Introduction to Security Proof of Cryptosystems

D. J. Guan

November 16, 2007

Abstract

Provide proof of security is the most important work in the design of cryptosystems. Problem reduction is a tool to prove the security of cryptosystems. However, most cryptosystems have not been able to find a suitable way to prove their security based on the standard notions of problem reduction. Recently random oracle model has been proposed for the prove of the security of cryptosystems. In this talk, I will use examples to illustrate the prove of the security of the cryptosystems by using random oracle model.

Introduction

A cryptosystem $\mathcal{C}$ is defined by (P, C, K, E, D), where

1. P is a finite set of *pliantexts*,
2. C is a finite set of *cyphertext*,
3. K is a finite set of *keys*,
4. E is a finite set of *encryption functions*, and
5. D is a finite set of *decryption functions*.

RSA Cryptosystem

$n = p \cdot q$; p, q: large distinct primes

$P = C = \mathbf{Z}_n^*$

$K = \{(n, e, d) \mid e \cdot d \equiv 1 \pmod{\phi(n)}\}$

$e_k(x) = x^e \bmod n$

$d_k(y) = y^d \bmod n$

To prove a cryptosystem $\mathcal{C}$ is secure we must

1. formally define the security requirements of $\mathcal{C}$, and
2. show that $\mathcal{C}$ satisfies the security requirements.

The security of RSA

RSA is secure if for every $y \in C$ it is *hard* to compute x,

$$x^e \equiv y \pmod{n}.$$

By the above definition, RSA is not secure, since $1^e \equiv 1 \pmod{n}$

Any acceptable security definitions to *make* RSA secure?

Is RSA secure?

How to Show a Problem is Hard?

$L = 4600000000 \times 365 \times 86400 \times 4700000000 \times 1000000000 < 10^{36}$

> **Show that it needs at least L computational steps to solve the problem.**

Time complexity $T(n)$, or space complexity $S(n)$, of a problem.

Lower bound of a problem.

Very few problems' lower bounds have been proved.

How to Show a Problem is Hard?

> **Show that the most efficient algorithm known today to solve the problem needs at least L computational steps.**

Some problems are *hard* to solve by the best known algorithms.

NP-Complete problems:
No polynomial time algorithms have been discovered.

Factoring an integer, Discrete logarithm problem:
$O(e^{\alpha(\log n)^{\beta}(\log\log n)^{(1-\beta)}})$, where $\beta = 1/3$ for number field sieve.

Problem Reduction

1. select a problem $\mathcal{P}$ which is known to be *hard* to solve,
2. reduce the problem $\mathcal{P}$ to the security of $\mathcal{C}$.

Since $\mathcal{P}$ is hard to solve, the cryptosystem $\mathcal{C}$ is hard to break.

This is called *problem reduction*.

Problem Reduction

$\mathcal{A} \leq_p \mathcal{B}$: problem $\mathcal{A}$ is polynomial time reducible to problem $\mathcal{B}$ if

1. Given an instance $\mathcal{I}_A$ of $\mathcal{A}$ we can construct an instance $\mathcal{I}_B$ of $\mathcal{B}$ such that the solution $\mathcal{B}(\mathcal{I}_B)$ can be converted into the solution $\mathcal{A}(\mathcal{I}_A)$.
2. Both the construction of the instance and the conversion of the solution can be done in *polynomial time*.

This is called *many-one* reduction.

If $\mathcal{A} \leq_p \mathcal{B}$ then problem $\mathcal{B}$ is *harder* to solve than $\mathcal{A}$.

$\mathcal{A} \equiv_p \mathcal{B}$ if $\mathcal{A} \leq_p \mathcal{B}$ and $\mathcal{B} \leq_p \mathcal{A}$.

Problem Reduction, an Example

Let $n = p \cdot q$, where p and q are unknown distinct primes.

Problem $\mathcal{A}$: factor n

Problem $\mathcal{B}$: solve $x^2 \equiv a \pmod{n}$ for x

Theorem 1 *$\mathcal{A} \equiv_p \mathcal{B}$.*

$\mathcal{A} \leq_p \mathcal{B}$: Assume $\mathcal{B}$ can be solved in polynomial time, Then the following is a polynomial time probabilistic algorithm to factor n.

```
input: an integer $n$
output: a factor of $n$
method:
    1. Randomly select an integer $y$.
    2. If $d = \gcd(y, n) > 1$ then $d$ is a factor of $n$; stop.
    3. Compute $a = y^2 \bmod n$.
    4. Solve $x^2 \equiv a \pmod{n}$ for $x$ by using algorithm $\mathcal{B}$.
    5. If $x = \pm y$ then fail; otherwise $\gcd(x + y, n)$ is a factor of $n$.
```

$\mathcal{B} \leq_p \mathcal{A}$: Assume $\mathcal{A}$ can be solved in polynomial time. The following is an algorithm to solve $x^2 \equiv a \pmod{n}$ for x in polynomial time.

input: two integers a and n

output: x, $x^2 \equiv a \pmod{n}$

method:

1. Factor $n = pq$ by using algorithm $\mathcal{A}$.
2. Solve $x_p^2 \equiv a \pmod{p}$ for x_p.
3. Solve $x_q^2 \equiv a \pmod{q}$ for x_q.
4. Compute x such that $x \bmod p = x_p$ and $x \bmod q = x_q$ by using Chinese remainder theorem.

