CS276: Graduate Cryptography[1]

This draft is continually being updated.

Sanjam Garg
University of California, Berkeley

September 11, 2018

[1]Based on scribe notes by students from taking CS276 in fall 2014. Also, thanks to Peihan Miao and Akshayaram Srinivasan for helping improve these notes.

Contents

1 One-Way Functions **7**
- 1.1 Noticeable and Negligible Functions . . . 7
- 1.2 Probabilistic Polynomial Time . . . 8
- 1.3 One-Way Functions . . . 9
- 1.4 Composability of One-Way Functions . . . 10
- 1.5 Examples . . . 10
- 1.6 Hardness Amplification . . . 12
- 1.7 Levin's One-Way Function . . . 14

2 Pseudorandomness **17**
- 2.1 Hard Core Bit . . . 17
- 2.2 Hard Core Bit of any One-Way Functions . . . 18
- 2.3 Computational Indistinguishability . . . 24
- 2.4 Pseudorandom Generators . . . 25
  - 2.4.1 PRG Extension . . . 25
  - 2.4.2 PRG from OWP (One-Way Permutations) . . . 26
- 2.5 Pseudorandom Functions . . . 28
  - 2.5.1 Definitions . . . 28
  - 2.5.2 Construction of PRF from PRG . . . 28

3 Digital Signatures **31**
- 3.1 Definition . . . 31
- 3.2 Digital Signature for q = 1 . . . 32
- 3.3 Collision Resistant Hash Functions . . . 32
  - 3.3.1 Definition of a family of CRHF . . . 32
  - 3.3.2 Discrete log CRHF . . . 33
- 3.4 Multiple-Message Digital Signature . . . 33

4 Public Key Encryption **35**
- 4.1 Correctness . . . 35
- 4.2 Indistinguishability and Semantic Security . . . 36
  - 4.2.1 Indistinguishability Security . . . 36
  - 4.2.2 Semantic Security . . . 36
  - 4.2.3 Equivalence of Definitions . . . 37
- 4.3 Public Key Encryption from Trap-Door OWP . . . 37
- 4.4 Indistinguishability in a Chosen Plaintext Attack . . . 37
- 4.5 Chosen Ciphertext Attack for Public Key Encryption . . . 38

4.6 One Time Secure Signature Scheme . . . 42
4.7 CCA2 . . . 43
4.7.1 Model . . . 43
4.7.2 Security . . . 44
4.7.3 Use and Efficiency . . . 47

5 Bilinear Maps **49**
5.1 Diffie-Hellman Key Exchange . . . 49
5.1.1 Discussion 1 . . . 49
5.1.2 Discussion 2 . . . 50
5.2 Bilinear Maps . . . 50
5.2.1 Discussion 1 . . . 50
5.3 Tripartite Diffie-Hellman . . . 51
5.4 IBE: Identity-Based Encryption . . . 51
5.4.1 Security Descriptions . . . 52
5.4.2 Discussion 1 . . . 53

6 Zero-Knowledge Proofs **57**
6.1 Interactive Proofs . . . 57
6.2 Zero Knowledge Proofs . . . 58
6.3 Graph Isomorphism . . . 63
6.4 Zero-Knowledge for NP . . . 63
6.4.1 Commitment Schemes . . . 64
6.4.2 3COL Protocol . . . 65

7 Secure Computation **69**
7.1 Introduction . . . 69
7.2 Real/Ideal Paradigm . . . 69
7.3 Oblivious transfer . . . 71
7.3.1 1-out-of-2 oblivious transfer . . . 71
7.3.2 1-out-of-4 oblivious transfer . . . 72
7.4 Yao's Two Party Computation Protocol . . . 73
7.4.1 Construction: . . . 73
7.4.2 Proof of Security . . . 74
7.5 GMW Protocol . . . 75
7.6 Malicious attacker intead of semi-honest attacker . . . 75
7.6.1 Zero-knowledge proof of knowledge (ZK-PoK) . . . 76

8 Secure RAM Computation **79**
8.1 Oblivious RAM . . . 79
8.2 Introduction to Secure RAM Computation . . . 81
8.2.1 Secret-Shared Database Construction . . . 81
8.2.2 Encrypted Database Construction . . . 81
8.2.3 Fixed Construction . . . 82

9 Witness Encryption **85**
9.1 A Story . . . 85
9.2 A Simple Language . . . 86
9.3 An NP Complete Language . . . 87
9.3.1 Exact Cover . . . 87
9.3.2 Multilinear Maps . . . 87
9.3.3 The n-MDDH Assumption . . . 87
9.3.4 Decisional Multilinear No-Exact-Cover Assumption . . . 88
9.3.5 The Encryption Scheme . . . 88

10 Obfuscation **89**
10.1 VBB Obfuscation . . . 89
10.2 Indistinguishability Obfuscation . . . 91
10.3 $i\mathcal{O}$ for Polynomial-sized Circuits . . . 92
10.3.1 Construction . . . 92
10.3.2 Proof of Security . . . 93
10.4 Identity-Based Encryption . . . 93
10.5 Digital Signature Scheme via Indistinguishable Obfuscation . . . 94
10.6 Public Key Encryption via Indistinguishable Obfuscation . . . 95
10.7 Indistinguishable Obfuscation Construction from NC^1 iO . . . 96

Chapter 1

One-Way Functions

Cryptography enables many paradoxical objects such as public key encryption, verifiable electronic signatures, zero-knowledge protocols and fully homomorphic encryption. The two main steps in the development of such seemingly impossible primitives are: (i) defining the desired security properties formally, and (ii) obtaining a construction satisfying the security property provably. The second step assumes (unproven) computational assumptions which are believed to be hard. In this course we will define several cryptographic primitives and argue their security based on well-defined computational hardness assumption. However, we will largely ignore the mathematics underlying the assumed computational intractability assumptions.

