Introduction to Security Reduction

Lecture 6: Simulation and Solution

(Contradiction, Trapdoor, and Random Oracles)

Adversary

My IQ is up to 186.

My interest is breaking schemes.

You want me to help you solve problem?

Fool me first!

Journal of WoCrypt

Lecture 12: Flaws in Papers
Lecture 11: Revision of Security Reduction
Lecture 10: Security Proofs for Encryption (Computational)
Lecture 9: Security Proofs for Encryption (Decisional)
Lecture 8: Security Proofs for Digital Signatures
Lecture 7: Analysis (Towards A Correct Reduction)
Lecture 6: Simulation and Solution
Lecture 5: Difficulties in Security Reduction
Lecture 4: Entry to Security Reduction
Lecture 3: Preliminaries (Hard Problem and Secure Scheme)
Lecture 2: Preliminaries (Field, Group, Pairing, and Hash Function)
Lecture 1: Definitions (Algorithm and Security Model)

Computational Complexity Theory

Outline

1 Simulation: 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

Outline

Simulation and Contradiction

- Security reduction is full of contradictions.
- We must fully understand how these paradoxes are solved.
- Otherwise, the security reduction will be possibly incorrect.

We are going to introduce two very popular paradoxes.

- The secret key in the simulated scheme is unknown.
- The secret key in the simulated scheme is known.

Outline

Secret Key is Unknown

Taking a simulated signature scheme as the example.

In simulation,

- The simulated scheme is generated from a problem instance.
- The simulator doesn't know the secret key.
- Computing signatures for the adversary needs the secret key if the simulator follows the signing algorithm.

Secret Key is Unknown: Two Contradictions

The first Paradox

- In simulated scheme, the simulator can compute signatures on messages queried by the adversary without knowing the secret key.
- In real scheme, signatures must be computed with secret key. Otherwise, the signature scheme must be insecure!

Why the signature computing in simulated scheme is so special?

The second Paradox

- In simulated scheme, the simulator uses the signature generated by the adversary to solve hard problem.
- In simulated scheme, the simulator can also generate some signatures by itself but the simulator cannot use these signatures to solve hard problem.

Why the forged signature from the adversary is so special?

Outline

Secret Key is Known

- Most security reductions are programmed in the way that the secret key is unknown to the simulator.
- However, security reductions where the simulator knows the secret key can work for some special schemes.
- The points of paradox are completely different.

Secret Key is Known: Two Contradictions

Paradox in Signature

- In simulated scheme, the simulator uses the signature generated by the adversary to solve hard problem.
- In simulated scheme, the simulator knows the secret key and can compute signatures on all messages.

Why the forged signature from the adversary is so special?

Paradox in Encryption

- In simulated scheme, the simulator uses the adversary's decryption result to solve hard problem.
- In simulated scheme, the simulator knows the secret key and can decrypt all ciphertexts including the challenge ciphertext.

Why the decryption result from the adversary is so special?

Outline

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.
- We give several important simulations from the problem instance.

Note: The given 8 examples are very important in the literature.

Simulation Example (1)

Let $f(x)$ be a polynomial in $\mathbb{Z}_p[x]$ of degree n. Let $a \in \mathbb{Z}_p$ be a random and unknown exponent.

Problem: Given $g, g^a, g^{a^2}, \cdots, g^{a^n} \in \mathbb{G}$, and $f(x) \in \mathbb{Z}_p[x]$, we can compute the group element

$$g^{f(a)}.$$

Computation:The polynomial $f(x)$ can be written as

$$f(x) = f_n x^n + f_{n-1} x^{n-1} + \cdots + f_1 x + f_0,$$

where $f_i \in \mathbb{Z}_p$ is the coefficient of x^i for all $i \in [0, n]$. Therefore, this element is computable by computing

$$g^{f(a)} = \prod_{i=0}^{n} (g^{a^i})^{f_i}.$$

Simulation Example (2)

Let $f(x)$ be a polynomial in $\mathbb{Z}_p[x]$ of degree n. Let $a \in \mathbb{Z}_p$ be a random and unknown exponent.

Problem: Given $g, g^a, g^{a^2}, \cdots, g^{a^{n-1}} \in \mathbb{G}$, $f(x) \in \mathbb{Z}_p[x]$, and any $w \in \mathbb{Z}_p$ satisfying $f(w) = 0$, we can compute the group element

$$g^{\frac{f(a)}{a-w}}.$$

Computation: If $f(w) = 0$ for an integer $w \in \mathbb{Z}_p$, we have that $x - w$ divides $f(x)$, and

$$\frac{f(x)}{x-w}$$

is a polynomial of degree $n-1$, where all coefficients are computable. Therefore, this element is computable because $\frac{f(x)}{x-w}$ is a polynomial of degree $n-1$.

