值全部计算异或，得到  $w_k$  的份额  $s_k^\ell = (s_i^\ell \wedge s_j^\ell) \oplus (\bigoplus_{\ell' \neq \ell} r_{\ell, \ell'})$ .

由式 (6.1) 可知，这样得到的秘密份额满足  $w_k = s_k^1 \oplus \dots \oplus s_k^n$ .

<sup>3</sup>这可以通过真值表来验证。

**输出重建。**输出重建的过程与两方的情形完全类似。设  $P_i$  的输出为  $y_i$ , 对于  $y_i$  的每个比特  $y_{i,k} \in \{0, 1\}$ , 其他参与方将自己的份额  $s_{i,k}^\ell$ ,  $\ell \neq i$  发给  $P_i$ , 然后  $P_i$  计算  $y_{i,k} = s_{i,k}^1 \oplus \dots \oplus s_{i,k}^n$ . 每个参与方都通过这样的方式重建输出。

最后, 完整的布尔电路 GMW 协议如图 6.3 所示。

### 布尔电路 GMW 协议 $\Pi_{\text{GMW}}^{\text{bool}}$

协议分为三个阶段: 输入分享阶段、计算阶段和输出重建阶段。参与方已经把要计算的函数转化成了包含非门、异或门和与门的布尔电路。

**输入分享阶段。**设  $P_i$  的输入为  $x_i$ . 对于  $x_i$  的每个比特  $x_{i,k} \in \{0, 1\}$ ,  $P_i$  选择随机比特  $r_{i,k}^1, \dots, r_{i,k}^n$  满足  $\bigoplus_{\ell=1}^n r_{i,k}^\ell = x_{i,k}$ , 然后将  $r_{i,k}^\ell$  发给  $P_\ell$  作为秘密份额。

**计算阶段。**重复以下步骤, 直到所有门都被计算。根据门的类型, 执行以下操作之一:

- 非门 (NOT)。假设输入导线上的值为  $w_i$ , 输出导线上的值为  $w_k$ . 参与方  $P_\ell$  持有的秘密份额为  $s_i^\ell$ .  $P_1$  设置  $s_k^1 = s_i^1 \oplus 1$  为输出导线上的份额, 其他参与方  $P_\ell$ ,  $\ell \neq 1$  设置  $s_k^\ell = s_i^\ell$  为输出导线上的份额。
- 异或门 (XOR)。假设输入导线上的值为  $w_i, w_j$ , 输出导线上的值为  $w_k$ . 参与方  $P_\ell$  持有的秘密份额为  $s_i^\ell, s_j^\ell$ .  $P_\ell$  设置  $s_k^\ell = s_i^\ell \oplus s_j^\ell$  为输出导线上的份额。
- 与门 (AND)。假设输入导线上的值为  $w_i, w_j$ , 输出导线上的值为  $w_k$ . 参与方  $P_\ell$  持有的秘密份额为  $s_i^\ell, s_j^\ell$ . 参与方执行如下操作。
  - 对于任意的  $\ell_1 \neq \ell_2$ ,  $P_{\ell_1}$  与  $P_{\ell_2}$  执行如下操作: 令  $S(\alpha, \beta) = (s_i^{\ell_1} \wedge \alpha) \oplus (s_j^{\ell_1} \wedge \beta)$ ,  $P_{\ell_1}$  均匀随机选择  $r \leftarrow \{0, 1\}$  并计算表

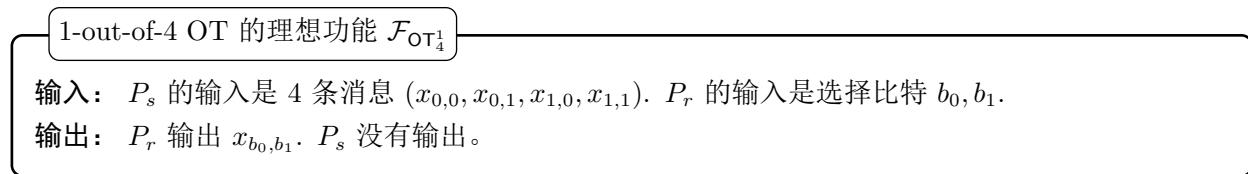
$$T = \begin{pmatrix} r \oplus S(0, 0) \\ r \oplus S(0, 1) \\ r \oplus S(1, 0) \\ r \oplus S(1, 1) \end{pmatrix}$$

$P_{\ell_1}$  以表  $T$  为输入作为发送方,  $P_{\ell_2}$  作为接收方以  $s_j^{\ell_2}, s_i^{\ell_2}$  作为选择比特运行 1-out-of-4 OT,  $P_{\ell_2}$  得到  $r \oplus S(s_j^{\ell_2}, s_i^{\ell_2})$ .  $P_{\ell_1}$  设置  $r_{\ell_1, \ell_2} = r$ ,  $P_{\ell_2}$  设置  $r_{\ell_2, \ell_1} = r \oplus S(s_j^{\ell_2}, s_i^{\ell_2})$ .

2.  $P_\ell$  设置  $s_k^\ell = (s_i^\ell \wedge s_j^\ell) \oplus (\bigoplus_{\ell' \neq \ell} r_{\ell, \ell'})$  为输出导线上的份额。

**输出重建阶段。**设  $P_i$  的输出为  $y_i$ , 对于  $y_i$  的每个比特  $y_{i,k}$ , 其他参与方将自己的份额  $s_{i,k}^\ell$ ,  $\ell \neq i$  发给  $P_i$ ,  $P_i$  计算  $y_{i,k} = s_{i,k}^1 \oplus \dots \oplus s_{i,k}^n$ .

图 6.3: 布尔电路 GMW 协议  $\Pi_{\text{GMW}}^{\text{bool}}$

图 6.4: 1-out-of-4 OT 的理想功能  $\mathcal{F}_{\text{OT}_4^1}$ 

### 6.2.2.2 安全性证明

这里，我们不考虑 1-out-of-4 OT 的具体实现，而是将其抽象成理想功能  $\mathcal{F}_{\text{OT}_4^1}$  (如图 6.4 所示)。也就是说，当参与方需要运行 1-out-of-4 OT 时，他们调用理想功能  $\mathcal{F}_{\text{OT}_4^1}$ . 下面我们证明，当  $t \leq n - 1$  时，布尔电路 GMW 协议  $\Pi_{\text{GMW}}^{\text{bool}}$  在  $\mathcal{F}_{\text{OT}_4^1}$ -混合模型下对于静态半诚实敌手 UC-安全实现了安全函数计算理想功能  $\mathcal{F}_{\text{sfe}}$  (图 6.1)。

这里的证明思路与 BGW 协议非常类似。首先，由布尔电路 GMW 协议的构造可知，参与方计算完每个门之后都得到了正确的秘密份额，协议满足正确性。对于隐私性，在输入分享阶段，协议采用的加法秘密分享使得任意不超过  $n - 1$  个参与方得到的份额都是均匀随机值。在计算阶段，非门、异或门的计算可以在本地进行，不会泄露信息；与门的计算中 1-out-of-4 OT 需要参与方的交互，由于加入了随机掩码  $r$ ,  $P_{\ell_2}$  得到的也只是均匀随机值。最后的输出重建阶段，诚实方的份额可以根据被攻陷方已知的份额和协议输出来高效地模拟。

注意，与 BGW 协议不同，GMW 协议不是信息论安全的，因为底层的 OT 协议不可能做到信息论安全 (见定理 5.1)。

**定理 6.2.** 假设被攻陷方数量不超过  $n - 1$ ，图 6.3 中的协议  $\Pi_{\text{GMW}}^{\text{bool}}$  在  $\mathcal{F}_{\text{OT}_4^1}$ -混合模型下对于静态半诚实敌手 UC-安全实现了理想功能  $\mathcal{F}_{\text{sfe}}$  (图 6.1)。

**证明.** 要证明该定理，我们需要构造模拟器  $\mathcal{S}$ ，使得对所有 PPT 的环境  $\mathcal{Z}$ ，都有

$$\text{EXEC}_{\Pi_{\text{GMW}}^{\text{bool}}, \mathcal{A}, \mathcal{Z}}^{\mathcal{F}_{\text{OT}_4^1}} \approx \text{EXEC}_{\mathcal{F}_{\text{sfe}}, \mathcal{S}, \mathcal{Z}}$$

其中  $\mathcal{A}$  是平凡敌手。

**模拟器  $\mathcal{S}$ .** 模拟器  $\mathcal{S}$  在内部运行  $\mathcal{A}$ ，转发  $\mathcal{Z}$  与  $\mathcal{A}$  之间的消息。因为是  $\mathcal{F}_{\text{OT}_4^1}$ -混合模型， $\mathcal{S}$  还需要模拟  $\mathcal{F}_{\text{OT}_4^1}$ . 令  $C$  为被攻陷方的集合。 $\mathcal{S}$  的工作方式如下。

- 当从  $\mathcal{F}_{\text{sfe}}$  收到  $(\text{INPUT}, \text{sid}, P_i)$  时，如果  $P_i \in C$ ，那么  $P_i$  是被攻陷的， $\mathcal{S}$  可以得到  $P_i$  的输入  $x_i$ .  $\mathcal{S}$  以  $x_i$  为输入模拟  $P_i$  遵循协议执行输入分享。
- 当从  $\mathcal{F}_{\text{sfe}}$  收到  $(\text{INPUT}, \text{sid}, P_j)$  时，如果  $P_j \notin C$ ，那么  $P_j$  是诚实的。 $\mathcal{S}$  设置  $P_j$  的输入为  $x_j = 0$ ，然后模拟  $P_j$  遵循协议执行输入分享，将秘密份额发送给被攻陷方。
- 对于与门的 1-out-of-4 OT， $\mathcal{S}$  模拟被攻陷方遵循协议执行。如果被攻陷方是 1-out-of-4 OT 的接收方， $\mathcal{S}$  模拟  $\mathcal{F}_{\text{OT}_4^1}$  输出一个随机比特给被攻陷方。(如果被攻陷方是 1-out-of-4 OT 的发送方，那么他在 1-out-of-4 OT 中没有得到输出，因此  $\mathcal{S}$  什么都不用做。)

- 在输出重建阶段重建  $P_i$  的输出时,  $\mathcal{S}$  发送  $(\text{OUTPUT}, \text{sid}, P_i)$  给  $\mathcal{F}_{\text{sfe}}$ . 如果  $P_i$  是被攻陷的,  $\mathcal{S}$  将得到  $P_i$  的输出  $y_i$ . 对于  $y_i$  的每个比特  $y_{i,k}$ ,  $\mathcal{S}$  随机选择诚实方的份额  $s_{i,k}^\ell$ ,  $P_\ell \notin C$ , 使得  $s_{i,k}^1 \oplus \dots \oplus s_{i,k}^n = y_{i,k}$ . 然后将诚实方的份额发送给  $P_i$ .

**不可区分性。**根据前面的分析, 很容易看出  $\mathcal{Z}$  在真实世界和理想世界的视图是不可区分的。首先, 协议的构造满足正确性, 故理想世界与真实世界中诚实方的输出相同。然后, 不论在真实世界还是理想世界, 输入分享阶段和执行 1-out-of-4 OT 时, 被攻陷方得到的都是均匀随机值。在输出重建阶段, 被攻陷方收到的诚实方份额也具有相同的分布。因此, 真实世界和理想世界是不可区分的。

□

### 6.2.3 多方算术电路 GMW 协议

#### 6.2.3.1 协议描述

在布尔电路中, 每一根导线上的值是一个比特; 在算术电路中, 每一根导线上的值是一个有限域  $\mathbb{F}$  上的元素。与布尔电路 GMW 协议的思路相同, 我们可以使用类似的方法将协议拓展到算术电路的场景。不同之处在于, 计算乘法门时, 我们使用 OLE (Oblivious Linear Evaluation, 茫然线性计算) 协议<sup>[34]</sup>来替换 OT 协议。在某个有限域  $\mathbb{F}$  上的 OLE 协议中, 发送方  $P_s$  的输入是  $\alpha, \beta \in \mathbb{F}$ , 接收方  $P_r$  的输入是  $x \in \mathbb{F}$ , 协议运行结束后,  $P_r$  获得  $\gamma = \alpha x + \beta$ ,  $P_s$  不获取任何输出。也就是说, 发送方和接收方以  $\alpha, \beta, x$  为输入做了线性计算  $\gamma = \alpha x + \beta$ , 但是发送方不知道接收方的输入  $x$ , 接收方也不知道发送方的输入  $\alpha, \beta$ . OLE 的理想功能  $\mathcal{F}_{\text{OLE}}$  如图 6.5 所示。

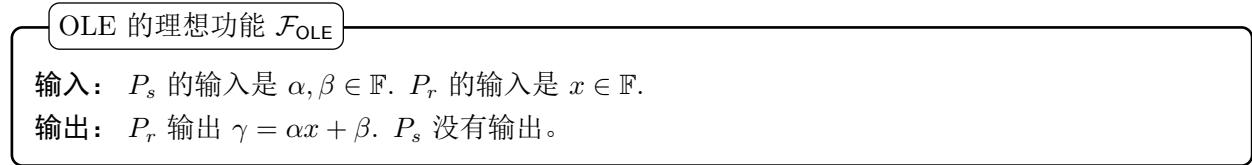


图 6.5: OLE 的理想功能  $\mathcal{F}_{\text{OLE}}$

**OLE 的实现。**读者可能会好奇 OLE 是如何实现的。实现 OLE 的朴素的方法包括: (1) 使用通用 MPC 协议 (例如第8.1节将要介绍的姚氏混淆电路协议); (2) 使用 (加法) 同态加密。(加法) 同态加密是一种公钥加密方案, 并且满足加法同态性, 即  $\text{Enc}_{\text{pk}}(x) + \text{Enc}_{\text{pk}}(y) = \text{Enc}_{\text{pk}}(x+y)$ . 基于这个工具,  $P_r$  可以生成  $(\text{pk}, \text{sk})$ , 然后计算  $\text{Enc}_{\text{pk}}(x)$  并发送给  $P_s$ ,  $P_s$  通过加法同态性计算  $\alpha \cdot \text{Enc}_{\text{pk}}(x) + \text{Enc}_{\text{pk}}(\beta) = \text{Enc}_{\text{pk}}(\alpha x + \beta)$  并发送给  $P_r$ , 最后  $P_r$  将其解密得到  $\alpha x + \beta$ . 关于更高效的 OLE 构造现在仍是密码学的研究课题之一, 这里我们不过多关注 OLE 的具体实现, 而是把它抽象为理想功能  $\mathcal{F}_{\text{OLE}}$ , 然后在  $\mathcal{F}_{\text{OLE}}$ -混合模型下构造协议。

现在, 我们介绍算术电路 GMW 协议。假设有  $n$  个参与方  $P_1, \dots, P_n$ , 他们想要计算函数  $f : \mathbb{F}^{M_{\text{in}}^1} \times \dots \times \mathbb{F}^{M_{\text{in}}^n} \rightarrow \mathbb{F}^{M_{\text{out}}^1} \times \dots \times \mathbb{F}^{M_{\text{out}}^n}$ , 其中  $P_i$  有  $M_{\text{in}}^i$  个输入和  $M_{\text{out}}^i$  个输出。 $C$  是函数  $f$  对应的算术电路。协议的流程如下。

**加法秘密分享。**对于  $P_i$  的每个输入  $x_{i,k} \in \mathbb{F}$ ,  $k \in [M_{\text{in}}^i]$ ,  $P_i$  选择  $n$  个域上的随机元素  $r_{i,k}^1, \dots, r_{i,k}^n$  满足  $\sum_{\ell=1}^n r_{i,k}^\ell = x_{i,k}$ , 然后将  $r_{i,k}^\ell$  发送给  $P_\ell$  作为秘密份额。每个参与方都以这样的方式进行秘密分享。可以看出, 这样的加法秘密分享是  $(n-1, n)$ -门限秘密分享 (见第2.3.1节的定义)。

逐门计算。考虑具有输入导线  $w_i, w_j$  以及输出导线  $w_k$  的门  $G$ , 每一根输入导线  $w_x$  ( $x \in \{i, j\}$ ) 的值都被秘密分享为  $s_x^1, \dots, s_x^n$ , 满足  $s_x^1 + \dots + s_x^n = w_x$ .  $P_\ell$  持有  $w_i$  和  $w_j$  上的  $s_i^\ell$  和  $s_j^\ell$ , 协议需要使得门  $G$  计算完成后,  $P_\ell$  持有  $w_k$  的秘密份额  $s_k^\ell$ . 不失一般性地, 假设  $\mathcal{C}$  由加法门、常数乘法门、常数加法门、乘法门组成。

加法门。对于加法门, 我们有

$$w_i + w_j = (s_i^1 + \dots + s_i^n) + (s_j^1 + \dots + s_j^n) = (s_i^1 + s_j^1) + \dots + (s_i^n + s_j^n)$$

也就是说, 参与方只需要将自己的秘密份额在本地相加, 即  $P_\ell$  设置  $s_k^\ell = s_i^\ell + s_j^\ell$ .

常数乘法门。设常数乘法门对应的常数为  $\alpha$ . 常数乘法门只有一个输入  $w_i$ , 输出为  $w_k = c \cdot w_i$ . 我们有

$$\alpha \cdot w_i = \alpha \cdot (s_i^1 + \dots + s_i^n) = \alpha \cdot s_i^1 + \dots + \alpha \cdot s_i^n$$

也就是说, 参与方只需要将自己的秘密份额在本地乘以常数  $\alpha$ , 即  $P_\ell$  设置  $s_k^\ell = \alpha \cdot s_i^\ell$ .

常数加法门。设常数加法门对应的常数为  $\alpha$ . 常数加法门只有一个输入  $w_i$ , 输出为  $w_k = w_i + \alpha$ . 我们有

$$w_i + \alpha = (s_i^1 + \dots + s_i^n) + \alpha = (s_i^1 + \alpha) + s_i^2 + \dots + s_i^n$$

也就是说, 参与方  $P_1$  将自己的秘密份额设置为  $s_k^1 = s_i^1 + \alpha$ , 其他参与方  $P_\ell$ ,  $\ell \neq 1$  设置  $s_k^\ell = s_i^\ell$ . 乘法门。乘法门的计算需要参与方的交互。对于乘法门, 我们有

$$w_k = w_i w_j = (s_i^1 + \dots + s_i^n)(s_j^1 + \dots + s_j^n) = \left( \sum_{\ell=1}^n s_i^\ell s_j^\ell \right) + \left( \sum_{\ell_1 \neq \ell_2} s_i^{\ell_1} s_j^{\ell_2} \right)$$

这里的想法与计算布尔电路与门非常相似。首先,  $P_\ell$  可以本地计算  $s_i^\ell s_j^\ell$ . 对于  $s_i^{\ell_1} s_j^{\ell_2}$ , 需要  $P_{\ell_1}$  和  $P_{\ell_2}$  的交互。具体来说,  $P_{\ell_1}$  选择均匀随机数  $r_{\ell_1, \ell_2} \leftarrow \mathbb{F}$ , 然后  $P_{\ell_1}$  和  $P_{\ell_2}$  运行 OLE 协议,  $P_{\ell_1}$  作为发送方以  $s_i^{\ell_1}$  和  $-r_{\ell_1, \ell_2}$  作为输入,  $P_{\ell_2}$  作为接收方以  $s_j^{\ell_2}$  作为输入, OLE 运行结束后,  $P_{\ell_2}$  得到  $\gamma_{\ell_1, \ell_2} = s_i^{\ell_1} s_j^{\ell_2} - r_{\ell_1, \ell_2}$ . 此时  $\gamma_{\ell_1, \ell_2}$  和  $r_{\ell_1, \ell_2}$  就构成了  $s_i^{\ell_1} s_j^{\ell_2}$  的加法秘密分享。最后,  $P_\ell$  设置  $s_k^\ell = s_i^\ell s_j^\ell + \sum_{\ell' \neq \ell} (r_{\ell, \ell'} + \gamma_{\ell', \ell})$  为  $w_k$  的秘密份额。由计算过程可知, 这样得到的秘密份额满足  $w_k = s_k^1 + \dots + s_k^n$ .

输出重建。对于  $P_i$  的每一个输出  $y_{i,k}$ , 其他参与方将自己的份额  $s_{i,k}^\ell$ ,  $\ell \neq i$  发给  $P_i$ , 然后  $P_i$  计算  $y_{i,k} = s_{i,k}^1 + \dots + s_{i,k}^n$ . 每一个输出都以这样的方式重建。

最后, 完整的算术电路 GMW 协议如图 6.6 所示。

### 6.2.3.2 安全性证明

不难发现, 假设  $t \leq n - 1$ , 在  $\mathcal{F}_{\text{OLE}}$  混合模型下, 算术电路 GMW 协议  $\Pi_{\text{GMW}}^{\text{arith}}$  对于静态半诚实敌手 UC-安全实现了安全函数计算的理想功能  $\mathcal{F}_{\text{sfe}}$ . 分析思路与布尔电路相同: 首先, 协议构造满足正确性。对于隐私性, 在秘密分享和计算乘法门时, 被攻陷方看到的是均匀随机值; 在输出重建时, 根据协议输出和被攻陷方的份额可以模拟诚实方的份额, 其分布与真实协议中完全相同。

**定理 6.3.** 假设被攻陷方数量不超过  $n - 1$ , 图 6.6 中的协议  $\Pi_{\text{GMW}}^{\text{arith}}$  在  $\mathcal{F}_{\text{OLE}}$ -混合模型下对于静态半诚实敌手 UC-安全实现了理想功能  $\mathcal{F}_{\text{sfe}}$  (图 6.1)。

此定理的证明与定理 6.2 完全类似, 不再赘述。

### 算术电路 GMW 协议 $\Pi_{\text{GMW}}^{\text{arith}}$

协议分为三个阶段：输入分享阶段、计算阶段和输出重建阶段。参与方  $P_1, \dots, P_n$  想要计算函数  $f : \mathbb{F}^{M_{\text{in}}^1} \times \dots \times \mathbb{F}^{M_{\text{in}}^n} \rightarrow \mathbb{F}^{M_{\text{out}}^1} \times \dots \times \mathbb{F}^{M_{\text{out}}^n}$  并已经将其转化成了包含加法门、常数乘法门、乘法门的算术电路。

**输入分享阶段。**对于  $P_i$  的每个输入  $x_{i,k} \in \mathbb{F}$ ,  $k \in [M_{\text{in}}^i]$ ,  $P_i$  选择域上的随机元素  $r_{i,k}^1, \dots, r_{i,k}^n$  满足  $\sum_{\ell=1}^n r_{i,k}^\ell = x_{i,k}$ , 然后将  $r_{i,k}^\ell$  发给  $P_j$  作为秘密份额。

**计算阶段。**重复以下步骤，直到所有门都被计算。根据门的类型，执行以下操作之一：

- 加法门。假设输入导线上的值为  $w_i, w_j$ , 输出导线上的值为  $w_k$ . 参与方  $P_\ell$  持有的秘密份额为  $w_i^\ell, w_j^\ell$ .  $P_\ell$  设置  $s_k^\ell = s_i^\ell + s_j^\ell$  为输出导线上的份额。
- 常数乘法门。假设对应的常数为  $\alpha$ , 输入导线上的值为  $w_i$ , 输出导线上的值为  $w_k$ . 参与方  $P_\ell$  持有的秘密份额为  $w_i^\ell$ .  $P_\ell$  设置  $s_k^\ell = \alpha \cdot s_i^\ell$  为输出导线上的份额。
- 常数加法门。假设对应的常数为  $\alpha$ , 输入导线上的值为  $w_i$ , 输出导线上的值为  $w_k$ . 参与方  $P_\ell$  持有的秘密份额为  $w_i^\ell$ .  $P_1$  设置输出导线上的份额为  $s_k^1 = s_i^1 + \alpha$ , 其他参与方  $P_\ell$ ,  $\ell \neq 1$  设置输出导线上的份额为  $s_k^\ell = s_i^\ell$ .
- 乘法门。假设输入导线上的值为  $w_i, w_j$ , 输出导线上的值为  $w_k$ . 参与方  $P_\ell$  持有的秘密份额为  $w_i^\ell, w_j^\ell$ . 参与方执行以下操作：
  1. 对于所有的  $\ell_1 \in [n], \ell_2 \in [n] \setminus \{\ell_1\}$ ,  $P_{\ell_1}$  与  $P_{\ell_2}$  执行如下操作:  $P_{\ell_1}$  均匀随机选择  $r_{\ell_1, \ell_2} \leftarrow \mathbb{F}$ .  $P_{\ell_1}$  和  $P_{\ell_2}$  运行 OLE 协议, 其中  $P_{\ell_1}$  作为发送方以  $s_i^{\ell_1}, -r_{\ell_1, \ell_2}$  作为输入,  $P_{\ell_2}$  作为接收方以  $s_j^{\ell_2}$  作为输入,  $P_{\ell_2}$  得到  $\gamma_{\ell_1, \ell_2} = s_i^{\ell_1} s_j^{\ell_2} - r_{\ell_1, \ell_2}$ .
  2.  $P_\ell$  设置  $s_k^\ell = s_i^\ell s_j^\ell + \sum_{\ell' \neq \ell} (\gamma_{\ell, \ell'} + \gamma_{\ell', \ell})$  为输出导线上的份额。

**输出重建阶段。**对于  $P_i$  的每个输出  $y_{i,k} \in \mathbb{F}$ , 其他参与方将自己的份额  $s_{i,k}^\ell$ ,  $\ell \neq i$  发给  $P_i$ ,  $P_i$  计算  $y_{i,k} = s_{i,k}^1 + \dots + s_{i,k}^n$ .

图 6.6: 算术电路 GMW 协议  $\Pi_{\text{GMW}}^{\text{arith}}$

# Chapter 7

## 通过 Beaver 三元组实现 MPC

无论是 BGW 协议、GMW 协议，还是第1章所介绍的协议，可以发现：线性门（加法门、常数乘法门、常数加法门、异或门、非门）的开销很小，参与方仅需在本地进行计算即可；而非线性门（乘法门、与门）的开销很大，参与方需要两两进行交互，并调用较为复杂的子协议（例如 OT、OLE）。

为了降低非线性门的计算开销，Beaver 三元组 (Beaver triple)<sup>[35]</sup>是一种被广泛采用的实用技术。它的基本思想是：将 MPC 协议分为在线阶段和离线阶段（预处理阶段），离线阶段生成与输入无关的相关随机数（指具有特定关系的随机值，即 Beaver 三元组），在线阶段通过消耗这些相关随机数来加速乘法门的计算。这样的设计模式可以将大部分开销转移到离线阶段，从而大大提升在线阶段的效率。

下面，本章的介绍采用模块化的方法：首先，将 Beaver 三元组的生成抽象为理想功能  $\mathcal{F}_{\text{triple}}$ ，在  $\mathcal{F}_{\text{triple}}$ -混合模型下构造高效的协议并进行安全性证明。然后，我们讨论如何生成 Beaver 三元组，即如何实现  $\mathcal{F}_{\text{triple}}$ 。

### 7.1 基于 Beaver 三元组的 MPC 协议

#### 7.1.1 Beaver 三元组

Beaver 三元组由 Beaver 提出<sup>[35]</sup>，它指的是秘密份额集三元组  $([a], [b], [c])$ ，其中  $a, b$  为有限域  $\mathbb{F}$  上均匀随机选取的元素，满足  $c = ab$ .

为了叙述的简洁，我们先定义一些方便的符号来表示加法秘密分享的计算。我们用  $[x]$  表示秘密  $x \in \mathbb{F}$  的秘密份额集，即  $[x] = (x_1, \dots, x_n)$ ，满足  $x = x_1 + \dots + x_n$ . 我们采用与 BGW 协议类似的术语：参与方  $P_i$  分发  $x$ ，指的是  $P_i$  均匀随机选择  $x_1, \dots, x_n$  满足  $x = x_1 + \dots + x_n$ ，然后将  $x_\ell$  发送给  $P_\ell$ . 当参与方获得了  $x$  的份额时，我们说参与方持有  $[x]$ .

对于常数  $\alpha \in \mathbb{F}$ ，我们定义如下的逐项加法、常数乘法和常数加法：

$$\begin{aligned}[x] + [y] &= (x_1 + y_1, \dots, x_n + y_n) \\ \alpha[x] &= (\alpha x_1, \dots, \alpha x_n) \\ [x] + \alpha &= (x_1 + \alpha, x_2, \dots, x_n)\end{aligned}$$

很容易发现，加法秘密分享满足如下的性质：

$$[x] + [y] = [x + y]$$

$$\alpha[x] = [\alpha x]$$

$$[x] + \alpha = [x + \alpha]$$

让我们看看 Beaver 三元组是如何实现乘法门的计算的。假设乘法门的输入  $x, y$  已经被秘密分享在  $n$  个参与方之中，每个参与方还持有 Beaver 三元组  $([a], [b], [c])$  中自己对应的份额，即参与方持有  $([x], [y], [a], [b], [c])$ ，其中  $x, y$  是乘法门的输入， $([a], [b], [c])$  是 Beaver 三元组。

乘法门的计算分为如下三个步骤：

1. 参与方本地计算  $[x] - [a] = [x - a], [y] - [b] = [y - b]$ .
2. 对上一步计算的份额进行秘密重建，所有参与方获得  $d = x - a, e = y - b$ .
3. 参与方本地计算  $d[b] + e[a] + [c] + de$ ，作为输出导线的份额。

首先，这样计算获得的结果是正确的。根据加法秘密分享的性质，我们有  $[xy] = [(d + a)(e + b)] = [de + db + ea + ab] = d[b] + e[a] + [c] + de$ . 因此， $d[b] + e[a] + [c] + de$  就是  $xy$  的秘密份额集。关于隐私性，协议的第 1 和第 3 步都是本地计算，而在第 2 步中，参与方进行了秘密重建得到了  $d = x - a, e = y - b$ . 我们需要确保  $x, y$  不被泄露，协议确实满足了这一点，因为  $a, b$  都是均匀随机值，因此  $d, e$  也是均匀随机值，不会泄露任何信息。

与同样采用加法秘密分享的 GMW 协议相比，通过 Beaver 三元组实现的乘法门计算只需对两个值进行秘密重建，在半诚实模型下可以让所有参与方将秘密份额发给  $P_1$ ， $P_1$  再将重建结果发给每个参与方，只需  $O(n)$  的通信量。而 GMW 协议中乘法门的计算需要参与方两两执行 OLE，共  $O(n^2)$  个 OLE. 由此可见，Beaver 三元组显著降低了乘法门所需的通信量和计算量。

### 7.1.2 协议描述

根据以上计算乘法门的方法，我们很容易构造基于 Beaver 三元组的 MPC 协议，其加法门、常数乘法门以及输出重建的过程与 GMW 协议完全相同。完整的协议如图 7.1 所示。

### 7.1.3 安全性证明

我们在  $\mathcal{F}_{\text{triple}}$ -混合模型下证明，上述协议对于静态半诚实敌手 UC-安全实现了安全函数计算的理想功能  $\mathcal{F}_{\text{sfe}}$ . 也就是说，我们假设参与方在协议的离线阶段通过请求  $\mathcal{F}_{\text{triple}}$  获得了 Beaver 三元组。 $\mathcal{F}_{\text{triple}}$  的定义如图 7.2 所示。

**定理 7.1.** 假设被攻陷方数量不超过  $n - 1$ ，图 7.1 中的协议  $\Pi$  在  $\mathcal{F}_{\text{triple}}$ -混合模型下对于静态半诚实敌手 UC-安全实现了理想功能  $\mathcal{F}_{\text{sfe}}$ （图 6.1）。

证明. 要证明该定理，我们需要构造模拟器  $\mathcal{S}$ ，使得对所有 PPT 的环境  $\mathcal{Z}$ ，都有

$$\text{EXEC}_{\Pi, \mathcal{A}, \mathcal{Z}}^{\mathcal{F}_{\text{triple}}} \approx \text{EXEC}_{\mathcal{F}_{\text{sfe}}, \mathcal{S}, \mathcal{Z}}$$

其中  $\mathcal{A}$  是平凡敌手。

### 基于 Beaver 三元组的 MPC 协议

协议分为三个阶段：输入分享阶段、计算阶段和输出重建阶段。参与方  $P_1, \dots, P_n$  想要计算函数  $f : \mathbb{F}^{M_{\text{in}}^1} \times \dots \times \mathbb{F}^{M_{\text{in}}^n} \rightarrow \mathbb{F}^{M_{\text{out}}^1} \times \dots \times \mathbb{F}^{M_{\text{out}}^n}$  并已经将其转化成了包含加法门、常数乘法门、常数加法门、乘法门的算术电路。参与方已经生成并持有  $m$  个 Beaver 三元组  $([a^k], [b^k], [c^k]), k \in [1, m]$ 。设电路的乘法门数量不超过  $m$ 。

**输入分享阶段。**对于  $P_i$  的每个输入  $x_{i,k} \in \mathbb{F}, k \in [M_{\text{in}}^i]$ ,  $P_i$  分发  $x_{i,k}$ .

**计算阶段。**重复以下步骤，直到所有门都被计算。根据门的类型，执行以下操作之一：

- 加法门。假设参与方持有  $[w_i], [w_j]$ . 参与方计算  $[w_i] + [w_j] = [w_i + w_j]$ .
- 常数乘法门。假设对应的常数为  $\alpha$ , 参与方持有  $[w_i]$ . 参与方计算  $\alpha[w_i] = [\alpha w_i]$ .
- 常数加法门。假设对应的常数为  $\alpha$ , 参与方持有  $[w_i]$ . 参与方计算  $[w_i] + \alpha = [w_i + \alpha]$ .
- 乘法门。假设参与方持有  $[w_i], [w_j]$  以及 Beaver 三元组  $([a], [b], [c])$ . 参与方执行以下操作：
  1. 参与方本地计算  $[w_i] - [a] = [w_i - a], [w_j] - [b] = [w_j - b]$ .
  2. 对上一步计算的份额进行秘密重建，所有参与方获得  $d = w_i - a, e = w_j - b$ .
  3. 参与方本地计算  $d[b] + e[a] + [c] + de = [w_i w_j]$ .

**输出重建阶段。**对于  $P_i$  的每个输出  $y_{i,k} \in \mathbb{F}$ , 其他参与方将自己的份额  $s_{i,k}^\ell, \ell \neq i$  发给  $P_i$ ,  $P_i$  计算  $y_{i,k} = s_{i,k}^1 + \dots + s_{i,k}^n$ .

图 7.1: 基于 Beaver 三元组的 MPC 协议

### Beaver 三元组生成的理想功能 $\mathcal{F}_{\text{triple}}$

理想功能  $\mathcal{F}_{\text{triple}}$  与参与方  $P_1, \dots, P_n$  交互。

**输入：**参与方没有输入。

理想功能  $\mathcal{F}_{\text{triple}}$  执行如下操作：

1. 随机选取  $a, b \leftarrow_{\$} \mathbb{F}$ , 计算  $c = ab$ .
2. 随机选取  $a_1, b_1, c_1, \dots, a_n, b_n, c_n$ , 满足  $a = a_1 + \dots + a_n, b = b_1 + \dots + b_n, c = c_1 + \dots + c_n$ .
3. 对所有  $i \in [n]$ , 将  $a_i, b_i, c_i$  发送给  $P_i$ .

图 7.2: Beaver 三元组生成的理想功能  $\mathcal{F}_{\text{triple}}$

模拟器  $\mathcal{S}$ . 模拟器  $\mathcal{S}$  在内部运行  $\mathcal{A}$ , 转发  $\mathcal{Z}$  与  $\mathcal{A}$  之间的消息。此外， $\mathcal{S}$  模拟  $\mathcal{F}_{\text{triple}}$  诚实地生成 Beaver 三元组。令  $C$  为被攻陷方的集合。 $\mathcal{S}$  的工作方式如下。

- 当从  $\mathcal{F}_{\text{sfe}}$  收到  $(\text{INPUT}, \text{sid}, P_i)$  时, 如果  $P_i \in C$ , 那么  $P_i$  是被攻陷的,  $\mathcal{S}$  可以得到  $P_i$  的输入

$(x_{i,1}, \dots, x_{i,M_{\text{in}}^i})$ .  $\mathcal{S}$  以  $(x_{i,1}, \dots, x_{i,M_{\text{in}}^i})$  为输入模拟  $P_i$  遵循协议执行输入分享。

- 当从  $\mathcal{F}_{\text{sfe}}$  收到  $(\text{INPUT}, \text{sid}, P_j)$  时, 如果  $P_j \notin C$ , 那么  $P_j$  是诚实的。 $\mathcal{S}$  设置  $P_j$  的输入为  $(x_{j,1}, \dots, x_{j,M_{\text{in}}^j}) = (0, \dots, 0)$ , 然后模拟  $P_j$  遵循协议执行输入分享, 将秘密份额发给被攻陷方。
- 计算阶段,  $\mathcal{S}$  模拟诚实方和被攻陷方遵循协议执行。(也就是说,  $\mathcal{S}$  模拟诚实方以  $(0, \dots, 0)$  为输入执行协议)
- 输出重建阶段, 重建  $P_i$  的输出时,  $\mathcal{S}$  发送  $(\text{OUTPUT}, \text{sid}, P_i)$  给  $\mathcal{F}_{\text{sfe}}$ . 如果  $P_i$  是被攻陷的,  $\mathcal{S}$  将得到  $P_i$  的输出  $(y_{i,1}, \dots, y_{i,M_{\text{out}}^i})$ . 对于每个输出值  $y_{i,k}$ ,  $\mathcal{S}$  均匀随机地选择诚实方的份额  $s_{i,k}^\ell$ ,  $P_\ell \notin C$ , 使得  $s_{i,k}^1 + \dots + s_{i,k}^n = y_{i,k}$ . 然后将诚实方的份额发送给  $P_i$ .

**不可区分性。**真实世界和理想世界的不同在于: (1) 理想世界的诚实方输入为  $(0, \dots, 0)$ ; (2) 输出重建阶段, 理想世界的诚实方份额是  $\mathcal{S}$  模拟的。这对于环境  $\mathcal{Z}$  是不可区分的, 因为: 加法秘密分享的隐私性确保了秘密份额与诚实方的输入无关; 计算乘法门时, 因为 Beaver 三元组分享的秘密是均匀随机的, 所以秘密重建得到的  $d = w_i - a$ ,  $e = w_j - b$  是均匀随机值; 最后, 输出重建阶段  $\mathcal{S}$  模拟的份额与真实世界中诚实方的份额也具有相同的分布。因此, 真实世界和理想世界是不可区分的。

□

## 7.2 Beaver 三元组的生成

本节, 我们讨论一下 Beaver 三元组的生成方式。

### 7.2.1 基于可信第三方的方式

Beaver 三元组  $([a], [b], [c])$  本质上是一种满足  $c = ab$  的相关随机数 (Correlated Randomness), 它与协议的输入是无关的。因此, 在有可信第三方的场景下, 可以由可信第三方执行理想功能  $\mathcal{F}_{\text{triple}}$  的操作, 将 Beaver 三元组分发给参与方。这是一种非常高效的 Beaver 三元组生成方式。具体而言, 第三方执行以下操作:

1. 随机选取  $a, b \leftarrow_{\$} \mathbb{F}$ , 计算  $c = ab$ .
2. 随机选取  $a_1, b_1, c_1, \dots, a_n, b_n, c_n$ , 满足  $a = a_1 + \dots + a_n$ ,  $b = b_1 + \dots + b_n$ ,  $c = c_1 + \dots + c_n$ .
3. 对所有  $i \in [n]$ , 将  $a_i, b_i, c_i$  发送给  $P_i$ .

注意, 这里的可信第三方需要满足: (1) 他是半诚实的; (2) 他不会与参与方串通。假设 1 保证了生成的 Beaver 三元组是正确的; 而如果第三方与参与方互相串通, 由于第三方知道  $a, b, c$ , 计算乘法门时需要重建  $d = x - a$ ,  $e = y - b$ , 那么乘法门的输入  $x, y$  显然会被泄露。如果上述两个假设成立, 那么协议是安全的, 且可信第三方无法得到参与方的输入, 因此这与“参与方直接将输入交给可信第三方”是不同的。

### 7.2.2 分布式生成方式

在某些场景下，如果找不到半可信的第三方，那么就需要各个参与方分布式地生成 Beaver 三元组。在 Beaver 三元组  $([a], [b], [c])$  中， $a, b$  是均匀随机选取的，也就是说，参与方需要通过某种方式秘密分享一个均匀随机值，使得没有人知道被分享的随机值是什么。这与之前的情况不同，之前的秘密分享过程中，都存在一个秘密的持有者，它知道秘密并完成秘密份额的计算和分发。

这个问题的解决方法非常简单：只需让每个参与方  $P_i$  在本地均匀随机选择  $a_i \leftarrow_{\$} \mathbb{F}$  作为秘密份额，然后定义  $a = a_1 + \dots + a_n$ 。首先，只需有一个诚实方，那么  $a$  就是均匀随机的；其次，根据  $a$  的定义，秘密份额集  $\{a_i\}$  构成了  $a$  的正确的加法秘密分享。可以看出，参与方通过随机选择自己的份额隐式地定义了随机值  $a$ ，而没有参与方可以知道  $a$  的值。同样地，参与方  $P_i$  均匀随机选择  $b_i \leftarrow_{\$} \mathbb{F}$  作为秘密份额，并且定义  $b = b_1 + \dots + b_n$ 。

接下来，参与方还需要计算  $c = ab$  的秘密份额。这实质上就是一个乘法门的计算，参与方可以通过调用 GMW 协议的乘法门计算协议，在  $\mathcal{F}_{OLE}$ -混合模型下实现。注意到，每个乘法门需要消耗一个 Beaver 三元组，而分布式地计算一个 Beaver 三元组实质上就是计算了一个乘法门，显然，这样的方法无法降低协议的总开销。但是请读者注意，Beaver 三元组是与协议输入无关的，因此参与方可以提前生成大量的 Beaver 三元组，将绝大部分开销转移到离线阶段，从而大大提升在线阶段的效率。

更具体而言，生成 Beaver 三元组的方式包括：基于 OT 的方式<sup>[36-37]</sup>，基于同态加密的方式<sup>[38]</sup>等，它们本质上是高效地实现了底层的 OLE 协议。相关随机数的高效生成仍是一个广受关注的研究方向，我们将其作为读者的拓展阅读内容，这里不再展开。



# Chapter 8

## 基于混淆电路的协议

与基于秘密分享的方案不同，混淆电路是安全多方计算的另一种经典方法。它最早由图灵奖得主姚期智院士在 1986 年提出，该协议被称为姚氏混淆电路协议<sup>[39]</sup> (Yao's Garbled Circuits protocol, Yao's GC)。它是第一个通用两方安全计算协议，并成为了安全多方计算迅速发展的开端。

本章，我们首先介绍姚氏两方混淆电路协议。在进行安全性分析时，我们会发现姚氏混淆电路协议的原始版本并不满足通用可组合安全。因此，我们将介绍独立 (stand-alone) 模型，并介绍独立模型与通用可组合模型之间的关系，然后在独立模型下对姚氏混淆电路协议进行安全性证明。随着安全多方计算的发展，研究者提出了许多混淆电路的优化构造，本章我们也将介绍其中的几个经典方案。

最后，在第8.2节，我们介绍 BMR 协议<sup>[40]</sup>，它将姚氏混淆电路协议拓展到多方的场景，是基于混淆电路的多方计算方案。

### 8.1 姚氏混淆电路协议

#### 8.1.1 直观思想

计算一个与门。我们先考虑一个最简单的情况。 $P_1$  和  $P_2$  分别拥有一比特的输入  $x$  和  $y$ ,  $x, y \in \{0, 1\}$ . 他们想要计算  $z = x \wedge y$ , 即计算一个与门 (例如, 第1.1.3节中的配对场景)。那么, 输入和输出满足如表 8.1所示的真值表。

$x$	$y$	$z = x \wedge y$
0	0	0
0	1	0
1	0	0
1	1	1

表 8.1: 与门的真值表

参与双方不能泄露自己的输入，因此不能直接把输入发给对方。混淆电路的直观思想是：为每个输入的每个可能取值选择一个随机密钥，用对应的密钥加密输出值。具体而言，对于  $x$  的可能取值 0 和 1，随机选择两个密钥  $k_X^0$  和  $k_X^1$ ；同样地，对于  $y$  的可能取值 0 和 1，随机选择两个密钥  $k_Y^0$  和  $k_Y^1$ . 然

后根据真值表，用对应的密钥加密输出  $z$ . 于是，我们得到如表 8.2所示的加密后的真值表。

$x$ 的对应密钥	$y$ 的对应密钥	加密的 $z$
$k_X^0$	$k_Y^0$	$\text{Enc}_{k_X^0}(\text{Enc}_{k_Y^0}(0))$
$k_X^0$	$k_Y^1$	$\text{Enc}_{k_X^0}(\text{Enc}_{k_Y^1}(0))$
$k_X^1$	$k_Y^0$	$\text{Enc}_{k_X^1}(\text{Enc}_{k_Y^0}(0))$
$k_X^1$	$k_Y^1$	$\text{Enc}_{k_X^1}(\text{Enc}_{k_Y^1}(1))$

表 8.2: 与门加密后的真值表

不妨假设  $P_1$  为每个输入随机选取了密钥，然后计算了表  $T = [\text{Enc}_{k_X^0}(\text{Enc}_{k_Y^0}(0)), \text{Enc}_{k_X^0}(\text{Enc}_{k_Y^1}(0)), \text{Enc}_{k_X^1}(\text{Enc}_{k_Y^0}(0)), \text{Enc}_{k_X^1}(\text{Enc}_{k_Y^1}(1))]$ .  $P_1$  将表  $T$  打乱顺序（也就是进行随机排列）得到  $T'$ ，并将  $T'$  发送给  $P_2$ . 那么，如果  $P_2$  拥有了  $x$  和  $y$  的一组密钥，那么他就可以正确解密  $T'$  中对应的那项，得到输出  $z$ ；而对于  $T'$  中的其他项，解密结果是乱码（或错误）。

现在我们面临的问题就是：如何让  $P_2$  拥有输入  $x$  和  $y$  所对应的密钥？显然， $P_1$  知道自己的输入  $x$ ，如果  $x = 0$ ，那就把  $k_X^0$  发送给  $P_2$ ，如果  $x = 1$ ，那就发送  $k_X^1$ . 因为  $k_X^0$  和  $k_X^1$  都是随机选的值，这一步不会泄露输入  $x$ . 但是， $P_1$  不知道  $P_2$  的输入  $y$ ，要怎么让  $P_2$  得到  $y$  的对应密钥呢？方法是运行 OT 协议（其定义和构造参见第2.3.7节和第5章）， $P_1$  作为发送方，以  $k_Y^0, k_Y^1$  作为输入； $P_2$  作为接收方，以  $y$  作为输入。那么，协议结束后  $P_2$  就得到了  $y$  的对应密钥  $k_Y^y$ ，而  $P_1$  不知道  $P_2$  的输入  $y$ ， $P_2$  也不知道  $y$  的另一个密钥。最后， $P_2$  尝试解密  $T'$  中的每一项，正确解密的结果即为  $z$ .  $P_2$  再将  $z$  发送给  $P_1$  以完成协议。

不难看出，这样构造的协议满足正确性。让我们分析一下隐私性。从  $P_1$  的视角看，他只在 OT 协议中接收了  $P_2$  的消息（除去最后一步接收协议的输出  $z$ ），而 OT 协议的安全性保证了  $P_1$  不知道  $P_2$  的输入  $y$ . 从  $P_2$  的视角看，他接收到了表  $T'$  以及  $x$  的对应密钥，并且他从 OT 协议中获得了  $y$  的对应密钥。刚才分析过， $x$  的对应密钥不会泄露  $x$ ，OT 协议确保了  $P_2$  只能获得  $y$  的一个密钥，那么， $P_2$  只能正确解密  $T'$  中的一项，加密方案的安全性保证了  $P_2$  不能从  $T'$  中的另外三项获得任何信息<sup>1</sup>。

**将函数表示为查找表。**基于上述思想，我们可以将任意函数表示为查找表（真值表），然后为每一个可能的输入值选取随机密钥，将输出值加密。具体而言，假设参与方  $P_1$  和  $P_2$  想要计算函数  $z = f(x, y)$ ，其中  $P_1$  的输入为  $x$ ， $P_2$  的输入为  $y$ . 设  $x$  的取值范围为  $X$ ,  $y$  的取值范围为  $Y$ . 对于每一个可能的取值  $x \in X$ ,  $P_1$  选取随机密钥  $k_X^x$ ; 同样地，对于每一个可能的取值  $y \in Y$ , 选取随机密钥  $k_Y^y$ . 然后，用  $k_X^x$  和  $k_Y^y$  加密  $f(x, y)$ ，即  $\text{Enc}_{k_X^x}(\text{Enc}_{k_Y^y}(f(x, y)))$ .  $P_1$  将这张表（大小为  $|X| \cdot |Y|$ ）随机排列后发送给  $P_2$ ，并且将自己的输入  $x$  对应的密钥  $k_X^x$  一并发送，然后双方执行 1-out-of- $|Y|$  OT 协议使  $P_2$  获得输入  $y$  所对应的密钥。最后， $P_2$  尝试解密表中的每一项<sup>2</sup>得到输出  $z$ .

**降低查找表的大小。**上述朴素的方案只有当  $x, y$  的取值范围较小时才可行，如果  $x, y$  的取值范围较大，例如， $x, y$  的长度均为 64 比特，那么加密后的查找表的大小为  $|X| \cdot |Y| = 2^{64} \cdot 2^{64} = 2^{128}$ . 计算这么大的查找表显然是不可行的。

如何有效地降低查找表的大小呢？这里的思路是：我们可以不通过一张（巨大的）查找表来计算函数  $f$ ，而是将  $f$  转化为布尔电路，对布尔电路的每一个门计算一张（小）查找表。

<sup>1</sup>这里我们稍微有一点不严谨，我们说对  $T'$  的另外三项进行解密会得到乱码或错误，但加密方案的定义并不天然地保证这一点。之后的章节会有这个性质的形式化定义。

<sup>2</sup>这里我们暂时粗略地假设输出  $z$  的可能取值与解密其他项得到的乱码（或错误）是可以区分的。

让我们探讨一下如何让上述思路具体落地。也就是说，我们将函数  $f$  转化为了布尔电路，那么对于布尔电路的每一个门，应该如何构建查找表呢？注意，这里的情况与刚才有所不同。此时，查找表的输出应该是下一个布尔门的输入，而不是函数  $f$  的输出，也就是计算的中间结果，因此它也不能被泄露。方案不难想象：我们为布尔电路的每一条导线  $w_i$  随机选取密钥  $k_i^0, k_i^1$ ，分别代表值为 0 和 1。对于输入导线为  $w_a, w_b$ ，输出导线为  $w_c$  的布尔门，根据布尔门的真值表使用  $w_a, w_b$  对应的密钥来加密  $w_c$  的密钥。例如，对于与门而言，查找表如表 8.3 所示。

$w_a$ 的对应密钥	$w_b$ 的对应密钥	$w_c$ 的对应密钥	查找表
$k_a^0$	$k_b^0$	$k_c^0$	$\text{Enc}_{k_a^0}(\text{Enc}_{k_b^0}(k_c^0))$
$k_a^0$	$k_b^1$	$k_c^0$	$\text{Enc}_{k_a^0}(\text{Enc}_{k_b^1}(k_c^0))$
$k_a^1$	$k_b^0$	$k_c^0$	$\text{Enc}_{k_a^1}(\text{Enc}_{k_b^0}(k_c^0))$
$k_a^1$	$k_b^1$	$k_c^1$	$\text{Enc}_{k_a^1}(\text{Enc}_{k_b^1}(k_c^1))$

表 8.3: 与门的查找表

基于这样的构造，如果拥有了输入导线的一组密钥，就可以正确解密输出导线的对应密钥，然后再用这个密钥解密下一个门，从而计算整个电路。

**哪个解密结果是正确的？** 细心的读者可能会发现：当尝试解密（随机排列后的）查找表的每一项时，得到的可能是正确的密钥，也可能是乱码。但是，正确的密钥也是随机选择的值，如何确定哪个解密结果是正确的密钥呢？

一个简单的方法是：在密钥后面添加  $n$  个 0 再进行加密，其中  $n$  为安全参数。那么，对于输入导线为  $w_a, w_b$ ，输出导线为  $w_c$  的与门，查找表如表 8.4 所示。

$w_a$ 的对应密钥	$w_b$ 的对应密钥	$w_c$ 的对应密钥	查找表
$k_a^0$	$k_b^0$	$k_c^0$	$\text{Enc}_{k_a^0}(\text{Enc}_{k_b^0}(k_c^0    0^n))$
$k_a^0$	$k_b^1$	$k_c^0$	$\text{Enc}_{k_a^0}(\text{Enc}_{k_b^1}(k_c^0    0^n))$
$k_a^1$	$k_b^0$	$k_c^0$	$\text{Enc}_{k_a^1}(\text{Enc}_{k_b^0}(k_c^0    0^n))$
$k_a^1$	$k_b^1$	$k_c^1$	$\text{Enc}_{k_a^1}(\text{Enc}_{k_b^1}(k_c^1    0^n))$

表 8.4: 与门的查找表（添加  $n$  个 0）

这样一来，解密其他项得到的乱码的最后  $n$  位恰好都是 0 的概率是可忽略的，我们可以确定：当解密结果的最后  $n$  位都是 0 时得到的就是正确的密钥。不过，这个解决方案是较为低效的，我们在第 8.1.6 节将介绍更高效的方案。

**协议描述（非形式化）。** 至此，我们可以总结一下姚氏混淆电路协议的流程。假设参与方  $P_1$  和  $P_2$  分别有  $n$  比特的输入  $x$  和  $y$ ， $x, y \in \{0, 1\}^n$ 。他们想要计算函数  $f(x, y)$ ，其输出也是  $n$  比特，即  $f : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}^n$ 。函数  $f$  已经转换为了等价的布尔电路  $C$ 。

1.  $P_1$  为  $C$  的每一条导线<sup>3</sup>  $w_i$  随机选取密钥  $k_i^0, k_i^1$ 。然后根据布尔门的真值表，为每个布尔门按照上述方式计算查找表并进行随机排列。

<sup>3</sup> 如果某条导线同时充当多个地方的输入（例如，同时是多个门的输入导线，或同时是某个门的两条输入导线），那么我们将其视为一条导线。

2.  $P_1$  将所有（随机排列后的）查找表发送给  $P_2$ . 对于每条输出导线， $P_1$  发送一张解密表，导线  $w_i$  的解密表包含  $(0, k_i^0)$  和  $(1, k_i^1)$ .<sup>4</sup>
3. 假设  $x$  对应的输入导线为  $w_1, \dots, w_n$ ,  $P_1$  将  $x$  对应的密钥  $k_1^{x_1}, \dots, k_n^{x_n}$  发送给  $P_2$ .
4. 假设  $y$  对应的输入导线为  $w_{n+1}, \dots, w_{2n}$ . 对  $i \in [n]$ ,  $P_1$  与  $P_2$  执行 1-out-of-2 OT, 其中  $P_1$  为发送方, 输入是  $(k_{n+i}^0, k_{n+i}^1)$ ,  $P_2$  是接收方, 输入是  $y_i$ .
5.  $P_2$  通过依次解密每一个布尔门来解密整个电路。当  $P_2$  获得输出导线上的密钥时, 根据解密表得到输出  $z = f(x, y)$ .  $P_2$  将  $z$  发送给  $P_1$ .

这里, 每个布尔门的（加密的）查找表被称为混淆表 (garbled table), 所有混淆表构成的整个电路被称为混淆电路 (garbled circuit)。

### 8.1.2 协议描述

根据上面的直观思想, 当  $P_2$  尝试解密混淆表时, 如果使用了对应的密钥, 那么一定会得到正确的结果; 而如果使用的不是对应的密钥, 那么实质上是用  $k'$  去解密用  $k$  加密的密文, 我们说此时会得到随机乱码。但是根据加密方案的定义 (参见第2.3.4节), 它需要满足正确性和选择明文攻击安全, 这并不能天然地保证用一个密钥解密另一个密钥加密的密文会得到随机乱码 (尽管大多数加密方案都满足这一点)。我们需要更形式化地定义这个性质。

本节的协议描述主要参照了 Lindell 和 Pinkas 的论文<sup>[41]</sup>。

**“特殊的”加密方案。**为了使  $P_2$  能够正确地计算混淆电路, 我们使用的加密方案需要满足两个额外的性质: (1) 用某个密钥加密的密文落在另一个密钥的密文空间的概率是可忽略的; (2) 已知密钥  $k$ , 可以高效地验证一个密文是否在  $k$  的密文空间内。这两个人性化的形式化定义如下。

**定义 8.1.1.** 令  $(\text{Gen}, \text{Enc}, \text{Dec})$  是一个对称加密方案。我们用  $\text{Range}_n(k) \stackrel{\text{def}}{=} \{\text{Enc}_k(x)\}_{x \in \{0,1\}^n}$  表示密钥  $k$  的密文空间。那么,

1. 如果对任意 PPT 的算法  $\mathcal{A}$ , 任意多项式  $p$ , 以及足够大的  $n$  都有

$$\Pr[k \leftarrow \text{Gen}(1^n), \mathcal{A}(1^n) \in \text{Range}_n(k)] < \frac{1}{p(n)}$$

那么, 我们称  $(\text{Gen}, \text{Enc}, \text{Dec})$  有不可知的密文空间 (*elusive range*)。

2. 如果存在 PPT 的算法  $M$  满足  $M(1^n, k, c) = 1$  当且仅当  $c \in \text{Range}_n(k)$ , 那么, 我们称  $(\text{Gen}, \text{Enc}, \text{Dec})$  有可验证的密文空间 (*efficiently verifiable range*)。

我们用  $\perp$  表示解密失败。对于任意的  $c \notin \text{Range}_n(k)$ , 我们有  $\text{Dec}_k(c) = \perp$ .

需要注意的是, 这两个人性化并不难实现。下面我们展示一个加密方案, 它满足以上两个人性化。

令  $\mathcal{F} = \{f_k\}$  是以  $k$  为密钥的伪随机函数, 对于  $k \in \{0,1\}^n$ ,  $f_k : \{0,1\}^n \rightarrow \{0,1\}^{2n}$  的输入为  $n$  比特, 输出为  $2n$  比特<sup>5</sup>。定义加密算法如下

<sup>4</sup>这里我们需要假设每条输出导线不是其他门的输入导线, 否则  $P_2$  得到输出导线的两个密钥后可能会计算额外信息。如果电路  $C$  不满足这个条件, 我们可以在  $z_i$  输出之前增加一个与门  $g_{\text{AND}}(z_i, a)$ , 其中  $a = 1$ . 这最多让电路大小增加  $n$ .

<sup>5</sup>简单来说, 当不知道密钥  $k$  时, 伪随机函数的输出与随机输出在计算上不可区分。伪随机函数可以基于单向函数 (one-way function) 构造。

$$\text{Enc}_k(x) = \langle r, f_k(r) \oplus x0^n \rangle$$

其中  $x \in \{0,1\}^n$ ,  $r \leftarrow \{0,1\}^n$ ,  $x0^n$  表示  $x$  与  $0^n$  的连接<sup>6</sup>。

我们简要分析一下这个加密方案的性质:

- 不可知的密文空间 (*elusive range*): 如果 PPT 的敌手  $\mathcal{A}$  输出一个密文  $c = \langle r, s \rangle$ , 它属于密文空间, 这意味着解密结果  $s \oplus f_k(r)$  的最后  $n$  位都是 0. 也就是说, 敌手  $\mathcal{A}$  可以选择  $r$ , 然后成功地预测  $f_k(r)$  的最后  $n$  位。因为  $f_k$  是伪随机函数, 这在计算上是不可行的。
- 可验证的密文空间 (*efficiently verifiable range*): 给定密钥  $k$  和密文  $c = \langle r, s \rangle$ , 只需验证  $s \oplus f_k(r)$  的最后  $n$  位是否均为 0, 如果是, 那么  $c \in \text{Range}_n(k)$ , 否则  $c \notin \text{Range}_n(k)$ .
- 选择明文攻击安全 (*IND-CPA secure*, 定义参见第 2.3.4 节): 对于两条不同的消息  $x_0, x_1$ , 因为  $f_k$  是伪随机函数, 所以  $f_k(r) \oplus x_00^n$  和  $f_k(r) \oplus x_10^n$  在计算上和随机数是不可区分的, 故该加密方案是 IND-CPA 安全的。

可见, 此加密方案满足定义 8.1.1 所定义的性质。

在之前的直观描述中, 混淆表的每一项都由两个密钥进行了双重加密。 $P_2$  拥有一组密钥, 他可以正确解密对应那一项。我们必须确保另外三项不会泄露关于明文的任何信息。直观上看, 这应该成立, 因为每一项  $P_2$  都至少缺失了一个密钥。但是, 我们需要形式化地证明这个性质。

与第 2.1.1 节中类似, 我们通过游戏 (Game) 来形式化地定义这个性质。令  $(\text{Gen}, \text{Enc}, \text{Dec})$  是一个对称加密方案。为了方便, 我们定义符号  $E(k_0, k_1, m) = \text{Enc}_{k_0}(\text{Enc}_{k_1}(m))$  表示双重加密。我们有如下的双重加密安全游戏。

双重加密安全游戏  $\text{Game}_{\mathcal{A}}^{\text{double}}(n)$ :

1. 敌手  $\mathcal{A}$  以  $1^n$  为输入, 输出两个长度为  $n$  的密钥  $k_0$  和  $k_1$  以及两个三元组  $(x_0, y_0, z_0)$  和  $(x_1, y_1, z_1)$ , 并发送给 Challenger.
2. Challenger 均匀随机地选择比特  $b \leftarrow_{\$} \{0,1\}$ , 并随机选取两个密钥  $k'_0, k'_1 \leftarrow \text{Gen}(1^n)$ .
3. Challenger 发送  $\langle E(k_0, k'_1, x_b), E(k'_0, k_1, y_b), E(k'_0, k'_1, z_b) \rangle$  给敌手  $\mathcal{A}$ .
4.  $\mathcal{A}$  可以访问  $E(\cdot, k'_1, \cdot)$  和  $E(k'_0, \cdot, \cdot)^a$ , 输出一个比特  $b'$ . 如果  $b' = b$ , 游戏的输出为 1, 表示  $\mathcal{A}$  成功; 否则, 输出 0.

---

<sup>a</sup>这里, 访问  $E(\cdot, k'_1, \cdot)$  指的是  $\mathcal{A}$  可以任意选择  $k$  和  $m$ , 得到  $E(k, k'_1, m)$ .  $E(k'_0, \cdot, \cdot)$  的定义也同理。

**定义 8.1.2.** 对称加密方案  $(\text{Gen}, \text{Enc}, \text{Dec})$  如果对任意概率多项式时间的敌手  $\mathcal{A}$ , 都存在一个可忽略函数  $\text{negl}$ , 使得

$$\Pr[\text{Game}_{\mathcal{A}}^{\text{double}}(n) = 1] \leq \frac{1}{2} + \text{negl}(n)$$

那么, 我们说  $(\text{Gen}, \text{Enc}, \text{Dec})$  在选择明文攻击下具有双重加密安全 (*secure under chosen double encryption*)。

---

<sup>6</sup>实际上, 0 的数量只需要达到  $\omega(\log n)$  即可 (super-logarithmic), 为了方便我们将它设置为  $n$ .

下面，我们证明任何选择明文攻击安全 (IND-CPA secure) 的加密方案都满足上面所定义的双重加密安全。它的证明需要用到以下结论：如果加密方案对一条消息满足选择明文攻击安全，那么对多条消息也满足选择明文攻击安全（证明可参见文献<sup>[10]</sup>, Section 5.4）。这里，我们给出两条消息的选择明文攻击安全游戏，因为这已经足以证明我们的引理。

对称加密方案 IND-CPA 游戏  $\text{Game}_{\mathcal{A}}^{\text{CPA}}(n)$ :

1. Challenger 随机选取密钥  $k \leftarrow \text{Gen}(1^n)$ .
2. 敌手  $\mathcal{A}$  以  $1^n$  为输入，可以访问  $\text{Enc}_k(\cdot)$ ，发送两对消息  $(x_0, y_0)$  和  $(x_1, y_1)$  给 Challenger.
3. Challenger 均匀随机地选择比特  $b \leftarrow_{\$} \{0, 1\}$ ，计算  $c_1 \leftarrow \text{Enc}_k(x_b)$ ,  $c_2 \leftarrow \text{Enc}_k(y_b)$ ，然后把  $(c_1, c_2)$  发给  $\mathcal{A}$ .
4.  $\mathcal{A}$  仍可以访问  $\text{Enc}_k(\cdot)$ ，输出比特  $b'$ . 如果  $b' = b$ ，游戏的输出为 1，表示  $\mathcal{A}$  成功；否则，输出 0.

**定义 8.1.3.** 对称加密方案  $(\text{Gen}, \text{Enc}, \text{Dec})$  如果对任意概率多项式时间的敌手  $\mathcal{A}$ ，都存在一个可忽略函数  $\text{negl}$ ，使得

$$\Pr[\text{Game}_{\mathcal{A}}^{\text{CPA}}(n) = 1] \leq \frac{1}{2} + \text{negl}(n)$$

那么，我们说  $(\text{Gen}, \text{Enc}, \text{Dec})$  在选择明文攻击下具有不可区分性 (IND-CPA secure)。

**引理 8.1.** 如果对称加密方案  $(\text{Gen}, \text{Enc}, \text{Dec})$  在选择明文攻击下具有不可区分性 (IND-CPA secure)，那么它在选择明文攻击下具有双重加密安全 (secure under chosen double encryption)。

**证明概要：**这里，我们仅给出证明概要，根据证明概要很容易写出完整的证明。基本思路仍然是通过一系列混合游戏 (hybrid game) 来展示：当  $b$  等于 0 或 1 时，敌手  $\mathcal{A}$  在  $\text{Game}_{\mathcal{A}}^{\text{double}}(n)$  中的视图是不可区分的。

假设加密方案是 IND-CPA 安全的。首先，我们定义  $\text{Game}_{\mathcal{A}}^{\text{mod}}(n)$ ，它与  $\text{Game}_{\mathcal{A}}^{\text{double}}(n)$  的不同之处在于无论  $b$  为何值， $y_0$  总是被加密，即  $\mathcal{A}$  总是收到  $\langle E(k_0, k'_1, x_b), E(k'_0, k_1, y_0), E(k'_0, k'_1, z_b) \rangle$ . 我们证明  $\text{Game}_{\mathcal{A}}^{\text{mod}}(n)$  与  $\text{Game}_{\mathcal{A}}^{\text{double}}(n)$  的视图是不可区分的：当  $b = 0$  时，两者完全相同；当  $b = 1$  时，这等价于区分  $E(k'_0, k_1, y_0)$  和  $E(k'_0, k_1, y_1)$ ，因为  $\mathcal{A}$  不知道  $k'_0$ ，根据单条消息的 IND-CPA 安全性， $\mathcal{A}$  无法区分它们。

接着，我们证明  $\mathcal{A}$  在游戏  $\text{Game}_{\mathcal{A}}^{\text{mod}}(n)$  获胜的概率不超过  $\frac{1}{2} + \text{negl}(n)$ . 这里，我们假设  $\mathcal{A}$  可以知道  $k'_0$ .<sup>7</sup> 因为  $\mathcal{A}$  总是收到  $E(k'_0, k_1, y_0)$ ，这对区分  $b = 0$  与  $b = 1$  没有任何帮助，所以我们将其移除。此时，不难发现，如果  $\mathcal{A}$  在  $\text{Game}_{\mathcal{A}}^{\text{mod}}(n)$  中获胜，那么他也能在  $\text{Game}_{\mathcal{A}}^{\text{CPA}}(n)$  中获胜（因为  $\mathcal{A}$  知道  $k_0, k'_0$ ，他可以将  $\langle E(k_0, k'_1, x_b), E(k'_0, k'_1, z_b) \rangle$  做一层解密，此时得到的就是  $\text{Game}_{\mathcal{A}}^{\text{CPA}}(n)$  中的视图）。由于加密方案是 IND-CPA 安全的，所以  $\mathcal{A}$  在  $\text{Game}_{\mathcal{A}}^{\text{mod}}(n)$  获胜的概率也不超过  $\frac{1}{2} + \text{negl}(n)$ .

综合以上两个结论可知，加密方案  $(\text{Gen}, \text{Enc}, \text{Dec})$  具有双重加密安全。  $\square$

<sup>7</sup>  $\mathcal{A}$  知道  $k'_0$  时结论成立，那么  $\mathcal{A}$  不知道  $k'_0$  时结论仍成立。

混淆电路的构造。在正式描述姚氏混淆电路协议之前，我们先形式化地介绍混淆电路的构造方式。令  $\mathcal{C}$  是一个布尔电路，它的输入是  $x, y \in \{0, 1\}^n$ ，输出是  $\mathcal{C}(x, y) \in \{0, 1\}^n$ 。我们假设  $\mathcal{C}$  的每条输出导线不会是其他门的输入导线（如果  $\mathcal{C}$  不满足这个条件，在输出前和 1 做与运算即可）。

对于  $\mathcal{C}$  中的每一个门  $g$ ，混淆表的构造方式如下。由于  $g$  是布尔门，所以它可以表示为函数  $g : \{0, 1\} \times \{0, 1\} \rightarrow \{0, 1\}$ <sup>8</sup>。设  $g$  的输入导线是  $w_1$  和  $w_2$ ，输出导线是  $w_3$ 。 $k_1^0, k_1^1, k_2^0, k_2^1, k_3^0, k_3^1$  是运行  $\text{Gen}(1^n)$  得到的密钥。为了简单起见，我们假设密钥的长度也是  $n$ 。我们希望计算方能够从  $k_1^\alpha, k_2^\beta$  计算出  $k_3^{g(\alpha, \beta)}$ ，并且不能获得关于  $k_3^{g(1-\alpha, \beta)}, k_3^{g(\alpha, 1-\beta)}, k_3^{g(1-\alpha, 1-\beta)}$  的任何信息。因此，门  $g$  由以下四个值构成

$$\begin{aligned} c_{0,0} &= \text{Enc}_{k_1^0}(\text{Enc}_{k_2^0}(k_3^{g(0,0)})) \\ c_{0,1} &= \text{Enc}_{k_1^0}(\text{Enc}_{k_2^1}(k_3^{g(0,1)})) \\ c_{1,0} &= \text{Enc}_{k_1^1}(\text{Enc}_{k_2^0}(k_3^{g(1,0)})) \\ c_{1,1} &= \text{Enc}_{k_1^1}(\text{Enc}_{k_2^1}(k_3^{g(1,1)})) \end{aligned}$$

其中  $\text{Enc}$  是对称加密方案  $(\text{Gen}, \text{Enc}, \text{Dec})$  的加密算法，且该对称加密方案满足选择明文攻击安全 (IND-CPA secure)、有不可知的密文空间、可验证的密文空间（见定义 8.1.1）。最后，我们还需要将这四个值随机排列，令排列后的值为  $c_0, c_1, c_2, c_3$ ，我们将其称为门  $g$  的混淆表 (garbled table)。不难发现，在已知  $k_1^\alpha, k_2^\beta$  的情况下很容易计算  $k_3^{g(\alpha, \beta)}$ ，计算方式如下：对于每个  $i \in \{0, 1, 2, 3\}$ ，计算  $\text{Dec}_{k_2^\beta}(\text{Dec}_{k_1^\alpha}(c_i))$ ，如果有多个项解密成功（解密得到的不是  $\perp$ ），输出  $\text{abort}$ ，否则，将唯一解密成功得到的值定义为  $k_3^\gamma$ 。（如果只有一项解密成功，那么一定有  $k_3^\gamma = k_3^{g(\alpha, \beta)}$ ，后面我们将证明：有多个项解密成功的概率是可忽略的。）

完整的混淆电路包含了每个门的混淆表。设电路  $\mathcal{C}$  的导线数量为  $m$ ，设  $w_1, \dots, w_m$  为这些导线的标签 (label)。如果  $w_i$  和  $w_j$  都来自门  $g$  的输出导线，我们将其视为一条导线，即  $w_i = w_j$ ；同样地，如果某个输入比特是多个门的输入，我们也将其视为一条导线。对于每个标签  $w_i$ ，选取两个独立的密钥  $k_i^0, k_i^1 \leftarrow \text{Gen}(1^n)$ ；总共需要选取  $2m$  个密钥，它们都是互相独立的。根据这些密钥，我们可以按照上述方式计算每个门的混淆表。最后，对于输出导线，需要计算解密表。如果  $w_i$  是一条输出导线，那么它的解密表包括  $(0, k_i^0)$  和  $(1, k_i^1)$ 。

我们用  $G(\mathcal{C})$  来表示  $\mathcal{C}$  的混淆电路，它包括每个门的混淆表和输出的解密表。至此，我们完成了构造混淆电路的形式化描述。

下面，我们证明上述混淆电路的构造方式满足正确性。

**引理 8.2.** （混淆电路的正确性）令  $x = x_1 \dots x_n, y = y_1 \dots y_n$  是电路  $\mathcal{C}$  的两个长度为  $n$  的输入。令  $w_{\text{in}_1} \dots w_{\text{in}_n}$  是  $x$  对应的输入导线的标签， $w_{\text{in}_{n+1}} \dots w_{\text{in}_{2n}}$  是  $y$  对应的输入导线的标签。假设构造  $G(\mathcal{C})$  所使用的加密方案有不可知的密文空间、可验证的密文空间，那么，已知混淆电路  $G(\mathcal{C})$  和密钥  $k_{\text{in}_1}^{x_1}, \dots, k_{\text{in}_n}^{x_n}, k_{\text{in}_{n+1}}^{y_1}, \dots, k_{\text{in}_{2n}}^{y_n}$ ，可以计算  $\mathcal{C}(x, y)$ ，失败概率是可忽略的。

证明。首先，我们证明每个门都能被正确地“解密”。令门  $g$  的输入导线是  $w_1, w_2$ ，输出导线是  $w_3$ ，我们证明：对于任意的  $\alpha, \beta \in \{0, 1\}$ ，已知密钥  $k_1^\alpha, k_2^\beta$  和门  $g$  的混淆表，可以得到  $k_3^{g(\alpha, \beta)}$ ，失败概率是可忽略的。更形式化地说，令  $c_0, c_1, c_2, c_3$  是门  $g$  的混淆表，我们需要找到  $c_i$  满足  $c_i = \text{Enc}_{k_1^\alpha}(\text{Enc}_{k_2^\beta}(k_3^{g(\alpha, \beta)}))$ 。

<sup>8</sup>非门 (NOT) 只有一个输入，可以通过“和常数 1 异或”来实现，后面的第8.1.6节会讲到：可以将异或门的开销优化为免费的。因为电路中无需设置非门。

下面，我们证明：只存在唯一的  $i$  满足  $c_i \in \text{Range}_n(k_1^\alpha)$  且  $\text{Dec}_{k_1^\alpha}(c_i) \in \text{Range}_n(k_2^\beta)$ . 也就是说，混淆表只有一项能被正确地解密。

这里的证明基于加密方案有不可知的密文空间。假设有两个值  $c$  都满足  $c \in \text{Range}_n(k_1^\alpha)$  且  $\text{Dec}_{k_1^\alpha}(c) \in \text{Range}_n(k_2^\beta)$ , 我们将其记为  $c_i$  和  $c_j$ . 不失一般性地，假设  $c_i = \text{Enc}_{k_1^\alpha}(\text{Enc}_{k_2^\beta}(k_3^{g(\alpha,\beta)}))$ , 即  $c_i$  应该被正确地解密。那么对于  $c_j$ , 有两种可能的情况：

1.  $c_j = \text{Enc}_{k_1^\alpha}(\text{Enc}_{k_2^{1-\beta}}(z))$ ,  $z \in \{k_3^0, k_3^1\}$ :

这意味着  $\text{Enc}_{k_2^{1-\beta}}(z) \in \text{Range}_n(k_2^\beta)$ . 由于  $k_2^{1-\beta}, k_3^0, k_3^1$  都是独立随机选择的，与  $k_2^\beta$  无关。我们可以定义算法  $\mathcal{A}$ , 它随机选择两个密钥  $k', k'' \leftarrow \text{Gen}(1^n)$  并输出  $c = \text{Enc}_{k'}(k'')$ . 那么，对于随机的密钥  $k \leftarrow \text{Gen}(1^n)$ ,  $c \in \text{Range}_n(k)$  的概率等于  $\text{Enc}_{k_2^{1-\beta}}(z) \in \text{Range}_n(k_2^\beta)$  的概率。由于加密方案有不可知的密文空间，这个概率是可忽略的。

2.  $c_j \in \text{Enc}_{k_1^{1-\alpha}}(z)$ ,  $z = \text{Enc}_{k'}(k'')$ ,  $k' \in \{k_2^0, k_2^1\}$ ,  $k'' \in \{k_3^0, k_3^1\}$ :

此时， $\text{Enc}_{k_1^{1-\alpha}}(z) \in \text{Range}_n(k_1^\alpha)$ . 由于  $k_1^{1-\alpha}, k', k''$  都是独立随机选择的，与上一种情况同理可知：该事件发生的概率是可忽略的。

下面，我们论证整个电路可以被正确地“解密”。如果  $\alpha \in \{0, 1\}$  是电路  $\mathcal{C}$  以  $(x, y)$  为输入计算时导线  $w_i$  的正确的值，那么我们说  $k_i^\alpha$  是  $w_i$  正确的密钥。我们用归纳法来证明。首先，对于输入导线， $k_{\text{in}_1}^{x_1}, \dots, k_{\text{in}_n}^{x_n}, k_{\text{in}_{n+1}}^{y_1}, \dots, k_{\text{in}_{2n}}^{y_n}$  是给定的正确的密钥。根据前面的论证，对于输入导线为  $w_i, w_j$ , 输出导线为  $w_\ell$  的门  $g$ , 给定密钥  $k_i^\alpha$  和  $k_j^\beta$ , 可以唯一正确地解密  $k_\ell^{g(\alpha,\beta)} = \text{Dec}_{k_j^\beta}(\text{Dec}_{k_i^\alpha}(c_{\alpha,\beta}))$ . 也就是说，对于电路中的每个门  $g$ , 已知其输入导线的正确密钥，排除可忽略的失败概率，可以得到其输出导线的正确密钥。每个门出错的概率都是可忽略的，那么整个电路出错的概率（不超过每个门出错的概率之和）也是可忽略的。因此，可以计算电路中每条导线的正确密钥。

最后，排除可忽略的失败概率，根据输出导线的正确密钥和解密表，得到的值是  $\mathcal{C}(x, y)$ . 证毕。□

**姚氏混淆电路协议描述。**至此，我们可以给出姚氏混淆电路协议的形式化描述，如图 8.1 所示。

### 8.1.3 安全性分析

首先，让我们直观地分析一下姚氏混淆电路协议的安全性。

**正确性。**根据引理 8.2，如果加密方案具有不可知的密文空间、可验证的密文空间，那么  $P_2$  计算混淆电路得到的是正确的  $\mathcal{C}(x, y) = f(x, y)$ .

**隐私性。**协议中， $P_1$  收到的消息除了最后的输出  $f(x, y)$ , 只有 OT 协议中的消息。根据 OT 协议的安全性， $P_1$  无法知道  $P_2$  的输入。对  $P_2$  来说，他在协议中收到了混淆电路  $G(\mathcal{C})$ 、 $x$  对应的密钥  $k_1^{x_1}, \dots, k_n^{x_n}$ , 并执行了 OT 协议。根据引理 8.1，如果使用了 IND-CPA 安全的加密方案，它也满足双重加密安全，也就是说，无法成功解密的其他项不会泄露任何关于明文的信息；而  $x$  对应的密钥是随机选取的，也不会泄露  $x$ . 直观上看，协议满足隐私性。

如果要将以上的直观分析转化为形式化的证明，模拟器  $\mathcal{S}$  应该如何构造呢？我们仍采用模块化的思想，假设存在 OT 协议  $\Pi_{\text{OT}}$  安全实现了理想功能  $\mathcal{F}_{\text{OT}}$ , 然后我们在  $\mathcal{F}_{\text{OT}}$ -混合模型下进行证明。对于  $P_1$  被攻陷的情况， $P_1$  在 OT 协议外收到的消息只有  $f(x, y)$ , 且  $P_1$  是 OT 的发送方，在 OT 协议中没有输出，所以它的视图在  $\mathcal{F}_{\text{OT}}$ -混合模型下可以完美地模拟。

### 姚氏混淆电路协议

**公共输入:** 布尔电路  $\mathcal{C}$ , 它对于任意的  $x, y \in \{0, 1\}^n$ , 满足  $\mathcal{C}(x, y) = f(x, y)$ , 其中  $f : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}^{2n}$ .  $\mathcal{C}$  的每条输出导线不会是其他门的输入导线。

**输入:**  $P_1$  的输入是  $x \in \{0, 1\}^n$ ,  $P_2$  的输入是  $y \in \{0, 1\}^n$ .

**协议:**

1.  $P_1$  按照本节所述的方式计算混淆电路  $G(\mathcal{C})$ , 并将其发送给  $P_2$ .
2. 令  $w_1, \dots, w_n$  是  $x$  对应的输入导线,  $w_{n+1}, \dots, w_{2n}$  是  $y$  对应的输入导线。执行以下操作:
  - (a)  $P_1$  发送  $k_1^{x_1}, \dots, k_n^{x_n}$  给  $P_2$ .
  - (b) 对于  $i \in [n]$ ,  $P_1$  和  $P_2$  执行 1-out-of-2 OT 协议,  $P_1$  的输入是  $(k_{n+i}^0, k_{n+i}^1)$ ,  $P_2$  的输入是  $y_i$ . (OT 协议可以并行地执行)
3.  $P_2$  获得了混淆电路和对应于输入的  $2n$  个密钥, 他按照本节所述的方式计算混淆电路获得  $f(x, y)$ .  $P_2$  将  $f(x, y)$  发送给  $P_1$ .

图 8.1: 姚氏混淆电路协议

对于  $P_2$  被攻陷的情况,  $\mathcal{S}$  又应该如何构造呢? 此时, 模拟器  $\mathcal{S}$  需要发送混淆电路  $G(\mathcal{C})$  给  $P_2$ .  $\mathcal{S}$  不知道  $P_1$  的输入, 因此它不能遵循协议诚实地构造  $G(\mathcal{C})$ , 但是  $\mathcal{S}$  可以根据协议的输出  $\mathcal{C}(x, y)$ , 构造一个“假的”混淆电路, 使得  $P_2$  无论使用的是哪个密钥, 最后的输出总是  $\mathcal{C}(x, y)$ . (这个“假的”混淆电路的构造方式将在后文具体介绍。) 然后,  $\mathcal{S}$  发送密钥  $k_1^0, \dots, k_n^0$  给  $P_2$ , 并模拟 OT 协议使  $P_2$  得到  $k_{n+1}^{y_1}, \dots, k_{2n}^{y_n}$ .

细心的读者可能会发现, 这样的模拟方式在通用可组合模型下会出现问题。在通用可组合模型中, 环境  $\mathcal{Z}$  可以实时地与协议交互, 为协议提供输入, 并控制消息的发送顺序。但是, 当  $P_1$  发送混淆电路  $G(\mathcal{C})$  时,  $\mathcal{Z}$  可能还没有给  $P_2$  提供输入; 换句话说,  $\mathcal{Z}$  可以让  $P_1$  先执行第 1 步发送混淆电路, 然后在第 2 步执行 OT 时才给  $P_2$  提供输入。对于这样的情况,  $\mathcal{S}$  就无法构造一个输出总是  $\mathcal{C}(x, y)$  的“假的”混淆电路了, 因为此时  $P_2$  的输入还没有确定。

因此, 这个协议不能被证明是通用可组合安全的。我们需要在一个稍微弱一点的模型——独立 (stand-alone) 模型下进行安全性证明。下面, 我们介绍独立模型, 并展示它与通用可组合模型的不同之处。

#### 8.1.4 独立模型

直观地说, 在通用可组合模型中, 环境  $\mathcal{Z}$  可以实时地与协议交互, 因此模拟器  $\mathcal{S}$  也必须根据  $\mathcal{Z}$  的行为提供实时的模拟。而在独立 (stand-alone) 模型中, (对于半诚实敌手)  $\mathcal{S}$  的输入是被攻陷方的输入和输出, 它需要模拟被攻陷方的视图, 这里  $\mathcal{S}$  提供的是一种更“静态”的模拟, 因为协议的输入在一开始就确定了, 而不是在协议执行中实时选择的。

我们首先考虑两个参与方、半诚实敌手的情况, 给出其在独立模型下的安全定义。

**安全定义 (半诚实敌手)。** 我们定义以下符号:

- 令  $f = (f_1, f_2)$  是概率多项式时间功能。令  $\Pi$  是计算  $f$  的两方协议。
- 第  $i$  个参与方 ( $i \in \{1, 2\}$ ) 在双方以  $(x, y)$  为输入执行协议  $\Pi$  时的视图记为  $\text{view}_i^{\Pi}(x, y)$ , 它等于  $(x, r^i, m_1^i, \dots, m_t^i)$ , 其中  $r^i$  是第  $i$  个参与方的内部随机数,  $m_j^i$  是他收到的第  $j$  条消息。
- 第  $i$  个参与方 ( $i \in \{1, 2\}$ ) 在双方以  $(x, y)$  为输入执行协议  $\Pi$  时的输出记为  $\text{output}_i^{\Pi}(x, y)$ , 它可以从第  $i$  个参与方的视图中计算得到。记  $\text{output}^{\Pi}(x, y) = (\text{output}_1^{\Pi}(x, y), \text{output}_2^{\Pi}(x, y))$ .

**定义 8.1.4.** (独立模型下对于半诚实敌手的安全性) 令  $f = (f_1, f_2)$  是一个功能。如果存在 PPT 的算法  $\mathcal{S}_1$  和  $\mathcal{S}_2$  使得

$$\begin{aligned} \{\mathcal{S}_1(x, f_1(x, y)), f(x, y)\}_{x, y \in \{0, 1\}^*} &\stackrel{\text{comp}}{\approx} \{(\text{view}_1^{\Pi}(x, y), \text{output}_1^{\Pi}(x, y))\}_{x, y \in \{0, 1\}^*} \\ \{\mathcal{S}_2(y, f_2(x, y)), f(x, y)\}_{x, y \in \{0, 1\}^*} &\stackrel{\text{comp}}{\approx} \{(\text{view}_2^{\Pi}(x, y), \text{output}_2^{\Pi}(x, y))\}_{x, y \in \{0, 1\}^*} \end{aligned}$$

其中  $|x| = |y|$ , 那么, 我们说协议  $\Pi$  对于静态半诚实敌手安全实现了功能  $f$ .

