CS6222 Cryptography

Topic: Introduction | Date: Sep, 2024
Lecturer: Wei-Kai Lin (TA: Arup Sarker) | Scriber: Lei Gong, Chen Gong

1 One Way Function

1.1 Definition of One-Way Functions

Definition, One-Way Functions.

A deterministic function $f : \{0,1\}^* \to \{0,1\}^*$ is a one way function if it satisfies the following.

- Easy to compute: There is a PPT C that computes $f(x)$ on all inputs $x \in \{0,1\}^*$.
- Hard to invert: No nuPPT adversary $\mathcal{A}$, there exists a negligible function ϵ that for any $n \in \mathbb{N}$:

$$\Pr[\mathcal{A}(1^n, f(x)) \in f^{-1}(f(x))] = 1.$$

$\mathcal{A}$ represents an adversary (often a probabilistic polynomial-time (PPT) algorithm). The adversary's goal is to invert the function f. The adversary takes 1^n (which is just a string of 1's of length n, often used to indicate the input size) and y, which is the output of the function $f(x)$, and tries to find the original input x. Besides, the definition of $f^{-1}(y)$ is that,

$$f^{-1}(y) = \{x' \in \{0,1\}^n : f(x') = y\}.$$

The above definition is standard in literature, but it is still hard to construct: any adversary can only invert a tiny fraction. Many natural candidates, such as factoring, does not meet this. We relax it, and define the Weak One-Way Function.

1.2 Definition of Weak One-Way Function

Definition, Weak One-Way Function.

A deterministic function $f : \{0,1\}^* \to \{0,1\}^*$ is a weak one-way function if it satisfies the following.

- Easy to compute: There is a PPT C that computes $f(x)$ on all inputs $x \in \{0,1\}^*$.
- Hard to invert: There exists a polynomial $q : \mathbb{N} \to \mathbb{N}$ such that for any nuPPT adversary $\mathcal{A}$, for sufficiently large $n \in \mathbb{N}$,

$$\Pr[x \leftarrow \{0,1\}^n; y \leftarrow f(x) : f(\mathcal{A}(1^n, y)) = y] \leq 1 - 1/q(n).$$

It is noticed that $1 - 1/q$ is the same for all adv $\mathcal{A}$, but in the strong OWF, the prob. The ϵ is different and depends on $\mathcal{A}$.

1.3 Example: Some Functions Are Easy To Invert

For any string $x \in \{0,1\}^*$, there are many easy-to-compute functions:

- Identity, $f(x) := x$
- Constant, $f(x) := 0$
- Constant output length, $f : \{0,1\}^* \to \{0,1\}^4$

1.4 Output Length of OWF

Lemma, Output Length of OWF.

> We define that $f : \{0,1\}^n \to \{0,1\}^l$ is OWF for $l = l(n)$. Then, we can construct a new function, i.e., $g : \{0,1\}^n \to \{0,1\}^n$ is also OWF.

Proof: We are given a one-way function f that maps n-bit inputs to $l(n)$-bit outputs, where $l(n)$ could be different from n. Our goal is to construct a new function $g : \{0,1\}^n \to \{0,1\}^n$ that has the same input and output size (i.e., n-bit inputs and n-bit outputs) and is also a one-way function.

Case 1: $l(n) \geq n$. In this case, the output length of f is greater than or equal to n, meaning f maps to a space that has at least as many possible outputs as inputs. We can define g as:

$$g(x) = \text{first } n \text{ bits of } f(x)$$

That is, $g(x)$ takes the first n bits of the output of $f(x)$. Since f is one-way, finding a preimage of $g(x)$ (i.e., finding x from $g(x)$) would be as hard as inverting $f(x)$, because if you can invert $g(x)$, you can also invert $f(x)$ by reconstructing its output and checking consistency.

Thus, $g(x)$ is also a one-way function.

Case 2: $l(n) < n$. In this case, the output length of f is less than n, meaning f compresses the input space, potentially making inversion easier. To deal with this, we can construct a new function g that incorporates both $f(x)$ and some of the input x itself to ensure that g is still one-way.

