Foundations of Cryptography
89-856

Yehuda Lindell
Dept. of Computer Science
Bar-Ilan University, Israel.
lindell@cs.biu.ac.il

April 26, 2010

Abstract and Course Syllabus

Abstract

In this course, we will study the *theoretical foundations* of modern cryptography. The focus of the course is to understand what cryptographic problems can be solved, and under what assumptions. Most of the course will follow the presentation of the relevant material in Oded Goldreich's books on the foundations of cryptography [5, 6]. The course obligations include exercises and a final exam. In addition, there will be reading assignments on important material that we will not have time to cover in class.

Course Syllabus

1. (a) **Introduction and background:** a rigorous approach to cryptography, the focus of the foundations of cryptography, background on the computational model

   (b) **One-way functions I:** definitions of strong and weak one-way functions, candidates

2. **One-way functions II:** strong versus weak one-way functions, definitions of collections of one-way functions and trapdoor permutations, definition of hard-core predicates, preliminaries for Goldreich-Levin

3. **Hard-core predicates:** proof of existence (the Goldreich-Levin hardcore predicate).

4. **Computational indistinguishability and pseudorandomness:** definition of computational indistinguishability, multiple sample theorem, definition of pseudorandomness, definition and construction of pseudorandom generators, extending the expansion factor of pseudorandom generators

5. **Pseudorandomness II:** definition of pseudorandom functions, construction of pseudorandom functions from pseudorandom generators

6. **Zero knowledge I:** motivation, interactive proofs - definitions, perfect zero-knowledge proof for Diffie-Hellman tuples

7. **Zero knowledge II:** commitment schemes, zero-knowledge proofs for all languages in $\mathcal{NP}$

8. **Zero knowledge III:** proofs of knowledge, non-interactive zero-knowledge *(may be skipped)*

9. **Encryption schemes I:** definitions – indistinguishability, semantic security and their equivalence, security under multiple encryptions.

10. **Encryption schemes II:** constructions of secure private-key and public-key encryption schemes; definitions of security for more powerful adversaries

11. **Digital signatures I:** definitions, constructions
12. **Digital signatures II:** constructions, constructions of hash functions
13. **Secure multiparty computation:** motivation, definitions, semi-honest oblivious transfer, the GMW construction

A word about references. We do not provide full references for all of the material that we present. To make things worse, our choice of what to cite and what not to cite is arbitrary. More complete citations can be found in [5] and [6] in the "historical notes" section at the end of each chapter.

Course Text Books

1. O. Goldreich. *Foundations of Cryptography: Volume 1 – Basic Tools.* Cambridge University Press, 2001.
2. O. Goldreich. *Foundations of Cryptography: Volume 2 – Basic Applications.* Cambridge University Press, 2004.

Contents

Lecture 1

Introduction and One-Way Functions

1.1 Introduction

In this course, we will study the theoretical foundations of cryptography. The main questions we will ask are *what cryptographic problems can be solved* and *under what assumptions*. Thus the main focus of the course is the presentation of "feasibility results" (i.e., proofs that a certain cryptographic task *can* be realized under certain assumptions). We will typically not relate to issues of efficiency (beyond equating efficiency with polynomial-time). There are a number of significant differences between this course and its prerequisite "Introduction to Cryptography" (89-656) given last semester:

1. First, our presentation here will be rigorous, and so we will only present constructions that have been proven secure.

2. Second, we will not begin with cryptographic applications like encryption and signatures, but will rather conclude with them. Rather, we start by studying one-way functions and their variants, and then show how different cryptographic primitives can be built from these. (Continuing the analogy of "foundations", we begin by building from the foundations and up, rather than starting with applications and working down to show how they can be securely realized.)

3. Third, the aim of the course is to provide the students with a deep understanding of how secure cryptographic solutions are achieved, rather than with a basic understanding of the important concepts and constructions.[1] Thus, we will cover far fewer topics in this course.

We note that the need for a rigorous approach in cryptography is especially strong. First, intuitive and heuristic arguments of security have been known to fail dismally when it comes to cryptography. Personally, my intuition has failed me many times. (I therefore do not believe anything until I have a full proof, and then I start thinking that it may be correct.) Second, in contrast to many other fields, the security of a cryptographic construction cannot be tested empirically. (By running a series of tests, you can see if something works under "many" conditions. However, such tests are of no help in seeing if a protocol can or cannot be maliciously attacked.) Finally, we note

* Lecture notes for a graduate course in the theory of cryptography. Yehuda Lindell, Bar-Ilan University, Israel, 2005.

My intention here is not at all to belittle the importance and place of the introductory course. Rather, the aim is different. I view the aim of the introductory course to provide students with an understanding of cryptographic problems and solutions that will guide them as *consumers* of cryptography. In contrast, the aim of this course is to be a *first step* on the way to learning how to build cryptographic solutions and prove them secure.

that the potential damage of implementing an insecure solution is often too great to warrant the chance. (In this way, cryptography differs from algorithms. A rigorous approach to algorithms is also important. However, heuristic solutions that almost always provide optimal solutions are often what is needed. In contrast, a cryptographic protocol that prevents most attacks is worthless, because an adversary can maliciously direct its attack at the weakest link.)

1.1.1 Preliminaries

We assume familiarity with complexity classes like $\mathcal{P}$, $\mathcal{NP}$, $\mathcal{BPP}$ and $\mathcal{P}/\text{poly}$. In general, we equate the notion of "efficient computation" with probabilistic polynomial-time. Thus, adversaries are assumed to be probabilistic polynomial-time Turing machines. (Recall that a Turing machine M runs in polynomial-time if there exists a single polynomial $p(\cdot)$ such that for every input x, the computation of $M(x)$ halts within $p(|x|)$ steps.) We will sometimes also consider non-uniform adversaries. Such an adversary can be modelled in one of two equivalent ways:

1. *Turing machine with advice:* In this formalization, a non-uniform machine is a pair $(M, \overline{a})$ where M is a two-input polynomial-time Turing machine and $\overline{a}$ is an infinite sequence of strings such that for every $n \in \mathbb{N}$, $|a_n| = \text{poly}(n)$.[2] The string a_n is the advice that M receives upon inputs of length n (note that for all inputs of length n, M receives the same advice).

2. *Families of polynomial-size circuits:* In this formalization, a non-uniform "machine" is represented by an infinite sequence (or family) of Boolean circuits $\mathcal{C} = (C_1, C_2, \ldots)$ such that for every $n \in \mathbb{N}$, $|C_n| = \text{poly}(n)$. Then, the computation upon input x is given by $C_{|x|}(x)$. We note that the size of a circuit is given by the number of edges that it has, and that there is a single polynomial that bounds the size of all circuits in the family.

Recall that $\mathcal{BPP} \subseteq \mathcal{P}/\text{poly}$. Therefore, for many tasks (like deciding a language or carrying out a well-defined adversarial attack), it holds that anything that a probabilistic polynomial-time machine can do, a non-uniform polynomial-time machine can also do. Thus, non-uniform adversaries are stronger than probabilistic polynomial-time ones. It is not clear whether adversaries should be modelled as probabilistic polynomial-time or non-uniform polynomial-time (or whether this makes any difference). The tradeoff between them, however, is clear: security guarantees against non-uniform adversaries are stronger, but almost always rely on stronger hardness assumptions (see "intractability assumptions" below). Another important point to make is that proofs of security for probabilistic polynomial-time adversaries hold also for non-uniform polynomial-time adversaries. Therefore, "uniform" proofs of security are preferable (where they are known).

Negligible functions. We will almost always allow "bad events" to happen with small probability. Since our approach here is *asymptotic*, we say that an event happens with small probability if for all sufficiently large n's, it occurs with probability that is smaller than $1/p(n)$ for every polynomial $p(\cdot)$. Formally:

Definition 1.1 (negligible functions): *A function $\mu : \mathbb{N} \to \mathbb{R}$ is* negligible in n *(or just* negligible*) if for every positive polynomial $p(\cdot)$ there exists an N such that for every $n > N$, $\mu(n) < 1/p(n)$.*

[2] We will repeatedly use the notation poly(n) during the course. It is important to understand the quantification that is intended here. What we mean is that there exists a *single* polynomial $p(\cdot)$ such that for every n, $|a_n| \leq p(n)$.

We will abuse notation with respect to negligible functions and will often just write $f(n) < \mu(n)$ when analyzing the function f. Our intention here is to say that there *exists* a negligible function μ such that $f(n) < \mu(n)$. When being more explicit, we will also often write that for every polynomial $p(\cdot)$ and all sufficiently large n's $f(n) < 1/p(n)$. Our intention here is the same as in Definition 1.1.

We note that a function f is non-negligible if there exists a polynomial $p(\cdot)$ such that for infinitely many n's it holds that $f(n) \geq 1/p(n)$. This is not to be confused with a noticeable function f for which it holds that there exists a polynomial $p(\cdot)$ such that for all n, $f(n) \geq 1/p(n)$. Notice that there exist non-negligible functions that are not noticeable. For example, consider the function $f(n)$ defined by $f(n) = 1/n$ for even n, and $f(n) = 2^{-n}$ for odd n.

Intractability assumptions. We will consider a task as intractable or infeasible if it cannot be carried out by a probabilistic polynomial-time machine (except with negligible probability). Thus, an encryption scheme will be secure if the task of "breaking it" is intractable in this sense. We note that in the non-uniform model, a task will be considered intractable if it cannot be carried out be a non-uniform polynomial-time machine (except with negligible probability).

As we discussed in the course "Introduction to Cryptography", most of the cryptographic tasks that we consider are impossible if $\mathcal{P} = \mathcal{NP}$. Therefore, almost all theorems that we prove will rely on an *initial hardness assumption*. We note that today, it is unknown whether cryptography can be based on $\mathcal{NP}$-hardness. There are a number of reasons for this. On the most simple level, $\mathcal{NP}$-completeness only provides for *worst-case* hardness, whereas we are interested in *average-case* hardness. (In particular, it does not suffice for us to construct an encryption scheme that cannot always be broken. Rather, we need it to be unbreakable almost all the time.) In addition, we will need a hardness assumption that provides efficiently samplable hard instances. However, we do not know how to efficiently sample hard instances of $\mathcal{NP}$-complete problems.

Shorthand and notation:

- PPT: probabilistic polynomial-time

- $\mu(n)$: an arbitrary negligible function (interpret $f(n) < \mu(n)$ as that there exists a negligible function $\mu(n)$ such that $f(n) < \mu(n)$).

- poly(n): an arbitrary polynomial (interpret $f(n) = \text{poly}(n)$ as that there exists a polynomial $p(n)$ such that $f(n) \leq p(n)$).

- U_n denotes a random variable that is uniformly distributed over $\{0,1\}^n$. We note that if we write U_n twice in the same equation, then we mean the same random variable. (When we wish to refer to two independent instances of the random variable, we will write $U_n^{(1)}$ and $U_n^{(2)}$.)

- Negligible, non-negligible, noticeable and overwhelming probability: we say that an event occurs with negligible, non-negligible or noticeable probability if there exists a negligible, non-negligible or noticeable function (respectively), such that the event occurs with the probability given by the function. We say that an event occurs with overwhelming probability, if it occurs except with negligible probability.

1.2 Computational Difficulty – One-Way Functions

As we have mentioned, it is currently not known whether it is possible to base cryptography on the assumption that $\mathcal{P} \neq \mathcal{NP}$. Rather, the most basic assumption used in cryptography is that of the existence of a one-way function. Loosely speaking, such a function has the property that it is easy to compute, but (almost always) hard to invert. One may wonder why we choose such a primitive as an assumption. On a very simplistic and informal level we argue that given the tasks that we wish to carry out, it is a natural choice. Cryptographic tasks often involve the honest parties carrying out some computation (that must be efficient), with the result being that some "information" is hidden from the adversary. Thus, the "easy" direction of computing the one-way function is carried out by the honest parties. Furthermore, the desired information is hidden so that it can only be revealed by inverting the one-way function. Since the function is hard to invert, no adversary can obtain this information. A theoretically more sound answer to the question "Why one-way functions?" is due to the fact that the existence of many of the secure cryptographic primitives that we would like to construct (like pseudorandom generators, encryption schemes, signatures schemes and so on) actually implies the existence of one-way functions. Thus, the existence of one-way functions is a *minimal assumption* when it comes to constructing these primitives.

1.2.1 One-Way Functions – Definition

One-way functions (or strong one-way functions) have the property that they are easy to compute, but hard to invert. Since we are interested in a computational task that is almost always hard to solve, the hard-to-invert requirement is formalized by saying that an adversary will fail to invert the function (i.e., find *some* preimage), except with negligible probability. (Note that it is always possible to succeed with negligible probability, by just guessing a preimage of the appropriate length.)

Definition 1.2 (one-way functions): *A function $f : \{0,1\}^* \rightarrow \{0,1\}^*$ is called* (strongly) one-way *if the following two conditions hold:*

1. Easy to compute: *There exists a polynomial-time algorithm A such that on input x, A outputs $f(x)$; i.e., $A(x) = f(x)$.*
2. Hard to invert: *For every probabilistic polynomial-time algorithm A, every positive polynomial $p(\cdot)$ and all sufficiently large n's*

$$\Pr\left[A(f(U_n), 1^n) \in f^{-1}(f(U_n))\right] < \frac{1}{p(n)} \tag{1.1}$$

We note that when we say *one-way functions*, by default we mean *strong* one-way functions. The qualifier "strong" is only used to differentiate them from *weak* one-way functions, as defined below. Note also that a function that is *not* one-way is not necessarily easy to invert all the time (or even "often"). Rather, the converse of Definition 1.2 is that there exists a PPT algorithm A and a positive polynomial $q(\cdot)$ such that for *infinitely many n's*, $\Pr\left[A(f(U_n), 1^n) \in f^{-1}(f(U_n))\right] \geq \frac{1}{q(n)}$.

Comments on the definition. First, notice that the quantification in the hard-to-invert requirement is over *all* PPT algorithms. Thus, we have assumed something about the *power* of the adversary, but nothing about its *strategy*. This distinction is of prime importance when it comes to defining security. Next, notice that the adversary A is not required to output the same x used in computing $f(x)$; rather any preimage (any value in the set $\{f^{-1}(f(x))\}$) suffices.

On a different note, we remark that the probability in Eq. (1.1), although not explicitly stated, is over the choice of U_n and the uniformly distributed coins on A's random tape. It is important to always understand the probability space being considered (and therefore to explicitly state it where it is not clear). Finally, we explain why the algorithm A is also given an auxiliary input 1^n. This is provided in order to rule out trivial one-way functions that shrink their input to such an extent, that A simply doesn't have time to write a preimage. For example, consider the length function $f_{\text{len}}(x) = |x|$, where $|x|$ is the binary representation of the number of bits in x (i.e., f_{len} applied to any string of length n is the binary representation of the integer n, which is a string of length $\lceil \log n \rceil$). Such a function is easy to invert, as long as the inverting algorithm is allowed to run for n steps. However, since the running-time of algorithms is measured as a function of the length of their input, an inverting algorithm for f_{len} must run in exponential-time. Providing A with the auxiliary input 1^n rules out such functions. This technicality is a good example of the difficulty of properly defining cryptographic primitives.

1.2.2 Weak One-Way Functions

An important goal of the theory of cryptography is to understand the *minimal requirements* necessary for obtaining security. A natural question to ask is therefore whether it is possible to weaken the requirement that a one-way function be almost always hard to invert. In particular, what about functions that are just hard to invert with some noticeable probability?

Loosely speaking, a weak one-way function is one that is sometimes hard to invert. More exactly, there exists a polynomial $p(\cdot)$ such that every adversary fails to invert the function with probability at least $1/p(n)$. This seems much weaker than the notion of (strong) one-way functions above, and it is natural to ask what such a function can be used for. The good news here is that it turns out that the existence of weak one-way functions is *equivalent* to the existence of strong one-way functions. Therefore, it suffices to demonstrate (or assume) that some function is weakly one-way, and we automatically obtain *strong* one-wayness.

Definition 1.3 (weak one-way functions): *A function $f : \{0,1\}^* \rightarrow \{0,1\}^*$ is called* weakly one-way *if the following two conditions hold:*

1. Easy to compute: *As in Definition 1.2.*
2. Hard to invert: *There exists a polynomial $p(\cdot)$ such that for every probabilistic polynomial-time algorithm A and all sufficiently large n's*

$$\Pr\left[A(f(U_n), 1^n) \notin f^{-1}(f(U_n))\right] > \frac{1}{p(n)}$$

Thus, if f is weakly one-way, it follows that every algorithm A will fail to invert with *noticeable probability*. Note that there is a single polynomial $p(\cdot)$ that bounds the success of all adversaries. (This order of quantification is crucial in the proof that the existence of weak one-way functions implies the existence of strong one-way functions.)

1.2.3 Candidates

One-way functions are only of interest if they actually exist. Since we cannot prove that they exist, we conjecture or assume their existence. This conjecture (assumption) is based on some very natural problems that have received much attention, and have yet to yield polynomial-time algorithms. Perhaps the most famous of these problems is that of integer factorization. This problem relates to the difficult of finding the prime factors of a number that is the product of long (and equal-length) uniformly distributed primes. This leads us to define the function $f_{\text{mult}}(x, y) = x \cdot y$, where $|x| = |y|$. That is, f_{mult} takes its random input, divides it into two equal parts and multiplies them together.

How hard is it to invert f_{mult}? First, note that there are many numbers for which it is easy to find their prime factors. For example, these include prime numbers themselves, numbers that have only small prime factors, and numbers p for which $p - 1$ has only small prime factors. Next, note that if x and y are prime (i.e., the input happens to be two primes), then by the hardness of the integer factorization problem, the output of the function will be hard to invert. However, x and y are uniformly distributed. Nevertheless, it is easy to show that f_{mult} is weakly one-way. In order to see this, recall the density-of-primes theorem that guarantees that at least $N/\log_2 N$ integers smaller than N are primes. Taking $N = 2^n$, where $2n$ is the length of the input, we have that the probability that x is prime equals at least $(2^n/n)/2^n = 1/n$, and so the probability of both x and y being prime equals $1/n^2$. It follows that f_{mult} is *weakly* one-way. Applying the hardness amplification of Theorem 2.1 below, we obtain the existence of (strong) one-way functions, based on the hardness of integer factorization problem. We note that it is actually possible to show that f_{mult} as it is, without any amplification, is strongly one-way (but this is more involved).

1.3 Strong Versus Weak One-Way Functions

In this section, we study the relation between weak and strong one-way functions.

1.3.1 Weak One-Way Functions Are Not Necessarily Strong

Although it seems intuitively clear that there should exist weak one-way functions that are not strong, we are going to prove this fact. This demonstrates that the notions of weak and strong one-way function are different. This will be our first formal proof, so even though it is intuitively clear, we will go through it slowly.

Proposition 1.4 *Assuming the existence of one-way functions, there exists a weakly one-way function that is not strongly one-way.*

Proof: The idea is to take a one-way function f and construct a function g from f, such that g is only weakly one-way. Intuitively, we do this by making g hard-to-invert only sometimes.

Let f be a strong one-way function. Then, define[3]

$$g(\sigma, x) = \begin{cases} \sigma f(x) & \text{if } \sigma = 0^{\log_2 |x|}, \\ \sigma x & \text{otherwise.} \end{cases}$$

In our analysis below, we always assume that the input length equals $n + \log_2 n$ for some integer n. This is justified by the fact that it is possible to define $g(x) = f(x)$ for x that is not of the required length, and otherwise it is as defined here. In this way, we will obtain that g is not a strong one-way function (because for infinitely many n's it is possible to invert it with high probability). Furthermore, g will clearly be weak for n's that are not of the required length, because in this case $g(x) = f(x)$. It therefore suffices to analyze g for inputs of length $n + \log_2 n$ and we can ignore this technicality from now on.

Clearly, g is not (strongly) one-way, because with probability $1 - 1/|x|$ it is easy to invert (in fact, with this probability it is just the identity function). It remains to show that g is weakly one-way. It may be tempting to just say that in the case that $\sigma = 0^{\log_2 |x|}$, g is hard to invert because f is hard to invert. However, this is not a formal proof. Rather, we need to show that there exists a polynomial $p(\cdot)$ such that if g can be inverted with probability greater than $1 - 1/p(n)$, then f can be inverted with non-negligible probability. This is called a *proof by reduction* and almost all of the proofs that we will see follow this line of reasoning.

We prove that for inputs of length $n + logn$, the function g is hard to invert for $p(n) = 2n$. That is, we show that for every algorithm A and all sufficiently large n's

$$\Pr\left[A(g(U_{n+\log n}), 1^{n+\log n}) \notin g^{-1}(g(U_{n+\log n}))\right] > \frac{1}{2n}$$

Assume, by contradiction, that there exists an algorithm A' such that for infinitely many n's

$$\Pr\left[A'(g(U_{n+\log n}), 1^{n+\log n}) \in g^{-1}(g(U_{n+\log n}))\right] \geq 1 - \frac{1}{2n}$$

We use A' to construct an algorithm A'' that inverts f on infinitely many n's. Upon input $(y, 1^n)$, algorithm A'' invokes A' with input $(0^{\log_2 n} y, 1^{n+\log n})$ and outputs the last n bits of A''s output. Intuitively, if A' fails with probability less than $1/(2n)$ over uniformly distributed strings, then it should fail with probability at most $1/2$ over strings that start with $0^{\log n}$ (because these occur with probability $1/n$). We therefore have that A'' will succeed to invert with probability at least $1/2$.

Let S_n denote the subset of all strings of length $n + \log n$ that start with $\log n$ zeroes (i.e., $S_n = \{0^{\log_2 n}\alpha \mid \alpha \in \{0,1\}^n\}$). Noting that $\Pr[U_{n+\log n} \in S_n] = 1/n$, we have that

$$\begin{aligned}
\Pr[A''(f(U_n), 1^n) \in f^{-1}(f(U_n))] &= \Pr[A'(0^{\log_2 n} f(U_n), 1^{n+\log n}) \in (0^{\log_2 n} f^{-1}(f(U_n)))] \\
&= \Pr[A'(g(U_{n+\log n}), 1^{n+\log n}) \in g^{-1}(g(U_{n+\log n})) \mid U_{n+\log n} \in S_n] \\
&\geq \frac{\Pr[A'(g(U_{n+\log n}), 1^{n+\log n}) \in g^{-1}(g(U_{n+\log n}))] - \Pr[U_{n+\log n} \notin S_n]}{\Pr[U_{n+\log n} \in S_n]}
\end{aligned}$$

where the inequality follows from the fact that $\Pr[A|B] = \Pr[A \wedge B]/\Pr[B]$ and $\Pr[A \wedge B] \geq \Pr[A] - \Pr[\neg B]$.

By our contradicting assumption on A', we have that the last value in the equation is greater than or equal to:

$$\frac{\left(1 - \frac{1}{2n}\right) - \left(1 - \frac{1}{n}\right)}{\frac{1}{n}} = \frac{1/2n}{1/n} = \frac{1}{2}$$

and so for infinitely many n's, the algorithm A'' inverts f with probability at least $1/2$. This contradicts the fact that f is a one-way function. ■

We note that the reduction that we have shown here is similar in spirit to the classic $\mathcal{NP}$-reductions that you have all seen. However, it also differs in a fundamental way. Specifically, an $\mathcal{NP}$-reduction states that if there is an algorithm that *always* solves one problem, then it can be used to *always* solve another problem. In contrast, in cryptography, reductions state that if there is an algorithm that solves one problem with some probability ϵ, there exists an algorithm that solves another problem with some probability δ. This makes quite a difference (as we will especially see in the proof of the Goldreich-Levin hardcore bit next week).

Lecture 2

One-Way Functions (continued)

2.1 Strong Versus Weak One-Way Functions

We continue to study the relation between weak and strong one-way functions.

2.1.1 Equivalence of Weak and Strong One-Way Functions

In this section, we state an important (and very non-trivial) theorem stating that strong one-way functions exist if and only if weak one-way functions exist. The interesting direction involves showing how a strong one-way function can be constructed from a weak one. This technique is called hardness amplification. The proof of this theorem is left as a reading assignment.

Theorem 2.1 *Strong one-way functions exists if and only if weak one-way functions exist.*

We will not present the proof, but just provide some intuition into the construction and why it works. Let f be a weak one-way function and let $p(\cdot)$ be such that all PPT algorithms fail to invert $f(U_n)$ with probability at least $p(n)$. Then, a strong one-way function g can be constructed from f as follows. Let the input of g be a string of length $n^2p(n)$ and denote it $x_1,\ldots,x_{np(n)}$ where for every i, $x_i \in \{0,1\}^n$. Then, define $g(x) = (f(x_1),\ldots,f(x_{np(n)}))$.

The intuition behind this construction is that if f is hard to invert with probability $1/p(n)$, then at least some of the $f(x_i)$'s should be hard to invert. (The function f is applied many times in order to lower the success probability to be negligible in n.) We note that it is easy to show that any algorithm that inverts g by inverting each $f(x_i)$ independently contradicts the weak one-wayness of f. This is due to the fact that each $f(x_i)$ can be inverted with probability at most $1 - 1/p(n)$. Therefore, the probability of succeeding on all $f(x_i)$'s is at most $(1-\frac{1}{p(n)})^{np(n)} < e^{-n}$. However, such an argument assumes something about the *strategy* of the inverting algorithm, whereas we can only assume something about its computational power. Therefore, the proof must work by reduction, showing that any algorithm that can invert g with non-negligible probability can invert f with probability greater than $1/p(n)$. This then contradicts the weak one-wayness of f with respect to $p(\cdot)$.

2.2 Collections of One-Way Functions

The formulation of one-way functions in Definition 1.2 is very useful due to its simplicity. However, most candidates that we know are not actually functions from $\{0,1\}^*$ to $\{0,1\}^*$. This motivates

[*] Lecture notes for a graduate course in the theory of cryptography. Yehuda Lindell, Bar-Ilan University, Israel, 2005.

the definition of *collections of one-way functions*. Such functions can be defined over an arbitrary (polynomial-time samplable) domain, and there may be a different function for each domain. In order to make this more clear, think about the RSA one-way function $f_{e,N}(x) = x^e \ \mathsf{mod} N$. In order to define the function, one first needs to choose e and N. Then, both the computation of the function and the domain of the function depend on these values. Indeed, there is no single RSA function that works over an infinite domain. Rather, the RSA family is an *infinite set* of *finite functions*.

Definition 2.2 (collections of one-way functions): *A* collection of functions *consists of an infinite set of indices $\overline{I}$, a corresponding set of functions $\{f_i\}_{i\in\overline{I}}$, and a set of finite domains $\{D_i\}_{i\in\overline{I}}$, where the domain of f_i is D_i.*

A collection of functions $(\overline{I}, \{f_i\}, \{D_i\})$ is called one-way *if there exist three probabilistic polynomial-time algorithms I, D and F such that the following conditions hold:*

1. Easy to sample and compute: *The output distribution of algorithm I on input 1^n is a random variable assigned values in the set $\overline{I} \cap \{0,1\}^n$. The output distribution of algorithm D on input $i \in \overline{I}$ is a random variable assigned values in the set D_i. On input $i \in \overline{I}$ and $x \in D_i$, algorithm F always outputs $f_i(x)$; i.e., $F(i,x) = f_i(x)$.*

2. Hard to invert: *For every probabilistic polynomial-time algorithm A', every positive polynomial $p(\cdot)$ and all sufficiently large n's,*

$$\Pr\left[A'(I_n, f_{I_n}(X_n)) \in f_{I_n}^{-1}(f_{I_n}(X_n))\right] < \frac{1}{p(n)}$$

where I_n is a random variable denoting the output distribution of $I(1^n)$ and X_n is a random variable denoting the output distribution of D on input (random variable) I_n.

We denote a collection of one-way functions by its algorithms (I, D, F).

Note that the probability in the equation is over the coin-tosses of A', I and D. There are a few relaxations of this definition that are usually considered. First, we allow I to output indices of length poly(n) rather than of length strictly n. Second, we allow all algorithms to fail with negligible probability (this is especially important for algorithm I).

Variants: There are a number of variants of one-way functions that are very useful. These include length-preserving one-way functions where $|f(x)| = |x|$, length-regular one-way functions where for every x, y such that $|x| = |y|$ it holds that $|f(x)| = |f(y)|$, 1–1 one-way functions, and one-way permutations that are bijections (i.e., 1–1 and onto).

We note that if one-way functions exists, then length-preserving and length-regular one-way functions exist. (We can therefore assume these properties without loss of generality.) In contrast, there is evidence that proving the existence of one-way functions does not suffice for proving the existence of one-way permutations [16].

2.3 Trapdoor One-Way Permutations

A collection of trapdoor one-way permutations, usually just called trapdoor permutations, are collections of one-way permutations with an addition "trapdoor" property. Informally speaking, the trapdoor t is an additional piece of information that is output by the index sampler I such that given t, it is possible to invert the function. Of course, without knowing t, the function should be hard to invert, since it is one-way. Recall that the RSA family is defined by (e, N), but given $d = e^{-1} \bmod \varphi(n)$, it is possible to invert $f_{e,N}$. Thus, d is the RSA trapdoor.

Definition 2.3 (collection of trapdoor permutations): *Let $I : 1^* \to \{0,1\}^* \times \{0,1\}^*$ be a probabilistic algorithm, and let $I_1(1^n)$ denote the first element of the pair output by $I(1^n)$. A triple of algorithms (I, D, F) is called a* collection of trapdoor permutations *if the following two conditions hold:*

1. The algorithms induce a collection of one-way permutations: *The triple (I_1, D, F) constitutes a collection of one-way permutations, as in Definition 2.2.*

2. Easy to invert with trapdoor: *There exists a (deterministic) polynomial-time algorithm, denoted F^{-1} such that for every (i, t) in the range of I and for every $x \in D_i$, it holds that $F^{-1}(t, f_i(x)) = x$.*

As with collections of one-way functions, it is possible to relax the requirements and allow F^{-1} to fail with probability that is negligible in n.

Recommended Exercises for Sections 2.2 and 2.3

1. Show that under the RSA assumption, the RSA family is a collection of one-way functions. (That is, fully define and analyze each of the (I, D, F) algorithms.) It is easier to use the above relaxations for this.

   Do the same for other candidates that we saw in the course "Introduction to Cryptography".

2. Show that the RSA family is actually a collection of trapdoor one-way *permutations*.

3. Show that if there exist collections of one-way functions as in Definition 2.2, then there exist one-way functions as in Definition 1.2.

2.4 Hard-Core Predicates

Intuitively, a one-way functions hides information about its preimage; otherwise, it would be possible to invert the function. However, it does *not* necessarily hide its entire preimage. For example, let f be a one-way function and define $g(x_1, x_2) = x_1, f(x_2)$, where $|x_1| = |x_2|$. Then, it is easy to show that g is also a one-way function (exercise: prove this). However, g reveals half of its input. We therefore need to define a notion of information that is *guaranteed to be hidden* by the function; this is exactly the purpose of a hard-core predicate.

Loosely speaking, a hard-core predicate b of a function f is a function outputting a single bit with the following property: If f is one-way, then upon input $f(x)$ it is infeasible to correctly guess $b(x)$ with any non-negligible advantage above $1/2$. (Note that it is always possible to guess $b(x)$ correctly with probability $1/2$.)

