Introduction to Modern Cryptography

Luca Trevisan

luca@cs.columbia.edu
Computer Science Department, Columbia University

January 17, 2000

Forewords

This book contains a set of lecture notes that I wrote for an offering of a "Topics in Cryptography" course at Columbia University, in the Spring 1999.

It is not clear what use one can make of these notes. The main purpose was to integrate the treatment of the Goldwasser-Bellare lecture notes with some additional proofs, without getting into the level of rigor of Goldreich's book. I tried to use the concrete security framework as far as possible. This might as well be the first written concrete security presentation of the Goldwasser-Micali cryptosystem.

These notes would have not been possible without the help of Oded Goldreich, Salil Vadhan and Moti Yung, who answered a countless number of questions.

Luca Trevisan, New York, 1999

Contents

Chapter 1

Discrete Probability

The following notes cover, mostly without proofs, the basic notions and results of discrete probability. Two important topics (the *birthday paradox* and *universal hash functions*) are not covered, and they will be the subject of future notes.

1.1 Basic Definitions

In cryptography we typically want to prove that an adversary that tries to break a certain protocol has only minuscule (technically, we say "negligible") probability of succeeding. In order to prove such results, we need some formalism to talk about the probability that certain events happen, and also some techniques to make computations about such probabilities.

In order to model a probabilistic system we are interested in, we have to define a *sample space* and a *probability distribution*. The sample space is the set of all possible *elementary events*, i.e. things that can happen. A probability distribution is a function that assigns a non-negative number to each elementary event, this number being the *probability* that the event happen. We want probabilities to sum to 1. Formally,

Definition 1.1 *For a finite sample space* Ω *and a function* $\mathbf{Pr} : \Omega \to \mathbf{R}$*, we say that* $\mathbf{Pr}$ *is a* probability distribution *if*

1. $\mathbf{Pr}(a) \geq 0$ *for every* $a \in \Omega$*;*
2. $\sum_{a \in \Omega} \mathbf{Pr}(a) = 1$.

For example, if we want to model a sequence of three coin flips, our sample space will be $\{Head, Tail\}^3$ (or, equivalently, $\{0,1\}^3$) and the probability distribution will assign $1/8$ to each element of the sample space (since each outcome is equally likely).

If we model an algorithm that first chooses at random a number in the range $1, \ldots, 10^{200}$ and then does some computation, our sample space will be the set $\{1, 2, \ldots, 10^{200}\}$, and each element of the sample space will have probability $1/10^{200}$.

We will always restrict ourselves to *finite* sample spaces, so we will not remark it each time. *Discrete probability* is the restriction of probability theory to finite sample spaces. Things are much more complicated when the sample space can be infinite.

An *event* is a subset $A \subseteq \Omega$ of the sample space. The probability of an event is defined in the intuitive way

$$\mathbf{Pr}[A] = \sum_{a \in A} \mathbf{Pr}(a)$$

(Conventially, we set $\mathbf{Pr}[\emptyset] = 0$.)

We use square brackets to remind us that now we are considering a different function: while $\mathbf{Pr}(\cdot)$ is a function whose inputs are *elements* of the sample space, $\mathbf{Pr}[\cdot]$ is a function whose inputs are *subsets* of the sample space.

For example, suppose that we want to ask what is the probability that, when flipping three coins, we get two heads. Then $\Omega = \{0,1\}^3$, $\mathbf{Pr}(a) = 1/8$ for every $a \in \Omega$, we define A as the subset of $\{0,1\}^3$ containing strings with exactly two 1s, and we ask what is $\mathbf{Pr}[A]$. As it turns out, A has 3 elements, that is $011, 101, 110$, and so $\mathbf{Pr}[A] = 3/8$. Very often, as in this example, computing the probability of an event reduces to counting the number of elements of a set.

When $\mathbf{Pr}(\cdot)$ assigns the same value $1/|\Omega|$ to all the elements of the sample space, then it is called the *uniform distribution over* Ω.

The following distribution arises very often, and it is good to know about it. Consider the situation where you flip n biased coins, and the probability that each coin turns out head is p (where $0 \leq p \leq 1$ is some fixed parameter). The outcome of each flip is independent of all the other outcomes. Then the sample space is $\Omega = \{0,1\}^n$, identifying heads with 1s; the probability distribution is

$$\mathbf{Pr}(a) = p^k(1-p)^{n-k} \text{ where } k \text{ is the number of 1s in } a$$

When $p = 1/2$ then we have the uniform distribution over $\{0,1\}^n$. If $p = 0$ then all a have probability zero, except $00\cdots0$, which has probability one. (Similarly if $p = 1$.) The other cases are more interesting.

These distributions are called *Bernoulli* distributions or *binomial* distributions.

If we have a binomial distribution with parameter p, and we ask what is the probability of the event A_k that we get a string with k ones, then such a probability is

$$\mathbf{Pr}[A_k] = \binom{n}{k} p^k (1-p)^{n-k}$$

1.2 Random Variables and Expectation

Very often, when studying a probabilistic system (say, a randomized algorithm) we are interested in some values that depend on the elementary event that takes place. For example, when we play dice, we are interested in the probabilistic system where two dice are rolled, and the sample space is $\{1,2,\ldots,6\}^2$, with the uniform distribution over the 36 elements of the sample space, and we are interested in the *sum* of the outcomes of the two dice. Similarly, when we study a randomized algorithm that makes some internal random choices, we are interested in the *running time* of the algorithm, or in its *output*. The notion of a *random variable* gives a tool to formalize questions of this kind.

A *random variable* X is a function $X : \Omega \to V$ where Ω is a sample space and V is some arbitrary set (V is called the *range* of the random variable). One should think of a random variable as an algorithm that on input an elementary event returns some output. Typically, V will either be a subset of the set of real numbers or of the set of binary strings of a certain length.

Let Ω be a sample space, $\mathbf{Pr}$ a probability distribution on Ω and X be a random variable on Ω. If v is in the range of X, then the expression $X = v$ denotes an event, namely the event $\{a \in \Omega : X(a) = v\}$, and thus the expression $\mathbf{Pr}[X = v]$ is well defined, and it is something interesting to try to compute.

Let's look at the example of dice. In that case, $\Omega = \{1, \ldots, 6\}^2$, for every $(a, b) \in \Omega$ we have $\mathbf{Pr}(a, b) = 1/36$. Let us define X as the random variable that associates $a + b$ to an elementary event (a, b). Then the range of X is $\{2, 3, \ldots, 12\}$. For every element of the range we can compute the probability that X take such value. By counting the number of elementary events in each event we get

$$\mathbf{Pr}[X = 2] = 1/36 \ , \ \mathbf{Pr}[X = 3] = 2/36 \ , \ \mathbf{Pr}[X = 4] = 3/36$$

$$\mathbf{Pr}[X = 5] = 4/36 \ , \ \mathbf{Pr}[X = 6] = 5/36 \ , \ \mathbf{Pr}[X = 7] = 6/36$$

and the other probabilities can be computed by observing that

$$\mathbf{Pr}[X = v] = \mathbf{Pr}[X = 14 - v]$$

It is possible to define more than one random variable over the same sample space, and consider expressions more complicated than equalities.

When the range of a random variable X is a subset of the real numbers (e.g. if X is the running time of an algorithm — in which case the range is even a subset of the integers) then we can define the *expectation* of X. The expectation of a random variable is a number defined as follows.

$$\mathbf{E}[X] = \sum_{v \in V} v \mathbf{Pr}[X = v]$$

where V is the range of X. We can assume without loss of generality that V is finite, so that the expression above is well defined (if it were an infinite series, it could diverge or even be undefined).

Expectations can be understood in terms of betting. Say that I am playing some game where I have a probability 2/3 of winning, a probability 1/6 of losing and a probability 1/6 of a draw. If I win, I win \$ 1; if I lose I lose \$ 2; if there is a draw I do not win or lose anything. We can model this situation by having a sample space $\{L, D, W\}$ with probabilities defined as above, and a random variable X that specifies my wins/losses. Specifically $X(L) = -2$, $X(D) = 0$ and $X(W) = 1$. The expectation of X is

$$\mathbf{E}[X] = \frac{1}{6} \cdot (-2) + \frac{1}{6} \cdot 0 + \frac{2}{3} \cdot 1 = \frac{1}{3}$$

so if I play this game I "expect" to win \$ 1/3. The game is more than fair on my side.

When we analyze a randomized algorithm, the running time of the algorithm typically depends on its internal random choices. A complete analysis of the algorithm would be a specification of the running time of the algorithm for *each* possible sequence of internal choices. This is clearly impractical. If we can at least analyse the *expected* running time of the algorithm, then this will be just a single value, and it will give useful information about the typical behavior of the algorithm (see Section 1.4 below).

Here is a very useful property of expectation.

Theorem 1.2 (Linearity of Expectation) *Let X be a random variable and a be real; then $\mathbf{E}[aX] = a\mathbf{E}[X]$. Let $X_1, \ldots, X_n$ be random variables over the same sample space; then $\mathbf{E}[X_1 + \cdots + X_n] = \mathbf{E}[X_1] + \cdots \mathbf{E}[X_n]$.*

Example 1 Consider the following question: if we flip a coin n times, what is the expected number of heads? If we try to answer this question without using the linearity of expectation we have to do a lot of work. Define $\Omega = \{0,1\}^n$ and let $\mathbf{Pr}$ be the uniform distribution; let X be the random variable such that $X(a) =$ the number of 1s in $a \in \Omega$. Then we have, as a special case of Bernoulli distribution, that

$$\mathbf{Pr}[X = k] = \binom{n}{k} 2^{-n}$$

In order to compute the average of X, we have to compute the sum

$$\sum_{k=0}^{n} \binom{n}{k} k 2^{-n} \tag{1.1}$$

which requires quite a bit of ingenuity. We now show how to solve Expression (1.1) just to see how much work can be saved by using the linearity of expectation. An inspection of Expression (1.1) shows that it looks a bit like the expressions that one gets out of the Binomial Theorem, except for the presence of k. In fact it looks pretty much like the *derivative* of an expression coming from the Binomial Theorem (this is a standard trick). Consider $(1/2 + x)^n$ (we have in mind to substitute $x = 1/2$ at some later point), then we have

$$\left(\frac{1}{2} + x\right)^n = \sum_{k=0}^{n} \binom{n}{k} 2^{-(n-k)} x^k$$

and then

$$\frac{d((1/2+x)^n)}{dx} = \sum_{k=0}^{n} \binom{n}{k} 2^{-(n-k)} k x^{k-1}$$

but also

$$\frac{d((1/2+x)^n)}{dx} = n\left(\frac{1}{2} + x\right)^{n-1}$$

and putting together

$$\sum_{k=0}^{n} \binom{n}{k} 2^{-(n-k)} k x^{k-1} = n\left(\frac{1}{2} + x\right)^{n-1} .$$

Now we substitute $x = 1/2$, and we have

$$\sum_{k=0}^{n} \binom{n}{k} 2^{-(n-k)} k 2^{-(k-1)} = n .$$

Here we are: dividing by 2 we get

$$\sum_{k=0}^{n} \binom{n}{k} k 2^{-n} = \frac{n}{2} .$$

So much for the definition of average. Here is a better route: we can view X as the sum of n random variables $X_1, \ldots, X_n$, where X_i is 1 if the i-th coin flip is 1 and X_i is 0 otherwise. Clearly, for every i, $\mathbf{E}[X_i] = \frac{1}{2} \cdot 0 + \frac{1}{2} \cdot 1 = \frac{1}{2}$, and so

$$\mathbf{E}[X] = \mathbf{E}[X_1 + \cdots + X_n] = \mathbf{E}[X_1] + \cdots + \mathbf{E}[X_n] = \frac{n}{2} .$$

■

1.3 Independence

1.3.1 Conditioning and Mutual Independence

Suppose I toss two coins, without letting you see the outcome, and I tell you that at least one of the coins came up heads, what is the probability that both coin are heads?

In order to answer to this question (I will give it away that the answer is 1/3), one needs some tools to reason about the probability that a certain event holds *given* (or *conditioned* on the fact) that a certain other event holds.

Fix a sample space Ω and a probability distribution $\mathbf{Pr}$. Suppose we are given that a certain event $A \subseteq \Omega$ holds. Then the probability of an elementary event a given the fact that A holds (written $\mathbf{Pr}(a|A)$) is defined as follows: if $a \notin A$, then it is impossible that a holds, and so $\mathbf{Pr}(a|A) = 0$; otherwise, if $a \in A$, then $\mathbf{Pr}(a|A)$ has a value that is proportional to $\mathbf{Pr}(a)$. One realizes that the factor of proportionality has to be $1/\mathbf{Pr}[A]$, so that probabilities sum to 1 again. Our definition of conditional probability of an elementary event is then

$$\mathbf{Pr}(a|A) = \begin{cases} 0 & \text{If } a \notin A \\ \frac{\mathbf{Pr}(a)}{\mathbf{Pr}[A]} & \text{Otherwise} \end{cases}$$

The above formula already lets us solve the question asked at the beginning of this section. Notice that probabilities conditioned on an event A such that $\mathbf{Pr}[A] = 0$ are undefined.

Then we extend the definition to arbitrary events, and we say that for an event B

$$\mathbf{Pr}[B|A] = \sum_{b \in B} \mathbf{Pr}(b|A)$$

One should check that the following (more standard) definition is equivalent

$$\mathbf{Pr}[B|A] = \frac{\mathbf{Pr}[A \cap B]}{\mathbf{Pr}[A]}$$

Definition 1.3 *Two events A and B are independent if*

$$\mathbf{Pr}[A \cap B] = \mathbf{Pr}[A] \cdot \mathbf{Pr}[B]$$

If A and B are independent, and $\mathbf{Pr}[A] > 0$, then we have $\mathbf{Pr}[B|A] = \mathbf{Pr}[B]$. Similarly, if A and B are independent, and $\mathbf{Pr}[B] > 0$, then we have $\mathbf{Pr}[A|B] = \mathbf{Pr}[A]$. This motivates the use of the term "independence." If A and B are independent, then whether A holds or not is not influenced by the knowledge that B holds or not.

When we have several events, we can define a generalized notion of independence.

Definition 1.4 *Let $A_1, \ldots, A_n \subseteq \Omega$ be events is a sample space Ω; we say that such events are* mutually *independent if for every subset of indices $I \subseteq \{1, \ldots n\}$, $I \neq \emptyset$, we have*

$$\mathbf{Pr}[\bigcap_{i \in I} A_i] = \prod_{i \in I} \mathbf{Pr}[A_i]$$

All this stuff was just to prepare for the definition of independence for random variables, which is a very important and useful notion.

Definition 1.5 *If X and Y are random variables over the same sample space, then we say that X and Y are independent if for any two values v, w, the event $(X = v)$ and $(Y = w)$ are independent.*

Therefore, if X and Y are independent, knowing the value of X, no matter which value it is, does not tell us anything about the distribution of Y (and vice versa).

Theorem 1.6 *If X and Y are independent, then $\mathbf{E}[XY] = \mathbf{E}[X]\mathbf{E}[Y]$.*

This generalizes to several random variables

Definition 1.7 *Let $X_1, \ldots, X_n$ be random variables over the same sample space, then we say that they are* mutually *independent if for any sequence of values $v_1, \ldots, v_n$, the events $(X_1 = v_1)$, $\ldots$, $(X_n = v_n)$ are mutually independent.*

Theorem 1.8 *If $X_1, \ldots, X_n$ are mutually independent random variables, then*

$$\mathbf{E}[X_1 \cdot X_2 \cdots X_n] = \mathbf{E}[X_1] \cdot \mathbf{E}[X_2] \cdots \mathbf{E}[X_n]$$

1.3.2 Pairwise Independence

It is also possible to define a weaker notion of independence.

Definition 1.9 *Let $X_1, \ldots, X_n$ be random variables over the same sample space, then we say that they are* pairwise *independent if for every $i, j \in \{1, \ldots, n\}$, $i \neq j$, we have that X_i and X_j are independent.*

It is important to note that a collection of random variables can be pairwise independent without being mutually independent. (But a collection of mutually independent random variables is always pairwise independent for a stronger reason.)

Example 2 Consider the following probabilistic system: we toss 2 coins, and we let the random vaiables X, Y, Z be, respectively, the outcome of the first coin, the outcome of the second coin, and the XOR of the outcomes of the two coins (as usual, we interpret outcomes of coins as 0/1 values). Then X, Y, Z are not mutually independent, for example

$$\mathbf{Pr}[Z = 0 | X = 0, Y = 0] = 1$$

while

$$\mathbf{Pr}[Z = 0] = 1/2$$

in fact, intuitively, since the value of Z is *totally determined* by the values of X and Y, the three variables cannot be mutually independent. On the other hand, we will now show that X, Y, Z are pairwise independent. By definition, X and Y are independent, so we have to focus of X and Z and on Y and Z. Let us prove that X and Z are independent (the proof for Y and Z is identical). We have to show that for each choice of two values $v, w \in \{0, 1\}$, we have

$$\mathbf{Pr}[X = v, Z = w] = \mathbf{Pr}[X = v]\mathbf{Pr}[Z = w] = \frac{1}{4}$$

and this is true, since, in order to have $Z = w$ and $X = v$, we must have $Y = w \oplus v$, and the event that $X = v$ and $Y = w \oplus v$ happens with probability 1/4. ■

Let us see two additional, more involved, examples.

Example 3 Suppose we flip k coins, whose outcomes be $a_1, \ldots, a_k \in \{0,1\}^k$. Then for every non-empty subset $I \subseteq \{0,1\}^k$ we define a random variable X_I, whose value is $\bigoplus_{i \in I} a_i$. It is possible to show that $\{X_I\}_{I \subseteq \{0,1\}^k, I \neq \emptyset}$ is a pairwise independent collection of random variables. Notice that we have $2^k - 1$ random variables defined over a sample space of only 2^k points. ■

Example 4 Let p be a prime number; suppose we pick at random two elements $a, b \in \mathbf{Z}_p$ — that is, our sample space is the set of pairs $(a,b) \in \mathbf{Z}_p \times \mathbf{Z}_p = \Omega$, and we consider the uniform distribution over this sample space. For every $z \in \mathbf{Z}_p$, we define one random variable X_z whose value is $az + b \pmod p$. Thus we have a collection of p random variables. It is possible to show that such random variables are pairwise independent. ■

1.4 Deviation from the Expectation

1.4.1 Markov's Inequality

Say that X is the random variable expressing the running time in seconds of an algorithm on inputs of a certain size, and that we computed $\mathbf{E}[X] = 10$. Since this is the order of magnitude of the time that we expect to spend while running the algorithm, it would be devastating if it happened that, say, $X \geq 1,000,000$ (i.e. more than 11 days) with large probability. However, we quickly realize that if $\mathbf{E}[X] = 10$, then it must be $\mathbf{Pr}[X \geq 1,000,000] \leq 1/100,000$, as otherwise the contribution to the expectation of the only events where $X \geq 1,000,000$ would already exceed the value 10. This reasoning can be generalized as follows.

Theorem 1.10 (Markov's Inequality) *If X is a non-negative random variable then*

$$\mathbf{Pr}[X \geq k] \leq \frac{\mathbf{E}[X]}{k}$$

Sometimes the bound given by Markov's inequality are extremely bad, but the bound is as strong as possible if the only information that we have is the expectation of X.

For example, suppose that X counts the number of heads in a sequence of n coin flips. Formally, Ω is $\{0,1\}^n$ with the uniform distribution, and X is the number of ones in the string. Then $\mathbf{E}[X] = n/2$. Suppose we want to get an upper bound on $\mathbf{Pr}[X \geq n]$ using Markov. Then we get

$$\mathbf{Pr}[X \geq n] \leq \frac{\mathbf{E}[X]}{n} = \frac{1}{2}$$

This is ridiculous! The right value is 2^{-n}, and the upper bound given by Markov's inequality is totally off, and it does not even depend on n.

However, consider now the experiment where we flip n coins that are *glued* together, so that the only possible outcomes are n heads (with probability $1/2$) and n tails (with probability $1/2$). Define X again as the number of heads. We still have that $\mathbf{E}[X] = n/2$, and we can apply Markov's inequality as before to get

$$\mathbf{Pr}[X \geq n] \leq \frac{\mathbf{E}[X]}{n} = \frac{1}{2}$$

But, now, the above inequality is tight, becuase $\mathbf{Pr}[X \geq n]$ is precisely $1/2$.

The moral is that Markov's inequality is very useful because it applies to every non-negative random variables having a certain expectation, so we can use it without having to study our random variable too much. On the other hand, the inequality will be accurate when applied to a random variable that typically deviates a lot from its expectation (say, the number of heads that we get when we toss n glued coins) and the inequality will be very bad when we apply it to a random variable that is concentrated around its expectation (say, the number of heads that we get in n independent coin tosses). In the latter case, if we want accurate estimations we have to use more powerful methods. One such method is described below.