Turing Reduction and Oracle Computation

An *oracle* $\mathcal{O}_{\mathcal{B}}$ is a *hypothetical* algorithm for solving problem $\mathcal{B}$.

To show that a problem $\mathcal{A}$ is solvable by a Turing machine with oracle $\mathcal{O}_{\mathcal{B}}$ is the same as to show $\mathcal{A} \leq_p \mathcal{B}$

The oracle can be *called* many times.

Public-Key Cryptosystem

A *public-key encryption scheme* consists of

1. key generation algorithm $\mathcal{K}$:
   It is a probabilistic algorithm. On input the security parameter w, $\mathcal{K}$ generates public key k_p and secrete key k_s.
2. encryption algorithm $\mathcal{E}_{k_p}$:
   On input a message m, $\mathcal{E}_{k_p}$ computes the ciphertext $c = \mathcal{E}_{k_p}(m)$. The encryption algorithm may be probabilistic. That is, in addition to the message m, it can take a random number r as its input.
3. decryption algorithm $\mathcal{D}_{k_s}$:
   On input a ciphertext c, $\mathcal{D}_{k_s}$ computes the original message m.

Semantically Secure

A public key cryptosystem is *semantically secure* if no polynomial-time attacker can learn any information about the plaintext from the ciphertext.

Define a game $\mathcal{G}$ played by two players $\mathcal{A}$ and $\mathcal{B}$.

1. $\mathcal{A}$ selects two messages m_0 and m_1, and sends them to $\mathcal{B}$.
2. $\mathcal{B}$ randomly selects one of the message $m_j \in \{m_0, m_1\}$ to encode. The ciphertext $c = \mathcal{E}_{k_p}(m_j)$ is then given to the $\mathcal{A}$.
3. $\mathcal{A}$ determines which one of the messages whose encryption is c, that is. $c = \mathcal{E}_{k_p}(m_0)$ or $c = \mathcal{E}_{k_p}(m_1)$.

$\mathcal{A}$ wins if $\mathcal{A}$ can determine the message whose encryption is c; otherwise $\mathcal{B}$ wins.

Semantically Secure

A cryptosystem is *semantically secure* if the game $\mathcal{G}$ is a fair game for both players.

That is, $\mathcal{A}$ has no strategy with the probability of winning $\geq \frac{1}{2} + \epsilon$ for any $\epsilon > 0$.

In the above game, $\mathcal{A}$ should be regarded as a *polynomial-time probabilistic Turing machine*.

That is, $\mathcal{A}$ can use only a limited computing resources.

Random Oracles

As mentioned above, it is difficult to find a problem reduction to show that a cryptosystem is secure.

In some cases, we can show that a cryptosystem is secure by adding more oracles.

These additional oracles are usually *hash functions*.

On input x, the hash function usually responds a random number.

On some specific input, the response is a special number which is useful to solve the hard problem.

An Example

Define a cryptosystem $\mathcal{C}$ as follows.

1. public functions:

   f: one-way trap-door function, and

   h: random number generator.
2. encryption: $E_{k_p}(m, r) = (f(r), m \oplus h(r))$, where r is a random number.
3. decryption: (exercise)

The one-way trap-door function f is considered to be *hard* to invert without the trap-door (key).

Theorem 2 *The cryptosystem $\mathcal{C}$ is semantically secure.*

Assume that there is an attacker $\mathcal{A}$ with probability of winning $\frac{1}{2}+\epsilon$, then there is a Turing machine with random oracle $\mathcal{O}$ (or a hash function h) which can invert the one-way trap-door function f with probability of success ϵ.

There are two oracles, the attacker $\mathcal{A}$ and the random oracle $\mathcal{O}$.

Note that ϵ should be non-negligible. That is, $\epsilon \geq 1/p(w)$, where $p(w)$ is a polynomial of the security parameter w.

Construct a Turing machine, $\mathcal{T}$ with oracle $\mathcal{A}$ and $\mathcal{O}$.

On input f and y, $\mathcal{T}$ finds r such that $f(r) = y$.

1. The oracle $\mathcal{O}$.
   Whenever $\mathcal{A}$ makes a query to $h(r)$
   (a) If $f(r) = y$ then print r and stop.
   (b) Otherwise return a random number t.
2. The attacker $\mathcal{A}$.
   Play the role of $\mathcal{A}$ in the game $\mathcal{G}$.
3. After receiving the two messages m_0 and m_1 from $\mathcal{A}$,
   $\mathcal{T}$ randomly selects an s and let $c = (y, s)$.