在本书的定理中, 我们用“安全实现”表示独立模型下的安全实现, 与“UC-安全实现”加以区分。

注意, 如果  $f$  是概率功能, 那么仅仅要求  $\mathcal{S}_i$  的输出与  $\text{view}_i^{\Pi}(x, y)$  不可区分是不够的, 必须考虑它与协议输出的联合分布 (我们在第4.1.1节讨论过这一点)。但是, 对于确定性功能, 可以使用以下简化的定义。

**对于确定性功能的简化定义。**对于确定性功能, 我们无需考虑联合分布, 只需要  $\mathcal{S}_i$  的输出与  $\text{view}_i^{\Pi}(x, y)$  不可区分, 其定义如下。

**定义 8.1.5.** (安全实现确定性功能) 令  $f = (f_1, f_2)$  是一个确定性功能。如果存在 PPT 的算法  $\mathcal{S}_1$  和  $\mathcal{S}_2$  使得

$$\begin{aligned} \{\mathcal{S}_1(x, f_1(x, y))\}_{x, y \in \{0, 1\}^*} &\stackrel{\text{comp}}{\approx} \{\text{view}_1^{\Pi}(x, y)\}_{x, y \in \{0, 1\}^*} \\ \{\mathcal{S}_2(y, f_2(x, y))\}_{x, y \in \{0, 1\}^*} &\stackrel{\text{comp}}{\approx} \{\text{view}_2^{\Pi}(x, y)\}_{x, y \in \{0, 1\}^*} \end{aligned}$$

其中  $|x| = |y|$ , 那么, 我们说协议  $\Pi$  对于静态半诚实敌手安全实现了功能  $f$ .

**单一输出确定性功能。**功能  $f = (f_1, f_2)$  如果满足  $f_1 = f_2$ , 我们称之为单一输出的功能。下面我们说明: 单一输出确定性功能可以实现任意的概率功能。

首先, 我们说明如何实现概率功能。令  $f = (f_1, f_2)$  是一个单一输出概率功能。定义确定性功能  $f'((x, r), (y, s)) = f(x, y, r \oplus s)$ , 并假设  $\Pi'$  安全实现了  $f'$ . 实现  $f$  的协议  $\Pi$  构造如下: 当收到协议输入  $x, y \in \{0, 1\}^n$  时,  $P_1$  均匀随机选取  $r \leftarrow_{\$} \{0, 1\}^{q(n)}$ ,  $P_2$  均匀随机选取  $s \leftarrow_{\$} \{0, 1\}^{q(n)}$ , 其中  $q(n)$  是所需随机比特的上限。然后  $P_1$  和  $P_2$  分别以  $(x, r)$  和  $(y, s)$  为输入执行协议  $\Pi'$ , 协议结束后, 双方就得到了  $f'((x, r), (y, s)) = f(x, y, r \oplus s)$ .

下面, 我们展示如何实现双方获得不同输出的情况, 即实现功能  $f = (f_1, f_2)$ , 其中  $f_1 \neq f_2$ . 我们定义确定性功能  $f'((x, r), (y, s)) = (f_1(x, y) \oplus r || f_2(x, y) \oplus s)$ , 并假设  $\Pi'$  安全实现了  $f'$ . 实现  $f$  的协议  $\Pi$  构造如下: 当收到协议输入  $x, y \in \{0, 1\}^n$  时,  $P_1$  均匀随机选取  $r \leftarrow_{\$} \{0, 1\}^{q(n)}$ ,  $P_2$  均匀随机选取  $s \leftarrow_{\$} \{0, 1\}^{q(n)}$ , 其中  $q(n)$  是输出长度的上限。然后  $P_1$  和  $P_2$  分别以  $(x, r)$  和  $(y, s)$  为输入执行协议  $\Pi'$ , 协议结束后, 双方得到  $f'((x, r), (y, s)) = (f_1(x, y) \oplus r || f_2(x, y) \oplus s)$ . 令  $(v, w) = f'((x, r), (y, s))$ .  $P_1$  计算  $v \oplus r$  得到  $f_1(x, y)$ ,  $P_2$  计算  $w \oplus s$  得到  $f_2(x, y)$ .

综上所述, 单一输出确定性功能可以实现任意的概率功能。因此, 虽然第8.1.2节的姚氏混淆电路协议只考虑了单一输出确定性功能, 但我们可以基于它实现任意的概率功能。

**并发组合的（不）安全性。**我们前面讲到，与通用可组合模型相比，独立模型提供了弱一些的安全保证。具体而言，在独立模型中被证明安全的协议，当与其他协议并发运行时，可能是不安全的。我们通过一个例子来展示这一点。

考虑密钥交换 (key exchange) 的场景。 $P_1$  和  $P_2$  希望生成一个共同的密钥，这个密钥对于可以观察网络通信的敌手是未知的。在实际场景中，生成的密钥往往用于消息的加密或认证。假设  $\Pi$  是在独立模型下被证明安全的密钥交换协议。定义协议  $\Pi'$  如下： $\Pi'$  与  $\Pi$  相同，但增加了以下指令：“如果密钥已经生成，接收一条敌手的消息，如果这条消息等于生成的密钥，回复 yes，否则回复 no”。

首先，我们论证  $\Pi'$  在独立模型下仍然是安全的。注意，在独立模型中，协议输入必须在协议开始执行前就确定（最后的那条消息是敌手的输入）。因此，敌手在协议开始前猜对密钥的概率是可忽略的，双方最后几乎总是回复 no，因而协议  $\Pi'$  与协议  $\Pi$  的效果相同（排除可忽略的猜对密钥的概率）。

下面，让我们看一下  $\Pi'$  与加密方案组合使用时的情况。假设参与方使用的是“一次一密”，即给定消息  $m$  和密钥  $k$ ，输出密文  $c = k \oplus m$ 。这是一个完美安全的加密方案。假设加密的消息只可能是“buy”或“sell”。也就是说， $P_1$  和  $P_2$  首先运行协议  $\Pi'$  生成密钥  $k$ ，然后  $P_1$  发送密文  $c$  给  $P_2$ ， $c$  只可能是  $k \oplus \text{buy}$  或  $k \oplus \text{sell}$ 。

此时，敌手就可以利用  $\Pi'$  中增加的指令进行攻击。敌手观察到密文  $c$ ，然后计算  $c' = c \oplus \text{buy}$  并发送给  $P_1$ 。此时，如果加密的是“buy”，那么  $c' = c \oplus \text{buy} = k \oplus \text{buy} \oplus \text{buy} = k$ ， $P_1$  会回复 yes；如果加密的是“sell”， $P_1$  则会回复 no。这意味着在协议  $\Pi'$  与“一次一密”并发组合的情况下，敌手可以知道加密的消息！

这个例子说明了：当两个协议  $\Pi_1$  和  $\Pi_2$  并发执行且  $\Pi_2$  使用了  $\Pi_1$  的输出时，敌手可能会利用与  $\Pi_1$  的交互来获取额外的信息，从而攻击  $\Pi_2$ 。独立模型只考虑了协议单独运行的情况，因此没有涵盖这种攻击方式。

最后，我们再来看看通用可组合模型中  $\Pi'$  为什么不安全。在通用可组合模型中，环境  $\mathcal{Z}$  可以实时地与协议交互， $\mathcal{Z}$  首先让  $P_1$  和  $P_2$  运行协议生成密钥（ $P_1$  和  $P_2$  都是诚实方）， $\mathcal{Z}$  获得诚实方的输出，即密钥  $k$ ，然后  $\mathcal{Z}$  让敌手  $\mathcal{A}$  发送消息  $k'$  给  $P_1$ ， $k'$  有  $1/2$  概率等于  $k$ ，有  $1/2$  概率为随机密钥。如果在真实世界，当  $k' = k$  时  $P_1$  会回复 yes，当  $k'$  为随机密钥时（以压倒性的概率）回复 no；如果在理想世界， $P_1$  是模拟器  $\mathcal{S}$  模拟的，它不可能知道理想功能生成的密钥  $k$ ，因此  $\mathcal{S}$  不知道应该回复 yes 还是 no，它有  $1/2$  的概率模拟失败。从这个例子可以看出，环境  $\mathcal{Z}$  与协议的实时交互准确地体现了并发组合时协议可能面对的任何外部环境。

**顺序组合的安全性。**幸运的是，独立模型虽然不能保证并发组合的安全性，但也提供了不错的安全性保证——独立模型下证明安全的协议在顺序组合时仍然是安全的。具体而言，令  $p(n)$  是一个多项式， $\rho_1, \dots, \rho_{p(n)}$  是一系列协议，其中  $\rho_i$  安全实现了理想功能  $f_i$ 。如果协议  $\Pi$  在  $f_1, \dots, f_{p(n)}$ -混合模型下安全实现了理想功能  $g$ ，那么  $\Pi^{\rho_1, \dots, \rho_{p(n)}}$  安全实现了理想功能  $g$ ， $\Pi^{\rho_1, \dots, \rho_{p(n)}}$  表示将协议中对  $f_j$  的请求替换为执行子协议  $\rho_j$ ，且这些子协议顺序执行。

**定理 8.3.** (顺序组合的安全性) 令  $p(n)$  是一个多项式， $f_1, \dots, f_{p(n)}$  是两方 PPT 理想功能。令  $\rho_1, \dots, \rho_{p(n)}$  是一系列协议，满足  $\rho_i$  安全实现功能  $f_i$ 。令  $g$  是一个两方功能，协议  $\Pi$  在  $f_1, \dots, f_{p(n)}$ -混合模型下安全实现了  $g$ 。那么， $\Pi^{\rho_1, \dots, \rho_{p(n)}}$  安全实现了理想功能  $g$ 。

该定理的证明思路与定理 4.3 非常类似，区别在于这里我们限制了敌手的能力，敌手不能实时地与协议交互，且子协议必须顺序执行。本书不再给出该定理的形式化证明，感兴趣的读者可以参阅<sup>[10]</sup>，Sections 7.3.1 和 7.4.1。不过，我们可以以上面的密钥交换协议为例，直观地展示协议  $\Pi'$  在顺序组合

下是安全的。当协议  $\Pi'$  与“一次一密”顺序组合时，参与方先执行完成协议  $\Pi'$ ，再使用“一次一密”进行加密。那么，当敌手观察到密文  $c$  时，密钥交换协议  $\Pi'$  已经运行结束了，敌手无法利用  $\Pi'$  获取明文信息，因而上述攻击方式在顺序组合时就不存在了。

**对于恶意敌手的安全性定义。**与通用可组合模型类似，独立模型下的安全性定义很容易拓展到恶意敌手的情形。在恶意敌手模型下，敌手可以任意地违背协议，其造成的影响始终等价于“输入替换”。但是，独立模型下敌手不能实时地与协议交互，也就是说，敌手必须在协议一开始就确定诚实方的输入，以及被攻陷方攻击协议的方式。形式化地说，对于每个被攻陷方  $P_i$ ，敌手需要在协议开始前确定它运行的算法  $A_i$ 。安全性定义要求：存在一个模拟器，它能够模拟被攻陷方的视图，并提取被攻陷方的输入，使得协议输出与理想功能的输出相一致。

下面，我们给出形式化的定义。仍然考虑两个参与方的情况。设功能  $f = (f_1, f_2)$ 。第一个参与方  $P_1$  的输入是  $x$ ，第二个参与方  $P_2$  的输入是  $y$ 。他们希望计算功能  $f$  使  $P_1$  得到  $f_1(x, y)$ ， $P_2$  得到  $f_2(x, y)$ 。由于敌手可以任意地违背协议，有些影响是协议无法避免的：(1) 敌手可以替换自己的输入；(2) 敌手可以不参与协议；(3) 敌手可以执行过程中提前结束协议。也就是说，在恶意敌手模型下协议无法实现公平性 (fairness)<sup>[42]</sup>，协议有可能让恶意的参与方获得输出，但诚实的参与方没有获得输出。因此，理想世界的运行过程如下。

- 输入：每个参与方获得一个输入，记为  $w$ （对  $P_1$  而言  $w = x$ ，对  $P_2$  而言  $w = y$ ）。
- 向可信方 (*trusted party*) 发送输入：诚实的参与方总是将  $w$  发给可信方。恶意的参与方可以根据  $w$  选择结束协议或发送  $w' \in \{0, 1\}^{|w|}$  给可信方。
- 可信方回复第一个参与方。当可信方收到输入对  $(x, y)$  后，他首先向第一个参与方发送  $f_1(x, y)$ 。如果可信方只收到一个合法的输入，他向两个参与方都发送  $\perp$ 。
- 可信方回复第二个参与方。如果第一个参与方是恶意的，他可以基于自己的输入和可信方的回复，决定是否让可信方终止。如果第一个参与方选择终止，可信方向第二个参与方发送  $\perp$  并终止；否则，可信方向第二个参与方发送  $f_2(x, y)$ 。
- 输出：诚实的参与方总是将他从可信方收到的消息作为输出。恶意的参与方可以输出一个关于他的输入和来自可信方消息的（概率多项式时间可计算的）函数。

令  $f : \{0, 1\}^* \times \{0, 1\}^* \rightarrow \{0, 1\}^* \times \{0, 1\}^*$  是一个功能，其中  $f = (f_1, f_2)$ 。令  $\bar{M} = (M_1, M_2)$  是一对非统一的 (non-uniform)<sup>9</sup> 概率期望多项式时间 (probabilistic expected polynomial-time)<sup>10</sup> 的机器（代表理想世界中的参与方）。如果存在至少一个  $i \in \{1, 2\}$  满足  $M_i$  是诚实的（即遵循上述理想世界中诚实方的指令），那么我们说  $\bar{M}$  是可允许的 (admissible)。我们将  $\bar{M}$  在理想世界中对  $f$  的执行（以  $(x, y)$  为输入）记为  $\text{IDEAL}_{f, \bar{M}}(x, y)$ ，它被定义为上述理想世界运行中  $M_1$  和  $M_2$  的输出。

真实世界的运行过程如下。令  $f : \{0, 1\}^* \times \{0, 1\}^* \rightarrow \{0, 1\}^* \times \{0, 1\}^*$  是一个功能，其中  $f = (f_1, f_2)$ 。令  $\Pi$  是一个计算  $f$  的两方协议。令  $\bar{M} = (M_1, M_2)$  是一对非统一的概率多项式时间的机器（代表真实世界中的参与方）。如果存在至少一个  $i \in \{1, 2\}$  满足  $M_i$  是诚实的（即遵循协议  $\Pi$  中的指令），那么

<sup>9</sup> 非统一 (non-uniform) 指的是一个算法无需支持任意的输入，而是可以对每个输入定义单独的算法。例如，如果一个 RSA 加密算法只支持 2048 位的密钥就是非统一的，如果支持任意长度的密钥就是统一的。因此，非统一的算法相较于统一的算法，实际上放宽了限制。

<sup>10</sup> 一个概率期望多项式时间算法，它的运行时间的期望是多项式的，但不保证对每一个输入的运行时间都是多项式的。

我们说  $\overline{M}$  是可允许的。我们将  $\overline{M}$  在真实世界中对  $\Pi$  的执行（以  $(x, y)$  为输入）记为  $\text{REAL}_{\Pi, \overline{M}}(x, y)$ ，它被定义为真实世界运行中  $M_1$  和  $M_2$  的输出。

安全性基于真实世界模拟了理想世界，即，对于每一对真实世界中可允许的机器，存在一对理想世界中可允许的机器，使得两个世界的执行是不可区分的。

**定义 8.1.6.** 令  $f : \{0, 1\}^* \times \{0, 1\}^* \rightarrow \{0, 1\}^* \times \{0, 1\}^*$  是一个功能，其中  $f = (f_1, f_2)$ . 令  $\Pi$  是一个计算  $f$  的两方协议。如果对于每一对真实世界中可允许的非统一的概率多项式时间的 (*admissible non-uniform probabilistic polynomial-time*) 机器  $\overline{A} = (A_1, A_2)$ ，都存在一对理想世界中可允许的非统一的概率期望多项式时间的 (*admissible non-uniform probabilistic expected polynomial-time*) 机器  $\overline{B} = (B_1, B_2)$ ，使得

$$\{\text{IDEAL}_{f, \overline{B}}(x, y)\}_{x, y \text{ s.t. } |x|=|y|} \stackrel{\text{comp}}{\approx} \{\text{REAL}_{f, \overline{A}}(x, y)\}_{x, y \text{ s.t. } |x|=|y|}$$

那么，我们说协议  $\Pi$  对于静态恶意敌手安全实现了  $f$ .

假设  $P_i$  是恶意的，那么定义 8.1.6 中的机器  $B_i$  就扮演了模拟器  $\mathcal{S}$  的角色，它需要模拟被攻陷方的视图，并提取被攻陷方的输入，使得诚实方在两个世界中的输出相一致。

**独立模型与通用可组合模型的关系。** 独立模型与通用可组合模型最主要的不同之处在于：通用可组合模型中，环境  $\mathcal{Z}$  可以与协议实时地交互，它模拟了协议可能面对的任何外部环境；而独立模型没有定义环境  $\mathcal{Z}$ ，其考虑的是协议单独运行的安全性。这样的设定导致的一个技术上的区别在于：在通用可组合模型中，模拟器  $\mathcal{S}$  不可以将环境  $\mathcal{Z}$  “倒带” (rewind)；而在独立模型中，对于恶意敌手时，“倒带”是模拟器  $\mathcal{S}$  提取敌手（被攻陷方）输入的一种常用技术。具体而言，“倒带”指的是当协议运行到某一步骤时， $\mathcal{S}$  在此处为敌手设置一个断点，然后继续运行协议。之后， $\mathcal{S}$  可以使敌手的程序回到断点处，然后与敌手再次运行协议（但是这次会发送不同的消息），从而提取敌手的输入。

现在，我们对环境  $\mathcal{Z}$  施加一些限制，定义一个弱一些的环境。这个弱一些的环境只能与协议交互两次：第一次在协议开始前， $\mathcal{Z}$  为诚实方提供输入，并决定敌手  $\mathcal{A}$  攻击协议的方式；第二次在协议结束后， $\mathcal{Z}$  接收诚实方的输出以及  $\mathcal{A}$  的消息，并输出一个比特  $b$ 。我们称这样的环境为非并发的 (non-concurrent) 环境。

可以证明，协议  $\Pi$  对于非并发的环境安全实现了理想功能  $f$ ，当且仅当协议  $\Pi$  在独立模型下安全实现了  $f$ 。其原理不难理解：在独立模型中，敌手必须在协议开始前决定诚实方的输入和被攻陷方执行的算法，这就等价于限制环境  $\mathcal{Z}$  只能在协议开始时和协议结束时交互两次。完整的证明参见文献<sup>[8,43]</sup> (eprint version, Section 7.4)。

### 8.1.5 安全性证明

现在，我们给出姚氏混淆电路协议在独立模型下的安全性证明。

**定理 8.4.** 令  $f$  是单一输出确定性功能。假设使用的加密方案满足选择明文攻击安全 (*IND-CPA secure*)，且有不可知的密文空间、可验证的密文空间。那么，图 8.1 中的姚氏混淆电路协议  $\Pi$  在  $\mathcal{F}_{OT}$ -混合模型下对于静态半诚实敌手安全实现了  $f$ .

证明. 我们考虑  $P_1$  被攻陷和  $P_2$  被攻陷两种情况。因为实现的是确定性功能，我们可以使用更简化的定义 8.1.5。

**情况 1:**  $P_1$  被攻陷。

$P_1$  在协议中的视图只包括他的输入、使用的随机数、OT 的视图、以及最后收到的协议输出。因为  $P_1$  是 OT 的发送方，在 OT 协议中没有输出，在  $\mathcal{F}_{\text{OT}}$ -混合模型下模拟器  $\mathcal{S}_1$  无需模拟 OT 中的视图。因此  $\mathcal{S}_1$  的构造非常简单： $\mathcal{S}_1$  的输入是  $(x, f(x, y))$ ，它随机选择  $r_C$  ( $P_1$  以此为随机数生成混淆电路  $G(\mathcal{C})$ )，然后输出

$$(x, r_C, f(x, y))$$

显然， $\mathcal{S}_1$  选择的随机数与  $P_1$  诚实运行选择的随机数有相同的分布。因此，

$$\{\mathcal{S}_1(x, f(x, y))\}_{x,y \in \{0,1\}^n} \stackrel{\text{comp}}{\approx} \{\text{view}_1^{\Pi}(x, y)\}_{x,y \in \{0,1\}^n}$$

**情况 2:**  $P_2$  被攻陷。

这种情况下，我们需要构造模拟器  $\mathcal{S}_2$ ，其输入是  $(y, f(x, y))$ ，输出是  $P_2$  在协议中的视图。 $P_2$  在协议的第 1 步收到了混淆电路，所以  $\mathcal{S}_2$  需要模拟这个混淆电路，并且使得  $P_2$  按照协议计算后能得到  $f(x, y)$ 。 $\mathcal{S}_2$  不知道  $P_1$  的输入  $x$ ，所以它不能诚实地生成混淆电路，它需要生成一个“假的”混淆电路，使得无论  $P_2$  用的是哪组密钥，计算的结果都是  $f(x, y)$ 。 $\mathcal{S}_2$  采用的方法是让混淆表的 4 项都加密同一个密钥，这样输入导线的值就不会影响输出导线的值了。我们需要证明：这样生成的“假的”混淆电路与真实的混淆电路是不可区分的。

我们给出  $\mathcal{S}_2$  的形式化描述。 $\mathcal{S}_2$  首先构造一个“假的”混淆电路，记为  $\tilde{G}(\mathcal{C})$ 。 $\tilde{G}(\mathcal{C})$  的构造方式如下。对于电路  $\mathcal{C}$  的每条导线  $w_i$ ， $\mathcal{S}_2$  选择两个随机密钥  $k_i, k'_i$ 。令  $g$  是以  $w_i, w_j$  为输入导线， $w_\ell$  为输出导线的门。 $g$  的混淆表的 4 项都加密了  $k_\ell$ ，无论输入导线的密钥是什么（也就是  $k'_\ell$  不会被加密）。换句话说， $\mathcal{S}_2$  计算以下 4 个值

$$\begin{aligned} c_{0,0} &= \text{Enc}_{k_i}(\text{Enc}_{k_j}(k_\ell)) \\ c_{0,1} &= \text{Enc}_{k_i}(\text{Enc}_{k'_j}(k_\ell)) \\ c_{1,0} &= \text{Enc}_{k'_i}(\text{Enc}_{k_j}(k_\ell)) \\ c_{1,1} &= \text{Enc}_{k'_i}(\text{Enc}_{k'_j}(k_\ell)) \end{aligned}$$

并将其随机排列。对于电路中的每个门， $\mathcal{S}_2$  都执行这样的操作。最后， $\mathcal{S}_2$  还需要生成解密表。我们将输出  $f(x, y)$  表示为二进制比特  $z_1 \dots z_n$ ，设  $w_{m-n+1}, \dots, w_m$  为输出导线。对于  $i = 1, \dots, n$ ，令  $k_{m-n+i}$  是以  $w_{m-n+i}$  为输出导线的门的混淆表中加密的（唯一）密钥， $k'_{m-n+i}$  是另一个密钥。那么，如果  $z_i = 0$ ，解密表为  $[(0, k_{m-n+i}), (1, k'_{m-n+i})]$ ；如果  $z_i = 1$ ，解密表为  $[(0, k'_{m-n+i}), (1, k_{m-n+i})]$ 。以上就是“假的”混淆电路  $\tilde{G}(\mathcal{C})$  的构造方式。

$\mathcal{S}_2$  还需要模拟  $P_2$  收到  $P_1$  发送的密钥、以及 OT 协议中的视图。 $\mathcal{S}_2$  令  $P_2$  从  $P_1$  收到的密钥为  $k_1, \dots, k_n$ （注意， $w_1, \dots, w_n$  对应的密钥是  $k_1, k'_1, \dots, k_n, k'_n$ 。实际上， $P_2$  从  $P_1$  收到的密钥也可以是  $k'_1, \dots, k'_n$ ，或者任意的组合）。 $\mathcal{F}_{\text{OT}}$ -混合模型下， $\mathcal{S}_2$  令  $P_2$  在 OT 协议中获得的输出是  $k_{n+1}, \dots, k_{2n}$ （它们是输入  $y$  对应的密钥，同样地，它们也可以是  $k'_{n+1}, \dots, k'_{2n}$ ，或者任意的组合）。最后， $\mathcal{S}_2$  输出

$$(y, \tilde{G}(\mathcal{C}), k_1, \dots, k_n, k_{n+1}, \dots, k_{2n})$$

以上是  $\mathcal{S}_2$  的形式化描述。我们需要证明

$$\{\mathcal{S}_2(y, f(x, y))\}_{x, y \in \{0,1\}^n} \stackrel{\text{comp}}{\approx} \{\text{view}_2^\Pi(x, y)\}_{x, y \in \{0,1\}^n}$$

其中，

$$\begin{aligned}\{\mathcal{S}_2(y, f(x, y))\}_{x, y \in \{0,1\}^n} &= \{(y, \tilde{G}(\mathcal{C}), k_1, \dots, k_n, k_{n+1}, \dots, k_{2n})\}_{x, y \in \{0,1\}^n} \\ \{\text{view}_2^\Pi(x, y)\}_{x, y \in \{0,1\}^n} &= \{(y, G(\mathcal{C}), k_1^{x_1}, \dots, k_n^{x_n}, k_{n+1}^{y_1}, \dots, k_{2n}^{y_n})\}_{x, y \in \{0,1\}^n}\end{aligned}$$

我们通过混合论证 (hybrid argument) 来证明这两个的分布是不可区分的。我们定义一系列混合实验  $H_i(x, y)$ ，每次将真实混淆电路  $G(\mathcal{C})$  中的一个门替换为  $\tilde{G}(\mathcal{C})$  中的门。在进行证明之前，首先考虑构造  $\tilde{G}(\mathcal{C})$  的另一种方式。这种构造方式使用了双方的输入  $x$  和  $y$ ，但是构造的电路与  $\mathcal{S}_2$  只根据  $y$  和  $f(x, y)$  所构造出的  $\tilde{G}(\mathcal{C})$  完全一样 ( $\mathcal{S}_2$  不可能用这种方式来构造  $\tilde{G}(\mathcal{C})$ ，这只是一个思想实验)。

另一种构造方式如下：首先，根据输入  $x, y$  计算整个电路，将所有密钥标记为活跃的 (active) 和不活跃的 (inactive)。如果导线  $w_a$  的值是  $\alpha$ ，那么  $k_a^\alpha$  是活跃的， $k_a^{1-\alpha}$  是不活跃的。然后，诚实地构造混淆电路  $G(\mathcal{C})$ 。最后，对每个门  $g$  的混淆表做如下改动：令  $w_a$  是门  $g$  的输出导线，门  $g$  的混淆表中的每项都加密了  $w_a$  的活跃密钥。这样，就完成了  $\tilde{G}(\mathcal{C})$  的构造。

我们论证这样构造的电路与  $\mathcal{S}_2$  构造的  $\tilde{G}(\mathcal{C})$  分布完全相同。首先，对于每个门的混淆表，两者都加密了单一的密钥；其次，每个门的密文的顺序都是随机的；再次，两者计算后得到的结果都是  $f(x, y)$ ；最后，这种构造方式中，解密表将活跃密钥解密为  $f(x, y)$ ，电路输出导线对应的门只加密了活跃密钥，而同样地， $\mathcal{S}_2$  构造的  $\tilde{G}(\mathcal{C})$  中，电路输出导线对应的门也只加密了一种密钥，这些密钥会被解密为  $f(x, y)$ 。综上所述，这样构造的电路与  $\mathcal{S}_2$  构造的  $\tilde{G}(\mathcal{C})$  分布相同。

在定义混合实验  $H_i(x, y)$  之前，我们需要将电路中的门排序。排序方式如下：电路中的门被排序为  $g_1, \dots, g_{|\mathcal{C}|}$ ，如果  $g_\ell$  的输入导线来自门  $g_i$  和  $g_j$ ，那么  $i < \ell$  且  $j < \ell$ 。这种排序方式称为电路的拓扑排序。混合实验  $H_i(x, y)$  定义如下：

**混合实验  $H_i(x, y)$** . 在这个实验中，混淆电路的构造方式如下：首先，根据输入  $x, y$  计算整个电路，将所有密钥标记为活跃的和不活跃的。然后，将前  $i$  个门  $g_1, \dots, g_i$  的混淆表按照上述另一种构造方式进行改动：即对于  $1 \leq j \leq i$ ，如果  $w_a$  是门  $g_j$  的输出导线，那么  $g_j$  的混淆表只加密  $w_a$  的活跃密钥。其余门  $g_{i+1}, \dots, g_{|\mathcal{C}|}$  的混淆表保持不变。

显然， $H_0(x, y) = \text{view}_2^\Pi(x, y)$ 。而我们前面论证过，另一种构造方式构造的混淆电路与  $\mathcal{S}_2$  构造的  $\tilde{G}(\mathcal{C})$  分布完全相同，因此  $H_{|\mathcal{C}|}(x, y) = \mathcal{S}_2(y, f(x, y))$ 。下面，我们只需要证明  $H_0(x, y) \stackrel{\text{comp}}{\approx} H_{|\mathcal{C}|}(x, y)$ 。

这里的证明方法是一个标准的混合论证。假设存在 PPT 的区分器  $D$  和多项式  $p(\cdot)$ ，对于无数多个  $n$ ，都有  $|\Pr[D(H_0(x, y)) = 1] - \Pr[D(H_{|\mathcal{C}|}(x, y)) = 1]| > \frac{1}{p(n)}$ ，那么一定存在某个  $i$ ，对于无数多个  $n$ ，有  $|\Pr[D(H_{i-1}(x, y)) = 1] - \Pr[D(H_i(x, y)) = 1]| > \frac{1}{|\mathcal{C}|p(n)}$ 。于是，我们可以通过  $D, x, y, i$  来构造一个可以攻破加密方案  $(\text{Gen}, \text{Enc}, \text{Dec})$  的双重加密安全性的对手  $\mathcal{A}_E$ ： $\mathcal{A}_E$  可以选择两组明文，一组对应于  $H_{i-1}(x, y)$ ，另一组对应于  $H_i(x, y)$ 。那么，区分  $H_{i-1}(x, y)$  和  $H_i(x, y)$  就等价于区分密文，也等价于攻破加密方案  $(\text{Gen}, \text{Enc}, \text{Dec})$  的双重加密安全性。下面，我们将以上分析转化为形式化的证明。

**一个具体的例子。**我们先考虑一个具体的例子。设  $g_i$  是一个或门 (OR)， $g_i$  的输入导线是  $w_a$  和  $w_b$ ，输出导线是  $w_c$ ，且  $w_a$  和  $w_b$  不是其他门的输入导线。假设电路的输入是  $x$  和  $y$ ，导线  $w_a$  的值为 0，导线  $w_b$  的值为 1。那么， $k_a^0, k_b^1$  是活跃的， $k_a^1, k_b^0$  是不活跃的（我们用粗体表示不活跃的密钥）。对于导

线  $w_c$ ,  $k_c^1$  是活跃的 (因为  $0 \vee 1 = 1$ ),  $k_c^0$  是不活跃的。 $g_i$  真实的混淆表与“假的”混淆表的区别在于加密的值不同。具体而言,  $g_i$  真实的混淆表包括

$$\text{Enc}_{k_a^0}(\text{Enc}_{k_b^0}(k_c^0)), \text{Enc}_{k_a^0}(\text{Enc}_{k_b^1}(k_c^1)), \text{Enc}_{k_a^1}(\text{Enc}_{k_b^0}(k_c^1)), \text{Enc}_{k_a^1}(\text{Enc}_{k_b^1}(k_c^1)) \quad (8.1)$$

而“假的”混淆表包括

$$\text{Enc}_{k_a^0}(\text{Enc}_{k_b^0}(k_c^1)), \text{Enc}_{k_a^0}(\text{Enc}_{k_b^1}(k_c^1)), \text{Enc}_{k_a^1}(\text{Enc}_{k_b^0}(k_c^1)), \text{Enc}_{k_a^1}(\text{Enc}_{k_b^1}(k_c^1)) \quad (8.2)$$

不难发现, 在这个例子中, 不可区分性基于加密方案在密钥  $k_b^0$  下的 IND-CPA 安全性 (选择明文  $k_c^0$  和  $k_c^1$ )。不过这里我们仍然展示如何构造一个攻击双重加密安全性的敌手  $\mathcal{A}_E$ , 它满足

$$\Pr[\text{Game}_{\mathcal{A}_E}^{\text{double}}(n) = 1] > \frac{1}{2} + \frac{1}{2|\mathcal{C}|p(n)}$$

$\mathcal{A}_E$  的构造如下。它的输入是  $1^n$ , 输出密钥  $k_a^0, k_b^1 \leftarrow \text{Gen}(1^n)$  以及消息  $(k_c^0, k_c^1, k_c^1)$  和  $(k_c^1, k_c^1, k_c^1)$ 。根据双重加密游戏  $\text{Game}^{\text{double}}$ , Challenger 选择随机比特  $b$  和两个随机密钥, 为保持一致性我们将这两个密钥记作  $k_a^1, k_b^0$ 。那么, 根据  $b = 0$  或  $b = 1$ ,  $\mathcal{A}_E$  收到的消息分别是

$$\langle \text{Enc}_{k_a^0}(\text{Enc}_{k_b^0}(k_c^0)), \text{Enc}_{k_a^1}(\text{Enc}_{k_b^1}(k_c^1)), \text{Enc}_{k_a^1}(\text{Enc}_{k_b^0}(k_c^1)) \rangle \quad (8.3)$$

或者

$$\langle \text{Enc}_{k_a^0}(\text{Enc}_{k_b^0}(k_c^1)), \text{Enc}_{k_a^1}(\text{Enc}_{k_b^1}(k_c^1)), \text{Enc}_{k_a^1}(\text{Enc}_{k_b^0}(k_c^1)) \rangle \quad (8.4)$$

将  $\mathcal{A}_E$  收到的密文记作  $(c_1, c_2, c_3)$ .  $\mathcal{A}_E$  计算  $c = \text{Enc}_{k_a^0}(\text{Enc}_{k_b^1}(k_c^1))$  ( $\mathcal{A}_E$  知道  $k_a^0, k_b^1, k_c^1$ ), 然后生成  $\langle c_1, c, c_3, c_2 \rangle$ . 注意, 如果  $\mathcal{A}_E$  收到的是式 (8.3), 那么  $\langle c_1, c, c_3, c_2 \rangle$  等于式 (8.1); 如果  $\mathcal{A}_E$  收到的是式 (8.4), 那么  $\langle c_1, c, c_3, c_2 \rangle$  等于式 (8.2)。因此, 如果有敌手能区分式 (8.1) 和式 (8.2), 它就能区分式 (8.3) 和式 (8.4), 从而攻破加密方案的双重加密安全性。

但是  $\mathcal{A}_E$  的构造还没有结束, 因为  $\mathcal{A}_E$  需要基于收到的消息 (对应于式 (8.3) 或式 (8.4)) 构造混淆电路 (对应于  $H_{i-1}(x, y)$  或  $H_i(x, y)$ ), 然后将混淆电路发给区分器  $D$ . 我们还需要展示  $\mathcal{A}_E$  可以在不知道  $k_a^1, k_b^0$  的情况下构造电路中其余门的混淆表。我们分两种情况讨论:

1.  $w_a$  (或  $w_b$ ) 是电路的输入导线。由于  $w_a$  和  $w_b$  不是其他门的输入导线, 因此  $w_a$  和  $w_b$  对应的密钥不会出现在其他门的混淆表中, 只可能是  $P_1$  发给  $P_2$  的密钥或  $P_2$  在 OT 协议中收到的密钥。而  $P_1$  发给  $P_2$  的密钥和  $P_2$  在 OT 协议中收到的密钥都是活跃密钥, 因此  $\mathcal{A}_E$  不需要知道  $k_a^1, k_b^0$ .
2.  $w_a$  (或  $w_b$ ) 不是电路的输入导线。此时,  $w_a$  (或  $w_b$ ) 对应的密钥只可能出现在  $g_j$  中, 其中  $g_j$  的输出导线是  $w_a$  (或  $w_b$ )。根据电路中门的排序的要求, 有  $j < i$ . 因此,  $g_j$  已经被替换为“假的”混淆表, 不会加密非活跃密钥。所以  $\mathcal{A}_E$  不需要知道  $k_a^1, k_b^0$ .

综上,  $\mathcal{A}_E$  可以构造电路中其余门的混淆表。那么, 如果  $\mathcal{A}_E$  在  $\text{Game}^{\text{double}}$  中的情况是  $b = 0$ , 其生成的混淆电路就对应  $H_{i-1}(x, y)$ ; 如果  $\mathcal{A}_E$  在  $\text{Game}^{\text{double}}$  中的情况是  $b = 1$ , 其生成的混淆电路就对应  $H_i(x, y)$ .  $\mathcal{A}_E$  将生成的混淆电路发给  $D$ , 令  $\mathcal{A}_E$  的输出等于  $D$  的输出。如果存在区分器  $D$  满足  $\Pr[D(H_i(x, y)) = 1] - \Pr[D(H_{i-1}(x, y)) = 1] > \frac{1}{|\mathcal{C}|p(n)}$ , 我们有

$$\begin{aligned}
\Pr[\text{Game}_{\mathcal{A}_E}^{\text{double}}(n) = 1] &= \Pr[b = 1] \Pr[\mathcal{A}_E \text{ 输出 } 1 \mid b = 1] + \Pr[b = 0] \Pr[\mathcal{A}_E \text{ 输出 } 0 \mid b = 0] \\
&= \Pr[b = 1] \Pr[D(H_i(x, y)) = 1] + \Pr[b = 0] (1 - \Pr[D(H_{i-1}(x, y)) = 1]) \\
&= \frac{1}{2} \Pr[D(H_i(x, y)) = 1] + \frac{1}{2} (1 - \Pr[D(H_{i-1}(x, y)) = 1]) \\
&= \frac{1}{2} + \frac{1}{2} (\Pr[D(H_i(x, y)) = 1] - \Pr[D(H_{i-1}(x, y)) = 1]) \\
&> \frac{1}{2} + \frac{1}{2|C|p(n)}
\end{aligned}$$

即  $\mathcal{A}_E$  攻破了加密方案的双重加密安全性。下面，我们将这个例子推广到一般的情况。

**一般的情况。**令  $g_i$  是任意的门，其输入导线是  $w_a$  和  $w_b$ ，输出导线是  $w_c$ 。令  $\alpha$  和  $\beta$  分别是  $w_a$  和  $w_b$  上的值。那么， $k_a^\alpha, k_b^\beta$  是活跃密钥， $k_a^{1-\alpha}, k_b^{1-\beta}$  是不活跃密钥。 $g_i$  真实的混淆表包含以下值（以随机的排列）

$$\text{Enc}_{k_a^\alpha}(\text{Enc}_{k_b^\beta}(k_c^{g_i(\alpha, \beta)})), \text{Enc}_{k_a^\alpha}(\text{Enc}_{k_b^{1-\beta}}(k_c^{g_i(\alpha, 1-\beta)})), \text{Enc}_{k_a^{1-\alpha}}(\text{Enc}_{k_b^\beta}(k_c^{g_i(1-\alpha, \beta)})), \text{Enc}_{k_a^{1-\alpha}}(\text{Enc}_{k_b^{1-\beta}}(k_c^{g_i(1-\alpha, 1-\beta)}))$$

而  $g_i$  “假的”混淆表包含以下值

$$\text{Enc}_{k_a^\alpha}(\text{Enc}_{k_b^\beta}(k_c^{g_i(\alpha, \beta)})), \text{Enc}_{k_a^\alpha}(\text{Enc}_{k_b^{1-\beta}}(k_c^{g_i(\alpha, \beta)})), \text{Enc}_{k_a^{1-\alpha}}(\text{Enc}_{k_b^\beta}(k_c^{g_i(\alpha, \beta)})), \text{Enc}_{k_a^{1-\alpha}}(\text{Enc}_{k_b^{1-\beta}}(k_c^{g_i(\alpha, \beta)}))$$

两者的不可区分性基于双重加密的安全性。如果  $w_a, w_b$  只作为  $g_i$  的输入导线，而不是其他门的输入导线，那么  $\mathcal{A}_E$  生成混淆电路的方式与上面的具体例子中完全一样。然而，如果  $w_a, w_b$  是多个门的输入导线， $\mathcal{A}_E$  就无法在不知道  $k_a^{1-\alpha}, k_b^{1-\beta}$  的情况下自己生成整个混淆电路了。具体而言，令  $w_a$  是  $g_{i_1}^a, \dots, g_{i_j}^a$  的输入导线， $w_b$  是  $g_{i_1}^b, \dots, g_{i_\ell}^b$  的输入导线，那么  $k_a^{1-\alpha}, k_b^{1-\beta}$  就可能会用于加密这些门的混淆表。幸运的是，注意到双重加密游戏 Game<sup>double</sup> 中，敌手可以访问  $E(\cdot, k'_1, \cdot)$  和  $E(k'_0, \cdot, \cdot)$ ，所以  $\mathcal{A}_E$  可以在不知道  $k_a^{1-\alpha}, k_b^{1-\beta}$  的情况下通过这两个接口来生成用它们加密的密文，从而构造完整的混淆电路。于是，如果  $\mathcal{A}_E$  在 Game<sup>double</sup> 中的情况是  $b = 0$ ，其生成的混淆电路就对应  $H_{i-1}(x, y)$ ；如果  $\mathcal{A}_E$  在 Game<sup>double</sup> 中的情况是  $b = 1$ ，其生成的混淆电路就对应  $H_i(x, y)$ 。由此可知  $H_{i-1}(x, y)$  与  $H_i(x, y)$  不可区分，故  $H_0(x, y) \stackrel{\text{comp}}{\approx} H_{|\mathcal{C}|}(x, y)$ 。

**总结整个证明。**由  $H_0(x, y) \stackrel{\text{comp}}{\approx} H_{|\mathcal{C}|}(x, y)$  可知， $\{\mathcal{S}_2(y, f(x, y))\}_{x, y \in \{0, 1\}^n} \stackrel{\text{comp}}{\approx} \{\text{view}_2^{\Pi}(x, y)\}_{x, y \in \{0, 1\}^n}$ 。综上所述，我们构造了 PPT 的模拟器  $\mathcal{S}_1$  和  $\mathcal{S}_2$  使得

$$\begin{aligned}
\{\mathcal{S}_1(x, f_1(x, y))\}_{x, y \in \{0, 1\}^n} &\stackrel{\text{comp}}{\approx} \{\text{view}_1^{\Pi}(x, y)\}_{x, y \in \{0, 1\}^n} \\
\{\mathcal{S}_2(y, f_2(x, y))\}_{x, y \in \{0, 1\}^n} &\stackrel{\text{comp}}{\approx} \{\text{view}_2^{\Pi}(x, y)\}_{x, y \in \{0, 1\}^n}
\end{aligned}$$

证毕。 □

上一节（第8.1.4节）讲过，基于单一输出确定性功能可以实现任意的概率功能。我们有以下推论。

**推论 8.5.** 令  $f = (f_1, f_2)$  是任意的概率功能。假设存在加密方案满足选择明文攻击安全 (IND-CPA secure)，且有不可知的密文空间、可验证的密文空间。那么，存在协议  $\Pi$  在  $\mathcal{F}_{\text{OT}}$ -混合模型下对于半诚实敌手安全实现了  $f$ 。

### 8.1.6 混淆电路的优化

在运行姚氏混淆电路协议时，生成混淆电路和计算混淆电路是主要的计算开销，而传输混淆电路是主要的通讯开销。为了提升性能，研究人员提出了多种混淆电路的优化方案。

**标识置换 (point-and-permute)。**在经典方案中，计算方 ( $P_2$ ) 在计算混淆电路时，并不知道需要对混淆表的哪一行进行解密，因此只能对每一行都进行解密尝试，对于每个混淆表，平均需要 2.5 次尝试才能够成功解密。标识置换方案<sup>[40]</sup>为每一个密钥附加了一个标识比特。具体而言，对于导线  $w_a$ ，它的密钥  $k_a^0$  附带了标识比特  $p_a^0$ ，密钥  $k_a^1$  附带了标识比特  $p_a^1$ ，并且导线值 0 对应的标识比特和导线值 1 对应的标识比特不同，即  $p_a^0 = 1 - p_a^1$ 。我们将  $k_a^0||p_a^0$  称为导线  $w_a$  的值 0 对应的标签。

标识比特指示了应该解密混淆表的哪一行，从而使得计算方无需逐行尝试。对于输入导线为  $w_a, w_b$ ，输出导线为  $w_c$  的布尔门，当计算方持有一对标签  $(k_a^\alpha||p_a^\alpha, k_b^\beta||p_b^\beta)$  时， $p_a^\alpha||p_b^\beta$  指示了应当解密的行——如果混淆表的行序号从 1 开始计数（即混淆表有第 1, 第 2, 第 3, 第 4 行），那么计算方直接解密第  $2p_a^\alpha + p_b^\beta + 1$  行即可。加入标识比特后，混淆表的生成方式也需要做对应的修改：首先，混淆电路生成方为每条导线随机选择密钥和标识比特，满足每条导线值 0 对应的标识比特和值 1 对应的标识比特不同。混淆表的各行加密的不仅仅是密钥，而是密钥和标识比特（即标签），例如，与门的某一行应当是  $\text{Enc}_{k_a^0}(\text{Enc}_{k_b^0}(k_c^0||p_c^0))$ 。其次，混淆表的各行也不再是随机排列，而是根据标识比特，对于标签  $(k_a^\alpha||p_a^\alpha, k_b^\beta||p_b^\beta)$ ，将其对应的密文放在第  $2p_a^\alpha + p_b^\beta + 1$  行。表 8.5 展示了当  $p_a^0 = 1, p_b^0 = 0$  时，标识置换方案下与门的混淆表，注意最右边一列的混淆表已经根据标识比特进行排列。使用标识置换技术后，加密方案也无需在明文后添加  $n$  个 0 以保证不可知的密文空间了。

$w_a$ 的对应标签	$w_b$ 的对应标签	混淆表
$k_a^0  (p_a^0 = 1)$	$k_b^0  (p_b^0 = 0)$	$\text{Enc}_{k_a^1}(\text{Enc}_{k_b^0}(k_c^0  p_c^0))$
$k_a^0  (p_a^0 = 1)$	$k_b^1  (p_b^1 = 1)$	$\text{Enc}_{k_a^1}(\text{Enc}_{k_b^1}(k_c^1  p_c^1))$
$k_a^1  (p_a^1 = 0)$	$k_b^0  (p_b^0 = 0)$	$\text{Enc}_{k_a^0}(\text{Enc}_{k_b^0}(k_c^0  p_c^0))$
$k_a^1  (p_a^1 = 0)$	$k_b^1  (p_b^1 = 1)$	$\text{Enc}_{k_a^0}(\text{Enc}_{k_b^1}(k_c^1  p_c^1))$

表 8.5: 标识置换方案，与门的混淆表

**混淆表行缩减 (Garbled Row Reduction, GRR)。**混淆表行缩减最早由 Naor, Pinkas 和 Sumner 以  $4 \rightarrow 3$  混淆表行缩减（缩写为 *GRR3*）的形式提出<sup>[44]</sup>。它的思想是：由于混淆表加密的标签是随机数，我们可以通过巧妙地选择这些随机数，使得混淆表的第一行固定为全 0 字符串。例如，对于索引为  $i$  的门  $g_i$ ，假设其输入导线为  $w_a, w_b$ ，输出导线为  $w_c$ 。它混淆表的双重加密  $\text{Enc}_{k_a^\alpha, k_b^\beta}^i(k_c^{g_i(\alpha, \beta)}||p_c^{g_i(\alpha, \beta)})$  可以实现为  $H(k_a^\alpha||k_b^\beta||i) \oplus (k_c^{g_i(\alpha, \beta)}||p_c^{g_i(\alpha, \beta)})$ ，其中  $H$  是可以理想化为随机谕示机的哈希函数（参见第 2.3.2 节的定义）。当我们根据标识置换技术对混淆表进行排列后，假设  $H(k_a^\alpha||k_b^\beta||i) \oplus (k_c^{g_i(\alpha, \beta)}||p_c^{g_i(\alpha, \beta)})$  是混淆表的第一行，那么，由于  $k_c^{g_i(\alpha, \beta)}||p_c^{g_i(\alpha, \beta)}$  也是随机选择的，我们只需将其选择为  $k_c^{g_i(\alpha, \beta)}||p_c^{g_i(\alpha, \beta)} = H(k_a^\alpha||k_b^\beta||i)$ ，即可使混淆表的第一行为全 0 字符串，从而对于每个混淆表，只需传输 3 条密文。

$4 \rightarrow 2$  混淆表行缩减（缩写为 *GRR2*）<sup>[45]</sup>通过多项式插值的方式来生成混淆表，生成的混淆表只需要 2 行。GRR2 首先根据布尔门真值表的输出值中 0 和 1 的个数，将布尔门分为“奇门”和“偶门”，例如，与门、或门的真值表输出值中 0 和 1 的个数是 1 个或 3 个，因此它们是“奇门”；异或门的真值表输出值中 0 和 1 的个数都是 2 个，因此它是“偶门”。对于奇门和偶门，需要采用不同的方式构造混淆表。

我们首先定义一些符号。假设门  $g_i$  的混淆表的加密方式为  $\text{Enc}_{k_a^\alpha, k_b^\beta}^i(k_c^{g_i(\alpha, \beta)} || p_c^{g_i(\alpha, \beta)}) = H(k_a^\alpha || k_b^\beta || i) \oplus (k_c^{g_i(\alpha, \beta)} || p_c^{g_i(\alpha, \beta)})$ , 假设密钥的长度为  $\kappa$  比特。定义  $r = 2p_a^\alpha + p_b^\beta + 1$  为要解密的行的索引。定义  $K_r || M_r$  如下

$$K_r || M_r = H(k_a^\alpha || k_b^\beta || i) \quad (8.5)$$

其中  $K_r$  长度为  $\kappa$  比特,  $M_r$  长度为 1 比特。也就是说, 对于密文  $\text{Enc}_{k_a^\alpha, k_b^\beta}^i(k_c^{g_i(\alpha, \beta)} || p_c^{g_i(\alpha, \beta)})$ , 我们将  $H(k_a^\alpha || k_b^\beta || i)$  视为加密密钥和标识比特的掩码 (mask), 其中  $K_r$  是用于加密密钥的掩码,  $M_r$  是用于加密标识比特的掩码。

我们以与门为例, 展示奇门的混淆表的构造。设布尔门  $g_i$  是与门, 其输入导线为  $w_a, w_b$ , 输出导线为  $w_c$ . 假设它的标识比特为  $p_a^0 = 0, p_b^0 = 0$ , 那么, 此时排列后的混淆表如下

$$\begin{aligned} \text{Enc}_{k_a^0, k_b^0}^i(k_c^0 || p_c^0) &= (K_1 || M_1) \oplus (k_c^0 || p_c^0) \\ \text{Enc}_{k_a^0, k_b^1}^i(k_c^0 || p_c^0) &= (K_2 || M_2) \oplus (k_c^0 || p_c^0) \\ \text{Enc}_{k_a^1, k_b^0}^i(k_c^0 || p_c^0) &= (K_3 || M_3) \oplus (k_c^0 || p_c^0) \\ \text{Enc}_{k_a^1, k_b^1}^i(k_c^1 || p_c^1) &= (K_4 || M_4) \oplus (k_c^1 || p_c^1) \end{aligned}$$

其中,  $K_r || M_r$  按照式 (8.5) 所定义。

观察混淆表, 发现其 1~3 行对应的输出密钥都是  $k_c^0$ . 因此对  $(1, K_1), (2, K_2), (3, K_3)$  这三个点进行插值, 得到二次多项式  $P(X)$ . 然后计算  $K_5 = P(5), K_6 = P(6)$ . 再对  $(4, K_4), (5, K_5), (6, K_6)$  这三个点进行插值, 得到二次多项式  $Q(X)$ . 最后, 选取密钥  $k_c^0 = P(0), k_c^1 = Q(0)$ , 将混淆表设置为  $(K_5, K_6)$  以及加密的 4 个标识比特  $(c_1, c_2, c_3, c_4) = (M_1 \oplus p_c^0, M_2 \oplus p_c^0, M_3 \oplus p_c^0, M_4 \oplus p_c^1)$ . 也就是说, 混淆表的总大小为  $2\kappa + 4$  比特。

当计算方对混淆表进行计算时, 他持有  $(k_a^\alpha || p_a^\alpha, k_b^\beta || p_b^\beta)$ . 首先, 计算  $r = 2p_a^\alpha + p_b^\beta + 1$  和  $K_r || M_r = H(k_a^\alpha || k_b^\beta || i)$ . 然后, 他对  $(r, K_r), (5, K_5), (6, K_6)$  这三个点进行插值。不难发现, 如果  $r = 1, 2, 3$ , 那么得到的多项式就是  $P(X)$ ; 如果  $r = 4$ , 那么得到的多项式就是  $Q(X)$ . 因此, 计算方设置输出密钥  $k_c^{g_i(\alpha, \beta)}$  为这个多项式在 0 处的值, 并计算标识比特  $p_c^{g_i(\alpha, \beta)} = c_r \oplus M_r$ . 计算方至此完成该混淆表的计算。不难发现, 这样计算的结果是正确的。图 8.2 是上述方案的示意图。如果标识比特不是  $p_a^0 = 0, p_b^0 = 0$ , 只需在构造混淆表时做对应的排列 (例如, 表 8.5 展示的  $p_a^0 = 1, p_b^0 = 0$  的情况, 只需分别对  $(1, K_1), (3, K_3), (4, K_4)$  和  $(2, K_2), (5, K_5), (6, K_6)$  进行插值即可)。

偶门的混淆表的构造方式与奇门有所不同。设布尔门  $g_i$  是异或门, 其输入导线为  $w_a, w_b$ , 输出导线为  $w_c$ . 我们仍假设它的标识比特为  $p_a^0 = 0, p_b^0 = 0$ , 那么, 排列后的混淆表如下

$$\begin{aligned} \text{Enc}_{k_a^0, k_b^0}^i(k_c^0 || p_c^0) &= (K_1 || M_1) \oplus (k_c^0 || p_c^0) \\ \text{Enc}_{k_a^0, k_b^1}^i(k_c^1 || p_c^1) &= (K_2 || M_2) \oplus (k_c^1 || p_c^1) \\ \text{Enc}_{k_a^1, k_b^0}^i(k_c^1 || p_c^1) &= (K_3 || M_3) \oplus (k_c^1 || p_c^1) \\ \text{Enc}_{k_a^1, k_b^1}^i(k_c^0 || p_c^0) &= (K_4 || M_4) \oplus (k_c^0 || p_c^0) \end{aligned}$$

其中,  $K_r || M_r$  按照式 (8.5) 所定义。

观察混淆表, 发现第 1、第 4 行对应的输出密钥是  $k_c^0$ , 第 2、第 3 行对应的输出密钥是  $k_c^1$ . 因此对  $(1, K_1), (4, K_4)$  进行插值得到  $P(X)$ , 选取密钥  $k_c^0 = P(0)$ , 如果  $k_c^0$  对应的标识比特  $p_c^0 = 0$ , 就把  $P(5)$

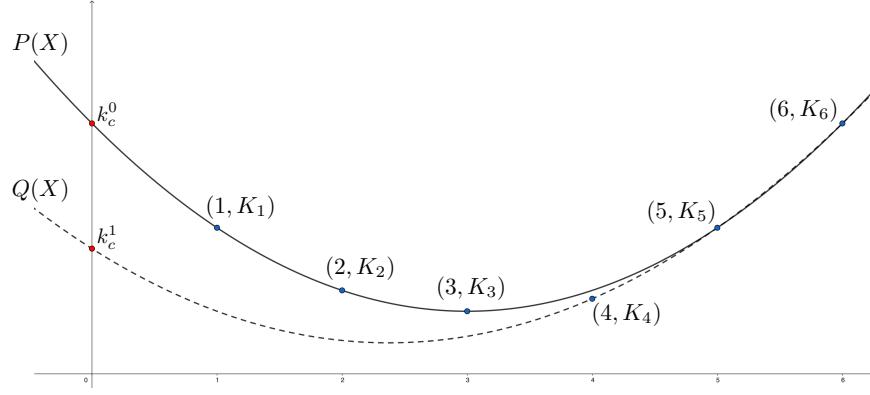


图 8.2: GRR2 示意图 (奇门)

放在混淆表的第 1 行, 否则放在第 2 行; 类似地, 对  $(2, K_2), (3, K_3)$  进行插值得到  $Q(X)$ , 选取密钥  $k_c^1 = Q(0)$ , 然后把  $Q(5)$  放在混淆表的另一行。混淆表被设置为两个长度为  $\kappa$  比特的值以及加密的 4 个标识比特  $(c_1, c_2, c_3, c_4) = (M_1 \oplus p_c^0, M_2 \oplus p_c^1, M_3 \oplus p_c^1, M_4 \oplus p_c^0)$ , 混淆表的总大小仍为  $2\kappa + 4$  比特。

当计算方对混淆表进行计算时, 他持有  $(k_a^\alpha \parallel p_a^\alpha, k_b^\beta \parallel p_b^\beta)$ . 首先, 计算  $r = 2p_a^\alpha + p_b^\beta + 1$  和  $K_r \parallel M_r = H(k_a^\alpha \parallel k_b^\beta \parallel i)$ . 然后, 计算  $p_c^{g_i(\alpha, \beta)} = c_r \oplus M_r$ . 如果该值为 0, 则取混淆表第 1 行的值; 如果该值为 1, 则取混淆表第 2 行的值。记所取的值为  $K_5$ . 最后, 计算方对  $(r, K_r), (5, K_5)$  进行插值。不难发现, 如果  $r = 1, 4$ , 那么得到的多项式就是  $P(X)$ ; 如果  $r = 2, 3$ , 那么得到的多项式就是  $Q(X)$ . 计算方设置  $k_c^{g_i(\alpha, \beta)}$  为这个多项式在 0 处的值, 完成该混淆表的计算。图 8.3 是上述方案的示意图。同样地, 如果标识比特不是  $p_a^0 = 0, p_b^0 = 0$ , 只需在构造混淆表时做对应的排列即可。

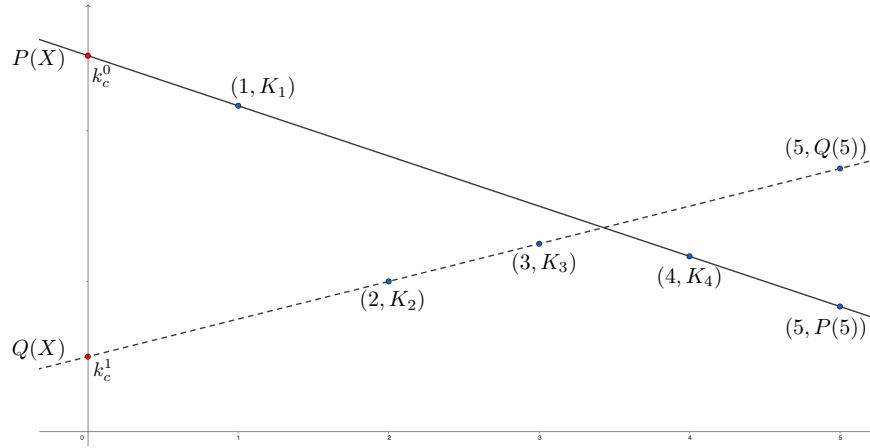


图 8.3: GRR2 示意图 (偶门)

**free-XOR.** 顾名思义, free-XOR 技术<sup>[46]</sup>可以将异或门的开销变为免费的。在刚才 GRR2 方案中, 我们发现, 异或门和其他门有所不同: 异或门是偶门, 与门、或门都是奇门。而且,  $a \oplus (b \oplus 1) = (a \oplus b) \oplus 1$ , 也就是将异或门的一个输入翻转, 会导致异或门的输出也翻转。异或门的这个性质使研究者思考: 当异或门的输入标签是  $(k_a^\alpha \parallel p_a^\alpha, k_b^\beta \parallel p_b^\beta)$  时, 能否使输出标签  $k_c^{g(\alpha, \beta)} \parallel p_c^{g(\alpha, \beta)}$  就等于  $(k_a^\alpha \parallel p_a^\alpha) \oplus (k_b^\beta \parallel p_b^\beta)$  呢? 如果这可以实现, 异或门的计算就无需任何解密操作了。

为此, 混淆电路生成方在生成混淆电路时首先随机选取一个全局偏移量  $\Delta \leftarrow_{\$} \{0, 1\}^\kappa$ . 对于每条导

线的标签，算法强制要求对应于 0 的密钥和对应于 1 的密钥的异或结果等于全局偏移量，即任意导线  $w_a$  都满足  $k_a^0 \oplus k_a^1 = \Delta$ . 当生成方随机选择了 0、1 密钥中的一个，就同时确定了另一个。对于异或门  $g$ ，假设其输入导线为  $w_a, w_b$ ，输出导线为  $w_c$ ，其输出密钥设置为  $k_c^0 = k_a^0 \oplus k_b^0$ ,  $k_c^1 = k_a^0 \oplus k_b^0 \oplus \Delta$ ，输出导线的标识比特设置为  $p_c^0 = p_a^0 \oplus p_b^0$ ,  $p_c^1 = p_a^0 \oplus p_b^0 \oplus 1$ . 这样，计算方在计算异或门时，可以通过计算  $k_c^{g(\alpha, \beta)} || p_c^{g(\alpha, \beta)} = (k_a^\alpha || p_a^\alpha) \oplus (k_b^\beta || p_b^\beta)$  直接得到输出标签。

需要注意的是，标识置换技术、GRR3 和 free-XOR 是相互兼容的。free-XOR 选取了全局偏移量  $\Delta$ ，将异或门的开销变为免费的，这并不影响与门、或门仍使用 GRR3 进行优化。GRR3 需要将一个密钥设置为哈希的输出，这与全局偏移量  $\Delta$  并不冲突。当同时使用标识置换技术、GRR3 和 free-XOR 时，混淆电路的构造方式如下：

1. 随机选取全局偏移量  $\Delta \leftarrow_{\$} \{0, 1\}^\kappa$ .
2. 对于电路的每条输入导线  $w_i$ ，随机选择 0 对应的导线标签  $k_i^0 || p_i^0 \leftarrow_{\$} \{0, 1\}^{\kappa+1}$ . 设置 1 对应的导线标签为  $k_i^1 || p_i^1 = (k_i^0 \oplus \Delta) || (p_i^0 \oplus 1)$ .
3. 按照拓扑顺序对电路中的每个门  $g_i$  按照如下方式构造混淆表（假设  $g_i$  的输入导线为  $w_a, w_b$ ，输出导线为  $w_c$ ，输入标签为  $(k_a^0 || p_a^0, k_a^1 || p_a^1), (k_b^0 || p_b^0, k_b^1 || p_b^1)$ ）：
  - 如果  $g_i$  是异或门，设置输出密钥为  $k_c^0 = k_a^0 \oplus k_b^0$ ,  $k_c^1 = k_a^0 \oplus k_b^0 \oplus \Delta$ ，设置输出导线的标识比特为  $p_c^0 = p_a^0 \oplus p_b^0$ ,  $p_c^1 = p_a^0 \oplus p_b^0 \oplus 1$ .
  - 如果  $g_i$  不是异或门，按照标识比特排列其混淆表。假设第一行为  $H(k_a^\alpha || k_b^\beta || i) \oplus (k_c^{g_i(\alpha, \beta)} || p_c^{g_i(\alpha, \beta)})$ ，设置  $(k_c^{g_i(\alpha, \beta)} || p_c^{g_i(\alpha, \beta)}) = H(k_a^\alpha || k_b^\beta || i)$ ，设置  $k_c^{1-g_i(\alpha, \beta)} = k_c^{g_i(\alpha, \beta)} \oplus \Delta$ ,  $p_c^{1-g_i(\alpha, \beta)} = p_c^{g_i(\alpha, \beta)} \oplus 1$ .
4. 最后，对于电路的每条输出导线，生成解密表。导线  $w_i$  的解密表包含  $(0, H(k_i^0))$  和  $(1, H(k_i^1))$ ，其中  $H$  是随机谕示机。这是为了防止解密表泄漏全局偏移量  $\Delta$ .

但是，GRR2 和 free-XOR 不兼容。因为 GRR2 中的密钥是插值多项式在 0 处的值，这是一个随机值，所以无法使电路中的所有密钥对都有相同的全局偏移量  $\Delta$ . 因此，同时使用标识置换技术、GRR3 和 free-XOR 一种很好的优化方案。不过，需要注意的是，由于 free-XOR 技术要求密钥对都有相同的全局偏移值，这使得底层的加密方案需要假设  $H$  是随机谕示机<sup>11</sup>，而无法基于伪随机函数构造<sup>12</sup>。GRR2 不需要假设  $H$  是随机谕示机（或满足关联健壮性）<sup>13</sup>。如果使用者不希望做出过强的假设，或是电路中异或门的比例很小，那么 GRR2 也是一种很好的优化方案。

**半门 (half-gates)。** 上面介绍的优化方案中，GRR2 可以将混淆表缩减为 2 条密文，但与 free-XOR 不兼容；GRR3 与 free-XOR 兼容，但是只能将混淆表缩减为 3 条密文。也就是说，上述方案无法同时实现免费的异或门并且将混淆表缩减为 2 条密文。半门<sup>[47]</sup>是一种巧妙的优化方案，它可以使奇门（包括与门、或门、与非门、或非门）的混淆表缩减为 2 条，且与 free-XOR 完全兼容。

我们先以与门为例展示半门方案的构造。半门的思想是将与门表示为两个半门异或的结果。每个半门都是一个与门，但是有一个参与方知道该门的一个输入，因此称之为半门。两个半门分别为生成方

<sup>11</sup>更准确地说， $H$  只需要满足关联健壮性 (correlation robustness)，读者可参阅文献<sup>[46]</sup>。

<sup>12</sup>这是因为，如果使用的密钥有相同的偏移值，那么伪随机函数的定义无法保证加密方案的隐私性。

<sup>13</sup>具体而言，对于索引为  $i$  的门  $g_i$ ，如果  $H$  是随机谕示机， $\text{Enc}_{k_1, k_2}^i(m)$  可以实现为  $H(k_1 || k_2 || i) \oplus m$ ；如果  $H$  是伪随机函数， $\text{Enc}_{k_1, k_2}^i(m)$  可以实现为  $H(k_1 || i) \oplus H(k_2 || i) \oplus m$ ，这里  $H(k || i)$  表示伪随机函数  $H$  以  $k$  为密钥， $i$  为输入时的输出。当密钥都是随机选取的时候，这两种构造都是安全的。但是当密钥有关联性时（存在一个全局偏移值），必须假设  $H$  是随机谕示机（或满足关联健壮性）。

半门（生成方知道其中一个输入）和计算方半门（计算方知道其中一个输入）。因为已知一个输入，所以半门的混淆表只包含两条密文，并且可以通过 GRR3 技术缩减到一条。最后，两个半门的输出通过免费的异或运算结合起来，得到整个与门的输出。

对于导线  $w_i$ ，我们用  $W_i^0$  表示值 0 对应的标签。假设生成方半门要计算  $v_\ell = v_i \wedge v_j$ ，其中生成方已知  $v_i$  的值。如果  $v_i = 0$ ，则无论  $v_j$  为何值  $v_\ell$  一定为 0；如果  $v_i = 1$ ，则  $v_\ell = v_j$ 。于是，（未经过排列的）混淆表构造如下：

$$\begin{aligned} H(W_j^0) \oplus W_\ell^0 \\ H(W_j^1) \oplus W_\ell^0 \oplus v_i \Delta \end{aligned}$$

其中， $v_i \Delta$  表示：当  $v_i = 1$  时， $v_i \Delta = \Delta$ ，当  $v_i = 0$  时， $v_i \Delta = 0^\kappa$ 。生成方根据  $W_j$  的标识比特对密文进行排列，并使用 GRR 技术缩减一条密文。当计算方对该半门进行计算时，如果其持有  $W_j^0$ ，他将解密得到  $W_\ell^0$ ；如果其持有  $W_j^1$ ，他将解密得到  $W_\ell^0 \oplus v_i \Delta$ 。

我们再考虑计算方半门。假设计算方半门要计算  $v_\ell = v_i \wedge v_j$ ，其中计算方已知  $v_i$  的值。生成方不知道  $v_i, v_j$  中任何一个值，他构造如下的混淆表：

$$\begin{aligned} H(W_i^0) \oplus W_\ell^0 \\ H(W_i^1) \oplus W_\ell^0 \oplus W_j^0 \end{aligned}$$

此时，生成方不能打乱密文的顺序。当计算方对该半门进行计算时，如果他知道  $v_i = 0$ ，那么他就知道自己持有的是  $W_i^0$ ，他解密第 1 行得到  $W_\ell^0$ ；如果他知道  $v_i = 1$ ，那么他就知道自己持有的是  $W_i^1$ ，他首先解密第二行得到  $W_\ell^0 \oplus W_j^0$ ，再将这个结果与他持有的  $w_j$  的标签计算异或。不难发现，当他持有  $W_j^0$  时，得到的输出标签是  $W_\ell^0$ ；当他持有  $W_j^1$  时，得到的输出标签是  $W_\ell^0 \oplus \Delta = W_\ell^1$ ，即这样计算的结果是正确的。同样地，生成方使用 GRR 技术缩减一条密文。

最后，与门的计算需要合并两个半门。对于某个与门  $v_c = v_a \wedge v_b$ ，选取随机比特  $r$ ，由于与操作对异或操作满足分配律，它可以表示为  $v_c = (r \wedge v_a) \oplus ((r \oplus v_b) \wedge v_a)$ 。因为  $r$  是生成方随机选取的，生成方知道  $r$  的值，因此左边的与门  $(r \wedge v_a)$  是生成方半门。如果右边的与门  $((r \oplus v_b) \wedge v_a)$  能成为计算方半门，就完成了整个构造，但是这需要让计算方知道  $r \oplus v_b$ （注意， $r$  是随机比特， $r \oplus v_b$  不会泄露任何信息）。这里采用的方式是将  $r$  设置为导线  $w_b$  值 0 对应的标识比特  $p_b^0$ （标识比特本身就是随机选取的）。计算方持有  $W_b^{v_b}$ ，他不知道  $v_b$  的值，但是  $p_b^{v_b}$  就等于  $r \oplus v_b$ （当  $v_b = 0$  时， $p_b^{v_b} = p_b^0 = r \oplus 0 = r \oplus v_b$ ；当  $v_b = 1$  时， $p_b^{v_b} = p_b^1 = r \oplus 1 = r \oplus v_b$ ）。至此，生成方可以按照上文所述的方式分别构造生成方半门和计算方半门的混淆表，就完成了半门方案的构造。

半门方案可以扩展到任意的奇门（与门、或门、与非门、或非门）。对于奇门  $g$ ，可以表示为

$$g(v_a, v_b) = (\alpha_a \oplus v_a) \wedge (\alpha_b \oplus v_b) \oplus \alpha_c$$

（先计算与，再计算异或），其中  $\alpha_a, \alpha_b, \alpha_c$  是根据门的类型选择的（公开的）常数。例如， $\alpha_a, \alpha_b, \alpha_c$  全为 0 就构成了与门； $\alpha_a, \alpha_b, \alpha_c$  全为 1 就构成了或门。然后，我们可以将奇门的输出表示成两个半门输出的异或： $g(v_a, v_b) = f_G(v_a, r) \oplus f_E(v_a, r \oplus v_b)$ ，其中生成方半门为

$$f_G(v_a, r) = (v_a \oplus \alpha_a) \wedge (r \oplus \alpha_b) \oplus \alpha_c$$

计算方半门为

$$f_E(v_a, r \oplus v_b) = (v_a \oplus \alpha_a) \wedge (r \oplus v_b)$$

于是，生成方半门的（未经过排列的）混淆表相应地修改为

$$\begin{aligned} H(W_a^0) \oplus f_G(0, r)\Delta \oplus W_{G_c}^0 \\ H(W_a^1) \oplus f_G(1, r)\Delta \oplus W_{G_c}^0 \end{aligned}$$

计算方半门的混淆表相应地修改为

$$\begin{aligned} H(W_b^r) \oplus W_{E_c}^0 \\ H(W_b^{r \oplus 1}) \oplus W_{E_c}^0 \oplus W_a^{\alpha_a} \end{aligned}$$

其中， $w_{G_c}$  是生成方半门的输出导线， $w_{E_c}$  是计算方半门的输出导线。与上文中与门的例子类似，计算方将生成方半门的混淆表按照标识比特排列，不打乱计算方半门的混淆表，并通过 GRR 技术将混淆表缩减一行。

半门方案不仅同时实现了 2 行的混淆表和免费的异或计算，它的作者还证明了<sup>[47]</sup>：任何线性类混淆电路方案<sup>14</sup>中，奇门所需的密文数量不可能少于 2 个。因此，在线性假设下，半门方案是通讯开销最优的方案。

表 8.6 对本节介绍的混淆电路优化方案进行了总结。

表 8.6：混淆电路优化方案及其开销。列在标识置换之后的方案同样采用了标识置换技术。混淆表大小以表中密文数量为度量，忽略额外的常数项；计算开销以调用加密/解密算法的次数为度量。

方案	混淆表大小		计算开销			
			生成		计算	
	异或门	与门	异或门	与门	异或门	与门
经典方案 <sup>[39]</sup>	4	4	4	4	2.5	2.5
标识置换 <sup>[40]</sup>	4	4	4	4	1	1
4 → 3 混淆表行缩减 (GRR3) <sup>[44]</sup>	3	3	4	4	1	1
4 → 2 混淆表行缩减 (GRR2) <sup>[45]</sup>	2	2	4	4	1	1
free-XOR + GRR3 <sup>[46]</sup>	0	3	0	4	0	1
半门 <sup>[47]</sup>	0	2	0	4	0	2

## 8.2 BMR 协议

上一节介绍的姚氏混淆电路协议只适用于两个参与方的情形。如果希望拓展到  $n$  个参与方，其中每个参与方  $P_i$  拥有隐私输入  $x_i$ ，他们希望安全地计算  $f(x_1, \dots, x_n)$ 。令  $\mathcal{C}$  是函数  $f$  所对应的布尔电路，我们可能有以下直观的想法。

<sup>14</sup>直观地说，“线性”指的是混淆电路的生成和计算（除去对随机谕示机的请求）所需要对称密码学 (symmetric-key) 操作数量与电路大小成线性关系。

1.  $P_1$  生成混淆电路  $G(\mathcal{C})$ .
2.  $P_1$  与其他参与方  $P_2, \dots, P_n$  执行 OT 协议, 使  $P_2, \dots, P_n$  得到各自的输入所对应的标签 (密钥)。
3.  $P_1$  将混淆电路  $G(\mathcal{C})$  发给  $P_2$ , 参与方  $P_i, i \neq 2$  将自己的输入所对应的标签发给  $P_2$ .
4.  $P_2$  计算混淆电路, 得到  $f(x_1, \dots, x_n)$  并将其广播。

不难看出, 上述协议在只有一个参与方被攻陷的情况下确实是安全的。但是, 如果  $P_1$  和  $P_2$  同时被攻陷, 那么敌手就同时知道了生成混淆电路的秘密信息 (标签与导线值的对应关系) 和所有参与方的输入所对应的标签, 因而可以获得所有参与方的输入。这样的安全性是无法令人满意的, 我们希望协议能够与基于秘密分享的协议类似, 存在一个门限值  $t$ , 使得任意不超过  $t$  个参与方都不能从协议中获得任何额外信息。

不过, 我们可以从这个直观想法中得到一些启发: 协议不应该由某个参与方单独生成混淆电路, 而应该由所有参与方共同生成混淆电路, 使得任意不超过  $t$  个参与方都不能得到混淆电路的秘密信息 (导线标签与导线值的对应关系)。之前的章节讲过, 基于秘密分享的 MPC 协议 (例如 BGW、GMW) 可以进行任意通用的计算。我们发现, 混淆电路  $G(\mathcal{C})$  是关于所有导线标签的一个函数, 而输入值对应的导线标签是关于参与方输入以及输入导线标签的函数。那么, 通过秘密分享导线标签和输入值, 应当可以分布式地生成混淆电路和输入值对应的导线标签。

协议基本思路如下。对于每条导线  $w_i$ , 参与方  $P_j$  随机选择子标签  $s_{i,j}^0, s_{i,j}^1$ , 定义  $w_i$  的标签  $W_i^0 = s_{i,1}^0 \oplus \dots \oplus s_{i,n}^0, W_i^1 = s_{i,1}^1 \oplus \dots \oplus s_{i,n}^1$ .<sup>15</sup> 然后参与方调用 MPC 协议 (例如 GMW) 共同生成混淆电路。随后, 参与方再对输入进行秘密分享, 调用 MPC 协议共同计算输入值对应的导线标签。最后, 每个参与方基于混淆电路和输入值的导线标签, 计算混淆电路得到协议输出。

这样的协议确实使得不超过  $n - 1$  个被攻陷参与方无法获得额外信息, 但是其效率是很低的。注意到, 计算混淆电路时, 对于门  $g_i$ , 混淆表的每一项为  $H(k_a^\alpha || k_b^\beta || i) \oplus (k_c^{g_i(\alpha, \beta)} || p_c^{g_i(\alpha, \beta)})$  的形式, 其中  $H$  是哈希函数。按照上述思路,  $k_a^\alpha, k_b^\beta$  都是秘密分享的, 那么参与方需要对其进行复杂的哈希计算, 效率将非常低。也就是说, 以上方式并不是共同生成混淆电路的“好的”方式。

### 8.2.1 直观思想

BMR(Beaver-Micali-Rogaway) 协议<sup>[40]</sup>展示了一种“更好的”方式, 其思路是: 对于每条导线  $w_i$ , 参与方  $P_j$  随机选择子标签 (又称为 seed)  $s_{i,j}^0, s_{i,j}^1$ , 定义  $w_i$  的标签 (又称为 superseed)  $S_i^0 = s_{i,1}^0 || \dots || s_{i,n}^0, S_i^1 = s_{i,1}^1 || \dots || s_{i,n}^1$ , 也就是说, 标签是子标签的连接而不是异或。当使用  $S_i^0$  加密消息  $M$  时, 将其实现为  $M \bigoplus_{j=1}^n \mathcal{G}(s_{i,j}^0)$ , 其中  $\mathcal{G}$  是伪随机数生成器 (pseudo-random generator, PRG)。这样一来, 参与方  $P_j$  可以在本地计算  $\mathcal{G}(s_{i,j}^0)$ , 然后将其作为底层 MPC 协议的输入, 计算混淆表, 计算过程只需要对秘密分享的值进行简单的异或计算即可, 大大提升了运行效率。

然而, 这样的构造中, 从  $P_j$  的视角看, 每个标签中有一段子标签是  $P_j$  贡献的, 因此他可以知道标签对应的值是 0 还是 1, 这将直接泄露导线上的秘密值!

<sup>15</sup> 这里需要保证  $W_i^0$  和  $W_i^1$  的最后一位标识比特是相反的, 可以通过以下方式实现: 当参与方数量  $n$  为奇数时, 要求所有参与方选取的  $s_{i,j}^0, s_{i,j}^1$  最后一位都相反; 当参与方数量  $n$  为偶数时, 要求  $P_1$  选取的  $s_{i,1}^0, s_{i,1}^1$  最后一位相同, 其他参与方选取的  $s_{i,j}^0, s_{i,j}^1 (j \neq 1)$  最后一位相反。

为了解决这个问题，BMR 协议为每一条导线  $w_i$  均匀随机选取掩码比特 (masking bit)  $\lambda_i$ ，它掩盖了导线的真实值 (real value)  $v_i$ ，使得参与方只能知道导线的外部值 (external value)  $p_i$ 。三者满足  $p_i = v_i \oplus \lambda_i$ 。由于  $\lambda_i$  是均匀随机选择的，所以外部值不会泄露真实值。下面，我们进行更详细的阐述。

**看待标识置换的另一种视角。**之前介绍标识置换技术时，我们说标识置换为每一个密钥附加了一个标识比特。对于导线  $w_a$ ，密钥  $k_a^0$  附带了  $p_a^0$ ，密钥  $k_a^1$  附带了  $p_a^1$ ，且  $p_a^0 = 1 - p_a^1$ 。在计算混淆电路时，计算方持有  $w_a$  的一个标签  $k_a||p_a$ ，他并不知道标签对应的真实值，但是他能看到标识比特  $p_a$ ，因此， $p_a$  可以看成导线  $w_a$  的外部值（计算方可以看到的值）。外部值与真实值有两种可能的关系：相等或相反。由此可以定义掩码比特  $\lambda_a$ 。如果  $\lambda_a = 0$ ，那么外部值等于真实值；如果  $\lambda_a = 1$ ，那么外部值等于 1 减真实值。如果将  $w_a$  的真实值记为  $v_a$ ，外部值记为  $p_a$ ，掩码比特记为  $\lambda_a$ ，它们满足  $p_a = v_a \oplus \lambda_a$ 。也就是说，在这种视角下，我们把标识比特  $p_a$  看作导线的外部值，并定义掩码比特  $\lambda_a$ 。

接下来，让我们看看 BMR 协议中混淆表的构造方式。首先，对于每条导线  $w_i$ ，每个参与方  $P_j$ ,  $j \in [n]$  随机选择两个子标签 (seed)  $s_{i,j}^0, s_{i,j}^1 \leftarrow \{0,1\}^\kappa$  以及随机比特  $\lambda_{i,j} \leftarrow \{0,1\}$ 。子标签  $s_{i,j}^0, s_{i,j}^1$  分别对应导线的外部值 0 和 1。请注意，这里  $s_{i,j}^0$  对应的是外部值 0，而在之前章节使用的符号中， $k_i^0$  或  $W_i^0$  对应的是真实值 0，请读者注意这里的符号区别。定义  $w_i$  的标签 (superseed) 为  $S_i^0 = s_{i,1}^0 || \dots || s_{i,n}^0 || 0$ ,  $S_i^1 = s_{i,1}^1 || \dots || s_{i,n}^1 || 1$ ，定义  $w_i$  的掩码比特为  $\lambda_i = \lambda_{i,1} \oplus \dots \oplus \lambda_{i,n}$ 。也就是说，标签是子标签的连接再连接上外部值（标识比特）<sup>16</sup>，而掩码比特由所有参与方秘密分享。

混淆表通过如下方式构造。令  $\mathcal{G} : \{0,1\}^\kappa \rightarrow \{0,1\}^{2n\kappa+2}$  是输入长度为  $\kappa$ ，输出长度为  $2n\kappa + 2$  的伪随机数生成器（见第2.3.3节）。定义  $g_{i,j}^p = \mathcal{G}(s_{i,j}^p)[1:n\kappa+1]$ ,  $h_{i,j}^p = \mathcal{G}(s_{i,j}^p)[n\kappa+2:2n\kappa+2]$ ，即  $g_{i,j}^p$  是伪随机数生成器的前一半输出， $h_{i,j}^p$  是后一半输出。对于布尔门  $g$ ，假设其输入导线为  $w_a, w_b$ ，输出导线为  $w_c$ ，其混淆表如下。

$$\begin{aligned} A_g &= g_{a,1}^0 \oplus \dots \oplus g_{a,n}^0 \oplus g_{b,1}^0 \oplus \dots \oplus g_{b,n}^0 \oplus \begin{cases} S_c^0 & \text{if } g(\lambda_a, \lambda_b) = \lambda_c \\ S_c^1 & \text{otherwise} \end{cases} \\ B_g &= h_{a,1}^0 \oplus \dots \oplus h_{a,n}^0 \oplus g_{b,1}^1 \oplus \dots \oplus g_{b,n}^1 \oplus \begin{cases} S_c^0 & \text{if } g(\lambda_a, \overline{\lambda_b}) = \lambda_c \\ S_c^1 & \text{otherwise} \end{cases} \\ C_g &= g_{a,1}^1 \oplus \dots \oplus g_{a,n}^1 \oplus h_{b,1}^0 \oplus \dots \oplus h_{b,n}^0 \oplus \begin{cases} S_c^0 & \text{if } g(\overline{\lambda_a}, \lambda_b) = \lambda_c \\ S_c^1 & \text{otherwise} \end{cases} \\ D_g &= h_{a,1}^1 \oplus \dots \oplus h_{a,n}^1 \oplus h_{b,1}^1 \oplus \dots \oplus h_{b,n}^1 \oplus \begin{cases} S_c^0 & \text{if } g(\overline{\lambda_a}, \overline{\lambda_b}) = \lambda_c \\ S_c^1 & \text{otherwise} \end{cases} \end{aligned}$$

让我们分析一下混淆表为什么这样构造。首先，这里的  $A_g$  是用  $S_a^0$  和  $S_b^0$  加密  $S_c^0$ （或  $S_c^1$ ）得到的密文， $B_g$  是用  $S_a^0$  和  $S_b^1$  加密得到的密文， $C_g, D_g$  也同理。混淆表应当根据标识比特（即外部值）进行排列，而这里  $S_a^0, S_b^0, S_c^0$  对应的就是外部值 0（再次强调，之前章节中的  $k_a^0$  和  $W_a^0$  对应的是真实值 0，请一定注意符号区别），因此，混淆表  $(A_g, B_g, C_g, D_g)$  已经正确地排列了。其次，混淆表需要确保计算方得到导线  $w_c$  的正确标签，即真实值  $v_a, v_b, v_c$  应当满足  $g(v_a, v_b) = v_c$ 。由于真实值、外部值、掩码比特满足  $p_a = v_a \oplus \lambda_a$ ，我们有

<sup>16</sup> 我们在标签的最后附加了它的外部值，但是对于生成混淆电路的参与方来说，其实无需这么做，因为对于  $P_j$  来说，每个标签中有一段子标签是他自己选取的，因此他能够知道对应的外部值。我们仍然在标签后附加外部值的原因是，这使得生成的混淆电路可以由任何人计算，即使他没有参与混淆电路的生成。

$$g(p_a \oplus \lambda_a, p_b \oplus \lambda_b) = p_c \oplus \lambda_c$$

即

$$p_c = g(p_a \oplus \lambda_a, p_b \oplus \lambda_b) \oplus \lambda_c$$

对于混淆表的第一项  $A_g$ , 它对应于外部值  $p_a = 0, p_b = 0$  的情况, 因此当  $g(\lambda_a, \lambda_b) = \lambda_c$  时,  $p_c = 0$ , 计算方应该获得导线  $w_c$  外部值 0 对应的标签  $S_c^0$ ; 当  $g(\lambda_a, \lambda_b) \neq \lambda_c$  时,  $p_c = 1$ , 计算方获得导线  $w_c$  外部值 1 对应的标签  $S_c^1$ . 对于混淆表的第二项  $B_g$ , 它对应于外部值  $p_a = 0, p_b = 1$  的情况, 因此当  $g(\lambda_a, \overline{\lambda_b}) = \lambda_c$  时,  $p_c = 0$ , 计算方应该获得导线  $w_c$  外部值 0 对应的标签。 $C_g, D_g$  的构造也同理。换句话说, 混淆表本质上仍是姚氏混淆电路的混淆表, 区别是: (1) 符号表示有所不同, 这里  $S_i^0$  的上标代表外部值; (2) 加密方式有所不同, 当使用  $S_i^0$  加密消息  $M$  时, 将其实现为  $M \bigoplus_{j=1}^n \mathcal{G}(s_{i,j}^0)$ , 并且由于每个标签需要加密混淆表中的两项, 我们不能重复使用 PRG 的输出<sup>17</sup>, 因此需要取  $g_{i,j}^p = \mathcal{G}(s_{i,j}^p)[1 : n\kappa + 1], h_{i,j}^p = \mathcal{G}(s_{i,j}^p)[n\kappa + 2 : 2n\kappa + 2]$  分别为 PRG 输出的前一半和后一半。

可以发现, 对于每条导线  $w_i$ , 参与方  $P_j$  持有子标签  $s_{i,j}^0, s_{i,j}^1$  以及掩码比特的份额  $\lambda_{i,j}$ ,  $P_j$  可以计算  $g_{i,j}^0, h_{i,j}^0, g_{i,j}^1, h_{i,j}^1$ . 而门  $g$  (假设其输入导线为  $w_a, w_b$ , 输出导线为  $w_c$ ) 的混淆表是关于

$$\{\{g_{i,j}^0, h_{i,j}^0, g_{i,j}^1, h_{i,j}^1\}_{i \in \{a,b\}}, \{s_{c,j}^0, s_{c,j}^1\}, \{\lambda_{i,j}\}_{i \in \{a,b,c\}}\}_{j \in [n]}$$

以及布尔门类型的函数, 因此参与方可以调用 MPC 协议 (例如 BGW、GMW) 安全地计算所有混淆表。

最后, 协议还需要计算输入值对应的标签, 以及解密输出值。对于每条输入导线  $w_i$ , 其值  $v_i$  是某个参与方的输入。该参与方将  $v_i$  秘密分享, 然后所有参与方安全地计算  $w_i$  的外部值  $p_i = v_i \oplus \lambda_i$  ( $\lambda_i$  也是被秘密分享的) 并公布。然后, 每个参与方  $P_j$  广播  $s_{i,j}^{p_i}$ . 标签  $S_i^{p_i} = s_{i,1}^{p_i} || \dots || s_{i,n}^{p_i} || p_i$  即为该输入导线的标签。根据输入导线的标签, 计算方可以按照拓扑顺序解密每个混淆表, 得到所有输出导线的外部值。设某条输出导线为  $w_j$ , 计算方得到了它的外部值  $p_j$ , 所有参与方恢复出该输出导线的掩码比特  $\lambda_j$ , 于是,  $v_j = p_j \oplus \lambda_j$  就是该输出导线的值。

### 8.2.2 协议描述

我们给出 BMR 协议的正式描述, 如图 8.4 和图 8.5 所示。

**轮数复杂度 (round complexity).** 之前章节介绍的基于秘密分享的协议 (例如 BGW、GMW), 每个乘法门 (或与门) 都需要参与方之间的交互。协议由输入层向输出层一层层地计算, 因此, 协议的轮数与电路的深度成线性关系。不过, 对于姚氏混淆电路协议和 BMR 协议, 它们的轮数复杂度是常数! 即协议轮数与电路的深度无关。我们对此做一个简单的分析。

在姚氏混淆电路协议中, 电路生成方  $P_1$  和计算方  $P_2$  的交互包括: (1)  $P_1$  向  $P_2$  发送混淆电路; (2) 对于  $P_2$  的每个输入比特,  $P_1$  与  $P_2$  执行 OT 协议; (3)  $P_2$  将协议输出发给  $P_1$ . 其中, (1) 和 (3) 各需要一轮, 而 (2) 中的 OT 协议可以并行执行, 无论  $P_2$  的输入有多长, 协议轮数都是常数 (它等于 OT 协议的轮数 +2)。

<sup>17</sup> 如果重复使用了 PRG 的输出异或不同的消息  $M_1, M_2$ , 那么将得到密文  $C_1 = M_1 \oplus \mathcal{G}(S_i^p), C_2 = M_2 \oplus \mathcal{G}(S_i^p)$ . 参与方可计算  $C_1 \oplus C_2 = M_1 \oplus M_2$ . 这就形成了 two-time pad 问题。

## BMR 协议（第一部分）

**公共输入：**布尔电路  $\mathcal{C}$ , 满足  $\mathcal{C}(x_1, \dots, x_n) = f(x_1, \dots, x_n)$ . 伪随机数生成器  $\mathcal{G} : \{0,1\}^\kappa \rightarrow \{0,1\}^{2n\kappa+2}$ .