1.4.2 Variance

For a random variable X, the random variable

$$X' = |X - \mathbf{E}[X]|$$

gives all the information that we need in order to decide whether X is likely to deviate a lot from its expectation or not. All we need to do is to prove that X' is typically small. However this idea does not lead us very far (analysing X' does not seem to be any easier than analysing X).

Here is a better tool. Consider

$$(X - \mathbf{E}[X])^2$$

This is again a random variable that tells us how much X deviates from its expectation. In particular, if the *expectation* of such an auxliary random variable is small, then we expect X to be typically close to its expectation. The *variance* of X is defined as

$$\mathbf{Var}(X) = \mathbf{E}[(X - \mathbf{E}[X])^2]$$

Here is an equivalent epxression (we use linearity of expectation in the derivation of the final result):

$$\begin{aligned}
\mathbf{Var}(X) &= \mathbf{E}[(X - \mathbf{E}[X])^2] \\
&= \mathbf{E}[X^2 - 2X\mathbf{E}[X] + (\mathbf{E}[X])^2] \\
&= \mathbf{E}[X^2] - 2\mathbf{E}[X\mathbf{E}[X]] + (\mathbf{E}[X])^2 \\
&= \mathbf{E}[X^2] - 2\mathbf{E}[X]\mathbf{E}[X] + (\mathbf{E}[X])^2 \\
&= \mathbf{E}[X^2] - (\mathbf{E}[X])^2
\end{aligned}$$

The variance is a useful notion for two reasons: it is often easy to compute and it gives rise to sometimes strong estimations on the probability that a random variable deviates from its expectation.

Theorem 1.11 (Chebyshev's Inequality)

$$\mathbf{Pr}[|X - \mathbf{E}[X]| \geq k] \leq \frac{\mathbf{Var}(X)}{k^2}$$

The proof uses Markov's inequality and a bit of ingenuity.

$$\begin{aligned}\mathbf{Pr}[|X - \mathbf{E}[X]| \geq k] &= \mathbf{Pr}[(X - \mathbf{E}[X])^2 \geq k^2] \\ &\leq \frac{\mathbf{E}[(X - \mathbf{E}[X])^2]}{k^2} \\ &= \frac{\mathbf{Var}(X)}{k^2}\end{aligned}$$

The nice idea is in the first step. The second step is just an application of Markov's inequality and the last step uses the definition of variance.

The value $\sigma(X) = \sqrt{\mathbf{Var}(X)}$ is called the *standard deviation* of X. One expects the value of a random variable X to be around the interval $\mathbf{E}[X] \pm \sigma(X)$. We can restate Chebyshev's Inequality in terms of standard deviation

Theorem 1.12 (Chebyshev's Inequality, Alternative Form)

$$\mathbf{Pr}[|X - \mathbf{E}[X]| \geq c \cdot \sigma(X)] \leq \frac{1}{c^2}$$

Let Y be a random variable that is equal to 0 with probability 1/2 and to 1 with probability 1/2. Then $\mathbf{E}[Y] = 1/2$, $Y = Y^2$, and

$$\mathbf{Var}(Y) = \mathbf{E}[Y^2] - (\mathbf{E}[Y])^2 = \frac{1}{2} - \frac{1}{4} = \frac{1}{4}$$

Let X the random variable that counts the number of heads in a sequence of n independent coin flips. We have seen that $\mathbf{E}[X] = n/2$. Computing the variance according to the definition would be painful. We are fortunate that the following result holds.

Lemma 1.13 (Tools to Compute Variance)

1. *Let X be a random variable, a, b be reals, then*

$$\mathbf{Var}(aX + b) = a^2\mathbf{Var}(X)$$

2. *Let $X_1, \ldots, X_n$ be* pairwise *independent random variables on the same sample space. Then*

$$\mathbf{Var}(X_1 + \cdots + X_n) = \mathbf{Var}(X_1) + \cdots + \mathbf{Var}(X_n)$$

Then we can view X as $X_1 + \cdots + X_n$ where X_i are mutually independent random variables such that for each i X_i takes value 1 with probability 1/2 and value 0 with probability 1/2. As computed before, $\mathbf{Var}(X_i) = 1/4$. Therefore $\mathbf{Var}(X) = n/4$ and the standard deviation is $\sqrt{n}/2$. This means that when we flip n coins we expect to get about $n \pm \sqrt{n}$ heads.

Let us test Chebyshev's inequality on the same example of the previous subsection. Let X be a random variable defined over $\Omega = \{0, 1\}^n$, where $\mathbf{Pr}$ is uniform, and X counts the number of 1s in the elementary event: suppose we want to compute $\mathbf{Pr}[X \geq n]$. As computed above, $\mathbf{Var}(X) = n/4$, so

$$\mathbf{Pr}[X \geq n] \leq \mathbf{Pr}[|X - \mathbf{E}[X]| \geq n/2] \leq \frac{1}{n}$$

This is still much less than the correct value 2^{-n}, but at least it is a value that decreases with n. It is also possible to show that Chebyshev's inequality is as strong as possible given its assumption.

Let $n = 2^k - 1$ for some integer k and let $X_1, \ldots, X_n$ be the collection of pairwise independent random variables as defined in Example 3. Let $X = X_1 + \ldots + X_n$. Suppose we want to compute $\mathbf{Pr}[X = 0]$. Since each X_i has variance $1/4$, we have that X has variance $n/4$, and so

$$\mathbf{Pr}[X = 0] \leq \mathbf{Pr}[|X - \mathbf{E}[X]| \geq n/2] \leq \frac{1}{n}$$

which is almost the right value: the right value is $2^{-k} = 1/(n+1)$.

1.5 Additional Material

1.5.1 Some Combinatorial Facts

Consider a set Ω with n elements. Ω has 2^n subsets (including the empty set and Ω itself).

For every $0 \leq k \leq n$, Ω has $\binom{n}{k}$ subsets of k elements. The symbol $\binom{n}{k}$ is read "n choose k" and is defined as

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

Then we must have

$$\sum_{k=0}^{n} \binom{n}{k} = 2^n \tag{1.2}$$

which is a special case of the following result

Theorem 1.14 (Binomial Theorem) *For every two reals a, b and non-negative integer n,*

$$(a+b)^n = \sum_{k=0}^{n} \binom{n}{k} a^k b^{n-k}$$

We can see that Equation (1.2) follows from the Binomial Theorem by simply substituting $a = 1$ and $b = 1$.

Sometimes we have to deal with summations of the form $1 + 1/2 + 1/3 + \ldots + 1/n$. It's good to know that $\sum_{k=1}^{n} 1/k \approx \ln n$. More precisely

Theorem 1.15 $\lim_{n\to\infty} \frac{\sum_{k=1}^{n} 1/k}{\ln n} = 1$.

In particular, $\sum_{k=1}^{n} 1/k \leq 1 + \ln n$ for every n, and $\sum_{k=1}^{n} 1/k \geq \ln n$ for sufficiently large n.

The following inequality is exceedingly useful in computing upper bounds of probabilities of events:

$$1 + x \leq e^x \tag{1.3}$$

This is easy to prove by looking at the Taylor series of e^x:

$$e^x = 1 + x + \frac{1}{2}x^2 + \ldots + \frac{1}{k!}x^k + \ldots$$

Observe that Equation (1.3) is true for *every* real x, not necessarily positive (but it becomes trivial for $x < -1$).

Here is a typical application of Equation (1.3). We have a randomized algorithm that has a probability ϵ over its internal coin tosses of succeeding in doing something (and when it succeeds, we notice that it does, say because the algorithm is trying to invert a one-way function, and when it succeeds then we can check it efficiently); how many times do we have to run the algorithm before we have probability at least 3/4 that the algorithm succeeds?

The probability that it never succeeds in k runs is

$$(1-\epsilon)^k \leq e^{-\epsilon k}$$

If we choose $k = 2/\epsilon$, the probability of k consecutive failures is less than $e^{-2} < 1/4$, and so the probability of succeeding (at least once) is at least 3/4.

1.5.2 Examples of Analysis of Error Probability of Algorithms

Example 5 Suppose that we have an algorithm whose worst-case running time (on inputs of a certain length) is bounded by a random variable T (whose sample space is the set of random choices made by the algorithm). For concreteness, suppose that we are considering the randomized algorithm that given a prime p and an element $a \in \mathbf{Z}^*_p$ decides whether a is a quadratic residue or not. Suppose that we are given $t = \mathbf{E}[T]$ but no additional information on the algorithm, and we would like to know how much time we have to wait in oder to have a probability at least $1 - 10^{-6}$ that the algorithm terminates. If we only know $\mathbf{E}[T]$, then we can just use Markov's inequality and say that

$$\mathbf{Pr}[T \geq kt] \leq \frac{1}{k}$$

and if we choose $k = 10^6$ we have that

$$\mathbf{Pr}[T \geq 10^6 t] \leq 10^{-6} \ .$$

However there is a much faster way of guaranteeing termination with high probability. We let the program run for $2t$ time. There is a probability 1/2 that the algorithm will stop before that time. If so we are happy. If not, we terminate the computation, and start it over (in the second iteration, we let the algorithm use independent random bits). If the second computation does not terminate within $2t$ time, we reset it once more, and so on.[1] Let T' be the random variable that gives the time taken by this new version of the algorithm (with the stop and reset actions). Now we have that the probability that we use more than $2kt$ time is equal to the probability that for k consecutive (independent) times the algorithm takes more than $2t$ time. Each of these events happen with probability at most 1/2, and so

$$\mathbf{Pr}[T' \geq 2kt] \leq 2^{-k}$$

and if take $k = 20$, the probability is less than 10^{-6}, and the time is only $40t$ rather than $1,000,000t$.

Suppose that $t = t(n)$ is the average running time of our algorithm on inputs of length n, and that we want to find another algorithm that finishes always in time $t'(n)$ and that reports a failure

[1] This is reminescent of the way one works with Windows'98.

only with negligible probability, say with probability at most $n^{-\log n}$. How large do we have choose t', and what the new algorithm should be like?

If we just put a timeout t' on the original algorithm, then we can use Markov's inequality to say that $t'(n) = n^{\log n} t(n)$ will suffice, but now t' is not polynomial in n (even if t was). Using the second method, we can put a timeout $2t$ and repeat the algorithm $(\log n)^2$ times. Then the failure probability will be as requested and $t'(n) = 2(\log n)^2 t(n)$. ■

If we know how the algorithm works, then we can make a more direct analysis.

Example 6 Suppose that our goal is, given n, to find a number $2 \leq a \leq n-1$ such that $\gcd(a, n) = 1$. To simplify notation, let $l = ||n|| \approx \log n$ be the number of digits of n in binary notation (in a concrete application, l would be a few hundreds). Our algorithm will be as follows:

- Repeat no more than k times:
  1. Pick uniformly at random $a \in \{2, \ldots, n-1\}$;
  2. Use Euclid's algorithm to test whether $\gcd(a, n) = 1$.
  3. If $\gcd(a, n) = 1$ then output a and halt.
- Output "failure".

We would like to find a value of k such that the probability that the algorithm reports a failure is negligible in the size of the input (i.e. in l).

At each iteration, the probability that algorithm finds an element that is coprime with n is

$$\frac{\phi(n)}{n-2} \geq \frac{1}{6 \log \log n} = \frac{1}{6 \log l}$$

So the probability that there is a failure in one iteration is at most

$$\left(1 - \frac{1}{6 \log l}\right)$$

and the probability of k consecutive independent failures is at most

$$\left(1 - \frac{1}{6 \log l}\right)^k \leq e^{-k/6 \log l}$$

if we set $k = (\log l)^3$ then the probability of k consecutive failures is at most

$$e^{-(\log l)^3/6 \log l} = l^{-(\log l)/6}$$

that is negligible in l.

■

Chapter 2

Algebra

This chapter contains as little theory as possible, and most results are stated without proof. Any introductory book on algebra will contain proofs and put the results in a more general, and more beautiful framework.

For example, a book by Childs [**?**] covers all the required material without getting too abstract. It also points out the cryptographic applications.

2.1 Prime Numbers

By *integer*, we mean a positive or negative integer. We denote by $\mathbf{Z}$ the set $\mathbf{Z} = \{\ldots, -3, -2, -1, 0, 1, 2, 3, \ldots\}$. A *natural* number is a non-negative integer. We denote by $\mathbf{N}$ the set $\mathbf{N} = \{0, 1, 2, 3 \ldots\}$. We also denote by $\mathbf{Z}^+$ the set $\mathbf{Z}^+ = \{1, 2, 3, \ldots\}$ of positive integers.

For an integer n, we denote by $||n||$ the *length* of n, i.e. the number of bits needed to represent it, i.e. $||n|| = \lceil \log_2 n \rceil$. Logarithms will always be to the base 2, so we will omit the base hereafter. We will denote by $\ln n$ the natural logarithm of n, i.e. the logarithm taken to the base $e = 2.71828\ldots$

For integers k, n, we say that k *divides* n (or that k is *divisor* of n) if n is a multiple of k. For example 5 divides 35. We write $k|n$ when k divides n.

A *prime number* is a positive integer $p \geq 2$ whose only divisors are 1 and p. Notice that 2 is the only even prime number.

The first few prime numbers are $2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, \ldots$.

When a number is not prime, it is called *composite*. A composite can always be written (in a unique way) as a product of primes, possibly with repetitions. E.g. $300 = 2 \times 2 \times 5 \times 5$.

There are infinitely many prime numbers (which is very easy to prove), and in fact there are quite a lot of them (which is harder to prove). Specifically, if we define $\pi(n)$ to be the number of prime numbers p such that $2 \leq p \leq n$, then $\pi(n)$ is about $n/\ln n$. Formally

Theorem 2.1 (Prime Numbers Theorem) $\lim_{n\to\infty} \frac{\pi(n)}{n/\ln n} = 1$

The following bounds are also known

$$\pi(n) \geq \frac{n}{\ln n}$$

and, for $n \geq 17$,

$$\pi(n) \leq 1.10555 \frac{n}{\ln n}$$

There is an efficient randomized algorithm that on input an integer tests whether it is prime or not. Therefore if we want to generate a large prime (in the interval from 1 to n, where n can be thought of as a number around 10^{200}) we can just pick a random number in the set $\{1, \ldots, n = 10^{200}\}$ and then test whether it is prime. If it is not, we try again. Each time we have a probability $\approx 1/\ln n \approx 1/460$ of succeeding, so we expect to succeed after less than 500 attempts. Big prime numbers are very important in applied cryptography, and the Prime Number Theorem is a very useful tools to analyze certain cryptographic protocols.

The Prime Number Theorem has exceedingly difficult proofs, but it is easy to prove at least that there are infinitely many primes. Suppose, by contradiction, that there are only finitely many primes, and let them be $p_1, \ldots, p_n$. Consider the number $m = p_1 \cdot p_2 \cdots p_n + 1$. Since m is bigger than any prime, it must be composite, and hence it must be divisible by some prime. We note that m is not divisible by p_1, as when we divide m by p_1 we get the quotient $p_2 \cdots p_n$ and the remainder 1. Similarly, m is not divisible by p_2, neither by p_3, ..., neither by p_n. So we get a contradiction.

2.2 Modular Arithmetic

Let a, n be integers ($n \geq 2$). If we try to divide a by n using the grade-school algorithm we end up with two numbers q and r (the quotient and the remainder) such that $aq + r = n$ and $0 \leq r \leq n - 1$. For example, if we divide 15 by 7 we get a quotient 2 and a remainder 1 and the equation $2 \cdot 7 + 1 = 15$. Such numbers q and r are unique. For integers a, b, n we write

$$a = b \pmod{n}$$

if a and b have the same remainder when divided by n (equivalently, if $a - b$ is a multiple of n). For example $15 = 8 \pmod{7}$.

For a fixed integer n, the relation $\cdot = \cdot \pmod{n}$ has several properties of ordinary equality. For example,

- For every $a \in \mathbf{Z}$, $a = a \pmod{n}$;
- For every $a, b, c \in \mathbf{Z}$, if $a = b \pmod{n}$ and $b = c \pmod{n}$, then $a = c \pmod{n}$;
- For every $a, a', b, b' \in \mathbf{Z}$, if $a = a' \pmod{n}$ and $b = b' \pmod{n}$ then $a + b = a' + b'$;
- For every $a, a', k, k' \in \mathbf{Z}$, if $a = a' \pmod{n}$ and $k = k' \pmod{n}$, then $ak = a'k' \pmod{n}$.

The last two properties imply that when we do arithmetic operations modulo n, then we obtain the same result if we replace one term by another one that is equal modulo n. In particular, it is the same if every term a is replaced by the remainder of its division by n. So, when doing operations modulo n, we can restrict ourselves to use only the integers $0, \ldots, n - 1$.

We denote by $\mathbf{Z}_n = \{0, 1, \ldots, n - 1\}$, and we define on this set operations of addition and multiplication modulo n. Therefore, for every two elements $a, b \in \mathbf{Z}_n$ we define the element $a + b \pmod{n}$, which is defined as the (unique) element c of $\mathbf{Z}_n$ such that $c = a + b \pmod{n}$.

For example, $5 + 4 = 2 \pmod{7}$ and $2 + 1 = 0 \pmod{3}$.

Similarly, we define a product operation in $\mathbf{Z}_n$. For example, $3 \cdot 4 = 3 \pmod{9}$.

Addition in $\mathbf{Z}_n$ is "invertible." Specifically, for each element $a \in \mathbf{Z}_n$ there is an element $a' \in \mathbf{Z}_n$ such that $a + a' = 0 \pmod{n}$ (one can take $a' = n - a$). This gives an analog in $\mathbf{Z}_n$ of the subtraction operation.

Multiplication, alas, is not necessarily invertible. That is, it is not necessarily true that for an element $a \in \mathbf{Z}_n$ there is an element $a' \in \mathbf{Z}_n$ such that $a \cdot a' = 1 \pmod{n}$.

Consider for example $\mathbf{Z}_6$ and the element 2. If there was an element $a \in \mathbf{Z}_6$ such that $2 \cdot a = 1$, then we would have $3 \cdot 2 \cdot a = 3 \cdot 1 = 3 \pmod{6}$. But we also have $3 \cdot 2 \cdot a = 6 \cdot a = 0 \cdot a = 0 \pmod{6}$, and so we have a contradiction. It is possible to characterize precisely the cases where an element a has an inverse with respect to multiplication in $\mathbf{Z}_n$. To this aim, we need a result that is also useful for its algorithmic aspect. Recall that the greatest common divisor (abbreviated gcd) of two numbers n and m is the largest integer that is both a divisor of n and a divisor of m.

Theorem 2.2 (Euclid's algorithm) *There exists an algorithm that on input two positive integers m and n returns $k = \gcd(m, n)$ and two integers α, β such that $\alpha n + \beta m = k$. The algorithm runs in time polynomial in the number of digits of n and m.*

Example 7 On input 14 and 10, Euclid's algorithm returns $2 = \gcd(10, 14)$ and the coefficients $\alpha = 3$ and $\beta = -2$. Indeed, $3 \times 10 - 2 \times 14 = 2$. ■

Example 8 On input 60 and 17, Euclid's algorithm returns $1 = \gcd(60, 17)$ and the coefficients $\alpha = 2$ and $\beta = -7$. Indeed, $2 \times 60 - 7 \times 17 = 1$. ■

Let us return to the issue of inverses in $\mathbf{Z}_n$. Suppose a and n are such that $\gcd(a, n) = k > 1$, we will show that a cannot have an inverse. Let us call $b = n/k$. Note that $b \in \mathbf{Z}_n$, $b \neq 0$. Assume by contradiction that there exists $a' \in \mathbf{Z}_n$ such that $a \cdot a' = 1 \pmod{n}$, then $b \cdot (a \cdot a') = b \pmod{n}$, but also (doing operations over the integers now) $b \cdot a \cdot a' = (n/k) \cdot a \cdot a' = n \cdot (a/k) \cdot a'$ which is a multiple of n (since a/k is an integer), and therefore we have $(b \cdot a) \cdot a' = 0 \pmod{n}$.

Consider now the case $\gcd(a, n) = 1$. Then, using Euclid's algorithm, we can find coefficients α, β such that $\alpha a + \beta n = 1$, that is $\alpha a = 1 \pmod{n}$, that is α is an inverse of a. In this case not only does a have an inverse, but we can also find it efficiently using Euclid's algorithm.

