

Introduction to Security Reduction (Fuchun Guo's Course)

cryptowjm@gmail.com

2022 年 5 月 17 日

目录

1 Lecture 1 Definitions (Algorithm and Security Model)	4
1.1 Classical Cryptography vs Modern Cryptography	4
1.2 How to Define Algorithms	5
1.3 How to Define Security Models	6
2 Lecture 2 Preliminaries(Field, Group, Pairing, and Hash Function)	11
2.1 Finite Field	11
2.2 Group	12
2.2.1 Definition and Description	12
2.2.2 Easy Problem and Hard Problem	13
2.2.3 Two Group Choices	14
2.2.4 Computations Over Group	15
2.3 Bilinear Pairings	15
2.3.1 Symmetric Pairing	15
2.3.2 Asymmetric Pairing	16
2.3.3 Computations Over Pairing	17
2.4 Hash Function	17
2.4.1 The Application of Hash Functions	17
2.4.2 The Classification of Hash Functions	17
2.5 (Pseudo)Random Number Generator	18
3 Lecture 3 Preliminaries(Hard Problem and Secure Scheme)	19
3.1 Preliminaries	19
3.1.1 Computational and Decisional	19
3.1.2 Deterministic and Probabilistic	20
3.1.3 Polynomial and Exponential	20
3.1.4 Negligible and Non-Negligible	20
3.1.5 Probability and Advantage	21
3.2 Hard Problem	22
3.2.1 Hardness Definition	22
3.2.2 Examples	23
3.2.3 Hardness Analysis	25
3.3 Secure Scheme	25
3.3.1 Security Parameter	25

3.3.2	Security Level	26
3.3.3	Definition of Security	27
3.3.4	Definition of Hard Problem(Hardness)	27
4	Lecture 4 Entry to Security Reduction	28
4.1	Reduction in Computational Complexity	28
4.2	Security Reduction in Cryptography	30
4.2.1	Overview	30
4.2.2	Framework	30
4.2.3	From Breaking to Solving	32
4.3	Evaluation of Security Reduction	33
4.3.1	Cost and Loss in Reduction	33
4.3.2	Concrete Security	33
4.3.3	Ideal Security Reduction	34
5	Lecture 5 Difficulties in Security Reduction	36
5.1	Understanding Security Reduction	36
5.2	Simulation and After Simulation	37
5.2.1	Something Unknown	37
5.2.2	Simulation Results	37
5.2.3	Classification of Attacks	38
5.2.4	Relations	39
5.2.5	Classification of Security Reduction	40
5.3	Adversary	41
5.3.1	From Black-Box Adversary to Malicious Adversary	41
5.3.2	Malicious Adversary and Useful Attack	42
5.3.3	Adversary's Computing Ability	43
5.3.4	What the Adversary Knows and Never Knows	44
5.3.5	Summary of Adversary	45
6	Lecture 6 Simulation and Solution(Contradiction, Trapdoor, and Random Oracle)	46
6.1	Simulation and Contradiction	46
6.1.1	Secret key is Unknown	46
6.1.2	Secret key is Known	47
6.1.3	Simulation with Problem Instance	47
6.1.4	Simulation Examples	47
6.1.5	Trapdoor	47
6.2	Random Oracle	49
6.2.1	What is Random Oracle	49
6.2.2	How to use Random Oracle	50
6.2.3	Application of the RO	50
6.2.4	*Oracle Response	52

7 Lecture 6 Simulation and Solution(Contradiction, Trapdoor, and Random Oracle)	54
7.1 Simulation and Contradiction	54
7.1.1 Secret key is Unknown	54
7.1.2 Secret key is Known	55
7.1.3 Simulation with Problem Instance	55
7.1.4 Simulation Examples	55
7.1.5 Trapdoor	55
7.2 Random Oracle	57
7.2.1 What is Random Oracle	57
7.2.2 How to use Random Oracle	58
7.2.3 Application of the RO	58
7.2.4 *Oracle Response	60

1 Lecture 1 Definitions (Algorithm and Security Model)

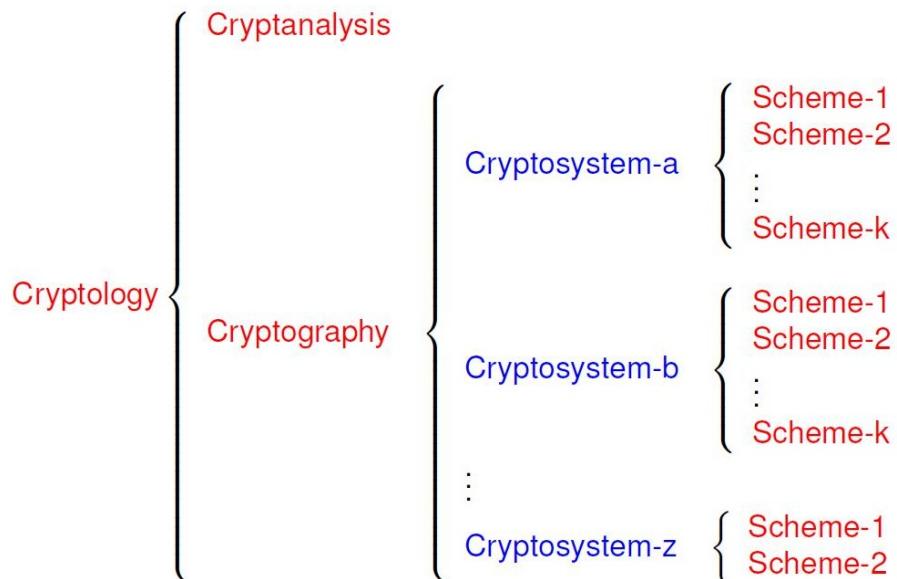
Outline:

- Classical Cryptography vs Modern Cryptography
- How to Define Algorithms
- How to Define Security Models
- Examples for Practice

1.1 Classical Cryptography vs Modern Cryptography

古典密码学 (Classical Cryptography) 曾经被认为是一门艺术，在计算机出现之后渐渐地成为了一门科学，称为现代密码学 (Modern Cryptography)。主要分为以下几个部分：

- Cryptology = Cryptography(构造) + Cryptanalysis(分析, analyze existing cryptography).
- Cryptography: 例如公钥密码学 (public-key cryptography), 基于群的密码学 (group-based cryptography), 椭圆曲线密码学 (elliptic-curve cryptography) 等, 是提供机密性、认证、完整性等安全服务的措施, 即不同的实现安全目标的途径.
- Cryptosystem: 例如数字签名、公钥加密、IBE 等, 是提供一项或多项安全服务的一系列算法.
- Scheme: 例如 BLS scheme, 是 Cryptosystem 中的具体算法构建的实例. 与 protocol 的区别: scheme 侧重于单方就能执行, protocol 侧重于交互的协议.



不同于古典密码学，现代密码学的通常侧重：

1. 定义：Definitions of algorithms, hard assumptions, advantage and so on.
2. 构造（模型）：Computational Model, Security Model and so on. 注：其余的模型也都是计算模型或者安全模型，随机预言机是计算模型。
3. 证明：Security Reduction, game-hopping proof and so on. 不同的密码体制常常有不同证明方法，并不存在通用的方法，这里主要介绍“归约证明” (Security Reduction).

在现代密码学的研究中，对于一个密码方案/协议，需要先给出算法和密码方案/协议的定义，再证明安全性。公钥密码的安全性证明经常会用到“归约证明”，需要使用安全模型将对密码方案/协议的攻击归约为计算问题。其中安全模型包括：

- 敌手能力：刻画敌手的计算能力等。
- 安全目标：安全性概念，说明敌手攻破方案的安全性意味着什么，怎么样才算攻破了该方案。

Recall：安全定义 + 归约证明，我们只需要一些简单的底层模块，例如单向函数等，就可以构造出加密、数字签名、消息鉴别等方案，以及密钥交换、身份认证等协议。对上层方案或者协议的攻击，可以有效归约为对底层模块的攻击，而对底层模块的攻击一般涉及某个数学困难问题，因此保证了上层方案或者协议的安全性。底层模块安全性的假设相当于数学中的公理，归约证明就是逻辑推理，密码学大厦终于有了坚实的基础和严密的逻辑保障。

下面主要分成 How to Define Algorithms 和 How to Define Security Models 两块展开。

1.2 How to Define Algorithms

定义一个新的 Cryptosystem 时（用来提供某种安全服务），首先需要通过若干个算法对其进行描述。如果是研究数字签名、公钥加密这些现有的 Cryptosystem，可以直接使用现有的算法。如果是定义全新的 Cryptosystem，则需要自己重新定义，例如曾经出现的代理签名、环签名等。

定义算法的步骤如下（以讲故事的形式）：

1. Fully understand the security service(s) (motivation). 理解安全服务的动机。
2. Which entities (e.g. signer) are involved. 参与方有哪些（加密者，解密者，签名者，验证者等）、是否有可信第三方等。
3. How many algorithms are involved. 需要多少种算法才能描述这个 cryptosystem?
4. What is the name of each algorithm. 每个算法的名字是什么？类似于编程的函数名。
5. Who will run each algorithm. 谁来运行每一个算法，对应前面的 entities。
6. What are inputs and outputs of each algorithm (objects). 每一个算法的输入和输出。e.g. 验证签名的时候要输入私钥吗？
7. What correctness must the algorithms satisfy? 算法需要满足的正确性。

以数字签名算法为例，How to Define Algorithms for Digital Signatures？

- A party, say Alice, wants to convince all other parties that a message m is published by her.
- To do so, Alice generates a public/secret key pair $(pk; sk)$ and publishes the public key pk to all verifiers.
- To generate a signature σ_m on m , she digitally signs m with her secret key sk .
- Upon receiving $(m; \sigma_m)$, any receiver who already knows pk can verify the signature σ_m and confirm the origin of the message m .

1. Motivation 就是上面故事中的描述。

2. entities 包括 Signer 和 Verifier(Alice 和 Bob 只是故事中的人物, 描述 motivation 时用. 实际算法中是 Signer 和 Verifier, 或者加密者、解密者、云服务提供商、数据拥有者、数据接收者等).
3. 1 (system parameter generation) + 3 (generate, sign, verify).
4. SysGen, KeyGen, Sign, Verify.
5. SysGen (Authority), KeyGen (Signer), Sign (Signer), Verify (Verifier). A signer can generate a system parameter for him/her to use alone so that SysGen is run by Signer.
6. 将算法抽象化, 这里输出是用 “ \rightarrow ” 而不是 “ $=$ ” .

```
SysGen( $\lambda$ ) $\rightarrow$  SP.  

KeyGen(SP) $\rightarrow$  (pk, sk).  

Sign(SP, sk, m)  $\rightarrow$   $\sigma_m$ .  

Verify(m,  $\sigma_m$ , SP, pk) $\rightarrow$  {0, 1}.
```

input and output OBJECTS only for algorithm definition.

How to Define Algorithms for Digital Signatures?

表达上类似, 但是需要略作修改

SysGen: The system parameter generation algorithm takes as input a security parameter λ . It returns the system parameters SP.

KeyGen: The key generation algorithm takes as input the system parameters SP. It returns a public/secret key pair (pk, sk).

Sign: The signing algorithm takes as input a message m from its message space, the secret key sk, and the system parameters SP. It returns a signature of m denoted by σ_m .

Verify: The verification algorithm takes as input a pair (m, σ_m), the public key pk, and the system parameters SP. It returns “accept” if σ_m is a valid signature of m signed with sk; otherwise, “reject.”

Note: All algorithms are probabilistic polynomial time algorithms.

7. 正确性: 在定义完算法后, 需要描述算法的正确性. 除了加密(解密)和签名(验签)算法之外, 密钥生成算法也具有**正确性 (Correctness)**: 由其产生的公私密钥对, 拥有密钥时, 签名算法能被验证, 加密算法能被解密. 以数字签名为例, 如果一个公钥能产生签名, 但这个签名不能被验证, 那就是没有意义的, “密钥生成算法的正确性”就是想保证生成的公私密钥对一定能用, 而不需要运行很多次密钥生成算法将可能的公私密钥对逐一尝试哪个能用. 有可能签名算法需要运行很多次才能产生出一个签名, 而“密钥生成算法的正确性”能保证产生的这个签名一定是可以验证的.

1.3 How to Define Security Models

配合卧报第 29 期食用.

- How to analyze the security of a scheme? 如何分析一个方案的安全性?

- Can a scheme (secure in a security model) resist any attack? 一个（在安全模型下）安全的方案能抵抗所有的攻击吗？
- Is a security model defined for a scheme? 安全模型是为了方案定义的吗？

基于复杂性理论的密码是没有绝对安全的概念，只有攻击密码方案所需时间和空间的复杂性，当攻击一个方案的复杂性更高时就认为这个方案更加安全。而复杂性的分析是对计算问题进行的，因此我们需要把攻击密码方案转化为计算问题，才可以分析其复杂性。**安全模型**的作用就是把对一个密码方案的攻击描述成一个计算问题，通过归约 (reduction) 的方式证明解决一个问题比另一个问题**更难 (复杂性更高)**。



什么是安全模型 当提出一个方案时，我们并不能证明它在所有的攻击（例如：重放攻击、碰撞攻击等）下安全，相反，我们能在**一个安全模型中**证明提出的方案是安全的。

安全模型可以看成是对一个 cryptosystem 的一系列攻击，将具体的攻击看作一个个点，则安全模型是包含了一些点的集合，可能会包含很多点，但不会是全集。“**证明了在某种安全模型下安全**”即表示“**能抵抗这个安全模型中的一系列攻击**”，但是每个安全模型都不会 capture 所有的攻击。

安全模型是定义在一个 cryptosystem（如数字签名、公钥加密、基于身份的加密）上的，而不是定义在一个 scheme 上的，必须要 **match** 整个 cryptosystem。比如一个数字签名的 scheme 用到了哈希函数，但不是所有的数字签名都包含哈希函数，那么数字签名这个 cryptosystem 的安全模型中就不能有哈希函数；又比如一个公钥加密的 scheme 用到了群或者 pairing，但不是所有的公钥加密的 scheme 都用到这些，因此在公钥加密的 cryptosystem 的安全模型中就不能有它们。

安全模型的定义 安全模型的定义一般是通过实验 (experient) 或者游戏 (game) 的形式，实验的形式一般写成伪代码，看起来比较简洁，但对新手理解较为困难，这里我们用游戏的形式。**对于提供不同的 security service 的 cryptosystem，有不同的安全模型的定义，具体取决于安全目标和敌手能力。**

How to Define a Security Model?

只有 Adversary 和 Challenger 两者，没有加密者解密者，没有签名者验证者，没有可信第三方。

We can use a game played between adversary and challenger to describe a security model.

由 Challenger 产生公私钥对

- The challenger is the secret key owner of a cryptosystem.
- The adversary is trying to break the cryptosystem.

A security model defines:

cryptosystem 而不是 scheme

- The adversary's capabilities: Challenger 知道一些 Adversary 不知道的东西，信息差！否则 Adversary 容易做到 Challenge 做的
 - What information the adversary can query
 - When the adversary can query information
- The adversary's security goal:
 - How the adversary wins the game (breaks the cryptosystem).

The definition of a security model is composed of four parts:

初始化

- **Setup:** The initialization between the adversary and the challenger.
 敌手进行询问，Challenger必须如实回答敌手的询问
- **Capabilities:** Describe what and when the adversary queries.
- **Security Goal:** The condition of winning the game for the adversary.
- **Advantage:** Define a parameter satisfying.
 - If the parameter is **negligible**, the cryptosystem is secure.
 - Otherwise, the parameter is **non-negligible** and it is insecure.

Note: Advantage and (non)negligible will be introduced in later lectures.

敌手赢得游戏的“优势”，当优势可忽略→安全，当优势不可忽略→不安全

在定义安全模型时，需要注意：

- 只考虑算法的 input 和 output，不考虑算法的具体执行过程和实例。例如当敌手询问消息 m 的签名时，challenger 必须通过将 (pk, sk, m) 输入到签名算法运行得到签名 σ_m 并回复给敌手，在这个过程中我们并不关心 σ_m 是怎么生成的和 σ_m 长什么样（这取决于具体的签名算法），我们关注的重点是敌手询问的东西（例如消息 m ）。
- 注：密码学中的攻击侧重于敌手能获得什么信息，而不是敌手如何获得这些信息和信息具体内容。
- 不考虑敌手的 trivial attack 和 strategy。
 - trivial attack 指的是敌手直接询问一些能轻松打破 cryptosystem 的问题，例如敌手让 challenger 分享自己的私钥。
 - strategy 指的是敌手的策略，challenger 不用管敌手为什么要询问这个，敌手要什么 challenger 就给什么，每次都 correctly and honestly 地返回敌手的查询就够了。
- 安全模型的初始优势必须是可忽略的 (negligible)，不能让敌手有一种策略询问之后变成不可忽略。否则敌手在这个安全模型下可以直接 break 这个 cryptosystem，这样无论构造怎么样的 scheme 在这个安全模型下敌手都能攻破。如果敌手有一种方法直接让安全模型变成 nonnegligible 的，则安全模型的定义出错了！

定义数字签名的安全模型 下面以定义数字签名的安全模型为例。

1. Setup: A key pair is generated and the adversary knows pk.
 2. Capabilities: The adversary obtains signatures of some messages.
 3. Goal: The adversary cannot forge signature of a new message.
 4. Advantage: The probability of successful forgery.

注：new 指的是这个消息敌手没拿去问过 challenger，否则可以直接拿 challenger 的回答；敌手也不能直接向 challenger 索要 signing key，这些都是 trivial attack。

Any attack that will make advantage = 1 must be forbidden.

数字签名的安全模型的形式化定义：

Security Model of Digital Signatures

The security model of existential unforgeability against chosen-message attacks (**EU-CMA**) can be described as follows.

Setup. Let SP be the system parameters. The challenger runs the key generation algorithm to generate a key pair (pk, sk) and sends pk to the adversary. The challenger keeps sk to respond to signature queries.

Query. The adversary makes signature queries on messages that are adaptively chosen by the adversary itself. For a signature query on the message m_i , the challenger runs the signing algorithm to compute σ_{m_i} and then sends it to the adversary.

Forgery. The adversary returns a forged signature σ_{m^*} on some m^* and wins the game if

- σ_{m^*} is a valid signature of the message m^* .
- A signature of m^* has not been queried in the query phase.

The advantage ϵ of winning the game is the probability of returning a valid forged signature.

与算法类似，写法上可以完全借鉴他人的，但是切记不能copy



定义新系统的安全模型 前面已经提到，有的 cryptosystem 可以提供不止一个 security service，此时每个 security service 都需要一个安全模型。例如签密 signcryption = signature + encryption：既要将消息保密不让第三方看到，也要对消息签名。

在这个 cryptosystem 的安全模型中，存在两个安全模型：cannot break signature 和 cannot break encryption，即第三方不能伪造签名也无法看到加密后的消息。

在一些特殊的情况下，一个安全模型也有可能 capture 多个 security services，例如 anonymous receiver with message indistinguishability in IBE.

