Modern Cryptography

600.442

Lecture #10

Dr. Christopher Pappacena

Fall 2013

Chapter 6: One-Way Functions

Inverting a Function

Let $f : \{0,1\}^* \to \{0,1\}^*$ be a function. The *inverting experiment* for f is defined as follows:

- Choose n and $x \leftarrow \{0,1\}^n$, and compute $y = f(x)$.
- $\mathcal{A}$ is given n and y and returns x'
- $\mathcal{A}$ *succeeds* if $f(x') = y$; we write $\mathrm{Invert}_{\mathcal{A},f}(n) = 1$ if $\mathcal{A}$ succeeds.

Note that the returned value x' need not equal x.

One-Way Functions

Definition: A function $f : \{0,1\}^* \to \{0,1\}^*$ is called a *one-way function* if:

- $f(x)$ can be computed in polynomial time in $|x|$.
- For all PPT algorithms $\mathcal{A}$, $\Pr[\mathrm{Invert}_{\mathcal{A},f}(n) = 1] \leq \mathtt{negl}(n)$.

NB: A function f that is not one-way is not necessarily *always* easy to invert - it is just *non-negligibly* invertible infinitely often.

Connection with $P \neq NP$

The existence of a one-way function implies that $P \neq NP$, since confirming that $f(x') = y$ can be done in polynomial time.

The converse is not true – $P \neq NP$ is a statement about *worst-case* complexity, while one-way functions are *almost always* hard to invert.

So the existence of a one-way function is *stronger* than $P \neq NP$!

Nevertheless, we believe that one-way functions exist.

Example: Define $f(x)$ for $x \in \{0,1\}^n$ as follows:

- Write $x = r||s$ with r and s each $n/2$ bits.
- Set $p = \text{NextPrime}(1||r)$ and $q = \text{NextPrime}(1||s)$, and $f(x) = pq$.
- Inverting f amounts to factoring $N = pq$, which is believed to be hard (though believed to *not* be NP-complete).

Example: Given n-bit strings $x_1, \ldots, x_n$ and J, define f by

$$f(x_1, \ldots, x_n, J) = (x_1, \ldots, x_n, sum_{j \in J} x_j)$$

where we identify J with the subset $\{i : J_i = 1\}$ of $\{1, \ldots, n\}$.

Inverting f means, given $(x_1, \ldots, x_n, y)$, find J with $\sum_{j \in J} x_j = y$.

This is known as the *subset sum* problem and is known to be NP-complete.

One-Way Functions and Cryptography

The existence of one-way functions implies the existence of:

- Pseudorandom generators
- Pseudorandom functions
- Strong pseudorandom permutations
- CCA-secure encryption schemes

Conversely, the existence of an encryption scheme with indistinguishable encryptions in the presence of an eavesdropper implies that one-way functions exist.

Hiding Inputs

A one-way function is not necessarily good at hiding its input, despite being hard to invert:

Example: If f is one-way and we define $g(x, y) = (x, f(y))$ for $|x| = |y|$, then g is one-way, even though g "gives away" half of its input.

Intuitively, there must be *something* hard to figure out about x from $f(x)$, otherwise we should be able to invert f.

This leads to the notion of a *hard-core predicate* (also called a *hard-core bit*).

Hard-Core Predicates

Definition: Let f be a one-way function. A function $\mathsf{hc} : \{0,1\}^* \to \{0,1\}$ is called a *hard-core predicate* for f if $\mathsf{hc}(x)$ can be computed in polynomial time in $|x|$ and, for all PPT algorithms $\mathcal{A}$,

$$\Pr[\mathcal{A}(f(x)) = \mathsf{hc}(x)] \leq \frac{1}{2} + \mathtt{negl}(n).$$

In other words, given $f(x)$, a PPT algorithm $\mathcal{A}$ can determine $\mathsf{hc}(x)$ with only negligible improvement over a random guess.

Hard-Core Predicates Exist

Theorem: Let f be a one-way function and define g by $g(x,r) = (f(x),r)$ for $|r| = |x|$. Define $\mathrm{gl}(x,r) = x \cdot r = \oplus_{i=1}^{n} x_i r_r$. Then $\mathrm{gl}(x,r)$ is a hard-core predicate for g.

The book writes $\mathrm{gl}(x)$ for the function $\oplus_{i=1}^{n} x_i r_i$, after Goldreich and Levin, who first introduced the concept and proved the theorem.

The idea of the proof is to show that if $\mathcal{A}$ can guess $\mathrm{gl}(x,r)$ successfully, then this knowledge can be used to invert f.

A Simple Idea Which Doesn't Quite Work

We can see the value r from $g(x, r)$, so once we know $g(x, r)$ we can generate $g(x, r')$ for other $r' \in \{0, 1\}^n$.

