Introduction to Cryptography

CSCI-GA 3210 / MATH-GA 2170

Oded Regev

Transcribed by Patrick Lin

ABSTRACT. These notes are from an introductory course in Cryptography, as offered in Fall 2014 at the Courant Institute of Mathematical Sciences, a school of New York University. The professor for the course was Oded Regev.

More information about the course can be found at the course website: `http://www.cims.nyu.edu/~regev/teaching/crypto_fall_2014/`.

In this course the two main topics covered are encryption and authentication. Additional topics may include to zero-knowledge proofs. A very popular modern topic is secure computation in the cloud, specifically homomorphic encryption.

The text for the course is *Introduction to Cryptography* Katz and Lindell; a good reference is *Foundations of Cryptography, Vol. 1 and 2* by Goldreich. This course follows closely the lecture notes by Yevgeniy Dodis.

These notes were transcribed live (with minor modifications) in LaTeX by Patrick Lin. The notes are in a draft state, and thus there are likely many errors and inconsistencies. These are corrected as they are found.

Revision: 10 Dec 2014 20:48.

Contents

CHAPTER 1

Introduction: Perfect Secrecy

Lecture 1, 8 Sep

This was created by Shannon in 1948. Shannon considered a formal model.

Roughly, consider two parties Alice and Bob, and Alice wants to send a message to Bob. In between, there is an eavestropper Eve listening to the communication trying to figure out what is being sent. We will formally define the model.

Once we have a model, we need to define functionality and security, and then as a third step construct the solution (protocol) and prove that it works.

So what is going on in the model? We represent Alice by an algorithm or function $A(\cdot)$ that takes a message in some message space, $m \in \mathcal{M}$, and maps it to some cipher in a cipher space $c \in \mathcal{C}$. Bob is an algorithm or function $B(\cdot)$ that takes $c \in \mathcal{C}$ and maps it to $m \in \mathcal{M}$. So we will assume that for all $m \in \mathcal{M}$, $B(A(m)) = m$. Eve is represented by $E(\cdot)$ that takes $c \in \mathcal{C}$ and outputs some $m' \in \mathcal{M}$.

A problem is that we could just have $E = B$, so we could propose that Eve does not know the system A, B. As it turns out, history has shown that this is a terrible idea, as the algorithm can be leaked. Furthermore, secret algorithms cannot be analyzed by outside researchers to find weaknesses. A good cryptography system should be such that even if the enemy knows the system, they cannot decrypt the message.

So what is missing from this model is the secret information used in modern systems, the key. This is the break in symmetry between Bob and Eve: Bob knows the key but Eve does not. Then even if Eve finds the key, it is possible to just use a new one that Eve does not know.

Now we will consider this new model. We have:

- Generator procedure Gen: randomized algorithm that outputs key $k \in \mathcal{K}$.
- Encryption procedure $\mathrm{Enc}(k, m)$
- Decryption procedure $\mathrm{Dec}(k, c)$

with the functionality that $\forall k \in \mathcal{K}, m \in \mathcal{M}$, $\mathrm{Dec}(k, \mathrm{Enc}(k, m)) = m$.

Notice that this changes the model: if Alice and Bob want to have this advantage over Eve, they need to first agree on the key. This is called *shared-key cryptography*: they need to first share the key and then they can communicate securely.

So how do we define security? There are some possibilities, though each is bad in some way:

- Eve should not be able to recover the key.
- Eve should not be able to output the message.
- The ciphertext should look like "gibberish".

It is easy to construct a system where Eve cannot recover the key but the system is not secure. For example, we can send the message in plain text: Eve cannot recover the key but can easily decrypt the message. If Eve cannot output the message, Eve might still be able to output some important information about the message.

What Shannon came up with is the following:

DEFINITION 1.1 (Shannon). A system $(\mathrm{Gen}, \mathrm{Enc}, \mathrm{Dec})$ is *perfectly secure* if for all distributions $\mathcal{D}$ over $\mathcal{M}$, and $\forall \bar{m} \in \mathcal{M}, \bar{c} \in \mathcal{C}$, we have

$$\Pr_{\substack{k \leftarrow \mathrm{Gen} \\ m \leftarrow \mathcal{D}}}[m = \bar{m} \mid \mathrm{Enc}_k(m) = \bar{c}] = \Pr_{m \leftarrow \mathcal{D}}[m = \bar{m}].$$

In other words, knowing the ciphertext tells us nothing about the message. This seems like a cumbersome definition but it captures the intuition that Shannon had. The following is an equivalent definition:

DEFINITION 1.2. A system $(\mathrm{Gen}, \mathrm{Enc}, \mathrm{Dec})$ is *perfectly secure* if for all distributions $\mathcal{D}$ over $\mathcal{M}$, and $\forall m_0, m_1 \in \mathcal{M}, \bar{c} \in \mathcal{C}$,

$$\Pr_{k \leftarrow \mathrm{Gen}}[\mathrm{Enc}_k(m_0) = \bar{c}] = \Pr_{k \leftarrow \mathrm{Gen}}[\mathrm{Enc}_k(m_1) = \bar{c}].$$

EXERCISE 1.3. Prove that the two definitions are equivalent.

CONSTRUCTION 1.4. The One-Time Pad (OTP):

- Gen: $k \leftarrow$ uniform from $\{0,1\}^n$
- $\mathrm{Enc}_k(m) = m \oplus k \in \{0,1\}^n$
- $\mathrm{Dec}_k(m) = c \oplus k \in \{0,1\}^n$

THEOREM 1.5. *The OTP is a perfectly secure shared-key encryption scheme.*

PROOF. We first verify functionality:

$$\mathrm{Dec}_k(\mathrm{Enc}_k(m) = (m \oplus k) \oplus k = m \oplus (k \oplus k) = m$$

as desired.

We next show security, using the second definition. $\forall m_0, m_1, \bar{c}$,

$$\Pr_{k \leftarrow \mathrm{Gen}}[\mathrm{Enc}_k(m_0) = \bar{c}] = \Pr_{k \leftarrow \mathrm{Gen}}[(m_0 \oplus k) = \bar{c}] = \Pr_{k \leftarrow \mathrm{Gen}}[k = \bar{c} \oplus m_0] = 2^{-n}.$$

It is easy to verify that $\Pr_{k \leftarrow \mathrm{Gen}}[\mathrm{Enc}_k(m_1) = \bar{c}] = 2^{-n}$ as well. □

So should we just stop here and use this for everything? Well as it turns out it is not so convenient to use this: the length of the key must be the same as the length of the message. So we need to know in advance the amount of communication.

The Soviets ran into this issue, because they started just using the key again when they ran out of keys. However, the key can only be used once: suppose the eavesdropper gets $c_1 = m_1 \oplus k$ and $c_2 = m_2 \oplus k$, then $c_1 \oplus c_2 = m_1 \oplus m_2$. As it turns out it is not so difficult to recover m_1 and m_2 from $m_1 \oplus m_2$, and this is actually what happened during the Cold War.

So we see that definitions of security are subtle. Even if we proved that the system is secure, the proof counts only as much as the system does. If we follow the system to the letter, it is perfectly secure, but even if we try to abuse it a little bit the entire system falls apart.

So we can try to improve this, but Shannon proved that this is impossible.

THEOREM 1.6. *(Shannon) In any perfectly secure scheme,* $|\mathcal{K}| \geqslant |\mathcal{M}|$.

EXERCISE 1.7. Prove this theorem.

So no system can be more secure than OTP.

The main loophole we will use is the following: just because we can get information about the message from the ciphertext does not mean that we can do so *efficiently*. So we move from information-theoretic cryptography to computational cryptography. This

completely changes the picture, and allows us to do many paradoxical things such as zero-knowledge proofs or homomorphic encryption.

So we will assume that Eve is bounded by polynomial-time computations. Since we have good reason to believe that $\mathsf{P} \neq \mathsf{NP}$, we will often use NP-intermediate problems. Most of the time we will use factoring.

CHAPTER 2

Important Background

2.1. The Computational Model

We quickly review the computational model. So formally an algorithm is a Turing Machine; the Church-Turing thesis states that the Turing Machine captures everything that is computable.

We do not just care about computability, but also efficiency, but for our purposes we will consider efficient to mean polynomial-time, that is, $\exists c > 0$ such that for all input of size n, the running time is $O(n^c)$. This is a robust notion because we can compose polynomial-time algorithms and get a polynomial-time.

Our model is usually what is known as *Probabilistic Polynomial Time (PPT)*, which is a polynomial-time algorithm that is also allowed randomization. Otherwise we are restricted to deterministic algorithms.

A question is whether probabilistic algorithms are more powerful than deterministic algorithms. In the past people thought they were, but more recently people think that it is not helpful. A common example is Primality Testing, which was recently shown to be solvable in deterministic polynomial-time (albeit slower than the probabilistic algorithm).

Sometimes we will be sloppy and have a *security parameter* n, so that the larger n is, the more secure the system. Often n will be the size of the key. This parameter will be implicitly given to all algorithms (in unary[1]).

We define $\mathrm{poly}(n)$ as a class of functions: $T \in \mathrm{poly}(n)$ if there exists some c, C such that for all n sufficiently large, $T(n) \leqslant C \cdot n^c$.

We say that T is *negligible* if $\forall c$, $T(n) = o(n^{-c})$, i.e. $\lim_{n\to\infty} n^c \cdot T(n) = 0$. The class of negligible functions is $\mathrm{negl}(n)$. Just as polynomial-time algorithms are closed under composition, so are negligible algorithms. Further, it is easy to see that $\mathrm{negl} \cdot \mathrm{poly} = \mathrm{negl}$.

We say that $\delta(n)$ is *noticeable* if there exists some c such that $\delta(n) = \Omega(n^{-c})$. This is roughly non-negligible, but strictly speaking this is not the negation.

2.2. Number Theory

We need to know the concept of a prime number, eg. $2, 3, 5, 7, \ldots$.

An important question is algorithmic number theory is the following: suppose we are given a number $2^{57885161} - 1$. Is this number prime? It is hard to say. As it turns it this is currently the largest known Mersenne prime, but the reason we know it is prime is because there is an algorithm that can check for primality efficiently.

Another important question is how to choose a prime number. Usually in cryptographic systems, the key should be a prime number. So how do we choose one, say, that is 100 digits long, or even longer?

[1] This is just a technicality so that the algorithm runs in polynomial time.

The naïve solution is to choose a uniform n-bit number and check that it is prime. The question is how many prime numbers there are in $\{1, \dots, N\}$ where N is very very large. Usually we have $n = \log_2 N$. So if $N^{0.99}$ of the numbers in $\{1, \dots, N\}$ were not prime, then we would be very sad, because we would have to test many numbers before finding one. Fortunately, within $\{1, \dots, N\}$ there are approximately $\frac{N}{\ln N}$ primes. In fact it is easier to prove that there are $\geqslant \frac{N}{2 \log_2 N}$ and this is good enough. So on average it is only necessary to repeat the process $O(n)$ times.

Another important question is factoring[2], so for example $15 = 3 \cdot 5$. This is believed to a relatively difficult problem. The naïve algorithm tests divisibility by the first $\sqrt{N}$ numbers, but this takes exponential time: a n-bit number is factored in time $2^{n/2}$. The fastest known algorithm factors a n-bit number in time $2^{n^{1/3}}$ (the number field sieve). This is the reason that, for example with RSA, we often work with numbers of 1000 bits.

Another problem is to find the $\gcd$ of two numbers.

DEFINITION 2.1. For integers $a, b \geqslant 1$, define $\gcd(a, b)$ to be the largest d that divides both a and b.

Euclid's algorithm finds the $\gcd$ of two numbers efficiently: it takes time $\mathrm{poly}(\log a + \log b)$. As a byproduct, the algorithm also gives two numbers x, y such that $ax + by = d$. The reason this is nice is because if a, b are coprime (that is, $\gcd(a, b) = 1$), then we get x, y such that $ax + by = 1$, that is, $ax = 1 \bmod b$. So x is the multiplicative inverse of a modulo b.

