CSE 594 : Modern Cryptography 01/31/2017

Lecture 3: One Way Functions - I

Instructor: Omkant Pandey *Scribe: Meghana Doppalapudi, Arun Ramachandran*

1 Modeling the adversary

A cryptosystem is information-theoretically secure if an adversary with infinite computational power cannot break it. One-time pad is an example of such a cryptosystem. In practice, adversaries have bounded computational resources. In modern cryptography we deal with schemes which are good for practical purposes i.e., it is hard for a bounded adversary without infinite computational resources to break and also key length needed not be as long as message length.
We know that any feasible computation can be modeled by Turing machine (with infinite tape and a predefined set of operations) and so is our adversary.

2 Algorithms and running time

2.1 Definition (Algorithm)

An Algorithm is a deterministic Turing machine whose input and output are strings over the binary alphabet $\Sigma = \{0,\ 1\}$.

2.2 Definition (Running Time)

An algorithm A is said to run in time $T(n)$ if $\forall\ x \in \{0,1\}^n$, $A(x)$ halts within $T(|x|)$ steps. A runs in polynomial time if there exists a constant c such that A runs in time $T(n) = n^c$
An algorithm is said to be efficient if it runs in polynomial time.
i.e.,$log_n(T(n)) = c$ where c is a constant which is independent of input length $n = |x|$

2^n, n^{logn}, $n^{loglogn}$ are few examples of non-polynomial functions

2.3 Definition (Randomized Algorithm)

A *randomized algorithm*, also called a probabilistic polynomial time Turing machine (PPT) is a Turing machine equipped with an extra *randomness* tape. Each bit of the randomness tape is uniformly and independently chosen.
The runtime of a randomized algorithm may depend on random tape that is selected. The output of a randomized algorithm is a distribution.

2.3.1 Adversary

An adversary can have many algorithms, one for each input length. This is deemed non-uniform as the algorithm is not uniform across all input sizes. However, it is still efficient as it runs in polynomial time. So, security against non-uniform adversaries must be stronger than uniform ones.

2.4 Definition(Non-Uniform PPT)

A *non-uniform probabilistic polynomial time Turing machine* is a Turing machine A, which is a sequence of probabilistic machines $A = A_1, A_2, \ldots$ for which there exists a polynomial $p(.)$ such that for every $A_i \in A$, the description size $|A_i|$ and the running time of A_i are at most $p(i)$. We write $A(x)$ to denote the distribution obtained by running $A_{|x|}(x)$.

Most often adversaries are non-uniform PPT.

2.5 One Way Functions

The following are two desired properties of an encryption scheme:

- easy to generate cipher text c, given a message m and key k
- hard to recover the message m and key k from cipher text only

Such an encryption scheme is called a One Way function.

2.5.1 Definition(One Way function)

A function $f : \{0,1\}^* \to \{0,1\}^*$ is a one-way function (OWF) if it satisfies the following conditions:

- **Easy to compute:** there is a PPT algorithm C such that $\forall\ x \in \{0,1\}^*$,
  $$\Pr[\ C(x) = f(x)\ ] = 1$$
- **Hard to invert:** For every non-uniform PPT adversary A, for any input length $n \in N$,
  $$\Pr[\ A \text{ inverts } f(x) \text{ for random } x\ ] \leq \text{small}$$

$$\text{Rewritten as } \Pr[\ x \xleftarrow{\$} \{0,1\}^n;\ A \text{ inverts } f(x)\ ] \leq \text{small}$$

We can define *small* as a **fast decaying function** $\nu(.)$ such that for any input length $n \in N$

$$\Pr[\ x \xleftarrow{\$} \{0,1\}^n;\ A \text{ inverts } f(x)\ ] \leq \nu(n)$$

2.5.2 How fast should it decay?

- Is 10 % good? No, as it is easy to invert 1 in every 10 cases.
- Is 0.0000001 % good? It is easy for 1 in 10^6 cases i.e., $\approx$ for every 1MB data
- Is $\frac{1}{n^{100}}$ good? No. A polynomial in denominator implies easy cases occur soon enough.