We say that two integers n and m are co-prime if $\gcd(n, m) = 1$. Putting everything together we have

Theorem 2.3 *For an element $a \in \mathbf{Z}_n$, there exists an element $a' \in \mathbf{Z}_n$ such that $a \cdot a' = 1 \pmod{n}$ if and only if a and n are co-prime.*

2.3 Groups

Definition 2.4 (Group) *A group is a set G endowed with an operation, that we denote, say, by $\otimes$, that given two elements of G returns an element of G (i.e. for every $a, b \in G, (a \otimes b) \in G$; the operation must satisfy the following properties:*

1. *For every $a, b \in G$, $a \otimes b = b \otimes a$;*
2. *For every $a, b, c \in G$, $(a \otimes b) \otimes c = a \otimes (b \otimes c)$;*
3. *There exists an element $u \in G$ such that for every $a \in G$, $a \otimes u = u \otimes a = a$;*
4. *For every element $a \in G$ there exists an element $a' \in G$ such that $a \otimes a' = u$.*

Remark. To be precise, what we have just defined is the notion of Abelian group, that is a special type of groups. In general, G can be a group even if Property 1 is not satisfied (in such a case, it will be called a non-Abelian group). In this course we will never consider non-Abelian group, so it is not necessary to insist on the difference. An example of a non-Abelian group is the set of $n \times n$ matrices, together with the matrix multiplication operation. ■

Our canonical example of a group is $\mathbf{Z}_n$ with the operation $\cdot + \cdot \pmod n$.

Here is another interesting example. For a prime p, define $\mathbf{Z}^*{}_p = \{1, 2, \ldots, p-1\}$.

Theorem 2.5 *If p is a prime, then $\mathbf{Z}^*{}_p$ together with multiplication $\pmod p$ is a group.*

To prove the theorem we have to check the required properties of a group. First of all, it is definitely true that $a \cdot b = b \cdot a \pmod p$ and that $a \cdot (b \cdot c) = (a \cdot b) \cdot c \pmod p$. Furthermore we have a special element u, namely 1, such that $a \cdot 1 = 1 \cdot a = a \pmod p$. Using the results of the previous section, and the fact that a prime number is co-prime with every other number small than itself, we also have that for every $a \in \mathbf{Z}^*{}_p$ there is an $a' \in \mathbf{Z}^*{}_p$ such that $a \cdot a' = 1 \pmod p$. In fact, we should also check one more property, namely that for every $a, b \in \mathbf{Z}^*{}_p$ it is true that $a \cdot b \pmod p \in \mathbf{Z}^*{}_p$, i.e. that is impossible that $a \cdot b = 0 \pmod p$. This follows by contradiction: if $a \cdot b = 0 \pmod p$, this means that $a \cdot b$ (taking the product over the integers) is a multiple of p. Since p is a prime number, it means that either a or b is a multiple of p. But both a and b are smaller than p, and so we have a contradiction.

It would be nice to have a similar result for arbitrary n, and say that $\mathbf{Z}_n - \{0\}$ is a group with respect to multiplication $\pmod n$. Unfortunately this is not true when n is a composite number (elements of $\mathbf{Z}_n$ having some common factor with n do not have an inverse, as seen in the previous section). Yet, it is still possible to define a group.

Define $\mathbf{Z}^*{}_n = \{a : 1 \leq a \leq n-1 \text{ and } \gcd(a, n) = 1\}$. For example, $\mathbf{Z}^*{}_6 = \{1, 5\}$ and $\mathbf{Z}^*{}_{10} = \{1, 3, 7, 9\}$. Note that the definition of $\mathbf{Z}^*{}_p$ for p prime is a special case of the previous definition.

Theorem 2.6 *For every positive integer n, $\mathbf{Z}^*{}_n$ is a group with respect to multiplication.*

We denote by $\phi(n)$ the number of elements of $\mathbf{Z}^*{}_n$, i.e. the number of elements of $\{1, 2, \ldots, n-1\}$ that are co-prime with n. It is easy to compute $\phi(n)$ given a factorization of n (but is hard otherwise).

Theorem 2.7

1. *If p is prime and $k \geq 1$ then $\phi(p^k) = (p-1)p^{k-1}$.*
2. *If n and m are co-prime then $\phi(nm) = \phi(n)\phi(m)$.*
3. *If the factorization of n is $\prod_i q_i^{k_i}$ then $\phi(n) = \prod_i (q_i - 1) q_i^{k_i - 1}$.*

Note that the third item in the theorem follows from the first two.

Example 9 . In order to compute $\phi(45)$ we compute the factorization $45 = 3^2 \cdot 5$ and then we apply the formula $\phi(45) = (3-1) \cdot 3^{2-1} \cdot (5-1) = 24$. Indeed, we can check that $\mathbf{Z}^*{}_{45} = \{1, 2, 4, 7, 8, 11, 13, 14, 16, 17, 19, 22, 23, 26, 28, 29, 31, 32, 34, 37, 38, 41, 43, 44\}$ has 24 elements. ■

For every n, the group $\mathbf{Z}_n$ with respect to the sum has the following nice property: every element $k \in \mathbf{Z}_n$ can be obtained by summing 1 to itself k times. One can define a generalization of this property for arbitrary groups.

Definition 2.8 *let G be a group with n elements and operation $\otimes$. Suppose there exists an element $g \in G$ such that g, $g \otimes g$, $g \otimes g \otimes g$,..., $\underbrace{g \otimes g \otimes \cdots \otimes g}_{n \text{ times}}$ are all different (and so cover all the elements of G). Then G is said to be* cyclic *and g is said to be a generator of G.*

Therefore $\mathbf{Z}_n$ is always cyclic and 1 is a generator. Quite interestingly, there is a similar result for $\mathbf{Z}^*{}_p$, p prime.

Theorem 2.9 *For every prime p, $\mathbf{Z}^*{}_p$ is a cyclic wit respect to multiplication* $\pmod p$.

We denote by $a^k \pmod n$ the value $\underbrace{a \cdot a \cdots a}_{k \text{ times}} \pmod n$.

Example 10 The group $\mathbf{Z}^*{}_7$ has generators 3 and 5. Indeed, we have

$$3^1 = 3 \pmod 7 \ , \ 3^2 = 2 \pmod 7 \ , \ 3^3 = 6 \pmod 7 \ ,$$

$$3^4 = 4 \pmod 7 \ , \ 3^5 = 5 \pmod 7 \ , \ 3^6 = 1 \pmod 7$$

The sequence of powers of 5 is $5, 4, 6, 2, 3, 1$. ■

The following results are useful in the analysis of RSA.

Theorem 2.10 (Fermat's Little Theorem) *If p is a prime and $a \in \mathbf{Z}_p^*$, then $a^{p-1} = 1 \pmod p$.*

Fermat's theorem is a special case of the following result.

Theorem 2.11 (Euler's theorem) *If $n \geq 2$ and $a \in \mathbf{Z}_n^*$, then $a^{\phi(n)} = 1 \pmod n$.*

In fact the definition of the function $\phi()$ is due to Euler.

Some of our interest in the notions of cyclic groups and generators is related to the exponentiation function (a candidate one-way function). In that application, one has to be able to generate at random a generator of a group $\mathbf{Z}^*{}_p$ (p prime). For this random generation, it is important to be able to test whether a given element is a generator and to know how many elements are generators.

Given n, $a \in \mathbf{Z}^*{}_n$ *and the factorization of* $\phi(n)$, there is an efficient algorithm that checks whether a is a generator for $\mathbf{Z}^*{}_n$. (By "efficient" algorithm we mean an algorithm that runs in time polynomial in the number of digits of n and a — and in the number of bits needed to represent the factorization of $\phi(n)$, which is polynomial in the number of digits of n anyway.)

Furthermore, the following results guarantee that there are several generators.

Theorem 2.12 *For a prime p, $\mathbf{Z}^*{}_p$ has $\phi(p-1)$ generators.*

Theorem 2.13 *For every n, $\phi(n) \geq n/6 \ln \ln n$.*

Using the prime number theorem, it is a triviality to prove (for $n \geq 17$) the weaker bound $\phi(n) \geq \frac{(n-1)}{\ln(n-1)} - \log n$. Indeed, there are $\pi(n-1) \geq \frac{(n-1)}{\ln(n-1)}$ primes in the interval $2, \ldots, n-1$, and all of them are co-prime with n, except those that are factors of n. But n has at most $\log n$ factors.

2.4 Some more algorithmic tools

2.4.1 The Chinese Remainders Theorem

The following is a very useful algorithmic result.

Theorem 2.14 (Chinese Remainders Theorem) *Consider a system of congruences of the form*

$$\begin{aligned} x &= a_1 \pmod{n_1} \\ x &= a_2 \pmod{n_1} \\ &\cdots \\ x &= a_k \pmod{n_k} \end{aligned}$$

Where $n_1, \ldots, n_k$ are pairwise co-prime. Then there is always a solution x in the interval $1, \ldots, n_1 \cdot n_2 \cdots n_k - 1$, and this solution is the only one in the interval. Such a solution x is efficiently computable given $a_1, \ldots, a_k, n_1, \ldots, n_k$. Furthermore, the set of all solutions is precisely the set of integers y such that $y = x \pmod{n_1 \cdot n_2 \cdots n_k}$.

For example, consider the system

$$\begin{aligned} x &= 5 \pmod 7 \\ x &= 2 \pmod 6 \\ x &= 1 \pmod 5 \end{aligned}$$

Then: $x = 26$ is a solution; 26 it is the only solution in the interval $1, \ldots, 209$; for every integer i we have that $26 + 210i$ is a solution; there is no other solution.

The algorithm to find the solution is simple. Let us call $N = n_1 \cdot n_2 \cdots n_k$ and $N_i = N/n_i$. Let us also call $y_i = (N_i)^{-1} \pmod{n_i}$, that is, y_i is such that $y_i \cdot N_i = 1 \pmod{n_i}$. By our assumptions on $n_1, \ldots, n_k$ it must be that $\gcd(N_i, n_i) = 1$, so y_i is well defined. Then a solution to the system is

$$\sum_{i=1}^{k} a_i N_i y_i \pmod{n_1 \cdot n_2 \cdots n_k}$$

2.4.2 Quadratic Residues

A (positive) integer x is said to be a *perfect square* if there is some integer $y \in \mathbf{Z}$ such that $x = y^2$; if so, y is said to be a *square root* of x. For example 25 is a perfect square, whose square roots are 5 and -5, while 20 is not a perfect square.

If an integer x is a perfect square then it has precisely two square roots, and they are efficiently computable given x. Unfortunately, things are not so easy with modular arithmetic.

Definition 2.15 *An element $x \in \mathbf{Z}_n$ is said to be a quadratic residue $\pmod n$ if there exists an element $y \in \mathbf{Z}_n$ such that $x = y \cdot y \pmod n$. If so, y is said to be a square root of $x \pmod n$.*

Let us first consider the case of $\mathbf{Z}_p$ with p prime. Then things are not too different from the case of the integers.

Theorem 2.16 *Let p be a prime number. If $x \in \mathbf{Z}_p$ ($x \neq 0$) is a quadratic residue, then it has exactly two square roots.*

Theorem 2.17 *There exists an efficient randomized algorithm that on input p prime and $x \in \mathbf{Z}_p$ tests whether x is a quadratic residue and, if so, returns the two square roots of x.*

Among the integers, perfect squares are quite rare, and they get sparser and sparser: there are only about $\sqrt{n}$ perfect squares in the interval $1, \ldots, n$. The situation is quite different in the case of $\mathbf{Z}_p$.

Theorem 2.18 *Let $p \geq 3$ be a prime. Then $\mathbf{Z}_p$ has $(p-1)/2$ non-zero quadratic residues.*

This is easy to see: each one of the $p-1$ element $y \neq 0$ is the square root of $y \cdot y$, on the other hand every quadratic residue has two square roots, and so there must be $(p-1)/2$ quadratic residues.

When we are in $\mathbf{Z}_n$ the situation gets more involved, and it is conjectured that no efficient algorithm, on input x and n, can determine whether x is a quadratic residue, let alone find a root. The same conjecture holds when n is the product of two large primes.

On the other hand, if $n = pq$ is the product of two primes, and *the factorization of n is known*, then extracting square roots becomes feasible. The next two results are proved using the Chinese Remainders Theorem. The algorithm of Theorem 2.20 uses the algorithm of Theorem 2.17 as a subroutine.

Theorem 2.19 *Let n be the product of two primes. If x is a quadratic residue $\pmod{n}$, then x has precisely 4 square roots in $\mathbf{Z}_n$.*

Theorem 2.20 *There is an efficient randomized algorithm that on input (x, p, q) (where p and q are prime and $x \in \mathbf{Z}_{pq}$) tests whether x is a quadratic residue $\pmod{pq}$; if so, the algorithm finds all the four square roots of x.*

Chapter 3

Computational Models

3.1 Circuits

In these notes we consider proofs of security that work even if the adversary can write a *different program* to break the system for each *different security parameter*, i.e. length of the key. The possibility of having a different program for each length of the key has a natural formalization in terms of *size of a circuit that realizes the attack in hardware*.

In the Goldwasser-Bellare notes, instead, the security is against an adversary that is modeled as a *single efficient algorithm*. The notion of security considered here is then stronger than the notion used in Goldwasser-Bellare, but this requires to make *stronger assumptions* on our primitives.

For concreteness, here we will show how to achieve semantic security against circuits, by assuming that we have a trapdoor permutation that is secure against circuits. Goldwasser and Bellare state without proof that one can have semantic security against algorithms by assuming that one has a trapdoor permutation that is secure against algorithms. Se we have incomparable results because from a stronger assumption one gets a stronger conclusion. A final remark: when an implication of the form "a certain cryptographic problem is solvable starting from a certain primitive" is proved assuming the adversary is an algorithm, then essentially the same proof will work assuming that the adversary is a circuit, but in general not vice-versa.

Chapter 4

Pseudorandomness

This chapter contains a definitional treatment of one-way functions, pseudorandom generators and pseudorandom functions.

We also include the construction of the Blum-Micali-Yao pseudorandom generator and of the Goldreich-Goldwasser-Micali pseudorandom functions. We present an analysis (that works for non-uniform adversaries) of the Blum-Micali-Yao construction.

4.1 One-Way Functions

A one-way function is a function that is easy to compute and hard to invert. First, let us give a definition that makes sense for a finite function.

Definition 4.1 *For an integer S and a number $0 < \epsilon < 1$, we say that a function $f : D \to R$ is (S, ϵ)-one way with respect to a distribution X of inputs, if for every circuit C of size $\leq S$ (that tries to invert f) we have*

$$\mathbf{Pr}[C(f(x)) = x' \text{ such that } f(x') = f(x)] \leq \epsilon$$

where the probability is taken over the random choices of x from the distribution X.

When the distribution is not specified, then it means that we are referring to the uniform distribution.

So for example if a function is $(10^9, 10^{-5})$-one way, then every circuit containing a billion gates or less can invert it on at most a fraction 1 over 100 thousands of the inputs. A $(2^{100}, 2^{-100})$-one way function is just *not* invertible for all practical purposes (provided that we are considering a realistic distribution of inputs).

In general it is interesting to look at *families* of functions, having increasingly large domains and increasingly strong security properties.

If we want to look at families of finite functions over particular domains (e.g. functions mapping $\mathbf{Z}^*_n$ on itself), then we must also be able to generate descriptions of functions from the family, and to sample elements from their domain.

Definition 4.2 (Family of One-Way Functions) *For an infinite set of indices (also called* names*) $I \subseteq \{0,1\}^*$, a* family of one-way functions *is a family $\{f_i\}_{i \in I}$ of functions $f_i : D_i \to R_i$ indexed by I, such that each D_i is finite, and such that the following properties hold:*

1. *(Names are samplable) There exists a randomized polynomial time algorithm S_1 that on input a positive integer k produces a random $i \in I \cap \{0,1\}^k$. (S_1 runs in time polynomial in k, not in the length of k.)*

2. *(Inputs are samplable) There exists a randomized polynomial time algorithm S_2 than on input a name $i \in I$ produces a random element $S_2(i) \in D_i$.*

3. *(Evaluation is easy) There exists a polynomial time algorithm A_1 that on input a name $i \in I$ and an element $x \in D_i$ returns $A_1(i,x) = f_i(x)$.*

4. *(Inversion is hard) For every polynomial P and sufficiently large k, if we take $i = S_1(k)$, then f_i is $(P(k), \nu(k))$-one way with respect to the distribution $S_2(i)$, where ν is negligible. That is, if A is computed by a circuit of size $\leq P(k)$ then if we pick $i = S_1(k)$, and $x = S_2(i)$ then*

$$\mathbf{Pr}[A(i, f_i(x)) = x' \text{ such that } f_i(x) = f_i(x')] \leq \nu(k)$$

We will often restrict to one-way permutations, i.e. one-way functions that are 1-1 and onto. In this case, the probability in item 4 can be written

$$\mathbf{Pr}[A(i, f_i(x)) = x] \leq \nu(k)$$

since there is only one x' such that $f_i(x') = f_i(x)$ (namely, $x = x'$).

4.2 Hard Core Predicates

For a one-way function f, a hard core predicate is an easy-to-compute 0/1 function B such that given $f(x)$ it is hard to compute $B(x)$. A finite definition of hard-core predicate for a permutation follows.

Definition 4.3 *A predicate $B : D \to \{0,1\}$ is (S, ϵ)-hard core for a permutation $f : D \to D$ and a distribution X on D if for every circuit C of size $\leq S$ (that tries to compute $B(x)$ given $f(x)$) we have*

$$\mathbf{Pr}[C(f(x)) = B(x)] \leq \frac{1}{2} + \epsilon$$

where the probability is taken over the random choices of x according to X.

Since we want to deal with families of one-way functions, we need the usual hurdle of names, samplers and so on. We will give the definition only for one-way permutations, but a more general definition can be given.

Definition 4.4 (Family of Hard-Core Predicates) *For an infinite set of indices (also called* names*) $I \subseteq \{0,1\}^*$, and a family of one-way permutations $\{f_i\}_{i \in I}$, $f_i D_i \to D_i$, a family of predicates $\{B_i\}_{i \in I}$ with $B_i : D_i \to \{0,1\}$ is said to be a family of hard-core predicates for f if the following conditions hold (algorithms S_1 and S_2 are according to the definition of f being one-way):*

1. *(Evaluation is easy) There exists a polynomial time algorithm A_1 that on input a name $i \in I$ and an element $x \in D_i$ returns $A_1(i,x) = B_i(x)$.*

2. *(Inversion is hard) For every polynomial P and sufficiently large k, if we pick $i = S_1(k)$ then B is $(P(k), \nu(k))$-hard core for f, where ν is negligible. That is, if A is computed by a circuit of size $\leq P(k)$ then if we pick $i = S_1(k)$, and $x = S_2(i)$ then*

$$\mathbf{Pr}[A(i, f_i(x)) = B_i(x)] \leq \frac{1}{2} + \nu(k)$$

where ν is negligible.

4.3 Goldreich-Levin

This section contain a complete proof of a weaker (but simpler) version of the result of Goldreich and Levin.

Goldreich and Levin prove[1] that if f is a strong one-way injective function then it is hard to guess the XOR of a random subset of the bits of x given $f(x)$. By "hard" we mean that the best guessing probability for an efficient algorithm is at most $1/2+\nu(n)$, where $\nu()$ is negligible and n is the length of x. In this section we present a much simpler calculation that shows that the guessing probability for an efficient algorithm is at most $3/4 + \nu(n)$.[2] This is a worthwhile calculation to study, because it gives some flavor of the techniques used to prove more general results and is a good way to familiarize with (applications of) results on expectations of random variables.

4.3.1 Statement of the Result

We use the notation $GL(y, r) = \sum_i x_i r_i \pmod 2$ where $y = f_n(x)$.

Lemma 4.5 *Let $f_n : \{0,1\}^n \to \{0,1\}^n$ be a family of permutations. Suppose there is an algorithm G such that*

$$\mathbf{Pr}[G(f_n(x), r) = GL(x, r)] \geq \frac{3}{4} + \epsilon(n)$$

(where the probability is taken over uniform x and r and over the internal randomization of G) then there is an algorithm A such that

$$\mathbf{Pr}[A(f_n(x)) = x] \geq \frac{\epsilon(n)}{4}$$

(where the probability is taken over uniform x and over the internal randomization of A.) Furthermore A runs by making $O(n(\log n)/\epsilon^2)$ calls to G.

As an immediate consequence we have the following theorem.

Theorem 4.6 *Suppose that f_n is a strong one-way permutation, then a polynomial time algorithm that is trying to compute $GL(x, r)$ given $f(x)$ and r has a guessing probability at most $3/4 + \nu(n)$ where ν is negligible and n is the length of x.*

[1] Their result is even more general than the one that we state.

[2] The observation that this weaker result is much simpler to prove is not new, and has appeared in several places.

4.3.2 Description of the Algorithm

In order to prove Lemma 4.5 we have to describe algorithm A and analyze its probability of success.

We first describe the algorithm

```
A(y)
begin
  for i = 1 to n
    // Tries to guess x_i where x = f^{-1}(y)
    for j = 1 to k = (ln 4n)/2ε^2
      pick at random r ∈ {0,1}^n
      let r' be equal to r except r'_i = 1 − r_i
      b = G(y, r)
      b' = G(y, r')
      g_j = b ⊕ b'
    x_i = value occurring more often in g_1, ..., g_k
  return x
end
```

4.3.3 Proof of Lemma 4.5

We shall call

$$I_n = \left\{ x \in \{0,1\}^n : \mathbf{Pr}[G(f_n(x), r) = GL(x, r)] \geq \frac{3}{4} + \frac{\epsilon}{2} \right\}$$

where the probability is taken over uniform r, and over the randomness of G.