The next important concept is that of *Chinese Remainder Theorem (CRT)*, which we will use a lot. We often work with the ring $(\mathbb{Z}_N, +, \cdot)$ of integers modulo N.[3]

LEMMA 2.2 (CRT). *For $N = pq$ product of two coprimes, the ring $\mathbb{Z}_N$ is isomorphic to the product ring $\mathbb{Z}_p \times \mathbb{Z}_q$. More precisely, there is an isomorphism $h : \mathbb{Z}_N \to \mathbb{Z}_p \times \mathbb{Z}_q$, $h(a) := (a \bmod p, a \bmod q)$.*

Lecture 2, 15 Sep

This is efficiently computable assuming the factorization of N is known. h is also efficiently invertible: we first compute $c_p, c_q \in \mathbb{Z}_N$ (using Euclid's algorithm) such that $h(c_p) = (1, 0)$ and $h(c_q) = (0, 1)$, then $h^{-1}(a, b) = ac_p + bc_q \in \mathbb{Z}_N$.

Sometimes we will wish to work with the elements of $\mathbb{Z}_N$ under multiplication, but not all elements in $\mathbb{Z}_N$ have multiplicative inverses. So we take the multiplicative group $\mathbb{Z}_N^* = \{x \in \mathbb{Z}_N \mid x \text{ is invertible } \bmod N, \text{ equivalently } \gcd(x, N) = 1\}$. Commonly we use the Euler φ: $\varphi(N) = |\mathbb{Z}_N^*|$. For prime p, $\mathbb{Z}_p^* = \{1, 2, \dots, p-1\}$, so $\varphi(p) = p - 1$. As it turns out, this group is always cyclic, i.e. there exists some g such that $\mathbb{Z}_p^* = \{g^0 = 1, g^1 = g, g^2, \dots, g^{p-2}\}$. So $\mathbb{Z}_p^* \cong \mathbb{Z}_{p-1}$.

EXAMPLE 2.3. $\mathbb{Z}_5^* = \{1, 2, 3, 4\} = \{2^0, 2^1, 2^2, 2^3\}$, so 2 is a generator of $\mathbb{Z}_5^*$. $\mathbb{Z}_7^* = \{1, \dots, 6\} \neq \{2^0 = 1, 2^1 = 2, 2^2 = 4\}$, so 2 is not a generator, but 3 is.

In $\mathbb{Z}_p^*$ generated by g, $-1 = g^{\frac{n-1}{2}}$, since $(-1)^2 = g^{n-1} = 1$.

By the Chinese Remainder Theorem, $\mathbb{Z}_N^* \cong \mathbb{Z}_p^* \times \mathbb{Z}_q^*$ for $N = pq$, for odd primes $p \neq q$. So $\varphi(N) = \varphi(p)\varphi(q) = (p-1)(q-1)$.

An important subgroup of $\mathbb{Z}_N^*$ that we will use is the group of quadratic residues $\mathbb{QR}_N^* = \{y \in \mathbb{Z}_N^* \mid \exists x \in \mathbb{Z}_N^*, y = x^2\}$. For odd prime p, $\left|\mathbb{QR}_p^*\right| = \frac{p-1}{2}$. That is, the

[2]Factoring here is well-defined: every integer can be factored into a unique set of prime factors. In other number fields this is not necessarily true, but for integers it is.

[3]Mathematical texts will never write the integers modulo N as $\mathbb{Z}_N$; rather, it will be written as $\mathbb{Z}/N\mathbb{Z}$.

function $x \mapsto x^2$ is a 2-to-1 function. For $N = pq$ for odd primes $p \neq q$, the Chinese Remainder Theorem gives $\mathbb{QR}_N^* \cong \mathbb{QR}_p^* \times \mathbb{QR}_q^*$.

EXAMPLE 2.4. $|\mathbb{QR}_{35}^*| = |\mathbb{QR}_5^*|\,|\mathbb{QR}_7^*| = 2 \cdot 3 = 6$.

So $|\mathbb{QR}_N^*| = \left|\mathbb{QR}_p^*\right|\left|\mathbb{QR}_q^*\right| = \frac{p-1}{2} \cdot \frac{q-1}{2} = \frac{\varphi(N)}{4}$, so that $x \mapsto x^2$ is 4-to-1 in $\mathbb{Z}_N^*$. If we take more primes, this becomes 8-to-1, and so on.

An important question is, when is $-1 \in \mathbb{QR}_p^*$? This is only when $p = 1 \bmod 4$.

EXERCISE 2.5. Check this fact.

CHAPTER 3

One-Way Functions

Almost every scheme in cryptography implies the use of one-way functions, including public-key encryption and authentication. Many other ideas follow from one-way functions. So it is important to talk about this primitive.

We want to say that a function is a one-way function if it is easy to compute but hard to invert. We need to make this precise.

DEFINITION 3.1. A function $f : \{0,1\}^* \to \{0,1\}^*$ is a *one-way function (OWF)* if it is:

(1) Easy to compute: there exists a polynomial-time algorithm that, given x, outputs $f(x)$.
(2) Hard to invert: for any (non-uniform[1]) PPT algorithm I, the advantage is negligible, $\mathrm{Adv}_f(I) = \Pr_{x\in\{0,1\}^n}[I(1^n, f(x)) \in f^{-1}(f(x))] \leqslant \mathrm{negl}(n)$[2].

The use of $\{0,1\}^*$ for the domain is not important; it is easy to modify the definition to a restriction of $\{0,1\}^*$. For a more rigorous treatment, we would have a domain D, with the restriction that we have a good way of sampling D.

The reason we allow any element in the preimate is because if we required I to output x, then the constant function $f = 0$ would be the model one-way function.

Why do we have $\mathrm{Adv}_f(I) \leqslant \mathrm{negl}(n)$? We cannot ask for $\mathrm{Adv}_f(I) = 0$, because no function can satisfy that, since even a random guess would not disqualified. Even $\mathrm{Adv}_f(I) \leqslant 10 \cdot 2^{-n}$ is too restrictive.

What if we just require $\mathrm{Adv}_f(I) < 1$? This is far more modest requirement. This runs into the difficulty that we can construct one-way functions based on NP-hard questions, so this is more like a worst-case analysis than an average-case analysis. Furthermore, it is not useful for cryptography because f can be inverted on almost all x.

There is a notion of a *weak one-way function*, in which $\mathrm{Adv}_f(I) \leqslant 1 - \frac{1}{\mathrm{poly}(n)}$. As it turns out, the existence of a weak one-way funciton implies the existence of a one-way function.

If f is a permutation (or bijection) on $\{0,1\}^n$, and f is one-way, then we call f a *one-way permuation (OWP)*.

The definition of OWF allows for the inverter to recover, for example, the first half of the bits of x, but not the second half.

It would be good to go through some examples of one-way functions, but in fact we do not know of any. We can, however, look at some OWF candidates.

[1] Non-uniformity just means that the algorithm is allowed to be different for inputs of different length.
[2] We input 1^n to I as a technicality to allow I to run in time $\mathrm{poly}(n)$.

3.1. Examples of OWF Candidates

EXAMPLE 3.2. Subset Sum[3]: $f_{\text{ss}} : \mathbb{Z}_N^n \times \{0,1\}^n \to \mathbb{Z}_N^n \times \mathbb{Z}_N$ where $N = 2^n$,[4] where $f_{\text{ss}}(a_1, \ldots, a_n, b_1, \ldots, b_n) = (a_1, \ldots, a_n, \sum_{i=1}^n b_i a_i)$. This is clearly efficiently computable. However, inverting this seems difficult, since the inverter has to do the following: given uniform iid $a_1, \ldots, a_n \in \mathbb{Z}_N$ and $\sum b_i a_i$ for uniform iid $b_1, \ldots, b_n \in \{0,1\}$, output $b_1', \ldots, b_n' \in \{0,1\}$ such that $\sum b_i' a_i = \sum b_i a_i$. It is important to output the a_i's, since otherwise the inverter can just output $(\sum_{i=1}^n b_i a_i, 0, \ldots, 0, 1, 0, \ldots)$.

REMARK 3.3. What if we work with $\mathbb{Z}_2^n$ instead of $\mathbb{Z}_N$? Well, it is easy to set up a system of linear equations modulo 2, and invert using Gaussian elimination.

EXAMPLE 3.4. Multiplication: $f_{\text{mult}} : \mathbb{N}^2 \to \mathbb{N}$, $f_{\text{mult}}(x, y) = xy$. Clearly this is efficient. This is easy to invert, since we can output $(1, xy)$. So let us forbid 1. Then it is still easy, since with probability $\frac{3}{4}$, xy is even, so we can output $(2, xy/2)$. To make this hard, we usually pick x, y prime. So if we define $\Pi_n = \{p \mid p \in \{1, \ldots, 2^n\} \text{ prime}$, then $f_{\text{mult}} : \Pi_n \times \Pi_n \to \mathbb{N}$ is a OWF based on the following conjecture:

CONJECTURE 3.5 (Factoring Assumption). *For every (non-uniform) PPT A,*

$$\Pr_{p,q \leftarrow \Pi_n}[A(pq) = (p, q)] \leqslant \text{negl}(n).$$

Another way to fix the problem is to restrict the domain to $\{1, \ldots, 2^n\}^2$ instead of restricting to Π_n^2.

EXERCISE 3.6 (Goldreich). Show that $f_{\text{mult}} : \{1, \ldots, 2^n\}^2 \to \mathbb{N}$ is a OWF.

REMARK 3.7. Although $\frac{3}{4}$ of the inputs are easily inverted, $> 1 - \frac{1}{\text{poly}(n)}$ inputs cannot be.

Often we do not know the domains that are hard. Maybe only some inputs are hard. How do we protect ourselves when we do not know? This is the question of converting weak one-way functions into one-way functions by way of some sort of amplification.

3.2. Weak OWFs imply (Strong) OWFs

Lecture 3, 22 Sep

Recall that what makes a (strong) one-way function strong is that no matter what attacker is brought in, the attacker can only invert a negligible amount of inputs. On the other hand, a δ-weak[5] one-way function is such that the attacker cannot invert δ fraction of inputs, i.e. for all non-uniform PPT inverter I,

$$\Pr[I(f(x)) \in f^{-1}(f(x))] \leqslant 1 - \delta.$$

We say f is a weak OWF if there is some $c > 0$ such that f is a n^{-c}-OWF.

We will show the following: that the existence of a weak OWF implies the existence of a strong OWF. To that end, for f, define its m-wise direct product by $f'(x_1, \ldots, x_m) = f^{(m)}(x_1, \ldots, x_m) = (f(x_1), \ldots, f(x_m))$.

THEOREM 3.8. *If f is a δ-weak OWF for some $\delta = 1/\text{poly}(n)$ then $f^{(m)}$ is a (strong) OWF for $m = 2n/\delta$.*

---

[3]Subset-sum is NP-hard, but this is irrelevant.

[4]If N is too big, then the sum reveals too much information; if N is too small, then the problem becomes to easy by just trying random bits. $N = 2^n$ seems to be about right.

[5]Say, $\delta = 1/\text{poly}(n)$.

The intuition is that if we have to do this many times, we make it hard. We will make this precise.

PROOF IDEA. Towards contradiction, we assume that there exists an efficient inverter I' that inverts $f^{(m)}$ with non-negligible probability $\alpha(n)$. Our goal is to construct I that inverts f with probability $> 1 - \delta$.

The intuition is that if we try to invert each of the elements, we succeed with probability $(1-\delta)^m = (1-\delta)^{2n/\delta} \approx e^{-2n}$.

First attempt: $y = f(x)$, I applies I' to $y' = (y_1 = y, y_2, \ldots, y_m)$, where for $i \geqslant 2$, $y_i = f(x_i)$, and $x_2, \ldots, x_m$ are iid uniform from the domain. With probability α, we get a preimage to y. The trouble is that we need this to amplify this probabliity to $> 1 - \delta$.

We can try to repeat the above many times with the same y and fresh $y_2, \ldots, y_m$. But this is still problematic, because the area that I' solves might be very bad and only solves α fraction of the first coordinate. So this attempt is still not enough, but we can fix by trying to "plant" y in all possible coordinates. □

So let us go to the proof proper.

PROOF. First, we must prove that $f^{(m)}$ is polynomial-time computable. But this is easy, because f is, and $m = \mathrm{poly}(n)$.

Next, towards contradiction assume that there exists an efficient inverter I' that inverts $f^{(m)}$ with non-negligible probability $\alpha(n)$. Our goal is to construct I that inverts f with probability $> 1 - \delta$.