安全模型的分类

- 标准安全模型 (Standard security model): the capability and security goal are acceptable by **most applications** (or by cryptography community).
- 强安全模型 (Strong security model): the adversary has a strong capability and/or a easier security goal than the standard one. (Resist more attacks)
- 弱安全模型 (Weak security model): the adversary has a weak capability and/or a harder security goal than the standard one. (Resist less attacks)

注：

1. Standard security model 和 Standard model 不同，前者是安全模型，后者是计算模型。
2. 安全模型的强和弱具备**相对性**，强安全模型可以理解为敌手更强的模型，在更强的安全模型下安全的方案能抵抗更多的攻击，在相对弱的模型下也安全。但也不是一个方案在更强的模型下提出就是更好的，在选择安全模型时**还要考虑效率、应用场景等**。非常著名的例子：数字签名中，抗泄露数字签名模型（允许敌手知道部分 signing key）比标准数字签名模型强，但应用场景更局限。标准安全模型并不一定是最强的，但是是符合大多数场景、最适用的。详细见卧报 29。

3. Weak security model 的典型例子：敌手被限制在只允许知道某些消息的签名，而不是可以询问任意消息的签名。除此以外，在安全模型中，敌手可以自己选择消息询问还是只能由 challenger 给定等，这些在证明上影响都非常大。
4. Weak security model 的出现是因为一些方案能在 Weak security model 下构建效率更高，更适合应用。有时解决不了的 open problem 可以先走半步：弱化安全模型 Boneh 身份加密是典型的例子。

安全模型的意义 安全模型是安全归约的核心，没有安全模型就无法将攻破方案抽象成解决计算问题，也就无法进行安全归约。

安全模型的核心本质：Breaking a scheme in a security model \Leftrightarrow solving a computing problem.

2 Lecture 2 Preliminaries(Field, Group, Pairing, and Hash Function)

Outline:

- 1. Finite Field
 - Symmetric and Asymmetric
 - Computations Over Pairing
- 2. Cyclic Groups
 - Definition and Description
 - Easy Problem and Hard Problem
 - Two Group Choices
 - Computations Over Group
- 3. Bilinear Pairings
 - 4. Hash Functions
 - The Application of Hash Functions
 - The Classification of Hash Functions
 - 5. *(Pseudo)Random Number Generator
 - 6. *Insecure Schemes

2.1 Finite Field

有限域 (Finite Field, Galois Field) 记作 $(F, +, *)$, 是一个包含了有限个元素的集合, 以及两个二元运算“+”(addition) and “*”(multiplication), 满足: 吸收、结合律、交换律、加乘法单位元、加乘法逆元、分配律.

- $\forall u, v \in F$, we have $u + v \in F$ and $u * v \in F$.
- $\forall u_1, u_2, u_3 \in F$, $(u_1 + u_2) + u_3 = u_1 + (u_2 + u_3)$ and $(u_1 * u_2) * u_3 = u_1 * (u_2 * u_3)$.
- $\forall u, v \in F$, we have $u + v = v + u$, $u * v = v * u$
- $\exists 0_F, 1_F \in F$ (identity elements), $\forall u \in F$, we have
 $u + 0_F = u$ and $u * 1_F = u$.
- $\forall u \in F$, $\exists -u \in F$ such that $u + (-u) = 0_F$.
- $\forall u \in F^*$, $\exists u^{-1} \in F^*$ such that $u * u^{-1} = 1_F$. Here, $F^* = F \setminus \{0_F\}$.
- $\forall u_1, u_2, v \in F$, we have $(u_1 + u_2) * v = u_1 * v + u_2 * v$.

利用逆元可以将“+”和“*”扩展得到“-”(subtraction) 和“/”(division), 共有这四种基本运算.

- $\forall u, v \in F$, we have

$$u - v = u + (-v),$$

which is the addition of u and the additive inverse of v .

- $\forall u \in F, v \in F^*$, we have

$$u/v = u * v^{-1},$$

which is the multiplication of u and the multiplicative inverse of v .

我们也经常用 $(F_{q^n}, +, *)$ 来表示一个有限域, 这里 q 是一个素数称为该有限域的特征, n 是一个正整数. F_{q^n} 中的每一个元素都是一个 n 维向量, 向量里的每个元素都在 F_q 中, 即有限域 $(F_{q^n}, +, *)$ 共有 q^n 个元素, 这些元素的 bit length 是 $n|q|$, $|q|$ 表示将 (十进制) q 的值用二进制表示的长度.

除此之外，也经常会令 $n = 1$ ，得到素数域 (Prime Field) F_q .

$$(\mathbb{F}_q, +, *)$$

- There are q elements in this field $\mathbb{Z}_q = \{0, 1, 2, \dots, q - 1\}$.
- $u + v = u + v \pmod{q}$.
- $u * v = u * v \pmod{q}$.
- $-u = q - u$.
- $u^{-1} = u^{q-2} \pmod{q}$.

Note: Prime field is important due to the use of a group of prime order.

2.2 Group

2.2.1 Definition and Description

群的描述：通常用 (G, g, p) 来描述一个群，其中 G 是群的空间， g 是群的生成元， p 是群的阶数， (G, g, p) 是群的基本组成部分，对于一些特殊的群（例如模乘群）需要更多的信息来描述。

Size of Group Element: 在描述群时，需要考虑用多少个比特能表示一个群（或者说群中的元素），对于素数 p , $G = \{g^0, g^1, \dots, g^{p-1}\}$ 中有 p 个元素，每个群元素都有相同的 size，能编码成不同的比特串。以 $p = 2^{160}$ 为例，为了表示 $p = 2^{160}$ 个元素至少需要一个 160 位的比特串，这只是最理想的情况，通常情况下要求 $|g| \geq 160$ ，这里 $|g|$ 表示将（十进制） g 的值用二进制表示的长度，这还只是 80 比特安全（教科书级别），目前一般要求 128 比特安全，安全等级参见卧报 30 期。

Definition (Abelian Group)

An abelian group, denoted by (\mathbb{H}, \cdot) , is a set of elements with one binary operation “.” defined as follows.

- $\forall u, v \in \mathbb{H}$, we have $u \cdot v \in \mathbb{H}$. 这里的“·”可以是“+”也可以是“*”，也可以是第三种运算
- $\forall u_1, u_2, u_3 \in \mathbb{H}$, we have $(u_1 \cdot u_2) \cdot u_3 = u_1 \cdot (u_2 \cdot u_3)$.
- $\forall u, v \in \mathbb{H}$, we have $u \cdot v = v \cdot u$.
- $\exists 1_{\mathbb{H}} \in \mathbb{H}, \forall u \in \mathbb{H}$, we have $u \cdot 1_{\mathbb{H}} = u$.
- $\forall u \in \mathbb{H}, \exists u^{-1} \in \mathbb{H}$, such that $u \cdot u^{-1} = 1_{\mathbb{H}}$.

Definition (Abelian Group with Cyclic)

An abelian group \mathbb{H} is a cyclic group if there exists (at least) one generator, denoted by h , which can generate the group \mathbb{H} :

群里的任何一个元素都可以通过 h 的某次方得到

$$\mathbb{H} = \{h^1, h^2, \dots, h^{|\mathbb{H}|}\} = \{h^0, h^1, h^2, \dots, h^{|\mathbb{H}|-1}\},$$

where $|\mathbb{H}|$ denotes the group order of \mathbb{H} and $h^{|\mathbb{H}|} = h^0 = 1_{\mathbb{H}}$.

Definition (Abelian Group with Cyclic of Prime Order)

A group \mathbb{G} is a cyclic subgroup of prime order if it is a subgroup of a cyclic group \mathbb{H} and $|\mathbb{G}|$ is a prime number, where

- $|\mathbb{G}|$ is a divisor of $|\mathbb{H}|$; 取 \mathbb{H} 的一个子集 \mathbb{G} , \mathbb{G} 是素数阶循环群, \mathbb{H} 不一定是素数阶
- There exists a generator $g \in \mathbb{H}$, which generates \mathbb{G} . 一定是素数阶

因为 \mathbb{H} 有可能是合数阶，所以我们才取素数阶子群 \mathbb{G} ，素数阶的好处见下一页
Abelian Group with Cyclic of Prime Order is short as Cyclic Group.

素数阶的好处：可以避免特殊的Attack以及任意一个群中的元素（除了单位元）都是生成元，都有逆元。

$$\mathbb{G} = \{g^0, g^1, \dots, g^{p-1}\} \text{ for a prime } p.$$

G本身是自己的最小子群

- The group \mathbb{G} is the smallest subgroup without confinement attacks.
- Any group element except g^0 is a generator of \mathbb{G} .
- Any integer in $\{1, 2, \dots, p-1\}$ has a modular multiplicative inverse.
For any $x \in \mathbb{Z}_p^* = \{1, 2, \dots, p-1\}$, we can definitely compute

$$g^{\frac{1}{x}}. \quad \text{即在G中任取一个x, x的逆元一定也在G中}$$

Note: We don't have to use a group (in prime order) in cryptography.

在密码构造中，素数阶不是必须要求的，但是可以提供一些好处

2.2.2 Easy Problem and Hard Problem

想要用循环群构建密码，首先要考虑群中的计算问题：一些计算问题是简单的，例如群上的加法、乘法或者指数运算 (group exponentiation)；而另一些计算问题必须是复杂的，否则密码不安全 (例如可以容易地通过公钥计算私钥)，最常见的群上的困难问题是 Discrete Logarithm Problem(离散对数问题, DLP)，这里要注意 DLP 也只是在一些特定的群中困难，并不是随便找一个群都有复杂的 DLP.

Easy(group exponentiation) 计算 x 个 g 相乘 $g^x = g \cdot g \cdots g \cdot g$.

直接逐个乘需要进行 $x - 1$ 次群基本运算，当 x 很大时，例如 $x = 2^{160}$ ，我们不可能逐个做乘法，常见的解决方案是 square-and-multiply algorithm:

Given $g \in G$ and $x \in \mathbb{Z}_p$, we can compute g^x as follows:

1. Convert x into an n-bit string $x = x_{n-1} \dots x_1 x_0 = \sum_{i=0}^{n-1} x_i 2^i$, let $g_i = g^{2^i}$, compute $g_i = g_{i-1} \cdot g_{i-1}$ for $i \in [1, n-1]$, 只需要至多 $n - 1$ 次群基本运算.
2. Compute g^x by $g^x = g^{\sum_{i=0}^{n-1} x_i 2^i} = \prod_{i=0}^{n-1} g_i^{x_i}$, 只需要至多 $n - 1$ 次群基本运算.
3. 分析: 共 $2n - 2$ 次群基本运算, 这里 n 是 x 的比特长度 (二进制展开的位数), 时间复杂度为 $O(\log x)$, 比前面的 $x - 1 = O(x)$ 快很多.

Hard(discrete logarithm problem) 给定 $g, h \in G \setminus 1_G$, 整数 x 满足 $g^x = h$ 被称为离散对数 (discrete logarithm, DL), 通过 g 和 h 求解 x 的问题称为离散对数问题 (DLP). 如果 G 是一个素数阶的群，则对于群中的两个元素 g 和 h , 一定存在符合条件的 x (这是选用素数阶的优点).

离散对数问题的困难性分析: 对于群 (G, g, p) , 目前求解离散对数问题的最有效的算法 (Pollard Rho Algorithm) 需要 $\Omega(\sqrt{p})$ 步指数运算 (exponentiation), 其中 p 是群的阶数.

- This algorithm can solve DLP over any group (order of the group is known), and the step number should be $c \cdot \sqrt{p}$ for some positive coefficient c in computational complexity.
- DLP over some specific groups could take less than \sqrt{p} steps, even could be easy $O(1)$ steps. 一些特定的群上的 DLP 问题是简单的，因此**需要考虑群的选择**.

Recall: $\Omega(\sqrt{p})$ steps means “lower bound” \sqrt{p} (at least), and $O(\sqrt{p})$ steps means “upper bound” \sqrt{p} (at most).

DLP 是基于群的密码中最基本的困难问题，基于群的密码方案的安全性都取决于 DLP 的困难性，如果 DLP 能被高效解决，那么所有基于群的密码方案都不再安全.

2.2.3 Two Group Choices

如何选择一个合适的群，使得 DLP 是困难的。为了构造一个群 (G, g, p) ，使得 DLP 问题是困难的（方案是安全的），必须考虑解决 DLP 的 generic attack（一般攻击）和 specific attack（特殊攻击，基于特殊的群结构）。以 80 比特安全为例，敌手必须进行至少 2^{80} 步才能攻破方案。

- generic attacks: generic attack 指的是只需要知道 group order 就能求解 DLP 的一般方法，和 group 的具体构造没有关系。例如上面提到的 Pollard Rho Algorithm，当群 G 的 order 为 2^{160} 时，解决群 G 上的 DLP 需要 2^{80} 步，因此为了抵抗 generic attack，参数需要满足 $p \geq 2^{160}$ 。换句话说，当群的 order $< 2^{160}$ 时，敌手可能可以用小于 2^{80} 步才能攻破这个方案。
- specific attacks: specific attacks 指的是一些基于特殊的群的结构来解决 DLP 的方法，在设置群时，其它的参数（例如群中元素的尺寸）也必须足够大以抵抗一些特殊的攻击。

对于一个有限域 $(F_{q^n}, +, *)$ ，已经可以得到两个群 $(F_{q^n}, +)$ 和 $(F_{q^n}, *)$ ，为什么我们不直接用这两个群构造密码方案，而是定义一些更加复杂和高级的新的有限域上的循环群，再构造密码方案呢？原因很复杂，例如椭圆曲线群可以在安全等级相同的情况下让群的元素表示更短（short representation）。

常选择的循环群包括乘法群（Multiplicative Group）和椭圆曲线群（Elliptic Curve Group）。

乘法群（Multiplicative Group）

A multiplicative group is defined as (\mathbb{G}, g, q, p) .

- **Group Elements.** The group elements are integers from

$$\mathbb{Z}_q^* = \{1, 2, \dots, q-1\}, |g| = \log^q.$$

■ **Group Generator.** g is from \mathbb{Z}_q^* (some integers from this set are not the generators of \mathbb{G}). q是一个大素数，p是个比q小的素数且满足p|(q-1)

■ **Group Order.** p satisfying $p|(q-1)$.

■ **Group Operation.** We have $u \cdot v = u \times v \bmod q$.

Note: The integer q significantly affects the hardness of DLP in this special construction and q must be at least 1024 bits. Otherwise, solving its DLP takes less than 2^{80} steps. 为了抵抗generic attack,大素数p至少160比特,而为了防止存在一种特殊的攻击specific attack能解决DL问题, q至少需要1024比特(保证至少 2^{80} 步指数运算才能解决DL问题,这个长度和RSA差不多) 

椭圆曲线群（Elliptic Curve Group）

An elliptic curve group is defined as (\mathbb{G}, g, p) .

- **Group Elements.** The group elements are points (represented with x-coordinate and y-coordinate) on the elliptic curve. When the curve is given, we can use the x-coordinate and one more bit only to represent a group element. 如果不考虑效率只是为了减少长度,有时我们会只记录横坐标和一个参数计算纵坐标和点

■ **Group Generator.** g is also a point.

■ **Group Order.** p a prime order.

■ **Group Operation.** We have $u \cdot v$ defined by elliptic curves.

椭圆曲线的好处：群元的长度和群的阶数order几乎一样，在这种情况下还能保证解决DLP需要 2^{80} 步运算

Note: The size of group element can be as short as the group order. That is, $|g| = |p| = 160$ where solving its DLP requires 2^{80} steps.

2.2.4 Computations Over Group

群上的计算 (素数阶群)

- **Group Operation.** Given $g, h \in \mathbb{G}$, compute $g \cdot h$.
- **Group Inverse.** Given $g \in \mathbb{G}$, compute $\frac{1}{g} = g^{-1}$.
Since $g^p = g \cdot g^{p-1} = 1$ (not the integer 1), we have $g^{-1} = g^{p-1}$.
- **Group Division.** Given $g, h \in \mathbb{G}$, compute $\frac{g}{h} = g \cdot h^{-1}$.
- **Group Exponentiation.** Given $g \in \mathbb{G}$ and $x \in \mathbb{Z}_p$, compute g^x .

Question: Given $g \in \mathbb{G}, (x, y, z) \in \mathbb{Z}_p$, do you know how to compute

$$g^{-\frac{y-z}{x+z}}?$$

把这种复杂的复合运算用来当作高级的密码方案所需要的计算

2.3 Bilinear Pairings

Bilinear Pairing(双线性对) 的思想是把两个”椭圆曲线群”中的群元素映射到一个”乘法群”中，并保持同态性，这个思想最早来源于将”椭圆曲线群”上的困难问题映射到”乘法群”中解决，本来是用来做密码攻击的，但能攻击的方案需要满足的限制条件较多，现在已经被用来做密码方案构造。

Bilinear Pairing 主要有以下三大类：

Bilinear pairing ($\mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$) falls into the following three types.

- Symmetric. $\mathbb{G}_1 = \mathbb{G}_2 = \mathbb{G}$. Denoted by $\mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ 对称:不考虑将size变短时可以直接取对称
- Asymmetric 1. $\mathbb{G}_1 \neq \mathbb{G}_2$ with homomorphism $\psi : \mathbb{G}_2 \rightarrow \mathbb{G}_1$.
- Asymmetric 2. $\mathbb{G}_1 \neq \mathbb{G}_2$ with no efficient homomorphism.
非对称: 有时构造方案想要更短的密文/更短的签名/发送更短的长度, 就考虑非对称
Note: Homomorphism might be needed in scheme construction.

2.3.1 Symmetric Pairing

Symmetric Pairing (对称双线性对): 将椭圆曲线群 G 中的元映射到乘法群 G_T 中，满足下面三条性质，两个群上的 DLP 都是困难的。

Let $\mathbb{PG} = (\mathbb{G}, \mathbb{G}_T, g, p, e)$ be a symmetric-pairing group. Here, \mathbb{G} is an elliptic curve group, \mathbb{G}_T is a multiplicative subgroup, $|\mathbb{G}| = |\mathbb{G}_T| = p$, g is a generator of \mathbb{G} , and e is a map satisfying the following three properties.

- For all $u, v \in \mathbb{G}, a, b \in \mathbb{Z}_p$, $e(u^a, v^b) = e(u, v)^{ab}$.
- $e(g, g)$ is a generator of group \mathbb{G}_T .
- For all $u, v \in \mathbb{G}$, there exist efficient algorithms to compute $e(u, v)$.

Two types of DLP:

- Compute x from g and g^x . 两个不同群上的离散对数问题
- Compute x from $e(g, g)$ and $e(g, g)^x$.

To make sure solving any DLP takes at least 2^{80} steps, it requires that

$$|g| \geq 512(\text{bits}), |e(g, g)| \geq 1024(\text{bits}).$$

Note: 1024 is just a textbook size. We need a larger parameter now.

2.3.2 Asymmetric Pairing

Asymmetric Pairing(非对称双线性对): 将两个椭圆曲线群 G_1, G_2 中的元映射到乘法群 G_T 中, 满足下面三条性质, 三个群上的 DLP 都是困难的.

Let $\mathbb{PG} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, p, e)$ be an asymmetric-pairing group. Here, $\mathbb{G}_1, \mathbb{G}_2$ are elliptic curve groups, \mathbb{G}_T is a multiplicative subgroup, $|\mathbb{G}_1| = |\mathbb{G}_2| = |\mathbb{G}_T| = p$, g_1 is a generator of \mathbb{G}_1 , g_2 is a generator of \mathbb{G}_2 , and e is a map satisfying the following three properties.