Lemma 4.7 *I_n contains at least $(\epsilon/2)2^n$ elements.*

Proof. For a fixed x, call $succ(x) = \mathbf{Pr}[G(f_n(x), r) = GL(x, r)]$. This is a non-negative random variable such that $\mathbf{E}[succ(x)] \geq 3/4 + \epsilon$ (by the assumption of Lemma 4.5). Using Lemma 4.12 below, we can say

$$\mathbf{Pr}[succ(x) \geq 3/4 - \epsilon/2] \geq \epsilon/2$$

and so

$$\mathbf{Pr}[x \in I] \geq \epsilon/2$$

■

Lemma 4.8 *When algorithm $A()$ receives in input $f_n(x)$ with $x \in I_n$, then, for every i and every j, there is a probability $\geq 1/2 + \epsilon$ that when the following piece of code is executed, g_j contains the right value of x_i.*

```
pick at random r ∈ {0,1}^n
let r' be equal to r except r'_i = 1 − r_i
b = G(y, r)
b' = G(y, r')
g_j = b ⊕ b'
```

Proof. We note that it is always true that $GL(y,r) \oplus GL(y,r') = x_i$. This is because

$$GL(y,r) \oplus GL(y,r') = \sum_h x_h r_h + \sum_h x_h r'_h = \sum_h (r_h + r'_h) x_h = x_i \pmod 2$$

where the last step is due to the fact that $r_h + r'_h = 0 \pmod 2$ for all $h \neq i$, by definition of r'. So every time both $b = GL(y,r)$ and $b' = GL(y,r')$ we have that $b \oplus b'$ is correct. By our assumption that $x \in I_n$, there are at most a fraction $1/4 - \epsilon/2$ of r for which $b \neq GL(y,r)$ and also a fraction at most $1/4 - \epsilon/2$ of r' for which $b' \neq GL(y,r')$, and so there is at a probability $\leq 1/2 - \epsilon$ that either $b \neq GL(y,r)$ or $b' \neq GL(y,r')$. We conclude that there is at least a probability $1/2 + \epsilon$ that both $b = GL(y,r)$ and $b' = GL(y,r')$. For a stronger reason there is at least a probability $1/2 + \epsilon$ that $b \oplus b'$ is correct. ■

Lemma 4.9 *When algorithm $A()$ receives in input $f_n(x)$ with $x \in I_n$, then, for every i, there is a probability $\geq 1 - 1/2n$ that the decoding of x_i given by the following piece of code is correct.*

for $j = 1$ **to** $k = (\ln 4n)/2\epsilon^2$
  pick at random $r \in \{0,1\}^n$
  let r' be equal to r except $r'_i = 1 - r_i$
  $b = G(y,r)$
  $b' = G(y,r')$
  $g_j = b \oplus b'$
x_i = value occurring more often in $g_1, \ldots, g_k$

Proof. Consider the random variables $X_1, \ldots, X_k$ where X_j is 1 if g_j is correct and $X_j = 0$ otherwise. By the previous lemma, we have $\mathbf{E}[X_j] \geq 1/2 + \epsilon$ for every j. Call $X = \sum_j X_j$. Using Chernoff, the probability of incorrect decoding is

$$\mathbf{Pr}\left[\sum_j X_j \leq k/2\right] \leq \mathbf{Pr}\left[|X - \mathbf{E}[X]| \geq \epsilon k\right] \leq 2e^{-2\epsilon^2 k} = e^{-\ln(4n)} = 1/2n$$

■

Lemma 4.10 *When algorithm $A()$ receives in input $f_n(x)$ with $x \in I_n$, then there is a probability $\geq 1/2$ that the decoding of x is correct.*

Proof. For each particular index i, the probability of making an error in decoding x_i is at most $1/2n$ by the previous lemma. Then the probability of making an error in at least one of the n coordinates is at most $n \cdot \frac{1}{2n} = \frac{1}{2}$. So the probability of making no error is at least $1 - 1/2 = 1/2$. ■

Lemma 4.11 *Algorithm $A()$ has a probability $\geq \epsilon/4$ of decoding x correctly.*

Proof. There is a probability at least $\epsilon/2$ that a randomly chosen x is in I_n. Conditioned on this, the probability of correct decoding is at least $1/2$. So the probability of correct decoding is at least $\epsilon/4$. ■

4.3.4 Probability Results

Lemma 4.12 *Let X be a random variable such that $0 \leq X \leq 1$. Then for every $0 < \epsilon < \mathbf{E}[X]$ we have*

$$\mathbf{Pr}[X \geq \mathbf{E}[X] - \epsilon] \geq \epsilon$$

Proof. Consider $\mathbf{Pr}[X < \mathbf{E}[X] - \epsilon]$. Then we have

$$\begin{aligned}
\mathbf{Pr}[X < \mathbf{E}[X] - \epsilon] &= \mathbf{Pr}[1 - X > 1 - \mathbf{E}[X] + \epsilon] \\
&\leq \frac{\mathbf{E}[1 - X]}{1 - \mathbf{E}[X] + \epsilon} \\
&= \frac{1 - \mathbf{E}[X]}{1 - \mathbf{E}[X] + \epsilon} \\
&= 1 - \frac{\epsilon}{1 - \mathbf{E}[X] + \epsilon} \\
&\leq 1 - \epsilon
\end{aligned}$$

And so

$$\mathbf{Pr}[X \geq \mathbf{E}[X] - \epsilon] = 1 - \mathbf{Pr}[X < \mathbf{E}[X] - \epsilon] \geq 1 - (1 - \epsilon) = \epsilon$$

■

Lemma 4.13 (Chernoff's Bound) *Let $X_1, \ldots, X_k$ be independent 0/1 random variables with the same distribution. Let $X = \sum_i X_i$. Then*

$$\mathbf{Pr}[|X - \mathbf{E}[X]| \geq \epsilon n] \leq e^{-2\epsilon^2 k}$$

4.4 Pseudorandomness

4.4.1 Computational Indistinguishability

Two distributions X and Y with the same range are said to be (S, ϵ)-indistinguishable if for every circuit C of size $\leq S$ (that tries to behave differently on input X or on input Y) we have

$$|\mathbf{Pr}[C(X) = 1] - \mathbf{Pr}[C(Y) = 1]| \leq \epsilon$$

Two families of distributions X_k and Y_k defined over $\{0,1\}^k$ are said to be computationally indistinguishable if

- For every polynomial P and for every sufficiently large k,
- For every circuit C of size $\leq P(k)$,
- X_k and Y_k are $(P(k), \nu(k))$-indistinguishable, where ν is negligble. That is,

$$|\mathbf{Pr}[C(X_k) = 1] - \mathbf{Pr}[C(Y_k) = 1]| \leq \nu(k)$$

A good way to see computational indistinguishability is to say that the *effect* of an element sampled from either distribution on an efficient computation is essentially the same. Therefore the two distribution may be interchanged without noticeable effect.

Another way to see the difference is to imagine the following game. Suppose X and Y are (S, ϵ)-indistinguishable. We generate a string z as follows:

- With probability $1/2$ we sample z from X;
- With probability $1/2$ we sample z from Y.

Then we want to find a small circuit C that on input z sampled as above tries to determine whether it is coming from X or from Y. By convention, let us assume that C outputs 1 when it believes that z is coming from X and 0 otherwise. Then we can verify that the probability that C makes the right guess is

$$\frac{1}{2}\mathbf{Pr}[C(X) = 1] + \frac{1}{2}\mathbf{Pr}[C(Y) = 0] = \frac{1}{2} + \frac{1}{2}(\mathbf{Pr}[C(X) = 1] - \mathbf{Pr}[C(Y) = 1]) < \frac{1}{2} + \epsilon$$

where the last step uses the computational indistinguishibility. Then we see that the probability that our circuit guesses correctly is only marginally better than half. Guessing probability $1/2$ is achievable by always answering 1, so this means that no non-trivial guessing is possible.

4.4.2 Definition of Pseudorandom Generator

A pseudorandom generator of expansion $l(\cdot)$ is an algorithm that on input k and $x \in \{0,1\}^k$ computes $G_k(x)$ where $G_k : \{0,1\}^k \to \{0,1\}^{l(k)}$, with the property that $G_k(U_k)$ and $U_{l(k)}$ are computationally indistinguishable.

4.5 The Blum-Micali-Yao Construction

4.5.1 One More Bit

We first see how to get $l(k) = k + 1$.

Let f be a one-way permutation and B a hard-core predicate. Then the proposed construction is

$$G(x) = f_n(x), B_n(x) \ .$$

The analysis of the previous construction consists in the proof of the following result.

Lemma 4.14 *Let $B : \{0,1\}^k \to \{0,1\}$ be a (S, ϵ)-hard core predicate of a permutation $f : \{0,1\}^k \to \{0,1\}^k$. Then the output of the generator*

$$G(x) = f(x), B(x)$$

is $(S - O(1), \epsilon)$-indistingishable from uniform.

Remark. The reader should check that when we apply Lemma 4.14 to a *family* of permutation with a family of hard-core predicates, then we get a pseudorandom generator according to the definition given above. Indeed Lemma 4.14 is particularly strong since it is showing that the two

fundamental security parameters (the size of the adversaries and its "success probability") are preserved. ■

We prove the previous statement by contradiction. Namely, we prove the following equivalent statement.

Lemma 4.15 *Let $f : \{0,1\}^k \to \{0,1\}^k$ be a permutation and $B : \{0,1\}^k \to B$ be a predicate. Let also C be a circuit of size S such that*

$$|\mathbf{Pr}[C(f(x), B(x)) = 1] - \mathbf{Pr}[C(f(x), b)] = 1]| > \epsilon$$

Then there exists a circuit D of size $S + O(1)$ such that

$$\mathbf{Pr}[D(f(x)) = B(x)] > \frac{1}{2} + \epsilon$$

The reader is strongly advised to convince himself/herself that Lemmas 4.14 and 4.15 are really saying the same thing. Good familiarity with the definitions of indistinguishable distributions and hard-core predicates is important to understand the logic of the proof that follows (the single steps are easy to understand).

Let us prove Lemma 4.15. Let C be such that

$$|\mathbf{Pr}[C(f(x), B(x)) = 1] - \mathbf{Pr}[C(f(x), b) = 1]| > \epsilon$$

then without loss of generality we can assume

$$\mathbf{Pr}[C(f(x), B(x)) = 1] - \mathbf{Pr}[C(f(x), b) = 1] > \epsilon$$

(Otherwise we just complement the output of C.)

An alternative way of looking at C is to say that it disinguishes the cases when its input is $f(x), B(x)$ and when it is $f(x), \bar{B}(x)$. Let us prove this.

$$\begin{aligned}
\epsilon &< \mathbf{Pr}[C(f(x), B(x)) = 1] - \mathbf{Pr}[C(f(x), b) = 1] \\
&= \mathbf{Pr}[C(f(x), B(x)) = 1] \\
&\quad -(\mathbf{Pr}[C(f(x), b) = 1 | B(x) = b]\mathbf{Pr}[B(x) = b] + \mathbf{Pr}[C(f(x), b) = 1 | \bar{B}(x) = b]\mathbf{Pr}[\bar{B}(x) = b]) \\
&= \mathbf{Pr}[C(f(x), B(x)) = 1] - \frac{1}{2}\mathbf{Pr}[C(f(x), B(x)) = 1] - \frac{1}{2}\mathbf{Pr}[C(f(x), \bar{B}(x)) = 1] \\
&= \frac{1}{2}\mathbf{Pr}[C(f(x), B(x)) = 1] - \frac{1}{2}\mathbf{Pr}[C(f(x), \bar{B}(x)) = 1]
\end{aligned}$$

So that

$$\mathbf{Pr}[C(f(x), B(x)) = 1] - \mathbf{Pr}[C(f(x), \bar{B}(x)) = 1] > 2\epsilon$$

Hope this looks natural and not just the result of algebraic tricks. If C is able to distinguish $B(x)$ from random given $f(x)$, then it better be able to distinguish $B(x)$ from its complement.

Consider the algorithm D described as follows.

```
D(y)
// D receives in input y = f(x) and tries to guess B(x)
begin
  pick a random b ∈ {0,1}
  if C(y,b) = 1 then return b
  else return b̄
end
```

Observe that the output of D on input y and random guess b is always $1 \oplus b \oplus C(x,b)$. Let us see there is a good probability that our algorithm D guesses correctly.

$$\begin{aligned}
\mathbf{Pr}[D(f(x)) = B(x)] &= \mathbf{Pr}[1 \oplus b \oplus C(x,b) = B(x)] \\
&= \mathbf{Pr}[C(x,b) = 1 \oplus b \oplus B(x) | b = B(x)]\mathbf{Pr}[b = B(x)] \\
&\quad + \mathbf{Pr}[C(x,b) = 1 \oplus b \oplus B(x) | b \neq B(x)]\mathbf{Pr}[b \neq B(x)] \\
&= \frac{1}{2}\mathbf{Pr}[C(x,B(x)) = 1] + \frac{1}{2}\mathbf{Pr}[C(x,\bar{B}(x)) = 0] \\
&= \frac{1}{2}\mathbf{Pr}[C(x,B(x)) = 1] + \frac{1}{2}(1 - \mathbf{Pr}[C(x,\bar{B}(x)) = 1]) \\
&= \frac{1}{2} + \frac{1}{2}(\mathbf{Pr}[C(x,B(x)) = 1] - \mathbf{Pr}[C(x,\bar{B}(x)) = 1]) \\
&> \frac{1}{2} + \epsilon
\end{aligned}$$

The probability was taken both over the choice of x and over the choice of b. If we want D to be completely deterministic (as in the definition of circuit), we note that

$$\begin{aligned}
\mathbf{Pr}[D(f(x)) = B(x)] &= \frac{1}{2}\mathbf{Pr}[D(f(x)) = B(x) | b = 0] + \frac{1}{2}\mathbf{Pr}[D(f(x)) = B(x) | b = 1] \\
&\leq \max\{\mathbf{Pr}[D(f(x)) = B(x) | b = 0] \ , \ \mathbf{Pr}[D(f(x)) = B(x) | b = 1]\}
\end{aligned}$$

So either by fixing b to be always 0 or by fixing b to be always 1 we have a probability of guessing that is at least as good as if b was taken at random.

4.5.2 Several More Bits

We will now see how to get a generator that stretches its input by more than bit.

Our starting point will be once more a permutation f and a hard-core predicate B. This time we will only sketch the analysis.

So let $f : \{0,1\}^k \to \{0,1\}^k$ be a permutation and $B : \{0,1\}^k \to \{0,1\}$ be a predicate.

We describe a generator $G_l : \{0,1\}^k \to \{0,1\}^l$. The generator is as follows.

$$G_l(x) = B(x)B(f(x)), \cdots B(f^{(l-1)}(x))$$

The analysis of such a generator is given by the following result.

Theorem 4.16 *Suppose that B is a (ϵ, S)-hard core predicate for f, and that both B and f are computable by circuits of size $\leq P$, then $G_l(\cdot)$ is $(\epsilon l, S - O(lP))$-indistinguishable from uniform.*

As usual, the proof is by contradiction.

Lemma 4.17 *Suppose there is a circuit C of size S such that*

$$|\mathbf{Pr}[C(G_l(x)) = 1] - \mathbf{Pr}[C(r) = 1]| > \epsilon \tag{4.1}$$

where x is uniform in $\{0,1\}^k$ and r is uniform in $\{0,1\}^l$. Suppose also that B and f are computable by circuits of size at most P.

Then there is a circuit D of size $\leq S + O(lP)$ such that

$$\mathbf{Pr}[D(f(x)) = B(x)] > \frac{1}{2} + \frac{\epsilon}{l}$$

As a first step, we note that without loss of generality we can assume (posibly raising the size of C to $S+1$) that Expression (4.1) can be written as

$$\mathbf{Pr}[C(G_l(x)) = 1] - \mathbf{Pr}[C(r) = 1]| > \epsilon \tag{4.2}$$

We now do a hybrid argument. We define $l+1$ distributions $X_0, \ldots, X_l$. The distribution X_i is defined by computing $g = G_l(x)$ for a random $x \in \{0,1\}^k$ and picking $r \in \{0,1\}^l$ at random; then the first i bits of r are concatenated with the last $l-i$ bits of g. We have by definition that X_0 is distributed like $G_l(x)$ and X_l is uniform.

So we can rewrite Expression (4.2) as

$$\mathbf{Pr}[C(X_0) = 1] - \mathbf{Pr}[C(X_l) = 1] > \epsilon$$

As in the analysis of the Goldwasser-Micali cryptosystem, the hybrid argument proceeds by noting that we can write

$$\begin{aligned} \epsilon &< \mathbf{Pr}[C(X_0) = 1] - \mathbf{Pr}[C(X_l) = 1] \\ &= \sum_{j=0}^{l-1} \mathbf{Pr}[C(X_j) = 1] - \mathbf{Pr}[C(X_{j+1}) = 1] \end{aligned}$$

and so there exists one j for which

$$\mathbf{Pr}[C(X_j) = 1] - \mathbf{Pr}[C(X_{j+1}) = 1] > \frac{\epsilon}{l}$$

which means that C can distinguish

$$X_j = b_1, \ldots, b_j, B(f^{(j)}(x)), \cdots, B(f^{(l)}(x))$$

from

$$X_{j+1} = b_1, \ldots, b_j, b_{j+1}, B(f^{(j+1)}(x)), \cdots, B(f^{(l)}(x))$$

(where b_h are random bits)

Now we use a neat trick. Since x is uniformly random and f is a permutation, then $f(x)$ is uniformly random. By induction, $f^{(i)}(x)$ is uniform and random for every i. This means that the two distributions above can be equivalently redefined if we substitute $f^{(j+1)}(x)$ with a uniformly random element y. All this is giving us that C can distinguish

$$b_1, \ldots, b_j, B(f^{(-1)}(y)), B(y), \cdots, B(f^{(l-j-2)}(y))$$

from

$$b_1, \ldots, b_j, b_{j+1}, B(y), \cdots, B(f^{(l-j-2)}(y))$$

On input y we can compute $f(y), \cdots, f^{(l-j-2)}(y)$ and also $B(y), \cdots, B(f^{(l-j-2)}(y)$.
Here is algorithm D

```
D(y)
// D receives in input y = f(z) and tries to guess B(z) = B(f^{(-1)}(z))
begin
  pick random b_1, ..., b_{j+1} ∈ {0,1}
  if C(b_1, ..., b_{j+1}, B(y), ..., B(f^{(l-j-2)}(y))) = 1 then return b
  else return b̄
end
```

The analysis proceeds as in the previous section and it is omitted. (Additional details may be provided in a future version of these notes.)

4.5.3 Several Samples

In the definition of indistinguishability (the intuitive definition, not its quantitative version), we say that when two distributions X and Y are indistinguishable then the computation of a small circuit on input X or on input Y are essentially the same. It is a good question to ask what happens when we give to a small circuit (X_1, X_2) (where X_1 and X_2 are independent samples from X) or (Y_1, Y_2) (where Y_1 and Y_2 are independent samples from Y). Is it still true that a small circuit cannot distinguish these bigger distributions? Perhaps it is intuitively true that the answer is yes. At any rate a rigorous proof is necessary. We might as well prove a more general result.

Theorem 4.18 *If X and Y are (S, ϵ)-indistinguishable distributions whose range is B^k, then for every integer t, the distributions $(X_1, \ldots, X_t)$ and $(Y_1, \ldots, Y_t)$, obtained by taking t independent copies of X and Y respectively, are $(t\epsilon, S - O(kt))$-indistinguishable.*

We cannot help proving the theorem by contradiction, using the following result.

Lemma 4.19 *Let X and Y be distributions with the same range $\{0,1\}^k$. Suppose that C is a circuit of size $\leq S$ such that*

$$|\mathbf{Pr}[C(X_1, \ldots, X_t) = 1] - \mathbf{Pr}[C(Y_1, \ldots, Y_t) = 1]| > \epsilon$$

(where X_j and Y_j are independet copies of X and Y, respectively).

Then there exists a circuit D of size $\leq S + O(kt)$ such that

$$|\mathbf{Pr}[D(X) = 1] - \mathbf{Pr}[D(Y) = 1]| > \frac{\epsilon}{t}$$

And we cannot help proving the lemma using a hybrid argument. We first assume that C is such that

$$\mathbf{Pr}[C(X_1, \ldots, X_t) = 1] - \mathbf{Pr}[C(Y_1, \ldots, Y_t) = 1] > \epsilon$$

We consider $t+1$ distributions $Z^0, \ldots, Z^t$. The distribution Z^i is defined as $X_1, \ldots, X_i, Y_{i+1}, \ldots, Y_t$. That is, in order to construct the distribution Z^i we pick i independent samples according to X and then $t - i$ independent samples from Y.