Define for $i = 1, \ldots, m$ $G_i = \{x_i \mid \Pr_{x' : x'_i = x_i}[I'(f'(x')) \text{ succeeds}] \geqslant \frac{\alpha}{2m}\}$. We claim that for some i, $\Pr_x[x \in G_i] \geqslant 1 - \delta/2$.

Suppose not. Then

$$\begin{aligned}
\Pr_{x'}[I'(f'(x')) \text{ succeeds}] &= \Pr_x[I'(f'(x')) \text{ succeeds} \wedge \forall i, x_i \in G_i] \\
&\quad + \Pr[[I'(f'(x')) \text{ succeeds} \vee \exists i, x_i \notin G_i] \\
&\leqslant \Pr[\forall i, x_i \in G_i] + \sum_{i=1}^{m} \Pr[I'(f'(x')) \text{ succeeds} \wedge x_i \notin G_i] \\
&\leqslant \Pr[\forall i, x_i \in G_i] + \sum_{i=1}^{m} \Pr[I'(f'(x')) \text{ succeeds} \mid x_i \notin G_i] \\
&\leqslant (1 - \delta/2)^m + m\frac{\alpha}{2m} \\
&\leqslant e^{-n} + \alpha/2 < \alpha.
\end{aligned}$$

Now define I as follows:

For $i = 1, \ldots, m$, repeat the following $2mn/\alpha$ times: applies I' to $(y_1, \ldots, y_m)$ where $y_i = y$ and for $j \neq i$, $y_j = f(x_j)$, for random x_j.

Notice that if $x \in G_i$, where $y = f(x)$ is our input, then for some i, the algorithm succeeds with probability $\geqslant 1 - (1 - \frac{\alpha}{2m})^{2mn/\alpha} \geqslant 1 - e^{-n}$. So if $x \in G_i$ we can surely find a preimage. We showed that $x \in G_i$ with probability $\geqslant 1 - \delta/2$, so overall our success probability is $\geqslant (1 - \delta/2)(1 - e^{-n}) > 1 - \delta$, in contradiction to that f is a weak one-way function. □

Note that this blows up the input size and is very inefficient. Goldreich's book contains a construction using expanders that is much more efficient.

3.3. Collections of OWFs

For motivation, recall the subset-sum function $f : \mathbb{Z}_N^n \times \{0,1\}^n \to \mathbb{Z}_N^{n+1}$ given by $f(a_1, \ldots, a_n, b_1, \ldots, b_n) = (a_1, \ldots, a_n, \sum a_i b_i)$. But this seems strange. It seems more natural and efficient to fix $a_1, \ldots, a_n$ and define the problem as $f_{a_1,\ldots,a_n}(b_1, \ldots, b_n) = \sum a_i b_i$. So to do this we introduce the notion of a family of OWFs.

DEFINITION 3.9. A *collection of OWFs* is a family $\mathcal{F} = \{f_s : D_s \to R_s\}_{s \in S}$ satisfying:

(1) It is easy to sample from the functions: there exists a PPT algorithm Gen that outputs $s \in S$ from some distribution.
(2) It is easy to sample from the domain: there exists a PPT algorithm D such that $D(s)$ outputs some $x \in D_s$.
(3) It is easy to compute: there exists a PPT algorithm F such that for $s \in S, x \in D_s$, $F(s, x) = f_s(x)$.
(4) It is hard to invert: for all non-uniform PPT I,

$$\Pr_{\substack{s \leftarrow \mathrm{Gen}(1^n), \\ x \leftarrow D_s}}[I(s, f_s(x)) \in f_s^{-1}(f_s(x))] \leqslant \mathrm{negl}(n).$$

EXAMPLE 3.10. We will formally capture subset-sum using this notation. Set $S = \bigcup_{i=1}^{\infty} (\mathbb{Z}_N)^n$ where $N = 2^n$. For all $\bar{a} \in S$, we have $D_{\bar{a}} = \{0,1\}^n$, $R_{\bar{a}} = \mathbb{Z}_N$, and $f_{\bar{a}}(b_1, \ldots, b_n) = \sum a_i b_i$. Gen, D, and F are straightforward.

Lecture 4, 29 Sep

Note that some $\bar{a}$ are bad, but the point is that on average the collection is hard to invert.

EXAMPLE 3.11. Multiplication (in primes): we have $S = \mathbb{N}$, $D_n = \Pi_n \times \Pi_n$, $R_n = \{1, \ldots, 2^{2n}\}$, $\mathrm{Gen}(1^n) = n$, $D(n)$ samples a pair of primes, and $F(x, y) = x \cdot y$.

An important example is *Rabin's function*.

EXAMPLE 3.12 (Rabin's Function). Let $S = \{p \cdot q \mid p, q \text{ prime}\}$, $f_N : \mathbb{Z}_N^* \to \mathbb{QR}_N^*$ be given by $f_N(x) = x^2 \bmod N$. So f_N is a 4-to-1 mapping.

The RSA formula is very similar (the power that x is raised to is different), but is not known to be one-way.

THEOREM 3.13. *The Rabin function is one-way assuming the factoring assumption.*

Note that the opposite is easy: if we can factor efficiently, the Rabin function is not one-way.

CLAIM 3.14. *Let $N = pq$ be a product of two distinct primes, then given any $x_1, x_2 \in \mathbb{Z}_N^*$ satisfying $x_1^2 \equiv x_2^2 \bmod N$ and $x_1 \not\equiv \pm x_2 \bmod N$, then we can efficiently factor N.*[6]

PROOF. By CRT we have $x_1^2 \equiv x_2^2 \bmod p$ and $x_1^2 \equiv x_2^2 \bmod q$. This implies that $x_1 \equiv \pm x_2 \bmod p$ and $x_1 \equiv \mp x_2 \bmod q$, but they are not both $+$ or both $-$. Without loss of generality, assume $x_1 \equiv x_2 \bmod p$ and $x_1 \equiv -x_2 \bmod q$. So $p \mid x_1 - x_2$, and $q \nmid x_1 - x_2$ where $x_1 - x_2 = 2x_1$. So we have $\gcd(x_1 - x_2, N) = p$. □

[6]This idea is used for some of the most efficient factoring algorithms currently known, including the Number Field Sieve.

PROOF OF THEOREM 3.13. First, the Rabin function is clearly efficient.

To show security: assume that there exists a PPT inverter I for the Rabin function, i.e.

$$\Pr_{\substack{N\leftarrow \mathrm{Gen}(1^n),\\ x\leftarrow \mathbb{Z}_N^*}}[I(N, y = x^2 \bmod N) \in \sqrt{y}] \geqslant \delta(n)$$

for some noticeable (non-negligible) $\delta(n)$. The goal is to factor N using I with non-negligible probability.

Now f is 4-to-1, sending $\pm x_1, \pm x_2$ to the same y. So if we can find one element from each of $\pm x_1$ and $\pm x_2$, then we are done from the claim. We construct a factoring algorithm $\mathcal{A}(N)$ as follows:

- Generate $x_1 \in \mathbb{Z}_N^*$ uniformly
- Compute $x_2 = I(N, y = x_1^2 \bmod N)$
- If $x_1^2 = x_2^2$ and $x_1 \neq \pm x_2$, then factor N as in Claim 3.14

We analyze the algorithm. The first thing we want to check is that I succeeds with some noticeable probability. We apply I with N and y distributed correctly, hence it succeeds with probability $\geqslant \delta$. The next step is to show that conditioned on that, the probability of getting $x_1 \neq \pm x_2$ is $\frac{1}{2}$. Intuitively, this is because I does not know x_1 and all four preimages are equally likely.

Overall, our factoring algorithm succeeds with probability at least $\frac{\delta}{2}$, which is non-negligible, contradicting the factoring assumption. □

REMARK 3.15. The problem with factoring is that not all N are created equal, in that some numbers are easier to factor that others. So we do not know how to amplify this algorithm to factor all numbers efficiently.

In some applications we really want to work with a OWP.

EXAMPLE 3.16 (Rabin's permutation). Suppose $p, q \equiv 3 \bmod 4$. This makes sure that $-1 \notin \mathbb{QR}_p^*$ and $-1 \notin \mathbb{QR}_q^*$. This gives a canonical representative of the four preimages of a quadratic residue of $\mathbb{Z}_N^*$. So if we restrict Rabin's function to $\mathbb{QR}_N^*$, we get a bijection.

The question is if this function is still one-way given this restriction. Note that the proof that Rabin's function is hard requires these collisions, but Rabin's permutation does not have them.

EXERCISE 3.17. Determine of Rabin's permutation is still one-way.

Primes of the form $3 \bmod 4$ are sometimes referred to as Rabin primes.

3.4. Applications of OWFs

An important application of OWFs is in storing passwords.

Obviously we do not want to store passwords in the clear. Another important thing is to never reinvent the wheel and use tried-and-true methods instead. Finally, even storing encryptions is bad, because if the decryption key is found, then the password can be decrypted.

So what can be done instead is to store a *hash*, say $y = f(x)$, where f is a OWF. In theory, this is perfectly fine. However in practice, the input is not uniformly random. So in practice people tend to use functions that are somewhat stronger than OWFs. A common hashing function used is the SHA-1 function.

One thing that is done is called *salting*. If a user picks a very simple password, this is a problem, because people have already built what are called *rainbow tables* that allow finding simple passwords via lookup by hash. What salting does is prevent the same function from being used for all passwords. For example, for a user id, we can map password $\mapsto f(\text{id} \circ \text{password})$. If the user id is too simple, we can even fix some unique identifier for the user.

Another thing that is done is *key strengthening*. The idea here is this: say someone wants to attack the database. Then they can just run f on many simple passwords. To prevent the attacker from computing many hashes very quickly, we can just store, for example, $f^{1000}(x) = \overbrace{f \circ f \circ \cdots \circ f}^{1000 \text{ times}}(x)$.

But another problem is that even if the hashing is secure, the password must be sent over some communication link. So we need a secure communication link. After all, if an attacker can get into the server, instead of attacking the database, they can just listen for passwords to be sent by the user. In Zero-Knowledge Proofs, we can prove to the server that we know a preimage of the hash (the password), but the server never gets to see it.

CHAPTER 4

Pseudo-Random Generators

Informally, the idea is that we want a function $f : \{0,1\}^n \to \{0,1\}^{n+1}$ with the following properties: it should be efficiently computable, and for x chosen uniformly from $\{0,1\}^n$, $f(x)$ looks uniform in $\{0,1\}^{n+1}$.

This is difficult: half the range is empty. How could f possibly look uniform? Recall that in our computational cryptography model, we work with an adversary that is limited. This leads to the notion if indistinguishability. So in fact we can even map from n bits into n^2 bits and still seem uniform.

4.1. Indistinguishability and PRGs

This is an essential notion in cryptography. For example, say we have two four-sided die with two different probability distributions. How different are these distributions?

DEFINITION 4.1 (Variational Distance). The *variational distance*[1] between two distributions X and Y is defined to be

$$\Delta(X,Y) = \sup_{A \subset \Omega} |X(A) - Y(A)| .$$

CLAIM 4.2. $\Delta(X,Y) = \frac{1}{2}\sum_{w\in\Omega} |X(w) - Y(w)|$.

EXERCISE 4.3. Prove this.

LEMMA 4.4. *For any (possibly random) function f, $\Delta(f(X), f(Y)) \leqslant \Delta(X,Y)$.*

LEMMA 4.5. *Δ is a metric, i.e., it satisfies the triangle inequality: for all X, Y, Z,*

$$\Delta(X,Z) \leqslant \Delta(X,Y) + \Delta(Y,Z).$$

DEFINITION 4.6. We say that $\{X_n\}, \{Y_n\}$ are statistically indistinguishable if we have $\Delta(X_n, Y_n) \leqslant \mathrm{negl}(n)$.

EXAMPLE 4.7. Take X as uniform over $\{0,1\}^n$, and Y as uniform over $\{0,1\}^n \backslash \{0^n\}$. Then X and Y are statistically indistinguishable since $\Delta(X,Y) = 2^{-n}$.

For distributions X, Y and an algorithm A (possibly randomized), then we have the notion of the *advantage*:

$$\mathrm{Adv}_{X,Y}(A) = |\Pr[A(X) = 1] - \Pr[A(Y) = 1]| .$$

DEFINITION 4.8. $\{X_n\}$ and $\{Y_n\}$ are *computationally indistinguishable* (denoted $X \overset{c}{\approx} Y$) if for all PPT A, $\mathrm{Adv}_{X_n,Y_n}(A) \leqslant \mathrm{negl}(n)$.