The weakest computational hardness assumptions (and also the simplest) of interest in cryptography is the existence of one-way functions. We study this objects in this chapter. In order to define one-way functions we need new formal vocabulary that we describe in Sections 1.1 and 1.2. Finally, we define one way functions in Section 1.3.

1.1 Noticeable and Negligible Functions

Noticeable and negligible functions are used to characterize the "largeness" or "smallness" of a function describing the probability of some event. Intuitively, a noticeable function is required to be larger than some inverse-polynomially function in the input parameter. On the other hand, a negligible function must be smaller than any inverse-polynomial function of the input parameter. More formally:

Definition 1.1 (Noticeable Function) *A function* $\mu(\cdot) : \mathbb{Z}^+ \to [0,1]$ *is noticeable iff* $\exists c \in \mathbb{Z}^+, n_0 \in \mathbb{Z}^+$ *such that* $\forall n \geq n_0,\ \mu(n) \geq n^{-c}$.

Example. Observe that $\mu(n) = n^{-3}$ is a noticeable function. (Notice that the above definition is satisfied for $c = 3$ and $n_0 = 1$.)

Definition 1.2 (Negligible Function) *A function* $\mu(\cdot) : \mathbb{Z}^+ \to [0,1]$ *is negligible iff* $\forall c \in \mathbb{Z}^+ \ \exists n_0 \in \mathbb{Z}^+$ *such that* $\forall n \geq n_0,\ \mu(n) < n^{-c}$.

Example. $\mu(n) = 2^{-n}$ is an example of a negligible function. This can be observed as follows. Consider an arbitrary $c \in \mathbb{Z}^+$ and set $n_0 = c^2$. Now, observe that for all $n \geq n_0$, we have that $\frac{n}{\log_2 n} \geq \frac{n_0}{\log_2 n_0} \geq \frac{n_0}{\sqrt{n_0}} = \sqrt{n_0} = c$. This allows us to conclude that

$$\mu(n) = 2^{-n} = n^{-\frac{n}{\log_2 n}} \leq n^{-c}.$$

Thus, we have proved that for any $c \in \mathbb{Z}^+$, there exists $n_0 \in \mathbb{Z}^+$ such that for any $n \geq n_0$, $\mu(n) \leq n^{-c}$.

Gap between Noticeable and Negligible Functions. At first thought it might seem that a function that is not negligible (or, a non-negligible function) must be a noticeable. This is not true! Negating the definition of a negligible function, we obtain that a non-negligible function $\mu(\cdot)$ is such that $\exists c \in \mathbb{Z}^+$ such that $\forall n_0 \in \mathbb{Z}^+$, $\exists n \geq n_0$ such that $\mu(n) \geq n^{-c}$. Note that this requirement is satisfied as long as $\mu(n) \geq n^{-c}$ for infinitely many choices of $n \in \mathbb{Z}^+$. However, a noticeable function requires this condition to be true for every $n \geq n_0$.

Below we give example of a function $\mu(\cdot)$ that is neither negligible nor noticeable.

$$\mu(n) = \begin{cases} 2^{-n} & : x \mod 2 = 0 \\ n^{-3} & : x \mod 2 \neq 0 \end{cases}$$

This function is obtained by interleaving negligible and noticeable functions. It cannot be negligible (resp., noticeable) because it is greater (resp., less) than an inverse-polynomially function for infinitely many input choices.

Properties of Negligible Functions. Sum and product of two negligible functions is still a negligible function. We argue this for the sum function below and defer the problem for products to Exercise 1.2.

Lemma 1.1 *If $\mu(n)$ and $\nu(n)$ are negligible functions from domain $\mathbb{Z}^+$ to range $[0, 1/2]$ then $\psi(n) = \mu(n) + \nu(n)$ is also a negligible function.*

Proof. We need to show that for any $c \in \mathbb{Z}^+$, we can find n_0 such that $\forall n \geq n_0$, $\psi(n) \leq n^{-c}$. Our argument proceeds as follows. Given the fact that μ and ν are negligible we can conclude that there exist n_1 and n_2 such that $\forall n \geq n_1$, $\mu(n) \leq n^{-(c+1)}$ and $\forall n \geq n_2$, $g(n) \leq n^{-(c+1)}$. Combining the above two fact and setting $n_0 = \max(n_1, n_2, 2)$ we have that for every $n \geq n_0$, $\psi(n) = \mu(n) + \nu(n) \leq n^{-(c+1)} + n^{-(c+1)} = 2n^{-(c+1)} \leq n \cdot n^{-(c+1)}$ (since, $2 \leq n_0 \leq n$). Thus, $\psi(n) \leq n^{-c}$ and hence is negligible. ■

1.2 Probabilistic Polynomial Time

A polynomial time Turing Machine is one which halts in time polynomial in its input length. A probabilistic Turing Machine is allowed to make random choices in its execution. More formally:

Definition 1.3 (Probabilistic Polynomial Time) *A Turing Machine M is a said to be PPT (Probabilistic Polynomial Time) Turing Machine if $\exists c \in \mathbb{Z}^+$ such that $\forall x \in \{0,1\}^*$, $M(x)$ halts in $|x|^c$ steps.*

A *non-uniform* PPT Turing Machine is a collection of machines one for each input length, as opposed to a single machine that must work for all input lengths.

Definition 1.4 (Non-uniform PPT) *A non-uniform PPT machine is a sequence of Turing Machines $\{M_1, M_2, \cdots\}$ such that $\exists c \in \mathbb{Z}^+$ such that $\forall x \in \{0,1\}^*$, $M_{|x|}(x)$ halts in $|x|^c$ steps.*

1.3 One-Way Functions

A one-way function $f : \{0,1\}^n \to \{0,1\}^m$ is a function that is easy to compute (computable by a polynomial time machine) but hard to invert. We formalize this by saying that there *cannot* exist a machine that can invert f in polynomial time.