- For all $u \in \mathbb{G}_1, v \in \mathbb{G}_2, a, b \in \mathbb{Z}_p$, $e(u^a, v^b) = e(u, v)^{ab}$.
- $e(g_1, g_2)$ is a generator of group \mathbb{G}_T .
- For all $u \in \mathbb{G}_1, v \in \mathbb{G}_2$, there exist efficient algo. to compute $e(u, v)$.

Three types of DLP:

- Compute x from g_1 and g_1^x .
- Compute x from g_2 and g_2^x .
- Compute x from $e(g_1, g_2)$ and $e(g_1, g_2)^x$.

To make sure solving any DLP takes at least 2^{80} steps, it requires that

$$|g_1| \geq 160(\text{bits}), |g_2| \geq 1024(\text{bits}), |e(g, g)| \geq 1024(\text{bits}).$$

为了80比特安全和 g_1 群上的元越短越好, g_2 和 e 都必须比较长.(很多论文错写 g_2 160bits)

Note: We have to set $|g_2| = |e(g_1, g_2)|$ due to asymmetric construction.

2.3.3 Computations Over Pairing

A symmetric-pairing group is composed of groups $(\mathbb{G}, \mathbb{G}_T)$ of prime order p and a bilinear map e . All computations over a pairing group are summarized as follows.

- All modular operations over \mathbb{Z}_p (prime field). 素数域上的模加、模乘
- All group operations over the groups $(\mathbb{G}, \mathbb{G}_T)$.
群上的运算：乘法、除法、群指数运算
- The pairing computation $e(u, v)$ for all $u, v \in \mathbb{G}$.

Question: Given $g, g^a \in \mathbb{G}, (x, y) \in \mathbb{Z}_p$, do you know how to compute
可以将这么多的计算综合起来做一些高级的计算，用来做密码方案，很多密码方案都是由基本运算构成 $e(g, g)^{(a+x)(a+y)}$?

2.4 Hash Function

A hash function takes an arbitrary-length string as an input and returns a much shorter string as an output. 输入任意长度的串 → 输出固定长度的串（短）.

2.4.1 The Application of Hash Functions

大多数公钥密码方案的构造都需要用到哈希函数. 一方面可以提高效率，另一方面在安全证明上也可以用到. 如果选取的哈希函数不安全则这些密码方案就都不安全了.

- 构建密码方案: 在构建密码方案时，我们通常不能将所有的值全部嵌入到 \mathbb{Z}_p 或 G 的有限空间中，可以使用哈希函数: 当 $m \notin \mathbb{Z}_p$ 时，用 $g^{H(m)}$ 代替 g^m . 在数字签名时，假设算法对消息是 1bit 1bit 地进行签名，1GB 的电影需要签名很久，如果使用 Hash Function 压缩到 160bits 就可以很快签名.
- 安全性证明: 在安全归约中，哈希函数经常被用作随机预言机 (RO, random oracle)，RO 是一种计算模型，不是安全模型.

2.4.2 The Classification of Hash Functions

- Security-Based Classification: 哈希函数根据安全性可以分为 One-Way Hash Function(OWHF, 单向哈希函数) 和 Collision-Resistant Hash Function (CRHF, 抗碰撞哈希函数) 两大类.
 1. One-Way Hash Function. Given a one-way hash function H and an output string y , it is hard to find a pre-image input x satisfying $y = H(x)$. (敌手很难找到原象)
 2. Collision-Resistant Hash Function. Given a collision-resistant hash function H , it is hard to find two different inputs x_1 and x_2 satisfying $H(x_1) = H(x_2)$. (敌手很难找到一对碰撞)
 3. 在描述方案时，可以直接写成 Cryptographic Hash Function，指满足题目应用场景和安全性的哈希函数 (有上面两个性质中的一个或都有). 不需要描述得太清楚，可以适当简化.
- Application-Based Classification: 哈希函数可以根据输出空间和应用分成下面三类，其中输入空间可以是任意串 (any arbitrary strings).

1. $H : \{0, 1\}^* \rightarrow \{0, 1\}^n$, 输出空间是一个 n 比特的串, 主要用在混合加密方案中生成 $\{0, 1\}^n$ 空间的对称密钥, 当作对称加密算法的 key 进行加密解密.
2. $H : \{0, 1\}^* \rightarrow \mathbb{Z}_p$, 输出空间是 $\{0, 1, 2, \dots, p-1\}$, 其中 p 是群的阶数, 用来嵌入 hash 值到 group exponents $g^{H(m)}$.
3. $H : \{0, 1\}^* \rightarrow G$, 输出空间是循环群 G , 主要将输入串 hash 成群中的元素, 这样的哈希函数只在一些群中存在. 与 hash 到 \mathbb{Z}_p 的区别, \mathbb{Z}_p 中可能会出现不安全, 后续会介绍到.

2.5 (Pseudo)Random Number Generator

在构造一些方案时, 算法需要用到规定空间内的随机数 (random number) 进行计算, 例如:

- A random n-bit string from $\{0, 1\}^n$.
- A random integer from \mathbb{Z}_p .
- A random element from such as \mathbb{G} .

random: 设 x 是一个随机变量, w_1 和 w_2 是取值空间中的随机的两个可能的取值, “random choose” 指的是 $Pr[x = w_1] = Pr[x = w_2]$.

key generation 经常要用到随机数当作私钥, 以达到 IND 安全.

在一些方案的算法中, 可以假设选择了满足真正相等概率的真随机数 (real random numbers), 而在现实世界中只能用选用伪随机数生成器生成的伪随机数 (pseudorandom numbers), 在安全规约时, 我们假定生成的随机数都是真随机的 (truly random). 密码学总是依赖于很多假设!!!

3 Lecture 3 Preliminaries(Hard Problem and Secure Scheme)

Outline:

- 1. Preliminaries
 - Computational and Decisional
 - Deterministic and Probabilistic
 - Polynomial and Exponential
 - Negl. and Non-Negl.
 - Probability and Advantage
- 2. Hard Problem
 - Hardness Definition
- 3. Secure Scheme
 - Examples
 - Hardness Analysis
 - Security Parameter
 - Security Level
 - Definition of Security
 - Definition of Hard Problem(Hardness)

3.1 Preliminaries

3.1.1 Computational and Decisional

Computing Problem: 定义在某种数学原语上的数学对象，以一种问题和答案的形式，每一个 Computing Problem 都有无穷多对 instance-solution pairs (x,y) , 其中 x 是 instance (问题实例), y 是 solution (问题实例对应的解). 对于大部分问题，随着 x 长度的增加，求解 y 的计算代价和时间消耗都会增加，在密码学中，我们一般考虑 instance 长度相同的 instance-solution pairs (有限多对) .

Computing Problems = Computational Problems + Decisional Problems. 在一些文献中会用 Computational 表示 Computing，这里为了防止混淆用 Computing.

Computing Problem 通常会用“关系函数” $R(x,y)$ 进行严格定义：当 (x,y) 是一对正确的 instance-solution pair, 令 $R(x,y)=1$, 否则 $R(x,y)=0$, 即 y 不是 x 的解. 但是要注意，对于一个 x , 我们允许有多个不同的 y 当作 x 的答案，这和普通的 $y=f(x)$ 不同.

例：整数分解是 Computing Problem，而在具体选择整数进行分解时有无穷多个选择，即有无穷多个 instance-solution pair，在密码学中为了达到一定的安全性，我们选取固定长度的整数（太短不安全，太长没必要），即有限个 instance-solution pair.

Computational Problem In a computational problem, the solution y to the problem instance x is from a large space (as large as the space of x). For example, the space size is up to $2^{|x|}$ (随便举的例子，这里 $|x|$ 指的 x 的比特长度).

例如 CDH problem: instance $x = (g, g^a, g^b) \in G$; solution $y = g^{ab} \in G$.

computational problem = search problem = function problem.

Decisional Problem In a decisional problem, The solution y to the problem instance x is from the set $\{0, 1\}$, $y = 1$ 表示 x 是一个正确的实例 (true instance), $y = 0$ 表示 x 是一个错误的实例 (false instance). 这样将所有的问题实例和解分成了 true 和 false 两大类.

例如 DDH problem: instance $x = (g, g^a, g^b, Z) \in G$; solution $y = \begin{cases} 1 & Z = g^{ab} \\ 0 & Z \neq g^{ab} \end{cases}$

Formal Language The set containing all true instances is a Formal Language L and the decisional problem is to decide whether $x \in L$. 可以理解为将一些字符按某一类规则组合得到的集合，P 和 NP 问题的很多定义都是采用 Formal Language 的形式.

3.1.2 Deterministic and Probabilistic

算法可以根据其计算结果分为确定性算法 (deterministic algorithms) 和概率性算法 (probabilistic algorithms) 两大类.

- deterministic algorithms : 给定一个输入的 problem instance x , 返回一个正确的答案 solution y , 无论运行多少次, 只要输入相同, 输出就必定相同, 而且都是正确答案.
- probabilistic (randomized) algorithm : 给定一个输入的 problem instance x 以一定的可能性 (likelihood) 输出正确答案.
- 解决一些解决困难问题 (攻破安全方案) 时, probabilistic algorithms 比 deterministic algorithms 更有效, 目前也无法给出解释. (例如在素数测试中, 目前效率最高的算法是概率性算法, 而没有找到和它一样有效的确定性算法, 当然也有可能未来找到了这样的确定算法, 目前也无法证明概率算法一定比确定算法更有效).

概率算法中, 用 (t, ϵ) 表示一个算法运行时间 t 后一定会停止, 并输出一个答案, 这个答案是正确答案的概率是 ϵ . 密码学中常用 ϵ 表示概率或者优势, 取决于语境.

Algorithms Classifications 密码学中所有的行为 (construct scheme, break scheme, solve problem, and program reduction) 都是为了提出算法, 算法可以粗略分为下面四类, 当然这个分类也只是帮助我们理解安全归约 (security reduction), 更清楚的理解算法的运行结构, 分类会有交叉, 不需分的太清.

- Scheme Algorithm. To construct a cryptosystem (scheme).
- Attack Algorithm. To break a scheme.
- Solution Algorithm. To solve a hard problem.
- Reduction Algorithm. To describe a security reduction.

3.1.3 Polynomial and Exponential

Let $f(\lambda)$ be a function in positive number λ .

- Polynomial: exists $c, k > 0$ such that $f(\lambda) \leq c\lambda^k$, 上限的定义.
- Exponential: exists $c > 0$ such that $f(\lambda) \geq 2^{c\lambda}$, 下限的定义.

注: 在密码中暂时只考虑多项式曲线以下和指数曲线以上两种, 例如时间消耗 (time cost) 是 polynomial time 还是 exponential time 等. Polynomial time should be exactly understood as “polynomial speed”, 是一个速度, 而不是某一个具体数值. e.g. 2^{160} is not an exponential time but 2^λ is!

我们希望进行分界, 合法的用户的计算时间在“多项式时间”以下, 而敌手攻破方案的时间在“指数时间”以上.

3.1.4 Negligible and Non-Negligible

函数 $\text{negl}(\lambda)$ 是一个可忽略函数 (negligible function), 如果满足:

1. $\text{negl}(\lambda) = \frac{1}{f(\lambda)}$.
2. 不能用任何的多项式限制 $f(\lambda)$ (super-polynomial), 换句话说, 不能用任何的多项式盖住 $f(\lambda)$.
3. negligible 的意思是随着多项式的增加, 以指数级的速度接近于 0.

函数 $\text{non-negl}(\lambda)$ 是一个不可忽略函数 (non-negligible function), 如果满足:

1. $\text{non-negl}(\lambda) = \frac{1}{f(\lambda)}$.
2. $f(\lambda)$ 是一个关于 λ 的多项式.
3. $1/99999999999999999999999999$ 是 non-negligible.
4. non-negligible = noticeable.

3.1.5 Probability and Advantage

Probability 是一个事件 (event) 发生的可能性 (likelihood), event 主要包括 given a problem instance, output a problem solution (in computing problem) or a successful attack on a scheme (in breaking a scheme).

之所以引入概率, 是因为敌手的 solution algorithm/attack algorithm 是一个概率性算法, 而不是确定性算法. 而目前概率性算法比确定性算法更有效, 因此不得不采用概率性算法来分析, 或许未来某天能证明确定性算法和概率性算法的等价性, 就会对密码学的基础和定义产生巨大的变化和影响.

概率会随着问题的不同而变化, solving different hard problems 或者 breaking different secure schemes 的概率值都是不一样的, 例如:

- Digital Signatures. $Pr[Win_{Sig}]$ 是指敌手成功伪造一个签名的概率值, 满足 $0 \leq Pr[Win_{Sig}] \leq 1$, 0 是因为敌手随便输入一个 random string, 只要在签名空间中, 就有可能是一个有效的签名.
- Encryption. $Pr[Win_{Enc}]$ 是指敌手能正确猜中一个加密后的消息的概率值, 如果在 IND 安全的模型下 (indistinguishability security model for encryption), 即给定一个密文, 敌手知道对应的消息是 m_0 或 m_1 并进行猜测, 则有 $\frac{1}{2} \leq Pr[Win_{Enc}] \leq 1$.

对于同一个困难问题, 不同的 event 的概率的范围也可能不同, 以一个 Decisional Problem 为例: 设 $Pr[Win_D]$ 是猜中 DDH 问题实例 (g, g_a, g_b, Z) 中的 Z 的概率.

- 假设 Z 是从一个“含有两个元素的空间”中随机选取的, 其中一个元素是 true($= g^{ab}$), 另一个元素是 false($\neq g^{ab}$), 则猜中的概率 $\frac{1}{2} \leq Pr[Win_D] \leq 1$.
- 假设 Z 是从一个“含有 p 个元素的群 G ”中随机选取的, 其中只有一个元素是 true($= g^{ab}$), 则有 $1 - \frac{1}{p} \leq Pr[Win_D] \leq 1$. 因为 G 中只有一个元素等于 g^{ab} 且 Z 是随机选取的, 因此 false 的概率非常大, 猜测为 false, 则猜中的概率为 $1 - \frac{1}{p}$.

上述两点说明概率的范围不仅与困难问题相关, 还取决于困难问题的具体定义. 我们无法仅仅根据“概率值”就区分出对一个密码系统的攻击是成功的还是失败的, 即使敌手能成功攻击一个方案的概率是 $\frac{1}{2}$ 甚至 $1 - \frac{1}{p}$, 仍不知道到底是真的产生威胁了还是没产生威胁.

因此, 引入 优势 (advantage) 来代替概率. 优势是一种调整后的概率, 最小的优势为 0, 如果 advantage 是 negligible, 则密码系统是安全的, 如果 advantage 是 non-negligible, 则密码系统是不安全的.

优势: 假设 P_{ideal} 是 breaking a secure scheme 或者 solving a hard problem 的最大的自然概率, 可以理解为不看方案和参数 (甚至不知道公钥长什么样), 直接猜中的最大概率 (就像上面的 $1 - \frac{1}{p}$), 这里是 trivial attack: 连公钥长什么样子都不知道就要对其进行攻击.

- If P_{ideal} is non-negligible, advantage = Probability of Successful Attack - P_{ideal} . (例如加密 IND 模型下的定义)
- If P_{ideal} is negligible, advantage = Probability of Successful Attack. (例如签名不可伪造模型下的定义)

注: non-negligible 中有时也会将优势定义为右边乘以 2. 即 $\text{advantage} = 2 \cdot (\text{Probability of Successful Attack} - P_{\text{ideal}})$, 乘以 2 后最大的 advantage 为 1, 能使加密、签名等各种安全模型的 advantage 统一, 满足任意的攻击的 advantage 都在 0 到 1 之间. 密码学中并不关心最大的 advantage, 只关心 advantage 是 negligible 还是 non-negligible.

引入 advantage 后, 可以得到在多项式时间内, 如果敌手的 advantage 是可忽略的, 则问题是困难的/方案是安全的, 如果敌手的 advantage 是不可忽略的, 则方案不安全.

3.2 Hard Problem

3.2.1 Hardness Definition

Computational Hardness Let x be a random instance of a computational problem. We say that the computational problem is **hard** if for any probabilistic polynomial time (PPT) adversary A , $\Pr[A(x) = y] = \epsilon(\lambda)$, 即 “敌手将给定的输入 x 输入到算法 A 中, 输出 y ” 这个事件发生的概率.

- 敌手运行概率性算法是因为概率性算法比确定性算法更有效.
- 所有公钥密码方案都只考虑多项式时间安全, 不考虑指数时间安全, 所以只考虑多项式时间的敌手.
- λ 是 problem instance x 的长度. $\epsilon(\lambda)$ 是一个关于 λ 的可忽略函数, 是针对所有可能出现的 x, y 的平均统计得到的值, 不是针对具体的 x, y .

Decisional Hardness Let x be a random instance of a decisional problem. We say that the decisional problem is **hard** if for any probabilistic polynomial time adversary (PPT) A , $|\Pr[A(x) = \text{true}|x = \text{true}] - \Pr[A(x) = \text{true}|x = \text{false}]| = \epsilon(\lambda)$, 当 x 是一个 true instance 时敌手判断 x 是 true (敌手猜中) 和当 x 是一个 false instance 时敌手判断 x 是 true (敌手未猜中) 的概率之差.

- 敌手运行概率性算法.
- 只考虑多项式时间的敌手.
- λ 是 problem instance x 的长度. $\epsilon(\lambda)$ 是一个关于 λ 的可忽略函数.

例: $x = (g, g^a, g^b, Z)$

- $\Pr[A(x) = \text{True}|x = \text{True}]$ is the probability of correctly guessing Z on the condition that $Z = g^{ab}$.
- $\Pr[A(x) = \text{True}|x = \text{False}]$ is the probability of wrongly guessing Z on the condition that $Z \neq g^{ab}$.

$$\text{advantage} = |\Pr[A(x) = \text{True}|x = \text{True}] - \Pr[A(x) = \text{True}|x = \text{False}]| = \epsilon(\lambda)$$

选用该定义的好处是无论 Z 取什么值, advantage 具有一致性, 能够包含所有的 decisional problem. 不管 Z 是从两个元中选取, 还是从一个集合 G 中选取, 虽然概率值不一样, 但是 advantage 的定义都包含了, 只要 advantage 是可忽略的, 那么该 decisional problem 就是困难的.

Hardness Assumption The concepts of hard problem and hardness assumption are equivalent. However, the descriptions of these two concepts are slightly different.

- 固定搭配 “solve a problem”, “break a assumption” .
- We can say that “breaking a proposed scheme implies solving an underlying hard problem”, “Therefore, the scheme is secure under the hardness assumption” .

- We can say that “a hardness assumption is a weak assumption or a strong assumption”. 这里说的 weak 和 strong 不是和问题相关，而是和困难性的强度相关 (the strength of the hardness)，同时 weak 和 strong 也具有相对性，对用户好还是对敌手好，要结合语境和对象.

All hardness assumptions can be classified into weak assumptions and strong assumptions, but the classification is not very precise.

- Weak assumptions = a computing problem is hard if and only if a few conditions hold. 只需要满足一两个条件就能使这个假设是困难的.
- Strong assumptions = a computing problem is hard if and only if lots of conditions hold. 要满足很多条件同时成立，这个假设才是困难的，有一个不成立就不是困难的.
- Breaking a strong assumption is easier than breaking a weak assumption.
- 困难问题的实例的强度也与敌手询问的次数有关.