Define g as follows:

$$g(x) = (f(x), \text{first } n - l(n) \text{ bits of } x)$$

In this construction:

- $f(x)$ is the output of the original one-way function, which has $l(n)$ bits.
- The second part consists of the first $n - l(n)$ bits of the input x, ensuring that the total length of the output is exactly n bits.

To invert $g(x)$, an adversary would need to invert $f(x)$, which is computationally hard (since f is a one-way function), and also recover the first $n - l(n)$ bits of x, which are directly part of $g(x)$.

Thus, inverting g would require inverting f, which by assumption is difficult. Therefore, $g(x)$ is also a one-way function.

In both cases—whether $l(n) \geq n$ or $l(n) < n$—we can construct a new function $g : \{0,1\}^n \to \{0,1\}^n$ from the given one-way function $f : \{0,1\}^n \to \{0,1\}^l$, and this new function g is also a one-way function.

1.5 Example: Any PRG is one-way

Theorem: If $g : \{0,1\}^* \to \{0,1\}^*$ is a PRG, then g is a OWF.

Proof: Suppose g is a PRG, which implies it stretches input by a polynomial amount and is computationally indistinguishable from a random string of the same length. We aim to show that $f(x) = g(x)$ is a one-way function (OWF).

1. f is a polynomial-time function because g is polynomial-time computable.
2. Assume for contradiction that there exists a polynomial-time algorithm A, such that for infinitely many $n \in \mathbb{N}$,

$$\Pr[A(1^n, y) \in g^{-1}(y) : y = g(x)] \geq \frac{1}{p(n)}.$$

   Construct an algorithm $B(t)$ as follows:

$$B(t) = \begin{cases} g(x), & \text{if } t \in g(\{0,1\}^n), \\ U = \{0,1\}^{2n}, & \text{if } t \in U_{2n}. \end{cases}$$

   Steps:

   (a) $x' \leftarrow A(t)$

   (b) Output: $g(x') = t$

   Then, we have the following probabilities:

$$\Pr[B(t) = 1 \mid t \leftarrow g(x)] = \frac{1}{p(n)},$$

   and

$$\Pr[B(t) = 1 \mid t \leftarrow U_{2n}] \leq \frac{2^n}{2^{2n}} = \frac{1}{2^n}.$$

   This leads to a contradiction because if A can invert g with non-negligible probability, it would contradict the pseudo-randomness of g.

1.6 Fact: $\exists OWF \Rightarrow NP \neq P$

Theorem: Suppose that there exists a one-way function (OWF) f, then there exists a language $L \in NP$ such that $L \notin P$.

Proof: The idea is to:

1. Define a language $L \in NP$ using f.
2. Assume for contradiction that $NP = P$, so there exists a polynomial-time algorithm D that decides L.
3. Construct a reduction that uses D to invert f, leading to a contradiction.

A first attempt is to define the language as:

$$L := \{f(x) : x \in \{0,1\}^n\}.$$

This implies $L \in NP$, since for every $y \in L$, the witness of y is x such that $f(x) = y$. However, this L does not help invert f, because D only outputs a single bit, giving no information about x. Moreover, if f is a permutation (i.e., a one-way permutation), deciding L becomes trivial.

To address this, we augment L with all prefixes of x, defining:

$$L := \{(f(x), x[1...i]) : x \in \{0,1\}^n, i \in [|x|]\}.$$

Thus, for any $y = f(x)$, the reduction can retrieve the first bit of x by running D iteratively as follows:

$$D(y, b_1), D(y, b_1 b_2), D(y, b_1 b_2 b_3), \ldots, D(y, b_1 b_2 \ldots b_n).$$

In each iteration, the next bit b_i of x is learned if $D(y, b_1 \ldots b_{i-1} b_i)$ accepts. This process reconstructs x, which contradicts the assumption that f is a one-way function. Hence, $NP \not\subseteq P$.

Furthermore, this proof can be extended to imply that $NP \not\subseteq BPP$, since similar reasoning applies for probabilistic polynomial time.