This implies that for some values adversary A can invert the function. However, we want functions that for a randomly chosen x, probability of inverting it is small. Hence, ν must decay faster than any polynomial. Now, we proceed to formally define negligible function $\nu(.)$

2.5.3 Definition: (Negligible function)

A function $\nu(n)$ is negligible if for every c, there exists some n_0 such that for all n greater than n_0, $\nu(n) \leq \frac{1}{n^c}$

- Negligible functions decay faster than inverse polynomial functions i.e., they are asymptotically smaller than inverse of a polynomial function.
- 2^{-n}, $n^{-loglogn}$ are examples of negligible functions.

3 One Way Functions - Attempt I

Attempt 1: As mentioned earlier, a function $f : \{0,1\}^* \rightarrow \{0,1\}^*$ is a one-way function (OWF) if it satisfies the following two conditions :

- **Easy to compute:** there is a PPT algorithm C such that $\forall x \in \{0,1\}^*$,
$$Pr[C(x) = f(x)] = 1.$$
- **Hard to invert:** for every non-uniform PPT adversary A, there exists a negligible function $\nu(.)$ such that for any input length $\forall n \in N$:
$$Pr[x \xleftarrow{\$} \{0,1\}^n;\ A\ inverts\ f(x)] \leq \nu(|x|).$$
$|x| = n.$

  Let us focus on the problem of A's input.

3.1 A's Input

Let $y = f(x)$. The adversary A, performing inversion, gets $f(x)$. So the input length to A is $|f(x)|$ = $|y|$. Let the following conditions hold good.

- **Condition 1:** A, on input y must run in polynomial time on the input, $poly(|y|)$.
- **Condition 2:** A cannot output x' such that $f(x') = y$.
- However, $f(x)$ need not be an injective function(might not have an exact 1-1 mapping, such as a constant function which is a many-1 mapping). $|y|$ could be much smaller than $n = |x|$.

  In this case, A will easily be able to find an inverse(correctly as per requirements) but not be able to write it in polynomial time.
- For instance, let us consider $f(x)$ = first $log|x|$ bits of x. It is trivial to invert this, as any output $f^{-1}(y)$such that it starts with y is an answer.
$$f^{-1}(y) = y||\underbrace{00...0}_{n-logn},$$
  Any output that starts with $y(log(n)$ bits) followed by 0s($n - log(n)$ of them), where $n = 2^{|y|}$ is a valid inversion. $f(x)$ is easy to compute (satisfies condition 1), but A cannot invert it in polynomial time($poly(|y|)$) (does not satisfy condition 2). For inversion, the input is of size $|y|$, and $f^{-1}(y)$ is of size $n = 2^{|y|}$. So, construction of the inverse takes a time exponential in $|y|$

3.2 Fixing the definition

For the inversion, A must be given a long enough input to invert(which is not too small compared to $n = |x|$). If the input y is too short, then it is padded with 1s in the beginning. Now, A will have enough time to write the answer, as the input size for the inversion is no more $|y|$ but $(|y| + n)$. Conventionally, the padding of 1s is represented as : $A(1^n, y)$.

Redefining One Way Functions as follows :

Definition 1 *A function $f : \{0,1\}^* \to \{0,1\}^*$ is a one-way function (OWF) if it satisfies the following two conditions :*

- ***Easy to compute :*** *there is a PPT algorithm C such that $\forall x \in \{0,1\}^*$,*

$$Pr[C(x) = f(x)] = 1.$$

- ***Hard to invert :*** *there exists a negligible function $\nu : N \to R$ such that for every non-uniform PPT adversary A and any input length $\forall n \in N$:*

$$Pr[x \leftarrow \{0,1\}^n, x^{'} \leftarrow A(1^n, f(x)) : f(x^{'}) = f(x)] \leq \nu(n).$$

This definition is also called a strong One Way Function(OWF). This means that all efficient adversaries (non-uniform PPT adversaries) will almost always fail to invert.

3.3 Injective OWFs and One Way Permutations (OWP)

- **Injective or 1-1 OWFs:** each image has a unique pre-image:

$$f(x_1) = f(x_2) \implies x_1 = x_2$$

  However, in injective OWFs, there could be an image $f(x_k)$ without a pre-image x_k

- **One Way Permutations (OWP):** These are 1-1 OWFs with the additional condition that every image has a pre-image, (i.e.) for every image $f(x_k)$, there exists an x_k. In other terms, size of the domain = size of the range.

3.4 Existence of OWFs

OWFs are not unconditional, they require proving that $P \neq NP$. But, we can construct them assuming that certain problems are hard, so such constructions are called "candidates" as they are based on assumptions.