	Good to us	Bad to us
Hardness Assumption	Weak Assumption	Strong Assumption
Computational Model	Strong Model	Weak Model

- weak Hardness Assumption is good to us, strong is bad to us.
- strong Computational model is good to us, weak is bad to us.
- 对于用户来说，一个在“弱困难假设”和“强的计算模型”下是可证明安全的方案是更好的. a scheme secure under a weak hardness assumption in a strong security model is better.

3.2.2 Examples

掌握困难问题是重要的，重要性详见密码学报那篇文章，很多密码方案功能越多性质越多，他的证明就相对复杂，就只能采用一些像 q-SDH 这样复杂的困难问题，才能做证明，比如广播加密. 当然这也是灌水文章的首选，一些功能比较复杂的密码方案，对应的困难问题也比较奇葩.

下面是一些问题实例，更多的例子参见书.

Computationally Hard Problems (1)

Let \mathbb{G} be the group from $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ unless specified otherwise.

Instance: $g, g^a \in \mathbb{G}$, where \mathbb{G} is a general cyclic group
Compute: a

Instance: $g, g^a, g^b \in \mathbb{G}$, where \mathbb{G} is a general cyclic group
Compute: g^{ab}

Instance: $g, g^a, g^{a^2}, \dots, g^{a^q} \in \mathbb{G}$
Compute: $(s, g^{\frac{1}{a+s}}) \in \mathbb{Z}_p \times \mathbb{G}$ for any s

Computationally Hard Problems (2)

Let \mathbb{G} be the group from $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ unless specified otherwise.

q -Strong Diffie-Hellman Inversion Problem (q -SDHI)
 Instance: $g, g^a, g^{a^2}, \dots, g^{a^q} \in \mathbb{G}$
 Compute: $g^{\frac{1}{a}}$

q -Bilinear Diffie-Hellman Inversion Problem (q -BDHI)
 Instance: $g, g^a, g^{a^2}, \dots, g^{a^q} \in \mathbb{G}$
 Compute: $e(g, g)^{\frac{1}{a}}$

q -Bilinear Diffie-Hellman Problem (q -SDH)
 Instance: $g, g^a, g^{a^2}, \dots, g^{a^q}, g^{a^{q+2}}, g^{a^{q+3}}, \dots, g^{a^{2q}}, h \in \mathbb{G}$
 Compute: $e(g, h)^{a^{q+1}}$

Decisionally Hard Problems (1)

Let \mathbb{G} be the group from $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ unless specified otherwise.

Decisional Diffie-Hellman Problem (DDH)
 Instance: $g, g^a, g^b, Z \in \mathbb{G}$, where \mathbb{G} is a general cyclic group
 Decide: $Z \stackrel{?}{=} g^{ab}$

Variant Decisional Diffie-Hellman Problem (Variant DDH)
 Instance: $g, g^a, g^b, g^{ac}, Z \in \mathbb{G}$, where \mathbb{G} is a general cyclic group
 Decide: $Z \stackrel{?}{=} g^{bc}$

Decisional Bilinear Diffie-Hellman Problem (DBDH)
 Instance: $g, g^a, g^b, g^c \in \mathbb{G}, Z \in \mathbb{G}_T$
 Decide: $Z \stackrel{?}{=} e(g, g)^{abc}$

Decisionally Hard Problems (2)

Let \mathbb{G} be the group from $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ unless specified otherwise.

Decisional Linear Problem
 Instance: $g, g^a, g^b, g^{ac_1}, g^{bc_2}, Z \in \mathbb{G}$
 Decide: $Z \stackrel{?}{=} g^{c_1+c_2}$

q -DABDHE Problem
 Instance: $g, g^a, g^{a^2}, \dots, g^{a^q}, h, h^{a^{q+2}} \in \mathbb{G}, Z \in \mathbb{G}_T$
 Decide: $Z \stackrel{?}{=} e(g, h)^{a^{q+1}}$

Decisional (f, g, F) -GDDHE Problem
 Instance: $g, g^a, g^{a^2}, \dots, g^{a^{n-1}}, g^{af(a)}, g^{b \cdot af(a)} \in \mathbb{G}$
 $h, h^a, h^{a^2}, \dots, h^{a^k}, h^{b \cdot g(a)} \in \mathbb{G}$
 $Z \in \mathbb{G}_T$
 $f(x), g(x)$ are co-prime polynomials of degree n, k .
 Decide: $Z \stackrel{?}{=} e(g, h)^{b \cdot f(a)}$

3.2.3 Hardness Analysis

前面都是介绍的困难问题的定义和实例，然而如何证明困难性呢？公钥密码学目前基于 $NP \neq P$ ，这是还没严格证明 $NP \neq P$ 的结论。如果我们能证明一个问题的困难的，那么就证明了 $NP \neq P$ 。

因此，公钥密码学中的困难问题都是“believed-to-be hard only”而不是真正困难的，只是密码学家都相信它困难。然而，我们也可以在某些条件下证明问题的困难性（给定一些额外条件或者用一些更强的假设）。

注：Some problems analyzed to be hard could be actually easy. (Because the analysis is wrong!) 卧报中有两个例子大家可以去看看，困难问题的设计并不容易，一不小心就会出错。

想要证明 computing problem A 的困难性，常见的方法包括：

- 归约：证明解决了问题 A 就解决了问题 B，而问题 B 是大家承认的困难问题 (solving problem A implies solving an existing problem B that is well believed to be hard, 下一章会详细介绍)。这里只是归约，安全归约比归约复杂得多。

- 弱化计算模型：在一个弱化敌手能力的模型中证明解决问题 A 是困难的（本来敌手可以做 12345678 八种计算，现在只能做 1234 四种计算，在这种情况下分析安全性）。One of well-known models is the generic group model for proving hard problems defined over a cyclic group.

注：这里是计算模型而不是安全模型，标准的计算模型是 Turing machine。

- membership proof：某一类的问题已经被别人（比如 Dan Boneh）证明是困难的，我们直接证明困难问题 A 属于这一大类中，如下图，具体的方案构造可能会用特殊的问题，但是它属于下面的集合，因此能用这个困难问题来构造一些多功能的比较花哨的方案（非常好用！），证明用这种困难问题非常容易达到，反而归约证明到 CDH, DDH 是非常困难甚至不可能的。

A Family of decisionally hard problem :

Decisional (P, Q, f) -GDHE Problem

Instance: $g^{P(x_1, x_2, \dots, x_m)} \in \mathbb{G}$, $e(g, g)^{Q(x_1, x_2, \dots, x_m)}$, $Z \in \mathbb{G}_T$
 $P = (p_1, p_2, \dots, p_s) \in \mathbb{Z}_p[X_1, \dots, X_m]^s$ is an s -tuple of m -variate polynomials
 $Q = (q_1, q_2, \dots, q_s) \in \mathbb{Z}_p[X_1, \dots, X_m]^s$ is an s -tuple of m -variate polynomials
 $f \in \mathbb{Z}_p[X_1, X_2, \dots, X_m]$
 $f \neq \sum a_{i,j} p_i p_j + \sum b_i q_i$ holds for $\forall a_{i,j}, b_i$

Decide: $Z \stackrel{?}{=} e(g, g)^{f(x_1, x_2, \dots, x_m)}$

3.3 Secure Scheme

3.3.1 Security Parameter

- key generation algorithm 输入 security parameter λ , 输出 a public key and a secret key.
- security parameter λ 是 input string 的长度 (bit length).
- security parameter can be roughly seen as instance length. 例如在基于群的密码中 security parameter λ 特指一些群元的长度比如 160bits 或者 1024bits.
- 哈希函数的 security parameter: The size n in $H : \{0, 1\}^* \rightarrow \{0, 1\}^n$ is a security parameter.
- 一般而言，随着 security parameter λ 的增加，攻破一个方案的困难性也在增加。例如 Hash Function 中，输出是 1024bits 比 256bits 更难找到碰撞。

注: security parameter \neq public key size. 前者可以理解为基本单元, 例如前者为 160bits, 那么后者(公钥)最短就是由若干个基本单元 160bits 构成, 不可能是 150bits or 120bits.

3.3.2 Security Level

用来表示敌手 breaking a scheme or solving a problem 的强度 (strength), 主要指的是时间消耗.

security level: λ is security parameter, we have $f(\lambda)$ is called security level. 这里的 f 与方案或者问题的定义相关, 因此 security level 取决于 security parameter λ 和具体的方案以及问题的定义.

k-bit security: a scheme or a problem has k-bit security if an adversary must take 2^k steps/operations to break the scheme or to solve the problem (with success probability at least $2/3$). 详见卧报 34 期.

注: security parameter 为 λ 的方案一般不能达到 λ -bit security. 事实上, 当一个方案的 input 是 λ -bit 时, 大多数情况下能达到 $\frac{\lambda}{2}$ -bit security 就不错了, 这里指的是公钥密码, 对于 AES 那种对称密码 128bits 的密钥可以达到 128bits 的 security level.

Security Level (from the view of attack algorithm): 对于一个提出的方案, 目前能找到的所有的 attack algorithms 分别需要的 time cost 和 advantage 为 $(t_1, \epsilon_1), (t_2, \epsilon_2), \dots, (t_l, \epsilon_l)$.

k-bit security: For a simple analysis, we say that the proposed scheme has k-bit security if $2^k = \min\left\{\frac{t_1}{\epsilon_1}, \frac{t_2}{\epsilon_2}, \dots, \frac{t_l}{\epsilon_l}\right\}$.

安全等级是指用已知的攻击方案对其攻击得到的安全等级, 而不是最终真实安全等级, 有可能有新提出的方案使得 $\frac{t_{l+1}}{\epsilon_{l+1}}$ 更小, 那么安全等级就随之下降.

Security Level (Upper Bound): Suppose a scheme is constructed over a cyclic group.

- Solving the Discrete Log problem can definitely break the scheme.
- The security level of scheme is not higher than DL problem.
- The discrete log hardness is the upper bound security level.

Summary: If solving problem B can immediately help breaking the scheme, the hardness of problem B is the upper bound security of scheme.

Security Level (Lower Bound): Suppose there is a scheme and a security reduction.

- Suppose breaking the scheme implies solving a hard problem A.
- The security level of scheme is “higher” than the problem A.
- The hardness of problem A is the lower bound security of scheme.

注: “higher” has some conditions that will be explained later when introducing concrete security.

Security Level summary Suppose there is a scheme.

- The known security level is based on all known attack algorithms. 基于现有的攻击
- The exact security level is based on the best attack algorithm (it is hard to find the best one. Otherwise, proving $NP \neq P$ is easy). Therefore, the exact security is usually unknown. 最终准确的 security level, 必须用最好的攻击算法才能计算出, 但目前无法证明 $P \neq NP$, 找不到 best attack.
- The upper bound security level is based on a computing problem harder than breaking the scheme, such as the DL problem. 例如方案的安全性达不到离散对数 \rightarrow 只需要分析离散对数的安全性级别.

- The lower bound security level is based on the underlying hardness assumption adopted in security reduction. 攻破方案归约到解决另一个困难问题，例如 QHDS，那么 QHDS 的困难性就是方案的安全级别下限。

一般情况下，密码学家对方案的安全下限 Lower Bound 比较感兴趣，因为一个方案的安全必须是对所有的对手安全，要考虑最聪明最狡猾能力最强的对手，能使安全性最低，这样做的密码方案才有意义。

3.3.3 Definition of Security

我们称一个安全参数为 λ 的方案在某个安全模型中是安全的 (Security)，如果对于所有的 PPT 敌手 A ，都只有 $\epsilon(\lambda)$ 的优势攻破该方案，其中 $\epsilon(\lambda)$ is a negligible function in λ .

Security and Security Model

- 安全模型描述了“敌手能力”和“安全目标”，对于不同的安全模型，攻击同一个方案的困难性是不同的，例如：
 - Easy: forge a signature with a secret key.
 - Hard: Forge a signature without knowing public key.
- 定义 advantage/定义一个方案安全/讨论一个方案的安全性时必须先清楚安全模型，安全模型是“密码方案”和“计算困难问题”之间的桥梁：把对密码方案的攻击转化成计算问题，再利用计算复杂性理论的方法进行分析，参考卧报 29.

Security and Security Level Security definition doesn't use the concept of security level. security level is used to understand the concrete security of scheme when λ is set with a concrete number. “安全等级”是用来理解当“安全参数” λ 设定为具体的数时的具体方案的安全性。

3.3.4 Definition of Hard Problem(Hardness)

我们称一个 computing problem 是困难的 (Hard Problem/Hardness)，如果对于所有的 PPT 敌手 A ，都只有 $\epsilon(\lambda)$ 的优势攻破该方案，其中 $\epsilon(\lambda)$ is a negligible function in λ and the instance length is λ .

注：

- 这里的 λ 指 instance length 而不是 security parameter, security parameter 仅被用在密码学中，与 computing problem 无关，描述 computing problem 时也不能把 security parameter 放在问题中。
- Computing problem 本身并没有安全模型，因此上述困难问题的定义对 computation problem 和 decisional problem 都是合适的。注意到这里用的 advantage 而不是 probability，一致性！

4 Lecture 4 Entry to Security Reduction

Outline:

- 1. Reduction in Computational Complexity
- 2. Security Reduction in Cryptography
 - Overview
 - Framework
 - From Breaking to Solving
- 3. Evaluation of Security Reduction
 - Cost and Loss in Reduction
 - Concrete Security
 - Ideal Security Reduction

终于到了安全规约的入门部分！

4.1 Reduction in Computational Complexity

在计算复杂性理论中，如果没有一些额外的假设，我们无法证明某一类计算问题是困难的（因为我们对最基本的 P 和 NP 问题是否等价都不清楚）。但是在给定一些假设后，例如已知某些问题是困难的，我们就可以通过归约 Reduction（反证法的思想，Proof by Contradiction）来做一些困难性的证明。

证明思想：The problem A is reducible to the problem B.

1. A computing problem A is believed to be hard.
2. If problem B is easy, we prove that problem A is also easy. 矛盾！
3. The assumption is then false, and the problem B must be hard.

The problem A is reducible to the problem B. 即 A 可以归约到 B，记作 $A \leq_p B$ ，表示解决了问题 B 就可以解决问题 A，A 比 B 简单 / B 比 A 困难。

具体过程：

1. 给定 problem A 的一个 problem instance x_A .
2. 将 problem instance x_A 转化成 problem instance x_B .
3. 求解 problem solution y_B 并转化成 problem solution y_A .
4. 从而解决了问题 B 就可以解决问题 A.

归约主要是研究一个新问题的困难度下限：对于一个不知道难度的新问题，证明其不会比一个已知难度的问题简单。

Problem A: $x_A = (g, g^a, g^b)$, $y_A = g^{ab}$

Problem B: $x_B = (g, g^c, g^d)$, $y_B = g^{c^2+cd}$

We have:

The problem A is reducible to the problem B
假如有一个敌手能帮我们解决B,

我们就能利用它帮我们解决A

Question: Do you know how to prove this?

Problem A: $x_A = (g, g^a, g^b)$, $y_A = g^{ab}$

Problem B: $x_B = (g, g^c, g^d)$, $y_B = g^{c^2+cd}$

Proof. Given the instance (g, g^a, g^b) , we work as follows.

- 1 Compute $\frac{g^b}{g^a} = g^{b-a}$.
- 2 Set $g^c = g^a$
- 3 Set $g^d = g^{b-a}$
- 4 Send the problem instance $(g, g^a, g^{b-a}) = (g, g^c, g^d)$ to algorithm \mathcal{B} that can compute its problem solution g^{c^2+cd} .

We have

$$g^{c^2+cd} = g^{a^2+a(b-a)} = g^{a^2+ab-a^2} = g^{ab},$$

which is the solution to problem A. This completes the proof. \square

Problem A: $x_A = (g, g^a, g^{a^2}, \dots, g^{a^q})$, $y_A = (c, g^{\frac{1}{a+c}})$ for any $c \in \mathbb{Z}_p$.

Problem B: $x_B = (g, h^{\frac{1}{a}}, h, h^a, \dots, h^{a^{q-1}})$, $y_B = (c, g^{\frac{1}{a+c}})$ for any $c \in \mathbb{Z}_p$.

We have:

The problem A is reducible to the problem B

Question: Do you know how to prove this?

Problem A: $x_A = (g, g^a, g^{a^2}, \dots, g^{a^q})$, $y_A = (c, g^{\frac{1}{a+c}})$ for any $c \in \mathbb{Z}_p$.

Problem B: $x_B = (g, h^{\frac{1}{a}}, h, h^a, \dots, h^{a^{q-1}})$, $y_B = (c, g^{\frac{1}{a+c}})$ for any $c \in \mathbb{Z}_p$.

Proof. Given the instance $(g, g^a, g^{a^2}, \dots, g^{a^q})$, we work as follows.

- 1 Randomly choose $w \in \mathbb{Z}_p$ and set $h = (g^a)^w$.
- 2 Send the problem instance

$$(g, h^{\frac{1}{a}}, h, h^a, \dots, h^{a^{q-1}}) = (g, g^w, g^{wa}, \dots, g^{wa^q})$$

to algorithm \mathcal{B} that can compute its problem solution $(c, g^{\frac{1}{a+c}})$.

- 3 We have the output is also the solution to problem A.

This completes the proof. \square

只有 randomly choose w 才能保证 xB 是从 instance space 中随机选取，要保证每个 instance 都是随机选取的

4.2 Security Reduction in Cryptography

4.2.1 Overview

Security reduction is similar to reduction in computational complexity.			
Reduction	Problem	Reducible to	Problem
Security Reduction	Problem	Reducible to	Scheme
	解决问题	比	攻破方案 简单

证明思想：The problem P is reducible to the scheme.

1. A computing problem P is believed to be hard.
2. If a proposed scheme is insecure, we prove that the problem P is easy. 矛盾！
3. The assumption is then false, and the scheme must be secure.

security reduction 和 security analysis 的区别: security analysis 更倾向于具体的攻击方法（例如重放攻击、中间人攻击等），得到的安全性只是基于已知的攻击算法，没考虑尚未得到/尚未公开的攻击算法。而 security reduction 包括了所有的攻击方法，只要能被安全模型 capture.