[1] This is also sometimes called the statistical distance, though this is ambiguous, since there are many statistical distances. Another name is the ℓ_1-distance.

Note that if we do not assume that A is PPT, then we get exactly statistical indistinguishability.

With this we can finally define a *pseudo-random generator*:

DEFINITION 4.9. A deterministic function $G : \{0,1\}^* \to \{0,1\}^*$ is a PRG with output length $\ell(n) > n$ if:

(1) G is efficiently computable
(2) $|G(x)| = \ell(n)$ if $x \in \{0,1\}^n$.
(3) $G(U_n) \overset{c}{\approx} U_{\ell(n)}$.

Lecture 5, 6 Oct

LEMMA 4.10 (Composition). *For all PPT B, if we have $\{X_n\} \overset{c}{\approx} \{Y_n\}$, then also $\{B(X_n)\} \overset{c}{\approx} \{B(Y_n)\}$.*

PROOF. Let D be any PPT attempting to distinguish $\{B(X_n)\}$ from $\{B(Y_n)\}$. Our goal is to show that its advantage is negligible. Consider an algorithm A that outputs $D(B(\cdot))$. Clearly A is a PPT and by construction,

$$\mathrm{Adv}_{X_n,Y_n}(A) = \mathrm{Adv}_{B(X_n),B(Y,n)}(D)$$

where the left hand side is negligible by assumption. □

The next lemma is basically the triangle inequality, but it is very important and used all over the place.

LEMMA 4.11 (Hybrid Lemma). *If $X^1 \overset{c}{\approx} X^2$ and $X^2 \overset{c}{\approx} X^3$, then $X^1 \overset{c}{\approx} X^3$.*

PROOF. Let D be any PPT attempting to distinguish X^1 from X^3. Our goal is to show that its advantage is negligible. Define $p_i = \Pr[D(X^i) = 1]$. Well, $|p_1 - p_3| \leqslant |p_1 - p_2| + |p_2 - p_3| \leqslant \mathrm{negl} + \mathrm{negl} = \mathrm{negl}$. □

4.2. Expanding PRGs

THEOREM 4.12. *Suppose G is a PRG with output length $\ell(n) = n + 1$. Then for any function $t(n) = \mathrm{poly}(n)$, there exists a PRG G_t of output length $t(n)$.*

This is quite remarkable, because it means that we can take 2^n inputs and output, say $2^{n^{10}}$ outputs such that no PPT can tell the difference between the outputs and uniform.

PROOF. First, define G_t in the following way: Take the seed s and feed it into G. This outputs n bits and an extra bit b_1. Take the first n bits and feed it into G again, which outputs n bits and an extra bit b_2. We continue in this manner for a total $t(n)$ times to obtain $b_1 b_2 \cdots b_t$.

Clearly $|G_t(s)| = t$, and G_t can be computed efficiently. We need to prove that G_t is pseudo-random. The proof will use the Hybrid Lemma.

Define "hybrid distributions" H_i, $i = 0, \ldots, t$ as follows: $H_i = U_i \circ G_{t-i}(U_n)$, that is, the first i bits $b_1 b_2 \cdots b_i$ of H_i are iid uniform, and $b_{i+1} b_{i+2} \cdots b_t$ are given by G_{t-i}. Well, $H_0 = G_t(U_n)$ is exactly what we want to analyze. Our goal is to show that $H_0 \overset{c}{\approx} H_t$.

Towards contradiction, let D be a PPT distinguishing between H_0 and H_t with advantage $\delta \geqslant 1/\mathrm{poly}(n)$. Define $p_i = \Pr[D(H_i) = 1]$, so $|p_t - p_0| = \delta$. By the triangle inequality, there must exist an i such that $|p_i - p_{i-1}| \geqslant \delta/t$, which is noticeable.

Consider the procedure E_i that, given $(x, b) \in \{0,1\}^n \times \{0,1\}$, outputs $U_{i-1} \circ b \circ G_{t-i}(x)$. This is clearly efficient, and $E_i(U_{n+1}) = U_i \circ G_{t-i}(U_n) = H_i$. On the other hand, $E_i((x, b) = G(U_n)) = U_{i-1} \circ b \circ G_{t-i}(x) = H_{i-1}$.

Therefore, the efficient procedure $D \circ E_i(\cdot)$ has advantage $\geqslant \delta/t$ in distinguishing U_{n+1} from $G(U_n)$, in contradiction. □

There is a delicate issue here, which is that in practice, we do not know the good i; we just know that there exists a good i.

One way to deal with this, although it is kind of cheating, is to use non-uniformity. More convincingly, we can estimate the p_i's efficiently ourselves, given D, and find the good i; this is inefficient.

The most elegant solution is to consider the procedure D' that chooses i uniformly from $\{1, \dots, t\}$ and outputs $D(E_i(\cdot))$. Then we have $\Pr[D'(U_{n+1}) = 1] = \frac{1}{t}\sum_{i=1}^{t} p_i$, $\Pr[D'(G(U_n)) = 1] = \frac{1}{t}\sum_{i=0}^{t-1} p_i$, and

$$\left|\frac{1}{t}\sum_{i=1}^{t} p_i - \frac{1}{t}\sum_{i=0}^{t-1} p_i\right| = \frac{1}{t}\left|p_t - p_0\right| = \frac{\delta}{t} \geqslant \frac{1}{\text{poly}}.$$

4.3. Constructing PRGs

We will discuss a construction by Blum and Micali. This will be based on the discrete logarithm.

Recall that $\mathbb{Z}_p^*$ for p prime is a cyclic group, i.e. there exists some $g \in \mathbb{Z}_p^*$ such that $\mathbb{Z}_p^* = \{g^0 = 1, g^1, g^2, \dots, g^{p-2}\}$.

CONJECTURE 4.13 (Discrete Log Assumption). *Let* $\text{Gen}(1^n)$ *be a procedure that outputs a* n*-bit prime* p *with a generator* g *of* $\mathbb{Z}_p^*$*. Then for any PPT* A*,*

$$\Pr_{\substack{p,g\leftarrow \text{Gen},\\ y\leftarrow\mathbb{Z}_p^*}}[A(p,g,y) = \log_g y] = \text{negl}(n).$$

DEFINITION 4.14. The *exponentiation* OWF is the collection $\{f_{p,g}\}$ where $f_{p,g} : \{0, \dots, p-2\} \to \mathbb{Z}_p^*$, $f_{p,g}(x) = g^x \mod p$.

This function is hard to invert by the Discrete Log Assumption. It is also easy to compute by iterated squaring. Now, given p, there is a $\frac{1}{\log\log p}$ fraction of $\mathbb{Z}_p^*$ that are generators. But we do not know how to check this. Instead, we can choose p together with a factorization of $p-1$. This is due to Kalai '02. Another way is to work with Sophie Germain numbers, which are primes of the form $p = 2q+1$ for prime q. The Discrete Log assumption is believed to hold (in fact, even more strongly) for Sophie-Germain primes. It seems with probability $1/n$, if q is a n-bit prime, then $p = 2q+1$ will also be prime. This is not known to be true but works in practice.

The exponentiation OWF is in fact a OWP by identifying $\mathbb{Z}_p^*$ with $\{0, \dots, p-2\}$.

Now the question is, how do we transform this OWP into a PRG?

One idea is to map $x \mapsto (f_{p,g}(x), x)$. But this is not a PRG, because given (a, b), we can check if $a = g^b \mod p$.

So maybe for some function $h : \{0,1\}^n \to \{0,1\}$, we have $x \mapsto (f_{p,g}(x), h(x))$ is a PRG. The first n bits are uniform, but mathematically, the first n bits deterministically determine the last bit. But maybe we can make it hard to predict computationally.

DEFINITION 4.15. A function $h : \{0,1\}^* \to \{0,1\}$ is a *hard-core predicate* (HCP) for f if for all PPT A,

$$\Pr[A(f(x)) = h(x)] \leqslant \frac{1}{2} + \text{negl}(n).$$

EXERCISE 4.16. Show that if h is a HCP for a OWP f, then

$$(f(x), h(x)) \overset{c}{\approx} (U_n.U_1) = U_{n+1}.$$

In other words, $x \mapsto (f(x), h(x))$ is a PRG.

Let us try to find a HCP for the exponentiation OWP.

Our first attempt will be to take the least significant bit of x, i.e. whether x is even or odd. But this is not a hard-core predicate since $(g^x)^{\frac{p-1}{2}} = 1$ if and only if x is even.

But it turns out the most significant bit of x is a HCP based on the Discrete Log Assumption, where the most significant bit can be computed as "$x \geqslant \frac{p-1}{2}$". So $x \mapsto (g^x \bmod p, \text{"}x \geqslant \frac{p-1}{2}\text{"})$ is a PRG.

THEOREM 4.17. *The most significant bit is a hard-core predicate for exponentiation based on the Discrete Log Assumption.*

PROOF SKETCH. Assume towards contradiction that there is an algorithm A that can predict $h(x)$, i.e.,

$$\Pr\left[A(g^x \bmod p) = \text{"}x \geqslant \tfrac{p-1}{2}\text{"}\right] \geqslant \frac{1}{2} + n^{-c}$$

for some $c > 0$.

For simplicity, assume A is always correct, and let us see how to compute the Discrete Log.

By example, suppose $p = 13$, $q = 2$, $y = 9 = g^?$. So say we split the possibilities, say into $0, 1, 2, 3, 4, 5$ and $6, 7, 8, 9, 10, 11$. We can ask A which half, then take y^2, which gives us $0, 2, 4$ and $6, 8, 10$. Then we take y^4 and get $0, 4, 8$, and we learn that the answer is 8. Then tracing back, we hope that $9 = g^8$, assuming that A worked correctly. □

Lecture 6, 20 Oct

A natural question is if there is some $h : \{0,1\}^n \to \{0,1\}$ is a universal HCP for all OWFs. The answer is no, as proved by Goldreich and Levin in 1989: If some f is a OWF then $g(x) = f(x) \circ h(x)$ is also a OWF but h is clearly not a HCP for g.

We therefore want to consider a family of HCPs. One way to make this precise is to define

$$g_f(x, r) = f(x)|r.$$

Then we get the following:

THEOREM 4.18. *If f is a OWF then $h(x, r) := \sum_{i=1}^n x_i r_i \bmod 2$ is a HCP for g_f.*

PROOF. Assume there exists a PPT A that given $f(x)$ and r for random x, r outputs $h(x, r)$ with probability $\geqslant \frac{1}{2} + \varepsilon$ for some $\varepsilon = 1/\operatorname{poly}(n)$. Our goal is to invert f with noticeable probability using A, i.e., given $f(x)$ for uniform x, find a preimage.

Assume that A always succeeds. Then we can simply query A on $A(f(x), e_1) = x_1$ through $A(f(x), e_n) = x_n$. So we manage to recover all of x, and we are happy.

If A does not always work, then we need more work.

So assume that A succeeds with probability $\geqslant \frac{3}{4} + \varepsilon$. Therefore, by a Markov argument, $\Pr_x[x \text{ is good}] \geqslant \frac{\varepsilon}{2}$ where x is "good" if $\Pr_r[A(f(x), r) = h(x, r)] \geqslant \frac{3}{4} + \varepsilon$: otherwise $\Pr_{x,r}[A(f(x), r) = h(x, r)] \leqslant \Pr_x[x \text{ is good}] + \Pr_{x,r}[A(f(x), r) = h(x, r) \mid x \text{ is bad}] < \frac{\varepsilon}{2} + \frac{3}{4} + \frac{\varepsilon}{2}$, in contradiction.

So it suffices to show how to invert f on good x. We show how to find x_i:

The procedure is as follows: choose a random r, and query A on r and on $r \oplus e_i$, then output the XOR of both answers. If both questions are successful the output is x_i as needed. The first query is successful with probability $\geqslant \frac{3}{4} + \frac{\varepsilon}{2}$ because it is uniform.