**输入：** $P_i$  的输入是  $x_i$ .

**协议：**协议分为两个阶段。

阶段 1:

1. 每个参与方  $P_i$  将输入  $x_i$  的每一个比特秘密分享。
2. 对于电路  $\mathcal{C}$  的每条导线  $w_i$ , 参与方  $P_j, j \in [n]$  随机选择两个子标签  $s_{i,j}^0, s_{i,j}^1 \leftarrow \$_{\{0,1\}^\kappa}$  以及随机比特  $\lambda_{i,j} \leftarrow \$_{\{0,1\}}$ . 定义  $w_i$  的标签为  $S_i^0 = s_{i,1}^0 || \dots || s_{i,n}^0 || 0$ ,  $S_i^1 = s_{i,1}^1 || \dots || s_{i,n}^1 || 1$ ,  $w_i$  的掩码比特为  $\lambda_i = \lambda_{i,1} \oplus \dots \oplus \lambda_{i,n}$ . 定义  $g_{i,j}^p = \mathcal{G}(s_{i,j}^p)[1 : n\kappa + 1]$ ,  $h_{i,j}^p = \mathcal{G}(s_{i,j}^p)[n\kappa + 2 : 2n\kappa + 2], j \in [n], p \in \{0,1\}$ .
3. 对于电路  $\mathcal{C}$  的每个门  $g$ , 假设其输入导线为  $w_a, w_b$ , 输出导线为  $w_c$ , 每个参与方  $P_j$  以  $\{g_{i,j}^0, h_{i,j}^0, g_{i,j}^1, h_{i,j}^1\}_{i \in \{a,b\}}, \{s_{c,j}^0, s_{c,j}^1\}, \{\lambda_{i,j}\}_{i \in \{a,b,c\}}$  为输入, 调用 MPC 协议 II 安全计算混淆表  $(A_g, B_g, C_g, D_g)$

$$A_g = g_{a,1}^0 \oplus \dots \oplus g_{a,n}^0 \oplus g_{b,1}^0 \oplus \dots \oplus g_{b,n}^0 \oplus \begin{cases} S_c^0 & \text{if } g(\lambda_a, \lambda_b) = \lambda_c \\ S_c^1 & \text{otherwise} \end{cases}$$

$$B_g = h_{a,1}^0 \oplus \dots \oplus h_{a,n}^0 \oplus g_{b,1}^1 \oplus \dots \oplus g_{b,n}^1 \oplus \begin{cases} S_c^0 & \text{if } g(\lambda_a, \overline{\lambda_b}) = \lambda_c \\ S_c^1 & \text{otherwise} \end{cases}$$

$$C_g = g_{a,1}^1 \oplus \dots \oplus g_{a,n}^1 \oplus h_{b,1}^0 \oplus \dots \oplus h_{b,n}^0 \oplus \begin{cases} S_c^0 & \text{if } g(\overline{\lambda_a}, \lambda_b) = \lambda_c \\ S_c^1 & \text{otherwise} \end{cases}$$

$$D_g = h_{a,1}^1 \oplus \dots \oplus h_{a,n}^1 \oplus h_{b,1}^1 \oplus \dots \oplus h_{b,n}^1 \oplus \begin{cases} S_c^0 & \text{if } g(\overline{\lambda_a}, \overline{\lambda_b}) = \lambda_c \\ S_c^1 & \text{otherwise} \end{cases}$$

4. 对于每条输入导线  $w_i$ , 假设其值为  $v_i$  (已经被秘密分享). 参与方安全计算  $p_i = v_i \oplus \lambda_i$  并公布。参与方  $P_j$  广播  $s_{i,j}^{p_i}$ . 设置  $S_i^{p_i} = s_{i,1}^{p_i} || \dots || s_{i,n}^{p_i} || p_i$  为输入导线  $w_i$  的标签。

阶段 2:

5. 参与方公布以下内容
  - (a) 对于每条输出导线  $w_j$ , 公布其掩码比特  $\lambda_j$ ;
  - (b) 第 3 步中计算的每个门  $g$  的混淆表;
  - (c) 第 4 步中计算的每条输入导线的标签。

5(a) 和 5(b) 构成混淆电路, 5(c) 为混淆输入 (garbled input)。

图 8.4: BMR 协议 (第一部分)

## BMR 协议（第二部分）

6. 每个参与方计算混淆电路，步骤如下：

- (a) 对于每条输入导线  $w_i$ , 参与方持有  $S_i^0 = s_{i,1}^0 \parallel \dots \parallel s_{i,n}^0 \parallel 0$  或  $S_i^1 = s_{i,1}^1 \parallel \dots \parallel s_{i,n}^1 \parallel 1$ .
- (b) 对于每个门  $g$ , 假设其输入导线为  $w_a, w_b$ , 输出导线为  $w_c$ . 假设参与方持有  $S_a^{p_a} = s_{a,1}^{p_a} \parallel \dots \parallel s_{a,n}^{p_a} \parallel p_a$  和  $S_b^{p_b} = s_{b,1}^{p_b} \parallel \dots \parallel s_{b,n}^{p_b} \parallel p_b$ . 令  $g_{a,j}^{p_a} = \mathcal{G}(s_{a,j}^{p_a})[1 : n\kappa + 1]$ ,  $h_{a,j}^{p_a} = \mathcal{G}(s_{a,j}^{p_a})[n\kappa + 2 : 2n\kappa + 2]$ ,  $g_{b,j}^{p_b} = \mathcal{G}(s_{b,j}^{p_b})[1 : n\kappa + 1]$ ,  $h_{b,j}^{p_b} = \mathcal{G}(s_{b,j}^{p_b})[n\kappa + 2 : 2n\kappa + 2]$ ,  $j \in [n]$ . 门  $g$  的混淆表为  $(A_g, B_g, C_g, D_g)$ . 参与方计算  $w_c$  的标签为

$$S_c^{p_c} = \begin{cases} g_{a,1}^{p_a} \oplus \dots \oplus g_{a,n}^{p_a} \oplus g_{b,1}^{p_b} \oplus \dots \oplus g_{b,n}^{p_b} \oplus A_g & \text{if } p_a = 0 \text{ and } p_b = 0 \\ h_{a,1}^{p_a} \oplus \dots \oplus h_{a,n}^{p_a} \oplus g_{b,1}^{p_b} \oplus \dots \oplus g_{b,n}^{p_b} \oplus B_g & \text{if } p_a = 0 \text{ and } p_b = 1 \\ g_{a,1}^{p_a} \oplus \dots \oplus g_{a,n}^{p_a} \oplus h_{b,1}^{p_b} \oplus \dots \oplus h_{b,n}^{p_b} \oplus C_g & \text{if } p_a = 1 \text{ and } p_b = 0 \\ h_{a,1}^{p_a} \oplus \dots \oplus h_{a,n}^{p_a} \oplus h_{b,1}^{p_b} \oplus \dots \oplus h_{b,n}^{p_b} \oplus D_g & \text{if } p_a = 1 \text{ and } p_b = 1 \end{cases}$$

- (c) 参与方按照拓扑顺序计算每一个门。对于每条输出导线  $w_j$ , 其标签  $S_j^{p_j}$  的最后一位为  $p_j$ . 计算  $v_j = p_j \oplus \lambda_j$  为输出值。

图 8.5: BMR 协议（第二部分）

对于 BMR 协议（如图 8.4 和图 8.5 所示），参与方的交互包括：将输入比特秘密分享（步骤 1）；计算每个门的混淆表（步骤 3）；计算每条输入导线的标签（步骤 4）。步骤 1 的秘密分享可以并行执行，只需 1 轮。我们着重分析步骤 3 和 4. 可以发现，无论电路规模有多大，门的混淆表和输入导线的标签都可以并行地计算。特别地，对于门  $g$  的混淆表（假设其输入导线为  $w_a, w_b$ , 输出导线为  $w_c$ ），它只与  $\{g_{i,j}^0, h_{i,j}^0, g_{i,j}^1, h_{i,j}^1, \lambda_{i,j}\}_{i \in \{a,b,c\}}$  和门的类型有关，与其他门的输入输出无关。而  $\{g_{i,j}^0, h_{i,j}^0, g_{i,j}^1, h_{i,j}^1, \lambda_{i,j}\}_{i \in \{a,b,c\}}$  是由参与方在步骤 2 中选择的随机子标签和随机比特定义的。因此，步骤 3 中的每个混淆表都可以并行地计算，只需要常数轮（取决于计算混淆表的电路与底层的 MPC 协议 II）。整个 BMR 协议也需要常数轮的交互。实际上，BMR 的原始论文<sup>[40]</sup>的题目是 *The Round Complexity of Secure Protocols*, 文章重点从理论上论证了：常数轮安全多方计算协议是存在的。

姚氏混淆电路协议和 BMR 协议只需常数轮交互的特点，使其在网络延迟较大的场景下，与基于秘密分享的协议相比常常具有性能优势。

### 8.2.3 安全性分析

最后，我们分析一下 BMR 协议的安全性。由于在计算混淆表时调用了底层 MPC 协议 II，因此 BMR 协议的安全性基于 II 的安全性。

模拟器  $\mathcal{S}$  的构造如下。

- 步骤 1 中， $\mathcal{S}$  模拟诚实方以 0 为输入进行秘密分享。
- 步骤 3 中， $\mathcal{S}$  调用底层 MPC 协议 II 的模拟器  $\mathcal{S}_{\Pi}$ ，使得步骤 3 输出的是一个“假的”混淆电路

$\tilde{G}(\mathcal{C})$ .<sup>18</sup> 它无论输入导线的标签是什么，输出总是  $f(x_1, \dots, x_n)$ .  $\tilde{G}(\mathcal{C})$  的构造思路与定理 8.4 完全类似，这里不再赘述。

- 步骤 4 中， $\mathcal{S}$  模拟诚实方遵循协议计算输入导线的标签。

不难发现，由于“假的”混淆电路  $\tilde{G}(\mathcal{C})$  与真实的混淆电路是不可区分的（详细证明可参考定理 8.4）， $\mathcal{S}$  模拟的视图与真实的视图不可区分。

BMR 协议重点论证了常数轮 MPC 协议的存在性。假设步骤 3 使用的是 BGW 协议，那么被攻陷方数量不能超过  $t$ ,  $t < n/2$ . 此时，BMR 协议只需要假设伪随机数生成器存在（以及安全信道，这是基本假设），就实现了常数轮的 MPC 协议。

**定理 8.6.** 假设参与方之间存在点对点安全信道。假设被攻陷方数量不超过  $t$ ,  $t < n/2$ . 假设伪随机数生成器存在。令  $f : (\{0, 1\}^\ell)^n \rightarrow (\{0, 1\}^\ell)^n$  是一个函数， $\mathcal{C}$  是  $f$  对应的布尔电路。那么，存在 const 轮的协议  $\Pi$  对于静态半诚实对手安全实现了  $f$ ，其中 const 是一个常数。此外，协议  $\Pi$  的描述可以由一个固定的算法在关于  $|\mathcal{C}|$  的多项式时间之内计算；协议  $\Pi$  的计算复杂度是  $O(\text{poly}(|\mathcal{C}|, \kappa))$ ，其中  $\text{poly}$  是一个固定的多项式， $\kappa$  是安全参数。

---

<sup>18</sup> 读者可以回顾 BGW 协议、GMW 协议的模拟器，它可以基于协议输出模拟被攻陷方的视图。因此这里可以调用  $\mathcal{S}_\Pi$  使步骤 3 输出  $\tilde{G}(\mathcal{C})$ .



# Chapter 9

## 恶意安全性

本章，我们介绍恶意安全的协议。我们首先介绍 *GMW* 编译器。这是一个通用的编译器，它使用零知识证明技术 (zero-knowledge proofs)，可以将任意半诚实安全的协议转化为恶意安全的协议。接着，我们介绍切分选择技术 (*cut-and-choose*)，它的思想是让某个参与方（假设为  $P_1$ ）生成多个副本，然后由另一个参与方（假设为  $P_2$ ）随机选择一部分副本让  $P_1$  打开，如果这部分都是正确的，双方才使用剩余的副本继续执行协议，从而避免  $P_1$  做出违背协议的行为。使用切分选择技术使姚氏混淆电路协议获得恶意安全性，是切分选择的一个典型应用。接下来我们介绍恶意安全的 *BGW* 协议。在 Ben-Or, Goldwasser, Wigderson(BGW) 的论文中<sup>[33]</sup>，对于半诚实敌手和恶意敌手分别设计了协议，而第6.1节介绍的是半诚实安全的协议。恶意安全的 *BGW* 协议使用了可验证秘密分享 (verifiable secret sharing)，对于  $t < n/3$  的（恶意的）被攻陷方，实现了完美的安全性。最后，我们介绍 *BDOZ* 和 *SPDZ* 协议。它们使用了消息认证码 (message authentication codes, MACs) 以获得恶意安全性。

### 9.1 GMW 编译器

*GMW* 编译器<sup>[20]</sup>使用零知识证明技术，可以将任意的半诚实安全的协议转化为恶意安全的协议。零知识证明是密码学中的一个重要工具，它可以让证明者 (prover) 向验证者 (verifier) 证明某个陈述 (statement) 为真，但不泄露陈述的正确性以外的任何信息。这个工具乍一看可能会令读者感到非常神奇和神秘，我们先通过一个简单的例子来带领读者了解零知识证明。

一个零知识证明的例子。形式化地说，零知识证明是一个两方协议，其中一方为证明者，另一方为验证者。证明者希望证明某个陈述为真。我们将这个陈述记为  $x$ 。在数学上，陈述为真意味着存在见证 (witness)  $w$ ，满足  $\mathcal{R}(x, w) = 1$ ，其中  $\mathcal{R}$  是一个二元 NP 关系 (relation)。而证明过程不能向验证者泄露  $w$  以及其他任何信息。

下面，我们给出一个零知识证明的具体例子。它的公共陈述是一个 Pedersen 承诺和一个 Lifted ElGamal 密文，证明者要证明承诺的值和加密的值是相同的。这个协议可能的具体场景是这样的：参与方首先用 Pedersen 承诺方案对自己的输入  $m$  进行承诺，然后协议的第一步需要对输入  $m$  加密，参与方需要证明自己遵循协议加密了承诺的值，当然，他不能泄露自己的输入。

我们在第2章已经介绍过 Pedersen 承诺（第2.3.6节）和 ElGamal 加密方案（第2.1.3节）。简单回顾一下。首先，公共参数包括一个阶为  $q$  的循环群  $\mathbb{G}$ 。我们将 Pedersen 承诺的承诺密钥记为  $\text{ck} \in \mathbb{G}$ ，

Pedersen 承诺定义为  $\text{Com}_{\text{ck}}(m) = g^m \text{ck}^r$ , 其中  $g$  是群  $\mathbb{G}$  的生成元,  $r \leftarrow_{\$} \mathbb{Z}_q$  是均匀随机选择的。Lifted ElGamal 加密是 ElGamal 加密的变体。我们将公钥记为  $\text{pk}, \text{pk} = g^{\text{sk}}$ 。Lifted ElGamal 加密定义为  $\text{Enc}_{\text{pk}}(m) = (c_1, c_2) = (g^r, g^m \text{pk}^r)$ , 其中  $g$  是群  $\mathbb{G}$  的生成元,  $r \leftarrow_{\$} \mathbb{Z}_q$  是均匀随机选择的。不难看出, Pedersen 承诺和 Lifted ElGamal 加密都具有加法同态的性质。此外, 由于对 Lifted ElGamal 密文解密时需要计算  $\text{DLog}(c_2/c_1^{\text{sk}}) = m$ , 因此实际使用中  $m$  的值不能太大。

我们将证明者的 Pedersen 承诺记为  $c, c = g^m \text{ck}^s$ ; 将 Lifted ElGamal 密文记为  $(c_1, c_2), (c_1, c_2) = (g^t, g^m \text{pk}^t)$ 。验证者对于  $c$  中的承诺值和  $(c_1, c_2)$  中的加密值一无所知 (根据 Pedersen 承诺的隐藏性和 ElGamal 加密的 IND-CPA 安全性), 证明者知道  $m, s, t$  (但不知道私钥  $\text{sk}$ ), 他希望证明承诺值等于加密值, 并且不泄露任何额外信息 (特别是不能泄露  $m$ )。

在这个场景下, 陈述是所有公开的信息, 包括  $(g, \text{ck}, \text{pk}, c, c_1, c_2)$ , 见证是证明者的私有信息  $m, s, t$ , 关系  $\mathcal{R}$  的定义如下

$$\mathcal{R} = \{(g, \text{ck}, \text{pk}, c, c_1, c_2), m, s, t) \mid c = g^m \text{ck}^s \wedge c_1 = g^t \wedge c_2 = g^m \text{pk}^t\}$$

我们给出关系  $\mathcal{R}$  的零知识证明协议, 如图 9.1 所示。

零知识证明需要满足三个性质: 完备性 (completeness), 可靠性 (soundness) 和零知识性 (zero-knowledge)。完备性指的是: 如果证明者和验证者都是诚实的, 那么验证者接受证明的概率为 1; 可靠性指的是: 如果验证者接受了证明者的证明, 那么 (以压倒性的概率) 陈述为真; 零知识性指的是: 验证者不能获得除了陈述为真以外的任何信息。

我们发现, 图 9.1 的协议分为 3 步: 证明者先发送初始消息  $a = (a_1, a_2, a_3)$ , 验证者再发送随机值  $e$ , 然后证明者发送回答消息  $z = (z_1, z_2, z_3)$ . 其中, 验证者发送的消息是随机值, 与证明者发送的消息无关。如果零知识证明协议中验证者的消息都是均匀随机选取的, 那么它被称为公共硬币 (*public-coin*) 的协议。像图 9.1 这类 3 步的公共硬币零知识证明协议被称为 Sigma 协议 (*Sigma protocols*)。下面, 我们形式化地定义 Sigma 协议以及它的完备性、可靠性、零知识性。

**定义 9.1.1.** Sigma 协议是一类 3 步公共硬币证明系统。对于关系  $\mathcal{R}$  的 Sigma 协议由三个算法  $(\mathcal{C}, \mathcal{Z}, \mathcal{V})$  组成, 且满足下述定义的完备性、2-特殊可靠性 (*2-special soundness*)、诚实验证者零知识性 (*special honest verifier zero-knowledge*):

- $a \leftarrow \mathcal{C}(x, w; r)$ : 该算法以陈述  $x$ 、见证  $w$  和随机硬币  $r$  作为输入, 输出初始消息  $a$ .
- $z \leftarrow \mathcal{Z}(x, w, r, e)$ : 该算法以陈述  $x$ 、见证  $w$ 、随机硬币  $r$  以及挑战值  $e \in \{0, 1\}^\lambda$  作为输入, 输出回答消息  $z$ .
- $0/1 \leftarrow \mathcal{V}(x, a, e, z)$ : 该算法以陈述  $x$ 、初始消息  $a$ 、挑战值  $e$  以及回答消息  $z$  作为输入, 输出 0 或 1 分别代表拒绝或接受。
- 完备性: 对于所有  $(x, w) \in \mathcal{R}, e \in \{0, 1\}^\lambda$ , 有

$$\Pr[a \leftarrow \mathcal{C}(x, w; r); z \leftarrow \mathcal{Z}(x, w, r, e) : \mathcal{V}(x, a, e, z) = 1] = 1$$

- 2-特殊可靠性: 存在确定性多项式时间提取器  $\mathcal{X}$ , 使得对任意 PPT 的敌手  $\mathcal{A}$  都有

$$\Pr \left[ \begin{array}{l} (x, a) \leftarrow \mathcal{A}(1^\lambda); e, e' \leftarrow_{\$} \{0, 1\}^\lambda; \\ z, z' \leftarrow \mathcal{A}(e, e'); w \leftarrow \mathcal{X}(x, a, \{e, z\}, \{e', z'\}) \end{array} : \begin{array}{l} e \neq e' \wedge \mathcal{V}(x, a, e, z) = 1 \wedge \\ \mathcal{V}(x, a, e', z') = 1 \wedge (x, w) \notin \mathcal{R} \end{array} \right] = 0$$

### 关系 $\mathcal{R}$ 的零知识证明协议

**陈述:**  $g, \mathbf{ck}, \mathbf{pk}, c, c_1, c_2 \in \mathbb{G}$ .

**见证:**  $m, s, t$  s.t.  $c = g^m \mathbf{ck}^s \wedge c_1 = g^t \wedge c_2 = g^m \mathbf{pk}^t$ .

**协议:**

证明者  $P$  执行如下操作:

- $m', s', t' \leftarrow_{\$} \mathbb{Z}_q$ ;
- 计算  $a_1 = g^{m'} \mathbf{ck}^{s'}, a_2 = g^{t'}, a_3 = g^{m'} \mathbf{pk}^{t'}$ ;
- $P \rightarrow V : a_1, a_2, a_3$ .

验证者  $V$  执行如下操作:

- $e \leftarrow_{\$} \mathbb{Z}_q$ ;
- $V \rightarrow P : e$ .

证明者  $P$  执行如下操作:

- 计算  $z_1 = m' + e \cdot m, z_2 = s' + e \cdot s, z_3 = t' + e \cdot t$ ;
- $P \rightarrow V : z_1, z_2, z_3$ .

验证者  $V$  验证以下等式是否成立, 如果都成立, 则输出 1 (表示接受), 否则输出 0 (表示拒绝):

- $g^{z_1} \mathbf{ck}^{z_2} \stackrel{?}{=} a_1 \cdot c^e$ ;
- $g^{z_3} \stackrel{?}{=} a_2 \cdot c_1^e$ ,
- $g^{z_1} \mathbf{pk}^{z_3} \stackrel{?}{=} a_3 \cdot c_2^e$ .

图 9.1: 关系  $\mathcal{R}$  的零知识证明协议

- 诚实验证者零知识性：存在 PPT 的算法  $\mathcal{S}$ ，使得对任意 PPT 的敌手  $\mathcal{A}$  都有

$$\begin{aligned} & \Pr[(x, e) \leftarrow \mathcal{A}(1^\lambda); a \leftarrow \mathcal{C}(x, w; r); z \leftarrow \mathcal{Z}(x, w, r, e) : \mathcal{A}(a, z) = 1] \\ & \approx \Pr[(x, e) \leftarrow \mathcal{A}(1^\lambda); (a, z) \leftarrow \mathcal{S}(x, e) : \mathcal{A}(a, z) = 1] \end{aligned}$$

其中  $r$  是均匀随机选取的， $\mathcal{A}$  输出的值必须满足  $(x, w) \in \mathcal{R}$  和  $e \in \{0, 1\}^\lambda$ .

完备性很容易理解：在证明者和验证者都诚实执行协议的情况下，验证者接受证明的概率为 1. 2-特殊可靠性的直观含义是：给定陈述  $x$ ，如果对于同一个初始消息  $a$ ，有两组可以通过验证的会话  $(a, e, z)$  和  $(a, e', z')$ ，那么提取器  $\mathcal{X}$  就能够提取出见证  $w$ . 换句话说，我们可以构造这样的提取器，它先与证明者  $P$  执行协议，得到  $(a, e, z)$ ，再将  $P$  倒带 (rewind) 到第 2 步，发送一个新的挑战值  $e'$ ，得到第二组会话  $(a, e', z')$ ，即可提取出见证  $w$ . 诚实验证者零知识性要求存在一个模拟器  $\mathcal{S}$ ，给定挑战值  $e$ ， $\mathcal{S}$  需要在不知道见证  $w$  的情况下模拟协议会话  $(a, e, z)$ ，且与真实的协议会话不可区分。因为  $\mathcal{S}$  不知道  $w$ ，但可以模拟出与真实会话不可区分的协议会话，那么我们可以知道：真实的协议会话不会泄露  $w$ .

从模拟器  $\mathcal{S}$  的定义中，我们可以看出消息顺序的微妙之处。在实际执行协议时，证明者必须先发送初始消息  $a$ ，再接收随机挑战值  $e$ ，最后发送回答消息  $z$ ；而模拟器是先知道了挑战值  $e$ ，再模拟出消息  $a$  和  $z$ . 也就是说，证明者在发送初始消息  $a$  的时候必须不知道挑战值  $e$ ，协议才有可靠性；而如果先知道了挑战值  $e$ ，模拟器就可以在不知道见证  $w$  的情况下模拟出协议会话  $(a, e, z)$ ，这体现了协议的零知识性。

下面，我们证明图 9.1 中的协议的安全性。

**定理 9.1.** 图 9.1 中的协议是对关系  $\mathcal{R}$  的 Sigma 协议，满足完备性、2-特殊可靠性、诚实验证者零知识性。

证明. 首先证明完备性。当证明者和验证者均诚实执行协议时，有

$$\begin{aligned} a_1 \cdot c^e &= g^{m'} \mathsf{ck}^{s'} \cdot (g^m \mathsf{ck}^s)^e = g^{m'+e \cdot m} \mathsf{ck}^{s'+e \cdot s} = g^{z_1} \mathsf{ck}^{z_2} \\ a_2 \cdot (c_1)^e &= g^{t'} \cdot (g^t)^e = g^{t'+e \cdot t} = g^{z_3} \\ a_3 \cdot (c_2)^e &= g^{m'} \mathsf{pk}^{t'} \cdot (g^m \mathsf{pk}^t)^e = g^{m'+e \cdot m} \mathsf{pk}^{t'+e \cdot t} = g^{z_1} \mathsf{pk}^{z_3} \end{aligned}$$

完备性得证。

接着证明 2-特殊可靠性。如果对于初始消息  $a_1, a_2, a_3$ ，有  $(e, z_1, z_2, z_3)$  和  $(e', z'_1, z'_2, z'_3)$  都能通过验证，根据验证等式，有

$$g^{z_1} \mathsf{ck}^{z_2} = a_1 \cdot c^e \quad (9.1)$$

$$g^{z_3} = a_2 \cdot c_1^e \quad (9.2)$$

$$g^{z_1} \mathsf{pk}^{z_3} = a_3 \cdot c_2^e \quad (9.3)$$

$$g^{z'_1} \mathsf{ck}^{z'_2} = a_1 \cdot c^{e'} \quad (9.4)$$

$$g^{z'_3} = a_2 \cdot c_1^{e'} \quad (9.5)$$

$$g^{z'_1} \mathsf{pk}^{z'_3} = a_3 \cdot c_2^{e'} \quad (9.6)$$

将式 (9.1) 除以式 (9.4)，得到

$$g^{z_1 - z'_1} \mathsf{ck}^{z_2 - z'_2} = c^{e - e'}$$

即

$$c = g^{(z_1 - z'_1)(e - e')^{-1}} \mathbf{ck}^{(z_2 - z'_2)(e - e')^{-1}} \quad (9.7)$$

同理, 将式 (9.2) 除以式 (9.5), 得到

$$g^{z_3 - z'_3} = c_1^{e - e'}$$

即

$$c_1 = g^{(z_3 - z'_3)(e - e')^{-1}} \quad (9.8)$$

最后, 将式 (9.3) 除以式 (9.6), 得到

$$g^{z_1 - z'_1} \mathbf{pk}^{z_3 - z'_3} = c_2^{e - e'}$$

即

$$c_2 = g^{(z_1 - z'_1)(e - e')^{-1}} \mathbf{pk}^{(z_3 - z'_3)(e - e')^{-1}} \quad (9.9)$$

由式 (9.7)、式 (9.8) 和式 (9.9), 我们提取出了见证  $m = (z_1 - z'_1)(e - e')^{-1}, s = (z_2 - z'_2)(e - e')^{-1}, t = (z_3 - z'_3)(e - e')^{-1}$  满足  $c = g^m \mathbf{ck}^s \wedge c_1 = g^t \wedge c_2 = g^m \mathbf{pk}^t$ .

最后证明诚实验证者零知识性。我们需要构造算法  $\mathcal{S}$ , 给定挑战值  $e$ , 输出  $(a_1, a_2, a_3, z_1, z_2, z_3)$ , 使得  $(a_1, a_2, a_3, e, z_1, z_2, z_3)$  与真实会话不可区分。 $\mathcal{S}$  的构造如下。它的输入是挑战值  $e$ , 它均匀随机选取  $z_1, z_2, z_3 \leftarrow \mathbb{Z}_q$ , 然后计算  $a_1 = \frac{g^{z_1} \mathbf{ck}^{z_2}}{c^e}, a_2 = \frac{g^{z_3}}{c_1^e}, a_3 = \frac{g^{z_1} \mathbf{pk}^{z_3}}{c_2^e}$ , 然后输出  $(a_1, a_2, a_3, z_1, z_2, z_3)$ . 下面我们论证  $\mathcal{S}$  模拟的会话与真实会话有相同的分布。首先, 无论是模拟的会话还是真实会话,  $e$  都是均匀随机值。 $\mathcal{S}$  模拟的会话中,  $z_1, z_2, z_3$  是均匀随机选取的, 真实协议中,  $z_1 = m' + e \cdot m, z_2 = s' + e \cdot s, z_3 = t' + e \cdot t$ , 由于  $m', s', t'$  是均匀随机选取的, 所以真实会话中的  $z_1, z_2, z_3$  也是均匀随机值。最后, 根据验证等式  $g^{z_1} \mathbf{ck}^{z_2} = a_1 \cdot c^e, g^{z_3} = a_2 \cdot c_1^e, g^{z_1} \mathbf{pk}^{z_3} = a_3 \cdot c_2^e$ , 在给定  $e$  和  $z_1, z_2, z_3$  的情况下,  $a_1, a_2, a_3$  的值也唯一确定。故  $\mathcal{S}$  模拟的会话与真实会话有相同的分布。

证毕。 □

从这个简单的例子可以看出, 零知识证明确实是可以构造的, 它并不神秘。实际上, 对于任意的 NP 关系, 都存在零知识证明协议<sup>[48]</sup>。因此, 在安全多方计算中, 参与方也可以通过零知识证明协议来证明自己诚实地遵循协议执行。

**GMW 编译器。**给定一个半诚实安全的协议  $\Pi$ , GMW 编译器将其转化为恶意安全的协议  $\Pi'$ .  $\Pi'$  的执行过程如下。

1. **输入承诺:** 每个参与方对自己的输入值进行承诺。即参与方  $P_i$  计算承诺值  $c_i = \text{Com}(x_i, \rho_i)$ , 其中  $x_i$  是  $P_i$  的输入,  $\rho_i$  是随机数。 $P_i$  将  $c_i$  公开。
2. **抛硬币:** 每个参与方  $P_i$  均匀选择随机数  $r_{i,i}$  并承诺, 将承诺公开。其他参与方  $P_j, j \neq i$  均匀选择随机数  $r_{i,j}$  并承诺。所有参与方完成承诺后,  $P_j, j \neq i$  将  $r_{i,j}$  打开。协议执行过程中,  $P_i$  使用  $r_i = \bigoplus_{j=1}^n r_{i,j}$  作为随机数。这样的构造使得只要有一个参与方是诚实的, 那么  $r_i$  就是均匀随机的。
3. **执行协议:** 在协议执行过程中, 当参与方  $P_i$  发送消息时, 通过零知识证明协议证明该消息是使用  $x_i$  为输入,  $r_i$  为随机数, 诚实地遵循协议执行得到的。

GMW 编译器给出了一个理论上完美的结论: 任何半诚实安全的协议都可以转化为恶意安全的协议。但是, 使用这样的通用方法得到的协议往往运行效率较低, 不适用于实际场景。

## 9.2 切分选择

### 9.2.1 直观思想

姚氏混淆电路协议对于半诚实的敌手是安全的，但是无法抵御恶意敌手。具体而言，协议中电路生成方  $P_1$  需要为电路  $C$  生成混淆电路，恶意的  $P_1$  可以生成另一个电路  $C'$  的混淆电路并发送给  $P_2$ 。对  $C'$  求值可能会泄露额外的信息，甚至直接泄露  $P_2$  的输入！

切分选择 (cut-and-choose) 技术<sup>1</sup>的基本思想是： $P_1$  对电路  $C$  生成多个混淆电路的副本，然后  $P_2$  随机选择一部分要求  $P_1$  打开。只要有一个被打开的电路是错误的， $P_2$  就知道  $P_1$  在作弊，协议中止。如果打开的电路都是正确的，那么  $P_1$  和  $P_2$  使用未被打开的电路继续执行协议，此时， $P_2$  有信心认为未被打开的电路大多数是正确的。

**如果电路的计算结果不一致？**当  $P_1$  和  $P_2$  使用切分选择技术执行协议时，假设  $P_1$  打开的电路全部都是正确的，那么他们将继续执行协议，对所有未打开的电路进行计算。此时，有两种可能的情况：(1) 所有电路的计算结果都一致；(2) 电路的计算结果不一致。对于情况 1， $P_2$  可以将这个结果作为协议的输出发送给  $P_1$ 。对于情况 2，出现不一致的计算结果的唯一可能就是  $P_1$  在作弊，我们的第一反应也许是： $P_2$  应该立刻中止协议。

然而，这样做是不安全的， $P_1$  有如下的攻击方式。假设协议中  $P_1$  需要生成  $s$  个混淆电路， $P_2$  随机选择  $s/2$  个电路让  $P_1$  打开。 $P_1$  构造一个特殊的混淆电路，当  $P_2$  的输入的第一个比特为 1 时，它会输出错误的结果，如果  $P_2$  的输入的第一个比特为 0，那么它的输出是正确的。然后，对于其余  $s - 1$  个混淆电路， $P_1$  正确地生成。此时，当  $P_2$  随机选择  $s/2$  个电路时，这个错误的混淆电路被检测到的概率为  $1/2$ 。如果它没有被检测到，协议将继续运行，这会泄露  $P_2$  的输入的第一个比特：如果这个比特是 1，电路将出现不一致的计算结果， $P_2$  将中止协议；如果这个比特是 0， $P_2$  将完成协议运行。也就是说， $P_1$  有  $1/2$  的概率可以获得  $P_2$  的输入的第一个比特。

这个例子告诉我们，即使  $P_2$  知道  $P_1$  在作弊，他也不能直接中止协议，而是应该输出大多数电路的计算结果。我们来计算一下此时  $P_1$  攻击成功的概率。假设  $P_1$  生成  $s$  个混淆电路， $P_2$  随机选择  $s/2$  个电路让  $P_1$  打开。我们将打开的电路全部正确记为事件 noAbort，将计算的电路有一半或更多错误电路记为事件 badMaj，将错误电路的数量记为 badTotal。那么，当打开的  $s/2$  个电路全都正确且计算的  $s/2$  个电路有一半或更多错误的概率为

$$\begin{aligned} \Pr[\text{noAbort} \wedge \text{badMaj}] &= \sum_{i=s/4}^{s/2} \Pr[\text{noAbort} \wedge \text{badTotal} = i] \leq \sum_{i=s/4}^{s/2} \frac{\binom{s-i}{s/2}}{\binom{s}{s/2}} = \frac{1}{\binom{s}{s/2}} \sum_{i=s/4}^{s/2} \binom{s-i}{s/2} \\ &= \frac{1}{\binom{s}{s/2}} \sum_{i=0}^{s/4} \binom{s/2+i}{s/2} = \frac{1}{\binom{s}{s/2}} \sum_{i=0}^{3s/4} \binom{i}{s/2} = \frac{1}{\binom{s}{s/2}} \binom{3s/4+1}{s/2+1} \end{aligned}$$

其中，倒数第二个等号是因为当  $i < s/2$  时， $\binom{i}{s/2} = 0$ 。最后一个等号是因为  $\sum_{k=0}^n \binom{k}{m} = \binom{n+1}{m+1}$ ，这个公式的证明参见书<sup>[50]</sup>，第 174 页。接下来，我们推导其上界。

<sup>1</sup> cut-and-choose 这个名称，最初类比的是互不信任的双方公平地分蛋糕的方法<sup>[49]</sup>：一方将蛋糕切为两块 (cut)，另一方选择自己的那份 (choose)。姚氏混淆电路协议 cut-and-choose 的过程与分蛋糕已经有所不同。

$$\begin{aligned}
\frac{\binom{\frac{3s}{4}+1}{\frac{s}{2}+1}}{\binom{s}{\frac{s}{2}}} &= \frac{(\frac{3s}{4}+1)!}{(\frac{s}{2}+1)!(\frac{s}{4})!} \cdot \frac{(\frac{s}{2})!(\frac{s}{2})!}{s!} = \frac{(\frac{3s}{4}+1)!}{s!} \cdot \frac{(\frac{s}{2})!}{(\frac{s}{4})!} \cdot \frac{(\frac{s}{2})!}{(\frac{s}{2}+1)!} \\
&= \frac{(\frac{s}{2})(\frac{s}{2}-1)\cdots(\frac{s}{4}+1)}{s(s-1)\cdots(\frac{3s}{4}+2)} \cdot \frac{1}{(\frac{s}{2}+1)} \\
&= \frac{(\frac{s}{2})}{s} \cdot \frac{(\frac{s}{2}-1)}{s-1} \cdots \frac{(\frac{s}{4}+2)}{(\frac{3s}{4}+2)} \cdot \frac{(\frac{s}{4}+1)}{(\frac{s}{2}+1)}
\end{aligned}$$

令  $t = s/4$ , 上式等于

$$\frac{2t}{4t} \cdot \frac{2t-1}{4t-1} \cdots \frac{t+2}{3t+2} \cdot \frac{t+1}{2t+1} = \left( \prod_{i=2}^t \frac{t+i}{3t+i} \right) \cdot \frac{t+1}{2t+1}$$

对于所有的  $i < t$ , 都有  $\frac{t+i}{3t+i} < \frac{1}{2}$ , 故上式的上界为  $\frac{1}{2^{t-1}}$ , 因此我们有

$$\Pr[\text{noAbort} \wedge \text{badMaj}] < \frac{1}{2^{s/4-1}}$$

如果我们希望这个概率小于  $2^{-40}$ , 只需取  $s = 164$  即可。

**输入一致性 (input consistency)**。在基于切分选择技术的协议中,  $P_2$  要对多个混淆电路的副本进行计算。我们需要确保每个混淆电路的输入是相同的, 即输入一致性。 $P_2$  的输入一致性比较容易实现: 对于  $P_2$  的每个输入比特,  $P_1$  和  $P_2$  仅执行一次 OT 协议一次性地发送所有混淆电路中该输入导线的密钥。但是,  $P_1$  有可能在不同的混淆电路提供不同的输入, 从而试图获取额外信息。例如, 假设  $P_1$  和  $P_2$  的输入为  $x, y \in \{0, 1\}^n$ , 他们希望计算  $x$  和  $y$  的内积, 即  $f(x, y) = \langle x, y \rangle = x_1y_1 + \cdots + x_ny_n$ , 其中  $x_i$  是  $x$  的第  $i$  位,  $y_i$  是  $y$  的第  $i$  位。假设  $P_1$  和  $P_2$  恰好需要计算  $n$  个混淆电路。对于第  $i$  个混淆电路,  $P_1$  将其输入的第  $i$  位设置为 1, 其余位为设置为 0. 于是,  $P_2$  计算第  $i$  个电路得到的结果就是  $y_i$ . 然后,  $P_2$  输出大多数电路的结果, 这将泄露  $P_2$  的输入中大多数比特是 0 还是 1.

零知识证明可以解决  $P_1$  的输入一致性问题。一种可行 (但不太高效的) 方法如下: 假设  $P_1$  生成了  $s$  个混淆电路。在进行切分选择检查之前, 对于  $P_1$  的每条输入导线,  $P_1$  对每个电路中该导线值 0 对应的密钥和 1 对应的密钥进行承诺, 所有承诺值构成了“0 集合”和“1 集合” (“0 集合”和“1 集合”的大小均为  $s$ ). 然后,  $P_2$  随机选择  $s/2$  个电路让  $P_1$  打开。 $P_1$  在打开电路的同时也打开“0 集合”和“1 集合”中对应电路的密钥,  $P_2$  检查打开的电路导线值 0 的密钥确实在“0 集合”中, 导线值 1 的密钥确实在“1 集合”中。最后, 当  $P_1$  向  $P_2$  发送  $P_1$  的输入在未打开的电路中的密钥时,  $P_1$  用零知识证明协议证明: 这些密钥都在“0 集合”中或这些密钥都在“1 集合”中。不难看出, 这样的构造确保了用于计算的大多数电路中,  $P_1$  的输入是一致的。然而, 证明密钥都在“0 集合”中或都在“1 集合”中的零知识证明协议是比较低效的, 我们将在第9.2.2节介绍一种更高效的零知识证明构造。

**选择性中止攻击 (selective abort attack/selective failure attack)**。现在, 我们通过零知识证明确保了  $P_1$  的输入一致性。但是,  $P_2$  的输入对应的密钥需要  $P_1$  在 OT 协议中提供, 恶意的  $P_1$  可以在 OT 协议中提供错误的输入来获取额外信息。假设  $w_i$  是  $P_2$  的一条输入导线, 它的密钥是  $k_i^0, k_i^1$ ,  $P_1$  应该将  $(k_i^0, k_i^1)$  作为 OT 协议的输入<sup>2</sup>。但是,  $P_1$  可以将 OT 协议的输入设置为  $(k_i^0, 0)$ . 这样一来, 如果  $P_2$  的这个输入比特是 0, 他将完成协议, 不会发现任何异常; 如果  $P_2$  的这个输入比特是 1, 他就没

<sup>2</sup>这里做了简化, 实际上应该有  $s/2$  对密钥。

有得到对应的密钥，无法计算混淆电路，只能中止协议。 $P_1$  通过观察  $P_2$  是否中止协议从而知道  $P_2$  的输入比特。

为了抵御选择性中止攻击，我们需要让  $P_1$  在执行 OT 协议的同时对 OT 协议的输入进行承诺（这种 OT 称为 committing OT<sup>[51]</sup>），然后在进行切分选择检查时将打开的电路对应的承诺一起打开。具体而言， $P_1$  和  $P_2$  首先对  $P_2$  的每个输入比特执行 committing OT 协议。然后， $P_1$  发送  $s$  个混淆电路给  $P_2$ ， $P_2$  随机选择  $s/2$  个电路让  $P_1$  打开。 $P_1$  同时打开这些电路在 committing OT 协议中的两个输入（利用 committing OT 的绑定性）， $P_2$  检查这些输入都是正确的。这样的方法可以确保  $P_1$  在大多数 OT 协议中使用了正确的输入。

$P_1$  的输出真实性 (output authenticity)。上述方法可以确保  $P_2$  进行电路计算后大多数电路的输出（以压倒性的概率）是正确的。在混淆电路协议中， $P_2$  在最后一步将协议输出发送给  $P_1$ ，在半诚实敌手模型下，这没有任何问题，但是，在恶意敌手模型下，恶意的  $P_2$  可以发送任意的错误输出给  $P_1$ 。

需要注意的是，在恶意敌手模型下进行两方安全计算时，我们无法实现公平性 (fairness)<sup>[42]</sup>。也就是说，协议无法使参与双方同时获得输出，恶意的参与方可以在获得输出后停止执行协议，使诚实参与方不获得输出。以混淆电路协议为例，我们无法阻止  $P_2$  在获得输出后停止执行协议。但是，我们能够实现的是：如果  $P_2$  向  $P_1$  发送协议输出， $P_1$  可以检查输出的真实性 (authenticity)。

这里采用的技术是消息认证码 (message authentication codes, MACs)，并且，如果我们能安全地计算任意单一输出功能  $f(x, y)$  使得只有  $P_2$  得到输出，那么我们可以基于其构造一个实现功能  $f = (f_1, f_2)$  的协议，其中  $P_1$  得到  $f_1(x, y)$ ， $P_2$  得到  $f_2(x, y)$ 。

协议构造如下。令  $f = (f_1, f_2)$  是一个功能。我们需要构造一个安全的协议使  $P_1$  得到  $f_1(x, y)$ ， $P_2$  得到  $f_2(x, y)$ ；我们有一个能够安全计算任何功能的协议，但是只有  $P_2$  能获得输出。令  $\mathbb{F}$  是包含  $f_1$  所有取值  $\{f_1(x, y)\}_{x, y \in \{0,1\}^n}$  的域，令  $p, a, b \leftarrow_{\$} \mathbb{F}$  是在  $\mathbb{F}$  上均匀随机选取的值。定义单一输出功能  $g$  如下： $g((p, a, b, x), y) = (\alpha, \beta, f_2(x, y))$ ，其中  $\alpha = p + f_1(x, y)$ ， $\beta = a \cdot \alpha + b$ ，运算符号  $\langle +, \cdot \rangle$  由域  $\mathbb{F}$  所定义。这里的  $\alpha$  是以  $p$  为密钥使用“一次一密”加密的密文， $\beta$  是  $\alpha$  的消息认证码。双方使用只有  $P_2$  能获得输出的协议计算功能  $g$ ，然后  $P_2$  将  $(\alpha, \beta)$  发给  $P_1$ 。 $P_1$  检查  $\beta \stackrel{?}{=} a \cdot \alpha + b$ ，如果是，输出  $\alpha - p$ ，否则输出  $\perp$ 。

显然，“一次一密”的完美安全性保证了  $P_2$  不知道  $f_1(x, y)$ ，而  $P_2$  成功伪造消息认证码  $\beta = a \cdot \alpha + b$  的概率为  $1/|\mathbb{F}|$ 。回顾一下，在第8.1.4节中，（独立模型下）对于恶意敌手的安全性定义考虑的是两个参与方都获得输出的情况。根据上述构造，我们可以使用一种更简单的等价定义，这种定义下只有一个参与方获得输出。我们在之后的协议构造和证明中，也会采用这种更简单的定义。

**协议概览。**综合以上所有构造，基于切分选择技术的混淆电路协议整体流程如下。

1.  $P_1$  生成  $s$  个混淆电路。
2. 对于  $P_2$  的每个输入比特， $P_1$  和  $P_2$  执行 committing OT 协议使  $P_2$  获得对应的密钥。
3.  $P_1$  向  $P_2$  发送  $s$  个混淆电路，以及  $P_1$  的输入导线密钥的承诺值。
4.  $P_2$  随机选择  $s/2$  个电路让  $P_1$  打开。
5.  $P_1$  打开这些电路，同时打开这些电路在 committing OT 协议中的输入，打开这些电路中  $P_1$  的输入导线密钥的承诺。 $P_2$  检查正确性。
6.  $P_1$  向  $P_2$  发送未打开的电路中  $P_1$  的输入导线密钥，并证明输入一致性。

7.  $P_2$  对未打开的电路进行计算，取大多数电路的计算结果为输出。
8. (如果  $P_1$  也需要获得输出)  $P_2$  向  $P_1$  发送  $P_1$  的输出和消息认证码， $P_1$  检查消息认证码。

上述流程是基于切分选择的协议的概览。实际上，研究人员对切分选择协议做了大量的优化<sup>[52-61]</sup>。本书将介绍 Lindell 和 Pinkas 在 2011 年提出的一个比较有代表性的切分选择协议<sup>[56]</sup>，并给出安全性证明。

### 9.2.2 LP11 协议

LP11 协议<sup>[56]</sup>也遵循上述基本流程，但有以下两点不同：

1. 在第 2 步中，LP11 协议没有执行 committing OT，而是引入了一个新的原语——切分选择 OT(Cut-and-Choose OT, CCOT)。在切分选择 OT 中，发送方的输入是  $(x_1^0, x_1^1), \dots, (x_s^0, x_s^1)$ ，接收方的输入是选择比特  $\sigma_1, \dots, \sigma_s$  以及一个大小为  $s/2$  的集合  $\mathcal{J} \subset [s]$ 。对于  $i \in [s]$ ，接收方得到  $x_i^{\sigma_i}$  (就像普通的 OT 那样)；对于  $j \in \mathcal{J}$ ，接收方得到发送方的两个值  $(x_j^0, x_j^1)$ 。发送方不能得到  $\sigma_1, \dots, \sigma_s$  和  $\mathcal{J}$  的任何信息。这里的集合  $\mathcal{J}$  实际上就是要打开的电路的索引。切分选择 OT 将 OT 协议与切分选择检查结合在一起，有效地抵御了选择性中止攻击（实际上，切分选择 OT 和 committing OT 具有类似的作用）。
2. LP11 协议设计了更高效的零知识证明以证明  $P_1$  的输入一致性。假设  $P_1$  的输入长度为  $\ell$  比特， $P_1$  选择值  $g^{a_1^0}, g^{a_1^1}, \dots, g^{a_\ell^0}, g^{a_\ell^1}$  以及  $g^{r_1}, \dots, g^{r_s}$ ，然后将第  $i$  个输入比特在第  $j$  个电路中的密钥设置为  $g^{a_i^0 \cdot r_j}, g^{a_i^1 \cdot r_j}$ ， $i \in [\ell], j \in [s]$ 。基于 DDH 假设，在已知所有的  $\{g^{a_i^0}, g^{a_i^1}, g^{r_j}\}$  和  $P_1$  的输入对应于密钥的情况下，其余密钥仍是伪随机的。这样的构造使得  $P_1$  的输入一致性零知识证明非常高效。

下面，我们详细介绍这两点。

#### 9.2.2.1 切分选择 OT

**切分选择 OT 理想功能。**直观地说，切分选择 OT 协议是一个批处理的 OT 协议（它能够对多对输入同时执行 OT 协议），但有一个额外属性：接收方可以选择一个子集，对于这个子集中的索引，接收方获得两个值。在基于切分选择的协议中，这样的构造非常自然。接收方选择的子集就是让  $P_1$  打开的电路的索引，接收方获得这些电路在 OT 协议中的两个输入，防止  $P_1$  进行选择性中止攻击。

图 9.2 展示了切分选择 OT 的理想功能  $\mathcal{F}_{CCOT}$  以及单一选择切分选择 OT(single-choice cut-and-choose OT)，记为  $\mathcal{F}_{CCOT}^S$ 。在单一选择切分选择 OT 中，接收方必须在每一个 OT 中使用相同的选择比特  $\sigma$ （这是为了保证切分选择协议中  $P_2$  的输入一致性）。

**切分选择 OT 构造思路。**切分选择 OT 基于 Peikert, Vaikuntanathan, Waters (PVW) 的 OT 协议<sup>[21]</sup>构造。PVW 协议是在公共参考字符串 (Common Reference String, CRS)<sup>3</sup>模型下构造的，它的 CRS 是元组  $(g_0, g_1, h_0, h_1)$ ，其中  $g_0$  是一个阶为  $q$  的群的生成元 (DDH 问题在该群中是困难的)， $g_1 = (g_0)^y$ ， $y$  是一个随机数， $h_0 = (g_0)^a$ ， $h_1 = (g_1)^b$  且  $a \neq b$ 。而切分选择 OT 协议可以在朴素模型 (plain model)

<sup>3</sup> 公共参考字符串是从某个分布中预先随机选择的字符串，它对协议的所有参与方公开；朴素模型 (plain model) 没有任何初始设置。相比于朴素模型，在公共参考字符串模型下可以构造出实现更强理想功能的协议。

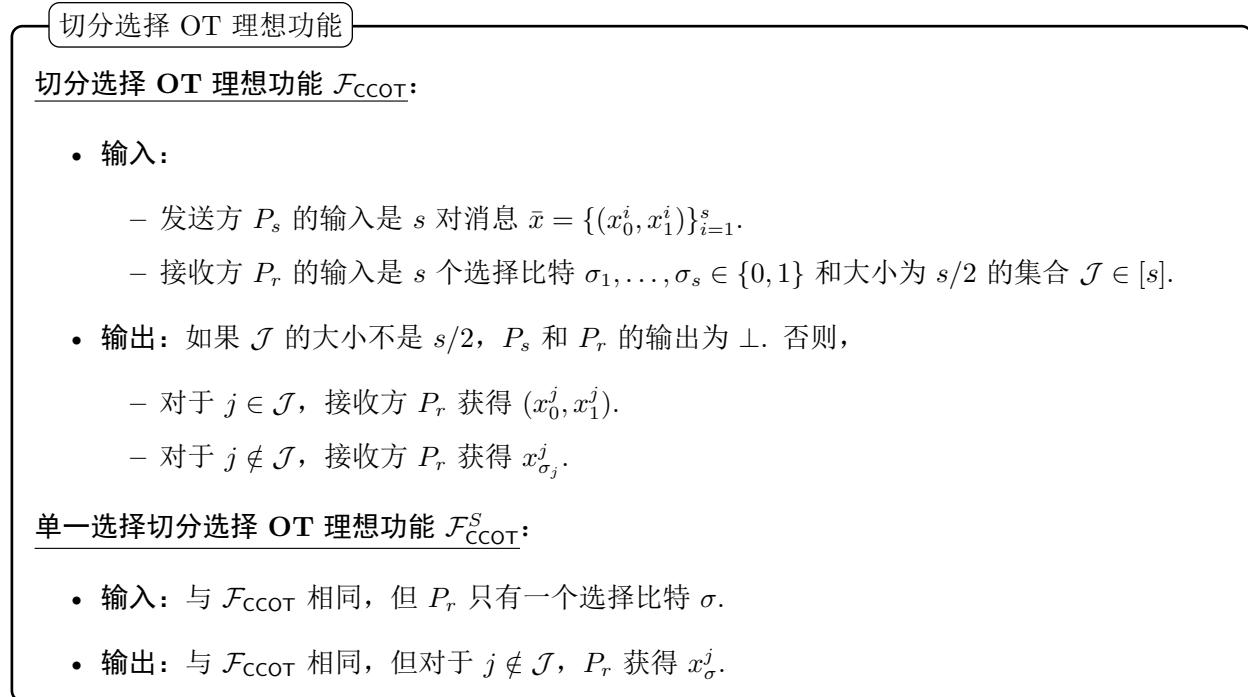


图 9.2: 切分选择 OT 理想功能

下构造, 这是因为协议让接收方选择元组  $(g_0, g_1, h_0, h_1)$ , 其中  $h_0 = (g_0)^a$ ,  $h_1 = (g_1)^b$ , 然后证明它满足  $a \neq b$  (也就是证明它不是 Diffie-Hellman 元组); 具体而言, 协议设置  $b = a + 1$ , 然后用 Sigma 协议证明  $(g_0, g_1, h_0, \frac{h_1}{g_1})$  是 Diffie-Hellman 元组, 使得该证明非常高效。

此外, 当  $(g_0, g_1, h_0, h_1)$  是 Diffie-Hellman 元组 (即  $a = b$ ) 时, PVW 协议的接收方可以获得发送方的两个输入。这恰恰符合切分选择 OT 的需要! 我们让接收方选取  $s/2$  对满足  $a \neq b$  的  $(h_0, h_1)$  (他只能获得一个输入), 选取  $s/2$  对满足  $a = b$  的  $(h_0, h_1)$  (他可以获得两个输入), 这就构成了切分选择 OT 的雏形。当然, 接收方需要证明有  $s/2$  对  $(h_0, h_1)$  满足  $a \neq b$ , 也就是需要证明  $s$  对  $(h_0, h_1)$  中有一部分满足某个陈述, 这是一个部分知识的证明 (proof of partial knowledge), 可以通过 CDS 组合<sup>[62]</sup>高效地实现。下面, 我们对上述思路的每一部分详细阐释。

**PVW 协议 (的变体)。** 图 9.3展示了 PVW 协议在朴素模型下的变体。

我们先分析 DH 元组的零知识证明协议 (图 9.4)。它实质上是在 Sigma 协议的基础上让验证者对挑战值进行承诺, 以实现 (面对任意验证者的) 零知识性。DH 元组 Sigma 协议的初始消息是  $A = (g_0)^r$ ,  $B = (g_1)^r$ , 挑战值是  $s$ , 回答消息是  $z = s \cdot w + r$ , 验证者最后验证  $(g_0)^z \stackrel{?}{=} A \cdot u^s$ ,  $(g_1)^z \stackrel{?}{=} B \cdot v^s$ 。在此基础上, 协议的第 1、2 步让验证者对挑战值  $s$  使用 Pedersen 承诺方案 (参见第2.3.6节) 进行承诺, Pedersen 承诺方案的承诺密钥为  $\alpha = (g_0)^a$  由证明者选取; 在协议的第 4、5 步, 证明者需要额外打开 Pedersen 承诺 (公布  $s, t$ ); 在协议的第 6 步, 证明者需要额外公布承诺密钥  $\alpha$  的离散对数  $a$ 。这样做的目的是: Sigma 协议仅满足诚实验证者零知识性 (special honest verifier zero-knowledge), 但对于任意的验证者, 模拟器无法预知挑战值; 图 9.4的协议让验证者首先对挑战值进行承诺, 使协议获得了 (面对任意验证者的) 零知识性<sup>4</sup>。

<sup>4</sup>对挑战值进行承诺, 使得模拟器将验证者倒带后, 得到的仍是同样的挑战值, 感兴趣的读者可以参阅书<sup>[63]</sup>, Chapter 6 中完整的证明。

### PVW 协议（朴素模型变体）

**输入：**发送方  $P_s$  的输入是  $(x_0, x_1)$ , 接收方  $P_r$  的输入是一个比特  $\sigma$ .

**辅助输入：**安全参数  $1^n$  和  $(\mathbb{G}, q, g_0)$ , 其中  $\mathbb{G}$  是一个阶为  $q$ , 以  $g_0$  为生成元的群,  $q$  的长度为  $n$ .  
**协议：**

1. 接收方  $P_r$  随机选择  $y, \alpha_0 \leftarrow_{\$} \mathbb{Z}_q$ , 设置  $\alpha_1 = \alpha_0 + 1$ , 然后计算  $g_1 = (g_0)^y, h_0 = (g_0)^{\alpha_0}, h_1 = (g_1)^{\alpha_1}$ .  $P_r$  发送  $(g_1, h_0, h_1)$  给  $P_s$ .
2.  $P_r$  使用零知识证明协议证明  $(g_0, g_1, h_0, h_1)$  不是 DH 元组 ( $P_r$  实际上证明  $(g_0, g_1, h_0, \frac{h_1}{g_1})$  是 DH 元组, 见图 9.4)。
3.  $P_r$  选择随机值  $r$  并计算  $g = (g_\sigma)^r, h = (h_\sigma)^r$ .  $P_r$  发送  $(g, h)$  给  $P_s$ .
4.  $P_s$  执行如下操作:
  - 定义函数  $\text{RAND}(w, x, y, z) = (u, v)$ , 其中  $u = (w)^s \cdot (y)^t, v = (x)^s \cdot (z)^t, s, t \leftarrow_{\$} \mathbb{Z}_q$  是均匀随机数。
  - $P_s$  计算  $(u_0, v_0) = \text{RAND}(g_0, g, h_0, h), (u_1, v_1) = \text{RAND}(g_1, g, h_1, h)$ .
  - $P_s$  将  $(u_0, w_0)$  和  $(u_1, w_1)$  发给  $P_r$ , 其中  $w_0 = v_0 \cdot x_0, w_1 = v_1 \cdot x_1$ .
5.  $P_r$  输出  $x_\sigma = w_\sigma / (u_\sigma)^r$ .

图 9.3: PVW 协议（朴素模型变体）

### DH 元组的零知识证明协议

**陈述：**  $(\mathbb{G}, g_0, g_1, u, v)$ , 其中  $\mathbb{G}$  是一个阶为  $q$ , 以  $g_0$  为生成元的群,  $g_0, g_1, u, v \in \mathbb{G}$ .

**见证：**  $w$  s.t.  $u = (g_0)^w \wedge v = (g_1)^w$ .

**协议：**

1. 证明者  $P$  随机选择  $a \leftarrow_{\$} \mathbb{Z}_q$ , 计算  $\alpha = (g_0)^a$  并将  $\alpha$  发送给验证者  $V$ .
2.  $V$  随机选择  $s, t \leftarrow_{\$} \mathbb{Z}_q$ , 计算  $c = (g_0)^s \cdot \alpha^t$  并将  $c$  发送给  $P$ .
3.  $P$  随机选择  $r \leftarrow_{\$} \mathbb{Z}_q$ , 计算  $A = (g_0)^r, B = (g_1)^r$  并将  $(A, B)$  发送给  $V$ .
4.  $V$  发送  $s, t$  给  $P$ .
5.  $P$  检查  $c \stackrel{?}{=} (g_0)^s \cdot \alpha^t$ . 如果不是, 那么终止协议。如果是,  $P$  计算  $z = s \cdot w + r$  并发送  $z$  和  $a$  给  $V$ .
6. 当且仅当  $\alpha = (g_0)^a, (g_0)^z = A \cdot u^s, (g_1)^z = B \cdot v^s$  时,  $V$  接受证明。

图 9.4: DH 元组的零知识证明协议

接着，我们分析一下 PVW 协议。首先，接收方在最后一步能接收到正确的消息，因为

$$\frac{w_\sigma}{(u_\sigma)^r} = \frac{v_\sigma \cdot x_\sigma}{(u_\sigma)^r} = \frac{g^s \cdot h^t \cdot x_\sigma}{((g_\sigma)^s \cdot (h_\sigma)^t)^r} = \frac{g^s \cdot h^t \cdot x_\sigma}{((g_\sigma)^r)^s \cdot ((h_\sigma)^r)^t} = \frac{g^s \cdot h^t \cdot x_\sigma}{g^s \cdot h^t} = x_\sigma$$

关于安全性，这基于 RAND 函数的性质：如果 RAND 的输入不是 DH 元组，那么它的输出是两个独立的均匀随机分布的群元素。因此， $(u_0, w_0)$  和  $(u_1, w_1)$  中有一个是均匀随机分布的，所以接收方只能获得发送方的一个输入。相反，如果  $(g_0, g_1, h_0, h_1)$  是 DH 元组且接收方知道  $y = \log_{g_0} g_1$ ，那么接收方可以获得发送方的两个输入。例如，对于  $\sigma = 0$  的情况，接收方可以首先执行协议步骤计算  $x_0$ ；并且，接收方还可以计算  $x_1 = w_1 / (u_1)^{ry^{-1}}$ ，这是因为

$$\begin{aligned} \frac{w_1}{(u_1)^{ry^{-1}}} &= \frac{g^s \cdot h^t \cdot x_1}{((g_1)^s \cdot (h_1)^t)^{ry^{-1}}} = \frac{g^s \cdot h^t \cdot x_1}{((g_1)^{y^{-1}})^{s \cdot r} \cdot ((h_1)^{y^{-1}})^{t \cdot r}} \\ &= \frac{g^s \cdot h^t \cdot x_1}{((g_0)^s \cdot (h_0)^t)^r} = \frac{g^s \cdot h^t \cdot x_1}{g^s \cdot h^t} = x_1 \end{aligned}$$

同理，对于  $\sigma = 1$  的情况，接收方可以执行协议步骤计算  $x_1$ ，并通过计算  $w_0 / (u_0)^{ry}$  得到  $x_0$ 。协议中，接收方需要通过零知识证明协议证明  $(g_0, g_1, h_0, \frac{h_1}{g_1})$  是 DH 元组，从而确保接收方只能获得发送方的一个输入。

安全性证明需要分别考虑发送方  $P_s$  和接收方  $P_r$  被攻陷的情况。当发送方  $P_s$  被攻陷时，模拟器  $\mathcal{S}$  需要提取  $P_s$  的两个输入。 $\mathcal{S}$  设置  $\alpha_1 = \alpha_0$ （即  $(g_0, g_1, h_0, h_1)$  是 DH 元组）并且“伪造”它的零知识证明，从而提取  $P_s$  的两个输入。当接收方  $P_r$  被攻陷时，模拟器  $\mathcal{S}$  可以提取零知识证明中的见证  $\alpha_0$  并计算  $\alpha_1 = \alpha_0 + 1$ 。通过检查  $h = g^{\alpha_0}$  还是  $h = g^{\alpha_1}$ ，模拟器  $\mathcal{S}$  可以提取出接收方的选择比特  $\sigma$ .<sup>5</sup>

**证明某个子集是 DH 元组 (CDS 组合)。**之前讲到，在切分选择 OT 中，接收方需要构造  $s$  对  $(h_0, h_1)$ ，并证明有  $s/2$  对不是 DH 元组。证明“不是 DH 元组”较为困难，接收方实际上证明  $(g_0, g_1, h_0, \frac{h_1}{g_1})$  是 DH 元组。也就是说，接收方首先构造  $\{(h_0^j, h_1^j)\}_{j \in [s]}$ ，然后证明有  $s/2$  个元组  $(g_0, g_1, h_0^j, \frac{h_1^j}{g_1})$  是 DH 元组。为了符号的简洁，我们可以将接收方构造的元组记为  $\{(h_0^j, \tilde{h}_1^j)\}_{j \in [s]}$ ，令  $\tilde{h}_1^j = \frac{h_1^j}{g_1}$ 。于是，我们需要构造关系  $\mathcal{R}$  的零知识证明来证明某个子集是 DH 元组，其中

$$\mathcal{R} = \left\{ ((\mathbb{G}, g_0, g_1, (h_0^1, h_1^1), \dots, (h_0^s, h_1^s)), \{(i_j, w_{i_j})\}) \mid \begin{array}{l} i_j \text{ 互不相同，将 } i_j \text{ 的集合记为 } I, |I| = s/2 \\ \text{且对每个 } i_j \in I \text{ 有 } h_0^{i_j} = (g_0)^{w_{i_j}}, h_1^{i_j} = (g_1)^{w_{i_j}} \end{array} \right\}$$

我们刚才已经构造了 DH 元组的零知识证明（图 9.4）。现在，我们需要证明某个大小为  $s/2$  的子集是 DH 元组，并且不泄漏这个子集。这里采用的方法是 Cramer, Damgård, Schoenmakers 提出的 CDS 组合<sup>[62]</sup>。CDS 组合将 Sigma 协议与秘密分享相结合，构成了部分知识的证明 (proof of partial knowledge)。协议如图 9.5 所示。

协议中执行了  $s$  个 DH 元组的零知识证明的实例，但实际上只有  $s/2$  个元组是 DH 元组。对于这  $s/2$  个元组，证明者正常地执行 DH 元组的零知识证明协议；对于另外  $s/2$  个元组，证明者调用模拟器的算法模拟协议的会话，但这需要证明者先知道挑战值。为此，CDS 组合巧妙地与秘密分享相结合：

<sup>5</sup> 我们对这里的“伪造证明”和“提取见证”作一点补充说明：考虑图 9.4 的协议。需要“伪造证明”时， $\mathcal{S}$  先与验证者交互得到挑战值  $s$ ，然后将验证者倒带，此时  $\mathcal{S}$  预先知道了  $s$ ，就可以模拟出会话  $(A, B, s, z)$ ，它与真实会话不可区分（参考定理 9.1 的证明）。需要“提取见证”时， $\mathcal{S}$  先与证明者交互得到会话  $(A, B, s, z)$ ，然后将证明者倒带，将挑战值换成  $s'$ ，得到会话  $(A, B, s', z')$ ，基于 2-特殊可靠性，可以提取见证  $w$ （参考定理 9.1 的证明），同时，因为在第一次与证明者交互时获得了  $\alpha$  的离散对数  $a$ ，所以  $\mathcal{S}$  在第二次交互时可以将 Pedersen 承诺  $c$  打开为  $s'$ 。

### 子集是 DH 元组的零知识证明协议

**陈述:**  $(\mathbb{G}, g_0, g_1, (h_0^1, h_1^1), \dots, (h_0^s, h_1^s))$ , 其中  $\mathbb{G}$  是一个阶为  $q$ , 以  $g_0$  为生成元的群。

**见证:**  $W = \{(i_j, w_{i_j})\}$  满足  $i_j$  互不相同, 将  $i_j$  的集合记为  $I$ ,  $|I| = s/2$  且对每个  $i_j \in I$  有  $h_0^{i_j} = (g_0)^{w_{i_j}}, h_1^{i_j} = (g_1)^{w_{i_j}}$ .

**协议:**

1. 验证者对挑战值承诺:

- (a) 证明者  $P$  随机选择  $a \leftarrow_{\$} \mathbb{Z}_q$ , 计算  $\alpha = (g_0)^a$  并将  $\alpha$  发送给验证者  $V$ .
- (b)  $V$  随机选择  $t, c \leftarrow_{\$} \mathbb{Z}_q$ , 计算  $C = (g_0)^c \cdot \alpha^t$  并将  $C$  发送给  $P$ .

2. 证明者发送第 1 条消息:

- (a) 对于  $i \notin I$ ,  $P$  随机选择  $c_i, z_i \leftarrow_{\$} \mathbb{Z}_q$  并计算  $A_i = \frac{(g_0)^{z_i}}{(h_0^i)^{c_i}}, B_i = \frac{(g_1)^{z_i}}{(h_1^i)^{c_i}}$ .
- (b) 对于  $i \in I$ ,  $P$  随机选择  $\rho_i \leftarrow_{\$} \mathbb{Z}_q$  并计算  $A_i = (g_0)^{\rho_i}, B_i = (g_1)^{\rho_i}$ .
- (c)  $P$  发送  $(A_1, B_1), \dots, (A_s, B_s)$  给  $V$ .

3. 验证者打开挑战值:

- (a)  $V$  发送  $t, c$  给  $P$ .

4. 证明者发送第 2 条消息:

- (a)  $P$  检查  $C \stackrel{?}{=} (g_0)^c \cdot \alpha^t$ . 如果不是, 那么终止协议。如果是,  $P$  将  $c$  作为挑战值。
- (b) 将  $\{c_i\}_{i \notin I}$  和  $c$  视为  $(s/2, s)$ -门限秘密分享的秘密份额和秘密值。它们唯一定义了其他的秘密份额。证明者计算出所有份额  $c_1, \dots, c_s$ .
- (c) 对于  $i \notin I$ ,  $z_i$  为步骤 2(a) 中选择的  $z_i$ .
- (d) 对于  $i \in I$ , 计算  $z_i = c_i \cdot w_i + \rho_i$ .
- (e)  $P$  发送  $(c_1, z_1), \dots, (c_s, z_s)$  和步骤 1(a) 中选择的  $a$  给  $V$ .

5. 验证者进行验证: 当且仅当以下等式成立时,  $V$  接受证明:

- (a)  $\alpha = g^a$
- (b)  $c_1, \dots, c_s$  是  $c$  合法的秘密份额
- (c) 对于  $i \in [s]$ ,  $A_i = \frac{(g_0)^{z_i}}{(h_0^i)^{c_i}}, B_i = \frac{(g_1)^{z_i}}{(h_1^i)^{c_i}}$

图 9.5: 子集是 DH 元组的零知识证明协议

将验证者的总挑战值  $c$  看作秘密，将子挑战值  $c_1, \dots, c_s$  看作秘密份额。如果采用的是  $(s/2, s)$ -门限秘密分享方案，那么证明者可以自由选择  $s/2$  个秘密份额，从而模拟这些实例的会话；如果证明者自由选择了超过  $s/2$  个秘密份额，那么这些秘密份额就已经决定了秘密值  $c$ ，它恰好与验证者随机选择的挑战值相等的概率是可忽略的。因此，上述协议具有可靠性。同时，模拟的协议会话和真实的协议会话不可区分，因此验证者无法知道哪些实例是 DH 元组，哪些实例是模拟的，确保了协议的零知识性。

**切分选择 OT 的构造。**至此，我们介绍了切分选择 OT 的所有子部件。我们给出切分选择 OT 协议的描述，如图 9.6 所示。

**定理 9.2.** 假设 DDH 问题在群  $\mathbb{G}$  中是困难的。图 9.6 中的协议对于静态恶意敌手安全实现了理想功能  $\mathcal{F}_{CCOT}$ （图 9.2）。

注意，这里的“安全实现”考虑的是独立模型。根据前文所述的思路，模拟器  $\mathcal{S}$  的构造非常直接。当发送方  $P_s$  被攻陷时， $\mathcal{S}$  模拟接收方构造  $s$  对  $\{(h_0^j, h_1^j)\}_{j \in [s]}$ ，它们全都是 DH 元组，然后“伪造”零知识证明，从而提取  $P_s$  的两个输入。当接收方  $P_r$  被攻陷时， $\mathcal{S}$  从  $P_r$  的零知识证明中提取出见证，从而提取出集合  $\mathcal{J}$  和  $\{\sigma_j\}_{j \notin \mathcal{J}}$ 。这里不再给出完整的证明，留作读者练习<sup>6</sup>。

**单一选择切分选择 OT.**如果我们希望接收方在所有 OT 实例中使用同样的选择比特  $\sigma$ ，很自然地，图 9.6 中的协议中传输阶段的第 1 步应该修改为：

$P_r$  随机选择  $r \leftarrow_{\$} \mathbb{Z}_q$  并计算  $g' = (g_\sigma)^r$ . 对于  $j \in [s]$ , 计算  $h_j = (h_\sigma^j)^r$ .  $P_r$  发送  $(g', h_1, \dots, h_s)$  给  $P_s$  并零知识地证明其正确性。

这里的零知识证明协议证明的是：存在  $r \in \mathbb{Z}_q$  满足 “ $g' = (g_0)^r$  且对于  $j \in [s]$  有  $h_j = (h_0^j)^r$ ” 或 “ $g' = (g_1)^r$  且对于  $j \in [s]$  有  $h_j = (h_1^j)^r$ ”。换句话说，需要证明  $\{(g_0, g', h_0^j, h_j)\}_{j \in [s]}$  都是 DH 元组或者  $\{(g_1, g', h_1^j, h_j)\}_{j \in [s]}$  都是 DH 元组。该协议如图 9.7 所示。

**定理 9.3.** 假设 DDH 问题在群  $\mathbb{G}$  中是困难的。图 9.6 中的协议经上述修改后对于静态恶意敌手安全实现了理想功能  $\mathcal{F}_{CCOT}^S$ （图 9.2）。

该定理的证明同样非常直接，留作读者练习。

**批处理单一选择切分选择 OT.**最后，我们的协议需要让计算方对每一条导线执行单一选择切分选择 OT，并且在每一个实例中使用同样的集合  $\mathcal{J}$ . 我们将这个理想功能记为  $\mathcal{F}_{CCOT}^{S,B}$ ，如图 9.8 所示。

要实现理想功能  $\mathcal{F}_{CCOT}^{S,B}$ ，我们只运行一次切分选择 OT 协议（图 9.6）的准备阶段，这样就确保了在所有实例中都使用了相同的集合  $\mathcal{J}$ . 然后，对于每个  $i \in [\ell]$ ，运行单一选择切分选择 OT 协议的传输阶段（可以并行地运行）。如此，我们就完成了批处理单一选择切分选择 OT 协议的构造。

**定理 9.4.** 假设 DDH 问题在群  $\mathbb{G}$  中是困难的。图 9.6 中的协议经上述修改后对于静态恶意敌手安全实现了理想功能  $\mathcal{F}_{CCOT}^{S,B}$ （图 9.8）。

**总结。**本节，我们从 PVW 协议开始，利用其特殊的性质（当公共参考字符串是 DH 元组时可以获得两个输入），构造了切分选择 OT. 然后，我们又在此基础上构造了单一选择切分选择 OT 和批处理单一选择切分选择 OT. 在协议中，我们多次使用了基于 Sigma 协议的零知识证明。至此，我们完成了协议中最重要的原语的构造。

<sup>6</sup>如果读者在证明时遇到问题，本书的前言中有我的邮箱，欢迎与我交流。

## 切分选择 OT 协议

**公共输入:**  $(\mathbb{G}, q, g_0)$ , 其中  $\mathbb{G}$  是一个阶为  $q$ , 以  $g_0$  为生成元的群。 $q$  的长度为  $n$ .

**输入:** 发送方  $P_s$  的输入是  $\{x_0^j, x_1^j\}_{j \in [s]}$ ; 接收方  $P_r$  的输入是  $s$  个比特  $\sigma_1, \dots, \sigma_s$  和一个大小为  $s/2$  的集合  $\mathcal{J}$ ,  $\mathcal{J} \subset [s]$ .

**协议:**

- 准备阶段:

1. 接收方  $P_r$  随机选择  $y \leftarrow_{\$} \mathbb{Z}_q$  并计算  $g_1 = (g_0)^y$ .
2. 对于  $j \in \mathcal{J}$ ,  $P_r$  随机选择  $\alpha_j \leftarrow_{\$} \mathbb{Z}_q$  并计算  $h_0^j = (g_0)^{\alpha_j}$ ,  $h_1^j = (g_1)^{\alpha_j}$ .
3. 对于  $j \notin \mathcal{J}$ ,  $P_r$  随机选择  $\alpha_j \leftarrow_{\$} \mathbb{Z}_q$  并计算  $h_0^j = (g_0)^{\alpha_j}$ ,  $h_1^j = (g_1)^{\alpha_j+1}$ .
4.  $P_r$  发送  $(g_1, h_0^1, h_1^1, \dots, h_0^s, h_1^s)$  给  $P_s$ .
5.  $P_r$  通过零知识证明向  $P_s$  证明  $s/2$  个  $(g_0, g_1, h_0^j, h_1^j)$  不是 DH 元组 ( $P_r$  实际上证明  $s/2$  个  $(g_0, g_1, h_0^j, \frac{h_1^j}{g_1})$  是 DH 元组, 见图 9.5)。

- 传输阶段: 对于  $j \in [s]$ , 执行以下操作 (可以并行执行):

1.  $P_r$  随机选择  $r_j \leftarrow_{\$} \mathbb{Z}_q$  并计算  $\tilde{g}_j = (g_{\sigma_j})^{r_j}$ ,  $\tilde{h}_j = (h_{\sigma_j}^j)^{r_j}$ .  $P_r$  发送  $(\tilde{g}_j, \tilde{h}_j)$  给  $P_s$ .

2.  $P_s$  执行以下操作:

- 定义函数  $\text{RAND}(w, x, y, z) = (u, v)$ , 其中  $u = (w)^s \cdot (y)^t$ ,  $v = (x)^s \cdot (z)^t$ ,  $s, t \leftarrow_{\$} \mathbb{Z}_q$  是随机数。
- $P_s$  计算  $(u_0^j, v_0^j) = \text{RAND}(g_0, \tilde{g}_j, h_0^j, \tilde{h}_j)$ ,  $(u_1^j, v_1^j) = \text{RAND}(g_1, \tilde{g}_j, h_1^j, \tilde{h}_j)$ .
- $P_s$  将  $(u_0^j, w_0^j)$  和  $(u_1^j, w_1^j)$  发给  $P_r$ , 其中  $w_0^j = v_0^j \cdot x_0^j$ ,  $w_1^j = v_1^j \cdot x_1^j$ .

- 输出阶段:

1. 对于  $j \in [s]$ ,  $P_r$  计算  $x_{\sigma_j}^j = \frac{w_{\sigma_j}^j}{(u_{\sigma_j}^j)^{r_j}}$ .
2. 对于  $j \in \mathcal{J}$ ,  $P_r$  计算  $x_{1-\sigma_j}^j = \frac{w_{1-\sigma_j}^j}{(u_{1-\sigma_j}^j)^{r_j \cdot z_j}}$ , 其中, 当  $\sigma_j = 0$  时  $z_j = y^{-1} \bmod q$ , 当  $\sigma_j = 1$  时  $z_j = y$ .

图 9.6: 切分选择 OT 协议

### 拓展的 DH 元组的零知识证明协议

**陈述:**  $(\mathbb{G}, g_0, g_1, g', u_1, v_1, \dots, u_s, v_s, h_1, \dots, h_s)$ , 其中  $\mathbb{G}$  是一个阶为  $q$  的群,  $g_0, g_1$  是其生成元。

**见证:**  $r$  s.t. “ $g' = (g_0)^r$  且对于  $j \in [s]$  有  $h_j = (u_j)^r$ ” 或 “ $g' = (g_1)^r$  且对于  $j \in [s]$  有  $h_j = (v_j)^r$ ”

**协议:**

1. 验证者  $V$  随机选择  $\gamma_1, \dots, \gamma_s \leftarrow_{\$} \mathbb{Z}_q$  并发送给证明者  $P$ .

2.  $P$  和  $V$  本地计算

$$\tilde{u} = \prod_{i=1}^s (u_i)^{\gamma_i}, \quad \tilde{v} = \prod_{i=1}^s (v_i)^{\gamma_i}, \quad \tilde{h} = \prod_{i=1}^s (h_i)^{\gamma_i}$$

3.  $P$  向  $V$  零知识地证明  $(g_0, g', \tilde{u}, \tilde{h})$  或  $(g_1, g', \tilde{v}, \tilde{h})$  是 DH 元组 (只需在图 9.5 的协议中取  $s = 2$ )。

图 9.7: 拓展的 DH 元组的零知识证明协议

批处理单一选择切分选择 OT 理想功能  $\mathcal{F}_{\text{CCOT}}^{S,B}$

- **输入:**

- 发送方  $P_s$  的输入是  $\ell$  个长度为  $s$  的向量  $\vec{x}_i, i \in [\ell]$  (每个向量由  $s$  对消息构成)。

- 接收方  $P_r$  的输入是  $\ell$  个选择比特  $\sigma_1, \dots, \sigma_\ell \in \{0, 1\}$  和大小为  $s/2$  的集合  $\mathcal{J} \in [s]$ .