We note that some functions have "trivial" hard-core predicates. For example, let f be a function and define $g(\sigma, x) = f(x)$ where $\sigma \in \{0,1\}$ and $x \in \{0,1\}^n$. Then, g clearly "hides" σ. In contrast, a 1–1 function g has a hard-core predicate only if it is one-way. Intuitively, this is the case because when a function is 1–1, all the "information" about the preimage x is found in $f(x)$. Therefore, it can only be hard to compute $b(x)$ if f cannot be inverted. We will be interested in hard-core predicates, where the hardness is due to the difficulty of inverting f.

Definition 2.4 (hard-core predicate): *A polynomial-time computable predicate $b : \{0,1\}^* \to \{0,1\}$ is called a* hard-core of a function *f if for every probabilistic polynomial-time algorithm A', every positive polynomial $p(\cdot)$ and all sufficiently large n's*

$$\Pr\left[A'(f(U_n), 1^n) = b(U_n)\right] < \frac{1}{2} + \frac{1}{p(n)}$$

We remark that hard-core predicates of collections of functions are defined in an analogous way, except that b is also given the index i of the function.

2.5 Hard-Core Predicates for Any One-Way Function

In this section, we will present the Goldreich-Levin construction of a hard-core predicate for any one-way function [8]. We note that the Goldreich-Levin construction does not actually work for any one-way function. Rather, it works for a specific type of one-way function with the property that any one-way function can be transformed into one of this type, without any loss of efficiency. Furthermore, if the initial one-way function was 1–1 or a bijection, then so is the resulting one-way function. We will present the full proof of this theorem.

Theorem 2.5 (Goldreich-Levin hard-core predicate): *Let f be a one-way function and let g be defined by $g(x,r) = ((f(x),r))$, where $|x| = |r|$. Let $b(x,r) = \sum_{i=1}^{n} x_i \cdot r_i \bmod 2$ be the inner product function, where $x = x_1 \cdots x_n$ and $r = r_1 \cdots r_n$. Then, the predicate b is a hard-core of the function g.*

In order to motivate the construction, notice that if there exists a procedure A that *always* succeeds in computing $b(x,r)$ from $g(x,r) = (f(x),r)$, then it is possible to invert f. Specifically, upon input (y,r) where $y = f(x)$, it is possible to invoke A on $(f(x),r)$ and $(f(x), r \oplus e^i)$ where e_i is the vector with a 1 in the i^{th} place, and zeroes in all other places. Then, since A always succeeds, we obtain back $b(x,r)$ and $b(x, r \oplus e^i)$ and can compute

$$b(x,r) \oplus b(x, r \oplus e^i) = \sum_{j=1}^{n} x_j \cdot r_j + \sum_{j=1}^{n} x_j \cdot (r_j \oplus e^i) = x_i \cdot r_i + x_i \cdot (r_i \oplus 1) = x_i$$

Repeating the procedure for every $i = 1, \ldots, n$ we obtain $x = x_1, \ldots, x_n$ and so have inverted $f(x)$. Unfortunately, however, the negation of b being a hard-core predicate is only that there exists an algorithm that correctly computes $b(x,r)$ with probability $1/2 + \text{poly}(n)$. This case is much harder to deal with; in particular, the above naive approach fails because the chance of obtaining the correct x_i for every i is very small.

2.5.1 Preliminaries – Markov and Chebyshev Inequalities

Before proceeding to the proof of Theorem 3.1, we prove two important inequalities that we will use. These inequalities are used to measure the probability that a random variable will significantly deviate from its expectation.

Markov Inequality: *Let X be a non-negative random variable and v a real number. Then:*

$$\Pr[X \geq v \cdot \mathsf{Exp}[X]] \leq \frac{1}{v}$$

Equivalently: $\Pr[X \geq v] \leq \mathsf{Exp}[X]/v$.

Proof:

$$\begin{aligned}
\mathsf{Exp}[X] &= \sum_{x} Pr[X = x] \cdot x \\
&\geq \sum_{x<v} \Pr[X = x] \cdot 0 + \sum_{x \geq v} \Pr[X = x] \cdot v \\
&= \Pr[X \geq v] \cdot v
\end{aligned}$$

■

The Markov inequality is extremely simple, and is useful when very little information about X is given. However, when an upper-bound on its variance is known, better bounds exist. Recall that $\mathsf{Var}(X) \stackrel{\text{def}}{=} \mathsf{Exp}[(X - \mathsf{Exp}[X])^2]$, that $\mathsf{Var}(X) = \mathsf{Exp}[X^2] - \mathsf{Exp}[X]^2$, and that $\mathsf{Var}[aX + b] = a^2 \mathsf{Var}[X]$.

Chebyshev's Inequality: *Let X be a random variable and $\delta > 0$. Then:*

$$\Pr[|X - \mathsf{Exp}[X]| \geq \delta] \leq \frac{\mathsf{Var}(X)}{\delta^2}$$

Proof: We define a random variable $Y \stackrel{\text{def}}{=} (X - \mathsf{Exp}[X])^2$ and then apply the Markov inequality.

$$\begin{aligned}
\Pr[|X - \mathsf{Exp}[X]| \geq \delta] &= \Pr[(X - \mathsf{Exp}[X])^2 \geq \delta^2] \\
&\leq \frac{\mathsf{Exp}[(X - \mathsf{Exp}[X])^2]}{\delta^2}
\end{aligned}$$

■

An important corollary of Chebyshev's inequality relates to pairwise independent random variables. A series of random variables $X_1, \ldots, X_m$ are called pairwise independent if for every $i \neq j$ and every a and b it holds that

$$\Pr[X_i = a \;\&\; X_j = b] = \Pr[X_i = a] \cdot \Pr[X_j = b]$$

We note that for pairwise independent random variables $X_1, \ldots, X_m$ it holds that $\mathsf{Var}[\sum_{i=1}^m X_i] = \sum_{i=1}^m \mathsf{Var}[X_i]$ (this is due to the fact that every pair of variables are independent and so their covariance equals 0). (Recall that $\mathsf{cov}(X, Y) = \mathsf{Exp}[XY] - \mathsf{Exp}[X]\mathsf{Exp}[Y]$ and $\mathsf{Var}[X+Y] = \mathsf{Var}[X] + \mathsf{Var}[Y] - 2\mathsf{cov}(X, Y)$. This can be extended to any number of random variables.)

Corollary 2.6 (pairwise-independent sampling): *Let $X_1, \ldots, X_m$ be pairwise-independent random variables with the same expectation μ and the same variance σ^2. Then, for every $\epsilon > 0$,*

$$\Pr\left[\left|\frac{\sum_{i=1}^m X_i}{m} - \mu\right| \geq \epsilon\right] \leq \frac{\sigma^2}{\epsilon^2 m}$$

Proof: By the linearity of expectations, $\mathsf{Exp}[\sum_{i=1}^m X_i/m] = \mu$. Applying Chebyshev's inequality, we have

$$\Pr\left[\left|\frac{\sum_{i=1}^m X_i}{m} - \mu\right| \geq \epsilon\right] \leq \frac{\mathsf{Var}\left(\sum_{i=1}^m \frac{X_i}{m}\right)}{\epsilon^2}$$

By pairwise independence, it follows that

$$\mathsf{Var}\left(\sum_{i=1}^m \frac{X_i}{m}\right) = \sum_{i=1}^m \mathsf{Var}\left(\frac{X_i}{m}\right) = \frac{1}{m^2}\sum_{i=1}^m \mathsf{Var}(X_i) = \frac{1}{m^2}\sum_{i=1}^m \sigma^2 = \frac{\sigma^2}{m}$$

The inequality is obtained by combining the above two equations. ■

Lecture 3

Hard-Core Predicates for Any One-Way Function

3.1 Proof of the Goldreich-Levin Hard-Core Predicate [8]

We now prove that the Goldreich-Levin construction indeed constitute a hard-core predicate for any one-way function of the defined type.

Theorem 3.1 (Goldreich-Levin hard-core predicate – restated): *Let f be a one-way function and let g be defined by $g(x,r) = ((f(x),r))$, where $|x| = |r|$. Let $b(x,r) = \sum_{i=1}^{n} x_i \cdot r_i \bmod 2$ be the inner product function, where $x = x_1 \cdots x_n$ and $r = r_1 \cdots r_n$. Then, the predicate b is a hard-core of the function g.*

Proof: Assume by contradiction, that there exists a probabilistic polynomial-time algorithm A and a polynomial $p(\cdot)$ such that for infinitely many n's

$$\Pr\left[A(f(X_n), R_n) = b(X_n, R_n)\right] \geq \frac{1}{2} + \frac{1}{p(n)}$$

where X_n and R_n are independent random variables that are uniformly distributed over $\{0,1\}^n$. We denote $\epsilon(n) = \Pr[A(f(X_n), R_n) = b(X_n, R_n)] - \frac{1}{2}$ and so $\epsilon(n) \geq 1/p(n)$. By the assumption, A succeeds for infinitely many n's; denote this (infinite) set by N. From now on, we restrict ourselves to $n \in N$.

We first prove that there exists a noticeable fraction of inputs x for which A correctly computes $b(x, R_n)$ upon input $(f(x), R_n)$ with noticeable probability. Notice that this claim enables us to focus on a set of concrete "good inputs" upon which A often succeeds, and so we reduce the probability distribution to be over R_n (and not over X_n and R_n), which makes things easier. The claim below (and the rest of the proof) holds for $n \in N$.

Claim 3.2 *There exists a set $S_n \subseteq \{0,1\}^n$ of size at least $\frac{\epsilon(n)}{2} \cdot 2^n$ such that for every $x \in S_n$ it holds that*

$$s(x) \stackrel{\text{def}}{=} \Pr[A(f(x), R_n) = b(x, R_n)] \geq \frac{1}{2} + \frac{\epsilon(n)}{2}$$

[*] Lecture notes for a graduate course in the theory of cryptography. Yehuda Lindell, Bar-Ilan University, Israel, 2005.

Proof: Denote by S_n the set of all x's for which $s(x) \geq 1/2+\epsilon(n)/2$. We show that $|S_n| \geq \frac{\epsilon(n)}{2}\cdot 2^n$. This follows from a simple averaging argument. (That is, if A inverts with probability $1/2+\epsilon(n)$, then there must be at least an $\epsilon(n)/2$ fraction of inputs for which it succeeds with probability $1/2+\epsilon(n)/2$.) We have:

$$\begin{aligned}\Pr[A(f(X_n),R_n)=b(X_n,R_n)] &= \Pr[A(f(X_n),R_n)=b(X_n,R_n) \mid X_n \in S_n]\cdot\Pr[X_n\in S_n] \\ &\quad +\Pr[A(f(X_n),R_n)=b(X_n,R_n) \mid X_n \notin S_n]\cdot\Pr[X_n\notin S_n] \\ &\leq \Pr[X_n\in S_n]+\Pr[A(f(X_n),R_n)=b(X_n,R_n) \mid X_n \notin S_n]\end{aligned}$$

and so

$$\Pr[X_n\in S_n] \geq \Pr[A(f(X_n),R_n)=b(X_n,R_n)] - \Pr[A(f(X_n),R_n)=b(X_n,R_n) \mid X_n \notin S_n]$$

By the definition of S_n, it holds that for every $x \notin S_n$, $\Pr[A(f(x),R_n)=b(x,R_n)] < 1/2+\epsilon(n)/2$. Therefore, $\Pr[A(f(X_n),R_n)=b(X_n,R_n) \mid X_n \notin S_n] < 1/2+\epsilon(n)/2$, and we have that

$$\Pr[X_n\in S_n] \geq \frac{1}{2}+\epsilon(n)-\frac{1}{2}-\frac{\epsilon(n)}{2}=\frac{\epsilon(n)}{2}$$

This implies that S_n must be at least of size $\frac{\epsilon(n)}{2}\cdot 2^n$ (because X_n is uniformly distributed in $\{0,1\}^n$). ■

From now on, we will consider only "good inputs" from S_n (this suffices because a random input is from S_n with noticeable probability).

A motivating discussion. For a moment, we will consider a simplified scenario where it holds that for every $x \in S_n$, $s(x) \geq 3/4+\epsilon(n)/2$. This mental experiment is only for the purpose of demonstrating the proof technique. In such a case, notice that $\Pr[A(f(x),R_n) \neq b(x,R_n)] < 1/4-\epsilon(n)/2$ and $\Pr[A(f(x),R_n\oplus e^i) \neq b(x,R_n\oplus e^i)] < 1/4-\epsilon(n)/2$ (since $R_n\oplus e^i$ is also uniformly distributed. Therefore, the probability that A fails on at least one of $(f(x),R_n)$ is less than $1/2-\epsilon(n)$ (by using the union bound). Therefore, A correctly computes $b(x,R_n)$ and $b(x,R_n\oplus e^i)$ with probability at least $1/2+\epsilon(n)$. Recall that $b(x,R_n)\oplus b(x,R_n\oplus e^i)=x_i$. Now, if we repeat this procedure many times, we have that the majority result will equal x_i with high probability. (Specifically, repeating $\ln 4n/(2\epsilon^2)$ times and using the Chernoff bound, we obtain that the majority result is x_i with probability at least $1-1/2n$.)

The problem with this procedure when we move to the case that $s(x) \geq 1/2+\epsilon(n)/2$ is that the probability of getting a correct answer will not be greater than 1/2 (in fact, using the union bound, we will only guarantee a success probability of $\epsilon(n)$). Therefore, the majority result will not necessarily be the correct one.[1] We therefore must somehow compute $b(x,R_n)$ and $b(x,R_n\oplus e^i)$ without invoking A twice. The way we do this is to invoke A on $b(x,R_n)$ and "guess" the value $b(x,R_n\oplus e^i)$ ourselves. This guess is generated in a special way so that the probability of the guess being correct (for all i) is noticeable. (Of course, a naive way of guessing would be correct with only negligible probability, because we need to guess $b(x,r)$ for a polynomial number of r's.) The strategy for generating the guesses is via *pairwise independent* sampling. As we have already seen, Chebyshev's inequality can be applied to this case in order to bound the deviation from the expected.

Note that the events of successfully guessing $b(x,R_n)$ and $b(x,R_n\oplus e^i)$ are not independent. Furthermore, we don't know that the minority guess will be correct; rather, we know nothing at all.

Continuing with this discussion, we show how the pairwise independent r's are generated. In order to generate $m = \text{poly}(n)$ many r's, we select $l = \log_2(m+1)$ independent uniformly distributed strings in $\{0,1\}^n$; denote them by $s^1, \ldots, s^l$. Then, for every possible non-empty subset $I \subseteq \{1,\ldots,l\}$, we define $r^I = \oplus_{i\in I} s^i$. Notice that there are $2^l - 1$ non-empty subsets, and therefore we have defined $2^{\log_2(m+1)} - 1 = m$ different strings. We now claim that all of the strings r^I are pairwise independent. In order to see this, notice that for every two subsets $I \neq J$, there exists an index j such that $j \notin I \cap J$. Without loss of generality, assume that $j \in J$. Then, given r^I, it is clear that r^J is uniformly distributed because it contains a uniformly distributed string s^j that does not appear in r^I. Likewise, r^I is uniformly distributed given r^J because s^j "hides" r^I. (A formal proof of pairwise independence is left as an exercise.) Finally, we note that the values $b(x,s_1), \ldots, b(x,s_l)$ can be correctly guessed with probability $1/2^l$ which is noticeable. In addition, given $b(x,s_1), \ldots, b(x,s_l)$ and any non-empty subset I, it is possible to compute $b(x,r^I) = b(x, \oplus_{i\in I} s^i) = \oplus_{i\in I} b(x,s^i)$. (We note that an alternative strategy to guessing all the $b(x,s_i)$ values is to try all possibilities, checking if we have succeeded in inverting $y = f(x)$. Since there are only $m+1 = \text{poly}(n)$ different possibilities, we have enough time to do this.)

The inversion algorithm B. We now provide a full description of the algorithm B that receives an input y and uses algorithm A in order to find $f^{-1}(y)$. Upon input y, B computes n (recall that we assume that n is implicit in y) and $l = \lceil \log_2(2n/\epsilon(n)^2 + 1) \rceil$, and proceeds as follows:

1. Uniformly choose $s^1, \ldots, s^l \in_R \{0,1\}^n$ and $\sigma^1 \ldots, \sigma^l \in_R \{0,1\}$ (σ^i is a guess for $b(x,s^i)$).

2. For every non-empty subset $I \subseteq \{1,\ldots,l\}$, define $r^I = \oplus_{i\in I}\, s^i$ and compute $\tau^I = \oplus_{i\in I}\, \sigma^i$.

3. For every $i \in \{1,\ldots,n\}$, obtain a guess for x_i as follows:

   (a) For every non-empty subset $I \subseteq \{1,\ldots,l\}$, set $v_i^I = \tau^I \oplus A(y, r^I \oplus e^i)$.

   (b) Guess $x_i = \mathsf{majority}_I\{v_i^I\}$

4. Output $x = x_1 \cdots x_n$.

Analyzing B's success probability. It remains to compute the probability that B successfully outputs $x \in f^{-1}(y)$. Before proceeding with the formal analysis, we provide an intuitive explanation. First, consider the case that the τ^I's are all correct (recall that this occurs with noticeable probability). In such a case, we have that $v_i^I = x_i$ with probability at least $1/2 + \epsilon(n)/2$ (this is due to the fact that A is invoked only once in computing v_i^I; the τ^I factor is already assumed to be correct). It therefore follows that a majority of the v_i^I values will equal the real value of x_i. Our analysis will rely on Chebyshev's inequality for the case of pairwise independent variables, because we need to compute the probability that the majority equals the correct x_i, where this majority is due to all the pairwise independent r^I's. We now present the formal proof.

Claim 3.3 *Assume that for every I, it holds that $\tau^I = b(x,r^I)$. Then for every $x \in S_n$ and every $1 \leq i \leq n$, the probability that the majority of the v_i^I values equal x_i is at least $1 - 1/2n$. That is,*

$$\Pr\left[\left|\left\{J \,:\, b(x,r^J) \oplus A(f(x), r^J \oplus e^i) = x_i\right\}\right| > \frac{1}{2}\cdot\left(2^l - 1\right)\right] > 1 - \frac{1}{2n}$$

Proof: For every I, define a 0-1 random variable X^I such that $X^I = 1$ if and only if $A(y, r^I \oplus e^i) = b(x, r^I \oplus e^i)$. Notice that if $X^I = 1$, then $b(x, r^I) \oplus A(y, r^I \oplus e^i) = x_i$. Since each r^I and $r^I \oplus e^i$ are uniformly distributed in $\{0,1\}^n$ (when considered in isolation), we have that $\Pr[X^I = 1] = s(x)$, and so for $x \in S_n$, we have that $\Pr[X^I = 1] \geq 1/2 + \epsilon(n)/2$ implying that $\mathsf{Exp}[X^I] \geq 1/2 + \epsilon(n)/2$. Furthermore, we claim that all the X^I random variables are pairwise independent. This follows from the fact that the r^I values are pairwise independent. (Notice that if r^I and r^J are truly independent, then clearly so are X^I and X^J. Thus, the same is true of pairwise independence.)

Let $m = 2^l - 1$ and let X be a random variable that is distributed the same as all of the X^I's. Then, using Chebyshev's inequality, we have:

$$\begin{aligned}
\Pr\left[\sum_I X^I \leq \frac{1}{2}\cdot m\right] &\leq \Pr\left[\left|\frac{\sum_I mX^I}{m} - \left(\frac{1}{2}+\frac{\epsilon(n)}{2}\right)\cdot m\right| \geq m\cdot\frac{\epsilon(n)}{2}\right] \\
&\leq \frac{\mathsf{Var}[mX]}{(m\cdot\epsilon(n)/2)^2\cdot m} \\
&= \frac{m^2\mathsf{Var}[X]}{(\epsilon(n)/2)^2\cdot m^3}
\end{aligned}$$

Since $m = 2^l - 1 = 2n/\epsilon(n)^2$, it follows from the above that:

$$\begin{aligned}
\Pr\left[\sum_I X^I \leq \frac{1}{2}\cdot m\right] &= \frac{\mathsf{Var}[X]}{(\epsilon(n)/2)^2\cdot 2n/\epsilon(n)^2} \\
&= \frac{\mathsf{Var}[X]}{n/2} \\
&< \frac{1/4}{n/2} = \frac{1}{2n}
\end{aligned}$$

where $\mathsf{Var}[X] < 1/4$ because $\mathsf{Var}[X] = E[X^2] - E[X]^2 = E[X] - E[X]^2 = E[X](1 - E[X]) = (1/2 + s(x))(1/2 - s(x)) = 1/4 - s(x)^2 < 1/4$. This completes the proof of the claim because $\sum_I X^I$ is exactly the number of correct v_i^I values. ■

By Claim 3.3, we have that *if* all of the τ^I values are correct, then each x_i computed by B is correct with probability at least $1 - 1/2n$. By the union bound over the failure probability of $1/2n$ for each i, we have that if all the τ^I values are correct, then the entire $x = x_1 \cdots x_n$ is correct with probability at least $1/2$. Notice now that the probability of the τ^I values being correct is independent of the analysis of Claim 3.3 and that this event happens with probability

$$\frac{1}{2^l} = \frac{1}{2n/\epsilon(n)^2 + 1} > \frac{1}{2np(n)^2 + 1} > \frac{1}{4np(n)^2}$$

Therefore, for $x \in S_n$, algorithm B succeeds in inverting $y = f(x)$ with probability at least $1/8np(n)^2$. Recalling that $|S_n| > \frac{\epsilon(n)}{2}\cdot 2^n$, we have that $x \in S_n$ with probability $\epsilon(n)/2 > 1/2p(n)$ and so the overall probability that B succeeds in inverting $f(U_n)$ is greater than or equal to $1/16np(n)^3 = 1/\text{poly}(n)$. Finally, noting that B runs in polynomial-time, we obtain a contradiction to the (strong) one-wayness of f. ■

Lecture 4

Computational Indistinguishability & Pseudorandomness

4.1 Computational Indistinguishability

We introduce the notion of computational indistinguishability [11, 17]. Informally speaking, two distributions are computationally indistinguishable if no efficient algorithm can tell them apart (or *distinguish* them). This is formalized as follows. Let D be some PPT algorithm, or distinguisher. Then, D is provided either a sample from the first distribution or the second one. We say that the distributions are computationally indistinguishable if every such PPT D outputs 1 with (almost) the same probability upon receiving a sample from the first or second distribution.

The actual definition refers to *probability ensembles*. These are infinite series of finite probability distributions (similar to the notion of "collections of one-way functions"). This formalism is necessary because distinguishing two finite distributions is easy (an algorithm can just have both distributions explicitly hardwired into its code).

Definition 4.1 (probability ensemble): *Let I be a countable index set. A* probability ensemble indexed by *I is a sequence of random variables indexed by I.*

Typically, the set I will either be $\mathbb{N}$ or an efficiently computable subset of $\{0,1\}^*$. Furthermore, we will typically refer to an ensemble $X = \{X_n\}_{n\in\mathbb{N}}$, where X_n ranges over strings of length $\text{poly}(n)$. (Recall, this means that there is a single polynomial $p(\cdot)$ such that X_n ranges over strings of length $p(n)$, for *every* n.) We present the definition for the case that $I = \mathbb{N}$.

Definition 4.2 (computational indistinguishability): *Two probability ensembles $X = \{X_n\}_{n\in\mathbb{N}}$ and $Y = \{Y_n\}_{n\in\mathbb{N}}$ are* computationally indistinguishable, *denoted $X \overset{\text{c}}{\equiv} Y$, if for every probabilistic polynomial-time distinguisher D, every positive polynomial $p(\cdot)$ and all sufficiently large n's*

$$|\Pr[D(X_n, 1^n) = 1] - \Pr[D(Y_n, 1^n) = 1]| < \frac{1}{p(n)}$$

We note that in the usual case where $|X_n| = \Omega(n)$ and the length n can be derived from a sample of X_n, it is possible to omit the auxiliary input 1^n.

[*] Lecture notes for a graduate course in the theory of cryptography. Yehuda Lindell, Bar-Ilan University, Israel, 2005.

4.1.1 Multiple Samples

We say that an ensemble $X = \{X_n\}_{n\in\mathbb{N}}$ is efficiently samplable if there exists a PPT algorithm S such that for every n, the random variables $S(1^n)$ and X_n are identically distributed.

In this section, we prove that if two *efficiently samplable* ensembles X and Y are computationally indistinguishable, then a polynomial number of (independent) samples of X are computationally indistinguishable from a polynomial number of (independent) samples of Y. We stress that this theorem does not hold in the case that X and Y are not efficiently samplable. We present two different proofs; the first for the non-uniform case and the second for the uniform case. (We present both because the first is more simple.)

Theorem 4.3 (multiple samples – non-uniform version): *Let X and Y be efficiently samplable ensembles such that $X \overset{\mathrm{c}}{\equiv} Y$ for non-uniform distinguishers. Then, for every polynomial $p(\cdot)$, the ensembles $\overline{X} = \{(X_n^{(1)}, \ldots, X_n^{(p(n))})\}_{n\in\mathbb{N}}$ and $\overline{Y} = \{(Y_n^{(1)}, \ldots, Y_n^{(p(n))})\}_{n\in\mathbb{N}}$ are computationally indistinguishable for non-uniform distinguishers.*

Proof: The proof is by reduction. We show that if there exists a (non-uniform) PPT distinguisher D that distinguishes $\overline{X}$ from $\overline{Y}$ with non-negligible success, then there exists a non-uniform PPT distinguisher D' that distinguishes a single sample of X from a single sample of Y with non-negligible success. Our proof uses a very important proof technique, called a *hybrid argument*, first used in [11].

Assume by contradiction that there exists a (non-uniform) PPT distinguisher D and a polynomial $q(\cdot)$ such that for infinitely many n's

$$\left|\Pr\left[D(X_n^{(1)}, \ldots, X_n^{(p(n))}) = 1\right] - \Pr\left[D(Y_n^{(1)}, \ldots, Y_n^{(p(n))}) = 1\right]\right| \geq \frac{1}{q(n)}$$

For every i, we define a *hybrid* random variable H_n^i as a sequence containing i independent copies of X_n followed by $p(n) - i$ independent copies of Y_n. That is:

$$H_n^i = \left(X_n^{(1)}, \ldots, X_n^{(i)}, Y_n^{(i+1)}, \ldots, Y_n^{(p(n))}\right)$$

Notice that $H_n^0 = \overline{Y}_n$ and $H_n^{p(n)} = \overline{X}_n$. The main idea behind the hybrid argument is that if D can distinguish these extreme hybrids, then it can also distinguish neighbouring hybrids (even though it was not "designed" to do so). In order to see this, and before we proceed to the formal argument, we present the basic hybrid analysis. Denote $\overline{X}_n = (X_n^{(1)}, \ldots, X_n^{(p(n))})$ and likewise for $\overline{Y}_n$. Then, we have:

$$\left|\Pr[D(\overline{X}_n) = 1] - \Pr[D(\overline{Y}_n) = 1]\right| = \left|\sum_{i=0}^{p(n)-1} \Pr[D(H_n^i) = 1] - \sum_{i=0}^{p(n)-1} \Pr[D(H_n^{i+1}) = 1]\right|$$

This follows from the fact that the only remaining terms in this telescopic sum are $\Pr[D(H_n^0) = 1]$ and $\Pr[D(H_n^{p(n)}) = 1]$. By our contradicting assumption, for infinitely many n's we have that:

$$\begin{aligned}
\frac{1}{q(n)} &\leq \left|\Pr[D(\overline{X}_n) = 1] - \Pr[D(\overline{Y}_n) = 1]\right| \\
&= \left|\sum_{i=0}^{p(n)-1} \Pr[D(H_n^i) = 1] - \sum_{i=0}^{p(n)-1} \Pr[D(H_n^{i+1}) = 1]\right| \\
&\leq \sum_{i=0}^{p(n)-1} \left|\Pr[D(H_n^i) = 1] - \Pr[D(H_n^{i+1}) = 1]\right|
\end{aligned}$$

Therefore, there must exist neighbouring hybrids for which D distinguishes with non-negligible probability (or the entire sum would be negligible which is not the case). This fact will be used to construct a D' that will distinguish a single sample. Indeed, the only difference between neighbouring hybrids is a single sample.

Formally, as we have already seen above,

$$\sum_{i=0}^{p(n)-1} \left|\Pr[D(H_n^i) = 1] - \Pr[D(H_n^{i+1}) = 1]\right| \geq \frac{1}{q(n)}$$

Thus, there exists a value k ($0 \leq k < q(n)$) such that D distinguishes H_n^k from H_n^{k+1} with probability at least $1/p(n)q(n)$. Otherwise, the sum above would not reach $1/q(n)$. That is, we have that for some k,

$$\left|\Pr[D(H_n^k) = 1] - \Pr[D(H_n^{k+1}) = 1]\right| \geq \frac{1}{p(n)q(n)}$$

Once again, note that D was not "designed" to work on such hybrids and may not "intend" to receive such inputs. The argument here has nothing to do with what D "means" to do. What is important is that if D distinguishes the extreme hybrids, then for some k it distinguishes the k^{th} hybrid from the $k+1^{\text{th}}$ hybrid. Now, since we are considered the non-uniform setting here, we can assume that the distinguisher D' has the value of k as part of its advice tape (note that this k may be different for every k). Thus, upon input α, D' generates the vector $\overline{H}_n = (X_n^{(1)}, \ldots, X_n^{(k)}, \alpha, Y_n^{(k+2)}, \ldots, Y_n^{(p(n))})$, invokes D on the vector $\overline{H}_n$, and outputs whatever D does.[1] Now, if α is distributed according to X_n, then $\overline{H}_n$ is distributed exactly like H_n^{k+1}. In contrast, if α is distributed according to Y_n, then $\overline{H}_n$ is distributed exactly like H_n^k. We therefore have that

$$\left|\Pr[D'(X_n) = 1] - \Pr[D'(Y_n) = 1]\right| = \left|\Pr[D(H_n^k) = 1] - \Pr[D(H_n^{k+1}) = 1]\right| \geq \frac{1}{p(n)q(n)}$$

in contradiction to the computational indistinguishability of a single sample of X from a single sample of Y with respect to non-uniform PPT distinguishers. ■

We stress that the above proof is inherently non-uniform because for every n, the distinguisher D' must know the value of k for which D distinguishes well between H_n^k and H_n^{k+1}. We now present a proof of the same theorem for the uniform case.