Similarly, the second query is also successful with probability $\geqslant \frac{3}{4} + \frac{\varepsilon}{2}$, since $r \oplus e_i$ is also uniform. Therefore by union bound,

$$\Pr[\text{both successful}] \geqslant 1 - \left(\frac{1}{4} - \frac{\varepsilon}{2}\right) - \left(\frac{1}{4} = \frac{\varepsilon}{2}\right) = \frac{1}{2} + \varepsilon.$$

To amplify this, we can use the Chernoff bound[2]: repeating this procedure independently for $\ell = O((\log n)/\varepsilon^2)$ times gives us a guess for x_i that is correct with probability $\geqslant 1 - \frac{1}{n^2}$. Then doing this for all bits x_i and by union bound, we can recover all bits correctly with probability $\geqslant 1 - \frac{1}{n}$.

Finally, assume A is correct with probability $\frac{1}{2} + \varepsilon$ (this is our goal). As before, by a Markov argument, $\frac{\varepsilon}{2}$ of the x's are good in the sense that A is correct for them with probability $\geqslant \frac{1}{2} + \frac{\varepsilon}{2}$.

Pick $r_1, \dots, r_t \in \{0,1\}^n$ uniformly for $t = O((\log n)/\varepsilon^2)$. Then, guess the values of $h(x, r_1), \dots, h(x, r_t)$ (or try all options), and for $i = 1, \dots, n$, the i-th bit of the output is the majority

$$\mathrm{MAJ}(A(f(x), r_1 \oplus e_i) \oplus h(x, r_1), \dots, A(f(x), r_t \oplus e_i) \oplus h(x, r_t)).$$

Let us see why this works. With probability $2^{-t} = n^{-O(1/\varepsilon^2)}$, all our guesses are correct. Conditioned on that, for each i, by Chernoff again we have success probability $\geqslant 1 - \frac{1}{n^2}$ in predicting x_i correctly. Then again by union bound again with probability $\geqslant 1 - \frac{1}{n}$ we output x. Since for constant ε, $n^{-O(1/\varepsilon^2)}$ is noticeable, we are done. □

There is actually a problem, that we assumed that ε is constant. We also have to deal with $\varepsilon = \frac{1}{\mathrm{poly}(n)}$. For that, we can observe that guessing h at t locations actually gives the values of h in 2^t locations, since we can just take linear combinations. But the trouble is that now we lose independence, so we cannot use Chernoff's bound. Luckily, they are pairwise independent so we can instead use Chebyshev's inequality, and $t = \log(4n/\varepsilon^2)$ suffices.

What we have shown is that existence of OWPs implies existence of PRGs.

4.4. Pseudo-Random Functions

Lecture 7, 27 Oct

A PRF can be thought of as a PRG with exponentially long output. But this does not really make sense, since we cannot output this in polynomial time. Furthermore, there is an issue of security since the adversary can be given time 2^n. So to fix this instead of sequentially outputting the string, we just require that we can query the bit (or polynomial-size block) at any location index.

DEFINITION 4.19. A family $\{f_s : \{0,1\}^{\ell_1(n)} \to \{0,1\}^{\ell_2(n)}\}_{s \in \{0,1\}^n}$ (or equivalently, "$f_s \in \{0,1\}^{\ell_1(n)} \times \{0,1\}^{2^{\ell_2(n)}}$") is a PRF family if

(1) It is efficiently computable, i.e. there is a deterministic poly-time algorithm F such that $F(s, x) = f_s(x)$.
(2) It is pseudorandom: for all PPT D,

$$\mathrm{Adv}_{\{f_s\},\{U\}} := \left| \Pr_{s \in \{0,1\}^n}[D^{f_s} = 1] - \Pr_U[D^U = 1] \right| \leqslant \mathrm{negl}(n).$$

(Sometimes this is called *oracle indistinguishability*.)

[2]THEOREM (Chernoff). *If $z_1, \dots, z_\ell$ are iid Boolean random variables with expectation μ then* $\Pr\left[\left|\sum z_i/\ell - \mu\right| \geqslant \varepsilon\right] \leqslant 2^{-\ell\varepsilon^2}$.

So why do we need a family? Why can't we just use a fixed function, as is sometimes done in applied cryptography? The reason is because otherwise we do not know how to define security.

When $\ell_1 = O(\log n)$ the PRF is in fact a PRG. This is because in polynomial time we can just query the oracle on all locations. Hence existence of PRFs imply existence of PRGs.

So how do we construct PRFs? We will produce one using PRGs (which are in turn built on OWFs).

THEOREM 4.20. *If PRGs exist, then PRFs exist for any ℓ_1, ℓ_2 that are* $\mathrm{poly}(n)$.

PROOF. We showed before that using a PRG we can take n bits and output $2n$ bits. The idea is that we can take each n-bit block, and double them again, and so on. This gives exponential growth. The problem is how to actually compute this.

Assume for simplicity that $\ell_1(n) = \ell_2(n) = n$. Let G be a PRG and without loss of generality assume that G has output length $2n$.

Denote $G(s) = G_0(s)|G_1(s)$, that is, $G_0(s)$ is the first half of the output of $G(s)$, and $G_1(s)$ the second half. We can draw a tree that branches by applying G_0 and G_1 on the parent node. At the bottom we have 2^n leaves, $G_0(G_0(\dots(s)))$ through $G_1(G_1(\dots(s)))$. So these leaves are indexed by binary strings of length n. We can compute a block by tracing the path in the tree to the corresponding leaf. This takes polynomial time. More formally, $f_s(x) = G_{x_n}(G_{x_{n-1}}(\dots(G_{x_1}(s))))$.

To prove security is harder. As a first attempt following the proof for PRGs, define H_i for $i = 0, \dots, 2^n$ to be the distribution in which the first i blocks are replcaed by uniform blocks. The idea is to show that $H_0 \overset{c}{\approx} H_{2^n}$, by first showing that for all i, $H_i \overset{c}{\approx} H_{i+1}$, and then conclude by the Hybrid Lemma that $H_0 \overset{c}{\approx} H_{2^n}$. Although it is true that $H_i \overset{c}{\approx} H_{i+1}$ (the proof for this is subtle), the problem is that the chain is exponentially long; the Hybrid Lemma only works for polynomially long chains.[3]

The way to proceed is to define H_i as top-to-bottom on the levels of the tree instead of as left-to-right. So let H_0 be the original tree for the PRF, H_1 be the tree with level 1 replaced by two blocks of uniform strings, and similarly for $H_2, \dots, H_n$. More formally the hybrid is $f(x)_{s_y} = G_{x_n}(G_{x_{n-1}}(\dots(G_{x_{i+1}}(s_{x_i,\dots,x_1}))))$ where $\{s_y\}_{y\in\{0,1\}^i}$ are iid uniform.

Now assume towards contradiction that $H_0 \overset{c}{\not\approx} H_1$, i.e. there exists a PPT D that distinguishes H_0 from H_1. Our goal is to use D to distinuisgh $G(s)$ from $(z_0, z_1) \in \{0,1\}^{2n}$ uniform. Our input is $\{0,1\}^{2n}$ and our output is a procedure that takes index x and outputs $G_{x_n}(G_{x_{n-1}}(\dots G_{x-1}(y))$. If (z_0, z_1) are taken from $G(U_n)$ then the output is H_0. If (z_0, z_1) is uniform, our output is H_1. Therefore by applying D to the procedure, we can break the PRG in contradiction.

The next step is to show that $H_1 \overset{c}{\approx} H_2$. This is similar, but for H_1 we now need two independent inputs, either both from U_{2n} or both from $G(U_n)$. But it is easy to show by the Hybrid Lemma that $(G(U_n), G(U_n)) \overset{c}{\approx} (U_{2n}, U_{2n})$, so we are done.

We can try to continue this, but at some point we run into a serious issue: we will need $G(U_n)^{2^i} \overset{c}{\approx} U_{2n}^{2^i}$. To fix this, instead of doing everything in advance, we can just do this on demand; indeed when querying we compute blocks on demand anyway. Notice that: in H_i, all the trees growing from the i-th level are iid; furthermore the distinguisher D attacking the PRF can only make polynomially many queries.

[3]EXAMPLE. $H_i = U(\{i, i+1, \dots, i+2^n\})$. Then $H_i \approx H_{i+1}$, even statistically. However, H_0 is easily distinguishable from H_{2^n}.

So we will use a simulator S_i for simulating H_{i-1} and H_i as follows: The input is $(z_0^1, z_1^1), \ldots, (z_0^q, z_1^q) \in \{0,1\}^{2n}$ for some large enough $q = \text{poly}(n)$. Then:
- $j \leftarrow 1$
- Given a query $x \in \{0,1\}^n$:
  - If this is the first time we see $(x_1, \ldots, x_{i-1})$, associate with j and increment j.
  - Output $G_{x_n}(\ldots(G_{x_{i+1}}(z_{x_i}^j))\ldots)$ where j is associated with $x_1, \ldots, x_{i-1}$.

Notice that we have enough enputs so that j never overflows. Furthermore, if the inputs are iid from $G(U_n)$ then we simulate H_{i-1}, and if the inputs are iid from U_{2n} we simulate H_i.

Hence $H_{i-1} \overset{c}{\approx} H_i$, and so by the Hybrid Lemma $H_0 \overset{c}{\approx} H_n$. □

One nice implication for this is in learning theory: PRFs are strongly unlearnable. But notice that we can construct PRFs from very simple circuits (in fact, PRFs exist in NC_1), and there are many classes of problems that include PRFs, so it is hopeless to try to learn them.

In application, there are serious issues of loss in security when moving from one construction to another. Naor and Reingold solved this issue in 1997.

4.5. Pseudo-Random Permutations

DEFINITION 4.21. A family of permutations $\{f_s : \{0,1\}^n \to \{0,1\}^n\}_{s\in\{0,1\}^n}$ is a PRP if

(1) It is efficiently computable and invertible.
(2) It is pseudorandom: for all PPT D,

$$\text{Adv}_{\{f_s\},\{U\}} := \left| \Pr_{s\in\{0,1\}^n}[D^{f_s} = 1] - \Pr_U[D^U = 1] \right| \leqslant \text{negl}(n).$$

In a strong PRP, we allow access to both f_s and f_s^{-1}.

This differs from OWPs in that it is f_s that is a permutation, not the mapping from the seed s to f_s.

In applied cryptography a strong PRP is more or less what is referred to as a "block cipher". AES is an example.

CLAIM 4.22. *A random permutation is (oracle) indistinguishable from a random function.*

PROOF IDEA. Assume that a distinguisher D performs t queries for $t = \text{poly}(n)$. In a random function it gets t iid uniform values in $\{0,1\}^n$. So it gets a uniform view among all 2^{nt} views. In a random permutation, only $(2^n)!/(2^n - t)!$ views are possible and we get one uniformly. The claim that the two distributions are within negligible statistical difference (in fact, it is $\leqslant \frac{1}{2^n} + \frac{2}{2^n} + \cdots + \frac{t}{2^n} \leqslant \frac{t^2}{2^n}$). □

So PRFs are indistinguishable from PRPs, so why do we want PRPs at all? Well, the reason is because it is invertible, which is useful for encryption.

We can construct PRPs from PRFs, using what is known as a *Feistel round*. What we do is take a string and break it into two parts L and R, and then obtain $L' = R$ and $R' = L \oplus f(R)$ where f is a PRF. This is not a random function, but we will see that applying this three times gives a PRP. We will not show it, but in fact applying it a fourth time gives a strong PRP. Lecture 8, 3 Nov

TODO: Fill in

CHAPTER 5

Encryption

5.1. Symmetric-Key Encryption

TODO: Fill in Lecture 9, 10 Nov

DEFINITION 5.1 (IND-CPA). (Chosen Plaintext Attack)

We now construct a PRF-Based SKE.
Assume $\{f_k : \{0,1\}^n \to \{0,1\}^n\}_k$ is a PRF family. We have

- Gen: outputs a uniform $k \leftarrow \{0,1\}^n$
- $\mathrm{Enc}_k(m)$: chooses $s \leftarrow U_n$, and outputs $(r, f_k(r) \oplus m)$
- $\mathrm{Dec}_k(r, c)$: outputs $f_k(r) \oplus c$

Notice that we can encrypt the same message many many times but it still looks random, since r is uniform and $f_k(r)$ looks random.

THEOREM 5.2. *This scheme is IND-CPA secure.*

PROOF. We will show that for fixed b, $(\mathrm{Enc}_k^b(\cdot,\cdot))_k \overset{c}{\approx} (U(\cdot,\cdot))$. We use the Hybrid Lemma.