Simulation Example (3)

Let $f(x)$ be a polynomial in $\mathbb{Z}_p[x]$ of degree n. Let $a \in \mathbb{Z}_p$ be a random and unknown exponent.

Problem: Given $g, g^a, g^{a^2}, \cdots, g^{a^{n-1}} \in \mathbb{G}$, $f(x) \in \mathbb{Z}_p[x]$, and any $w \in \mathbb{Z}_p$, we can compute the group element

$$g^{\frac{f(a)-f(w)}{a-w}}.$$

Computation: It is easy to see that $x - w$ divides $f(x) - f(w)$ and then

$$\frac{f(x) - f(w)}{x - w}$$

is a polynomial of degree $n - 1$, where all coefficients are computable. Therefore, this element is computable because $\frac{f(x)-f(w)}{x-w}$ is a polynomial of degree $n - 1$.

Simulation Example (4)

Let $f(x)$ be a polynomial in $\mathbb{Z}_p[x]$ of degree n. Let $a \in \mathbb{Z}_p$ be a random and unknown exponent.

Problem:Given $g, g^a, g^{a^2}, \cdots, g^{a^{n-1}}, g^{\frac{f(a)}{a-w}} \in \mathbb{G}$, $f(x) \in \mathbb{Z}_p[x]$, and any $w \in \mathbb{Z}_p$ satisfying $f(w) \neq 0$, we can compute the group element

$$g^{\frac{1}{a-w}}.$$

Computation: Since $x - w$ divides $f(x) - f(w)$, we have

$$\frac{f(x)}{x-w} = \frac{f(x)-f(w)+f(w)}{x-w} = \frac{f(x)-f(w)}{x-w} + \frac{f(w)}{x-w},$$

$$\frac{f(x)}{x-w} = f'_{n-1}x^{n-1} + f'_{n-2}x^{n-2} + \cdots + f'_1 x + f'_0 + \frac{d}{x-w}$$

for coefficients $f'_i \in \mathbb{Z}_p$, which are computable, and $d = f(w)$ which is a nonzero integer. Therefore, this element is computable by computing

$$g^{\frac{1}{a-w}} = \left(\frac{g^{\frac{f(a)}{a-w}}}{\prod_{i=0}^{n-1} (g^{a^i})^{f'_i}} \right)^{\frac{1}{d}}.$$

Simulation Example (5)

Let $f(x)$ be a polynomial in $\mathbb{Z}_p[x]$ of degree n. Let $a \in \mathbb{Z}_p$ be a random and unknown exponent.

Problem:Given $g, g^a, g^{a^2}, \cdots, g^{a^{n-1}}, h^{as} \in \mathbb{G}, f(x) \in \mathbb{Z}_p[x]$ and $e(g,h)^{f(a)s} \in \mathbb{G}_T$ where $f(0) \neq 0$, we can compute the group element

$$e(g,h)^s.$$

Computation: Let $f(x) = f_n x^n + f_{n-1}x^{n-1} + \cdots + f_1 x + f_0$. This polynomial can be rewritten as

$$f(x) = x(f_n x^{n-1} + f_{n-1}x^{n-2} + \cdots + f_1) + f_0.$$

Since $f_0 = f(0) \neq 0$, this element is computable by computing

$$e(g,h)^s = \left(\frac{e(g,h)^{f(a)s}}{e\left(h^{as}, \prod_{i=0}^{n-1}(g^{a^i})^{f_{i+1}}\right)} \right)^{\frac{1}{f_0}}.$$

Simulation Example (6)

Problem:Given $g, g^a \in \mathbb{G}$, we can compute a pair $\left(g^{\frac{1}{a+r}},\ g^r\right)$ for an integer $r \in \mathbb{Z}_p$, where a correct pair, denoted by (u, v), satisfies

$$e\left(u,\ g^a \cdot v\right) = e(g, g).$$

Computation: We solve this problem by randomly choosing $r' \in \mathbb{Z}_p^*$ and computing

$$\left(g^{\frac{1}{r'}}, g^{r'-a}\right).$$

Let $r = r' - a \in \mathbb{Z}_p$. We have

$$\left(g^{\frac{1}{a+r}},\ g^r\right) = \left(g^{\frac{1}{a+r'-a}},\ g^{r'-a}\right) = \left(g^{\frac{1}{r'}}, g^{r'-a}\right),$$

and thus the computed pair is the solution to the problem instance.

