可证明安全 - 密码学中的安全性证明

钱宸
网络空间安全学院
山东大学

2025.09.22

Contents

Contents

基于模拟的证明定义

基于模拟的证明

什么是模拟?

- ”现实世界”与”理想世界”

基于模拟的证明

什么是模拟?

- ”现实世界”与”理想世界”
- 将所需要的性质抽象成为一个“理想世界”

基于模拟的证明

什么是模拟?

- ”现实世界”与”理想世界”
- 将所需要的性质抽象成为一个“理想世界”

基于模拟的证明

什么是模拟?

- ”现实世界”与”理想世界”
- 将所需要的性质抽象成为一个“理想世界”

真实世界	理想世界
实际执行的协议 运算复杂 可能不安全	定义上就安全的环境 运算简单 定义上就安全

基于模拟的证明

模拟的逻辑

- 如果一个敌手在“真实世界”中攻击一个协议，那么就存在一个敌手在“理想世界”中攻击这个环境，使得两个敌手获得的结果“无法区分”.

基于模拟的证明

模拟的逻辑

- 如果一个敌手在“真实世界”中攻击一个协议，那么就存在一个敌手在“理想世界”中攻击这个环境，使得两个敌手获得的结果“无法区分”.
- **关键:** 理想世界中的敌手不可能成功地攻击这个环境.

基于模拟的证明

模拟的逻辑

- 如果一个敌手在“真实世界”中攻击一个协议，那么就存在一个敌手在“理想世界”中攻击这个环境，使得两个敌手获得的结果“无法区分”.
- **关键:** 理想世界中的敌手不可能成功地攻击这个环境.
- 因此，真实世界中的敌手也不可能成功地攻击这个协议.

基于模拟的证明

模拟的逻辑

- 如果一个敌手在“真实世界”中攻击一个协议，那么就存在一个敌手在“理想世界”中攻击这个环境，使得两个敌手获得的结果“无法区分”.
- **关键:** 理想世界中的敌手不可能成功地攻击这个环境.
- 因此，真实世界中的敌手也不可能成功地攻击这个协议.

基于模拟的证明

模拟的逻辑

- 如果一个敌手在“真实世界”中攻击一个协议，那么就存在一个敌手在“理想世界”中攻击这个环境，使得两个敌手获得的结果“无法区分”.
- **关键:** 理想世界中的敌手不可能成功地攻击这个环境.
- 因此，真实世界中的敌手也不可能成功地攻击这个协议.

与规约的关系

- 在证明这类定义的时候, 需要构造一个模拟器“规约”

基于模拟的证明

模拟的逻辑

- 如果一个敌手在“真实世界”中攻击一个协议，那么就存在一个敌手在“理想世界”中攻击这个环境，使得两个敌手获得的结果“无法区分”.
- **关键:** 理想世界中的敌手不可能成功地攻击这个环境.
- 因此，真实世界中的敌手也不可能成功地攻击这个协议.

与规约的关系

- 在证明这类定义的时候, 需要构造一个模拟器“规约”
- **需要证明:** ”攻击真实环境中的协议”比攻击“攻击理想环境中的协议”更难.

$$\exists \mathcal{A} : \Pr\left[\mathbf{G}_{\mathcal{A}}^{real} \Rightarrow 1\right] > 1 - \mathsf{negl}(\lambda) \implies \exists \mathcal{B} : \Pr\left[\mathbf{G}_{\mathcal{B}}^{ideal} \Rightarrow 1\right] > 1 - \mathsf{negl}(\lambda)$$

基于模拟的证明

模拟的逻辑

- 如果一个敌手在“真实世界”中攻击一个协议，那么就存在一个敌手在“理想世界”中攻击这个环境，使得两个敌手获得的结果“无法区分”.
- **关键:** 理想世界中的敌手不可能成功地攻击这个环境.
- 因此，真实世界中的敌手也不可能成功地攻击这个协议.

与规约的关系

- 在证明这类定义的时候, 需要构造一个模拟器“规约”
- **需要证明:** ” 攻击真实环境中的协议”比攻击“攻击理想环境中的协议”更难.

$$\exists \mathcal{A} : \Pr\left[\mathbf{G}_{\mathcal{A}}^{real} \Rightarrow 1\right] > 1 - \mathsf{negl}(\lambda) \implies \exists \mathcal{B} : \Pr\left[\mathbf{G}_{\mathcal{B}}^{ideal} \Rightarrow 1\right] > 1 - \mathsf{negl}(\lambda)$$

- $\exists R \in \mathsf{PPT} : R(\mathcal{A}) = \mathcal{B}$.

语义安全性与模拟

基于模拟的证明 - 例子: 加密安全性

加密安全性

- “收到密文的敌手获得的信息” VS “什么都没有收到的敌手获得的信息”

基于模拟的证明 - 例子: 加密安全性

加密安全性

- “收到密文的敌手获得的信息” VS “什么都没有收到的敌手获得的信息”
- **关键:** 什么都没有收到的敌手获得的不了任何关于明文的信息.

基于模拟的证明 - 例子: 加密安全性

加密安全性

- “收到密文的敌手获得的信息” VS “什么都没有收到的敌手获得的信息”
- **关键:** 什么都没有收到的敌手获得的不了任何关于明文的信息.
- 如果一个收到密文的敌手获得的信息和一个什么都没有收到的敌手获得的信息“无法区分”，那么这个加密加密算法就是安全的.

基于模拟的证明 - 例子: 加密安全性

加密安全性

- “收到密文的敌手获得的信息” VS “什么都没有收到的敌手获得的信息”
- **关键:** 什么都没有收到的敌手获得的不了任何关于明文的信息.
- 如果一个收到密文的敌手获得的信息和一个什么都没有收到的敌手获得的信息“无法区分”，那么这个加密加密算法就是安全的.

基于模拟的证明 - 例子: 加密安全性

加密安全性

- “收到密文的敌手获得的信息” VS “什么都没有收到的敌手获得的信息”
- **关键:** 什么都没有收到的敌手获得的不了任何关于明文的信息.
- 如果一个收到密文的敌手获得的信息和一个什么都没有收到的敌手获得的信息“无法区分”，那么这个加密加密算法就是安全的.

好像非常**多此一举**

- 为什么不直接定义“如果敌手不能获得明文的任何信息, 则加密算法是安全的”

基于模拟的证明 - 例子: 加密安全性

加密安全性

- “收到密文的敌手获得的信息” VS “什么都没有收到的敌手获得的信息”
- **关键:** 什么都没有收到的敌手获得的不了任何关于明文的信息.
- 如果一个收到密文的敌手获得的信息和一个什么都没有收到的敌手获得的信息“无法区分”，那么这个加密加密算法就是安全的.

好像非常**多此一举**

- 为什么不直接定义“如果敌手不能获得明文的任何信息, 则加密算法是安全的”
- **不行!** 直接定义“敌手不能获得明文的任何信息”是**不对的**

基于模拟的证明 - 例子: 加密安全性

加密安全性

- “收到密文的敌手获得的信息” VS “什么都没有收到的敌手获得的信息”
- **关键:** 什么都没有收到的敌手获得的不了任何关于明文的信息.
- 如果一个收到密文的敌手获得的信息和一个什么都没有收到的敌手获得的信息“无法区分”，那么这个加密加密算法就是安全的.

好像非常**多此一举**

- 为什么不直接定义“如果敌手不能获得明文的任何信息, 则加密算法是安全的”
- **不行!** 直接定义“敌手不能获得明文的任何信息”是**不对的**
- 因为敌手可能获得一些关于明文的信息.

基于模拟的证明 - 例子: 加密安全性

加密安全性

- “收到密文的敌手获得的信息” VS “什么都没有收到的敌手获得的信息”
- **关键:** 什么都没有收到的敌手获得的不了任何关于明文的信息.
- 如果一个收到密文的敌手获得的信息和一个什么都没有收到的敌手获得的信息“无法区分”，那么这个加密加密算法就是安全的.

好像非常**多此一举**

- 为什么不直接定义“如果敌手不能获得明文的任何信息, 则加密算法是安全的”
- **不行!** 直接定义“敌手不能获得明文的任何信息”是**不对的**
- 因为敌手可能获得一些关于明文的信息.
- 例如: 明文的长度, 明文是否为 0,1 等

基于模拟的证明 - 例子: 加密安全性

- **合理的加密安全性:** 敌手从密文中不能获得关于明文的任何“额外信息”.
- **“额外信息”:** 敌手从密文中获得的信息, **减去**敌手什么都没有收到时获得的信息.

与模拟的关系

- **从规约的角度:** $\exists R \in \mathsf{PPT} : R(\mathcal{A}^{real}) = \mathcal{B}^{ideal}$

基于模拟的证明 - 例子: 加密安全性

- **合理的加密安全性:** 敌手从密文中不能获得关于明文的任何“额外信息”.
- **“额外信息”:** 敌手从密文中获得的信息, **减去**敌手什么都没有收到时获得的信息.

与模拟的关系

- **从规约的角度:** $\exists R \in \mathsf{PPT} : R(\mathcal{A}^{real}) = \mathcal{B}^{ideal}$
- R **需要满足的性质:**

基于模拟的证明 - 例子: 加密安全性

- **合理的加密安全性:** 敌手从密文中不能获得关于明文的任何“额外信息”.
- **“额外信息”:** 敌手从密文中获得的信息, **减去**敌手什么都没有收到时获得的信息.

与模拟的关系

- **从规约的角度:** $\exists R \in \mathsf{PPT} : R(\mathcal{A}^{real}) = \mathcal{B}^{ideal}$
- R **需要满足的性质:**
  1. **调用** $\mathcal{A}^{real}$**:** R 需要在只与理想游戏交互的情况下为 $\mathcal{A}^{real}$**模拟**一个真实的环境.

基于模拟的证明 - 例子: 加密安全性

- **合理的加密安全性:** 敌手从密文中不能获得关于明文的任何“额外信息”.
- **“额外信息”:** 敌手从密文中获得的信息, **减去**敌手什么都没有收到时获得的信息.

与模拟的关系

- **从规约的角度:** $\exists R \in \mathsf{PPT} : R(\mathcal{A}^{real}) = \mathcal{B}^{ideal}$
- R **需要满足的性质:**
  1. **调用** $\mathcal{A}^{real}$**:** R 需要在只与理想游戏交互的情况下为 $\mathcal{A}^{real}$**模拟**一个真实的环境.
  2. **输出结果:** R 需要输出 $\mathcal{A}^{real}$ 的输出, 作为 $\mathcal{B}^{ideal}$ 的输出.

基于模拟的证明 - 例子: 加密安全性

- **合理的加密安全性:** 敌手从密文中不能获得关于明文的任何“**额外信息**”.
- “**额外信息**”: 敌手从密文中获得的信息, **减去**敌手什么都没有收到时获得的信息.