Fix b. We will show $H_0 \overset{c}{\approx} H_1 \overset{c}{\approx} H_2$ where H_0 chooses $k \leftarrow \{0,1\}^n$ and answers query (m_0, m_1) by choosing $r \leftarrow U_n$ and outputting $(r, f_k(r) \oplus m_b)$; H_1 uses a random function $F : \{0,1\}^n \to \{0,1\}^n$ and answers a random query (m_0, m_1) by choosing $r \leftarrow U_n$ and outputting $(r, F(r) \oplus m_b)$; and H_2 answers all queries with a uniform element in $\{0,1\}^{2n}$.

$H_0 \overset{c}{\approx} H_1$ by the PRF assumption: if there is a distinguisher that can distinguish H_0 from H_1 we can construct a distinguisher between $\{f_k\}$ and a truly random F, by implementing H_0 or H_1 given the function (f_k or F).

H_1 and H_2 are almost the same, except that in H_1, if the first n bits are the same then the second n bits must also be the same. But this probability is small, so in fact $H_1 \overset{s}{\approx} H_2$. Conditioned on all the r's being different, the two are identical. This happens with probability $\geqslant 1 - \frac{q^2}{2n}$, since there are q^2 locations for collision and $\frac{1}{2^n}$ probability for collision. So the distance is $\leqslant \frac{q^2}{2^n} = \mathrm{negl}(n)$. □

Another way of constructing SKEs is using PRGs. This is often called a *stream cipher*. Assume $G : \{0,1\}^n \to \{0,1\}^{2n}$ is a PRG and write $G(k) = G_1(k)|G_2(k)$. We have

- Gen: outputs a uniform $k \leftarrow \{0,1\}^n$
- $\mathrm{Enc}_k(m)$: outputs $c = m \oplus G_1(k)$ and updates $k \leftarrow G_2(k)$
- $\mathrm{Dec}_k(c)$: outputs $c \oplus G_1(k)$ and updates $k \leftarrow G_2(k)$

This is an example of what is called a *stateful encryption scheme*, which is perfectly fine in some applications. The advantage is that it is deterministic, which makes it easy to implement. The statefulness is why the scheme is secure. Note that there are issues with synchronicity: if the receiver misses a message the keys may go out of sync. The scheme is

also *forward-secure*: if the key is leaked at some point, past messages are still secure, since $G_1(k), G_1(G_2(k)), \ldots, G_1(G_2(\ldots G_2(k)))$ and $G_2(G_2(\ldots G_2(k)))$ are pseudorandom, and $G_2(G_2(\ldots G_2(k)))$ is the key that is leaked.

Another nice implementation is DDH, see Dodis Lecture 8.

Some more ciphers based on PRPs and PRFs are:

- ECB (electronic codebook), which uses a PRP with $\mathrm{Enc}_k(m) = f_k(m)$ and $\mathrm{Dec}_k(c) = f_k^{-1}(c)$. This is deterministic, so it is not multimessage secure.
- R-CNT (we saw this before) which uses a PRF with $\mathrm{Enc}_k(m) = (r, f_k(r) \oplus m)$ and $\mathrm{Dec}_k(r, c) = f_k(r) \oplus c$.
- CNT, a stateful version of R-CNT which uses a PRP with $\mathrm{Enc}_k(m)$ that outputs $(\text{counter}, f_k(\text{counter}) \oplus m)$ and implementing the counter.
- R-nonce, which uses a PRP with $\mathrm{Enc}_k(m) = f_k(m \circ r)$ and $\mathrm{Dec}_k(c)$ outputs the first half of $f_k^{-1}(c)$.
- R-IV, which uses a PRP with $\mathrm{Enc}_k(m) = (r, f_k(r \oplus m))$, and $\mathrm{Dec}_k(r, c) = f_k^{-1}(c) \oplus r$.

Some of these are wasteful, for example R-IV would send as many random bits as message bits. There are ways to get around this, for example sending r_1 once and then using r_2 then r_3 etc. This is known as Modes of Operations. For example:

- R-CTR: given messages $(m_1, \ldots, m_n)$ the scheme picks a random r and outputs $(r, c_1, \ldots, c_n)$ where $c_i = f_k(r + i - 1) \oplus m_i$.

5.2. Public-Key Encryption

We showed Weak OWF $\implies$ OWF $\implies$ PRG $\implies$ PRF $\implies$ SKE $\implies$ OWF.

As it turns out we cannot have Weak OWF $\implies$ PKE.

The idea for is as follows: Alice chooses a pair $(\mathfrak{sk}, \mathfrak{pk})$ and broadcasts $\mathfrak{pk}$, then Bob encrypts any mesage using $\mathfrak{pk}$ but only Alice can decrypt it using sk.

More formally:

- $\mathrm{Gen}(1^m)$ outputs public key $\mathfrak{pk}$ and secret key $\mathfrak{sk}$
- $\mathrm{Enc}_{\mathfrak{pk}}(m)$ outputs c
- $\mathrm{Dec}_{\mathfrak{sk}}(m)$ outputs m

where Correctness is defined as: for $(\mathfrak{pk}, \mathfrak{sk}) \leftarrow \mathrm{Gen}$, $\forall m\ \mathrm{Dec}_{\mathfrak{sk}}(\mathrm{Enc}_{\mathfrak{pk}}(m)) = m$.

Security is somewhat harder to define. Some possibilities are:

- Weak definition: cannot extract m from $\mathrm{Enc}_{\mathfrak{pk}}(m)$ ("one-wayness")
- SKE definition: $(\mathrm{Enc}^0_{\mathfrak{pk}}(\cdot,\cdot))_{\mathfrak{pk}} \overset{c}{\approx} (\mathrm{Enc}^1_{\mathfrak{pk}}(\cdot,\cdot))_{\mathfrak{pk}}$, but this bad since it does not capture that $\mathfrak{pk}$ is public (in fact any IND-CPA secure SKE is also secure under this definition if we take $\mathfrak{pk} = \mathfrak{sk}$).
- $\forall q, m_1, \ldots, m_q, m_0, m_0'$,

$$(\mathfrak{pk}, \mathrm{Enc}_{\mathfrak{pk}}(m_1), \ldots, \mathrm{Enc}_{\mathfrak{pk}}(m_q), \mathrm{Enc}_{\mathfrak{pk}}(m_0))_{\mathfrak{pk}} \overset{c}{\approx} (\mathfrak{pk}, \mathrm{Enc}_{\mathfrak{pk}}(m_1), \ldots, \mathrm{Enc}_{\mathfrak{pk}}(m_q), \mathrm{Enc}_{\mathfrak{pk}}(m_0'))_{\mathfrak{pk}}.$$

  But this is equivalent to single-message security: $\forall m, m'$, $(\mathfrak{pk}, \mathrm{Enc}_{\mathfrak{pk}}(m)) \overset{c}{\approx} (\mathfrak{pk}, \mathrm{Enc}_{\mathfrak{pk}}(m'))$.

Finally this is the definition we will use:

DEFINITION 5.3 (IND-CPA). A PKE is IND-CPA secure if the following is indistinguishable for $b = 0, 1$:

$$(\mathfrak{pk}, C^b_{\mathfrak{pk}}(m_0, m_1) := \mathrm{Enc}_{\mathfrak{pk}}(m_b))_{\mathfrak{pk}}.$$

As always encryption must be randomized. Statefulness in PKE makes no sense.

This is equivalent to what is known as *semantic security*.

We often work with encryptions of one bit, $\mathcal{M} = \{0, 1\}$ (this implies $\mathrm{poly}(n)$ many-bit encryption).

To construct PKEs we are no longer able to use OWFs and PRFs. What we will use are called Trapdoor OWPs (TDP)s, which are a family $\{f_s : \{0,1\}^n \to \{0,1\}^n\}$, such that when choosing s we can also choose t such that $f_s^{-1} : \{0,1\}^n \to \{0,1\}^n$ is efficiently computable using the information t. For example in the Rabin permutation we used $N = pq$; the trapdoor is p and q. For the function $x \mapsto g^x \bmod p$, it is unclear what the trapdoor would be, but in fact we can still use it for PKE. Lecture 10, 17 Nov

DEFINITION 5.4. $\{f_s : D_s \to D_s\}$ is a family of trapdoor one-way permutations (TDPs) if

- There is a PPT sampler $S(1^n)$ that outputs s and trapdoor t
- There is a PPT inverter I such that $I(t, y) = f_s^{-1}(y)$ for all $y \in D_s$
- It is a OWP (with respect to s) and as usual, the adversary is given s and $f_s(x)$ for random x.
- We have hard-core bits for the OWP (either specific or Goldreich-Levin)

Using OWP with a HCP h, we construct a PKE for $\mathcal{M} = \{0, 1\}$ as follows:

- Gen: takes $(s, t) \leftarrow S$ and outputs $\mathfrak{pk} = s$, $\mathfrak{sk} = t$.
- $\mathrm{Enc}_s(m)$: Choose $r \in D_s$ and output $c = (f_s(r), h(r) \oplus m)$.
- $\mathrm{Dec}_t(c = (y, c'))$: output $h(f_s^{-1}(y)) \oplus c'$.[1]

THEOREM 5.5. *This PKE is IND-CPA secure, i.e.* $(\mathfrak{pk}, \mathrm{Enc}(0)) \overset{c}{\approx} (\mathfrak{pk}, \mathrm{Enc}(1))$.

PROOF. By the definition of HCP,

$$(s, f_s(r), h(r) \oplus 0) \overset{c}{\approx} (s, f_s(r), U_1) \equiv (s, f_s(r), U_1 \oplus 1) \overset{c}{\approx} (s, f_s(r), h(r) \oplus 1).$$

□

Some candidate TDPs are:

- Rabin's function $f_N : \mathbb{QR}_N^* \to \mathbb{QR}_N^*$, $f_N(x) = x^2 \bmod N$, which is believed to be a OWP; both the least and most significant bits are HCPs. The trapdoor is the factorization $N = pq$ where $p, q \equiv 3 \bmod 4$: given p, q we can compute square roots.
- RSA $f_{N,e} : \mathbb{Z}_N^* \to \mathbb{Z}_N^*$, $f_{N,e} = x^e \bmod N$, $N = pq$ and e coprime to $\varphi(N) = (p-1)(q-1)$, this guarantees $f_{N,e}$ is a permutation. The trapdoor is $d = e^{-1} \bmod \varphi(N)$: we can compute $(x^e)^d = x^{ed \bmod \varphi(N)} = x$. Each bit of x is hard-core.
- Modular exponentiation $f_{p,g}(x) = g^x \bmod p$, but there is no known trapdoor.

[1] An incorrect attempt would be to pick $r \in \{0,1\}^n$ and output $(f_s(r), r \oplus m)$; this is incorrect because OWP can allow $f_s(r)$ to reveal a lot of information about r.

5.3. Historical SKEs and PKEs

5.3.1. Diffie-Hellman's SKE. Historically, one of the first SKEs was Diffie-Hellman. Let G be a (multiplicative) cyclic group with known generator g and order q. The protocol is as follows:

- Alice chooses $a \leftarrow \mathbb{Z}_q$, and Bob chooses $b \leftarrow \mathbb{Z}_q$.
- Alice sends $\alpha = g^a$, and Bob sends $\beta = g^b$.
- They agree on $g^{ab} = \alpha^b = \beta^a$.

It is not obvious how to obtain g^{ab} from knowing only g^a and g^b, but this does not mean security.

CONJECTURE 5.6 (Computational Diffie-Hellman (CDH)). *Given g, g^a, g^b it is hard to compute g^{ab}.*

But this is not that great, because we can obtain incomplete information about g^{ab}; recall that such a case was the motivation for HCPs. So we need a stronger assumption:

CONJECTURE 5.7 (Decisional Diffie-Hellman (DDH)). *For $a, b, c \leftarrow \mathbb{Z}_q$ uniformly, $(g, g^a, g^b, g^{ab}) \overset{c}{\approx} (g, g^a, g^b, g^c)$.*

DDH says that Diffie-Hellman is a secure key exchange.

Actually, DDH is false for $\mathbb{Z}_p^*$! Note that while g^a and g^b are uniform, g^{ab} is a quadratic residue with probability $\frac{3}{4}$, whereas g^c is a quadratic residue with probability $\frac{1}{2}$. Instead, we work with $\mathbb{QR}_p^*$ for $p = 2q + 1$, which makes $\mathbb{QR}_p^*$ a group of prime order q.