Definition 1.5 (One-Way Functions) *A function $f : \{0,1\}^n \to \{0,1\}^m$ is said to be one-way function if:*

- *f is computable by a polynomial time machine, and*
- *$\forall$ non-uniform PPT adversaries $\mathcal{A}$ we have that*

$$\mu_{\mathcal{A},f}(n) = \Pr_{x \xleftarrow{\$} \{0,1\}^n} [\mathcal{A}(1^n, f(x)) \in f^{-1}(f(x))] \tag{1.1}$$

is a negligible function, where $f^{-1}(y)$ is a set such that for all $x \in f^{-1}(y)$ we have that $f(x) = y$ and $x \xleftarrow{\$} \{0,1\}^n$ denotes that x is drawn uniformly at random from the set $\{0,1\}^n$.

The above definition is rather delicate. We next describe problems in the slight variants of this definition that are insecure.

1. What if we require that $\Pr_{x \xleftarrow{\$} \{0,1\}^n}[\mathcal{A}(1^n, f(x)) \in f^{-1}(f(x))] = 0$ instead of being negligible?

   This condition is false for every function f. An adversary $\mathcal{A}$ that outputs an arbitrarily fixed value x_0 succeed in outputting a value in $f^{-1}(f(x))$ with probability at least $1/2^n$.

2. What if we drop the input 1^n to $\mathcal{A}$ in Equation 1.1?

   Consider the function $f(x) = |x|$. In this case, we have that $m = \log_2 n$, or $n = 2^m$. Intuitively, f should not be considered a one-way function, because it is easy to invert f. Namely, given a value y any x such that $|x| = y$ is such that $x \in f^{-1}(y)$. However, according to this definition the adversary gets an m bit string as input, and hence is restricted to running in time polynomial in m. Since each possible x is of size $n = 2^m$, the adversary doesn't even have enough time to write down the answer! Thus, according to the flawed definition above, f would be a one-way function.

   Providing the attacker with 1^n (n repetitions of the 1 bit) as additional input avoids this issue. In particular, it allows the attacker to run in time polynomial in m and n.

A Candidate One-way Function. It is not known whether one-way functions exist. The existence of one-way functions would imply that $P \neq NP$ (see Exercise 1.3), and so of course we do not know of any concrete functions that have been proved to be one-way.

However, there are candidates of functions that could be one-way functions. One example is based on the hardness of factoring. Multiplication can be done easily in $O(n^2)$ time, but so far no polynomial time algorithm is known for factoring.

One candidate might be to say that given an input x, split x into its left and right halves x_1 and x_2, and then output $x_1 \times x_2$. However, this is not a one-way function, because with probability $\frac{3}{4}$, 2 will be a factor of $x_1 \times x_2$, and in general the factors are small often enough that a non-negligible number of the outputs could be factored in polynomial time.

To improve this, we again split x into x_1 and x_2, and use x_1 and x_2 as seeds in order to generate large primes p and q, and then output pq. Since p and q are primes, it is hard to factor pq, and so it is hard to retrieve x_1 and x_2. This function is believed to be one-way.

1.4 Composability of One-Way Functions

Given a one-way function $f : \{0,1\}^n \to \{0,1\}^n$, is the function $f^2(x) = f(f(x))$ also a one-way function? Intuitively, it seems that if it is hard to invert $f(x)$, then it would be just as hard to invert $f(f(x))$. However, this intuition is incorrect and highlights the delicacy when working with cryptographic assumptions and primitives. In particular, assuming one-way functions exists we describe a one-way function $f : \{0,1\}^{n/2} \times \{0,1\}^{n/2} \to \{0,1\}^n$ such that f^2 can be efficiently inverted. Let $g : \{0,1\}^n \to \{0,1\}^n$ be a one-way function then we set f as follows:

$$f(x_1, x_2) = \begin{cases} 0^n & : \text{if } x_1 = 0^{n/2} \\ 0^{n/2} \| g(x_2) & : \text{otherwise} \end{cases}$$

Two observations follow:

1. f^2 is not one-way. This follows from the fact that for all inputs x_1, x_2 we have that $f^2(x_1, x_2) = 0^n$. This function is clearly not one-way!

2. f is one-way. This can be argued as follows. Assume that there exists an adversary $\mathcal{A}$ such that $\mu_{\mathcal{A},f}(n) = \Pr_{x \xleftarrow{\$} \{0,1\}^n}[\mathcal{A}(1^n, f(x)) \in f^{-1}(f(x))]$ is non-negligible. Using such an $\mathcal{A}$ we will describe a construction of adversary $\mathcal{B}$ such that $\mu_{\mathcal{B},g}(n) = \Pr_{x \xleftarrow{\$} \{0,1\}^n}[\mathcal{B}(1^n, g(x)) \in g^{-1}(g(x))]$ is also non-negligible. This would be a contradiction thus proving our claim.

   Description of $\mathcal{B}$: $\mathcal{B}$ on input $y \in \{0,1\}^n$ outputs the n lower-order bits of $\mathcal{A}(1^{2n}, 0^n \| y)$.

   Observe that if $\mathcal{A}$ successfully inverts f then we have that $\mathcal{B}$ successfully inverts g. More formally, we have that:

$$\mu_{\mathcal{B},g}(n) = \Pr_{x \xleftarrow{\$} \{0,1\}^n} \left[\mathcal{A}(1^{2n}, 0^n || g(x)) \in \{0,1\}^n || g^{-1}(g(x)) \right].$$