By definition, Z^0 is the same as $Y_1, \ldots, Y_t$ and Z^t is the same as $X_1, \ldots, X_t$. We can then write

$$\mathbf{Pr}[C(Z^t) = 1] - \mathbf{Pr}[C(Z^0) = 1] > \epsilon$$

And we also have

$$\begin{aligned} \epsilon &< \mathbf{Pr}[C(Z^t) = 1] - \mathbf{Pr}[C(Z^0) = 1] \\ &= \sum_{j=1}^{t} \mathbf{Pr}[C(Z^j) = 1] - \mathbf{Pr}[C(Z^{j-1}) = 1] \end{aligned}$$

and so there better exist a j such that

$$\mathbf{Pr}[C(Z^j) = 1] - \mathbf{Pr}[C(Z^{j-1}) = 1] > \frac{\epsilon}{t}$$

If we write the previous expression in full, we have

$$\mathbf{Pr}[C(x_1, \ldots, x_{j-1}, x_j, y_{j+1}, \ldots, y_t) = 1] - \mathbf{Pr}[C(x_1, \ldots, x_{j-1}, y_j, y_{j+1}, \ldots, y_t) = 1] > \frac{\epsilon}{t}$$

where x_i and y_i are picked randomly and independently according to X and Y, respectively.

Now comes the important idea. *There must exist some fixed values* $v_1, \ldots, v_{j-1}, v_{j+1}, \ldots, v_t$ such that

$$\mathbf{Pr}[C(v_1, \ldots, v_{j-1}, x_j, v_{j+1}, \ldots, v_t) = 1] - \mathbf{Pr}[C(v_1, \ldots, v_{j-1}, y_j, v_{j+1}, \ldots, v_t) = 1] > \frac{\epsilon}{t}$$

where the probability is taken over the choice of x_j from X and y_j from Y.

So now we just define

$$D(z) = C(v_1, \ldots, v_{j-1}, z, v_{j+1}, \ldots, v_y)$$

and we see that

$$\mathbf{Pr}[D(X) = 1] - \mathbf{Pr}[D(Y) = 1] > \frac{\epsilon}{t}$$

and that D is computable by a circuit of size $S + O(tk)$.

4.6 Pseudorandom Functions

A very powerful paradigm invented during the development of modern cryptography is the "simulation" paradigm, that arises in several different contexts.

The general idea is that we would like to have a protocol that works according to some idealized (and typically impossible to achieve) model. Then the approach is to construct a realistic protocol and then prove that whatever a bounded adversary can do in the realistic model or in the idealized model is essentially the same. Hence our realistic protocol is effectively as good as the idealized one.

For concreteness, in public-key cryptography we would like the adversary not too see anything related to the message that we are sending, wheras in reality the adversary may get the encryption. The definition of semantic security gives us the guarantee that whathever the adversary can

understand about the message given the encryption, he could without looking at the encryption. So semantically secure public-key cryptography is "equivalent" to an idealized situation where the adversary sees nothing of (the encryption of) the message.

In pseudorandomness, we would like the output of the generator to be *the* uniform distribution. But then a generator with $l(k) > k$ could not exist. Instead, we ask that the outcome of the computation of an efficient algorithm (or circuit) on input a truly random string or on input the string computed by the generator, are essentially the same. Again we have a realistic situation that is close enough to the idealized one that we have in mind.

In this section, the idealized model is that of a function $f : \{0,1\}^l \to \{0,1\}^l$ taken at random among all the functions with the same domain and range (in a typical application, we may have $l = 1,000$). It takes $l2^l$ bits to store such a function, and as much time to generate it at random (and random functions cannot be compressed). In practice, we would like "pseudorandom" functions that have a short representation, an efficient computability, and work as the idealized truly random ones. This is formalized by looking at efficient non-uniform[3] algorithm (small circuits) that "interact" with the function, i.e. use it as a subroutine. If the distribution of outputs of such programs is essentially the same on interaction with a random or pseudorandom function, then we are done.

4.6.1 Formal Definition

The basic idea is that we want to define functions of the form $f : \{0,1\}^l \to \{0,1\}^l$ whose description is of length k much smaller than $l2^l$. So for every $s \in \{0,1\}^k$ we have a function $f_s : \{0,1\}^l \to \{0,1\}^l$.

If we pick f_s for a random s, and we let an efficient algorithm try to "distinguish" f_s from a random function, then we would like it to fail. This needs further formalization. We are looking at algorithms that have access to a function f given as a "black-box" (think of a remote procedure call — this is an example to help understand the concept, not an intended application), so that the algorithm can evaluate f on points of its choice but not access the code used to compute f. Each access to a value of f is charged as if it happens in unit time. At the end of the interaction, the algorithm decides whether it thinks that the function in the black box is truly random or is coming from the distribution f_s. In order for f_s to be pseudorandom, the algorithm must have essentially the same probability of outputs on input f_s for a random s or in input a completely random f. It remains to formalize the notion of black-box access to f for a circuit. We consider a generic circuits $C(\cdot)$ having a special gate of fan-in l and with l outputs. For a function $f : \{0,1\}^l \to \{0,1\}^l$, we denote by $C(f)$ the circuit obtained by plugging into the special gates of the generic circuit a gate that computes f. Generic circuits are technically called "oracle" circuits.

Definition 4.20 *A collection $\{f_s\}_{s\in\{0,1\}^k}$ of functions $f_s : \{0,1\}^k \to \{0,1\}^k$ is (S,ϵ)-pseudorandom if for every oracle circuit C of size $\leq S$ we have*

$$|\mathbf{Pr}[C(f_s) = 1] - \mathbf{Pr}[C(g) = 1]| \leq \epsilon$$

where s is picked randomly in $\{0,1\}^k$ and g is picked randomly among all the functions $g : \{0,1\}^l \to \{0,1\}^l$.

Now the definition has to be extended to families, and be given in an asymptotic sense.

[3]Warning: here we are using "uniform" with the meaning that the word has in complexity theory, where it gives the distinction between algorithms and families of circuits. This is *not* the same meaning as in probability, where a distribution is uniform if all the points in the sample space have the same probability.

Definition 4.21 *An ensemble $\{\{f_s^k\}_{s\in\{0,1\}^k}\}_{k\geq 1}$ of functions such that for every k and every $s \in \{0,1\}^k$ we have $f_s^k : \{0,1\}^{l(k)} \to \{0,1\}^{l(k)}$ where $l(\cdot)$ is some arbitrary function is an ensemble of pseudorandom functions if*

- *There is a polynomial-time algorithm F that on input k, $s \in \{0,1\}^k$, and $x \in \{0,1\}^{l(k)}$ computes $F(k,s,x) = f_s^k(x)$.*
- *For every polynomial P, for sufficiently large k and for a negligible function $\nu()$ we have that $\{f_s^k\}_{s\in\{0,1\}^k}$ is a $(P(k),\nu(k))$-pseudorandom collection of functions.*

4.6.2 GGM Construction

Our starting point is a pseudorandom generator $G : \{0,1\}^k \to \{0,1\}^{2k}$.

Then our idea is to define a function $f_s : \{0,1\}^l \to \{0,1\}^k$ in the following way.

We compute $G(s)$, and this gives a string of length $2k$. We view the output of G as two strings of length k. We give each string in input to one of two copies of G. This way we get two strings of length $2k$, that we view as four strings of length k. We give each string in input to one of four copies of G, and we get in output four strings ... This process goes on l times. At the end we get 2^l strings of length k. These strings specify f_s. Indeed, f_s has 2^l possible inputs, and for each of them we have to specify an output of length k.

Of course, in order to compute $f_s(x)$ for a particular x, it is not necessary to unfold the whole computation. One can follow the particular "path" corresponding to x.

Now we get this more formal. Let us call $G_0 : \{0,1\}^k \to \{0,1\}^k$ the function such that $G_0(s)$ is the first k bits of $G(s)$, and let $G_1 : \{0,1\}^k \to \{0,1\}^k$ be the function such that $G_1(s)$ is the last k bits of $G(s)$. Then

$$f_s(x) = G_{x_l}(G_{x_{l-1}}(\cdots G_{x_1}(s)\cdots))$$

4.6.3 Analysis of the GGM Construction

Theorem 4.22 *Suppose $G : \{0,1\}^k \to \{0,1\}^{2k}$ is a (ϵ, S)-pseudorandom generator computable by a circuit of size P. Then for every T such that $T \leq (S - lP)/kc$, the construction of the previous section gives $(Tl\epsilon, T)$-pseudorandom functions. The constant c is independent of all the other parameters.*

The statement to be proved is

Lemma 4.23 *Suppose C is a circuit of size T such that*

$$|\mathbf{Pr}[C(f_s) = 1] - \mathbf{Pr}[C(g) = 1]| > \epsilon$$

Then there exists a circuit D of size $O(lP + kT)$ such that

$$|\mathbf{Pr}[D(G(s)) = 1] - \mathbf{Pr}[D(r) = 1]| > \frac{\epsilon}{lT}$$

In the previous lemma, one should think of S and $1/\epsilon$ as huge (say, 2^{200}), and of l, P, k as much smaller, and of the same order of magnitude (say, l, k are around 1,000 and P is around 100,000). So we can have a huge T, say 2^{100}, and still a very small value of $Tl\epsilon$ (around 2^{-75}).

The proof is fairly complicated and goes beyond the scope of this course.

Credits

The constructions of pseudorandom generators in these notes are due to Blum and Micali and to Yao. The definition and construction of pseudorandom functions is due to Goldreich, Goldwasser and Micali.

The explicit reference to size and success probability of the adversary has been recently advocated and practiced by Bellare and Rogaway. A related explicit notion of security has been earlier advocated by Levin, and used e.g. in Luby's book.

These notes are probably the first elementary treatment of pseudorandom generators with concrete security in the Bellare-Rogaway notation.

Chapter 5

Trapdoor Permutations and Public Key Encryption

This chapter contains a definitional treatment of trapdoor permutations and semantically secure public-key encryption.

We also present some simplified proofs that work in the case of *non-uniform adversaries*. The meaning of the term "uniform" in this framework, and the notion of "circuit size" are explained in Section 3.1.

For simplicity, instead of considering families of functions and predicates defined over arbitrary domains, we define them over binary strings. We are also assuming that the length of the key is equal to the length of the input. This causes no major loss of generality, and at the price of some more notational weight one can get the treatment for the fully general case from the treatment of these notes.

5.1 Trapdoor Permutations

For an infinite set of indices (also called *names*, or *public keys*) $I \subseteq \{0,1\}^*$, a *family of trapdoor permutations* is a family $\{f_i\}_{i \in I}$ of functions $f_i : \{0,1\}^k \to \{0,1\}^k$ indexed by I, where $k = ||i||$, each f_i is 1-1 and onto (a "permutation") and such that the following properties hold:

1. (Names are samplable) There exists a randomized polynomial time algorithm S_1 that on input a positive integer k produces a random pair (i, t_i) where $i \in I \cap \{0,1\}^k$. (S_1 runs in time polynomial in k, not in the length of k.)

2. (Inputs are samplable) There exists a randomized polynomial time algorithm S_2 than on input a name $i \in I$ produces a random element $S_2(i) \in \{0,1\}^k$.

3. (Evaluation is easy) There exists a randomized polynomial time algorithm A_1 that on input a name $i \in I$ and an element $x \in \{0,1\}^k$ returns $A_1(i,x) = f_i(x)$.

4. (Inversion with trapdoor is easy) There exists a polynomial time algorithm A_2 that on input a pair (i, t_i) as generated by S_1, and an element $y \in \{0,1\}^k$ returns $A_2(i, , t_i, y) = f_i^{-1}(y)$.

5. (Inversion without trapdoor is hard) For every polynomial P and sufficiently large k, if A is computed by a circuit of size $\leq P(k)$ then if we pick $(i, t_i) = S_1(k)$, $x = S_2(i)$ then

$$\mathbf{Pr}[A(i, f_i(x)) = x] \leq \nu(k)$$

where ν is negligible.

5.1.1 Concrete Security

At first sight, it seems that the concrete security definition of trapdoor permutation would not be different from the corresponding definition of one-way function. Instead, there is a difference, since we have to define the hardness of inversion with respect to a randomly chosen index i, even when we look at the finite case.

The reason is that for a fixed i, there is a small circuit that inverts f_i: the trapdoor information for i can be hard-wired into the circuit. This is why the following definition looks a bit more complicated than one would expect.

Definition 5.1 (Trapdoor Permutation: Concrete Security) *A family $\{f_i\}_{i\in I}$ of permutations $f_i : \{0,1\}^k \to \{0,1\}^k$ is (S,ϵ)-trapdoor with respect to a distribution Z on names $i \in I$ and to a random function $X : I \to \{0,1\}^k$ if for every circuit C of size $\leq S$ (that tries to invert f_i) we have*

$$\mathbf{Pr}[C(i, f_i(x)) = x] \leq \epsilon$$

where i is sampled from Z and x from $X(i)$.

In the above definition, Z is playing the role of the random key generation done by $S_1(k)$ in the asymptotic definition, and $X(i)$ is playing the role of the random element generation done by $S_2(i)$.

5.2 Trapdoor Predicates

For an infinite set of indices (also called *names*, or *public keys*) $I \subseteq \{0,1\}^*$, a *family of trapdoor predicates* is a family $\{B_i\}_{i\in I}$ of functions $B_i : \{0,1\}^k \to \{0,1\}$ indexed by I, where $k = ||i||$, such that the following properties hold:

1. (Names are samplable) There exists a randomized polynomial time algorithm S_1 that on input a positive integer k produces a random pair (i, t_i) where $i \in I \cap \{0,1\}^k$. (S_1 runs in time polynomial in k, not in the length of k.)

2. (Inputs are samplable) There exists a randomized polynomial time algorithm S_2 than on input a name $i \in I$ and a bit $b \in \{0,1\}$ produces a random element $x = S_2(i,b) \in \{0,1\}^k$ such that $B_i(x) = b$.

3. (Evaluation with trapdoor is easy) There exists a polynomial time algorithm A_1 that on input a pair (i, t_i) as generated by S_1, and an element $x \in \{0,1\}^k$ returns $A_2(i, , t_i, x) = B_i(x)$.

4. (Evaluation without trapdoor is hard) For every polynomial P and sufficiently large k, if A is computed by a circuit of size $\leq P(k)$ then if we pick $(i, t_i) = S_1(k)$, b randomly in $\{0,1\}$, $x = S_2(i,b)$ then

$$\mathbf{Pr}[A(i,x) = B_i(x)] \leq \frac{1}{2} + \nu(k)$$

where ν is negligible.

5.2.1 Concrete Security

The concrete security definition becomes cumbersome. We will restrict to one particular k, and we will have a distribution Z on the names $I \cap \{0,1\}^k$, and a random function Y that on input name i and bit b, produces an element $Y(i,b) \in \{0,1\}^k$ (Y is modeling S_2).

Definition 5.2 *For a set of names $I \subseteq \{0,1\}^k$, a family $\{B_i\}_{i\in I}$ of predicates $B_i\{0,1\}^k \to \{0,1\}$, a distribution X on $\{0,1\}^k$, a distribution Z on i, and a random function $Y : I \times \{0,1\} \to \{0,1\}^k$, we say that $\{B_i\}$ is a family of (S,ϵ)-trapdoor predicates if*

- *for every $i \in I$ and $b \in \{0,1\}$ we have $\mathbf{Pr}[B_i(Y(i,b)) = b] = 1$;*
- *for every circuit C of size $\leq S$,*

$$\mathbf{Pr}[C(i,x) = B_i(x)] \leq \frac{1}{2} + \epsilon$$

where i is sampled from Z, b is taken at random from $\{0,1\}$ and x is sampled from $Y(i,b)$.

The last condition is equivalent to

$$\mathbf{Pr}[C(i,Y(x,1)) = 1] - \mathbf{Pr}[C(i,Y(x,0)) = 1] \leq 2\epsilon$$

5.3 Public Key Encryption

A public key cryptosystem is made of three randomized polynomial time algorithms (K,E,D) called, respectively, the *key-generation* algorithm, the *encryption* algorithm, and the *decryption* algorithm.

We assume the existence of a set of indices I and a set of domains D_i.

1. On input a positive integer k, $K(k)$ produces a pair (i,t_i) where $i \in \{0,1\}^k$. (The key-generation algorithm works in time polynomial in the number k, not in the number of digits of k).

2. On input i and $x \in \{0,1\}^k$, $E(i,x)$ produces an encryption m. The encryption is a randomized process, and so m is to be considered a random variable even when i and x are fixed.

3. On input i, t_i and $m = E(i,x)$, $D(i,t_i,m) = x$. Typically D does not make random choices. If it does, it must still guarantee that the decoding is always correct (but the polynomial running time might be on average instead of in the worst case).

We have to find a formalization of the following necessary requirement:

- On input $i = S(k)$ and $E(i,m)$, an adversary should be unable to get information about m. This is to be formalized later.

5.3.1 Definition of Semantic Security

A public key cryptosystem (K, E, D) is semantically secure if

- For every polynomial P, there exists a polynomial P', such that for sufficiently large k and i generated by $S_1(k)$,
- For every distribution X over $\{0,1\}^k$,
- For every functions $h : \{0,1\}^k \to \{0,1\}^*$ (partial information) and $f : \{0,1\}^k \to \{0,1\}^*$ (interesting property) whose output is of length at most $P(k)$
- For every circuit C of size $\leq P(k)$ (that computes $f(x)$ given $h(x)$ and an encryption of x),
- There exists a circuit C' of size $\leq Q(k)$ (that computes $f(x)$ given $h(x)$ only) such that

$$\mathbf{Pr}[C(i, h(x), E(i, x)) = f(x)] \leq \mathbf{Pr}[C'(i, h(x)) = f(x)] + \nu(k)$$

  where i is distributed as in $S_1(k)$, x is taken according to the distribution X, and the probability is also taken over the internal random choices of $E()$. The function ν has to be negligible.

The intuitive motivation of the definition of semantic security is to provide a formalization of the notion that the if the adversary's goal is to get partial information about the message m (formalized by the ability of guessing the value of $f(m)$) then looking at the encryption does not help.

A Treatment Using Concrete Security

We now give a way to state the condition of message indistinguishability for a fixed length of the key and of the message, and with explicit security parameter.

Definition 5.3 (Concrete Semantic Security) *For a subset $I_k \subseteq \{0,1\}^k$, and a distribution Z over I_k, consider a random function $E : \{0,1\}^k \to \{0,1\}^l$ (i.e. a function such that for every $i \in I_k$ and $m \in \{0,1\}^k$, $E(i,m)$ is still a random variable). Then we say that E is (S, S', ϵ)-semantically secure with respect to Z if for every two functions $f : \{0,1\}^l \to \{0,1\}^*$ and $h : \{0,1\}^l \to \{0,1\}^*$, every distribution X over $\{0,1\}^l$ and every circuit C of size $\leq S$, there is a circuit C' of size S' such that*

$$|\mathbf{Pr}[C(i, E(i,m), h(m)) = f(m)] \leq \mathbf{Pr}[C'(i, h(m)) = f(m)] + \epsilon$$

where the probability is taken over the randomness of $E()$ and over the random choices of i from Z and m from X.

To find the connection with the asymptotic definition, Z is playing the role of $K(k)$, I_k is $I \cap \{0,1\}^k$, and E is the same as before, except restricted to keys of length k and messages of length l. Also note that in the definition of security we are not asking that Z and E be computed by efficient algorithms, and that E be invertible using a trapdoor. These are *efficiency* requirement that can be stated independently of the *security* requirement. (Clearly we need both, we are just saying that the requirements can be *defined* and *analysed* separately.)

5.3.2 Definition of Message-Indistinguishability

- For every polynomial P, sufficiently large k,
- For every two messages m_0 and m_1,
- For every i as chosen by $S_1(k)$,
- For every circuit C (that tries to decide whether its input encrypts m_0 or m_1) of size $\leq P(k)$,

$$\mathbf{Pr}[C(i, E(i, m)) = m] \leq \frac{1}{2} + \nu(k)$$

where the probability is taken over the randomness of $E()$ and the random choice of m in $\{m_0, m_1\}$.

Intuitively, in a message-indistinguishable cryptosystem the encryption of any two messages are hard to tell apart.