Security Reduction (Proof Strategy)

Recalling that

$$\begin{array}{l} \text{problem } A \text{ is reducible to problem } B \\ \Downarrow \\ \text{problem instance } x_A \text{ can be used to create problem instance } x_B \end{array}$$

In security reduction,

$$\begin{array}{l} \text{a problem } P \text{ is reducible to proposed scheme} \\ \Downarrow \quad \text{只 simulate 了结果的值, 而不是整个方案} \\ \text{problem instance } x_P \text{ can be used to "simulate" a scheme} \\ \Downarrow \quad \text{simulate what the adversary needs} \\ \text{breaking the simulated scheme can be transformed to solution } y_P \\ \text{敌手能用攻破 simulated scheme 的方法 得到问题 P 的解 } y_P \end{array}$$

4.2.2 Framework

Security Reduction 沿着下面的框架：

Breaking Assumption: 在证明之前，首先假设存在一个敌手能攻破方案（在规定的安全模型中）

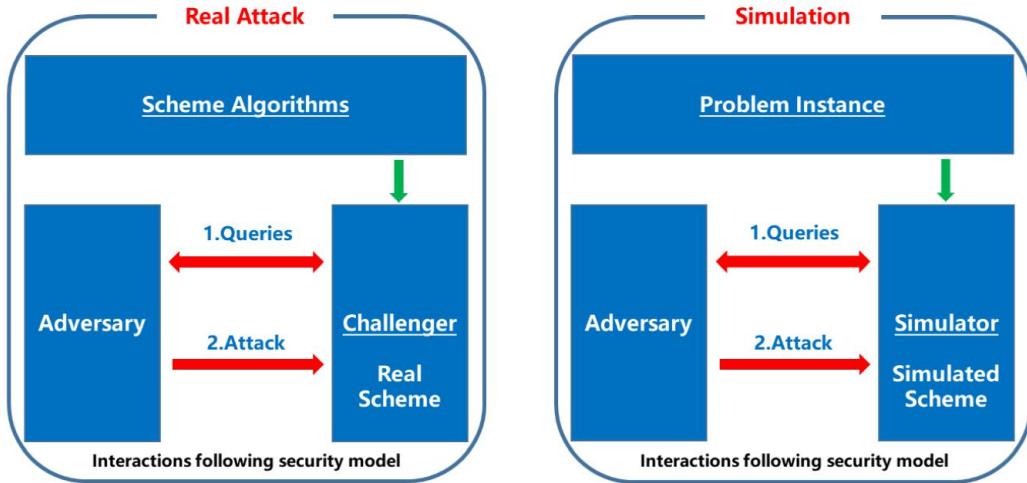
1. Simulation. 用 problem P 的一个 instance x_P 生成 simulated scheme，然后和敌手交互（按安全模型的定义，敌手询问允许询问的东西，就给敌手）。
2. Solution. 从敌手对 simulated scheme 的 attack 抽取出 problem solution y_P .
3. Analysis. 计算解决困难问题 P 的优势 advantage，如果是 non-negligible，则成功攻破了假设。

Question: Can we program a security reduction without analysis? 部分早期或者新手的论文并没有“Analysis”，没有“Analysis”的安全归约证明不完整！安全归约并不是数学上的严格的形式化证明，只是提出了一个归约算法，然后指出如何用归约算法把“敌手对方案的攻击”转化为“解决困难问题的能力”。

安全归约的“矛盾”：我们无法说明一个安全归约是能进行的，因为安全归约的过程必须有敌手配合，而敌手不一定是真实存在的。如果存在敌手，方案就直接不安全了。如果不存在敌手，归约算法无法执行。因此我们仅仅做理论上的分析：如果存在敌手，就一定能用来解决困难问题。

如果这个宇宙存在孙悟空，我就能证明存在外星人。

我，只证明了关系，而没办法说明外星人存在还是不存在。



Concepts in real attacks and simulation.

Real Scheme	vs	Simulated Scheme
Challenger	vs	Simulator
Real attack	vs	Simulation

Real Scheme vs Simulated Scheme

- Real Scheme 是将 security parameter 作为输入，运行方案算法 (scheme algorithm) 得到的一套具体的公钥私钥等参数。
- Simulated Scheme 是用一个 problem instance 根据归约算法 (reduction algorithm) 模拟得到的，problem instance is a random instance of an underlying hard problem. 归约算法怎么定义，Simulated Scheme 就怎么产生。

Challenger vs Simulator

- 真实方案中：The adversary interacts with **real scheme** controlled by the **challenger**。
- 安全证明中：The adversary interacts with **simulated scheme** controlled by the **simulator**。
- challenger 和 simulator 只是方便交互用的代号，**challenger** 只出现在攻击真实方案的 **security model** 中，**simulator** 只出现在安全证明的 **security reduction** 中。不同的场景请用不同的 entities 描述。

Real attack vs Simulation

- The **real attack** (上图中左边的房间，现实场景) is the interaction between the adversary and the **challenger**。根据安全模型提供敌手需要的信息。
- The **simulation** (上图中右边的房间，模拟场景) is the interaction between the adversary and the **simulator**。根据安全模型提供敌手需要的信息。

Indistinguishable

- From the view of the adversary, the responses by **real scheme** and **simulated scheme** could be indistinguishable (looks the same). 理解为将敌手的眼睛遮住, 丢到上面的左边和右边两个房间之一, 敌手与 challenger/simulator 进行交互, success simulation 使得敌手分不清对方到底是谁.(indistinguishable)
- The responses by **real scheme** is following **scheme algorithm** and the response by **simulated scheme** is following **reduction algorithm**. 除此以外, 归约算法并不需要模拟整个 **scheme algorithm**, 只用模拟出敌手需要的计算结果. 也可能只需要模拟部分算法的结果 (例如证明加密方案的 CPA 安全, 可以不模拟解密算法的结果), 具体需要模拟哪些算法取决于安全模型.

解释 indistinguishable 的一个小例子:

Approach 1:

Randomly chooses a from $\{0, 1, \dots, 9\}$ and computes $b = a + 1 \bmod 10$

Approach 2:

Randomly chooses a from $\{0, 1, \dots, 9\}$ and computes $b = a + 7 \bmod 10$

Question: Given an integer b , can we know it is computed from Approach 1 or Approach 2 ?

不能判断b是通过哪一种方式得到的: indistinguishable
注意 randomly chooses a 很重要,

反例:选取 a 时 $a=1$ with probability 99%, 判断 b 是通过方式1还是方式2选取时就有优势了: 方式1 $b=2$ 的概率非常大, 方式2 $b=8$ 的概率非常大, $b=2$ 则猜方式1, $b=8$ 则猜方式2, 虽然也有猜错的可能, 但是优势是"不可忽略的".

4.2.3 From Breaking to Solving

security reduction 的目标是将敌手的攻击 reduce(转化) 为解决困难问题. 攻击可以是 computational attack 或者 decisional attack.

- A **computational attack**, such as forging a valid signature, requires the adversary to find a correct answer from an exponential-size answer space.
- A **decisional attack**, such as guessing the message $m_b \in \{m_0, m_1\}$ in the challenge ciphertext in the IND-CPA security model, only requires the adversary to guess b (0 or 1).
- 注: Security against a **decisional attack** = **Indistinguishability** security.

Three Types: 根据攻击的类型, 将归约转化为不同类型的计算问题.

	Attack		Hard Problem
Type 1	Computational Attacks	→	Computational Hard Problems
Type 2	Decisional Attacks	→	Decisional Hard Problems
Type 3	Decisional Attacks	→	Computational Hard Problems

Note: The third type is very special because it is only available in the random oracle model, where the simulator uses hash queries made by the adversary to solve a computational hard problem.

4.3 Evaluation of Security Reduction

4.3.1 Cost and Loss in Reduction

Reduction Cost and Reduction Loss

Suppose there exists an adversary who can (t, ϵ) -break the scheme.

Generally speaking, we solve an underlying hard problem with

$$t : \text{多项式时间} \quad t' = t + T, \quad \epsilon' = \frac{\epsilon}{L}.$$

ϵ : 不可忽略的优势

- T is *reduction cost* (simulation time). We have

归约的时间消耗.

$$T = O(q)$$

目前已知的安全归约
中都是 $O(q)$, 没办法
降到1或者常数

T 和 L 的范
围是由已
知的安
全规
约的特
征进行总
结得到的.

- L is *reduction loss* (success probability $\frac{1}{L}$). We have

并不是敌手的每一次成功的攻击

都能转化为解决困难问题的能力,

中间可能失败, L 次成功一次.

有很多特殊的安
全归约能让 Loss 和询问次
数 q 无关, 恒为 1, 2 等

Here, q is the number of queries, such as signature queries or hash queries or private key queries.

Reduction 可以根据 reduction loss 分为 Tight Reduction 和 Loose Reduction:

- Tight Reduction: $L << q$, 例如 $L = O(1)$, $L = O(\log q)$.
- Loose Reduction: $L = O(q)$.

注: 只按照 L 进行分类, 而不按照 T 进行分类是因为目前所有已知的安全归约中的时间消耗 $O(q)$ 都消不掉, 所以我们暂时不管... 或许未来找到了时间消耗恒为 $O(1)$ 的归约, 则开辟了新的研究方向.

Tight Reduction: Necessary or Not?

- A tight reduction is better than a loose reduction.
- How to program a tight reduction is a hot research topic in the Theory of Cryptography research. However, Applied cryptography (e.g. cloud computing security, IoT Security) doesn't care loose reduction too much.
- A scheme with tight reduction is usually less efficient in computations than a scheme with a loose reduction. 因此在寻找计算效率高的方案时不应该考虑 tight reduction.
- IEEE Trans 的文章通常 motivation 不要是 Tighter reduction. 理论的点不适合于应用层次的期刊.

4.3.2 Concrete Security

Recall: 安全级别下限 Lower Bound Security Level

Suppose breaking the scheme S can be reduced to solving an underlying hard problem B , The lower bound security level of the scheme S is calculated from the underlying hard problem B . However, lower bound security level of $S \neq$ Security level of the problem B . 攻破方案 S 能解决困难问题 B , 则 S 的安全级别下限可以用 B 计算, 然而具体的安全级别 (是否取等) 要取决于 reduction cost 和 reduction loss.

Suppose the hard problem B has k -bit security.

Suppose the scheme S can be broken with (t, ϵ) .

Suppose a reduction solves the hard problem B with

假如存在一个安全归约能够把敌手的攻击转化为解决困难问题B

$$t' = t + T, \quad \epsilon' = \frac{\epsilon}{L}.$$

We have

$$\frac{t+T}{\frac{\epsilon}{L}} \geq 2^k$$

We obtain

$$\frac{t}{\epsilon} \geq 2^{k-\log L} - \frac{T}{\epsilon}.$$

which is the **concrete security** of scheme via formal reduction.

- 当 L 和 T 很大的时候, 方案 S 的 Lower Bound Security Level ($\frac{t}{\epsilon}$ 的下界 $2^{k-\log L} - \frac{T}{\epsilon}$) 很小, 失去意义.
- 如果能找到一个更好的归约 (L 和 T 更小), 方案 S 的 Lower Bound Security Level 将变得很大.
- 在安全归约时希望 L 和 T 越小越好, 小到 $2^{k-\log L} - \frac{T}{\epsilon}$ 无限接近 2^k , 那么方案 S 的安全级别就和困难问题 B 的安全级别一样了.

通过安全归约, 我们能证明方案的 concrete security 至少是 (for example) $2^{\lambda-30}$, 并不能说 the concrete security is $2^{\lambda-30}$. 这个值只是根据我们提出的安全归约方法得出来的, 有可能有更好的安全归约方法, 就能提高这个下限. 具体的例子参见书 P54 下方.

4.3.3 Ideal Security Reduction

An ideal security reduction (完美的安全归约) is the best security reduction that we can program for a proposed scheme.

- Security Model. The security model that allows the adversary to maximally, flexibly, and adaptively make queries to the challenger and win the game with a minimum requirement. 最强的安全模型, 前面已经提到过.
- Hard Problem. The underlying hard problem adopted must be the hardest one among all hard problems defined over the same mathematical primitive. For example, the DL problem. 即 weak hardness assumption.
- Reduction Cost and Reduction Loss. The reduction cost T and the reduction loss L are the minimized values 越小越好. $T = O(q), L = 1, q$ 是敌手查询的次数.
- Computational Restrictions on Adversary. There is no computational restriction on the adversary except time and advantage (归约的开始: 假如存在一个敌手能以时间 t 和不可忽略的优势 ϵ 攻破方案, 除了时间和优势外对敌手没有限制). 在一些特殊的证明中, 如在 random oracle model 中可能会 restrict the adversary 不能计算哈希值, 要得到哈希值只能通过 Oracle 得到.

Unfortunately, an inherent trade-off among these ideal features is very common in all security reductions proposed in the literature.

- For example, we can construct an efficient signature scheme whose security is under a weak hardness assumption, but the security reduction must use random oracles. 如 BLS 签名方案.
- For example, we can also construct a signature scheme without random oracles in the security reduction, but it is accompanied with a strong assumption or a long public key.

Currently, it seems technically impossible to construct a scheme with an ideal security reduction satisfying all four features mentioned above. how to argue your contribution where there is a trade-off then is very important!

Let us create a human-like AI Robot with one line code only!

5 Lecture 5 Difficulties in Security Reduction

Outline:

- | | |
|--|---|
| 1. Understanding Security Reduction | 3. Adversary |
| 2. Simulation and After Simulation | <ul style="list-style-type: none">• From Black-Box Adversary to Malicious Adversary• Malicious Adversary and Useful Attack• Adversary's Computing Ability• What Adversary Knows and Never Knows• Summary of Adversary |
| <ul style="list-style-type: none">• Something Unknown• Simulation Results• Classification of Attacks• Relations | |

5.1 Understanding Security Reduction

安全归约始于 breaking assumption: 假如存在一个敌手能攻破提出的方案 (在安全模型中). 则对于给定的 problem instance x_p , 我们 (证明者) 能构造一个 simulator, simulator 可以产生一个 simulated scheme 并利用敌手对 simulated scheme 的攻击计算出 y_p , 即解决困难问题.

矛盾: 敌手能攻破 real scheme, 我们 (证明者) 想要敌手攻破 simulated scheme, 如何让敌手对 simulated scheme 感兴趣? 这就是安全归约的重点和难点.

Proof. Suppose there exists an adversary who can break the proposed scheme (in a security model). Given a problem instance x_p , we can construct a simulator to generate a simulated scheme and use the adversary's attack on the simulated scheme to compute y_p .

⋮

- The adversary is able to break the proposed (real) scheme.
- The adversary is asked to break the simulated scheme.

Silly, honey bee likes flowers but not the plastic one!
how to make bee interested in plastic flowers?—安全归约



The adversary is attacking a given scheme $\in \{\text{real scheme, simulated scheme}\}$.

1. 假设敌手能攻破任何给定的方案 (在安全模型中), 如果这个方案 looks like a real scheme.
 2. 在安全归约中, given scheme 是一个 simulated scheme, 即敌手在和一个 simulated scheme 交互, 我们想要敌手相信这个方案是 real scheme, 并攻破该方案, 然后利用敌手的攻击来解决困难问题.
- 当 given scheme 看起来不像 real scheme 时: 我们不知道“如果敌手知道自己攻击的方案是一个 simulated scheme (而不是 real scheme) 时, 是否还会有相同的优势”(敌手是否还有一样的热情和态度去攻击这个 simulated scheme)——不知道攻击态度.
 - 当 given scheme 看起来像 real scheme 时: 我们也不知道当给出的 simulated scheme 在敌手看来 looks like a real scheme 时, 敌手会怎么攻击这个“real scheme”——不知道攻击过程.



Adversary

My IQ is up to 186.

My interest is breaking schemes.

You want me to help you solve problem?

我们 想利用敌手来解决困难问题

Fool me first!

但是敌手的兴趣是攻破方案，
所以我们必须先“欺骗”敌手。

对安全归约的理解：敌手是一个只对“攻破方案”感兴趣的恶魔，我们想利用敌手“解决困难问题”，就得用“模拟方案”来“欺骗”敌手。

5.2 Simulation and After Simulation

本节主要介绍模拟和模拟之后发生的事情。

5.2.1 Something Unknown

- Real Scheme: the **challenger** generates the scheme and **knows everything including the secret key**.
- Simulated Scheme: the **simulator** generates the scheme with a problem instance and **doesn't know something**.

simulator 必然有一些不知道的信息。反过来想，如果 simulator 已经知道了所有信息，那么敌手输出的信息肯定也属于 simulator，压根就不需要敌手的帮助了!!!

For example, a simulated scheme is generated with a problem instance (g, g^a, g^b) and the **simulator** therefore **doesn't know (a, b)** in simulation.

The unknown value might affect the simulation, 之后 simulation 部分会介绍。

5.2.2 Simulation Results

Successful Simulation 怎样的模拟是成功的？ no abort + correctness

- no abort: simulation 的过程中没有 abort，即 simulator 能响应敌手的所有询问。
abort(中止) 取决于我们(证明者)设计的归约算法，归约算法中描述了必须中止的情况。例如归约算法定义“当 simulator 不知道如何回答敌手的签名询问时，simulator 会中止”，那么当敌手询问一个消息的签名，而 simulator 不知道如何计算该消息的签名，simulation 就会中止。
- correctness: simulation 的过程中 simulator 对敌手的响应必须是正确的，不知道不能瞎答，只能中止。例如敌手询问一个消息对应的签名，simulator 必须返回正确的签名，而不能随便返回一个假的签名，企图达到不中止 simulation 的目的。

Indistinguishable Simulation 怎样的模拟是不可区分的？

A simulation is indistinguishable from real attack \Leftrightarrow 敌手无法区分 simulated scheme 和 real scheme.
unsuccessful simulation must be distinguishable from the real attack.

Whether Indistinguishable Or Not A Successful Simulation is an Indistinguishable Simulation if

- correctness: simulator 对敌手询问的响应满足正确性, 例如 simulator 产生的签名 (simulated signature) 是合法的、能通过签名验证算法.
- randomness: simulated scheme 产生的随机数 simulated random numbers 和 real random numbers 一样服从随机数的特点—random and independent.

注:

- 目前认为敌手能区分一个 given scheme 是 simulated scheme 还是 real scheme 主要是通过 correctness 和 randomness 两种方法, 暂不考虑如 real scheme 和 simulated scheme 的反应时间等. (以签名为例, 或许 simulated scheme 中签名明显慢于 real scheme, 敌手能借此区分).
- 需要满足 randomness 的原因在于数字签名算法等可能会带有随机数, 敌手或许能通过很多个签名的随机数不是“真的随机的”来区分 (例如 100 个签名的随机数都是 1, 则敌手能判断出不是 real scheme 中的真正的随机数, 进而知道是 simulated scheme).

5.2.3 Classification of Attacks

According to results of attack on simulated scheme, we have 攻击模拟方案层面

- Failed Attack. An attack fails if the attack cannot break the simulated scheme following the security model. For example, the forged signature is invalid.
- Successful Attack. An attack is successful if the attack can break the simulated scheme following the security model. For example, the forged signature is valid.

According to results of solving hard problem, we have 解决困难问题层面

- Useless Attack. A useless attack is an attack by the adversary that cannot be reduced to solving an underlying hard problem. 无法通过转化用来解决困难问题.
- Useful Attack. A useful attack is an attack by the adversary that can be reduced to solving an underlying hard problem. 转化后能用来解决困难问题, 得到困难问题的解.

Example: 思考为什么给 3 个, 而不是 1 个或 2 个? 自己出题, 方案设计与出题原理类似.

We define a real attack as follows.

- The adversary is given $(g, g^a, g^{b_1}, g^{b_2}, g^{b_3})$ generated by the challenger who knows (b_1, b_2, b_3) . challenger 可能知道 a 也可能不知道
- The adversary can adaptive query an integer $j \in \{1, 2, 3\}$ and the challenger responds with g^{ab_j} . adaptive: 自适应选取, 即选取各值的概率由选取者自己定义, 而不是服从均匀分布 ($j=1, 2, 3$ 各 $1/3$)
- The adversary breaks the scheme if it can compute $g^{ab_{i^*}}$ satisfying:

敌手询问了 1.2.3 中的某一个, $i^* \in \{1, 2, 3\} \wedge i^* \neq j$.
必须输出另外两个中的一个

Reduction: We can reduce adversary's attack to solving CDH problem.

CDH problem is reducible to this scheme

Proof. Given (g, g^a, g^b) , the simulator works as follows.