5.3.2. El Gamal's PKE. A similar idea is the El Gamal PKE, where for known g:

- Gen: $\mathfrak{sk} = a \leftarrow \mathbb{Z}_q$, $\mathfrak{pk} = \alpha = g^a$
- $\mathrm{Enc}_\alpha(m)$: choose $b \leftarrow \mathbb{Z}_q$ and output $c = (g^b, \alpha^b \cdot m)$
- $\mathrm{Dec}_a(\beta, c')$: output c'/β^a

The correctness of this scheme is obvious.

PROPOSITION 5.8. *The scheme is IND-CPA secure based on DDH.*

PROOF. The goal is to show that R_0 and R_1 are indistinguishable, where $R_0 = (g, g^a, (m_0, m_1) \overset{b}{\mapsto} (g^b, g^{ab} \cdot m_0))$ and $R_1 = (g, g^a, (m_0, m_1) \overset{b}{\mapsto} (g^b, g^{ab} \cdot m_1))$. More precisely, we show that both are indistinguishable from $U = (g, g^a, (m_0, m_1) \overset{b,c}{\mapsto} (g^b, g^c))$, then we have $R_0 \overset{c}{\approx} U \overset{c}{\approx} R_1$.

Without loss of generality, we show that $R_0 \overset{c}{\approx} U$. Assume by contradiction that there is a distinguisher. We will break DDH. We get (g, g^a, g^b, g^c) where $a, b \leftarrow \mathbb{Z}_q$ where c is either ab or uniform from $\mathbb{Z}_q$. Construct an oracle in the following way: $(g, g^a, (m_0, m_1) \mapsto (g^b, g^c \cdot m_0)$. If $c = ab$ this is R_0, and if c is uniform this is U. □

5.4. Semantic Security

It is not at all obvous that IND-CPA is the right definition of security for PKE. Semantic security was defined roughly to say that knowing the encryption scheme is useless; this leads to zero-knowledge proofs. It turns out that semantic security is equivalent to IND-CPA (Goldwasser-Micali '84)

Lecture 11, 24 Nov

DEFINITION 5.9. A PKE scheme is *semantically secure* if for all PPT attackers A_0, A_1 there exists a PPT simulator S such that

$$|P_a - P_s| \leqslant \mathrm{negl}(n),$$

where P_a and P_s are defined in the model below.

The model is as follows: There is a known public key $\mathfrak{pk}$. A_0 outputs a message m and some "test procedure" R and some "memory snapshot" α. A_1 takes α_1 and $\mathrm{Enc}_{\mathfrak{pk}}(m)$, and outputs some z. They are successful if $R(m,z)=1$, and $P_a = \Pr[R(m,z)=1]$. On the other hand, S is only given α, and outputs z. S is successful if $R(m,z)=1$ and $P_s = \Pr[R(m,z)=1]$.

THEOREM 5.10. *A PKE is semantically secure if and only if it is IND-CPA secure.*

PROOF. We first show that IND-CPA security implies semantic security. By the contrapositive, assume that there exists A_0, A_1 such that for all S, $|P_a - P_s| > \frac{1}{\mathrm{poly}(n)}$. Since we can choose S, pick S that calls A_1 with α and $\mathrm{Enc}_{\mathfrak{pk}}(0)$. Then by assumption there must be a gap in the probability that $R(m,z)=1$. The attack on IND-CPA is as follows: run A_0 and get $\alpha, \mathrm{Enc}_{\mathfrak{pk}}(m), R$ then calls the challenge oracle with $(m_0 = m, m_1 = 0)$. Use the result as input to A_1 and accept if and only if $R(m,z)=1$.

Next, we show the other direction: semantic security implies IND-CPA security. Again by the contrapositive, assume we have an attacker B on IND-CPA. Partition B into two procedures: B_0 is whatever happens before the challenge, which outputs (m_0, m_1, β) where m_0, m_1 are the challenge messages and β is a memory snapshot. B_1 is what happens after the challenge, with input (m_0, m_1, β) and $\mathrm{Enc}_{\mathfrak{pk}}(m_b)$ and guesses b with probability $> \frac{1}{2} + \frac{1}{\mathrm{poly}(n)}$. Then A_0 takes as input $(m_0, m_1, \beta) \leftarrow B_0$ and outputs $(m_b, R, \alpha = (m_0, m_1, \beta))$ where $b \leftarrow \{0,1\}$ uniformly and R is "equality". A_1 gets input $(c, (m_0, m_1, \beta))$ and outputs $z = m_b$ where $b = B_1(c, \beta)$. By our construction, $P_a \geqslant \frac{1}{2} + \frac{1}{\mathrm{poly}(n)}$. On the other hand without $c = \mathrm{Enc}_{\mathfrak{pk}}(m_b)$, there is no information about b, so $P_s = \frac{1}{2}$ for all S. □

Of course, the IND-CPA definition is much easier to use.

CHAPTER 6

Authentication

The idea is that we want to verify that a message has not been tampered with. The way we do this is we associate with each message a signature in such a way that it is not possible to predict the signatures of future messages.

6.1. Message Authentication Codes

The MAC (Message Authentication Code) is a secret key model defined as follows:

- Gen outputs a key k
- $\mathrm{Tag}_k(m)$ outputs a "tag" $t \in \mathcal{T}$
- $\mathrm{Ver}_k(m, t)$ either accepts or rejects.

For correctness, we should have for all m, $k \leftarrow \mathrm{Gen}$, $t \leftarrow \mathrm{Tag}_k(m)$ then $\mathrm{Ver}_k(m, t) =$ Accept.

For security, we have some possibilities:

- For all PPT A outputting some m, t, $\Pr[\mathrm{Ver}_k(A(\cdot)) = \text{Accept}] \leqslant \mathrm{negl}(n)$. This is very weak, as it is easy to achieve by giving the attacker nothing, and just using a fixed random string.
- Since it is not fair for the attacker to know nothing, so maybe we give the attacker the key, then $\Pr[\mathrm{Ver}_k(A(k)) = \text{Accept}] \leqslant \mathrm{negl}(n)$. But this is impossible.
- Somewhat similar to IND-CPA, $\Pr[\mathrm{Ver}_k(A^{\mathrm{Tag}_k, \mathrm{Ver}_k}) = \text{Accept}] \leqslant \mathrm{negl}(n)$. The only problem is that we don't want the attacker to sign their own documents and then give it to the verifier. What we need is for the attacker to produce something new.

Our information-theoretic definition of security will be as follows:

DEFINITION 6.1 (Perfect Unforgeability). A MAC is perfectly unforgeable if for all (unbounded) $\mathcal{F}$, and $m \in \mathcal{M}$,

$$\mathrm{Adv}(\mathcal{F}) = \Pr_{\substack{k \leftarrow \mathrm{Gen}, \\ t \leftarrow \mathrm{Tag}_k(m)}} [\mathcal{F}(m, t) \text{ succeeds}] \leqslant \frac{1}{|\mathcal{T}|}$$

where $\mathcal{F}(m, t)$ succeeds means $\mathcal{F}$ outputs (m', t') such that $\mathrm{Ver}_k(m', t') =$ Accept and $m \neq m'$.

This is the most basic setting, where we only use the system once.

REMARK 6.2. The goal of the attacker was the sign a new message. We can ask for a more modest task: either sign a different message, or sign the same message with a different tag, that is output (m, t') such that $t \neq t'$ and $\mathrm{Ver}_k(m, t') =$ Accept. This is called *strong unforgeability*. When Tag is deterministic and Ver is canonical, that is, $t \stackrel{?}{=} \mathrm{Tag}(m)$, then strong unforgeability is equal to perfect unforgeability.

REMARK 6.3. The term *existential unforgeability* refers to the model in which the adversary just needs to successfully tag one message of its choice.

DEFINITION 6.4. A family of functions $\mathcal{H} = \{h_k : \mathcal{M} \to \mathcal{T}\}_k$ is *pairwise independent* if for all $m, m' \in \mathcal{M}$, $m \neq m'$, the random variable $(h_k(m), h_k(m'))_k$ is uniform in $\mathcal{T}^2$.

A totally random family of functions satisfies this. But we want something better.

EXAMPLE 6.5. Suppose we have three messages and four functions:

k \ m			
	0	0	0
	0	1	1
	1	0	1
	1	1	0

EXAMPLE 6.6. Let p be a prime. Let $h_{a,b}(x) = ax + b \bmod p$ for $(a, b) \in \mathbb{Z}_p^2$, $\mathcal{M} = \mathcal{T} = \mathbb{Z}_p$. For all distinct m, m',

$$\begin{aligned}\Pr[h_{a,b}(m) = t \text{ and } h_{a,b}(m') = t'] &= \Pr\left[\begin{pmatrix} m & 1 \\ m' & 1 \end{pmatrix}\begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} t \\ t' \end{pmatrix}\right] \\ &= \Pr\left[\begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} m & 1 \\ m' & 1 \end{pmatrix}^{-1}\begin{pmatrix} t \\ t' \end{pmatrix}\right] = \frac{1}{p^2}.\end{aligned}$$

Then we can create a MAC where

- Gen outputs $h_k \leftarrow \mathcal{H}$
- $\mathrm{Tag}_k(m) = h_k(m)$
- $\mathrm{Ver}_k(m, t) = (h_k(m) \stackrel{?}{=} t)$

Lecture 12, 1 Dec

THEOREM 6.7. *This MAC is perfectly unforgeable (even strongly).*

PROOF. For all $\mathcal{F}$, $m \in \mathcal{M}$,

$$\mathrm{Adv}(\mathcal{F}) = \frac{1}{|\mathcal{M}|}\sum_{t\in\mathcal{T}} \Pr[\mathcal{F}(m, t) \text{ succeeds} \mid h_k(m) = t] = \frac{1}{|\mathcal{T}|}\sum_{t\in\mathcal{T}} \frac{1}{|\mathcal{T}|} = \frac{1}{|\mathcal{T}|}. \quad \square$$

We will now move on to computational notions of unforgeability.

DEFINITION 6.8 (UF-CMA). A MAC is unforgeable under chosen-message attack (UF-CMA) if for all PPT $\mathcal{F}$,

$$\mathrm{Adv}(\mathcal{F}) := \Pr[\mathcal{F}^{\mathrm{Tag}_k} \text{ succeeds}] \leqslant \mathrm{negl}(n)$$

where $\mathcal{F}^{\mathrm{Tag}_k}$ succeeding as before, $\mathcal{F}^{\mathrm{Tag}_k}$ outputs (m', t') such that $\mathrm{Ver}_k(m', t') = 1$ and m' was not given to Tag_k.

REMARK 6.9. Giving $\mathcal{F}$ in addition the verificaiton oracle Ver_k does not help.

THEOREM 6.10. *The same MAC as above with $\mathcal{H}$ a PRF family is UF-CMA.*

PROOF. As is standard, we want to replace PRF with a truly random function. So if a forger has noticeable advantage with a PRF, it should still have noticeable advantage with a truly random function.

By contrapositive assume $\mathcal{F}$ is a forger for the MAC. We construct a distinguisher $\mathcal{D}$ between PRF and uniform function. $\mathcal{D}$ is given a function $g : \{0,1\}^n \to \{0,1\}^n$ that is either from the PRF family or uniform. Then $\mathcal{D}$ runs $\mathcal{F}$ and answers $\mathrm{Tag}(m)$ by $g(m)$.

$\mathcal{F}$ outputs (m', t') and we check that m' is new and that $g(m') = t'$. If so, output 1, else 0. By assumption, if g is PRF then $\mathcal{D}$ outputs 1 with noticeable probability, and if g is uniform then $\mathcal{D}$ outputs 1 with probability $\leqslant 2^{-n}$. □

REMARK 6.11. PRF is a bit of an overkill. It suffices to use a family of "unpredictable" functions, which are functions that no adversary can guess the value at any new location. We essentially just showed that PRF implies unpredictability. One can also go backwards: take an unpredictable family $\{f_k\}$ and turn it into the family $\{g_{k,r}(x) = \langle r, f_k(x)\rangle\}$. This family is a PRF family using a proof similar to that of Goldreich-Levin.