- **输出:** 如果  $\mathcal{J}$  的大小不是  $s/2$ ,  $P_s$  和  $P_r$  的输出为  $\perp$ . 否则,

- 对于  $i \in [\ell]$  和  $j \in \mathcal{J}$ , 接收方  $P_r$  获得  $\vec{x}_i$  的第  $j$  对消息。

- 对于  $i \in [\ell]$  和  $j \notin \mathcal{J}$ , 接收方  $P_r$  获得  $\vec{x}_i$  的第  $j$  对消息的第  $\sigma_i$  个值 ( $\sigma_i \in \{0, 1\}$ , 这里的第  $\sigma_i$  个值从 0 开始计数)。

图 9.8: 批处理单一选择切分选择 OT 理想功能  $\mathcal{F}_{\text{CCOT}}^{S,B}$

### 9.2.2.2 输入一致性零知识证明

接下来，我们来构造另一个原语——输入一致性零知识证明。我们前面讲过，LP11 协议中， $P_1$  首先选择值  $g^{a_1^0}, g^{a_1^1}, \dots, g^{a_\ell^0}, g^{a_\ell^1}$  以及  $g^{r_1}, \dots, g^{r_s}$ ，然后将第  $i$  个输入比特在第  $j$  个电路中的密钥设置为  $g^{a_i^0 \cdot r_j}, g^{a_i^1 \cdot r_j}$ ,  $i \in [\ell]$ ,  $j \in [s]$ . 对于每个输入比特（即对于每个  $i \in [\ell]$ ），将  $P_1$  的密钥记为  $k_{i,1}, \dots, k_{i,s}$ . 令  $h_{i,0} = g^{a_i^0}$ ,  $h_{i,1} = g^{a_i^1}$ ,  $u_1 = g^{r_1}, \dots, u_s = g^{r_s}$ . 该零知识证明协议证明的是：存在  $a \in \mathbb{Z}_q$  满足 “ $h_{i,0} = g^a$  且对于  $j \in [s]$  有  $k_{i,j} = (u_j)^a$ ” 或 “ $h_{i,1} = g^a$  且对于  $j \in [s]$  有  $k_{i,j} = (u_j)^a$ ”。该证明与图 9.7 的零知识证明非常类似，如图 9.9 所示。

**输入一致性零知识证明**

**陈述：**  $(\mathbb{G}, g, h_0, h_1, u_1, \dots, u_s, k_1, \dots, k_s)$ , 其中  $\mathbb{G}$  是一个阶为  $q$ , 以  $g$  为生成元的群。

**见证：**  $a$  s.t. “ $h_0 = g^a$  且对于  $j \in [s]$  有  $k_j = (u_j)^a$ ” 或 “ $h_1 = g^a$  且对于  $j \in [s]$  有  $k_j = (u_j)^a$ ”

**协议：**

1. 验证者  $V$  随机选择  $\gamma_1, \dots, \gamma_s \leftarrow_{\$} \mathbb{Z}_q$  并发送给证明者  $P$ .
2.  $P$  和  $V$  本地计算
$$\tilde{u} = \prod_{i=1}^s (u_i)^{\gamma_i}, \quad \tilde{k} = \prod_{i=1}^s (k_i)^{\gamma_i}$$
3.  $P$  向  $V$  零知识地证明  $(g, h_0, \tilde{u}, \tilde{k})$  或  $(g, h_1, \tilde{u}, \tilde{k})$  是 DH 元组（只需在图 9.5 的协议中取  $s = 2$ ）。

图 9.9: 输入一致性零知识证明

### 9.2.2.3 协议描述

现在，我们给出 LP11 协议的完整描述。我们首先给出协议每一步的直观解释。我们将打开的电路称为“检查电路”(check-circuit)，未打开的电路称为“计算电路”(evaluation-circuit)。

**步骤 1：** $P_1$  构造  $s$  个混淆电路。这些混淆电路中， $P_1$  的输入导线密钥通过特殊的方式来选择： $P_1$  首先选择随机数  $a_1^0, a_1^1, \dots, a_\ell^0, a_\ell^1$  ( $P_1$  的输入长度为  $\ell$ ) 和  $r_1, \dots, r_s$ ，然后将第  $j$  个电路中的第  $i$  个输入导线的密钥设置为  $g^{a_i^0 \cdot r_j}, g^{a_i^1 \cdot r_j}$ . 此时， $g^{a_1^0}, g^{a_1^1}, \dots, g^{a_\ell^0}, g^{a_\ell^1}, g^{r_1}, \dots, g^{r_s}$  构成了  $P_1$  的输入密钥的承诺<sup>7</sup>。

**步骤 2：**参与双方执行批处理单一选择切分选择 OT，其中  $P_1$  的输入是所有电路中  $P_2$  的所有输入导线的密钥， $P_2$  的输入是他的协议输入和大小为  $s/2$  的随机集合  $\mathcal{J} \subset [s]$ . 于是，对于“检查电路”（索引在  $\mathcal{J}$  中的电路）， $P_2$  获得两个密钥，对于“计算电路”， $P_2$  获得其输入对应的密钥。

**步骤 3：** $P_1$  给  $P_2$  发送  $s$  个混淆电路以及  $P_1$  的输入密钥的承诺  $g^{a_1^0}, g^{a_1^1}, \dots, g^{a_\ell^0}, g^{a_\ell^1}, g^{r_1}, \dots, g^{r_s}$ . 此时， $P_1$  已经对  $s$  个电路做了承诺，但还不知道要打开哪些电路。

**步骤 4：** $P_2$  向  $P_1$  展示“检查电路”的集合  $\mathcal{J}$ ，并发送这些电路中  $P_2$  的第一个输入比特对应的两个密钥以证明其正确性。注意，只有“检查电路” $P_2$  才能知道两个密钥。

<sup>7</sup>实际上， $P_1$  的密钥设置为  $H(g^{a_i^0 \cdot r_j}), H(g^{a_i^1 \cdot r_j})$ ,  $H$  是随机数提取器，参见图 9.10 的协议描述。

**步骤 5:** 对于“检查电路”， $P_2$  需要知道  $P_1$  的输入导线密钥才能检查正确性。因此，对于  $j \in \mathcal{J}$ ， $P_1$  发送  $r_j$  给  $P_2$ 。于是， $P_2$  可以根据  $g^{a_i^0}, g^{a_i^1}, r_j$  来计算  $g^{a_i^0 \cdot r_j}, g^{a_i^1 \cdot r_j}$ ，并且这不会泄露“计算电路”中的密钥。

**步骤 6:**  $P_2$  检查  $s/2$  个“检查电路”的正确性，如果存在错误， $P_2$  终止协议。如果“检查电路”都是正确的，那么大多数“计算电路”（以压倒性的概率）是正确的。

**步骤 7:**  $P_1$  向  $P_2$  发送“计算电路”中  $P_1$  的输入密钥，即，对于“计算电路” $j$  的第  $i$  条导线， $P_1$  向  $P_2$  发送  $g^{a_i^{x_i} \cdot r_j}$ ，其中  $x_i$  是  $P_1$  的第  $i$  个输入比特。 $P_1$  通过上述输入一致性零知识证明（图 9.9）高效地证明所有密钥使用了相同的  $a_i^{x_i}$ 。

**步骤 8:**  $P_2$  计算所有“计算电路”，取大多数电路的输出作为协议输出。注意，即使输出不一致， $P_2$  也不能中止协议，否则可能泄露信息（见第9.2.1节）。由于在恶意敌手模型下，双方获得输出和只有一个参与方获得输出是等价的，因此我们考虑只有  $P_2$  获得输出的简单情况（见第9.2.1节）。

协议描述如图 9.10 所示。

### 9.2.3 安全性证明

直观上看，如果所有的“检查电路”都是正确的，那么大多数“计算电路”以压倒性的概率是正确的。另一方面，基于 DDH 假设， $P_2$  从  $g^{a_1^0}, g^{a_1^1}, \dots, g^{a_\ell^0}, g^{a_\ell^1}, g^{r_1}, \dots, g^{r_s}$  不能获得输入导线密钥的任何信息。因此，协议有两个安全参数：混淆电路的数量  $s$  是统计安全参数， $n = |q|$  是计算安全参数，其中  $q$  是群  $\mathbb{G}$  的阶。

下面，我们首先在独立模型下给出严格的安全性证明（考虑只有  $P_2$  获得输出的等价定义）。然后，我们讨论协议的 UC 安全性。

**定理 9.5.** 假设 DDH 问题在群  $\mathbb{G}$  中是困难的，假设步骤 7(b) 的零知识证明协议具有完备性、可靠性和零知识性，假设构造混淆电路使用的加密方案是安全的。那么，图 9.10 中的协议在  $\mathcal{F}_{\text{CCOT}}^{S,B}$ -混合模型下对于静态恶意敌手安全实现了功能  $f$ 。

证明. 考虑  $P_1$  被攻陷和  $P_2$  被攻陷两种情况。

**情况 1:**  $P_1$  被攻陷。

直观上看， $P_1$  只能通过构造错误的混淆电路来攻击协议。在协议中， $P_1$  在知道集合  $\mathcal{J}$  之前已经对所有电路做了承诺，因此，所有“检查电路”均正确但大多数“计算电路”不正确的概率是关于  $s$  的可忽略函数。如第9.2.1节的分析，这个概率小于  $2^{-s/4+1}$ 。注意，如果  $P_2$  发现“检查电路”的错误而终止协议，这与  $P_2$  的输入是无关的，因此不会泄露  $P_2$  的输入。下面，我们给出模拟器  $\mathcal{S}$  的构造。

**模拟器  $\mathcal{S}$ .** 令  $\mathcal{A}$  是控制  $P_1$  的敌手。模拟器  $\mathcal{S}$  在其内部运行  $\mathcal{A}$ ，模拟  $P_2$  向  $P_1$  发送的消息和理想功能  $\mathcal{F}_{\text{CCOT}}^{S,B}$ 。 $\mathcal{S}$  的工作方式如下：

1.  $\mathcal{S}$  记录  $P_1$  向  $\mathcal{F}_{\text{CCOT}}^{S,B}$  发送的输入（步骤 2）。这些输入构成一个  $\ell \times s$  的矩阵，矩阵的每个元素是一对消息，记为  $\{(z_0^{i,j}, z_1^{i,j})\}_{i \in [\ell], j \in [s]}$ 。
2.  $\mathcal{S}$  记录  $P_1$  向  $P_2$  发送的混淆电路  $GC_1, \dots, GC_s$  和  $\{(i, 0, u_i^0), (i, 1, u_i^1)\}_{i=1}^\ell, \{(j, h_j)\}_{j=1}^s$ （步骤 3）。
3.  $\mathcal{S}$  随机选择大小为  $s/2$  的集合  $\mathcal{J} \in [s]$ ，并模拟  $P_2$  发送集合  $\mathcal{J}$  和  $\{(z_0^{1,j}, z_1^{1,j})\}_{j \in \mathcal{J}}$  给  $P_1$ （步骤 4）。

## LP11 协议

**公共输入:** 统计安全参数  $s$ , 函数  $f$  等价的布尔电路  $\mathcal{C}, (\mathbb{G}, q, g)$ , 其中  $\mathbb{G}$  是阶为  $q$ , 以  $g$  为生成元的群。 $|q| = n$ 。  
**输入:**  $P_1$  的输入是  $x \in \{0, 1\}^\ell$ ,  $P_2$  的输入是  $y \in \{0, 1\}^\ell$ .

**协议:**

1. 构造混淆电路:

- (a)  $P_1$  选择随机值  $a_1^0, a_1^1, \dots, a_\ell^0, a_\ell^1 \leftarrow_{\$} \mathbb{Z}_q$  和  $r_1, \dots, r_s \leftarrow_{\$} \mathbb{Z}_q$ .
- (b) 设  $w_1, \dots, w_\ell$  是  $P_1$  在电路  $\mathcal{C}$  中的输入导线。用  $w_{i,j}$  表示第  $j$  个混淆电路中的导线  $w_i$ , 用  $k_{i,j}^b$  表示导线  $w_{i,j}$  对应于比特  $b$  的密钥。 $P_1$  将这些输入导线的密钥设置为

$$k_{i,j}^0 = H(g^{a_i^0 \cdot r_j}), k_{i,j}^1 = H(g^{a_i^1 \cdot r_j})$$

其中,  $H$  是一个随机数提取器<sup>a</sup>。

- (c)  $P_1$  按照第8.1.2节中所述的方式构造  $s$  个混淆电路, 记为  $GC_1, \dots, GC_s$ , 其中导线  $w_1, \dots, w_\ell$  的密钥按照上述方式确定。

2. 茫然传输:  $P_1$  和  $P_2$  以参数  $\ell$  和  $s$  执行批处理单一选择切分选择 OT (协议见第9.2.2.1节):

- (a)  $P_1$  定义向量  $\vec{z}_1, \dots, \vec{z}_\ell$ , 其中  $\vec{z}_i$  包含了  $P_2$  的输入比特  $y_i$  在  $GC_1, \dots, GC_s$  中的  $s$  对密钥。
- (b)  $P_2$  的输入是大小为  $s/2$  的随机集合  $\mathcal{J} \subset [s]$  和  $\sigma_1, \dots, \sigma_\ell \in \{0, 1\}$ , 其中  $\sigma_i = y_i, i \in [\ell]$ .
- (c) 对于  $GC_j, j \in \mathcal{J}$ ,  $P_2$  获得他的输入导线的两个密钥; 对于其他电路,  $P_2$  获得他的输入对应的密钥。

3. 发送电路和承诺:  $P_1$  向  $P_2$  发送  $s$  个混淆电路, 随机数提取器  $H$  的定义, 以及  $P_1$  的输入导线密钥的承诺  $\{(i, 0, g^{a_i^0}), (i, 1, g^{a_i^1})\}_{i=1}^\ell, \{(j, g^{r_j})\}_{j=1}^s$ .

4. 发送切分选择挑战:  $P_2$  向  $P_1$  发送集合  $\mathcal{J}$  以及电路  $\{GC_j\}_{j \in \mathcal{J}}$  中  $P_2$  的第一个输入比特  $y_1$  对应的两个密钥。如果密钥不正确,  $P_1$  输出  $\perp$  并终止协议。 $\{GC_j\}_{j \in \mathcal{J}}$  称为“检查电路”,  $\{GC_j\}_{j \notin \mathcal{J}}$  称为“计算电路”。

5. 发送“检查电路”中所有输入密钥: 对于每个“检查电路”  $GC_j$ ,  $P_1$  向  $P_2$  发送  $r_j$ .  $P_2$  检查该值与第 3 步中的  $(j, g^{r_j})$  是否一致。如果不一致,  $P_2$  输出  $\perp$  并终止协议。

6. “检查电路”正确性检验: 对于  $j \in \mathcal{J}$ ,  $P_2$  使用第 3 步得到的  $g^{a_i^0}, g^{a_i^1}$  和第 5 步得到的  $r_j$  计算  $GC_j$  中  $P_1$  的输入导线密钥  $k_{i,j}^0 = H(g^{a_i^0 \cdot r_j}), k_{i,j}^1 = H(g^{a_i^1 \cdot r_j})$ .  $P_2$  将自己的输入导线密钥设置为切分选择 OT 中获得的值。已知所有的输入导线密钥,  $P_2$  解密所有的混淆表来检查混淆电路的正确性。如果存在不正确的混淆电路,  $P_2$  输出  $\perp$  并终止协议。

7. 发送“计算电路”中的密钥:

- (a) 对于  $j \notin \mathcal{J}, i \in [\ell]$ ,  $P_1$  发送  $k'_{i,j} = g^{a_i^{x_i} \cdot r_j}$  给  $P_2$ .  $P_2$  设置  $k_{i,j} = H(k'_{i,j})$ .
- (b)  $P_1$  证明输入密钥的一致性: 对于  $i \in [\ell]$ ,  $P_1$  使用图 9.9 的协议并行地证明存在  $\sigma_i \in \{0, 1\}$  使得  $k'_{i,j} = g^{a_i^{\sigma_i} \cdot r_j}$  对  $j \notin \mathcal{J}$  都成立。如果证明不通过,  $P_2$  输出  $\perp$  并终止协议。

8. 对电路计算:  $P_2$  用第 7 步获得的  $P_1$  的输入密钥和第 2 步获得的  $P_2$  的输入密钥计算所有的“计算电路”  $\{GC_j\}_{j \notin \mathcal{J}}$ . 如果某个电路的计算失败了,  $P_2$  将其输入记为  $\perp$ . 最后,  $P_2$  取大多数电路的输出作为协议的输出。

<sup>a</sup>随机数提取器能从不完美的随机源中提取随机数。举例来说, 在 DDH 假设下, 已知  $g^x, g^y$  时,  $g^{xy}$  和随机群元素在计算上不可区分, 因此可以认为  $g^{xy}$  有  $\log_2(\text{order}(g)) = |q|$  位的熵 (entropy)。但是,  $g^{xy}$  的长度可能远远大于  $|q|$ , 例如,  $g$  是  $\mathbb{Z}_p^*$  中阶为  $q$  的一个元素,  $p, q$  是素数且  $q$  整除  $p - 1$ , 并且  $|p| = 1024$ ,  $|q| = 512$ . 此时, 512 位的熵分布在  $g^{xy}$  的 1024 位中, 因此我们不能直接将  $g^{xy}$  作为密钥, 而需要使用随机数提取器。读者可参阅文献<sup>[64-66]</sup>。

图 9.10: LP11 协议

4.  $\mathcal{S}$  记录  $P_1$  向  $P_2$  发送的  $\{r_j\}_{j \in \mathcal{J}}$ , 检查  $h_j = g_j^r$  对  $j \in \mathcal{J}$  是否均成立 (步骤 5)。如果不成立,  $\mathcal{S}$  向可信方发送  $\perp$ , 模拟  $P_2$  终止协议, 然后输出  $\mathcal{A}$  的输出。
5.  $\mathcal{S}$  按照协议规定检查  $\{GC_j\}_{j \in \mathcal{J}}$  的正确性 (步骤 6)。如果不正确,  $\mathcal{S}$  向可信方发送  $\perp$ , 模拟  $P_2$  终止协议, 然后输出  $\mathcal{A}$  的输出。
6.  $\mathcal{S}$  记录  $P_1$  向  $P_2$  发送的  $\{k'_{i,j}\}_{i \in [\ell], j \notin \mathcal{J}}$  (步骤 7(a)) .
7.  $\mathcal{S}$  从步骤 7(b) 的零知识证明中提取见证, 即, 对于  $i \in [\ell]$ ,  $\mathcal{S}$  提取见证  $a_i$  满足  $k'_{i,j} = (h_j)^{a_i}$  对  $j \notin \mathcal{J}$  都成立, 且  $u_i^0 = g^{a_i}$  或  $u_i^1 = g^{a_i}$ .
  - (a) 如果对于某个  $i$ ,  $\mathcal{S}$  没有提取合法的见证, 那么  $\mathcal{S}$  向可信方发送  $\perp$ , 模拟  $P_2$  终止协议, 然后输出  $\mathcal{A}$  的输出。
  - (b) 否则, 对于  $i \in [\ell]$ , 如果  $u_i^0 = g^{a_i}$  则设置  $x_1 = 0$ , 如果  $u_i^1 = g^{a_i}$  则设置  $x_1 = 1$ .
8.  $\mathcal{S}$  代表  $P_1$  向可信方发送  $x = x_1 \dots x_\ell$ , 然后输出  $\mathcal{A}$  的输出 (步骤 8)。

我们将图 9.10 中的协议记作  $\Pi$ , 我们需要证明

$$\{\text{IDEAL}_{f,\mathcal{S}}(x,y,n,s)\}_{x,y \in \{0,1\}^\ell; n,s \in \mathbb{N}} \stackrel{\text{comp}}{\approx} \{\text{REAL}_{\Pi,\mathcal{A}}(x,y,n,s)\}_{x,y \in \{0,1\}^\ell; n,s \in \mathbb{N}}$$

也就是任意 PPT 的敌手区分这两个随机变量的概率是关于  $n$  和  $s$  的可忽略函数。

在继续证明之前, 我们先定义一下什么是“坏的”电路。对于混淆电路  $GC_j$ , 它的输入导线密钥定义如下:

1.  $P_1$  的输入导线密钥: 令  $(1,0,g^{a_1^0}),(1,1,g^{a_1^1}), \dots, (\ell,0,g^{a_\ell^0}),(\ell,1,g^{a_\ell^1}),(j,g^{r_j})$  是步骤 3 中  $P_1$  发给  $P_2$  的值, 那么,  $GC_j$  中  $P_1$  的输入导线密钥为  $(g^{a_1^0 \cdot r_j},g^{a_1^1 \cdot r_j}), \dots, (g^{a_\ell^0 \cdot r_j},g^{a_\ell^1 \cdot r_j})$ .
2.  $P_2$  的输入导线密钥: 令  $(z_0^{1,j},z_1^{1,j}), \dots, (z_0^{\ell,j},z_1^{\ell,j})$  是第 2 步中  $P_1$  在切分选择 OT 中的输入 ( $\vec{z}_1, \dots, \vec{z}_\ell$  的第  $j$  个元素), 那么, 这些值是  $GC_j$  中  $P_2$  的输入导线密钥。

如果  $P_1$  和  $P_2$  的输入导线密钥不能将  $GC_j$  打开为正确的布尔电路  $\mathcal{C}$ , 那么我们说  $GC_j$  是“坏的”电路。注意, 当执行完步骤 3 时, 电路是否为“坏的”已经确定了。

我们将“检查电路”全部正确记为事件 noAbort, 将“计算电路”有一半或更多是“坏的”电路记为事件 badMaj, 将“坏的”电路的数量记为 badTotal. 下面的断言展示了 badMaj 和 noAbort 同时发生的概率的上界。

**断言 1.** 对于每个  $s \in \mathbb{N}$ , 有

$$\Pr[\text{noAbort} \wedge \text{badMaj}] = \frac{\binom{\frac{3s}{4} + 1}{\frac{s}{2} + 1}}{\binom{s}{\frac{s}{2}}} < \frac{1}{2^{s/4 - 1}}$$

证明. 该断言在第9.2.1节已经推导过。  $\square$

最后，我们说明如果  $\text{noAbort} \wedge \text{badMaj}$  没有发生，那么

$$\{\text{IDEAL}_{f,\mathcal{S}}(x,y,n,s)\}_{x,y \in \{0,1\}^\ell; n,s \in \mathbb{N}} \stackrel{\text{comp}}{\approx} \{\text{REAL}_{\Pi,\mathcal{A}}(x,y,n,s)\}_{x,y \in \{0,1\}^\ell; n,s \in \mathbb{N}}$$

根据零知识证明的可靠性， $\mathcal{S}$  未能提取合法见证的概率是可忽略的。因此  $\mathcal{S}$  提取的  $x$  就是  $P_1$  的输入。又因为  $\text{noAbort} \wedge \text{badMaj}$  没有发生，那么真实世界中大多数电路的输出就是可信方计算  $f(x,y)$  的结果。 $\mathcal{S}$  模拟  $P_2$  发送的集合  $\mathcal{J}$  和  $\{(z_0^{1,j}, z_1^{1,j})\}_{j \in \mathcal{J}}$  与真实世界完全相同，且  $\mathcal{S}$  只有在  $P_2$  终止协议时才向可信方发送  $\perp$ 。综上，我们完成了“ $P_1$  被攻陷”时的证明。

### 情况 2: $P_2$ 被攻陷。

$P_2$  被攻陷的情况更为简单。 $P_2$  在协议中收到了  $s/2$  个打开的“检查电路”和  $s/2$  个“计算电路”，对于“计算电路” $P_2$  只能获得输入导线的一个密钥，并且这些密钥对应于一致的  $x$  和  $y$ 。直观上看， $P_2$  除了计算这些电路之外做不了其他的事。在切分选择 OT 中，“单一选择”的性质使  $P_2$  无法为不同的电路输入不同的  $y$ 。此外，在步骤 4 中  $P_2$  可能试图发送  $\mathcal{J}' \neq \mathcal{J}$ ，但是他必须猜对混淆电路中输入比特  $y_1$  的另一个密钥，这个概率是可忽略的。下面，我们给出模拟器  $\mathcal{S}$  的构造。

**模拟器  $\mathcal{S}$ .** 令  $\mathcal{A}$  是控制  $P_2$  的敌手。模拟器  $\mathcal{S}$  在其内部运行  $\mathcal{A}$ ，模拟  $P_1$  向  $P_2$  发送的消息和理想功能  $\mathcal{F}_{\text{CCOT}}^{S,B}$ 。 $\mathcal{S}$  的工作方式如下：

1.  $\mathcal{S}$  记录  $P_2$  向  $\mathcal{F}_{\text{CCOT}}^{S,B}$  发送的输入  $\mathcal{J}$  和选择比特  $\sigma_1, \dots, \sigma_\ell$ 。如果  $\mathcal{J}$  的大小不是  $s/2$ ，那么  $\mathcal{S}$  向可信方发送  $\perp$ ，模拟  $P_1$  终止协议，然后输出  $\mathcal{A}$  的输出。
2.  $\mathcal{S}$  选择随机值  $a_1^0, a_1^1, \dots, a_\ell^0, a_\ell^1, r_1, \dots, r_s \leftarrow_{\$} \mathbb{Z}_q$ ，然后构造一个  $\ell \times s$  的矩阵，矩阵的每个元素是一对长度为  $n$  的密钥。对于  $P_1$  的输入导线密钥（即  $i \in [\ell]$ ），设置为  $(x_{i,j}^0, x_{i,j}^1) = (H(g^{a_i^0 \cdot r_j}), H(g^{a_i^1 \cdot r_j}))$ ，对于  $P_2$  的输入导线密钥，随机选取。
3.  $\mathcal{S}$  将这个矩阵作为  $P_1$  给  $\mathcal{F}_{\text{CCOT}}^{S,B}$  的输入，然后让  $P_2$  获得合理的输出（步骤 2），即，对于  $j \in \mathcal{J}$ ， $P_2$  获得  $\{(x_{i,j}^0, x_{i,j}^1)\}_{i \in [\ell]}$ ，对于  $j \notin \mathcal{J}$ ， $P_2$  获得  $\{x_{i,j}^{\sigma_i}\}_{i \in [\ell]}$ 。
4.  $\mathcal{S}$  代表  $P_2$  发送  $y = \sigma_1 \dots \sigma_\ell$  给可信方，并获得协议输出  $z$ 。
5. 对于  $j \in \mathcal{J}$ ， $\mathcal{S}$  诚实地构造  $GC_j$ 。
6. 对于  $j \notin \mathcal{J}$ ， $\mathcal{S}$  按照定理 8.4 中的方式构造“假的”混淆电路  $\widetilde{GC}_j$ ，它的输出总是  $z$ 。其中， $P_1$  和  $P_2$  的输入导线密钥使用上述步骤中选择的密钥。
7.  $\mathcal{S}$  模拟  $P_1$  发送混淆电路和  $\{(i, 0, g^{a_i^0}), (i, 1, g^{a_i^1})\}_{i=1}^\ell, \{(j, g^{r_j})\}_{j=1}^s$  给  $P_2$ （步骤 3）。
8.  $\mathcal{S}$  记录  $P_2$  向  $P_1$  发送的集合  $\mathcal{J}'$  以及密钥  $\{x_{1,j}^0, x_{1,j}^1\}_{j \in \mathcal{J}'}$ （步骤 4），做如下检查：
  - (a) 如果  $\mathcal{J}' \neq \mathcal{J}$  且两个密钥均正确， $\mathcal{S}$  输出  $\text{fail}$  并停止模拟。
  - (b) 如果  $\mathcal{J}' = \mathcal{J}$  但有密钥不正确， $\mathcal{S}$  向可信方发送  $\perp$ ，模拟  $P_1$  终止协议，然后输出  $\mathcal{A}$  的输出。
  - (c) 如果  $\mathcal{J}' = \mathcal{J}$  且密钥均正确， $\mathcal{S}$  继续模拟。
9.  $\mathcal{S}$  模拟  $P_1$  向  $P_2$  发送第 2 步中选择的  $\{r_j\}_{j \in \mathcal{J}}$ （步骤 5）。
10. 对于  $i \in [\ell], j \notin \mathcal{J}$ ， $\mathcal{S}$  模拟  $P_1$  向  $P_2$  发送  $k'_{i,j} = g^{a_i^0 \cdot r_j}$ ，并诚实地执行零知识证明协议证明输入一致性（步骤 7）。（ $\mathcal{S}$  不知道  $P_1$  的输入，它取  $x = 0^\ell$  为输入。）

11. 最后,  $\mathcal{S}$  输出  $\mathcal{A}$  的输出。

我们需要证明

$$\{\text{IDEAL}_{f,\mathcal{S}}(x,y,n,s)\}_{x,y \in \{0,1\}^\ell; n,s \in \mathbb{N}} \stackrel{\text{comp}}{\approx} \{\text{REAL}_{\Pi,\mathcal{A}}(x,y,n,s)\}_{x,y \in \{0,1\}^\ell; n,s \in \mathbb{N}}$$

首先,  $\mathcal{S}$  输出 `fail` 的概率是  $\mathcal{A}$  猜对随机密钥的概率, 它是可忽略的。其次, 在  $P_2$  的视图中, 唯一的区别在于“计算电路”被替换成了“假的”混淆电路。定理 8.4 中已经证明了, (一个)“假的”混淆电路与真实的混淆电路是不可区分的。现在  $P_2$  的视图中有  $s/2$  个“假的”混淆电路, 我们通过标准的混合论证可以证明: 它们与真实的混淆电路仍是不可区分的。注意, 上述证明必须在  $P_2$  获得一致的输入密钥时才成立, 而理想功能  $\mathcal{F}_{\text{CCOT}}^{S,B}$  的“单一选择”的性质保证了这一点。

证毕。 □

**UC 安全性。**到目前为止, 本节的所有定理考虑的都是独立模型下的安全性。但是, 注意到我们的证明中, 只有提取零知识证明的见证时使用了倒带技术, 其他步骤无需将敌手倒带。因此, 如果能将 Sigma 协议替换成 UC 安全的零知识证明协议, 那么整个协议就变成 UC 安全的了。在公共参考字符串 (Common Reference String, CRS) 模型下, 存在将 Sigma 协议转化为 UC 安全的零知识证明的高效构造<sup>[67-68]</sup>。于是, 我们有以下定理。

**定理 9.6.** 对于任意的输入长度为  $\ell$  的两方功能  $f$ , 在公共参考字符串模型和  $DDH$  假设下, 存在协议  $\Pi$  对于静态恶意敌手  $UC$ -安全实现了功能  $f$ .

## 9.3 BGW 协议 (恶意安全)

### 9.3.1 直观思想

在第3章和第6.1节, 我们介绍了 BGW 协议的半诚实安全版本, 它对于半诚实的敌手具有完美安全性。然而, 如果敌手是恶意的, 那么他有多种攻击协议的方式。假设参与方  $P_1$  是被攻陷的, 且输入为  $s$ 。首先, 在输入分享阶段,  $P_1$  应该选择一个  $t$  阶<sup>8</sup>多项式  $q(\cdot)$ , 满足  $q(0) = s$ , 并将秘密份额  $q(\alpha_i)$  发送给  $P_i$ 。但是, 恶意的  $P_1$  可以选择一个阶数大于  $t$  的多项式  $q(\cdot)$ , 将  $q(\alpha_i)$  发送给  $P_i$ , 这样一来, 任意  $t+1$  个秘密份额都会重建出不同的值, 因而  $P_1$  的输入是没有被良好定义的。其次, 在输出重建阶段, 假设输出值为  $y$ , 参与方持有  $y$  的秘密份额, 他们应该公布这些秘密份额来重建输出。但是, 恶意的参与方可以公布错误的秘密份额, 其他参与方无法知道哪些份额是正确的, 哪些是错误的。因此, 协议必须提供以下性质: 假设一共有  $n$  个参与方, 被攻陷的参与方不超过  $t$  个, 输出重建时可以基于  $n-t$  个正确的份额和  $t$  个错误的份额重建输出值。

对于前一种攻击, 解决方案为可验证秘密分享 (Verifiable Secret Sharing, VSS)。顾名思义, 可验证秘密分享 VSS 允许其他参与方“验证”(恶意的)秘密分发者 (dealer) 正确地分享了某个秘密, 即秘密值是被良好定义的<sup>9</sup>。对于后一种攻击, 解决方案为纠错码 (error-correcting code)。一个纠错码的码

<sup>8</sup>更准确地说, “ $t$  阶”应该是“至多  $t$  阶”。在均匀随机选取的情况下, 多项式阶数不为  $t$  的概率为  $1/|\mathbb{F}|$ , 其中  $\mathbb{F}$  是使用的域。为了表述的简洁, 本章中通常省略“至多”。

<sup>9</sup>对于恶意的参与方, 我们无法阻止其替换自己的输入。

字 (codeword) 包含多个元素，当其中有部分元素丢失或错误时<sup>10</sup>，如果数量不超过阈值，我们就可以将其“纠正”为原始的正确码字。

**可验证秘密分享。**回想一下，在  $(t, n)$ -Shamir 秘密分享方案中，参与方首先选定  $n$  个互不相同且不为 0 的值  $(\alpha_1, \dots, \alpha_n)$ 。要分享某个值  $s$ ，秘密分享算法首先选择阶数最多为  $t$  的随机多项式  $q(\cdot)$ ，满足  $q(0) = s$ ，然后将秘密份额  $q(\alpha_i)$  发送给  $P_i$ 。然而，恶意的秘密分发者可能会违背协议，例如，选择阶数大于  $t$  的多项式  $q(\cdot)$ ，要想可验证地进行秘密分享，我们可能有以下想法：

- 参与方共同确定一个以  $g$  为生成元的循环群  $\mathbb{G}$ 。算法随机选择  $q(x) = a_0 + a_1x + \dots + a_tx^t$ ，其中  $a_0 = s$ ，并公布验证值  $A_k = g^{a_k}$ ,  $k = 0, \dots, t$ ，然后将秘密份额  $s_i = q(\alpha_i)$  发送给  $P_i$ .  $P_i$  验证

$$g^{s_i} \stackrel{?}{=} \prod_{k=0}^t (A_k)^{(\alpha_i)^k}$$

这种方式被称为 Feldman VSS.

- 参与方共同确定一个以  $g$  为生成元的循环群  $\mathbb{G}$ ，以及元素  $h$ ，满足  $\log_g h$  未知。算法随机选择  $q(x) = a_0 + a_1x + \dots + a_tx^t$  和  $q'(x) = b_0 + b_1x + \dots + b_tx^t$ ，其中  $a_0 = s$ ，并公布验证值  $C_k = g^{a_k}h^{b_k}$ ,  $k = 0, \dots, t$ ，然后将秘密份额  $(s_i, s'_i) = (q(\alpha_i), q'(\alpha_i))$  发送给  $P_i$ .  $P_i$  验证

$$g^{s_i}h^{s'_i} \stackrel{?}{=} \prod_{k=0}^t (C_k)^{(\alpha_i)^k}$$

这种方式被称为 Pedersen VSS.

直观上看，上述两种 VSS 方案在离散对数 (Discrete-Log) 假设下，对于多项式时间的敌手实现了可验证性<sup>1112</sup>。然而，它们都不能实现完美安全性。对于 Feldman VSS，如果参与方有无限的计算力，他可以求解  $A_k = g^{a_k}$  的离散对数，得到  $a_k$ ，从而打破隐私性；对于 Pedersen VSS，虽然验证值  $C_k = g^{a_k}h^{b_k}$  完美地隐藏了  $a_k$ ，但如果秘密分发者有无限的计算力，他求解  $h$  的离散对数，然后就可以任意选择  $s_i$ ，再计算满足  $g^{s_i}h^{s'_i} = \prod_{k=0}^t (C_k)^{(\alpha_i)^k}$  的  $s'_i$ ，从而打破正确性。

BGW 的可验证秘密分享在  $t < n/3$  的条件下实现了完美安全性，这是因为 Shamir 秘密分享本质上是 Reed-Solomon 纠错码。下面，我们详细阐述。

**纠错码 (Reed-Solomon 码)。** Reed-Solomon 码是一种线性码 (linear code)，线性码是一种特殊的纠错码，线性码的码字的线性组合也是合法的码字。线性码有三个参数  $[n, k, d]$ 。在大小为  $q$  的有限域  $\mathbb{F}$  上的  $[n, k, d]$ -线性码，指的是其长度为  $n$  (每个码字包含  $n$  个域上的元素)，维度 (dimension) 为  $k$  (共有  $q^k$  个不同的码字)，距离为  $d$  (两个码字的汉明距离 (Hamming distance) 至少为  $d$ )。

BGW 协议需要构造一种长度为  $n$ ，维度为  $k = t + 1$ ，距离为  $n - t$  的线性码。其对应的 Reed-Solomon 码构造如下。令  $\mathbb{F}$  是一个有限域，满足  $|\mathbb{F}| > n$ 。令  $\alpha_1, \dots, \alpha_n$  为域上各不相同的值。令  $m = (m_0, \dots, m_t)$  是要被编码的消息，其中  $m_i \in \mathbb{F}$ ,  $i \in [t]$ . 编码算法如下：

1. 定义最高  $t$  阶的多项式  $p_m(x) = m_0 + m_1x + \dots + m_tx^t$ .

<sup>10</sup>一个具体的例子是，在通信传输的过程中，可能有部分信息发生丢失或错误。

<sup>11</sup>可验证性指：如果验证通过，那么任意  $t+1$  个秘密份额插值将得到同样的（不超过  $t$  阶的）多项式，即秘密分发者正确地进行了秘密分享。

<sup>12</sup>如果  $P_i$  发现秘密份额不满足验证条件，他将提出投诉 (complaint)，然后，参与方将执行协议的“投诉解决”机制，这部分并非本段的重点，因此略去，读者可参阅文献<sup>[69]</sup>。

2. 计算码字  $C(m) = \langle p_m(\alpha_1), \dots, p_m(\alpha_n) \rangle$ .

不难看出, 该编码的距离为  $n-t$  (因为任意两个至多  $t$  阶的多项式, 最多有  $t$  个  $\alpha$  点满足  $p_1(\alpha) = p_2(\alpha)$ . 而两个不同的消息  $m \neq m'$  定义了两个不同的多项式  $p_m \neq p_{m'}$ , 故  $C(m)$  和  $C(m')$  最多有  $t$  处相同)。用  $d(x, y)$  来表示码字  $x, y \in \mathbb{F}^n$  的汉明距离, 我们有以下编码理论中著名的定理。

**定理 9.7.** Reed-Solomon 码是有限域  $\mathbb{F}$  上的  $[n, t+1, n-t]$ -线性码, 且存在高效的解码算法能够纠正不超过  $\frac{n-t-1}{2}$  个错误, 即, 对于每个  $m \in \mathbb{F}^{t+1}$  和  $x \in \mathbb{F}^n$ , 如果  $d(x, C(m)) \leq \frac{n-t-1}{2}$ , 那么解码算法以  $x$  为输入时, 输出  $m$ .

当  $t < n/3$  时,  $n \geq 3t+1$ . 将  $n = 3t+1$  代入上述定理可知, 此时我们可以高效地纠正  $\frac{3t+1-t-1}{2} = t$  个错误。在 BGW 协议中, 参与方使用 Shamir 秘密分享方案。假设  $n$  个参与方持有秘密份额  $\{q(\alpha_i)\}_{i \in [n]}$ , 其中  $q(\cdot)$  是至多  $t$  阶的多项式,  $q(0) = s$ . 我们可以把秘密份额  $\langle q(\alpha_1), \dots, q(\alpha_n) \rangle$  看成 Reed-Solomon 码字。当需要重建秘密值时, 参与方直接广播自己的秘密份额。诚实的参与方一定会广播正确的份额, 而被攻陷方可能会广播错误值。但是, 由于被攻陷方的数量  $t < n/3$ , 最多有  $t$  个错误值, 因此诚实方可以运行 Reed-Solomon 解码算法高效地恢复出正确的多项式  $q(\cdot)$ , 并计算  $q(0) = s$ .

综上, 我们可以得出结论: 当  $t < n/3$  时, 在输出重建阶段, 被攻陷方无法攻击协议——即使他们提供错误的值, 诚实方也能 (以概率 1) 重建正确的秘密值。但是, 我们还需要一个有效的可验证秘密分享方案, 来强制 (被攻陷的) 秘密分发者分发的秘密份额位于  $t$  阶多项式上。下面我们详细阐述 BGW 的可验证秘密分享。

**二元多项式。** 二元多项式是 BGW 可验证秘密分享的核心工具。一个  $t$  阶的二元多项式, 指的是有两个变量的多项式, 且每个变量的阶数最多为  $t$ . 它可以写成如下的形式:

$$f(x, y) = \sum_{i=0}^t \sum_{j=0}^t a_{i,j} \cdot x^i \cdot y^j$$

我们用  $\mathcal{B}^{s,t}$  表示常数项为  $s$  的  $t$  阶二元多项式的集合。那么, 对于一个属于  $\mathcal{B}^{s,t}$  的二元多项式, 它的系数的数量为  $(t+1)^2 - 1 = t^2 + 2t$  (总共有  $(t+1)^2$  个系数, 但常数项已经被固定为  $s$ )。

在第3章中, 我们论证了:  $t+1$  个 (横坐标不同的) 点唯一确定一个  $t$  阶多项式, 即, 给定  $t+1$  个点  $\{(\alpha_k, \beta_k)\}_{k=1}^{t+1}$ , 存在唯一的  $t$  阶多项式  $f$  满足  $f(\alpha_k) = \beta_k$ . 将其推广到二元多项式, 直觉上, 我们可能会猜测以下结论成立:  $t+1$  个  $t$  阶一元多项式唯一确定一个  $t$  阶二元多项式。具体而言, 对于  $t$  阶二元多项式  $S(x, y)$ , 固定  $y$  为某个值  $\alpha$  就得到一个  $t$  阶一元多项式  $f(x) = S(x, \alpha)$ . 于是,  $t+1$  个 (互不相同的) 值  $\alpha_1, \dots, \alpha_{t+1}$  定义了  $t+1$  个  $t$  阶一元多项式  $f_k(x) = S(x, \alpha_k)$ . 反过来, 给定  $t+1$  个 (互不相同的) 值  $\alpha_1, \dots, \alpha_{t+1}$  和  $t+1$  个  $t$  阶一元多项式  $f_1(x), \dots, f_{t+1}(x)$ , 应该存在唯一的  $t$  阶二元多项式  $S(x, y)$ , 满足  $S(x, \alpha_k) = f_k(x)$ ,  $k = 1, \dots, t+1$ . 我们把上述猜想形式化地写成以下引理。

**引理 9.8.** 令  $t$  是一个非负整数。令  $\alpha_1, \dots, \alpha_{t+1}$  是  $t+1$  个域  $\mathbb{F}$  上的不同元素。令  $f_1(x), \dots, f_{t+1}(x)$  是  $t+1$  个  $t$  阶多项式。那么, 存在唯一的  $t$  阶二元多项式  $S(x, y)$ , 对  $k = 1, \dots, t+1$  都满足

$$S(x, \alpha_k) = f_k(x)$$

可以提前告诉读者: 该引理是正确的。但为了在本节一次性地连贯介绍 BGW 协议的总体思路, 我们将该引理的证明放在下一小节。

使用二元多项式进行可验证秘密分享。沿着引理 9.8 的思路，我们首先将  $t$  阶一元多项式  $q(z)$  “嵌入”二元多项式  $S(x, y)$  中，其中  $q(0) = s$ . 具体而言， $S(x, y)$  是均匀随机选择的  $t$  阶二元多项式，但满足条件  $S(0, z) = q(z)$ . 于是， $q(\alpha_1), \dots, q(\alpha_n)$  就是嵌入  $S(x, y)$  的 Shamir 秘密份额。然后，秘密分发者向  $P_i$  发送两个一元多项式  $f_i(x) = S(x, \alpha_i)$ ,  $g_i(y) = S(\alpha_i, y)$ . 根据定义，它们满足  $f_i(\alpha_j) = S(\alpha_j, \alpha_i) = g_j(\alpha_i)$ ,  $g_i(\alpha_j) = S(\alpha_i, \alpha_j) = f_j(\alpha_i)$ . 这样一来，任意两个参与方  $P_i$  和  $P_j$  就可以通过检查  $f_i(\alpha_j) \stackrel{?}{=} g_j(\alpha_i)$  和  $g_i(\alpha_j) \stackrel{?}{=} f_j(\alpha_i)$  来验证他们收到的多项式是否两两一致 (consistent)。 $P_i$  还需要检查自己的多项式是否满足  $f_i(\alpha_i) \stackrel{?}{=} g_i(\alpha_i)$ . 如果验证均通过， $P_i$  输出自己的秘密份额为  $f_i(0) = q(\alpha_i)$ .

接下来，我们分析一下该方案的正确性和隐私性。

**引理 9.9.** (*BGW VSS 正确性*) 令  $K \subseteq [n]$  是索引的集合，满足  $|K| \geq t + 1$ ，令  $\{f_k(x), g_k(y)\}_{k \in K}$  是  $t$  阶多项式对的集合，令  $\{\alpha_k\}_{k \in K}$  是域  $\mathbb{F}$  上互不相同的非 0 元素。如果对于所有  $i, j \in K$ ，都有  $f_i(\alpha_j) = g_j(\alpha_i)$ ，那么存在唯一的  $t$  阶二元多项式  $S$ ，满足  $f_k(x) = S(x, \alpha_k)$  和  $g_k(y) = S(\alpha_k, y)$  对所有  $k \in K$  都成立。

基于引理 9.8，不难证明引理 9.9 是正确的。同样，我们将该引理的证明放在下一小节。

我们还需要证明隐私性，也就是说，被攻陷方接收到的多项式  $\{f_i(x), g_i(y)\}_{i \in I}, |I| \leq t$  不会泄露关于  $s$  的信息。实际上，我们可以证明一个更强的引理：对于任意两个多项式  $q_1, q_2$ ，满足  $q_1(\alpha_i) = q_2(\alpha_i) = f_i(0)$  对  $i \in I$  都成立，当  $S(x, y)$  基于  $q_1(z)$  选择时，多项式集合  $\{f_i(x), g_i(y)\}_{i \in I}$  的分布与  $S(x, y)$  基于  $q_2(z)$  选择时  $\{f_i(x), g_i(y)\}_{i \in I}$  的分布是相同的。由此得到的推论是： $\{f_i(x), g_i(y)\}_{i \in I}$  不会泄露关于秘密是  $s_1 = q_1(0)$  还是  $s_2 = q_2(0)$  的任何信息。

**引理 9.10.** (*BGW VSS 隐私性*) 令  $\alpha_1, \dots, \alpha_n \in \mathbb{F}$  是  $n$  个互不相同的非 0 元素，集合  $I \subset [n]$  且  $|I| \leq t$ . 令  $q_1$  和  $q_2$  是两个域  $\mathbb{F}$  上的  $t$  阶多项式，满足  $q_1(\alpha_i) = q_2(\alpha_i)$  对  $i \in I$  都成立。那么，

$$\left\{ \{(i, S_1(x, \alpha_i), S_1(\alpha_i, y))\}_{i \in I} \right\} \stackrel{\text{perf}}{=} \left\{ \{(i, S_2(x, \alpha_i), S_2(\alpha_i, y))\}_{i \in I} \right\}$$

其中， $S_1(x, y)$  和  $S_2(x, y)$  是在约束条件  $S_1(0, z) = q_1(z)$  和  $S_2(0, z) = q_2(z)$  下均匀随机选择的  $t$  阶二元多项式。

该引理的证明与一元多项式的情形类似，实际上是二元情形的推广。同样，我们将该引理的证明放在下一小节。

**如果有人提出投诉 (complaint)?** 按照上述协议，如果所有验证都通过，没有参与方提出投诉，那么一切非常美好——被分享的秘密是良好定义的，且不会泄露任何信息。但是，如果有参与方提出投诉，表示他接收到的份额不满足  $f_i(\alpha_j) = g_j(\alpha_i)$  和  $g_i(\alpha_j) = f_j(\alpha_i)$ ，此时，协议该如何处理呢？

这里我们假设存在认证的 (authenticated) 广播信道（即广播者的身份是已知的）。对于参与方  $P_j$ ，按照协议，他从  $P_i$  处收到  $f_i(\alpha_j), g_i(\alpha_j)$ ，如果  $f_j(\alpha_i) \neq g_i(\alpha_j)$  或  $g_j(\alpha_i) \neq f_i(\alpha_j)$ ，这有可能是因为秘密分发者是被攻陷的，或  $P_i$  是被攻陷的。此时， $P_j$  广播  $(j, i, f_j(\alpha_i), g_j(\alpha_i))$  来提出投诉。注意，这里的  $f_j(\alpha_i), g_j(\alpha_i)$  是由秘密分发者发给  $P_j$  的多项式  $f_j(x), g_j(y)$  所定义的。然后，秘密分发者检查  $f_j(\alpha_i), g_j(\alpha_i)$  的值是否正确，如果正确，那就什么都不做；如果不正确，秘密分发者广播正确的多项式  $f_j(x), g_j(y)$ ，然后其他参与方检查  $f_j(x), g_j(y)$  与自己的多项式是否一致。

在秘密分发者是诚实的情况下，容易看出上述方案是可行的。因为这种情况下，所有诚实方一定会获得一致的多项式，投诉只可能是因为被攻陷方提出错误的投诉（此时秘密分发者会广播他的多项式，这不会给被攻陷方带来更多信息），或者被攻陷方向诚实方发送错误的值（此时秘密分发者不会广播诚

实方的多项式)。但是, 在秘密分发者是被攻陷的情况下, 协议必须保证: 在投诉解决阶段, 除非秘密分发者重新向所有诚实方发送了一致的多项式, 否则所有诚实方都应该拒绝并输出  $\perp$ 。因此, 投诉解决的方案需要小心的设计。我们先给出完整的协议, 如图 9.11 所示, 然后再解释其原理。

首先, 对于秘密分发者是诚实的情况, 正如上面的分析, 所有诚实方都将获得一致的多项式, 因此至少  $n - t$  个参与方会广播 `consistent` (至少有  $n - t$  个诚实方), 最后诚实的参与方  $P_j$  将输出  $f_j(0) = S(0, \alpha_j) = q(\alpha_j)$ 。

对于秘密分发者是被攻陷的情况, 诚实方  $P_j$  和  $P_k$  有可能收到不一致的多项式。如果他们收到了不一致的多项式, 即  $f_j(x), g_j(y)$  和  $f_k(x), g_k(y)$  满足  $f_j(\alpha_k) \neq g_k(\alpha_j)$  或  $g_j(\alpha_k) \neq f_k(\alpha_j)$ , 那么他们都会投诉, 构成 `joint complaint`。然后, 秘密分发者必须对此广播一条 `reveal` 消息来解决投诉。要让  $n - t$  个参与方广播 `consistent`, 至少要有  $(n - t) - t \geq t + 1$  个诚实方广播 `consistent`。这也就代表着,  $t + 1$  个 (或更多) 诚实方在第 1 轮就收到了与 `reveal` 消息中的新多项式两两一致的多项式, 根据引理 9.9, 这  $t + 1$  个 (或更多) 多项式已经唯一确定了二元多项式  $S(x, y)$ 。

最后, 我们还需要说明: 那些没有广播 `consistent` 的诚实方的多项式, 也是来自同样的二元多项式  $S(x, y)$  的。这里用反证法。假设存在诚实方  $P_j$  满足  $f_j(x) \neq S(x, \alpha_j)$  ( $g_j(y) \neq S(\alpha_j, y)$  的情况同理), 由于  $f_j(x)$  至多为  $t$  阶, 因此  $f_j(\alpha_k) = S(\alpha_k, \alpha_j)$  最多对于  $t$  个  $\alpha_k$  成立。所以,  $P_j$  的多项式上的点最多与  $t$  个广播了 `consistent` 的诚实方相一致 (广播了 `consistent` 的诚实方都有  $g_k(\alpha_j) = S(\alpha_k, \alpha_j)$ )。而现在至少有  $t + 1$  个诚实方广播了 `consistent`, 因此,  $P_j$  必定和某个广播了 `consistent` 的诚实方有 `joint complaint`, 并且已经被 `reveal` 消息解决了, 那么,  $P_j$  更新后的多项式也是与  $S(x, y)$  相一致的。至此, 我们完成了 BGW 可验证秘密分享的分析。

**乘法协议。**但是协议到这里还没有结束, 回想一下, 在半诚实安全的 BGW 协议中, 对输入进行秘密分享后, 我们需要逐个计算电路中的算术门。加法门与常数乘法门是线性操作, 可以在本地完成, 我们还需要设计一个对于恶意敌手安全的乘法协议。

假设乘法门的输入导线值为  $a$  和  $b$ , 参与方  $P_i$  持有  $t$  阶的秘密份额  $a_i$  和  $b_i$ 。秘密份额的乘积  $a_i \cdot b_i$  定义了一个  $2t$  阶的多项式, 其常数项为  $a \cdot b$ 。在半诚实安全的 BGW 协议中,  $P_i$  对份额  $a_i \cdot b_i$  再进行秘密分享, 然后将子份额线性重组, 从而降阶为  $t$  阶多项式, 其常数项仍为  $a \cdot b$ 。在恶意敌手模型下, 难题在于: 被攻陷方不一定会秘密分享正确的  $a_i \cdot b_i$ 。上文的可验证秘密分享只能保证被分享的秘密是被良好定义的, 我们还需要一个机制来强制参与方秘密分享  $a_i \cdot b_i$ 。

为此, 乘法协议总体分为三步:

1. 每个参与方首先可验证地秘密分享他们输入导线上的份额, 即,  $P_i$  对秘密份额  $a_i$  和  $b_i$  进行秘密分享。因为输入份额位于  $t$  阶多项式上, 它们构成以  $[n, t + 1, n - t]$  为参数的 Reed-Solomon 码, 因此可以通过纠错码的性质纠正最多  $t$  个错误, 使每个参与方得到正确的子份额。
2. 然后, 每个参与方秘密分享  $a_i \cdot b_i$ 。由于上一步已经对  $a_i$  和  $b_i$  正确地秘密分享, 因此这一步也能够可验证的进行。换句话说, 此步骤中, 参与方在已知  $a_i$  和  $b_i$  的子份额的情况下, 验证秘密分发者正确地秘密分享了  $a_i \cdot b_i$ 。
3. 最后, 每个参与方持有了  $a_i \cdot b_i$  的 ( $t$  阶的) 子份额。与半诚实安全的 BGW 协议中相同, 他们将子份额线性重组, 得到的份额落在新的  $t$  阶多项式上, 其常数项为  $a \cdot b$ 。

这里, 步骤 1 和 2 讲得有些模糊, 因为协议较为复杂, 实在无法用几句话概括总体思想。后文中会对其详细阐释。

### BGW 可验证秘密分享协议

**输入：**秘密分发者  $D = P_1$  持有至多  $t$  阶的多项式  $q(x)$ ,  $t < n/3$ . 其他参与方  $P_2, \dots, P_n$  没有输入。

**公共输入：**域  $\mathbb{F}$  和  $n$  个互不相同的非 0 元素  $\alpha_1, \dots, \alpha_n \in \mathbb{F}$ .

**协议：**

1. **第 1 轮（发送多项式）。**秘密分发者执行以下操作：

- (a) 均匀随机选择二元多项式  $S(x, y) \in \mathcal{B}^{q(0), t}$ , 满足条件  $S(0, z) = q(z)$ .
- (b) 对于  $i \in [n]$ , 定义多项式  $f_i(x) = S(x, \alpha_i)$ ,  $g_i(y) = S(\alpha_i, y)$ . 向  $P_i$  发送多项式  $f_i(x)$  和  $g_i(y)$ .

2. **第 2 轮（交换子份额）。**每个参与方  $P_i$  执行以下操作：

- (a) 保存从秘密分发者处收到的多项式  $f_i(x)$  和  $g_i(y)$  (如果阶数大于  $t$ , 则截断高于  $t$  阶的项)。
- (b) 对于  $j \in [n]$ , 发送  $f_i(\alpha_j)$  和  $g_i(\alpha_j)$  给  $P_j$  (注意, 这里涉及到发送给自己)。

3. **第 3 轮（投诉广播）。**每个参与方  $P_i$  执行以下操作：

- (a) 对于  $j \in [n]$ , 令  $(u_j, v_j)$  是在第 2 轮中从  $P_j$  处收到的值, 如果  $u_j \neq g_i(\alpha_j)$  或  $v_j \neq f_i(\alpha_j)$ , 广播  $\text{complaint}(i, j, f_i(\alpha_j), g_i(\alpha_j))$ . (注意, 当  $f_i(\alpha_i) \neq g_i(\alpha_i)$  时,  $P_i$  会广播  $\text{complaint}(i, i, f_i(\alpha_i), g_i(\alpha_i))$ .)
- (b) 如果没有参与方广播  $\text{complaint}$ , 每个参与方  $P_i$  输出  $f_i(0)$  并终止。

4. **第 4 轮（投诉解决）。**对于每条  $\text{complaint}$  消息, 秘密分发者执行以下操作：

- (a) 对于  $P_i$  广播的消息  $\text{complaint}(i, j, u, v)$ , 检查  $u \stackrel{?}{=} S(\alpha_j, \alpha_i)$  和  $v \stackrel{?}{=} S(\alpha_i, \alpha_j)$ . 如果都成立, 什么都不做; 否则, 广播  $\text{reveal}(i, f_i(x), g_i(y))$ .

5. **第 5 轮（评估投诉解决）。**每个参与方  $P_i$  执行以下操作：

- (a) 对于所有  $j \neq k$ , 如果看到  $P_k$  和  $P_j$  广播的两条消息  $\text{complaint}(k, j, u_1, v_1)$  和  $\text{complaint}(j, k, u_2, v_2)$  满足  $u_1 \neq v_2$  或  $v_1 \neq u_2$ , 那么将  $(j, k)$  记为 joint complaint. 对于  $j = k$ , 如果看到  $P_j$  广播的消息  $\text{complaint}(j, j, u, v)$  满足  $u \neq v$ , 那么将  $(j, j)$  也记为 joint complaint. 如果存在 joint complaint  $(j, k)$  且秘密分发者既没有广播  $\text{reveal}(k, f_k(x), g_k(y))$  也没有广播  $\text{reveal}(j, f_j(x), g_j(y))$ , 那么跳到步骤 6 (且不广播 consistent); 否则, 执行下一步。

- (b) 考虑秘密分发者广播的消息集合  $\text{reveal}(j, f_j(x), g_j(y))$  (如果阶数大于  $t$ , 则截断高于  $t$  阶的项):
  - i. 如果存在一条消息满足  $j = i$ , 那么将保存的多项式  $f_i(x), g_i(y)$  更新为  $f_j(x), g_j(y)$ , 然后跳到步骤 6 (且不广播 consistent)。
  - ii. 如果存在一条消息满足  $j \neq i$ , 并且  $f_i(\alpha_j) \neq g_j(\alpha_i)$  或  $g_i(\alpha_j) \neq f_j(\alpha_i)$ , 那么跳到步骤 6 (且不广播 consistent)。

如果秘密分发者广播的  $\text{reveal}$  消息均不满足以上任何一个条件, 那么执行下一步。

- (c) 广播 consistent.

6. **输出决定。**每个参与方  $P_i$  执行以下操作：

如果至少  $n - t$  个参与方广播了 consistent, 输出  $f_i(0)$ ; 否则, 输出  $\perp$ .

图 9.11: BGW 可验证秘密分享协议

协议概览。最后，我们总结一下恶意安全的 BGW 协议的总体流程。假设被攻陷方的数量不超过  $t$ ,  $t < n/3$ , 参与方存在点对点的安全信道和认证的广播信道。

1. 输入分享：每个参与方使用上文所述的可验证秘密分享协议分享自己的输入。
2. 逐门计算：对于加法门和常数乘法门，在本地对秘密份额进行相应的计算；对于乘法门，调用乘法协议进行计算。
3. 输出重建：对于输出值  $y$ , 每个参与方广播自己的秘密份额  $y_i$ , 然后调用 Reed-Solomon 解码算法恢复出  $y$  值。

下面，我们开始进行形式化的协议描述和安全性证明。我们将采用模块化的方式，将可验证秘密分享和乘法协议抽象为理想功能  $\mathcal{F}_{\text{VSS}}$  和  $\mathcal{F}_{\text{mult}}$ , 然后在  $\{\mathcal{F}_{\text{VSS}}, \mathcal{F}_{\text{mult}}\}$ -混合模型下实现安全计算协议<sup>13</sup>。

### 9.3.2 准备工作——Shamir 秘密分享的性质与引理证明

在开始正式的协议描述之前，让我们先做一些准备工作：我们给出关于 Shamir 秘密分享性质的几个引理和推论，以及在上一节中省略的引理证明。

**Shamir 秘密分享的性质。**在第3.1节中，我们证明了：任意  $t$  个 Shamir 秘密份额是  $\mathbb{F}^t$  上的均匀分布。我们用  $\mathcal{P}^{s,t}$  表示常数项为  $s$ , 阶数不超过  $t$  的所有多项式的集合。上述性质可以写成如下引理。

**引理 9.11.** 对于任意秘密  $s \in \mathbb{F}$ , 任意互不相同的非 0 元素  $\alpha_1, \dots, \alpha_n \in \mathbb{F}$ , 任意子集  $I \subset [n]$  满足  $|I| = \ell \leq t$ , 有

$$\left\{ \{f(\alpha_i)\}_{i \in I} \right\} \stackrel{\text{perf}}{=} \left\{ U_{\mathbb{F}}^{(1)}, \dots, U_{\mathbb{F}}^{(\ell)} \right\}$$

其中  $f(x) \leftarrow_{\$} \mathcal{P}^{s,t}$ ,  $U_{\mathbb{F}}^{(1)}, \dots, U_{\mathbb{F}}^{(\ell)}$  是  $\ell$  个独立的服从  $\mathbb{F}$  上的均匀分布的随机变量。

很直接地，对于任意的两个秘密  $s, s'$ , 有如下推论。

**推论 9.12.** 对于任意互不相同的非 0 元素  $\alpha_1, \dots, \alpha_n \in \mathbb{F}$ , 任意两个值  $s, s' \in \mathbb{F}$ , 任意子集  $I \subset [n]$  满足  $|I| = \ell \leq t$ , 以及任意  $\vec{y} \in \mathbb{F}^\ell$ , 有

$$\Pr_{f(x) \leftarrow_{\$} \mathcal{P}^{s,t}} [\vec{y} = (\{f(\alpha_i)\}_{i \in I})] = \Pr_{g(x) \leftarrow_{\$} \mathcal{P}^{s',t}} [\vec{y} = (\{g(\alpha_i)\}_{i \in I})] = \frac{1}{|\mathbb{F}|^\ell}$$

在实际协议中，每个参与方都需要分享自己的输入。假设共有  $m$  个输入，被攻陷方将收到  $m \cdot |I|$  个秘密份额。我们将上述结论推广： $m \cdot |I|$  个秘密份额也不会泄露关于秘密的任何信息。这是因为这些份额服从  $\mathbb{F}^{m \cdot |I|}$  上的均匀分布。

**推论 9.13.** 对于任意  $m \in \mathbb{N}$ , 任意互不相同的非 0 元素  $\alpha_1, \dots, \alpha_n \in \mathbb{F}$ , 任意两个秘密集合  $(a_1, \dots, a_m) \in \mathbb{F}^m$  和  $(b_1, \dots, b_m) \in \mathbb{F}^m$ , 任意子集  $I \subset [n]$  满足  $|I| = \ell \leq t$ , 有

$$\left\{ \{(f_1(\alpha_i), \dots, f_m(\alpha_i))\}_{i \in I} \right\} \stackrel{\text{perf}}{=} \left\{ \{(g_1(\alpha_i), \dots, g_m(\alpha_i))\}_{i \in I} \right\}$$

其中，对于每个  $j$ ,  $f_j(x), g_j(x)$  都是分别从  $\mathcal{P}^{a_j,t}$  和  $\mathcal{P}^{b_j,t}$  中均匀随机选取的。

<sup>13</sup>本节参考了文献<sup>[70]</sup>。

在 Shamir 秘密分享方案中，秘密分发者随机选取至多  $t$  阶的多项式，其常数项为秘密  $s$ 。我们考虑另一种选取多项式的方式，它将最高次项系数 (leading coefficient) 固定为秘密  $s$ ，即  $f(x) = a_0 + a_1x + \dots + a_{t-1}x^{t-1} + sx^t$ 。我们用  $\mathcal{P}_{s,t}^{\text{lead}}$  表示最高次项  $x^t$  系数为  $s$  的所有  $t$  阶多项式的集合。与引理 9.11 的证明过程类似，此时的秘密份额  $\{f(\alpha_i)\}_{i \in I}$  仍为  $\mathbb{F}^{|I|}$  上的均匀随机分布。<sup>14</sup>

**引理 9.14.** 对于任意秘密  $s \in \mathbb{F}$ ，任意互不相同的非 0 元素  $\alpha_1, \dots, \alpha_n \in \mathbb{F}$ ，任意子集  $I \subset [n]$  满足  $|I| = \ell \leq t$ ，有

$$\left\{ \{f(\alpha_i)\}_{i \in I} \right\} \stackrel{\text{perf}}{=} \left\{ U_{\mathbb{F}}^{(1)}, \dots, U_{\mathbb{F}}^{(\ell)} \right\}$$

其中  $f(x) \leftarrow_{\$} \mathcal{P}_{s,t}^{\text{lead}}$ ， $U_{\mathbb{F}}^{(1)}, \dots, U_{\mathbb{F}}^{(\ell)}$  是  $\ell$  个独立的服从  $\mathbb{F}$  上的均匀分布的随机变量。

**引理证明。**下面，我们给出上一节中省略的引理证明。

**引理 9.8.** 令  $t$  是一个非负整数。令  $\alpha_1, \dots, \alpha_{t+1}$  是  $t+1$  个域  $\mathbb{F}$  上的不同元素。令  $f_1(x), \dots, f_{t+1}(x)$  是  $t+1$  个  $t$  阶多项式。那么，存在唯一的  $t$  阶二元多项式  $S(x, y)$ ，对  $k = 1, \dots, t+1$  都满足

$$S(x, \alpha_k) = f_k(x) \quad (9.10)$$

证明. 通过拉格朗日插值，定义二元多项式  $S(x, y)$  如下：

$$S(x, y) = \sum_{i=1}^{t+1} f_i(x) \cdot \frac{\prod_{j \neq i} (y - \alpha_j)}{\prod_{j \neq i} (\alpha_i - \alpha_j)}$$

不难看出， $S(x, y)$  为  $t$  阶二元多项式，且对  $k = 1, \dots, t+1$  有

$$\begin{aligned} S(x, \alpha_k) &= \sum_{i=1}^{t+1} f_i(x) \cdot \frac{\prod_{j \neq i} (\alpha_k - \alpha_j)}{\prod_{j \neq i} (\alpha_i - \alpha_j)} \\ &= f_k(x) \cdot \frac{\prod_{j \neq k} (\alpha_k - \alpha_j)}{\prod_{j \neq k} (\alpha_k - \alpha_j)} + \sum_{i \in [t+1] \setminus \{k\}} f_i(x) \cdot \frac{\prod_{j \neq i} (\alpha_k - \alpha_j)}{\prod_{j \neq i} (\alpha_i - \alpha_j)} \\ &= f_k(x) + 0 \\ &= f_k(x) \end{aligned}$$

故  $S(x, y)$  满足式 (9.10)。下面只需要证明  $S(x, y)$  是唯一的。假设存在两个不同的  $t$  阶二元多项式  $S_1, S_2$  均满足式 (9.10)。定义多项式  $R(x, y)$  如下

$$R(x, y) = S_1(x, y) - S_2(x, y) = \sum_{i=0}^t \sum_{j=0}^t r_{i,j} \cdot x^i y^j$$

我们证明  $R(x, y) = 0$ 。首先，对于  $k \in [t+1]$ ，有

$$R(x, \alpha_k) = \sum_{i,j=0}^t r_{i,j} \cdot x^i (\alpha_k)^j = S_1(x, \alpha_k) - S_2(x, \alpha_k) = f_k(x) - f_k(x) = 0$$

我们可以将  $R(x, \alpha_k)$  重新写为

$$R(x, \alpha_k) = \sum_{i=0}^t \left( \left( \sum_{j=0}^t r_{i,j} \cdot (\alpha_k)^j \right) \cdot x^i \right)$$

<sup>14</sup>这个引理是为证明后文中的定理做准备。

因为  $R(x, \alpha_k) = 0$ , 所以  $R(x, \alpha_k)$  的系数全为 0, 即, 对每个固定的  $i \in \{0, 1, \dots, t\}$  有  $\sum_{j=0}^t r_{i,j} \cdot (\alpha_k)^j = 0$ . 也就是说, 对每个固定的  $i \in \{0, 1, \dots, t\}$ , 多项式  $h_i(x) = \sum_{j=0}^t r_{i,j} \cdot x^j$  在  $t+1$  个点  $\alpha_1, \dots, \alpha_{t+1}$  处的值都为 0, 因而  $h_i(x)$  也是 0 多项式, 它的系数  $r_{i,j}$  均为 0 ( $j \in \{0, 1, \dots, t\}$ ). 因此, 对于  $i, j \in \{0, 1, \dots, t\}$ , 都有  $r_{i,j} = 0$ . 于是我们得出结论, 对于所有的  $x, y$  都有  $R(x, y) = 0$ , 即  $S_1(x, y) = S_2(x, y)$ .

证毕。  $\square$

**引理 9.9.** (BGW VSS 正确性) 令  $K \subseteq [n]$  是索引的集合, 满足  $|K| \geq t + 1$ , 令  $\{f_k(x), g_k(y)\}_{k \in K}$  是  $t$  阶多项式对的集合, 令  $\{\alpha_k\}_{k \in K}$  是域  $\mathbb{F}$  上互不相同的非 0 元素. 如果对于所有  $i, j \in K$ , 都有  $f_i(\alpha_j) = g_j(\alpha_i)$ , 那么存在唯一的  $t$  阶二元多项式  $S$ , 满足  $f_k(x) = S(x, \alpha_k)$  和  $g_k(y) = S(\alpha_k, y)$  对所有  $k \in K$  都成立.

证明. 令  $L$  是一个大小恰好为  $t+1$  的  $K$  的子集. 由引理 9.8 可知, 存在唯一的  $t$  阶二元多项式  $S(x, y)$  满足  $S(x, \alpha_\ell) = f_\ell(x)$  对  $\ell \in L$  都成立. 下面我们证明, 如果  $f_i(\alpha_j) = g_j(\alpha_i)$  对  $i, j \in K$  都成立, 那么对所有  $k \in K$  都有  $f_k(x) = S(x, \alpha_k)$  和  $g_k(y) = S(\alpha_k, y)$ .

首先, 根据引理的条件, 对于所有  $k \in K$  和  $\ell \in L$  都有  $g_k(\alpha_\ell) = f_\ell(\alpha_k)$ . 根据  $S(x, y)$  的定义, 有  $f_\ell(\alpha_k) = S(\alpha_k, \alpha_\ell)$ . 因此, 对于所有  $k \in K$  和  $\ell \in L$  都有  $g_k(\alpha_\ell) = S(\alpha_k, \alpha_\ell)$ . 由于  $g_k(y)$  和  $S(\alpha_k, y)$  都是  $t$  阶多项式, 它们在  $t+1$  个点  $\alpha_\ell$  处满足  $g_k(\alpha_\ell) = S(\alpha_k, \alpha_\ell)$ , 因此对于  $k \in K$  都有  $g_k(y) = S(\alpha_k, y)$ .

我们还需要证明  $f_k(x) = S(x, \alpha_k)$  对  $k \in K$  都成立. 根据  $S(x, y)$  的定义, 当  $k \in L$  时该式一定成立, 我们需要证明  $k \in K \setminus L$  的情况. 同样, 根据引理的条件, 对于所有  $j, k \in K$  都有  $f_k(\alpha_j) = g_j(\alpha_k)$ . 而上面已经证明  $g_j(\alpha_k) = S(\alpha_j, \alpha_k)$  对  $j, k \in K$  都成立. 因此, 对  $j, k \in K$  都有  $f_k(\alpha_j) = S(\alpha_j, \alpha_k)$ . 由于  $f_k(x)$  和  $S(x, \alpha_k)$  都是  $t$  阶多项式, 它们在至少  $t+1$  个点  $\alpha_j$  处满足  $f_k(\alpha_j) = S(\alpha_k, \alpha_j)$ , 因此对于  $k \in K$  都有  $f_k(x) = S(x, \alpha_k)$ .

证毕。  $\square$

**引理 9.10.** (BGW VSS 隐私性) 令  $\alpha_1, \dots, \alpha_n \in \mathbb{F}$  是  $n$  个互不相同的非 0 元素, 集合  $I \subset [n]$  且  $|I| \leq t$ . 令  $q_1$  和  $q_2$  是两个域  $\mathbb{F}$  上的  $t$  阶多项式, 满足  $q_1(\alpha_i) = q_2(\alpha_i)$  对  $i \in I$  都成立. 那么,

$$\left\{ \{(i, S_1(x, \alpha_i), S_1(\alpha_i, y))\}_{i \in I} \right\} \stackrel{\text{perf}}{=} \left\{ \{(i, S_2(x, \alpha_i), S_2(\alpha_i, y))\}_{i \in I} \right\}$$

其中,  $S_1(x, y)$  和  $S_2(x, y)$  是在约束条件  $S_1(0, z) = q_1(z)$  和  $S_2(0, z) = q_2(z)$  下均匀随机选择的  $t$  阶二元多项式。

证明. 定义概率集合 (probability ensemble)  $\mathbb{S}_1$  和  $\mathbb{S}_2$  如下:

$$\mathbb{S}_1 = \left\{ \{(i, S_1(x, \alpha_i), S_1(\alpha_i, y))\}_{i \in I} \mid S_1 \leftarrow_{\$} \mathcal{B}^{q_1(0), t} \text{ s.t. } S_1(0, z) = q_1(z) \right\}$$

$$\mathbb{S}_2 = \left\{ \{(i, S_2(x, \alpha_i), S_2(\alpha_i, y))\}_{i \in I} \mid S_2 \leftarrow_{\$} \mathcal{B}^{q_2(0), t} \text{ s.t. } S_2(0, z) = q_2(z) \right\}$$

上述引理等价于证明  $\mathbb{S}_1 \stackrel{\text{perf}}{=} \mathbb{S}_2$ .

我们首先证明, 对于任意的多项式对集合  $Z = \{(i, f_i(x), g_i(y))\}_{i \in I}$ ,  $\mathbb{S}_1$  的支持集 (support) 中与  $Z$  一致的二元多项式的数量等于  $\mathbb{S}_2$  的支持集中与  $Z$  一致的二元多项式的数量, 这里的一致指的是  $f_i(x) = S(x, \alpha_i)$  且  $g_i(y) = S(\alpha_i, y)$ .

注意到, 如果存在  $i, j \in I$  满足  $f_i(\alpha_j) \neq g_j(\alpha_i)$ , 那么  $\mathbb{S}_1$  和  $\mathbb{S}_2$  的支持集中都不存在与  $Z$  一致的二元多项式. 另外, 如果存在  $i \in I$  满足  $f_i(0) \neq q_1(\alpha_i)$ , 那么  $\mathbb{S}_1$  和  $\mathbb{S}_2$  的支持集中也不存在与  $Z$  一

致的二元多项式（这是因为对于  $\mathbb{S}_1$  而言  $f_i(0) = S(0, \alpha_i) = q_1(\alpha_i)$  应该成立，对于  $\mathbb{S}_2$  也同理，因为  $q_1(\alpha_i) = q_2(\alpha_i)$  对  $i \in I$  都成立）。

令  $Z = \{(i, f_i(x), g_i(y))\}_{i \in I}$  是  $t$  阶多项式对的集合，且满足  $f_i(\alpha_j) = g_j(\alpha_i)$  对  $i, j \in I$  都成立， $f_i(0) = q_1(\alpha_i) = q_2(\alpha_i)$  对  $i \in I$  都成立。首先，我们计算  $\mathbb{S}_1$  中多项式的数量。因为  $Z$  包含  $I$  个  $t$  阶多项式  $\{f_i(x)\}_{i \in I}$ ，并且  $t+1$  个这样的多项式唯一确定一个  $t$  阶二元多项式，因此，需要再选择  $t+1-|I|$  个与  $q_1(z)$ ,  $\{g_i(y)\}_{i \in I}$  相一致的多项式  $f_j(x)$  ( $j \neq i$ )，即  $f_j(x)$  必须满足  $f_j(\alpha_i) = g_i(\alpha_j)$  对  $i \in I$  都成立，且  $f_j(0) = q_1(\alpha_j)$ 。由此可知， $f_j$  有  $|I|+1$  个点已经确定了，而  $t+1$  个点唯一确定一个  $t$  阶多项式，所以其余的  $t-|I|$  个点可以任意地选择。于是，总共有  $(|\mathbb{F}|^{t-|I|})^{(t+1-|I|)}$  种方式选择  $S_1$ ，使其与  $Z$  相一致。接着我们计算  $\mathbb{S}_2$  中多项式的数量，其过程与  $\mathbb{S}_1$  的计算完全相同，我们得到  $S_2$  的选择方式也有  $(|\mathbb{F}|^{t-|I|})^{(t+1-|I|)}$  种。

至此，我们证明了  $\mathbb{S}_1$  的支持集中与  $Z$  一致的多项式的数量与  $\mathbb{S}_2$  相等。又因为  $S_1$  和  $S_2$  分别是从  $\mathbb{S}_1$  和  $\mathbb{S}_2$  的支持集中均匀随机选取的，所以无论概率集合是  $\mathbb{S}_1$  还是  $\mathbb{S}_2$ ，得到  $Z$  的概率都是相等的。证毕。  $\square$

### 9.3.3 BGW 可验证秘密分享

接下来，我们正式进入 BGW 协议的介绍。我们首先介绍 BGW 可验证秘密分享，它的协议已经在上文的讨论中给出，如图 9.11 所示。现在，我们要形式化地证明其安全性。

首先，我们定义理想功能  $\mathcal{F}_{VSS}$ ，如图 9.12 所示。

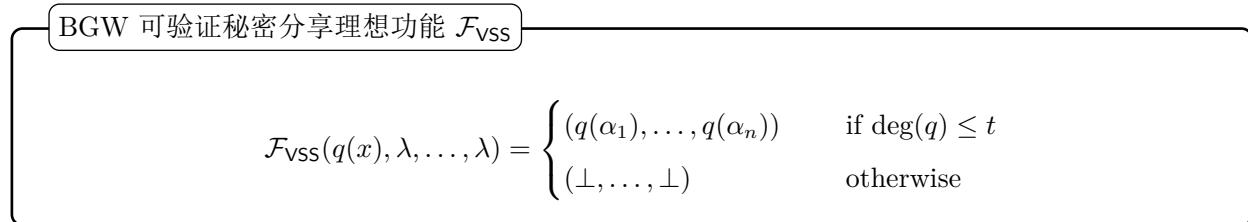


图 9.12: BGW 可验证秘密分享理想功能

在理想功能  $\mathcal{F}_{VSS}$  中，秘密分发者的输入是多项式  $q(x)$ ，其他参与方没有输入；协议的输出是  $P_i$  获得秘密份额  $q(\alpha_i)$ 。这里的“可验证性”体现在：(1) 如果  $q$  的阶数高于  $t$ ，那么所有参与方都会拒绝并输出  $\perp$ ；(2) 秘密  $s = q(0)$  被隐式地定义了，因而一定是良好定义的。当参与方需要可验证地分享秘密  $s$  时，秘密分发者首先随机选取多项式  $q \leftarrow_{\$} \mathcal{P}^{s,t}$ ，然后以  $q(x)$  为输入运行  $\mathcal{F}_{VSS}$  即可。

注意，这里的  $\mathcal{F}_{VSS}$  是专门为 BGW 协议而定义的，我们也可以定义一种更广义的 VSS（类似于多方场景下的承诺）。但因为 BGW 协议只需要对秘密份额  $q(\alpha_1), \dots, q(\alpha_n)$  做计算，所以我们采用这种更简单的定义方式。

我们有以下安全性定理。

**定理 9.15.** 假设参与方之间存在点对点安全信道、认证的广播信道。假设被攻陷方数量不超过  $t$ ,  $t < n/3$ ，图 9.11 中的协议  $\Pi_{VSS}$  对于静态恶意敌手  $UC$ -安全实现了理想功能  $\mathcal{F}_{VSS}$ （图 9.12）。

证明. 要证明此定理，我们需要构造模拟器  $\mathcal{S}$ ，使得对所有环境  $\mathcal{Z}$ ，都有

$$\text{EXEC}_{\Pi_{VSS}, \mathcal{A}, \mathcal{Z}} \approx \text{EXEC}_{\mathcal{F}_{VSS}, \mathcal{S}, \mathcal{Z}}$$

其中  $\mathcal{A}$  是平凡敌手。

我们分两种情况考虑：(1) 秘密分发者是诚实的，(2) 秘密分发者是被攻陷的。对于第 1 种情况，我们证明诚实方总会接受分发的秘密份额，即敌手无法通过错误的投诉来影响最终结果；第 2 种情况更复杂些，我们需要证明：如果秘密分发者解决了投诉，使得至少  $n - t$  个参与方广播了 `consistent`，那么此时所有诚实方一定持有着一致的秘密份额。

### 情况 1：秘密分发者是诚实的。

在这种情况的理想世界中，秘密分发者向  $\mathcal{F}_{\text{VSS}}$  发送  $q(x)$ ，每个诚实方  $P_j$  从  $\mathcal{F}_{\text{VSS}}$  处接收到  $q(\alpha_j)$ 。因为此时被攻陷方没有输入，所以敌手无法影响诚实方的输出。我们要证明：在真实世界中，诚实方  $P_j$  也总是输出  $q(\alpha_j)$ ，不会输出  $\perp$ 。

由于秘密分发者是诚实的，他将遵循协议选择二元多项式，并向每个参与方发送协议规定的一元多项式。此时，诚实方  $P_j$  只可能输出  $f_j(0) = S(0, \alpha_j) = q(\alpha_j)$  或  $\perp$  ( $P_j$  的多项式  $f_j(x)$  发生改变只可能是秘密分发者在解决投诉阶段发送了  $\text{reveal}(j, f'_j(x), g_j(y))$  且  $f'_j(x) \neq f_j(x)$ ，但是诚实的秘密分发者不会这么做)。下面，我们说明  $P_j$  不会输出  $\perp$ 。在协议中，当至少  $n - t$  个参与方广播 `consistent` 时，诚实方就会输出  $f_j(0)$  而不是  $\perp$ 。我们只需证明所有诚实方都会广播 `consistent`。对于诚实方  $P_j$ ，他广播 `consistent` 当且仅当以下条件成立：

1. 秘密分发者解决了所有投诉：如果存在 `joint complaint` 消息  $\text{complaint}(k, \ell, u_1, v_1)$  和  $\text{complaint}(\ell, k, u_2, v_2)$  满足  $u_1 \neq v_2$  或  $v_1 \neq u_2$ ，秘密分发者在步骤 4(a)，对  $k$  或  $\ell$  或两者广播了 `reveal` 消息。（见步骤 5(a)）
2. 秘密分发者没有广播  $\text{reveal}(j, f_j(x), g_j(y))$ 。（见步骤 5(b)i）
3. `reveal` 消息与  $P_j$  的多项式相一致：每条 `reveal` 消息  $\text{reveal}(k, f_k(x), g_k(y))$  都满足  $g_k(\alpha_j) = f_j(\alpha_k)$  和  $f_k(\alpha_j) = g_j(\alpha_k)$ 。（见步骤 5(b)ii）

在秘密分发者是诚实的情况下，对于每个 `joint complaint`，他都会广播至少一条 `reveal` 消息。这是因为当  $u_1 \neq v_2$  或  $v_1 \neq u_2$  时，不可能  $(u_1, v_1)$  和  $(u_2, v_2)$  都与  $S(x, y)$  相一致，故条件 1 满足。而且，因为秘密分发者是诚实的，条件 3 显然满足。最后，只有当  $P_j$  的 `complaint` 消息中包含错误的  $(u, v)$  值时，秘密分发者才会广播  $\text{reveal}(j, f_j(x), g_j(y))$ 。由于  $P_j$  是诚实的，这种情况不会发生，故条件 2 也满足。综上，所有诚实方都会广播 `consistent`，因而每个诚实方  $P_j$  都会输出  $f_j(0) = q(\alpha_j)$ 。

现在，我们可以给出模拟器  $\mathcal{S}$  的构造。假设被攻陷方的集合为  $I$ ，模拟器  $\mathcal{S}$  的工作方式如下：

- $\mathcal{S}$  在内部运行  $\mathcal{A}$ ，转发  $\mathcal{A}$  与环境  $\mathcal{Z}$  的消息。
- 当  $\mathcal{S}$  得到被攻陷方的输出  $\{q(\alpha_i)\}_{i \in I}$  时，它需要模拟被攻陷方的视图： $\mathcal{S}$  随机选择多项式  $q'(x)$ ，满足条件  $q'(\alpha_i) = q(\alpha_i)$  对  $i \in I$  都成立。然后， $\mathcal{S}$  以  $q'(x)$  作为秘密分发者的输入，模拟所有诚实方遵循协议执行。

我们证明  $\mathcal{Z}$  在理想世界和真实世界的视图是完美不可区分的。首先，观察到模拟的被攻陷方视图是由敌手  $\mathcal{A}$  和多项式对  $\{(f_i(x), g_i(y))\}_{i \in I}$  所决定的。这是因为除  $\{(f_i(x), g_i(y))\}_{i \in I}$  以外的消息只有投诉消息、投诉解决消息和 `consistent` 消息。由于秘密分发者是诚实的，所以诚实方  $P_j$  提出投诉只可能是他收到了来自被攻陷方的错误的  $(u_i, v_i)$  值，此时投诉消息为  $\text{complaint}(j, i, f_j(\alpha_i), g_j(\alpha_i))$ ，它等于

$\text{complaint}(j, i, g_i(\alpha_j), f_i(\alpha_j))$  (因为秘密分发者是诚实的), 故这条投诉消息已经由  $\{(f_i(x), g_i(y))\}_{i \in I}$  所决定了。对于投诉解决消息, 秘密分发者只会发送正确的  $\text{reveal}(i, f_i(x), g_i(y))$ , 它也由  $\{(f_i(x), g_i(y))\}_{i \in I}$  所决定了。由此可知, 模拟视图中的消息由  $\{(f_i(x), g_i(y))\}_{i \in I}$  所决定, 该结论对真实世界也成立。因此, 理想世界与真实世界的唯一区别在于  $\{(f_i(x), g_i(y))\}_{i \in I}$  是基于  $q'(x)$  还是  $q(x)$  所选择的。但是根据引理 9.10, 两者的分布完全相同, 故  $\mathcal{Z}$  在理想世界和真实世界的视图是完美不可区分的。

#### 情况 2: 秘密分发者是被攻陷的。

在这种情况下, 敌手  $\mathcal{A}$  控制了秘密分发者,  $\mathcal{S}$  需要模拟所有诚实方。注意到, 除了秘密分发者以外, 其他参与方都没有输入且行为是确定性的。 $\mathcal{S}$  的构造思路是这样的: 如果理想世界模拟的运行中, 参与方输出  $\perp$ , 那么  $\mathcal{S}$  代表秘密分发者发送一个不合法的多项式 (例如  $q'(x) = x^{2t}$ ) 给理想功能  $\mathcal{F}_{\text{VSS}}$ ; 如果模拟的运行中, 参与方输出了秘密份额, 那么  $\mathcal{S}$  根据其看到的秘密份额插值得到  $S(x, y)$ , 然后代表秘密分发者发送  $q(x) = S(0, x)$  给  $\mathcal{F}_{\text{VSS}}$ .

具体而言, 模拟器  $\mathcal{S}$  的工作方式如下:

1.  $\mathcal{S}$  在内部运行  $\mathcal{A}$ , 转发  $\mathcal{A}$  与环境  $\mathcal{Z}$  的消息。
2.  $\mathcal{S}$  模拟  $n - |I|$  个诚实方遵循协议执行。
3. 令  $\text{num}$  为理想世界的模拟运行中广播了 `consistent` 的 (诚实的和被攻陷的) 参与方数量:
  - (a) 如果  $\text{num} < n - t$ , 那么  $\mathcal{S}$  代表秘密分发者向  $\mathcal{F}_{\text{VSS}}$  发送多项式  $q'(x) = x^{2t}$  作为输入 (这会让  $\mathcal{F}_{\text{VSS}}$  向所有参与方发送  $\perp$  作为输出)。
  - (b) 如果  $\text{num} \geq n - t$ , 那么  $\mathcal{S}$  通过以下方式计算  $t$  阶多项式  $q'(x)$ 。令  $K \subset [n] \setminus I$  是所有在模拟运行中广播了 `consistent` 的诚实方的集合。根据引理 9.9,  $\mathcal{S}$  找到集合  $K$  所唯一确定的  $t$  阶二元多项式  $S(x, y)$ . 然后,  $\mathcal{S}$  设置  $q'(x) = S(0, x)$  并代表秘密分发者将其发送给  $\mathcal{F}_{\text{VSS}}$ . 最后,  $\mathcal{S}$  从  $\mathcal{F}_{\text{VSS}}$  得到被攻陷方的输出  $\{q'(\alpha_i)\}_{i \in I}$ .  $\mathcal{S}$  已经知道这些值, 所以它们不会被使用。(但是  $\mathcal{S}$  需要代表秘密分发者将  $q'(x)$  发送给  $\mathcal{F}_{\text{VSS}}$ , 来让理想世界的诚实方接收到来自  $\mathcal{F}_{\text{VSS}}$  的输出。)

注意到, 协议中除了秘密分发者外, 其他参与方的行为都是确定性的, 并且, 此时诚实方没有输入, 因此在真实世界的运行和模拟运行中, 敌手  $\mathcal{A}$  的视图是完全一致的。我们只需要证明, 在真实世界和理想世界中, 诚实方的输出是一致的。

**断言 1.** 令  $J = [n] \setminus I$  是诚实方的索引。对于任一控制集合  $I$  (包括秘密分发者) 的敌手  $\mathcal{A}$ , 任一多项式  $q(x)$ , 任何辅助输入  $z \in \{0, 1\}^*$ , 都有

$$\text{OUTPUT}\left(\text{REAL}_{\Pi_{\text{VSS}}, \mathcal{A}(z), I}(q(x), \lambda, \dots, \lambda)\right) = \text{OUTPUT}\left(\text{IDEAL}_{\mathcal{F}_{\text{VSS}}, \mathcal{S}(z), I}(q(x), \lambda, \dots, \lambda)\right)$$

证明. 令  $\vec{x} = (q(x), \lambda, \dots, \lambda)$  是输入向量。我们分两种情况讨论: (1) 存在诚实方输出了  $\perp$ ; (2) 没有诚实方输出  $\perp$ .