Theorem 4.4 (multiple samples – uniform version): *Let X and Y be efficiently samplable ensembles such that $X \overset{c}{\equiv} Y$. Then, for every polynomial $p(\cdot)$, the ensembles $\overline{X} = \{(X_n^{(1)}, \ldots, X_n^{(p(n))})\}_{n\in\mathbb{N}}$ and $\overline{Y} = \{(Y_n^{(1)}, \ldots, Y_n^{(p(n))})\}_{n\in\mathbb{N}}$ are computationally indistinguishable.*

Proof: The proof begins exactly as above. That is, based on a contradicting assumption that there exists a PPT distinguisher D that distinguishes $\overline{X}$ from $\overline{Y}$ with non-negligible probability, we have that for some polynomial q and infinitely many n's

$$\left|\sum_{i=0}^{p(n)-1} \Pr[D(H_n^i) = 1] - \Pr[D(H_n^{i+1}) = 1]\right| \geq \frac{1}{q(n)}$$

where the hybrid variable are as defined above. We now construct a PPT distinguisher D' for a single sample of X_n and Y_n. Upon input α, D' chooses a random $i \in_R \{0, \ldots, p(n)-1\}$, generates

[1] The efficient samplability of X and Y is needed for constructing the vector $\overline{H}_n$.

the vector $\overline{H}_n = (X_n^{(1)}, \ldots, X_n^{(i)}, \alpha, Y_n^{(i+2)}, \ldots, Y_n^{(p(n))})$, invokes D on the vector $\overline{H}_n$, and outputs whatever D does. Now, if α is distributed according to X_n, then $\overline{H}_n$ is distributed exactly like H_n^{i+1}. In contrast, if α is distributed according to Y_n, then $\overline{H}_n$ is distributed exactly like H_n^i. (Note that we use the independence of the samples in making this argument.) Furthermore, each i is chosen with probability exactly $1/p(n)$. Therefore,

$$\Pr[D'(X_n) = 1] = \frac{1}{p(n)} \cdot \sum_{i=0}^{p(n)-1} \Pr[D(H_n^{i+1}) = 1]$$

and

$$\Pr[D'(Y_n) = 1] = \frac{1}{p(n)} \cdot \sum_{i=0}^{p(n)-1} \Pr[D(H_n^{i}) = 1]$$

It therefore follows that:

$$\begin{aligned}
\left|\Pr[D'(X_n) = 1] - \Pr[D'(Y_n) = 1]\right| &= \frac{1}{p(n)} \cdot \left|\sum_{i=0}^{p(n)-1} \Pr[D(H_n^{i+1}) = 1] - \sum_{i=0}^{p(n)-1} \Pr[D(H_n^{i}) = 1]\right| \\
&= \frac{1}{p(n)} \cdot \left|\Pr[D(H_n^{p(n)}) = 1] - \Pr[D(H_n^{0}) = 1]\right| \\
&= \frac{1}{p(n)} \cdot \left|\Pr[D(\overline{X}_n) = 1] - \Pr[D(\overline{Y}_n) = 1]\right| \\
&\geq \frac{1}{p(n)q(n)}
\end{aligned}$$

in contradiction to the indistinguishability of a single sample. ■

The hybrid technique. The hybrid proof technique is used in many proofs of cryptographic constructions and is considered a basic technique. Note that there are three conditions for using it. First, the extreme hybrids are the same as the original distributions (for the multiple sample case). Second, the capability of distinguishing neighbouring hybrids can be translated into the capability of distinguishing single samples of the distribution. Finally, the number of hybrids is polynomial (and so the degradation of distinguishing success is only polynomial).

On the danger of induction arguments. A natural way to prove Theorem 4.4 is by induction. Namely, the base case is $X \overset{c}{\equiv} Y$. Now, for every i, denote by $\overline{Z}_i$ the prefix of vector $\overline{Z}$ of length i. Then, by the indistinguishability of single samples of X and Y, it follows that $\overline{X}_i \overset{c}{\equiv} \overline{Y}_i$ implies that $\overline{X}_{i+1} \overset{c}{\equiv} \overline{Y}_{i+1}$. In order to prove this inductive step, note that $\overline{X}_i$ can be efficiently constructed (using efficient samplability). The proof that this suffices follows from a similar argument to our proof above.

We note that the above argument, as such, may fail. In particular, what we obtain is that for every distinguisher of $\overline{X}_{i+1}$ from $\overline{Y}_{i+1}$ there exists a distinguisher of $\overline{X}_i$ from $\overline{Y}_i$. Applying this $p(n)$ times, we obtain a distinguisher for X and Y, from a distinguisher for $\overline{X}$ and $\overline{Y}$, thereby providing the necessary contradiction. The problem is that the resulting distinguisher for the single-sample case of X and Y may not run in polynomial-time. For example, consider the case that the induction is such that the distinguisher for $\overline{X}_i$ and $\overline{Y}_i$ runs twice as long as the distinguisher for $\overline{X}_{i+1}$ from $\overline{Y}_{i+1}$. Then, if the original distinguisher for $\overline{X}$ and $\overline{Y}$ runs in time $q(n)$, the final distinguisher for the single-sample case would run in time $2^{p(n)}q(n)$. Thus, no contradiction to the single-sample case is obtained. The same problem arises with the distinguishing probability. That is, consider the case that the induction is such that the distinguisher for $\overline{X}_i$ and $\overline{Y}_i$ succeeds with half the probability that the distinguisher for $\overline{X}_{i+1}$ from $\overline{Y}_{i+1}$ succeeds. Then, if the original distinguisher succeeds with probability $1/2$, the single-sample distinguisher still only succeeds with probability $1/2 \cdot 1/2^{p(n)}$. For the above reasons, induction arguments are typically not used. If they *are* used, then these issues must be *explicitly* dealt with.

Uniform versus non-uniform reductions. We remark that the above two theorems are incomparable. The first makes a stronger assumption (namely, indistinguishability for non-uniform distinguishers for a single sample), but also has a stronger conclusion (again, indistinguishability for non-uniform distinguishers for multiple samples). In contrast, the second theorem makes a weaker assumption (requiring indistinguishability only for uniform distinguishers) but reaches a weaker conclusion. Despite this, it is important to note that a uniform reduction is always preferable over a non-uniform one. This is because the *proof* that we supplied for the uniform version proves both theorems simultaneously. Thus, we also prefer uniform proofs, when we have them. Having said this, we don't always know how to provide uniform reductions, and in some cases (as we will see later in the course) it is actually impossible as the non-uniformity is inherent.

4.1.2 Pseudorandomness

Given the definition of computational indistinguishability, it is easy to define pseudorandomness:

Definition 4.5 (pseudorandom ensembles): *An ensemble $X = \{X_n\}_{n\in\mathbb{N}}$ is called* pseudorandom *if there exists a polynomial $l(n)$ such that X is computationally indistinguishable from the uniform ensemble $U = \{U_{l(n)}\}_{n\in\mathbb{N}}$.*

The reason that we don't just define $\{X_n\} \overset{c}{\equiv} \{U_n\}$ is because X_n may range over strings of length poly(n) and not just n.

We stress that pseudorandom ensembles may be very far from random. The point is that they cannot be distinguished in *polynomial-time*. In the next lecture, we will construct pseudorandom generators from one-way permutations. Such generators yield pseudorandom distributions that are clearly far from random. However, the construction relies on the existence of one-way functions. We remark that the existence of pseudorandom ensembles that are far from random can be proved *unconditionally*; see [5, Section 3.2.2].

Further Reading

There is much to be said about computational indistinguishability and pseudorandomness that we will *not* have time to cover in class. It is highly recommended to read Sections 3.1 and 3.2 in [5]. (Much of this material is covered in class, but there is other important material that we skip, like statistical closeness and its relation to computational indistinguishability. Also, more motivation is provided in [5] than we have time to cover here.)

4.2 Pseudorandom Generators

Intuitively speaking, a pseudorandom generator is an *efficient* deterministic algorithm G that *stretches* a short random seed into a long *pseudorandom* string. Pseudorandom generators were first defined in [3].

Definition 4.6 (pseudorandom generators): *A* pseudorandom generator *is a deterministic polynomial-time algorithm G satisfying the following two conditions:*

1. Expansion: *There exists a function $l : \mathbb{N} \to \mathbb{N}$ such that $l(n) > n$ for all $n \in \mathbb{N}$, and $|G(s)| = l(|s|)$ for all s.*

2. Pseudorandomness: *The ensemble $\{G(U_n)\}_{n\in\mathbb{N}}$ is pseudorandom.*

We note that constructing a pseudorandom generator even for the case of $l(n) = n+1$ is non-trivial. Specifically, in this case, there are 2^n possible pseudorandom strings of length $n+1$, in contrast to $2 \cdot 2^n$ possible random strings. Thus, the pseudorandom strings make up only half the possible space. This implies that "with enough time", it is trivial to distinguish $\{G(U_n)\}$ from $\{U_{n+1}\}$. We will also see later that the existence of such a pseudorandom generator (with $l(n) = n+1$) already implies the existence of one-way functions.

4.2.1 Pseudorandom Generators from One-Way Permutations

In this section we will show how to construct pseudorandom generators that stretch the seed by one bit, under the assumption that one-way *permutations* exist (this result was proven in [17]). In the next section, we will show how to then expand this to any polynomial expansion factor.

Let f be a one-way permutation and let b be a hard-core predicate of f. The idea behind the construction is that given $f(U_n)$, it is hard to guess the value of $b(U_n)$ with probability that is non-negligibly higher than 1/2. Thus, intuitively, $b(U_n)$ is indistinguishable from U_1. Since f is a permutation, $f(U_n)$ is uniformly distributed. Therefore, $\{(f(U_n), b(U_n))\}$ is indistinguishable from U_{n+1} and so constitutes a pseudorandom generator.

Theorem 4.7 *Let f be a one-way permutation, and let b be a hard-core predicate of f. Then, the algorithm $G(s) = (f(s), b(s))$ is a pseudorandom generator with $l(n) = n+1$.*

Proof: We have already seen the intuition and therefore begin directly with the proof. Assume, by contradiction, that there exists a PPT distinguisher D and a polynomial $p(\cdot)$ such that for infinitely many n's

$$|\Pr[D(f(U_n), b(U_n)) = 1] - \Pr[D(U_{n+1}) = 1]| \geq \frac{1}{p(n)}$$

As a first step to constructing an algorithm A to guess $b(x)$ from $f(x)$, we show that D can distinguish $(f(x), b(x))$ from $(f(x), \overline{b}(x))$ where $\overline{b}(x) = 1 - b(x)$. In order to see this, first note that

$$\Pr[D(f(U_n), U_1) = 1] = \frac{1}{2} \cdot \Pr[D(f(U_n), b(U_n)) = 1] + \frac{1}{2} \cdot \Pr[D(f(U_n), \overline{b}(U_n)) = 1]$$

because with probability 1/2 the bit U_1 equals $b(U_n)$, and with probability 1/2 it equals $\overline{b}(U_n)$. Given this, we have:

$$\begin{aligned}
&|\Pr[D(f(U_n), b(U_n)) = 1] - \Pr[D(f(U_n), U_1) = 1]| \\
&= |\Pr[D(f(U_n), b(U_n)) = 1] - \frac{1}{2} \cdot \Pr[D(f(U_n), b(U_n)) = 1] - \frac{1}{2} \cdot \Pr[D(f(U_n), \overline{b}(U_n)) = 1]| \\
&= \frac{1}{2}|\Pr[D(f(U_n), b(U_n)) = 1] - \Pr[D(f(U_n), \overline{b}(U_n)) = 1]|
\end{aligned}$$

By our contradicting assumption, and noting that $\{(f(U_n), U_1)\} \equiv \{U_{n+1}\}$, we have that for infinitely many n's

$$\left|\Pr[D(f(U_n), b(U_n)) = 1] - \Pr[D(f(U_n), \overline{b}(U_n)) = 1]\right| \geq \frac{2}{p(n)}$$

Without loss of generality, assume that for infinitely many n's it holds that

$$\Pr[D(f(U_n), b(U_n)) = 1] - \Pr[D(f(U_n), \overline{b}(U_n)) = 1] \geq \frac{2}{p(n)} \ .$$

We now use D to construct an algorithm A that guesses $b(x)$. Upon input $y = f(x)$ for some x, algorithm A works as follows:

1. Uniformly choose $\sigma \in_R \{0, 1\}$
2. Invoke D upon (y, σ).
3. If D returns 1, then output σ. Otherwise, output $\overline{\sigma}$.

It remains to analyze the success probability of A. Intuitively, A succeeds because D outputs 1 when $\sigma = b(x)$ with probability $2/p(n)$ higher than it outputs 1 when $\sigma = \overline{b}(x)$. Formally,

$$\begin{aligned}
\Pr[A(f(U_n)) {=} b(U_n)] &= \frac{1}{2}\Pr[A(f(U_n)) = b(U_n) \mid \sigma = b(U_n)] + \frac{1}{2}\Pr[A(f(U_n)) = b(U_n) \mid \sigma = \overline{b}(U_n)] \\
&= \frac{1}{2} \cdot \Pr[D(f(U_n), b(U_n)) = 1] + \frac{1}{2} \cdot \Pr[D(f(U_n), \overline{b}(U_n)) = 0] \\
&= \frac{1}{2} \cdot \Pr[D(f(U_n), b(U_n)) = 1] + \frac{1}{2}\Big(1 - \Pr[D(f(U_n), \overline{b}(U_n)) = 1]\Big) \\
&= \frac{1}{2} + \frac{1}{2} \cdot \Pr[D(f(U_n), b(U_n)) = 1] - \frac{1}{2} \cdot \Pr[D(f(U_n), \overline{b}(U_n)) = 1] \\
&\geq \frac{1}{2} + \frac{1}{2} \cdot \frac{2}{p(n)} = \frac{1}{2} + \frac{1}{p(n)}
\end{aligned}$$

in contradiction to the assumption that b is a hard-core predicate of f. ■

4.2.2 Increasing the Expansion Factor

In this section, we show that the expansion factor of any pseudorandom generator can be increased by any polynomial amount. We do not prove this theorem and provide only an outline of the idea behind the proof. See [5, Section 3.3.2] for more details.

Theorem 4.8 *If there exists a pseudorandom generator G_1 with $l_1(n) = n + 1$, then for any polynomial $p(n) > n$, there exists a pseudorandom generator G with $l(n) = p(n)$.*

Proof Idea: The construction of G from G_1 works as follows:

1. Let $s \in \{0, 1\}^n$ be the seed, and denote $s_0 = s$.
2. For every $i = 1, \ldots, p(n)$, compute $G_1(s_{i-1}) = (\sigma_i, s_i)$, where $\sigma_i \in \{0, 1\}$ and $s_i \in \{0, 1\}^n$.
3. Output $\sigma_1, \ldots, \sigma_{p(n)}$

In other words, G works by extracting a single bit σ_1 from $G_1(s)$ and using the remaining n bits in G_1's output as the seed in the next iteration. By the pseudorandomness of G_1, it follows that one cannot distinguish the case that G_1's input seed is truly random from the case that it is pseudorandom. Thus, the output of $G_1(s_1)$ is also pseudorandom, and so on for any polynomial number of iterations.

The actual proof is by a hybrid argument, where we define H_n^i to be a string with a length i prefix that is truly random, and a length $p(n) - i$ suffix that is pseudorandom. Note that if neighbouring hybrids can be distinguished, then this can be reduced to distinguishing $\{G_1(U_n)\}$ from $\{U_{n+1}\}$. ■

4.2.3 Pseudorandom Generators and One-Way Functions

We note, without proof, that the existence of pseudorandom generators implies the existence of one-way functions. (Here, we see the "minimality" of one-way functions as an assumption.) Intuitively, given a pseudorandom generator with $l(n) = 2n$, the function $f(x) = G(x)$ is one-way. (Note that by Section 4.2.2, this implies that the existence of a generator that stretches by even one bit implies the existence of one-way functions.) The idea behind this construction is that if it is possible to invert $f(U_n)$ with probability $1/\text{poly}(n)$, then this advantage can be used to distinguish $G(U_n)$ from U_{2n} with non-negligible advantage. Namely, construct a distinguisher D that upon input y, runs the inverter for f upon y. Then, if the inverter for f succeeds in finding a "seed" x such that $y = G(x)$, then it is almost certain that y is pseudorandom and so D outputs 1. In every other case, D outputs 0. It follows that D outputs 1 upon input $G(U_n)$ with probability that is non-negligibly higher than when it receives input U_{2n} (because in such a case, there is almost certainly no preimage seed). More details can be found in [5, Section 3.3.6].

We also note that pseudorandom generators can be constructed from *any* one-way function [13]. (Unfortunately, this is a very complex construction and so we will not see it here.) We therefore obtain the following theorem:

Theorem 4.9 *Pseudorandom generators exist if and only if one-way functions exist.*

Lecture 5

Pseudorandom Functions and Zero Knowledge

In this lecture, we introduce the notion of pseudorandom functions and show how to construct them from pseudorandom generators (which can in turn be constructed from any one-way function). The notion of pseudorandom functions, and the construction that we will see here, were presented in [7].

5.1 Pseudorandom Functions

5.1.1 Definitions

Intuitively, a pseudorandom function is one that cannot be distinguished from a random one. Defining this notion, however, is non-trivial because it is not possible to hand a distinguisher a description of the function and ask it to decide whether or not it is random. This is due to the fact that the description of a random function from n input bits to a single output bit is of size 2^n. We therefore provide the distinguisher with *oracle access* to a function that is either random or the one that we have constructed. An efficient function f is said to be *pseudorandom* if no efficient oracle-machine/distinguisher can tell whether its oracle computes a truly random function or the function f. For simplicity, we present the definition for the case of functions that map n bit inputs to n bit outputs (modifying it to map n bit inputs to $l(n)$ bit outputs, for any polynomial $l(n)$ is straightforward).

Preliminaries. We begin by defining the notion of a function ensemble. A function ensemble is a sequence $F = \{F_n\}_{n\in\mathbb{N}}$ of random variables such that the random variable F_n assumes values in the set of functions mapping n-bit inputs to n-bit outputs. We denote the uniform function ensemble by $H = \{H_n\}_{n\in\mathbb{N}}$.

A function ensemble is called efficiently computable, if it has a succinct representation, and if it can be efficiently evaluated. More formally, we require that there exists a PPT algorithm I and a mapping ϕ from strings to functions so that $\phi(I(1^n))$ and F_n are identically distributed. Here, I is the description of the function, and we denote by f_i the function $\phi(i)$ for i in the range of I. In addition to the above, we require the existence of a polynomial-time algorithm V such that $V(i, x) = f_i(x)$ for every i in the range of $I(1^n)$ and for every $x \in \{0, 1\}^n$. Thus, V is an efficient evaluation algorithm.

We are now ready to present the definition:

[*] Lecture notes for a graduate course in the theory of cryptography. Yehuda Lindell, Bar-Ilan University, Israel, 2005.

Definition 5.1 (pseudorandom function ensembles): *A function ensemble $F = \{F_n\}_{n\in\mathbb{N}}$ is* pseudorandom *if for every probabilistic polynomial-time oracle machine D, every polynomial $p(\cdot)$ and all sufficiently large n's*

$$\left|\Pr\left[D^{F_n}(1^n) = 1\right] - \Pr\left[D^{H_n}(1^n) = 1\right]\right| < \frac{1}{p(n)}$$

We will always consider *efficiently computable* pseudorandom ensembles in this course. For shorthand, when we refer to pseudorandom functions, we really mean efficiently computable pseudorandom function ensembles. We will also refer to the key of the function, which is just the succinct representation as output by $I(1^n)$.

For simplicity, we have presented the definition for the case that the key-length, input-length and output-length are all the same. We remark that this can be easily generalized so that the input and output lengths are allowed to be different polynomials in the length of the key.

5.2 Constructions of Pseudorandom Functions

We now show how to construct pseudorandom functions from pseudorandom generators. In order to motivate the construction, consider the following toy example. Let G be a pseudorandom generator with $l(n) = 2n$ (i.e., G is length doubling), and denote $G(s) = (G_0(s), G_1(s))$, where $|s| = |G_0(s)| = |G_1(s)| = n$. Then, the four strings $G_0(G_0(s))$, $G_0(G_1(s))$, $G_1(G_0(s))$, and $G_1(G_1(s))$ are all pseudorandom, *even when viewed all together.* In order to see this, consider a hybrid distribution of $G_0(U_n^{(0)})$, $G_0(U_n^{(1)})$, $G_1(U_n^{(0)})$, and $G_1(U_n^{(1)})$. In this hybrid distribution, the random variable $U_n^{(b)}$ takes the place of $G_b(s)$. Therefore, if it is possible to distinguish the hybrid distribution from the original distribution, then we would be able to distinguish between $\{(G_0(s), G_1(s))\}$ and U_{2n} (in contradiction to the pseudorandomness of G). Likewise, if we could distinguish the hybrid distribution from U_{4n}, then we would distinguish either $G(U_n^{(0)}) = G_0(U_n^{(0)}), G_1(U_n^{(0)})$ from U_{2n}, or $G(U_n^{(1)}) = G_0(U_n^{(1)}), G_1(U_n^{(1)})$ from U_{2n}. Once again, this contradicts the pseudorandomness of G.

Looking at this differently, it follows that we have obtained a pseudorandom function mapping two bits to n bits. Specifically, let s be the key. Then, define $f_s(b_1, b_2) = G_{b_2}(G_{b_1}(s))$. By our above argument, this constitutes a pseudorandom function. (In order to be convinced of this, notice that a random function is just a long random string, where a different part of the string is allocated for every possible input.) The full construction below works in the same way, except that the pseudorandom generator is applied n times, once for each input bit.

Construction 5.2 (pseudorandom functions): *Let G be a deterministic function that maps inputs of length n into outputs of length $2n$. Denote by $G_0(s)$ the first n bits of G's output, and by $G_1(s)$ the second n bits of G's output. For every $s \in \{0,1\}^n$, define the function $f_s : \{0,1\}^n \to \{0,1\}^n$ as:*

$$f_s(\sigma_1\sigma_2\cdots\sigma_n) = G_{\sigma_n}(\cdots(G_{\sigma_2}(G_{\sigma_1}(s)))\cdots)$$

Let F_n be the random variable defined by uniformly selecting $s \in_R \{0,1\}^n$ and setting $F_n = f_s$. Let $F = \{F_n\}_{n\in\mathbb{N}}$ be the resulting function ensemble.

This construction can be viewed as a full binary tree of depth n, defined as follows. The value at the root equals the key/seed s. For any node of value s', the left son of s' has value $G_0(s')$ and the right son of s' has value $G_1(s')$. The function on an input value $x = x_1 \cdots x_n$ is then equal to the value at the *leaf* that is reached by traversing the tree according to x (that is, $x_i = 0$ means "go left in the tree", and $x_i = 1$ means "go right"). We stress that the function has a *fixed input length,* and only values in the leaves are output. (Exercise: show that if the internal nodes of the tree are also output, then the construction is no longer secure.) Notice also that the size of the tree is exponential in n; in particular, there are 2^n leaves. Nevertheless, we never need to construct and hold the tree explicitly. Rather, the values on the path (and so the value of the appropriate leaf) can be efficiently obtained given the key s.

We now claim that the above construction yields a pseudorandom function when G is "properly" instantiated.

Theorem 5.3 *If the function G is a pseudorandom generator with $l(n) = 2n$, then Construction 5.2 is an efficiently computable pseudorandom function ensemble.*

Proof Sketch: The proof of this theorem works by a hybrid argument. Let H_n^i be a full binary tree of depth n where the nodes of levels 0 to i are labelled with truly random values, and the nodes of levels $i+1$ to n are constructed as in Construction 5.2 (given the labels of level i). We note that in H_n^i, the labels in nodes 0 to $i-1$ are actually irrelevant. The function associated with this tree is obtained as in Construction 5.2 by outputting the appropriate values in the leaves.

Notice that H_n^n equals the truly random function H_n, because all the leaves are given truly random values. On the other hand, H_n^0 equals Construction 5.2 exactly (because only the key is random). Using a hybrid argument, we obtain that if Construction 5.2 can be distinguished from a truly random function with non-negligible probability, then there must be a k such H_n^k can be distinguished from H_n^{k+1} with non-negligible probability. We use this to distinguish the pseudorandom generator from random. Intuitively this follows because the only difference between the distributions is that in H_n^{k+1} the pseudorandom generator G is applied one more time on the way from the root to the leaves of the tree. The actual proof is more tricky than this because we cannot hold the entire $(k+1)^{\text{th}}$ level of the tree (it may be exponential in size). Rather, let $t(n)$ be the maximum running-time of the distinguisher D who manages to distinguish Construction 5.2 from a random function. It follows that D makes at most $t(n)$ oracle queries. Now, let D' be a distinguisher for G that receives an input of length $2n \cdot t(n)$ that is either truly random or $t(n)$ independent samples of $G(U_n)$. (Recall that by Theorem 4.4, all of these samples together should be indistinguishable from $U_{2n \cdot t(n)}$.) Then, D' answers D's oracle queries as follows, initially holding an empty binary tree. Upon receiving a query $x = x_1 \cdots x_n$ from D, distinguisher D' uses $x_1 \cdots x_k$ to reach a node on the k^{th} level (filling all values to that point with arbitrary values – they are of no consequence). Then, D' takes one of its input samples (of length $2n$) and labels the left son of the reached node with the first half of the sample and the right son with the second half of the sample. D' then continues to compute the output as in Construction 5.2. Note that in future queries, if the input x brings D' to a node that has already been filled, then D' answers consistently to the value that already exists. Otherwise, D' uses a new sample from its input. (Notice that D' works by filling the tree "on the fly" and depending on D's queries. It does this because the full tree is too large to hold.)

Now, if D' receives random input, then it answers D' exactly according to the distribution H_n^{k+1}. This holds because all the values in level $k+1$ in the tree (dynamically) constructed by D' are random. On the other hand, if D' receives pseudorandom input, then it answers D' exactly according to H_n^k because the values in level $k+1$ are pseudorandom. (Notice that the seeds to these pseudorandom values are not known to D' but this makes no difference to the result.) We conclude that D' distinguishes multiple samples of G from random, in contradiction. ■

Combining Theorem 5.3 with Theorem 4.9, (and noting trivially that one-way functions can be constructed from pseudorandom functions), we obtain the following:

Corollary 5.4 *Pseudorandom functions exist if and only if one-way functions exist.*

Variations. We note that a number of variations of pseudorandom functions have been considered and are very useful. First, it is possible to construct pseudorandom functions that have variable input length. Loosely speaking, this is obtained by following Construction 5.2, and then outputting the output of a pseudorandom generator G' applied to the label in the reached node. Pseudorandom permutations are also of importance (e.g., as so-called block ciphers), and can be constructed from pseudorandom functions. We refer the reader to [5, Sections 3.6 and 3.7] for details (as well as for a full proof of Theorem 5.3).

5.2.1 Applications

Pseudorandom functions have many applications. We have already seen in the course "Introduction to Cryptography" (89-656) that pseudorandom functions can be used to obtain *secure private-key encryption* (under CPA or even CCA-security), and secure *message authenticate codes.* Thus, relying on this prior knowledge, we already have that the basic tasks of private-key cryptography can be achieved assuming only the *minimal assumption* of the existence of (weak) one-way functions.

Another interesting application of pseudorandom functions is in challenge/response protocols for entity authentication. Specifically, in order to prove the identity of a user, it is possible to first share the key k of a pseudorandom function between the user and the server. Then, upon a login request, the server can send a random challenge c and allow access if and only if it receives the response $r = f_k(c)$ back. Due to the pseudorandom property of the function $f_k(\cdot)$, the probability that an adversary can guess r without knowing k is negligible.

Using pseudorandom functions – a general paradigm. The above example regarding challenge/response protocols brings us to a general paradigm regarding the use of pseudorandom functions. In the first step of designing a system, a truly random function is used. The security of the system is proven in this case. Next, the truly random function is replaced by a pseudorandom one, and it is proved that this can make at most a negligible difference. This paradigm is a very powerful and useful one for designing secure protocols.

5.3 Zero-Knowledge Interactive Proof Systems

General remark: *The notions of interactive proofs and zero-knowledge are very strange at first, and they require a significant amount of motivating discussion. I have decided to keep this discussion short in the lecture notes, and rely on motivation that will be provided in class. The notes here are therefore not fully self-contained and should be considered together with what is taught in class.*

Classically, proofs are "strings" that demonstrate the validity of some statement. This is clearly the case in the world of mathematics, but is also true in computer science. For example, consider the notion of NP-proofs. Informally speaking, let L be an NP-language and let R_L be an appropriate NP-relation for L. Then, by the definition of $\mathcal{NP}$, there exists a polynomial-time algorithm A that outputs 1 upon receiving $(x, w) \in R_L$, and for $x \notin L$ outputs 0 when receiving (x, v) for every possible v. Essentially, the string x is a statement, and the string w is a *proof* that $x \in$

L. Furthermore, the behaviour of the algorithm A ensures that these proofs have the following properties (that are essential for proofs to have meaning):

1. **Completeness:** A proof system is complete if every valid statement $x \in L$ has a proof of this fact. By the definition of $\mathcal{NP}$, every $x \in L$ has a w such that $(x, w) \in R_L$. Therefore, it indeed holds that for every $x \in L$, there exists a proof causing A to output 1 (and so to "be convinced").

2. **Soundness:** A proof system is sound if invalid statements $x \notin L$ do not have false proofs. Notice that A never outputs 1 when $x \notin L$ (irrespective of the value of v). Therefore, A is never convinced of false claims, and so soundness holds.

This above view of proofs (as strings) is actually rather limiting. In particular, a more general definition of a proof is any method of verifying the validity of a statement, while preserving the properties of completeness and soundness. Such a view would allow, for example, an interactive process between the prover and verifier (rather than forcing the prover to write a static string that is later checked by the verifier). A further relaxation would allow the completeness and soundness properties to hold *except with negligible probability* (instead of always). These relaxations of the notion of a proof system turn out to be very useful. Beyond increasing the power of what can be proven and efficiently verified (this is not our topic here), it enables us to consider additional properties of the proof system.

Zero-knowledge proof systems. In the cryptographic context, we are interested in constructing proofs that reveal nothing beyond the validity of the statement being proved. For example, assume that Alice and Bob communicate with each other via encrypted email, using a shared secret key K. Furthermore, assume that at some stage they are ordered (say, by a court order) to reveal one of the encrypted messages. This order causes the following problem. If Alice and Bob just produce the message, then how can the court know that the produced message was the one encrypted. On the other hand, if Alice and Bob present their secret key K, then *all* of their encrypted email can be read (whereas the court only required the revealing of a single message). A solution to this problem is therefore for Alice and Bob to present the plaintext message, and *prove* that this message was indeed the encrypted one. Notice that in order to protect the other encrypted messages, this proof must have the property that it reveals nothing about the encryption key (or the other encrypted messages).

This motivating discussion leads us to another question of how to define what it means for a proof to reveal nothing beyond the validity of the statement being proved. The definition of this notion follows what is known as the *simulation paradigm.* Loosely speaking, this states that a verifier learns nothing from a proof if it can efficiently generate everything that it saw in the proof, by itself. In other words, the verifier's view in the protocol can be simulated, given only its input. Notice that this implies that if the verifier learned something from the proof (e.g., something about a different encrypted message), then it could have learned this by itself (without seeing the proof). In our above example, it is therefore possible to conclude that the verifier could not have learned anything about a different encrypted message from the proof (or this would contradict the security of the encryption scheme).

This notion seems strange initially, but will become clearer after we see some examples. See [5, Section 4.1] for more motivating discussion. The notion of zero-knowledge interactive proofs was introduced by [12] (at the same time, interactive proofs were independently studied by [1]). These papers have had far-reaching ramifications on both cryptography and complexity.