- Randomly choose $(c_1, c_2, c_3) \in \mathbb{Z}_p$ and $k \in \{1, 2, 3\}$. Set

$$b_i = \begin{cases} c_i & \text{if } i \neq k \\ b & \text{if } i = k \end{cases}$$

随机选取 c1 模拟 b1, c2 模拟 b2, c3 模拟 b3, 对其中的一个 k 将 ck 用 b 替换, 这样有三个数中有两个是 simulator 知道的, 还有一个 simulator 不知道, 至于是哪个取决于随机选取的 k 的结果

- We have $(g^{b_1}, g^{b_2}, g^{b_3})$ can be simulated from (c_1, c_2, c_3, g^b) .
- Send $(g, g^a, g^{b_1}, g^{b_2}, g^{b_3})$ to the adversary.

- The adversary makes an adaptive choice j . The simulator

$j=k$ 时 $b_j=b$, simulator 需要计算 g^{ab} 给对手, 这是 simulator 无法计算的, 所以 simulator 只能 abort. 这个计算过程中 simulator 不需要计算 g^{ab} 即 CDH 问题的解

- The adversary outputs $g^{ab_{i^*}}$. We have $g^{ab_{i^*}} = g^{ab}$ if $i^* = k$. 要知道 a 也需要知道 b

This completes the simulation and the solution. \square

- For simulator, the simulation is a **successful simulation** if the query satisfies $j \neq k$.

注: 此时不会中止, 且 simulator 能够正确地计算出值并发送给对手.(no abort + correctness)

- For adversary, the attack is a **successful attack** if the adversary can output $g^{ab_{i^*}}$, and the attack is a **useless attack** when $i^* \neq k$ (**useful attack** when $i^* = k$).

注: $i^* \neq k$ 时, simulator 可以自行计算 $g^{ab_{i^*}}$, 利用对手进行的攻击是无用的. $i^* = k$ 时, 是 simulator 所需要的计算, 此时对手的攻击是有用的.

5.2.4 Relations

simulator

We create a simulated scheme with a problem instance.

\Downarrow no abort + correctness

Successful Simulation

\Downarrow correctness + randomness

Indistinguishable Simulation

\Downarrow can break the simulated scheme

Successful Attack

\Downarrow can be reduced to solving an underlying hard problem

Useful Attack

\Downarrow

Solve Hard Problem

Security reduction is complicated because this is not the only logic!

这不是安全归约的唯一逻辑, 有些特殊的归约方法不用 follow 这个规则

The relations between attack and simulation

- **Suc Sim → Suc or Failed Attack.** An attack at the end of a **successful simulation** can be either a **failed attack** or a **successful attack**, which is dependent on whether the simulation is distinguishable or not. 成功的模拟不一定是成功的攻击, 还要看模拟是否是可区分的.
- **IND Sim → Suc or Failed Attack.** An attack at the end of an **indistinguishable simulation** must be a **successful attack** with probability defined in the breaking assumption. 不可区分的模拟是否对应成功的攻击, 取决于 break assumption, 与 break assumption 中定义的敌手成功攻击的概率相同.
- **DIS Sim → Suc or Failed Attack.** An attack at the end of a **distinguishable simulation** can be either a **failed attack** or a **successful attack**. That is, the adversary can decide which attack it will launch. This doesn't contradict the breaking assumption. 即使敌手发现是 simulated scheme, 仍有可能要攻破这个方案, 敌手的行为是不可预测的. 具体为什么敌手仍要攻破的原因非常复杂, 以后再讨论.
- **Suc Attack → Useful or Useless Attack.** The attack on the indistinguishable **simulated scheme** can be either a **useful attack** or a **useless attack**. An indistinguishable simulation cannot ensure a useful attack. 即使是 indistinguishable simulation, 也无法保证攻击一定对安全归约有用, 这是归约的难点.

奇葩的反例。。。。

Relations

We might receive a **useful attack** from the adversary in some very special security reductions, **even**

- The simulated scheme is distinguishable from real scheme, **or**
- The attack from the adversary is a failed attack. 敌手失败的攻击有时都能利用。。。

We might receive a **useless attack** from the adversary in many security reductions, **even**

- The simulated scheme is indistinguishable from real scheme, **and**
- The attack from the adversary is a successful attack.

Unfortunately, no example or further explanation now. :(

5.2.5 Classification of Security Reduction

- **Successful Security Reduction.** We say that a security reduction is **successful** if the adversary outputs a **useful attack**. 敌手能输出一个有用的攻击.
- **Correct Security Reduction.** We say that a security reduction for a proposed scheme is **correct** if the advantage of solving an underlying hard problem using the adversary's attack is non-negligible in polynomial time. 通过运行这个归约算法, 我们解决困难问题的优势是不可忽略的.

Successful Reduction 可以理解为 Security Reduction 中发生的好事件. 除了分析好处之外, 我们还需要保证该事件能以不可忽略 (non-negligible) 的概率出现, 否则解决困难问题的优势几乎没有.

5.3 Adversary

5.3.1 From Black-Box Adversary to Malicious Adversary

假设存在一个敌手以 (t, ϵ) 攻破方案 (在安全模型中), 我们 (证明者) 构建一个 simulator 来利用敌手解决困难问题. 首先敌手满足:

- 除了时间 t 和优势 ϵ , 对敌手没有任何限制.
- 我们不知道敌手如何攻破方案, 即使从 simulator 的角度不可能攻破的.
- Security Reduction 中的敌手是 **黑盒敌手 (Black-Box Adversary)**. 能抵抗黑盒敌手, 意味着能抵抗任意敌手.

黑盒敌手的特点: adaptive output, 不服从均匀分布, 也不可预测.

例如 a 是一个从 $\{0, 1\}$ 中选取的数.

- a is randomly chosen: $Pr[a = 0] = Pr[a = 1] = \frac{1}{2}$.
- a is adaptively chosen: $Pr[a = 0], Pr[a = 1]$ unknown.

Adaptive = Unpredictable \neq Adaptive Chosen-Message Attacks

黑盒敌手的 adaptive output 主要表现在 adaptive query 和 adaptive attack. 例子如下图.

The adaptive output from the black-box adversary includes

- Adaptive query.
- Adaptive attack.

Example. Suppose the message space is $\{m_1, m_2, m_3, m_4, m_5\}$ with five messages. An adversary will query the signature of one message before it forges a valid signature of a new message.

- Adaptive query. The adversary will query the signature of message m_i with **unknown probability**. 询问 $m_1 \sim m_5$ 中的每一个的概率值是不知道的
- Adaptive attack. If the adversary makes a signature query on m_4 , the adversary will forge a signature of one of the messages from $\{m_1, m_2, m_3, m_5\}$ with **unknown probabilities** between $[0, 1]$ satisfying
假设敌手询问了 m_4 , 则需要从 $m_1 m_2 m_3 m_5$ 中选取一个进行伪造, 不知道是哪个
 $Pr[m^* = m_1] + Pr[m^* = m_2] + Pr[m^* = m_3] + Pr[m^* = m_5] = 1$.

从黑盒敌手到恶意敌手. 在安全归约中, 我们希望构建一个 simulator 来欺骗所有的黑盒敌手 (“fool” any Black-box Adversary), 从而帮助我们解决困难问题. 如果有某个特殊的敌手无法欺骗, 我们就失败了! 必须能欺骗最狡猾/聪明的敌手.

假如敌手能输出两个对 simulated scheme 的攻击, one attack is useful and the other is useless(对安全归约有用/无用), 因为黑盒敌手 adaptive attack, 我们无法预测敌手输出 useful attack 的概率.

然而 correct reduction 又要求我们计算出返回 useful attack(solving hard problem) 的概率, 因此我们只能将黑盒敌手放缩成恶意敌手 (Malicious Adversary): 考虑输出一个 useless attack 的最大概率.

Malicious Adversary:

- The malicious adversary will try its best to launch a useless attack unless the adversary doesn't know which attack is useless. 恶意敌手尽可能地对 simulated scheme 发起对安全归约 useless 的攻击.

- If the maximum probability of returning a useless attack is not the overwhelming probability 1, the probability of returning a useful attack must be noticeable (non-negligible).
- If a security reduction works against such a malicious adversary, the security reduction definitely works against any adversary who can break the proposed scheme. 能抵抗 Malicious Adversary 的安全归约肯定能抵抗任意的敌手.

总之，在安全归约中我们需要计算概率，但是黑盒敌手的行为不可预测，我们无法计算概率，因此我们直接放大到恶意的敌手，考虑恶意的敌手发动 useless attack 的 advantage.

回到上面这个例子，恶意敌手在输出 $g^{ab_{i^*}}$ 前会尽力去猜出 k . (从而实现 useless attack).

Proof. Given (g, g^a, g^b) , the simulator works as follows.

- Randomly choose $(c_1, c_2, c_3) \in \mathbb{Z}_p$ and $k \in \{1, 2, 3\}$. Set

$$b_i = \begin{cases} c_i & \text{if } i \neq k \\ b & \text{if } i = k \end{cases}$$

- We have $(g^{b_1}, g^{b_2}, g^{b_3})$ can be simulated from (c_1, c_2, c_3, g^b) .
- Send $(g, g^a, g^{b_1}, g^{b_2}, g^{b_3})$ to the adversary.
- The adversary makes an adaptive choice j . The simulator aborts if $j = k$. Otherwise, we have $b_j = c_j$ and the simulator can compute $g^{ab_j} = (g^a)^{c_j}$ for the adversary.
- The adversary outputs $g^{ab_{i^*}}$. We have $g^{ab_{i^*}} = g^{ab}$ if $i^* = k$.

In this reduction, the malicious adversary will try its best to guess k before it outputs $g^{ab_{i^*}}$ (try to launch a useless attack).



5.3.2 Malicious Adversary and Useful Attack

Let P_e be the probability of **returning a successful attack** on the proposed scheme under the breaking assumption. P_e 是 breaking assumption 中定义的返回成功攻击的概率.

- 不可区分时: If the simulated scheme is **indistinguishable** from the real scheme, according to the breaking assumption, the adversary will **return a successful attack** on the simulated scheme with probability P_e .
- 可区分时: If the simulated scheme is **distinguishable** from the real scheme, the adversary will **return a successful attack** on the simulated scheme with malicious and adaptive probability $P^* \in [0, 1]$.
 - 敌手发现 scheme 是一个 simulated scheme 后, 会以一个不可预测的概率 $P^* \in [0, 1]$ 返回成功的攻击 (恶意的, 尽量不返回). 这里返回成功的攻击的概率不是 0 或者 1, 而是 P^* . 与 breaking assumption 并不矛盾, 因为可区分时不是一个 real scheme, 成功攻击 real scheme 的概率为 P_e .
 - 具体地, 不可预测的概率 $P^* \in [0, 1]$ 和 advantage 一样有统一性, 在数字签名和加密等不同方案中, 概率 $P^* \in [0, 1]$ 是不同的. 数字签名中 $P^* = 0$, 加密方案中 $P^* = P_e$ 等.

5.3.3 Adversary's Computing Ability

Theorem: If the mathematical problem P is hard, then the proposed scheme is secure and there exists no adversary who can break the proposed scheme in polynomial time with non-negligible advantage.

我们只讨论“问题 P 是困难的”时提出的方案是不是安全的，并不关心“ P 是简单的”时方案安不安全。就好比如果离散对数问题是简单的，还需要去考虑群上基于离散对数问题构造的方案的安全性吗？对“方案的安全性”的讨论，仅局限在“问题 P 是困难的”时。

Question: Can the adversary solve the problem P in reduction? amplify the adversary's computational ability in security reduction

粗看起来敌手应该不能解决问题 P , 否则敌手能轻松攻破方案. 然而“安全归约的目的”是证明“提出的方案”能够抵抗任意的“无法在多项式时间内解决困难问题 P 的敌手”. 在安全归约中, 我们并不关心“提出的方案”是否被攻破, 相反, 仅关心“困难问题 P ”能否在多项式时间内解决. 即使敌手真的能解决“困难问题 P ”也没影响, 只要 simulator 能解决“困难问题 P ”就行了.

放大敌手计算能力 (amplification): 方案中的敌手 (计算能力有限) → 归约中的敌手 (计算能力无限).

- 构建公钥密码提出的方案一般是 computationally secure, 针对计算能力有限的敌手. (方案的安全性)
- 如果能解决困难问题, 那么攻破该方案肯定是容易的.
- 在归约时需要满足: 即使敌手的计算能力是无限的 (computationally unbounded), 归约也要是有效的. (归约的有效性)

注: The “computationally unbounded” adversary can solve all computationally hard problems in polynomial time with advantage 1.

- Benefit: The amplification will **simplify** the **analysis of reduction**.
- Dis-Benefit: Programming a **correct security reduction** becomes **harder**.

放大的好处: 在安全归约的 simulation 中, e.g. 对签名查询的响应中会包含一些 secret information(记作 \mathbb{I}). \mathbb{I} 或许会泄露“how to generate a useless attack on the simulated scheme”. 换句话说, 敌手只要知道了信息 \mathbb{I} , 就能发起一次 useless attack. 因此在正确性分析时, 我们需要分析敌手 cannot obtain \mathbb{I} , 否则归约就失败了. 现有的安全归约方案都无法避免这个问题, 表现了分析“敌手无法从 \mathbb{I} 中获得有价值的线索”的必要性. According to **all proposed** security reductions in the literature, it seems **impossible not to include** the information \mathbb{I} in responses to the adversary. 下面是两个例子, 卧报较早的一篇文章有介绍.

Proof. Given (g, g^a, g^b) , the simulator works as follows.

- Randomly choose $(c_1, c_2, c_3) \in \mathbb{Z}_p$ and $k \in \{1, 2, 3\}$. Set

$$b_i = \begin{cases} c_i & \text{if } i \neq k \\ b & \text{if } i = k \end{cases}$$

- We have $(g^{b_1}, g^{b_2}, g^{b_3})$ can be simulated from (c_1, c_2, c_3, g^b) .
- Send $(g, g^a, g^{b_1}, g^{b_2}, g^{b_3})$ to the adversary.
- The adversary makes an adaptive choice j . The simulator aborts if $j = k$. Otherwise, we have $b_j = c_j$ and the simulator can compute $g^{ab_j} = (g^a)^{c_j}$ for the adversary.
- The adversary outputs $g^{ab_{i^*}}$. We have $g^{ab_{i^*}} = g^{ab}$ if $i^* = k$.

有一个 b_i 是由 b 模拟得到，包含 b , 由 b 能计算 k 要证明敌手无
The elements $(g, g^a, g^{b_1}, g^{b_2}, g^{b_3})$ carries the information $\mathbb{I} = k$ 法从中得到 k

Prove that the adversary cannot compute a from the following elements:

$$\left(g, g^{x+z}, g^{x+a+y}, g^{z+a^2} \right),$$

where x, y, z are randomly chosen from \mathbb{Z}_p .

放大后, 拥有无限计算能力的敌手, 能解决离散对数问题
After the amplification, we only need to prove that the adversary cannot compute a from the following elements:

$$\left(x + z, \quad x + a + y, \quad z + a^2 \right).$$

抽取后利用数学的工具, 和原先的群、离散对数等无关
The algebra methodology will be applied in analysis (in later lectures).

5.3.4 What the Adversary Knows and Never Knows

安全归约开始于 break assumption, 我们 (证明者) 构造了一个 simulator 来生成一个 simulated scheme, 然后利用敌手的攻击来解决困难问题 (underlying hard problem).

- 敌手有着无界的计算能力 (unbounded computational power), 可以解决所有的 computational hard problems.(不包含 secure primitive, e.g. 假设安全的 Hash 函数)
- 敌手是恶意的 (maliciously), 它将尽其所能发起一次攻击, 既能攻破 simulated scheme, 又能使得归约失败 (无法用来解决困难问题). (The adversary will maliciously try its best to launch a useless attack to break the simulated scheme and make the reduction fail.)
- 安全归约的难点在于: 我们必须依靠这样的一个敌手去解决困难问题, 还要以一个不可忽略的优势.

What the Adversary Knows 敌手知道的: 三大块

- 方案算法 (Scheme Algorithm). The adversary knows the scheme algorithm of the proposed scheme, such as signing and verification algorithm.
- 归约算法 (Reduction Algorithm). The adversary knows the reduction algorithm for proving the security of the proposed scheme. 如果 given scheme 是一个 simulated scheme, 敌手知道 simulator 具体是怎么算的. For example, the adversary knows how signatures are simulated. 因为我们无法证明敌手不知道归约算法, 因此只能假设其知道.
- 能计算所有的计算困难问题 (How to Solve All Computational Hard Problems). 我们放大了敌手的能力, 因此其能解决所有计算困难问题. For example, suppose (g, g^a) is given to the adversary. We assume that the adversary can compute a before launching an attack.

What the Adversary Never Knows 敌手不知道的: 三大块

- 随机数 (Random Numbers). 敌手不知道 simulator 选取的随机数, 除非 simulator 将其发送给敌手. The adversary doesn't know those random numbers (including group elements) chosen by the simulator unless they can be computed by the adversary.
- 问题实例 (Problem Instance). The adversary doesn't know the random instance of the underlying hard problem given to the simulator. 简化证明的描述和书写.

- 敌手无法求解绝对困难问题 (How to Solve an Absolutely Hard Problem). Such as computing (x, y) from the group elements (g, g^{x+y}) . (This type of problems will be introduced later.)
注: If the simulator randomly chooses $x, y \in \mathbb{Z}_p$, they are **unknown** to the adversary. However, once (g, g^{x+y}) are given to the adversary, the adversary can compute and **cannot obtain** the (x, y) that simulator chooses.

Proof. Given (g, g^a, g^b) , the simulator works as follows.

- Randomly choose $(c_1, c_2, c_3) \in \mathbb{Z}_p$ and $k \in \{1, 2, 3\}$. Set

$$b_i = \begin{cases} c_i & \text{if } i \neq k \\ b & \text{if } i = k \end{cases}$$

- We have $(g^{b_1}, g^{b_2}, g^{b_3})$ can be simulated from (c_1, c_2, c_3, g^b) .
- Send $(g, g^a, g^{b_1}, g^{b_2}, g^{b_3})$ to the adversary.
- The adversary makes an adaptive choice j . The simulator aborts if $j = k$. Otherwise, we have $b_j = c_j$ and the simulator can compute $g^{ab_j} = (g^a)^{c_j}$ for the adversary.
- The adversary outputs $g^{ab_{i^*}}$. We have $g^{ab_{i^*}} = g^{ab}$ if $i^* = k$.

The adversary knows a, b_1, b_2, b_3 before it launches an attack.



5.3.5 Summary of Adversary

将敌手的眼睛遮住带入一个房间, 给敌手一个 given scheme, 可能是真实世界中的 real scheme, 也可能是模拟世界中的 simulated scheme. 敌手利用他知道的三大块以及他能询问的 (在安全模型下) 来判断这个 given scheme 是 real scheme 还是 simulated scheme.

- 当敌手无法区分 given scheme 是 real scheme 还是 simulated scheme 时: 根据 breaking assumption, 敌手将对 given scheme 发起一次攻击, 是成功的攻击 (可以用来攻破方案) 的概率为 P_e .
- 当敌手发现 given scheme 是一个 simulated scheme 时: 敌手将以恶意且不可预测的概率 $P^* \in [0, 1]$ 发起一次成功的攻击, 恶意指敌手会尽量让 simulator 没办法用这个攻击来解决困难问题.

理想中的安全归约: Security Reduction. Suppose there exists an adversary who can break the proposed scheme. We can construct a simulator to solve an underlying hard problem.