与模拟的关系

- **从规约的角度:** $\exists R \in \mathsf{PPT} : R(\mathcal{A}^{real}) = \mathcal{B}^{ideal}$
- R **需要满足的性质:**
  1. **调用** $\mathcal{A}^{real}$**:** R 需要在只与理想游戏交互的情况下为 $\mathcal{A}^{real}$**模拟**一个真实的环境.
  2. **输出结果:** R 需要输出 $\mathcal{A}^{real}$ 的输出, 作为 $\mathcal{B}^{ideal}$ 的输出.
  3. **无法区分:** $\mathcal{A}^{real}$ 在真实环境中的输出和 $\mathcal{B}^{ideal}$ 在理想环境中的输出“无法区分”.

加密算法的语义安全性

Recall: 敌手从密文中不能获得关于明文的任何“额外信息”.

定义 (语义安全性 [Goldreich, 2004, Def. 5.2.1])

一个对称加密算法 (G, E, D) 是语义安全的, 如果对于任意非均匀多项式时间敌手 $\mathcal{A}$, 存在一个非均匀多项式时间敌手 $\mathcal{B}$, 使得对于任意离散概率分布 $\{X_n\}_{n\in\mathbb{N}}$ 其中 $|X_n| \le \mathsf{poly}(n)$, 对于任意多项式大小的函数族 $f, h : \{0,1\}^* \to \{0,1\}^*$, 存在一个多项式 $p(\cdot)$ 使得对于所有 n 足够大, 有

$$\Pr\Big[\mathcal{A}(1^n, E_k(X_n), 1^{|X_n|}, h(1^n, X_n)) = f(1^n, X_n)\Big] - \Pr\Big[\mathcal{B}(1^n, 1^{|X_n|}, h(1^n, X_n)) = f(1^n, X_n)\Big] < \frac{1}{p(n)}$$

加密算法的语义安全性

Recall: 敌手从密文中不能获得关于明文的任何“额外信息”.

定义 (语义安全性 [Goldreich, 2004, Def. 5.2.1])

一个对称加密算法 (G, E, D) 是语义安全的, 如果对于任意非均匀多项式时间敌手 $\mathcal{A}$, 存在一个非均匀多项式时间敌手 $\mathcal{B}$, 使得对于任意离散概率分布 $\{X_n\}_{n\in\mathbb{N}}$ 其中 $|X_n| \le \mathsf{poly}(n)$, 对于任意多项式大小的函数族 $f, h : \{0,1\}^* \to \{0,1\}^*$, 存在一个多项式 $p(\cdot)$ 使得对于所有 n 足够大, 有

$$\Pr\Big[\mathcal{A}(1^n, E_k(X_n), 1^{|X_n|}, h(1^n, X_n)) = f(1^n, X_n)\Big] - \Pr\Big[\mathcal{B}(1^n, 1^{|X_n|}, h(1^n, X_n)) = f(1^n, X_n)\Big] < \frac{1}{p(n)}$$

注意: 这里 h 用来表示敌手已经知道的关于明文的信息.

语义安全性与模拟

$$\Pr\Big[\mathcal{A}(1^n, E_k(X_n), 1^{|X_n|}, h(1^n, X_n)) = f(1^n, X_n)\Big] - \Pr\Big[\mathcal{B}(1^n, 1^{|X_n|}, h(1^n, X_n)) = f(1^n, X_n)\Big] < \frac{1}{p(n)}$$

语义安全性与模拟

$$\Pr\Big[\mathcal{A}(1^n, E_k(X_n), 1^{|X_n|}, h(1^n, X_n)) = f(1^n, X_n)\Big] - \Pr\Big[\mathcal{B}(1^n, 1^{|X_n|}, h(1^n, X_n)) = f(1^n, X_n)\Big] < \frac{1}{p(n)}$$

模拟

$\mathcal{B} = R(\mathcal{A})$, 其中 R 满足:

- **调用 $\mathcal{A}$:** R 需要在只与理想游戏交互的情况下为 $\mathcal{A}$ 模拟一个真实的环境.

语义安全性与模拟

$$\Pr\Big[\mathcal{A}(1^n, E_k(X_n), 1^{|X_n|}, h(1^n, X_n)) = f(1^n, X_n)\Big] - \Pr\Big[\mathcal{B}(1^n, 1^{|X_n|}, h(1^n, X_n)) = f(1^n, X_n)\Big] < \frac{1}{p(n)}$$

模拟

$\mathcal{B} = R(\mathcal{A})$, 其中 R 满足:

- **调用 $\mathcal{A}$:** R 需要在只与理想游戏交互的情况下为 $\mathcal{A}$ 模拟一个真实的环境.
- **输出结果:** R 需要输出 $\mathcal{A}$ 的输出, 作为 $\mathcal{B}$ 的输出.

语义安全性与模拟

$$\Pr\Big[\mathcal{A}(1^n, E_k(X_n), 1^{|X_n|}, h(1^n, X_n)) = f(1^n, X_n)\Big] - \Pr\Big[\mathcal{B}(1^n, 1^{|X_n|}, h(1^n, X_n)) = f(1^n, X_n)\Big] < \frac{1}{p(n)}$$

模拟

$\mathcal{B} = R(\mathcal{A})$, 其中 R 满足:

- **调用 $\mathcal{A}$:** R 需要在只与理想游戏交互的情况下为 $\mathcal{A}$ 模拟一个真实的环境.
- **输出结果:** R 需要输出 $\mathcal{A}$ 的输出, 作为 $\mathcal{B}$ 的输出.
- **无法区分:** $\mathcal{A}$ 在真实环境中的输出和 $\mathcal{B}$ 在理想环境中的输出“无法区分”.

语义安全性与模拟

$$\Pr\Big[\mathcal{A}(1^n, E_k(X_n), 1^{|X_n|}, h(1^n, X_n)) = f(1^n, X_n)\Big] - \Pr\Big[\mathcal{B}(1^n, 1^{|X_n|}, h(1^n, X_n)) = f(1^n, X_n)\Big] < \frac{1}{p(n)}$$

语义安全性与模拟

$$\Pr\Big[\mathcal{A}(1^n, E_k(X_n), 1^{|X_n|}, h(1^n, X_n)) = f(1^n, X_n)\Big] - \Pr\Big[\mathcal{B}(1^n, 1^{|X_n|}, h(1^n, X_n)) = f(1^n, X_n)\Big] < \frac{1}{p(n)}$$

尝试构造 $\mathcal{B}$

$R(\mathcal{A}) = \mathcal{B}(1^n, 1^{|X_n|}, h(1^n, X_n))$:

1. 调用密钥生成算法 $k \xleftarrow{\$} G(1^n)$.

如果我们有 $E_k(0^{|X_n|}) \approx_c E_k(X_n)$, 那么我们就构造出了一个有效的 $\mathcal{B}$.

语义安全性与模拟

$$\Pr\Big[\mathcal{A}(1^n, E_k(X_n), 1^{|X_n|}, h(1^n, X_n)) = f(1^n, X_n)\Big] - \Pr\Big[\mathcal{B}(1^n, 1^{|X_n|}, h(1^n, X_n)) = f(1^n, X_n)\Big] < \frac{1}{p(n)}$$

尝试构造 $\mathcal{B}$

$R(\mathcal{A}) = \mathcal{B}(1^n, 1^{|X_n|}, h(1^n, X_n))$:

1. 调用密钥生成算法 $k \xleftarrow{\$} G(1^n)$.
2. 计算 ct $:= E_k(0^{|X_n|})$. **(其实 $\mathcal{A}$ 需要 $E_k(X_n)$)**

如果我们有 $E_k(0^{|X_n|}) \approx_c E_k(X_n)$, 那么我们就构造出了一个有效的 $\mathcal{B}$.

语义安全性与模拟

$$\Pr\Big[\mathcal{A}(1^n, E_k(X_n), 1^{|X_n|}, h(1^n, X_n)) = f(1^n, X_n)\Big] - \Pr\Big[\mathcal{B}(1^n, 1^{|X_n|}, h(1^n, X_n)) = f(1^n, X_n)\Big] < \frac{1}{p(n)}$$

尝试构造 $\mathcal{B}$

$R(\mathcal{A}) = \mathcal{B}(1^n, 1^{|X_n|}, h(1^n, X_n))$:

1. 调用密钥生成算法 $k \xleftarrow{\$} G(1^n)$.
2. 计算 $\mathsf{ct} := E_k(0^{|X_n|})$. **(其实 $\mathcal{A}$ 需要 $E_k(X_n)$)**
3. 运行 $result \xleftarrow{\$} \mathcal{A}(1^n, \mathsf{ct}, 1^{|X_n|}, h(1^n, 0^{|X_n|}))$. 并输入 $result$.

如果我们有 $E_k(0^{|X_n|}) \approx_c E_k(X_n)$, 那么我们就构造出了一个有效的 $\mathcal{B}$.

两方安全计算：对于半诚实敌手的模拟

两方安全计算

两方安全计算

- Alice 和 Bob 分别有各自的输入 x 和 y.

两方安全计算

两方安全计算

- Alice 和 Bob 分别有各自的输入 x 和 y.
- 他们想要计算一个函数 $f(x, y)$.

两方安全计算

两方安全计算

- Alice 和 Bob 分别有各自的输入 x 和 y.
- 他们想要计算一个函数 $f(x, y)$.
- 但是他们不想泄露各自的输入.

两方安全计算

两方安全计算

- Alice 和 Bob 分别有各自的输入 x 和 y.
- 他们想要计算一个函数 $f(x, y)$.
- 但是他们不想泄露各自的输入.
- **目标:** 设计一个协议, 使得 Alice 和 Bob 在不泄露各自输入的情况下, 计算出 $f(x, y)$.

两方安全计算

两方安全计算

- Alice 和 Bob 分别有各自的输入 x 和 y.
- 他们想要计算一个函数 $f(x, y)$.
- 但是他们不想泄露各自的输入.
- **目标:** 设计一个协议, 使得 Alice 和 Bob 在不泄露各自输入的情况下, 计算出 $f(x, y)$.

两方安全计算

两方安全计算

- Alice 和 Bob 分别有各自的输入 x 和 y.
- 他们想要计算一个函数 $f(x, y)$.
- 但是他们不想泄露各自的输入.
- **目标:** 设计一个协议, 使得 Alice 和 Bob 在不泄露各自输入的情况下, 计算出 $f(x, y)$.

形式化定义

- 两方安全计算: $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$, 其中 $f = (f_A, f_B)$,
- $f_A : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$, $f_B : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$.
- $f(x, y) = (f_A(x, y), f_B(x, y))$

两方安全计算

安全性质

- Alice 和 Bob 分别有各自的输入 x 和 y.
- 他们想要计算一个函数 $f(x, y)$.
- 但是他们**不想泄露各自的输入.**
- **目标:** 设计一个协议, 使得 Alice 和 Bob 在不泄露各自输入的情况下, 计算出 $f(x, y)$.

两方安全计算

安全性质

- Alice 和 Bob 分别有各自的输入 x 和 y.
- 他们想要计算一个函数 $f(x, y)$.
- 但是他们**不想泄露各自的输入.**
- **目标:** 设计一个协议, 使得 Alice 和 Bob 在不泄露各自输入的情况下, 计算出 $f(x, y)$.