If we can determine $\mathsf{gl}(x, r)$ and $\mathsf{gl}(x, r \oplus e_j)$ for some r and j, then we know x_j:

$$\mathsf{gl}(x, r) \oplus \mathsf{gl}(x, r \oplus e_j) = x \cdot r \oplus x \cdot (r \oplus e_j) = x \cdot e_j = x_j.$$

If we can do this for every j then we have inverted f!

What Goes Wrong?

Two main problems:

- The elements r and $r \oplus e_j$ are not independent, so it is hard to get good bounds for the probability of successfully recovering x_j from the probability of correctly determining $\mathsf{gl}(x, r)$.

- If the probability that we succeed in recovering x_j is $\epsilon(n)$, then the probability of recovering x is $\epsilon(n)^n$. This is negligible even if $\epsilon(n) = 0.9999999$.

Let $\mathcal{A}$ be an algorithm that can compute $\mathsf{gl}(x, r)$ from $g(x, r)$ with advantage $\epsilon(n)$. This says that

$$\Pr_{(x,r)\leftarrow\{0,1\}^{2n}}[\mathcal{A}(f(x), r) = \mathsf{gl}(x, r)] \geq \epsilon(n).$$

We first show that we can remove the dependence on x by identifying a suitable subset of $\{0, 1\}^n$:

Lemma: There exists a set $S_n \subseteq \{0, 1\}^n$, of size at least $\epsilon(n)2^n/2$, such that

$$\Pr_{r\leftarrow\{0,1\}^n}[\mathcal{A}(f(x), r) = \mathsf{gl}(x, r)] \geq \epsilon(n)/2$$

for all $x \in S_n$.

Proof of Lemma

Let S_n be the set of all x for which $\text{Pr}_r[\mathcal{A}(f(x), r) = \text{gl}(x, r)] \geq \epsilon(n)/2$. Then write

$$\Pr_{(x,r)}[\mathcal{A}(f(x), r) = \text{gl}(x, r)] \leq \Pr_x[x \in S_n] + \Pr_{(x,r)}[\mathcal{A}(f(x), r) = \text{gl}(x, r) | x \notin S_n].$$

Then

$$\begin{aligned}\Pr_x[x \in S_n] &\geq \Pr_{(x,r)}[\mathcal{A}(f(x), r) = \text{gl}(x, r)] - \Pr_{(x,r)}[\mathcal{A}(f(x), r) = \text{gl}(x, r) | x \notin S_n] \\ &\geq \frac{1}{2} + \epsilon(n) - \left(\frac{1}{2} + \frac{\epsilon(n)}{2}\right) = \epsilon(n)/2.\end{aligned}$$

This says that the size of S_n is at least $\epsilon(n)2^n/2$.

Making Independent Choices

Lemma: For any ℓ, let $s_1, \ldots, s_\ell$ be chosen uniformly at random from $\{0,1\}^\ell$ and, for each nonempty subset $I \subseteq \{1, \ldots, \ell\}$, let $r_I = \oplus_{i \in I} s_i$. Then the $2^\ell - 1$ values $\{r_I\}$ are pairwise independent and uniformly distributed.

Proof: Board.

If we guess the ℓ bits $\{\mathsf{gl}(x, s_1), \ldots, \mathsf{gl}(x, s_\ell)\}$ correctly, then we also know the correct values of the $2^\ell - 1$ bits $\{\mathsf{gl}(x, r_I)\}$.

Inverting f

Given $g(x, r)$, run the following algorithm $\mathcal{A}'$.

- Set $n = |x|$ and $\ell = \lceil \log(2n/\epsilon(n)^2 + 1) \rceil$.

- Choose values $s_1, \ldots, s_\ell \in \{0, 1\}^n$ and guess the bits $\sigma_i = \mathsf{gl}(x, s_i)$.

- For every j and every nonempty subset I of $\{1, \ldots, \ell\}$, use $\mathcal{A}$ to produce a guess for $\mathsf{gl}(x, r_I \oplus e_j)$.

- Set x_j equal to the majority vote of $\{\sigma_j \oplus \mathsf{gl}(x, r_I \oplus e_j)\}$.

- Return $x = x_1 \ldots x_n$.

When Does $\mathcal{A}'$ Succeed?

The algorithm will succeed when the following events happen:

- The value x belongs to S_n.
- $\mathcal{A}'$ correctly guesses the ℓ values $\sigma_i = \mathsf{gl}(x, s_i)$.
- $\mathcal{A}$ computes the correct value of $\mathsf{gl}(x, r_I \oplus e_j)$ for a majority of the indices I.

Note that $\mathcal{A}'$ can also succeed if some of these events don't happen: Mod 2, two wrongs make a right!