Remark. Goldwasser and Micali originally used the term "polynomial security" instead of "message indistinguishability." The modern term reflects better the property of the system. Interestingly, NP-complete problems were originally called, with a similarly confusing name, "polynomially complete." ■

Observe that the condition

$$\mathbf{Pr}[C(i, E(i, m)) = m] \leq \frac{1}{2} + \nu(k)$$

on the distinguishing probability can be equivalently stated as

$$|\mathbf{Pr}[C(i, E(i, m_0)) = 1] - \mathbf{Pr}[C(i, E(i, m_1)) = 1]| \leq \nu(k)$$

(where the function $\nu()$ is not the same but it is still negligible) and it is again equivalent to the statement

$$\mathbf{Pr}[C(i, E(i, m_0)) = 1] - \mathbf{Pr}[C(i, E(i, m_1)) = 1] \leq \nu(k)$$

(where the circuit C may not be the same, but have a comparable size)

A Treatment Using Concrete Security

Definition 5.4 (Message Indistinguishability: Concrete Security) *For a subset $I_k \subseteq \{0,1\}^k$, and a distribution Z over I_k, consider a random function $E : \{0,1\}^k \to \{0,1\}^l$ (i.e. a function such that for every $i \in I_k$ and $m \in \{0,1\}^k$, $E(i, m)$ is still a random variable). Then we say that E is (S, ϵ)-message indistinguishable with respect to Z if for every two messages $m_0, m_1 \in B^l$ and every circuit C of size $\leq S$, we have*

$$|\mathbf{Pr}[C(i, E(i, m_0)) = 1] - \mathbf{Pr}[C(i, E(i, m_1)) = 1]| \leq \epsilon$$

where the probability is taken over the randomness of $E()$ and the random choice of i from Z.

It would be the same to say that the distributions $(Z, E(Z, m_0))$ and $(Z, E(Z, m_1))$ are (S, ϵ)-indistinguishable.

To find the connection with the asymptotic definition, Z is playing the role of $K(k)$, I_k is $I \cap \{0,1\}^k$, and E is the same as before, except restricted to keys of length k and messages of length l.

Note that the condition in the message indistinguishability is the same as asking that if m is chosen randomly in $\{m_0, m_1\}$, then

$$\mathbf{Pr}[C(i, E(i, m)) = m] \leq \frac{\epsilon}{2}$$

5.3.3 Semantic Security Implies Message-Indistinguishability

We have to prove that if a system is not message-indistinguishable then it is not semantically secure. This will follow from the lemma below.

Lemma 5.5 *Suppose $E : \{0,1\}^k \to \{0,1\}^l$ is not (S, ϵ)-message indistinguishable with respect to Z. Then* for every S', *E is not $(S, S', \epsilon/2)$-semantically secure.*

Let C be a circuit of size S and m_1, m_2 be two messages in $\{0,1\}^l$ such that

$$\mathbf{Pr}[C(i, E(i, m)) = m] > \frac{1}{2} + \frac{\epsilon}{2}$$

where the probability is taken over the random choices of $E()$ and the random choice of m in $\{m_0, m_1\}$.

Then consider the distribution X that is uniform in $\{m_0, m_1\}$, the function $h()$ whose output is empty, and the function f such that $f(x) = x$.

Now note that for every C', regardless of its size,

$$\mathbf{Pr}[C'(i, h(X)) = X] = \frac{1}{2}$$

and so

$$\mathbf{Pr}[C(i, h(X), E(i, X)) = X] > \mathbf{Pr}[C'(i, h(X)) = X] + \frac{\epsilon}{2}$$

5.3.4 Message-Indistinguishability Implies Semantic Security

This is going to be a bit harder (and we are going to use in an essential way the fact that the adversary is a family of circuits, rather than a single algorithm)

For a change, the proof is not by contradiction in its higher-level structure, though one of the claims will be proved by contradiction.

Assume (K, D, E) is message-indistinguishable, we want to prove that it is semantically secure. We shall prove the lemma below.

Lemma 5.6 *Suppose $E : I_k \times \{0,1\}^l \to \{0,1\}^k$ is (S, ϵ) message indistinguishable with respect to Z, and that E is computed by a circuit of size P. Then E is $(T, T', 2\epsilon)$-semantically secure where $T = S - O(l)$ and $T' = T + P + O(l)$.*

Remark. We are a bit abusing our definitions when we say that E is "computed" by a circuit of size P. What we mean is that there is some circuit $C_E(\cdot,\cdot,\cdot)$ that for every fixed $i \in I_k$ and $m \in \{0,1\}^k$ is such that $C_E(i,m,y)$ (where the input y is chosen at random) is distributed like $E(i,m)$. In practice, E is computed by a randomized algorithm, i.e. an algorithm that, like C_E, takes in input i, m, and random bits, and creates the distribution $E(i,m)$. This will be clarified when we see actual constructions of secure encryption systems. ■

Consider a distribution X on the message space $\{0,1\}^k$ and arbitrary functions f and h, consider a circuit C of size T and let us prove that there exists a circuit C' of size $T + P + O(l)$ such that

$$\mathbf{Pr}[C(i, E(i,m), h(m)) = f(m)] \leq \mathbf{Pr}[C'(i, h(m)) = f(m)] + \epsilon \tag{5.1}$$

where i is sampled from Z and m is sampled from X.

The circuit C' is defined as follows ($\mathbf{0}$ is the string of length k all whose entries are 0)

```
C'(i, z)
// C' receives in input i and z = h(m) and tries to guess f(m)
begin
  compute c = E(i, 0)
  return C(i, c, h(m))end
```

We have to prove that indeed C' satisfies Expression (5.1). (Here we turn to an argument by contradiction.)

Suppose

$$\mathbf{Pr}[C(i, E(i,m), h(m)) = f(m)] - \mathbf{Pr}[C'(i, h(m)) = f(m)] > \epsilon \tag{5.2}$$

where m is sampled from X and i from Z.

Consider a message $m_{\max}$ that maximizes the expression below

$$\max_{m \text{ in the range of } X}\{\mathbf{Pr}[C(i, E(i,m), h(m)) = f(m)] - \mathbf{Pr}[C'(i, h(m)) = f(m)]\}$$

where the probability is taken on the random choices of E and on the choices of i from Z.

Then we must have

$$\mathbf{Pr}[C(i, E(i, m_{\max}), h(m_{\max})) = f(m_{\max})] - \mathbf{Pr}[C'(i, h(m_{\max})) = f(m_{\max})] > \epsilon$$

We now show that, if the above expression holds, then the encryptions of $\mathbf{0}$ and $m_{\max}$ are distinguishable.

Consider the circuit D defined as follows

```
D(i, c)
// D receives in input i and c = E(i, m)
// and tries to guess whether m = 0 or m = m_max
begin
  if C(i, c, h(m_max)) = f(m_max) then return 1
  else return 0
end
```

Note that D has size $T + O(l) \leq S$. We have

$$\mathbf{Pr}[D(i, E(i, m_{\max})) = 1] - \mathbf{Pr}[D(i, E(i, \mathbf{0})) = 1]$$

$$
\begin{aligned}
&= \mathbf{Pr}[C(i, E(i, m_{\max}), h(m_{\max}))) = f(m_{\max})] - \mathbf{Pr}[C(i, E(i, \mathbf{0}), h(m_{\max}))) = f(m_{\max})] \\
&= \mathbf{Pr}[C(i, E(i, m_{\max}), h(m_{\max})) = f(m_{\max})] - \mathbf{Pr}[C'(i, h(m_{\max})) = f(m_{\max})] \\
&> \epsilon
\end{aligned}
$$

And here is a contradiction to the assumption that E was (S, ϵ)-message indistinguishable.

We must conclude that Expression (5.1) must hold, and the proof is complete.

Remark. [Use of the non-uniform model] The fact that the adversary needs only be a small circuit, or a program specialized on a certain input length, makes the above proof go smoothly on a couple of delicate points. Specifically, we are able to use in the circuit the special message $m_{\max}$ and the value $h(m_{\max})$. We can do that even if $m_{\max}$ may be very hard to find and $h(m_{\max})$ may be very hard to compute, yet both have a small description and can hard-wired into a circuit. So a small circuit using them *exists* even though it may be very hard to construct it. The *non-constructive* nature of the circuit model may not be immediately communicated by the definition, but reveals itself clearly in this kind of proof. ■

5.4 The Goldwasser-Micali Cryptosystem

5.4.1 Construction

Let $\{B_i\}_i i \in I$ be a family of trapdoor predicates, with associated algorithms S_1, S_2, A_2.

Then the cryptosystem is defined as follows:

- (Key Generation) For a length parameter k, sample $(i, t_i) = S_1(k)$, and use i as public key and t_i as private key.
- (Encryption) On input a message $m = m_0, \ldots, m_l$ of length l and public key i, compute $x_1, \ldots, x_l$ given by $x_i = S_2(i, m_i)$. Output the concatenation $x_1, \ldots, x_l$.
- (Decryption) On input a ciphertext $x_1, \ldots, x_l$, public key i and private key t_i, compute plaintext $m = m_1, \ldots, m_l$ by computing $m_i = A_1(i, t_i, x_i)$.

5.4.2 Analysis

We prove that the Goldwasser-Micali cryptosystem is message-indistinguishable (and hence semantically secure).

Lemma 5.7 *Suppose $B_i : \{0,1\}^k \to \{0,1\}$ is a finite family of (S, ϵ)-trapdoor predicates with respect to Z and Y, and suppose that Y is computed by a circuit of size P. Then the Goldwasser-Micali encryption is $(S - O(Pkl), 2\epsilon l)$-message indistinguishable.*

The finite equivalent of the Goldwasser-Micali encryption of a mesage $m = (m_0, \ldots, m_l) \in \{0,1\}^l$ is to sample i from Z and then output the concatenation $(Y(i, m_1), \ldots, Y(i, m_l)) \in \{0,1\}^{kl}$, where each Y is evaluated independently.

The proof is by contradiction. If the system is not message-indistinguishable then B_i is not a family of trapdoor predicates. Specifically, we prove the following result.

Lemma 5.8 *Let Z be a distribution of keys, $m, m' \in \{0,1\}^l$ be two messages and C be a circuit of size S such that*

$$\mathbf{Pr}[C(i, E(i, m)) = 1] - \mathbf{Pr}[C(i, E(i, m')) = 1] > \epsilon$$

Then there exists a circuit C' of size $S + O(lkP)$ such that

$$\mathbf{Pr}[C'(i, x) = B_i(x)] > \frac{1}{2} + \frac{\epsilon}{2l}$$

where i is sampled according to Z and x is sampled by first picking a random bit $b \in \{0, 1\}$ and then sampling from $Y(i, b)$.

We use the hybrid argument, and define $l+1$ messages $m^0, \ldots, m^l$. For $j = 0, \ldots, l$, the message m^j is defined so that the first j bits of m^j are equal to the first j bits of m and the last $l - j$ bits of m^j are equal to the last $l - j$ bit of m'. So $m^l = m$ and $m^0 = m'$.

Then we can write

$$\begin{aligned}
\epsilon &< \mathbf{Pr}[C(i, E(i, m)) = 1] - \mathbf{Pr}[C(i, E(i, m')) = 1] \\
&= \mathbf{Pr}[C(i, E(i, m^l)) = 1] - \mathbf{Pr}[C(i, E(i, m^0)) = 1] \\
&= \sum_{j=1}^{l} (\mathbf{Pr}[C(i, E(i, m^j)) = 1] - \mathbf{Pr}[C(i, E(i, m^{j-1})) = 1])
\end{aligned}$$

And there will be some particular j for which

$$\mathbf{Pr}[C(i, E(i, m^j)) = 1] - \mathbf{Pr}[C(i, E(i, m^{j-1})) = 1] > \frac{\epsilon}{l}$$

Now it should be noted that m^j and m^{j-1} differ only in the j-th bit, by definition. We are ready to define the circuit C' that is what the lemma is asking for.

```
C'(i, x)
// D receives in input i and x from Y(i, b)
// and tries to behave differently if b = 0 or b = 1 begin
   sample "encryptions" c_1, ..., c_{j-1}, c_{j+1}, c_l
      where c_h = Y(i, m_h^j)
   if C(i, c_1, ..., c_{j-1}, x, c_{j+1}, ..., c_j) = z_j^j then return 1
   else return 0
end
```

By definition of C', previous calculations, and the way the Goldwasser-Micali cryptosystem works, we have

$$\mathbf{Pr}[C'(i, Y(i, z_j^j)) = 1] - \mathbf{Pr}[C'(i, Y(i, z_j^{j-1})) = 1] > \frac{\epsilon}{l}$$

which implies

$$|\mathbf{Pr}[C'(i, Y(i, 0)) = 1] - \mathbf{Pr}[C'(i, Y(i, 1)) = 1]| > \frac{\epsilon}{l}$$

and the lemma is proved.

Credits

All the results of this note come from [GM84], except the result that semantic security implies message indistinguishability. The treatment in [GM84] was uniform, which made proofs tremendously harder. The proofs in this notes are from [Gol95]. The concrete security treatment is more or less new.

Chapter 6

Zero Knowledge

The definitive introductory treatment of Zero Knowledge is Chapter 4 of Goldreich's book [Gol95]. This chaper summarizes some of the main ideas and sketch a couple of proofs.

A warning about notation: there a few, unavoidable, collisions of notation through these notes, hopefully the context will avoid any confusion. We use π to denote both a *p*ermutation and a *p*roof. We use G to denote both a *g*raph and a pseudorandom *g*enerator. We use V to denote both a *v*erifier and the set of *v*ertices of a graph. In previous notes we used P to denote a *p*olynomial, while here it denotes a *p*rover.

6.1 Motivations

We all know (or at least we think so) what a "mathematical proof" of a statement is. A mathematical proof has two desirable properties. The first property is that a correct proof *definitely* and *undisputedly* establishes the truth of the statement it refers to. The other property is communicability: after learning a proof of a statement, we can explain it to another person, and convince her of the truth of the statement as much as we were convinced of the truth of the statement when we first saw the proof. Mathematical education relies on this.

In zero knowledge proofs, quite amazingly, we can be convinced by a "prover" party of the validity of a certain statement, but then we are not able to replicate the proof. As a matter of fact, even though we become convinced that a proof (in the conventional sense) exists, and that the prover knows it, we learn nothing about such proof. In fact, even more dramatically, we learn nothing at all that we did not know before, or that we would have not been able to compute by ourselves, except the fact that the statement is true. This is possible because the proof is not just written down and handed to us (in which case replicability would be obvious) but is the result of an *interaction* between two parties: the "prover" entity, and us, the "verifier" entity. Rather than presenting an argument, the prover *answers questions*.

Suppose you attend one of the interview talks for faculty candidates that are going on right now, and take note of what the candidate says, how he answers questions, etc. Would you be able to impersonate the candidate at an interview at another place? Probably not, because questions would be different and impredictable.[1]

[1]This is a somewhat misleading example: a faculty candidate would probably not try to hide all information about his work from the audience, in fact he hopes to do quite the opposite. The example is just trying to motivate the fact that being convinced of the truth of a result by asking impredictable questions is not quite the same as by reading a written proof.

Zero knowledge proofs are more than a mathematical oddity, and their motivation stems from the area of *secure distributed protocols*.

6.1.1 A Relative of Zero Knowledge: Identification Schemes

Before considering these advanced applications, let us consider how the notion of zero knowledge can be applied to a familiar context. Suppose we want to telnet to a remote computer. Since whatever passes through the internet is essentially public, we are not very happy of having to digit our password to gain access to the computer (somebody might be snooping our connection, and he would get our password). Suppose now that instead of sending our password, we *prove in zero knowledge* to the remote computer that we *know the password*.[2] Then if a person is snooping the connection, he is learning nothing about our password. To get a sense of how this can work, suppose we have a good message authentication scheme (MAC) $T(\cdot, \cdot)$, and that our password is the secret key s. Then, at login time, the other computer will send us a random message m, and we reply with the tag $T(s, m)$. So we are always able to get into the system, and by definition of MAC, an adversary that monitors our logins is unable (unless with very low probability) to gain access to the system.

The protocol above works well for the sake of password access, but is not really "zero-knowledge" in the sense that we would like to define in these notes. If the adversary monitors our logins, he learns something about our password: he learns the values of the tag of a few random messages. This is definitely something that he would have not been able to compute by himself. This is not *useful* information, still it is information. We want to do even better than that (for more advanced protocols, where such a level of security is desirable or even mandatory).

We shall now consider a protocol that is not very good, but conveys a sense of how zero knowledge works. Suppose that our password is the trapdoor for a one-way permutation f. The permutation f is public. When we login, the other computer sends us a random element y, and we reply with $f^{-1}(y)$. We are definitely able to gain access to the computer, an adversary that does not have the trapdoor is unable to do so, and even if the adversary monitors our login sessions, he only sees pairs $(y, f^{-1}(y))$, where y is uniformly distributed. This is the same as seeing pairs of the form $(f(x), x)$, where x is uniformly distributed. Seeing such pairs can be of no use, since the adversary *would have been able to produce them by himself*. This is the kind of security that we want to achieve. Note that the above password protocol has a serious drawback: if the adversary *impersonates* the other computer, instead of just monitoring our connections, then he can break the system if, say, f is Rabin's function.

In essence, we want the following properties:

- A legitimate user (like us) should gain access.
- An illegitimate user should not gain access.
- An adversary that impersonates the computer we want to login to gains no useful information. In fact, he could have reconstructed the content of the interaction with us by himself.

The third requirement is similar in spirit to the definition of semantic security. In the definition of semantic security we say that whatever property of the plaintext the adversary was able to compute given the ciphertext, he could have computed it without looking at the ciphertext. So he might as well not bother snooping our communication.

[2] This notion of *proving that we know something* is slightly different, but strongly related, to the notion of *proving the truth of a statement* which we consider in these notes

6.1.2 Other Examples

In this section we review a few contexts where it is useful to be able to prove something without leaking any additional information.

In some applications, a user has to generate a number $n = pq$ where p and q are primes such that $p = 2p' + 1$ and $q = 2q' + 1$, where p' and q' are again prime numbers. In such applications, the user reveals n and keeps p and q secret. It is important for the other parties to make sure that n does have the above described property, but there is no efficient algorithm to test it. The user who generated n can *prove* that n was generated in the right way, by revealing p and q. But, in the application, p and q must be kept secret. Zero knowledge gives a way for the user to prove that n has the required property without giving away any information about n.

A growing area of interest in cryptography is the design of secure distributed protocols, where several users are connected in a network, each one has private information that does not want to share, they distrust each other, and they want to perform some computation that involve their secret information. Consider an auction. Each user has a price that he is willing to pay. The users want to be able to determine who has the higher price, in such a way that at the end this user and his price are announced, but all the other bids remain secret. Furthermore no user (or largish coalition of users) should be able to cheat, e.g. by raising the price they are willing to pay along the way.

There is a generic result that says that if some computation has an efficient implementation in case there are no security requirement, then there is also an implementation that offers the strongest possible security requirements in terms of secrecy of the data and protection against cheating coalitions. This generic result is totally impractical (as any result of this generality is bound to be), but it gives the very important implication that every cryptographic distributed protocol does have a solution (albeit an inefficient one) and that, quite probably, it also has an efficient ad hoc one, which it is good to look for. The general result is established in two stages: there is a first construction that shows that every protocol can be implemented securely if the users behave in a semi-honest way (which has a certain technical meaning); in the full-fledged construction, the users use the previous protocol and also add at any round a zero-knowledge proof that what they have been doing until then is "semi-honest."

Zero knowledge proofs have also be used to give (inefficient) encryption schemes secure against lunch-time attacks. Often, zero-knowledge is a good tool to prove that general (inefficient) solutions exist for certain protocols, and to motivate the search for more efficient constructions.

6.1.3 These Notes

The most important things to learn about zero-knowledge are the definition (especially the concept of simulation), examples showing how it is achievable, and a general theorem saying that whenever we have a proof (in a standard sense) of a statement, then the statement is also provable in zero knowledge. In the rest of these notes we will give definitions of zero knowledge protocols, examples of protocols for graph isomorphism and non-isomorphism and the proof of the main result. The proof will make use of "commitment schemes," a cryptographic construction of independent interest, that also solves the unrelated nice problem of "coin flipping over the telephone."

6.2 Definitions

A brief review about decision problems and languages follows. We want to give tools to prove that certain assertions are true (e.g. a given number n is the product of two primes, two given graphs

are isomorphic). We will always encode statements as strings of bits. We can identify a *property* of strings with the set $L \subseteq \{0,1\}^*$ of strings that have this property. A set of strings is called a *language*. Our goal is to fix a language L we are interested in (e.g. $L = \{n : n$ is an integer, represented as binary string, that is the product of two primes $\}$) and then give a way to prove, for every $x \in L$, that indeed it is the case that $x \in L$.

Definition 6.1 (Interactive Proof System) *Let us fix a language L. Consider two communicating randomized processes P and V. We say that (P, V) are an* interactive proof system *for L if for every x of length k and for a negligible function $\nu()$:*

- Completeness property. *If $x \in L$, then with probability $\geq 1 - \nu(k)$ the interaction of $P(x)$ and $V(x)$ ends with V accepting.*
- Soundness property. *If $x \notin L$, then for every randomized process P', there is a probability $\leq \nu(k)$ that the interaction of $P'(x)$ and $V(x)$ ends with V accepting.*
- Efficient verification property. *V runs in time polynomial in k.*

The idea is that on common input x, prover P tries to convince verifier V of the truth of the statement "$x \in L$." When V is convinced, then it stops the protocol and it accepts, otherwise not. When the statement is true, the first condition says that, except with negligible probability, the prover succeeds in convincing the verifier. The second condition says that if the statement is false, then, even if the prover cheats by not following the prescribed protocol, the verifier will not be fooled and will not accept the incorrect statement.