基于模拟的隐私保护

- “任何用户**能够**计算的内容, 在只知道其输入输出的情况下也能算出来”

两方安全计算

安全性质

- Alice 和 Bob 分别有各自的输入 x 和 y.
- 他们想要计算一个函数 $f(x, y)$.
- 但是他们**不想泄露各自的输入.**
- **目标:** 设计一个协议, 使得 Alice 和 Bob 在不泄露各自输入的情况下, 计算出 $f(x, y)$.

基于模拟的隐私保护

- “任何用户**能够**计算的内容, 在只知道其输入输出的情况下也能算出来”
- **半诚实敌手:** 敌手会严格按照协议执行, 但是会试图从中获得额外的信息.

两方安全计算

安全性质

- Alice 和 Bob 分别有各自的输入 x 和 y.
- 他们想要计算一个函数 $f(x, y)$.
- 但是他们**不想泄露各自的输入.**
- **目标:** 设计一个协议, 使得 Alice 和 Bob 在不泄露各自输入的情况下, 计算出 $f(x, y)$.

基于模拟的隐私保护

- “任何用户**能够**计算的内容, 在只知道其输入输出的情况下也能算出来”
- **半诚实敌手:** 敌手会严格按照协议执行, 但是会试图从中获得额外的信息.
- **注意:** 这里我们假设敌手是**半诚实**的. 因此敌手会根据输入计算输出.

两方安全计算：对于半诚实敌手的模拟

一些记号与约定

- $f = (f_1, f_2)$ 为一个 PPT 函数, π 是一个计算 f 的两方计算协议.
- 令安全参数为 λ, 用户 i 的在协议 $\pi(x, y)$ 的运行过程中的视图 $(view)$ 记作 $\mathsf{view}_i^{\pi}(x, y, \lambda) = (w, r^i; \mathsf{m}_1^i, \ldots, \mathsf{m}_t^i)$ 其中
  - w为用户 i 的输入,
  - r^i为用户 i 的随机输入,
  - m_j^i为用户 i 在第 j 轮发送的消息.

两方安全计算：安全性定义

$\mathsf{view}_i^{\pi}(x, y, \lambda) = (w, r^i; \mathsf{m}_1^i, \ldots, \mathsf{m}_t^i)$ 其中

- w为用户 i 的输入，
- r^i为用户 i 的随机输入，
- m_j^i为用户 i 在第 j 轮发送的消息.

两方安全计算：安全性定义

$\mathsf{view}_i^{\pi}(x, y, \lambda) = (w, r^i; \mathsf{m}_1^i, \ldots, \mathsf{m}_t^i)$ 其中

- w为用户 i 的输入，
- r^i为用户 i 的随机输入，
- m_j^i为用户 i 在第 j 轮发送的消息.

定义 (安全两方计算)

令 $f = (f_1, f_2)$ 为一个 PPT 函数, π 是一个计算 f 的两方计算协议. 我们说 π 是**对于半诚实静态敌手的安全两方计算协议**, 如果对于每一个 $i \in \{1, 2\}$, 存在一个 PPT 算法 S_i, 使得对于任意 $x, y \in \{0, 1\}^*$, 有

$$\{S_i(x_i, f_i(x, y), \lambda), f(x, y)\}_{\lambda, x, y} \approx_c \{\mathsf{view}_i^{\pi}(x, y, \lambda), f(x, y)\}_{\lambda, x, y}$$

其中 $x_1 = x, x_2 = y$, 且 $|x| = |y|$, $\lambda \in \mathbb{N}$.

两方安全计算：安全性定义

定义 (安全两方计算)

令 $f = (f_1, f_2)$ 为一个 PPT 函数, π 是一个计算 f 的两方计算协议. 我们说 π 是**对于半诚实静态敌手的安全两方计算协议**, 如果对于每一个 $i \in \{1, 2\}$, 存在一个 PPT 算法 S_i, 使得对于任意 $x, y \in \{0, 1\}^*$, 有

$$\{S_i(x_i, f_i(x, y), \lambda), f(x, y)\}_{\lambda, x, y} \approx_c \{\mathsf{view}_i^{\pi}(x, y, \lambda), \mathsf{output}^{\pi}(x, y, \lambda)\}_{\lambda, x, y}$$

其中 $x_1 = x, x_2 = y$, 且 $|x| = |y|$, $\lambda \in \mathbb{N}$.

- **注意:** S_i 光输出 $S_i(x_i, f_i(x, y), \lambda)$ 能够模拟出用户 i 的视图是不够的.

两方安全计算：安全性定义

定义 (安全两方计算)

令 $f = (f_1, f_2)$ 为一个 PPT 函数, π 是一个计算 f 的两方计算协议. 我们说 π 是**对于半诚实静态敌手的安全两方计算协议**, 如果对于每一个 $i \in \{1, 2\}$, 存在一个 PPT 算法 S_i, 使得对于任意 $x, y \in \{0, 1\}^*$, 有

$$\{S_i(x_i, f_i(x, y), \lambda), f(x, y)\}_{\lambda,x,y} \approx_c \{\mathsf{view}_i^{\pi}(x, y, \lambda), \mathsf{output}^{\pi}(x, y, \lambda)\}_{\lambda,x,y}$$

其中 $x_1 = x, x_2 = y$, 且 $|x| = |y|$, $\lambda \in \mathbb{N}$.

- **注意:** S_i 光输出 $S_i(x_i, f_i(x, y), \lambda)$ 能够模拟出用户 i 的视图是不够的.
- 还需要保证联合分布 $\{S_i(x_i, f_i(x, y), \lambda), f(x, y)\}_{\lambda,x,y}$ 是不可区分的. **(是否存在攻击?)**

两方安全计算：安全性定义

定义 (确定性安全两方计算)

令 $f = (f_1, f_2)$ 为一个多项式时间的确定性函数, π 是一个计算 f 的两方计算协议. 我们说 π 是**对于半诚实静态敌手的安全两方计算协议**, 如果对于每一个 $i \in \{1, 2\}$, 存在一个 PPT 算法 S_i, 使得对于任意 $x, y \in \{0, 1\}^*$, 有

$$\{S_i(x_i, f_i(x, y), \lambda)\}_{\lambda,x,y} \approx_c \{\mathsf{view}_i^{\pi}(x, y, \lambda)\}_{\lambda,x,y}$$

其中 $x_1 = x, x_2 = y$, 且 $|x| = |y|$, $\lambda \in \mathbb{N}$.

- **注意:** 正确性保证 $\mathsf{output}^{\pi}(x, y, \lambda) = f(x, y)$.

样例：不经意传输

样例：半诚实敌手安全的不经意传输

目标： Alice 有两个消息 m_0, m_1, Bob 想要获得其中一个 m_b. 但是 Alice 不想知道 Bob 想要哪个消息. **工具：** 需要增强陷门置换

定义 (增强陷门置换)

一个**增强陷门置换**是一个四元组 (G, S, F, F^{-1}), 其中

- $G(1^\lambda) \to (\mathsf{pk}, \mathsf{sk})$ 输入安全参数 1^n, 输出一个陷门置换的描述符 pk 和一个陷门 sk. pk 定义了一个在 $\{0,1\}^{poly(n)}$ 上的单射 f_{pk}.
- $S(\mathsf{pk}) \to y$: 输入 pk, 输出一个几乎随机的 $y \in f_{\mathsf{pk}}(\{0,1\}^{poly(n)})$.
- $F(\mathsf{pk}, x) \to y$ 输入 (pk, x), 其中 $x \in \{0,1\}^{poly(n)}$, 输出 $y = f_{\mathsf{pk}}(x)$.
- $F^{-1}(\mathsf{sk}, y) \to x$ 输入 (sk, y), 其中 $y \in \{0,1\}^{poly(n)}$, 输出 $x = f_{\mathsf{pk}}^{-1}(y)$.

并且满足:

- 对于任意 $x \in \{0,1\}^{poly(n)}$, 有 $F^{-1}(\mathsf{sk}, F(\mathsf{pk}, x)) = x$.
- 对于任意 $x, x' \in \{0,1\}^{poly(n)}$, 有 $f_{\mathsf{pk}}(x) = f_{\mathsf{pk}}(x') \iff x = x'$.

样例：半诚实敌手安全的不经意传输

安全性： 对于任意非均匀多项式时间敌手 $\mathcal{A}$, 有

$$\Pr\Big[\mathcal{A}(1^n, \mathsf{pk}, r) = f^{-1}(S(\mathsf{pk}; r))\Big] < \mathsf{negl}(\lambda)$$

其中 $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} G(1^n)$, $r \xleftarrow{\$} \{0,1\}^{poly(n)}$.

样例：半诚实敌手安全的不经意传输

安全性： 对于任意非均匀多项式时间敌手 $\mathcal{A}$, 有

$$\Pr\Big[\mathcal{A}(1^n, \mathsf{pk}, r) = f^{-1}(S(\mathsf{pk}; r))\Big] < \mathsf{negl}(\lambda)$$

其中 $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} G(1^n)$, $r \xleftarrow{\$} \{0,1\}^{poly(n)}$.

定义 (Hardcore Bit)

存在一个多项式时间可计算的函数 $B : \{0,1\}^{poly(n)} \to \{0,1\}$, 使得对于任意非均匀多项式时间敌手 $\mathcal{A}$, 有

$$\Pr[\mathcal{A}(1^n, \mathsf{pk}, f_{\mathsf{pk}}(x)) = B(x)] < \frac{1}{2} + \mathsf{negl}(\lambda)$$

其中 $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} G(1^n)$, $x \xleftarrow{\$} \{0,1\}^{poly(n)}$.

协议构造

不经意传输 [Even et al., 1985]

- **输入:** P_1 拥有 $b_0, b_1 \in \{0,1\}$, P_2 拥有 $\sigma \in \{0,1\}$.

协议构造

不经意传输 [Even et al., 1985]

- **输入:** P_1 拥有 $b_0, b_1 \in \{0,1\}$, P_2 拥有 $\sigma \in \{0,1\}$.
- **协议:**

协议构造

不经意传输 [Even et al., 1985]

- **输入:** P_1 拥有 $b_0, b_1 \in \{0,1\}$, P_2 拥有 $\sigma \in \{0,1\}$.
- **协议:**
  1. P_1 生成 $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} G(1^n)$, 并发送 pk 给 P_2.

协议构造

不经意传输 [Even et al., 1985]

- **输入:** P_1 拥有 $b_0, b_1 \in \{0,1\}$, P_2 拥有 $\sigma \in \{0,1\}$.
- **协议:**
  1. P_1 生成 $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} G(1^n)$, 并发送 pk 给 P_2.
  2. P_2 选择 $r_0, r_1 \xleftarrow{\$} \{0,1\}^{poly(n)}$, 计算 $x_\sigma = S(\mathsf{pk}; r_\sigma), y_{1-\sigma} = S(\mathsf{pk}; r_{1-\sigma})$, 并计算 $y_\sigma = f(\mathsf{pk}, x_\sigma)$ 发送 (y_0, y_1) 给 P_1.