5.3.1 Interactive Proofs

In the subsequent sections, we will refer to interactive Turing machines. Intuitively, these are Turing machines that have communication tapes, in addition to their input, random, work and output tapes. More specifically, a Turing machine has a read-only input tape, a write-only output tape, a read-only random tape (modelling its random coin tosses), and a read-and-write work tape. An interactive Turing machine is a Turing machine with two additional tapes:

1. A write-only outgoing communication tape
2. A read-only incoming communication tape

An interaction between a pair of interactive Turing machines is defined by making the outgoing communication tape of one machine equal the incoming communication tape of the other (and vice versa). Thus, the machines can communicate via these communication tapes. We note that in an interaction between these machines, only one machine is active at any one time. There are a number of ways of modelling this and we have decided to leave it at an intuitive level here. It suffices to think that one machine is designated to start the computation, and when it enters a special wait state (typically, after writing on its outgoing communication tape or halting after writing output), the other machine is activated. For formal definitions of interactive machines, see [5, Section 4.2.1].

We now introduce notation that will be used later. Let A and B be interactive machines. Then, we denote by $\langle A, B\rangle(x)$ the output of B after interacting with A upon *common input* x (i.e., both parties have x written on their input tape).

Interactive proofs. An interactive proof is a protocol between two parties (i.e., interactive Turing machines). One party is known as the prover, and the other as the verifier. The verifier is always required to be efficient (i.e., polynomial-time), whereas the prover is sometimes allowed to run in superpolynomial-time. This is consistent with the notion of proofs that are efficient to check (like in $\mathcal{NP}$), even though they may be hard to generate. We note that in cryptographic contexts, we will typically only really be interested in the case that all parties run in polynomial-time. We will call a protocol an interactive proof system if it has completeness (meaning that an honest prover will successfully convince an honest verifier of a true statement with “high” probability) and soundness (meaning that a dishonest prover will only be able to convince an honest verifier of a false statement with “low” probability). We note that the output of the (honest) verifier is always 1 (meaning accept) or 0 (meaning reject).

Definition 5.5 (interactive proof system): *A pair of interactive machines (P, V) is called an* interactive proof system for a language L *if machine V runs in probabilistic polynomial-time and the following two conditions hold:*

- **Completeness:** *For every $x \in L$,* $\Pr[\langle P, V\rangle(x) = 1] \geq 2/3$
- **Soundness:** *For every $x \notin L$ and every interactive machine P^*,* $\Pr[\langle P^*, V\rangle(x) = 1] \leq 1/3$

Proof systems come in many variants and with many different properties. Some of the more important variants and properties (for our purposes) are defined as follows:

1. *A proof system is said to have* perfect completeness *if for every $x \in L$,* $\Pr[\langle P, V\rangle(x) = 1] = 1$.
2. *A proof system is said to have* negligible soundness error *if for every $x \notin L$ and every interactive machine P^*,* $\Pr[\langle P^*, V\rangle(x) = 1] < \mu(|x|)$*, for some negligible function $\mu(\cdot)$.*

3. *A proof system is said to have* computational soundness *if the soundness condition is only guaranteed to hold for probabilistic polynomial-time interactive machines P^*. A system with computational soundness is called an* argument system.

4. *A proof system is said to have an* efficient prover *if there exists some auxiliary input w such that the honest prover P can be implemented in probabilistic polynomial-time, given x and w.*

In this course, we will only see proofs with perfect completeness. Regarding soundness, we are typically only interested in proofs with negligible soundness. However, in the exercise, you are asked to show that the soundness error can be reduced to negligible (you are asked to prove this only in the more simple case where there is perfect completeness; however, the claim also holds in the general case). Finally, we remark that in cryptography efficient provers are typically required and computational soundness typically suffices.

We note that interactive proofs within themselves are fascinating, and one could easily spend a number of lectures studying them. Due to lack of time, we restrict our attention to the cryptographic context only.

Lecture 6

Zero-Knowledge Proofs and Perfect Zero-Knowledge

6.1 Zero Knowledge Proofs – Definitions

As we have described, zero-knowledge proofs are proof systems with the additional property that the verifier learns nothing from the proof, except for being convinced that the statement being proved in indeed true. This is formalized by showing that for every (possibly adversarial) polynomial-time verifier V^*, there exists a *non-interactive simulator* that is given only the common input and outputs a string that is *very close* to the view of V^* in a real proof with P.[1] The requirement on how close the output of the simulator must be to the real view of V^* depends on whether we are interested in *perfect* zero-knowledge (where the output must be identically distributed), *statistical* zero-knowledge (where the output must be statistically close), or *computational* zero-knowledge (where the output must be computationally indistinguishable). We present the definition here for the case of computational zero-knowledge.

Definition 6.1 (computational zero-knowledge): *Let (P, V) be an interactive proof system for some language L. We say that (P, V) is* computational zero-knowledge *if for every probabilistic polynomial-time interactive machine V^* there exists a probabilistic polynomial-time simulator S^* such that*

$$\{\langle P, V^* \rangle(x)\}_{x\in L} \overset{c}{\equiv} \{S^*(x)\}_{x\in L} \ .$$

If the above ensembles are identically distributed then we say that (P, V) is perfect zero-knowledge*, and if they are statistically close then we say that (P, V) is* statistical zero-knowledge.

Note that the probability ensembles in the above definition are indexed over true statements only (i.e., over $x \in L$).

There are many important variants of this definition. For example, the simulator is often allowed to run in expected polynomial-time (rather than strict polynomial-time). Also, *auxiliary-input* is typically considered. This means that the prover and verifier are provided auxiliary input, and the zero-knowledge property must be preserved for every common input x and *every* auxiliary input z

[*] Lecture notes for a graduate course in the theory of cryptography. Yehuda Lindell, Bar-Ilan University, Israel, 2005.

[1] Formally, the view of the verifier includes its input, random coins and the messages that it receives during the execution. Our formal definition below will require that the simulator generates output that is close to the output of V^*. Since V^* could just output its view (and since its output can be efficiently computed from its view), these formalizations are actually equivalent.

that the verifier V^* receives. We remark that auxiliary-input zero-knowledge is the default, as it is needed for obtaining properties like closure under sequential composition.

One more variant that is often considered is black-box zero-knowledge. This considers the case that the simulator receives only black-box (or oracle) access to the adversarial verifier. Furthermore, a single simulator works for all verifiers. We note that black-box zero-knowledge implies auxiliary-input zero-knowledge, and until recently, all zero-knowledge protocols were black-box. Nonblack-box constructions of zero-knowledge protocols are beyond the scope of this course, and all the proofs that we will see are black-box zero-knowledge. Formally,

Definition 6.2 (black-box zero-knowledge): *Let (P, V) be an interactive proof system for some language L. We say that (P, V) is* black-box zero-knowledge *if there exists a probabilistic polynomial-time simulator $\mathcal{S}$ such that for every probabilistic polynomial-time interactive machine V^* it holds that*

$$\left\{\langle P(x,y), V^*(x,z,r)\rangle\right\}_{x\in L, y\in R_L(x), z,r\in\{0,1\}^*} \overset{\rm c}{\equiv} \left\{\mathcal{S}^{V^*(x,z,r;\cdot)}(x)\right\}_{x\in L, y\in R_L(x), z,r\in\{0,1\}^*}$$

where $V^*(x,z,r;\cdot)$ denotes the next-message function *of the interactive machine V^* with common input x, auxiliary input z and random-tape r (i.e., the next message function of V^* receives a message history h and outputs $V^*(x,z,r;h)$).*

Trivial zero-knowledge proofs. We note that any language $L \in \mathcal{BPP}$ has a zero-knowledge proof in which the verifier just runs the $\mathcal{BPP}$-decision machine and outputs its answer. We will therefore be interested only in zero-knowledge proofs for languages that are *not* in $\mathcal{BPP}$.

6.2 Perfect Zero-Knowledge for Diffie-Hellman Tuples

We begin by recalling the decisional Diffie-Hellman problem. Let $\mathcal{G}$ be a probabilistic polynomial-time generation algorithm that on input 1^n outputs the description of a group G of prime order q, with $|q| = n$, and a generator g of G. Then, the decisional Diffie-Hellman assumption relative to $\mathcal{G}$ states that:

$$\{(\mathcal{G}(1^n), g^a, g^b, g^{ab})\}_{n\in\mathbb{N}; a,b\in_R\mathbb{Z}_q} \overset{\rm c}{\equiv} \{(\mathcal{G}(1^n), g^a, g^b, g^c)\}_{n\in\mathbb{N}, a,b,c\in_R\mathbb{Z}_q}$$

Recall that under this assumption, it is possible to carry out secure key exchange, as follows. Let g^a and g^b be A and B's respective public keys, and let a and b be their respective secret keys. Then, in order to communicate, A and B simply locally compute $K = g^{ab}$. In the case that A and B do not already know each other's public-keys, certificates can be used to first exchange g^a and g^b.

An interesting problem that we will consider now is how to prove that a given tuple $(\mathcal{G}(1^n), g^a, g^b, g^c)$ is a Diffie-Hellman tuple. In other words, the aim is to prove that $c = ab$. The reason why such a proof may be required is the same as in our motivating discussion from the previous lecture. Assume, that by court order, A and B are required to reveal the secret key K that they use to communicate with each other. Essentially, this requires proving that $K = g^{ab}$, or in other words, that $(\mathcal{G}(1^n), g^a, g^b, K)$ is a Diffie-Hellman tuple. As we have already discussed, A and B do not want to reveal their secret keys a and b, because this would actually reveal the keys that they have generated with all other parties as well. Therefore, they want to prove that K is their generated key, and nothing else.

For simplicity from now on we will denote a Diffie-Hellman tuple by (g, g^a, g^b, g^{ab}) and will assume that the order q and the description of the group are known. Furthermore, we will denote a tuple by (g, h, y_1, y_2) and say that it is of the Diffie-Hellman type if there exists a value x such that $y_1 = g^x$ and $y_2 = h^x$. (In order to see that this is equivalent to the above formulation, note that for every $h \in G$, there exists a value a such that $g^a = h$. Thus, the existence of an x as required is equivalent to saying that $h = g^a$, $y_1 = g^x$ and $y_2 = h^x = g^{ax}$, as above.) Finally we assume that given any value h and a description of the group G it is possible to efficiently determine if $h \in G$. Let DH denote the set of all Diffie-Hellman tuples. We now present the protocol for the language DH:

Protocol 6.3 (perfect zero-knowledge for Diffie-Hellman tuples):

- **Common input:** *a tuple* $(g, h, y_1, y_2) \in DH$*.*
- **Prover's auxiliary input:** *a value x such that $y_1 = g^x$ and $y_2 = h^x$.*
- **The protocol:**
  1. *The prover P chooses a random value $r \in \mathbb{Z}_q$, computes $A = g^r$ and $B = h^r$, and sends (A, B) to the verifier V.*
  2. *V chooses a random bit $\sigma \in_R \{0, 1\}$ and sends σ to P.*
  3. *P sends $s = \sigma \cdot x + r \bmod q$ to V. (That is, if $\sigma = 0$ then P sends r to V, and if $\sigma = 1$ then P sends $x + r \bmod q$ to V.)*
  4. *V accepts if and only if $A = g^s/y_1^\sigma$ and $B = h^s/y_2^\sigma$. (That is, if $\sigma = 0$ then V accepts if and only if $A = g^s$ and $B = h^s$; if $\sigma = 1$ then V accepts if and only if $A = g^s/y_1$ and $B = h^s/y_2$.)*

Before proceeding to prove the above protocol, we motivate the construction. First, notice that it is an interactive proof. In order to see this, we explain the meaning behind the "challenge" bit σ from the verifier. When V sends $\sigma = 0$, this should be interpreted as a check that P constructed A and B *properly* (in particular, that $\log_g A = \log_h B = r$ for some r). On the other hand, when V sends $\sigma = 1$, this is the actual proof. That is, *assume* that A and B are properly constructed (which is checked in the case that $\sigma = 0$), and that $A = g^s/y_1$ and $B = h^s/y_2$ (which holds if V accepts). Then, it follows that for some r, $g^r = g^s/y_1$ and $h^r = h^s/y_2$. Thus, $y_1 = g^{s-r}$ and $y_2 = h^{s-r}$. Taking $x = s - r$, we have that (g, h, y_1, y_2) is indeed a Diffie-Hellman tuple. In other words, if the prover attempts to cheat by sending an improperly formed first message, then V will detect this with probability 1/2. In contrast, if the prover sends a properly formed first message, then it will only be able to answer in the case of $\sigma = 1$ if the input is really a Diffie-Hellman tuple. This implies a soundness error of 1/2 that can be lowered by *sequentially* repeating the proof many times.

We will discuss the zero-knowledge property after we see the proof. Intuitively, the verifier learns nothing because it either sees a random value r (in the case of $\sigma = 0$), or the value $x + r$ (in the case of $\sigma = 1$), which is also random. Since it only sees one of these values, it learns nothing about x from the proof.

Proposition 6.4 *Protocol 6.3 is a black-box perfect zero-knowledge proof system for the language of Diffie-Hellman tuples, with an efficient prover, perfect completeness and soundness error $\frac{1}{2}$.*

Proof Sketch: The fact that P is an efficient prover (when given b) is immediate from the protocol description. Likewise, when P is honest, it can always convince the honest verifier V. Thus, the protocol has perfect completeness.

In order to prove soundness, we show that if $(g, h, y_1, y_2) \notin DH$, then a cheating prover P^* can correctly answer for at most one choice of σ. Let $(g, h, y_1, y_2) \notin DH$ and let (A, B) be the prover message from P^* to V. There are two cases:

1. *There exists an $r \in \mathbb{Z}_q$ such that $A = g^r$ and $B = h^r$:* In this case, if V chooses $\sigma = 1$, then it always rejects. In order to see this, note that if V accepts, then there exists an s such that $A = g^s/y_1$ and $B = h^s/y_2$. Since in this case, $A = g^r$ and $B = h^r$, it follows that $g^r = g^s/y_1$ and $h^r = h^s/y_2$ and so $y_1 = g^{s-r}$ and $y_2 = h^{s-r}$. This contradicts the assumption that $(g, h, y_1, y_2) \notin DH$.

2. *There exist $r, r' \in \mathbb{Z}_q$, $r \neq r'$, such that $A = g^r$ and $B = h^{r'}$:* In this case, if V chooses $\sigma = 0$, then it always rejects. This is due to the fact that there does not exist any value $s \in \mathbb{Z}_q$ for which it holds that $A = g^s$ and $B = h^s$.

Since V chooses σ uniformly at random *after* receiving (A, B), it follows that for every possible pair (A, B), the verifier V will reject with probability 1/2. Thus, soundness holds.

It remains to prove that the protocol is zero-knowledge. We construct a black-box simulator $\mathcal{S}$ that is given input $(g, h, y_1, y_2) \in DH$ (recall that simulators only need to work for "correct" statements), oracle access to a verifier $V^*((g, h, y_1, y_2), r, z; \cdot)$ and works as follows:

1. $\mathcal{S}$ chooses $\tau \in_R \{0, 1\}$ and a random value $r \in_R \mathbb{Z}_q$.

   (a) If $\tau = 0$, then $\mathcal{S}$ computes $A = g^r$ and $B = h^r$.

   (b) If $\tau = 1$, then $\mathcal{S}$ computes $A = g^r/y_1$ and $B = h^r/y_2$.

   $\mathcal{S}$ queries its oracle V^* with the pair (A, B).

2. Let σ be the oracle reply from V^*.

3. If $\sigma = \tau$, then $\mathcal{S}$ sends its oracle V^* the value r, and outputs whatever V^* does (V^*'s output is its oracle reply given a full transcript).

4. If $\sigma \neq \tau$, then $\mathcal{S}$ returns to Step 1 and starts again with independent coin tosses.

We first claim that if $\sigma = \tau$, then the view of V^* in the simulation with $\mathcal{S}$ is *identical* to its view in a real execution with P. We start with the distribution over the message (A, B). The real prover always sends $(A = g^r, B = h^r)$. In contrast, $\mathcal{S}$ sometimes sends $(A = g^r, B = h^r)$ and sometimes sends $(A = g^r/y_1, B = h^r/y_2)$. However, both of these distributions are identical in the case that the input is a Diffie-Hellman tuple. Namely, since for some x, $y_1 = g^x$ and $y_2 = h^x$, it follows that either $\mathcal{S}$ sends $(A = g^r, B = h^r)$ or it sends $(A = g^{r-x}, B = h^{r-x})$. Since r is chosen uniformly and independently of x, these distributions are the same. The same argument implies that in the case that $\sigma = \tau$, the view of V^* of the last message is also the same as when interacting with the real prover. Specifically, if $\sigma = \tau = 0$, then $\mathcal{S}$ works exactly like the honest prover, and so the result is identical. However, even if $\sigma = \tau = 1$, we have that V^* receives $(A = g^{r-x}, B = h^{r-x})$ and afterwards r. Setting $r' = r - x$ we have that V^* receives $s = r' + x$ as it expects to receive when interacting with P. We conclude that

$$\left\{\mathcal{S}_1^{V^*((g,h,y_1,y_2),r,z;\cdot)}(g, h, y_1, y_2) \mid \sigma = \tau\right\} \equiv \left\{\langle P((g, h, y_1, y_2), x), V^*((g, h, y_1, y_2), z, r)\rangle\right\}$$

when $(g, h, y_1, y_2) \in DH$ and $\mathcal{S}_1$ is a simulator that works as $\mathcal{S}$ but runs only a single iteration. Since $\mathcal{S}$ continues until $\sigma = \tau$, and uses independent coin tosses each time, we have that its output is exactly a uniform sample from the distribution defined by

$$\left\{\mathcal{S}_1^{V^*((g,h,y_1,y_2),r,z;\cdot)}(g, h, y_1, y_2) \mid \sigma = \tau\right\}$$

and so is distributed exactly like $\{\langle P((g, h, y_1, y_2), x), V^*((g, h, y_1, y_2), z, r)\rangle\}$, as required.

It remains to show that $\mathcal{S}$ halts in (expected) polynomial-time. In order to see this, recall that the message (A, B) generated by $\mathcal{S}$ in the case that $\tau = 0$ is identically distributed to the first message generated in the case that $\tau = 1$. Thus, the message σ generated by V^* after receiving (A, B) is *independent* of the value τ. Thus, the probability that $\sigma = \tau$ is at most $1/2$. It follows that $\mathcal{S}$ expects to halt after two attempts, and so its expected running-time is polynomial. ■

Reducing the soundness error. As we have mentioned, the soundness error can be reduced by repeating the proof many times sequentially. However, we must prove that the zero-knowledge property is preserved in such a case. Fortunately, it has been shown that any *auxiliary-input* zero-knowledge protocol remains zero-knowledge under sequential composition. Without relying on this general theorem, it is easy to see that the simulation strategy above can be generalized in a straightforward way to the case of many sequential executions. Specifically, the above strategy for a single execution is carried out until it succeeds. Once this happens, the simulator fixes this part of the transcript and continues with the same strategy for the second execution. As before, the expected number of attempts is two, following which the simulator proceeds to the third execution and so on.

Discussion. We note that the simulation by S^* is somewhat counterintuitive. In particular, S^* works without ever knowing if (g, x, y, z) is really a Diffie-Hellman tuple, and would "succeed" in proving even if it is not. This is in stark contrast to the soundness requirement of the proof. This contradiction is reconciled by the fact that soundness must hold in the setting of a real interaction between a verifier and prover. In contrast, the simulator is given additional power due to the fact that it can *rewind* the verifier. This power is not given to a real prover, and so it could not follow a similar strategy in order to cheat.

On the power of perfect zero-knowledge proofs. It has been shown that every language that has a perfect zero-knowledge proof is contained in $\mathcal{AM} \cap$ co-$\mathcal{AM}$ (where $\mathcal{AM}$ is the class of all languages having two-round public-coin proof systems). We note that $\mathcal{AM}$ is conjectured to be not much larger than $\mathcal{NP}$. Furthermore, if an NP-complete language is contained in $\mathcal{AM} \cap$co-$\mathcal{AM}$, then the polynomial-hierarchy collapses. An interesting by-product of this result is that it can be used to declare that a given language (that is not known to be in $\mathcal{P}$) is unlikely to be $\mathcal{NP}$-complete. For example, the *graph isomorphism* language is not known to be in $\mathcal{P}$, and is also not known to be $\mathcal{NP}$-complete. However, there exists a perfect zero-knowledge proof for this language (see [5, Section 4.3.2]). Therefore, graph isomorphism is unlikely to be $\mathcal{NP}$-complete, because this would cause the polynomial hierarchy to collapse. This is a good example of where cryptography and complexity meet, and benefit from each other.

Lecture 7

Zero-Knowledge for all $\mathcal{NP}$

In this lecture, we present one of the most fundamental and amazing theorems in the theory of cryptography. The theorem states that any NP-language has a zero-knowledge proof, and was proved in [9]. The importance of this theorem is that it means that zero-knowledge proofs have a wide use, and are not just specific to some peculiar languages.

The theorem is also the first *positive* use of the notion of NP-completeness. That is, rather than using NP-completeness to show that something cannot be done, here it is used to accomplish something positive; namely, the existence of zero-knowledge proofs for all $\mathcal{NP}$. This is achieved by presenting a zero-knowledge proof for an NP-complete language (namely, 3-colouring). Then, any NP-language can be proven in zero-knowledge by first applying a Cook reduction to the input (obtaining an instance of 3-colouring), and then proving that the resulting instance is indeed 3-colourable. (There are some subtleties that must be addressed here, which will be discussed later.) We note that the proof that we will present is computational zero-knowledge (and not perfect or statistical). As we have mentioned, perfect or statistical zero-knowledge proofs do not exist for NP-complete languages, unless the polynomial-hierarchy collapses.

7.1 Commitment Schemes

The construction of zero-knowledge proofs for 3COL uses a (perfectly binding) commitment scheme. Commitment schemes are a basic ingredient in many cryptographic protocols. They are used to enable a party, known as the *sender*, to commit itself to a value while keeping it secret from the *receiver* (this property is called hiding). Furthermore, the commitment is binding, and thus in a later stage when the commitment is opened, it is guaranteed that the "opening" or "decommitment" can yield only a single value determined in the committing phase. One can think of a commitment scheme as a digital envelope. Placing a value in an envelope and sealing it binds the sender to the value. However, in addition, the receiver learns nothing about the value until the envelope is opened.

In a *perfectly binding* commitment scheme, the binding property holds even for an all-powerful sender, while the hiding property is only guaranteed with respect to a polynomial-time bounded receiver. Note that to some extent, the hiding and binding requirements contradict each other. That is, if a scheme is hiding, then no "information" about the committed value should be contained in the commitment value. However, in such a case, it should be possible to reveal any value in the decommitment stage. This contradiction is overcome by the use of computational assumptions, as we will see below.

[*] Lecture notes for a graduate course in the theory of cryptography. Yehuda Lindell, Bar-Ilan University, Israel, 2005.

For simplicity, we begin by presenting the definition for a non-interactive, perfectly-binding commitment scheme for a single bit. String commitment can be obtained by separately committing to each bit in the string. We denote by $C(\sigma; r)$ the output of the commitment scheme C upon input $\sigma \in \{0,1\}$ and using the random string $r \in_R \{0,1\}^n$ (for simplicity, we assume that C uses n random bits where n is the security parameter).

Definition 7.1 (non-interactive perfectly-binding bit commitment): *A* non-interactive perfectly binding commitment scheme *is a probabilistic polynomial-time algorithm C satisfying the following two conditions:*

1. Perfect Binding: $C(0;r) \neq C(1;s)$ *for every* $r, s \in \{0,1\}^n$ *and for every* $n \in \mathbb{N}$. (*Equivalently, it is required that* $\{C(0;r)\}_{r\in\{0,1\}^*} \cap \{C(1;r)\}_{r\in\{0,1\}^*} = \phi$.)
2. Computational Hiding: *The probability ensembles* $\{C(0;U_n)\}_{n\in\mathbb{N}}$ *and* $\{C(1;U_n)\}_{n\in\mathbb{N}}$ *are computationally indistinguishable to* non-uniform *polynomial-time distinguishers.*

A decommitment *to a commitment value c is a pair (b, r) such that $c = C(b; r)$.*

Constructing bit commitment. We now show how to construct non-interactive perfectly-binding commitment schemes.

Proposition 7.2 *Assuming the existence of 1–1 one-way functions, there exist non-interactive perfectly-binding commitment schemes.*

Proof: Let f be a 1–1 one-way function and let b be a hard-core predicate of f (such a predicate exists, as we have seen in Theorem 3.1). Then, define

$$C(\sigma; r) = (f(r), b(r) \oplus \sigma)$$

The binding property of C follows immediately from the 1–1 property of f. In particular, for every $r \neq s$, it holds that $f(r) \neq f(s)$ and so $C(0;r) \neq C(1;s)$. Furthermore, $f(r)$ fully defines $b(r)$, and so $C(0;r) = (f(r), b(r)) \neq (f(r), b(r) \oplus 1) = C(1;r)$. We conclude that for *every* $r, s \in \{0,1\}^n$ (both in the case that $r \neq s$ and in the case that $r = s$), it holds that $C(0;r) \neq C(1;s)$.

The hiding property follows immediately from the fact that b is a hard-core predicate of f. In particular, if it is possible to distinguish $\{f(r), b(r))$ from $\{f(r), \overline{b}(r)\}$, then it is possible to guess $b(r)$ given $f(r)$. In fact, in the proof of Theorem 4.7 we have already formally proven this fact. This completes the proof. ■

We note that allowing some minimal interaction (in which the receiver first sends a single message), it is possible to construct almost perfectly-binding commitment schemes from any one-way function [14].

String commitment. As we have mentioned, it is possible to construct secure commitment schemes by concatenating bit commitments. However, in order to prove this, we need a definition of security for *string commitment*. In the homework, you are asked to formulate the notion of perfect binding for string commitment. Here, we will present a definition of hiding through a "game" between the distinguisher and a *commitment oracle*. For a commitment scheme C, an adversary $\mathcal{A}$, a security parameter n and a bit $b \in \{0,1\}$, consider the following experiment:

The commitment experiment $\mathsf{ComExp}^b_{\mathcal{A},C}(n)$:

1. *Upon input 1^n, the adversary $\mathcal{A}$ outputs a pair of messages m_0, m_1 that are of the same length.*
2. *The commitment $c = C(m_b; r)$ is computed, where r is uniformly chosen, and is given to $\mathcal{A}$.*
3. *$\mathcal{A}$ outputs a bit b' and this is the output of the experiment.*

We now have the following definition:

Definition 7.3 *A string commitment scheme C is* computationally hiding *if for every probabilistic polynomial-time machine $\mathcal{A}$, every polynomial $p(\cdot)$ and all sufficiently large n's*

$$\left|\Pr\left[\mathsf{ComExp}^0_{\mathcal{A},C}(n) = 1\right] - \Pr\left[\mathsf{ComExp}^1_{\mathcal{A},C}(n) = 1\right]\right| < \frac{1}{p(n)}$$

If the above holds for every non-uniform polynomial-time machine $\mathcal{A}$, then the scheme is computationally hiding for non-uniform adversaries*.*

In the homework, you are asked to prove the following proposition:

Proposition 7.4 *Let C be a bit commitment scheme that fulfills Definition 7.1. Then, the string commitment scheme C' that is defined by $C'(x) = C(x_1), \ldots, C(x_n)$ where $x = x_1, \ldots, x_n$ fulfills Definition 7.3.*

A further extension to Definition 7.3 is to consider the case that $\mathcal{A}$ outputs a pair of vectors of commitments $\overline{m}_0$ and $\overline{m}_1$ where each vector contains the same number of elements and for every i, the length of the message m_0^i equals the length of the message m_1^i. This definition is equivalent to Definition 7.3 as can be shown via a standard hybrid argument.

Applications. Commitment schemes are used in many places in cryptography. One famous use is in secure coin-tossing [2]. Specifically, in order to toss a coin, the first party commits to a random bit σ_1 and sends the commitment value c to the second party. The second party then sends a random value σ_2 to the first party. Finally, the first party decommits, revealing σ_1, and the result is $\sigma_1 \oplus \sigma_2$. The idea behind this protocol is that the second party cannot make σ_2 depend on σ_1 because of the hiding property of the commitment scheme. Likewise, the first party cannot make its revealed value depend on σ_2 because it chose σ_1 first, and is bound to this value by the commitment scheme. Thus, the result of the protocol is essentially the XOR of two independently chosen coins, yielding the desired result.

7.2 Zero-Knowledge for the Language 3COL

Let $G = (V, E)$ be a graph. We say that $G \in 3COL$ (or G is 3-colourable) if there exists a function $\phi : V \to \{1, 2, 3\}$ such that for every $(u, v) \in E$ it holds that $\phi(u) \neq \phi(v)$. The function ϕ is called a colouring of G. It is well known that $3COL$ is NP-complete.

The idea behind the zero-knowledge proof for $3COL$ is as follows. The prover commits to a random 3-colouring of the graph G; more specifically, it commits to $\psi(v)$ for every $v \in V$, where ψ is a random permutation of the colours in ϕ. Next, the verifier asks to see the colours of the endpoints of a randomly chosen edge $(u, v) \in E$. Finally, the prover opens the commitments of $\psi(u)$ and $\psi(v)$ and the verifier accepts if and only if $\psi(u) \neq \psi(v)$.

In order to see that this is an interactive proof, notice that if the graph is not 3-colourable, then for any commitment to a function ψ, there must be at least one edge $(u, v) \in E$ for which $\psi(u) = \psi(v)$. It follows that the verifier will detect a cheating prover with probability at least $1/|E|$. By repeating the proof many times with different random colourings each time (say, $n \cdot |E|$ times), the soundness error can be made negligible. Regarding zero-knowledge, notice that in each execution, the only thing that the verifier sees is a pair of numbers that are different. Since the colouring is random and different in each execution, we have that the verifier learns nothing (in particular, the verifier cannot create a picture of how the colouring looks in the graph).

Protocol 7.5 (computational zero-knowledge for $3COL$):

- **Common input:** *a graph $G = (V, E) \in 3COL$ where $|V| = n$. Denote $V = \{v_1, \ldots, v_n\}$.*
- **Prover's auxiliary input:** *a colouring ϕ of G.*
- **The protocol:**
  1. *The prover P chooses a random permutation π over the set $\{1, 2, 3\}$ and defines a colouring $\psi = \pi \circ \phi$ of G (i.e., $\psi(v) = \pi(\phi(v))$).*

     For every $i = 1, \ldots, n$, the prover P computes $c_i = C(\psi(v_i); U_n)$. P then sends the vector $(c_1, \ldots, c_n)$ to the verifier V. (Formally, each commitment value is two bits long and so by our above construction of commitment schemes, $2n$ random bits are actually needed. For simplicity, however, we will assume that only n bits are used.)
  2. *The verifier V chooses a random edge $e = (v_i, v_j) \in_R E$ and sends e to P.*
  3. *Let $e = (v_i, v_j)$ be the edge received by P.*[1] *Then, it opens c_i and c_j, revealing $\psi(v_i)$ and $\psi(v_j)$ to the verifier V.*
  4. *The verifier V accepts if and only if it received valid decommitments to c_i and c_j, and it holds that $\psi(v_i), \psi(v_j) \in \{1, 2, 3\}$ and $\psi(v_i) \neq \psi(v_j)$.*

Proposition 7.6 *Assume that C used in Protocol 7.5 is a perfectly-binding commitment scheme. Then, Protocol 7.5 is a computational zero-knowledge proof system for the language 3COL, with an efficient prover, perfect completeness and soundness error $1 - 1/|E|$.*

Proof Sketch: We first prove that Protocol 7.5 is an efficient-prover interactive proof. The efficient-prover and perfect completeness properties are immediate. Regarding soundness, notice that if $G \notin 3COL$, then for every series of commitments $(c_1, \ldots, c_n)$ there exists at least one $e = (v_i, v_j) \in E$ such that c_i and c_j are either commitments to values that are not in $\{1, 2, 3\}$ or they are commitments to the same value. Otherwise, C_G defines a valid 3-colouring of G, in contradiction to the assumption that $G \notin 3COL$.