   Note that

$$\begin{aligned}
\mu_{\mathcal{A},f}(2n) = & \Pr_{x_1, x_2 \xleftarrow{\$} \{0,1\}^{2n}} [\mathcal{A}(1^{2n}, f(x_1, x_2)) \in f^{-1}(f(\tilde{x}))] \\
\leq & \Pr_{x_1 \xleftarrow{\$} \{0,1\}^n} [x_1 = 0^n] + \Pr_{x_1 \xleftarrow{\$} \{0,1\}^n} [x_1 \neq 0^n] \Pr_{x_2 \xleftarrow{\$} \{0,1\}^n} [\mathcal{A}(1^{2n}, 0^n || g(x_2)) \in \{0,1\}^n || g^{-1}(g(x_2))] \\
= & \frac{1}{2^n} + \left(1 - \frac{1}{2^n}\right) \cdot \Pr_{x_2 \xleftarrow{\$} \{0,1\}^n} [\mathcal{A}(1^{2n}, 0^n || g(x_2)) \in \{0,1\}^n || g^{-1}(g(x_2))] \\
= & \frac{1}{2^n} + \left(1 - \frac{1}{2^n}\right) \cdot \mu_{\mathcal{B},g}(n).
\end{aligned}$$

   Rewriting the above expression, we have that $\mu_{\mathcal{B},g}(n) = \frac{\mu_{\mathcal{A},f}(2n) - \frac{1}{2^n}}{1 - \frac{1}{2^n}}$ which is non-negligible as long as $\mu_{\mathcal{A},f}(2n)$ is non-negligible.

1.5 Examples

The goal of this section is to illustrate the general strategy for the problems of the form,

"If f is one-way function, then show that f' (derived from f) is not a one-way function"

Some of the examples include:

- If f is a one-way function, prove that f' defined as $f(f(\cdot))$ is not one-way.
- If f is a one-way function, prove that f' defined by dropping the first bit the output of f is not one-way.

In order to give such a proof, we need to give an example of an one-way function f and show that f' (derived from f) is not one-way. The general strategy for these types of problems is the following:

1. Come up with a contrived function g and show that g is one-way.
2. Construct the new function g' that is derived from g.
3. Show that g' can be inverted with non-negligible probability and thus show that g' is not one-way.

The reason why we need to come-up with a contrived function is that for specific one-way function f, f' (derived from f) could be one-way. To see why this is the case, consider a one-way function $f : \{0,1\}^n \to \{0,1\}^n$ that is additionally injective. Then, one can show that $f^2(\cdot)$ is in fact a one-way function.[1] On the other hand, in the previous section, we showed that there exists a (contrived) function g such that g is one-way but g^2 is not one-way. Hence, we might not always be able to start from any one-way function f and show that f' (derived from f) is not one-way. The first step where we come up a suitable g requires some ingenuity. Once that is done, the second and the third steps would generally be not so hard.

To illustrate these three steps, let us consider a concrete example. We want to show that if f is one-way then f' that is defined by dropping the first bit of the output of f is not one-way.

Step-1: Designing the function g. We want to come up with a (contrived) function g and prove that it is one-way. Let us assume that there exists a one-way function $h : \{0,1\}^n \to \{0,1\}^n$. We define the function $g : \{0,1\}^{2n} \to \{0,1\}^{2n}$ as follows:

$$g(x\|y) = \begin{cases} 0^n\|y & \text{if } x = 0^n \\ 1\|0^{n-1}\|g(y) & \text{otherwise} \end{cases}$$

Claim 1.1 *If h is a one-way function, then so is g.*

Proof. Assume for the sake of contradiction that g is not one-way. Then there exists a polynomial time adversary $\mathcal{A}$ and a non-negligible function $\mu(\cdot)$ such that:

$$\Pr_{x,y}[\mathcal{A}(1^n, g(x\|y)) \in g^{-1}(g(x\|y))] = \mu(n)$$

We will use such an adversary $\mathcal{A}$ to invert h with some non-negligible probability. This contradicts the one-wayness of h and thus our assumption that g is not one-way function is false.

Let us now construct an $\mathcal{B}$ that uses $\mathcal{A}$ and inverts h. $\mathcal{B}$ is given $1^n, h(y)$ for a randomly chosen y and its goal is to output $y' \in h^{-1}(h(y))$ with some non-negligible probability. $\mathcal{B}$ works as follows:

1. It samples $x \leftarrow \{0,1\}^n$ randomly.
2. If $x = 0^n$, it samples a random $y' \leftarrow \{0,1\}^n$ and outputs it.

[1] Try to prove this!

3. Otherwise, it runs $\mathcal{A}(10^{n-1}\|h(y))$ and obtains $x'\|y'$. It outputs y'.

Let us first analyze the running time of $\mathcal{B}$. The first two steps are clearly polynomial (in n) time. In the third step, $\mathcal{B}$ runs $\mathcal{A}$ and uses its output. Note that the running time of since $\mathcal{A}$ runs in polynomial (in n) time, this step also takes polynomial (in n) time. Thus, the overall running time of $\mathcal{B}$ is polynomial (in n).

Let us now calculate the probability that $\mathcal{B}$ outputs the correct inverse. If $x = 0^n$, the probability that y' is the correct inverse is at least $\frac{1}{2^n}$ (because it guesses y' randomly and probability that a random y' is the correct inverse is $\geq 1/2^n$). On the other hand, if $x \neq 0^n$, then the probability that $\mathcal{B}$ outputs the correct inverse is $\mu(n)$. Thus,

$$\begin{aligned}
\Pr[\mathcal{B}(1^n, h(y)) \in h^{-1}(h(y))] &\geq \Pr[x = 0^n](\frac{1}{2^n}) + \Pr[x \neq 0^n]\mu(n) \\
&= \frac{1}{2^{2n}} + (1 - \frac{1}{2^n})\mu(n) \\
&\geq \mu(n) - (\frac{1}{2^n} - \frac{1}{2^{2n}})
\end{aligned}$$

Since $\mu(n)$ is a non-negligible function and $(\frac{1}{2^n} - \frac{1}{2^{2n}})$ is a negligible function, their difference is non-negligible.[2] This contradicts the one-wayness of h.