协议构造

不经意传输 [Even et al., 1985]

- **输入:** P_1 拥有 $b_0, b_1 \in \{0,1\}$, P_2 拥有 $\sigma \in \{0,1\}$.
- **协议:**
  1. P_1 生成 $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} G(1^n)$, 并发送 pk 给 P_2.
  2. P_2 选择 $r_0, r_1 \xleftarrow{\$} \{0,1\}^{poly(n)}$, 计算 $x_\sigma = S(\mathsf{pk}; r_\sigma), y_{1-\sigma} = S(\mathsf{pk}; r_{1-\sigma})$, 并计算 $y_\sigma = f(\mathsf{pk}, x_\sigma)$ 发送 (y_0, y_1) 给 P_1.
  3. P_1 利用 sk 计算 x_0, x_1, 并计算

协议构造

不经意传输 [Even et al., 1985]

- **输入:** P_1 拥有 $b_0, b_1 \in \{0,1\}$, P_2 拥有 $\sigma \in \{0,1\}$.
- **协议:**
  1. P_1 生成 $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} G(1^n)$, 并发送 pk 给 P_2.
  2. P_2 选择 $r_0, r_1 \xleftarrow{\$} \{0,1\}^{poly(n)}$, 计算 $x_\sigma = S(\mathsf{pk}; r_\sigma), y_{1-\sigma} = S(\mathsf{pk}; r_{1-\sigma})$, 并计算 $y_\sigma = f(\mathsf{pk}, x_\sigma)$ 发送 (y_0, y_1) 给 P_1.
  3. P_1 利用 sk 计算 x_0, x_1, 并计算
     - $\beta_0 := B(\mathsf{pk}, x_0) \oplus b_0$

协议构造

不经意传输 [Even et al., 1985]

- **输入:** P_1 拥有 $b_0, b_1 \in \{0,1\}$, P_2 拥有 $\sigma \in \{0,1\}$.
- **协议:**
  1. P_1 生成 $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} G(1^n)$, 并发送 pk 给 P_2.
  2. P_2 选择 $r_0, r_1 \xleftarrow{\$} \{0,1\}^{poly(n)}$, 计算 $x_\sigma = S(\mathsf{pk}; r_\sigma), y_{1-\sigma} = S(\mathsf{pk}; r_{1-\sigma})$, 并计算 $y_\sigma = f(\mathsf{pk}, x_\sigma)$ 发送 (y_0, y_1) 给 P_1.
  3. P_1 利用 sk 计算 x_0, x_1, 并计算
     - $\beta_0 := B(\mathsf{pk}, x_0) \oplus b_0$
     - $\beta_1 := B(\mathsf{pk}, x_1) \oplus b_1$

协议构造

不经意传输 [Even et al., 1985]

- **输入:** P_1 拥有 $b_0, b_1 \in \{0,1\}$, P_2 拥有 $\sigma \in \{0,1\}$.
- **协议:**
  1. P_1 生成 $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} G(1^n)$, 并发送 pk 给 P_2.
  2. P_2 选择 $r_0, r_1 \xleftarrow{\$} \{0,1\}^{poly(n)}$, 计算 $x_\sigma = S(\mathsf{pk}; r_\sigma), y_{1-\sigma} = S(\mathsf{pk}; r_{1-\sigma})$, 并计算 $y_\sigma = f(\mathsf{pk}, x_\sigma)$ 发送 (y_0, y_1) 给 P_1.
  3. P_1 利用 sk 计算 x_0, x_1, 并计算
     - $\beta_0 := B(\mathsf{pk}, x_0) \oplus b_0$
     - $\beta_1 := B(\mathsf{pk}, x_1) \oplus b_1$
     - P_1 发送 (β_0, β_1) 给 P_2.

协议构造

不经意传输 [Even et al., 1985]

- **输入:** P_1 拥有 $b_0, b_1 \in \{0,1\}$, P_2 拥有 $\sigma \in \{0,1\}$.
- **协议:**
  1. P_1 生成 $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} G(1^n)$, 并发送 pk 给 P_2.
  2. P_2 选择 $r_0, r_1 \xleftarrow{\$} \{0,1\}^{poly(n)}$, 计算 $x_\sigma = S(\mathsf{pk}; r_\sigma), y_{1-\sigma} = S(\mathsf{pk}; r_{1-\sigma})$, 并计算 $y_\sigma = f(\mathsf{pk}, x_\sigma)$ 发送 (y_0, y_1) 给 P_1.
  3. P_1 利用 sk 计算 x_0, x_1, 并计算
     - $\beta_0 := B(\mathsf{pk}, x_0) \oplus b_0$
     - $\beta_1 := B(\mathsf{pk}, x_1) \oplus b_1$
     - P_1 发送 (β_0, β_1) 给 P_2.
  4. P_2 计算 $b_\sigma = \beta_\sigma \oplus B(x_\sigma)$ 并输出

协议分析

定理 ([Even et al., 1985])

如果 (G, S, F, F^{-1}) 是一个增强陷门置换, 且存在 *Hardcore Bit*, 则上述协议是一个对于半诚实敌手安全的两方计算协议.

协议分析

定理 ([Even et al., 1985])

如果 (G, S, F, F^{-1}) 是一个增强陷门置换, 且存在 *Hardcore Bit*, 则上述协议是一个对于半诚实敌手安全的两方计算协议.

需要**分别证明**对于 P_1 和 P_2 的安全性. 分别需要对应构造模拟器 S_1 和 S_2 模拟 $\mathsf{view}_1^{\pi}((b_0, b_1), \sigma) = ((b_0, b_1), r; y_0, y_1)$ 和 $\mathsf{view}_2^{\pi}((b_0, b_1), \sigma) = (\sigma, r; \mathsf{pk}, (\beta_0, \beta_1))$.

协议分析

定理 ([Even et al., 1985])

如果 (G, S, F, F^{-1}) 是一个增强陷门置换, 且存在 *Hardcore Bit*, 则上述协议是一个对于半诚实敌手安全的两方计算协议.

需要**分别证明**对于 P_1 和 P_2 的安全性. 分别需要对应构造模拟器 S_1 和 S_2 模拟 $\mathsf{view}_1^{\pi}((b_0, b_1), \sigma) = ((b_0, b_1), r; y_0, y_1)$ 和 $\mathsf{view}_2^{\pi}((b_0, b_1), \sigma) = (\sigma, r; \mathsf{pk}, (\beta_0, \beta_1))$.

证明.

首先证明 P_1 被腐化时的安全性. 构造模拟器 $S_1(b_0, b_1, 1^{\lambda})$:

1. 挑选随机带 r, 计算 $(\mathsf{pk}, sk) \xleftarrow{\$} G(1^{\lambda}; r)$.

□

协议分析

定理 ([Even et al., 1985])

如果 (G, S, F, F^{-1}) 是一个增强陷门置换, 且存在 *Hardcore Bit*, 则上述协议是一个对于半诚实敌手安全的两方计算协议.

需要**分别证明**对于 P_1 和 P_2 的安全性. 分别需要对应构造模拟器 S_1 和 S_2 模拟 $\mathsf{view}_1^{\pi}((b_0, b_1), \sigma) = ((b_0, b_1), r; y_0, y_1)$ 和 $\mathsf{view}_2^{\pi}((b_0, b_1), \sigma) = (\sigma, r; \mathsf{pk}, (\beta_0, \beta_1))$.

证明.

首先证明 P_1 被腐化时的安全性. 构造模拟器 $S_1(b_0, b_1, 1^{\lambda})$:

1. 挑选随机带 r, 计算 $(\mathsf{pk}, sk) \xleftarrow{\$} G(1^{\lambda}; r)$.
2. 选择 $r_0, r_1 \xleftarrow{\$} \{0,1\}^{poly(n)}$, 计算 $y_0 = S(\mathsf{pk}; r_0), y_1 = S(\mathsf{pk}; r_1)$.

□

协议分析

定理 ([Even et al., 1985])

如果 (G, S, F, F^{-1}) 是一个增强陷门置换, 且存在 *Hardcore Bit*, 则上述协议是一个对于半诚实敌手安全的两方计算协议.

需要**分别证明**对于 P_1 和 P_2 的安全性. 分别需要对应构造模拟器 S_1 和 S_2 模拟 $\mathsf{view}_1^{\pi}((b_0, b_1), \sigma) = ((b_0, b_1), r; y_0, y_1)$ 和 $\mathsf{view}_2^{\pi}((b_0, b_1), \sigma) = (\sigma, r; \mathsf{pk}, (\beta_0, \beta_1))$.

证明.

首先证明 P_1 被腐化时的安全性. 构造模拟器 $S_1(b_0, b_1, 1^{\lambda})$:

1. 挑选随机带 r, 计算 $(\mathsf{pk}, sk) \xleftarrow{\$} G(1^{\lambda}; r)$.
2. 选择 $r_0, r_1 \xleftarrow{\$} \{0,1\}^{poly(n)}$, 计算 $y_0 = S(\mathsf{pk}; r_0), y_1 = S(\mathsf{pk}; r_1)$.
3. $\mathsf{view}_1^{\pi} = ((b_0, b_1), r; y_0, y_1)$.

□

协议分析

Part 2.

P_2 被腐化的安全性. 构造 $S_2(\sigma, b_\sigma)$: $\mathsf{view}_2^{\pi}((b_0, b_1), \sigma) = (\sigma, r; \mathsf{pk}, (\beta_0, \beta_1))$

1. 接收输入 σ. 生成随机带 $r = (r_0, r_1)$.

□

协议分析

Part 2.

P_2 被腐化的安全性. 构造 $S_2(\sigma, b_\sigma)$: $\mathsf{view}_2^{\pi}((b_0, b_1), \sigma) = (\sigma, r; \mathsf{pk}, (\beta_0, \beta_1))$

1. 接收输入 σ. 生成随机带 $r = (r_0, r_1)$.
2. 计算 $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} G(1^\lambda)$

□

协议分析

Part 2.

P_2 被腐化的安全性. 构造 $S_2(\sigma, b_\sigma)$: $\mathsf{view}_2^{\pi}((b_0, b_1), \sigma) = (\sigma, r; \mathsf{pk}, (\beta_0, \beta_1))$

1. 接收输入 σ. 生成随机带 $r = (r_0, r_1)$.
2. 计算 $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} G(1^\lambda)$
3. 生成 $x_\sigma = S(\mathsf{pk}; r_\sigma)$, $y_{1-\sigma} = S(\mathsf{pk}; r_{1-\sigma})$. 计算 $x_{1-\sigma} = F^{-1}(\mathsf{sk}, y_{1-\sigma})$.

□

协议分析

Part 2.

P_2 被腐化的安全性. 构造 $S_2(\sigma, b_\sigma)$: $\mathsf{view}_2^\pi((b_0, b_1), \sigma) = (\sigma, r; \mathsf{pk}, (\beta_0, \beta_1))$

1. 接收输入 σ. 生成随机带 $r = (r_0, r_1)$.
2. 计算 $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} G(1^\lambda)$
3. 生成 $x_\sigma = S(\mathsf{pk}; r_\sigma)$, $y_{1-\sigma} = S(\mathsf{pk}; r_{1-\sigma})$. 计算 $x_{1-\sigma} = F^{-1}(\mathsf{sk}, y_{1-\sigma})$.
4. 计算 $\beta_\sigma = b_\sigma \oplus B(x_\sigma)$, $\beta_{1-\sigma} = B(x_{1-\sigma})$.