We now sketch the proof that Protocol 7.5 is zero-knowledge (the proof is rather complete, but some details are left out). We construct a probabilistic polynomial-time $\mathcal{S}$ who receives input $G \in 3COL$, oracle access to a possibly adversarial probabilistic polynomial-time V^*, and works as follows:

[1] If the reply of the verify is not a valid edge, then P interprets it to be a pre-specified default edge.

1. $\mathcal{S}$ attempts the following at most $2n|E|$ times (using independent random coin tosses each time):
   (a) $\mathcal{S}$ chooses a random edge $e' = (v_i, v_j) \in_R E$. Simulator $\mathcal{S}$ then chooses $\psi(v_i) \in_R \{1,2,3\}$ and $\psi(v_j) \in_R \{1,2,3\} \setminus \{\psi(v_i)\}$, and defines $\psi(v_k) = 1$ for every $k \neq i, j$.
   For every $i = 1, \ldots, n$, simulator $\mathcal{S}$ computes $c_i = C(\psi(v_i); U_n)$ and hands the verifier V^* the commitments $(c_1, \ldots, c_n)$.
   (b) Let e be the edge that V^* sends as a reply.
      i. If $e = e'$, then $\mathcal{S}$ decommits to $\psi(v_i)$ and $\psi(v_j)$ and outputs whatever V^* outputs.
      ii. If $e \neq e'$, then $\mathcal{S}$ returns back to Step 1 for another attempt (using independent and random coin tosses).
2. If all $2n|E|$ attempts fail, then $\mathcal{S}$ outputs fail and halts.

It is clear that $\mathcal{S}$ runs in (strict) polynomial time: there are $2n|E|$ iterations in the simulation and each iteration involves a polynomial amount of work.

We begin by showing that $\mathcal{S}$ outputs fail with at most negligible probability. We prove this by demonstrating that $\mathcal{S}$ succeeds in each iteration with "good enough" probability. This argument is similar to the case of Diffie-Hellman tuples above, but is computational (rather than information-theoretic). Specifically, if the probability that V^* replies with $e = e'$ is lower than $1/2|E|$, then this fact can be used to contradict the hiding property of the commitment scheme.[2] Formally, assume that for infinitely many graphs G, it holds that V^* upon input G outputs $e = e'$ with probability less than $1/2|E|$ when interacting with the simulator. We show how this contradicts the hiding property of the commitment scheme. We begin by constructing a modified simulator S' who chooses a random edge e like $\mathcal{S}$. However, S' sets $\psi(v) = 1$ for *all* $v \in V$ (including the endpoints of e). When V^* interacts with S', we have that in V^*'s view the chosen edge e is uniformly distributed in E (even given the commitments). Therefore, V^* replies with $e = e'$ with probability exactly $1/|E|$. It remains to show that if V^* replies with $e = e'$ with probability that is less than $1/2|E|$ when interacting with $\mathcal{S}$, then this can be used to distinguish commitments. This can be demonstrated using Definition 7.3 of hiding (with the extension to vectors of messages). Informally speaking, a distinguisher D for the commitment scheme works in the same way as $\mathcal{S}$ and S', except that it generates the vector $\overline{m}_0$ to contain two 1's and the vector $\overline{m}_1$ to contain the colours $\psi(v_i)$ and $\psi(v_j)$. Then, D hands V^* the commitments it received from its oracle along with $n-2$ commitments to 1 that it generates itself (the commitments are placed in the appropriate order). Finally, D outputs 1 if and only if V^* replies with $e = e'$. Now, if D received back a commitment to two ones, then the distribution is exactly that generated by S' and so $e = e'$ with probability exactly $1/|E|$. That is,

$$\Pr[\mathsf{ComExp}^0_{D,C}(n) = 1] = \frac{1}{|E|}$$

In contrast, if D received back a commitment to the colours $\psi(v_i)$ and $\psi(v_j)$, then the view of V^* is exactly as in an interaction with $\mathcal{S}$. Thus, if V^* sends $e = e'$ with probability less than $1/2|E|$ it follows that

$$\Pr[\mathsf{ComExp}^1_{D,C}(n) = 1] < \frac{1}{2|E|}$$

Combining the above we have that for infinitely many G's (and thus infinitely many n's, the distinguisher D distinguishes between the commitments, in contradiction to the hiding property

[2] In fact it can be shown that e must equal e' with probability that is at most negligibly far from $1/|E|$ but it is easier to take the concrete $1/2|E|$.

of the commitment scheme. Thus, $e = e'$ with probability at least $1/2|E|$. This implies that the probability that $\mathcal{S}$ fails in all of these attempts is at most $(1-1/2|E|)^{2n|E|} < e^{-n}$ and so is negligible. That is, $\mathcal{S}$ outputs fail with negligible probability.

Next, we prove that $\mathcal{S}$'s output is *computationally indistinguishable* from the output of V^* in a real execution with the honest prover P. In order to see this, we first consider a modified simulator $\tilde{\mathcal{S}}$ who receives a real 3-colouring ϕ of the input graph. Then, $\tilde{\mathcal{S}}$ works in exactly the same way as $\mathcal{S}$ except that it commits to a random permutation of the valid 3-colouring. (Of course, $\tilde{\mathcal{S}}$ is not a valid simulator, but this is just a mental experiment.) It is clear that the output distribution of $\tilde{\mathcal{S}}$, *given that it doesn't output* fail is identical to that of a real transcript between the honest prover and V^* (the rewinding until $e = e'$ makes no difference). That is, we have:

$$\{\langle P, V^*\rangle(G)\}_{G\in 3COL} \equiv \left\{\tilde{\mathcal{S}}(G) \mid \neg\mathsf{fail}\right\}_{G\in 3COL}$$

However, since $\mathcal{S}$ and $\tilde{\mathcal{S}}$ output fail with at most negligible probability (where the latter is shown in the same way as for $\mathcal{S}$), we have that there can be at most a negligible difference between the two distributions. Thus they are computationally indistinguishable (in fact, even statistically close). That is:

$$\{\langle P, V^*\rangle(G)\}_{G\in 3COL} \stackrel{\rm c}{\equiv} \left\{\tilde{\mathcal{S}}(G)\right\}_{G\in 3COL}$$

We proceed to show that the output distribution of $\tilde{\mathcal{S}}$ is computationally indistinguishable from the output distribution of $\mathcal{S}$. We use Definition 7.3 in order to prove this, or actually, its extension to the case that the adversary outputs a pair of *vectors* of messages. Assume by contradiction that there exists a (non-uniform) polynomial-time distinguisher D and a polynomial p such that for infinitely many graphs $G \in 3COL$ (where n denotes the number of nodes in each such graph) it holds that

$$\left|\Pr[D(\tilde{\mathcal{S}}(G)) = 1] - \Pr[D(\mathcal{S}(G)) = 1]\right| \geq \frac{1}{p(n)}$$

We now use D to construct a *non-uniform* probabilistic polynomial-time distinguisher D' for the commitment scheme C. Distinguisher D' is given a graph $G \in 3COL$ *along with its colouring* as auxiliary input and works as follows:

1. D' fixes the random-tape of V^* to a uniformly distributed string R.

2. D' prepares $2n|E|$ vectors of commitments as follows. For each vector it chooses an independent random edge $e' = (v_i, v_j) \in_R E$. It then constructs vectors of messages as follows. The first vector consists of $n-2$ messages of value 1. In contrast, the second vector is constructed by first choosing a random colouring ψ of G (like the honest prover) and setting the messages to be the colours of all nodes except for v_i and v_j; the colours are given in the order of the nodes from v_1 to v_n excluding v_i, v_j. Note that there are $n-2$ values in this vector as well. The vectors $\overline{m}_0, \overline{m}_1$ are constructed by concatenating all of the above. That is, $\overline{m}_0$ consists of all of the vectors of the first type, and $\overline{m}_1$ consists of all of the vectors of the second type.

   D' hands $\overline{m}_0, \overline{m}_1$ to its commitment oracle and receives back a vector of commitments $\overline{c}$. Denote the $m \stackrel{\rm def}{=} 2n|E|$ vectors of commitments inside $\overline{c}$ by $\overline{c}^1, \ldots, \overline{c}^m$, and denote the commitments in the vector $\overline{c}^\ell$ by c_k^ℓ for $k = 1, \ldots, n$, $k \neq i, j$. Our intention here is that c_k^ℓ is the "message" associated with node v_k inside $\overline{c}^\ell$.

3. For $\ell = 1, \ldots, 2n|E|$, distinguisher D' works as follows:

(a) Given $\overline{c}^\ell$ with random edge $e'_\ell = (v_i, v_j)$ that was chosen for this ℓ^{th} part, distinguisher D' computes commitments $c_i^\ell = \psi(v_i)$ and $c_j^\ell = \psi(v_j)$. D' then hands the commitments $c_1^\ell, \ldots, c_n^\ell$ to V^*.

(b) Let e be the edge that V^* sends as a reply.

i. If $e = e'_\ell$, then D' decommits to c_i^ℓ and c_j^ℓ to V^* (D' can do this because it generated the commitments c_i and c_j itself). Then, D' receives the output generated by V^* and outputs whatever D outputs on this.

ii. If $e \neq e'^\ell$, then D' returns back to the beginning of the loop.

4. If D' "fails" on all attempts, then it invokes D on the string fail and outputs whatever D outputs.

It is not difficult to ascertain that when D' is interacting in experiment $\mathsf{ComExp}^0_{\mathcal{A},C}(n)$ and so it receives commitments to $\overline{m}_0$, the distribution generated by D' is exactly the same as $\mathcal{S}$. Thus,

$$\Pr[\mathsf{ComExp}^0_{D'(1^n,G,\phi),C}(n) = 1] = \Pr[D(\mathcal{S}(G)) = 1]$$

On the other hand, when D' interacts in experiment $\mathsf{ComExp}^1_{\mathcal{A},C}(n)$ and so it receives commitments to $\overline{m}_1$ that constitute a correct colouring, the distribution generated by D' is exactly the same as $\tilde{\mathcal{S}}$. That is,

$$\Pr[\mathsf{ComExp}^1_{D'(1^n,G,\phi),C}(n) = 1] = \Pr[D(\tilde{\mathcal{S}}(G)) = 1]$$

Combining the above, we have that for infinitely many $G \in 3COL$ it holds that

$$\left|\Pr[\mathsf{ComExp}^0_{D'(1^n,G,\phi),C}(n) = 1] - \Pr[\mathsf{ComExp}^1_{D'(1^n,G,\phi),C}(n) = 1]\right| \geq \frac{1}{p(n)}$$

in contradiction to the extended hiding property of the commitment scheme. We conclude that the output distribution of $\mathcal{S}$ is computationally indistinguishable from the output of V^* in a real execution of the protocol. This completes the proof. ■

We remark that by repeating the proof many times sequentially, the soundness error can be made negligible.

7.3 Zero-Knowledge for every Language $L \in \mathcal{NP}$

A protocol for any NP-language L is obtained as follows. Let $x \in L$. Then, both the prover and verifier first compute a Cook reduction of x to G, via the reduction to 3-colouring. Next, they run the $3COL$ protocol described above. (In order for the prover to be efficient, the witness to $x \in L$ must also be "translated" into a witness for $G \in 3COL$.)

We note that there is a subtlety here that needs to be addressed. Specifically, the verifier now has *additional* information about G that is not provided to the verifier in the setting of $3COL$. This additional information is a value $x \in L$ such that the Cook reduction of L to $3COL$ transforms the value x to the input graph G. Now, if the protocol used is zero-knowledge also for the case of *auxiliary inputs* (as indeed Protocol 7.5 is), then the protocol for L is also zero-knowledge. This can be seen by just defining the auxiliary input of V^* to be such an x. An alternative approach to solving this problem is to notice that 3-colouring has a *Levin reduction*, meaning that given G it is possible to efficiently go "back" and find x. In such a case, x can be efficiently computed from G and so this information can be obtained by V^* itself. Thus, the original verifier actually does not have any additional information beyond the input G itself.

Recalling that commitment schemes can be constructed from any one-way function [14], we conclude with the following theorem:

Theorem 7.7 *Assume the existence of one-way functions. Then, every language $L \in \mathcal{NP}$ has a computational zero-knowledge proof system with an efficient prover, perfect completeness, and negligible soundness error.*

7.4 More on Zero-Knowledge

In the material presented in class, we have barely touched the tip of the iceberg with respect to zero-knowledge. Two topics of utmost importance that we did not relate to at all are **zero-knowledge proofs of knowledge** and **non-interactive zero-knowledge proofs**. We refer students to [5, Section 4.7] and [5, Section 4.10], respectively, for material. It is also recommended to read [5, Sections 4.5,4.6,4.8] regarding negative results on zero-knowledge, witness indistinguishable proofs, and zero-knowledge arguments.

Lecture 8

Proofs of Knowledge and Non-Interactive Zero Knowledge

* Lecture notes for a graduate course in the theory of cryptography. Yehuda Lindell, Bar-Ilan University, Israel, 2005.

Lecture 9

Encryption Schemes I

In this lecture, we will consider the problem of secure encryption. This lecture will be mainly definitional in nature, and will present the definitions for private-key and public-key encryption in parallel. We will consider two definitions, *semantic security* and *indistinguishability,* and will prove their equivalence. Finally, we will prove that any *public-key* encryption scheme that is secure for a single message is also secure under multiple encryptions. An analogous claim does not hold for private-key encryption schemes. We note that since the course "Introduction to Cryptography" (89-656) is a prerequisite to this one, we assume that all students are familiar with the basic notions and settings of private-key and public-key encryption.

9.1 Definitions of Security

Before defining the notion of *security* for encryption schemes, we present the syntax of what constitutes an encryption scheme to begin with.

Definition 9.1 *An* encryption scheme *consists of a triple of probabilistic polynomial-time algorithms (G, E, D) satisfying the following conditions:*

1. *On input 1^n, the* key-generator *algorithm G outputs a pair of keys (e, d).*
2. *For every pair (e, d) in the range of $G(1^n)$ and for every $\alpha \in \{0,1\}^*$, the encryption and decryption algorithms E and D satisfy*

   $$\Pr[D(d, E(e, \alpha)) = \alpha] = 1$$

   where the probability is taken over the internal coin tosses of algorithms E and D.

The integer n serves as the security parameter *of the scheme. The key e is called the* encryption key *and the key d is called the* decryption key*. The string α is the* plaintext *and $E(e, \alpha)$ is the* ciphertext*. For shorthand, we will denote $E_e(\alpha) = E(e, \alpha)$ and $D_d(\beta) = D(d, \beta)$.*

We note that the above definition does not differentiate between public and private key encryption schemes; this difference will come into the definition of security. We also note that the definition can be relaxed to allow for a negligible failure error in decryption, and/or that correct decryption only needs to hold for all but a negligible fraction of key pairs.

[*] Lecture notes for a graduate course in the theory of cryptography. Yehuda Lindell, Bar-Ilan University, Israel, 2005.

9.1.1 Semantic Security

Intuitively, an encryption scheme is secure if a ciphertext reveals no information about the encrypted plaintext. An immediate difficulty arises when trying to formulate this notion. Specifically, it is possible that a priori information is known about the plaintext (e.g., it is English text or it is a work contract). Therefore, it is not possible to require that the adversary know nothing about the plaintext given the ciphertext (because it already knows something). Rather, it must be required that the adversary learn nothing *more* about the plaintext from the ciphertext, than what is already a priori known. This is formalized using the simulation paradigm. That is, an encryption scheme is said to be secure (under semantic security) if everything the adversary can learn about the plaintext given the ciphertext, it could learn about the plaintext using its a priori knowledge alone. More formally, it is required that for every adversary who receives the ciphertext and outputs some information about the plaintext, there exists another adversary who does *not* receive the ciphertext but succeeds in outputting essentially the same information about the plaintext. This second adversary is a "simulator" and its existence demonstrates that the information that was output by the initial adversary was not learned from the ciphertext.[1] One caveat to the above is that the length of the plaintext can always be learned by the adversary. More specifically, an n-bit random plaintext cannot be encrypted into an $n/2$-bit ciphertext (stated otherwise, a giant cannot dress up as a dwarf). However, this is the only information that the adversary is allowed to learn. (We note that it is possible to pad the plaintext to some pre-determined length if the exact length is something that we wish to hide.)

The actual definition. In the definition below, the function f represents the information about the plaintext that the adversary attempts to learn. In contrast the function h (or "history" function) represents the adversary's a priori knowledge regarding the plaintext. The adversary's inability to learn information about the plaintext should hold for any distribution of plaintexts, and such a distribution is represented by the probability ensemble $\{X_n\}_{n\in\mathbb{N}}$. As we will see, the definition will quantify over all possible functions f and h, and all possible probability ensembles $\{X_n\}$.

The security of an encryption scheme is only required to hold for polynomial-length plaintexts; this is formalized by requiring that $|X_n| \leq \text{poly}(n)$. (Recall that this means that there exists a polynomial p such that for all sufficiently large n's, $|X_n| \leq p(n)$.) Likewise, we restrict f and h to be such that for every z, $|f(z)|, |h(z)| \leq \text{poly}(|z|)$. Such distributions and functions are called polynomially bounded. Denote by $G_1(1^n)$ the first key output by G (i.e., the encryption key), and by $G_2(1^n)$ the second key output by G (i.e., the decryption key). We now present the definition.

Definition 9.2 (semantic security – private-key model): *An encryption scheme* (G, E, D) *is* semantically secure in the private-key model *if for every probabilistic polynomial-time algorithm* $\mathcal{A}$ *there exists a probabilistic polynomial-time algorithm* $\mathcal{A}'$ *such that for every polynomially-bounded probabilistic ensemble* $\{X_n\}_{n\in\mathbb{N}}$*, every pair of polynomially-bounded functions* $f, h : \{0,1\}^* \to \{0,1\}^*$*, every positive polynomial* $p(\cdot)$ *and all sufficient large* n*'s*

$$\begin{aligned}\Pr[\mathcal{A}(1^n, E_{G_1(1^n)}(X_n), 1^{|X_n|}, h(1^n, X_n)) = f(1^n, X_n)]&\\ < \Pr[\mathcal{A}'(1^n, 1^{|X_n|}, h(1^n, X_n)) = f(1^n, X_n)] + \frac{1}{p(n)}&\end{aligned}$$

where the probabilities are taken over X_n *and the internal coin tosses of* G*,* E*,* $\mathcal{A}$ *and* $\mathcal{A}'$*.*

[1] The fact that this second adversary is a simulator will become more clear later. Specifically, this second adversary is typically constructed by invoking the first adversary on a "dummy" ciphertext that encrypts nothing. Thus, the second adversary provides the first adversary with a "simulated ciphertext".

Notice that the algorithm $\mathcal{A}$ (representing the real adversary) is given the ciphertext $E_{G(1^n)}(X_n)$ as well as the history function $h(1^n, X_n)$, where this latter function represents whatever a priori knowledge of the plaintext X_n the adversary may have. The adversary $\mathcal{A}$ then attempts to guess the value of $f(1^n, X_n)$. Now, the algorithm $\mathcal{A}'$ also attempts to guess the value of $f(1^n, X_n)$. However, in contrast to $\mathcal{A}$, the adversary $\mathcal{A}'$ is given only the history function $h(1^n, X_n)$ and *not* the ciphertext. The security requirement states that $\mathcal{A}$'s success in guessing $f(1^n, X_n)$ should not exceed $\mathcal{A}'$'s success in guessing $f(1^n, X_n)$ by a non-negligible factor. Stated otherwise, $\mathcal{A}'$ can guess the value of $f(1^n, X_n)$ with almost the same success as $\mathcal{A}'$, without ever receiving the ciphertext. Thus, the ciphertext $E_{G(1^n)}(X_n)$ does not reveal anything (non-negligible) about $f(1^n, X_n)$.

Security for public-key encryption. The definition for public-key encryption is almost identical; we include it only for the sake of completeness. Recall that $G_1(1^n)$ denotes the first key output by G (i.e., the encryption key). Then, the only difference in the definition here is that the adversary is also given the public encryption-key $G_1(1^n)$. That is:

Definition 9.3 (semantic security – public-key model): *An encryption scheme* (G, E, D) *is* semantically secure in the public-key model *if for every probabilistic polynomial-time algorithm* $\mathcal{A}$ *there exists a probabilistic polynomial-time algorithm* $\mathcal{A}'$ *such that for every polynomially-bounded probabilistic ensemble* $\{X_n\}_{n\in\mathbb{N}}$*, every pair of polynomially-bounded functions* $f, h : \{0,1\}^* \to \{0,1\}^*$*, every positive polynomial* $p(\cdot)$ *and all sufficient large n's*

$$\Pr[\mathcal{A}(1^n, G_1(1^n), E_{G_1(1^n)}(X_n), 1^{|X_n|}, h(1^n, X_n)) = f(1^n, X_n)] \\ < \Pr[\mathcal{A}'(1^n, 1^{|X_n|}, h(1^n, X_n)) = f(1^n, X_n)] + \frac{1}{p(n)}$$

where the probabilities are taken over X_n *and the internal coin tosses of* G*,* E*,* $\mathcal{A}$ *and* $\mathcal{A}'$*.*

Note that there is no point giving $\mathcal{A}'$ the public-key $G_1(1^n)$, since $\mathcal{A}'$ can generate it by itself.

9.1.2 Indistinguishability

The definition of semantic security is an intuitive one. In contrast, the definition of indistinguishability that we will present now is less intuitive. Its advantage over semantic security is that it is much easier to work with. Fortunately, as we will prove, the definitions turn out to be equivalent. We can therefore utilize the easier-to-handle definition of indistinguishability, while retaining the intuitive appeal of semantic security. We note that the definition here is explicitly non-uniform.

Definition 9.4 (indistinguishability of encryptions – private-key model): *An encryption scheme* (G, E, D) *has* indistinguishable encryptions in the private-key model *if for every polynomial-size circuit family* $\{C_n\}_{n\in\mathbb{N}}$*, every pair of positive polynomials* $\ell(\cdot)$ *and* $p(\cdot)$*, all sufficient large n's, and every* $x, y \in \{0,1\}^{\ell(n)}$

$$\left|\Pr[C_n(E_{G_1(1^n)}(x)) = 1] - \Pr[C_n(E_{G_1(1^n)}(y)) = 1]\right| < \frac{1}{p(n)}$$

where the probabilities are taken over the internal coin tosses of algorithms G *and* E*.*

The definition for the public-key model is almost the same, except that C_n also receives $G_1(1^n)$.

9.1.3 Equivalence of the Definitions

In this section we prove the perhaps surprising theorem that the definitions of semantic security and indistinguishability are *equivalent*. We note that this result holds for both the public-key and private-key models. We present a proof sketch for the private-key case only, and leave the extensions to the public-key case as an exercise.

Theorem 9.5 (equivalence of definitions – private-key): *A private-key encryption scheme is semantically secure if and only if it has indistinguishable encryptions.*

Proof Sketch: We first show that any encryption scheme that has indistinguishable encryptions is semantically secure. The idea behind this proof is that by indistinguishability, an adversary $\mathcal{A}$ cannot distinguish between $E(X_n)$ and $E(1^{|X_n|})$ with non-negligible probability. Therefore, $\mathcal{A}$ will output $f(1^n, X_n)$ upon receiving $E(1^{|X_n|})$ with almost the same probability as it outputs $f(1^n, X_n)$ upon receiving $E(X_n)$. This fact is then used to construct $\mathcal{A}'$ who does not receive $E(X_n)$ but is still able to output $f(1^n, X_n)$ with almost the same probability as $\mathcal{A}$ who does receive $E(X_n)$.

Let $\mathcal{A}$ be a PPT algorithm that tries to learn some information (i.e., $f(1^n, X_n)$) about a plaintext. Recall that $\mathcal{A}$ receives $(1^n, E_{G_1(1^n)}(X_n), 1^{|X_n|}, h(1^n, X_n))$ and attempts to output $f(1^n, X_n)$, for some functions h and f. We construct $\mathcal{A}'$ who receives $(1^n, 1^{|X_n|}, h(1^n, X_n))$ and works as follows:

1. $\mathcal{A}'$ invokes the key generator G upon input 1^n and receives back an encryption key e.
2. $\mathcal{A}'$ computes $\beta = E_e(1^{|X_n|})$.
3. $\mathcal{A}'$ invokes $\mathcal{A}$ upon input $(1^n, \beta, 1^{|X_n|}, h(1^n, X_n))$ and outputs whatever $\mathcal{A}$ outputs.

Notice that $\mathcal{A}'$ does not depend on f or h and is therefore PPT as long as $\mathcal{A}$ is PPT. We claim that $\mathcal{A}'$ fulfills the requirement of Definition 9.2. In order to simplify notation, we will drop the 1^n and $1^{|X_n|}$ inputs of $\mathcal{A}$ and $\mathcal{A}'$ (one can assume that they are part of the output of h). Assume, by contradiction, that for some polynomial p and for infinitely many n's we have that

$$\left|\Pr[\mathcal{A}(E_{G_1(1^n)}(X_n), h(1^n, X_n)) = f(1^n, X_n)] - \Pr[\mathcal{A}'(h(1^n, X_n)) = f(1^n, X_n)]\right| \geq \frac{1}{p(n)} \tag{9.1}$$

Let

$$\Delta_n(X_n) = \left|\Pr[\mathcal{A}(E_{G_1(1^n)}(X_n), h(1^n, X_n)) = f(1^n, X_n)] - \Pr[\mathcal{A}(E_{G_1(1^n)}(1^{|X_n|}), h(1^n, X_n)) = f(1^n, X_n)]\right|$$

Then, by the construction of $\mathcal{A}$, the contradicting assumption in Eq. (9.1) is equivalent to assuming that $\Delta_n(X_n) \geq 1/p(n)$.

Fix n. Then, let $x_n \in \{0,1\}^{\text{poly}(n)}$ be the string in the support of X_n that maximizes the value of $\Delta_n(X_n)$. Next, we construct a circuit C_n that has the values x_n, $f(1^n, x_n)$ and $h(1^n, x_n)$ hardwired into it. The circuit C_n will be used to distinguish an encryption of x_n from an encryption of $1^{|x_n|}$. Upon receiving input ciphertext β, circuit C_n invokes $\mathcal{A}(\beta, h(1^n, x_n))$ and outputs 1 if $\mathcal{A}$ outputs $f(1^n, x_n)$, and 0 otherwise.

Notice that C_n is polynomial size because it only runs $\mathcal{A}$ and compares its output to a hardwired value. (Notice also that the circuit is strictly non-uniform because it uses the hardwired values that cannot necessarily be obtained efficiently.) By the construction of C_n, we have that

$$\Pr[C_n(\beta) = 1] = \Pr[\mathcal{A}(\beta, h(1^n, x_n)) = f(1^n, x_n)]$$

Therefore, by our contradicting assumption, we have that for infinitely many n's

$$\left|\Pr[C_n(E_{G_1(1^n)}(x_n)) = 1] - \Pr[C_n(E_{G_1(1^n)}(1^{|x_n|})) = 1]\right| \geq \frac{1}{p(n)}$$

in contradiction to the assumption that E has indistinguishable encryptions.

The other direction. We now show that any encryption scheme that is semantically secure has indistinguishable encryptions. We sketch this direction more briefly. Intuitively, indistinguishability can be viewed as a special case of semantic security. That is, let f be a function that distinguishes encryptions; that is, define $f(1^n, z) = 1$ if and only if $z = x_n$. Then, the ability to predict f can be directly translated into the ability to distinguish an encryption of x_n from an encryption of something else. Thus, semantic security implies indistinguishability. We note that the actual proof must also take into account the fact that indistinguishability is cast in the non-uniform model, whereas semantic security considers uniform machines. This issue is handled by including the non-uniformity into the history function h.

More specifically, assume that there exists a circuit family $\{C_n\}$ such that for some polynomial p and for infinitely many pairs (x_n, y_n)

$$\left|\Pr[C_n(E_{G_1(1^n)}(x_n)) = 1] - \Pr[C_n(E_{G_1(1^n)}(y_n)) = 1]\right| \geq \frac{1}{p(n)}$$

Then, we define a distribution X_n that chooses x_n with probability 1/2 and y_n with probability 1/2. Furthermore, we define a function f such that $f(1^n, x_n) = 1$ and $f(1^n, y_n) = 0$. Finally, we define h such that $h(1^n, X_n) = C_n$. Notice now that a machine $\mathcal{A}$ who receives a ciphertext $\beta = E(X_n)$ and $C_n = h(1^n, X_n)$ is able to compute $C_n(\beta)$ and obtain a guess for the value of $f(1^n, x_n)$. We refer to [6, Section 5.2.3] for more details. ■

We note that the first direction in the proof (indistinguishability implies semantic security) suffices for using indistinguishability as a definition that is more comfortable to work with. The second direction (semantic security implies indistinguishability) is important in that it show that we do not "lose" secure encryption schemes by insisting on a proof of security via indistinguishability. (If equivalence did not hold, then it could be that there exist encryption schemes that are semantically secure, but do not have indistinguishable encryptions. Such schemes would be "lost".)

Notice that the above proof is *inherently non-uniform*. Nevertheless, uniform analogues exist; see [6, Section 5.2.5].

Equivalence in the public-key setting. For completeness, we state the theorem also for the public-key case:

Theorem 9.6 (equivalence of definitions – public-key): *A public-key encryption scheme is semantically secure if and only if it has indistinguishable encryptions.*

9.2 Security Under Multiple Encryptions

Notice that the basic definition of encryption considers only a single encrypted message. In many cases this is unsatisfactory, and we need security to hold even if many encrypted messages are sent. We begin by defining the notion of indistinguishability under multiple encryptions (the equivalence of semantic security and indistinguishability holds also in this setting). We will present the definition for the private-key setting, and note that the extension to the public-key setting is straightforward.

Definition 9.7 (indistinguishability under multiple encryptions – private key): *An encryption scheme (G, E, D) has* indistinguishable encryptions for multiple messages in the private-key model *if for every polynomial-size circuit family, all positive polynomials $\ell(\cdot)$, $t(\cdot)$ and $p(\cdot)$, all sufficiently large n's, and every $x_1, \ldots, x_{t(n)}, y_1, \ldots, y_{t(n)} \in \{0,1\}^{\ell(n)}$, it holds that*

$$\left|\Pr[C_n(E_{G_1(1^n)}(x_1), \ldots, E_{G_1(1^n)}(x_{t(n)})) = 1] - \Pr[C_n(E_{G_1(1^n)}(y_1), \ldots, E_{G_1(1^n)}(y_{t(n)})) = 1]\right| < \frac{1}{p(n)}$$

We now show that in the public-key setting, indistinguishability for a single message implies indistinguishability for multiple messages. Unfortunately, the same is not true for the private-key setting.