Step-2: Constructing g'. We construct the new function $g' : \{0,1\}^{2n} \to \{0,1\}^{2n-1}$ by dropping the first bit of g. That is,

$$g'(x\|y) = \begin{cases} 0^{n-1}\|y & \text{if } x = 0^n \\ 0^{n-1}\|g(y) & \text{otherwise} \end{cases}$$

Step-3: Inverting g'. We now want to prove that g' is not one-way. That is, we want to design an adversary $\mathcal{C}$ such that given 1^{2n} and $g'(x\|y)$ for a randomly chosen x, y, it outputs an element in the set $g^{-1}(g(x\|y))$. The description of $\mathcal{C}$ is as follows:

- On input 1^{2n} and $g'(x\|y)$, the adversary $\mathcal{C}$ parses $g'(x\|y)$ as $0^{n-1}\|\overline{y}$.
- It outputs $0^n\|\overline{y}$ as the inverse.

Notice that $g'(0^n\|\overline{y}) = 0^{n-1}\|\overline{y}$. Thus, $\mathcal{C}$ succeeds with probability 1 and this breaks the one-wayness of g'.

1.6 Hardness Amplification

In this section, we show that even a very *weak* form of one-way functions suffices from constructing one-way functions as defined previously. For this section, we refer to this previously defined notion as strong one-way functions.

Definition 1.6 (Weak One-Way Functions) *A function $f : \{0,1\}^n \to \{0,1\}^m$ is said to be a weak one-way function if:*

- f is computable by a polynomial time machine, and

[2]Exercise: Prove that if $\alpha(\cdot)$ is a non-negligible function and $\beta(\cdot)$ is a negligible function, then $(\alpha - \beta)(\cdot)$ is a non-negligible function.

- *There exists a noticeable function $\alpha_f(\cdot)$ such that $\forall$ non-uniform PPT adversaries $\mathcal{A}$ we have that*

$$\mu_{\mathcal{A},f}(n) = \Pr_{x \xleftarrow{\$} \{0,1\}^n}[\mathcal{A}(1^n, f(x)) \in f^{-1}(f(x))] \leq 1 - \alpha_f(n).$$

Theorem 1.1 *If there exists a weak one-way function, then there exists a (strong) one-way function.*

Proof. We prove the above theorem constructively. Suppose $f : \{0,1\}^n \to \{0,1\}^m$ is a weak one-way function, then we prove that the function $g : \{0,1\}^{nq} \to \{0,1\}^{mq}$ for $q = \lceil \frac{2n}{\alpha_f(n)} \rceil$

$$g(x_1, x_2, \cdots, x_q) = f(x_1)||f(x_2)||\cdots||f(x_q),$$

is a strong one-way function.

Assume for the sake of contradiction that there exists an adversary $\mathcal{B}$ such that $\mu_{\mathcal{B},g}(nq) = \Pr_{x \xleftarrow{\$} \{0,1\}^{nq}}[\mathcal{B}(1^{nq}, g(x)) \in g^{-1}(g(x))]$ is non-negligible. Then we use $\mathcal{B}$ to construct $\mathcal{A}$ (see Figure 1.1) that breaks f, namely $\mu_{\mathcal{A},f}(n) = \Pr_{x \xleftarrow{\$} \{0,1\}^n}[\mathcal{A}(1^n, f(x)) \in f^{-1}(f(x))] > 1 - \alpha_f(n)$ for sufficiently large n.

```
1: loop T = 4n^2 / (α_f(n) μ_{B,g}(nq)) times
2:     i ←$ {1, 2, ··· , q}
3:     x_1, ··· , x_{i−1}, x_i, ··· , x_q ←$ {0,1}^n
4:     (x'_1, x'_2, ··· , x'_q) := B(f(x_1), f(x_2), ··· , f(x_q))
5:     if f(x'_i) = y then
6:         return x'_i
7: return ⊥
```

Figure 1.1: Construction of $\mathcal{A}(1^n, y)$

Note that: (1) $\mathcal{A}(1^n, y)$ iterates at most $T = \frac{4n^2}{\alpha_f(n)\mu_{\mathcal{B},g}(nq)}$ times each call is polynomial time. (2) $\mu_{\mathcal{B},g}(nq)$ is a non-negligible function. This implies that for infinite choices of n this value is greater than some noticeable function. Together these two facts imply that for infinite choices of n the running time of $\mathcal{A}$ is bounded by a polynomial function in n.

It remains to show that $\Pr_{x \xleftarrow{\$} \{0,1\}^n}[\mathcal{A}(1^n, f(x)) = \perp] < \alpha_f(n)$ for arbitrarily large n. A natural way to argue this is by showing that at least one execution of $\mathcal{B}$ should suffice for inverting $f(x)$. However, the technical challenge in proving this formally is that these calls to $\mathcal{B}$ aren't independent. Below we formalize this argument even when these calls aren't independent.

Define the set S of "bad" x's, which are hard to invert:

$$S := \left\{ x \,\middle|\, \Pr_{\mathcal{B}}[\mathcal{A} \text{ inverts } f(x) \text{ in a single iteration}] \leq \frac{\alpha_f(n)\mu_{\mathcal{B},g}(nq)}{4n} \right\}.$$

We start by proving that the size of S is small. More formally,

$$\Pr_{x \xleftarrow{\$} \{0,1\}^n}[x \in S] \leq \frac{\alpha_f(n)}{2}.$$

Assume, for the sake of contradiction, that $\Pr_{x \xleftarrow{\$} \{0,1\}^n}[x \in S] > \frac{\alpha_f(n)}{2}$. Then we have that:

$$\begin{aligned}
\mu_{\mathcal{B},g}(nq) =& \Pr_{(x_1,\cdots,x_q) \xleftarrow{\$} \{0,1\}^{nq}}[\mathcal{B}(1^{nq}, g(x_1,\cdots,x_q)) \in g^{-1}(g(x_1,\cdots,x_q))] \\
=& \Pr_{x_1,\cdots,x_q}[\mathcal{B}(1^{nq}, g(x_1,\cdots,x_q)) \in g^{-1}(g(x_1,\cdots,x_q)) \wedge \forall i : x_i \notin S] \\
&+ \Pr_{x_1,\cdots,x_q}[\mathcal{B}(1^{nq}, g(x_1,\cdots,x_q)) \in g^{-1}(g(x_1,\cdots,x_q)) \wedge \exists i : x_i \in S] \\
\leq& \Pr_{x_1,\cdots,x_q}[\forall i : x_i \notin S] + \sum_{i=1}^{q} \Pr_{x_1,\cdots,x_q}[\mathcal{B}(1^{nq}, g(x_1,\cdots,x_q)) \in g^{-1}(g(x_1,\cdots,x_q)) \wedge x_i \in S] \\
\leq& \left(1 - \frac{\alpha_f(n)}{2}\right)^q + q \cdot \Pr_{x_1,\cdots,x_q,i}[\mathcal{B}(1^{nq}, g(x_1,\cdots,x_q)) \in g^{-1}(g(x_1,\cdots,x_q)) \wedge x_i \in S] \\
=& \left(1 - \frac{\alpha_f(n)}{2}\right)^{\frac{2n}{\alpha_f(n)}} + q \cdot \Pr_{x \xleftarrow{\$} \{0,1\}^n, \mathcal{B}}[\mathcal{A} \text{ inverts } f(x) \text{ in a single iteration} \wedge x \in S] \\
\leq& e^{-n} + q \cdot \Pr_x[x \in S] \cdot \Pr[\mathcal{A} \text{ inverts } f(x) \text{ in a single iteration} \mid x \in S] \\
\leq& e^{-n} + \frac{2n}{\alpha_f(n)} \cdot 1 \cdot \frac{\mu_{\mathcal{B},g}(nq) \cdot \alpha_f(n)}{4n} \\
\leq& e^{-n} + \frac{\mu_{\mathcal{B},g}(nq)}{2}.
\end{aligned}$$

Hence $\mu_{\mathcal{B},g}(nq) \leq 2e^{-n}$, contradicting with the fact that $\mu_{\mathcal{B},g}$ is non-negligible. Then we have

$$\begin{aligned}
&\Pr_{x \xleftarrow{\$} \{0,1\}^n}[\mathcal{A}(1^n, f(x)) = \perp] \\
&\quad = \Pr_x[x \in S] + \Pr_x[x \notin S] \cdot \Pr[\mathcal{B} \text{ fails to invert } f(x) \text{ in every iteration} | x \notin S] \\
&\quad \leq \frac{\alpha_f(n)}{2} + (\Pr[\mathcal{B} \text{ fails to invert } f(x) \text{ a single iteration} | x \notin S])^T \\
&\quad \leq \frac{\alpha_f(n)}{2} + \left(1 - \frac{\mu_{\mathcal{A},g}(nq) \cdot \alpha_f(n)}{4n}\right)^T \\
&\quad \leq \frac{\alpha_f(n)}{2} + e^{-n} \leq \alpha_f(n)
\end{aligned}$$

for sufficiently large n. This concludes the proof. ■

1.7 Levin's One-Way Function

Theorem 1.2 *If there exists a one-way function, then there exists an explicit function f that is one-way (constructively).*

Lemma 1.2 *If there exists a one-way function computable in time n^c for a constant c, then there exists a one-way function computable in time n^2.*

Proof. Let $f : \{0,1\}^n \to \{0,1\}^n$ be a one-way function computable in time n^c. Construct $g : \{0,1\}^{n+n^c} \to \{0,1\}^{n+n^c}$ as follows:

$$g(x, y) = f(x)||y$$

where $x \in \{0,1\}^n, y \in \{0,1\}^{n^c}$. $g(x,y)$ takes time $2n^c$, which is linear in the input length.

We next show that $g(\cdot)$ is one-way. Assume for the purpose of contradiction that there exists an adversary $\mathcal{A}$ such that $\mu_{\mathcal{A},g}(n+n^c) = \Pr_{(x,y)\xleftarrow{\$}\{0,1\}^{n+n^c}}[\mathcal{A}(1^{n+n^c}, g(x,y)) \in g^{-1}(g(x,y))]$ is non-negligible. Then we use $\mathcal{A}$ to construct $\mathcal{B}$ such that $\mu_{\mathcal{B},f}(n) = \Pr_{x\xleftarrow{\$}\{0,1\}^n}[\mathcal{B}(1^n, f(x)) \in f^{-1}(f(x))]$ is also non-negligible.

$\mathcal{B}$ on input $z \in \{0,1\}^n$, samples $y \xleftarrow{\$} \{0,1\}^{n^c}$, and outputs the n higher-order bits of $\mathcal{A}(1^{n+n^c}, z||y)$. Then we have

$$\begin{aligned}\mu_{\mathcal{B},g}(n) =& \Pr_{x\xleftarrow{\$}\{0,1\}^n, y\xleftarrow{\$}\{0,1\}^{n^c}}\left[\mathcal{A}(1^{n+n^c}, f(x)||y) \in f^{-1}(f(x))||\{0,1\}^{n^c}\right]\\ \geq& \Pr_{x,y}\left[\mathcal{A}(1^{n+n^c}, g(x,y)) \in f^{-1}(f(x))||y\right]\\ =& \Pr_{x,y}\left[\mathcal{A}(1^{n+n^c}, g(x,y)) \in g^{-1}(g(x,y))\right]\end{aligned}$$

is non-negligible. ■

Proof. [of Theorem 1.2] We first construct a weak one-way function $h : \{0,1\}^n \to \{0,1\}^n$ as follows:

$$h(M,x) = \begin{cases} M||M(x) & \text{if } M(x) \text{ takes no more than } |x|^2 \text{ steps} \\ M||0 & \text{otherwise}\end{cases}$$

where $|M| = \log n, |x| = n - \log n$ (interpreting M as the code of a machine and x as its input). If h is weak one-way, then we can construct a one-way function from h as we discussed in Section 1.6.