□

协议分析

Part 2.

P_2 被腐化的安全性. 构造 $S_2(\sigma, b_\sigma)$: $\mathsf{view}_2^\pi((b_0, b_1), \sigma) = (\sigma, r; \mathsf{pk}, (\beta_0, \beta_1))$

1. 接收输入 σ. 生成随机带 $r = (r_0, r_1)$.
2. 计算 $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} G(1^\lambda)$
3. 生成 $x_\sigma = S(\mathsf{pk}; r_\sigma)$, $y_{1-\sigma} = S(\mathsf{pk}; r_{1-\sigma})$. 计算 $x_{1-\sigma} = F^{-1}(\mathsf{sk}, y_{1-\sigma})$.
4. 计算 $\beta_\sigma = b_\sigma \oplus B(x_\sigma)$, $\beta_{1-\sigma} = B(x_{1-\sigma})$.
5. 输出 $(\sigma, r; \mathsf{pk}, (\beta_0, \beta_1))$

□

零知识证明 - HVZK

Honest Verifier Zero Knowledge (HVZK)

- 证明者 P 想要向验证者 V 证明一个命题 $x \in L$, 但是不想泄露任何关于 x 的信息.

零知识证明 - HVZK

Honest Verifier Zero Knowledge (HVZK)

- 证明者 P 想要向验证者 V 证明一个命题 $x \in L$, 但是不想泄露任何关于 x 的信息.
- P 和 V 通过交互式协议进行通信, 最终 V 输出一个比特, 表示是否接受证明.

零知识证明 - HVZK

Honest Verifier Zero Knowledge (HVZK)

- 证明者 P 想要向验证者 V 证明一个命题 $x \in L$, 但是不想泄露任何关于 x 的信息.
- P 和 V 通过交互式协议进行通信, 最终 V 输出一个比特, 表示是否接受证明.
- **目标:** 设计一个协议, 使得即使有半诚实敌手, 也能保证协议的正确性和隐私性.

恶意敌手

恶意敌手

恶意敌手

- 恶意敌手可以做任何事情, 包括偏离协议.

恶意敌手

恶意敌手

- 恶意敌手可以做任何事情, 包括偏离协议.
- 例如: 发送错误的消息, 不发送消息, 重复发送消息等.

恶意敌手

恶意敌手

- 恶意敌手可以做任何事情, 包括偏离协议.
- 例如: 发送错误的消息, 不发送消息, 重复发送消息等.
- **目标:** 设计一个协议, 使得即使有恶意敌手, 也能保证协议的正确性和隐私性.

恶意敌手

恶意敌手

- 恶意敌手可以做任何事情, 包括偏离协议.
- 例如: 发送错误的消息, 不发送消息, 重复发送消息等.
- **目标:** 设计一个协议, 使得即使有恶意敌手, 也能保证协议的正确性和隐私性.

恶意敌手

恶意敌手

- 恶意敌手可以做任何事情, 包括偏离协议.
- 例如: 发送错误的消息, 不发送消息, 重复发送消息等.
- **目标:** 设计一个协议, 使得即使有恶意敌手, 也能保证协议的正确性和隐私性.

- **关键:** 恶意敌手的行为是不可预测的.

恶意敌手

恶意敌手

- 恶意敌手可以做任何事情, 包括偏离协议.
- 例如: 发送错误的消息, 不发送消息, 重复发送消息等.
- **目标:** 设计一个协议, 使得即使有恶意敌手, 也能保证协议的正确性和隐私性.

- **关键:** 恶意敌手的行为是不可预测的.
- 因此需要模拟器能够模拟出恶意敌手的所有可能行为.

恶意敌手

恶意敌手

- 恶意敌手可以做任何事情, 包括偏离协议.
- 例如: 发送错误的消息, 不发送消息, 重复发送消息等.
- **目标:** 设计一个协议, 使得即使有恶意敌手, 也能保证协议的正确性和隐私性.

- **关键:** 恶意敌手的行为是不可预测的.
- 因此需要模拟器能够模拟出恶意敌手的所有可能行为.
- 尤其是敌手可能忽略所有给定的输入, 只输出一个任意的输出. 模拟器无法模拟出来这种行为.

恶意敌手的经典例子：零知识证明

零知识证明

- 证明者 P 想要向验证者 V 证明一个命题 $x \in L$, 但是不想泄露任何关于 x 的信息.

恶意敌手的经典例子：零知识证明

零知识证明

- 证明者 P 想要向验证者 V 证明一个命题 $x \in L$, 但是不想泄露任何关于 x 的信息.
- P 和 V 通过交互式协议进行通信, 最终 V 输出一个比特, 表示是否接受证明.

恶意敌手的经典例子：零知识证明

零知识证明

- 证明者 P 想要向验证者 V 证明一个命题 $x \in L$, 但是不想泄露任何关于 x 的信息.
- P 和 V 通过交互式协议进行通信, 最终 V 输出一个比特, 表示是否接受证明.
- **目标:** 设计一个协议, 使得即使有恶意敌手, 也能保证协议的正确性和隐私性.

恶意敌手的经典例子：零知识证明

零知识证明

- 证明者 P 想要向验证者 V 证明一个命题 $x \in L$, 但是不想泄露任何关于 x 的信息.
- P 和 V 通过交互式协议进行通信, 最终 V 输出一个比特, 表示是否接受证明.
- **目标:** 设计一个协议, 使得即使有恶意敌手, 也能保证协议的正确性和隐私性.

恶意敌手的经典例子：零知识证明

零知识证明

- 证明者 P 想要向验证者 V 证明一个命题 $x \in L$, 但是不想泄露任何关于 x 的信息.
- P 和 V 通过交互式协议进行通信, 最终 V 输出一个比特, 表示是否接受证明.
- **目标:** 设计一个协议, 使得即使有恶意敌手, 也能保证协议的正确性和隐私性.

- 如果 V 是恶意的, 它想获得关于秘密的消息 $\rightarrow$ 零知识性.

恶意敌手的经典例子：零知识证明

零知识证明

- 证明者 P 想要向验证者 V 证明一个命题 $x \in L$, 但是不想泄露任何关于 x 的信息.
- P 和 V 通过交互式协议进行通信, 最终 V 输出一个比特, 表示是否接受证明.
- **目标:** 设计一个协议, 使得即使有恶意敌手, 也能保证协议的正确性和隐私性.

- 如果 V 是恶意的, 它想获得关于秘密的消息 $\rightarrow$ 零知识性.
- 如果 P 是恶意的, 它想让 V 接受一个错误的命题 $\rightarrow$ 完备性.

恶意敌手的建模

理想功能

- 理想功能是一个可信的第三方, 它能够接收双方的输入, 并计算出正确的输出.

恶意敌手的建模

理想功能

- 理想功能是一个可信的第三方, 它能够接收双方的输入, 并计算出正确的输出.
- 理想功能能够检测恶意行为, 并采取相应的措施.

恶意敌手的建模

理想功能

- 理想功能是一个可信的第三方, 它能够接收双方的输入, 并计算出正确的输出.
- 理想功能能够检测恶意行为, 并采取相应的措施.
- **例如:** 如果一方发送了错误的消息, 理想功能可以拒绝该消息, 并通知另一方.

恶意敌手：理想功能

对于一个函数 $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$, 理想功能 F 的定义如下:

定义 (理想功能)

输入: F 接收两个输入 (x, y), 其中 x 来自用户 A, y 来自用户 B.

输入给 F: 诚实用户 P_j 将其输入发送给可信方, 敌手可以发出任意值, 或者终止信号.

可信方中止: 如果可信方收到终止信号, 它将中止计算, 并向双方发送中止通知. 否则继续.

可信输出: 可信方根据输入 (可能是腐化的) 计算出正确的输出并发送给双方.

中间中止: 敌手可发送中止信号, 使得可信方在计算输出前中止计算.

输出: 诚实用户输出收到的值, 腐化方不发出任何值, 敌手输出" 可计算的任意值.

恶意敌手建模

定义 (安全两方计算 - 理想功能)

令 $f = (f_1, f_2)$ 为一个 PPT 函数, i 是被腐化的用户, z 是给敌手 $\mathcal{A}$ 的辅助信息. 理想功能 $\text{IDEAL}_{f,\mathcal{A}(z),i}(x, y, n)$ 定义为诚实用户与敌手的共同输出.

真实执行过程

定义 (安全多方计算 - 真实执行)

令 $f = (f_1, f_2)$ 为一个 PPT 函数, π 是一个计算 f 的两方计算协议, i 是被腐化的用户, z 是给敌手 $\mathcal{A}$ 的辅助信息. 真实执行过程 $\text{REAL}_{\pi,\mathcal{A}(z),i}(x, y, n)$ 定义为诚实用户与敌手的共同输出.

通信模型: π 是按照轮来执行, 每一轮只有一个用户发送消息. 且另一个用户会在轮结束前收到消息.

恶意敌手

定义

令 f 为一个两方理想功能, π 是一个计算 f 的两方计算协议. 我们说 π 是**对于恶意静态敌手的安全两方计算协议**, 如果对于任意非均匀多项式时间敌手 $\mathcal{A}$, 存在一个非均匀多项式时间针对理想功能的敌手 $\mathcal{S}$, 使得对于任意 $i \in \{1,2\}$, $|x| = |y|$, 有

$$\left\{\textsc{Real}_{\pi,\mathcal{A}(z),i}(x,y,\lambda)\right\}_{\lambda,x,y,z} \equiv_c \left\{\textsc{Ideal}_{f,\mathcal{S}(z),i}(x,y,\lambda)\right\}_{\lambda,x,y,z}.$$

安全抛硬币

如何安全抛硬币?

[Blum, 1982] 给出了一个简单的协议. 计算函数 $f(\lambda, \lambda) = (U_1, U_1)$ where $U_1 \in \{0,1\}$ 是一个均匀分布的随机变量.

First Try

- P_1 选择 $b_1 \xleftarrow{\$} \{0,1\}$, 发送给 P_2.

如何安全抛硬币?

[Blum, 1982] 给出了一个简单的协议. 计算函数 $f(\lambda, \lambda) = (U_1, U_1)$ where $U_1 \in \{0,1\}$ 是一个均匀分布的随机变量.

First Try

- P_1 选择 $b_1 \xleftarrow{\$} \{0,1\}$, 发送给 P_2.
- P_2 选择 $b_2 \xleftarrow{\$} \{0,1\}$, 发送给 P_1.

如何安全抛硬币?

[Blum, 1982] 给出了一个简单的协议. 计算函数 $f(\lambda, \lambda) = (U_1, U_1)$ where $U_1 \in \{0,1\}$ 是一个均匀分布的随机变量.