9.2.1 Multiple Encryptions in the Public-Key Setting

Theorem 9.8 *A public-key encryption scheme has indistinguishable encryptions for multiple messages if and only if it has indistinguishable encryptions for a single message.*

Proof: The implication from multiple messages to a single message is trivial. The other direction is proven using a hybrid argument. That is, assume that (G, E, D) has indistinguishable encryptions for a single message. Furthermore, assume by contradiction, that there exists a circuit family $\{C_n\}$ and polynomials ℓ, t and p such that for infinitely many n's there exist $x_1, \ldots, x_{t(n)}, y_1, \ldots, y_{t(n)}$ such that

$$\Big|\Pr[C_n(G_1(1^n), E_{G_1(1^n)}(x_1), \ldots, E_{G_1(1^n)}(x_{t(n)})) = 1] \\ - \Pr[C_n(G_1(1^n), E_{G_1(1^n)}(y_1), \ldots, E_{G_1(1^n)}(y_{t(n)})) = 1]\Big| \geq \frac{1}{p(n)}$$

We define a hybrid sequence $h^i = (x_1, \ldots, x_i, y_{i+1}, \ldots, y_{t(n)})$, and a hybrid distribution $H_n^i = (G_1(1^n), E_{G_1(1^n)}(x_1), \ldots, E_{G_1(1^n)}(x_i), E_{G_1(1^n)}(y_{i+1}), \ldots, E_{G_1(1^n)}(y_{t(n)}))$. Then, by the contradicting assumption we have that

$$\left|\Pr[C_n(H_n^{t(n)}) = 1] - \Pr[C_n(H_n^0) = 1]\right| \geq \frac{1}{p(n)}$$

Using a standard hybrid argument it follows that there exists an $i \in \{0, \ldots, t(n)-1\}$ such that

$$\left|\Pr[C_n(H_n^i) = 1] - \Pr[C_n(H_n^{i+1}) = 1]\right| \geq \frac{1}{t(n) \cdot p(n)}$$

By hardwiring all of the x_j and y_k values into a circuit C'_n, we obtain a polynomial-size circuit family that will distinguish single encryptions. In particular, circuit C'_n receives an encryption β that is either of x_{i+1} or y_{i+1}. It then encrypts (by itself) the values $x_1, \ldots, x_i$ and $y_{i+2}, \ldots, y_{t(n)}$, and invokes C_n upon the vector of ciphertexts in which β is placed in the $(i+1)^{\text{th}}$ position. If β is an encryption of x_{i+1}, then it follows that C'_n invoked C_n upon input H_n^{i+1}. In contrast, if β is an encryption of y_{i+1}, then C'_n invoked C_n upon input H_n^i. It follows that C_n's ability to distinguish H_n^i from H_n^{i+1} can be translated into C'_n's ability to distinguish x_{i+1} from y_{i+1}. We stress that this reduction works only because C'_n is able to encrypt by itself, since it is given the public-key. ■

9.2.2 Multiple Encryptions in the Private-Key Setting

Theorem 9.9 *Assuming the existence of one-way functions (that are hard to invert for polynomial-size circuits), there exists an encryption scheme that has indistinguishable encryptions for a single message, but does not satisfy Definition 9.7.*

The idea behind the proof of this theorem is as follows. Let PRG be a pseudorandom generator (such a PRG exists assuming the existence of one-way functions). Then, define $e = G(1^n) = U_n$, $E_e(x) = PRG(e) \oplus x$ and $D_e(y) = PRG(e) \oplus y$. (We note that in order for this to work, the generator PRG must stretch e to a length that is any polynomial in n, and not just to a predetermined polynomial length.) It is not difficult to prove that Theorem 9.9 holds for this encryption scheme. We leave the details as an exercise.

Conclusion: *When constructing encryption schemes in the private-key setting, the security of the encryption scheme must be explicitly proven for multiple encryptions.*

Lecture 10

Encryption Schemes II

In this lecture, we will show how to construct secure encryption schemes in both the private and public-key models. We will also extend the definitions of secure encryption to deal with active adversaries that can carry out chosen-plaintext and chosen-ciphertext attacks.

10.1 Constructing Secure Encryption Schemes

10.1.1 Private-Key Encryption Schemes

In this section, we will show how to construct private-key encryption schemes that are secure for multiple messages. The construction uses pseudorandom functions (recall that these can be constructed from any one-way function). We will construct a scheme that encrypts plaintexts of length exactly n (and we assume that the pseudorandom function has input and output of exactly n bits). In order to obtain a general encryption scheme, where plaintexts can be of any polynomial length, it is possible to just parse the plaintext into blocks of at most n (with appropriate padding for the last block), and encrypt each block separately. This "multiple-encryption" approach is facilitated by the fact that we prove the security of the basic scheme for *multiple* encryptions.

Construction 10.1 (private-key encryption – block ciphers): *Let $F = \{F_n\}$ be an efficiently computable function ensemble, and let I and V be the sampling and evaluation functions, respectively, as in Definition 5.1. Then, define (G, E, D) as follows:*

- Key generation: $G(1^n) = (k, k)$*, where* $k \leftarrow I(1^n)$.
- Encryption of $x \in \{0,1\}^n$ using key k: $E_k(x) = (r, V(k,r) \oplus x)$ *where* $r \in_R \{0,1\}^n$.
- Decryption of (r, y) using key k: $D_k(r,y) = V(k,r) \oplus y$.

We proceed to prove the security of the above construction:

Theorem 10.2 *Assume that F is a family of functions that is pseudorandom to polynomial-size circuits. Then, (G, E, D) described in Construction 10.1 is a private-key encryption scheme with indistinguishable encryptions for multiple messages.*

[*] Lecture notes for a graduate course in the theory of cryptography. Yehuda Lindell, Bar-Ilan University, Israel, 2005.

Proof Sketch: Assume first that F is a truly random function family. Let $x_1, \ldots, x_{t(n)}$ and $y_1, \ldots, y_{t(n)}$ be two vectors of plaintexts. Now, assume that the r values appearing in all the ciphertexts are distinct. (That is, let $c_i = (r_i, f_i)$. Then, assume that for all $i \neq j$ it holds that $r_i \neq r_j$.) Then, it follows that $E_k(x_i)$ is distributed identically to $E_k(y_i)$. This is due to the fact that F is truly random and so if all r_i's are distinct, then all $F(r_i)$'s are independently and uniformly distributed in $\{0,1\}^n$. The probability of distinguishing a vector of encryptions of the x values from a vector of encryptions of the y values therefore equals at most $t(n)^2 \cdot 2^{-n}$ (which upper bounds the probability that at least one pair i and j are such that $r_i = r_j$).

Next, we replace the random function by a pseudorandom one and claim that if the scheme now becomes "distinguishable", then this yields a distinguisher for the pseudorandom function. ■

We note the use of random coin tosses in the encryption process. We will not elaborate on this, as it was discussed at length in the course "Introduction to Cryptography" (89-656).

10.1.2 Public-Key Encryption Schemes

In this section, we will show how to construct public-key encryption schemes. In contrast to private-key encryption, our constructions here require the use of trapdoor one-way permutations (rather than one-way functions). We will first show how to encrypt a single bit; encryption of arbitrary-length messages can be obtained by encrypting each bit separately. (Recall that in the public-key setting, the extension from the single-message to the multiple-message setting is automatic.)

Encrypting a single bit. Let (I, S, F, F^{-1}) be a collection of trapdoor one-way permutations, as in Definition 2.3 (we denote the domain sampling algorithm by S so as not to confuse it with the decryption algorithm D). The idea behind this construction is to encrypt a bit σ by outputting $(f(U_n), b(U_n) \oplus \sigma)$, where f is the sampled permutation and b is a hard-core predicate of f. Recall that this output is pseudorandom (as proven earlier in the course), and so intuitively σ will be "hidden" as required. Furthermore, notice that decryption can be carried out using the trapdoor. In particular, given $(f(r), \tau)$ where $\tau = b(r) \oplus \sigma$, it is possible to use the trapdoor to obtain r and then compute $b(r)$ and $\sigma = \tau \oplus b(r)$.

Construction 10.3 (public-key encryption): *Let (I, S, F, F^{-1}) be a collection of trapdoor permutations and let B be a predicate. Then, define (G, E, D) as follows:*

- Key generation: *$G(1^n) = (i, t)$, where $i = I_1(1^n)$ is the first element of the output of $I(1^n)$ and t is the second element of the output of $I(1^n)$, or the "trapdoor".*
- Encryption of $\sigma \in \{0,1\}$ using key i: *Sample a random element x according to $S(i)$ and compute $y = F(i, x)$. Output (y, τ) where $\tau = B(i, x) \oplus \sigma$.*
- Decryption of (y, τ) using key (i, t): *Compute $x = F^{-1}(t, y)$ and output $\sigma = B(i, x) \oplus \tau$.*

We now prove the security of the above construction:

Theorem 10.4 *Assume that (I, S, F, F^{-1}) is a collection of trapdoor one-way permutations and that B is a hard-core predicate of (I, S, F), where hardness holds for non-uniform adversaries. Then, (G, E, D) described in Construction 10.3 is a public-key encryption scheme with indistinguishable encryptions.*

Proof Sketch: The fact that (G, E, D) constitutes an encryption scheme is immediate. Regarding its security, it suffices to show that for $i \leftarrow I(1^n)$, it is hard to distinguish between the distributions $\{i, E_i(0)\} \equiv \{i, (F(i,x), B(i,x))\}$ and $\{i, E_i(1)\} \equiv \{i, (F(i,x), \overline{B}(i,x))\}$, where $i \leftarrow I(1^n)$ and $x \leftarrow S(i)$. In the proof of Theorem 4.7 we proved an almost identical fact. (The only difference here is that we are using a *collection* of permutations, instead of a single one-way permutation. However, this makes no difference to the specific fact being referred to.) We thus conclude that the constructed encryption scheme is secure. ■

The encryption scheme of Construction 10.3 is very inefficient. In particular, for every bit of the plaintext, the ciphertext contains $n+1$ bits. Thus, the size of the ciphertext is n times the size of the plaintext. We will now present a construction that is more efficient with respect to the bandwidth. (Unfortunately, it still requires an application of $F(i,\cdot)$ for every bit of the plaintext and so is still computationally very heavy.)

Construction 10.5 (public-key encryption – efficient construction): *Let (I, S, F, F^{-1}) be a collection of trapdoor permutations and let B be a predicate. Then, define (G, E, D) as follows:*

- Key generation: *$G(1^n) = (i, t)$, where $i = I_1(1^n)$ is the first element of the output of $I(1^n)$ and t is the second element of the output of $I(1^n)$, or the "trapdoor".*
- Encryption of $\sigma \in \{0,1\}^\ell$ using key i: *Let $\sigma = \sigma_0 \cdots \sigma_{\ell-1}$ where for every j, $\sigma_j \in \{0,1\}$. Sample a random element x according to $S(i)$ and set $x_0 = x$. Then, for $j = 1, \ldots, \ell$, set $x_j = F(i, x_{j-1})$. Finally, for $j = 0, \ldots, \ell-1$, set $\tau_j = B(i, x_j) \oplus \sigma_j$ and output $(x_\ell, \tau_0, \ldots, \tau_{\ell-1})$.*
- Decryption of $(x_\ell, \tau_0, \ldots, \tau_{\ell-1})$ using key (i, t): *Compute $x_0 = F^{-\ell}(t, x_\ell)$ and then reconstruct the series $B(i, x_0), \ldots, B(i, x_{\ell-1})$ and output $\sigma = \sigma_0 \cdots \sigma_{\ell-1}$ by computing $\sigma_j = \tau_j \oplus B(i, x_j)$ for every j.*

We prove the security of the above construction:

Theorem 10.6 *Assume that (I, S, F, F^{-1}) is a collection of trapdoor one-way permutations and that B is a hard-core predicate of (I, S, F), where hardness holds for non-uniform adversaries. Then, (G, E, D) described in Construction 10.5 is a public-key encryption scheme with indistinguishable encryptions.*

Proof Sketch: The fact that (G, E, D) constitutes an encryption scheme is straightforward. Denote by $G_i^\ell(x_0)$ the series $B(i, x_0), \ldots, B(i, x_{\ell-1})$. Then, in order to prove security, we need to prove that for every polynomial ℓ and every $\sigma, \sigma' \in \{0,1\}^{\ell(n)}$, it holds that

$$\left\{i, F^\ell(i, x_0), G_i^\ell(x_0) \oplus \sigma\right\}_{n\in\mathbb{N}} \stackrel{c}{\equiv} \left\{i, F^\ell(i, x_0), G_i^\ell(x_0) \oplus \sigma'\right\}_{n\in\mathbb{N}} \tag{10.1}$$

where $i \leftarrow I(1^n)$ and $x_0 \leftarrow S(i)$, and where indistinguishability holds with respect to polynomial-size circuits. We show that

$$\left\{i, F^\ell(i, x_0), G_i^\ell(x_0)\right\}_{n\in\mathbb{N}} \stackrel{c}{\equiv} \left\{i, F^\ell(i, x_0), U_\ell\right\}_{n\in\mathbb{N}} \tag{10.2}$$

which clearly implies Eq. (10.1). (We note that the entire distribution is not necessarily pseudorandom because $F^\ell(i, x_0)$ may not be pseudorandom. This is due to the fact that the domain of F is not necessarily $\{0,1\}^n$.) In order to prove Eq. (10.2) we rely on the fact that

$$\{i, F(i,x), B(i,x)\}_{n\in\mathbb{N}} \stackrel{c}{\equiv} \{i, F(i,x), U_1\}_{n\in\mathbb{N}} \tag{10.3}$$

where $i \leftarrow I(1^n)$, $x \leftarrow S(i)$ and indistinguishability relates to polynomial-size circuits. (From here on, all references to i and x will be like here, and so we will not explicitly write that they are chosen this way each time.) The proof of Eq. (10.3) is analogous to the proof of Theorem 4.7. We proceed to prove Eq. (10.1) by a hybrid argument. Define the hybrid distribution

$$H_i^\ell(x_0, j) = \Big(i, F^\ell(i, x_0), U_j, B(i, x_j), \ldots, B(i, x_{\ell-1})\Big)$$

Then, $\{H_i^\ell(x_0, 0)\} \equiv \{i, F^\ell(i, x_0), G_i^\ell(x_0)\}$ and $\{H_i^\ell(x_0, \ell)\} \equiv \{i, F^\ell(i, x_0), U_\ell\}$. A standard hybrid argument yields that if Eq. (10.2) does not hold, then there exists a j such that $H_i^\ell(x_0, j)$ can be distinguished from $H_i^\ell(x_0, j+1)$ with non-negligible probability; let D be such a distinguisher. We contradict Eq. (10.3) by constructing a distinguisher D' who receives (i, y, b) where $y = F(i, x)$, and distinguishes between the case that $b = B(i, x)$ from the case that b is uniformly distributed. D' works as follows. Given (i, y, b), distinguisher D' chooses a random $j \in_R \{1, \ldots, \ell-1\}$, defines $x_j = y$ and invokes D upon input $(i, F^{\ell-j}(x_j), U_{j-1}, b, B(i, x_j), \ldots, B(i, x_{\ell-1}))$ where $x_{j+1} = F(i, x_j)$ and so on. Notice that on the one hand, if b is uniformly distributed, then the distribution handed to D by D' is exactly $H_i^\ell(x_0, j)$. On the other hand, if $b = B(i, x) = B(i, F^{-1}(y)) = B(i, x_{j-1})$, then the distribution handed to D by D' is exactly $H_i^\ell(x_0, j-1)$. We therefore conclude that D' distinguishes the distributions in Eq. (10.3) with non-negligible probability, in contradiction.[1] ■

Notice that with respect to bandwidth, Construction 10.5 is very efficient. In particular, an encryption of a message of length $\ell(n)$ is of size only $\ell(n) + n$. However, the trapdoor permutation needs to be invoked once for every bit of the plaintext. Since this operation is very heavy, this encryption scheme has not been adopted in practice.

10.2 Secure Encryption for Active Adversaries

10.2.1 Definitions

The definitions of security that we have seen so far have referred to *eavesdropping adversaries* only. In many settings, this level of security does not suffice. In this section, we will introduce definitions of security that relate to more powerful adversaries. The adversary's power here is increased by allowing it to obtain encryptions (and possibly decryptions) of its choice during its attack. There are three different types of attacks that we will consider here:[2]

1. *Chosen plaintext attacks* (CPA): Here the adversary is allowed to obtain encryptions of any plaintext that it wishes. This is the default in the public-key setting, but adds power in the private-key setting.

2. *Passive chosen ciphertext attacks* (CCA1): Here the adversary is allowed to ask for decryptions of any ciphertext that it wishes, *up until the point* that it receives the "challenge ciphertext". (The challenge ciphertext is the encryption of one of the values; the aim of the adversary is to guess which plaintext was encrypted in this ciphertext.)

3. *Adaptive chosen ciphertext attacks* (CCA2): Here the adversary is allowed to ask for decryptions of any ciphertext that it wishes, *even after* it receives the "challenge ciphertext". The

[1] We note that the hybrid distribution in this proof was constructed with the uniformly distributed bits first, because otherwise D' would not be able to construct the hybrid distribution from the input (i, y, b).

[2] For motivation regarding these attacks and why they are interesting and/or important, see the lecture notes for the course "Introduction to Cryptography" (89-656).

only limitation is that the adversary is not allowed to ask for a decryption of the challenge ciphertext itself.

In the definition below, the adversary is given access to oracles for encryption and decryption. As for eavesdropping adversaries, semantic security is equivalent to indistinguishability for all three types of attacks; we do not prove this fact here (but note that the proof is essentially the same). We begin by defining the CPA experiment, for $b \in \{0,1\}$. We note that in order to differentiate between the adversary's computation before and after it receives the challenge ciphertext, we consider an adversary $\mathcal{A}$ that is actually a *pair* of PPT machines $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$. This is just a technicality and the given pair should be thought of as a single adversary.

$\mathbf{Expt}^{\mathbf{CPA}}_{\mathcal{A},n,z}(b)$**:**

1. *Key generation:* A key-pair $(e,d) \leftarrow G(1^n)$ is chosen.
2. *Oracle access and challenge plaintext generation:* The adversary $\mathcal{A}_1$ is given the auxiliary input z and access to an encryption oracle E_e, and outputs a pair of plaintexts $x_0, x_1 \in \{0,1\}^{\mathrm{poly}(n)}$, where $|x_0| = |x_1|$. In addition, $\mathcal{A}_1$ outputs some "state information" s for $\mathcal{A}_2$. Formally,
$$(x_0, x_1, s) \leftarrow \mathcal{A}_1^{E_e(\cdot)}(1^n, z)$$
3. *Compute the challenge ciphertext:* $c = E_e(x_b)$
4. *Challenge receipt and additional oracle access:* The adversary $\mathcal{A}_2$ is given state information s output by $\mathcal{A}_1$, an encryption of x_b and additional oracle access, and outputs a "guess" for which value was encrypted. Formally,
$$\beta \leftarrow \mathcal{A}_2^{E_e(\cdot)}(s, c)$$
5. *Experiment result:* Output β

The CCA1 and CCA2 experiments are defined analogously. In the definition of CCA2, the oracle $D_d^{\neg c}$ is a regular decryption oracle, except that it answers with the empty string if it receives c in an oracle query.

$\mathsf{Expt}^{\mathrm{CCA1}}_{\mathcal{A},n,z}(b)$:

$$\begin{aligned} &(x_0, x_1, s) \leftarrow \mathcal{A}_1^{E_e(\cdot), D_d(\cdot)}(1^n, z)\\ &c = E_e(x_b)\\ &\beta \leftarrow \mathcal{A}_2^{E_e(\cdot)}(s, c)\\ &\text{Output } \beta \end{aligned}$$

$\mathsf{Expt}^{\mathrm{CCA2}}_{\mathcal{A},n,z}(b)$:

$$\begin{aligned} &(x_0, x_1, s) \leftarrow \mathcal{A}_1^{E_e(\cdot), D_d(\cdot)}(1^n, z)\\ &c = E_e(x_b)\\ &\beta \leftarrow \mathcal{A}_2^{E_e(\cdot), D_d^{\neg c}(\cdot)}(s, c)\\ &\text{Output } \beta \end{aligned}$$

We now define the notion of indistinguishability under CPA, CCA1 and CCA2 attacks.

Definition 10.7 *A public-key encryption scheme* (G, E, D) *is said to have* indistinguishable encryptions under chosen plaintext attacks *if for every pair of probabilistic polynomial-time oracle machines* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, every positive polynomial* $p(\cdot)$*, and all sufficiently large* n *and* $z \in \{0,1\}^{\mathrm{poly}(n)}$*, it holds that*

$$\left|\Pr\left[\mathsf{Expt}^{\mathrm{CPA}}_{\mathcal{A},n,z}(0) = 1\right] - \Pr\left[\mathsf{Expt}^{\mathrm{CPA}}_{\mathcal{A},n,z}(1) = 1\right]\right| < \frac{1}{p(n)} \tag{10.4}$$

Analogously, (G, E, D) *has indistinguishable encryptions under passive (resp., adaptive) chosen-ciphertext attacks if* $\mathsf{Expt}^{\mathrm{CPA}}_{\mathcal{A},n,z}(b)$ *in Eq. (10.4) is replaced with* $\mathsf{Expt}^{\mathrm{CCA1}}_{\mathcal{A},n,z}(b)$ *(resp.,* $\mathsf{Expt}^{\mathrm{CCA2}}_{\mathcal{A},n,z}(b)$*).*

10.2.2 Constructions

We first note that the public-key encryption schemes described in Constructions 10.3 and 10.5 are secure under chosen-plaintext attacks. Intuitively, this follows from the fact that in the public-key setting, the adversary is anyway given the encryption key (and so can encrypt by itself). Technically, the definitions are actually not equivalent because the challenge plaintexts in a chosen-plaintext attack may depend on the public-key, which is not the case in the eavesdropping case. Nevertheless, it is not hard to see that the same proof goes through also for the case of CPA-attacks.

Next, we note that the *private-key* encryption scheme described in Construction 10.1 is also secure under chosen-plaintext attacks. This does *not* follow from the definitions, because the adversary's power in the eavesdropping setting is strictly weaker than in a CPA attack (even if it receives multiple encryptions). Nevertheless, this is also easily demonstrated and we leave it as an exercise.

CCA2-security in the private-key setting. In the course "Introduction to Cryptography", we stated that CCA2-secure private-key encryption can be obtained by first encrypting the message using a private-key encryption scheme that is CPA-secure, and then applying a message authentication code (MAC) to the result. We leave the formal definition and proof of this construction as an exercise.

CCA1 and CCA2 security in the public-key setting. Unfortunately, constructions for these settings are far more complicated than in the private-key setting. We leave these (specifically CCA2) as a reading assignment. This also involves learning non-interactive zero-knowledge which we will describe on a high-level in class. The reading material will be placed on the course website. Start with the material on non-interactive zero-knowledge, taken from the fragments of a book preceding [5], and focus on understanding the model and definitions. (It is also important to understand the constructions, but the definitions are crucial for the next stage.) Then, read the paper that presents a construction of a public-key encryption scheme that is CCA2-secure (read the appendix as well). Focus on the main ideas (and not technicalities that you may not be familiar with).

Lecture 11

Digital Signatures I

In this lecture, we will present definitions of digital signatures and show how to construct full-fledged digital signatures from length-restricted ones. As regarding encryption schemes, we will not present motivation to the notion of digital signatures, and will not discuss applications. This material can be found in the lecture notes for the course "Introduction to Cryptography" (89-656).

11.1 Defining Security for Signature Schemes

We begin by presenting the syntax of digital signature schemes.

Definition 11.1 *A* digital signature scheme *consists of a triple of probabilistic polynomial-time algorithms (G, S, V) satisfying the following conditions:*

1. *On input 1^n, the* key-generator *algorithm G outputs a pair of keys (s, v).*
2. *For every pair (s, v) in the range of $G(1^n)$ and for every $\alpha \in \{0,1\}^*$, the* signing *and* verification *algorithms S and V satisfy*
$$\Pr[V(v, \alpha, S(s, \alpha)) = 1] = 1$$
where the probability is taken over the internal coin tosses of algorithms S and V.

The integer n serves as the security parameter *of the scheme. The key s is called the* signing key*, the key v is called the* verification key*, and a string $\sigma = S(s, \alpha)$ is called a* digital signature*. For shorthand, we will often denote $S_s(\alpha) = S(s, \alpha)$ and $V_v(\alpha, \sigma) = V(v, \alpha, \sigma)$.*

As with encryption, the requirement that verification always succeeds can be relaxed to allow failure with negligible probability.

Defining security for signature schemes. The security requirement for a signature scheme states that a PPT adversary $\mathcal{A}$ should succeed in generating a valid "forgery" with at most negligible probability. In order to model the fact that the adversary $\mathcal{A}$ may see signatures generated by the real signer, $\mathcal{A}$ is given access to a signing oracle. Such an attack is called a chosen message attack (because $\mathcal{A}$ chooses the messages to be signed). In addition, the adversary $\mathcal{A}$ is said to succeed in generating an existential forgery if it generates a pair (α, σ) where $V(v, \alpha, \sigma) = 1$ and where α was not queried to the signing oracle. (The forgery is "existential" because there are no constraints on the message α.) More formally, the following experiment is defined:

* Lecture notes for a graduate course in the theory of cryptography. Yehuda Lindell, Bar-Ilan University, Israel, 2005.

1. A key-pair $(s,v) \leftarrow G(1^n)$ is chosen.
2. The adversary $\mathcal{A}$ is given input $(1^n, v)$ and access to the oracle $S_s(\cdot)$. Let $\mathcal{Q}$ denote the set of oracle queries made by $\mathcal{A}$ in this stage.
3. $\mathcal{A}$ outputs a pair (α,σ). We say that $\mathcal{A}$ succeeds, denoted $\mathsf{succeed}_{\mathcal{A}}(n) = 1$, if $\alpha \notin \mathcal{Q}$ and $V(v,\alpha,\sigma) = 1$. (If V is probabilistic, then $\mathcal{A}$ succeeds if $\Pr[V(v,\alpha,\sigma)=1]$ is non-negligible.)

We are now ready to present the definition:

Definition 11.2 *A signature scheme (G,S,V) is* existentially secure against chosen-message attacks *(or just* secure*) if for every probabilistic polynomial-time adversary $\mathcal{A}$, every polynomial $p(\cdot)$ and all sufficiently large n's*

$$\Pr[\mathsf{succeed}_{\mathcal{A}}(n) = 1] < \frac{1}{p(n)}$$

where the probability is taken over the coin-tosses of G, S, V and $\mathcal{A}$.

11.2 Length-Restricted Signatures

In the definition of what constitutes a signature scheme (Definition 11.1), the input message α can be of any length. Furthermore, since the adversary can carry out a chosen message attack, and it succeeds if it forges any message, the messages generated by the adversary may be of any polynomial length (depending only on the running-time of $\mathcal{A}$). It is useful when constructing signature schemes to first construct a scheme that is only defined for messages of a specified length. Then, such a "length-restricted" scheme is converted into a "full-fledged" signature scheme.

Definition 11.3 (length-restricted signature schemes): *Let $\ell : \mathbb{N} \rightarrow \mathbb{N}$. An* ℓ-length-restricted signature scheme *is a signature scheme as in Definition 11.1 with the additional requirement that for every n, every $(s,v) \leftarrow G(1^n)$, and all $\alpha \notin \{0,1\}^{\ell(n)}$, it holds that $S_s(\alpha) = \perp$ and $V_v(\alpha,\sigma) = 0$, for all σ. Security is defined exactly as in Definition 11.2.*

Notice that the difference between regular (or full-fledged) signature schemes is just that a length-restricted scheme is really only defined over strings of exactly $\ell(n)$ (for all other lengths, S and V just output default values). The fact that V always outputs 0 for strings of different lengths means that success (as in Definition 11.2) can only hold for messages of length exactly $\ell(n)$.

11.2.1 From Length-Restricted to Full-Fledged Signature Schemes

In this section, we show that as long as $\ell(\cdot)$ is a "large enough" function, then full-fledged signatures schemes can be constructed from any length-restricted signature scheme.

Theorem 11.4 *Assume that $\ell(\cdot)$ is a super-logarithmically growing function; i.e. $\ell(n) = \omega(\log n)$. Then, there exists a secure ℓ-restricted signature scheme if and only there exists a secure signature scheme.*

Proof: We show how to construct full-fledged signature scheme from an ℓ-restricted signature scheme, as long as $\ell(n) = \omega(\log n)$.[1] The idea behind the construction is to break the message up

[1] Recall that $f(n) \in \omega(g(n))$ if and only if for every constant $c > 0$ there exists an $N > 0$ such that for every $n > N$ it holds that $0 \leq cg(n) < f(n)$. In other words, $f(n)$ is asymptotically larger than $g(n)$. For our purposes here, one can think of $\ell(n) = n$ or $\ell(n) = \log^2 n$. We note that for $f(n) \in \omega(\log n)$, it holds that $2^{f(n)}$ is super-polynomial.

into blocks and apply the signature scheme separately to each block. This must be done carefully so that the order of the blocks cannot be rearranged and so that blocks from signatures on different messages cannot be intertwined. This is achieved by including additional information into every block. Specifically, in addition to part of the message, each block contains an index of its position in the series, in order to prevent rearranging the blocks. Furthermore, all the blocks in a signature contain the same random identifier. This prevents blocks from different signatures from being combined, because they will have different identifiers. Finally, all the blocks in a signature contain the total number of blocks, so that blocks cannot be dropped from the end of the message. We now proceed to the actual construction:

Construction 11.5 *Let ℓ be a super-logarithmic function and let (G, S, V) be an ℓ-restricted signature scheme. Let $\ell'(n) = \ell(n)/4$. Then, we construct a full-fledged signature scheme (G', S', V') as follows:*

1. Key generation algorithm G': *Set $G' = G$.*
2. Signing algorithm S': *On input a signing-key s in the range of $G'(1^n)$ and a message $\alpha \in \{0,1\}^*$, algorithm S' parses α into t blocks $\alpha_1, \ldots, \alpha_t$ each of length $\ell'(n)$. (In order to ensure unique encoding, the last block can always be padded with 10^*.)*

   Next, S' chooses a random $r \in_R \{0,1\}^{\ell'(n)}$. For $i = 1, \ldots, t$, algorithm S' computes $\sigma_i = S(r, t, i, \alpha_i)$, where i and t are uniquely encoded into strings of length $\ell'(n)$. Finally, S' outputs the signature $\sigma = (r, t, \sigma_1, \ldots, \sigma_t)$.[2]
3. Verification algorithm V': *On input (v, α, σ), where $\sigma = (r, t, \sigma_1, \ldots, \sigma_t)$, algorithm V' first parses α in the same way as S'. If there are t resulting blocks, then V' checks that for every i, $V(v, (r, t, i, \alpha_i), \sigma_i) = 1$. V outputs 1 if and only if both of these checks pass.*

Since (G', S', V') accepts messages of any length, it fulfills the requirements of being a "full-fledged" signature scheme, as in Definition 11.1. We proceed to prove that it is secure.