Simulation Example (7)

Problem:Given $g, g^a \in \mathbb{G}$ and $w \in \mathbb{Z}_p$, we can compute a pair $\left(g^{\frac{r}{a+w}},\ g^r\right)$ for an integer $r \in \mathbb{Z}_p$, where a correct pair, denoted by (u, v), satisfies

$$e\left(u,\ g^a \cdot g^w\right) = e(v, g).$$

Computation: We solve this problem by randomly choosing $r' \in \mathbb{Z}_p$ and computing

$$\left(g^{r'},\ g^{r'(a+w)}\right).$$

Let $r = r'(a + w) \in \mathbb{Z}_p$. We have

$$\left(g^{\frac{r}{a+w}},\ g^r\right) = \left(g^{\frac{r'(a+w)}{a+w}},\ g^{r'(a+w)}\right) = \left(g^{r'},\ g^{r'(a+w)}\right),$$

and thus the computed pair is the solution to the problem instance.

Simulation Example (8)

Problem:Given $g, g^a, g^b \in \mathbb{G}$ and $w \in \mathbb{Z}_p^*$, we can compute a pair

$$\left(g^{ab} g^{(wa+1)r},\ \ g^r\right)$$

for an integer $r \in \mathbb{Z}_p$, where a correct pair, denoted by (u, v), satisfies

$$e(u,\ g) = e(g^a, g^b) e(g^{wa} g, v).$$

Computation: We solve this problem by randomly choosing $r' \in \mathbb{Z}_p$ and computing

$$\left(g^{-\frac{1}{w}b + wr'a + r'}, g^{-\frac{b}{w} + r'}\right).$$

Let $r = -\frac{b}{w} + r' \in \mathbb{Z}_p$. We have

$$\left(g^{ab} g^{(wa+1)r},\ g^r\right) \quad = \quad \left(g^{ab} g^{(wa+1)(-\frac{b}{w}+r')},\ g^{-\frac{b}{w}+r'}\right) = \left(g^{-\frac{1}{w}b + wr'a + r'},\ g^{-\frac{b}{w}+r'}\right),$$

and thus the computed pair is the solution to the problem instance.

Outline

Trapdoor

- Trapdoor is the additional secret in simulated scheme for computing response when the secret key is unknown.
- We program the simulation in the way that computing using secret key can be replaced with computing using the trapdoor.
- The trapdoor is also used to extract problem solution from the adversary's attack on the simulated scheme.

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

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

- Then the function $f(x) = w(x+m)$ is called Trapdoor.

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

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

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

$$\left(m^*, g^{\frac{1}{a+m^*}}\right)$$

Trapdoor

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.

Important Observation: Let the signature on m with secret key α be

$$\left(g^r, g^{\frac{1}{r}}, g^{\frac{r}{\alpha+m}}\right),$$

where r is a random number chosen in signature generation.

Suppose the simulator is able to randomly choose $x = r$ from $\mathbb{Z}_p$ in the signature simulation. Why not simplify the signature construction into

$$g^{\frac{1}{\alpha+m}} ?$$

Outline

Random Oracle

Suppose a proposed scheme is constructed using a hash function H.
We can represent the proposed scheme with the following combination

$$Scheme + H.$$

Roughly speaking, a security proof with random oracles for this scheme is the security proof for the following combination

$$Scheme + \mathcal{O},$$

where the hash function H is set as a random oracle $\mathcal{O}$.

- We do not analyze the security of $Scheme + H$ but $Scheme + \mathcal{O}$.
- We believe that $Scheme + H$ is secure if $Scheme + \mathcal{O}$ is secure.

Outline

Random Oracle

- Proving security of a proposed scheme in the random oracle model requires at least one hash function to be set as a random oracle. However, it doesn't need all hash functions to be set as random oracles (it depends on security reduction).
- When a hash function is set as a random oracle, the adversary will not receive the hash function algorithm from the simulator in the security proof. The adversary can only access the "hash function" by asking the random oracle.
- A random oracle is an ideal hash function. However, we do not need to construct such an ideal hash function in the simulation. Instead, the main task in the simulator is to think how to respond to each input query.

Note: Random Oracle is a tool for programming security reduction.