**情况 1:** 存在  $j \in J$  满足  $\text{OUTPUT}_j(\text{REAL}_{\Pi_{\text{VSS}}, \mathcal{A}(z), I}(q(x), \lambda, \dots, \lambda)) = \perp$ . 根据协议, 诚实方  $P_j$  (在真实世界) 输出  $\perp$  当且仅当小于  $n - t$  个参与方广播了 `consistent`. 因为这些消息是广播的, 参与方都会接收到同样的消息, 因此所有诚实方在真实世界都会输出  $\perp$ .

下面证明，在理想世界中，诚实方也会输出  $\perp$ . 根据模拟器  $\mathcal{S}$  的构造，如果真实世界中小于  $n - t$  个参与方广播了 `consistent`，那么模拟运行中同样也会有小于  $n - t$  个参与方广播 `consistent`. 此时  $\mathcal{S}$  会代表秘密分发者向  $\mathcal{F}_{\text{VSS}}$  发送  $q'(x) = x^{2t}$ ，使理想世界的所有诚实方输出  $\perp$ .

情况 2：对于  $j \in J$  都有  $\text{OUTPUT}_j(\text{REAL}_{\Pi_{\text{VSS}}, \mathcal{A}(z), I}(q(x), \lambda, \dots, \lambda)) \neq \perp$ . 此时，模拟运行中也有至少  $n - t$  个参与方广播 `consistent`. 由  $n \geq 3t + 1$  可知  $n - t \geq 2t + 1$ ，又因为被攻陷方数量不超过  $t$ ，所以至少有  $t + 1$  个诚实方广播了 `consistent`. 我们在“秘密分发者是诚实的”情况中讨论过，诚实方  $P_j$  广播 `consistent` 当且仅当以下条件成立：

1. 秘密分发者解决了所有投诉。（见步骤 5(a)）
2. 秘密分发者没有广播  $\text{reveal}(j, f_j(x), g_j(y))$ . （见步骤 5(b)i）
3.  $\text{reveal}$  消息与  $P_j$  的多项式相一致。（见步骤 5(b)ii）

令  $K \subset [n]$  是模拟器的步骤 3(b) 中广播 `consistent` 的诚实方，对于他们而言，上述条件成立，即  $f_i(\alpha_j) = g_j(\alpha_i)$  对  $i, j \in K$  都成立。根据引理 9.9，存在唯一的  $t$  阶二元多项式  $S(x, y)$  满足  $S(x, \alpha_k) = f_k(x)$  和  $S(\alpha_k, y) = g_k(y)$  对  $k \in K$  都成立。由于  $S(x, y)$  是唯一的，它也唯一确定了  $q'(x) = S(0, x)$ . 在理想世界中，模拟器  $\mathcal{S}$  代表秘密分发者将  $q'(x)$  发送给  $\mathcal{F}_{\text{VSS}}$ ，使得理想世界的诚实方  $P_j$  输出  $q'(\alpha_j)$ .

下面证明，在真实世界中，诚实方  $P_j$  也会输出  $q'(\alpha_j)$ . 首先，对于  $k \in K$ ，显然  $P_k$  会输出  $q'(\alpha_k)$ . 我们只需证明其他没有广播 `consistent` 的诚实方  $P_j$ ,  $j \notin K$  也会输出  $q'(\alpha_j)$ . 令  $f'_j(x)$  和  $g'_j(y)$  是  $P_j$  在执行协议步骤 5(b)i 的替换后持有的多项式（该多项式可能与最初收到的多项式不同）。注意， $P_j$  没有广播 `consistent`，所以上述三个条件不一定成立。但是对于广播了 `consistent` 的参与方  $P_k$ ,  $k \in K$ ，他们的多项式  $f_k(x), g_k(y)$  一定与  $P_j$  的多项式相一致，即  $f_k(\alpha_j) = g'_j(\alpha_k)$ ,  $g_k(\alpha_j) = f'_j(\alpha_k)$ . 这意味着对于至少  $t + 1$  个值  $k \in K$ ，有  $f'_j(\alpha_k) = S(\alpha_k, \alpha_j)$ . 又因为  $f'_j(x)$  是至多  $t$  阶的多项式，可得  $f'_j(x) = S(x, \alpha_j)$ . 由此可知， $P_j$  的输出为  $f'_j(0) = S(0, \alpha_j) = q'(\alpha_j)$ . 证毕。

□

综上所述，模拟器  $\mathcal{S}$  的构造在秘密分发者诚实或被攻陷时，都使真实世界和理想世界不可区分，因此协议  $\Pi_{\text{VSS}}$  对于静态恶意敌手 UC-安全实现了理想功能  $\mathcal{F}_{\text{VSS}}$ . 定理证毕。

□

### 9.3.4 乘法协议

假设乘法门的输入导线值为  $a$  和  $b$ ，参与方  $P_i$  持有  $t$  阶的秘密份额  $a_i$  和  $b_i$ . 前文讲过，乘法协议分为三步：

1. 每个参与方  $P_i$  可验证地秘密分享他们输入导线上的份额  $a_i$  和  $b_i$ .
2. 在持有  $a_i$  和  $b_i$  的子份额的情况下，每个参与方  $P_i$  可验证地秘密分享  $a_i \cdot b_i$ .
3. 将  $a_i \cdot b_i$  的子份额线性重组，得到的份额落在新的  $t$  阶多项式上，其常数项为  $a \cdot b$ .

下面，我们详细介绍每一步的具体实现。我们的协议描述采用模块化的方式：步骤 1 被抽象为理想功能  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$ （见第9.3.4.3节），为实现该理想功能，我们先定义并行可验证秘密分享理想功能  $\mathcal{F}_{\text{VSS}}^n$ （见第9.3.4.1节）以及实现矩阵乘法理想功能  $\mathcal{F}_{\text{mat}}^A$ （见第9.3.4.2节）；步骤 2 被抽象为理想功能  $\mathcal{F}_{\text{VSS}}^{\text{mult}}$ （见第9.3.4.5节），为实现该理想功能，我们需要先实现多项式求值理想功能  $\mathcal{F}_{\text{eval}}$ （见第9.3.4.4节）；最后，基于  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  与  $\mathcal{F}_{\text{VSS}}^{\text{mult}}$ ，再加上步骤 3，我们在第9.3.4.6节实现了乘法理想功能  $\mathcal{F}_{\text{mult}}$ 。

### 9.3.4.1 定义并行可验证秘密分享理想功能 $\mathcal{F}_{\text{VSS}}^n$

在 BGW 协议的输入分享阶段，每个参与方作为秘密分发者运行可验证秘密分享协议，分发自己输入的秘密份额。也就是说，参与方运行  $n$  个 VSS 实例，在第  $i$  个实例中  $P_i$  是秘密分发者并以  $q_i(x)$  作为协议输入。我们需要定义一个理想功能来描述并行地运行 VSS 所满足的安全性。直观上看，它应该保证所有被分享的秘密是互相独立的，我们可能有以下初步的想法：

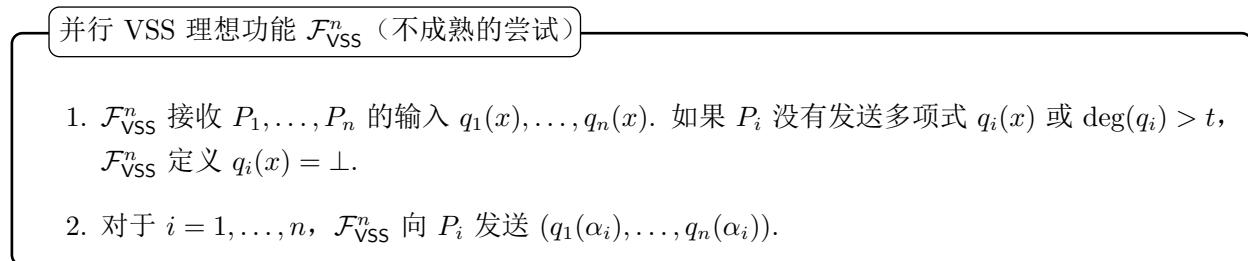


图 9.13: 并行 VSS 理想功能  $\mathcal{F}_{\text{VSS}}^n$  (不成熟的尝试)

直观上看，该理想功能似乎刻画了并行 VSS 所应该满足的安全性质。然而，在实际执行中，由于  $n$  个 VSS 实例是并行运行的，敌手可能采取这样的行为：他首先接收诚实方发送的秘密份额，然后再决定他要分享的秘密。也就是说，敌手可以在看到诚实方发给被攻陷方的秘密份额后，再基于这些秘密份额选择被攻陷方的输入多项式（这类敌手被称为 rushing adversary）。具体而言，令  $P_j$  是一个诚实方，其输入为  $q_j(x)$ ，令  $P_i$  是一个被攻陷方。在实际执行中，敌手可以首先看到  $P_i$  的份额  $q_j(\alpha_i)$ ，然后选择输入多项式  $q_i(x)$ ，满足  $q_i(\alpha_i) = q_j(\alpha_i)$ 。但是在理想世界中，敌手无法这么做，因为  $\mathcal{F}_{\text{VSS}}^n$  的定义使被攻陷方在提供输入多项式之前无法得到关于诚实方多项式的任何信息。

因此，并行地调用  $n$  次  $\mathcal{F}_{\text{VSS}}$  无法安全实现  $\mathcal{F}_{\text{VSS}}^n$ （因为在真实世界中存在某种攻击方式，在理想世界中无法实现），我们需要修改协议或理想功能的定义（或两者都修改）。让我们再次审视敌手的攻击方式，可以发现，这种攻击实际上并不会造成协议的不安全，因为秘密份额  $q_j(\alpha_i)$  与常数项  $q_j(0)$  是相互独立的，敌手让  $q_i(\alpha_i)$  依赖于  $q_j(\alpha_i)$  并不会对整个协议造成实际影响。因此，我们可以让协议保持不变（仍为并行调用  $n$  次  $\mathcal{F}_{\text{VSS}}$ ），而是将理想功能修改如下：令  $I$  是被攻陷方的集合。首先，诚实方发送输入多项式  $q_j(x)$ ,  $j \notin I$ ；然后，模拟器（理想敌手）收到诚实方向被攻陷方发送的秘密份额  $q_j(\alpha_i)$ ,  $j \notin I$ ,  $i \in I$ ；最后，被攻陷方向理想功能发送他们的输入多项式  $q_i(x)$ ,  $i \in I$ 。这样一来，修改后的理想功能就刻画了敌手 (rushing adversary) 可以基于  $q_j(\alpha_i)$  来选择输入多项式的能力。形式化地，平行 VSS 理想功能  $\mathcal{F}_{\text{VSS}}^n$  定义如图 9.14 所示。

并行 VSS 理想功能  $\mathcal{F}_{\text{VSS}}^n$

$\mathcal{F}_{\text{VSS}}^n$  接收被攻陷方的集合  $I \subset [n]$ , 并执行如下操作:

1. 对于每个诚实方  $P_j, j \notin I$ ,  $\mathcal{F}_{\text{VSS}}^n$  接收其输入多项式  $q_j(x)$ .
2. 基于诚实方的输入多项式,  $\mathcal{F}_{\text{VSS}}^n$  向对手  $\mathcal{S}$  发送被攻陷方的秘密份额  $\{q_j(\alpha_i)\}_{j \notin I, i \in I}$ .
3. 对于  $i \in I$ ,  $\mathcal{F}_{\text{VSS}}^n$  接收其输入多项式  $q_i(x)$ .
4. 对于  $j = 1, \dots, n$ ,  $\mathcal{F}_{\text{VSS}}^n$  向  $P_j$  发送  $(q_1(\alpha_j), \dots, q_n(\alpha_j))$ . 如果  $\deg(q_i(x)) > t$ , 那么发送上而不是  $q_i(\alpha_j)$ .

图 9.14: 并行 VSS 理想功能  $\mathcal{F}_{\text{VSS}}^n$

#### 9.3.4.2 实现矩阵乘法理想功能 $\mathcal{F}_{\text{mat}}^A$

接下来, 我们首先实现一个较简单的理想功能——矩阵乘法, 它将输入向量  $\vec{x}$  映射到  $\vec{x} \cdot A$ , 其中  $A$  是一个公开的固定矩阵, 第  $i$  个参与方  $P_i$  持有  $x_i$ , 所有参与方都以  $\vec{x} \cdot A$  作为输出。之后, 我们会把它作为实现理想功能  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  的子协议。

让我们首先讨论一下  $\mathcal{F}_{\text{mat}}^A$  的定义。 $\mathcal{F}_{\text{mat}}^A$  使得参与方安全地计算长度为  $m$  的向量  $(y_1, \dots, y_m) = (x_1, \dots, x_n) \cdot A$ , 其中  $x_1, \dots, x_n \in \mathbb{F}$  是各参与方的输入,  $A \in \mathbb{F}^{n \times m}$ . (这里  $m = 1$  的情况也是有意义的, 但在协议中我们需要取  $m = 2t$ .) 注意到, 矩阵乘法是线性运算 (仅包括加法和常数乘法), 我们可以沿用半诚实场景下的思路: 参与方首先 (可验证地) 秘密分享各自的输入, 然后每个参与方在本地对秘密份额进行线性运算, 最后, 公布计算后的份额来进行输出重建。与半诚实的协议相比, 仅有的一处不同在于——输入分享时使用了可验证秘密分享, 以及输出重建时使用了 Reed-Solomon 解码。因此, 矩阵乘法理想功能可以通过以下方式安全地计算:

1. 输入分享阶段: 每个参与方  $P_i$  选择随机多项式  $g_i(x)$ , 满足条件  $g_i(0) = x_i$ . 然后,  $P_i$  调用理想功能  $\mathcal{F}_{\text{VSS}}$  秘密分享  $g_i(x)$ . 多项式分享结束后, 参与方  $P_i$  持有份额  $g_1(\alpha_i), \dots, g_n(\alpha_i)$ .
2. 计算阶段: 参与方  $P_i$  计算输出向量的秘密份额  $\vec{y}^i = (g_1(\alpha_i), \dots, g_n(\alpha_i)) \cdot A$ . 注意到:

$$\vec{y}^i = (g_1(\alpha_i), \dots, g_n(\alpha_i)) \cdot A = \begin{bmatrix} g_1(\alpha_i), \dots, g_n(\alpha_i) \end{bmatrix} \cdot \begin{bmatrix} a_{1,1} & \dots & a_{1,m} \\ a_{2,1} & \dots & a_{2,m} \\ \vdots & & \vdots \\ a_{n,1} & \dots & a_{n,m} \end{bmatrix}$$

因此  $\vec{y}^i$  的第  $j$  个元素为  $\sum_{\ell=1}^n g_\ell(\alpha_i) \cdot a_{\ell,j}$ . 将  $\vec{y}^i$  的第  $j$  个元素记为  $y_j^i$ , 那么  $y_j^1, \dots, y_j^n$  是  $\vec{y} = (g_1(0), \dots, g_n(0)) \cdot A$  的第  $j$  个元素的 Shamir 秘密份额。

3. 输出重建阶段:

- (a) 每个参与方  $P_i$  将  $\vec{y}^i$  发给其他参与方。

(b) 每个参与方  $P_i$  根据收到的秘密份额, 运行 Reed-Solomon 解码算法重建输出  $\vec{g} = (g_1(0), \dots, g_n(0))$ .

A. 注意到, 参与方的秘密份额构成如下的矩阵:

$$\left[ \begin{array}{ccc|c} \leftarrow & \vec{y}^1 & \rightarrow & \sum_{\ell=1}^n g_\ell(\alpha_1) \cdot a_{\ell,1} \dots \sum_{\ell=1}^n g_\ell(\alpha_1) \cdot a_{\ell,m} \\ \leftarrow & \vec{y}^2 & \rightarrow & \sum_{\ell=1}^n g_\ell(\alpha_2) \cdot a_{\ell,1} \dots \sum_{\ell=1}^n g_\ell(\alpha_2) \cdot a_{\ell,m} \\ \vdots & & & \vdots \\ \leftarrow & \vec{y}^n & \rightarrow & \sum_{\ell=1}^n g_\ell(\alpha_n) \cdot a_{\ell,1} \dots \sum_{\ell=1}^n g_\ell(\alpha_n) \cdot a_{\ell,m} \end{array} \right]$$

矩阵的第  $j$  列构成  $\sum_{\ell=1}^n g_\ell(0) \cdot a_{\ell,j}$  的 Shamir 秘密份额, 而  $\sum_{\ell=1}^n g_\ell(0) \cdot a_{\ell,j}$  就是输出向量的第  $j$  个元素。参与方对矩阵的每一列运行 Reed-Solomon 解码算法得到正确的输出。

直观上看, 上述协议满足对于恶意敌手的安全性:  $\mathcal{F}_{\text{VSS}}$  确保被攻陷方的输入是良好定义的, 而 Reed-Solomon 解码使得被攻陷方无法影响协议输出。然而, 正如前面所提到的, 我们要将  $\mathcal{F}_{\text{mat}}^A$  作为实现  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  的子协议, 因此  $\mathcal{F}_{\text{mat}}^A$  的定义需要更复杂些: 首先, 每个参与方的输入不是  $x_i$ , 而是  $t$  阶多项式  $g_i(x)$  满足  $g_i(0) = x_i$  (这是由于  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  的需要); 其次, 参与方的输出不仅仅是  $\vec{y} = (g_1(0), \dots, g_n(0)) \cdot A$ , 除此之外  $P_i$  还需要输出收到的秘密份额  $g_1(\alpha_i), \dots, g_n(\alpha_i)$ . 于是, 理想功能  $\mathcal{F}_{\text{mat}}^A$  定义如下:

$$\mathcal{F}_{\text{mat}}^A(g_1, \dots, g_n) = \left( (\vec{y}, \{g_\ell(\alpha_1)\}_{\ell=1}^n), (\vec{y}, \{g_\ell(\alpha_2)\}_{\ell=1}^n), \dots, (\vec{y}, \{g_\ell(\alpha_n)\}_{\ell=1}^n) \right)$$

其中  $\vec{y} = (g_1(0), \dots, g_n(0)) \cdot A$ . 尽管只是作了很小的改动, 但这使得协议和理想功能的定义需要变得更复杂。

下面, 我们分析一下为什么上述改动会造成影响。上文的矩阵乘法协议中, 在输出重建阶段, 参与方  $P_i$  要发送输出向量的秘密份额  $\vec{y}^i$ . 注意到, 向量  $\vec{y}^1, \dots, \vec{y}^n$  是完全由输入多项式  $g_1(x), \dots, g_n(x)$  所决定的。但是, 在理想世界中, 模拟器只能获得 (被攻陷方的) 部分秘密份额/输入多项式, 因此不能模拟  $\vec{y}^1, \dots, \vec{y}^n$ . 具体而言, 假设只有  $P_1$  是被攻陷方。此时, 模拟器可以知道  $P_1$  的输出  $\vec{y} = (g_1(0), \dots, g_n(0)) \cdot A$  和  $g_1(\alpha_1), \dots, g_n(\alpha_1)$  以及  $P_1$  的输入多项式  $g_1(x)$ , 但不知道诚实方的输入多项式。然而, 在真实执行中, 敌手可以看到诚实方发送的所有向量  $\vec{y}^2, \dots, \vec{y}^n$ , 它们是基于  $g_1(x), \dots, g_n(x)$  和矩阵  $A$  的一个确定性函数, 因此理想世界的模拟器必须模拟诚实方发送完全一致的消息, 但基于模拟器拥有的信息, 这不可能做到!

让我们审视一下这里遇到的问题: 真实协议中的  $\vec{y}^1, \dots, \vec{y}^n$  泄露了输入多项式的一些信息, 但理想功能并没有这样的泄露。然而,  $\mathcal{F}_{\text{mat}}^A$  的目的是安全地实现  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  (见第9.3.4.3节的图 9.19), 敌手知道  $\vec{y}^1, \dots, \vec{y}^n$  对协议的安全性并没有什么影响。实际上, 我们甚至可以允许敌手知道向量  $\vec{Y}(x) = (Y_1(x), \dots, Y_m(x)) = (g_1(x), \dots, g_n(x)) \cdot A$ . 因此, 我们采取与定义  $\mathcal{F}_{\text{VSS}}^n$  时同样的方式: 修改理想功能, 允许  $\mathcal{F}_{\text{mat}}^A$  泄漏  $\vec{Y}(x)$  给敌手。于是, 矩阵乘法理想功能  $\mathcal{F}_{\text{mat}}^A$  定义如图 9.15所示。注意, 由于  $\mathcal{F}_{\text{mat}}^A$  让  $P_i$  输出了秘密份额  $\{g_\ell(\alpha_i)\}_{\ell=1}^n$ , 与上一节相同, 我们考虑 rushing adversary, 允许敌手基于收到的秘密份额选择输入多项式。

经过修改, 上文中的协议就可以安全地实现理想功能  $\mathcal{F}_{\text{mat}}^A$  了。我们给出协议在  $\mathcal{F}_{\text{VSS}}^n$ -混合模型下的形式化描述, 如图 9.16所示。

图 9.17展示了协议的第 5 步。 $P_i$  从每个参与方处收到一个向量, 这些向量按行构成一个矩阵, 矩阵的每一列与正确的码字的距离最多为  $t$ .

**定理 9.16.** 假设参与方之间存在点对点安全信道。假设被攻陷方的数量不超过  $t$ ,  $t < n/3$ . 图 9.16中的协议  $\Pi_{\text{mat}}^A$  在  $\mathcal{F}_{\text{VSS}}^n$ -混合模型下对于静态恶意敌手 UC-安全实现了理想功能  $\mathcal{F}_{\text{mat}}^A$  (图 9.15)。

矩阵乘法理想功能  $\mathcal{F}_{\text{mat}}^A$  (参数  $A \in \mathbb{F}^{n \times m}$ )

$\mathcal{F}_{\text{mat}}^A$  接收被攻陷方的集合  $I \subset [n]$ , 并执行如下操作:

1. 对于每个诚实方  $P_j$ ,  $j \notin I$ ,  $\mathcal{F}_{\text{mat}}^A$  接收其输入多项式  $g_j(x)$ . 如果  $\deg(g_j(x)) > t$ , 设置  $g_j(x) = 0$ .
2.  $\mathcal{F}_{\text{mat}}^A$  向敌手  $\mathcal{S}$  发送被攻陷方的秘密份额  $\{g_j(\alpha_i)\}_{j \notin I, i \in I}$ .
3. 对于  $i \in I$ ,  $\mathcal{F}_{\text{mat}}^A$  接收其输入多项式  $g_i(x)$ . 如果  $\deg(g_i(x)) > t$ , 设置  $g_i(x) = 0$ .
4.  $\mathcal{F}_{\text{mat}}^A$  计算  $\vec{Y}(x) = (Y_1(x), \dots, Y_m(x)) = (g_1(x), \dots, g_n(x)) \cdot A$ .
5. (a)  $\mathcal{F}_{\text{mat}}^A$  向敌手  $\mathcal{S}$  发送  $\vec{Y}(x)$ .
- (b) 对于  $j = 1, \dots, n$ ,  $\mathcal{F}_{\text{mat}}^A$  向  $P_j$  发送输出  $\vec{y} = \vec{Y}(0)$  和  $(g_1(\alpha_j), \dots, g_n(\alpha_j))$ .

图 9.15: 矩阵乘法理想功能  $\mathcal{F}_{\text{mat}}^A$ 

$\mathcal{F}_{\text{VSS}}^n$ -混合模型下的  $\Pi_{\text{mat}}^A$  协议

**输入:**  $P_i$  的输入为多项式  $g_i(x)$ .

**公共输入:** 域  $\mathbb{F}$ ,  $n$  个互不相同的非 0 元素  $\alpha_1, \dots, \alpha_n \in \mathbb{F}$ , 以及矩阵  $A \in \mathbb{F}^{n \times m}$ .

**协议:**

1. 每个参与方  $P_i$  检查自己的输入多项式, 如果大于  $t$  阶, 则设置  $g_i(x) = 0$ . 然后将  $g_i(x)$  作为输入发送给理想功能  $\mathcal{F}_{\text{VSS}}^n$ .
2. 每个参与方  $P_i$  从  $\mathcal{F}_{\text{VSS}}^n$  处接收到  $g_1(\alpha_i), \dots, g_n(\alpha_i)$ . 如果有的值为  $\perp$ , 将其替换为 0.
3. 令  $\vec{x}^i = (g_1(\alpha_i), \dots, g_n(\alpha_i))$ . 每个参与方  $P_i$  在本地计算  $\vec{y}^i = \vec{x}^i \cdot A$ . 等价地说, 对于  $k = 1, \dots, m$ ,  $P_i$  计算  $Y_k(\alpha_i) = \sum_{\ell=1}^n g_\ell(\alpha_i) \cdot a_{\ell,k}$ , 其中  $(a_{1,k}, \dots, a_{n,k})^T$  是矩阵  $A$  的第  $k$  列, 并保存  $\vec{y}^i = (Y_1(\alpha_i), \dots, Y_m(\alpha_i))$ .
4. 每个参与方  $P_i$  向所有参与方  $P_j$ ,  $1 \leq j \leq n$  发送  $\vec{y}^i$ .
5. 对于  $j = 1, \dots, n$ , 将  $P_i$  从  $P_j$  处收到的向量记为  $\hat{\vec{Y}} = (\hat{Y}_1(\alpha_j), \dots, \hat{Y}_m(\alpha_j))$ . 如果某些值缺失, 则将其替换为 0. 每个参与方执行如下操作:
  - 对于  $k = 1, \dots, m$ ,  $P_i$  以(可能损坏的)码字  $(\hat{Y}_k(\alpha_1), \dots, \hat{Y}_k(\alpha_n))$  为输入, 运行 Reed-Solomon 解码算法(参数  $d = 2t + 1$ ), 得到码字  $(Y_k(\alpha_1), \dots, Y_k(\alpha_n))$ , 如图 9.17 所示. 然后重建多项式  $Y_k(x)$  并计算  $y_k = Y_k(0)$ .

**输出:**  $P_i$  输出  $(y_1, \dots, y_m)$  以及秘密份额  $g_1(\alpha_i), \dots, g_n(\alpha_i)$ .

图 9.16:  $\mathcal{F}_{\text{VSS}}^n$ -混合模型下的  $\Pi_{\text{mat}}^A$  协议

$$\begin{bmatrix} \leftarrow & \hat{\vec{Y}}^1 & \rightarrow \\ \leftarrow & \hat{\vec{Y}}^2 & \rightarrow \\ \vdots & & \\ \leftarrow & \hat{\vec{Y}}^n & \rightarrow \end{bmatrix} = \begin{bmatrix} \hat{Y}_1(\alpha_1) & \dots & \hat{Y}_k(\alpha_1) & \dots & \hat{Y}_m(\alpha_1) \\ \hat{Y}_1(\alpha_2) & \dots & \hat{Y}_k(\alpha_2) & \dots & \hat{Y}_m(\alpha_2) \\ \vdots & & \vdots & & \vdots \\ \hat{Y}_1(\alpha_n) & \dots & \hat{Y}_k(\alpha_n) & \dots & \hat{Y}_m(\alpha_n) \end{bmatrix}$$

图 9.17:  $P_i$  收到的向量构成的矩阵, 对每一列执行 Reed-Solomon 解码

证明. 要证明此定理, 我们需要构造模拟器  $\mathcal{S}$ , 使得对所有环境  $\mathcal{Z}$ , 都有

$$\text{EXEC}_{\Pi_{\text{mat}}^A, \mathcal{A}, \mathcal{Z}}^{\mathcal{F}_{\text{VSS}}^n} \approx \text{EXEC}_{\mathcal{F}_{\text{mat}}^A, \mathcal{S}, \mathcal{Z}}^{\mathcal{F}_{\text{VSS}}^n}$$

其中  $\mathcal{A}$  是平凡敌手。

**模拟器  $\mathcal{S}$ .** 模拟器  $\mathcal{S}$  在内部运行  $\mathcal{A}$ , 转发  $\mathcal{Z}$  与  $\mathcal{A}$  之间的消息。在  $\mathcal{F}_{\text{VSS}}^n$ -混合模型下,  $\mathcal{S}$  还需要模拟  $\mathcal{F}_{\text{VSS}}^n$ .  $\mathcal{S}$  的工作方式如下。

1. 当诚实方向  $\mathcal{F}_{\text{mat}}^A$  发送输入后, 模拟器  $\mathcal{S}$  记录来自  $\mathcal{F}_{\text{mat}}^A$  的消息  $\{g_j(\alpha_i)\}_{j \notin I, i \in I}$  ( $\mathcal{F}_{\text{mat}}^A$  的步骤 2)。
2. 模拟  $\Pi_{\text{mat}}^A$  的步骤 1 和 2:  $\mathcal{S}$  按照如下方式模拟  $\mathcal{F}_{\text{VSS}}^n$ :
  - (a)  $\mathcal{S}$  模拟  $\mathcal{F}_{\text{VSS}}^n$  的步骤 2, 向敌手  $\mathcal{A}$  发送  $\{g_j(\alpha_i)\}_{j \notin I, i \in I}$ .
  - (b)  $\mathcal{S}$  模拟  $\mathcal{F}_{\text{VSS}}^n$  的步骤 3, 接收并记录被攻陷方的输入多项式  $\{g_i(x)\}_{i \in I}$ .
  - (c)  $\mathcal{S}$  模拟  $\mathcal{F}_{\text{VSS}}^n$  的步骤 4, 向每个被攻陷方  $P_i$ ,  $i \in I$  发送  $(g_1(\alpha_i), \dots, g_n(\alpha_i))$ . 如果  $\deg(g_k(x)) > t$ , 那么发送  $\perp$  而不是  $g_k(\alpha_i)$ .
3.  $\mathcal{S}$  代表被攻陷方向  $\mathcal{F}_{\text{mat}}^A$  发送输入多项式  $\{g_i(x)\}_{i \in I}$  ( $\mathcal{F}_{\text{mat}}^A$  的步骤 3)。
4.  $\mathcal{S}$  接收并记录来自  $\mathcal{F}_{\text{mat}}^A$  的消息: ( $\mathcal{F}_{\text{mat}}^A$  的步骤 5)
  - (a) 多项式向量  $\vec{Y}(x) = (g_1(x), \dots, g_n(x)) \cdot A$ ;
  - (b) 输出向量  $\vec{y} = (y_1, \dots, y_m)$ ;
  - (c) 被攻陷方  $P_i$ ,  $i \in I$  收到的秘密份额  $\{(g_1(\alpha_i), \dots, g_n(\alpha_i))\}_{i \in I}$ .
5. 模拟  $\Pi_{\text{mat}}^A$  的步骤 4: 对于  $j \notin I, i \in I$ ,  $\mathcal{S}$  模拟  $P_j$  向  $P_i$  发送  $\vec{y}^j = (Y_1(\alpha_j), \dots, Y_m(\alpha_j))$ .

**不可区分性.** 下面证明  $\mathcal{Z}$  在真实世界和理想世界中的视图是不可区分的。首先, 在  $\mathcal{F}_{\text{VSS}}^n$ -混合模型下, 观察到: 诚实方、模拟器  $\mathcal{S}$ 、理想功能  $\mathcal{F}_{\text{mat}}^A$  的行为都是确定性的 (因为随机性来源于  $\mathcal{F}_{\text{VSS}}^n$ )。因此, 我们证明敌手在理想世界和真实世界中的视图是相同的。由于模拟器  $\mathcal{S}$  从  $\mathcal{F}_{\text{mat}}^A$  收到了  $\{g_j(\alpha_i)\}_{j \notin I, i \in I}$  和  $\vec{Y}(x)$ , 这使得  $\mathcal{S}$  能够提供与真实执行中一样的消息。

接下来, 我们只需证明在真实世界和理想世界中诚实方的输出是一致的, 即  $\mathcal{F}_{\text{mat}}^A$  在步骤 4 中计算的  $\vec{Y}(x) = (Y_1(x), \dots, Y_m(x))$  与协议第 5 步计算的  $(Y_1(x), \dots, Y_m(x))$  是一样的。注意到, 每个多项式  $Y_k(x)$  是基于诚实方运行 Reed-Solomon 解码算法得到的码字来重建的, 我们只需证明解码算法会得到正确的码字。这是因为诚实方都会提供正确的  $\hat{Y}_k(\alpha_i)$ , 即  $\hat{Y}_k(\alpha_j) = Y_k(\alpha_j)$  对  $j \notin I$  都成立, 故码字  $(\hat{Y}_1(\alpha_1), \dots, \hat{Y}_k(\alpha_n))$  至少有  $n - t$  个元素是正确的。根据定理 9.7, 解码算法将输出正确的码字, 重建  $Y_k(x)$ . 证毕。  $\square$

### 9.3.4.3 实现子份额秘密分享理想功能 $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$

定义  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$ . 非形式化地,  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  允许参与方可验证地秘密分享一些值, 这些值本身就是秘密份额。具体而言, 假设参与方  $P_1, \dots, P_n$  分别持有  $f(\alpha_1), \dots, f(\alpha_n)$ , 其中  $f$  是  $t$  阶多项式, 然后, 每个参与方  $P_i$  要向其他参与方秘密分享  $f(\alpha_i)$ , 而不能是其他值, 如图 9.18 所示。

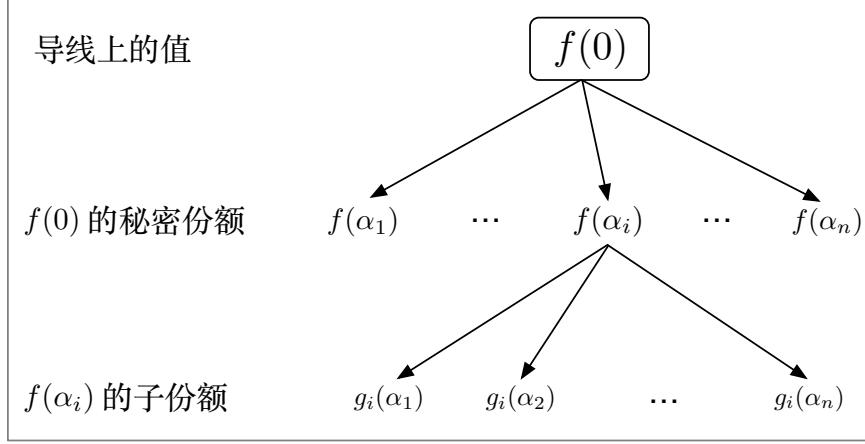


图 9.18: 子份额秘密分享:  $P_i$  分发  $f(\alpha_i)$  的子份额

在半诚实场景下, 这很容易做到: 每个参与方  $P_i$  选择常数项为  $f(\alpha_i)$  的随机多项式  $g_i(x)$ , 然后向  $P_j$  发送  $g_i(\alpha_j)$ . 但是, 在恶意敌手场景下, 主要的难点在于: 如何强制被攻陷方  $P_i$  分享正确的  $f(\alpha_i)$  而不是其他值。注意, 此时有超过  $t$  个诚实方, 它们的秘密份额唯一确定了  $f(x)$ , 因此可以令  $f(x)$  为诚实方的秘密份额所确定的  $t$  阶多项式, 它是良好定义的。我们要确保被攻陷方  $P_i$  在分发子份额时使用了常数项为  $f(\alpha_i)$  的  $t$  阶多项式。

我们先给出  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  的定义。当某个被攻陷方  $P_i$  向  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  提供了不合法的输入时 ( $g_i(x)$  超过  $t$  阶或  $g_i(0) \neq f(\alpha_i)$ ),  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  将其替换为常数多项式  $g'_i(x) = f(\alpha_i)$ . 这确保了  $P_i$  的输入多项式的常数项总是  $f(\alpha_i)$ . 另外, 由于我们在实现  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  的协议中调用了理想功能  $\mathcal{F}_{\text{mat}}^A$  来乘以 Reed-Solomon 码校验矩阵 (parity-check matrix)  $H$  (的转置), 因此  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  需要向模拟器  $\mathcal{S}$  泄漏  $\vec{Y}(x) = (g_1(x), \dots, g_n(x)) \cdot H^T$  使其能够模拟  $\mathcal{F}_{\text{mat}}^A$ . 最后, 与前面相同, 我们考虑 rushing adversary, 允许敌手基于诚实方向被攻陷方发送的秘密份额来选择输入多项式。 $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  的定义如图 9.19 所示。

**Reed-Solomon 编码背景知识。**令  $G \in \mathbb{F}^{(t+1) \times n}$  是长度  $n = 3t + 1$ , 维度  $k = t + 1$ , 距离  $d = 2t + 1$  的 Reed-Solomon 码的生成矩阵 (generator matrix)。对向量  $\vec{a} = (a_0, \dots, a_t) \in \mathbb{F}^{t+1}$  编码将输出  $\vec{a} \cdot G$ , 其中

$$G = \begin{pmatrix} 1 & 1 & \dots & 1 \\ \alpha_1 & \alpha_2 & \dots & \alpha_n \\ \vdots & \vdots & & \vdots \\ \alpha_1^t & \alpha_2^t & \dots & \alpha_n^t \end{pmatrix}$$

令  $f(x) = \sum_{\ell=0}^t a_\ell \cdot x^\ell$  是一个  $t$  阶多项式, 对向量  $\vec{a} = (a_0, \dots, a_t) \in \mathbb{F}^{t+1}$  的 Reed-Solomon 编码即为向量  $\langle f(\alpha_1), \dots, f(\alpha_n) \rangle$ . 令  $H \in \mathbb{F}^{2t \times n}$  是  $G$  的校验矩阵, 即  $H$  的秩 (rank) 为  $2t$ , 满足  $G \times H^T = 0^{(t+1) \times 2t}$ . 注意,  $H$  完全由  $\alpha_1, \dots, \alpha_n$  所决定, 因此是一个公开的常量矩阵。码字  $\vec{\beta} \in \mathbb{F}^n$  的伴随式 (syndrome)

子份额秘密分享理想功能  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$

$\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  接收被攻陷方的集合  $I \subset [n]$ , 并执行如下操作:

1. 对于每个诚实方  $P_j$ ,  $j \notin I$ ,  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  接收其输入  $\beta_j$ . 令  $f(x)$  是由点集  $\{(\alpha_j, \beta_j)\}_{j \notin I}$  所唯一确定的  $t$  阶多项式。
2. 对于  $j \notin I$ ,  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  均匀随机选择  $t$  阶多项式  $g_j(x)$ , 满足条件  $g_j(0) = \beta_j = f(\alpha_j)$ .
3.  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  向敌手  $\mathcal{S}$  发送被攻陷方的秘密份额  $\{g_j(\alpha_i)\}_{j \notin I, i \in I}$ .
4.  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  从（理想）敌手处接收被攻陷方的输入多项式  $\{g_i(x)\}_{i \in I}$ . 如果没有收到  $g_i(x)$  或  $\deg(g_i(x)) > t$ , 设置  $g_i(x) = 0$ .
5.  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  按照如下方式确定输出多项式  $g'_1(x), \dots, g'_n(x)$ :
  - (a) 对于  $j \notin I$ , 设置  $g'_j(x) = g_j(x)$ .
  - (b) 对于  $i \in I$ , 如果  $g_i(0) = f(\alpha_i)$  则设置  $g'_i(x) = g_i(x)$ , 否则设置  $g'_i(x) = f(\alpha_i)$ .
6. (a)  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  向敌手  $\mathcal{S}$  发送  $\vec{Y}(x) = (g_1(x), \dots, g_n(x)) \cdot H^T$ , 其中  $H$  是 Reed-Solomon 码的校验矩阵（见下文 Reed-Solomon 编码背景知识）。
- (b) 对于  $j = 1, \dots, n$ ,  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  向  $P_j$  发送输出  $g'_j(x)$  和  $(g'_1(\alpha_j), \dots, g'_n(\alpha_j))$ .

图 9.19: 子份额秘密分享理想功能  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$

为  $S(\vec{\beta}) = \vec{\beta} \cdot H^T \in \mathbb{F}^{2t}$ . 对于一个正确的码字  $\vec{\beta} = \vec{a} \cdot G$ , 一定有  $S(\vec{\beta}) = 0^{2t}$ ; 对于任意错误向量  $\vec{e} \in \mathbb{F}^n$ , 有  $S(\vec{\beta} + \vec{e}) = S(\vec{e})$ . 如果  $\vec{e}$  与  $0^n$  的距离不超过  $t$ , 那么我们可以对  $\vec{\beta} + \vec{e}$  进行纠错得到原始向量  $\vec{\beta}$ . 在 Reed-Solomon 解码算法中, 存在一个子程序可以从伴随式  $S(\vec{e})$  中提取  $\vec{e}$ . 也就是说, 给定一个可能损坏的码字  $\vec{\gamma} = \vec{\beta} + \vec{e}$ , 首先计算其伴随式  $S(\vec{\gamma}) = \vec{\gamma} \cdot H^T = S(\vec{e})$ , 然后将  $S(\vec{e})$  输入子程序提取  $\vec{e}$ . 最后, 根据  $\vec{e}$  和  $\vec{\gamma}$  可以提取原始向量  $\vec{\beta} = \vec{\gamma} - \vec{e}$ .

**协议构造.** 协议中, 每个参与方  $P_i$  选择随机多项式  $g_i(x)$ , 其常数项为协议输入值 (秘密份额)  $\beta_i$ . 令  $\vec{\beta} = (\beta_1, \dots, \beta_n)$ . 因为这些输入值位于某个多项式  $f(x)$  上, 所以对于所有诚实方  $P_j$  都有  $g_j(0) = \beta_j = f(\alpha_j)$ . 但是对于被攻陷方  $P_i$ , 他的多项式的常数项  $g_i(0)$  没有任何保证. 令  $\vec{\gamma} = (g_1(0), \dots, g_n(0))$ , 那么  $\vec{\gamma}$  与  $\vec{\beta} = (\beta_1, \dots, \beta_n)$  的距离最多为  $t$ . 而  $\vec{\beta}$  的长度为  $n = 3t + 1$ , 因此可以通过 Reed-Solomon 纠错算法来给  $\vec{\gamma}$  纠错. 参与方首先将多项式  $(g_1(x), \dots, g_n(x))$  发给  $\mathcal{F}_{\text{mat}}^H$  (它是图 9.15 定义的理想功能, 将参数矩阵换为校验矩阵  $H$  的转置), 然后每个参与方  $P_i$  得到输出  $(g_1(\alpha_i), \dots, g_n(\alpha_i))$  和  $(s_1, \dots, s_{2t}) = \vec{\gamma} \cdot H^T$ . 这里  $(s_1, \dots, s_{2t}) = \vec{\gamma} \cdot H^T$  就是  $\vec{\gamma}$  的伴随式  $S(\vec{\gamma})$ . 每个参与方基于该伴随式提取错误向量  $\vec{e} = (e_1, \dots, e_n) = \vec{\gamma} - \vec{\beta}$ . 注意,  $\vec{e}$  满足  $g_i(0) - e_i = f(\alpha_i)$  对  $i \in [n]$  都成立, 且  $\vec{e}$  可以由参与方在本地基于  $S(\vec{\gamma})$  提取出来. 于是, 错误向量  $\vec{e}$  为诚实方提供了计算输出所需要的信息. 具体而言, 如果  $e_i = 0$ , 那么  $P_i$  使用了正确的多项式  $g_i(x)$ , 参与方可以直接输出  $\mathcal{F}_{\text{mat}}^H$  的输出  $g_i(\alpha_j)$ ; 如果  $e_i \neq 0$ , 那么参与方就知道  $P_i$  是被攻陷的, 他们互相发送  $P_i$  的秘密份额  $g_i(\alpha_j)$ , 重建  $g_i(x)$  (通过 Reed-Solomon 解码), 然后计算  $g_i(0) - e_i = f(\alpha_i)$ . 最后, 参与方设置  $g'_i(x) = f(\alpha_i)$ , 与理想功能  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  相一致. 完整的协议如图 9.20 所示.

在进行安全性证明时, 有一点需要注意: 协议中泄漏了伴随式  $\vec{\gamma} \cdot H^T$ , 模拟器  $\mathcal{S}$  需要在理想世界中模拟该消息. 不过,  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  向模拟器  $\mathcal{S}$  泄露了  $\vec{Y}(x) = (g_1(x), \dots, g_n(x)) \cdot H^T$ , 而伴随式就等于  $\vec{Y}(0)$ , 因此  $\mathcal{S}$  可以模拟该条消息.

**定理 9.17.** 假设参与方之间存在点对点安全信道. 假设被攻陷方的数量不超过  $t$ ,  $t < n/3$ . 图 9.20 中的协议  $\Pi_{\text{VSS}}^{\text{subshare}}$  在  $\mathcal{F}_{\text{mat}}^H$ -混合模型下对于静态恶意敌手 UC-安全实现了理想功能  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  (图 9.19).

**证明.** 要证明此定理, 我们需要构造模拟器  $\mathcal{S}$ , 使得对所有环境  $\mathcal{Z}$ , 都有

$$\text{EXEC}_{\Pi_{\text{VSS}}^{\text{subshare}}, \mathcal{A}, \mathcal{Z}}^{\mathcal{F}_{\text{mat}}^H} \approx \text{EXEC}_{\mathcal{F}_{\text{VSS}}^{\text{subshare}}, \mathcal{S}, \mathcal{Z}}^{\mathcal{F}_{\text{VSS}}^{\text{subshare}}}$$

其中  $\mathcal{A}$  是平凡敌手.

**模拟器  $\mathcal{S}$ .** 模拟器  $\mathcal{S}$  在内部运行  $\mathcal{A}$ , 转发  $\mathcal{Z}$  与  $\mathcal{A}$  之间的消息. 在  $\mathcal{F}_{\text{mat}}^H$ -混合模型下,  $\mathcal{S}$  还需要模拟  $\mathcal{F}_{\text{mat}}^H$ .  $\mathcal{S}$  的工作方式如下.

1. 当诚实方向  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  发送输入后, 模拟器  $\mathcal{S}$  记录来自  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  的消息  $\{g_j(\alpha_i)\}_{j \notin I, i \in I}$  ( $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  的步骤 3).
2. 模拟  $\Pi_{\text{VSS}}^{\text{subshare}}$  的步骤 2:  $\mathcal{S}$  按照如下方式模拟  $\mathcal{F}_{\text{mat}}^H$ :
  - (a)  $\mathcal{S}$  模拟  $\mathcal{F}_{\text{mat}}^H$  的步骤 2, 向敌手  $\mathcal{A}$  发送  $\{g_j(\alpha_i)\}_{j \notin I, i \in I}$ .
  - (b)  $\mathcal{S}$  模拟  $\mathcal{F}_{\text{mat}}^H$  的步骤 3, 接收并记录被攻陷方的输入多项式  $\{g_i(x)\}_{i \in I}$ .
3.  $\mathcal{S}$  向  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  发送被攻陷方的输入多项式  $\{g_i(x)\}_{i \in I}$  ( $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  的步骤 4). 在接下来的运行中, 如果  $\deg(g_i(x)) > t$ ,  $\mathcal{S}$  设置  $g_i(x) = 0$ .

### $\mathcal{F}_{\text{mat}}^H$ -混合模型下的 $\Pi_{\text{VSS}}^{\text{subshare}}$ 协议

**输入:**  $P_i$  的输入为  $\beta_i$ , 诚实方的点集  $(\alpha_j, \beta_j)$  位于  $t$  阶多项式上。

**公共输入:** 域  $\mathbb{F}$ ,  $n$  个互不相同的非 0 元素  $\alpha_1, \dots, \alpha_n \in \mathbb{F}$ , 它们定义了 Reed-Solomon 码的校验矩阵  $H \in \mathbb{F}^{2t \times n}$  (见上文 Reed-Solomon 编码背景知识)。

**协议:**

1. 每个参与方  $P_i$  均匀随机选择  $t$  阶多项式  $g_i(x)$ , 满足条件  $g_i(0) = \beta_i$ .
2. 每个参与方  $P_i$  将  $g_i(x)$  作为输入发送给  $\mathcal{F}_{\text{mat}}^H$ , 然后从  $\mathcal{F}_{\text{mat}}^H$  处接收到  $g_1(\alpha_i), \dots, g_n(\alpha_i)$  以及  $\vec{s} = (s_1, \dots, s_{2t}) = (g_1(0), \dots, g_n(0)) \cdot H^T$ . 注意, 这里的  $\vec{s}$  是 (可能损坏的) 码字  $\vec{\gamma} = (g_1(0), \dots, g_n(0))$  的伴随式。
3. 每个参与方在本地以  $\vec{s}$  为输入运行 Reed-Solomon 解码算法, 获得错误向量  $\vec{e} = (e_1, \dots, e_n)$ .
4. 对于  $k$  满足  $e_k = 0$ ,  $P_i$  设置  $g'_k(\alpha_i) = g_k(\alpha_i)$ .
5. 对于  $k$  满足  $e_k \neq 0$ :
  - (a)  $P_i$  向所有其他参与方发送  $g_k(\alpha_i)$ .
  - (b)  $P_i$  接收到  $g_k(\alpha_1), \dots, g_k(\alpha_n)$ , 如果某些值缺失, 将其设置为 0.  $P_i$  运行 Reed-Solomon 解码算法重建  $g_k(x)$ .
  - (c)  $P_i$  计算  $g_k(0)$ , 然后设置  $g'_k(\alpha_i) = g_k(0) - e_k$ .

**输出:**  $P_i$  输出  $g_i(x)$  和  $g'_1(\alpha_i), \dots, g'_n(\alpha_i)$ .

图 9.20:  $\mathcal{F}_{\text{mat}}^H$ -混合模型下的  $\Pi_{\text{VSS}}^{\text{subshare}}$  协议

4.  $\mathcal{S}$  接收并记录来自  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  的消息，包括多项式向量  $\vec{Y}(x) = (g_1(x), \dots, g_n(x)) \cdot H^T$  和被攻陷方的输出： $\{g'_i(x)\}_{i \in I}$  和  $\{g'_1(\alpha_i), \dots, g'_n(\alpha_i)\}_{i \in I}$ . ( $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  的步骤 6)。
5. 继续模拟  $\Pi_{\text{VSS}}^{\text{subshare}}$  的步骤 2：
  - (a)  $\mathcal{S}$  模拟  $\mathcal{F}_{\text{mat}}^H$  的步骤 5(a)：向敌手  $\mathcal{A}$  发送  $\vec{Y}(x) = (g_1(x), \dots, g_n(x)) \cdot H^T$ .
  - (b)  $\mathcal{S}$  模拟  $\mathcal{F}_{\text{mat}}^H$  的步骤 5(b)： $\mathcal{S}$  计算  $\vec{y} = \vec{Y}(0)$ . 此时， $\mathcal{S}$  已经从  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  获得了秘密份额  $\{g_j(\alpha_i)\}_{j \notin I, i \in I}$ ,  $\mathcal{S}$  根据  $\{g_i(x)\}_{i \in I}$  计算其余的秘密份额，然后模拟  $\mathcal{F}_{\text{mat}}^H$  向  $P_i, i \in I$  发送  $\vec{y}$  和  $(g_1(\alpha_i), \dots, g_n(\alpha_i))$ .
6. 模拟  $\Pi_{\text{VSS}}^{\text{subshare}}$  的步骤 5(a)：令  $\vec{s} = \vec{Y}(0)$ .  $\mathcal{S}$  运行 Reed-Solomon 解码算法从伴随式  $\vec{s}$  中提取错误向量  $\vec{e} = (e_1, \dots, e_n)$ . 对于所有  $i$  满足  $e_i \neq 0$ ,  $\mathcal{S}$  模拟  $P_j, j \notin I$  发送  $g_i(\alpha_j)$  给所有其他参与方。

**不可区分性。**下面证明  $\mathcal{Z}$  在真实世界和理想世界中的视图是不可区分的。我们先观察模拟器  $\mathcal{S}$  的模拟方式。首先， $\mathcal{S}$  需要模拟  $\mathcal{F}_{\text{mat}}^H$ . 基于  $\mathcal{S}$  从  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  处收到的诚实方的秘密份额  $\{g_j(\alpha_i)\}_{j \notin I, i \in I}$  和  $\vec{Y}(x) = (g_1(x), \dots, g_n(x)) \cdot H^T$ ,  $\mathcal{S}$  可以模拟出  $\mathcal{F}_{\text{mat}}^H$  向敌手和被攻陷方发送的消息。此外，在协议  $\Pi_{\text{VSS}}^{\text{subshare}}$  的步骤 5(a) 中， $\mathcal{S}$  需要模拟诚实方向被攻陷方发送消息，由于伴随式  $\vec{s} = \vec{Y}(0)$ , 故  $\mathcal{S}$  可以提取错误向量  $\vec{e}$  从而进行模拟。

也就是说，真实世界与理想世界的唯一不同在于诚实方多项式的选择——理想世界中，这些多项式由理想功能  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  选择；真实世界中，它们由诚实方自己选择。无论在真实世界还是理想世界，诚实方  $P_j$  的多项式  $g_j(x)$  都是在  $g_j(0) = \beta_j$  的条件下均匀随机选择的，它们分布完全相同。而敌手在协议中能做的事只有选择被攻陷方的多项式并发送给  $\mathcal{F}_{\text{mat}}^H$ . 我们发现，给定诚实方的多项式，敌手的视图与诚实方的输出是关于被攻陷方的多项式的确定性函数。于是，我们只需要证明，给定诚实方的多项式  $\{g_j(x)\}_{j \notin I}$ ,  $\mathcal{Z}$  在真实世界和理想世界中的视图是完全相同的（因为理想世界和真实世界中，诚实方的多项式有相同的分布）。

首先，根据  $\mathcal{S}$  模拟  $\mathcal{F}_{\text{mat}}^H$  的方式， $\mathcal{A}$  在两个世界中收到来自  $\mathcal{F}_{\text{mat}}^H$  的消息是相同的（ $\mathcal{S}$  仅仅是转发）。其次， $\mathcal{F}_{\text{mat}}^H$  确保了所有诚实方得到同样的伴随式  $\vec{s}$ , 提取出同样的错误向量  $\vec{e} = (e_1, \dots, e_n)$ , 因此敌手  $\mathcal{A}$  在协议步骤 5(a) 看到的视图也是相同的。最后，考虑诚实方的输出。在理想世界中，每个诚实方  $P_j$  的输出是由  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  计算的  $g'_j(x)$  和  $(g'_1(\alpha_j), \dots, g'_n(\alpha_j))$ . 在真实世界中，每个诚实方  $P_j$  的输出是自己选择的多项式  $g_j(x)$  和纠错后的秘密份额  $(g'_1(\alpha_j), \dots, g'_n(\alpha_j))$ . 显然，根据  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  的计算方式，对于诚实方  $P_j$ , 有  $g'_j(x) = g_j(x)$ . 令  $f(x)$  是诚实方的输入份额唯一确定的多项式。令  $K \subset I$  是所有满足  $e_k \neq 0$  的索引的集合。根据 Reed-Solomon 解码算法的基本性质，对于  $k \in K$ , 有  $g_k(0) \neq f(\alpha_k)$ , 因此，根据  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  的计算方式，会设置  $g'_k(x) = f(\alpha_k)$ ; 在真实世界中，同样地， $P_i$  也设置  $g'_i(\alpha_i) = g_i(0) - e_i = f(\alpha_i)$ . 故真实世界和理想世界中，诚实方会输出同样的秘密份额。

证毕。  $\square$

#### 9.3.4.4 实现多项式求值理想功能 $\mathcal{F}_{\text{eval}}$

在第9.3.4.5节的协议（实现  $\mathcal{F}_{\text{VSS}}^{\text{mult}}$ ）中，参与方需要解决“投诉”(complaint)（因为被攻陷方可能提供错误的值）。解决这些投诉的方式是：计算多项式在投诉方的点处的值。具体而言，给定多项式  $f(x)$  的份额  $f(\alpha_1), \dots, f(\alpha_n)$ , 参与方需要计算  $f(\alpha_k)$ , 其中  $k$  是一个公开值，且不泄漏任何其他信息。

我们首先定义理想功能  $\mathcal{F}_{\text{eval}}^k$ 。它的参数是索引  $k$ ,  $k$  指定了需要求多项式在哪个点处的值。 $\mathcal{F}_{\text{eval}}^k$  的形式化定义如图 9.21 所示。在该定义中,  $\mathcal{F}_{\text{eval}}^k$  只接收了诚实方的输入, 忽略了被攻陷方的输入。我们也可以让  $\mathcal{F}_{\text{eval}}^k$  接收所有参与方的输入, 然后运行 Reed-Solomon 解码算法。但实际上敌手的输入不会对输出有任何影响, 因此我们采用这种更简洁的定义方式。

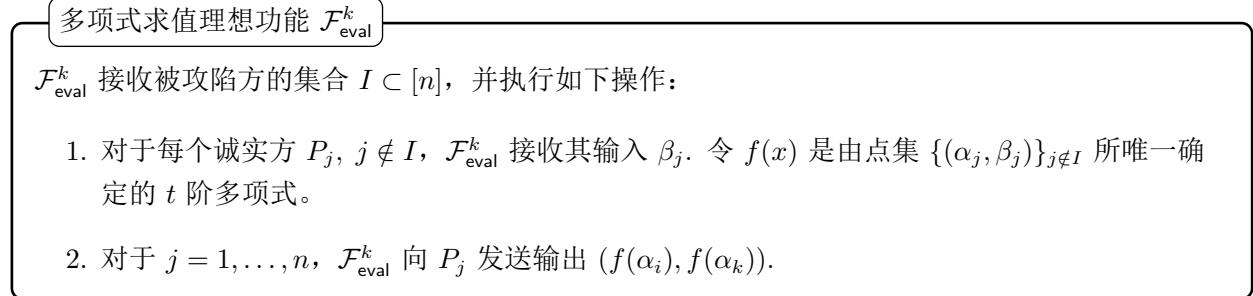


图 9.21: 多项式求值理想功能  $\mathcal{F}_{\text{eval}}^k$

$\mathcal{F}_{\text{eval}}^k$  可以写成以下等价的函数表示:

$$\mathcal{F}_{\text{eval}}^k(\beta_1, \dots, \beta_n) = \left( (f(\alpha_1), f(\alpha_k)), \dots, (f(\alpha_n), f(\alpha_k)) \right)$$

其中  $f$  是对  $(\beta_1, \dots, \beta_n)$  进行 Reed-Solomon 解码的输出。注意, 虽然  $f(\alpha_i)$  已经是  $P_i$  的输入了, 但为了让模拟器可以进行模拟, 我们还是需要将其作为输出的一部分 (这只是为了安全性证明的需要, 对实际协议不会有任何影响)。

**Reed-Solomon 编码背景知识。**下面我们说明:  $f(\alpha_k)$  是输入秘密份额  $(\beta_1, \dots, \beta_n)$  的线性组合<sup>15</sup>。定义参与方的输入向量为  $\vec{\beta} = (\beta_1, \dots, \beta_n)$ . 对于  $j \notin I$ , 有  $\beta_j = f(\alpha_j)$ , 因此, 参与方的输入由下式计算

$$\vec{\beta} = V_{\vec{\alpha}} \cdot \vec{f}^T$$

其中,  $V_{\vec{\alpha}}$  是 Vandermonde 矩阵, 定义如下

$$V_{\vec{\alpha}} = \begin{pmatrix} 1 & \alpha_1 & \dots & (\alpha_1)^{n-1} \\ 1 & \alpha_2 & \dots & (\alpha_2)^{n-1} \\ \vdots & \vdots & & \vdots \\ 1 & \alpha_n & \dots & (\alpha_n)^{n-1} \end{pmatrix}$$

$\vec{f}$  是多项式  $f(x)$  的系数构成的向量。注意,  $\vec{f}$  的长度为  $n$ , 它的第  $t+1$  项之后用 0 填充。令  $\vec{\alpha}_k = (1, \alpha_k, (\alpha_k)^2, \dots, (\alpha_k)^{n-1})$  是  $V_{\vec{\alpha}}$  的第  $k$  行。于是, 理想功能的输出为

$$f(\alpha_k) = \vec{\alpha}_k \cdot \vec{f}^T$$

我们有

$$\vec{\alpha}_k \cdot \vec{f}^T = \vec{\alpha}_k \cdot (V_{\vec{\alpha}}^{-1} \cdot V_{\vec{\alpha}}) \cdot \vec{f}^T = (\vec{\alpha}_k \cdot V_{\vec{\alpha}}^{-1}) \cdot (V_{\vec{\alpha}} \cdot \vec{f}^T) = (\vec{\alpha}_k \cdot V_{\vec{\alpha}}^{-1}) \cdot \vec{\beta}$$

<sup>15</sup>这里的秘密份额  $(\beta_1, \dots, \beta_n)$  是经过纠错的正确的秘密份额。

故存在常量向量  $\vec{\alpha}_k \cdot V_{\vec{\alpha}}^{-1}$ , 它与输入向量  $\vec{\beta}$  的内积就是要计算的输出。也就是说,  $\mathcal{F}_{\text{eval}}^k$  只需对参与方的输入做线性运算。

**协议构造。**由于  $\mathcal{F}_{\text{eval}}^k$  是对参与方的输入进行线性运算, 我们似乎可以采用与  $\mathcal{F}_{\text{mat}}^A$  同样的方法。但是, 请注意,  $\mathcal{F}_{\text{mat}}^A$  允许被攻陷方输入任何值, 而  $\mathcal{F}_{\text{eval}}^k$  需要保证被攻陷方的输入是正确的秘密份额 (即所有秘密份额位于相同的  $t$  阶多项式上)。幸运的是, 我们可以调用理想功能  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  来解决该问题—— $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  保证了输入分享阶段中参与方正确地分发了秘密份额的子份额; 协议的计算阶段与输出重建阶段与协议  $\Pi_{\text{mat}}^A$  完全一致。完整的协议如图 9.22 所示。

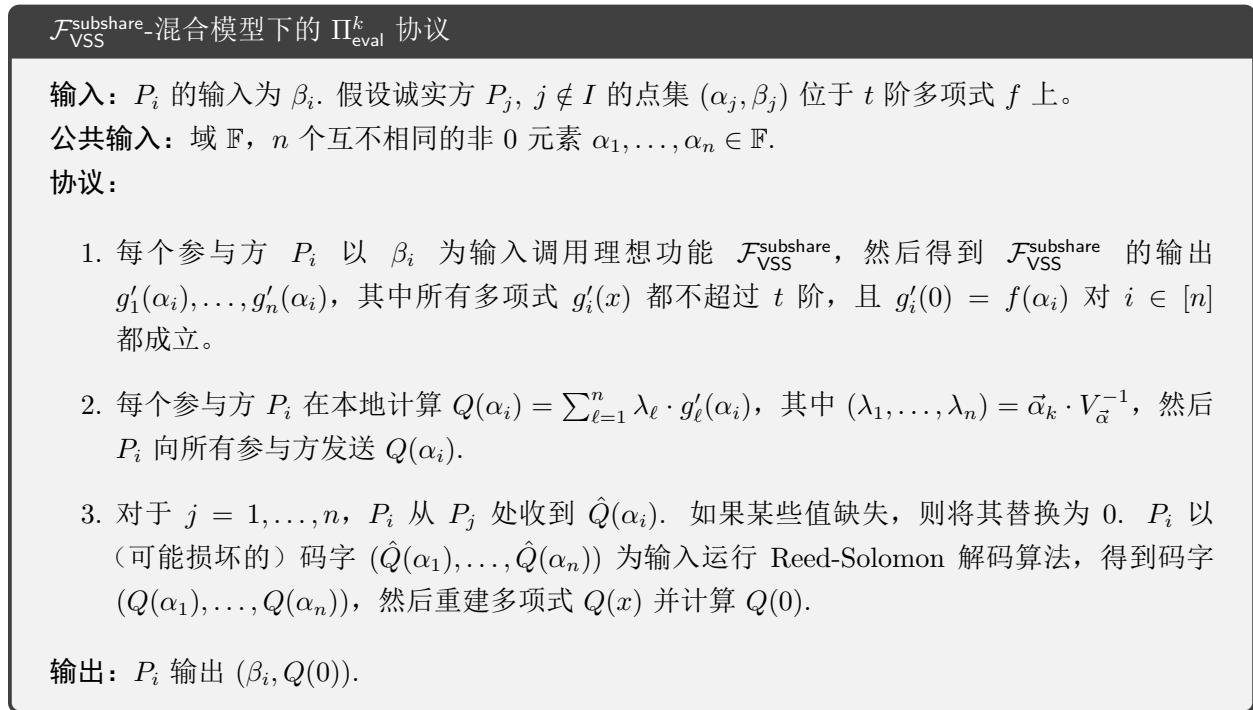


图 9.22:  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$ -混合模型下的  $\Pi_{\text{eval}}^k$  协议

直观上看, 该协议是安全的, 参与方调用  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  获得了输入的子份额, 而这些子份额不会泄露输入份额的任何信息。不过, 在构造模拟器  $\mathcal{S}$  时, 有一些细节需要考虑:  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  除了向  $P_i$  发送输出值  $g'_1(\alpha_i), \dots, g'_n(\alpha_i)$ , 还向敌手泄露了向量  $\vec{Y}(x) = (g_1(x), \dots, g_n(x)) \cdot H^T$ , 其中  $H$  是 Reed-Solomon 编码的校验矩阵,  $g_i(x)$  是  $P_i$  的输入多项式 (被攻陷方  $P_i$  的多项式  $g_i(x)$  如果常数项不正确, 则  $g_i(x) \neq g'_i(x)$ ; 对于诚实方  $P_j$  都有  $g_j(x) = g'_j(x)$ )。模拟器需要模拟  $\vec{Y}(x)$ . 注意,  $\vec{Y}(0) = (g_1(0), \dots, g_n(0)) \cdot H^T$  为输入值的伴随式。可以模拟的原因是伴随式仅依赖于错误向量  $\vec{e}$ . 下面展开说明。令  $\vec{\gamma} = (\gamma_1, \dots, \gamma_n)$  是参与方的输入向量 (对于  $i \in I$  可能有  $\gamma_i \neq f(\alpha_i)$ )。令 “正确的” 输入向量为  $\vec{\beta} = (f(\alpha_1), \dots, f(\alpha_n))$ . 向量  $\vec{\gamma}$  决定了错误向量  $\vec{e}$ ,  $\vec{e}$  满足  $\vec{\gamma} - \vec{e} = (f(\alpha_1), \dots, f(\alpha_n)) = \vec{\beta}$ , 且  $\vec{e}$  的汉明重量最多为  $t$ . 伴随式函数  $S(\vec{x}) = \vec{x} \cdot H^T$  满足  $S(\vec{\gamma}) = S(\vec{\beta} + \vec{e}) = S(\vec{e})$ , 即  $(\gamma_1, \dots, \gamma_n) \cdot H^T = \vec{e} \cdot H^T$ , 而模拟器可以自己计算  $\vec{e}$  (模拟器从  $\mathcal{F}_{\text{eval}}^k$  的输出中得到被攻陷方的  $f(\alpha_i)$ ,  $i \in I$ , 通过模拟  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  获得被攻陷方的输入多项式  $\{g_i(x)\}_{i \in I}$ , 而  $e_i = g_i(0) - f(\alpha_i)$ ; 对于  $j \notin I$ , 都有  $e_j = 0$ )。故模拟器可以计算  $\vec{e} \cdot H^T = \vec{\gamma} \cdot H^T = (g_1(0), \dots, g_n(0)) \cdot H^T = \vec{Y}(0)$ .

上面我们说明了模拟器可以计算  $\vec{Y}(0)$ . 除此之外, 对于  $i \in I$ , 模拟器知道  $g_1(\alpha_i), \dots, g_n(\alpha_i)$ , 它

可以计算  $\vec{Y}(\alpha_i) = (g_1(\alpha_i), \dots, g_n(\alpha_i)) \cdot H^T$ . 我们将证明:  $\vec{Y}(x)$  是在  $\vec{Y}(0)$  和  $\{\vec{Y}(\alpha_i)\}_{i \in I}$  的约束条件下的均匀随机  $t$  阶多项式向量 (当  $|I| = t$  时有  $t+1$  个约束条件, 此时  $\vec{Y}(x)$  是确定的); 对于  $Q(x)$  而言也是如此:  $Q(x)$  是在  $Q(0)$  和  $Q(\alpha_i) = \sum_{\ell=1}^n \lambda_\ell \cdot g'_\ell(\alpha_i)$ ,  $i \in I$  的约束条件下的均匀随机多项式 (当  $|I| = t$  时  $Q(x)$  是确定的)。故模拟器可以模拟协议中所有消息。

我们有以下定理。

**定理 9.18.** 假设被攻陷方的数量不超过  $t$ ,  $t < n/3$ . 图 9.22 中的协议  $\Pi_{\text{eval}}^k$  在  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$ -混合模型下对于静态恶意对手  $UC$ -安全实现了理想功能  $\mathcal{F}_{\text{eval}}^k$  (图 9.21)。

证明. 要证明此定理, 我们需要构造模拟器  $\mathcal{S}$ , 使得对所有环境  $\mathcal{Z}$ , 都有

$$\text{EXEC}_{\Pi_{\text{eval}}^k, \mathcal{A}, \mathcal{Z}}^{\mathcal{F}_{\text{VSS}}^{\text{subshare}}} \approx \text{EXEC}_{\mathcal{F}_{\text{eval}}^k, \mathcal{S}, \mathcal{Z}}^{\mathcal{F}_{\text{VSS}}^{\text{subshare}}}$$

其中  $\mathcal{A}$  是平凡敌手。

**模拟器  $\mathcal{S}$ .** 模拟器  $\mathcal{S}$  在内部运行  $\mathcal{A}$ , 转发  $\mathcal{Z}$  与  $\mathcal{A}$  之间的消息。在  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$ -混合模型下,  $\mathcal{S}$  还需要模拟  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$ .  $\mathcal{S}$  的工作方式如下。

1. 当诚实方向  $\mathcal{F}_{\text{eval}}^k$  发送输入后, 模拟器  $\mathcal{S}$  得到  $\mathcal{F}_{\text{eval}}^k$  向被攻陷方发送的输出  $\{(f(\alpha_i), f(\alpha_k))\}_{i \in I}$  ( $\mathcal{F}_{\text{eval}}^k$  的步骤 2)。
2. 模拟  $\Pi_{\text{eval}}^k$  的步骤 1:  $\mathcal{S}$  按照如下方式模拟  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$ :
  - (a)  $\mathcal{S}$  模拟  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  的步骤 3: 对于  $j \notin I$ , 随机选择多项式  $g_j(x) \leftarrow_{\$} \mathcal{P}^{0,t}$ , 然后向敌手  $\mathcal{A}$  发送  $\{g_j(\alpha_i)\}_{j \notin I, i \in I}$ .
  - (b)  $\mathcal{S}$  模拟  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  的步骤 4, 接收并记录被攻陷方的输入多项式  $\{g_i(x)\}_{i \in I}$ . 如果没有收到  $g_i(x)$ , 设置  $g_i(x) = 0$ .
  - (c) 对于  $i \in I$ , 检查  $\deg(g_i(x)) \leq t$  与  $g_i(0) = f(\alpha_i)$  是否成立, 如果都成立, 设置  $g'_i(x) = g_i(x)$ , 否则, 设置  $g'_i(x) = f(\alpha_i)$ . ( $\mathcal{S}$  已经从  $\mathcal{F}_{\text{eval}}^k$  的输出中得到了  $f(\alpha_i)$ )
  - (d) 对于  $j \notin I$ , 设置  $g'_j(x) = g_j(x)$ .
  - (e)  $\mathcal{S}$  模拟  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  的步骤 6:
    - i.  $\mathcal{S}$  模拟  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  向敌手  $\mathcal{A}$  发送多项式向量  $\vec{Y}(x)$ , 通过如下方式选取:
      - 计算  $(e_1, \dots, e_n)$ : 对于  $j \notin I$  设置  $e_j = 0$ , 对于  $i \in I$  设置  $e_i = g_i(0) - f(\alpha_i)$ .
      - 均匀随机选择  $t$  阶多项式向量  $\vec{Y}(x)$ , 满足约束条件  $\vec{Y}(0) = (e_1, \dots, e_n) \cdot H^T$  以及  $\vec{Y}(\alpha_i) = (g_1(\alpha_i), \dots, g_n(\alpha_i)) \cdot H^T$ ,  $i \in I$ .
    - ii.  $\mathcal{S}$  模拟  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  向被攻陷方  $P_i, i \in I$  发送输出  $g'_i(x)$  和  $g'_1(\alpha_i), \dots, g'_n(\alpha_i)$ .
3. 模拟  $\Pi_{\text{eval}}^k$  的步骤 3:
  - (a)  $\mathcal{S}$  均匀随机选择  $t$  阶多项式  $Q(x)$ , 满足约束条件  $Q(0) = f(\alpha_i)$  以及  $Q(\alpha_i) = \sum_{\ell=1}^n \lambda_\ell \cdot g'_\ell(\alpha_i)$ ,  $i \in I$ , 其中  $(\lambda_1, \dots, \lambda_n) = \vec{\alpha}_k \cdot V_{\vec{\alpha}}^{-1}$ .
  - (b) 对于  $j \notin I$ ,  $\mathcal{S}$  模拟  $P_j$  向所有参与方发送  $Q(\alpha_j)$ .

不可区分性。真实世界的运行和理想世界的运行有三个不同：(1) 对于  $j \notin I$ ,  $\mathcal{S}$  选择的多项式  $g_j(x)$  的常数项为 0 而不是  $f(\alpha_j)$ ; (2)  $\mathcal{S}$  是基于约束条件随机选择的  $\vec{Y}(x)$ , 而真实世界中  $\vec{Y}(x)$  是由  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  基于  $(g_1(x), \dots, g_n(x))$  计算得到的; (3)  $\mathcal{S}$  是基于约束条件随机选择的  $Q(x)$ , 而真实世界中  $Q(x)$  是基于  $(g'_1(x), \dots, g'_n(x))$  计算出来的。我们通过三个混合实验证明真实世界和理想世界是不可区分的。

**混合实验  $\mathcal{H}_0$ :** 这是理想世界的执行  $\text{EXEC}_{\mathcal{F}_{\text{eval}}^k, \mathcal{S}, \mathcal{Z}}$ .

**混合实验  $\mathcal{H}_1$ :**  $\mathcal{H}_1$  与  $\mathcal{H}_0$  相同, 除了模拟器  $\mathcal{S}$  在步骤 2(a) 中, 选取诚实方  $P_j$  多项式  $g_j(x) \leftarrow_{\$} \mathcal{P}^{\beta_j, t}$  而不是  $g_j(x) \leftarrow_{\$} \mathcal{P}^{0, t}$ , 其中  $\beta_j = f(\alpha_j)$  是  $P_j$  的输入。

**断言 1:** 如果  $|I| \leq t$ , 那么  $\mathcal{H}_1$  与  $\mathcal{H}_0$  完美不可区分。

证明. 由于  $|I| \leq t$ , 根据推论 9.13,  $\mathcal{H}_1$  与  $\mathcal{H}_0$  中的敌手得到的秘密份额  $\{g_j(\alpha_i)\}_{j \notin I, i \in I}$  具有相同的分布, 因此  $\mathcal{H}_1$  与  $\mathcal{H}_0$  完美不可区分。  $\square$

**混合实验  $\mathcal{H}_2$ :**  $\mathcal{H}_2$  与  $\mathcal{H}_1$  相同, 除了  $\vec{Y}(x)$  的计算方式改用真实世界中  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  的计算方式, 即: 对于  $j \notin I$ , 均匀随机选取  $t$  阶多项式  $g_j(x)$  满足  $g_j(0) = f(\alpha_j)$ , 对于  $i \in I$ , 使用被攻陷方提供的多项式  $g_i(x)$ , 然后基于  $(g_1(x), \dots, g_n(x))$  计算。

**断言 2:** 如果  $|I| \leq t$ , 那么  $\mathcal{H}_2$  与  $\mathcal{H}_1$  完美不可区分。

证明. 这是基于伴随式函数  $S(\vec{x}) = \vec{x} \cdot H^T$  的性质。令  $\vec{\gamma}$  是参与方的实际输入, 令  $\vec{e} = (e_1, \dots, e_n)$  是错误向量, 那么, 我们有  $S(\vec{\gamma}) = S(\vec{e})$ . 因此, 在  $\mathcal{H}_2$  和  $\mathcal{H}_1$  中, 对于  $\vec{Y}(0)$  和  $\{\vec{Y}(\alpha_i)\}_{i \in I}$  的约束条件实际上是相同的。接下来的证明与半诚实的情况同理, 根据定理 6.1 中的证明, 对于  $\vec{Y}(x)$  中的每个多项式  $Y_\ell(x)$ ,  $\ell = 1, \dots, 2t$ , 基于不超过  $t+1$  个约束条件均匀随机选择的多项式与真实世界的多项式有相同的分布, 故  $\mathcal{H}_2$  与  $\mathcal{H}_1$  完美不可区分。  $\square$

**混合实验  $\mathcal{H}_3$ :**  $\mathcal{H}_3$  与  $\mathcal{H}_2$  相同, 除了  $Q(x)$  的计算方式改用真实世界中的计算方式, 即基于  $(g'_1(x), \dots, g'_n(x))$  计算。

**断言 3:** 如果  $|I| \leq t$ , 那么  $\mathcal{H}_3$  与  $\mathcal{H}_2$  完美不可区分。

证明. 与断言 2 同理,  $\mathcal{H}_3$  与  $\mathcal{H}_2$  中  $Q(x)$  的分布完全相同。  $\square$

**混合实验  $\mathcal{H}_4$ :** 这是真实世界的执行  $\text{EXEC}_{\Pi_{\text{eval}}^k, \mathcal{A}, \mathcal{Z}}^{\mathcal{F}_{\text{VSS}}^{\text{subshare}}}$ .

**断言 4:** 如果  $|I| \leq t$ , 那么  $\mathcal{H}_4$  与  $\mathcal{H}_3$  完美不可区分。

证明.  $\mathcal{H}_4$  与  $\mathcal{H}_3$  中, 敌手  $\mathcal{A}$  的视图完全相同 (根据定义)。我们只需要证明真实世界与理想世界中诚实方的输出相同, 即真实协议中诚实方的输出一定是  $f(\alpha_k)$ . 不难发现,  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  保证了输入秘密份额的子份额被正确地分发了, 而  $|I| \leq t$ , 码字  $(\hat{Q}(\alpha_1), \dots, \hat{Q}(\alpha_n))$  至少有  $n-t$  个正确的元素, 根据 Reed-Solomon 编码的性质, 参与方可以重建正确的  $Q(0) = f(\alpha_k)$ .  $\square$

综上所述, 真实世界与理想世界完美不可区分, 证毕。  $\square$

### 9.3.4.5 实现乘积子份额秘密分享理想功能 $\mathcal{F}_{\text{VSS}}^{\text{mult}}$

理想功能  $\mathcal{F}_{\text{VSS}}^{\text{mult}}$  允许秘密分发者在已经分享了  $t$  阶多项式  $A(x)$  和  $B(x)$  的情况下，可验证地分享随机  $t$  阶多项式  $C(x)$ ，满足  $C(0) = A(0) \cdot B(0)$ 。这是协议中关键的一步（参见第9.3.1节的乘法协议概览）。 $\mathcal{F}_{\text{VSS}}^{\text{mult}}$  的定义如图 9.23 所示。

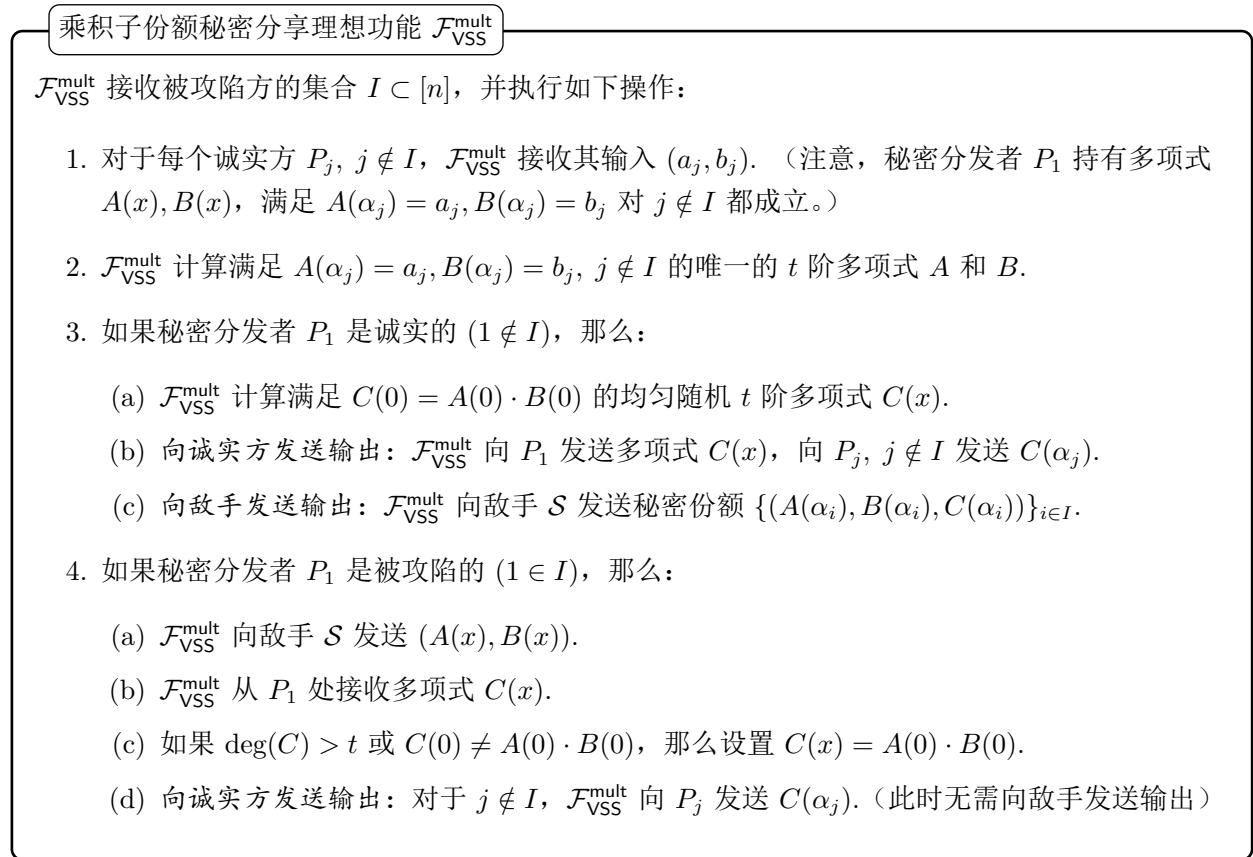


图 9.23: 乘积子份额秘密分享理想功能  $\mathcal{F}_{\text{VSS}}^{\text{mult}}$

尽管秘密分发者  $P_1$  本来就应该知道  $A(x), B(x)$ ，参与方  $P_i$  本来就应该知道  $A(\alpha_i), B(\alpha_i)$ ，在上述定义中  $\mathcal{F}_{\text{VSS}}^{\text{mult}}$  还是将这些信息提供给了敌手  $\mathcal{S}$ ，这是因为  $\mathcal{S}$  需要知道被攻陷方正确的份额才能够进行模拟。在整个乘法协议中，敌手原本就知道这些信息，因此  $\mathcal{F}_{\text{VSS}}^{\text{mult}}$  再次发送它们不会造成额外的信息泄露。

请读者注意， $\mathcal{F}_{\text{VSS}}^{\text{mult}}$  适用于参与方已经拥有（导线值的）秘密份额  $a$  和  $b$  时，通过  $t$  阶多项式  $C(x)$ （满足  $C(0) = a \cdot b$ ）可验证地分发  $a \cdot b$  的子份额。这里的  $a$  和  $b$  不是导线上的值，而是导线值的秘密份额。

**协议思想。**令  $A(x)$  和  $B(x)$  是用于分享  $a$  和  $b$  的多项式，满足  $A(0) = a, B(0) = b$ 。协议的思想是：秘密分发者首先选择  $t$  个多项式  $D_1(x), \dots, D_t(x)$ ，它们均为  $t$  阶，使得  $C(x) = A(x) \cdot B(x) - \sum_{\ell=1}^t x^\ell \cdot D_\ell(x)$  是常数项为  $a \cdot b$  的随机  $t$  阶多项式。注意， $A(x)$  和  $B(x)$  均为  $t$  阶，故  $A(x) \cdot B(x)$  的阶数为  $2t$ ，秘密分发者选择的  $D_1(x), \dots, D_t(x)$  需要使  $A(x) \cdot B(x)$  高于  $t$  阶的项全部被抵消，这样  $C(x)$  的阶数才能为  $t$ 。然后，秘密分发者分享多项式  $D_1(x), \dots, D_t(x)$ ，每个参与方在本地计算  $C(x)$  的秘密份额。

注意到，每一个  $D_\ell(x)$  都乘以了  $x^\ell$ ，因此无论  $D_1(x), \dots, D_t(x)$  如何选择， $C(x)$  的常数项始终等于  $A(0) \cdot B(0) = a \cdot b$ 。也就是说，即使恶意的秘密分发者错误地选择  $D_1(x), \dots, D_t(x)$ ，也不会影响  $C(x)$  的常数项，但  $C(x)$  的阶数可能超过  $t$ 。

更具体地，在确定  $D_1(x), \dots, D_t(x)$  后，秘密分发者调用  $\mathcal{F}_{\text{VSS}}$  分享这些多项式， $\mathcal{F}_{\text{VSS}}$  保证了多项式的阶数为  $t$  且参与方持有正确的份额。而参与方本来就持有  $A(x)$  和  $B(x)$  的份额。给定  $A(\alpha_j), B(\alpha_j)$  以及  $D_1(\alpha_j), \dots, D_t(\alpha_j)$ ， $P_j$  可以本地计算  $C(\alpha_j) = A(\alpha_j) \cdot B(\alpha_j) - \sum_{\ell=1}^t (\alpha_j)^\ell \cdot D_\ell(\alpha_j)$ 。此时，如果秘密分发者是诚实的，那么参与方的秘密份额位于常数项为  $a \cdot b$  的  $t$  阶多项式上；如果秘密分发者是被攻陷的，那么这些秘密份额仍然位于常数项为  $a \cdot b$  的多项式上（但阶数不能保证）。因此，只需要再设计一种方法使参与方验证  $C(x)$  确实是  $t$  阶多项式即可。

一个直接的想法是：让秘密分发者再调用  $\mathcal{F}_{\text{VSS}}$  分享  $C(x)$ ，然后参与方验证自己收到的份额与基于  $A(\alpha_j), B(\alpha_j), D_1(\alpha_j), \dots, D_t(\alpha_j)$  计算得到的  $C(\alpha_j)$  相等。具体而言，将  $P_j$  从  $\mathcal{F}_{\text{VSS}}$  收到的份额记为  $C'(\alpha_j)$ ，将计算  $A(\alpha_j) \cdot B(\alpha_j) - \sum_{\ell=1}^t (\alpha_j)^\ell \cdot D_\ell(\alpha_j)$  所得的值记为  $C(\alpha_j)$ 。如果  $C(\alpha_j) \neq C'(\alpha_j)$ ，那么参与方就像 VSS 协议中（图 9.11）那样广播“投诉”。如果有超过  $t$  个参与方投诉，那么秘密分发者就是被攻陷的，所有参与方广播自己的份额以重建  $A(x)$  和  $B(x)$ ，并输出  $A(0) \cdot B(0)$ 。但是，需要注意的是， $C(x)$  可以是阶数为  $2t$  的多项式，它与另一个  $t$  阶多项式  $C'(x)$  最多可以相交于  $2t$  个点。因此，恶意的秘密分发者可以通过  $\mathcal{F}_{\text{VSS}}$  分享  $t$  阶多项式  $C'(x)$ ，其中  $C'(x)$  与  $C(x)$  在  $2t$  个位置  $\alpha_j$  处的值相等，且  $C'(0) \neq a \cdot b$ 。由于诚实方的数量至少为  $2t+1$ ，此时，我们只能保证至少有 1 个诚实方会广播投诉，但投诉的数量不足以让参与方确定秘密分发者是被攻陷的并重建其份额。

这个问题的解决方法是：参与方要明确地解决每一个投诉，验证其是否为合法的。只要有一个合法的投诉，参与方就重建  $a$  和  $b$  并输出  $a \cdot b$ ；如果投诉的不合法的，那么忽略它。这保证了如果没有诚实方提出合法的投诉，那么秘密分发者通过  $\mathcal{F}_{\text{VSS}}$  分享的  $t$  阶多项式  $C'(x)$  与  $C(x)$  在至少  $2t+1$  个点处的值一致，而  $C(x)$  的阶数最多为  $2t$ ，所以一定满足  $C(x) = C'(x)$ ，即  $C(x)$  的阶数为  $t$ 。

这里，我们需要调用第9.3.4.4节实现的理想功能  $\mathcal{F}_{\text{eval}}^k$  来明确地解决投诉。当  $P_k$  提出投诉时，参与方调用  $\mathcal{F}_{\text{eval}}^k$  来计算  $A(\alpha_k), B(\alpha_k), D_1(\alpha_k), \dots, D_t(\alpha_k)$  以及  $C'(\alpha_k)$ ，然后检查  $C'(\alpha_k) \stackrel{?}{=} A(\alpha_k) \cdot B(\alpha_k) - \sum_{\ell=1}^t (\alpha_k)^\ell \cdot D_\ell(\alpha_k)$ 。如果等式不成立，那么秘密分发者就是被攻陷的，参与方重建  $a \cdot b$  并输出。上述方法可以明确地验证每个投诉是否为合法的。注意，参与方一定持有  $C'(x), D_1(x), \dots, D_t(x)$  的正确份额，因为它们是通过  $\mathcal{F}_{\text{VSS}}$  分享的，而根据输入的假设，参与方持有的  $A(x)$  和  $B(x)$  的份额也是正确的，因此可以调用  $\mathcal{F}_{\text{eval}}^k$  进行计算。

在秘密分发者是诚实的情况下，参与方不可能提出合法的投诉。此时，如果被攻陷方提出错误的投诉，投诉解决的过程只会公布（被攻陷的）投诉者自己的秘密份额，因而不会让敌手获得任何额外信息。**构造多项式  $C(x)$** 。上文中说到，秘密分发者需要选择特殊构造的  $t$  个  $t$  阶多项式  $D_1(x), \dots, D_t(x)$  使得  $C(x) = A(x) \cdot B(x) - \sum_{\ell=1}^t x^\ell \cdot D_\ell(x)$  是常数项为  $a \cdot b$  的随机  $t$  阶多项式，其中  $a = A(0), b = B(0)$ 。下面我们介绍选择这些多项式的方式。首先，定义多项式  $D(x)$  如下：

$$D(x) = A(x) \cdot B(x) = a \cdot b + d_1 x + \cdots + d_{2t} x^{2t}$$

因为  $A(x)$  和  $B(x)$  为  $t$  阶多项式，所以  $D(x)$  的阶数为  $2t$ 。然后，秘密分发者定义  $D_1(x), \dots, D_t(x)$  如下：

$$\begin{aligned}
D_t(x) &= r_{t,0} + r_{t,1}x + \cdots + r_{t,t-1}x^{t-1} + d_{2t}x^t \\
D_{t-1}(x) &= r_{t-1,0} + r_{t-1,1}x + \cdots + r_{t-1,t-1}x^{t-1} + (d_{2t-1} - r_{t,t-1}) \cdot x^t \\
D_{t-2}(x) &= r_{t-2,0} + r_{t-2,1}x + \cdots + r_{t-2,t-1}x^{t-1} + (d_{2t-2} - r_{t-1,t-1} - r_{t,t-2}) \cdot x^t \\
&\vdots \\
D_1(x) &= r_{1,0} + r_{1,1}x + \cdots + r_{1,t-1}x^{t-1} + (d_{t+1} - r_{t,1} - r_{t-1,2} - \cdots - r_{2,t-1}) \cdot x^t
\end{aligned}$$

其中  $r_{i,j} \leftarrow_{\$} \mathbb{F}$  都是均匀随机值,  $d_i$  是  $D(x) = A(x) \cdot B(x)$  的系数。也就是说,  $D_\ell(x)$  除了  $x^t$  的系数是特殊选择的, 其他项的系数都是均匀随机选择的。我们将  $D_\ell(x)$ ,  $1 \leq \ell \leq t$  重新写为以下形式:

$$D_\ell(x) = r_{\ell,0} + r_{\ell,1} \cdot x + \cdots + r_{\ell,t-1} \cdot x^{t-1} + \left( d_{t+\ell} - \sum_{m=\ell+1}^t r_{m,t+\ell-m} \right) \cdot x^t$$

多项式  $C(x)$  通过下式计算得到:

$$C(x) = D(x) - \sum_{\ell=1}^t x^\ell \cdot D_\ell(x)$$

让我们验算一下, 当  $D_1(x), \dots, D_t(x)$  按照上述方式选择时,  $C(x)$  是常数项为  $a \cdot b$  的  $t$  阶多项式, 即:  $D(x)$  中高于  $t$  阶的项都被抵消了。令

$$R_{\ell,t} = d_{t+\ell} - \sum_{m=\ell+1}^t r_{m,t+\ell-m} \quad (9.11)$$

那么

$$D_\ell(x) = r_{\ell,0} + r_{\ell,1} \cdot x + \cdots + r_{\ell,t-1} \cdot x^{t-1} + R_{\ell,t} \cdot x^t$$

我们来分析一下多项式  $\sum_{\ell=1}^t x^\ell \cdot D_\ell(x)$ . 首先, 它的阶数为  $2t$ , 常数项为 0. 我们将它的系数画成一张表, 如表 9.1 所示。

	$x$	$x^2$	$x^3$	$\dots$	$x^t$	$x^{t+1}$	$x^{t+2}$	$\dots$	$x^{2t-2}$	$x^{2t-1}$	$x^{2t}$
$D_t$					$r_{t,0}$	$r_{t,1}$	$r_{t,2}$	$\dots$	$r_{t,t-2}$	$r_{t,t-1}$	$R_{t,t}$
$D_{t-1}$				$\dots$	$r_{t-1,1}$	$r_{t-1,2}$	$r_{t-1,3}$	$\dots$	$r_{t-1,t-1}$	$R_{t-1,t}$	
$D_{t-2}$				$\dots$	$r_{t-2,2}$	$r_{t-2,3}$	$r_{t-2,4}$	$\dots$	$R_{t-2,t}$		
$\vdots$				$\ddots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$			
$D_3$			$r_{3,0}$	$\dots$	$r_{3,t-3}$	$r_{3,t-2}$	$r_{3,t-1}$	$\dots$			
$D_2$		$r_{2,0}$	$r_{2,1}$	$\dots$	$r_{2,t-2}$	$r_{2,t-1}$	$R_{2,t}$				
$D_1$	$r_{1,0}$	$r_{1,1}$	$r_{1,2}$	$\dots$	$r_{1,t-1}$	$R_{1,t}$					

表 9.1: 多项式  $\sum_{\ell=1}^t x^\ell \cdot D_\ell(x)$  的系数

表格中,  $D_\ell(x)$  的系数向右移了  $\ell$  格, 这是因为在多项式  $\sum_{\ell=1}^t x^\ell \cdot D_\ell(x)$  中  $D_\ell(x)$  要乘以  $x^\ell$ . 因此, 将表中每一列全部相加即可得到对应项的系数。于是, 对于  $k = 1, \dots, t$ , 将  $x^{t+k}$  对应的列全部相加, 得到其系数为

$$R_{k,t} + r_{k+1,t-1} + r_{k+2,t-2} + \cdots + r_{t,k} = R_{k,t} + \sum_{m=k+1}^t r_{m,t+k-m}$$

根据式 (9.11) 中  $R_{k,t}$  的定义, 这个值等于  $d_{t+k}$ . 而  $d_{t+k}$  是  $D(x)$  中  $x^{t+k}$  的系数, 由此可知  $C(x) = D(x) - \sum_{\ell=1}^t x^\ell \cdot D_\ell(x)$  中大于  $t$  阶的项全部被抵消了, 因此  $C(x)$  是  $t$  阶多项式。最后, 我们注意到  $x, x^2, \dots, x^t$  的系数都是均匀随机的 (因为对于  $i = 1, \dots, t$ ,  $r_{i,0}$  都是均匀随机选取的)。故  $C(x)$  除了常数项为  $A(0) \cdot B(0) = a \cdot b$ , 其他项的系数都是均匀随机的。

**协议描述。**我们在  $\{\mathcal{F}_{\text{VSS}}, \mathcal{F}_{\text{eval}}^1, \dots, \mathcal{F}_{\text{eval}}^n\}$ -混合模型下给出完整的协议描述, 如图 9.24 所示。为了简洁, 我们用  $\mathcal{F}_{\text{eval}}$  表示  $n$  个理想功能  $\mathcal{F}_{\text{eval}}^1, \dots, \mathcal{F}_{\text{eval}}^n$ .