实际上的安全归约: Correct Security Reduction. Even if the attack on the simulated scheme is launched by a malicious adversary who has unbounded computational power, the advantage of solving the underlying hard problem is still non-negligible (in polynomial time).



能够干掉 simulated scheme 的同时, 又
不帮你解决困难问题, 让你欲哭无泪.
Can you survive from my useless-attack fooling?

6 Lecture 6 Simulation and Solution(Contradiction, Trapdoor, and Random Oracle)

Outline:

1. Simulation and Contradiction

- Secret Key is Unknown
- Secret Key is Known
- Simulation Examples
- Trapdoor

2. Random Oracle

- What is Random Oracle?
- How to use Random Oracle?
- Applications
- *Oracle Response

道高一尺，魔高一丈。

6.1 Simulation and Contradiction

安全归约充满着矛盾，在做安全归约时应当知道这些矛盾的点是如何被解决的，否则安全归约可能是错的。这里介绍两类常见的矛盾 (paradoxes = contradiction)

- The secret key in the simulated scheme is unknown. (在模拟方案时，simulator 不知道私钥)
- The secret key in the simulated scheme is known. (在模拟方案时，simulator 知道私钥)

6.1.1 Secret key is Unknown

Taking a simulated signature scheme as the example. In simulation,

1. simulated scheme 是由 a problem instance 产生的。
2. simulator 并不知道 secret key。
3. 如果 simulator 按照 signing algorithm 给对手计算签名，是需要私钥的。

simulator 没有 secret key，又怎么通过 signing algorithm 产生签名呢？

The first Paradox: Simulation 过程的悖论

- In simulated scheme, simulator 可以在没有私钥的情况下计算某些“对手查询的消息”的签名。
- In real scheme, 签名的产生必须要有私钥作为输入，否则这个签名方案肯定是不安全的。

Why the signature computing in simulated scheme is so special? 在 simulated scheme 中计算的签名有什么特殊？

The second Paradox: Solution 过程的悖论

- In simulated scheme, simulator 用对手产生的签名来解决困难问题。
- In simulated scheme, simulator 也能自己产生一些签名（因为在对应的安全模型中 simulator 需要为对手计算一些签名），为什么 simulator 不能自己产生签名来解决困难问题。而一定要用对手产生的签名来解决困难问题。

Why the forged signature from the adversary is so special? 故手要的签名有什么特殊？

6.1.2 Secret key is Known

大多数 security reduction 中 simulator 不知道 secret key. 然而, 一些特殊的方案对应的 security reduction 允许 simulator 知道 secret key. Unknown 和 Known 对应的自我矛盾的点是完全不同的.

Paradox in Signature

- Unknown: In simulated scheme, simulator 不知道 secret key, 用敌手产生的签名来解决困难问题.
- Known: In simulated scheme, simulator 知道 secret key, 可以计算所有消息的签名.

Why the forged signature from the adversary is so special? **为什么已经可以计算所有消息的签名了, simulator 还要用敌手伪造的签名?**

Paradox in Encryption

- Unknown: In simulated scheme, simulator 不知道 secret key, 用敌手的解密结果来解决困难问题 (Recall: IND 模型).
- Known: In simulated scheme, simulator 知道 secret, 可以解密所有的密文, 包括 challenge ciphertext.

Why the decryption result from the adversary is so special? **为什么已经可以解密所有的密文了, simulator 还要用敌手的解密结果?**

Secret key is Known 时的安全归约, 不建议刚入门的学生搞懂以上两种逆天的技术!

6.1.3 Simulation with Problem Instance

Completing some **tricky computations** without knowing some **secrets** is quite normal in security reduction. They are used in **computing responses** to the adversary's queries, and we give several important simulations from the problem instance.

这个技术是为了: simulation without knowing secret key!

6.1.4 Simulation Examples

8 个非常重要的例子, 已知一些输入, 在不知道某些 secrets 的前提下, 计算一些看起来不可能的输出. 都是从读过的论文中抽象总结出来. 仔细体会!

6.1.5 Trapdoor

这部分书上没有.

Trapdoor 是 simulated scheme 中, simulator 能获得的一种额外信息 (additional secret), 用来在不知道 secret key 时计算一些 (敌手询问对应的) response.

Trapdoor 的用途:

- **代替 secret key.** 在 simulation 时, 我们不知道 secret key, 而 computing 需要用到 secret key, trapdoor 能用来代替 secret key 以达到 computing 的结果. (We program the simulation in the way that computing using secret key can be replaced with computing using the trapdoor.)
- **将 attack 转化成 solution.** Trapdoor 还能被用来在 simulated scheme 中从敌手的攻击中 extract problem solution.

下面给出两个使用 Trapdoor 的例子:

Example: Trapdoor for Simulation

Let $pk = (g, g^\alpha, h)$ and $sk = \alpha$. The signature on message $m \in \mathbb{Z}_p$ is

$$\sigma_m = h^{\frac{1}{\alpha+m}}.$$

- Suppose (g, g^α) is simulated with problem instance (g, g^a) .
- The secret key α is unknown.
- When h is simulated with $h = g^{w(a+m)}$ where w is a random integer known by the simulator, we have
雪崩时, 没有一片雪花是无辜的.
归约时, 没有一个群元是随意的.
群元的设置必然有其目的 $\sigma_m = h^{\frac{1}{\alpha+m}} = g^{\frac{w(a+m)}{\alpha+m}} = g^w$.
- Then the function $f(x) = w(x + m)$ is called Trapdoor.
Trapdoor 是除了公钥外, simulator
额外知道的信息.



Example: Trapdoor for Solution

Let $pk = (g, g^\alpha, h)$ and $sk = \alpha$. The signature on message $m \in \mathbb{Z}_p$ is

$$\sigma_m = h^{\frac{1}{\alpha+m}}.$$

- The given problem instance is (g, g^a) from 1-SDH problem.
- Suppose (g, g^α, h) is simulated with $(g, g^a, g^{w(a+m)})$.
- Suppose the adversary forges a signature on $m^* \neq m$. We have

Simulator 知道 $\sigma_{m^*} = h^{\frac{1}{\alpha+m^*}} = g^{\frac{w(a+m)}{a+m^*}}$.

Trapdoor 信息

- Then the Trapdoor $f(x) = w(x + m)$ can be used to compute the following problem solution from the forged signature.

因为 Simulator 知道 Trapdoor, 因此其能在 $m^* \neq m$ 时抽取出一个困难

问题的解: $\xrightarrow{(m^*, g^{\frac{1}{a+m^*}})}$

如果 $h=g^b$, Simulator 不知道 b 和 Trapdoor, 无法计算出该解.



In the simulated scheme, all random elements (random group elements and random numbers) are not truly random elements. They are well computed following some ways from trapdoors.

假如用 secret key α 对消息 m 进行签名, 得到的签名为 $(g^r, g^{\frac{1}{r}}, g^{\frac{r}{\alpha+r}})$, 其中 r 是在生成签名时选取的一个随机数 (random number).

假如 simulator 在 signature simulation 中, 能从 \mathbb{Z}_p 中随机选择 $x = r$, 为什么不将签名简化为 $g^{\frac{1}{\alpha+r}}$ 呢?

如果归约中遇到了上述情况, 要么证明错了, 要么方案可以简化 (e.g. 减少元的个数).

6.2 Random Oracle

6.2.1 What is Random Oracle

假如一个方案在构造时用到了 Hash 函数 H , 在对其进行安全证明时需要用到 Random Oracles. 换句话说, 含有 Hash 函数的方案, 在安全证明时 Hash 函数 H 被设置为 RO(Random Oracle) \mathcal{O} , 在分析安全性时用的是 Random Oracle \mathcal{O} 而不是 Hash 函数 H , 我们相信当前者安全时后者就是安全的.(其实不一定满足, 但是应用密码学中一般直接用)

- 在证明一个方案的安全性时, **至少需要将一个 Hash 函数设置为 RO**, 并不需要将所有的 Hash 函数都设置为 RO (这取决于具体的安全归约的需求).
- 当一个 Hash 函数被设置为 RO 后, 敌手就不能从 simulator 处得到 Hash 函数算法了 (在安全证明中), 相反, 敌手只能通过询问 RO 得到 Hash 值.
- RO 是 ideal Hash function, 实际上是不存在的, 然而在 simulation 时并不需要考虑如何构造实现 ideal Hash function. 相反, simulator 主要考虑当敌手每次询问 x 时, 返回的 $H(x)$ 的值.
- RO 是做安全归约时的一种逆天的工具!

Hash Function vs Random Oracle 注意“表的长度”和“映射方式”.

Input	Hash Function	Output	Input	Random Oracle	Output
x_1		y_1	x_1		y_1
x_2		y_2	x_2		y_2
x_3	$H(x_i) = y_i$	y_3	x_3	Simulator	y_3
x_4		y_4	.		.
:		:	x_q		y_q

Hash 函数表的长度很大 (由定义, 例如 input space 为 160 bits, 可以输入 2^{160} 个 inputs), 通过 Hash 函数算法将 x_i 映射到 y_i . 而 RO 表的长度只有 q (远短于左边), 是安全归约中敌手询问的次数, x_i, y_i 之间的映射关系由 Simulator 决定.

区别:

- Knowledge. 给定一个输入 x , 如果 H 是一个 Hash 函数, 敌手知道 H 就可以自己计算 $H(x)$. 如果 H 被设置为 RO, 敌手只有 queries x to the RO 才能得到 $H(x)$ 的值, 而无法自己计算.
- Input. (见前面) Hash 函数和 RO 的 input space 相同, 但是 Hash 函数表的 input 的个数可以很多 (exponential), 而 RO 的 input 的个数为 q (polynomial), 即安全归约中敌手询问的次数, queries in polynomial time having polynomial number.
- Output. 给定一个 input, Hash 函数的 output 由 input 和 Hash 函数算法决定, 而 RO 的 output 由控制着 RO 的 simulator 决定.
- Representation. Hash 函数可以理解为一个通过 Hash 函数算法进行的映射, 从 input space 到 output space. RO 是没有映射的, 值由 simulator 决定, 并没有具体的映射规则.(如果说有, 那也是 simulator 制定的, 遵循归约算法).

6.2.2 How to use Random Oracle

Hash List 首先需要建立一个 Hash List 来记录每个 input x 和 output $y = H(x)$ 之间的关系, 这里 H 被设置为 RO.

- From the view of the adversary. the Hash List consists of $(x_1, y_1), (x_2, y_2), \dots, (x_q, y_q)$.
- From the view of the simulator. the Hash List could consist of $(x_1, y_1, S_1), (x_2, y_2, S_2), \dots, (x_q, y_q, S_q)$. 这里 S_i 是被用来计算 y_i 的 intermediate secret. intermediate secret 的例子见下面. 也有一些在计算 y 时不需要 S (这里书上表述不准确), 甚至每个 x 对应的 y 可以是完全随机的.

Example of intermediate secret in Hash List:

Let $H : \{0, 1\}^* \rightarrow \mathbb{G}$ be a random oracle.

- For a query on x_i , the simulator randomly chooses w_i .
- It computes and sets $y_i = H(x_i) = g^{w_i}$.
- The simulator adds (x_i, y_i, w_i) into the hash list.
- We define the intermediate secret $S_i = w_i$.

Note: The use of intermediate secret will be explained later.

Security Reduction with Random Oracle 在 Simulation 中, Simulator 需要增加 H-Query 阶段 (通常在 Setup 阶段之后), H-Query 阶段的作用是描述对手进行的 hash queries 和 responses.

H-Query. The adversary makes hash queries in this phase. The simulator prepares a hash list \mathcal{L} to record all queries and responses, where the hash list is empty at the beginning.

For a query x to the random oracle,

- If x is already in the hash list, the simulator responds to this query following the hash list. 需要保证对手重复询问时, 得到的结果的一致性.
- Otherwise, the simulator generates an intermediate secret S and uses it to compute y based on reduction algorithm. Then, the simulator responds to this query with $y = H(x)$ and adds the tuple (x, y, S) to the hash list. 并不是所有用到 RO 的安全归约都有 S .

注: H-Query phase 只在 Security Reduction 中出现, 不是在 Security Model 中出现! 安全模型中不能出现 Hash 函数.

6.2.3 Application of the RO

RO 怎么用 (三大类) 以及如何去掉 RO. 以下三个 RO 的应用, 在化成灰之前不可忘记!

1. We program the output and know a trapdoor. (simulator 在给出 output 时嵌入一些陷门.)
2. We program the output into a special space(simulator 能将 output 映射到一些特殊的集合/空间.)
3. We control the output and know the input.(simulator 能通过控制 output 来知道 input.) 如果不向我询问 x , 你就无法知道 $H(x)$, 因为 $H(x)$ 是由我决定, 但当你向我询问 x , 我就知道你涉及到了什么.

注: with RO 的安全归约相对容易 (适合先学习), without RO 通常是找到了代替 RO 的复杂技术.

Application 1: We program the output and know a trapdoor.

Let $H : \{0, 1\}^* \rightarrow \mathbb{G}$ be a random oracle. 由simulator控制

- For a query on x_i , the simulator randomly chooses w_i .
- It computes and sets $y_i = H(x_i) = g^{w_i}$. BLS签名的核心
- The simulator adds (x_i, y_i, w_i) into the hash list.
- The intermediate secret $S_i = w_i$ is the trapdoor.

Trapdoor: Even the simulator doesn't know a in (g, g^a) , it can still compute the element secret key 和 trapdoor 至少知道一个, 就能计算

Simulator 不知道 a 的情况下, 知道 trapdoor w_i , 计算 $H(x_i)^a$

$H(x_i)^a = (g^{w_i})^a = (g^a)^{w_i}$
如果 H 是真正的 Hash 函数, 在不知道 a 时, 无法计算 $H(x_i)$ 和 $H(x_i)^a$,
然而在 RO 模型中, H 是由 simulator 控制的一个随机预言机, 且 simulator 拥有陷门信息 w_i , 因此 simulator 能在不知道 a 的情况下, 利用 g^a 和 w_i 计算出 $H(x_i)^a$

Application 2: We program the output into a special space.

这是一种挖坑技术

Let $pk = (g, g^\alpha, h, H)$ and $sk = \alpha$. The signature on $m \in \mathbb{Z}_p$ is

$$\sigma_m = h^{\frac{1}{\alpha + H(m)}}.$$

Let $H : \{0, 1\}^* \rightarrow \mathbb{Z}_p$ be a random oracle.

- (g, g^α, h) is simulated with $(g, g^a, g^{w(a+z)})$, where w, z are random integers from \mathbb{Z}_p known by the simulator. (a is unknown.)
- Suppose the adversary can make one signature query on m after seeing public key.

simulator 能模拟出 m 对应的签名吗?
Mapping: The signature on m can be computed when $y \neq H(m)$ set z.
挖个坑, 隐藏自己: 能模拟出对应的签名, 让对手分不清是 simulated scheme 还是 real scheme
挖个坑, 让对手掉下去: 利用对手解决困难问题

Application 3: We control the output and know the input.

在基于“计算困难问题”的加密中非常重要

Let $H : \{0, 1\}^* \rightarrow \{0, 1\}^n$ be a random oracle.

- Suppose the adversary can break the following ciphertext and know the message with advantage 1.

$$CT = (g, g^a, g^b, H(g^{ab}) \oplus m).$$

- The adversary must ever make the query g^{ab} to the random oracle before it can break CT .
- The simulator knows that one of hash queries is equal to g^{ab} .
RO 是由 simulator 控制的, 因此当对手向 RO 询问时, simulator 知道对手询问了什么. 如果对手能攻破密文 CT , 那么 simulator 知道对手一定曾经询问过 $g^{\{ab\}}$, (某个 queries=g^{\{ab\}}), 否则对手没有办法(优势)攻破该密文. 而对手最多询问过多项式个 queries, simulator 能找到 $g^{\{ab\}}$ 的概率就变得很大.

6.2.4 *Oracle Response

设 H 是一个 random oracle, 签名查询次数 $q \ll \text{Hash}$ 查询 (random oracle 查询) 次数 q_H .

- 假如敌手进行了 q_H 次 random oracle 查询, 得到 $(x_1, y_1), (x_2, y_2), \dots, (x_{q_H}, y_{q_H})$.
- 假如 simulator 记录的 Hash List 为 $(x_1, y_1, c_1), (x_2, y_2, c_2), \dots, (x_{q_H}, y_{q_H}, c_{q_H})$, 其中 $c_i \in \{0, 1\}$ 是由 simulator 选取的 (不是随机选的).
- 假如敌手自适应地选取 $(x'_1, x'_2, \dots, x'_q, x^*)$. 它们都在 Hash List 中, 对应的 c_i 分别为 $(c'_1, c'_2, \dots, c'_q, c^*)$.

Question: How to set c_i for each x_i to obtain a high probability. 可观的, 不可忽略的值.

$$P = \Pr[c'_1 = c'_2 = \dots = c'_q = 0 \wedge c^* = 1]$$

有以下两种方法: 方法一较容易, 得到的概率较低. 方法二较为复杂, 能得到更高的概率.

Approach 1

Approach 1, 相对简单而概率较低

In the first approach, the simulator randomly chooses $i^* \in [1, q_H]$ and guesses that the adversary will output the i^* -th query as x^* . Then, for a query x_i , the simulator sets

simulator 建一个长为 q_H 的表, 猜测敌手会用表的第 i^* 个作为 x^* , 在设置 i^* 时不知道 x^* .

$$\begin{cases} c_i = 1 & \text{if } i = i^* \\ c_i = 0 & \text{otherwise} \end{cases}.$$

In this setting,

随机猜的一个设为1, 其余都是0

$$P = \Pr[c'_1 = c'_2 = \dots = c'_q = 0 \wedge c^* = 1]$$

is equivalent to successfully guessing which query is chosen as x^* .

Since the adversary makes q_H queries, and one of queries is chosen to be x^* , we have

simulator 是随机猜的, 敌手也是不可预测的, 随机猜中的 x^* 恰为敌手选的 x^* 的概率是 $1/q_H$, 此时 event 发生

$$P = \Pr[c'_1 = c'_2 = \dots = c'_q = 0 \wedge c^* = 1] = \frac{1}{q_H},$$

which is linear in the number of all hash queries.



复杂点，但结果概率高

Approach 2

抛一个不均匀的硬币

In the second approach, the simulator flips a bit $b_i \in \{0, 1\}$ in such a way that $b_i = 0$ occurs with probability P_b , and $b_i = 1$ occurs with probability $1 - P_b$. Then, for a query x_i , the simulator sets

选择多个 c_i 并设置为 1
剩下的设置为 0

$$\begin{cases} c_i = 1 & \text{if } b_i = 1 \\ c_i = 0 & \text{otherwise} \end{cases}.$$

Since all b_i are chosen according to the probability P_b , we have

$$\begin{aligned} P &= \Pr[c'_1 = c'_2 = \dots = c'_q = 0 \wedge c^* = 1] \\ &= \Pr[b'_1 = b'_2 = \dots = b'_q = 0 \wedge b^* = 1] \\ &= P_b^q(1 - P_b). \end{aligned}$$

The value is maximized at $P_b = 1 - 1/(1 + q)$, and then we get

$$P \approx \frac{1}{(1 + \frac{1}{q})^q \cdot q} = \frac{1}{e \cdot q}, \quad \text{这里 } e \text{ 是自然对数, } e=2.71\dots$$

which is linear in the number of chosen hash queries.