It remains to show that if one-way functions exist, then h is a weak one-way function, with $\alpha_h(n) = \frac{1}{n^2}$. Assume for the purpose of contradiction that there exists an adversary $\mathcal{A}$ such that $\mu_{\mathcal{A},h}(n) = \Pr_{(M,x)\xleftarrow{\$}\{0,1\}^n}[\mathcal{A}(1^n, h(M,x)) \in h^{-1}(h(M,x))] \geq 1 - \frac{1}{n^2}$ for all sufficiently large n. By the existence of one-way functions and Lemma 1.2, there exists a one-way function $\tilde{M}$ that can be computed in time n^2. Let $\tilde{M}$ be the uniform machine that computes this one-way function. We will consider values n such that $n > 2^{|\tilde{M}|}$. In other words for these choices of n, $\tilde{M}$ can be described using $\log n$ bits. We construct $\mathcal{B}$ to invert $\tilde{M}$: on input y outputs the $(n - \log n)$ lower-order bits of $\mathcal{A}(1^n, \tilde{M}||y)$. Then

$$\begin{aligned}\mu_{\mathcal{B},\tilde{M}}(n-\log n) =& \Pr_{x\xleftarrow{\$}\{0,1\}^{n-\log n}}\left[\mathcal{A}(1^n, \tilde{M}||\tilde{M}(x)) \in \{0,1\}^{\log n}||\tilde{M}^{-1}(\tilde{M}((x))\right]\\ \geq& \Pr_{x\xleftarrow{\$}\{0,1\}^{n-\log n}}\left[\mathcal{A}(1^n, \tilde{M}||\tilde{M}(x)) \in \tilde{M}||\tilde{M}^{-1}(\tilde{M}((x))\right].\end{aligned}$$

Observe that for sufficiently large n it holds that

$$\begin{aligned}1 - \frac{1}{n^2} \leq& \mu_{\mathcal{A},h}(n)\\ =& \Pr_{(M,x)\xleftarrow{\$}\{0,1\}^n}\left[\mathcal{A}(1^n, h(M,x)) \in h^{-1}(h(M,x))\right]\\ \leq& \Pr_M[M = \tilde{M}] \cdot \Pr_x\left[\mathcal{A}(1^n, \tilde{M}||\tilde{M}(x)) \in \tilde{M}||\tilde{M}^{-1}(\tilde{M}((x))\right] + \Pr_M[M \neq \tilde{M}]\\ \leq& \frac{1}{n} \cdot \mu_{\mathcal{B},\tilde{M}}(n-\log n) + \frac{n-1}{n}.\end{aligned}$$

Hence $\mu_{\mathcal{B},\tilde{M}}(n - \log n) \geq \frac{n-1}{n}$ for sufficiently large n which is a contradiction. ■

Exercises

Exercise 1.1 *If $\mu(\cdot)$ and $\nu(\cdot)$ are negligible functions then show that $\mu(\cdot) \cdot \nu(\cdot)$ is a negligible function.*

Exercise 1.2 *If $\mu(\cdot)$ is a negligible function and $f(\cdot)$ is a function polynomial in its input then show that $\mu(f(\cdot))$[3] are negligible functions.*

Exercise 1.3 *Prove that the existence of one-way functions implies $P \neq NP$.*

Exercise 1.4 *Prove that there is no one-way function $f : \{0,1\}^n \to \{0,1\}^{\lfloor \log_2 n \rfloor}$.*

Exercise 1.5 *Let $f : \{0,1\}^n \to \{0,1\}^n$ be any one-way function then is $f'(x) \stackrel{def}{=} f(x) \oplus x$ necessarily one-way?*

Exercise 1.6 *Prove or disprove: If $f : \{0,1\}^n \to \{0,1\}^n$ is a one-way function, then $g : \{0,1\}^n \to \{0,1\}^{n-\log n}$ is a one-way function, where $g(x)$ outputs the $n - \log n$ higher order bits of $f(x)$.*

Exercise 1.7 *Explain why the proof of Theorem 1.1 fails if the attacker $\mathcal{A}$ in Figure 1.1 sets $i = 1$ and not $i \stackrel{\$}{\leftarrow} \{1, 2, \cdots, q\}$.*

Exercise 1.8 *Given a (strong) one-way function construct a weak one-way function that is not a (strong) one-way function.*

Exercise 1.9 *Let $f : \{0,1\}^n \to \{0,1\}^n$ be a weak one-way permutation (a weak one way function that is a bijection). More formally, f is a PPT computable one-to-one function such that $\exists$ a constant $c > 0$ such that $\forall$ non-uniform PPT machine A and $\forall$ sufficiently large n we have that:*

$$\Pr_{x,A}[A(f(x)) \notin f^{-1}(f(x))] > \frac{1}{n^c}$$

Show that $g(x) = f^T(x)$ is not a strong one way permutation. Here f^T denotes the T times self composition of f and T is a polynomial in n.

Interesting follow up reading if interested: With some tweaks the function above can be made a strong one-way permutation using explicit constructions of expander graphs. See Section 2.6 in `http://www.wisdom.weizmann.ac.il/~oded/PSBookFrag/part2N.ps`

[3] Assume that μ and f are such that $\mu(f(\cdot))$ takes inputs from $\mathbb{Z}^+$ and outputs values in $[0,1]$.

Chapter 2

Pseudorandomness

2.1 Hard Core Bit

We start by asking the following question: Is it possible to concentrate the strength of a one-way function into one bit? In particular, given a one-way function f, does there exist one bit that can be computed efficiently from the input x, but is hard to compute given $f(x)$?