Proposition 11.6 *If (G, S, V) is a secure ℓ-restricted signature scheme and ℓ is a function that grows super-logarithmically, then (G', S', V') is a secure signature scheme.*

The remainder of the proof of Theorem 11.4 involves proving Proposition 11.6. We present a full proof (but rather informally) and refer to [6, Section 6.2] for full details. The intuition behind the proof is that if the random identifier r is different in every signature that the adversary receives from the oracle, then a forgery must either contain a new identifier or it must somehow manipulate the blocks of a signed message. In both cases, the adversary must generate a forgery for the underlying scheme (G, S, V) because the encoding of α prevents any modification to the order or number of blocks. More formally, an adversary $\mathcal{A}'$ that successfully forges a signature for the scheme (G', S', V') can be used to construct an adversary $\mathcal{A}$ that successfully forges a signature for the length-restricted scheme (G, S, V). Adversary $\mathcal{A}$ internally invokes $\mathcal{A}'$ and answers $\mathcal{A}'$'s oracle queries using its own oracle. Specifically, whenever $\mathcal{A}$ queries its signing oracle S' with a message α, adversary $\mathcal{A}$ chooses a random r and parses α in the same way as S' does. Next, $\mathcal{A}$ queries its own signing oracle S with each block, obtaining signatures σ_i, and returns the signature $\sigma = (r, t, \sigma_1, \ldots, \sigma_t)$ to $\mathcal{A}'$. Finally, when $\mathcal{A}'$ outputs (α, σ) where $\sigma = (r, t, \sigma_1, \ldots, \sigma_t)$, adversary $\mathcal{A}$ checks if it is a successful forgery. If yes, it searches the blocks in the output to see if there is an

[2] Notice that i and t can be encoded in $\ell'(n)$ bits because $t = \mathrm{poly}(n)$ and $\ell'(n) = \omega(\log n)$.

i such that $\mathcal{A}$ did not query (α_i, i, t, r) with its oracle, where α_i is constructed in the usual way. If yes, it outputs $((r, t, i, \alpha_i), \sigma_i)$ and halts. If not, it output fail and halts.

It remains to show that if $\mathcal{A}'$ generated a successful forgery, then with overwhelming probability, so did $\mathcal{A}$. Let (α, σ) where $\sigma = (r, t, \sigma_1, \ldots, \sigma_t)$ be a successful forgery for $\mathcal{A}'$ in the "game" with $\mathcal{A}$. We have the following cases:

1. *The identifier in the forged signature of $\mathcal{A}'$ is different from all identifiers generated by $\mathcal{A}$:* In this case, it is clear that every block of the signature constitutes a successful forgery for $\mathcal{A}$.

2. *The identifier in the forged signature equals the identifier in exactly one of the signatures supplied to $\mathcal{A}'$:* Let σ' be the signature received by $\mathcal{A}'$ with the same identifier as in the forgery, and let α' be the corresponding signed message. If the output of $\mathcal{A}'$ is a successful forgery, then the message α must be different to all previously signed messages, and in particular must be different to α'. Let t and t' be the number of blocks in the parsing of α and α', respectively. There are two subcases here:

   (a) *Case 1 – $t = t'$:* In this case, the α-content of one of the blocks must be different (i.e., for some i it must be that $(\alpha_i, i) \neq (\alpha'_i, i)$). This block constitutes a successful forgery for $\mathcal{A}$. In order to see this, note that $\mathcal{A}$ queried its oracle with (α'_i, i, t, r), and furthermore that there was only a single query of the form $(*, i, *, r)$. (This is due to the fact that r was only used once.) Since $(\alpha'_i, i) \neq (\alpha_i, i)$, we have that this block constitutes a forgery for $\mathcal{A}$.

   (b) *Case 2 – $t \neq t'$:* In this case, each block constitutes a successful forgery for $\mathcal{A}$. This follows from the following two facts. First, in this case, the only signature generated by $\mathcal{A}$ for $\mathcal{A}'$ that has the identifier r is the signature on the message α'. Second, $t' \neq t$. Combining these facts we have that $\mathcal{A}$ never queried its oracle with a message of the form $(r, t, *, *)$. However all of the blocks in $\mathcal{A}'$'s forgery are of this form. They therefore all constitute a successful forgery for $\mathcal{A}$.

3. *The identifier in the forged signature equals the identifier in at least two signatures supplied to $\mathcal{A}'$:* We rule out this case by showing the two signatures (generated legally) have the the same identifier with at most negligible probability. Now, the length of a random identifier is $\ell'(n) = \ell(n)/4$ and $\ell(n)$ is super-logarithmic. Therefore $2^{-\ell'(n)}$ is a negligible function. Now, for m signatures, the probability that at least two signatures have the same identifier is $\binom{m}{2} \cdot 2^{-\ell'(n)} = \text{poly}(n) \cdot \mu(n)$ which is negligible.

This completes the proof. ■

Notice that Theorem 11.4 is very strong in that it requires no additional assumptions for constructing a full-fledge signature scheme from an ℓ-restricted one. It is therefore useful for proving feasibility results.

11.2.2 Collision-Resistant Hash Functions and Extending Signatures

In this section, we present an alternative way of constructing full-fledged signature schemes from length-restricted ones. Specifically, the message is first hashed with a collision-resistant hash function, and the result is then signed. This methodology requires additional complexity assumptions (namely, the existence of collision-resistant hash functions), but is more efficient, at least with respect to bandwidth. Another advantage of the hash-based extension is that it is also applicable to *one-time* signatures schemes (these are signature schemes that can be used to sign only a single message). We will study one-time signatures schemes in Section 12.2.

We begin by defining the notion of collision-resistant hash functions. (Once again, we refer students to the lecture notes for the course "Introduction to Cryptography" (89-656) for background material and motivation.)

Definition 11.7 (collision-resistant hash functions): *Let $\ell : \mathbb{N} \to \mathbb{N}$. A collection of functions $H = \{h_r : \{0,1\}^* \to \{0,1\}^{\ell(|r|)}\}$ is called a family of* collision-resistant hash functions *if there exists a probabilistic polynomial-time sampling algorithm I such that the following holds:*

1. *There exists a polynomial-time algorithm that, given r and x, returns $h_r(x)$.*

2. *For every probabilistic polynomial-time algorithm $\mathcal{A}$, every positive polynomial $p(\cdot)$ and all sufficiently large n's*

$$\Pr\left[\mathcal{A}(I(1^n), 1^n) = (x, x') \ \& \ x \neq x' \ \& \ h_{I(1^n)}(x) = h_{I(1^n)}(x')\right] < \frac{1}{p(n)}$$

where the probability is taken over the coin tosses of I and $\mathcal{A}$.[3]

The function $\ell(\cdot)$ is called the range-length *of H.*

We note that any family of collision-resistant hash functions is a collection of one-way functions. We leave the proof of this as an exercise.

The hash-and-sign construction for signatures is simple: In order to sign on an arbitrary-length message α, first compute $h_r(\alpha)$ in order to obtain a message of length ℓ and then apply an ℓ-restricted signature scheme. In order to simplify the description, we assume that $|r| = n$ and thus the output of the hash function is exactly $\ell(n)$ for all $r \leftarrow I(1^n)$. A more formal description of the construction is as follows:

Construction 11.8 *Let (G, S, V) be an ℓ-restricted signature scheme and let H be a family of hash functions with range-length ℓ. Then, define (G', S', V') as follows:*

1. Key generation G': *Upon input 1^n, compute $(s, v) \leftarrow G(1^n)$ and $r \leftarrow I(1^n)$. Let $s' = (s, r)$ and $v' = (v, r)$; output (s', v').*

2. Signature algorithm S': *Given key (s, r) and message α, output $\sigma = S_s(h_r(\alpha))$.*

3. Verification algorithm V': *Given key (v, r), message α and signature σ, output $V_v(h_r(\alpha), \sigma)$.*

We have the following proposition:

Proposition 11.9 *Let $\ell : \mathbb{N} \to \mathbb{N}$. Assume that (G, S, V) is a secure ℓ-restricted signature scheme and that H is a family of collision-resistant hash functions with range-length ℓ. Then, (G', S', V') from Construction 11.8 is a secure signature scheme.*

Note that the "and" in the probability is not between two random variables, because the right-hand side has no probability. Rather it should be interpreted as "such that".

Proof Sketch: We present only a very brief overview of the proof. Given an adversary $\mathcal{A}'$ who finds a collision in (G', S', V') we construct $\mathcal{A}$ who either finds a collision in H or computes a forgery in (G, S, V). Adversary $\mathcal{A}$ invokes $\mathcal{A}'$ and upon receiving an oracle query α, it queries its own oracle with $h_r(\alpha)$. Assume that $\mathcal{A}'$ outputs a successful forgery (α', σ'). Then there are two cases:

1. There exists a message α that was previously queried such that $h_r(\alpha) = h_r(\alpha')$: In this case, $\mathcal{A}'$ found a collision in H.

2. There does not exist such a message α: In this case, $\mathcal{A}'$ obtained a forgery in (G, S, V).

The full proof works by constructing two adversaries: one corresponding to each case. Then, if $\mathcal{A}'$ succeeds in generating a forgery with non-negligible probability, one of the two adversaries must also succeed with non-negligible probability. ■

11.2.3 Constructing Collision-Resistant Hash Functions

Unfortunately, due to lack of time, we do not have time to show how to construct collision-resistant hash functions (with proven security). The construction is not complicated and is well-worth reading. This material can be found in [6, Section 6.2.3]; it refers to claw-free permutations that are presented in [5, Section 2.4.5]. We note that claw-free permutations, and thus collision-resistant hash functions, can be constructed assuming that the factoring or discrete log problems are hard.

Lecture 12

Digital Signatures II

In this lecture, we will show how to construct secure digital signatures.

12.1 Minimal Assumptions for Digital Signatures

Until now, we have seen that "private-key cryptography" can be obtained assuming only the existence of one-way functions. This is true of primitives, such as pseudorandom generators and functions, as well as applications such as private-key encryption and message authentication codes. In contrast, it is not known how to construct public-key encryption from one-way functions (and black-box constructions of public-key encryption from one-way functions have been proven to not exist). Rather, known (general) constructions of public-key encryption schemes require the existence of trapdoor one-way permutations. Intuitively, this is due to the fact that decryption involves inverting something, and so some trapdoor information is needed.

In contrast to encryption, the "public-key" problem of digital signatures can be solved relying only on the existence of one-way functions. That is, we have the following theorem:

Theorem 12.1 *Secure digital signature schemes exists if and only if one-way functions exist.*

We will not prove the above result, but will rather present a simpler construction that relies on collision-resistant hash functions. Due to lack of time will also only present a *memory-dependent* signature scheme, meaning that the signing algorithm must record the history of all previous signatures. This actually suffices for many applications, although it is clearly undesirable. We will also outline the construction of a full memoryless signature scheme, and refer to [6, Section 6.4] for a full description.

12.2 Secure One-Time Signature Schemes

We begin by constructing signature schemes that are secure as long as only a single message is signed. These are of independent interest, but also form the basis of our general construction. We omit a formal definition of security for one-time signature schemes, but note that it suffices to limit the adversary to a *single oracle query*, and everything else remains the same.

[*] Lecture notes for a graduate course in the theory of cryptography. Yehuda Lindell, Bar-Ilan University, Israel, 2005.

12.2.1 Length-Restricted One-Time Signature Schemes

Construction 12.2 (an ℓ-restricted one-time signature scheme): *Let $\ell : \mathbb{N} \to \mathbb{N}$ be polynomially bounded and polynomial-time computable, and let $f : \{0,1\}^* \to \{0,1\}^*$ be a polynomial-time computable function. The scheme (G, S, V) is defined as follows:*

1. Key generation G: *On input 1^n, G uniformly chooses strings $s_1^0, s_1^1, \ldots, s_{\ell(n)}^0, s_{\ell(n)}^1 \in_R \{0,1\}^n$, and computes $v_i^b = f(s_i^b)$, for every $1 \le i \le \ell(n)$ and $b \in \{0,1\}$. G outputs the keys (s, v) where $s = ((s_1^0, s_1^1), \ldots, (s_{\ell(n)}^0, s_{\ell(n)}^1))$ and $v = ((v_1^0, v_1^1), \ldots, (v_{\ell(n)}^0, v_{\ell(n)}^1))$.*

2. Signing algorithm S: *Upon input s and $\alpha \in \{0,1\}^{\ell(n)}$, algorithm S outputs $\sigma = (s_1^{\alpha_1}, \ldots, s_{\ell(n)}^{\alpha_{\ell(n)}})$ where $\alpha = \alpha_1 \cdots \alpha_{\ell(n)}$.*

3. Verification algorithm V: *Upon input v, $\alpha \in \{0,1\}^{\ell(n)}$ and σ, check that for every $i = 1, \ldots, \ell(n)$ it holds that $f(s_i^{\alpha_i}) = v_i^{\alpha_i}$.*

Proposition 12.3 *If f is a one-way function, then Construction 12.2 constitutes a secure ℓ-restricted one-time signature scheme.*

Proof Sketch: The intuition behind the security of Construction 12.2 is that a forgery can only be obtained by inverting the one-way function f. Specifically, recall that the adversary can only see a single signature before it outputs its forgery; let α be this message and let α' be the message of the forgery. Now, if the adversary succeeds, it must be that $\alpha' \neq \alpha$ and so for some i it holds that $\alpha'_i \neq \alpha_i$. The signature that the adversary received for α contained the pre-image $s_i^{\alpha_i}$ and *not* the pre-image $s_i^{1-\alpha_i} = s_i^{\alpha'_i}$. Thus, if the adversary presented a valid forgery for α', it follows that it must have inverted the value $v_i^{\alpha'_i}$. This contradicts the assumption that f is one-way.

The actual proof works as follows. An adversary $\mathcal{A}_{\rm owf}$ who wishes to invert the one-way function receives some image y and attempts to output $x \in f^{-1}(y)$. Adversary $\mathcal{A}_{\rm owf}$ chooses a random position (j, β) for y and $2\ell(n) - 1$ random values s_i^b (here we need ℓ to be polynomially bounded and computable). Then, for every $i \neq j$, adversary $\mathcal{A}_{\rm owf}$ defines $v_i^b = f(s_i^b)$. In addition, $\mathcal{A}_{\rm owf}$ defines $v_j^\beta = y$ and $v_j^{1-\beta} = f(s_j^{1-\beta})$. The adversary $\mathcal{A}_{\rm sig}$ is then invoked with the public-key defined by the strings $((v_1^0, v_1^1), \ldots, (v_{\ell(n)}^0, v_{\ell(n)}^1))$. Now, with probability 1/2, the message α that $\mathcal{A}_{\rm sig}$ queries to its oracle can be signed by $\mathcal{A}_{\rm owf}$ (specifically, $\mathcal{A}_{\rm owf}$ can sign as long as $\alpha_j \neq \beta$). Following this, $\mathcal{A}_{\rm sig}$ outputs a pair (α', σ'). If the forgery is "good", and $\alpha'_j = \beta$, then the signature must contain $f^{-1}(y)$. In such a case, we have that $\mathcal{A}_{\rm owf}$ has succeeded in inverting f, as required. Since α' must differ from α in at least one position, and since j is randomly chosen (and the choice is independent of $\mathcal{A}_{\rm sig}$'s view), we have that $\alpha'_j = \beta$ with probability at least $1/\ell(n)$. We conclude that $\mathcal{A}_{\rm owf}$ succeeds in inverting f with $1/2\ell(n)$ times the probability that $\mathcal{A}_{\rm sig}$ succeeds in generating a forgery. This completes the proof sketch. ■

12.2.2 General One-Time Signature Schemes

We have already seen that by first hashing an arbitrary-length message and then applying a secure length-restricted signature scheme, we obtain a secure signature scheme. The same holds for one-time signatures (observe that the proof of Proposition 11.9 goes through also for one-time signature schemes). Recalling that any family of collision-resistant hash functions is one-way, we conclude with the following theorem:

Theorem 12.4 *If there exist collision-resistant hash functions, then there exist secure one-time signature schemes.*

12.3 Secure Memory-Dependent Signature Schemes

In this section, we will construct a secure memory-dependent signature scheme from any one-time signature scheme. A memory-dependent scheme has the property that the signing algorithm S maintains internal state between generating signatures. Stated otherwise, S maintains a record of all the previously generated signatures. We stress that the signing oracle provided to the adversary maintains state in the same way. We omit a formal definition of memory-dependent signature schemes.

Memory-dependent signature schemes are clearly weaker than memoryless ones. Nevertheless, in many applications (like, where a smartcard generates the signatures and the signing key resides only there), memory-dependent schemes suffice. We present only a memory-dependent scheme due to lack of time (and not because we really think that such schemes suffice). We will also present the high-level idea behind the construction of a memoryless scheme.

Motivation for the construction. The idea behind the construction is to essentially use the one-time signature in order to sign on the first message *and* a new (fresh) key of the one-time signature scheme. The second message is then signed by choosing another fresh key, and then signing on the new message/key pair with the key from the previous signature. The signing process continues in this way, and so we actually obtain chains of signatures. Signatures are verified by first checking that the chain of keys to the current one were all validly signed (i.e., check that the first fresh key was signed with the original key, that the second fresh key was signed with the first fresh key, and so on). Such a valid chain should convince us that the original signer signed all messages. This methodology is called the *refreshing paradigm*. We now present the actual construction.

Construction 12.5 *Let (G, S, V) be a one-time signature scheme. Then, define (G', S', V') as follows:*

1. Key generation G': *Set $G' = G$.*
2. Signing algorithm S': *Upon input signing key s and message α:*
   (a) *If α is the first message to be signed, thus denoted α_1, then S' runs $G(1^n)$ and obtains (s_1, v_1). Next, S' computes $\sigma_1 = S_s(\alpha_1, v_1)$ and outputs $\sigma'_1 = (\alpha_1, v_1, \sigma_1)$. (Note that the signature here contains the message as well; this is needed below.)*

   (b) *Let $(\alpha_1, \sigma'_1), \ldots, (\alpha_j, \sigma'_j)$ be the series of previously generated signatures, where each $\sigma'_j = (\alpha_j, v_j, \sigma_j)$. The current message is therefore the $(j+1)^{\text{th}}$ in the series and is denoted α_{j+1}. The signing algorithm S' first runs $G(1^n)$ and obtains (s_{j+1}, v_{j+1}). Then, S' computes $\sigma_{j+1} = S_{s_j}(\alpha_{j+1}, v_{j+1})$ and outputs $\sigma'_{j+1} = (\sigma'_j, \alpha_{j+1}, v_{j+1}, \sigma_{j+1})$.*[1]
3. Verification algorithm V': *Upon input a key v and a pair (α, σ'), algorithm V' first parses σ' into a series of triples $(\alpha_1, v_1, \sigma_1), \ldots, (\alpha_j, v_j, \sigma_j)$. Then, V' outputs 1 if and only if $V_v((\alpha_1, v_1), \sigma_1) = 1$ and for every $i = 2, \ldots, j$ it holds that $V_{v_{i-1}}((\alpha_i, v_i), \sigma_i) = 1$.*

[1] Note that $\sigma'_j = (\sigma'_{j-1}, \alpha_j, v_j, \sigma_j)$ and so recursively we obtain the entire chain back to σ'_1.

We note that this scheme is horribly inefficient. In particular, the size of the signature grows linearly with the number of messages signed. Nevertheless, its relative simplicity is useful for using it to demonstrate a feasibility result regarding memory-dependent signature schemes. We prove the following theorem:

Theorem 12.6 *Assume that (G, S, V) is a secure one-time signature scheme. Then, Construction 12.5 constitutes a secure memory-dependent signature scheme.*

Proof: Let $\mathcal{A}'$ be an adversary that generates a forgery for (G', S', V') with non-negligible probability. We construct a forger $\mathcal{A}$ for the one-time signature scheme (G, S, V) as follows. Let v be the input of $\mathcal{A}$, where v is in the range of $G(1^n)$. Let $t(n)$ be an upper-bound on the number of oracle queries made by $\mathcal{A}'$ (for example, take $t(n)$ to be the running time of $\mathcal{A}'$). Then, $\mathcal{A}$ chooses a random index $i \in_R \{0, \ldots, t(n)\}$ and works as follows. If $i = 0$, then $\mathcal{A}$ invokes $\mathcal{A}'$ with v as its public verification-key. Otherwise, $\mathcal{A}$ computes $(v_0, s_0) \leftarrow G(1^n)$ and invokes $\mathcal{A}'$ with v_0 as its public verification-key. Recall that by the construction, the key-pair (s_j, v_j) is used to sign on the $(j+1)^{\text{th}}$ message. The index i chosen randomly is the position in the chain that $\mathcal{A}$ places the key v that it received as input. Thus, if $i = 0$, adversary $\mathcal{A}$ places v as the public-key (since it is used to sign on the first message). For general i, the key v is placed together with the $(i-1)^{\text{th}}$ message (signed by s_{i-1}), and so the i^{th} message is to be signed by the signing key s associated with v. (Of course, $\mathcal{A}$ does not know s, but can use its oracle to generate this signature.) We differentiate between the first $i-1$ oracle queries, the i^{th} query, the $(i+1)^{\text{th}}$ query, and the remaining queries:

1. In order to answer the first $i-1$ oracle queries (if $i > 0$), $\mathcal{A}$ behaves exactly like the honest signer. It can do this because it chose (v_0, s_0).

2. In order to answer the i^{th} oracle query α_i, adversary $\mathcal{A}$ sets $v_i = v$ (where v is its input verification-key) and continues like the honest signer. That is, $\mathcal{A}$ computes $\sigma_i = S_{s_{i-1}}(\alpha_i, v)$ and outputs $\sigma'_i = (\sigma'_{i-1}, \alpha_i, v, \sigma_i)$. $\mathcal{A}$ knows s_{i-1} so this is no problem.

3. In order to answer the $(i+1)^{\text{th}}$ oracle query α_{i+1} (if $i < t(n)$), adversary $\mathcal{A}$ computes $(v_{i+1}, s_{i+1}) \leftarrow G(1^n)$ and queries its oracle with the message (α_{i+1}, v_{i+1}) obtaining σ_{i+1}.[2] Then $\mathcal{A}$ replies with $\sigma'_{i+1} = (\sigma'_i, \alpha_{i+1}, v_{i+1}, \sigma_{i+1})$.

4. In order to answer the remaining oracle queries, $\mathcal{A}'$ works in the same way as for the first $i-1$ queries. It can do this because it knows s_{i+1} and all subsequent signing keys.

We stress that the distribution of all the signatures generated by $\mathcal{A}$ is the same. The only difference is if the verification-key and signature is generated by $\mathcal{A}$, or obtained externally.

Let (α, σ') be the output of $\mathcal{A}'$ and assume that $V'(v, \alpha, \sigma') = 1$ and $\alpha \notin \mathcal{Q}$ (by the assumption, this event occurs with non-negligible probability). Parse σ' into triples $(\alpha_1, v_1, \sigma_1), \ldots, (\alpha_j, v_j, \sigma_j)$, where $\alpha_j = \alpha$. If $j = 1$, then it follows that $\mathcal{A}'$ output a pair (α, σ') such that $\sigma' = (\alpha, v_1, \sigma_1)$ and $V_v((\alpha, v_1), \sigma) = 1$. It follows that if $i = 0$ in the simulation by $\mathcal{A}$, then $\mathcal{A}'$ generated a forgery for verification-key $v_0 = v$ that constitutes a forgery for $\mathcal{A}$ (because v is its input verification-key). Since $i = 0$ with probability $1/(t(n) + 1)$, we have that $\mathcal{A}$ succeeds with non-negligible probability (note that if α is a forgery then it must be that $\mathcal{A}'$ never queried its signing oracle with α and so $\mathcal{A}$ did not query its signing oracle with (α, v_1) for any v_1). So far, we have dealt with the case that $j = 1$. In case $j > 1$, we have the following two cases:

Note that that $v_i = v$ and that $\mathcal{A}$'s oracle answers using the signing-key corresponding to v.

1. The keys $v_1, \ldots, v_j$ are exactly the same verification keys appearing in the signatures generated by $\mathcal{A}$ in the above game with $\mathcal{A}'$ (and they appear in the same order). Since $\alpha \notin \mathcal{Q}$, we have that $\mathcal{A}'$ has forged a signature with respect to v_{j-1} (note that the forgery is with respect to v_{j-1} and not v_j because v_j is the "new key" to be used for the next signature). Since $\mathcal{A}$ chose i randomly, it follows that $i = j-1$ with probability $1/(t(n)+1)$. Thus, $\mathcal{A}$ succeeds in forging a signature with respect to v with non-negligible probability. (Note that this argument relies on the fact that the view of $\mathcal{A}'$ is independent of $\mathcal{A}$'s choice of i.)

2. The keys $v_1, \ldots, v_j$ are not the same as those appearing in the sequence generated by $\mathcal{A}$. Let $l \geq 1$ be such that $v_1, \ldots, v_{l-1}$ are the same keys as generated by $\mathcal{A}$, and v_l is different. It follows then that $\mathcal{A}$ did not give $\mathcal{A}'$ a signature σ' containing a sub-signature σ on the message (α_l, v_l). Now, if $i = l-1$ (which occurs with probability $1/(t(n)+1)$), it follows that $\mathcal{A}'$ generated a forgery with respect to the verification-key $v_{l-1} = v_i = v$. Once again, this implies that $\mathcal{A}$ succeeds in generating a forgery with respect to v with non-negligible probability.

This completes the proof. ■

The above yields the following corollary:

Corollary 12.7 *If there exist one-time signature schemes, then there exist secure memory-dependent signature schemes.*

Note that since we need a one-time signature scheme that is capable of signing on a message as well as another key for the one-time signature scheme, our original construction based on one-way functions only cannot be used. Rather, we have to use the generalized scheme that uses collision-resistant hash functions.

12.4 Secure Memoryless Signature Schemes

We only sketch the main idea behind the construction. The first step in this construction is to change the memory-dependent scheme of the previous section. Specifically, instead of generating a linear *chain* of signatures, we construct a *tree* of signatures. The construction below is presented for the case of fixed-length messages. A full-fledged signature scheme can either be obtained by applying Theorem 11.4, or by making relatively small modifications to the construction.

The construction of the tree is as follows. We consider a full binary tree of depth n. Each node in the tree is labelled by a binary string $\tau_1, \ldots, \tau_i$ according to the following structure. The root is labelled the empty string λ. Let τ be the label of some internal node. Then, the left son of this node is labelled $\tau 0$ and the right son is labelled $\tau 1$. Each node is assigned a pair of keys from the underlying one-time signature scheme; the keys for a node labelled τ are denoted (s_τ, v_τ). The verification-key of the entire signature scheme is v_λ and the signing-key is s_λ. We note that the only keys that are chosen at the onset are s_λ and v_λ; the rest are generated on the fly during the signing process.

Now, in order to sign on a document $\alpha \in \{0,1\}^n$, we go to the node that has label α. We then choose key-pairs for all the nodes on the path from α to the root, and for the all the siblings of these nodes. Next, we use s_λ to sign on the verification-keys of both of its sons. Similarly, the signing-key of the son of λ that is on the path to α is used to sign on the verification-keys of both of its sons. This procedure continues until we reach α, at which time the message α itself is signed with the key s_α. The chain of these signatures constitutes the signature on α.

In order to maintain consistency, all of the chosen key-pairs must be recorded by the signing algorithm. For the next signature, only new nodes are assigned new key-pairs. This ensures that every signing-key is used to generate only one signature. The proof that the above constitutes a memory-dependent signature scheme uses similar ideas to our above construction, but is significantly more involved.

Before proceeding further, we remark that the above construction is already an improvement over that of Section 12.3 because the length of the signature does not grow over time.

Obtaining memoryless signature schemes. Unlike the linear-chain construction, the tree-based construction lends itself to a memoryless transformation. Specifically, we use a pseudorandom function F_k (where k is part of the signing key) and define the key-pair associated with a node τ to be the result of running $G(1^n)$ with random-tape $F_k(\mathsf{key}, \tau)$. Furthermore, the random-coins used by S to generate a signature with s_τ are set to be $F_k(\mathsf{sign}, \tau)$ (note, key and sign can be any distinct symbols). Notice that the scheme may now be memoryless because the key-pairs can be reproduced from scratch for each signature. Furthermore, because F_k is a fixed function, the key-pair for a node τ, and the coins used by S in computing a signature with s_τ, are always the same. Thus, we are guaranteed that each signing-key is only used to generate a single signature. We conclude that if a truly random function is used, the construction would be identical to above. Security when using a pseudorandom function follows using standard arguments. We obtain the following theorem:

Theorem 12.8 *If there exist secure one-time signature schemes, then there exist secure (memoryless) signature schemes.*

12.5 Removing the Need for Collision-Resistant Hash Functions

We have proven that the existence of one-time signature schemes (that are not length restricted) implies the existence of general memoryless secure signature schemes. However, our construction of a not length-restricted one-time signature scheme relied on the existence of collision-resistant hash functions (which seems to be a much stronger assumption than just one-way functions). We provide a very brief sketch showing how this assumption can be removed. Consider a weak type of hash function, called a universal one-way hash function which provides the following level of collision-resistance. An adversary $\mathcal{A}$ outputs a value x and is then given the index r of a hash function that is randomly sampled from the family. $\mathcal{A}$ is said to have succeeded if it then outputs y such that $h_r(x) = h_r(y)$.

Consider now the following one-time signature scheme that is based on a length-restricted one-time signature scheme: Upon input signing-key s and message α, first choose an index r for a hash function and compute $\sigma = S_s(r, h_r(\alpha))$ and define the signature to be (r, σ). The intuition behind the security of this construction is as follows. Let $\mathcal{A}'$ be an adversary that generates a successful forgery (α', σ') where $\sigma' = (r', S_s(r', h_{r'}(\alpha')))$, and let α denote the single message that it queried its signing oracle. Then, we have the following cases:

1. With non-negligible probability, the forgery contains r' that is different to the r that it received in the signature it received from its oracle: In this case, $\mathcal{A}'$ can be used to generate a forgery in the original one-time signature scheme (take $\mathcal{A}$ that invokes $\mathcal{A}'$ and simulates the game above – the forgery generated by $\mathcal{A}'$ is good for $\mathcal{A}$ because $\mathcal{A}$ never queried its oracle with (r', x) for any x).

2. With non-negligible probability, the forgery contains the same r that it received in the signature from its oracle, but $h_r(\alpha) \neq h_r(\alpha')$: In this case, once again $\mathcal{A}'$ can be used to obtain a forgery in the original scheme.

3. With non-negligible probability, the forgery contains the same r as that generated by $\mathcal{A}$ during the simulation game and $h_r(\alpha) = h_r(\alpha')$: In this case, $\mathcal{A}'$ can be used to find a collision in h. Specifically, invoke $\mathcal{A}'$ and take the message α to be x for the hash function experiment. Then, upon receiving r, give (r, σ) to $\mathcal{A}'$ where $\sigma = S_s(r, h_r(\alpha))$. If $\mathcal{A}'$ outputs a forgery with the same r, then we have found $\alpha' \neq \alpha$ such that $h_r(\alpha) = h_r(\alpha')$ in contradiction to the weaker notion of collision resistance described above.

The important point here is that universal one-way hash functions can be constructed from one-way functions alone. This therefore yields the desired result.

Lecture 13

Secure Multiparty Computation

In this lecture, we will present a very high-level overview of the secure multiparty computation. (The notes of this lecture have been cut-and-paste from other places, and so the style is a little different from the rest of the course.)