$q \ll qH$

$1/qH \ll 1/e \cdot q$

入门时，最好学习如何使用 RO 做证明，之后再考虑如何去掉 RO.

7 Lecture 6 Simulation and Solution(Contradiction, Trapdoor, and Random Oracle)

Outline:

1. Simulation and Contradiction

- Secret Key is Unknown
- Secret Key is Known
- Simulation Examples
- Trapdoor

2. Random Oracle

- What is Random Oracle?
- How to use Random Oracle?
- Applications
- *Oracle Response

道高一尺，魔高一丈。

7.1 Simulation and Contradiction

安全归约充满着矛盾，在做安全归约时应当知道这些矛盾的点是如何被解决的，否则安全归约可能是错的。这里介绍两类常见的矛盾 (paradoxes = contradiction)

- The secret key in the simulated scheme is unknown. (在模拟方案时，simulator 不知道私钥)
- The secret key in the simulated scheme is known. (在模拟方案时，simulator 知道私钥)

7.1.1 Secret key is Unknown

Taking a simulated signature scheme as the example. In simulation,

1. simulated scheme 是由 a problem instance 产生的。
2. simulator 并不知道 secret key。
3. 如果 simulator 按照 signing algorithm 给对手计算签名，是需要私钥的。

simulator 没有 secret key，又怎么通过 signing algorithm 产生签名呢？

The first Paradox: Simulation 过程的悖论

- In simulated scheme, simulator 可以在没有私钥的情况下计算某些“对手查询的消息”的签名。
- In real scheme, 签名的产生必须要有私钥作为输入，否则这个签名方案肯定是不安全的。

Why the signature computing in simulated scheme is so special? 在 simulated scheme 中计算的签名有什么特殊？

The second Paradox: Solution 过程的悖论

- In simulated scheme, simulator 用对手产生的签名来解决困难问题。
- In simulated scheme, simulator 也能自己产生一些签名（因为在对应的安全模型中 simulator 需要为对手计算一些签名），为什么 simulator 不能自己产生签名来解决困难问题。而一定要用对手产生的签名来解决困难问题。

Why the forged signature from the adversary is so special? 故手要的签名有什么特殊？

7.1.2 Secret key is Known

大多数 security reduction 中 simulator 不知道 secret key. 然而, 一些特殊的方案对应的 security reduction 允许 simulator 知道 secret key. Unknown 和 Known 对应的自我矛盾的点是完全不同的.

Paradox in Signature

- Unknown: In simulated scheme, simulator 不知道 secret key, 用敌手产生的签名来解决困难问题.
- Known: In simulated scheme, simulator 知道 secret key, 可以计算所有消息的签名.

Why the forged signature from the adversary is so special? **为什么已经可以计算所有消息的签名了, simulator 还要用敌手伪造的签名?**

Paradox in Encryption

- Unknown: In simulated scheme, simulator 不知道 secret key, 用敌手的解密结果来解决困难问题 (Recall: IND 模型).
- Known: In simulated scheme, simulator 知道 secret, 可以解密所有的密文, 包括 challenge ciphertext.

Why the decryption result from the adversary is so special? **为什么已经可以解密所有的密文了, simulator 还要用敌手的解密结果?**

Secret key is Known 时的安全归约, 不建议刚入门的学生搞懂以上两种逆天的技术!

7.1.3 Simulation with Problem Instance

Completing some **tricky computations** without knowing some **secrets** is quite normal in security reduction. They are used in **computing responses** to the adversary's queries, and we give several important simulations from the problem instance.

这个技术是为了: simulation without knowing secret key!

7.1.4 Simulation Examples

8 个非常重要的例子, 已知一些输入, 在不知道某些 secrets 的前提下, 计算一些看起来不可能的输出. 都是从读过的论文中抽象总结出来. 仔细体会!

7.1.5 Trapdoor

这部分书上没有.

Trapdoor 是 simulated scheme 中, simulator 能获得的一种额外信息 (additional secret), 用来在不知道 secret key 时计算一些 (敌手询问对应的) response.

Trapdoor 的用途:

- **代替 secret key.** 在 simulation 时, 我们不知道 secret key, 而 computing 需要用到 secret key, trapdoor 能用来代替 secret key 以达到 computing 的结果. (We program the simulation in the way that computing using secret key can be replaced with computing using the trapdoor.)
- **将 attack 转化成 solution.** Trapdoor 还能被用来在 simulated scheme 中从敌手的攻击中 extract problem solution.

下面给出两个使用 Trapdoor 的例子:

Example: Trapdoor for Simulation

Let $pk = (g, g^\alpha, h)$ and $sk = \alpha$. The signature on message $m \in \mathbb{Z}_p$ is

$$\sigma_m = h^{\frac{1}{\alpha+m}}.$$

- Suppose (g, g^α) is simulated with problem instance (g, g^a) .
- The secret key α is unknown.
- When h is simulated with $h = g^{w(a+m)}$ where w is a random integer known by the simulator, we have
雪崩时, 没有一片雪花是无辜的.
归约时, 没有一个群元是随意的.
群元的设置必然有其目的 $\sigma_m = h^{\frac{1}{\alpha+m}} = g^{\frac{w(a+m)}{\alpha+m}} = g^w$.
- Then the function $f(x) = w(x + m)$ is called Trapdoor.
Trapdoor 是除了公钥外, simulator
额外知道的信息.



Example: Trapdoor for Solution

Let $pk = (g, g^\alpha, h)$ and $sk = \alpha$. The signature on message $m \in \mathbb{Z}_p$ is

$$\sigma_m = h^{\frac{1}{\alpha+m}}.$$

- The given problem instance is (g, g^a) from 1-SDH problem.
- Suppose (g, g^α, h) is simulated with $(g, g^a, g^{w(a+m)})$.
- Suppose the adversary forges a signature on $m^* \neq m$. We have

Simulator 知道 $\sigma_{m^*} = h^{\frac{1}{\alpha+m^*}} = g^{\frac{w(a+m)}{a+m^*}}$.
Trapdoor 信息

- Then the Trapdoor $f(x) = w(x + m)$ can be used to compute the following problem solution from the forged signature.

因为 Simulator 知道 Trapdoor, 因此其能在 $m^* \neq m$ 时抽取出一个困难

问题的解: $\xrightarrow{(m^*, g^{\frac{1}{a+m^*}})}$

如果 $h=g^b$, Simulator 不知道 b 和 Trapdoor, 无法计算出该解.



In the simulated scheme, all random elements (random group elements and random numbers) are not truly random elements. They are well computed following some ways from trapdoors.

假如用 secret key α 对消息 m 进行签名, 得到的签名为 $(g^r, g^{\frac{1}{r}}, g^{\frac{r}{\alpha+r}})$, 其中 r 是在生成签名时选取的一个随机数 (random number).

假如 simulator 在 signature simulation 中, 能从 \mathbb{Z}_p 中随机选择 $x = r$, 为什么不将签名简化为 $g^{\frac{1}{\alpha+r}}$ 呢?

如果归约中遇到了上述情况, 要么证明错了, 要么方案可以简化 (e.g. 减少元的个数).

7.2 Random Oracle

7.2.1 What is Random Oracle

假如一个方案在构造时用到了 Hash 函数 H , 在对其进行安全证明时需要用到 Random Oracles. 换句话说, 含有 Hash 函数的方案, 在安全证明时 Hash 函数 H 被设置为 RO(Random Oracle) \mathcal{O} , 在分析安全性时用的是 Random Oracle \mathcal{O} 而不是 Hash 函数 H , 我们相信当前者安全时后者就是安全的.(其实不一定满足, 但是应用密码学中一般直接用)

- 在证明一个方案的安全性时, **至少需要将一个 Hash 函数设置为 RO**, 并不需要将所有的 Hash 函数都设置为 RO (这取决于具体的安全归约的需求).
- 当一个 Hash 函数被设置为 RO 后, 敌手就不能从 simulator 处得到 Hash 函数算法了 (在安全证明中), 相反, 敌手只能通过询问 RO 得到 Hash 值.
- RO 是 ideal Hash function, 实际上是不存在的, 然而在 simulation 时并不需要考虑如何构造实现 ideal Hash function. 相反, simulator 主要考虑当敌手每次询问 x 时, 返回的 $H(x)$ 的值.
- RO 是做安全归约时的一种逆天的工具!

Hash Function vs Random Oracle 注意“表的长度”和“映射方式”.

Input	Hash Function	Output	Input	Random Oracle	Output
x_1		y_1	x_1		y_1
x_2		y_2	x_2		y_2
x_3	$H(x_i) = y_i$	y_3	x_3	Simulator	y_3
x_4		y_4	.		.
:		:	x_q		y_q

Hash 函数表的长度很大 (由定义, 例如 input space 为 160 bits, 可以输入 2^{160} 个 inputs), 通过 Hash 函数算法将 x_i 映射到 y_i . 而 RO 表的长度只有 q (远短于左边), 是安全归约中敌手询问的次数, x_i, y_i 之间的映射关系由 Simulator 决定.

区别:

- Knowledge. 给定一个输入 x , 如果 H 是一个 Hash 函数, 敌手知道 H 就可以自己计算 $H(x)$. 如果 H 被设置为 RO, 敌手只有 queries x to the RO 才能得到 $H(x)$ 的值, 而无法自己计算.
- Input. (见前面) Hash 函数和 RO 的 input space 相同, 但是 Hash 函数表的 input 的个数可以很多 (exponential), 而 RO 的 input 的个数为 q (polynomial), 即安全归约中敌手询问的次数, queries in polynomial time having polynomial number.
- Output. 给定一个 input, Hash 函数的 output 由 input 和 Hash 函数算法决定, 而 RO 的 output 由控制着 RO 的 simulator 决定.
- Representation. Hash 函数可以理解为一个通过 Hash 函数算法进行的映射, 从 input space 到 output space. RO 是没有映射的, 值由 simulator 决定, 并没有具体的映射规则.(如果说有, 那也是 simulator 制定的, 遵循归约算法).

7.2.2 How to use Random Oracle

Hash List 首先需要建立一个 Hash List 来记录每个 input x 和 output $y = H(x)$ 之间的关系, 这里 H 被设置为 RO.

- From the view of the adversary. the Hash List consists of $(x_1, y_1), (x_2, y_2), \dots, (x_q, y_q)$.
- From the view of the simulator. the Hash List could consist of $(x_1, y_1, S_1), (x_2, y_2, S_2), \dots, (x_q, y_q, S_q)$. 这里 S_i 是被用来计算 y_i 的 intermediate secret. intermediate secret 的例子见下面. 也有一些在计算 y 时不需要 S (这里书上表述不准确), 甚至每个 x 对应的 y 可以是完全随机的.

Example of intermediate secret in Hash List:

Let $H : \{0, 1\}^* \rightarrow \mathbb{G}$ be a random oracle.

- For a query on x_i , the simulator randomly chooses w_i .
- It computes and sets $y_i = H(x_i) = g^{w_i}$.
- The simulator adds (x_i, y_i, w_i) into the hash list.
- We define the intermediate secret $S_i = w_i$.

Note: The use of intermediate secret will be explained later.

Security Reduction with Random Oracle 在 Simulation 中, Simulator 需要增加 H-Query 阶段 (通常在 Setup 阶段之后), H-Query 阶段的作用是描述对手进行的 hash queries 和 responses.

H-Query. The adversary makes hash queries in this phase. The simulator prepares a hash list \mathcal{L} to record all queries and responses, where the hash list is empty at the beginning.

For a query x to the random oracle,

- If x is already in the hash list, the simulator responds to this query following the hash list. 需要保证对手重复询问时, 得到的结果的一致性.
- Otherwise, the simulator generates an intermediate secret S and uses it to compute y based on reduction algorithm. Then, the simulator responds to this query with $y = H(x)$ and adds the tuple (x, y, S) to the hash list. 并不是所有用到 RO 的安全归约都有 S .

注: H-Query phase 只在 Security Reduction 中出现, 不是在 Security Model 中出现! 安全模型中不能出现 Hash 函数.

7.2.3 Application of the RO

RO 怎么用 (三大类) 以及如何去掉 RO. 以下三个 RO 的应用, 在化成灰之前不可忘记!

1. We program the output and know a trapdoor. (simulator 在给出 output 时嵌入一些陷门.)
2. We program the output into a special space(simulator 能将 output 映射到一些特殊的集合/空间.)
3. We control the output and know the input.(simulator 能通过控制 output 来知道 input.) 如果不向我询问 x , 你就无法知道 $H(x)$, 因为 $H(x)$ 是由我决定, 但当你向我询问 x , 我就知道你涉及到了什么.

注: with RO 的安全归约相对容易 (适合先学习), without RO 通常是找到了代替 RO 的复杂技术.

Application 1: We program the output and know a trapdoor.

Let $H : \{0, 1\}^* \rightarrow \mathbb{G}$ be a random oracle. 由simulator控制

- For a query on x_i , the simulator randomly chooses w_i .
- It computes and sets $y_i = H(x_i) = g^{w_i}$. BLS签名的核心
- The simulator adds (x_i, y_i, w_i) into the hash list.
- The intermediate secret $S_i = w_i$ is the trapdoor.

Trapdoor: Even the simulator doesn't know a in (g, g^a) , it can still compute the element secret key 和 trapdoor 至少知道一个, 就能计算

Simulator 不知道 a 的情况下, 知道 trapdoor w_i , 计算 $H(x_i)^a$

$H(x_i)^a = (g^{w_i})^a = (g^a)^{w_i}$
如果 H 是真正的 Hash 函数, 在不知道 a 时, 无法计算 $H(x_i)$ 和 $H(x_i)^a$,
然而在 RO 模型中, H 是由 simulator 控制的一个随机预言机, 且 simulator 拥有陷门信息 w_i , 因此 simulator 能在不知道 a 的情况下, 利用 g^a 和 w_i 计算出 $H(x_i)^a$

Application 2: We program the output into a special space.

这是一种挖坑技术

Let $pk = (g, g^\alpha, h, H)$ and $sk = \alpha$. The signature on $m \in \mathbb{Z}_p$ is

$$\sigma_m = h^{\frac{1}{\alpha + H(m)}}.$$

Let $H : \{0, 1\}^* \rightarrow \mathbb{Z}_p$ be a random oracle.

- (g, g^α, h) is simulated with $(g, g^a, g^{w(a+z)})$, where w, z are random integers from \mathbb{Z}_p known by the simulator. (a is unknown.)
- Suppose the adversary can make one signature query on m after seeing public key.

simulator 能模拟出 m 对应的签名吗?
Mapping: The signature on m can be computed when $y \neq H(m)$ set z.
挖个坑, 隐藏自己: 能模拟出对应的签名, 让对手分不清是 simulated scheme 还是 real scheme
挖个坑, 让对手掉下去: 利用对手解决困难问题

Application 3: We control the output and know the input.

在基于“计算困难问题”的加密中非常重要

Let $H : \{0, 1\}^* \rightarrow \{0, 1\}^n$ be a random oracle.

- Suppose the adversary can break the following ciphertext and know the message with advantage 1.

$$CT = (g, g^a, g^b, H(g^{ab}) \oplus m).$$

- The adversary must ever make the query g^{ab} to the random oracle before it can break CT .
- The simulator knows that one of hash queries is equal to g^{ab} .
RO 是由 simulator 控制的, 因此当对手向 RO 询问时, simulator 知道对手询问了什么. 如果对手能攻破密文 CT , 那么 simulator 知道对手一定曾经询问过 $g^{\{ab\}}$, (某个 queries=g^{\{ab\}}), 否则对手没有办法(优势)攻破该密文. 而对手最多询问过多项式个 queries, simulator 能找到 $g^{\{ab\}}$ 的概率就变得很大.

7.2.4 *Oracle Response

设 H 是一个 random oracle, 签名查询次数 $q \ll \text{Hash}$ 查询 (random oracle 查询) 次数 q_H .

- 假如敌手进行了 q_H 次 random oracle 查询, 得到 $(x_1, y_1), (x_2, y_2), \dots, (x_{q_H}, y_{q_H})$.
- 假如 simulator 记录的 Hash List 为 $(x_1, y_1, c_1), (x_2, y_2, c_2), \dots, (x_{q_H}, y_{q_H}, c_{q_H})$, 其中 $c_i \in \{0, 1\}$ 是由 simulator 选取的 (不是随机选的).
- 假如敌手自适应地选取 $(x'_1, x'_2, \dots, x'_q, x^*)$. 它们都在 Hash List 中, 对应的 c_i 分别为 $(c'_1, c'_2, \dots, c'_q, c^*)$.

Question: How to set c_i for each x_i to obtain a high probability. 可观的, 不可忽略的值.

$$P = \Pr[c'_1 = c'_2 = \dots = c'_q = 0 \wedge c^* = 1]$$

有以下两种方法: 方法一较容易, 得到的概率较低. 方法二较为复杂, 能得到更高的概率.

Approach 1

Approach 1, 相对简单而概率较低

In the first approach, the simulator randomly chooses $i^* \in [1, q_H]$ and guesses that the adversary will output the i^* -th query as x^* . Then, for a query x_i , the simulator sets

simulator 建一个长为 q_H 的表, 猜测敌手会用表的第 i^* 个作为 x^* , 在设置 i^* 时不知道 x^* .

$$\begin{cases} c_i = 1 & \text{if } i = i^* \\ c_i = 0 & \text{otherwise} \end{cases}.$$

In this setting,

随机猜的一个设为1, 其余都是0

$$P = \Pr[c'_1 = c'_2 = \dots = c'_q = 0 \wedge c^* = 1]$$

is equivalent to successfully guessing which query is chosen as x^* .

Since the adversary makes q_H queries, and one of queries is chosen to be x^* , we have

simulator 是随机猜的, 敌手也是不可预测的, 随机猜中的 x^* 恰为敌手选的 x^* 的概率是 $1/q_H$, 此时 event 发生

$$P = \Pr[c'_1 = c'_2 = \dots = c'_q = 0 \wedge c^* = 1] = \frac{1}{q_H},$$

which is linear in the number of all hash queries.



复杂点，但结果概率高

Approach 2

抛一个不均匀的硬币

In the second approach, the simulator flips a bit $b_i \in \{0, 1\}$ in such a way that $b_i = 0$ occurs with probability P_b , and $b_i = 1$ occurs with probability $1 - P_b$. Then, for a query x_i , the simulator sets

选择多个 c_i 并设置为 1
剩下的设置为 0

$$\begin{cases} c_i = 1 & \text{if } b_i = 1 \\ c_i = 0 & \text{otherwise} \end{cases}.$$

Since all b_i are chosen according to the probability P_b , we have

$$\begin{aligned} P &= \Pr[c'_1 = c'_2 = \dots = c'_q = 0 \wedge c^* = 1] \\ &= \Pr[b'_1 = b'_2 = \dots = b'_q = 0 \wedge b^* = 1] \\ &= P_b^q(1 - P_b). \end{aligned}$$

The value is maximized at $P_b = 1 - 1/(1 + q)$, and then we get

$$P \approx \frac{1}{(1 + \frac{1}{q})^q \cdot q} = \frac{1}{e \cdot q}, \quad \text{这里 } e \text{ 是自然对数, } e=2.71\dots$$

which is linear in the number of chosen hash queries.

$q \ll qH$

$1/qH \ll 1/e \cdot q$

入门时，最好学习如何使用 RO 做证明，之后再考虑如何去掉 RO.