Hash Function vs Random Oracle

Input	Hash Function	Output
x_1		y_1
x_2		y_2
x_3	$H(x_i) = y_i$	y_3
x_4		y_4
$\vdots$		$\vdots$

Input	Random Oracle	Output
x_1		y_1
x_2		y_2
x_3	Simulator	y_3
$\vdots$		$\vdots$
x_q		y_q

- **Knowledge.** Given any input x, if H is a hash function, the adversary knows H and so knows how to compute $H(x)$. However, if H is set as a random oracle, the adversary doesn't know $H(x)$ unless it queries x to the random oracle.
- **Input.** Hash functions and random oracles have the same input space. Although the number of inputs to a hash function is exponential, the number of inputs to a random oracle is polynomial. This is due to queries in polynomial time having polynomial number.

Hash Function vs Random Oracle

Input	Hash Function	Output
x_1		y_1
x_2		y_2
x_3	$H(x_i) = y_i$	y_3
x_4		y_4
$\vdots$		$\vdots$

Input	Random Oracle	Output
x_1		y_1
x_2		y_2
x_3	Simulator	y_3
$\vdots$		$\vdots$
x_q		y_q

- **Output.** Given an input, the output from a hash function is determined by the input and the hash function algorithm. However, the output from a random oracle for an input is defined by the simulator who controls the random oracle.
- **Representation.** A hash function is a mapping from an input space to an output space according to the hash function algorithm. A random oracle is a virtual hash function composed of input and output only. The random oracle doesn't have any mapping rule.

Outline

Hash List

A hash list is to record the relations between inputs x and outputs $y = H(x)$, when H is set as a random oracle.

From the view of the adversary, the hash list consists of

$$(x_1, y_1), (x_2, y_2), \cdots, (x_q, y_q).$$

From the view of the simulator, the hash list could consist of

$$(x_1, y_1, S_1), (x_2, y_2, S_2), \cdots, (x_q, y_q, S_q),$$

where S_i is the intermediate secret used to compute y_i.

Note: Some reductions must set a hash function as a random oracle but they do not need S to compute y.

Example of Secret in Hash List

Let $H : \{0,1\}^* \to \mathbb{G}$ 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

The simulator adds **H-Query** phase (usually after the Setup phase) in the simulation to describe hash queries and 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.
- 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.

Note: The **H-Query** phase only appears in the security reduction, and it should not appear in the security model.

Outline

Applications of Random Oracle

Input	Random Oracle	Output
x_1		y_1
x_2		y_2
x_3	Simulator	y_3
$\vdots$		$\vdots$
x_q		y_q

There are three different ways in the application. They are

- We program the output and know a trapdoor.
- We program the output into a special space.
- We control the output and know the input.

Applications of Random Oracle

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

Let $H : \{0,1\}^* \to \mathbb{G}$ 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.
- 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

$$H(x_i)^a = (g^{w_i})^a = (g^a)^{w_i}$$

Applications of Random Oracle

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

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 $H : \{0,1\}^* \to \mathbb{Z}_p$ 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.

Mapping: The signature on m can be computed when $y = H(m) = z$.

Applications of Random Oracle

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

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

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

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

- 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}.

Outline

Question

Let H be a random oracle and $q << q_H$.

- Suppose the adversary queries and obtains q_H pairs as follows.

$$(x_1, y_1), (x_2, y_2), \cdots, (x_{q_H}, y_{y_H}).$$

- Suppose the hash list recorded by the simulator is

$$(x_1, y_1, c_1), (x_2, y_2, c_2), \cdots, (x_{q_H}, y_{y_H}, c_{q_H}),$$

  where $c_i \in \{0, 1\}$ is chosen by the simulator in any way.
- Suppose the adversary adaptively picks $(x'_1, x'_2, \cdots, x'_q, x^*)$.
- Let the corresponding c_i in the pick be $(c'_1, c'_2, \cdots, c'_q, c^*)$.

Question: How to program c_i for each x_i to obtain a high probability

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

Approach 1

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

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

In this setting,

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

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

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

which is linear in the number of all hash queries.

Approach 2

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

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

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

$$\begin{aligned} P &= \Pr[c'_1 = c'_2 = \cdots = c'_q = 0 \wedge c^* = 1] \\ &= \Pr[b_1 = b_2 = \cdots = b_q = 0 \wedge b^* = 1] \\ &= P_b^q(1 - P_b). \end{aligned}$$

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

$$P \approx \frac{1}{(1+\frac{1}{q})^q \cdot q} = \frac{1}{e \cdot q},$$

which is linear in the number of chosen hash queries.

Oh, Dammit! You might be able to fool me too!


END