我们有以下定理。

**定理 9.19.** 假设参与方之间存在认证的广播信道。假设被攻陷方的数量不超过  $t$ ,  $t < n/3$ . 图 9.24 中的协议  $\Pi_{\text{VSS}}^{\text{mult}}$  在  $\{\mathcal{F}_{\text{VSS}}, \mathcal{F}_{\text{eval}}\}$ -混合模型下对于静态恶意敌手  $UC$ -安全实现了理想功能  $\mathcal{F}_{\text{VSS}}^{\text{mult}}$  (图 9.23)。

证明. 我们分两种情况考虑: (1) 秘密分发者是诚实的; (2) 秘密分发者是被攻陷的。

**情况 1:** 秘密分发者  $P_1$  是诚实的。此时, 由于秘密分发者是诚实的, 敌手  $\mathcal{A}$  在所有  $\mathcal{F}_{\text{VSS}}$  的调用中都没有输入, 只接收秘密份额。此外, 根据  $\mathcal{F}_{\text{VSS}}^{\text{mult}}$  的定义, 模拟器  $\mathcal{S}$  不向  $\mathcal{F}_{\text{VSS}}^{\text{mult}}$  提供输入,  $\mathcal{S}$  只需模拟敌手的视图。模拟器  $\mathcal{S}$  采用如下方式进行模拟。首先, 均匀随机选择  $t$  阶多项式  $D_2(x), \dots, D_t(x)$ , 然后, 在以下约束条件下均匀随机选择  $t$  阶多项式  $D_1(x)$ :

$$\alpha_i \cdot D_1(\alpha_i) = A(\alpha_i) \cdot B(\alpha_i) - C(\alpha_i) - \sum_{\ell=2}^t (\alpha_i)^\ell \cdot D_\ell(\alpha_i) \quad \text{对 } i \in I \text{ 都成立}$$

这样计算得到的  $D_1(\alpha_i), \dots, D_t(\alpha_i)$  具有正确的分布, 因为

$$C(x) = D(x) - \sum_{\ell=1}^t x^\ell \cdot D_\ell(x) = A(x) \cdot B(x) - x \cdot D_1(x) - \sum_{\ell=2}^t x^\ell \cdot D_\ell(x)$$

即

$$x \cdot D_1(x) = A(x) \cdot B(x) - C(x) - \sum_{\ell=2}^t x^\ell \cdot D_\ell(x)$$

我们将证明, 模拟器选择的  $D_\ell(x)$  与诚实的秘密分发者选择的  $D_\ell(x)$  具有相同的分布 (因为它们都是在约束条件  $C(\alpha_i) = A(\alpha_i) \cdot B(\alpha_i) - \sum_{\ell=1}^t (\alpha_i)^\ell \cdot D_\ell(\alpha_i)$ ,  $i \in I$  下的随机多项式)。 $\mathcal{S}$  还需要模拟投诉解决。注意到, 当秘密分发者是诚实的时候, 诚实方不会广播投诉。对于每个被攻陷方  $P_i$  ( $i \in I$ ) 广播的投诉 (complaint,  $i$ ),  $\mathcal{S}$  很容易进行模拟, 因为  $\mathcal{S}$  知道正确的  $\tilde{A}(\alpha_i) = A(\alpha_i)$ ,  $\tilde{B}(\alpha_i) = B(\alpha_i)$ ,  $\tilde{C}(\alpha_i) = C(\alpha_i)$ , 对于  $\ell = 1, \dots, t$ ,  $\mathcal{S}$  使用它选择的  $\tilde{D}_\ell(\alpha_i) = D_\ell(\alpha_i)$  作为  $\mathcal{F}_{\text{eval}}^i$  的输出。

模拟器  $\mathcal{S}$  的工作方式如下。

1. 当诚实方向  $\mathcal{F}_{\text{VSS}}^{\text{mult}}$  发送输入后,  $\mathcal{S}$  收到  $\mathcal{F}_{\text{VSS}}^{\text{mult}}$  发送的秘密份额  $\{(A(\alpha_i), B(\alpha_i), C(\alpha_i))\}_{i \in I}$  ( $\mathcal{F}_{\text{VSS}}^{\text{mult}}$  的步骤 3(c))。
2.  $\mathcal{S}$  选择  $t-1$  个均匀随机  $t$  阶多项式  $D_2(x), \dots, D_t(x)$ .

$\{\mathcal{F}_{\text{VSS}}, \mathcal{F}_{\text{eval}}\}$ -混合模型下的  $\Pi_{\text{VSS}}^{\text{mult}}$  协议

输入：

1. 秘密分发者  $P_1$  的输入是  $t$  阶多项式  $A(x)$  和  $B(x)$ ,  $A(0) = a, B(0) = b$ .
2. 参与方  $P_i$  的输入是  $a_i, b_i$ , 满足  $a_i = A(\alpha_i), b_i = B(\alpha_i)$ .

公共输入：域  $\mathbb{F}$ ,  $n$  个互不相同的非 0 元素  $\alpha_1, \dots, \alpha_n \in \mathbb{F}$ .

协议：

1. 分发阶段：

- (a) 秘密分发者  $P_1$  计算（阶数为  $2t$  的）多项式  $D(x) = A(x) \cdot B(x)$ , 记为  $D(x) = a \cdot b + \sum_{\ell=1}^{2t} d_\ell \cdot x^\ell$ .
- (b)  $P_1$  在域  $\mathbb{F}$  上均匀随机选择  $t^2$  个值  $\{r_{k,j}\}$ , 其中  $k = 1, \dots, t$ ,  $j = 0, \dots, t-1$ .
- (c) 对于  $\ell = 1, \dots, t$ ,  $P_1$  定义多项式  $D_\ell(x)$  如下：

$$D_\ell(x) = \left( \sum_{m=0}^{t-1} r_{\ell,m} \cdot x^m \right) + \left( d_{\ell+t} - \sum_{m=\ell+1}^t r_{m,t+\ell-m} \right) \cdot x^t$$

- (d)  $P_1$  计算多项式  $C(x) = D(x) - \sum_{\ell=1}^t x^\ell \cdot D_\ell(x)$ .
- (e)  $P_1$  作为秘密分发者以  $C(x)$  为输入调用  $\mathcal{F}_{\text{VSS}}$ ; 每个参与方  $P_i$  收到  $C(\alpha_i)$ .
- (f) 对于  $\ell = 1, \dots, t$ ,  $P_1$  作为秘密分发者以  $D_\ell(x)$  为输入调用  $\mathcal{F}_{\text{VSS}}$ ; 每个参与方  $P_i$  收到  $D_\ell(\alpha_i)$ .

2. 验证阶段：每个参与方  $P_i$  执行以下操作：

- (a) 如果任何一个份额  $C(\alpha_i), D_\ell(\alpha_i)$  为  $\perp$ , 跳到拒绝阶段（注意，如果有一个诚实方收到  $\perp$  那么所有诚实方收到的都是  $\perp$ ）。
- (b) 否则,  $P_i$  计算  $c'_i = a_i \cdot b_i - \sum_{\ell=1}^t (\alpha_i)^\ell \cdot D_\ell(\alpha_i)$ . 如果  $c'_i \neq C(\alpha_i)$ , 那么  $P_i$  广播 (complaint,  $i$ ).
- (c) 如果有参与方  $P_k$  广播 (complaint,  $k$ ), 那么进入投诉解决阶段; 否则, 进入输出阶段并输出  $C(\alpha_i)$ .

3. 投诉解决阶段：设置  $\text{reject} = \text{false}$ . 然后, 对于每条投诉 (complaint,  $k$ ), 执行以下操作：

- (a) 调用  $t+3$  次  $\mathcal{F}_{\text{eval}}^k$ : 第 1 次调用中  $P_i$  的输入为  $a_i$ , 第 2 次调用中  $P_i$  的输入为  $b_i$ , 第 3 次调用中  $P_i$  的输入为  $C(\alpha_i)$ , 第  $\ell+3$  次调用中  $P_i$  的输入为  $D_\ell(\alpha_i)$ ,  $\ell = 1, \dots, t$ .
- (b) 令  $\tilde{A}(\alpha_k), \tilde{B}(\alpha_k), \tilde{C}(\alpha_k), \tilde{D}_1(\alpha_k), \dots, \tilde{D}_t(\alpha_k)$  为参与方在  $t+3$  次调用  $\mathcal{F}_{\text{eval}}^k$  中得到的输出。计算  $\tilde{C}'(\alpha_k) = \tilde{A}(\alpha_k) \cdot \tilde{B}(\alpha_k) - \sum_{\ell=1}^t (\alpha_k)^\ell \cdot \tilde{D}_\ell(\alpha_k)$ .
- (c) 如果  $\tilde{C}'(\alpha_k) \neq \tilde{C}(\alpha_k)$ , 那么设置  $\text{reject} = \text{true}$ .

如果  $\text{reject} = \text{false}$ , 那么进入输出阶段并输出  $C(\alpha_i)$ ; 否则, 进入拒绝阶段。

4. 拒绝阶段：

- (a) 每个参与方  $P_i$  广播  $(a_i, b_i)$ . 令  $\vec{a} = (a_1, \dots, a_n)$  和  $\vec{b} = (b_1, \dots, b_n)$  是广播的值（缺失的值用 0 代替）。 $P_i$  以  $\vec{a}$  和  $\vec{b}$  为输入运行 Reed-Solomon 解码算法, 得到  $A'(x)$  和  $B'(x)$ .
- (b) 每个参与方  $P_i$  设置  $C(\alpha_i) = A'(0) \cdot B'(0)$ .

输出：每个参与方  $P_i$  输出  $C(\alpha_i)$ .

图 9.24:  $\{\mathcal{F}_{\text{VSS}}, \mathcal{F}_{\text{eval}}\}$ -混合模型下的  $\Pi_{\text{VSS}}^{\text{mult}}$  协议

3. 对于  $i \in I$ ,  $\mathcal{S}$  计算

$$D_1(\alpha_i) = (\alpha_i)^{-1} \cdot \left( A(\alpha_i) \cdot B(\alpha_i) - C(\alpha_i) - \sum_{\ell=2}^t (\alpha_i)^\ell \cdot D_\ell(\alpha_i) \right)$$

4. 模拟  $\Pi_{\text{VSS}}^{\text{mult}}$  的步骤 1(e) 和 1(f):  $\mathcal{S}$  模拟  $\mathcal{F}_{\text{VSS}}$  向每个被攻陷方  $P_i$ ,  $i \in I$  发送输出  $C(\alpha_i), D_1(\alpha_i), \dots, D_t(\alpha_i)$ .
5. 模拟  $\Pi_{\text{VSS}}^{\text{mult}}$  的步骤 2 和 3: 对于每个广播了  $(\text{complaint}, k)$  的被攻陷方  $P_k$ ,  $\mathcal{S}$  模拟投诉解决阶段:  $\mathcal{S}$  模拟  $t+3$  次  $\mathcal{F}_{\text{eval}}^k$  的调用, 对于  $i \in I$ , 模拟  $\mathcal{F}_{\text{eval}}^k$  向被攻陷方  $P_i$  发送输出  $(A(\alpha_i), A(\alpha_k)), (B(\alpha_i), B(\alpha_k)), (C(\alpha_i), C(\alpha_k))$  和  $\{(D_\ell(\alpha_i), D_\ell(\alpha_k))\}_{\ell=1}^t$ .

**不可区分性。** 我们证明理想世界与真实世界是不可区分的。我们首先证明诚实方的输出有相同的分布, 然后, 我们证明: 给定诚实方的输出, 敌手的视图在理想世界和真实世界的分布相同。

**诚实方的输出。** 令诚实方的输入所在的  $t$  阶多项式为  $A(x)$  和  $B(x)$ . 在理想世界中, 理想功能  $\mathcal{F}_{\text{VSS}}^{\text{mult}}$  从  $\mathcal{P}^{A(0) \cdot B(0), t}$  中均匀随机选择  $C(x)$ , 然后向每个参与方  $P_j$  发送输出  $(A(\alpha_j), B(\alpha_j), C(\alpha_j))$ . 在真实世界中, 秘密分发者按照协议  $\Pi_{\text{VSS}}^{\text{mult}}$  的步骤 1(a) 到 1(d) 来选择  $D_1(x), \dots, D_t(x)$  和  $C(x)$ , 根据定义, 其满足  $C(0) = A(0) \cdot B(0)$ . 因为秘密分发者是诚实的, 所有投诉解决的结果都会是  $\tilde{C}'(\alpha_k) = \tilde{C}(\alpha_k)$ , 所以不会进入拒绝阶段, 故每个诚实方  $P_j$  都会输出  $C(\alpha_j)$ , 即所有诚实方都会输出常数项正确的  $C(x)$  的秘密份额。最后, 根据上文中的分析,  $C(x)$  的阶数不超过  $t$  且除了常数项外其他项的系数都是均匀随机的。综上所述, 理想世界和真实世界中诚实方的输出具有相同的分布。

**敌手的视图。** 下面证明: 给定诚实方的输入和输出, 敌手在真实世界和理想世界的视图分布相同。对于  $j \notin I$ , 首先固定其输入份额  $(A(\alpha_j), B(\alpha_j))$  和输出份额  $C(\alpha_j)$ . 这些份额唯一确定了  $t$  阶多项式  $A(x), B(x), C(x)$ . 敌手在真实执行中的视图包括从  $\mathcal{F}_{\text{VSS}}$  收到的份额

$$\left\{ D_1(\alpha_i) \right\}_{i \in I}, \dots, \left\{ D_t(\alpha_i) \right\}_{i \in I}, \left\{ C(\alpha_i) \right\}_{i \in I} \quad (9.12)$$

以及在投诉解决阶段收到的消息。在秘密分发者是诚实的情况下, 敌手在投诉解决阶段得到的消息只是式 (9.12) 的子集, 并且, 诚实方的输出已经确定了  $C(x)$ , 因此, 我们只需要证明: 在真实世界和理想世界中,  $\{D_1(\alpha_i), \dots, D_t(\alpha_i)\}_{i \in I}$  具有相同的分布。

我们将模拟器  $\mathcal{S}$  选择的多项式记为  $D_1^S(x), \dots, D_t^S(x)$ , 将真实协议中秘密分发者选择的多项式记为  $D_1(x), \dots, D_t(x)$ . 我们需要证明

$$\left\{ D_1^S(\alpha_i), \dots, D_t^S(\alpha_i) \mid A(x), B(x), C(x) \right\}_{i \in I} \stackrel{\text{perf}}{=} \left\{ D_1(\alpha_i), \dots, D_t(\alpha_i) \mid A(x), B(x), C(x) \right\}_{i \in I} \quad (9.13)$$

为此, 我们证明对于  $\ell = 1, \dots, t$ , 有

$$\begin{aligned} & \left\{ D_\ell^S(\alpha_i) \mid A(x), B(x), C(x), D_{\ell+1}^S(\alpha_i), \dots, D_t^S(\alpha_i) \right\}_{i \in I} \\ & \stackrel{\text{perf}}{=} \left\{ D_\ell(\alpha_i) \mid A(x), B(x), C(x), D_{\ell+1}(\alpha_i), \dots, D_t(\alpha_i) \right\}_{i \in I} \end{aligned} \quad (9.14)$$

综合上式取  $\ell = t$  到  $\ell = 1$  的情况可得式 (9.13)。

我们首先证明式 (9.14) 对于  $\ell > 1$  的情况都成立。令  $\ell \in \{2, \dots, t\}$ . 显然,  $\{D_\ell^S(\alpha_i)\}$  是均匀随机分布的, 与  $A(x), B(x), C(x), D_{\ell+1}^S(x), \dots, D_t^S(x)$  无关。但是, 真实协议中选取的  $D_\ell(x)$  似乎和

$A(x), B(x), C(x), D_{\ell+1}(x), \dots, D_t(x)$  存在依赖关系。下面我们证明  $\{D_\ell(\alpha_i)\}$  仍然是均匀随机值。注意到

$$D_\ell(x) = r_{\ell,0} + r_{\ell,1} \cdot x + \dots + r_{\ell,t-1} \cdot x^{t-1} + \left( d_{\ell+t} - \sum_{m=\ell+1}^t r_{m,t+\ell-m} \right) \cdot x^t$$

其中  $r_{\ell,0}, \dots, r_{\ell,t-1}$  都是均匀随机选取的，不会出现在  $A(x), B(x), D_{\ell+1}(x), \dots, D_t(x)$  的系数中（见表 9.1）。只有最后一个系数与其他多项式存在依赖关系。但是，根据引理 9.14， $D_\ell(x)$  是  $t$  阶多项式，除了最高次项的系数外其他系数都是均匀随机选择的，那么任意不超过  $t$  个点处的值  $\{D_\ell(\alpha_i)\}_{i \in I}$  是均匀随机值，与  $A(x), B(x), D_{\ell+1}(x), \dots, D_t(x)$  无关。然后，我们还要证明  $\{D_\ell(\alpha_i)\}_{i \in I}$  与  $C(x)$  无关。这是因为对于  $1 \leq m \leq t$ ， $C(x)$  的第  $m$  个系数包含随机值  $r_{1,m-1}$ ，这个值不出现在其他多项式中 ( $r_{1,m-1}$  只出现在  $D_1(x)$  中，见表 9.1)，且  $C(x)$  的常数项是  $A(0) \cdot B(0)$ 。故  $D_\ell(x)$  与  $C(x)$  无关。至此我们完成了式 (9.14) 在  $\ell > 1$  时的证明。

接下来证明式 (9.14) 对于  $\ell = 1$  也成立，即给定  $A(x), B(x), C(x)$  以及  $\{D_2(\alpha_i), \dots, D_t(\alpha_i)\}_{i \in I}$ ， $\{D_1^S(\alpha_i)\}_{i \in I}$  与  $\{D_1(\alpha_i)\}_{i \in I}$  有相同的分布。在理想世界中， $\{D_1^S(\alpha_i)\}_{i \in I}$  满足  $D_1^S(\alpha_i) = (\alpha_i)^{-1} \cdot (A(\alpha_i) \cdot B(\alpha_i) - C(\alpha_i) - \sum_{\ell=2}^t (\alpha_i)^\ell \cdot D_\ell(\alpha_i))$ ；在真实协议中， $D_1(\alpha_i)$  也满足该式（将  $C(x) = A(x) \cdot B(x) - \sum_{\ell=1}^t x^\ell \cdot D_\ell(x)$  变形可得）。也就是说， $\{D_1(\alpha_i)\}_{i \in I}$  是由  $A(x), B(x), C(x)$  以及  $\{D_2(\alpha_i), \dots, D_t(\alpha_i)\}_{i \in I}$  唯一确定的，故两者分布相同。综上，式 (9.14) 成立。

我们前面已经证明了诚实方的输出分布相同。因此，真实世界和理想世界完美不可区分。

**情况 2：秘密分发者是被攻陷的。**在这种情况下，模拟器  $\mathcal{S}$  可以向  $\mathcal{F}_{\text{VSS}}^{\text{mult}}$  发送多项式  $C(x)$ 。如果  $C(x)$  的阶数不超过  $t$  且常数项为  $A(0) \cdot B(0)$ ，那么  $C(x)$  决定了诚实方的份额；否则 ( $C(x)$  是不合法的)，所有诚实方都会输出  $A(0) \cdot B(0)$ 。

直观上看，因为所有投诉都通过调用  $\mathcal{F}_{\text{eval}}$  明确地解决了，协议具有正确性。在  $\{\mathcal{F}_{\text{VSS}}, \mathcal{F}_{\text{eval}}\}$ -混合模型下，敌手在投诉阶段外不会收到诚实方的消息；又因为敌手知道  $A(x), B(x)$  以及他分发的多项式  $C(x), D_1(x), \dots, D_t(x)$ ，他本来就知道哪个诚实方会广播投诉，以及调用  $\mathcal{F}_{\text{eval}}$  会输出什么结果。因此敌手也无法获得额外的信息。

形式化地，模拟器  $\mathcal{S}$  的工作方式如下。

1. 当诚实方向  $\mathcal{F}_{\text{VSS}}^{\text{mult}}$  发送输入后， $\mathcal{S}$  收到  $\mathcal{F}_{\text{VSS}}^{\text{mult}}$  发送的多项式  $(A(x), B(x))$  ( $\mathcal{F}_{\text{VSS}}^{\text{mult}}$  的步骤 4(a))。
2. 模拟  $\Pi_{\text{VSS}}^{\text{mult}}$  的步骤 1(e) 和 1(f)： $\mathcal{S}$  记录  $P_1$  向  $\mathcal{F}_{\text{VSS}}$  发送的输入  $C(x), D_1(x), \dots, D_t(x)$ 。
3. 如果  $\deg(C(x)) > t$  或存在  $1 \leq \ell \leq t$  满足  $\deg(D_\ell(x)) > t$ ，跳到步骤 7 (模拟拒绝阶段)。
4. 模拟  $\Pi_{\text{VSS}}^{\text{mult}}$  的步骤 2 和 3 (诚实方)：对于每个  $k \notin I$ ，如果  $C(\alpha_k) \neq A(\alpha_k) \cdot B(\alpha_k) - \sum_{\ell=1}^t (\alpha_k)^\ell \cdot D_\ell(\alpha_k)$ ， $\mathcal{S}$  模拟  $P_k$  广播 (complaint,  $k$ )。然后， $\mathcal{S}$  模拟投诉解决阶段： $\mathcal{S}$  使用多项式  $A(x), B(x), C(x), D_1(x), \dots, D_t(x)$  来模拟  $\mathcal{F}_{\text{eval}}^k$  的输出。如果存在  $k \notin I$  满足  $C(\alpha_k) \neq A(\alpha_k) \cdot B(\alpha_k) - \sum_{\ell=1}^t (\alpha_k)^\ell \cdot D_\ell(\alpha_k)$ ，跳到步骤 7。
5. 模拟  $\Pi_{\text{VSS}}^{\text{mult}}$  的步骤 2 和 3 (被攻陷方)：对于每个被攻陷方广播的 (complaint,  $k$ )， $k \in I$ ， $\mathcal{S}$  和步骤 4 中一样使用多项式  $A(x), B(x), C(x), D_1(x), \dots, D_t(x)$  来模拟  $\mathcal{F}_{\text{eval}}^k$  的输出。如果  $C(\alpha_k) \neq A(\alpha_k) \cdot B(\alpha_k) - \sum_{\ell=1}^t (\alpha_k)^\ell \cdot D_\ell(\alpha_k)$ ，跳到步骤 7。
6. 如果  $\mathcal{S}$  执行到了这里 (没有跳到步骤 7)，它代表  $P_1$  向  $\mathcal{F}_{\text{VSS}}^{\text{mult}}$  发送  $C(x)$  ( $\mathcal{F}_{\text{VSS}}^{\text{mult}}$  的步骤 4(b))。然后跳过步骤 7。

7. 模拟  $\Pi_{\text{VSS}}^{\text{mult}}$  的步骤 4:  $\mathcal{S}$  执行如下操作模拟拒绝阶段:

- (a) 代表  $P_1$  向  $\mathcal{F}_{\text{VSS}}^{\text{mult}}$  发送  $\hat{C}(x) = x^{t+1}$  ( $\mathcal{F}_{\text{VSS}}^{\text{mult}}$  的步骤 4(b)).
- (b) 模拟每个诚实方  $P_j$  广播  $a_j = A(\alpha_j)$  和  $b_j = B(\alpha_j)$ .

因为模拟器从  $\mathcal{F}_{\text{VSS}}^{\text{mult}}$  处收到了  $A(x), B(x)$ , 所以它可以模拟诚实方  $P_j$  广播秘密份额  $a_j = A(\alpha_j)$  和  $b_j = B(\alpha_j)$ . 显然, 敌手在理想世界和真实世界的视图是相同的。我们只需要证明诚实方的输出是一致的。分两种情况讨论。

1. 情况 1:  $\mathcal{S}$  没有模拟拒绝阶段 (没有运行步骤 7)。此时, 以下条件成立

- (a) 多项式  $C(x), D_1(x), \dots, D_t(x)$  不超过  $t$  阶;
- (b) 对于  $j \notin I$  都有  $C(\alpha_j) = A(\alpha_j) \cdot B(\alpha_j) - \sum_{\ell=1}^t (\alpha_j)^\ell \cdot D_\ell(\alpha_j)$ ;
- (c) 如果某个被攻陷方  $P_i$  广播了  $(\text{complaint}, i)$ , 有  $C(\alpha_i) = A(\alpha_i) \cdot B(\alpha_i) - \sum_{\ell=1}^t (\alpha_i)^\ell \cdot D_\ell(\alpha_i)$ .

因为  $\mathcal{S}$  获得的多项式就是被攻陷方向  $\mathcal{F}_{\text{VSS}}$  发送的输入, 所以在真实世界中诚实方  $P_j$  显然会输出  $C(\alpha_j)$ . 在理想世界中, 如果  $\deg(C(x)) \leq t$  且  $C(0) = A(0) \cdot B(0)$ , 那么诚实方  $P_j$  就会输出  $C(\alpha_j)$ . 我们证明  $C(x)$  满足这两个条件。首先,  $\deg(C(x)) \leq t$  已经满足了。令  $C'(x) = A(x) \cdot B(x) - \sum_{\ell=1}^t x^\ell \cdot D_\ell(x)$ . 因为  $A(x), B(x), D_\ell(x)$  都不超过  $t$  阶, 故  $C'(x)$  不超过  $2t$  阶。又因为  $C(\alpha_j) = A(\alpha_j) \cdot B(\alpha_j) - \sum_{\ell=1}^t (\alpha_j)^\ell \cdot D_\ell(\alpha_j)$  对  $j \notin I$  都成立, 所以  $C'(x)$  与  $C(x)$  相交于至少  $2t+1$  个点, 由此可知  $C(x) = C'(x)$ , 因而  $C(0) = C'(0) = A(0) \cdot B(0)$ . 综上所述, 理想世界中诚实方  $P_j$  也会输出  $C(\alpha_j)$ .

2. 情况 2:  $\mathcal{S}$  模拟了拒绝阶段 (运行了步骤 7)。此时, 条件 (a)(b)(c) 至少有一个不满足, 而在真实协议中, 出现这样的情况时, 所有诚实方都会执行拒绝阶段并输出  $A(0) \cdot B(0)$ . 在理想世界中, 此时  $\mathcal{S}$  发送了大于  $t$  阶的多项式  $\hat{C}(x) = x^{t+1}$  给  $\mathcal{F}_{\text{VSS}}^{\text{mult}}$ , 故理想世界中的诚实方也会输出  $A(0) \cdot B(0)$ .

综上所述, 我们构造了模拟器  $\mathcal{S}$  在秘密分发者诚实或被攻陷时, 都使真实世界和理想世界不可区分。定理证毕。  $\square$

#### 9.3.4.6 实现乘法理想功能 $\mathcal{F}_{\text{mult}}$

在实现了  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  和  $\mathcal{F}_{\text{VSS}}^{\text{mult}}$  之后, 我们终于可以对于恶意敌手实现乘法理想功能  $\mathcal{F}_{\text{mult}}$ . 整体思路正如第9.3.1节中所讲, 参与方首先使用  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  分发自己的输入份额的子份额, 然后使用  $\mathcal{F}_{\text{VSS}}^{\text{mult}}$  分发输入份额乘积的子份额, 最后, 参与方在本地对子份额进行线性重组即可。

定义理想功能。我们首先在恶意敌手场景下定义理想功能。在半诚实场景下,  $\mathcal{F}_{\text{mult}}$  的函数表示如下:

$$\mathcal{F}_{\text{mult}} \left( (f_a(\alpha_1), f_b(\alpha_1)), \dots, (f_a(\alpha_n), f_b(\alpha_n)) \right) = \left( f_{ab}(\alpha_1), \dots, f_{ab}(\alpha_n) \right)$$

其中  $f_{ab}$  是常数项为  $a \cdot b$  的均匀随机  $t$  阶多项式。

在恶意敌手场景下, 理想功能  $\mathcal{F}_{\text{mult}}$  的定义需要更加小心。在协议中, 每个参与方需要作为秘密分发者调用  $\mathcal{F}_{\text{VSS}}^{\text{mult}}$ , 然后参与方在本地计算子份额的线性组合。当被攻陷方作为秘密分发者时, 它可以选择使用  $\mathcal{F}_{\text{VSS}}^{\text{mult}}$  分享的多项式  $C(x)$ . 我们考虑 rushing adversary, 他首先收到诚实方分发的子份额, 然后再选择要分享的多项式  $C(x)$ , 因此, 敌手可以控制被攻陷方在输出多项式  $f_{ab}$  中持有的份额。不过,

此时并不存在  $f_{ab}$  的秘密分发者，因此敌手只能决定被攻陷方自己的份额，而不能决定多项式  $f_{ab}$ . 理想功能  $\mathcal{F}_{\text{mult}}$  的定义如图 9.25 所示。

**乘法理想功能  $\mathcal{F}_{\text{mult}}$**

$\mathcal{F}_{\text{mult}}$  接收被攻陷方的集合  $I \subset [n]$ , 并执行如下操作:

1. 对于每个诚实方  $P_j, j \notin I$ ,  $\mathcal{F}_{\text{VSS}}^{\text{mult}}$  接收其输入  $(\beta_j, \gamma_j)$ . 令  $f_a(x)$  和  $f_b(x)$  分别是由点集  $\{(\alpha_j, \beta_j)\}_{j \notin I}$  和  $\{(\alpha_j, \gamma_j)\}_{j \notin I}$  唯一确定的  $t$  阶多项式。
2.  $\mathcal{F}_{\text{mult}}$  向敌手  $\mathcal{S}$  发送  $\{(f_a(\alpha_i), f_b(\alpha_i))\}_{i \in I}^a$ .
3.  $\mathcal{F}_{\text{mult}}$  从敌手  $\mathcal{S}$  处接收  $\{\delta_i\}_{i \in I}$  (如果某些值没有收到, 将其设置为 0.)
4.  $\mathcal{F}_{\text{mult}}$  基于以下约束条件均匀随机选择  $t$  阶多项式  $f_{ab}(x)$ :
  - (a)  $f_{ab}(0) = f_a(0) \cdot f_b(0)$ ;
  - (b)  $f_{ab}(\alpha_i) = \delta_i$  对  $i \in I$  都成立。

(当  $|I| \leq t$  时, 这样的多项式一定存在。)

5.  $\mathcal{F}_{\text{mult}}$  向  $P_j, j \in [n]$  发送输出  $f_{ab}(\alpha_j)$ .

<sup>a</sup>与  $\mathcal{F}_{\text{eval}}$  和  $\mathcal{F}_{\text{VSS}}^{\text{mult}}$  的情况类似, 敌手本来就知道  $\{(f_a(\alpha_i), f_b(\alpha_i))\}_{i \in I}$ , 这只是为了让模拟器能够模拟。

图 9.25: 乘法理想功能  $\mathcal{F}_{\text{mult}}$

尽管  $\mathcal{F}_{\text{mult}}$  的定义与半诚实的情况略有不同, 但这个定义也足以实现安全的电路计算: 这里敌手仅仅增加了决定被攻陷方的份额的能力, 由于  $f_{ab}$  是  $t$  阶多项式, 敌手决定  $f_{ab}$  的  $t$  个值不会泄露  $f_{ab}(0) = a \cdot b$  的任何信息。

**协议构造。**接下来, 我们在  $\{\mathcal{F}_{\text{VSS}}^{\text{subshare}}, \mathcal{F}_{\text{VSS}}^{\text{mult}}\}$  混合模型下构造乘法协议。参与方持有两条输入导线的秘密份额, 每个参与方首先通过  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  和  $\mathcal{F}_{\text{VSS}}^{\text{mult}}$  来分发秘密份额乘积的子份额, 然后将这些子份额在本地线性重组, 这与半诚实场景下的协议思路类似。具体而言, 令  $f_a(x)$  和  $f_b(x)$  是两个  $t$  阶多项式, 满足  $f_a(0) = a, f_b(0) = b$ . 令  $h(x) = f_a(x) \cdot f_b(x) = a \cdot b + h_1 x + h_2 x^2 + \cdots + h^{2t} x^{2t}$ , 令  $V_{\vec{\alpha}}$  是  $\vec{\alpha}$  的 Vandermonde 矩阵。因为 Vandermonde 矩阵是可逆的, 我们有

$$V_{\vec{\alpha}} \cdot \begin{pmatrix} ab \\ h_1 \\ \vdots \\ h_{2t} \\ 0 \\ \vdots \\ 0 \end{pmatrix} = \begin{pmatrix} h(\alpha_1) \\ h(\alpha_2) \\ \vdots \\ h(\alpha_n) \end{pmatrix}, \text{ 因而 } V_{\vec{\alpha}}^{-1} \cdot \begin{pmatrix} ab \\ h_1 \\ \vdots \\ h_{2t} \\ 0 \\ \vdots \\ 0 \end{pmatrix} = V_{\vec{\alpha}}^{-1} \cdot \begin{pmatrix} h(\alpha_1) \\ h(\alpha_2) \\ \vdots \\ h(\alpha_n) \end{pmatrix}$$

令  $\lambda_1, \dots, \lambda_n$  是  $V_{\vec{\alpha}}^{-1}$  的第 1 行, 我们有

$$a \cdot b = \lambda_1 \cdot h(\alpha_1) + \cdots + \lambda_n \cdot h(\alpha_n) = \lambda_1 \cdot f_a(\alpha_1) \cdot f_b(\alpha_1) + \cdots + \lambda_n \cdot f_a(\alpha_n) \cdot f_b(\alpha_n)$$

因此  $a \cdot b$  是  $f_a(\alpha_\ell) \cdot f_b(\alpha_\ell)$ ,  $\ell = 1, \dots, n$  的线性组合。参与方首先通过  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  和  $\mathcal{F}_{\text{VSS}}^{\text{mult}}$  分发  $f_a(\alpha_\ell) \cdot f_b(\alpha_\ell)$  的秘密份额，具体而言，令  $C_1(x), \dots, C_n(x)$  是满足  $C_\ell(0) = f_a(\alpha_\ell) \cdot f_b(\alpha_\ell)$ ,  $\ell = 1, \dots, n$  的随机  $t$  阶多项式，其中  $C_\ell(x)$  是  $P_\ell$  作为秘密分发者通过  $\mathcal{F}_{\text{VSS}}^{\text{mult}}$  分享的多项式。分享完成后，每个参与方  $P_i$  持有  $C_1(\alpha_i), \dots, C_n(\alpha_i)$ 。于是，每个参与方可以本地计算  $Q(\alpha_i) = \sum_{\ell=1}^n \lambda_\ell \cdot C_\ell(\alpha_i)$ ，从而参与方持有了多项式  $Q(x) = \sum_{\ell=1}^n \lambda_\ell \cdot C_\ell(x)$  的秘密份额。因为  $C_\ell(0) = f_a(\alpha_\ell) \cdot f_b(\alpha_\ell)$ ，故

$$Q(0) = \sum_{\ell=1}^n \lambda_\ell \cdot C_\ell(0) = \sum_{\ell=1}^n \lambda_\ell \cdot f_a(\alpha_\ell) \cdot f_b(\alpha_\ell) = a \cdot b$$

此外，因为  $C_\ell(x)$ ,  $\ell = 1, \dots, n$  都是  $t$  阶多项式，所有  $Q(x)$  也是  $t$  阶多项式，即参与方持有了  $a \cdot b$  合法的秘密份额。完整的协议描述如图 9.26 所示。

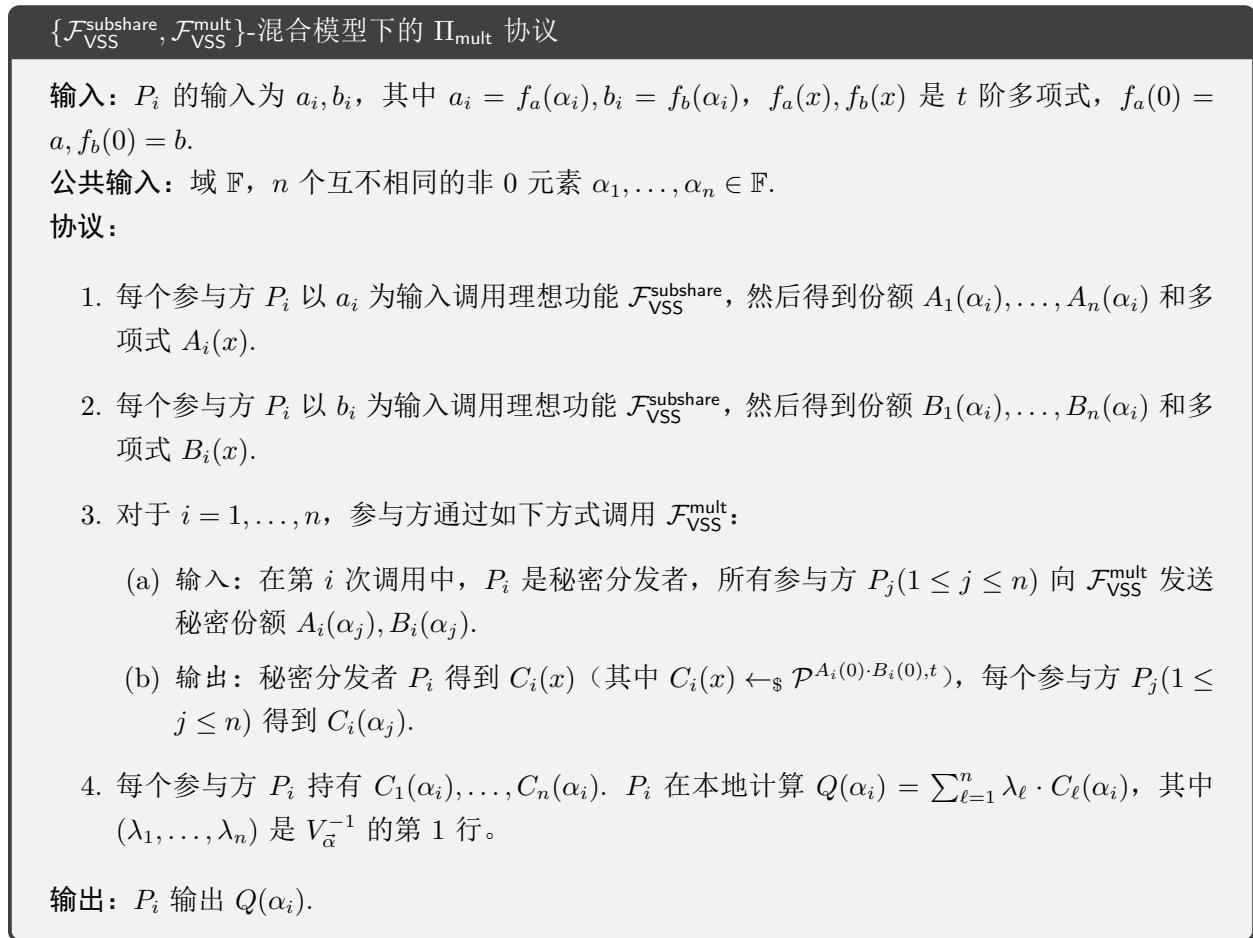


图 9.26:  $\{\mathcal{F}_{\text{VSS}}^{\text{subshare}}, \mathcal{F}_{\text{VSS}}^{\text{mult}}\}$ -混合模型下的  $\Pi_{\text{mult}}$  协议

直观上看， $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  和  $\mathcal{F}_{\text{VSS}}^{\text{mult}}$  确保了参与方能获得正确的秘密份额且不泄露额外信息。不过，请注意，因为  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  会向敌手泄露向量  $\vec{Y}(x) = (g_1(x), \dots, g_n(x)) \cdot H^T$ ，其中  $g_1, \dots, g_n$  是定义了参与方的输入份额的多项式， $H$  是 Reed-Solomon 码的校验矩阵，所以在协议  $\Pi_{\text{mult}}$  中敌手会获得

$\vec{Y}_A(x) = (A_1(x), \dots, A_n(x)) \cdot H^T$  和  $\vec{Y}_B(x) = (B_1(x), \dots, B_n(x)) \cdot H^T$ . 模拟器需要模拟这两条消息。这与  $\mathcal{F}_{\text{eval}}$  的情形类似，模拟器能够进行模拟，因为  $\vec{Y}_A(x)$  和  $\vec{Y}_B(x)$  是在约束条件下的随机多项式向量。

我们有以下定理。

**定理 9.20.** 假设被攻陷方的数量不超过  $t$ ,  $t < n/3$ . 图 9.26 中的协议  $\Pi_{\text{mult}}$  在  $\{\mathcal{F}_{\text{VSS}}^{\text{subshare}}, \mathcal{F}_{\text{VSS}}^{\text{mult}}\}$ -混合模型下对于静态恶意对手  $UC$ -安全实现了理想功能  $\mathcal{F}_{\text{mult}}$  (图 9.25)。

证明. 要证明此定理，我们需要构造模拟器  $\mathcal{S}$ ，使得对所有环境  $\mathcal{Z}$ ，都有

$$\text{EXEC}_{\Pi_{\text{mult}}, \mathcal{A}, \mathcal{Z}}^{\mathcal{F}_{\text{VSS}}^{\text{subshare}}, \mathcal{F}_{\text{VSS}}^{\text{mult}}} \approx \text{EXEC}_{\mathcal{F}_{\text{mult}}, \mathcal{S}, \mathcal{Z}}$$

其中  $\mathcal{A}$  是平凡敌手。

**模拟器  $\mathcal{S}$ .** 模拟器  $\mathcal{S}$  在内部运行  $\mathcal{A}$ ，转发  $\mathcal{Z}$  与  $\mathcal{A}$  之间的消息。在  $\{\mathcal{F}_{\text{VSS}}^{\text{subshare}}, \mathcal{F}_{\text{VSS}}^{\text{mult}}\}$ -混合模型下， $\mathcal{S}$  还需要模拟  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  和  $\mathcal{F}_{\text{VSS}}^{\text{mult}}$ .  $\mathcal{S}$  的工作方式如下。

1. 当诚实方向  $\mathcal{F}_{\text{mult}}$  提供输入后， $\mathcal{S}$  接收  $\mathcal{F}_{\text{mult}}$  发送的值  $\{(f_a(\alpha_i), f_b(\alpha_i))\}_{i \in I}$  ( $\mathcal{F}_{\text{mult}}$  的步骤 2)。
  2. 模拟  $\Pi_{\text{mult}}$  的步骤 1:  $\mathcal{S}$  通过如下方式模拟对  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  的第一次调用:
    - (a) 对于  $j \notin I$ ,  $\mathcal{S}$  均匀随机选择多项式  $A_j(x) \leftarrow_{\$} \mathcal{P}^{0,t}$ .
    - (b) 模拟  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  的步骤 3:  $\mathcal{S}$  模拟  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  向敌手  $\mathcal{A}$  发送  $\{A_j(\alpha_i)\}_{j \notin I, i \in I}$ .
    - (c) 模拟  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  的步骤 4:  $\mathcal{S}$  记录敌手  $\mathcal{A}$  向  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  发送的多项式  $\{A_i(x)\}_{i \in I}$ . 如果某些多项式没有收到，将其设置为常数多项式 0.
    - (d) 模拟  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  的步骤 5: 对于  $i \in I$ ,  $\mathcal{S}$  执行以下检查:
      - i.  $A_i(0) \stackrel{?}{=} f_a(\alpha_i)$ ;
      - ii.  $A_i(x)$  的阶数不超过  $t$ .
 如果检查都通过，那么设置  $A'_i(x) = A_i(x)$ ; 否则，设置  $A'_i(x) = f_a(\alpha_i)$  ( $\mathcal{S}$  在步骤 1 中从  $\mathcal{F}_{\text{mult}}$  接收到了这个值)。
    - 对于  $j \notin I$ ,  $\mathcal{S}$  设置  $A'_j(x) = A_j(x)$ .
  - (e) 模拟  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  的步骤 6(a): 首先， $\mathcal{S}$  按照如下方式计算错误向量  $e^A = (e_1^A, \dots, e_n^A)$ : 对于  $j \notin I$ , 设置  $e_j^A = 0$ ; 对于  $i \in I$ , 设置  $e_i^A = A_i(0) - f_a(\alpha_i)$ . 然后， $\mathcal{S}$  基于以下两个约束条件
    - (a)  $\vec{Y}_A(0) = (e_1^A, \dots, e_n^A) \cdot H^T$ ,
    - (b)  $\vec{Y}_A(\alpha_i) = (A_1(\alpha_i), \dots, A_n(\alpha_i)) \cdot H^T$ ,  $i \in I$ ,
 均匀随机选择多项式向量  $\vec{Y}_A(x)$ .  $\mathcal{S}$  模拟  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  向敌手  $\mathcal{A}$  发送  $\vec{Y}_A(x)$ .
  - (f) 模拟  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  的步骤 6(b): 对于  $i \in I$ ,  $\mathcal{S}$  模拟  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  向  $P_i$  发送输出  $A'_i(x)$  和  $A'_1(\alpha_i), \dots, A'_n(\alpha_i)$ .
3. 模拟  $\Pi_{\text{mult}}$  的步骤 2:  $\mathcal{S}$  模拟对  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  的第二次调用: 模拟的方式与步骤 2 完全相同，只是使用的点集为  $\{f_b(\alpha_i)\}_{i \in I}$ . 将  $\mathcal{S}$  在模拟中使用的多项式记为  $B_1(x), \dots, B_n(x)$  和  $B'_1(x), \dots, B'_n(x)$ , 将  $\mathcal{S}$  模拟  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  向敌手或被攻陷方发送的值记为  $\{B'_i(x)\}_{i \in I}$ ,  $\{B'_1(\alpha_i), \dots, B'_n(\alpha_i)\}_{i \in I}$  和  $\vec{Y}_B(x)$ . 此时， $\mathcal{S}$  持有  $t$  阶多项式集合  $\{A'_\ell(x), B'_\ell(x)\}_{\ell \in [n]}$ , 其中，对于  $j \notin I$  有  $A'_j(0) = B'_j(0) = 0$ , 对于  $i \in I$  有  $A'_i(0) = f_a(\alpha_i)$ ,  $B'_i(0) = f_b(\alpha_i)$ .
  4. 模拟  $\Pi_{\text{mult}}$  的步骤 3: 对于  $j \notin I$ ,  $\mathcal{S}$  模拟诚实方  $P_j$  作为秘密分发者对  $\mathcal{F}_{\text{VSS}}^{\text{mult}}$  的调用:

- (a)  $\mathcal{S}$  均匀随机选择多项式  $C'_j(x) \leftarrow_{\$} \mathcal{P}^{0,t}$ .
- (b)  $\mathcal{S}$  模拟  $\mathcal{F}_{\text{VSS}}^{\text{mult}}$  向敌手  $\mathcal{A}$  发送份额  $\{(A'_j(\alpha_i), B'_j(\alpha_i), C'_j(\alpha_i))\}_{i \in I}$  ( $\mathcal{F}_{\text{VSS}}^{\text{mult}}$  的步骤 3(c))。

5. (继续) 模拟  $\Pi_{\text{mult}}$  的步骤 3: 对于  $i \in I$ ,  $\mathcal{S}$  模拟被攻陷方  $P_i$  作为秘密分发者对  $\mathcal{F}_{\text{VSS}}^{\text{mult}}$  的调用:

- (a)  $\mathcal{S}$  模拟  $\mathcal{F}_{\text{VSS}}^{\text{mult}}$  向敌手  $\mathcal{A}$  发送多项式  $(A'_i(x), B'_i(x))$  ( $\mathcal{F}_{\text{VSS}}^{\text{mult}}$  的步骤 4(a)).
- (b)  $\mathcal{S}$  模拟  $\mathcal{F}_{\text{VSS}}^{\text{mult}}$  接收  $P_i$  发送的多项式  $C_i(x)$  ( $\mathcal{F}_{\text{VSS}}^{\text{mult}}$  的步骤 4(b)), 然后执行如下操作:
  - i. 如果  $\deg(C_i(x)) \leq t$  且  $C_i(0) = A'_i(0) \cdot B'_i(0) = f_a(\alpha_i) \cdot f_b(\alpha_i)$ , 那么  $\mathcal{S}$  设置  $C'_i(x) = C_i(x)$ .
  - ii. 否则, 设置  $C'_i(x) = f_a(\alpha_i) \cdot f_b(\alpha_i)$ .

此时,  $\mathcal{S}$  持有  $t$  阶多项式  $C'_1(x), \dots, C'_n(x)$ , 其中, 对于  $j \notin I$  有  $C'_j(0) = 0$ , 对于  $i \in I$  有  $C'_i(0) = f_a(\alpha_i) \cdot f_b(\alpha_i)$ .

6. 对于  $i \in I$ ,  $\mathcal{S}$  计算  $Q(\alpha_i) = \sum_{\ell=1}^n \lambda_\ell \cdot C'_\ell(\alpha_i)$ , 其中  $C'_1(x), \dots, C'_n(x)$  是上述定义的多项式。然后,  $\mathcal{S}$  向  $\mathcal{F}_{\text{mult}}$  发送  $\{Q(\alpha_i)\}_{i \in I}$  (这是  $\mathcal{F}_{\text{mult}}$  步骤 3 中的  $\{\delta_i\}_{i \in I}$ )。

**不可区分性。**真实世界与理想世界的不同之处如下: (1)对于  $j \notin I$ ,  $\mathcal{S}$  选择的多项式  $A'_j(x), B'_j(x), C'_j(x)$  的常数项为 0, 而真实协议中它们的常数项分别为  $f_a(\alpha_j), f_b(\alpha_j), f_a(\alpha_j) \cdot f_b(\alpha_j)$ ; (2)理想世界中的  $\vec{Y}_A(x)$  和  $\vec{Y}_B(x)$  是  $\mathcal{S}$  根据错误向量计算的, 而真实协议中它们是由  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  根据实际的多项式  $A_1(x), \dots, A_n(x)$  和  $B_1(x), \dots, B_n(x)$  来计算的; (3)在理想世界中,  $P_i$  的输出是  $f_{ab}(\alpha_i)$ , 其中  $f_{ab}(x)$  是  $\mathcal{F}_{\text{mult}}$  在约束条件  $f_{ab}(0) = f_a(0) \cdot f_b(0)$  和  $f_{ab}(\alpha_i) = \delta_i$ ,  $i \in I$  下均匀随机选择的, 而真实协议中的输出份额是由多项式  $Q(x) = \sum_{\ell=1}^n \lambda_\ell \cdot C'_\ell(x)$  决定的。我们需要通过一系列混合实验来证明真实世界与理想世界不可区分。

**混合实验  $\mathcal{H}_0$ :** 这是理想世界的执行  $\text{EXEC}_{\mathcal{F}_{\text{mult}}, \mathcal{S}, \mathcal{Z}}$ .

**混合实验  $\mathcal{H}_1$ :**  $\mathcal{H}_1$  与  $\mathcal{H}_0$  相同, 除了模拟器  $\mathcal{S}$  为诚实方  $P_j$  选取多项式的方式变为  $A_j(x) \leftarrow_{\$} \mathcal{P}^{f_a(\alpha_j), t}, B_j(x) \leftarrow_{\$} \mathcal{P}^{f_b(\alpha_j), t}, C'_j(x) \leftarrow_{\$} \mathcal{P}^{f_a(\alpha_j) \cdot f_b(\alpha_j), t}$  而不是  $A_j(x) \leftarrow_{\$} \mathcal{P}^{0,t}, B_j(x) \leftarrow_{\$} \mathcal{P}^{0,t}, C'_j(x) \leftarrow_{\$} \mathcal{P}^{0,t}$ .

**断言 1:** 如果  $|I| \leq t$ , 那么  $\mathcal{H}_1$  与  $\mathcal{H}_0$  完美不可区分。

**证明.**  $\mathcal{H}_1$  中的修改仅会影响敌手收到的份额  $\{(A'_j(\alpha_i), B'_j(\alpha_i), C'_j(\alpha_i))\}_{j \notin I, i \in I}$  (模拟器  $\mathcal{S}$  的步骤 4(b))。由于  $|I| \leq t$ , 根据推论 9.13,  $\mathcal{H}_1$  与  $\mathcal{H}_0$  中的敌手得到的秘密份额  $\{(A'_j(\alpha_i), B'_j(\alpha_i), C'_j(\alpha_i))\}_{j \notin I, i \in I}$  具有相同的分布, 因此  $\mathcal{H}_1$  与  $\mathcal{H}_0$  完美不可区分。  $\square$

**混合实验  $\mathcal{H}_2$ :**  $\mathcal{H}_2$  与  $\mathcal{H}_1$  相同, 除了  $\vec{Y}_A(x)$  与  $\vec{Y}_B(x)$  的计算方式改用真实世界中  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$  的计算方式, 即基于多项式  $A_1(x), \dots, A_n(x)$  和  $B_1(x), \dots, B_n(x)$  计算 (注意在  $\mathcal{H}_1$  中, 诚实方多项式的选取方式已经变为  $A_j(x) \leftarrow_{\$} \mathcal{P}^{f_a(\alpha_j), t}, B_j(x) \leftarrow_{\$} \mathcal{P}^{f_b(\alpha_j), t}$ )。

**断言 2:** 如果  $|I| \leq t$ , 那么  $\mathcal{H}_2$  与  $\mathcal{H}_1$  完美不可区分。

**证明.** 这与定理 9.18 中的断言 2 完全同理。  $\square$

**混合实验  $\mathcal{H}_3$ :** 这是真实世界的执行  $\text{EXEC}_{\Pi_{\text{mult}}, \mathcal{A}, \mathcal{Z}}^{\mathcal{F}_{\text{VSS}}^{\text{subshare}}, \mathcal{F}_{\text{VSS}}^{\text{mult}}}$ .

**断言 3:** 如果  $|I| \leq t$ , 那么  $\mathcal{H}_3$  与  $\mathcal{H}_2$  完美不可区分。

证明.  $\mathcal{H}_3$  与  $\mathcal{H}_2$  唯一的区别在于定义输出份额的多项式的选择方式: 在  $\mathcal{H}_2$  中,  $f_{ab}(x)$  由理想功能  $\mathcal{F}_{\text{mult}}$  在约束条件  $f_{ab}(0) = f_a(0) \cdot f_b(0)$  和  $f_{ab}(\alpha_i) = \delta_i, i \in I$  下选择的; 在  $\mathcal{H}_3$  中, 定义输出份额的多项式为  $Q(x) = \sum_{\ell=1}^n \lambda_\ell \cdot C'_\ell(x)$ . 根据模拟器  $\mathcal{S}$  的定义, 我们有  $\delta_i = Q(\alpha_i) = \sum_{\ell=1}^n \lambda_\ell \cdot C'_\ell(\alpha_i)$ , 其中  $C'_1(x), \dots, C'_n(x)$  的常数项都是正确的。于是, 我们只需要证明以下两个分布是相同的:

- 理想世界: 在约束条件  $f_{ab}(0) = f_a(0) \cdot f_b(0)$  和  $f_{ab}(\alpha_i) = Q(\alpha_i) = \sum_{\ell=1}^n \lambda_\ell \cdot C'_\ell(\alpha_i), i \in I$  下均匀随机选择  $t$  阶多项式  $f_{ab}(x)$ .
- 真实世界: 计算  $f_{ab}(x) = Q(x) = \sum_{\ell=1}^n \lambda_\ell \cdot C'_\ell(x)$ .

注意, 在这两种情况下, 多项式  $C'_1(x), \dots, C'_n(x)$  的分布是相同的。

首先, 当  $|I| = t$  时, 理想世界中的约束条件已经唯一确定了  $f_{ab}(x)$ , 故它与真实世界的多项式一定相同。

如果  $|I| < t$ , 在理想世界中,  $f_{ab}(x)$  的选择方式等价于选择  $t - |I|$  个均匀随机值  $\beta_\ell \leftarrow_{\$} \mathbb{F} (\ell \notin I)$ , 然后令  $f_{ab}(x)$  是经过  $(\alpha_\ell, \beta_\ell)$  的唯一多项式。我们观察真实世界中的  $f_{ab}(x)$ . 给定任意  $j \notin I$ , 根据协议  $\Pi_{\text{mult}}$  的定义,  $C'_j(x)$  是在约束条件  $C'_j(0) = f_a(\alpha_j) \cdot f_b(\alpha_j)$  下的均匀随机  $t$  阶多项式。根据引理 9.11, 给定点集  $\{(\alpha_i, C'_j(\alpha_i))\}_{i \in I}$  和“秘密”  $s = C'_j(0)$ , 任意  $t - |I|$  个位置处的值  $\{C'_j(\alpha_\ell)\}_{\ell \notin I}$  都是  $\mathbb{F}$  上的均匀随机分布 (注意, 敌手没有看到这些值)。这意味着, 在真实世界中, 任意  $t - |I|$  个位置  $\alpha_\ell (\ell \notin I)$  处的值  $f_{ab}(\alpha_\ell)$  也是均匀随机的。因此, 它等价于选择  $t - |I|$  个均匀随机值  $\beta_\ell \leftarrow_{\$} \mathbb{F} (\ell \notin I)$ , 然后令  $f_{ab}(x)$  是满足  $f_{ab}(\alpha_\ell) = \beta_\ell$  的唯一多项式。综上, 理想世界与真实世界中的  $f_{ab}(x)$  分布相同。  $\square$

定理证毕。  $\square$

**小结。** 我们对本节中的定理作一个小结:

- 定理 9.15 (在朴素模型 (plain model) 下 UC-安全实现  $\mathcal{F}_{\text{VSS}}$ )
- 定理 9.16 (在  $\mathcal{F}_{\text{VSS}}^n$ -混合模型下 UC-安全实现  $\mathcal{F}_{\text{mat}}^A$ )
- 定理 9.17 (在  $\mathcal{F}_{\text{mat}}^A$ -混合模型下 UC-安全实现  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$ )
- 定理 9.18 (在  $\mathcal{F}_{\text{VSS}}^{\text{subshare}}$ -混合模型下 UC-安全实现  $\mathcal{F}_{\text{eval}}$ )
- 定理 9.19 (在  $\{\mathcal{F}_{\text{VSS}}, \mathcal{F}_{\text{eval}}\}$ -混合模型下 UC-安全实现  $\mathcal{F}_{\text{VSS}}^{\text{mult}}$ )
- 定理 9.20 (在  $\{\mathcal{F}_{\text{VSS}}^{\text{subshare}}, \mathcal{F}_{\text{VSS}}^{\text{mult}}\}$ -混合模型下 UC-安全实现  $\mathcal{F}_{\text{mult}}$ )

根据通用组合定理 (定理 4.3), 我们有以下结论。

**定理 9.21.** 假设参与方之间存在点对点安全信道、认证的广播信道。假设被攻陷方的数量不超过  $t$ ,  $t < n/3$ . 那么, 在朴素模型 (plain model) 下存在协议对于静态恶意敌手 UC-安全实现了理想功能  $\mathcal{F}_{\text{mult}}$ .

### 9.3.5 $\{\mathcal{F}_{\text{VSS}}, \mathcal{F}_{\text{mult}}\}$ -混合模型下的安全计算协议

现在，我们可以给出  $\{\mathcal{F}_{\text{VSS}}, \mathcal{F}_{\text{mult}}\}$ -混合模型下的安全计算协议，它对于控制了  $t < n/3$  个被攻陷方的静态恶意敌手 UC-安全实现了函数计算功能。协议的思路与半诚实的情况非常类似，区别仅在于：(1) 在输入分享阶段，参与方调用理想功能  $\mathcal{F}_{\text{VSS}}$  进行秘密分享；(2) 对于乘法门，参与方调用理想功能  $\mathcal{F}_{\text{mult}}$  进行计算。 $\mathcal{F}_{\text{VSS}}$  确保了恶意的被攻陷方也执行了正确的 Shamir 秘密分享，而  $\mathcal{F}_{\text{mult}}$  确保了诚实方能获得乘法门的输出导线的正确份额。此外，根据  $\mathcal{F}_{\text{mult}}$  的定义，它向敌手泄漏了被攻陷方在乘法门的输入导线上的份额，但是在协议中，敌手原本就知道这些信息，故  $\mathcal{F}_{\text{mult}}$  不会造成任何额外的信息泄漏。最后，在输出重建阶段，对于每个输出值，参与方将秘密份额发送给输出值的接收方。此时，恶意的参与方可能会发送错误的份额（在半诚实场景下这不会发生），但是因为被攻陷方数量  $t < n/3$ ，参与方可以调用 Reed-Solomon 解码算法重建正确的输出。

协议的形式化描述如图 9.27 所示。

我们有以下定理。

**定理 9.22.** 假设参与方之间存在点对点安全信道。假设被攻陷方的数量不超过  $t$ ,  $t < n/3$ . 图 9.27 中的协议  $\Pi_{\text{sfe}}$  在  $\{\mathcal{F}_{\text{VSS}}, \mathcal{F}_{\text{mult}}\}$ -混合模型下对于静态恶意敌手 UC-安全实现了理想功能  $\mathcal{F}_{\text{sfe}}$  (图 6.1)。

证明. 要证明此定理，我们需要构造模拟器  $\mathcal{S}$ ，使得对所有环境  $\mathcal{Z}$ ，都有

$$\text{EXEC}_{\Pi_{\text{sfe}}, \mathcal{A}, \mathcal{Z}}^{\mathcal{F}_{\text{VSS}}, \mathcal{F}_{\text{mult}}} \approx \text{EXEC}_{\mathcal{F}_{\text{sfe}}, \mathcal{S}, \mathcal{Z}}$$

其中  $\mathcal{A}$  是平凡敌手。

**模拟器  $\mathcal{S}$ .** 模拟器  $\mathcal{S}$  在内部运行  $\mathcal{A}$ ，转发  $\mathcal{Z}$  与  $\mathcal{A}$  之间的消息。在  $\{\mathcal{F}_{\text{VSS}}, \mathcal{F}_{\text{mult}}\}$ -混合模型下， $\mathcal{S}$  还需要模拟  $\mathcal{F}_{\text{VSS}}$  和  $\mathcal{F}_{\text{mult}}$ .  $\mathcal{S}$  的工作方式如下。

- 首先， $\mathcal{S}$  选择大小为  $t$  的集合  $\hat{I} \supseteq I$ .
- 输入分享阶段：
  1. 对于  $j \notin I$ ,  $\mathcal{S}$  均匀随机选择常数项为 0 的  $t$  阶多项式  $q_j(x) \leftarrow_{\$} \mathcal{P}^{0,t}$ . 对于  $i \in I$ ,  $\mathcal{S}$  模拟  $\mathcal{F}_{\text{VSS}}$  向被攻陷方  $P_i$  发送份额  $q_j(\alpha_i)$ . 对于  $i \in \hat{I} \setminus I$ ,  $\mathcal{S}$  记录其秘密份额  $q_j(\alpha_i)$ .
  2. 对于  $i \in I$ ,  $\mathcal{S}$  记录  $P_i$  作为秘密分发者向  $\mathcal{F}_{\text{VSS}}$  发送的多项式  $q_i(x)$ . 如果  $\deg(q_i(x)) \leq t$ ,  $\mathcal{S}$  模拟  $\mathcal{F}_{\text{VSS}}$  向  $P_\ell$ ,  $\ell \in I$  发送份额  $q_i(\alpha_\ell)$ ; 否则,  $\mathcal{S}$  模拟  $\mathcal{F}_{\text{VSS}}$  向  $P_\ell$ ,  $\ell \in I$  发送  $\perp$ , 并设置  $q_i(x) = 0$ .
  3. 对于  $j \in [n]$ , 将  $P_j$  在电路中的输入导线记为  $w_j$ . 对于  $i \in I$ ,  $\mathcal{S}$  记录  $q_j(\alpha_i)$ , 将其作为  $P_i$  在导线  $w_j$  上的秘密份额。
- 与  $\mathcal{F}_{\text{sfe}}$  的交互：
  1.  $\mathcal{S}$  代表被攻陷方向  $\mathcal{F}_{\text{sfe}}$  发送输入  $\{x_i = q_i(0)\}_{i \in I}$ .
  2.  $\mathcal{S}$  接收  $\mathcal{F}_{\text{sfe}}$  向被攻陷方发送的输出  $\{y_i\}_{i \in I}$ .
- 计算阶段：令  $G_1, \dots, G_m$  是预先确定的电路门的拓扑顺序。对于  $k = 1, \dots, m$ ,  $\mathcal{S}$  根据门的类型执行以下操作之一：

$\{\mathcal{F}_{\text{VSS}}, \mathcal{F}_{\text{mult}}\}$ -混合模型下的安全函数计算协议  $\Pi_{\text{sfe}}$

**输入:**  $P_i$  的输入是  $x_i \in \mathbb{F}$ .

**公共输入:** 函数  $f: \mathbb{F}^n \rightarrow \mathbb{F}^n$  对应的算术电路  $C$ , 对于所有  $\vec{x} \in \mathbb{F}^n$  有  $C(\vec{x}) = f(\vec{x})$ . 域  $\mathbb{F}$ ,  $n$  个互不相同的非 0 元素  $\alpha_1, \dots, \alpha_n \in \mathbb{F}$ .

**协议:**

1. 输入分享阶段:

- (a) 每个参与方  $P_i$  均匀随机选择常数项为  $x_i$  的  $t$  阶多项式  $q_i(x) \leftarrow_{\$} \mathcal{P}^{x_i, t}$ , 然后作为秘密分发者以  $q_i(x)$  为输入调用  $\mathcal{F}_{\text{VSS}}$ .
- (b) 每个参与方记录从  $\mathcal{F}_{\text{VSS}}$  处接收到的值  $q_1(\alpha_i), \dots, q_n(\alpha_i)$ . 如果某些值为  $\perp$ , 将其替换为 0.

2. 计算阶段: 令  $G_1, \dots, G_m$  是预先确定的电路门的一个拓扑顺序。对于  $k = 1, \dots, m$ , 参与方根据门的类型执行以下操作之一:

- 情况 1:  $G_k$  是加法门。令  $\beta_i^k$  和  $\gamma_i^k$  是  $P_i$  持有的输入导线份额。 $P_i$  设置输出导线份额为  $\delta_i^k = \beta_i^k + \gamma_i^k$ .
- 情况 2:  $G_k$  是带有常数  $c$  的常数乘法门。令  $\beta_i^k$  是  $P_i$  持有的输入导线份额。 $P_i$  设置输出导线份额为  $\delta_i^k = c \cdot \beta_i^k$ .
- 情况 3:  $G_k$  是乘法门。令  $\beta_i^k$  和  $\gamma_i^k$  是  $P_i$  持有的输入导线份额。 $P_i$  以  $(\beta_i^k, \gamma_i^k)$  为输入调用  $\mathcal{F}_{\text{mult}}$  并得到  $\delta_i^k$ .  $P_i$  设置输出导线份额为  $\delta_i^k$ .

3. 输出重建阶段:

- (a) 令  $o_1, \dots, o_n$  是电路的输出导线, 其中  $P_i$  的输出是导线  $o_i$  上的值。对于  $i = 1, \dots, n$ , 将参与方持有的导线  $o_i$  的份额记为  $\beta_1^i, \dots, \beta_n^i$ . 每个参与方  $P_j$  向  $P_i$  发送  $\beta_j^i$ .
- (b) 当收到所有秘密份额后,  $P_i$  以  $(\beta_1^i, \dots, \beta_n^i)$  为输入运行 Reed-Solomon 解码算法得到正确的码字  $(\tilde{\beta}_1^i, \dots, \tilde{\beta}_n^i)$ . 然后,  $P_i$  使用  $(\tilde{\beta}_1^i, \dots, \tilde{\beta}_n^i)$  中任意  $t+1$  个点通过拉格朗日插值法计算出多项式  $g_i(x)$  并输出  $g_i(0)$ .

图 9.27:  $\{\mathcal{F}_{\text{VSS}}, \mathcal{F}_{\text{mult}}\}$ -混合模型下的安全函数计算协议  $\Pi_{\text{sfe}}$

1. 情况 1:  $G_k$  是加法门。令  $\beta_i^k$  和  $\gamma_i^k$  是  $\mathcal{S}$  为  $P_i$ ,  $i \in \hat{I}$  存储的  $G_k$  的输入导线份额。对于  $i \in \hat{I}$ ,  $\mathcal{S}$  计算并存储  $\delta_i^k = \beta_i^k + \gamma_i^k$ , 作为  $P_i$  在  $G_k$  的输出导线上的份额。
  2. 情况 2:  $G_k$  是带有常数  $c$  的常数乘法门。令  $\beta_i^k$  是  $\mathcal{S}$  为  $P_i$ ,  $i \in \hat{I}$  存储的  $G_k$  的输入导线份额。对于  $i \in \hat{I}$ ,  $\mathcal{S}$  计算并存储  $\delta_i^k = c \cdot \beta_i^k$ , 作为  $P_i$  在  $G_k$  的输出导线上的份额。
  3. 情况 3:  $G_k$  是乘法门。令  $\beta_i^k$  和  $\gamma_i^k$  是  $\mathcal{S}$  为  $P_i$ ,  $i \in I$  存储的  $G_k$  的输入导线份额。首先,  $\mathcal{S}$  模拟  $\mathcal{F}_{\text{mult}}$  向对手  $\mathcal{A}$  发送  $\{(\beta_i^k, \gamma_i^k)\}_{i \in I}$  ( $\mathcal{F}_{\text{mult}}$  的步骤 2); 然后,  $\mathcal{S}$  记录对手  $\mathcal{A}$  向  $\mathcal{F}_{\text{mult}}$  发送的  $\{\delta_i\}_{i \in I}$  ( $\mathcal{F}_{\text{mult}}$  的步骤 3), 如果某些值没有收到, 将其设置为 0. 最后,  $\mathcal{S}$  存储  $\delta_i^k$  作为  $P_i$ ,  $i \in I$  在  $G_k$  的输出导线上的份额, 并模拟  $\mathcal{F}_{\text{mult}}$  向  $P_i$ ,  $i \in I$  发送  $\delta_i$  ( $\mathcal{F}_{\text{mult}}$  的步骤 5)。对于  $i \in \hat{I} \setminus I$ ,  $\mathcal{S}$  为其生成随机份额  $\delta_i^k$ .
- 输出重建阶段: 对于  $i \in I$ ,  $\mathcal{S}$  执行如下操作。令  $o_i$  是参与方  $P_i$  在电路中的输出导线, 令  $\{\beta_\ell^i\}_{\ell \in \hat{I}, i \in I}$  是  $\mathcal{S}$  为  $P_\ell$ ,  $\ell \in \hat{I}$  存储的导线  $o_i$  上的份额。然后,  $\mathcal{S}$  在约束条件  $q'_i(\alpha_\ell) = \beta_\ell^i$ ,  $\ell \in \hat{I}$  和  $q'_i(0) = y_i$  下计算唯一的  $t$  阶多项式  $q'_i(x)$ , 其中  $y_i$  是  $\mathcal{S}$  从  $\mathcal{F}_{\text{sfe}}$  处接收到的  $P_i$  的输出。最后, 对于  $j \notin I$ ,  $\mathcal{S}$  模拟  $P_j$  向  $P_i$  发送  $q'_i(\alpha_j)$ .

不可区分性。我们通过一系列混合实验来证明真实世界与理想世界不可区分。

**混合实验  $\mathcal{H}_0$ :** 这是理想世界的执行  $\text{EXEC}_{\mathcal{F}_{\text{sfe}}, \mathcal{S}, \mathcal{Z}}$ .

**混合实验  $\mathcal{H}_1$ :**  $\mathcal{H}_1$  与  $\mathcal{H}_0$  相同, 除了在输入分享阶段, 模拟器  $\mathcal{S}$  为诚实方  $P_j$  选取多项式的方式变为  $q_j(x) \leftarrow_{\$} \mathcal{P}^{x_j, t}$  而不是  $q_j(x) \leftarrow_{\$} \mathcal{P}^{0, t}$ .

**断言 1:** 如果  $|I| \leq t$ , 那么  $\mathcal{H}_1$  与  $\mathcal{H}_0$  完美不可区分。

证明.  $\mathcal{H}_1$  中的修改仅会影响对手在输入分享阶段收到的份额  $\{(q_j(\alpha_i))\}_{j \notin I, i \in I}$ . 由于  $|I| \leq t$ , 根据推论 9.13,  $\mathcal{H}_1$  与  $\mathcal{H}_0$  中的对手得到的秘密份额具有相同的分布, 因此  $\mathcal{H}_1$  与  $\mathcal{H}_0$  完美不可区分。  $\square$

**混合实验  $\mathcal{H}_2$ :** 这是真实世界的执行  $\text{EXEC}_{\Pi_{\text{sfe}}, \mathcal{A}, \mathcal{Z}}^{\mathcal{F}_{\text{VSS}}, \mathcal{F}_{\text{mult}}}$ .

**断言 2:** 如果  $|I| \leq t$ , 那么  $\mathcal{H}_2$  与  $\mathcal{H}_1$  完美不可区分。

证明.  $\mathcal{H}_2$  与  $\mathcal{H}_1$  的区别在于输出重建阶段的计算方式。我们首先证明诚实方有相同的输出。在真实协议中,  $\mathcal{F}_{\text{VSS}}$  首先确保了输入向量  $\vec{x}$  是良好定义的; 其次,  $\mathcal{F}_{\text{mult}}$  使参与方计算乘法门后能得到正确的输出导线份额。因此, 当电路计算完成后, 诚实方持有电路输出导线的正确份额。最后, Reed-Solomon 解码算法可以纠正  $\frac{n-t}{2} > \frac{3t-t}{2} = t$  个错误, 因此对手在输出重建阶段无法影响诚实方的输出, 故  $\mathcal{H}_2$  与  $\mathcal{H}_1$  中诚实方的输出相同。

接下来证明对手  $\mathcal{A}$  有相同的视图。这与定理 9.20 的断言 3 同理。当  $|I| = t$  时, 被攻陷方的份额和输出值唯一确定了  $q'_i(x)$ ; 当  $|I| < t$  时,  $\mathcal{S}$  选择的多项式与真实协议中的多项式具有相同的分布。

注意, 这里的证明为了简洁性, 我们假设输出值都来自乘法门。如果输入值来自加法门或常数乘法门, 协议也是安全的, 但  $\mathcal{S}$  的模拟方式需要更加小心以满足一致性要求。我们在第 6.1.2 节的最后讨论过这一点。  $\square$

定理证毕。

$\square$

**总结。**在前文中，我们已经在朴素模拟下实现了理想功能  $\mathcal{F}_{\text{VSS}}$  和  $\mathcal{F}_{\text{mult}}$ . 根据通用组合定理（定理 4.3），我们有以下结论。

**定理 9.23.** 假设参与方之间存在点对点安全信道、认证的广播信道。假设被攻陷方的数量不超过  $t$ ,  $t < n/3$ . 那么，在朴素模型 (*plain model*) 下存在协议对于静态恶意敌手 UC-安全实现了理想功能  $\mathcal{F}_{\text{sfe}}$  (图 6.1)。

至此，我们完成了 BGW 协议（恶意安全）的全部介绍与 UC 安全性证明。

## 9.4 BDOZ 协议和 SPDZ 协议

第9.3节的 BGW 协议在  $t < n/3$  (诚实大多数) 的场景下对于 (静态) 恶意敌手实现了完美的安全性。而 BDOZ 协议<sup>[71]</sup>和 SPDZ 协议<sup>[72]</sup>能够对于控制  $t = n - 1$  个参与方的 (静态) 恶意敌手具有安全性。BDOZ 协议和 SPDZ 协议都使用了 Beaver 三元组技术，在此基础上，通过消息认证码实现了恶意安全性。BDOZ 协议的方法是对让每个参与方对每一个秘密份额都进行认证，即，当需要重建某个值时，各参与方公布自己的秘密份额以及它的  $n - 1$  个消息认证码标签，然后，每个参与方检查所有其他参与方的秘密份额的真实性 (总通信量和计算量均为  $O(n^2)$ ). SPDZ 消息认证码的设计更加巧妙，在重建某个值时，实现了线性的通信量和计算量。

下面，我们展开介绍。

### 9.4.1 BDOZ 协议

#### 9.4.1.1 直观思想

让我们以第7.1节的基于 Beaver 三元组的协议作为起点。为简单起见，我们先考虑两方计算的场景，并假设 Beaver 三元组是由可信第三方生成的。

**敌手的攻击。**我们已经证明，第7.1节的协议 (图 7.1) 是半诚实安全的。但是，如果参与方是恶意的，他可以进行怎样的攻击呢？假设参与方是  $P_1$  和  $P_2$ ，其中  $P_1$  是恶意的被攻陷方。首先，对于输入分享阶段，假设  $P_1$  的输入为  $x$ ,  $P_1$  应该随机选取  $x_1, x_2$ , 满足  $x_1 + x_2 = x$ , 然后将  $x_2$  发给  $P_2$ . 恶意的  $P_1$  可以任意选取  $x'_1, x'_2$ , 其中  $x'_1 + x'_2 \neq x$ , 并将  $x'_2$  发给  $P_2$ . 但这不是问题，因为这等价于  $P_2$  将自己的输入替换为  $x' = x'_1 + x'_2$ . 我们之前讨论过，“输入替换”是敌手唯一被允许的影响。

然而，在计算乘法门和输出重建时，参与方需要公布自己的份额以进行秘密重建。此时，我们没有任何机制来确保恶意参与方公布正确的份额。以输出重建阶段为例，假设某个公共输出值为  $y$ ,  $P_1$  持有  $y_1$ ,  $P_2$  持有  $y_2$ , 满足  $y_1 + y_2 = y$ . 此时，恶意的  $P_1$  可以发送另一个值  $y'_1 \neq y_1$  给  $P_2$ . 这样一来， $P_2$  得到的输出就变为  $y' = y'_1 + y_2$ . 这确实是一种对协议的攻击：该攻击使得  $P_2$  输出了错误的结果，但是在理想世界中，敌手无法做到这一点。在计算乘法门进行的秘密重建时，敌手也一样可以发起这种攻击，使参与方重建出错误的值。

因此，我们需要设计一种机制，使得参与方在秘密重建时必须公布正确的秘密份额。具体而言，假设参与方需要重建某个值  $y$ , 其中  $P_1$  持有  $y_1$ ,  $P_2$  持有  $y_2$ . 当  $P_1(P_2)$  公布任何  $y_1(y_2)$  以外的其他值时都会被检测到，然后参与方终止协议。

**消息认证码。**回想一下，在第9.2.1节的切分选择协议中，我们在实现输出真实性时也遇到了类似的问题，解决方法是使用消息认证码 (Message Authentication Codes, MACs). 现在，我们再次使用该技术，让

$P_2$  对  $P_1$  的秘密份额进行认证 ( $P_1$  也对  $P_2$  的秘密份额进行认证)。我们使用的消息认证码的形式仍然是  $\text{Mac}_{\Delta, K}(x) = \Delta \cdot x + K$ , 其中  $\Delta, K$  是密钥,  $x$  是认证的消息,  $\text{Mac}_{\Delta, K}(x)$  是标签,  $\Delta, K, x \in \mathbb{F}_p$ 。不难看出, 这是信息论安全的一次性消息认证码, 敌手在不知道密钥的情况下成功伪造标签的概率为  $1/p$  (“一次性”是因为如果敌手拥有同一密钥下的两个消息标签对  $(x_1, M_1), (x_2, M_2)$ , 他就可以求解密钥从而伪造标签)。

这个 MAC 的构造还具有一个很重要的性质。如果我们将不同密钥中的  $\Delta$  取相同的值 ( $K$  取不同的值), 那么该 MAC 就具有了加法同态性。具体而言, 设  $M_1 = \text{Mac}_{\Delta, K_1}(x_1) = \Delta \cdot x_1 + K_1$ ,  $M_2 = \text{Mac}_{\Delta, K_2}(x_2) = \Delta \cdot x_2 + K_2$ , 那么  $M_1 + M_2 = \Delta \cdot (x_1 + x_2) + (K_1 + K_2)$ , 即  $M_1 + M_2$  是  $x_1 + x_2$  在密钥  $(\Delta, K_1 + K_2)$  下合法的标签。而加法同态性正是我们计算算术电路时所需要的!