2 Primes and Factoring

Prime Number Theorem: Define $\pi(x)$ as the number of primes $\leq x$. The Prime Number Theorem (PNT) states that primes are dense.

$$\pi(N) \sim \frac{N}{\ln(N)} \quad \text{as } N \to \infty.$$

Here, $\pi(N)$ is the number of primes less than N.

Theorem (Chebyshev, 1848): For all $x > 1$,

$$\pi(x) \geq \frac{x}{2\log_2(x)}.$$

Note: The above form of Chebyshev's theorem is easier to prove, but the more famous version of the Prime Number Theorem is $\pi(x) \sim \frac{x}{\ln(x)}$ when $x \to \infty$, where the logarithm is base e. In Chebyshev's version, the logarithm is base 2.

Using this, if $x \in \{0,1\}^n$ and $x \in [2^n]$, we have:

$$\Pr[x \text{ is prime}] \geq \frac{\frac{2^n}{2\log(2^n)}}{2^n} = \frac{1}{2n}.$$

2.1 Assumption: Factoring

For any adversary $\mathcal{A}$, there exists a negligible function $\epsilon(n)$ such that:

$$\Pr[(p,q) \leftarrow \Pi_n^2; r \leftarrow p \cdot q : \mathcal{A}(r) \in \{p,q\}] < \epsilon(n),$$

where $\Pi_n := \{p < 2^n : p \text{ is prime}\}$ is the set of primes less than 2^n.

Define $\text{mul} : \mathbb{N}^2 \to \mathbb{N}$ as follows:

$$\text{mul}(x,y) = \begin{cases} 1 & \text{if } x = 1 \text{ or } y = 1 \text{ (eliminating trivial factors)}, \\ x \cdot y & \text{otherwise.} \end{cases}$$

It is easy to compute $\text{mul}(x,y)$. For many pairs (x,y), it is easy to invert with probability at least $3/4$ when $x \cdot y$ is even. However, it is not a strong OWF (One-Way Function).

Theorem: If the factoring assumption is true, then mul is a weak OWF.

Proof: The function mul is easy to compute, but is it hard to invert?

Assume for contradiction (AC) that for all polynomial $q(n)$, there exists a non-uniform PPT (nuPPT) adversary A such that for infinitely many $n \in \mathbb{N}$:

$$\Pr[(x,y) \leftarrow \{0,1\}^n; z = x \cdot y : \text{mul}(A(1^{2n}, z)) = z] > 1 - \frac{1}{q(n)}.$$

This is the negation of the weak OWF assumption.

Now, we construct an adversary B that breaks factoring:

Algorithm $B(1^{2n}, z)$:

1. Sample $(x,y) \leftarrow \{0,1\}^n$.
2. If both x and y are prime, let $\bar{z} \leftarrow z$; otherwise, let $\bar{z} \leftarrow \text{mul}(x,y)$.
3. Run $(x', y') \leftarrow A(1^{2n}, \bar{z})$.
4. Output (x', y') if both x and y are prime and $z = x' \cdot y'$.

We intentionally make the input to A uniform in $\{0,1\}^{2n}$.

By Chebyshev's Theorem, both x and y are prime with probability at least:

$$\frac{1}{(2\log(2^n))^2} = \frac{1}{4n^2}.$$

Hence, B fails to pass z to A with probability at most $1 - \frac{1}{4n^2}$.

By the assumption (AC), A fails to invert $\bar{z}$ with probability at most $\frac{1}{q(n)}$. Choose $q(n) = 8n^2$ and construct A correspondingly.

By the union bound, the failure probability of B is at most:

$$\Pr[z \neq \bar{z} \cup A \text{ fails}] \leq \Pr[z \neq \bar{z}] + \Pr[A \text{ fails}] \leq 1 - \frac{1}{4n^2} + \frac{1}{8n^2} = 1 - \frac{1}{8n^2}.$$

Thus, B breaks factoring with probability at least $\frac{1}{8n^2}$, which is greater than negligible, contradicting the factoring assumption.

Note: The above reduction assumes efficient primality testing, which is not necessary but left as an exercise.

Conclusion: The reduction from the factoring assumption to the construction of OWF (such as mul) is a common pattern in cryptography.