4 Factoring Problem

Considering the multiplication function, $f_X : NXN \to N$:

$$f_X(x,y) = \begin{cases} \perp, & if\ x = 1 \vee y = 1 \\ x.y, & otherwise \end{cases}$$

For the factoring problem, we exclude trivial cases like the trivial factor 1 (Case 1 above).
We can comprehend that f_X is not a OWF(f_X is not a strong OWF), as follows.
If the product is even, then 2 is a factor, and the second factor can easily be found out by dividing the product by 2. A random number of any fixed size is even with probability 1/2. The product $x.y$ is even with probability 3/4 (it is even if either x, or y, or both are even). For the inversion, if given a number $z = x.y$, the output could be (2,z/2) if z is even and (0,0) otherwise. This inversion mechanism would succeed 3/4 times (75% of the time).

Eliminating the trivial factors, let Π_n be the set of all prime numbers $< 2^n$. If p and q are randomly chosen from Π_n and multiplied, this is unlikely to have small trivial factors. This leads to the following assumption.

Assumption 1 ***(Factoring Assumption):*** *For every non-uniform PPT adversary A, there exists a negligible function ν such that*

$$Pr[p \xleftarrow{\$} \Pi_n;\ q \xleftarrow{\$} \Pi_n;\ N = pq :\ A(N) \in \{p, q\}] \le \nu(n).$$

This factoring assumption is a time-tested conjecture, that has been studied for a long time, and no "good" attack(which means no attack by any non-uniform PPT adversary) has been devised against it. The best known algorithms for breaking it are :

$$2^{O(\sqrt{nlogn})}(provable)$$

$$2^{O(\sqrt[3]{nlog^2n})}(heuristic)$$

Now, the question is if we can construct OWFs using this factoring assumption.

4.1 Back to the Multiplication Function

Reconsidering the function $f_X : N^2 \to N$, let us call the case when a randomly chosen x and randomly chosen y are both prime as the GOOD case, because no non-uniform PPT adversary A can invert a product of such an x and y.

If the GOOD case occurs with probability $> \epsilon$, then every A must fail to invert f_X with probability atleast ϵ.

If ϵ is a noticeable function, then every A must fail to invert f_X with a noticeable probability. This definition of f_X is called a weak One Way Function.

4.2 Weak One Way Functions

Definition 2 *A function $f : \{0,1\}^* \to \{0,1\}^*$ is a weak one-way function if it satisfies the following two conditions :*

- ***Easy to compute :*** *there is a PPT algorithm C such that $\forall x \in \{0,1\}^*$,*

$$Pr[C(x) = f(x)] = 1.$$

- ***Somewhat hard to invert :*** *there is a noticeable function $\epsilon : N \to R$ such that for every non-uniform PPT adversary A and for all sufficiently large n:*

$$Pr[x \leftarrow \{0,1\}^n, x^{'} \leftarrow A(1^n, f(x)) : f(x^{'}) \neq f(x)] \geq \epsilon(n).$$

  Noticeable means that $\exists c$ and integer N_c such that $\forall n > N_c : \epsilon(n) \geq 1/n^c$

Using the above, f_X can be proven to be a weak OWF, if we can show that the GOOD case (when x and y are prime) occurs with noticeable probability.

Theorem 1 *Assuming the factoring assumption, the function f_X is a weak OWF.*

Using the idea that the fraction of prime numbers between 1 and 2^n is noticeable, and according to Chebyshev's theorem which states that an n-bit number is prime with probability $\geq 1/2n$, the above theorem can be proved.

The next question that arises is whether we can construct normal (strong) OWFs from the factoring assumption, or put in other words, whether we can construct strong OWFs from any weak OWF. Yes, this could be done using Yao's theorem.

4.3 Weak to Strong OWFs

Theorem 2 *Strong OWFs exist if and only if weak OWFs exist.*

Converting a somewhat hard problem into a really hard one is called hardness amplification. This involves applying the weak OWF on many independent samples and using all of the outputs as the output of the strong OWF(hardness amplification),and using a proof by reduction technique: if an adversary A can break our strong OWF, we can come up with an algorithm B that breaks weak OWFs, thus contradicting our assumption of weak OWFs existing. Hence, by reverse implication, the strong OWF also exists.