**协议构造。**让我们回到协议。我们称  $\Delta$  是全局 MAC 密钥,  $K$  是本地 MAC 密钥。首先,  $P_1$  应该持有全局密钥  $\Delta_1$ ,  $P_2$  持有全局密钥  $\Delta_2$ . 对于某个秘密值  $x$ ,  $P_1$  应该持有秘密份额  $(x_1, K_1, M_1)$ ,  $P_2$  持有秘密份额  $(x_2, K_2, M_2)$ , 满足以下条件

- $x_1 + x_2 = x$
- $M_1 = \Delta_2 \cdot x_1 + K_2$
- $M_2 = \Delta_1 \cdot x_2 + K_1$

即  $x_1, x_2$  是  $x$  的加法秘密分享,  $M_1$  是  $P_1$  持有的 (密钥  $(\Delta_2, K_2)$  下的)  $x_1$  的 MAC 标签,  $M_2$  是  $P_2$  持有的 (密钥  $(\Delta_1, K_1)$  下的)  $x_2$  的 MAC 标签。当需要重建  $x$  时,  $P_1$  公布  $(x_1, M_1)$ ,  $P_2$  公布  $(x_2, M_2)$ , 然后  $P_1$  检查  $M_2 = ? \Delta_1 \cdot x_2 + K_1$ ,  $P_2$  检查  $M_1 = ? \Delta_2 \cdot x_1 + K_2$ . 我们将该步骤称为秘密值的“打开”(可验证地)。如果需要将秘密值打开给某一个参与方而不是打开给所有人, 只需将秘密份额和 MAC 标签发给那个参与方即可。

下面, 我们用符号  $[x]$  表示 BDOZ 认证秘密分享, 在两方场景下, 它的定义如下:

$$[x] = [\{x_i, K_i, M_i\}_{i=1,2}]$$

因为 MAC 满足加法同态性, 我们可以定义如下的线性运算。对于  $[x] = [\{x_i, K_i, M_i\}_{i=1,2}], [x'] = [\{x'_i, K'_i, M'_i\}_{i=1,2}]$  以及常数  $\alpha \in \mathbb{F}_p$ , 加法和常数乘法定义如下:

$$\begin{aligned} [x] + [x'] &= [\{x_i + x'_i, K_i + K'_i, M_i + M'_i\}_{i=1,2}] \\ \alpha[x] &= [\{\alpha x_i, \alpha K_i, \alpha M_i\}_{i=1,2}] \end{aligned}$$

显然, 该运算满足

$$[x] + [x'] = [x + x']$$

$$\alpha[x] = [\alpha x]$$

对于常数加法, 当需要加常数  $\alpha$  时,  $P_1$  需要将其秘密份额加上  $\alpha$ , 此时,  $P_2$  需要更新本地 MAC 密钥以满足 MAC 验证等式。其定义如下:

$$[x] + \alpha = [\{x_1 + \alpha, K_1, M_1\}, \{x_2, K_2 - \Delta_2 \alpha, M_2\}]$$

同样地, 该运算满足

$$[x] + \alpha = [x + \alpha]$$

三元组的生成与输入分享。至此，我们对于 BDOZ 认证秘密分享成功实现了线性运算！并且 MAC 的加入使得秘密重建时参与方必须揭示正确的秘密份额，否则 MAC 验证将无法通过。如果我们还能够生成认证 Beaver 三元组和输入值的认证秘密分享，那么基于加法同态性和 Beaver 三元组的性质，我们就可以实现恶意安全的算术电路计算协议了！

在可信第三方的假设下，可以由可信第三方生成认证 Beaver 三元组  $([a], [b], [c])$ 。具体而言，可信第三方首先为  $P_i$  随机选择全局 MAC 密钥  $\Delta_i$ ,  $i = 1, 2$ . 然后，他随机选择  $a, b \leftarrow_{\$} \mathbb{F}$  并计算  $c = ab$ . 接下来，可信第三方对  $a, b, c$  做加法秘密分享，并随机选择每一个本地 MAC 密钥，然后计算每一个秘密份额的 MAC 标签。最后，可信第三方向每个参与方分发全局 MAC 密钥  $\Delta_i$  以及认证秘密份额。

而对于输入值的分享，我们无法让参与方直接生成输入值的认证秘密分享。这里采取的方法是消耗一个随机值的认证秘密分享。协议如下：假设  $P_1$  的输入是  $x_1$ . 首先，可信第三方在生成认证 Beaver 三元组的同时还生成了足够的随机值的认证秘密分享。参与方选取一个随机值的认证秘密分享  $[a]$ ，然后将  $a$  打开给  $P_1$ .  $P_1$  广播  $\delta = x_1 - a$ . 参与方计算  $[a] + \delta = [a + \delta] = [x_1]$ ，得到  $x_1$  的认证秘密分享。因为  $a$  是均匀随机的，所以  $\delta = x_1 - a$  不会泄漏  $x_1$  的任何信息。

**两方协议小结。**综上所述，假设由可信第三方生成 Beaver 三元组，在两方场景下，我们介绍了 BDOZ 协议的构造。我们做一个小结。

预处理（离线）阶段：

- 可信第三方生成足够的认证 Beaver 三元组和随机值的认证秘密分享。

在线阶段：

- 输入分享：设  $P_i$  的输入为  $x_i$ . 参与方取一个随机值的认证秘密分享  $[a]$ ，将  $a$  打开给  $P_i$ .  $P_i$  广播  $\delta = x_i - a$ . 参与方计算  $[a] + \delta = [x_i]$ .

- 逐门计算：

– 对于加法门、常数乘法门、常数加法门，分别计算

$$[x] + [y] = [x + y], \alpha[x] = [\alpha x], [x] + \alpha = [x + \alpha]$$

– 对于乘法门，执行如下操作

- \* 取认证 Beaver 三元组  $([a], [b], [c])$ ，本地计算  $[x] - [a] = [x - a], [y] - [b] = [y - b]$ ;
- \* 向所有参与方打开  $d = x - a, e = y - b$ ;
- \* 参与方本地计算  $d[b] + e[a] + [c] + de = [xy]$ .

- 输出重建：对于  $P_i$  的输出  $y_i$ ，将  $y_i$  打开给  $P_i$ .

**拓展到多方场景。**不难发现，上述思路可以拓展到多方场景。假设参与方的数量为  $n$ . BDOZ 认证秘密分享的定义为：

$$[x] = [\{x_i, \{K_{x_j}^i, M_j(x_i)\}_{j \in [n], j \neq i}\}_{i=1}^n]$$

其中  $K_{x_j}^i$  是  $P_i$  持有的对  $(P_j$  的秘密份额  $x_j$  认证的密钥， $M_j(x_i) = \Delta_j \cdot x_i + K_{x_i}^j$  是  $x_i$  在  $P_j$  的密钥下的 MAC 标签， $\{x_i, \{K_{x_j}^i, M_j(x_i)\}_{j \in [n], j \neq i}\}$  是  $P_i$  持有的  $x$  的秘密份额。也就是说， $P_i$  需要对每个参与方的份额都进行认证，同时  $P_i$  的份额也需要被每个其他参与方认证。当需要将  $[x]$  打开给  $P_j$

时, 每个参与方  $P_i$  发送  $x_i, M_j(x_i)$  给  $P_j$ ,  $P_j$  检查  $M_j(x_i) \stackrel{?}{=} \Delta_j \cdot x_i + K_{x_i}^j$ . 如果检查均通过,  $P_j$  计算  $x = \sum_{i=1}^n x_i$ ; 否则,  $P_j$  广播 fail. 如果要将  $x$  打开给所有人, 那么对每个参与方执行上述打开操作。

与两方场景类似, 对于  $[x] = [\{x_i, \{K_{x_j}^i, M_j(x_i)\}_{j \in [n], j \neq i}\}_{i=1}^n]$ ,  $[x'] = [\{x'_i, \{K_{x'_j}^i, M_j(x'_i)\}_{j \in [n], j \neq i}\}_{i=1}^n]$  和常数  $\alpha \in \mathbb{F}_p$ , 我们定义秘密份额的加法、常数乘法、常数加法运算:

$$\begin{aligned}[x] + [x'] &= [\{x_i + x'_i, \{K_{x_j}^i + K_{x'_j}^i, M_j(x_i) + M_j(x'_i)\}_{j \in [n], j \neq i}\}_{i=1}^n] \\ \alpha[x] &= [\{\alpha x_i, \{\alpha K_{x_j}^i, \alpha M_j(x_i)\}_{j \in [n], j \neq i}\}_{i=1}^n] \\ [x] + \alpha &= [\{x_1 + \alpha, \{K_{x_1}^1, M_j(x_1)\}_{j \in [n], j \neq 1}\}, \{x_i, \{K_{x_1}^i - \Delta_i \alpha, \{K_{x_j}^i\}_{j=2, \dots, n, j \neq i}\}, \{M_j(x_i)\}_{j \in [n], j \neq i}\}_{i=2}^n]\end{aligned}$$

注意, 在常数加法中,  $P_1$  将自己的秘密份额更新为  $x_1 + \alpha$ , 其他参与方  $P_i, i \neq 1$  将认证  $P_1$  的份额所使用的本地密钥更新为  $K_{x_1}^i - \Delta_i \alpha$ .

上述运算满足

$$\begin{aligned}[x] + [x'] &= [x + x'] \\ \alpha[x] &= [\alpha x] \\ [x] + \alpha &= [x + \alpha]\end{aligned}$$

在预处理阶段, 由可信第三方生成所有认证 Beaver 三元组和随机值的认证秘密分享。在线阶段中, 计算方式与两方场景完全相同。至此, 我们在可信第三方生成认证 Beaver 三元组的假设下, 完成了 BDOZ 协议的介绍。

#### 9.4.1.2 协议描述

本节, 我们给出 BDOZ 协议的形式化描述。这里, 我们假设存在可信第三方生成 Beaver 三元组, 并将其抽象为理想功能  $\mathcal{F}_{\text{prep}}$ . 我们首先给出符号  $[.]$  的定义与运算规则, 然后给出理想功能  $\mathcal{F}_{\text{prep}}$  的定义, 最后在  $\mathcal{F}_{\text{prep}}$ -混合模型下给出完整的协议描述。

**符号表示与线性运算。**对于消息  $x \in \mathbb{F}_p$ , 密钥  $(\Delta, K) \in \mathbb{F}_p^2$ , 消息验证码 MAC 的定义为  $\text{Mac}_{\Delta, K}(x) = \Delta \cdot x + K$ . 我们将  $\Delta$  称为全局 MAC 密钥,  $K$  称为本地 MAC 密钥。每个参与方  $P_i$  持有全局 MAC 密钥的份额  $\Delta_i$ .<sup>16</sup>

值  $x$  的认证秘密分享记为  $[x]$ , 定义如下:

$$[x] = [\{x_i, \{K_{x_j}^i, M_j(x_i)\}_{j \in [n], j \neq i}\}_{i=1}^n]$$

其中,  $x = x_1 + \dots + x_n$ ,  $M_j(x_i) = \text{Mac}_{\Delta_j, K_{x_i}^j}(x_i)$ .

对于秘密分享  $[.]$ , 它的打开、加法、常数乘法、常数加法运算定义如图 9.28 所示。

不难看出, 图 9.28 中的运算满足

$$\begin{aligned}[x] + [x'] &= [x + x'] \\ \alpha[x] &= [\alpha x] \\ [x] + \alpha &= [x + \alpha]\end{aligned}$$

**理想功能  $\mathcal{F}_{\text{prep}}$ .** 我们将预处理阶段中生成认证 Beaver 三元组 (以及单个值的认证秘密分享) 抽象为理想功能  $\mathcal{F}_{\text{prep}}$ .  $\mathcal{F}_{\text{prep}}$  接受 3 种命令: INIT, SINGLES, TRIPLES, 分别用于初始化、生成单个值的认证

<sup>16</sup> 原始论文<sup>[71]</sup>中,  $P_i$  对每个参与方  $P_j, j \neq i$  生成了全局 MAC 密钥  $\Delta_j^i$ . 这没有必要,  $P_i$  对所有其他参与方  $P_j, j \neq i$  只需使用同一个全局 MAC 密钥  $\Delta_i$  即可, 因此本教材作了简化。

秘密分享  $[ \cdot ]$  的运算

**打开:** 当需要将  $[x] = [\{x_i, \{K_{x_j}^i, M_j(x_i)\}_{j \in [n], j \neq i}\}_{i=1}^n]$  打开给  $P_j$  时, 每个参与方  $P_i$  发送  $x_i, M_j(x_i)$  给  $P_j$ ,  $P_j$  检查  $M_j(x_i) \stackrel{?}{=} \Delta_j \cdot x_i + K_{x_i}^j$ . 如果检查均通过,  $P_j$  计算  $x = \sum_{i=1}^n x_i$ ; 否则,  $P_j$  广播 fail. 如果要将  $x$  打开给所有人, 那么对每个参与方执行上述打开操作。

**加法:** 对于  $[x] = [\{x_i, \{K_{x_j}^i, M_j(x_i)\}_{j \in [n], j \neq i}\}_{i=1}^n]$ ,  $[x'] = [\{x'_i, \{K_{x'_j}^i, M_j(x'_i)\}_{j \in [n], j \neq i}\}_{i=1}^n]$ , 加法运算定义为:

$$[x] + [x'] = [\{x_i + x'_i, \{K_{x_j}^i + K_{x'_j}^i, M_j(x_i) + M_j(x'_i)\}_{j \in [n], j \neq i}\}_{i=1}^n]$$

**常数乘法:** 对于  $[x] = [\{x_i, \{K_{x_j}^i, M_j(x_i)\}_{j \in [n], j \neq i}\}_{i=1}^n]$  和常数  $\alpha \in \mathbb{F}_p$ , 常数乘法定义为:

$$\alpha[x] = [\{\alpha x_i, \{\alpha K_{x_j}^i, \alpha \cdot M_j(x_i)\}_{j \in [n], j \neq i}\}_{i=1}^n]$$

**常数加法:** 对于  $[x] = [\{x_i, \{K_{x_j}^i, M_j(x_i)\}_{j \in [n], j \neq i}\}_{i=1}^n]$  和常数  $\alpha \in \mathbb{F}_p$ , 常数加法定义为:

$$[x] + \alpha = [\{x_1 + \alpha, \{K_{x_j}^1, M_j(x_1)\}_{j \in [n], j \neq 1}\}, \{x_i, \{K_{x_1}^i - \Delta_i \alpha, \{K_{x_j}^i\}_{j=2, \dots, n, j \neq i}\}, \{M_j(x_i)\}_{j \in [n], j \neq i}\}_{i=2}^n]$$

图 9.28: 秘密分享  $[ \cdot ]$  的运算BDOZ 预处理阶段理想功能  $\mathcal{F}_{\text{prep}}$ 

令  $C$  为被攻陷方的集合。

**初始化:** 当收到来自每个参与方  $P_i$  的消息  $(\text{INIT}, \text{sid}, p)$  时,  $\mathcal{F}_{\text{prep}}$  记录域的模数  $p$ . 对于每个诚实方  $P_i \notin C$ , 随机选择  $\Delta_i \leftarrow_{\$} \mathbb{F}_p$ ; 对于每个被攻陷方  $P_j \in C$ , 记录其发送的  $\Delta_j$ .

**单个值:** 当收到来自每个参与方  $P_i$  的消息  $(\text{SINGLES}, \text{sid}, u)$  时,  $\mathcal{F}_{\text{prep}}$  执行以下操作: 对于  $v = 1, \dots, u$ :

1. 执行期间, 如果收到来自敌手  $S$  的  $(\text{STOP}, \text{sid})$ , 那么终止并向所有参与方发送 fail.
2.  $\mathcal{F}_{\text{prep}}$  随机选择  $a \leftarrow_{\$} \mathbb{F}_p$ , 然后
  - (a) 对于每个被攻陷方  $P_j \in C$ , 记录其发送的  $a_j$ . 然后, 为每个诚实方  $P_i \notin C$  随机选择  $a_i \leftarrow_{\$} \mathbb{F}_p$ , 满足  $\sum_{i=1}^n a_i = a$ .
  - (b) 对于每个被攻陷方  $P_j \in C$ , 记录其发送的  $K_{a_i}^j$ ,  $i = 1, \dots, n$ . 对于每个诚实方  $P_i \notin C$ , 随机选择  $K_{a_j}^i$ ,  $j = 1, \dots, n$ .
  - (c) 对于  $i = 1, \dots, n$ ,  $j = 1, \dots, n$ , 计算  $M_j(a_i) = \Delta_j \cdot a_i + K_{a_i}^j$ .
  - (d) 至此  $[a]$  的所有数据均已生成。 $\mathcal{F}_{\text{prep}}$  向每个参与方  $P_i$  发送  $\{a_i, \{K_{a_j}^i, M_j(a_i)\}_{j \in [n], j \neq i}\}$ .

**三元组:** 当收到来自每个参与方  $P_i$  的消息  $(\text{TRIPLES}, \text{sid}, u)$  时,  $\mathcal{F}_{\text{prep}}$  执行以下操作: 对于  $v = 1, \dots, u$ :

1. 步骤 1 与 “单个值” 中相同。
2. 随机选择  $a, b \leftarrow_{\$} \mathbb{F}_p$ , 计算  $c = ab$ . 然后如 “单个值” 中的步骤 2 那样生成  $[a], [b], [c]$  并分发。

图 9.29: BDOZ 预处理阶段理想功能  $\mathcal{F}_{\text{prep}}$

秘密分享、生成认证 Beaver 三元组。有两点需要注意：(1)  $\mathcal{F}_{\text{prep}}$  允许被攻陷方选择自己的秘密份额和 MAC 密钥，对于诚实方，这些值由  $\mathcal{F}_{\text{prep}}$  随机选择；(2)  $\mathcal{F}_{\text{prep}}$  一次生成  $u$  个认证 Beaver 三元组（或单个值的认证秘密分享），其中  $u$  是一个参数。这是因为在实现  $\mathcal{F}_{\text{prep}}$  的协议中，参与方一次生成  $u$  个认证 Beaver 三元组， $\mathcal{F}_{\text{prep}}$  的定义需要与之吻合。

$\mathcal{F}_{\text{prep}}$  的定义如图 9.29 所示。

**协议描述。**下面，我们给出  $\mathcal{F}_{\text{prep}}$ -混合模型下，BDOZ 协议的完整描述，如图 9.30 所示。基于 BDOZ 认证秘密分享的线性同态性和可安全打开性，协议整体与基于 Beaver 三元组的协议非常类似，并实现了恶意安全性。

### $\mathcal{F}_{\text{prep}}$ -混合模型下的 BDOZ 协议 $\Pi_{\text{BDOZ}}$

参与方  $P_1, \dots, P_n$  想要计算函数  $f : \mathbb{F}^{M_{\text{in}}^1} \times \dots \times \mathbb{F}^{M_{\text{in}}^n} \rightarrow \mathbb{F}^{M_{\text{out}}^1} \times \dots \times \mathbb{F}^{M_{\text{out}}^n}$  并已经将其转化成了包含加法门、常数乘法门、常数加法门、乘法门的算术电路。

#### 预处理阶段：

参与方首先向  $\mathcal{F}_{\text{prep}}$  发送  $(\text{INIT}, \text{sid}, p)$ . 然后，参与方向  $\mathcal{F}_{\text{prep}}$  调用足够次数的  $(\text{TRIPLES}, \text{sid}, u)$  和  $(\text{SINGLES}, \text{sid}, u)$  以获得足够的认证三元组以及单个值的认证秘密分享。

#### 在线阶段：

- **输入分享：**假设  $P_i$  的某个输入为  $x_i$ , 取单个值的认证秘密分享  $[a]$ , 执行如下操作:

1. 向  $P_i$  打开  $[a]$ ;
2.  $P_i$  广播  $\delta = x_i - a$ ;
3. 参与方计算  $[a] + \delta = [x_i]$ .

对于  $P_i$  的每一个输入，都执行上述操作。

- **逐门计算：**重复以下步骤，直到所有门都被计算。根据门的类型，执行以下操作之一：

- 加法门。假设参与方持有  $[x], [y]$ . 参与方计算  $[x] + [y] = [x + y]$ .
- 常数乘法门。假设对应的常数为  $\alpha$ , 参与方持有  $[x]$ . 参与方计算  $\alpha[x] = [\alpha x]$ .
- 常数加法门。假设对应的常数为  $\alpha$ , 参与方持有  $[x]$ . 参与方计算  $[x] + \alpha = [x + \alpha]$ .
- 乘法门。假设参与方持有  $[x], [y]$ , 执行以下操作:

1. 取 Beaver 三元组  $([a], [b], [c])$ ;
2. 参与方本地计算  $[x] - [a] = [x - a], [y] - [b] = [y - b]$ ;
3. 对所有参与方打开  $[x - a], [y - b]$ , 记为  $d = x - a, e = y - b$ ;
4. 参与方本地计算  $d[b] + e[a] + [c] + de = [xy]$ .

- **输出重建：**假设  $P_i$  的某个输出为  $y_i$ , 将  $[y_i]$  打开给  $P_i$ . 对于  $P_i$  的每一个输出，都执行该操作。

图 9.30:  $\mathcal{F}_{\text{prep}}$ -混合模型下的 BDOZ 协议  $\Pi_{\text{BDOZ}}$

### 9.4.1.3 安全性证明

根据协议基于 Beaver 三元组的构造思想，不难发现，协议中泄漏的值都是均匀随机值。恶意敌手能够发动的攻击仅为公布错误的秘密份额。而根据 MAC 的构造，错误的秘密份额通过 MAC 验证的概率为  $1/p$ .

我们给出恶意敌手模型下的安全函数计算理想功能  $\mathcal{F}_{\text{sfe}}$ ，如图 9.31 所示。它与半诚实版本有两点区别：(1) 敌手  $\mathcal{S}$  可以随时终止协议；(2) 在输出阶段，敌手  $\mathcal{S}$  可以阻止诚实方获得输出。<sup>17</sup>

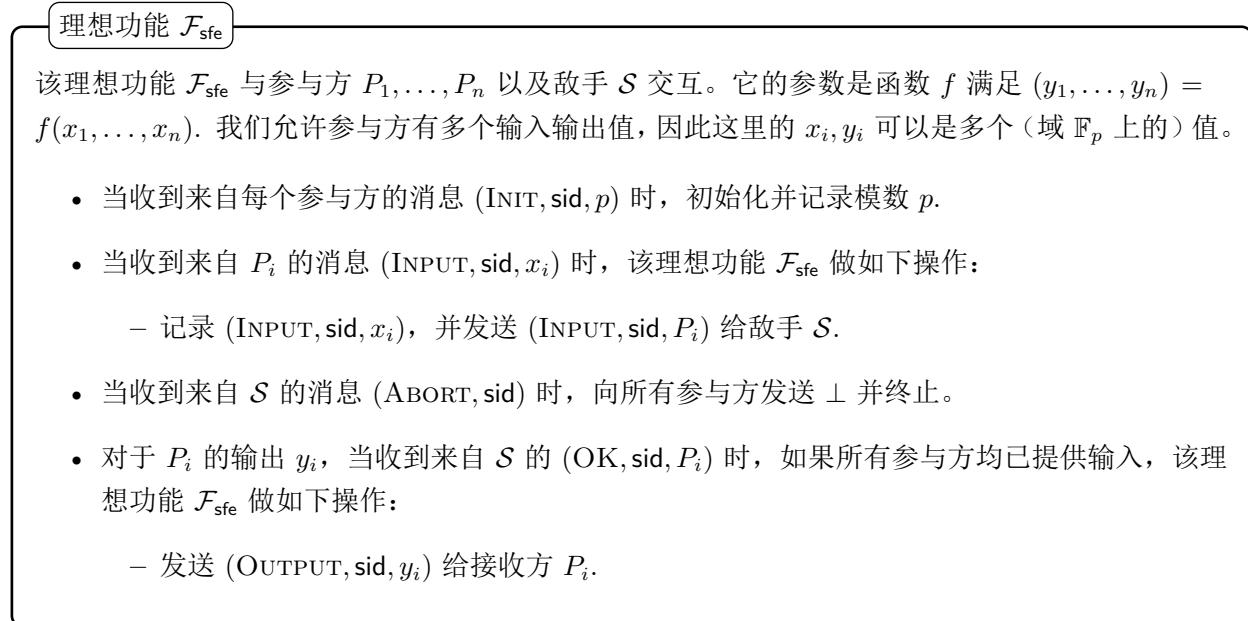


图 9.31: 安全函数计算理想功能  $\mathcal{F}_{\text{sfe}}$

我们有以下定理。

**定理 9.24.** 假设被攻陷方数量不超过  $n - 1$ ，图 9.30 中的协议  $\Pi_{\text{BDOZ}}$  在  $\mathcal{F}_{\text{prep}}$ -混合模型下对于静态恶意敌手  $UC$ -安全实现了理想功能  $\mathcal{F}_{\text{sfe}}$ （图 9.31）。

证明. 要证明该定理，我们需要构造模拟器  $\mathcal{S}$ ，使得对所有环境  $\mathcal{Z}$ ，都有

$$\text{EXEC}_{\Pi_{\text{BDOZ}}, \mathcal{A}, \mathcal{Z}}^{\mathcal{F}_{\text{prep}}} \approx \text{EXEC}_{\mathcal{F}_{\text{sfe}}, \mathcal{S}, \mathcal{Z}}$$

其中  $\mathcal{A}$  是平凡敌手。

**模拟器  $\mathcal{S}$ .** 模拟器  $\mathcal{S}$  在内部运行  $\mathcal{A}$ ，转发  $\mathcal{Z}$  与  $\mathcal{A}$  之间的消息。此外， $\mathcal{S}$  模拟  $\mathcal{F}_{\text{prep}}$  诚实地生成认证 Beaver 三元组和单个值的认证秘密分享。令  $C$  为被攻陷方的集合。 $\mathcal{S}$  的工作方式如下。

- 模拟  $\mathcal{F}_{\text{prep}}$ ：

- 当  $\mathcal{F}_{\text{prep}}$  收到来自被攻陷方  $P_i$  的  $(\text{INIT}, \text{sid}, p)$  时， $\mathcal{S}$  在理想世界代表  $P_i$  向  $\mathcal{F}_{\text{sfe}}$  发送  $(\text{INIT}, \text{sid}, p)$ .

<sup>17</sup>注意，第9.3节的协议虽然也是在恶意敌手模型下的，但是由于存在诚实大多数 ( $t < n/3$ )，因此敌手不能在协议执行时终止协议，也不能阻止诚实方获得输出。

- $\mathcal{S}$  模拟  $\mathcal{F}_{\text{prep}}$  记录每个被攻陷方发送的秘密份额和 MAC 密钥，并为每个诚实方随机选择秘密份额和 MAC 密钥，然后， $\mathcal{S}$  诚实地计算认证 Beaver 三元组和单个值的认证秘密分享，并分发给参与方。

- 输入阶段：

- 对于诚实方  $P_i \notin C$ ，当从  $\mathcal{F}_{\text{sfe}}$  收到  $(\text{INPUT}, \text{sid}, P_i)$  时， $\mathcal{S}$  模拟  $P_i$  以 0 为输入遵循协议执行。如果将  $[a]$  打开给  $P_i$  时 MAC 验证不通过， $\mathcal{S}$  模拟  $P_i$  广播 fail，并在理想世界向  $\mathcal{F}_{\text{sfe}}$  发送  $(\text{ABORT}, \text{sid})$ 。
- 对于被攻陷方  $P_j \in C$ ， $\mathcal{S}$  记录其广播的  $\delta$ 。由于  $\mathcal{S}$  模拟了  $\mathcal{F}_{\text{prep}}$ ，因此知道  $a$  的值。 $\mathcal{S}$  提取  $P_j$  的输入  $x_j = a + \delta$ ，并在理想世界代表  $P_j$  发送给  $\mathcal{F}_{\text{sfe}}$ 。
- 如果任何参与方广播 fail， $\mathcal{S}$  在理想世界向  $\mathcal{F}_{\text{sfe}}$  发送  $(\text{ABORT}, \text{sid})$ 。

- 计算阶段：

- 对于加法门、常数乘法门、常数加法门， $\mathcal{S}$  模拟诚实方遵循协议计算。
- 对于乘法门， $\mathcal{S}$  模拟诚实方遵循协议执行。如果秘密打开时诚实方的 MAC 验证不通过， $\mathcal{S}$  模拟诚实方广播 fail，并在理想世界向  $\mathcal{F}_{\text{sfe}}$  发送  $(\text{ABORT}, \text{sid})$ 。如果任何参与方广播 fail， $\mathcal{S}$  在理想世界向  $\mathcal{F}_{\text{sfe}}$  发送  $(\text{ABORT}, \text{sid})$ 。

- 输出阶段：

- 对于诚实方  $P_i \notin C$  的输出，如果打开时 MAC 验证不通过， $\mathcal{S}$  模拟  $P_i$  广播 fail，并在理想世界阻止  $P_i$  获得输出。否则（MAC 验证通过）， $\mathcal{S}$  向  $\mathcal{F}_{\text{sfe}}$  发送  $(\text{OK}, \text{sid}, P_i)$ 。
- 对于被攻陷方  $P_j \in C$  的输出， $\mathcal{S}$  首先在理想世界向  $\mathcal{F}_{\text{sfe}}$  发送  $(\text{OK}, \text{sid}, P_j)$  得到  $P_j$  的输出  $y$ 。然后， $\mathcal{S}$  需要模拟诚实方的份额使其与  $y$  相一致。 $\mathcal{S}$  的方法是只修改一个诚实方的份额，保持其他诚实方的份额不变。记这个诚实方为  $P_k$ 。 $\mathcal{S}$  计算  $P_k$  的新秘密份额为  $y'_k = y - \sum_{i \neq k} y_i$ （因为  $\mathcal{S}$  模拟了  $\mathcal{F}_{\text{prep}}$ ，所以知道每个参与方的秘密份额）。然后， $\mathcal{S}$  计算  $\text{Mac}_{K_{y_k}^j}(y'_k)$  作为  $y'_k$  的 MAC 标签（因为  $\mathcal{S}$  模拟了  $\mathcal{F}_{\text{prep}}$ ，所以知道  $P_j$  的全局 MAC 密钥  $\Delta_j$ ，并且可以计算本地密钥  $K_{y_k}^j$ ）。最后， $\mathcal{S}$  模拟诚实方向  $P_j$  发送秘密份额和 MAC 标签。

**不可区分性。**首先，当敌手不能成功伪造 MAC 标签时，真实世界与理想世界是完美不可区分的。此时，真实世界与理想世界的区别为：

1. 理想世界中诚实方的输入为 0；
2. 输出阶段，理想世界中的诚实方份额是  $\mathcal{S}$  模拟的。

两者不可区分是因为：

1. 协议中泄漏的值（包括输入分享阶段广播的  $\delta$ ，计算乘法门时打开的  $d, e$ ，以及秘密打开时公布的秘密份额和 MAC 标签）都是均匀随机值，与诚实方的输入无关；
2. 输出阶段  $\mathcal{S}$  模拟的诚实方份额与真实世界的诚实方份额具有完全相同的分布（具体而言，设某个输出为  $y$ ，无论在真实世界还是理想世界，参与方的份额都是在条件  $\sum_{i=1}^n y_i = y$  下的均匀随机值）。

下面，我们论证敌手成功伪造 MAC 标签的概率是可忽略的。协议中 MAC 的定义为  $\text{Mac}_{\Delta, K}(x) = \Delta \cdot x + K$ ，其中  $\Delta, K \leftarrow_{\$} \mathbb{F}_p$ ， $\Delta$  是全局密钥， $K$  是本地一次密钥。敌手持有一些认证秘密份额，记为  $x_1, x_2, \dots$ ，以及它们对应的 MAC 标签，记为  $t_1, t_2, \dots$ （其中  $t_i = \Delta \cdot x_i + K_i$ ，MAC 密钥  $\Delta, K_1, K_2, \dots$  是均匀随机的）；在进行秘密打开时，敌手应当发送的秘密份额一定是  $x_1, x_2, \dots$  的线性组合。对于  $x_i$  的任意线性组合，敌手能够声明一个错误的值但提供合法 MAC 标签的概率为  $1/p$ （形式化地说，假设敌手持有的（正确的）秘密份额和 MAC 标签为  $x$  和  $t$ ，满足  $t = \Delta \cdot x + K$ 。其中 MAC 密钥  $\Delta$  和  $K$  是均匀随机的，且敌手不知道它们的值。如果敌手能提供合法的  $x', t'$  满足  $x' \neq x$  且  $t' = \Delta \cdot x' + K$ ，那么等价于敌手猜对  $\Delta = (t - t')(x - x')^{-1}$ 。）。

证毕。  $\square$

#### 9.4.1.4 预处理阶段的实现

本节，我们要移除可信第三方，通过安全多方计算的方式实现预处理阶段，即实现  $\mathcal{F}_{\text{prep}}$ 。我们介绍 BDOZ 原文<sup>[71]</sup>所提出的基于同态加密 (semi-homomorphic encryption)<sup>18</sup>的方案。

**同态加密定义。** 同态加密方案 PKE 包含三个算法 ( $\text{Gen}, \text{Enc}, \text{Dec}$ )，其中：

- $\text{Gen}(1^\kappa, p)$  是一个随机算法，以安全参数  $\kappa$  和模数  $p$  为输入，输出公私钥对  $(\text{pk}, \text{sk})$  以及参数  $\mathcal{P} = (p, M, R, \mathcal{D}_\sigma^d, \mathbb{G})$ 。其中， $M, R$  是整数， $\mathcal{D}_\sigma^d$  是一个算法，其生成长度为  $d$  的随机向量（作为加密算法的随机数）。我们要求  $\mathcal{D}_\sigma^d$  输出的向量  $\mathbf{r}$  以压倒性的概率满足  $\|\mathbf{r}\|_\infty \leq \sigma$ ，其中  $\|\mathbf{r}\|_\infty = \max(|r_1|, \dots, |r_d|)$ ， $\sigma < R$ 。 $\mathbb{G}$  是密文所在的阿贝尔群。
- $\text{Enc}_{\text{pk}}(x, \mathbf{r})$  是一个确定性算法，以整数  $x \in \mathbb{Z}$  和向量  $\mathbf{r} \in \mathbb{Z}^d$  为输入，输出密文  $C \in \mathbb{G}$ 。当随机数不重要时，我们将其简写为  $\text{Enc}_{\text{pk}}(x)$ 。给定  $C_1 = \text{Enc}_{\text{pk}}(x_1, \mathbf{r}_1)$  和  $C_2 = \text{Enc}_{\text{pk}}(x_2, \mathbf{r}_2)$ ，有  $C_1 + C_2 = \text{Enc}_{\text{pk}}(x_1 + x_2, \mathbf{r}_1 + \mathbf{r}_2)$ 。对于密文  $C$ ，如果存在  $x, \mathbf{r}$  满足  $C = \text{Enc}_{\text{pk}}(x, \mathbf{r})$  且  $|x| \leq \tau$ ,  $\|\mathbf{r}\|_\infty \leq \rho$ ，那么我们说  $C$  是一个  $(\tau, \rho)$ -密文。
- $\text{Dec}_{\text{sk}}(C)$  是一个确定性算法，以密文  $C \in \mathbb{G}$  为输入，输出  $x' \in \mathbb{Z}_p \cup \{\perp\}$ 。

**正确性：**如果同态加密方案 PKE 对于任意的  $p$  满足

$$\Pr \left[ \begin{array}{l} (\text{pk}, \text{sk}, \mathbb{P}) \leftarrow \text{Gen}(1^\kappa, p); \\ x \in \mathbb{Z}, |x| \leq M; \mathbf{r} \in \mathbb{Z}^d, \|\mathbf{r}\|_\infty \leq R \end{array} : \text{Dec}_{\text{sk}}(\text{Enc}_{\text{pk}}(x, \mathbf{r})) \neq x \bmod p \right] < \text{negl}(\kappa),$$

那么我们说它满足正确性。

**IND-CPA 安全性：**同态加密方案 PKE 的 IND-CPA 安全性如定义 2.1.1 所示。

在实际使用中，Paillier<sup>[73]</sup>是一个著名的同态加密方案。

**零知识证明。** 预处理阶段协议需要用到两个零知识证明协议：

- $\Pi_{\text{PoPK}}$ (Proof of Plaintext Knowledge): 用于证明知道密文所加密的明文。
- $\Pi_{\text{PoCM}}$ (Proof of Correct Multiplication): 用于证明正确地进行了乘法运算。

<sup>18</sup>在学术界，同态加密 (homomorphic encryption) 通常指满足加法同态或乘法同态的加密方案，全同态加密 (fully homomorphic encryption) 指既满足加法同态又满足乘法同态的方案；在工业界，前者通常被称为半同态加密 (partially homomorphic encryption)。这里的 semi-homomorphic encryption 实际上是 homomorphic encryption，但额外要求被加密的值不能“太大”。我们仍将 semi-homomorphic encryption 称为同态加密而不是半同态加密，以避免混淆。

形式化地说,  $\Pi_{\text{PoPK}}$  的陈述 (statement) 是公钥  $\text{pk}_P$  和  $u$  个密文  $C_k$ ,  $k = 1, \dots, u$ . 证明者证明所有密文  $C_k$  都是  $(2^{2u+\log u}\tau, 2^{2u+\log u}\rho)$ -密文。也就是说,  $\Pi_{\text{PoPK}}$  证明的关系为:

$$\begin{aligned} \mathcal{R}_{\text{PoPK}}^{(u,\tau,\rho)} = & \{(x, w) \mid x = (\text{pk}_P, C_1, \dots, C_u); \\ & w = ((x_1, \mathbf{r}_1), \dots, (x_u, \mathbf{r}_u)) : C_k = \text{Enc}_{\text{pk}_P}(x_k, \mathbf{r}_k), \\ & |x_k| \leq 2^{2u+\log u}\tau, \|\mathbf{r}_k\|_\infty \leq 2^{2u+\log u}\rho\} \end{aligned}$$

$\Pi_{\text{PoCM}}$  的陈述是公钥  $\text{pk}_P, \text{pk}_V$  和  $u$  个密文三元组  $(A_k, B_k, C_k)$ ,  $k = 1, \dots, u$ , 其中  $A_k$  用  $\text{pk}_P$  加密,  $B_k, C_k$  用  $\text{pk}_V$  加密 (因此它们属于  $\mathbb{G}_V$ )。证明者证明  $A_k = \text{Enc}_{\text{pk}_P}(a_k, \mathbf{h}_k)$ ,  $C_k = a_k B_k + \text{Enc}_{\text{pk}_V}(r_k)$ , 其中  $r_k$  为随机数。也就是说,  $\Pi_{\text{PoCM}}$  证明的关系为:

$$\begin{aligned} \mathcal{R}_{\text{PoCM}}^{(u,\tau,\rho)} = & \{(x, w) \mid x = (\text{pk}_P, \text{pk}_V, (A_1, B_1, C_1), \dots, (A_u, B_u, C_u)); \\ & w = ((a_1, \mathbf{h}_1, r_1, \mathbf{t}_1), \dots, (a_u, \mathbf{h}_u, r_u, \mathbf{t}_u)) : \\ & A_k = \text{Enc}_{\text{pk}_P}(a_k, \mathbf{h}_k), B_k \in \mathbb{G}_V, C_k = a_k B_k + \text{Enc}_{\text{pk}_V}(r_k, \mathbf{t}_k), \\ & |a_k| \leq 2^{3u+\log u}\tau, \|\mathbf{h}_k\|_\infty \leq 2^{3u+\log u}\rho, \\ & |r_k| \leq 2^{4u+\log u}\tau^2, \|\mathbf{t}_k\|_\infty \leq 2^{4u+\log u}\tau\rho\} \end{aligned}$$

我们暂时将这两个零知识证明协议当作黑盒, 不关注其实现细节。

**直观思想。**在第7.2.2节, 我们介绍过半诚实场景下 Beaver 三元组的分布式生成方式。假设参与方是半诚实的, 基于同态加密方案, 参与方显然可以生成 Beaver 三元组<sup>19</sup>。我们用符号  $\langle \cdot \rangle$  表示未认证的秘密分享, 即

$$\langle a \rangle = \{a_1, \dots, a_n\}$$

其中  $\sum_{i=1}^n a_i = a$ . 于是, 参与方可以首先生成未认证的 Beaver 三元组  $\langle a \rangle, \langle b \rangle, \langle c \rangle$ .

接下来, 参与方需要对秘密份额进行认证。参与方可以自行选择自己的全局 MAC 密钥和本地 MAC 密钥。然后, 对于每一个份额, 参与方需要获得其 MAC 标签  $M_j(a_i) = \text{Mac}_{\Delta_j, K_{a_i}^j}(a_i) = \Delta_j \cdot a_i + K_{a_i}^j$ , 其中  $P_i$  知道  $a_i$ ,  $P_j$  知道  $\Delta_j, K_{a_i}^j$ . 由于 MAC 标签的计算实质上也是 OLE, 我们似乎可以再次基于同态加密来实现。

然而, 请注意, 我们这里考虑的是恶意敌手场景, 敌手可以任意地违背协议。因此, 敌手不一定会将正确的秘密份额  $a_i$  作为 OLE 的输入。具体而言, 对于被攻陷方  $P_j$ , 假设他秘密份额是  $a_j$ , 在为  $a_j$  计算 MAC 标签时, 他首先可以向不同的参与方发送不同的值, 此时, 认证秘密分享  $[a]$  就不是被良好定义的。

为此, 我们可以要求  $P_j$  先对  $a_j$  进行承诺 (或者加密, 加密也是一种承诺的方式), 从而强制  $P_j$  使用对不同参与方使用同样的值。但是, 对于认证 Beaver 三元组, 敌手仍然存在攻击的方式。考虑未认证的 Beaver 三元组  $\langle a \rangle, \langle b \rangle, \langle c \rangle$ , 假设  $P_j$  持有  $a_j, b_j, c_j$ , 满足  $\sum a_i = a$ ,  $\sum b_i = b$ ,  $\sum c_i = c$ ,  $c = ab$ . 恶意的  $P_j$  可以不对  $c_j$  进行承诺, 而是将其替换为  $c'_j$ , 然后其他参与方都会对份额  $c'_j$  进行认证。这就使参与方实际上分享了  $c' = c_1 + \dots + c'_j + \dots + c_n \neq c$ , 于是破坏了条件  $c = ab$ , 也破坏了协议的正确性。

<sup>19</sup>如果读者不清楚如何生成, 可以回顾一下第7.2.2节以及第6.2.3.1节 “OLE 的实现”。

上述攻击表明，敌手有能力对 Beaver 三元组中的  $[c]$  制造偏移量  $\Delta$ ，使得  $c = ab + \Delta$ 。为了解决这个问题，协议通过“消耗”一个 Beaver 三元组来验证另一个 Beaver 三元组的正确性。具体如下。参与方取（不一定正确的）认证 Beaver 三元组  $([a], [b], [c])$  和  $([f], [g], [h])$ ，他们希望验证  $c = ab$ 。首先，参与方产生（长度为  $u$  位）随机挑战值  $e$ ，然后（可验证地）打开  $\varepsilon = e[a] - [f]$ ,  $\delta = [b] - [g]$ 。最后，参与方打开  $e[c] - [h] - \delta[f] - \varepsilon[g] - \varepsilon\delta$ ，如果为 0，那么输出  $([a], [b], [c])$ ，否则，丢弃  $([a], [b], [c])$ 。上述检查的思路是：验证的式子  $ec - h - \delta f - \varepsilon g - \varepsilon\delta$  等于  $e(c - ab) - (h - fg)$ ，如果  $c = ab$  且  $h = fg$ ，那么验证一定通过；如果  $c \neq ab$ ，那么验证通过等价于  $e = (h - fg)/(c - ab)$ ， $e$  只有一个值能够满足该式，而  $e$  是长度为  $u$  的随机值，故上式成立的概率为  $2^{-u}$ 。

**协议描述。**下面，我们给出预处理阶段的协议描述。

预处理阶段需要用到两个理想功能：

- $\mathcal{F}_{\text{KeyReg}}$ : 用于参与方注册公私钥，如图 9.32 所示。
- $\mathcal{F}_{\text{Rand}}$ : 用于生成随机挑战值，如图 9.33 所示。

**理想功能  $\mathcal{F}_{\text{KeyReg}}$**

该理想功能  $\mathcal{F}_{\text{KeyReg}}$  的参数是密钥生成算法  $\text{Gen}$  和安全参数  $1^\kappa$ 。

**诚实方注册：**当从诚实方  $P_i$  收到  $(\text{REG}, \text{sid}, p)$  时， $\mathcal{F}_{\text{KeyReg}}$  计算  $(\text{pk}_i, \text{sk}_i) \leftarrow \text{Gen}(1^\kappa, p)$ ，然后向其他参与方  $P_j$ ,  $j \neq i$  发送  $(\text{REGISTERED}, \text{sid}, P_i, \text{pk}_i, \perp)$ ，向  $P_i$  发送  $(\text{REGISTERED}, \text{sid}, P_i, \text{pk}_i, \text{sk}_i)$ 。

**被攻陷方注册：**当从被攻陷方  $P_i$  收到  $(\text{REG}, \text{sid}, p, r^*)$  时， $\mathcal{F}_{\text{KeyReg}}$  执行同样的操作，但是将  $r^*$  作为密钥生成算法  $\text{Gen}$  的随机数。

图 9.32: 理想功能  $\mathcal{F}_{\text{KeyReg}}$

**理想功能  $\mathcal{F}_{\text{Rand}}$**

当从所有参与方收到  $(\text{RAND}, \text{sid}, u)$  时， $\mathcal{F}_{\text{Rand}}$  选取  $r \leftarrow_{\$} \{0, 1\}^u$  并向所有参与方发送  $(\text{RAND}, \text{sid}, r)$ 。

图 9.33: 理想功能  $\mathcal{F}_{\text{Rand}}$

我们将预处理阶段协议  $\Pi_{\text{prep}}$  的一些部分写成子协议，它们分别是：

- $\Pi_{\text{share}}$ : 用于生成随机值的（未认证的）加法秘密分享，如图 9.34 所示。
- $\Pi_{2\text{-mult}}$ : 用于两个参与方计算乘法，即 OLE，如图 9.35 所示。
- $\Pi_{n\text{-mult}}$ : 用于  $n$  个参与方两两计算乘法，构造（未认证的）Beaver 三元组，如图 9.36 所示。
- $\Pi_{\text{addMacs}}$ : 用于计算 MAC 标签，构造认证秘密分享，如图 9.37 所示。

最后，我们给出预处理阶段协议  $\Pi_{\text{prep}}$  的描述，如图 9.38 所示。

子协议  $\Pi_{\text{share}}$

**Share( $u$ ):**

1. 每个参与方  $P_i$  随机选择  $x_{k,i} \leftarrow \mathbb{F}_p$ ,  $k = 1, \dots, u$ , 然后广播  $(\tau, \rho)$ -密文  $\{\text{Enc}_{\text{pk}_i}(x_{k,i})\}_{k=1}^u$ .
2. 每两个参与方  $P_i, P_j (i \neq j)$  以  $\{\text{Enc}_{\text{pk}_i}(x_{k,i})\}_{k=1}^u$  为输入运行  $\Pi_{\text{PoPK}}(u, \tau, \rho)$ ,  $P_i$  为证明者, 证明密文是  $(2^{2u+\log u}\tau, 2^{2u+\log u}\rho)$ -密文。
3. 所有参与方输出  $\langle x_k \rangle = (\text{Enc}_{\text{pk}_1}(x_{k,1}), \dots, \text{Enc}_{\text{pk}_n}(x_{k,n}))$ ,  $k = 1, \dots, u$ , 其中  $x_k = \sum_{i=1}^n x_{k,i} \bmod p$ .  $P_i$  存储  $x_{k,i}$  以及加密使用的随机数, 作为私有输出。

图 9.34: 子协议  $\Pi_{\text{share}}$ 

子协议  $\Pi_{2\text{-mult}}$

**2-Mult( $u, \tau, \rho$ ):**

1.  $P_i$  和  $P_j$  以  $(\tau, \rho)$ -密文  $\{\text{Enc}_{\text{pk}_i}(x_k)\}_{k=1}^u, \{\text{Enc}_{\text{pk}_j}(y_k)\}_{k=1}^u$  为输入 (此时它们已被证明是  $(2^{2u+\log u}\tau, 2^{2u+\log u}\rho)$ -密文)。
2. 对于  $k = 1, \dots, u$ ,  $P_i$  向  $P_j$  发送  $C_k = x_k \text{Enc}_{\text{pk}_j}(y_k) + \text{Enc}_{\text{pk}_j}(r_k)$ , 其中  $\text{Enc}_{\text{pk}_j}(r_k)$  是用  $\text{pk}_j$  加密的随机  $(2^{3u+\log u}\tau^2, 2^{3u+\log u}\tau\rho)$ -密文。 $P_i$  以  $\text{Enc}_{\text{pk}_i}(x_k), \text{Enc}_{\text{pk}_j}(y_k), C_k$  作为输入调用  $\Pi_{\text{PoCM}}(u, \tau, \rho)$  证明  $C_k$  的计算是正确的。
3. 对于  $k = 1, \dots, u$ ,  $P_j$  解密  $C_k$  得到  $v_k$ , 并输出  $z_{k,j} = v_k \bmod p$ .  $P_i$  输出  $z_{k,i} = -r_k \bmod p$ .

图 9.35: 子协议  $\Pi_{2\text{-mult}}$ 

子协议  $\Pi_{n\text{-mult}}$

**n-Mult( $u$ ):**

1. 协议输入是子协议  $\Pi_{\text{share}}$  生成的  $\langle a_k \rangle, \langle b_k \rangle$ ,  $k = 1, \dots, u$ . 每个参与方  $P_i$  初始化变量  $c_{k,i} = a_{k,i}b_{k,i} \bmod p$ ,  $k = 1, \dots, u$ .
2. 每两个参与方  $P_i, P_j (i \neq j)$  以  $\text{Enc}_{\text{pk}_i}(a_{k,i}), \text{Enc}_{\text{pk}_j}(b_{k,j})$ ,  $k = 1, \dots, u$  为输入运行  $\Pi_{2\text{-mult}}$ , 并将输出加在  $c_{k,i}, c_{k,j}$  上, 即, 令  $z_{k,i}, z_{k,j}$  分别是  $P_i, P_j$  在协议  $\Pi_{2\text{-mult}}$  中的输出,  $P_i$  设置  $c_{k,i} = c_{k,i} + z_{k,i} \bmod p$ ,  $P_j$  设置  $c_{k,j} = c_{k,j} + z_{k,j} \bmod p$ .
3. 每个参与方  $P_i$  调用  $\Pi_{\text{share}}$ , 广播  $\text{Enc}_{\text{pk}_i}(c_{k,i})$ ,  $k = 1, \dots, u$ . 参与方输出子协议  $\Pi_{\text{share}}$  的输出, 即  $\langle c_k \rangle$ ,  $k = 1, \dots, u$ .

图 9.36: 子协议  $\Pi_{n\text{-mult}}$

### 子协议 $\Pi_{\text{addMacs}}$

**初始化:** 每个参与方  $P_i$  随机选取  $\Delta_i \leftarrow_{\$} \mathbb{F}_p$ , 然后向每个其他参与方  $P_j, j \neq i$  发送  $\text{Enc}_{\text{pk}_i}(\Delta_i)$  并以  $(\text{Enc}_{\text{pk}_i}(\Delta_i), \dots, \text{Enc}_{\text{pk}_i}(\Delta_i))$  为输入运行  $\Pi_{\text{PoPK}}(u, \tau, \rho)$ , 证明它是  $(2^{2u+\log u}\tau, 2^{2u+\log u}\rho)$ -密文。

#### **AddMacs( $u$ ):**

1. 协议输入为  $\langle a_k \rangle, k = 1, \dots, u$ . 每个参与方  $P_i$  持有  $a_k$  的份额  $a_{k,i}$ .  $P_i$  将存储  $a_{k,i}$  作为  $[a_k]$  的一部分。
2. 每两个参与方  $P_i, P_j (i \neq j)$  调用  $\Pi_{\text{2-mult}}(u, \tau, \rho)$ ,  $P_i$  的输入为  $\text{Enc}_{\text{pk}_i}(\Delta_i), \dots, \text{Enc}_{\text{pk}_i}(\Delta_i)$ ,  $P_j$  的输入是  $\text{Enc}_{\text{pk}_j}(a_{k,j}), k = 1, \dots, u$ . 然后,  $P_i$  获得  $z_{k,i}$ ,  $P_j$  获得  $z_{k,j}$ . 因为  $\Pi_{\text{2-mult}}$  保证了  $\Delta_i \cdot a_{k,j} = z_{k,i} + z_{k,j}$ , 因此  $P_i$  存储用于认证  $a_{k,j}$  的 MAC 密钥  $K_{a_{k,j}}^i = -z_{k,i} \bmod p$ ,  $P_j$  存储  $a_{k,j}$  的 MAC 标签  $M_i(a_{k,j}) = z_{k,j}$ .

图 9.37: 子协议  $\Pi_{\text{addMacs}}$

### $\{\mathcal{F}_{\text{KeyReg}}, \mathcal{F}_{\text{Rand}}\}$ -混合模型下的协议 $\Pi_{\text{prep}}$

**初始化:** 参与方首先向  $\mathcal{F}_{\text{KeyReg}}$  发送  $(\text{REG}, \text{sid}, p)$  注册公私钥, 然后调用  $\Pi_{\text{addMacs}}$  的初始化过程。

#### **Triples( $u$ ):**

1. 参与方首先调用 4 次  $\Pi_{\text{share}}$ , 获得  $\{\langle a_k \rangle, \langle b_k \rangle, \langle f_k \rangle, \langle g_k \rangle\}_{k=1}^u$ .
2. 参与方对  $\{\langle a_k \rangle, \langle b_k \rangle\}_{k=1}^u$  和  $\{\langle f_k \rangle, \langle g_k \rangle\}_{k=1}^u$  分别调用  $\Pi_{\text{n-mult}}$ , 获得  $\{\langle c_k \rangle\}_{k=1}^u$  和  $\{\langle h_k \rangle\}_{k=1}^u$ .
3. 参与方对每个秘密份额调用  $\Pi_{\text{addMacs}}$ , 获得  $\{[a_k], [b_k], [c_k], [f_k], [g_k], [h_k]\}_{k=1}^u$ .
4. 参与方通过“消耗”  $\{[f_k], [g_k], [h_k]\}$  来检查  $c_k = a_k b_k$  是否满足: 首先, 参与方调用  $\mathcal{F}_{\text{Rand}}$  获得长度为  $u$  位的随机挑战值  $e$ . 然后, 参与方打开  $e[a_k] - [f_k]$  记为  $\varepsilon_k$ , 打开  $[b_k] - [g_k]$  记为  $\delta_k$ , 最后打开  $e[c_k] - [h_k] - \delta_k[f_k] - \varepsilon_k[g_k] - \varepsilon_k\delta_k$ , 检查是否为 0. 如果这个值为 0, 参与方输出  $\{[a_k], [b_k], [c_k]\}_{k=1}^u$ .

#### **Singles( $u$ ):**

1. 参与方调用  $\Pi_{\text{share}}$ , 获得  $\{\langle a_k \rangle\}_{k=1}^u$ .
2. 参与方对  $\{\langle a_k \rangle\}_{k=1}^u$  调用  $\Pi_{\text{addMacs}}$ , 获得  $\{[a_k]\}_{k=1}^u$  并输出。

图 9.38:  $\{\mathcal{F}_{\text{KeyReg}}, \mathcal{F}_{\text{Rand}}\}$ -混合模型下的协议  $\Pi_{\text{prep}}$

为了正确解密，我们对同态加密方案的参数有一定的要求。如果一个同态加密方案能确保所有的密文都能被正确地解密，那么我们称这个同态加密方案是可容许的 (*admissible*)，即  $M \geq 2^{5u+2\log u}\tau^2$ ,  $R \geq 2^{4u+\log u}\tau\rho$ .

我们有以下安全性定理。

**定理 9.25.** 假设被攻陷方数量不超过  $n - 1$ ，假设使用的加密方案是模  $p$  同态的 (*semi-homomorphic*)，可容许的，且具有 *IND-CPA* 安全性，那么图 9.38 中的协议  $\Pi_{\text{prep}}$  在  $\{\mathcal{F}_{\text{KeyReg}}, \mathcal{F}_{\text{Rand}}\}$ -混合模型下对于静态恶意对手 *UC*-安全实现了理想功能  $\mathcal{F}_{\text{prep}}$  (图 9.29)。

该定理的证明通过归约到同态加密方案的一个性质，较为抽象，感兴趣的读者可以自行阅读文献<sup>[71]</sup>。

最后，为了协议描述的完整性，我们给出零知识证明协议  $\Pi_{\text{PoPK}}$  和  $\Pi_{\text{PoCM}}$  的构造，如图 9.39 和图 9.40 所示。关于这两个协议的完备性、可靠性、零知识性的证明可参阅文献<sup>[71]</sup>。

零知识证明协议  $\Pi_{\text{PoPK}}$

**PoPK**( $u, \tau, \rho$ ):

1. 输入是  $u$  个密文  $\{C_k = \text{Enc}_{\text{pk}_P}(x_k, \mathbf{r}_k)\}_{k=1}^u$ . 定义向量  $\mathbf{c} = (C_1, \dots, C_u)$ ,  $\mathbf{x} = (x_1, \dots, x_u)$  以及矩阵  $\mathbf{R} = (\mathbf{r}_1, \dots, \mathbf{r}_u)$ , 其中  $\mathbf{r}_k$  为矩阵的一行。
2. 定义  $m = 2u - 1$ . 证明者  $P$  构造  $m$  个  $(2^{u-1+\log u}\tau, 2^{u-1+\log u}\rho)$ -密文  $\{A_i = \text{Enc}_{\text{pk}_P}(y_i, \mathbf{s}_i)\}_{i=1}^m$  并发送给验证者  $V$ . 与第 1 步中相同，定义向量  $\mathbf{a} = (A_1, \dots, A_m)$ ,  $\mathbf{y} = (y_1, \dots, y_m)$  以及矩阵  $\mathbf{S} = (\mathbf{s}_1, \dots, \mathbf{s}_m)$ , 其中  $\mathbf{s}_k$  为矩阵的一行。
3.  $V$  均匀随机选择向量  $\mathbf{e} = (e_1, \dots, e_u) \leftarrow \$_{\{0,1\}^u}$  并发送给  $P$ .
4.  $P$  计算  $\mathbf{z} = \mathbf{y} + \mathbf{M}_e \cdot \mathbf{x}$  和  $\mathbf{T} = \mathbf{S} + \mathbf{M}_e \cdot \mathbf{R}$  并发送给  $V$ , 其中  $\mathbf{M}_e$  是一个  $m \times u$  的矩阵，对于  $1 \leq i - k + 1 \leq u$ ,  $\mathbf{M}_e$  的第  $(i, k)$  个位置  $\mathbf{M}_{e,i,k} = \mathbf{e}_{i-k+1}$ , 其他位置为 0.
5.  $V$  检查  $\mathbf{d} \stackrel{?}{=} \mathbf{a} + \mathbf{M}_e \cdot \mathbf{c}$ , 其中  $\mathbf{d} = (\text{Enc}_{\text{pk}_P}(z_1, \mathbf{t}_1), \dots, \text{Enc}_{\text{pk}_P}(z_m, \mathbf{t}_m))$ . 此外， $V$  检查  $|z_i| \leq 2^{u-1+\log u}\tau$  和  $\|\mathbf{t}_i\|_\infty \leq 2^{u-1+\log u}\rho$ .

图 9.39: 零知识证明协议  $\Pi_{\text{PoPK}}$

## 9.4.2 SPDZ 协议

BDOZ 协议中，参与方需要两两检验对方的秘密份额的 MAC 标签，因此存储量与参与方的数量成线性关系。SPDZ 协议（读作“speedz”）由 Damgård, Pastro, Smart, Zakarias 提出<sup>[72]</sup>，协议通过更巧妙的设计，将认证秘密份额的存储量降低为常数级。

### 9.4.2.1 直观思想

对秘密份额认证并无必要。BDOZ 协议通过对秘密份额进行认证来确保秘密值打开时的正确性。然而，我们实际上并不关心秘密份额是否正确，我们只需要打开的秘密值是正确的就可以了。基于这个观察，

### 零知识证明协议 $\Pi_{\text{PoCM}}$

**PoCM**( $u, \tau, \rho$ ):

1. 输入是  $u$  个密文三元组  $\{(A_k, B_k, C_k)\}_{k=1}^u$ , 其中  $A_k = \text{Enc}_{\text{pk}_P}(a_k, \mathbf{h}_k)$ ,  $C_k = a_k B_k + \text{Enc}_{\text{pk}_V}(r_k, \mathbf{t}_k)$ .
2. 证明者  $P$  构造  $u$  个均匀随机的  $(2^{3u-1+\log u}\tau, 2^{3u-1+\log u}\rho)$ -密文  $\{D_k = \text{Enc}_{\text{pk}_P}(d_k, \mathbf{s}_k)\}_{k=1}^u$  和  $u$  个密文  $F_k = d_k B_k + \text{Enc}_{\text{pk}_V}(f_k, \mathbf{y}_k)$ ,  $k = 1, \dots, u$  并发送给验证者  $V$ , 其中  $\text{Enc}_{\text{pk}_V}(f_k, \mathbf{y}_k)$  是均匀随机的  $(2^{4u-1+\log u}\tau^2, 2^{4u-1+\log u}\tau\rho)$ -密文。
3.  $V$  均匀随机选择向量  $\mathbf{e} = (e_1, \dots, e_u) \leftarrow \$_{\{0, 1\}^u}$  并发送给  $P$ .
4.  $P$  向  $V$  发送  $\{(z_k, \mathbf{v}_k)\}_{k=1}^u$  和  $\{(x_k, \mathbf{w}_k)\}_{k=1}^u$ , 其中  $z_k = d_k + e_k a_k$ ,  $\mathbf{v}_k = \mathbf{s}_k + e_k \mathbf{h}_k$ ,  $x_k = f_k + e_k r_k$ ,  $\mathbf{w}_k = \mathbf{y}_k + e_k \mathbf{t}_k$ .
5.  $V$  检查  $D_k + e_k A_k \stackrel{?}{=} \text{Enc}_{\text{pk}_P}(z_k, \mathbf{v}_k)$  和  $F_k + e_k C_k \stackrel{?}{=} z_k B_k + \text{Enc}_{\text{pk}_V}(x_k, \mathbf{w}_k)$ . 此外,  $V$  检查  $|z_k| \leq 2^{3u-1+\log u}\tau$ ,  $\|\mathbf{v}_k\|_\infty \leq 2^{3u-1+\log u}\rho$ ,  $|x_k| \leq 2^{4u-1+\log u}\tau^2$ ,  $\|\mathbf{w}_k\|_\infty \leq 2^{4u-1+\log u}\tau\rho$ .
6. 步骤 2~5 并行重复  $u$  次。

图 9.40: 零知识证明协议  $\Pi_{\text{PoCM}}$

SPDZ 协议直接对秘密值进行认证, 它的消息认证码定义为  $\text{Mac}_\Delta(x) = \Delta \cdot x$ , 其中  $x$  是秘密值,  $\Delta$  是(全局) MAC 密钥。

不过, 这样的改变带来了一个“先有鸡还是先有蛋”的问题——参与方必须知道 MAC 密钥才能检验 MAC 标签, 但是知道了 MAC 密钥就可以任意地伪造 MAC 标签。为了解决这个问题, MAC 密钥  $\Delta$  是被秘密分享的, 参与方在重建密钥  $\Delta$  之前必须先对秘密份额和 MAC 标签进行承诺(commit)。这样一来, 基于承诺的绑定性, 当敌手知道  $\Delta$  时已经无法修改秘密份额和 MAC 标签, 也就无法伪造 MAC 标签了。

也就是说, SPDZ 协议中, 秘密值  $x$  是被分享的, MAC 密钥  $\Delta$  是被分享的, 由于  $\text{Mac}_\Delta(x) = \Delta \cdot x$  即  $\Delta = \text{Mac}_\Delta(x)/x$ , 为了使打开秘密值时不泄露  $\Delta$  (计算乘法门时要进行秘密打开), MAC 密钥也需要是被分享的。我们用符号  $\langle \cdot \rangle$  来表示 SPDZ 认证秘密分享<sup>20</sup>, 它的定义如下:

$$\langle x \rangle = ((x_1, \dots, x_n), (M(x)_1, \dots, M(x)_n))$$

其中  $x = x_1 + \dots + x_n$ ,  $M(x)_1, \dots, M(x)_n$  是  $\text{Mac}_\Delta(x)$  的加法秘密分享, 即  $M(x)_1 + \dots + M(x)_n = \text{Mac}_\Delta(x) = \Delta \cdot x$ ,  $\Delta = \Delta_1 + \dots + \Delta_n$ ,  $P_i$  持有  $(x_i, M(x)_i)$  和 MAC 密钥的份额  $\Delta_i$ .<sup>21</sup> 这样一来, 参与方对一个秘密份额的存储量降为了常数级。

对于认证秘密分享  $\langle x \rangle = ((x_1, \dots, x_n), (M(x)_1, \dots, M(x)_n))$ ,  $\langle x' \rangle = ((x'_1, \dots, x'_n), (M(x')_1, \dots, M(x')_n))$

<sup>20</sup> 本节中使用的符号含义与上一节不同, 请读者避免混淆。

<sup>21</sup> 在原始论文<sup>[72]</sup>中, 秘密分享的定义还包含了一个公开值  $\delta$ . 但这个值是不必要的, 因此本教材的描述中将其删除了。

和常数  $\alpha \in \mathbb{F}_p$ , 定义加法、常数乘法、常数加法运算如下:

$$\begin{aligned}\langle x \rangle + \langle x' \rangle &= ((x_1 + x'_1, \dots, x_n + x'_n), (M(x)_1 + M(x')_1, \dots, M(x)_n + M(x')_n)) \\ \alpha \langle x \rangle &= ((\alpha x_1, \dots, \alpha x_n), (\alpha \cdot M(x)_1, \dots, \alpha \cdot M(x)_n)) \\ \langle x \rangle + \alpha &= ((x_1 + \alpha, \dots, x_n), (M(x)_1 + \Delta_1 \alpha, \dots, M(x)_n + \Delta_n \alpha))\end{aligned}$$

上述运算满足

$$\begin{aligned}\langle x \rangle + \langle x' \rangle &= \langle x + x' \rangle \\ \alpha \langle x \rangle &= \langle \alpha x \rangle \\ \langle x \rangle + \alpha &= \langle x + \alpha \rangle\end{aligned}$$

于是, 我们可以对 SPDZ 认证秘密分享进行线性运算, 并通过 Beaver 三元组技术计算乘法门。部分打开。回想一下, 在使用 Beaver 三元组计算乘法门时, 假设两根输入导线上的值是  $x$  和  $y$ , 计算过程是这样的:

- 取认证 Beaver 三元组  $(\langle a \rangle, \langle b \rangle, \langle c \rangle)$ , 参与方本地计算  $\langle x \rangle - \langle a \rangle = \langle x - a \rangle, \langle y \rangle - \langle b \rangle = \langle y - b \rangle$ ;
- 向所有参与方打开  $d = x - a, e = y - b$ ;
- 参与方本地计算  $d\langle b \rangle + e\langle a \rangle + \langle c \rangle + de = \langle xy \rangle$ .

注意到, 计算过程中需要打开  $d = x - a, e = y - b$ . 对于 SPDZ 认证秘密分享, 检验打开的秘密值需要重建 MAC 密钥  $\Delta$ , 但是一旦 MAC 密钥被重建, 之后敌手就可以任意伪造 MAC 标签了。也就是说, 如果重建了 MAC 密钥  $\Delta$ , 之后的乘法门计算和输出重建就无法保证正确性。

为了解决这个问题, SPDZ 协议对乘法门中需要打开的值做“部分打开”, 即, 只重建秘密值, 不重建 MAC 标签和 MAC 密钥。在协议最后的输出阶段, 对所有打开过的值进行检验(这意味着协议过程中, 参与方不一定正在对正确的值进行计算)。具体方式如下:

- 假设乘法门计算时需要打开  $\langle d \rangle$ , 参与方执行“部分打开”:  $P_i$  将  $d_i$  发送给  $P_1, P_1$  计算  $d = \sum_{i=1}^n d_i$  并广播。
- 输出阶段, 假设  $a_1, \dots, a_T$  是所有被部分打开的值, 其中  $\langle a_j \rangle = ((a_{j,1}, \dots, a_{j,n}), (M(a_j)_1, \dots, M(a_j)_n))$ . 参与方取一个随机值  $e$ , 对于  $i = 1, \dots, T$  设置  $e_i = e^i$ , 并计算  $a = \sum_{j=1}^T e_j a_j$ .
- $P_i$  计算  $M_i = \sum_{j=1}^T e_j \cdot M(a_j)_i$  并对这个值进行承诺。
- 参与方打开全局 MAC 密钥  $\Delta$ .
- $P_i$  打开承诺值  $M_i$ . 参与方检查  $\Delta \cdot a \stackrel{?}{=} \sum_{i=1}^n M_i$ . 若不满足, 则终止协议。

**MAC 密钥的打开。** 上述过程需要(可验证地)打开 MAC 密钥  $\Delta$ . 也就是说,  $\Delta$  的打开也需要让所有参与方验证正确性。因此,  $\Delta$  不能采用符号  $\langle \cdot \rangle$  定义的方式进行秘密分享。我们定义符号  $\llbracket \cdot \rrbracket$  如下:

$$\llbracket \Delta \rrbracket = ((\Delta_1, \dots, \Delta_n), (\beta_i, M(\Delta)_1^i, \dots, M(\Delta)_n^i)_{i \in [n]})$$

其中  $\Delta = \Delta_1 + \dots + \Delta_n$ ,  $M(\Delta)_1^1 + \dots + M(\Delta)_n^n = \beta_i \Delta$ . 参与方  $P_i$  持有  $\Delta_i, \beta_i, M(\Delta)_1^i, \dots, M(\Delta)_n^i$ . 这里,  $M(\Delta)_i = M(\Delta)_i^1 + \dots + M(\Delta)_i^n = \text{Mac}_{\beta_i}(\Delta) = \beta_i \Delta$  是  $\Delta$  在  $P_i$  的私有 MAC 密钥  $\beta_i$  下的

MAC 标签。因此， $\llbracket \Delta \rrbracket$  可以向任一参与方可验证地打开：如果要向  $P_i$  打开  $\llbracket \Delta \rrbracket$ ，每个参与方  $P_j$  发送  $\Delta_j, M(\Delta)_j^j$  给  $P_i$ ，然后  $P_i$  计算  $\Delta = \Delta_1 + \dots + \Delta_n$  并检查  $\sum_{j=1}^n M(\Delta)_j^j = \beta_i \Delta$ （如果要打开给所有人，那么对每个参与方执行该操作）。注意，对于  $\langle \cdot \rangle$  秘密分享，每个参与方的存储量是常数；对于  $\llbracket \cdot \rrbracket$  秘密分享，每个参与方的存储量是线性的。

**输入分享。**我们还需要解决输入分享的问题。假设  $P_i$  的输入为  $x_i$ ，在 BDOZ 协议中，参与方取一个随机值的秘密分享  $[a]$  并打开给  $P_i$ ，然后  $P_i$  广播  $\delta = x_i - a$ ，参与方计算  $[a] + \delta$ 。基于符号  $\langle \cdot \rangle$  和  $\llbracket \cdot \rrbracket$  的定义，我们可以采用类似的方法，通过一对秘密分享  $\langle r \rangle, \llbracket r \rrbracket$  来实现输入分享。方法如下：

- 设  $P_i$  的输入为  $x_i$ 。取随机值的一对秘密分享  $\langle r \rangle, \llbracket r \rrbracket$ 。
- 将  $\llbracket r \rrbracket$  打开给  $P_i$ 。
- $P_i$  广播  $\delta = x_i - r$ 。
- 参与方计算  $\langle r \rangle + \delta = \langle x_i \rangle$ 。

**三元组正确性检验。**最后，与第9.4.1.4节中类似，因为考虑的是恶意敌手，预处理阶段产生的 Beaver 三元组  $(\langle a \rangle, \langle b \rangle, \langle c \rangle)$  不一定满足  $c = ab$ 。我们需要消耗另一个 Beaver 三元组  $(\langle f \rangle, \langle g \rangle, \langle h \rangle)$  以检验其正确性。协议如下：

- 取两个 Beaver 三元组  $(\langle a \rangle, \langle b \rangle, \langle c \rangle)$  和  $(\langle f \rangle, \langle g \rangle, \langle h \rangle)$ ；
- 参与方取随机值  $t$ ；
- 部分打开  $t \cdot \langle a \rangle - \langle f \rangle$ ，记为  $\rho$ ；部分打开  $\langle b \rangle - \langle g \rangle$ ，记为  $\sigma$ ；
- 计算  $t \cdot \langle c \rangle - \langle h \rangle - \sigma \cdot \langle f \rangle - \rho \cdot \langle g \rangle - \sigma \cdot \rho$ ，并将结果部分打开；
- 如果结果不为 0，丢弃  $(\langle a \rangle, \langle b \rangle, \langle c \rangle)$ ；否则，输出  $(\langle a \rangle, \langle b \rangle, \langle c \rangle)$ 。

在协议的输出阶段，所有部分打开的值都需要检验正确性。

**小结。**我们对上述思路作一个小结，给出 SPDZ 协议的非形式化描述。

**预处理阶段：**

- 生成 MAC 密钥的认证秘密分享  $\llbracket \Delta \rrbracket$ ，足够的认证 Beaver 三元组  $(\langle a \rangle, \langle b \rangle, \langle c \rangle)$ ，足够的随机值的秘密分享对  $\langle r \rangle, \llbracket r \rrbracket$ ，以及两个随机值的认证秘密分享  $\llbracket t \rrbracket, \llbracket e \rrbracket$ 。

**在线阶段：**

- **输入分享：**设  $P_i$  的输入为  $x_i$ 。取随机值的秘密分享对  $\langle r \rangle, \llbracket r \rrbracket$ ，将  $\llbracket r \rrbracket$  打开给  $P_i$ 。 $P_i$  广播  $\delta = x_i - r$ 。参与方计算  $\langle r \rangle + \delta = \langle x_i \rangle$ 。
- **逐门计算：**
  - 对于加法门、常数乘法门、常数加法门，分别计算

$$\langle x \rangle + \langle y \rangle = \langle x + y \rangle, \alpha \langle x \rangle = \langle \alpha x \rangle, \langle x \rangle + \alpha = \langle x + \alpha \rangle$$

- 对于输入为  $\langle x \rangle, \langle y \rangle$  的乘法门，执行如下操作

1. 检验三元组正确性:

- \* 取两个 Beaver 三元组  $(\langle a \rangle, \langle b \rangle, \langle c \rangle)$  和  $(\langle f \rangle, \langle g \rangle, \langle h \rangle)$ ;
- \* 打开随机值  $\llbracket t \rrbracket$ ;
- \* 部分打开  $t \cdot \langle a \rangle - \langle f \rangle$ , 记为  $\rho$ ; 部分打开  $\langle b \rangle - \langle g \rangle$ , 记为  $\sigma$ ;
- \* 计算  $t \cdot \langle c \rangle - \langle h \rangle - \sigma \cdot \langle f \rangle - \rho \cdot \langle g \rangle - \sigma \cdot \rho$ , 并将结果部分打开;
- \* 如果结果不为 0, 丢弃  $(\langle a \rangle, \langle b \rangle, \langle c \rangle)$ ; 否则, 将  $(\langle a \rangle, \langle b \rangle, \langle c \rangle)$  用于乘法门计算。

2. 计算乘法门:

- \* 使用上一步检验过的三元组  $(\langle a \rangle, \langle b \rangle, \langle c \rangle)$ , 部分打开  $\langle x \rangle - \langle a \rangle$ , 记为  $d$ , 部分打开  $\langle y \rangle - \langle b \rangle$ , 记为  $\epsilon$ ;
- \* 参与方本地计算  $d\langle b \rangle + \epsilon\langle a \rangle + \langle c \rangle + d\epsilon = \langle xy \rangle$ .

- 输出阶段: 设  $P_i$  的输出为  $y_i$ . 参与方持有  $\langle y_1 \rangle, \dots, \langle y_n \rangle$ , 其中  $\langle y_j \rangle = ((y_{j,1}, \dots, y_{j,n}), (M(y_j)_1, \dots, M(y_j)_n))$ . 参与方执行如下操作:

- 假设  $a_1, \dots, a_T$  是所有被部分打开的值, 其中  $\langle a_j \rangle = ((a_{j,1}, \dots, a_{j,n}), (M(a_j)_1, \dots, M(a_j)_n))$ . 参与方打开随机值  $\llbracket e \rrbracket$ , 对于  $i = 1, \dots, T$  设置  $e_i = e^i$ , 并计算  $a = \sum_{j=1}^T e_j a_j$ .
- $P_i$  计算  $M_i = \sum_{j=1}^T e_j \cdot M(a_j)_i$  并对这个值进行承诺。对于  $j = 1, \dots, n$ ,  $P_i$  对  $y_{j,i}$  和  $M(y_j)_i$  进行承诺。
- 打开全局 MAC 密钥  $\llbracket \Delta \rrbracket$ .
- $P_i$  打开承诺值  $M_i$ . 参与方检查  $\Delta \cdot a \stackrel{?}{=} \sum_{i=1}^n M_i$ . 若不满足, 则终止协议。
- 对于  $j = 1, \dots, n$ ,  $P_i$  向  $P_j$  打开承诺值  $y_{j,i}$  和  $M(y_j)_i$ .  $P_j$  计算  $y_j = \sum_{i=1}^n y_{j,i}$  并检查  $\sum_{i=1}^n M(y_j)_i \stackrel{?}{=} \Delta \cdot y_j$ . 若检查通过, 则  $P_j$  输出  $y_j$ .

#### 9.4.2.2 协议描述

本节, 我们将预处理阶段抽象为理想功能  $\mathcal{F}_{\text{prep}}$ , 将承诺方案抽象为理想功能  $\mathcal{F}_{\text{com}}$ , 然后在  $\{\mathcal{F}_{\text{prep}}, \mathcal{F}_{\text{com}}\}$ -混合模型下给出 SPDZ 协议的形式化描述。

**符号表示。** 对于消息  $x \in \mathbb{F}_p$ , 密钥  $\Delta \in \mathbb{F}_p$ , 消息验证码 MAC 的定义为  $\text{Mac}_\Delta(x) = \Delta \cdot x$ .

秘密分享  $\langle \cdot \rangle$  的定义如下:

$$\langle x \rangle = ((x_1, \dots, x_n), (M(x)_1, \dots, M(x)_n))$$

其中  $x = x_1 + \dots + x_n$ ,  $M(x)_1 + \dots + M(x)_n = \text{Mac}_\Delta(x) = \Delta \cdot x$ ,  $P_i$  持有  $(x_i, M(x)_i)$  和 MAC 密钥的份额  $\Delta_i$ , 满足  $\Delta = \Delta_1 + \dots + \Delta_n$ .

秘密分享  $\langle \cdot \rangle$  的部分打开、加法、常数乘法、常数加法运算的定义如图 9.41 所示。

秘密分享  $\llbracket \cdot \rrbracket$  的定义如下:

$$\llbracket x \rrbracket = ((x_1, \dots, x_n), (\beta_i, M(x)_1^i, \dots, M(x)_n^i)_{i \in [n]})$$

其中  $x = x_1 + \dots + x_n$ ,  $M(x)_i^1 + \dots + M(x)_i^n = \text{Mac}_{\beta_i}(x) = \beta_i x$ . 参与方  $P_i$  持有  $x_i, \beta_i, M(x)_1^i, \dots, M(x)_n^i$ .

秘密分享  $\llbracket \cdot \rrbracket$  的打开操作的定义如图 9.42 所示。

### 秘密分享 $\langle \cdot \rangle$ 的运算

**部分打开:** 当需要将  $\langle x \rangle$  部分打开时, 每个参与方  $P_i$  将  $x_i$  发送给  $P_1$ ,  $P_1$  计算  $x = \sum_{i=1}^n x_i$  并广播。

**加法:** 对  $\langle x \rangle = ((x_1, \dots, x_n), (M(x)_1, \dots, M(x)_n))$ ,  $\langle x' \rangle = ((x'_1, \dots, x'_n), (M(x')_1, \dots, M(x')_n))$ , 加法运算定义为:

$$\langle x \rangle + \langle x' \rangle = ((x_1 + x'_1, \dots, x_n + x'_n), (M(x)_1 + M(x')_1, \dots, M(x)_n + M(x')_n))$$

**常数乘法:** 对  $\langle x \rangle = ((x_1, \dots, x_n), (M(x)_1, \dots, M(x)_n))$  和常数  $\alpha \in \mathbb{F}_p$ , 常数乘法定义为:

$$\alpha \langle x \rangle = ((\alpha x_1, \dots, \alpha x_n), (\alpha \cdot M(x)_1, \dots, \alpha \cdot M(x)_n))$$

**常数加法:** 对  $\langle x \rangle = ((x_1, \dots, x_n), (M(x)_1, \dots, M(x)_n))$  和常数  $\alpha \in \mathbb{F}_p$ , 常数加法定义为:

$$\langle x \rangle + \alpha = ((x_1 + \alpha, \dots, x_n), (M(x)_1 + \Delta_1 \alpha, \dots, M(x)_n + \Delta_n \alpha))$$

图 9.41: 秘密分享  $\langle \cdot \rangle$  的运算

### 秘密分享 $\llbracket \cdot \rrbracket$ 的运算

**打开:** 当需要将  $\llbracket x \rrbracket = ((x_1, \dots, x_n), (\beta_i, M(x)_1^i, \dots, M(x)_n^i)_{i \in [n]})$  打开给  $P_i$  时, 每个参与方  $P_j$  发送  $x_j, M(x)_j^i$  给  $P_i$ , 然后  $P_i$  计算  $x = x_1 + \dots + x_n$  并检查  $\sum_{j=1}^n M(x)_j^i \stackrel{?}{=} \beta_i x$ . 如果检查不通过,  $P_i$  广播 fail. 当需要将  $\llbracket x \rrbracket$  打开给所有人时, 对每个参与方执行该操作。

图 9.42: 秘密分享  $\llbracket \cdot \rrbracket$  的运算

**理想功能  $\mathcal{F}_{\text{prep}}$ .** 我们将预处理阶段抽象为理想功能  $\mathcal{F}_{\text{prep}}$ , 它接受 3 种命令: INIT, PAIRS, TRIPLES, 分别用于初始化、生成认证秘密分享对、生成认证 Beaver 三元组。需要注意的是,  $\mathcal{F}_{\text{prep}}$  允许被攻陷方选择自己的秘密份额和 MAC 密钥份额, 并且允许敌手在认证秘密分享中加入偏移量  $\delta$ . 也就是说, 敌手有能力让  $\mathcal{F}_{\text{prep}}$  生成错误的认证秘密分享, 但是错误的认证秘密分享在协议运行中通过验证的概率是可忽略的。因此, 这等价于敌手使协议终止。

$\mathcal{F}_{\text{prep}}$  的定义如图 9.44 所示。为了表示的方便, 我们定义两个函数 Bracket 和 Angle, 分别用于生成  $\llbracket \cdot \rrbracket$  秘密分享和  $\langle \cdot \rangle$  秘密分享。

**理想功能  $\mathcal{F}_{\text{com}}$ .** 我们给出承诺理想功能  $\mathcal{F}_{\text{com}}$  的定义, 如图 9.43 所示。 $\mathcal{F}_{\text{com}}$  允许参与方对某个值承诺, 然后打开给所有参与方, 或打开给某个指定参与方。其中,  $\text{cid}$  表示 commitment id, 用于给承诺编号。为了描述的简洁性, 在协议描述中, 我们直接称“ $P_i$  调用  $\mathcal{F}_{\text{com}}$  对  $x_i$  进行承诺”、“ $P_i$  调用  $\mathcal{F}_{\text{com}}$  (向  $P_i$ ) 打开  $x_i$ ”。 $\mathcal{F}_{\text{com}}$  可以通过任何 UC 安全的承诺方案来实现<sup>[74-75]</sup>。

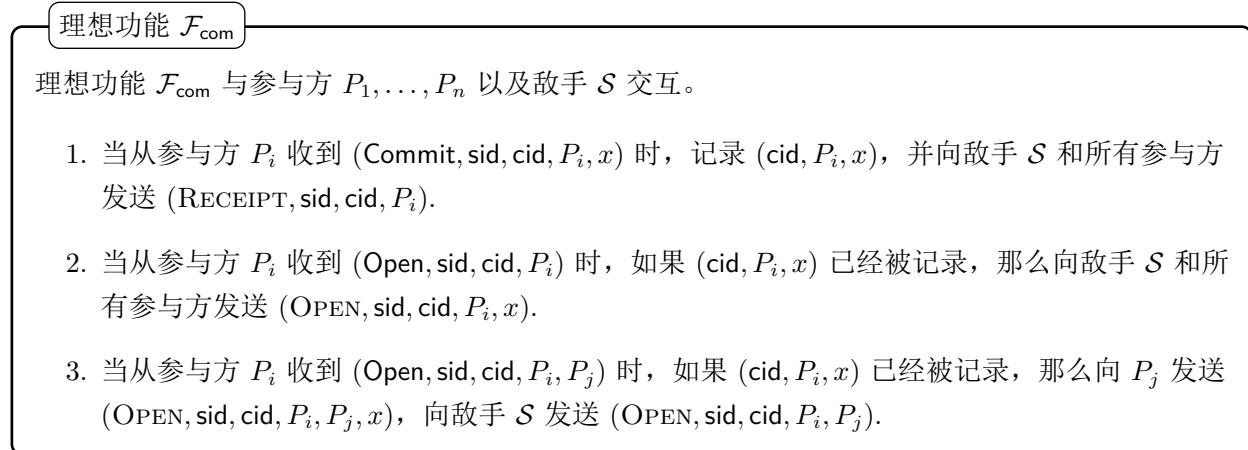


图 9.43: 理想功能  $\mathcal{F}_{\text{com}}$

**协议描述。**下面, 我们给出  $\{\mathcal{F}_{\text{prep}}, \mathcal{F}_{\text{com}}\}$ -混合模型下, SPDZ 协议的形式化描述, 如图 9.45 所示。

SPDZ 预处理阶段理想功能  $\mathcal{F}_{\text{prep}}$

令  $C$  为被攻陷方的集合。

$\text{Bracket}(\mathbf{v}_1, \dots, \mathbf{v}_n, \boldsymbol{\delta}_1, \dots, \boldsymbol{\delta}_n, \beta_1, \dots, \beta_n)$ : (其中  $\mathbf{v}_1, \dots, \mathbf{v}_n, \boldsymbol{\delta}_1, \dots, \boldsymbol{\delta}_n \in (\mathbb{F}_p)^s$ ,  $\beta_1, \dots, \beta_n \in \mathbb{F}_p$ )

1. 令  $\mathbf{v} = \sum_{i=1}^n \mathbf{v}_i$ .
2. 对于  $i = 1, \dots, n$ 
  - (a) 计算 MAC 标签  $\mathbf{M}(\mathbf{v})_i = \beta_i \cdot \mathbf{v}$ , 设置  $\mathbf{M}_i = \mathbf{M}(\mathbf{v})_i + \boldsymbol{\delta}_i$ .
  - (b) 对于每个被攻陷方  $P_j \in C$ , 记录其发送的  $\mathbf{M}_i^j$ .
  - (c) 对于每个诚实方  $P_j \notin C$ , 均匀随机选择  $\mathbf{M}_i^j$ , 满足  $\sum_{i=1}^n \mathbf{M}_i^j = \mathbf{M}_i$ .
3. 向每个参与方  $P_i$  发送  $(\mathbf{v}_i, (\beta_i, \mathbf{M}_1^i, \dots, \mathbf{M}_n^i))$ .