We need to show that

1. the random oracle $\mathcal{O}$ (or a hash function h) is consistent, and
2. $c = (y, s)$ is the encryption of m_0 or m_1.
   (Recall that $E_{k_p}(m, r) = (f(r), m \oplus h(r)))$
3. the algorithm will succeed with probability ϵ in inverting the function f.

Digital Signature Scheme

A *signature scheme* consists of

1. key generation algorithm $\mathcal{K}$
   It is a probabilistic algorithm. On input the security parameter w, $\mathcal{K}$ generates public key k_p and secrete key k_s.
2. signing algorithm $\mathcal{S}_{k_s}$
   On input a message m, $\mathcal{S}_{k_s}$ computes the signature $u = \mathcal{S}_{k_s}(m)$. The signing algorithm may be probabilistic.
3. verification algorithm $\mathcal{V}_{k_p}$
   On input a signature u and m, $\mathcal{V}_{k_p}(u, m) = 1$ if and only if u is a valid signature of m.

An Example

Define a signature scheme $\mathcal{S}$ as follows.

1. public functions:

   f: one-way trap-door function

   h: one-way hash function

2. signature: $u = S_{k_s}(m) = f^{-1}(h(m))$
3. verification: $V_{k_p}(u, m) = 1$ if and only if $f(u) = h(m)$.

Polynomial Time Attacker of $\mathcal{S}$

An attacker of the signature scheme $\mathcal{S}$ is an probabilistic polynomial time algorithm $\mathcal{F}$ that can compute a valid signature u for a message m.

The attacker $\mathcal{F}$ must call hash function for the signature of the message m.

In addition, $\mathcal{F}$ can also ask for the signature of messages other than m.

Theorem 3 *The signature scheme $\mathcal{S}$ is secure.*

Assume that there is an attacker $\mathcal{F}$ with probability of success ϵ, then there is a random oracle $\mathcal{O}$ (or a hash function h) that can be used to invert the one-way trap-door function f with probability of success ϵ/r, where r is the number of queries to the random oracle.

Assume that $\mathcal{F}$ makes r queries to the random oracle, and that it must query $h(m)$ before it can compute the signature of m.

Given f and y, find r such that $f(r) = y$.

$\mathcal{O}$ for $h(r)$:

1. Randomly chooses $s \in [1, r]$.
2. Checks if $f(r) = y$ or not. If it is, then print r and stop; otherwise
   (a) if $i = s$ then return y.
   (b) if $i \neq s$ then choose a random number t and return $f(t)$.

$\mathcal{S}$ for the signature:

1. if $m = m_s$ then fail; stop;
2. if $m \neq m_s$, it returns t;

We need to show that the random oracle is consistent and the algorithm will succeed with probability ϵ/r in inverting the function f.

$$\begin{array}{lcccccc}
m: & m_1 & m_2 & \cdots & m_s & \ldots & m_r \\
u: & t_1 & t_2 & \cdots & r & \ldots & t_r \\
h(m): & f(t_1) & f(t_2) & \cdots & y & \ldots & f(t_r)
\end{array}$$

Recall that the signature of m is $u = f^{-1}(h(m_i))$.

Summary of the Random Oracle Method

To prove that a cryptosystem or a signature scheme is secure, we need to do the following steps.

1. Define the security requirements of the cryptosystem formally.
2. Find a “hard” problem.
3. Make assumptions that the attacker must do.
4. Use the attacker to solve the hard problem.

In the above, (1), (2), and (4) are the same as the standard problem reduction.

If we make no assumptions on the attacker to break the cryptosystem, then the proof can be converted into a standard problem reduction.

In standard problem reduction, we take the attacker as a *black box*.

The theorem we proved in this way is the *strongest* one.

As mentioned before, standard problem reduction is difficult to find for almost all cryptosystems, signature schemes, and cryptographic protocols.

However, if we make some assumptions on the attacking algorithm, then we can prove the security of some cryptosystems, signature schemes, or cryptographic protocols.

The more assumptions we make, the weaker the theorems are.

Assumptions we can make on the attacking algorithm include

1. calls to the hash function,
2. calls to the signature function,
3. etc.

In addition to the above assumptions, we can also rewrite the hash function, etc.

In doing these, we must must make everything consistent.

We should also make these functions provided by us look *the same as* the real functions to the attacking algorithm.

Since it is believed that the attacking algorithm should work for every reasonable hash functions, the attacking algorithm should also work for the function provided by us.

What have we proved by using the random oracle model and play the role of generating the values for the hash function?

We have shown that there exists a hash functions which, together with the attacking algorithm, can allow us to solve some hard problems.

Since we believe that the attacking algorithm works for any reasonable hash functions, the attacking algorithm, if it exists, can be used to solve hard problems.

References

1. Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the First ACM Conference on computer and Communication*, pages 62–73, New York, November 1993. ACM Press.

2. Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28:270–299, 1984.