First Try

- P_1 选择 $b_1 \xleftarrow{\$} \{0,1\}$, 发送给 P_2.
- P_2 选择 $b_2 \xleftarrow{\$} \{0,1\}$, 发送给 P_1.
- 输出 $b_1 \oplus b_2$.

如何安全抛硬币?

[Blum, 1982] 给出了一个简单的协议. 计算函数 $f(\lambda, \lambda) = (U_1, U_1)$ where $U_1 \in \{0,1\}$ 是一个均匀分布的随机变量.

First Try

- P_1 选择 $b_1 \xleftarrow{\$} \{0,1\}$, 发送给 P_2.
- P_2 选择 $b_2 \xleftarrow{\$} \{0,1\}$, 发送给 P_1.
- 输出 $b_1 \oplus b_2$.

如何安全抛硬币?

[Blum, 1982] 给出了一个简单的协议. 计算函数 $f(\lambda, \lambda) = (U_1, U_1)$ where $U_1 \in \{0,1\}$ 是一个均匀分布的随机变量.

First Try

- P_1 选择 $b_1 \xleftarrow{\$} \{0,1\}$, 发送给 P_2.
- P_2 选择 $b_2 \xleftarrow{\$} \{0,1\}$, 发送给 P_1.
- 输出 $b_1 \oplus b_2$.

- P_2 可以选择 $b_2 = b_1$ 使得输出为 0.

如何安全抛硬币?

[Blum, 1982] 给出了一个简单的协议. 计算函数 $f(\lambda, \lambda) = (U_1, U_1)$ where $U_1 \in \{0,1\}$ 是一个均匀分布的随机变量.

First Try

- P_1 选择 $b_1 \xleftarrow{\$} \{0,1\}$, 发送给 P_2.
- P_2 选择 $b_2 \xleftarrow{\$} \{0,1\}$, 发送给 P_1.
- 输出 $b_1 \oplus b_2$.

- P_2 可以选择 $b_2 = b_1$ 使得输出为 0.
- P_1 可以选择 $b_1 = 1 - b_2$ 使得输出为 1.

如何安全抛硬币?

[Blum, 1982] 给出了一个简单的协议. 计算函数 $f(\lambda, \lambda) = (U_1, U_1)$ where $U_1 \in \{0,1\}$ 是一个均匀分布的随机变量.

First Try

- P_1 选择 $b_1 \xleftarrow{\$} \{0,1\}$, 发送给 P_2.
- P_2 选择 $b_2 \xleftarrow{\$} \{0,1\}$, 发送给 P_1.
- 输出 $b_1 \oplus b_2$.

- P_2 可以选择 $b_2 = b_1$ 使得输出为 0.
- P_1 可以选择 $b_1 = 1 - b_2$ 使得输出为 1.
- **不安全!**

安全抛硬币 - 承诺

定义 (承诺方案 [Goldreich, 2004, Def. 3.2.1])

一个**承诺方案**是一个二元组 (G, Com), 其中

- $G(1^n) \to \mathsf{ck}$: 输入安全参数 1^n, 输出一个承诺密钥 ck.
- $\mathsf{Com}(\mathsf{ck}, m; r) \to c$: 输入承诺密钥 ck, 消息 $m \in \{0,1\}$, 随机数 $r \in \{0,1\}^{poly(n)}$, 输出一个承诺 c.

安全抛硬币 - 承诺性质

定义

安全性质

- **绑定性** 对于任意 PPT 敌手 $\mathcal{A}$, 有

$$\Pr[\mathsf{Com}(\mathsf{ck}, m_0; r_0) = \mathsf{Com}(\mathsf{ck}, m_1; r_1)] < \mathsf{negl}(\lambda)$$

  其中 $(\mathsf{ck}) \xleftarrow{\$} G(1^\lambda)$, $(m_0, m_1) \xleftarrow{\$} \mathcal{A}(1^\lambda, \mathsf{ck})$, $m_0 \neq m_1$.

- **隐藏性** 对于任意 PPT 敌手 $\mathcal{A}$, 有

$$\{\mathsf{Com}(\mathsf{ck}, 0; r)\}_{\lambda, r} \approx_c \{\mathsf{Com}(\mathsf{ck}, 1; r)\}_{\lambda, r},$$

  其中 $(\mathsf{ck}) \xleftarrow{\$} G(1^\lambda)$, $r \leftarrow\!\!\shortmid \mathsf{U}\left(\{0,1\}^{\mathsf{poly}(\lambda)}\right)$.

安全抛硬币

Second Try

- P_1 选择 $b_1 \xleftarrow{\$} \{0,1\}$, 计算 $c = \mathsf{Com}(\mathsf{ck}, b_1; r)$, 发送 c 给 P_2.

安全抛硬币

Second Try

- P_1 选择 $b_1 \xleftarrow{\$} \{0,1\}$, 计算 $c = \mathsf{Com}(\mathsf{ck}, b_1; r)$, 发送 c 给 P_2.
- P_2 选择 $b_2 \xleftarrow{\$} \{0,1\}$, 发送 b_2 给 P_1.

安全抛硬币

Second Try

- P_1 选择 $b_1 \xleftarrow{\$} \{0,1\}$, 计算 $c = \mathsf{Com}(\mathsf{ck}, b_1; r)$, 发送 c 给 P_2.
- P_2 选择 $b_2 \xleftarrow{\$} \{0,1\}$, 发送 b_2 给 P_1.
- P_1 发送 (b_1, r) 给 P_2, P_2 验证 $c = \mathsf{Com}(\mathsf{ck}, b_1; r)$.

安全抛硬币

Second Try

- P_1 选择 $b_1 \xleftarrow{\$} \{0,1\}$, 计算 $c = \mathsf{Com}(\mathsf{ck}, b_1; r)$, 发送 c 给 P_2.
- P_2 选择 $b_2 \xleftarrow{\$} \{0,1\}$, 发送 b_2 给 P_1.
- P_1 发送 (b_1, r) 给 P_2, P_2 验证 $c = \mathsf{Com}(\mathsf{ck}, b_1; r)$.
- 输出 $b_1 \oplus b_2$.

安全抛硬币

Second Try

- P_1 选择 $b_1 \xleftarrow{\$} \{0,1\}$, 计算 $c = \mathsf{Com}(\mathsf{ck}, b_1; r)$, 发送 c 给 P_2.
- P_2 选择 $b_2 \xleftarrow{\$} \{0,1\}$, 发送 b_2 给 P_1.
- P_1 发送 (b_1, r) 给 P_2, P_2 验证 $c = \mathsf{Com}(\mathsf{ck}, b_1; r)$.
- 输出 $b_1 \oplus b_2$.

安全抛硬币

Second Try

- P_1 选择 $b_1 \xleftarrow{\$} \{0,1\}$, 计算 $c = \mathsf{Com}(\mathsf{ck}, b_1; r)$, 发送 c 给 P_2.
- P_2 选择 $b_2 \xleftarrow{\$} \{0,1\}$, 发送 b_2 给 P_1.
- P_1 发送 (b_1, r) 给 P_2, P_2 验证 $c = \mathsf{Com}(\mathsf{ck}, b_1; r)$.
- 输出 $b_1 \oplus b_2$.

- P_1 不能改变 b_1, 因为它已经被提交了.

安全抛硬币

Second Try

- P_1 选择 $b_1 \xleftarrow{\$} \{0,1\}$, 计算 $c = \mathsf{Com}(\mathsf{ck}, b_1; r)$, 发送 c 给 P_2.
- P_2 选择 $b_2 \xleftarrow{\$} \{0,1\}$, 发送 b_2 给 P_1.
- P_1 发送 (b_1, r) 给 P_2, P_2 验证 $c = \mathsf{Com}(\mathsf{ck}, b_1; r)$.
- 输出 $b_1 \oplus b_2$.

- P_1 不能改变 b_1, 因为它已经被提交了.
- P_2 不能改变 b_2, 因为它是随机选择的.

安全抛硬币

Second Try

- P_1 选择 $b_1 \xleftarrow{\$} \{0,1\}$, 计算 $c = \mathsf{Com}(\mathsf{ck}, b_1; r)$, 发送 c 给 P_2.
- P_2 选择 $b_2 \xleftarrow{\$} \{0,1\}$, 发送 b_2 给 P_1.
- P_1 发送 (b_1, r) 给 P_2, P_2 验证 $c = \mathsf{Com}(\mathsf{ck}, b_1; r)$.
- 输出 $b_1 \oplus b_2$.

- P_1 不能改变 b_1, 因为它已经被提交了.
- P_2 不能改变 b_2, 因为它是随机选择的.
- 输出是均匀分布的, 且双方都无法控制输出.

安全抛硬币 - 半诚实敌手证明

挑战

- 构造模拟器 S_1 模拟 $\mathsf{view}_1^{\pi}(b_1, b_2) = (\mathsf{ck}, r; b_2, b)$.

安全抛硬币 - 半诚实敌手证明

挑战

- 构造模拟器 S_1 模拟 $\mathsf{view}_1^{\pi}(b_1, b_2) = (\mathsf{ck}, r; b_2, b)$.
- 构造模拟器 S_2 模拟 $\mathsf{view}_2^{\pi}(b_1, b_2) = (\mathsf{ck}, r; c, (b_1, r_1))$.

安全抛硬币 - 半诚实敌手证明

挑战

- 构造模拟器 S_1 模拟 $\mathsf{view}_1^{\pi}(b_1, b_2) = (\mathsf{ck}, r; b_2, b)$.
- 构造模拟器 S_2 模拟 $\mathsf{view}_2^{\pi}(b_1, b_2) = (\mathsf{ck}, r; c, (b_1, r_1))$.

- S_1 比较简单, 直接选择随机 b_2 即可.

安全抛硬币 - 半诚实敌手证明

挑战

- 构造模拟器 S_1 模拟 $\mathsf{view}_1^{\pi}(b_1, b_2) = (\mathsf{ck}, r; b_2, b)$.
- 构造模拟器 S_2 模拟 $\mathsf{view}_2^{\pi}(b_1, b_2) = (\mathsf{ck}, r; c, (b_1, r_1))$.

- S_1 比较简单, 直接选择随机 b_2 即可.
- S_2 需要利用承诺的隐藏性来模拟 c.

安全抛硬币 - 半诚实敌手证明

挑战

- 构造模拟器 S_1 模拟 $\mathsf{view}_1^{\pi}(b_1, b_2) = (\mathsf{ck}, r; b_2, b)$.

安全抛硬币 - 半诚实敌手证明

挑战

- 构造模拟器 S_1 模拟 $\mathsf{view}_1^{\pi}(b_1, b_2) = (\mathsf{ck}, r; b_2, b)$.
- 构造模拟器 S_2 模拟 $\mathsf{view}_2^{\pi}(b_1, b_2) = (\mathsf{ck}, r; c, (b_1, r_1))$.

安全抛硬币 - 半诚实敌手证明

挑战

- 构造模拟器 S_1 模拟 $\mathsf{view}_1^{\pi}(b_1, b_2) = (\mathsf{ck}, r; b_2, b)$.
- 构造模拟器 S_2 模拟 $\mathsf{view}_2^{\pi}(b_1, b_2) = (\mathsf{ck}, r; c, (b_1, r_1))$.