In the definition, we ask that V runs in polynomial time. For starters, we are not putting restrictions on P (we will consider efficient implementations of P later). We will never make restrictions on the computational power of P'. This will give a very strong notion of correctness: not even an infinitely powerful adversary can manage to get a non-negligible probability of fooling the verifier into accepting an false statement.

Since P and V are communicating processes, their computation proceeds in a number of *rounds*: they start with, say, the prover sending a message to the verifier, then the verifier does some computation, then he sends a message to the prover, then he does some computation, etc.

Both P and V are randomized processes. Neither knows the random choices made by other. (This is important in order to make V be able to ask unexpected "tricky" questions to P.)

So much for the notion of proof in a randomized and interactive setting (indeed there would be much more to say). We now have to formalize the notion of zero knowledge.

Definition 6.2 (Weak Perfect Zero Knowledge) *We say that a proof system (P, V) for a language L is* weak perfect zero-knowledge *if there is a polynomial time randomized algorithm Sim such that for every $x \in L$ the output of $Sim(x)$ and the transcript of the communication between $P(x)$ and $V(x)$ are identical distributions.*

The above definition is saying that when P proves to V that $x \in L$, the communication between P and V is not giving to V any useful knowledge: he could have generated by himself the (distribution of the) messages that have been exchanged.

In the password example (which is not a special case of zero knowledge, but a similar and suggestive framework) the prover is the user who wants to login, the verifier is the remote computer, and the transcript of the communication is what a passive adversary can see. The completeness property says that a legitimate user gains access to the remote computer, the soundness property

says that a illegitimate user does not gain access to the remote computer, and a weak perfect zero knowledge requirement says that the adversary gains nothing in tapping the interaction between the user and the remote computer.

In the password protocol analogy, we are also interested in security against an adversary who is impersonating the computer the user is trying to login into. The corresponding problem of interest in zero knowledge is to be able to guarantee that the prover does not leak any information even to a cheating verifier who is sending messages according to a distribution that is not the same as in the secure protocol. In the strong definition of zero knowledge, the exchange of messages can be simulated even if the verifier deviates from the protocol.

Definition 6.3 (Perfect Zero Knowledge) *We say that a proof system (P, V) for a language L is* perfect zero-knowledge *if for every polynomial time communicating process V' there is a polynomial time randomized algorithm $Sim_{V'}$ such that for every $x \in L$ the output of $Sim_{V'}(x)$ and the transcript of the communication between $P(x)$ and $V'(x)$ are identical distributions.*

As a matter of fact, it is too conservative to insist that the distributions be the same. It serves the same purpose (and gives much more flexibility) to just assume they are indistinguishable.

Definition 6.4 (Weak Computational Zero Knowledge) *We say that a proof system (P, V) for a language L is* weak computational zero-knowledge *if there is a polynomial time randomized algorithm Sim such that for every $x \in L$ the output of $Sim(x)$ and the transcript of the communication between $P(x)$ and $V(x)$ are computationally indistinguishable.*

Definition 6.5 (Computational Zero Knowledge) *We say that a proof system (P, V) for a language L is* computational zero-knowledge *if for every polynomial time communicating process V' there is a polynomial time randomized algorithm $Sim_{V'}$ such that for every $x \in L$ the output of $Sim_{V'}(x)$ and the transcript of the communication between $P(x)$ and $V'(x)$ are computationally indistinguishable.*

6.3 Graph Isomorphism and Non-Isomorphism

A graph G is defined by a set of "vertices" V and by a set E of pairs of elements of V. The elements of E are also called edges.

Two graphs are *isomorphic* if they are the same up to a renaming of the vertices. Formally two graphs $G = (V, E)$ and $G' = (V, E')$ are isomorphic if there is a permutation $\pi : V \to V$ such that $(u, v) \in E$ iff $(\pi(u), \pi(v)) \in E'$. There is no known polynomial time algorithm to test whether two graphs are isomorphic. Furthermore there is no known way to certify that two given graphs are *not* isomorphic. Of course there is a simple way to certify that two given graphs are isomorphic: just give the permutation π.

If $G = (V, E)$ is a graph and $\pi : V \to V$ is a permutation, we denote by $\pi(G) = (V, \pi(E))$ the graph such that $(\pi(u), \pi(v)) \in \pi(E)$ iff $(u, v) \in E$.

6.3.1 A Weak Perfect Zero Knowledge Protocol for Graph Non-Isomorphism

In this section we show that there is a weak perfect zero knowledge proof system for graph non-isomorphism.

We first describe a basic protocol that works in two rounds: the verifier sends a message, the prover replies, and the protocol ends. This simplified protocol is such that an incorrect statement

may be accepted with probability up to 1/2. Repeating the protocol several times (with independent random choices each time) will give a negligible probability of error.

The basic protocol follows:

- P and V have as common input two graphs G_0 and G_1.
- V chooses at random one of the graph, G_b with $b \in \{0, 1\}$, applies a random permutation π to its vertices, and sends $\pi(G_b)$ to P.
- P guesses the value of b and tells it to V; the verifier accepts iff the guess was correct.

We now analyze the properties of the protocol

- Completeness. If G_0 and G_1 are not isomorphic, then the prover is always able to answer correctly, because a random permutation of G_0 is isomorphic to G_0 and not to G_1, and vice versa.
- Soundness. If G_0 and G_1 are isomorphic, then a random permutation of G_0 and a random permutation of G_1 have the same distribution. The answer of the prover is thus independent of b, and it is equal to b with probability 1/2.
- Weak Zero Knowledge. The simulator picks at random b and π, writes the corresponding message of V, and writes b as the answer of P.

By repeating the protocol k times the soundness probability becomes 2^{-k}.

The same protocol cannot be simulated in the case of a cheating verifier. It is still possible to give a perfect zero knowledge proof system for graph non-isomorphism, but it is considerably more complicated, and we will not present it here.

6.3.2 A Perfect Zero Knowledge Protocol for Graph Isomorphism

In this section we give a perfect zero knowledge protocol for the graph isomorphism protocol. Again there will be a basic protocol with soundness error 1/2, and then the protocol with negligible soundness error is obtained by doing several independent repetitions of the basic protocol. This time the basic protocol has three rounds: the prover sends a message first, then there is a reply from the verifier and then a last message from the prover.

- Prover P chooses a random permutation π and sends $H = \pi(G_1)$.
- Verifier V picks a random bit b, and sends it to the prover, asking the prover to show that G_b is isomorphic to H.
- The prover supplies the required permutation.

Analysis:

- If G_0 and G_1 are isomorphic, then they are also both isomorphic to H, and the prover can always give a satisfactory answer.
- If G_0 and G_1 are not isomorphic, and the prover is cheating, then no matter what the prover is sending as graph H, we have that either H is not isomorphic to G_0 or H is not isomorphic to G_1 (indeed H could be isomorphic to neither of them). Then there is probability (at least) 1/2 that the prover is unable to give a satisfactory answer to the verifier in the last round.

- Zero knowledge. This time the proof is non-trivial. Consider an arbitrary verifier V'. The simulator starts by choosing at random a graph among G_0 and G_1, that is, he chooses a random $b \in \{0,1\}$ and considers the graph G_b. Then he chooses a random permutation π, and sets $H = \pi(G_b)$. Then it simulates the behavior of V' on input H. If V' generates as an answer b, then the simulator writes b and then π. If not, the simulator erases what he has done to that point and starts again.

The proof of zero knowledge deserves further attention. The simulator works in several attempts. We claim that, at each attempt, he is successful with probability 1/2. Indeed, no matter what was the choice of b, the distribution of H as seen by V' is the same. Therefore the bit computed by V' is *independent* of b, and it equals b with probability 1/2, as claimed. On average two attempts are sufficient to complete the simulation. So the simulator runs in average polynomial time.

6.4 Commitment

In this section we consider a problem whose solution is a primitive used to build zero knowledge proof systems and other useful cryptographic protocols.

6.4.1 Motivations

This has happened sometimes: an astrologer (or psychic of some sort) makes predictions about some future event, closes them in a sealed envelope, and then leaves it to a notary. At some later time he has a press conference where the notary opens the envelope and the prediction is shown. He would normally cheat by making obvious predictions (the US basketball team will win the gold medal at the Olympic games; Dan Quayle will not win the presidential elections (actually, you never know); and so on). But there is another way to cheat: the psychic will write two different predictions (Dan Quayle wins, Dan Quayle does not win; just in case) and put them in two different envelopes to be given to two different notaries. At the press conference the right notary is invited. It would be more fun to let the psychic "commit" to a prediction in advance, in a way that cannot be withdrawn, and also without telling the prediction in advance. We are looking for the cryptographic analog of a sealed envelope that is in the public domain and that only the owner can open.

This looks easy: the guy chooses a private-key public-key pair, and encodes the prediction using the public key (so nobody can invert it), possibly using a semantically secure encryption system. At the right time, he reveals the original message and/or the secret key. This sounds good, but it requires trapdoor permutations. We will see an implementation that only uses one-way functions.

6.4.2 Definition

We need to describe two (interactive) protocols, involving a *sender* S and a *verifier* V.

- The **commitment** protocol. At the beginning S has some information a. At the end of this protocol the verifier stores his record v of what he has seen during the interaction.
- The **disclosure** protocol. At the beginning V has is record v of his interaction with S. At the end he knows the value of a.

The properties are that even if S cheats in the disclosure phase, he will not be able to convince V in the disclosure phase that the value represented in v is different from a. Furthermore, even if V cheats in the commitment phase, he will not be able to get the value of a (better than guessing at random).

6.4.3 Construction and Its Analysis

We first consider the case where the information of the sender is just a single bit b. Our first construction uses a one-way permutation $f : \{0,1\}^k \to \{0,1\}^k$.

Commitment:

- S picks a random $x \in \{0,1\}^k$ and a random $r \in \{0,1\}^k$ such that $\sum_i x_i r_i \pmod 2 = b$. Then S sends $(f(x), r)$.

Disclosure:

- S discloses x. Then V, given his record (y, r), can verify that $f(x) = y$ and that $\sum_i x_i r_i = b$.

Regarding the correctness of the protocol, it is impossible for S to cheat, because, since f is a permutation, only the right disclosure would convince the verifier. On the other hand, the probability that the verifier can guess the value of b after the commitment phase is only $1/2 + \nu(k)$, where $\nu()$ is a negligible function. This is because b is the Goldreich-Levin hard core bit of a one-way permutation.

We now see a commitment protocol based on the existence of a pseudorandom generator $G : \{0,1\}^k \to \{0,1\}^{3k}$. We have only seen how to construct pseudorandom generators based on the existence of one-way permutations (and using Goldreich-Levin), so there seems to be no advantage over the previous construction. Indeed, it has been shown that one can construct pseudorandom generators based only on the assumption that one-way functions exist, which is a weaker assumption that assuming that one-way permutations exist.

Commitment:

- V sends a random string r
- S picks a random x, then if $b = 0$ he sends $G(x)$, otherwise he sends $G(x) \oplus r$.

Disclosure

- S discloses x. Then the verifier, given his previous record (r, y), can see whether $y = G(x)$ or $y = r \oplus G(x)$.

Now, even if V cheats by sending a non-random string in the first round, he receives a string that is indistinguishable from uniform, regardless the value of r and b, so V cannot get a hint to the value of b by cheating during the commitment phase.

Let us consider the issue of S trying to cheat during the disclosure phase. Let us define $f(x, r, b)$ the reply of S when the secret is b, the verifier's message is r and the sender's random choice is x (that is, $f(x, r, b) = G(x)$ if $b = 0$ and $f(x, r, b) = G(x) \oplus r$ otherwise.) The only way the sender can possibly cheat is that, for the r that was sent by V, there are x, x' such that $f(x, r, 0) = f(x', r, 1)$. Let's say that, if so, (x, x') *fools* r, and that r is *non-safe*. Note that in order to fool two different strings we need two different pairs, and that we have at most $(2^k)^2 = 2^{2k}$ pairs, and so at most 2^{2k} of the 2^{3k} possible strings r are non-safe. So the probability of cheating is at most 2^{-k}.

If we want to commit to a a message $a = (b_1, \ldots, b_l) \in \{0,1\}^l$ then we can apply the commitment protocol to each b_i. For example, using the first approach, the sender chooses l pairs (x^i, r^i) where $x^i \in \{0,1\}^k$ are independent random strings and r^i are independent random strings such that for every i we have $\sum_j r^i_j x^i_j = b_i \pmod 2$. Then the commitment is $(f(x^1), r^1, \ldots, f(x^l), r^l)$ and

the disclosure is $(x^1, \ldots, x^l)$. The security of this composed protocol is based on the security of the base protocol and on a hybrid argument. We do not give the details. In a similar way one can extend the pseudorandom generator protocol to the case of several bits. There are also more efficient commitment protocols for the case of several bits.

6.4.4 Application

This is a nice application of commitment schemes: coin flipping over the telephone. Two people, say Alice and Bob, are over the phone and want to agree to a fair coin flip (maybe they are coauthors of a paper and want to decide who is going to give the talk[3]). If they distrust each other, this looks like a difficult problem: it's not like one of them can flip the coin and announce the result to the other.

Using commitment, here is a protocol: Alice flips her coin, and runs a commitment protocol with Bob to commit to the value of the coin. Bob, then, also flips a coin and announces the result to Alice. Then Alice discloses the commitment, and the xor of Alice's and Bob's coins is the final answer.

Suppose Alice is trying to cheat: then she will commit to a certain value (that she will not be able to change later) and then this value will be xor-ed with the fair coin flipped by Bob. The result will be a fair coin. If Bob tries to cheat, when he has to declare his coin flip he has no clue what Alice's coin is, and so the value of Alice's coin is (almost) independent of the value of Bob's. When the two values are xor-ed, the result is a fair coin. Of course if they *both cheat* then the result might be whatever, but this is true for every possible protocol.

6.5 Every Possible "Theorem"

In this section we show that if a statement has a short and easily checkable certificate (a "proof" in the standard sense) then there is a computational zero knowledge proof for the statement, with the additional feature that the prover can be efficiently implemented given the certificate.

We will show such a protocol for one particular problem: given a graph prove that it is 3-colorable. We shall briefly argue that if we can prove such kind of statements, then we can prove whatever we want.

6.5.1 Graph 3-Colorability and Statements with Short Proofs

A graph $G = (V, E)$ is 3-colorable if there is a way to assign to every vertex a color chosen from the set $\{Red, Green, Blue\}$ such that for each edge $(u, v) \in E$, u and v have been assigned different colors.

For example a path is 3-colorable (it is also 2-colorable) but a graph of four vertices with all the possible connections is not 3-colorable.

It is possible to give a *short certificate* that a graph is 3-colorable: just give the assignment of colors to the vertices. We have been using throughout these notes the notion of "proof in the standard sense" or of "certificates," it is time to give a rigorous definition.

Definition 6.6 (NP) *A language L belongs to the class NP if there exists a deterministic algorithm $V(\cdot, \cdot)$ running in time polynomial in the length of its first input such that for every x:*

[3]The example is unfair to scientists: they normally *trust* each other, and they would not need a cryptographic protocol to do the choice. Besides, the junior author would give the talk ...

- Completeness. *If $x \in L$ then there exists a "proof" π such that $V(x, \pi)$ accepts.*
- Soundness. *If $x \notin L$ then there is no proof π such that $V(x, \pi)$.*

Recall that a language stands for a set of true statement (in a certain context, and according to a certain representation). When a language is in NP, every statement of the form "$x \in L$" has a proof in the standard sense of the word: a short written document whose correctness can be efficiently tested. Algorithm V in the definition formalized the notion of efficient testing.

Note that V runs in time polynomial in the length of x, which means that when a proof exists that $x \in L$, the proof will be of length polynomial in the length of x (otherwise V would not even have time to read it).

NP stands for "non-deterministic polynomial time." This name is motivated by an equivalent definition of NP in terms of "non-deterministic Turing machines." We will not give this alternative definition.

The following theorem states that the problem of finding 3-colorings of graphs is "more general" than any problem in NP. To get a sense of what the Theorem is saying, try to substitute a concrete problem in place of L.

Theorem 6.7 *Let L be a language in NP, and $V(\cdot, \cdot)$ be the algorithm as in the definition of NP. Then there exists an polynomial time algorithm that on input an arbitrary x produces a graph G_x with the following properties:*

- *If $x \in L$, then G_x is 3-colorable, furthermore a 3-coloring of G_x can be efficiently computed from a π such that $V(x, \pi)$ accepts.*
- *If G_x is 3-colorable, then $x \in L$, furthermore a π such that $V(x, \pi)$ accepts can be efficiently computed from a 3-coloring of G_x.*

The theorem above is a more specific way of stating that 3-coloring is *NP-complete*. A proof (or even a discussion of the implications) is beyond the scope of these notes.

The following result is more or less a consequence of Theorem 6.8. The proof is not completely straightforward because there are certain technical details about the simulation that one has to treat carefully; see also Proposition 4.4.9 in [Gol95] and its proof.

Theorem 6.8 *Suppose that 3-coloring has a computational zero knowledge proof system such that the prover is implementable in polynomial time given the 3-coloring. Then every language in NP has a computational zero knowledge proof system such that the prover is implementable in polynomial time given a standard proof.*

The idea is to consider an arbitrary language L and the proof system (P, V) for graph 3-coloring. For an arbitrary verifier V^* we denote by Sim_{V^*} the simulator such that for every graph G the distribution of outputs of $Sim_{V^*}(G)$ and the distribution of interaction between $P(G)$ and $V(G)$ are equal. Then we can give a proof system (P_L, V_L) for L as follows: on common input x, P_L and V_L simulate the behavior of P and V, respectively, on common input G_x. The completeness, soundness, and efficiency conditions on (P_L, V_L) are implied by the completeness, soundness and efficiency conditions of (P, V) and by Theorem 6.8 above. For an arbitray verifier V' we should now show that the interaction between $P_L(x)$ and $V'(x)$ can be simulated. This is the non-trivial part. Let us denote by V'' the verifier than on input a graph G reconstructs the x such that $G = G_x$, and aborts if no such x exists. Even if the efficient implementation of such a procedure

is not apparent from the statement of Theorem 6.8, such "inversion" can indeed be efficiently implemented. We can see that from our definitions it follows that the distribution of outputs of $S_{V''}(G_x)$ and the distribution of interactions between $P(G_x)$ and $V''(G_x)$ are indentical. In turn, the latter distribution is identical to the distribution of interactions between $P_L(x)$ and $V'(x)$.

6.5.2 How to Prove that a Graph is 3-Colorable

Note: this section is really sketchy.

Construction:

- The common input of prover P and verifier V is a graph G. Prover P also has a 3-coloring of G.
- P randomly permutes the colors of his 3-coloring. For every vertex, P commits to its color.
- V chooses at random an edge, and asks the prover to reveal the color of its two endpoints.
- P discloses the two colors. V accepts iff they are different.

Analysis:

- Completeness is clear.
- Suppose the graph is not 3-colorable. Then there must be an edge such that the commitments of the colors of the endpoints are not different. Such an edge will be chosen with probability $1/|E|$. Assuming that the commitments are done using one-way permutations, there is a probability of error $1 - 1/|E|$.
- The simulator for verifier V' chooses a random color for every edge, and commits to this coloring. Then the commitments of the colors are written down. Verifier V' is simulated on input such commitments. If the verifier chooses an edge with two different colors at the endpoints, the simulator discloses the colors and terminates the simulation. If not, the simulator starts again.

Some notes:

- The probability of error can go from $1 - 1/|E|$ to negligible by making sufficiently many repetitions. In particular, if the protocol is repeated k times, then the probability of error is $\leq (1 - 1/|E|)^k < e^{-k/|E|}$ which is negligible if, e.g., we set $k = |E|^2$.
- The simulator, at the first round, produces a distribution of messages that is not the same as the distribution of messages produced by the prover. Yet, the two distributions are computationally indistinguishable since otherwise the verifier would be able to distinguish (with non-negligible probability) the commitment to certain colors from the commitment to other colors, contradicting the definition of commitment scheme. After the first round, the simulator perfectly simulates the protocol.
- We have to make sure that each attempt of the simulator to simulate the interaction has a reasonably good probability of success. Suppose, first, the verifier chooses the edge that he sends in the second round without considering the message received in the first round. Then the choice of the verifier is independent of the coloring chosen by the simulator, and

with probability 2/3 the edge chosen by the verifier is an edge whose colors are different. So on average the simulator has to make only 1.5 attempts to complete the simulation. The problem is that the verifier can look at the first round of communication before choosing his edge. But what the verifier sees is a distribution indistinguishable from uniform, and so what the verifier chooses is almost independent of the coloring chosen by the simulator. The probability of success of one attempt of simulation is therefore at least $2/3 - \nu(k)$ where k is the number of nodes of the graph and k is a negligible function.