6.2. Dealing with Long Messages

An important problem is dealing with long messages. We do not want to compute the PRF on the entire message. When encrypting we can just slice a long message into parts and encrypt each part individually.

So say we want a message in $\{0,1\}^N$ and a MAC in $\{0,1\}^n$.

Maybe we can chop the message into $m_1, m_2, \ldots, m_{N/n} \in \{0,1\}^n$ and output the tag as the concatenation $f_k(m_1)|\ldots|f_k(m_{N/n})$ where f_k is a PRF. But this is no good, because an attacker can output a valid tag of any permutation of the blocks.

Let us try something a bit more clever. Let us use a counter, say output instead the tag $f_k(1|m_1)|\ldots|f_k(\frac{N}{n}|m_{N/n})$, which takes care of the permutation attack. This also turns out to be not secure, because if the forger can just permute blocks between several message-tag pairs. This can be fixed (see the lecture notes of Dodis), but the disadvantage is that it creates even longer tags.

A cleaner solution is to take a PRF $\{0,1\}^n \to \{0,1\}^n$ and create another PRF $\{0,1\}^N \to \{0,1\}^n$. This is known as *domain extension*. The idea is to somehow choose a shrinking function $h : \{0,1\}^N \to \{0,1\}^n$ and use $f_k(h(\cdot))$. The advantage is that we keep a small tag size, and we are able to use trusted PRF constructions.

What we need is h for which it is hard to find collisions without knowing h. Definining what "hard to find collisions" means is somewhat delicate. Here are some candidates:

DEFINITION 6.12 (Collision-Resistant Hash Function). A family $\{h_k\}$ such that given k, no PPT can find collision in h_k, namely, find $x_1 \neq x_2$ such that $h_k(x_1) = h_k(x_2)$, with noticeable probability.

This is a strong primitive, and does not follow from OWF. For our application in fact this is too strong: there is no need for the adversary to know k. This leads to the notion of a "weak" CRHF:

DEFINITION 6.13 (Weak CRHF). A family $\{h_k\}$ such that given oracle access to h_k, no PPT can find collision in h_k, namely, find $x_1 \neq x_2$ such that $h_k(x_1) = h_k(x_2)$, with noticeable probability.

This is still too strong, since for extending the domain of PRF, the adversary doesn't even see $h_k(x)$. This leads to our weakest definition.

DEFINITION 6.14 (Almost Universal Hash Function). A family $\{h_k\}$ is called almost universal if for all $x \neq x' \in \{0,1\}^N$, $\Pr_k[h_k(x) = h_k(x')] \leqslant \text{negl}$.

THEOREM 6.15. *$\{f_{k'}(h_k(\cdot))\}_{k,k'}$ is a PRF, where $\{f_k\}$ is a PRF and $\{h_k\}$ is almost universal.*

In particular, this defines a MAC.

PROOF. First, since $\{f_{k'}\}$ is a PRF, $\{f_{k'}(h_k(\cdot))\}_{k,k'} \overset{c}{\approx} \{R(h_k(\cdot))\}_k$ where R is uniform $\{0,1\}^n \to \{0,1\}^n$. It suffices then to show that the latter is indistinguishable from a uniform function $\{Z : \{0,1\}^N \to \{0,1\}^n\}$.

Assume $\mathcal{D}$ is a distinguisher, and assumes that without loss of generality $\mathcal{D}$ never asks the same question twice. Then under Z it sees simly a sequence of iid uniform answers. Under $R(h_k(\cdot))$, it is the same assuming no collisions in h, namely all inputs are mapped to different outputs. The probability of collision is at most $\leqslant q^2 \operatorname{negl} = \operatorname{negl}$ where q is the number of queries (strictly speaking we are using that R, being a random function, hides all information about the output of h_k). □

The last thing to do is to construct an almost universal hash function.

CONSTRUCTION 6.16. Let $\mathbb{F}$ be the finite field with 2^n elements. View the message as $m = m_0, \ldots, m_{\ell-1}$ where $\ell = N/n$ and $m_i \in \mathbb{F}$. The family is $\{h_x\}_{x\in\mathbb{F}}$ given by

$$h_x(m_0, \ldots, m_{\ell-1}) = \sum_{i=0}^{\ell-1} m_i x^i$$

then for any $m \neq m'$ a collision is equivalent to $\sum_{i=0}^{\ell-1}(m_i - m'_i)x^i = 0$. Since a non-constant polynomial of degree $\ell - 1$ has at most $\ell - 1$ roots,

$$\Pr[h_x(m) = h_x(m')] \leqslant \frac{\ell - 1}{2^n} \leqslant \operatorname{negl}.$$

The reason this construction was so easy was because PRF is so strong. If we just used an unpredictable function, then we need something slightly stronger, say a (weak) CRHF.

6.3. Authenticated Encryption

Lecture 13, 8 Dec

This is basically SKE with the extra property that decryption fails unless it is sent by the original sender. It turns out that this is very strong.

The model is as in SKE: Gen, Enc, and Dec, but Dec can also output "$\perp$" meaning invalid input.

Security is defined as:

- IND-CPA (Secret Key)
- For all PPT $\mathcal{F}$, $\Pr_k[\mathcal{F}^{\mathrm{Enc}_k(\cdot)} \text{ forges}] \leqslant \operatorname{negl}$ where forging means outputting some c' such that $\mathrm{Dec}_k(c') \neq \perp$, and c' was not returned by the encryption oracle.

How do we construct Authenticated Encryption? Some possibilities:

- XOR cipher: $(r, f_k(r) \oplus m)$ where f_k is a PRF. This is obviously not authentic, since Dec does not check.
- g is a strong PRP, and $\mathrm{Enc}_k(m) = g_k(m|r)$; decryption inverts g_k and outputs the first half. Again, this is not authentic since there is no check. But if we modify it to also output r, $\mathrm{Enc}_k(m) = (r, g_k(m|r))$, then when decrypting we first can check that r matches the second half. This turns out to be a good Authenticated Encryption scheme.
- Since this is sort of SKE combined with MAC, let us try to combine them. "Encrypt-and-tag": Take IND-CPA SKE and strongly unforgeable MAC and output $(c \leftarrow \mathrm{SKE}.\mathrm{Enc}_{k_e}(m), t \leftarrow \mathrm{MAC}.\mathrm{Tag}_{k_a}(m))$. This is kind of authentic, but more importantly it is not IND-CPA secure: the MAC has no security and thus can reveal a lot about m.

- One way to fix the above is "Encrypt-then-tag", $(c \leftarrow \mathrm{SKE}.\mathrm{Enc}_{k_e}(m), t \leftarrow \mathrm{MAC}.\mathrm{Tag}_{k_a}(c))$. This is a good Authenticated Encryption scheme.

As it turns out Authenticated Encryption automatically satisfies IND-CCA2, in which an attacker is allowed access to the decryption oracle both before and after the challenge (but not on the encryption from the challenge).

THEOREM 6.17. *Any AE is also IND-CCA2 secure.*

The intuition is that the decryption oracle is useless: anything that was not output by the encryption oracle will result in $\perp$.

6.4. Digital Signatures

This is the public-key version of MAC. What we want is extremely natural: verification by anyone. This is what is used to verify the authenticity of a website, or an email.

The model is as follows:

- Gen outputs verification key $\mathfrak{vk}$, secret key $\mathfrak{sk}$.
- $\mathrm{Sign}_{\mathfrak{sk}}(m)$ outputs a signature σ
- $\mathrm{Ver}_{\mathfrak{vk}}(m, \sigma)$ just verifies if σ is a valid signature.

Correctness is defined as in a MAC, and security is UF-CMA as before: for all PPT $\mathcal{F}$, $\Pr[\mathcal{F}^{\mathrm{Sign}_{\mathfrak{sk}}(\cdot)}(\mathfrak{vk}) \text{ succeeds}] \leqslant \mathrm{negl}(n)$ where succeeds is as before: outputting (m', σ') such that $\mathrm{Ver}_{\mathfrak{vk}}(m', \sigma') =$ accept where m' is new.

Let us attempt to create a scheme. We can try to use a TDP, which we used to construct PKEs. This will not work, but intuitively we can think of this as authentication being the opposite of encryption:

- Gen outputs $\mathfrak{vk} = s$, $\mathfrak{sk} = t$ where (s, t) are chosen as in the TDP.
- $\mathrm{Sign}_t(m) = f_s^{-1}(m)$
- $\mathrm{Ver}_s(m, \sigma) = (f_s(\sigma) \stackrel{?}{=} m)$

This seems secure, until we realize that we can do the following: the attacker can pick some σ, compute $m = f_s(\sigma)$ and get a message-signature pair. We can have a weaker requirement: given a message, the attacker cannot sign it; under this the scheme is secure. But under our (existential unforgeability) definition, the attacker can still sign gibberish messages.

One way to fix this is to make sure that the message space is not dense. But another issue is that in our definitions we always what that it is hard to invert random m, but maybe for some messages it is easy. A third issue is that we provide the adversary with the signing oracle f_s^{-1}, but this is something that we never considered when defining TDPs.

Another attempt is using a One-Time Signature (Lamport '79). Let f be a OWF taking random strings in $\{0,1\}^n$. The secret key is a table of 2ℓ n-bit strings $x^{i,b}$, $i = 0, \ldots, \ell$ and $b = 0, 1$, and the verification key is a table $y^{i,b} = f(x^{i,b})$:

$x^{1,0}$	$x^{2,0}$	$\cdots$	$x^{\ell,0}$	$\xrightarrow{f}$	$y^{1,0}$	$y^{2,0}$	$\cdots$	$y^{\ell,0}$
$x^{1,1}$	$x^{2,1}$	$\cdots$	$x^{\ell,1}$		$y^{1,1}$	$y^{2,1}$	$\cdots$	$y^{\ell,1}$

The signer takes $m \in \{0,1\}^\ell$ and reveals x^{i,m_i} for $i = 1, \ldots, \ell$ and the verifier checks if $y^{i,m_i} = f(x^{i,m_i})$.

Security is proven by showing that if there is a forger, then the forger can be used to break OWF. The proof is more subtle, since we do not know what message the forger may want to sign. Obviously this only works for allowing the attacker to only see one signature, otherwise an attacker can mix and match.

But only being able to sign once is not very useful. The way to fix it is as follows: generate $\mathfrak{sk}_1, \mathfrak{vk}_1$ then when signing the first message m_1, generate another pair $\mathfrak{sk}_2, \mathfrak{vk}_2$ and sign $\mathfrak{vk}_2|m_1$ using $\mathfrak{sk}_1$, then when signing m_2 generate $\mathfrak{sk}_3, \mathfrak{vk}_3$ and sign $\mathfrak{vk}_3|m_2$ using $\mathfrak{sk}_2$, etc.

One problem is that a verifier needs to be stateful and remember the entire chain. So we really want to provide the entire chain at every step. It is not efficient, since signing ℓ bits takes $2\ell n$-bit long verification keys, but it's a start.

To deal with this recall that for MACs we could get verification keys of size that was independent of the message size. For MACs we could use Almost Universal Hash Functions. In this case, because the adversary needs to know the hash function in order to verify, we must use CRHFs. Note that a CRHF is a OWF.

By using a CRHF in Lamport's OTS, we get another OTS that can sign long messages with the same keysize. So this resolves the blowup issue in the chain idea.

Another improvement to the idea is to have a tree instead of a chain: instead of just signing one verification key at each level, we can sign two and then branch. As a result we can also just put the messages at the leaves. So we have logarithmic growth instead in the number of messages sent instead of linear growth. But we would still need to store the entire tree! So instead we can just use a PRF to compute the keys on demand.

The end after doing all of these optimizations is the following: using PRFs to store a "virtual" exponentially long random string, we can make the stateless while not being terribly inefficient.

The final problem is constructing a CRHF. The following is a candidate CRHF based on the Discrete Log assumption: take the family $h_{p,g,y} : \{0, \dots, p-2\} \times \{0,1\} \to \mathbb{Z}_p^*\}$ given by $h_{p,g,y}(x,b) = y^b g^x \bmod p$. Let us see what happens if we can come up with a collision $(x,b), (x',b')$ such that $y^b g^x \equiv y^{b'} g^{x'} \bmod p$. Note that $b \neq b'$, since $b = b' \implies x = x'$. So $yg^x \equiv g^{x'}$, that is $y = g^{x'-x}$ and we have solved the discrete log of y.