安全抛硬币 - 半诚实敌手证明

挑战

- 构造模拟器 S_1 模拟 $\mathsf{view}_1^{\pi}(b_1, b_2) = (\mathsf{ck}, r; b_2, b)$.
- 构造模拟器 S_2 模拟 $\mathsf{view}_2^{\pi}(b_1, b_2) = (\mathsf{ck}, r; c, (b_1, r_1))$.

证明.

□

安全抛硬币 - 恶意敌手证明

定理 ([Blum, 1982])

如果存在一个安全的承诺方案, 则上述协议是一个对于恶意静态敌手安全的两方计算协议.

安全抛硬币 - 恶意敌手证明

定理 ([Blum, 1982])

如果存在一个安全的承诺方案, 则上述协议是一个对于恶意静态敌手安全的两方计算协议.

挑战

恶意敌手可以偏离协议, 所以上述 Simulator**不再适用.** 构造恶意敌手环境中的模拟器:

安全抛硬币 - 恶意敌手证明

Part 1.

只考虑确定性敌手 $\mathcal{A}$, P_2 被腐化的安全性. 构造模拟器 $S(\mathcal{A})$:

1. 发送 λ 给信任方计算 f_{ct} 收回一个比特 b.

安全抛硬币 - 恶意敌手证明

Part 1.

只考虑确定性敌手 $\mathcal{A}$, P_2 被腐化的安全性. 构造模拟器 $S(\mathcal{A})$:

1. 发送 λ 给信任方计算 f_{ct} 收回一个比特 b.
2. 初始化 $i = 1$

安全抛硬币 - 恶意敌手证明

Part 1.

只考虑确定性敌手 $\mathcal{A}$, P_2 被腐化的安全性. 构造模拟器 $S(\mathcal{A})$:

1. 发送 λ 给信任方计算 f_{ct} 收回一个比特 b.
2. 初始化 $i = 1$
3. 调用 $\mathcal{A}$ 选择随机数 b_1, r_1, 计算 $c = \mathsf{Com}(\mathsf{ck}, b_1; r_1)$, 内部发送 c 给 $\mathcal{A}$.

安全抛硬币 - 恶意敌手证明

Part 1.

只考虑确定性敌手 $\mathcal{A}$, P_2 被腐化的安全性. 构造模拟器 $S(\mathcal{A})$:

1. 发送 λ 给信任方计算 f_{ct} 收回一个比特 b.
2. 初始化 $i = 1$
3. 调用 $\mathcal{A}$ 选择随机数 b_1, r_1, 计算 $c = \mathsf{Com}(\mathsf{ck}, b_1; r_1)$, 内部发送 c 给 $\mathcal{A}$.
4. 如果 $\mathcal{A}$ 回复 $b_2 = b \oplus b_1$, 则发送 (b_1, r_1) 给 $\mathcal{A}$, 并输出 $\mathcal{A}$ 的输出.

安全抛硬币 - 恶意敌手证明

Part 1.

只考虑确定性敌手 $\mathcal{A}$, P_2 被腐化的安全性. 构造模拟器 $S(\mathcal{A})$:

1. 发送 λ 给信任方计算 f_{ct} 收回一个比特 b.
2. 初始化 $i = 1$
3. 调用 $\mathcal{A}$ 选择随机数 b_1, r_1, 计算 $c = \mathsf{Com}(\mathsf{ck}, b_1; r_1)$, 内部发送 c 给 $\mathcal{A}$.
4. 如果 $\mathcal{A}$ 回复 $b_2 = b \oplus b_1$, 则发送 (b_1, r_1) 给 $\mathcal{A}$, 并输出 $\mathcal{A}$ 的输出.
5. 如果 $\mathcal{A}$ 回复 $b_2 \neq b \oplus b_1$, 则 $i := i + 1$, 返回第 3 步.

安全抛硬币 - 恶意敌手证明

Part 1.

只考虑确定性敌手 $\mathcal{A}$, P_2 被腐化的安全性. 构造模拟器 $S(\mathcal{A})$:

1. 发送 λ 给信任方计算 f_{ct} 收回一个比特 b.
2. 初始化 $i = 1$
3. 调用 $\mathcal{A}$ 选择随机数 b_1, r_1, 计算 $c = \mathsf{Com}(\mathsf{ck}, b_1; r_1)$, 内部发送 c 给 $\mathcal{A}$.
4. 如果 $\mathcal{A}$ 回复 $b_2 = b \oplus b_1$, 则发送 (b_1, r_1) 给 $\mathcal{A}$, 并输出 $\mathcal{A}$ 的输出.
5. 如果 $\mathcal{A}$ 回复 $b_2 \neq b \oplus b_1$, 则 $i := i + 1$, 返回第 3 步.
6. 如果 $i > \lambda$, 则输出失败.

安全抛硬币 - 恶意敌手证明

Part 1.

只考虑确定性敌手 $\mathcal{A}$, P_2 被腐化的安全性. 构造模拟器 $S(\mathcal{A})$:

1. 发送 λ 给信任方计算 f_{ct} 收回一个比特 b.
2. 初始化 $i = 1$
3. 调用 $\mathcal{A}$ 选择随机数 b_1, r_1, 计算 $c = \mathsf{Com}(\mathsf{ck}, b_1; r_1)$, 内部发送 c 给 $\mathcal{A}$.
4. 如果 $\mathcal{A}$ 回复 $b_2 = b \oplus b_1$, 则发送 (b_1, r_1) 给 $\mathcal{A}$, 并输出 $\mathcal{A}$ 的输出.
5. 如果 $\mathcal{A}$ 回复 $b_2 \neq b \oplus b_1$, 则 $i := i + 1$, 返回第 3 步.
6. 如果 $i > \lambda$, 则输出失败.

安全抛硬币 - 恶意敌手证明

Part 1.

只考虑确定性敌手 $\mathcal{A}$, P_2 被腐化的安全性. 构造模拟器 $S(\mathcal{A})$:

1. 发送 λ 给信任方计算 f_{ct} 收回一个比特 b.
2. 初始化 $i = 1$
3. 调用 $\mathcal{A}$ 选择随机数 b_1, r_1, 计算 $c = \mathsf{Com}(\mathsf{ck}, b_1; r_1)$, 内部发送 c 给 $\mathcal{A}$.
4. 如果 $\mathcal{A}$ 回复 $b_2 = b \oplus b_1$, 则发送 (b_1, r_1) 给 $\mathcal{A}$, 并输出 $\mathcal{A}$ 的输出.
5. 如果 $\mathcal{A}$ 回复 $b_2 \neq b \oplus b_1$, 则 $i := i + 1$, 返回第 3 步.
6. 如果 $i > \lambda$, 则输出失败.

为什么模拟器能够成功证明?

- 模拟器 $S(\mathcal{A})$ 失败的概率可忽略. (承诺的隐藏性)

安全抛硬币 - 恶意敌手证明

Part 1.

只考虑确定性敌手 $\mathcal{A}$, P_2 被腐化的安全性. 构造模拟器 $S(\mathcal{A})$:

1. 发送 λ 给信任方计算 f_{ct} 收回一个比特 b.
2. 初始化 $i = 1$
3. 调用 $\mathcal{A}$ 选择随机数 b_1, r_1, 计算 $c = \mathsf{Com}(\mathsf{ck}, b_1; r_1)$, 内部发送 c 给 $\mathcal{A}$.
4. 如果 $\mathcal{A}$ 回复 $b_2 = b \oplus b_1$, 则发送 (b_1, r_1) 给 $\mathcal{A}$, 并输出 $\mathcal{A}$ 的输出.
5. 如果 $\mathcal{A}$ 回复 $b_2 \neq b \oplus b_1$, 则 $i := i + 1$, 返回第 3 步.
6. 如果 $i > \lambda$, 则输出失败.

为什么模拟器能够成功证明?

- 模拟器 $S(\mathcal{A})$ 失败的概率可忽略. (承诺的隐藏性)
- 在不失败的情况下, 理想世界与真实世界不可区分

安全抛硬币 - 恶意敌手证明

Part 2.

只考虑确定性敌手 $\mathcal{A}$, P_1 被腐化的安全性. 构造模拟器 $S(\mathcal{A})$:

1. 发送 λ 给信任方计算 f_{ct} 收回一个比特 b.

安全抛硬币 - 恶意敌手证明

Part 2.

只考虑确定性敌手 $\mathcal{A}$, P_1 被腐化的安全性. 构造模拟器 $S(\mathcal{A})$:

1. 发送 λ 给信任方计算 f_{ct} 收回一个比特 b.
2. 调用 $\mathcal{A}$ 从内部收到 $\mathcal{A}$ 发送给 P_1 的 c.

安全抛硬币 - 恶意敌手证明

Part 2.

只考虑确定性敌手 $\mathcal{A}$, P_1 被腐化的安全性. 构造模拟器 $S(\mathcal{A})$:

1. 发送 λ 给信任方计算 f_{ct} 收回一个比特 b.
2. 调用 $\mathcal{A}$ 从内部收到 $\mathcal{A}$ 发送给 P_1 的 c.
3. 分两次发送 $b_2 = 0$ 和 $b_2 = 1$.

安全抛硬币 - 恶意敌手证明

Part 2.

只考虑确定性敌手 $\mathcal{A}$, P_1 被腐化的安全性. 构造模拟器 $S(\mathcal{A})$:

1. 发送 λ 给信任方计算 f_{ct} 收回一个比特 b.
2. 调用 $\mathcal{A}$ 从内部收到 $\mathcal{A}$ 发送给 P_1 的 c.
3. 分两次发送 $b_2 = 0$ 和 $b_2 = 1$.
   - 3.1 如果 $\mathcal{A}$ 回复 (b_1, r_1) 使得 $c = \mathsf{Com}(\mathsf{ck}, b_1; r_1)$, 则输出 $Continue$ 给理想功能, 并输出 $b_2 = b \oplus b_1$ 给 $\mathcal{A}$, 并输出 $\mathcal{A}$ 的输出.

安全抛硬币 - 恶意敌手证明

Part 2.

只考虑确定性敌手 $\mathcal{A}$, P_1 被腐化的安全性. 构造模拟器 $S(\mathcal{A})$:

1. 发送 λ 给信任方计算 f_{ct} 收回一个比特 b.
2. 调用 $\mathcal{A}$ 从内部收到 $\mathcal{A}$ 发送给 P_1 的 c.
3. 分两次发送 $b_2 = 0$ 和 $b_2 = 1$.
   - 3.1 如果 $\mathcal{A}$ 回复 (b_1, r_1) 使得 $c = \mathsf{Com}(\mathsf{ck}, b_1; r_1)$, 则输出 $Continue$ 给理想功能, 并输出 $b_2 = b \oplus b_1$ 给 $\mathcal{A}$, 并输出 $\mathcal{A}$ 的输出.
   - 3.2 如果 $\mathcal{A}$ 两次均回复 (b_1, r_1) 使得 $c \neq \mathsf{Com}(\mathsf{ck}, b_1; r_1)$, 则输出 $Abort$ 给理想功能, 并输出 $\mathcal{A}$ 的输出.