6.6 Credits

These notes follow the treatment of Goldreich [Gol95].

The notion of interactive proof system and of zero knowledge were introduced by Goldwasser, Micali and Rackoff [GMR89].

The protocols for graph isomorphism and non-isomorphism, as well as the general result about computational zero knowledge are due to Goldreich, Micali and Wigderson [GMW91].

Chapter 7

Digital Cash

This chapter considerable less formal than the other ones. On the one hand a formal treatment of a protocol with (at least) three mutually distrusting parties and a complicated definition of "security" is very hard (though not impossible); on the other hand the field has not stabilized to the point of providing: (i) a clean definition, (ii) an elegant theoretical solution, (iii) good practical solutions meeting the definition (as in the case, say, of public-key encryption).

7.1 Introduction

In order for the internet to become a profitable commercial medium, it is necessary to have a way for *payments* to be made between parties that communicate through a (insecure) network.

The need for payment protocols arise in "electronic stores" that sell books, videos, computers, up to stocks, cars and real estates online. Payments also arise in pay-per-use access to news, music files, computer programs, and other non-free digital goods.

Payments in real life are mostly done via cash, debit cards, personal checks and credit cards. Electronic payment systems borrow metaphors (and infrastructures) from some of these systems.

In payment systems we have three entities: a customer Alice, a Bank and a merchant. Alice and the merchant have accounts in the Bank. Eventually, a purchase done by Alice should result in a transfer of money from Alice's account to the Merchant's account.

Let us see high level descriptions of cash, debit cards and checks.

Cash. Cash is made by physical objects (bills) that are supposedly unforgeable.

- Alice withdraws cash from the Bank. She is required to identify herself (e.g. by swiping her ATM card and typing a password).
- Alice gives cash to the merchant.
- The merchant deposits his revenues at the bank.

The main advantages are that at any stage (withdrawal, purchase, deposit) only two of the parties have to interact, and that the purchase can be done without the need of a bank authorization. The unforgeability relies on a physical assumption.

Debit Card. Each user having an account in the Bank is given a card and a PIN.

- At the moment of the purchase, Alice swipes the card in the merchant's machine and types her password.

- This information, together with the amount of the purchase, is sent to the Bank that transfers money from Alice's account to the merchant's account.

An access to the Bank is needed for each transaction. Alice must have complete trust in the honesty of the merchant.

Personal Check. Each user having an account in the Bank is given a booklet with checks. Each check shows identifying information.

- At the moment of the purchase, Alice identifies herself as the owner of the account the check refers to, and signs the check for a certain amount of money.
- Later the merchant deposits the check on his account.

Alice needs not trust the merchant, since the amount of money written on the check cannot be changed. The merchant needs to partially trust Alice. If the check is not covered, Alice is identified and can be sued, but money is lost.

Electronic payments are mostly done via credit cards. When credit cards are used over a network, the protocol is similar to a debit card in implementation (the bank has to authorize the transaction) and features (the merchant has to know all identifying information, and has to be trusted).

The use of cryptographic techniques can make credit card transactions over the net reasonably secure. The user gets a receipt from the merchant for the payment, the merchant gets a receipt from the Bank for the deposit, the merchant cannot charge more than the user understands she is paying, the user cannot pay less than what the merchant understands he is being payed, an adversary tapping the communication, or breaking actively into the system, cannot make purchases impersonating users, or trick users into believe he (the adversary) is a merchants, and so on.

The credit card model of payments has a strong inconvenience: the user has to disclose her identity to the merchant, and the Bank can keep a record of all the purchases of all the clients.

This collection of data about people has several undesirable effects. The Bank can infer several personal information about the client, looking at her purchase, including info on medical conditions (meets with a psychiatrist, buys AZT) hobbies (practices rafting and bungee jumping), habits (had lots of speeding tickets, buyed a sport car), and use this information against the user when she applies for a life insurance, a loan or a job. From patterns in purchases one can often infer ethnicity, disabilities, religion, and other characteristic that a person may not want her Bank (and, potentially, junk mail senders) to know.

When a user pays in cash, she does not have to disclose her identity. We would like to design a system that guarantees anonymity for the user, and that be also secure against any type of dishonest behavior (the way cryptographic credit card-based systems are).

7.2 Definitions

A digital cash system consists of three protocols, a *withdrawal* protocol, a *spending* protocol and a *deposit* protocol. Each protocol involves (a subset of) three entities: the user (Alice), the merchant and the Bank. It is assumed that there is a single Bank and several users and merchants. It is also assumed that the user communicates to the merchant via an *anonymous channel*. An anonymous

channel is a mean of communication that does not disclose (even partial) information about the user. It could be implemented via a re-mailer system or anonymizing web browsing system.[1]

The three protocols are the digital implementation of Alice getting money from an ATM and purchasing some goods, and then the merchant depositing his revenues. In an off-line system the three protocols are as follows:

- Withdrawal. This is a protocol involving Alice and the Bank. At the end of the protocol Alice has some information C, that is a, say, \$1 digital coin, and the Bank removes \$1 from Alice's account.
- Spending. This is a protocol between Alice and the merchant. Alice uses C, and at the end of the protocol the merchant has some information C'.
- Deposit. This is a protocol between the merchant and the Bank. The merchant uses C' and at the end he has \$1 more on his account.

In an on-line system, the Spending and the Deposit protocols are merged into a single protocol between Alice, the merchant and the Bank. So a transaction between Alice and the merchant cannot be completed without connecting to the Bank.

The three protocols need to satisfy the following informal properties:

Anonymity. The merchant does not get any information about Alice's identity (this is the requirement where we we need to assume that Alice talks to the merchant through an "anonymous channel").

This condition should be guaranteed even if the Bank and the merchant collaborate to trace Alice's purchases.

No Double Spending. Neither Alice nor the merchant can spend the same electronic coin twice. The following condition is a generalization of this one.

No forging. If Alice withdraws k coins, she cannot spend $k+1$ successfully. If the merchant is involved in k spending protocols, he cannot deposit $k+1$ coins successfully.

It is *impossible* to avoid double spending from the user in a off-line system. We will have to settle for a system similar to personal checks: the user can cheat, but merchant and Bank end up having legal evidence against her if she tries.

It is difficult to integrate such feature in an anonymous system. We want to make sure that if Alice is honest, her identity is impossible to trace through her purchases. On the other hand, if she tries to double-spend an electronic coin, the records of the Bank and of the merchant will reveal her identity, and will be evidence against her (evidence that would convince a fourth party). The difficulty is in how the identity of Alice can be kept secret in ordinary transactions but revealed if she cheats. This is described in Section 7.5.

The main difficult is indeed in how to achieve anonymity: in order for the Bank to distinguish good coins from forged ones (let alone disclosing the identity of the forger), the Bank needs to put some sort of signature on the good coins during the withdrawal phase. But Alice cannot let the Bank know the coin that she is withdrawing, otherwise her anonymity is compromised. So,

[1]Note that indistinguishability of users is the only assumption that we make about the channel: adversaries may tap communication and, by the very nature of the system, inject messages of their choice.

somehow, the Bank has to sign a coin that it cannot see. This sounds impossible[2] but it can be achieved, as explained in Section 7.3.

7.3 Blind Signatures

A blind signature scheme is a protocol involving user Alice and a *signer* (in the application to electronic cash the singer will be the Bank). Using such protocol, Alice gets a signature from the signer without the signer what the message is. (Think of the metaphor of a person signing a document while blindfolded.)

Conventionally, in a signature system the user sends a message to the signer and the signer produces a signature for the message. The signature is convincing for the user as well as for any third party.

In a blind signature scheme, the user and the signer interact and, at the end of the interaction, the user gets a valid signature from the signer. Yet, the signer's view of the interaction is independent of the message.

7.3.1 Basic Construction

The basic construction is due to Chaum, and uses RSA. The version that we describe in this section is a simplified (and clearly insecure) version. We will see a better construction in the next section.

The signer picks the private key/public key pair as in RSA: the public key is a pair (e, n) where $n = pq$ and p, q are big primes and $\gcd(e, \phi(n)) = 1$; the private key is (d, n) where $de = 1 \pmod{\phi(n)}$.

Recall that in the plain RSA signature scheme the signature for a message $m \in \mathbf{Z}^*{}_n$ is $m^d \pmod n$. We will see how Alice can get from the signer the value $m^d \pmod n$ without giving away m.

- Alice wants to get a signature for a message $m \in \mathbf{Z}^*{}_n$. She picks a random $r \in \mathbf{Z}^*{}_n$ and then computes $y = r^e m$.
- The signer receives y and outputs $s = y^d \pmod n$.
- Alice receives $s = y^d = r^{ed} m^d = r m^d \pmod n$, computes r' such that $rr' = 1 \pmod n$ and then computes $sr' = m^d \pmod n$.

In this way, Alice gets the signature of the message she is interested in.

Let us consider the point of view of the signer. For a fixed message m, the signer receives $r^e m$ where r is random. Since the mapping from r to r^e is a permutation (it is RSA encryption), it follows that if r is random and uniform then also r^e is random and uniform. Then, since $\mathbf{Z}^*{}_n$ is a group, we also have that if r^e is random and uniform then also $r^e m$ is random and uniform. Thus the signer receives a message that is random and independent of m, so he has no information on m (in a very strong sense).

The only drawback of this system is that it is not robust against forgery, in the same way plain RSA is not. In fact, blind signatures are *based on* the fact that RSA is not existentially unforgeable: otherwise it would be impossible for Alice to obtain the signature of m without letting the signer know m.

[2] but several cryptographic protocols sound impossible, when you hear about them for the first time, like semantically secure encryption or zero knowledge proofs.

7.3.2 Secure Construction

Chaum's method can be made resistant to forgery in the same way plain RSA can: by hashing the message with an "ideal hash function."

Let h be a hash function that behaves "like a random function" and that maps from the set of possible messages to the domain of a RSA permutation. The Bellare-Rogaway way[3] of signing using RSA and a hash function is to sign m as $(h(m))^d \pmod n$. The application of the same approach to blind signatures follows.

- Alice wants to get a signature for a message $m \in \mathbf{Z}^*{}_n$. She picks a random $r \in \mathbf{Z}^*{}_n$ and then computes $y = r^e h(m)$.
- The signer receives y and outputs $s = y^d \pmod n$.
- Alice receives $s = y^d = r^{ed}(h(m))^d = r(h(m))^d \pmod n$, computes r' such that $rr' = 1 \pmod n$ and then computes $sr' = (h(m))^d \pmod n$.

The same argument as before proves that the signer sees a random message independently of m and h.

An analysis similar to that of Bellare and Rogaway essentially rules out any possible forging.

We have not given a formal definition of forging for blind signatures: informally, a scheme is (S, q, ϵ)-secure if an adversary that runs in total time S (not including the time it has to wait for an answer during the protocol) and that can get q signatures for arbitrary messages of his choice (the adversary may not follow the blind signature protocol, he may choose, adaptively, a sequence of q arbitrary values $y_1, \ldots, y_q$ and he gets back $y_1^d \pmod n, \ldots, y_q^d \pmod n$), there is a probability $\leq \epsilon$ that he is able to compute $q + 1$ valid signatures. In the asymptotic definition, S and q are replaced by any polynomial function in the length of the message, and ϵ is replaced by a negligible function in the length of the message.

7.4 Basic Protocol

We show how blind signatures give a simple *on-line* protocol, i.e. a protocol where the spending and the deposit protocol have to be executed simultaneously.

7.4.1 Description

- Withdrawal. Alice generates a random message m (can be thought of as a unique serial number for the digital bill to be withdrawn). Alice uses a blind signature protocol to get a signature σ of m from the Bank.
- Spending-Deposit. Alice gives (m, σ) to the merchant. The merchant forwards it to the Bank. The Bank confirms that the coin is good (the serial number was never spent before), and puts it on the merchant's account.

[3]The approach of hashing the message and then applying the inverse of a trapdoor permutation was not introduced by Bellare and Rogaway. They just analyzed it assuming the hash function is "ideal," i.e. completely random.

7.4.2 Analysis

- Unforgeability. Since the signature system is unforgeable, Alice cannot spend more than what she has withdrawn. Similarly the merchant cannot deposit more coins that the ones he has received.
- No Double Spending. The same coin cannot be spent twice, because the Bank checks the serial number.
- Anonymity. During each withdrawal protocol, the Bank only sees a random message, independent of the coin being signed. During the deposit-spending phase, the merchant and the Bank see a randomly generated coin, independent of the user. So it is impossible to gain any information about which user performed which transaction.

7.4.3 Drawbacks

This system has obvious drawbacks, the main one being the necessity of a performing Spending and Deposit simultaneously. Another minor drawback is the fact that all electronic coins come in the same denomination. It would be better to allow \$1 electronic coins, \$10 ones, and so on. Otherwise an exceedingly large number of coins have to be transmitted and verified for a real estate transaction (unless the granularity of payable amounts is very rough). This is achievable by letting the bank use different keys for different denominations. An alternative way is presented in next section.

7.5 Improvements

7.5.1 Coins of Different Denomination

Let us consider first the problem of having coins of different denominations.[4]

The idea is that a coin, instead of being just a signed random string, be a signed pair $((v, m), \sigma)$ where v is the denomination, m is the serial number of the coin, and σ is the signature of (v, m). Then Alice would declare m and do the blind signature protocol for (v, m) (for a random m). This is not so good for the Bank: Alice may claim that she is doing the blind signature for a \$1 coin while instead she is setting $v = \$1,000$. It is a feature of the blind signature scheme that the Bank does not get any information about the coin, not even its denomination.

The following solution, called cut-and-choose, has been proposed. We only describe the withdrawal protocol (the other is the same); in the description, k is a parameter of the construction and (e, d, n) are the parameter of an RSA permutation.

- Alice wants a coin of value v. She generates at random k serial numbers $m_1, \ldots, m_k$ and k random numbers $r_1, \ldots, r_k$. She computes $(y_1, \ldots, y_k) = (r_1^e h(v, m_1), \ldots, r_k^e h(v, m_k))$, where all operations are $\pmod n$, and sends $(y_1, \ldots, y_k)$ to the signer; she also sends v.
- The signer chooses at random an index i between 1 and k, and asks Alice to disclose the values of r_j and m_j for $j \neq i$.

[4]The use of different denominations is a slight harm to anonymity. If Alice is the only one asking for a \$193 denomination, whenever a \$193 is spent Alice is identified. Our aim in defining provably secure digital cash protocols with variable denomination is that when a coin of a certain denomination is being spent, the Bank and the merchant do not get any additional information on the person doing the purchase, *except* the fact that she is one of those who withdrew a coin of that particular denomination.

- Alice sends the required values.
- The signer checks that for every $j \neq i$ actually $y_j = r_j^e h(v, m_j) \pmod n$. If everything checks, the signer sends $s = (r_i y_i)^d \pmod n$.
- Alice reconstructs from s the signature $(h(v, m_i)^d \pmod n$ of the coin (v, m_i).

If Alice follows the protocol, she gets a signature for a coin of the required denomination. Furthermore the Bank gets no information about the coin (it only gets information about independently chosen random values).

Suppose that Alice tries to cheat by declaring a certain value v and having a different value of v in one of the coins. For example suppose that for some index j we have $y_j = r_j h(v', m_j)$ where v' is bigger than v. With probability $\geq 1 - 1/k$ we have that j is not the index i chosen by the Bank. Therefore Alice, to conceal her attempt of cheating, must provide r', m' such that $r' h(v, m') = r_i h(v', m_i)$. This is possible only with negligible probability when h is a random function. Putting the calculations together, the probability that Alice can succeed in her attempt of cheating is only $1/k$, the probability that she does not succeed but she is not caught in her attempt is negligible, the probability that she is caught is $1 - 1/k$ minus a negligible factor.

This system has the advantage of simplicity. The disadvantage is the fact that a much larger amount of communication and storage is requested. Besides, the probability of breaking the system decreases linearly (instead of exponentially, or as a negligible function) in the amount of additional storage.

A (theoretically) better way is to have Alice submit to the Bank only one message $y = r^e h(v, m)$ and v, and then prove in zero knowledge that there exists values m, r (that she knows) such that $y = r^e h(v, m)$. Using a zero knowledge protocol, the probability that Alice can convince the signer that the coin is for value v' while instead it was for $v \neq v'$ is only negligible in the length of the message.

7.5.2 Tracing Double-Spenders

In order to detect double spending, we use an approach that is similar to that used to have different denomination. We let Alice put some additional information in the coin, and then let her convince the signer the he is signing a coin that includes the required additional information. The additional information will be chosen so that it is of no use in tracing Alice if she is honest, but it discloses her identity if she is not honest. In the following, we need two "ideal" hash functions. One, call it h, is used in the signing process as before. The other, call it H has a different purpose.

When creating a coin, besides the serial number m (and the denomination v) Alice also generates l random pairs of strings $(x_1, z_1), \ldots, (x_l, z_l)$ with the property that for every i, we have that $x_i \oplus z_i$ discloses the identity of Alice (e.g. it is her Bank account number). A valid coin is of the form

$$(v, m, H(x_1), \ldots, H(x_l), H(z_1), \ldots, H(z_l))$$

together with its signature

$$(h(v, m, H(x_1), \ldots, H(x_l), H(z_1), \ldots, H(z_l)))^d \pmod n$$

We'll see how such a format helps separating spending and deposit; we first see how withdrawal can be implemented. The idea is again cut-and-choose.

Withdrawal:

- Alice generates k coins of value v, with independent choices of serial numbers m_i and additional values x_j^i and z_j^i. Alice also generates k random values r_i. Alice sends to the signer k messages $y_1, \ldots, y_k$ where

$$y_i = r_i^e h(v, m_i, H(x_1^i), \ldots, H(x_l^i), H(z_1^i), \ldots, H(z_l^i)) \pmod{n}$$

- The Bank picks a particular value of i, and asks Alice to reveal all her random choices used in the construction of y_j, for $j \neq i$. If Alice's disclosure is consistent, and the additional strings x_j, z_j have been chosen as prescribed, the signer signs y_i.
- Alice gets the signature of one valid coin as usual.

Here Alice can still cheat with probability $1/k$. Using zero knowledge the probability of Alice cheating can be reduced to a negligible factor.

Spending:

- Alice submits a coin

$$c = (v, m, H(x_1), \ldots, H(x_l), H(z_1), \ldots, H(z_l))$$

  together with its signature $c^d \pmod{n}$.
- The merchant checks the signature, then picks a random bit string $(b_1, \ldots, b_l) \in \{0,1\}^l$, he sends the string to Alice.
- For any i, if $b_i = 0$ Alice sends x_i to the merchant, otherwise she sends z_i.
- The merchant applies H to what Alice sends back, and checks that Alice's answers are consistent.

The idea is that when the transaction is successfully completed, the merchant has a coin, its signature, and some additional information from which it is impossible to recover information about Alice. However, the additional information can be used, in association with the Bank, to uncover Alice's identity in case she double-spent the coin.

Deposit:

- The merchant submits the coin

$$c = (v, m, H(x_1), \ldots, H(x_l), H(z_1), \ldots, H(z_l))$$

  together with its signature $c^d \pmod{n}$ and the xs and the zs disclosed by Alice.
- The Bank verifier the signature and checks that the serial number m is new.

What happens if the serial number is not new? First of all, except with negligible probability, the rest of the coin is also identical, since the signature is unforgeable. If the merchant is honest then, unless with exponentially small probability, the choice of $(b_1, \ldots, b_l)$ was different the other time the same coin was spent. So the bank, using its record and the new information, has some value i for which both x_i and y_i are known. This information discloses the name of Alice. Furthermore, the randomness of h is such that it is impossible for the merchant to "fabricate" evidence against Alice. If the values of $(b_1, \ldots, b_l)$ are equal both times, we can assume that what happens is that the merchant is trying to deposit the same coin twice.

7.6 Credits

These notes are based partly on Tsiounis' Ph.D. dissertation [Tsi97] and on the Goldwasser-Bellare lecture notes [GB97].

The idea of blind signatures and its application to electronic cash are due to Chaum [Cha83]

Bibliography

[Cha83] D. Chaum. Blind signatures for untraceable payments. In *Crypto'82*, pages 199–203, 1983.

[GB97] S. Goldwasser and M. Bellare. Lecture notes on cryptography. Unpublished Notes, 1997.

[GM84] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984. Preliminary Version in *Proc. of STOC'82*.

[GMR89] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989. Preliminary version in *Proc of STOC'85*.

[GMW91] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity. *Journal of the ACM*, 38(3), 1991.

[Gol95] O. Goldreich. Foundations of cryptography. Book in preparation, 1995.

[Tsi97] Y.S. Tsiounis. *Efficient Electronic Cash: New Notions and Techniques*. PhD thesis, Northeastern University, 1997.