$\text{Angle}(\mathbf{v}_1, \dots, \mathbf{v}_n, \boldsymbol{\delta}, \Delta)$ : (其中  $\mathbf{v}_1, \dots, \mathbf{v}_n, \boldsymbol{\delta} \in (\mathbb{F}_p)^s$ ,  $\Delta \in \mathbb{F}_p$ )

1. 令  $\mathbf{v} = \sum_{i=1}^n \mathbf{v}_i$ .
2. 计算 MAC 标签  $\mathbf{M}(\mathbf{v}) = \Delta \cdot \mathbf{v}$ , 设置  $\mathbf{M} = \mathbf{M}(\mathbf{v}) + \boldsymbol{\delta}$ .
3. 对于每个被攻陷方  $P_i \in C$ , 记录其发送的  $\mathbf{M}_i$ .
4. 对于每个诚实方  $P_i \notin C$ , 均匀随机选择  $\mathbf{M}_i$ , 满足  $\sum_{i=1}^n \mathbf{M}_i = \mathbf{M}$ .
5. 向每个参与方  $P_i$  发送  $(\mathbf{v}_i, \mathbf{M}_i)$ .

**初始化:** 当收到来自每个参与方  $P_i$  的消息  $(\text{INIT}, \text{sid}, p, s)$  时,  $\mathcal{F}_{\text{prep}}$  记录域的模数  $p$  和参数  $s$ , 执行如下操作:

1. 执行期间, 如果收到来自敌手  $S$  的  $(\text{STOP}, \text{sid})$ , 那么终止并向所有参与方发送 fail.
2. 对于每个被攻陷方  $P_i \in C$ , 记录其发送的  $\Delta_i$  和  $\beta_i$ .
3. 对于每个诚实方  $P_i \notin C$ , 均匀随机选择  $\Delta_i, \beta_i \leftarrow_{\$} \mathbb{F}_p$ . 令  $\Delta = \sum_{i=1}^n \Delta_i$ .
4. 记录敌手  $S$  发送的  $\boldsymbol{\delta}_1, \dots, \boldsymbol{\delta}_n \in (\mathbb{F}_p)^s$ .
5. 执行  $\text{Bracket}(\text{Diag}_s(\Delta_1), \dots, \text{Diag}_s(\Delta_n), \boldsymbol{\delta}_1, \dots, \boldsymbol{\delta}_n, \beta_1, \dots, \beta_n)$ , 其中  $\text{Diag}_s(x) = (x, x, \dots, x) \in (\mathbb{F}_p)^s$ .

**秘密分享对:** 当收到来自每个参与方  $P_i$  的消息  $(\text{PAIRS}, \text{sid})$  时,  $\mathcal{F}_{\text{prep}}$  执行以下操作:

1. 执行期间, 如果收到来自敌手  $S$  的  $(\text{STOP}, \text{sid})$ , 那么终止并向所有参与方发送 fail.
2. 对于每个被攻陷方  $P_i \in C$ , 记录其发送的  $\mathbf{r}_i$ .
3. 对于每个诚实方  $P_i \notin C$ , 均匀随机选择  $\mathbf{r}_i$ .
4. 记录敌手  $S$  发送的  $\boldsymbol{\delta}, \boldsymbol{\delta}_1, \dots, \boldsymbol{\delta}_n \in (\mathbb{F}_p)^s$ .
5. 执行  $\text{Bracket}(\mathbf{r}_1, \dots, \mathbf{r}_n, \boldsymbol{\delta}_1, \dots, \boldsymbol{\delta}_n, \beta_1, \dots, \beta_n)$  和  $\text{Angle}(\mathbf{r}_1, \dots, \mathbf{r}_n, \boldsymbol{\delta}, \Delta)$ .

**三元组:** 当收到来自每个参与方  $P_i$  的消息  $(\text{TRIPLES}, \text{sid})$  时,  $\mathcal{F}_{\text{prep}}$  执行以下操作: 对于  $v = 1, \dots, u$ :

1. 执行期间, 如果收到来自敌手  $S$  的  $(\text{STOP}, \text{sid})$ , 那么终止并向所有参与方发送 fail.
2. 对于每个被攻陷方  $P_i \in C$ , 记录其发送的  $\mathbf{a}_i, \mathbf{b}_i$ .
3. 对于每个诚实方  $P_i \notin C$ , 均匀随机选择  $\mathbf{a}_i, \mathbf{b}_i$ . 令  $\mathbf{a} = \sum_{i=1}^n \mathbf{a}_i, \mathbf{b} = \sum_{i=1}^n \mathbf{b}_i$ .
4. 记录敌手  $S$  发送的  $\boldsymbol{\delta}_a, \boldsymbol{\delta}_b, \boldsymbol{\delta} \in (\mathbb{F}_p)^s$ . 设置  $\mathbf{c} = \mathbf{a} \cdot \mathbf{b} + \boldsymbol{\delta}$ , 其中  $\mathbf{a} \cdot \mathbf{b}$  表示逐项乘法.
5. 对于每个被攻陷方  $P_i \in C$ , 记录其发送的  $\mathbf{c}_i$ .
6. 对于每个诚实方  $P_i \notin C$ , 均匀随机选择  $\mathbf{c}_i$ , 满足  $\sum_{i=1}^n \mathbf{c}_i = \mathbf{c}$ .
7. 记录敌手  $S$  发送的  $\boldsymbol{\delta}_c \in (\mathbb{F}_p)^s$ .
8. 执行  $\text{Angle}(\mathbf{a}_1, \dots, \mathbf{a}_n, \boldsymbol{\delta}_a, \Delta), \text{Angle}(\mathbf{b}_1, \dots, \mathbf{b}_n, \boldsymbol{\delta}_b, \Delta), \text{Angle}(\mathbf{c}_1, \dots, \mathbf{c}_n, \boldsymbol{\delta}_c, \Delta)$ .

图 9.44: SPDZ 预处理阶段理想功能  $\mathcal{F}_{\text{prep}}$

### $\{\mathcal{F}_{\text{prep}}, \mathcal{F}_{\text{com}}\}$ -混合模型下的 SPDZ 协议 $\Pi_{\text{SPDZ}}$

参与方  $P_1, \dots, P_n$  想要计算函数  $f : \mathbb{F}^{M_{\text{in}}^1} \times \dots \times \mathbb{F}^{M_{\text{in}}^n} \rightarrow \mathbb{F}^{M_{\text{out}}^1} \times \dots \times \mathbb{F}^{M_{\text{out}}^n}$  并已经将其转化成了包含加法门、常数乘法门、常数加法门、乘法门的算术电路。

#### 预处理阶段:

参与方首先向  $\mathcal{F}_{\text{prep}}$  发送  $(\text{INIT}, \text{sid}, p, s)$ . 然后, 参与方向  $\mathcal{F}_{\text{prep}}$  调用足够次数的  $(\text{PAIRS}, \text{sid})$  和  $(\text{TRIPLES}, \text{sid})$ , 获得足够的认证 Beaver 三元组  $(\langle a \rangle, \langle b \rangle, \langle c \rangle)$ , 足够的随机值的秘密分享对  $\langle r \rangle, \llbracket r \rrbracket$ , 两个随机值的认证秘密分享  $\llbracket t \rrbracket, \llbracket e \rrbracket$ .

#### 在线阶段:

- **输入分享:** 假设  $P_i$  的某个输入为  $x_i$ , 取随机值的秘密分享对  $\langle r \rangle, \llbracket r \rrbracket$ , 执行如下操作:

1. 向  $P_i$  打开  $\llbracket r \rrbracket$ ;
2.  $P_i$  广播  $\delta = x_i - r$ ;
3. 参与方计算  $\langle r \rangle + \delta = \langle x_i \rangle$ .

对于  $P_i$  的每一个输入, 都执行上述操作。

- **逐门计算:** 重复以下步骤, 直到所有门都被计算。根据门的类型, 执行以下操作之一:

- 加法门。假设参与方持有  $\langle x \rangle, \langle y \rangle$ . 参与方计算  $\langle x \rangle + \langle y \rangle = \langle x + y \rangle$ .
- 常数乘法门。假设对应的常数为  $\alpha$ , 参与方持有  $\langle x \rangle$ . 参与方计算  $\alpha \langle x \rangle = \langle \alpha x \rangle$ .
- 常数加法门。假设对应的常数为  $\alpha$ , 参与方持有  $\langle x \rangle$ . 参与方计算  $\langle x \rangle + \alpha = \langle x + \alpha \rangle$ .
- 乘法门。假设参与方持有  $\langle x \rangle, \langle y \rangle$ , 执行以下操作:

1. 检验三元组正确性 (这一步实际上可以在预处理阶段完成):
  - \* 取两个 Beaver 三元组  $(\langle a \rangle, \langle b \rangle, \langle c \rangle)$  和  $(\langle f \rangle, \langle g \rangle, \langle h \rangle)$ ;
  - \* 打开随机值  $\llbracket t \rrbracket$ ;
  - \* 部分打开  $t \cdot \langle a \rangle - \langle f \rangle$ , 记为  $\rho$ ; 部分打开  $\langle b \rangle - \langle g \rangle$ , 记为  $\sigma$ ;
  - \* 计算  $t \cdot \langle c \rangle - \langle h \rangle - \sigma \cdot \langle f \rangle - \rho \cdot \langle g \rangle - \sigma \cdot \rho$ , 并将结果部分打开;
  - \* 如果结果不为 0, 丢弃  $(\langle a \rangle, \langle b \rangle, \langle c \rangle)$ ; 否则, 将  $(\langle a \rangle, \langle b \rangle, \langle c \rangle)$  用于乘法门计算。
2. 计算乘法门:
  - \* 使用上一步检验过的三元组  $(\langle a \rangle, \langle b \rangle, \langle c \rangle)$ , 部分打开  $\langle x \rangle - \langle a \rangle$ , 记为  $d$ , 部分打开  $\langle y \rangle - \langle b \rangle$ , 记为  $\epsilon$ ;
  - \* 参与方本地计算  $d \langle b \rangle + \epsilon \langle a \rangle + \langle c \rangle + d\epsilon = \langle xy \rangle$ .

- **输出阶段:** 设  $P_i$  的输出为  $y_i$ . 参与方持有  $\langle y_1 \rangle, \dots, \langle y_n \rangle$ , 其中  $\langle y_j \rangle = ((y_{j,1}, \dots, y_{j,n}), (M(y_j)_1, \dots, M(y_j)_n))$ ,  $j \in [n]$ . (如果每个参与方有多个输出, 将  $\langle y_1 \rangle, \dots, \langle y_n \rangle$  相应地改为  $((\langle y_1^1 \rangle, \dots, \langle y_1^{M_{\text{out}}^1} \rangle), \dots, (\langle y_n^1 \rangle, \dots, \langle y_n^{M_{\text{out}}^n} \rangle))$ .) 参与方执行如下操作:

1. 假设  $a_1, \dots, a_T$  是所有被部分打开的值, 其中  $\langle a_j \rangle = ((a_{j,1}, \dots, a_{j,n}), (M(a_j)_1, \dots, M(a_j)_n))$ . 参与方打开随机值  $\llbracket e \rrbracket$ , 对于  $i = 1, \dots, T$  设置  $e_i = e^i$ , 并计算  $a = \sum_{j=1}^T e_j a_j$ .
2.  $P_i$  计算  $M_i = \sum_{j=1}^T e_j \cdot M(a_j)_i$  并调用  $\mathcal{F}_{\text{com}}$  对这个值进行承诺。对于  $j = 1, \dots, n$ ,  $P_i$  调用  $\mathcal{F}_{\text{com}}$  对  $y_{j,i}$  和  $M(y_j)_i$  进行承诺。
3. 打开全局 MAC 密钥  $\llbracket \Delta \rrbracket$ .
4.  $P_i$  调用  $\mathcal{F}_{\text{com}}$  打开承诺值  $M_i$ . 参与方检查  $\Delta \cdot a \stackrel{?}{=} \sum_{i=1}^n M_i$ . 若不满足, 则终止协议。
5. 对于  $j = 1, \dots, n$ ,  $P_i$  调用  $\mathcal{F}_{\text{com}}$  向  $P_j$  打开承诺值  $y_{j,i}$  和  $M(y_j)_i$ .  $P_j$  计算  $y_j = \sum_{i=1}^n y_{j,i}$  并检查  $\sum_{i=1}^n M(y_j)_i \stackrel{?}{=} \Delta \cdot y_j$ . 若检查通过, 则  $P_j$  输出  $y_j$ .

图 9.45:  $\{\mathcal{F}_{\text{prep}}, \mathcal{F}_{\text{com}}\}$ -混合模型下的 SPDZ 协议  $\Pi_{\text{SPDZ}}$

### 9.4.2.3 安全性证明

我们给出 SPDZ 协议的安全性证明。因为 SPDZ 协议也是基于 Beaver 三元组技术构造的，所以证明思路以及模拟器的构造与 BDOZ 协议中类似。

**定理 9.26.** 假设被攻陷方数量不超过  $n - 1$ ，图 9.45 中的协议  $\Pi_{\text{SPDZ}}$  在  $\{\mathcal{F}_{\text{prep}}, \mathcal{F}_{\text{com}}\}$ -混合模型下对于静态恶意对手 UC-安全实现了理想功能  $\mathcal{F}_{\text{sfe}}$ （图 9.31）。

证明. 要证明该定理，我们需要构造模拟器  $\mathcal{S}$ ，使得对所有环境  $\mathcal{Z}$ ，都有

$$\text{EXEC}_{\Pi_{\text{SPDZ}}, \mathcal{A}, \mathcal{Z}}^{\mathcal{F}_{\text{prep}}, \mathcal{F}_{\text{com}}} \approx \text{EXEC}_{\mathcal{F}_{\text{sfe}}, \mathcal{S}, \mathcal{Z}}$$

其中  $\mathcal{A}$  是平凡敌手。

**模拟器  $\mathcal{S}$ .** 模拟器  $\mathcal{S}$  在内部运行  $\mathcal{A}$ ，转发  $\mathcal{Z}$  与  $\mathcal{A}$  之间的消息。此外， $\mathcal{S}$  需要模拟  $\mathcal{F}_{\text{prep}}$  生成认证 Beaver 三元组和认证秘密分享对，以及模拟  $\mathcal{F}_{\text{com}}$ 。令  $C$  为被攻陷方的集合。 $\mathcal{S}$  的工作方式如下。

- 模拟  $\mathcal{F}_{\text{prep}}$ ：
  - 当  $\mathcal{F}_{\text{prep}}$  收到来自被攻陷方  $P_i$  的  $(\text{INIT}, \text{sid}, p, s)$  时， $\mathcal{S}$  在理想世界代表  $P_i$  向  $\mathcal{F}_{\text{sfe}}$  发送  $(\text{INIT}, \text{sid}, p)$ 。
  - $\mathcal{S}$  模拟  $\mathcal{F}_{\text{prep}}$  诚实地执行初始化，它记录每个被攻陷方发送的秘密份额和 MAC 密钥份额，并为每个诚实方随机选择秘密份额和 MAC 密钥份额。然后， $\mathcal{S}$  模拟  $\mathcal{F}_{\text{prep}}$  遵循描述计算认证 Beaver 三元组和认证秘密分享对，并分发给参与方。
- 模拟  $\mathcal{F}_{\text{com}}$ :  $\mathcal{S}$  模拟  $\mathcal{F}_{\text{com}}$  记录和打开承诺值。
- 输入阶段：
  - 对于诚实方  $P_i \notin C$ ，当从  $\mathcal{F}_{\text{sfe}}$  收到  $(\text{INPUT}, \text{sid}, P_i)$  时， $\mathcal{S}$  模拟  $P_i$  以 0 为输入遵循协议执行。如果将  $\llbracket r \rrbracket$  打开给  $P_i$  时 MAC 验证不通过， $\mathcal{S}$  模拟  $P_i$  广播 fail，并在理想世界向  $\mathcal{F}_{\text{sfe}}$  发送  $(\text{ABORT}, \text{sid})$ 。
  - 对于被攻陷方  $P_j \in C$ ， $\mathcal{S}$  记录其广播的  $\delta$ 。由于  $\mathcal{S}$  模拟了  $\mathcal{F}_{\text{prep}}$ ，因此知道  $r$  的值。 $\mathcal{S}$  提取  $P_j$  的输入  $x_j = r + \delta$ ，并在理想世界代表  $P_j$  发送给  $\mathcal{F}_{\text{sfe}}$ 。
    - 如果任何参与方广播 fail（打开  $\llbracket r \rrbracket$  时验证不通过）， $\mathcal{S}$  在理想世界向  $\mathcal{F}_{\text{sfe}}$  发送  $(\text{ABORT}, \text{sid})$ 。
- 计算阶段：
  - 对于加法门、常数乘法门、常数加法门、乘法门， $\mathcal{S}$  模拟诚实方遵循协议计算。
- 输出阶段：
  - 输出阶段的步骤 1~3 中， $\mathcal{S}$  模拟诚实方遵循协议运行，并模拟  $\mathcal{F}_{\text{com}}$  诚实地记录承诺值。如果步骤 4 的验证不通过， $\mathcal{S}$  模拟诚实方终止协议，并在理想世界向  $\mathcal{F}_{\text{sfe}}$  发送  $(\text{ABORT}, \text{sid})$ 。
  - 如果步骤 4 的验证通过，执行如下操作：
    - \* 对于诚实方  $P_i \notin C$  的输出，如果打开承诺值时验证不通过， $\mathcal{S}$  在理想世界阻止  $P_i$  获得输出。否则（MAC 验证通过）， $\mathcal{S}$  向  $\mathcal{F}_{\text{sfe}}$  发送  $(\text{OK}, \text{sid}, P_i)$ 。

- \* 对于被攻陷方  $P_j \in C$  的输出， $\mathcal{S}$  首先在理想世界向  $\mathcal{F}_{\text{sfe}}$  发送  $(\text{OK}, \text{sid}, P_j)$  得到  $P_j$  的输出  $y$ . 然后， $\mathcal{S}$  需要模拟诚实方的份额使其与  $y$  相一致。 $\mathcal{S}$  的方法是只修改一个诚实方的份额，保持其他诚实方的份额不变。记这个诚实方为  $P_k$ . 假设当诚实方使用 0 作为输入得到的输出为  $y'$ .  $\mathcal{S}$  将  $P_k$  的秘密份额加上  $y - y'$ ，将  $P_k$  的 MAC 标签份额加上  $\Delta \cdot (y - y')$  (因为  $\mathcal{S}$  模拟了  $\mathcal{F}_{\text{prep}}$ ，所以知道  $\Delta$  的值)，然后， $\mathcal{S}$  模拟  $\mathcal{F}_{\text{com}}$  向  $P_j$  打开秘密份额和 MAC 标签份额，其中  $P_k$  的份额是被修改过的。

**不可区分性。**首先，在输出阶段的步骤 5 之前，环境  $\mathcal{Z}$  在真实世界和理想世界中的视图具有相同的分布。这是因为：

1. 输入阶段，诚实方广播的都是均匀随机值；
2. 计算乘法门进行部分打开时，诚实方发送的也是均匀随机的秘密份额；
3. 在输出阶段的步骤 4，参与方打开的  $M_i$  是 MAC 标签份额的线性组合，其中敌手可能对 MAC 标签加入了偏移量  $\delta$ ，因为这些 MAC 标签份额是随机的，所以在真实世界和理想世界  $M_i$  有相同的分布；
4. 最后，如果真实世界中的验证不通过，协议终止，那么模拟的运行中验证也不通过， $\mathcal{S}$  会向  $\mathcal{F}_{\text{sfe}}$  发送  $(\text{ABORT}, \text{sid})$  使参与方如真实世界那样不获得输出。

如果协议进行到了输出阶段的步骤 5，参与方会获得输出，并接收到来自其他参与方的秘密份额、MAC 标签份额。如果真实世界和理想世界的输出是相同的，那么被攻陷方收到的秘密份额、MAC 标签份额在真实世界和理想世界中具有相同的分布。因此，我们只需要证明：真实世界中协议运行完成，但输出值与理想世界不同的概率是可忽略的。

显然，理想世界中的输出值一定是正确的。在真实世界中产生错误的输出只可能因为：(1) Beaver 三元组是错误的；(2) 被攻陷方发送了错误的秘密份额。下面论证当这两种情况发生时，通过验证的概率是可忽略的。对于第一种情况，协议在计算乘法门时对 Beaver 三元组进行了检验，检验条件等价于  $t(c - ab) = h - fg$ . 如果  $c \neq ab$ ，只有当  $t = (h - fg)/(c - ab)$  时该式成立，而  $t$  是随机的，因此等式满足的概率为  $1/p$ . 对于第二种情况，首先考虑  $\llbracket \cdot \rrbracket$  秘密分享，敌手对  $P_i$  作弊成功等价于猜到 MAC 密钥  $\beta_i$ ，而  $\beta_i$  是均匀随机选择的，敌手猜对的概率为  $1/p$ . 其次，考虑  $\langle \cdot \rangle$  秘密分享，将其抽象为如下的安全性游戏：

1. Challenger 生成密钥  $\Delta$  并计算  $M_i = \Delta \cdot m_i$ ,  $i = 1, \dots, T$ . 将消息  $m_1, \dots, m_T$  发送给敌手。
2. 敌手发送给 Challenger 消息  $m'_1, \dots, m'_T$ .
3. Challenger 生成随机值  $e_1, \dots, e_T \leftarrow_{\$} \mathbb{F}_p$  并发送给敌手。
4. 敌手发送给 Challenger 偏移量  $\delta$ .
5. 令  $m = \sum_{i=1}^T e_i m'_i$ ,  $M = \sum_{i=1}^T e_i M_i$ . Challenger 检查  $\Delta \cdot m \stackrel{?}{=} M + \delta$ .

如果存在  $i$  满足  $m'_i \neq m_i$  且最终验证通过，那么敌手获胜。

该游戏模拟了  $\langle \cdot \rangle$  秘密分享的验证：步骤 2 体现了被攻陷方可以发送错误的份额来改变打开的秘密值；步骤 4 体现了敌手有能力使  $\mathcal{F}_{\text{prep}}$  生成的秘密分享中有偏移量  $\delta$ ；注意，在这个游戏的步骤 3 中，

$e_i$  都是随机选择的，而协议中  $e_i = e^i$ ，其中  $e$  是随机选择的。我们先考虑  $e_i$  都随机时敌手的获胜概率，然后讨论  $e_i = e^i$  的情况。

我们来计算敌手获胜的概率。最终验证通过意味着  $\Delta \cdot \sum_{i=1}^T e_i(m'_i - m_i) = \delta$ 。我们首先考虑  $\sum_{i=1}^T e_i(m'_i - m_i) \neq 0$  的情况，此时  $\Delta = \delta / \sum_{i=1}^T e_i(m'_i - m_i)$ 。该式成立等价于敌手猜对  $\Delta$ ，而  $\Delta$  是均匀随机的，敌手猜对的概率为  $1/p$ 。接着，我们证明  $\sum_{i=1}^T e_i(m'_i - m_i) = 0$  发生的概率也是可忽略的。因为敌手在选择  $m'_i$  的时候不知道  $e_i$ ，而  $e_i$  都是均匀随机选择的，因此该式等于 0 的概率为  $1/p$ 。综上，敌手获胜的概率不超过  $2/p$ 。

在实际协议中， $e_i = e^i$ ，其中  $e$  是随机选择的。我们要证明  $\sum_{i=1}^T e_i(m'_i - m_i) = 0$  发生的概率仍然是很小的。定义  $\mu_i = m'_i - m_i$ ,  $\boldsymbol{\mu} = (\mu_1, \dots, \mu_T)$ ,  $\mathbf{e} = (e, e^2, \dots, e^T)$ ,  $f_\mu(\mathbf{e}) = \mathbf{e} \cdot \boldsymbol{\mu} = \sum_{i=1}^T e^i \mu_i$ 。这里  $f_\mu(\mathbf{e})$  是  $T$  次多项式，最多有  $T$  个根，因为  $e$  是均匀随机的，故  $f_\mu(\mathbf{e}) = 0$  的概率不超过  $T/p$ 。也就是说，在协议中，敌手将  $\langle \cdot \rangle$  秘密分享打开为错误的值且通过验证的概率不超过  $T/p$ ，其中  $T$  是打开的  $\langle \cdot \rangle$  秘密分享的数量。

证毕。 □

#### 9.4.2.4 \* 预处理阶段的实现<sup>22</sup>

本节将实现预处理阶段理想功能  $\mathcal{F}_{\text{prep}}$ 。我们介绍 SPDZ 原文<sup>[72]</sup>所提出的基于有限同态加密 (somewhat homomorphic encryption) 的方案。有限同态加密既满足加法同态也满足乘法同态，但是计算的次数是有限的，也就是要求计算的电路不能太大。因此它比较适合用于在预处理阶段计算大量的认证 Beaver 三元组。

**有限同态加密定义。**首先，定义密文空间  $M = (\mathbb{F}_p)^s$ 。我们用  $+$  和  $\cdot$  表示  $M$  中元素的逐项加法和逐项乘法。假设存在单射的编码函数  $\text{encode}$  将  $(\mathbb{F}_p)^s$  映射到环  $R = \mathbb{Z}^N$ ，其中  $N$  是一个整数；假设存在解码函数  $\text{decode}$  以  $\mathbb{Z}^N$  上的元素为输入，输出  $(\mathbb{F}_p)^s$  上的元素。并且，对于所有  $\mathbf{m} \in M$  满足  $\text{decode}(\text{encode}(\mathbf{m})) = \mathbf{m}$ 。此外，解密操作与域的参数是兼容的，即，对于所有  $\mathbf{x} \in \mathbb{Z}^N$  有  $\text{decode}(\mathbf{x}) = \text{decode}(\mathbf{x} \pmod p)$ 。最后，编码算法生成的向量是“短的”，即，对于所有  $\mathbf{m} \in (\mathbb{F}_p)^s$  有  $\|\text{encode}(\mathbf{m})\|_\infty \leq \tau$ ，其中  $\tau = p/2$ 。

环  $R$  上的两个操作记为  $+$  和  $\cdot$ 。我们假设  $R$  上的加法操作是逐元素的加法，而对乘法操作没有假设。我们要求如下性质成立：对于所有元素  $\mathbf{m}_1, \mathbf{m}_2 \in M$ ，满足

$$\begin{aligned}\text{decode}(\text{encode}(\mathbf{m}_1) + \text{encode}(\mathbf{m}_2)) &= \mathbf{m}_1 + \mathbf{m}_2 \\ \text{decode}(\text{encode}(\mathbf{m}_1) \cdot \text{encode}(\mathbf{m}_2)) &= \mathbf{m}_1 \cdot \mathbf{m}_2\end{aligned}$$

接下来的介绍中，当我们讨论明文空间  $M$  时，我们默认它隐含地带有某个整数  $N$  所对应的编码函数  $\text{encode}$  和解码函数  $\text{decode}$ 。对于  $x \in \mathbb{F}_p$ ，定义  $\text{Diag}_s(x) = (x, x, \dots, x) \in (\mathbb{F}_p)^s$ 。

有限同态加密方案以  $\kappa$  为安全参数，包含 5 个算法 ( $\text{ParamGen}, \text{KeyGen}, \text{KeyGen}^*, \text{Enc}, \text{Dec}$ )。

**ParamGen( $1^\kappa, M$ ):** 这是参数生成算法，它输出整数  $N$ ，函数  $\text{encode}$  和  $\text{decode}$  的定义，以及随机化算法  $D_\rho^d$  的描述，其中  $D_\rho^d$  的输出是  $\mathbb{Z}^d$  中的向量。我们假设  $D_\rho^d$  输出的  $\mathbf{r}$  以压倒性的概率满足  $\|\mathbf{r}\|_\infty \leq \rho$ 。  
 $D_\rho^d$  给加密算法提供所需的随机数。算法  $\text{ParamGen}$  还输出一个加法阿贝尔群  $\mathbb{G}$ 。群  $\mathbb{G}$  同时拥有一个乘法运算（不一定封闭），该运算是交换的，并且对  $\mathbb{G}$  上的加法满足分配律。假设密文位于群  $\mathbb{G}$  中。我们用符号  $\boxplus$  和  $\boxtimes$  来表示  $\mathbb{G}$  上的加法和乘法运算，并将这些运算自然地扩展到  $\mathbb{G}$  中元素构成的向量和矩阵上。最后， $\text{ParamGen}$  输出一个允许的算术电路集合  $C$ ，这些电路定义了能够在密文上计算的函数

<sup>22</sup>此部分授课老师/读者可以选择性地教学/阅读。

集合。我们假设所有其他算法都隐含地把  $\text{ParamGen}$  的输出  $P = (1^\kappa, N, \text{encode}, \text{decode}, D_\rho^d, \mathbb{G}, C)$  作为输入。

KeyGen(): 该算法生成公钥  $\text{pk}$  和私钥  $\text{sk}$ .

Enc<sub>pk</sub>(x, r): 这是一个确定性算法，输入是  $\mathbf{x} \in \mathbb{Z}^N, \mathbf{r} \in \mathbb{Z}^d$ , 输出是密文  $c \in \mathbb{G}$ . 这里的  $\mathbf{x}$  是  $\text{encode}$  的输出,  $\mathbf{r}$  是  $D_\rho^d$  的输出。因此对于  $\mathbf{m} \in M$ , 我们可以用  $\text{Enc}_{\text{pk}}(\mathbf{m})$  表示对其加密。当随机数不重要时, 我们简写为  $\text{Enc}_{\text{pk}}(\mathbf{x})$ . 为了使后续的零知识证明能够成立, 我们要求对  $V$  个“干净”的密文 (即  $V$  的值“较小”) 进行相加时, 所得密文等于将明文相加后再进行加密的结果, 即

$$\text{Enc}_{\text{pk}}(\mathbf{x}_1 + \cdots + \mathbf{x}_V, \mathbf{r}_1 + \cdots + \mathbf{r}_n) = \text{Enc}_{\text{pk}}(\mathbf{x}_1, \mathbf{r}_1) \boxplus \cdots \boxplus \text{Enc}_{\text{pk}}(\mathbf{x}_V, \mathbf{r}_V)$$

Dec<sub>sk</sub>(c): 解密算法以私钥  $\text{sk}$  和密文  $c$  为输入, 输出  $\mathbf{m} \in M$  或符号  $\perp$ .

我们现在可以定义上述加密方案所需满足的各种性质。在此之前, 先引入一些符号: 对于某个函数  $f \in C$ , 用  $n(f)$  表示函数  $f$  的变量个数, 并用  $\hat{f}$  表示  $f$  在群  $\mathbb{G}$  上对应的函数。具体地说, 给定  $f$ , 我们将其表达式中的加法  $+$  替换为群  $\mathbb{G}$  上的加法运算  $\boxplus$ , 乘法  $\cdot$  替换为群  $\mathbb{G}$  上的乘法运算  $\boxtimes$ , 常数  $c$  替换为  $\text{Enc}_{\text{pk}}(\text{encode}(c), \mathbf{0})$  得到  $\hat{f}$ . 此外, 给定  $n(f)$  个向量  $\mathbf{x}_1, \dots, \mathbf{x}_{n(f)}$ , 我们以自然的方式定义  $f(\mathbf{x}_1, \dots, \mathbf{x}_{n(f)})$ , 指的是在每个索引上计算函数  $f$ .

**正确性:** 直观地说, 正确性意味着: 如果我们对一个函数  $f \in C$  的结果进行解密, 该函数是作用在  $n(f)$  个加密向量上的, 那么解密结果应当与该函数直接作用在  $n(f)$  个明文向量上的结果相一致。然而, 为了在协议中使用该方案, 我们需要放宽这一要求: 即使我们所使用的密文并非由常规的加密算法生成, 只要它们包含的编码和随机数不过大, 并且这些编码可以被解码为合法的值, 那么解密结果也是正确的。形式化地, 如果一个有限同态加密方案满足以下条件

$$\begin{aligned} & \Pr[P \leftarrow \text{ParamGen}(1^\kappa, M), (\text{pk}, \text{sk}) \leftarrow \text{KeyGen}(), \text{for any } f \in C, \\ & \text{any } \mathbf{x}_i, \mathbf{r}_i, \text{ with } \|\mathbf{x}_i\|_\infty \leq B_{\text{plain}}, \|\mathbf{r}_i\|_\infty \leq B_{\text{rand}}, \text{decode}(\mathbf{x}_i) \in (\mathbb{F}_p)^s, \\ & i = 1, \dots, n(f), \text{ and } c_i \leftarrow \text{Enc}_{\text{pk}}(\mathbf{x}_i, \mathbf{r}_i), c \leftarrow \hat{f}(c_1, \dots, c_{n(f)}): \\ & \text{Dec}_{\text{sk}}(c) \neq f(\text{decode}(\mathbf{x}_1), \dots, \text{decode}(\mathbf{x}_{n(f)}))] < \text{negl}(\kappa) \end{aligned}$$

那么, 我们称它满足  $(B_{\text{plain}}, B_{\text{rand}}, C)$ -正确性。

如果一个密文可以在上述实验中作为密文  $c$  得到, 那么我们称它是  $(B_{\text{plain}}, B_{\text{rand}}, C)$ -可容许的 (*admissible*)。也就是说, 该密文是通过将函数  $f \in C$  应用于 (合法的) 受  $B_{\text{plain}}$  限制的编码和  $B_{\text{rand}}$  限制的随机数生成的密文而得到的。

KeyGen<sup>\*</sup>(): 这是一个随机算法, 输出一个无意义的公钥  $\widetilde{\text{pk}}$ , 满足对于任意消息  $\mathbf{x}$ ,  $\text{Enc}_{\widetilde{\text{pk}}}(\mathbf{x})$  与  $\text{Enc}_{\text{pk}}(\mathbf{0})$  是统计上不可区分的。此外, 令  $(\text{pk}, \text{sk}) \leftarrow \text{KeyGen}(), \widetilde{\text{pk}} \leftarrow \text{KeyGen}^*$ , 那么  $\text{pk}$  与  $\widetilde{\text{pk}}$  在计算上也是不可区分的。此性质暗含了该加密方案是 IND-CPA 安全的。

**分布式解密:** 我们希望参与方能够实现分布式解密。对此, 我们假设系统已经设置好了一个公钥, 并且对应的私钥在参与方之间秘密分享 (这是系统的初始设置 (set up assumption)), 使参与方可以共同对密文进行解密。我们假设参与方始终只对  $(B_{\text{plain}}, B_{\text{rand}}, C)$ -可容许的密文进行解密, 这一约束由我们的协议所保证。

具体而言, 我们假设存在理想功能  $\mathcal{F}_{\text{KeyGen}}$ , 如图 9.46 所示。它的基本功能是生成一个密钥对, 并让私钥在参与方之间秘密分享。这个秘密分享方案可以认为是加密系统的一个参数。

此外，我们还希望我们的加密系统能够实现理想功能  $\mathcal{F}_{\text{KeyGen}}$ ，如图 9.47 所示。通过调用该理想功能，参与方可以共同解密  $(B_{\text{plain}}, B_{\text{rand}}, C)$ -可容许的密文，但只在被动攻击下安全。也就是说：敌手能够获得正确的解密结果，并且有能力决定诚实方最终得到的结果是什么。

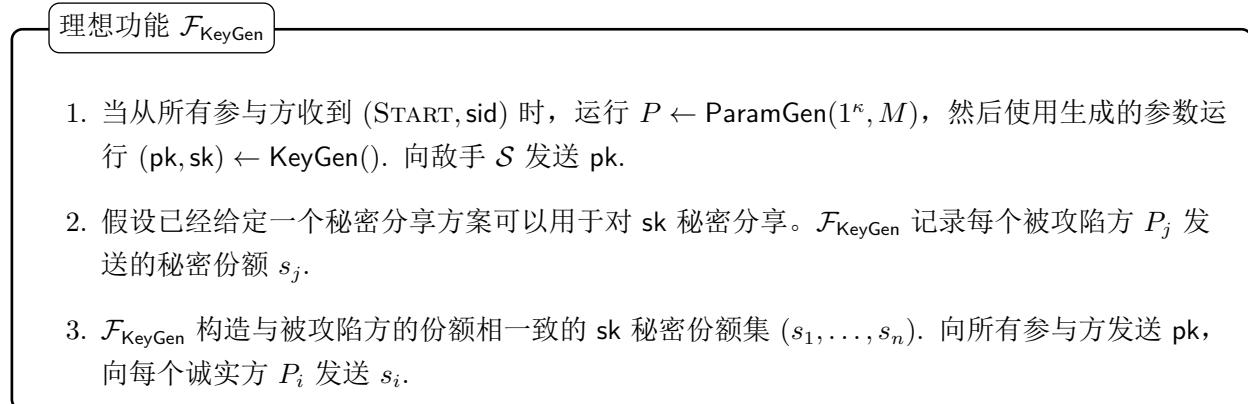


图 9.46: 理想功能  $\mathcal{F}_{\text{KeyGen}}$

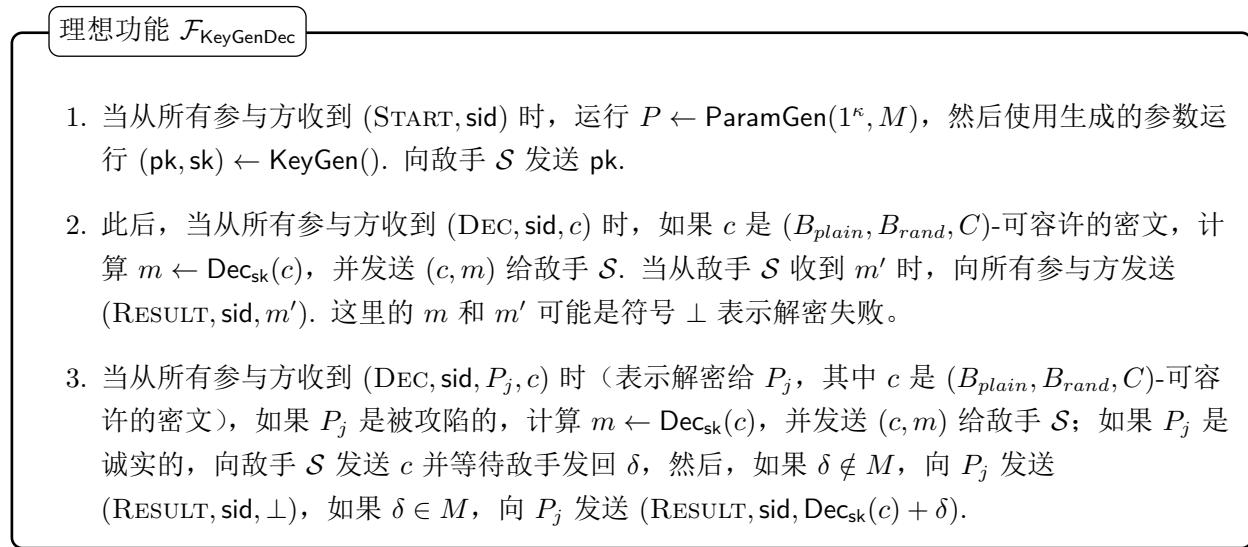


图 9.47: 理想功能  $\mathcal{F}_{\text{KeyGenDec}}$

现在，我们终于可以给出底层加密系统性质的定义。这是要在协议中使用该加密系统，必须满足的基本性质。这里我们引入信息论安全参数  $u$ ，用于控制零知识证明协议中出现的误差。

**定义 9.4.1.** (可容许加密系统) 令  $C$  包含形式为  $(x_1 + \dots + x_n) \cdot (y_1 + \dots + y_n) + z_1 + \dots + z_n$  或形式更简单的公式，“更简单的”指包含更少的加法、可能不包含乘法。如果一个加密系统由算法  $(\text{ParamGen}, \text{KeyGen}, \text{KeyGen}^*, \text{Enc}, \text{Dec})$  构成，满足上述定义的性质，满足  $(B_{\text{plain}}, B_{\text{rand}}, C)$ -正确性，其中

$$B_{\text{plain}} = N \cdot \tau \cdot u^2 \cdot 2^{(1/2+\nu)u}, \quad B_{\text{rand}} = d \cdot \rho \cdot u^2 \cdot 2^{(1/2+\nu)u}$$

其中  $\nu > 0$  为任意常数。此外，存在满足  $\mathcal{F}_{\text{KeyGen}}$  要求的秘密分享方案，存在协议  $\Pi_{\text{KeyGenDec}}$  在  $\mathcal{F}_{\text{KeyGen}}$ -混

合模型下  $UC$ -安全实现  $\mathcal{F}_{\text{KeyGenDec}}$ 。那么，我们称该加密系统是可容许的 (*admissible*)。

集合  $C$  是协议所需的所有对密文的计算操作。我们始终假定  $B_{plain}$  和  $B_{rand}$  以上述形式基于参数  $\tau, \rho, u$  所定义。这是因为我们能够通过零知识证明协议强制被攻陷方遵守这些约束。

在实际使用中， $\text{BGV}^{[76]}$  是一个著名的有限同态加密/全同态加密方案。

**零知识证明。** 预处理阶段协议需要用到一个关于明文知识的零知识证明 (zero-knowledge Proof of Plain-text Knowledge)，它的陈述是公钥  $\text{pk}$  和  $u$  个密文，证明者证明自己知道密文中的明文和随机数，且它们满足  $B_{plain}, B_{rand}$  的界限约束。形式化地说，它证明的关系为：

$$\begin{aligned}\mathcal{R}_{\text{PoPK}} = \{(x, w) \mid x = (\text{pk}, \mathbf{c}), w = ((\mathbf{x}_1, \mathbf{r}_1), \dots, (\mathbf{x}_u, \mathbf{r}_u)) : \\ \mathbf{c} = (c_1, \dots, c_u), c_i \leftarrow \text{Enc}_{\text{pk}}(\mathbf{x}_i, \mathbf{r}_i), \\ \|\mathbf{x}_i\|_\infty \leq B_{plain}, \text{decode}(\mathbf{x}_i) \in (\mathbb{F}_p)^s, \|\mathbf{r}_i\|_\infty \leq B_{rand}\}\end{aligned}$$

零知识证明协议  $\Pi_{\text{PoPK}}$  如图 9.48 所示。

### 零知识证明协议 $\Pi_{\text{PoPK}}$

- 定义  $V = 2u - 1$ . 对于  $i = 1, \dots, V$ , 证明者设置  $\mathbf{y}_i \leftarrow \mathbb{Z}^N, \mathbf{s}_i \leftarrow \mathbb{Z}^d$ , 满足  $\|\mathbf{y}_i\|_\infty \leq N \cdot \tau \cdot u^2 \cdot 2^{\nu u - 1}, \|\mathbf{s}_i\|_\infty \leq d \cdot \rho \cdot u^2 \cdot 2^{\nu u - 1}$ . 对于  $\mathbf{y}_i$ , 通过如下方式实现：选择随机消息  $\mathbf{m}_i \leftarrow_{\$} (\mathbb{F}_p)^s$ , 设置  $\mathbf{y}_i = \text{encode}(\mathbf{m}_i) + \mathbf{u}_i$ , 其中  $\mathbf{u}_i$  的每个位置都是  $p$  的倍数，在  $\|\mathbf{y}_i\|_\infty \leq N \cdot \tau \cdot u^2 \cdot 2^{\nu u - 1}$  的条件下均匀随机选择。如果  $\text{diag}$  被设置为 true, 那么选择  $\mathbf{m}_i$  时每个位置都相同。
- 对于  $i = 1, \dots, V$ , 证明者计算  $a_i \leftarrow \text{Enc}_{\text{pk}}(\mathbf{y}_i, \mathbf{s}_i)$ , 然后定义  $\mathbf{S} \in \mathbb{Z}^{V \times d}$  是第  $i$  行为  $\mathbf{s}_i$  的矩阵，并设置  $\mathbf{y} = (\mathbf{y}_1, \dots, \mathbf{y}_V), \mathbf{a} = (a_1, \dots, a_V)$ .
- 证明者向验证者发送  $\mathbf{a}$ .
- 验证者选择  $\mathbf{e} \leftarrow_{\$} \{0, 1\}^u$  并发送给证明者。
- 证明者设置  $\mathbf{z} = (\mathbf{z}_1, \dots, \mathbf{z}_V)$ , 其中  $\mathbf{z}^T = \mathbf{y}^T + \mathbf{M}_e \cdot \mathbf{x}^T, \mathbf{x} = (\mathbf{x}_1, \dots, \mathbf{x}_u)$ . 设置  $\mathbf{T} = \mathbf{S} + \mathbf{M}_e \cdot \mathbf{R}$ . 这里的矩阵  $\mathbf{R}$  的第  $i$  行是  $\mathbf{r}_i$ ; 矩阵  $\mathbf{M}_e$  是一个  $V \times u$  的矩阵，对于  $1 \leq i - k + 1 \leq u$ , 它的第  $(i, k)$  个位置是  $\mathbf{e}_{i-k+1}$ , 其他位置为 0. 证明者向验证者发送  $(\mathbf{z}, \mathbf{T})$ .
- 对于  $i = 1, \dots, V$ , 验证者计算  $d_i \leftarrow \text{Enc}_{\text{pk}}(\mathbf{z}_i, \mathbf{t}_i)$ , 其中  $\mathbf{t}_i$  是矩阵  $\mathbf{T}$  的第  $i$  行。设置  $\mathbf{d} = (d_1, \dots, d_V)$ .
- 验证者检查  $\text{decode}(\mathbf{z}_i) \in (\mathbb{F}_p)^s$  以及以下 3 个条件, 如果不成立则拒绝:
 
$$\mathbf{d}^T = \mathbf{a}^T \boxplus (\mathbf{M}_e \boxtimes \mathbf{c}^T), \|\mathbf{z}_i\|_\infty \leq N \cdot \tau \cdot u^2 \cdot 2^{\nu u - 1}, \|\mathbf{t}_i\|_\infty \leq d \cdot \rho \cdot u^2 \cdot 2^{\nu u - 1}$$
- 如果  $\text{diag}$  被设置为 true, 额外检查  $\text{decode}(\mathbf{z}_i)$  每个位置都相同, 如果不成立则拒绝。

图 9.48: 零知识证明协议  $\Pi_{\text{PoPK}}$

**直观思想。**因为有限同态加密既满足加法同态也满足乘法同态,是一个非常强大的密码学原语,因此基于它实现预处理阶段并不困难。简单来说,当生成认证 Beaver 三元组  $(\langle a \rangle, \langle b \rangle, \langle c \rangle)$  时,参与方  $P_i$  首先在本地随机生成  $a$  和  $b$  的秘密份额  $a_i$  和  $b_i$ ,然后所有参与方公布  $\text{Enc}_{\text{pk}}(a_1), \dots, \text{Enc}_{\text{pk}}(a_n), \text{Enc}_{\text{pk}}(b_1), \dots, \text{Enc}_{\text{pk}}(b_n)$ ,并通过零知识证明  $\Pi_{\text{PoPK}}$  证明合法性。基于加密方案的同态性,可以计算  $\text{Enc}_{\text{pk}}(c) = (\text{Enc}_{\text{pk}}(a_1) \boxplus \dots \boxplus \text{Enc}_{\text{pk}}(a_n)) \boxtimes (\text{Enc}_{\text{pk}}(b_1) \boxplus \dots \boxplus \text{Enc}_{\text{pk}}(b_n))$ ,然后调用子协议  $\text{Reshare}$  通过分布式解密获得  $c$  的秘密份额。对于秘密值的 MAC 标签,也可以通过类似的方法计算。

**协议描述。**我们给出预处理阶段的协议描述,它需要调用两个理想功能:  $\mathcal{F}_{\text{KeyGenDec}}$  (图 9.47) 和  $\mathcal{F}_{\text{Rand}}$  (图 9.33)。

我们将预处理阶段协议的一些部分写成子协议,它们分别是:

- $\Pi_{\text{Reshare}}$ : 它的输入是  $e_m = \text{Enc}_{\text{pk}}(m)$  和  $enc = \text{NewCiphertext}/\text{NoNewCiphertext}$ . 协议使参与方获得  $m$  的秘密分享,并且如果  $enc = \text{NewCiphertext}$ ,额外输出  $e'_m$ ,它将密文  $e_m$  重随机化(re-randomize)。协议如图 9.49 所示。
- $\Pi_{\text{PBracket}}$ : 用于生成  $\llbracket \cdot \rrbracket$  秘密分享,如图 9.50 所示。
- $\Pi_{\text{PAngle}}$ : 用于生成  $\langle \cdot \rangle$  秘密分享,如图 9.51 所示。

子协议  $\Pi_{\text{Reshare}}$

$\text{Reshare}(e_m, enc)$  :

1. 每个参与方  $P_i$  均匀随机选择  $\mathbf{f}_i \leftarrow_{\$} (\mathbb{F}_p)^s$ . 定义  $\mathbf{f} = \sum_{i=1}^n \mathbf{f}_i$ .
2. 每个参与方  $P_i$  计算并广播  $e_{\mathbf{f}_i} \leftarrow \text{Enc}_{\text{pk}}(\mathbf{f}_i)$ .
3. 每个参与方  $P_i$  作为证明者以  $e_{\mathbf{f}_i}$  为陈述运行  $\Pi_{\text{PoPK}}$ . 如果证明不通过则终止协议。
4. 参与方计算  $e_{\mathbf{f}} = e_{\mathbf{f}_1} \boxplus \dots \boxplus e_{\mathbf{f}_n}$  以及  $e_{m+\mathbf{f}} = e_m \boxplus e_{\mathbf{f}}$ .
5. 参与方调用  $\mathcal{F}_{\text{KeyGenDec}}$  解密  $e_{m+\mathbf{f}}$  得到  $\mathbf{m} + \mathbf{f}$ .
6.  $P_1$  设置  $\mathbf{m}_1 = \mathbf{m} + \mathbf{f} - \mathbf{f}_1$ , 其他参与方  $P_i, i \neq 1$  设置  $\mathbf{m}_i = -\mathbf{f}_i$  作为输出.
7. 如果  $enc = \text{NewCiphertext}$ , 参与方计算  $e'_m = \text{Enc}_{\text{pk}}(\mathbf{m} + \mathbf{f}) \boxminus e_{\mathbf{f}_1} \boxminus \dots \boxminus e_{\mathbf{f}_n}$  作为输出, 其中  $\text{Enc}_{\text{pk}}(\mathbf{m} + \mathbf{f})$  在计算时使用默认的固定随机数。

图 9.49: 子协议  $\Pi_{\text{Reshare}}$

最后,我们给出预处理阶段协议  $\Pi_{\text{prep}}$  的描述,如图 9.52 所示。

我们有以下安全性定理。

**定理 9.27.** 假设被攻陷方数量不超过  $n - 1$ . 图 9.52 中的协议  $\Pi_{\text{prep}}$  在  $\{\mathcal{F}_{\text{KeyGen}}^{23}, \mathcal{F}_{\text{Rand}}\}$ -混合模型下对于静态恶意对手  $UC$ -安全实现了理想功能  $\mathcal{F}_{\text{prep}}$  (图 9.44)。

该定理的证明请参阅文献<sup>[72]</sup>。

<sup>23</sup>因为  $\mathcal{F}_{\text{KeyGenDec}}$  可以基于  $\mathcal{F}_{\text{KeyGen}}$  实现,所以在定理中仅假设  $\mathcal{F}_{\text{KeyGen}}$ .

子协议  $\Pi_{\text{PBracket}}$

$\text{PBracket}(\mathbf{v}_1, \dots, \mathbf{v}_n, e_{\mathbf{v}})$  :

1. 对于  $i = 1, \dots, n$ 
  - (a) 所有参与方设置  $e_{M_i} = e_{\beta_i} \boxtimes e_{\mathbf{v}}$  ( $e_{\beta_i}$  是在初始化阶段生成的公开值)。
  - (b) 参与方计算  $(M_i^1, \dots, M_i^n) \leftarrow \text{Reshare}(e_{M_i}, \text{NoNewCiphertext})$ , 每个参与方  $P_j$  获得  $\beta_i \cdot \mathbf{v}$  的份额  $M_i^j$ .
2. 输出  $[\![\mathbf{v}]\!] = (\mathbf{v}_1, \dots, \mathbf{v}_n, (\beta_i, M_1^i, \dots, M_n^i)_{i \in [n]})$ .

图 9.50: 子协议  $\Pi_{\text{PBracket}}$ 

子协议  $\Pi_{\text{PAngle}}$

$\text{PAngle}(\mathbf{v}_1, \dots, \mathbf{v}_n, e_{\mathbf{v}})$  :

1. 所有参与方设置  $e_{\Delta \cdot \mathbf{v}} = e_{\Delta} \boxtimes e_{\mathbf{v}}$  ( $e_{\Delta}$  是在初始化阶段生成的公开值)。
2. 参与方计算  $(M_1, \dots, M_n) \leftarrow \text{Reshare}(e_{\Delta \cdot \mathbf{v}}, \text{NoNewCiphertext})$ , 每个参与方  $P_j$  获得  $\Delta \cdot \mathbf{v}$  的份额  $M_i$ .
3. 输出  $\langle \mathbf{v} \rangle = (\mathbf{v}_1, \dots, \mathbf{v}_n, M_1, \dots, M_n)$ .

图 9.51: 子协议  $\Pi_{\text{PAngle}}$

$\{\mathcal{F}_{\text{KeyGenDec}}, \mathcal{F}_{\text{Rand}}\}$ -混合模型下的协议  $\Pi_{\text{prep}}$

**初始化：**该步骤生成全局密钥  $\Delta$  和私人密钥  $\beta_i$ .

1. 参与方向  $\mathcal{F}_{\text{KeyGenDec}}$  发送  $(\text{Start}, \text{sid})$  获得公钥  $\text{pk}$ .
2. 每个参与方  $P_i$  生成 MAC 密钥  $\beta_i \leftarrow_{\$} \mathbb{F}_p$ .
3. 每个参与方  $P_i$  生成  $\Delta_i \leftarrow_{\$} \mathbb{F}_p$ . 令  $\Delta = \sum_{i=1}^n \Delta_i$ .
4. 每个参与方  $P_i$  计算并广播  $e_{\Delta_i} \leftarrow \text{Enc}_{\text{pk}}(\text{Diag}_s(\Delta_i)), e_{\beta_i} \leftarrow \text{Enc}_{\text{pk}}(\text{Diag}_s(\beta_i))$ .
5. 每个参与方  $P_i$  作为证明者分别以  $(e_{\Delta_1}, \dots, e_{\Delta_n})$  和  $(e_{\beta_1}, \dots, e_{\beta_n})$  为陈述运行  $\Pi_{\text{PoPK}}$  ( $\text{diag}$  设置为 true), 其中陈述中的  $e_{\Delta_i}$  和  $e_{\beta_i}$  有  $u$  个。
6. 所有参与方计算  $e_{\Delta} = e_{\Delta_1} \boxplus \dots \boxplus e_{\Delta_n}$ , 然后生成  $\llbracket \text{Diag}_s(\Delta) \rrbracket \leftarrow \text{PBracket}(\text{Diag}_s(\Delta_1), \dots, \text{Diag}_s(\Delta_n), e_{\Delta})$ .

**秘密分享对：**该步骤生成认证秘密分享对  $\langle \mathbf{r} \rangle, \llbracket \mathbf{r} \rrbracket$ , 也可以用于生成  $\llbracket \mathbf{r} \rrbracket$ .

1. 每个参与方生成  $\mathbf{r}_i \leftarrow_{\$} (\mathbb{F}_p)^s$ . 令  $\mathbf{r} = \sum_{i=1}^n \mathbf{r}_i$ .
2. 每个参与方  $P_i$  计算并广播  $e_{\mathbf{r}_i} \leftarrow \text{Enc}_{\text{pk}}(\mathbf{r}_i)$ . 令  $e_{\mathbf{r}} = e_{\mathbf{r}_1} \boxplus \dots \boxplus e_{\mathbf{r}_n}$ .
3. 每个参与方  $P_i$  作为证明者以他生成的密文为陈述运行  $\Pi_{\text{PoPK}}$ .
4. 参与方生成  $\langle \mathbf{r} \rangle \leftarrow \text{PAngle}(\mathbf{r}_1, \dots, \mathbf{r}_n, e_{\mathbf{r}}), \llbracket \mathbf{r} \rrbracket \leftarrow \text{PBracket}(\mathbf{r}_1, \dots, \mathbf{r}_n, e_{\mathbf{r}})$ .

**三元组：**该步骤生成认证 Beaver 三元组  $\langle \mathbf{a} \rangle, \langle \mathbf{b} \rangle, \langle \mathbf{c} \rangle$ .

1. 每个参与方生成  $\mathbf{a}_i, \mathbf{b}_i \leftarrow_{\$} (\mathbb{F}_p)^s$ . 令  $\mathbf{a} = \sum_{i=1}^n \mathbf{a}_i, \mathbf{b} = \sum_{i=1}^n \mathbf{b}_i$ .
2. 每个参与方  $P_i$  计算并广播  $e_{\mathbf{a}_i} \leftarrow \text{Enc}_{\text{pk}}(\mathbf{a}_i), e_{\mathbf{b}_i} \leftarrow \text{Enc}_{\text{pk}}(\mathbf{b}_i)$ .
3. 每个参与方  $P_i$  作为证明者以他生成的密文为陈述运行  $\Pi_{\text{PoPK}}$ .
4. 参与方计算  $e_{\mathbf{a}} = e_{\mathbf{a}_1} \boxplus \dots \boxplus e_{\mathbf{a}_n}, e_{\mathbf{b}} = e_{\mathbf{b}_1} \boxplus \dots \boxplus e_{\mathbf{b}_n}$ .
5. 参与方生成  $\langle \mathbf{a} \rangle \leftarrow \text{PAngle}(\mathbf{a}_1, \dots, \mathbf{a}_n, e_{\mathbf{a}}), \langle \mathbf{b} \rangle \leftarrow \text{PAngle}(\mathbf{b}_1, \dots, \mathbf{b}_n, e_{\mathbf{b}})$ .
6. 参与方计算  $e_{\mathbf{c}} = e_{\mathbf{a}} \boxtimes e_{\mathbf{b}}$ .
7. 参与方计算  $(\mathbf{c}_1, \dots, \mathbf{c}_n, e'_{\mathbf{c}}) \leftarrow \text{Reshare}(e_{\mathbf{c}}, \text{NewCiphertext})$ .
8. 参与方生成  $\langle \mathbf{c} \rangle \leftarrow \text{PAngle}(\mathbf{c}_1, \dots, \mathbf{c}_n, e'_{\mathbf{c}})$ .

图 9.52:  $\{\mathcal{F}_{\text{KeyGenDec}}, \mathcal{F}_{\text{Rand}}\}$ -混合模型下的协议  $\Pi_{\text{prep}}$



# 后记

当我们写下本书最后一个章节的句点时，内心无比激动与欣慰。从动笔到完稿，历时一年半有余，这段写作旅程充满挑战，也收获颇丰。在整个过程中，团队倾注了大量心血，力求结构连贯、内容深入浅出，真正做到为初学者的自学之路提供切实可行的帮助。在此，我们衷心感谢在本书写作过程中提供宝贵意见与帮助的课题组师生！是你们的付出，让这本书得以顺利完成。

让我们先一起回顾一下在本书中走过的历程吧！整体上，本书可以分成三个部分：

- 第一部分：第 1~3 章。这部分是一个引子，通过两个具有启发性的协议，引出安全多方计算的概念，以及对隐私性、恶意安全性的讨论。
- 第二部分：第 4 章。这部分在对隐私性和恶意安全性的讨论中，引出形式化的通用可组合安全框架。
- 第三部分：第 5~9 章。这部分介绍经典的安全多方计算协议，包括协议的直观思想、形式化描述、安全性证明。

第三部分所介绍的协议并不是互相独立的，它们存在着许多内在联系。例如，*BMR* 协议是姚氏混淆电路协议在多方场景下的拓展，它在计算混淆表时又需要调用其他协议（例如 *BGW* 协议）；*LP11* 协议通过切分选择技术使姚氏混淆电路协议获得恶意安全性；*BGW* 协议（恶意安全）通过纠错码和可验证秘密分享获得恶意安全性；*BDOZ* 协议和 *SPDZ* 协议在 *Beaver* 三元组的基础上通过消息认证码获得恶意安全性。希望读者在阅读完本书的时候，也能形成这样的知识体系和脉络。

本书最初拟定的书名是《安全多方计算——从入门到精通》。然而，在写作的调研过程中，我们愈发意识到，安全多方计算作为一个活跃且高速发展的研究领域，方向繁多、内容庞杂，任何研究者都难以在每个分支都达到“精通”。因此，我们认为本书不宜追求面面俱到、泛泛而谈，而应选择一个聚焦的切入口。最终，我们决定以“可证明安全”作为主线展开论述。由于时间所限，一些重要但未能覆盖的内容，例如 *ABY* 框架、隐匿求交 (Private Set Intersection, PSI)、隐匿信息检索 (Private Information Retrieval, PIR) 等协议，暂未纳入本书。对此我们感到略有遗憾，也期待未来有机会在第二版中将这些内容补充完善，与读者继续分享这一领域的精彩。

不过，情况并不悲观。一旦掌握了 MPC 的安全框架，并理解了基础的经典协议，你就建立起了属于自己的知识体系——如同一棵大树已然扎下坚实的树干。之后，你将能够自主阅读各类论文，追踪领域最新进展，提出自己的创新思路——在这棵树干上不断生长出新的枝条，逐步拓展广度与深度，最终成长为参天大树……



# 参考文献

- [1] CRAMER R, DAMGÅRD I B, NIELSEN J B. Secure multiparty computation and secret sharing [M]. Cambridge University Press, 2015.
- [2] BONEH D, SHOUP V. A graduate course in applied cryptography. Draft 0.6[M]. 2023.
- [3] KATZ J, LINDELL Y. Introduction to modern cryptography: principles and protocols[M]. Chapman, 2007.
- [4] BEIMEL A, CHOR B. Universally Ideal Secret Sharing Schemes (Preliminary Version)[C/OL]// BRICKELL E F. Lecture Notes in Computer Science: Advances in Cryptology – CRYPTO'92: vol. 740. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 1993: 183-195. DOI: 10.1007 /3-540-48071-4\_13.
- [5] BELLARE M, ROGAWAY P. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols[C/OL]//DENNING D E, PYLE R, GANESAN R, et al. ACM CCS 93: 1st Conference on Computer and Communications Security. Fairfax, Virginia, USA: ACM Press, 1993: 62-73. DOI: 10.1145/168588.168596.
- [6] CANETTI R, GOLDREICH O, HALEVI S. The Random Oracle Methodology, Revisited (Preliminary Version)[C/OL]//30th Annual ACM Symposium on Theory of Computing. Dallas, TX, USA: ACM Press, 1998: 209-218. DOI: 10.1145/276698.276741.
- [7] SHAMIR A. How to Share a Secret[J/OL]. Communications of the Association for Computing Machinery, 1979, 22(11): 612-613. DOI: 10.1145/359168.359176.
- [8] CANETTI R. Universally Composable Security: A New Paradigm for Cryptographic Protocols [C/OL]//42nd Annual Symposium on Foundations of Computer Science. Las Vegas, NV, USA: IEEE Computer Society Press, 2001: 136-145. DOI: 10.1109/SFCS.2001.959888.
- [9] CANETTI R. Security and Composition of Multiparty Cryptographic Protocols[J/OL]. Journal of Cryptology, 2000, 13(1): 143-202. DOI: 10.1007/s001459910006.
- [10] GOLDREICH O. The Foundations of Cryptography - Volume 2: Basic Applications[M/OL]. Cambridge University Press, 2004. <http://www.wisdom.weizmann.ac.il/%5C%7Eoded/foc-vol2.html>. DOI: 10.1017/CBO9780511721656.
- [11] ISHAI Y, KILIAN J, NISSIM K, et al. Extending Oblivious Transfers Efficiently[C/OL]//BONEH D. Lecture Notes in Computer Science: Advances in Cryptology – CRYPTO 2003: vol. 2729. Santa

- Barbara, CA, USA: Springer, Heidelberg, Germany, 2003: 145-161. DOI: 10.1007/978-3-540-45146-4\_9.
- [12] RABIN M O. How to Exchange Secrets with Oblivious Transfer[J]. Technical report, 1981.
- [13] CRÉPEAU C. Equivalence Between Two Flavours of Oblivious Transfers[C/OL]//POMERANCE C. Lecture Notes in Computer Science: Advances in Cryptology – CRYPTO'87: vol. 293. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 1988: 350-354. DOI: 10.1007/3-540-48184-2\_30.
- [14] NAOR M, PINKAS B. Oblivious Transfer and Polynomial Evaluation[C/OL]//31st Annual ACM Symposium on Theory of Computing. Atlanta, GA, USA: ACM Press, 1999: 245-254. DOI: 10.1145/301250.301312.
- [15] KILIAN J. Founding Cryptography on Oblivious Transfer[C/OL]//20th Annual ACM Symposium on Theory of Computing. Chicago, IL, USA: ACM Press, 1988: 20-31. DOI: 10.1145/62212.62215.
- [16] ISHAI Y, PRABHAKARAN M, SAHAI A. Founding Cryptography on Oblivious Transfer - Efficiently[C/OL]//WAGNER D. Lecture Notes in Computer Science: Advances in Cryptology – CRYPTO 2008: vol. 5157. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 2008: 572-591. DOI: 10.1007/978-3-540-85174-5\_32.
- [17] WOLF S, WULLSCHLEGER J. Oblivious Transfer Is Symmetric[C/OL]//VAUDENAY S. Lecture Notes in Computer Science: Advances in Cryptology – EUROCRYPT 2006: vol. 4004. St. Petersburg, Russia: Springer, Heidelberg, Germany, 2006: 222-232. DOI: 10.1007/11761679\_14.
- [18] GERTNER Y, KANNAN S, MALKIN T, et al. The Relationship between Public Key Encryption and Oblivious Transfer[C/OL]//41st Annual Symposium on Foundations of Computer Science. Redondo Beach, CA, USA: IEEE Computer Society Press, 2000: 325-335. DOI: 10.1109/SFCS.2000.892121.
- [19] EVEN S, GOLDREICH O, LEMPEL A. A Randomized Protocol for Signing Contracts[J/OL]. Commun. ACM, 1985, 28(6): 637-647. <https://doi.org/10.1145/3812.3818>. DOI: 10.1145/3812.3818.
- [20] GOLDREICH O, MICALI S, WIGDERSON A. How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority[C/OL]//AHO A. 19th Annual ACM Symposium on Theory of Computing. New York City, NY, USA: ACM Press, 1987: 218-229. DOI: 10.1145/28395.28420.
- [21] PEIKERT C, VAIKUNTANATHAN V, WATERS B. A Framework for Efficient and Composable Oblivious Transfer[C/OL]//WAGNER D. Lecture Notes in Computer Science: Advances in Cryptology – CRYPTO 2008: vol. 5157. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 2008: 554-571. DOI: 10.1007/978-3-540-85174-5\_31.
- [22] IMPAGLIAZZO R, RUDICH S. Limits on the Provable Consequences of One-way Permutations[C/OL]//GOLDWASSER S. Lecture Notes in Computer Science: Advances in Cryptology

- CRYPTO’88: vol. 403. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 1990: 8-26. DOI: 10.1007/0-387-34799-2\_2.
- [23] CHOU T, ORLANDI C. The Simplest Protocol for Oblivious Transfer[C/OL]//LAUTER K E, RODRÍGUEZ-HENRÍQUEZ F. Lecture Notes in Computer Science: Progress in Cryptology - LATINCRYPT 2015: 4th International Conference on Cryptology and Information Security in Latin America: vol. 9230. Guadalajara, Mexico: Springer, Heidelberg, Germany, 2015: 40-58. DOI: 10.1007/978-3-319-22174-8\_3.
- [24] HAUCK E, LOSS J. Efficient and Universally Composable Protocols for Oblivious Transfer from the CDH Assumption[Z]. Cryptology ePrint Archive, Report 2017/1011. <https://eprint.iacr.org/2017/1011>. 2017.
- [25] BARRETO P S L M, DAVID B, DOWSLEY R, et al. A Framework for Efficient Adaptively Secure Composable Oblivious Transfer in the ROM[Z]. Cryptology ePrint Archive, Report 2017/993. <https://eprint.iacr.org/2017/993>. 2017.
- [26] BEAVER D. Correlated Pseudorandomness and the Complexity of Private Computations[C/OL] //28th Annual ACM Symposium on Theory of Computing. Philadelphia, PA, USA: ACM Press, 1996: 479-488. DOI: 10.1145/237814.237996.
- [27] ASHAROV G, LINDELL Y, SCHNEIDER T, et al. More efficient oblivious transfer and extensions for faster secure computation[C/OL]//SADEGHI A R, GLIGOR V D, YUNG M. ACM CCS 2013: 20th Conference on Computer and Communications Security. Berlin, Germany: ACM Press, 2013: 535-548. DOI: 10.1145/2508859.2516738.
- [28] KELLER M, ORSINI E, SCHOLL P. Actively Secure OT Extension with Optimal Overhead [C/OL]//GENNARO R, ROBshaw M J B. Lecture Notes in Computer Science: Advances in Cryptology – CRYPTO 2015, Part I: vol. 9215. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 2015: 724-741. DOI: 10.1007/978-3-662-47989-6\_35.
- [29] BOYLE E, COUTEAU G, GILBOA N, et al. Efficient Two-Round OT Extension and Silent Non-Interactive Secure Computation[C/OL]//CAVALLARO L, KINDER J, WANG X, et al. ACM CCS 2019: 26th Conference on Computer and Communications Security. London, UK: ACM Press, 2019: 291-308. DOI: 10.1145/3319535.3354255.
- [30] ROY L. SoftSpokenOT: Quieter OT Extension from Small-Field Silent VOLE in the Minicrypt Model[C/OL]//DODIS Y, SHRIMPTON T. Lecture Notes in Computer Science: Advances in Cryptology – CRYPTO 2022, Part I: vol. 13507. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 2022: 657-687. DOI: 10.1007/978-3-031-15802-5\_23.
- [31] MASNY D, RINDAL P. Endemic Oblivious Transfer[C/OL]//CAVALLARO L, KINDER J, WANG X, et al. ACM CCS 2019: 26th Conference on Computer and Communications Security. London, UK: ACM Press, 2019: 309-326. DOI: 10.1145/3319535.3354210.
- [32] LibOTe. libOTe[EB/OL]. 2024 [2024-07-12]. <https://github.com/osu-crypto/libOTe>.

- [33] BEN-OR M, GOLDWASSER S, WIGDERSON A. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation (Extended Abstract)[C/OL]//20th Annual ACM Symposium on Theory of Computing. Chicago, IL, USA: ACM Press, 1988: 1-10. DOI: 10.1145/62212.62213.
- [34] DÖTTLING N, KRASCHEWSKI D, MÜLLER-QUADE J. Statistically Secure Linear-Rate Dimension Extension for Oblivious Affine Function Evaluation[C/OL]//SMITH A. Lecture Notes in Computer Science: ICITS 12: 6th International Conference on Information Theoretic Security: vol. 7412. Montreal, QC, Canada: Springer, Heidelberg, Germany, 2012: 111-128. DOI: 10.1007/978-3-642-32284-6\_7.
- [35] BEAVER D. Efficient Multiparty Protocols Using Circuit Randomization[C/OL]//FEIGENBAUM J. Lecture Notes in Computer Science: Advances in Cryptology – CRYPTO’91: vol. 576. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 1992: 420-432. DOI: 10.1007/3-540-46766-1\_34.
- [36] GILBOA N. Two Party RSA Key Generation[C/OL]//WIENER M J. Lecture Notes in Computer Science: Advances in Cryptology – CRYPTO’99: vol. 1666. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 1999: 116-129. DOI: 10.1007/3-540-48405-1\_8.
- [37] DEMMLER D, SCHNEIDER T, ZOHNER M. ABY - A Framework for Efficient Mixed-Protocol Secure Two-Party Computation[C]//ISOC Network and Distributed System Security Symposium – NDSS 2015. San Diego, CA, USA: The Internet Society, 2015.
- [38] PULLONEN P, BOGDANOV D, SCHNEIDER T. The design and implementation of a two-party protocol suite for Sharemind 3[J]. CYBERNETICA Institute of Information Security, Tech. Rep, 2012, 4: 17.
- [39] YAO A C C. How to Generate and Exchange Secrets (Extended Abstract)[C/OL]//27th Annual Symposium on Foundations of Computer Science. Toronto, Ontario, Canada: IEEE Computer Society Press, 1986: 162-167. DOI: 10.1109/SFCS.1986.25.
- [40] BEAVER D, MICALI S, ROGAWAY P. The Round Complexity of Secure Protocols (Extended Abstract)[C/OL]//22nd Annual ACM Symposium on Theory of Computing. Baltimore, MD, USA: ACM Press, 1990: 503-513. DOI: 10.1145/100216.100287.
- [41] LINDELL Y, PINKAS B. A Proof of Security of Yao’s Protocol for Two-Party Computation [J/OL]. Journal of Cryptology, 2009, 22(2): 161-188. DOI: 10.1007/s00145-008-9036-8.
- [42] CLEVE R. Limits on the Security of Coin Flips when Half the Processors Are Faulty (Extended Abstract)[C/OL]//18th Annual ACM Symposium on Theory of Computing. Berkeley, CA, USA: ACM Press, 1986: 364-369. DOI: 10.1145/12130.12168.
- [43] CANETTI R. Universally Composable Security: A New Paradigm for Cryptographic Protocols [Z]. Cryptology ePrint Archive, Report 2000/067. <https://eprint.iacr.org/2000/067>. 2000.
- [44] NAOR M, PINKAS B, SUMNER R. Privacy preserving auctions and mechanism design[C/OL] //FELDMAN S I, WELLMAN M P. Proceedings of the First ACM Conference on Electronic

- Commerce (EC-99), Denver, CO, USA, November 3-5, 1999. ACM, 1999: 129-139. <https://doi.org/10.1145/336992.337028>. DOI: 10.1145/336992.337028.
- [45] PINKAS B, SCHNEIDER T, SMART N P, et al. Secure Two-Party Computation Is Practical[C/OL]//MATSUI M. Lecture Notes in Computer Science: Advances in Cryptology – ASIACRYPT 2009: vol. 5912. Tokyo, Japan: Springer, Heidelberg, Germany, 2009: 250-267. DOI: 10.1007/978-3-642-10366-7\_15.
- [46] KOLESNIKOV V, SCHNEIDER T. Improved Garbled Circuit: Free XOR Gates and Applications [C/OL]//ACETO L, DAMGÅRD I, GOLDBERG L A, et al. Lecture Notes in Computer Science: ICALP 2008: 35th International Colloquium on Automata, Languages and Programming, Part II: vol. 5126. Reykjavik, Iceland: Springer, Heidelberg, Germany, 2008: 486-498. DOI: 10.1007/978-3-540-70583-3\_40.
- [47] ZAHUR S, ROSULEK M, EVANS D. Two Halves Make a Whole - Reducing Data Transfer in Garbled Circuits Using Half Gates[C/OL]//OSWALD E, FISCHLIN M. Lecture Notes in Computer Science: Advances in Cryptology – EUROCRYPT 2015, Part II: vol. 9057. Sofia, Bulgaria: Springer, Heidelberg, Germany, 2015: 220-250. DOI: 10.1007/978-3-662-46803-6\_8.
- [48] GOLDREICH O, MICALI S, WIGDERSON A. Proofs that Yield Nothing But their Validity and a Methodology of Cryptographic Protocol Design (Extended Abstract)[C/OL]//27th Annual Symposium on Foundations of Computer Science. Toronto, Ontario, Canada: IEEE Computer Society Press, 1986: 174-187. DOI: 10.1109/SFCS.1986.47.
- [49] BRASSARD G, CHAUM D, CRÉPEAU C. Minimum Disclosure Proofs of Knowledge[J/OL]. J. Comput. Syst. Sci., 1988, 37(2): 156-189. [https://doi.org/10.1016/0022-0000\(88\)90005-0](https://doi.org/10.1016/0022-0000(88)90005-0). DOI: 10.1016/0022-0000(88)90005-0.
- [50] GRAHAM R L, KNUTH D E, PATASHNIK O. Concrete Mathematics: A Foundation for Computer Science, 2nd Ed[M/OL]. Addison-Wesley, 1994. <https://www-cs-faculty.stanford.edu/%5C%7Eknuth/gkp.html>.
- [51] KIRAZ M, SCHOENMAKERS B. A protocol issue for the malicious case of Yao’s garbled circuit construction[C]//27th symposium on information theory in the Benelux: vol. 29: 4. 2006: 283-290.
- [52] LINDELL Y, PINKAS B. An Efficient Protocol for Secure Two-Party Computation in the Presence of Malicious Adversaries[C/OL]//NAOR M. Lecture Notes in Computer Science: Advances in Cryptology – EUROCRYPT 2007: vol. 4515. Barcelona, Spain: Springer, Heidelberg, Germany, 2007: 52-78. DOI: 10.1007/978-3-540-72540-4\_4.
- [53] SHELAT A, SHEN C H. Two-Output Secure Computation with Malicious Adversaries[C/OL]//PATERSON K G. Lecture Notes in Computer Science: Advances in Cryptology – EUROCRYPT 2011: vol. 6632. Tallinn, Estonia: Springer, Heidelberg, Germany, 2011: 386-405. DOI: 10.1007/978-3-642-20465-4\_22.
- [54] SHELAT A, SHEN C H. Fast two-party secure computation with minimal assumptions[C/OL]//SADEGHI A R, GLIGOR V D, YUNG M. ACM CCS 2013: 20th Conference on Computer and

- Communications Security. Berlin, Germany: ACM Press, 2013: 523-534. DOI: 10.1145/2508859.2516698.
- [55] KREUTER B, SHELAT A, SHEN C H. Billion-Gate Secure Computation with Malicious Adversaries[C]//KOHNO T. USENIX Security 2012: 21st USENIX Security Symposium. Bellevue, WA, USA: USENIX Association, 2012: 285-300.
- [56] LINDELL Y, PINKAS B. Secure Two-Party Computation via Cut-and-Choose Oblivious Transfer [C/OL]//ISHAI Y. Lecture Notes in Computer Science: TCC 2011: 8th Theory of Cryptography Conference: vol. 6597. Providence, RI, USA: Springer, Heidelberg, Germany, 2011: 329-346. DOI: 10.1007/978-3-642-19571-6\_20.
- [57] HUANG Y, KATZ J, EVANS D. Efficient Secure Two-Party Computation Using Symmetric Cut-and-Choose[C/OL]//CANETTI R, GARAY J A. Lecture Notes in Computer Science: Advances in Cryptology – CRYPTO 2013, Part II: vol. 8043. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 2013: 18-35. DOI: 10.1007/978-3-642-40084-1\_2.
- [58] LINDELL Y. Fast Cut-and-Choose Based Protocols for Malicious and Covert Adversaries[C/OL]//CANETTI R, GARAY J A. Lecture Notes in Computer Science: Advances in Cryptology – CRYPTO 2013, Part II: vol. 8043. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 2013: 1-17. DOI: 10.1007/978-3-642-40084-1\_1.
- [59] BRANDÃO L T A N. Secure Two-Party Computation with Reusable Bit-Commitments, via a Cut-and-Choose with Forge-and-Lose Technique - (Extended Abstract)[C/OL]//SAKO K, SARKAR P. Lecture Notes in Computer Science: Advances in Cryptology – ASIACRYPT 2013, Part II: vol. 8270. Bengalore, India: Springer, Heidelberg, Germany, 2013: 441-463. DOI: 10.1007/978-3-642-42045-0\_23.
- [60] FREDERIKSEN T K, JAKOBSEN T P, NIELSEN J B. Faster Maliciously Secure Two-Party Computation Using the GPU[C/OL]//ABDALLA M, PRISCO R D. Lecture Notes in Computer Science: SCN 14: 9th International Conference on Security in Communication Networks: vol. 8642. Amalfi, Italy: Springer, Heidelberg, Germany, 2014: 358-379. DOI: 10.1007/978-3-319-10879-7\_21.
- [61] AFSHAR A, MOHASSEL P, PINKAS B, et al. Non-Interactive Secure Computation Based on Cut-and-Choose[C/OL]//NGUYEN P Q, OSWALD E. Lecture Notes in Computer Science: Advances in Cryptology – EUROCRYPT 2014: vol. 8441. Copenhagen, Denmark: Springer, Heidelberg, Germany, 2014: 387-404. DOI: 10.1007/978-3-642-55220-5\_22.
- [62] CRAMER R, DAMGÅRD I, SCHOENMAKERS B. Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols[C/OL]//DESMEDT Y. Lecture Notes in Computer Science: Advances in Cryptology – CRYPTO’94: vol. 839. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 1994: 174-187. DOI: 10.1007/3-540-48658-5\_19.
- [63] HAZAY C, LINDELL Y. Efficient Secure Two-Party Protocols - Techniques and Constructions [M/OL]. Springer, 2010. <https://doi.org/10.1007/978-3-642-14303-8>. DOI: 10.1007/978-3-642-14303-8.

- [64] DODIS Y, GENNARO R, HÅSTAD J, et al. Randomness Extraction and Key Derivation Using the CBC, Cascade and HMAC Modes[C/OL]//FRANKLIN M. Lecture Notes in Computer Science: Advances in Cryptology – CRYPTO 2004: vol. 3152. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 2004: 494-510. DOI: 10.1007/978-3-540-28628-8\_30.
- [65] CARTER L, WEGMAN M N. Universal Classes of Hash Functions[J/OL]. *J. Comput. Syst. Sci.*, 1979, 18(2): 143-154. [https://doi.org/10.1016/0022-0000\(79\)90044-8](https://doi.org/10.1016/0022-0000(79)90044-8). DOI: 10.1016/0022-0000(79)90044-8.
- [66] HÅSTAD J, IMPAGLIAZZO R, LEVIN L A, et al. A Pseudorandom Generator from any One-way Function[J/OL]. *SIAM J. Comput.*, 1999, 28(4): 1364-1396. <https://doi.org/10.1137/S0097539793244708>. DOI: 10.1137/S0097539793244708.
- [67] DODIS Y, SHOUP V, WALFISH S. Efficient Constructions of Composable Commitments and Zero-Knowledge Proofs[C/OL]//WAGNER D. Lecture Notes in Computer Science: Advances in Cryptology – CRYPTO 2008: vol. 5157. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 2008: 515-535. DOI: 10.1007/978-3-540-85174-5\_29.
- [68] GARAY J A, MACKENZIE P D, YANG K. Strengthening Zero-Knowledge Protocols Using Signatures[C/OL]//BIHAM E. Lecture Notes in Computer Science: Advances in Cryptology – EUROCRYPT 2003: vol. 2656. Warsaw, Poland: Springer, Heidelberg, Germany, 2003: 177-194. DOI: 10.1007/3-540-39200-9\_11.
- [69] GENNARO R, JARECKI S, KRAWCZYK H, et al. Secure Distributed Key Generation for Discrete-Log Based Cryptosystems[J/OL]. *Journal of Cryptology*, 2007, 20(1): 51-83. DOI: 10.1007/s00145-006-0347-3.
- [70] ASHAROV G, LINDELL Y. A Full Proof of the BGW Protocol for Perfectly Secure Multiparty Computation[J/OL]. *Journal of Cryptology*, 2017, 30(1): 58-151. DOI: 10.1007/s00145-015-9214-4.
- [71] BENDLIN R, DAMGÅRD I, ORLANDI C, et al. Semi-homomorphic Encryption and Multiparty Computation[C/OL]//PATERSON K G. Lecture Notes in Computer Science: Advances in Cryptology – EUROCRYPT 2011: vol. 6632. Tallinn, Estonia: Springer, Heidelberg, Germany, 2011: 169-188. DOI: 10.1007/978-3-642-20465-4\_11.
- [72] DAMGÅRD I, PASTRO V, SMART N P, et al. Multiparty Computation from Somewhat Homomorphic Encryption[C/OL]//SAFAVI-NAINI R, CANETTI R. Lecture Notes in Computer Science: Advances in Cryptology – CRYPTO 2012: vol. 7417. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 2012: 643-662. DOI: 10.1007/978-3-642-32009-5\_38.
- [73] PAILLIER P. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes[C/OL]//STERN J. Lecture Notes in Computer Science: Advances in Cryptology – EUROCRYPT’99: vol. 1592. Prague, Czech Republic: Springer, Heidelberg, Germany, 1999: 223-238. DOI: 10.1007/3-540-48910-X\_16.
- [74] DAMGÅRD I, NIELSEN J B. Perfect Hiding and Perfect Binding Universally Composable Commitment Schemes with Constant Expansion Factor[C/OL]//YUNG M. Lecture Notes in Com-

- puter Science: Advances in Cryptology – CRYPTO 2002: vol. 2442. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 2002: 581-596. DOI: 10.1007/3-540-45708-9\_37.
- [75] LINDELL Y. Highly-Efficient Universally-Composable Commitments Based on the DDH Assumption[C/OL]//PATERSON K G. Lecture Notes in Computer Science: Advances in Cryptology – EUROCRYPT 2011: vol. 6632. Tallinn, Estonia: Springer, Heidelberg, Germany, 2011: 446-466. DOI: 10.1007/978-3-642-20465-4\_25.
- [76] BRAKERSKI Z, GENTRY C, VAIKUNTANATHAN V. (Leveled) fully homomorphic encryption without bootstrapping[C/OL]//GOLDWASSER S. ITCS 2012: 3rd Innovations in Theoretical Computer Science. Cambridge, MA, USA: Association for Computing Machinery, 2012: 309-325. DOI: 10.1145/2090236.2090262.