13.1 Motivation

Distributed computing considers the scenario where a number of distinct, yet connected, computing devices (or parties) wish to carry out a joint computation of some function. For example, these devices may be servers who hold a distributed database system, and the function to be computed may be a database update of some kind. The aim of *secure multi-party computation* is to enable parties to carry out such distributed computing tasks in a secure manner. Whereas distributed computing classically deals with questions of computing under the threat of machine crashes and other inadvertent faults, secure multi-party computation is concerned with the possibility of deliberately malicious behaviour by some adversarial entity. That is, it is assumed that a protocol execution may come under "attack" by an external entity, or even by a subset of the participating parties. The aim of this attack may be to learn private information or cause the result of the computation to be incorrect. Thus, two important requirements on any secure computation protocol are *privacy* and *correctness*. The privacy requirement states that nothing should be learned beyond what is absolutely necessary; more exactly, parties should learn their output and nothing else. The correctness requirement states that each party should receive its correct output. Therefore, the adversary must not be able to cause the result of the computation to deviate from the function that the parties had set out to compute.

The setting of secure multi-party computation encompasses tasks as simple as coin-tossing and broadcast, and as complex as electronic voting, electronic auctions, electronic cash schemes, contract signing, anonymous transactions, and private information retrieval schemes. Consider for a moment the tasks of voting and auctions. The privacy requirement for an election protocol ensures that no parties learn anything about the individual votes of other parties, and the correctness requirement ensures that no coalition of parties can influence the outcome of the election beyond just voting for their preferred candidate. Likewise, in an auction protocol, the privacy requirement ensures that only the winning bid is revealed (this may be desired), and the correctness requirement ensures that the highest bidder is indeed the party to win (and so the auctioneer, or any other party, cannot bias the outcome).

Due to its generality, the setting of secure multi-party computation can model almost every,

[*] Lecture notes for a graduate course in the theory of cryptography. Yehuda Lindell, Bar-Ilan University, Israel, 2005.

if not every, cryptographic problem (including the classic tasks of encryption and authentication). Therefore, questions of feasibility and infeasibility for secure multi-party computation are fundamental to the theory and practice of cryptography.

Security in multi-party computation. As we have mentioned above, the model that we consider is one where an adversarial entity controls some subset of the parties and wishes to attack the protocol execution. The parties under the control of the adversary are called corrupted, and follow the adversary's instructions. Secure protocols should withstand any adversarial attack (where the exact power of the adversary will be discussed later). In order to formally claim and prove that a protocol is secure, a precise definition of security for multi-party computation is required. A number of different definitions have been proposed and these definitions aim to ensure a number of important security properties that are general enough to capture most (if not all) multi-party computation tasks. We now describe the most central of these properties:

- *Privacy:* No party should learn anything more than its prescribed output. In particular, the only information that should be learned about other parties' inputs is what can be derived from the output itself. For example, in an auction where the only bid revealed is that of the highest bidder, it is clearly possible to derive that all other bids were lower than the winning bid. However, this should be the only information revealed about the losing bids.

- *Correctness:* Each party is guaranteed that the output that it receives is correct. To continue with the example of an auction, this implies that the party with the highest bid is guaranteed to win, and no party including the auctioneer can influence this.

- *Independence of Inputs:* Corrupted parties must choose their inputs independently of the honest parties' inputs. This property is crucial in a sealed auction, where bids are kept secret and parties must fix their bids independently of others. We note that independence of inputs is *not* implied by privacy. For example, it may be possible to generate a higher bid, without knowing the value of the original one. Such an attack can actually be carried out on some encryption schemes (i.e., given an encryption of \$100, it is possible to generate a valid encryption of \$101, without knowing the original encrypted value).

- *Guaranteed Output Delivery:* Corrupted parties should not be able to prevent honest parties from receiving their output. In other words, the adversary should not be able to disrupt the computation by carrying out a "denial of service" attack.

- *Fairness:* Corrupted parties should receive their outputs if and only if the honest parties also receive their outputs. The scenario where a corrupted party obtains output and an honest party does not should not be allowed to occur. This property can be crucial, for example, in the case of contract signing. Specifically, it would be very problematic if the corrupted party received the signed contract and the honest party did not.

We stress that the above list does *not* constitute a definition of security, but rather a set of requirements that should hold for any secure protocol. Indeed, one possible approach to defining security is to just generate a list of separate requirements (as above) and then say that a protocol is secure if all of these requirements are fulfilled. However, this approach is not satisfactory for the following reasons. First, it may be possible that an important requirement was missed. This is especially true because different applications have different requirements, and we would like a definition that is general enough to capture all applications. Second, the definition should be simple enough so that it is trivial to see that *all* possible adversarial attacks are prevented by the proposed definition.

The standard definition today therefore formalizes security in the following general way. As a mental experiment, consider an "ideal world" in which an external trusted (and incorruptible) party is willing to help the parties carry out their computation. In such a world, the parties can simply send their inputs to the trusted party, who then computes the desired function and passes each party its prescribed output. Since the only action carried out by a party is that of sending its input to the trusted party, the only freedom given to the adversary is in choosing the corrupted parties' inputs. Notice that all of the above-described security properties (and more) hold in this ideal computation. For example, privacy holds because the only message ever received by a party is its output (and so it cannot learn any more than this). Likewise, correctness holds since the trusted party cannot be corrupted and so will always compute the function correctly.

Of course, in the "real world", there is no external party that can be trusted by all parties. Rather, the parties run some protocol amongst themselves without any help. Despite this, a secure protocol should emulate the so-called "ideal world". That is, a real protocol that is run by the parties (in a world where no trusted party exists) is said to be secure, if no adversary can do more harm in a real execution that in an execution that takes place in the ideal world. This can be formulated by saying that for any adversary carrying out a successful attack in the real world, there exists an adversary that successfully carries out the same attack in the ideal world. However, successful adversarial attacks *cannot* be carried out in the ideal world. We therefore conclude that all adversarial attacks on protocol executions in the real world must also fail.

More formally, the security of a protocol is established by comparing the outcome of a real protocol execution to the outcome of an ideal computation. That is, for any adversary attacking a real protocol execution, there exists an adversary attacking an ideal execution (with a trusted party) such that the input/output distributions of the adversary and the participating parties in the real and ideal executions are essentially the same. Thus a real protocol execution "emulates" the ideal world. This formulation of security is called the ideal/real simulation paradigm. In order to motivate the usefulness of this definition, we describe why all the properties described above are implied. Privacy follows from the fact that the adversary's output is the same in the real and ideal executions. Since the adversary learns nothing beyond the corrupted party's outputs in an ideal execution, the same must be true for a real execution. Correctness follows from the fact that the honest parties' outputs are the same in the real and ideal executions, and from the fact that in an ideal execution, the honest parties all receive correct outputs as computed by the trusted party. Regarding independence of inputs, notice that in an ideal execution, all inputs are sent to the trusted party before any output is received. Therefore, the corrupted parties know nothing of the honest parties' inputs at the time that they send their inputs. In other words, the corrupted parties' inputs are chosen independently of the honest parties' inputs, as required. Finally, guaranteed output delivery and fairness hold in the ideal world because the trusted party always returns all outputs. The fact that it also holds in the real world again follows from the fact that the honest parties' outputs are the same in the real and ideal executions.

We remark that the above informal definition is actually "overly ideal" and needs to be relaxed in settings where the adversary controls a half or more of the participating parties (that is, in the case that there is no honest majority). When this number of parties is corrupted, it is known that it is *impossible* to obtain general protocols for secure multi-party computation that guarantee output delivery and fairness. Therefore, the definition is relaxed and the adversary is allowed to abort the computation (i.e., cause it to halt before termination), meaning that "guaranteed output delivery" is not fulfilled. Furthermore, the adversary can cause this abort to take place after it has already obtained its output, but before all the honest parties receive their outputs. Thus "fairness" is not achieved. Loosely speaking, the relaxed definition is obtained by modifying the ideal execution and giving the adversary the additional capability of instructing the trusted party to not send outputs to some of the honest parties. Otherwise, the definition remains identical and thus all the other properties are still preserved.

Adversarial power. The above informal definition of security omits one very important issue: the power of the adversary that attacks a protocol execution. As we have mentioned, the adversary controls a subset of the participating parties in the protocol. However, we have not described the corruption strategy (i.e., when or how parties come under the "control" of the adversary), the allowed adversarial behaviour (i.e., does the adversary just passively gather information or can it instruct the corrupted parties to act maliciously), and what complexity the adversary is assumed to be (i.e., is it polynomial-time or computationally unbounded). We now describe the main types of adversaries that have been considered:

1. **Corruption strategy:** The corruption strategy deals with the question of when and how parties are corrupted. There are two main models:
    (a) Static corruption model: In this model, the adversary is given a fixed set of parties whom it controls. Honest parties remain honest throughout and corrupted parties remain corrupted.

    (b) Adaptive corruption model: Rather than having a fixed set of corrupted parties, adaptive adversaries are given the capability of corrupting parties during the computation. The choice of who to corrupt, and when, can be arbitrarily decided by the adversary and may depend on its view of the execution (for this reason it is called adaptive). This strategy models the threat of an external "hacker" breaking into a machine during an execution. We note that in this model, once a party is corrupted, it remains corrupted from that point on.

2. **Allowed adversarial behaviour:** Another parameter that must be defined relates to the actions that corrupted parties are allowed to take. Once again, there are two main types of adversaries:
    (a) Semi-honest adversaries: In the semi-honest adversarial model, even corrupted parties correctly follow the protocol specification. However, the adversary obtains the internal state of all the corrupted parties (including the transcript of all the messages received), and attempts to use this to learn information that should remain private. This is a rather weak adversarial model. However, there are some settings where it can realistically model the threats to the system. Semi-honest adversaries are also called "honest-but-curious" and "passive".

    (b) Malicious adversaries: In this adversarial model, the corrupted parties can *arbitrarily* deviate from the protocol specification, according to the adversary's instructions. In general, providing security in the presence of malicious adversaries is preferred, as it ensures that no adversarial attack can succeed. Malicious adversaries are also called "active".

3. **Complexity:** Finally, we consider the assumed computational complexity of the adversary. As above, there are two categories here:
    (a) Polynomial-time: The adversary is allowed to run in (probabilistic) polynomial-time (and sometimes, expected polynomial-time). The specific computational model used differs,

depending on whether the adversary is uniform (in which case, it is a probabilistic polynomial-time Turing machine) or non-uniform (in which case, it is modelled by a polynomial-size family of circuits).

(b) Computationally unbounded: In this model, the adversary has no computational limits whatsoever.

The above distinction regarding the complexity of the adversary yields two very different models for secure computation: the *information-theoretic* model and the *computational* model. In the information-theoretic setting, the adversary is not bound to any complexity class (and in particular, is not assumed to run in polynomial-time). Therefore, results in this model hold unconditionally and do not rely on any complexity or cryptographic assumptions. The only assumption used is that parties are connected via ideally *private* channels (i.e., it is assumed that the adversary cannot eavesdrop or interfere with the communication between honest parties).

In contrast, in the computational setting the adversary is assumed to be polynomial-time. Results in this model typically assume cryptographic assumptions like the existence of trapdoor permutations. We note that it is not necessary here to assume that the parties have access to ideally private channels, because such channels can be implemented using public-key encryption. However, it is assumed that the communication channels between parties are authenticated; that is, if two honest parties communicate, then the adversary can eavesdrop but cannot modify any message that is sent. Such authentication can be achieved using digital signatures and a public-key infrastructure.

We remark that all possible combinations of the above types of adversaries have been considered in the literature.

13.2 Definition of Security

In this section we present the definition for secure two-party computation. We present the two-party, and not multiparty, case for the sake of simplicity.

Two-party computation. A two-party protocol problem is cast by specifying a random process that maps pairs of inputs to pairs of outputs (one for each party). We refer to such a process as a functionality and denote it $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$, where $f = (f_1, f_2)$. That is, for every pair of inputs (x, y), the output-pair is a random variable $(f_1(x, y), f_2(x, y))$ ranging over pairs of strings. The first party (with input x) wishes to obtain $f_1(x, y)$ and the second party (with input y) wishes to obtain $f_2(x, y)$. We often denote such a functionality by $(x, y) \mapsto (f_1(x, y), f_2(x, y))$. Thus, for example, the basic coin-tossing functionality is denoted by $(1^n, 1^n) \mapsto (U_n, U_n)$.

Adversarial behavior. Loosely speaking, the aim of a secure two-party protocol is to protect an honest party against dishonest behavior by the other party. The definition we present here considers the case of a probabilistic polynomial-time *malicious* adversary with *static* corruptions. When considering malicious adversaries, there are certain undesirable actions that cannot be prevented. Specifically, a party may refuse to participate in the protocol, may substitute its local input (and enter with a different input) and may abort the protocol prematurely. One ramification of the adversary's ability to abort, is that it is impossible to achieve "fairness". That is, the adversary may obtain its output while the honest party does not.

Execution in the ideal model. We now describe the ideal model for malicious adversaries. As we have mentioned, the ability of the adversary to abort early is built into the ideal model since this cannot be prevented. An ideal execution proceeds as follows:

Inputs: Each party obtains an input, denoted z.

Send inputs to trusted party: An honest party always sends z to the trusted party. A malicious party may, depending on z, either abort or sends some $z' \in \{0,1\}^{|z|}$ to the trusted party.

Trusted party answers first party: In case it has obtained an input pair, (x, y), the trusted party (for computing f), first replies to the first party with $f_1(x, y)$. Otherwise (i.e., in case it receives only one valid input), the trusted party replies to both parties with a special symbol $\perp$.

Trusted party answers second party: In case the first party is malicious it may, depending on its input and the trusted party's answer, decide to *stop* the trusted party. In this case the trusted party sends $\perp$ to the second party. Otherwise (i.e., if not stopped), the trusted party sends $f_2(x, y)$ to the second party.

Outputs: An honest party always outputs the message it has obtained from the trusted party. A malicious party may output an arbitrary (probabilistic polynomial-time computable) function of its initial input and the message obtained from the trusted party.

Let $f : \{0,1\}^* \times \{0,1\}^* \mapsto \{0,1\}^* \times \{0,1\}^*$ be a functionality, where $f = (f_1, f_2)$, and let $\overline{M} = (M_1, M_2)$ be a pair of non-uniform probabilistic polynomial-time machines (representing parties in the ideal model). Such a pair is admissible if for at least one $i \in \{1, 2\}$ we have that M_i is honest (i.e., follows the honest party instructions in the above-described ideal execution). Then, the joint execution of f under $\overline{M}$ in the ideal model (on input pair (x, y)), denoted $\text{IDEAL}_{f,\overline{M}}(x, y)$, is defined as the output pair of M_1 and M_2 from the above ideal execution. For example, in the case that M_1 is malicious and always aborts at the outset, the joint execution is defined as $(M_1(x, \perp), \perp)$. Whereas, in case M_1 never aborts, the joint execution is defined as $(M_1(x, f_1(x', y)), f_2(x', y))$ where $x' = M_1(x)$ is the input that M_1 gives to the trusted party.

Execution in the real model. We next consider the real model in which a real (two-party) protocol is executed (and there exists no trusted third party). In this case, a malicious party may follow an arbitrary feasible strategy; that is, any strategy implementable by non-uniform expected polynomial-time machines.

Let f be as above and let Π be a two-party protocol for computing f. Furthermore, let $\overline{M} = (M_1, M_2)$ be a pair of non-uniform probabilistic polynomial-time machines (representing parties in the real model). Such a pair is admissible if for at least one $i \in \{1, 2\}$ we have that M_i is honest (i.e., follows the strategy specified by Π). Then, the joint execution of Π under $\overline{M}$ in the real model (on input pair (x, y)), denoted $\text{REAL}_{\Pi,\overline{M}}(x, y)$, is defined as the output pair of M_1 and M_2 resulting from the protocol interaction.

Security as emulation of a real execution in the ideal model. Having defined the ideal and real models, we can now define security of protocols. Loosely speaking, the definition asserts that a secure two-party protocol (in the real model) emulates the ideal model (in which a trusted party exists). This is formulated by saying that admissible pairs in the ideal model are able to simulate admissible pairs in an execution of a secure real-model protocol.

Definition 13.1 (security in the malicious model): *Let f and Π be as above. Protocol Π is said to* securely compute f (in the malicious model) *if for every pair of admissible non-uniform probabilistic polynomial-time machines $\overline{A} = (A_1, A_2)$ for the real model, there exists a pair of admissible non-uniform probabilistic polynomial-time machines $\overline{B} = (B_1, B_2)$ for the ideal model, such that*

$$\left\{\text{IDEAL}_{f,\overline{B}}(x,y)\right\}_{x,y \text{ s.t. } |x|=|y|} \overset{\text{c}}{\equiv} \left\{\text{REAL}_{\Pi,\overline{A}}(x,y)\right\}_{x,y \text{ s.t. } |x|=|y|}$$

We note that the above definition assumes that the parties know the input lengths (this can be seen from the requirement that $|x| = |y|$). Some restriction on the input lengths is unavoidable, see [6, Section 7.1] for discussion.

13.3 Oblivious Transfer

Oblivious transfer was introduced by Rabin [15] and is a central tool in constructions of secure protocols. The variant of oblivious transfer that we will see here is called 1-out-of-2 oblivious transfer, and was introduced by [4]. The 1-out-of-2 oblivious transfer functionality is defined by $((x_0, x_1), \sigma) \mapsto (\lambda, x_\sigma)$ where λ denotes the empty string. We will briefly describe the oblivious transfer protocol of [4], that is secure in the presence of semi-honest adversaries. Our description will be for the case that $x_0, x_1 \in \{0, 1\}$; when considering semi-honest adversaries, the general case can be obtained by running the single-bit protocol many times in parallel.

Protocol 13.2 (oblivious transfer [4]):

- **Inputs:** *P_1 has $x_0, x_1 \in \{0,1\}$ and P_2 has $\sigma \in \{0,1\}$.*
- **The protocol:**
  1. *P_1 randomly chooses a permutation-trapdoor pair (f, t) from a family of enhanced trapdoor permutations.*[1] *P_1 sends f (but not the trapdoor t) to P_2.*
  2. *P_2 chooses a random v_σ in the domain of f and computes $w_\sigma = f(v_\sigma)$. In addition, P_2 chooses a random $w_{1-\sigma}$ in the domain of f, using the "enhanced" sampling algorithm (see Footnote 1). P_2 sends (w_0, w_1) to P_1.*
  3. *P_1 uses the trapdoor t and computes $v_0 = f^{-1}(w_0)$ and $v_1 = f^{-1}(w_1)$. Then, it computes $b_0 = B(v_0) \oplus x_0$ and $b_1 = B(v_1) \oplus x_1$, where B is a hard-core bit of f. Finally, P_1 sends (b_0, b_1) to P_2.*
  4. *P_1 computes $x_\sigma = B(v_\sigma) \oplus b_\sigma$ and outputs x_σ.*

We have the following theorem:

Theorem 13.3 *Assuming that (f, t) are chosen from a family of enhanced trapdoor permutations, Protocol 13.2 securely computes the 1-out-of-2 oblivious transfer functionality in the presence of static semi-honest adversaries.*

[1] Informally speaking, an enhanced trapdoor permutation has the property that it is possible to sample from the range, so that given the coins used for sampling it is still hard to invert the value. See [6, Appendix C.1] for more details.

Proof Sketch: We present a very brief proof sketch here only. Consider first the case that P_1 is corrupted. In this case, we construct a simulator S_1 (given no output) that internally invokes P_1 and receives f as sent by P_1 to P_2. Simulator S_1 then hands P_1 two random values w_0 and w_1 in the domain of f. Simulator S_1 then obtains P_1's reply and outputs whatever P_1 outputs. Notice that the view of P_1 in this simulation is identical to its view in a real execution because the distribution of a randomly chosen w_0 is equivalent to the distribution generated by first choosing a random v_0 and then computing $w_0 = f(v_0)$. We therefore conclude that S_1's output is distributed identically to P_1's output in a real execution.

Next, consider the case that P_2 is corrupted. In this case, we construct a simulator S_2 who receives a value x_σ as output and works as follows. S_2 chooses (f, t), internally invokes P_2 and hands f to P_2 as if it was sent from P_1. Then, after receiving (w_0, w_1) from P_2, simulator S_2 computes $b_0 = B(v_0) \oplus x_\sigma$ and $b_1 = B(v_1) \oplus x_\sigma$, and hands (b_0, b_1) to P_2. Finally, S_2 outputs whatever P_2 outputs. We claim that P_2's view in this simulation is computationally indistinguishable from its view in a real execution (and thus S_2's output in an ideal execution is indistinguishable from P_2's output in a real execution). In order to see this, recall that P_2 is semi-honest and so it only knows one of the preimages of (w_0, w_1); namely it only knows w_σ. By the hard-core property of B, it follows that $B(v_{1-\sigma})$ is indistinguishable from U_1. Therefore, the fact that S_2 handed P_2 the values $b_0 = B(v_0) \oplus x_\sigma$ and $b_1 = B(v_1) \oplus x_\sigma$, rather than the values $b_0 = B(v_0) \oplus x_0$ and $b_1 = B(v_1) \oplus x_1$, cannot be detected in polynomial-time. This completes the proof sketch. ■

Extensions. We note that Protocol 13.2 can be easily extended to 1-out-of-k oblivious transfer by having the receiver choose k values but where it still only knows a single preimage.

13.4 Constructions of Secure Protocols

Goldreich, Micali and Wigderson [10] showed that assuming the existence of (enhanced) trapdoor permutations, there are secure protocols (in the malicious model) for any multi-party functionality. Their methodology works by first presenting a protocol secure against semi-honest adversaries. Next, a *compiler* is applied that transforms *any* protocol secure against semi-honest adversaries into a protocol secure against malicious adversaries. In this section, we describe the construction of [10] for the case of semi-honest adversaries, and their compiler for transforming it into a protocol that is secure in the presence of malicious adversaries.

13.4.1 Security Against Semi-Honest Adversaries

Recall that in the case of semi-honest adversaries, even the corrupted parties follow the protocol specification. However, the adversary may attempt to learn more information than intended by examining the transcript of messages that it received during the protocol execution. Despite the seemingly weak nature of the adversarial model, obtaining protocols secure against semi-honest adversaries is a non-trivial task.

We now briefly describe the construction of [10] for secure two-party computation in the semi-honest adversarial model. Let f be the two-party functionality that is to be securely computed. Then, the parties are given an arithmetic circuit over $GF(2)$ that computes the function f. The protocol starts with the parties sharing their inputs with each other using simple bitwise-xor secret sharing, and thus following this stage, they both hold shares of the input lines of the circuit. That is, for each input line l, party A holds a value a_l and party B holds a value b_l, such that both a_l and b_l are random under the constraint that $a_l + b_l$ equals the value of the input into this line.

Next, the parties evaluate the circuit gate-by-gate, computing random shares of the output line of the gate from the random shares of the input lines to the gate. There are two types of gates in the circuit: addition gates and multiplication gates:

- Addition gates are evaluated by each party locally adding its shares of the input values. Note that this is fine because if A holds a_l and $a_{l'}$ and B holds b_l and $b_{l'}$, then $(a_l + a_{l'}) + (b_l + b_{l'}) = (a_l + b_l) + (a_{l'} + b_{l'})$ where the latter is exactly what the output wire of the addition gate should hold. Furthermore, neither party learns anything more than it knew already, because no information is transferred in evaluating this gate.

- Multiplication gates are evaluated using 1-out-of-4 oblivious transfer. Here the parties wish to compute random shares c_1 and c_2 such that $c_1 + c_2 = a \cdot b = (a_l + b_l)(a_{l'} + b_{l'})$. For this purpose, A chooses a random bit $\sigma \in_R \{0,1\}$, sets its share c_1 of the output line of the gate to σ, and defines the following table:

Value of $(b_l, b_{l'})$	Receiver input i	Receiver output c_2
(0,0)	1	$o_1 = \sigma + (a_l + 0) \cdot (a_{l'} + 0)$
(0,1)	2	$o_2 = \sigma + (a_l + 0) \cdot (a_{l'} + 1)$
(1,0)	3	$o_3 = \sigma + (a_l + 1) \cdot (a_{l'} + 0)$
(1,1)	4	$o_4 = \sigma + (a_l + 1) \cdot (a_{l'} + 1)$

  Having prepared this table, A and B use the 1-out-of-4 oblivious transfer functionality. Party A plays the sender and inputs the values (o_1, o_2, o_3, o_4) defined above, and party B plays the receiver and sets its input i appropriately as defined in the above table (e.g., for $b_l = 1$ and $b_{l'} = 0$, party B sets $i = 3$). Upon receiving its output o from the oblivious transfer, P_2 sets $c_2 = o$ to be its share of the output line of the gate. Notice that $c_1 + c_2 = (a_l + b_l)(a_{l'} + b_{l'})$ and the parties hold random shares of the output line of the gate.

In the above way, the parties jointly compute the circuit and obtain shares of the output gates. The protocol concludes with each party revealing the prescribed shares of the output gates to the other party (i.e, if a certain output gate provides a bit of A's input, then B will reveal its share of this output line to A).

13.4.2 The GMW Compiler

The GMW compiler takes for input a protocol secure against semi-honest adversaries; from here on we refer to this as the "basic protocol". Recall that this protocol is secure in the case that each party follows the protocol specification exactly, using its input and uniformly chosen random tape. Thus, in order to obtain a protocol secure against malicious adversaries, we need to enforce potentially malicious parties to behave in a semi-honest manner. First and foremost, this involves forcing the parties to follow the prescribed protocol. However, this only makes sense relative to a *given* input and random tape. Furthermore, a malicious party must be forced into using a *uniformly chosen* random tape. This is because the security of the basic protocol may depend on the fact that the party has no freedom in setting its own randomness.

An informal description of the GMW compiler. In light of the above discussion, the compiler begins by having each party commit to its input. Next, the parties run a coin-tossing protocol in order to fix their random tapes (clearly, this protocol must be secure against malicious adversaries). A regular coin-tossing protocol in which both parties receive the same uniformly distributed string is not sufficient here. This is because the parties' random tapes must remain secret. This is solved by augmenting the coin-tossing protocol so that one party receives a uniformly distributed string (to be used as its random tape) and the other party receives a commitment to that string. Now, following these two steps, each party holds its own uniformly distributed random-tape and a commitment to the other party's input and random-tape. Therefore, each party can be "forced" into working consistently with the committed input and random-tape.

We now describe how this behavior is enforced. A protocol specification is a deterministic function of a party's view consisting of its input, random tape and messages received so far. As we have seen, each party holds a commitment to the input and random tape of the other party. Furthermore, the messages sent so far are public. Therefore, the assertion that a new message is computed according to the protocol is of the $\mathcal{NP}$ type (and the party sending the message knows an adequate NP-witness to it). Thus, the parties can use zero-knowledge proofs to show that their steps are indeed according to the protocol specification. As the proofs used are zero-knowledge, they reveal nothing. On the other hand, due to the soundness of the proofs, even a malicious adversary cannot deviate from the protocol specification without being detected. We thus obtain a reduction of the security in the malicious case to the given security of the basic protocol against semi-honest adversaries.

In summary, the components of the compiler are as follows (where "secure" refers to security against malicious adversaries):

1. **Input Commitment:** In this phase the parties execute a secure protocol for the following functionality:
$$((x,r),1^n) \mapsto (\lambda, C(x;r))$$
where x is the party's input string (and r is the randomness chosen by the committing party).

   A secure protocol for this functionality involves the committing party sending $C(x;r)$ to the other party followed by a zero-knowledge proof of knowledge of (x,r). Informally, this functionality ensures that the committing party "knows" the value being committed to.

2. **Coin Generation:** The parties generate t-bit long random tapes (and corresponding commitments) by executing a secure protocol in which one party receives a commitment to a uniform string of length t and the other party receives the string itself (to be used as its random tape) and the decommitment (to be used later for proving "proper behavior"). That is, the parties compute the functionality:
$$(1^n,1^n) \mapsto ((U_t, U_{t\cdot n}), C(U_t; U_{t\cdot n}))$$
(where we assume that to commit to a t-bit string, C requires $t \cdot n$ random bits).

3. **Protocol Emulation:** In this phase, the parties run the basic protocol whilst proving (in zero-knowledge) that their steps are consistent with their input string, random tape and prior messages received.

Since a malicious party running in the "compiled protocol" must prove that every message that it sends is according to the protocol specification, it has only two strategies: it can either behave semi-honestly, or it can abort. (Note that cheating in a zero-knowledge proof is also considered aborting, since the adversary will be caught except with negligible probability.) The security of the basic protocol in the presence of semi-honest adversaries thus implies that the compiled protocol is secure even in the presence of malicious adversaries.

Bibliography

[1] L. Babai and S. Moran. Arthur-Merlin Games: a Randomized Proof System and a Hierarchy of Complexity Classes. In the *Journal of Computer and System Sciences,* 36(2):254–276, 1988.

[2] M. Blum. Coin Flipping by Phone. *IEEE Spring COMPCOM*, pages 133–137, 1982.

[3] M. Blum and S. Micali. How to Generate Cryptographically Strong Sequences of Pseudo-Random Bits. *SIAM Journal on Computing*, 13(4):850–864, 1984.

[4] S. Even, O. Goldreich and A. Lempel. A Randomized Protocol for Signing Contracts. In *Communications of the ACM,* 28(6):637–647, 1985.

[5] O. Goldreich. *Foundations of Cryptography: Volume 1 – Basic Tools.* Cambridge University Press, 2001.

[6] O. Goldreich. *Foundations of Cryptography: Volume 2 – Basic Applications.* Cambridge University Press, 2004.

[7] O. Goldreich, S. Goldwasser and S. Micali. How to Construct Random Functions. *Journal of the ACM*, 33(4):792–807, 1986.

[8] O. Goldreich and L.A. Levin. Hard-Core Predicates for Any One-Way Function. In *21st STOC*, pages 25–32, 1989.

[9] O. Goldreich, S. Micali and A. Wigderson. Proofs that Yield Nothing but their Validity or All Languages in NP Have Zero-Knowledge Proof Systems. *Journal of the ACM*, 38(1):691–729, 1991.

[10] O. Goldreich, S. Micali and A. Wigderson. How to Play any Mental Game – A Completeness Theorem for Protocols with Honest Majority. In 19*th STOC,* pages 218–229, 1987. For details see [6, Chapter 7].

[11] S. Goldwasser and S. Micali. Probabilistic Encryption. *JCSS*, 28(2):270–299, 1984.

[12] S. Goldwasser, S. Micali and C. Rackoff The Knowledge Complexity of Interactive Proof Systems. *SIAM Journal on Computing,* 18(1):186–208, 1989.

[13] J. Håstad, R. Impagliazzo, L.A. Levin and M. Luby. A Pseudorandom Generator from any One-way Function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999.

[14] M. Naor. Bit Commitment using Pseudorandom Generators. *Journal of Cryptology*, 4(2):151–158, 1991.

[15] M. Rabin. How to Exchange Secrets by Oblivious Transfer. Tech. Memo TR-81, Aiken Computation Laboratory, Harvard U., 1981.

[16] S. Rudich. *Limits on the Provable Consequences of One-way Functions.* Ph.D. thesis, UC Berkeley, 1988.

[17] A.C. Yao. Theory and Application of Trapdoor Functions. In 23*rd FOCS*, pages 80–91, 1982.

[18] A. Yao. How to Generate and Exchange Secrets. In 27*th FOCS*, pages 162–167, 1986.