安全抛硬币 - 恶意敌手证明

Part 2.

只考虑确定性敌手 $\mathcal{A}$, P_1 被腐化的安全性. 构造模拟器 $S(\mathcal{A})$:

1. 发送 λ 给信任方计算 f_{ct} 收回一个比特 b.
2. 调用 $\mathcal{A}$ 从内部收到 $\mathcal{A}$ 发送给 P_1 的 c.
3. 分两次发送 $b_2 = 0$ 和 $b_2 = 1$.
   - 3.1 如果 $\mathcal{A}$ 回复 (b_1, r_1) 使得 $c = \mathsf{Com}(\mathsf{ck}, b_1; r_1)$, 则输出 $Continue$ 给理想功能, 并输出 $b_2 = b \oplus b_1$ 给 $\mathcal{A}$, 并输出 $\mathcal{A}$ 的输出.
   - 3.2 如果 $\mathcal{A}$ 两次均回复 (b_1, r_1) 使得 $c \neq \mathsf{Com}(\mathsf{ck}, b_1; r_1)$, 则输出 $Abort$ 给理想功能, 并输出 $\mathcal{A}$ 的输出.
   - 3.3 如果 $\mathcal{A}$ 只在 $b_2 \oplus b_1 = b$ 的情况下回复一次 (b_1, r_1), 则输出 $Continue$ 给理想功能, 输出 $b_2 = b \oplus b_1$ 给 $\mathcal{A}$, 并输出 $\mathcal{A}$ 的输出.

安全抛硬币 - 恶意敌手证明

Part 2.

只考虑确定性敌手 $\mathcal{A}$, P_1 被腐化的安全性. 构造模拟器 $S(\mathcal{A})$:

1. 发送 λ 给信任方计算 f_{ct} 收回一个比特 b.
2. 调用 $\mathcal{A}$ 从内部收到 $\mathcal{A}$ 发送给 P_1 的 c.
3. 分两次发送 $b_2 = 0$ 和 $b_2 = 1$.
   - 3.1 如果 $\mathcal{A}$ 回复 (b_1, r_1) 使得 $c = \mathsf{Com}(\mathsf{ck}, b_1; r_1)$, 则输出 $Continue$ 给理想功能, 并输出 $b_2 = b \oplus b_1$ 给 $\mathcal{A}$, 并输出 $\mathcal{A}$ 的输出.
   - 3.2 如果 $\mathcal{A}$ 两次均回复 (b_1, r_1) 使得 $c \neq \mathsf{Com}(\mathsf{ck}, b_1; r_1)$, 则输出 $Abort$ 给理想功能, 并输出 $\mathcal{A}$ 的输出.
   - 3.3 如果 $\mathcal{A}$ 只在 $b_2 \oplus b_1 = b$ 的情况下回复一次 (b_1, r_1), 则输出 $Continue$ 给理想功能, 输出 $b_2 = b \oplus b_1$ 给 $\mathcal{A}$, 并输出 $\mathcal{A}$ 的输出.
   - 3.4 如果 $\mathcal{A}$ 只在 $b_2 \oplus b_1 \neq b$ 的情况下回复一次 (b_1, r_1), 则输出 $Abort$ 给理想功能, 输出 $b_2 = b_1 \oplus b \oplus 1$ 给 $\mathcal{A}$, 并输出 $\mathcal{A}$ 的输出.

一些更新的问题

组合和通用可组合性

挑战

- 上述的安全性定义只考虑了单一协议的执行, 但是在实际应用中, 协议可能会被多次执行, 并且可能会与其他协议交互.

组合和通用可组合性

挑战

- 上述的安全性定义只考虑了单一协议的执行, 但是在实际应用中, 协议可能会被多次执行, 并且可能会与其他协议交互.
- 例如: 在一个复杂的系统中, 一个协议可能会调用另一个协议, 或者多个协议可能会并行执行.

组合和通用可组合性

挑战

- 上述的安全性定义只考虑了单一协议的执行, 但是在实际应用中, 协议可能会被多次执行, 并且可能会与其他协议交互.
- 例如: 在一个复杂的系统中, 一个协议可能会调用另一个协议, 或者多个协议可能会并行执行.
- **目标:** 设计一个协议, 使得即使在这种复杂的环境中, 也能保证协议的正确性和隐私性.

组合和通用可组合性

挑战

- 上述的安全性定义只考虑了单一协议的执行, 但是在实际应用中, 协议可能会被多次执行, 并且可能会与其他协议交互.
- 例如: 在一个复杂的系统中, 一个协议可能会调用另一个协议, 或者多个协议可能会并行执行.
- **目标:** 设计一个协议, 使得即使在这种复杂的环境中, 也能保证协议的正确性和隐私性.

组合和通用可组合性

挑战

- 上述的安全性定义只考虑了单一协议的执行, 但是在实际应用中, 协议可能会被多次执行, 并且可能会与其他协议交互.
- 例如: 在一个复杂的系统中, 一个协议可能会调用另一个协议, 或者多个协议可能会并行执行.
- **目标:** 设计一个协议, 使得即使在这种复杂的环境中, 也能保证协议的正确性和隐私性.

- 组合性: 协议在多个实例中执行时, 仍然保持安全性.

组合和通用可组合性

挑战

- 上述的安全性定义只考虑了单一协议的执行, 但是在实际应用中, 协议可能会被多次执行, 并且可能会与其他协议交互.
- 例如: 在一个复杂的系统中, 一个协议可能会调用另一个协议, 或者多个协议可能会并行执行.
- **目标:** 设计一个协议, 使得即使在这种复杂的环境中, 也能保证协议的正确性和隐私性.

- 组合性: 协议在多个实例中执行时, 仍然保持安全性.
- 通用可组合性: 协议在任意环境中执行时, 仍然保持安全性.

通用可组合性

定义 (通用可组合性 [Canetti, 2001, Canetti, 2020])

令 f 为一个两方理想功能, π 是一个计算 f 的两方计算协议. 我们说 π 是**对于恶意静态敌手的通用可组合性安全两方计算协议**, 如果对于任意非均匀多项式时间敌手 $\mathcal{A}$, 存在一个非均匀多项式时间针对理想功能的敌手 $\mathcal{S}$, 使得对于任意 $i \in \{1,2\}$, $|x| = |y|$, 有

$$\left\{\textsc{Real}_{\pi,\mathcal{A}(z),i}(x,y,\lambda)\right\}_{\lambda,x,y,z} \equiv_c \left\{\textsc{Ideal}_{f,\mathcal{S}(z),i}(x,y,\lambda)\right\}_{\lambda,x,y,z}.$$

并且该等价关系在任意环境下都成立.

随机谕言机

定义 (随机谕言机 [Bellare and Rogaway, 1995])

一个**随机谕言机**是一个理想功能 F_{RO}, 它接收两个输入 $(\lambda, 1^n)$, 并输出一个均匀分布的随机字符串 $r \xleftarrow{\$} \{0,1\}^n$.

随机谕言机

定义 (随机谕言机 [Bellare and Rogaway, 1995])

一个**随机谕言机**是一个理想功能 F_{RO}, 它接收两个输入 $(\lambda, 1^n)$, 并输出一个均匀分布的随机字符串 $r \xleftarrow{\$} \{0,1\}^n$.

挑战

- 在实际应用中, 随机谕言机通常被实现为一个密码学哈希函数.

随机谕言机

定义 (随机谕言机 [Bellare and Rogaway, 1995])

一个**随机谕言机**是一个理想功能 F_{RO}, 它接收两个输入 $(\lambda, 1^n)$, 并输出一个均匀分布的随机字符串 $r \xleftarrow{\$} \{0,1\}^n$.

挑战

- 在实际应用中, 随机谕言机通常被实现为一个密码学哈希函数.
- 但是, 哈希函数并不是真正的随机函数, 它们可能存在碰撞, 以及其他结构性弱点.

随机谕言机

定义 (随机谕言机 [Bellare and Rogaway, 1995])

一个**随机谕言机**是一个理想功能 F_{RO}, 它接收两个输入 $(\lambda, 1^n)$, 并输出一个均匀分布的随机字符串 $r \xleftarrow{\$} \{0,1\}^n$.

挑战

- 在实际应用中, 随机谕言机通常被实现为一个密码学哈希函数.
- 但是, 哈希函数并不是真正的随机函数, 它们可能存在碰撞, 以及其他结构性弱点.
- **问题:** [Canetti et al., 2004] 证明了在随机谕言机模型中安全的协议, 在实际应用中可能并不安全.

随机谕言机应用：Hash-and-Sign 签名

Hash-and-Sign 签名

- 选择一个安全的签名方案 (G, S, V).

随机谕言机应用：Hash-and-Sign 签名

Hash-and-Sign 签名

- 选择一个安全的签名方案 (G, S, V).
- 选择一个随机谕言机 H.

随机谕言机应用：Hash-and-Sign 签名

Hash-and-Sign 签名

- 选择一个安全的签名方案 (G, S, V).
- 选择一个随机谕言机 H.
- 陷门单向置换 F.

随机谕言机应用：Hash-and-Sign 签名

Hash-and-Sign 签名

- 选择一个安全的签名方案 (G, S, V).
- 选择一个随机谕言机 H.
- 陷门单向置换 F.
- 签名算法 S: 输入消息 m, 计算 $h = H(m)$, 输出签名 $s = F_{\mathsf{sk}}^{-1}(h)$.

随机谕言机应用：Hash-and-Sign 签名

Hash-and-Sign 签名

- 选择一个安全的签名方案 (G, S, V).
- 选择一个随机谕言机 H.
- 陷门单向置换 F.
- 签名算法 S: 输入消息 m, 计算 $h = H(m)$, 输出签名 $s = F_{\mathsf{sk}}^{-1}(h)$.
- 验证算法 V: 输入消息 m 和签名 s, 验证 $s = F_{\mathsf{vk}}(H(m))$.

References I

Bellare, M. and Rogaway, P. (1995).
Optimal asymmetric encryption.
In De Santis, A., editor, *EUROCRYPT'94*, volume 950 of *LNCS*, pages 92–111. Springer, Berlin, Heidelberg.

Blum, M. (1982).
Coin flipping by phone. compcon.

Canetti, R. (2001).
Universally composable security: A new paradigm for cryptographic protocols.
In *42nd FOCS*, pages 136–145. IEEE Computer Society Press.

Canetti, R. (2020).
Universally composable security.
J. ACM, 67(5):28:1–28:94.

References II

Canetti, R., Goldreich, O., and Halevi, S. (2004).
On the random-oracle methodology as applied to length-restricted signature schemes.
In Naor, M., editor, *TCC 2004*, volume 2951 of *LNCS*, pages 40–57. Springer, Berlin, Heidelberg.

Even, S., Goldreich, O., and Lempel, A. (1985).
A randomized protocol for signing contracts.
Communications of the ACM, 28(6):637–647.

Goldreich, O. (2004).
Foundations of Cryptography, Volume 2.
Cambridge university press Cambridge.