Definition 2.1 (Hard Core Bit) *Let $f : \{0,1\}^n \to \{0,1\}^n$ be a one-way function. $B : \{0,1\}^n \to \{0,1\}$ is a hard core bit of f if:*

- *B is computable by a polynomial time machine, and*
- *$\forall$ non-uniform PPT adversaries $\mathcal{A}$ we have that*

$$\Pr_{x \xleftarrow{\$} \{0,1\}^n}[\mathcal{A}(1^n, f(x)) = B(x)] \leq \frac{1}{2} + \mathsf{negl}(n).$$

A simple example. Let f be a one-way function. Consider the one-way function $g(b,x) = 0||f(x)$ and a hard core bit $B(b,x) = b$. Intuitively, the value $g(b,x)$ does not reveal any information about the first bit b, thus no information about the value $B(b,x)$ can be ascertained. Hence $\mathcal{A}$ cannot predict the first bit with a non-negligible advantage than a random guess. However, we are more interested in the case where the hard core bit is hidden because of computational hardness and not information theoretic hardness.

Remark 2.1 *Given a one-way function f, we can construct another one-way function g with a hard core bit. However, we may not be able to find a hard core bit for f. In fact, it is an open question whether a hard core bit exists for every one-way function.*

Intuitively, if a function f is one-way, there should be a particular bit in the input x that is hard to compute given $f(x)$. But this is not true:

Claim 2.1 *If $f : \{0,1\}^n \to \{0,1\}^n$ is a one-way function, then there exists a one-way function $g : \{0,1\}^{n+\log n} \to \{0,1\}^{n+\log n}$ such that $\forall 1 \leq i \leq n + \log n$, $B_i(x) = x_i$ is not a hard core bit, where x_i is the i^{th} bit of x.*

Proof. Define $g : \{0,1\}^{n+\log(n)} \to \{0,1\}^{n+\log(n)}$ as follows.

$$g(x,y) = f(x_{\bar{y}})||x_y||y,$$

where $|x| = n, |y| = \log n$, $x_{\bar{y}}$ is all bits of x except the y^{th} bit, x_y is the y^{th} bit of x.

First, one can show that g is still a one-way function. (We leave this as an exercise!) We next show that B_i is not a hard core bit for $\forall 1 \leq i \leq n$ (clearly B_i is not a hard core bit for $n+1 \leq i \leq n+\log n$). Construct an adversary $\mathcal{A}_i(1^{n+\log n}, f(x_{\bar{y}})||x_y||y)$ that "breaks" B_i:

- If $y \neq i$ then output a random bit;
- Otherwise output x_y.

$$\begin{aligned}
&\Pr_{x,y}[\mathcal{A}(1^{n+\log n}, g(x,y)) = B_i(x)] \\
=&\Pr_{x,y}[\mathcal{A}(1^{n+\log n}, f(x_{\bar{y}})||x_y||y) = x_i] \\
=&\frac{n-1}{n} \cdot \frac{1}{2} + \frac{1}{n} \cdot 1 = \frac{1}{2} + \frac{1}{2n}.
\end{aligned}$$

Hence $\mathcal{A}_i$ can guess the output of B_i with greater than $\frac{1}{2} + \mathsf{negl}(n)$ probability. ■

Application: Coin tossing over the phone. We next describe an application of hard core bits to coin tossing. Consider two parties trying to perform a coin tossing over the phone. In this setting the first party needs to declare its choice as the second one flips the coin. However, how can the first party trust the win/loss response from the second party? In particular, if the first party calls out "head" and then the second party can just lie that it was "tails." We can use hard core bit of a (one-to-one) one-way function to enable this applications.

Let f be a (one-to-one) one-way function and B be a hard core bit for f. Consider the following protocol:

- Party P_1 samples x from $\{0,1\}^n$ uniformly at random and sends y, where $y = f(x)$, to party P_2.
- P_2 sends back a random bit b sampled from $\{0,1\}$.
- P_1 sends back $(x, B(x))$ to P_2. P_2 aborts if $f(x) \neq y$.
- Both parties output $B(x) \oplus b$.

Note that P_2 cannot guess $B(x)$ with a non-negligible advantage than $1/2$ as he sends back his b. On the other hand, P_1 cannot flip the value $B(x)$ once it has sent $f(x)$ to P_2 because f is one-to-one.

2.2 Hard Core Bit of any One-Way Functions

We now show that a slight modification of every one-way function has a hard core bit. More formally,

Theorem 2.1 *Let $f : \{0,1\}^n \to \{0,1\}^n$ be a one-way function. Define a function $g : \{0,1\}^{2n} \to \{0,1\}^{2n}$ as follows:*

$$g(x,r) = f(x)||r,$$

where $|x| = |r| = n$. Then we have that g is one-way and that it has a hard core bit, namely $B(x,r) = \sum_{i=1}^{n} x_i r_i \mod 2$.

Remark 2.2 *If f is a (one-to-one) one-way function, then g is also a (one-to-one) one-way function with hard core bit $B(\cdot)$.*

Proof. We leave it as an exercise to show that g is a one-way function and below we will prove that the function $B(\cdot)$ describe a hard core bit of g. More specifically, we need to show that if there exists a non-uniform PPT $\mathcal{A}$ s.t. $\Pr_{x,r}[\mathcal{A}(1^{2n}, g(x,r)) = B(x,r)] \geq \frac{1}{2} + \epsilon(n)$, where ϵ is non-negligible, then there exists a non-uniform PPT $\mathcal{B}$ such that $\Pr_{x,r}[\mathcal{B}(1^{2n}, g(x,r)) \in g^{-1}(g(x,r))]$ is non-negligible. Below we use E to denote the event that $\mathcal{A}(1^{2n}, g(x,r)) = B(x,r)$. We will provide our proof in a sequence of three steps of complexity: (1) the super simple case where we restrict to $\mathcal{A}$ such that $\Pr_{x,r}[E] = 1$, (2) the simple case where we restrict to $\mathcal{A}$ such that $\Pr_{x,r}[E] \geq \frac{3}{4} + \epsilon(n)$, and finally (3) the general case with $\Pr_{x,r}[E] \geq \frac{1}{2} + \epsilon(n)$.

Super simple case. Suppose that $\mathcal{A}$ breaks the B with perfect accuracy:

$$\Pr_{x,r}[E] = 1.$$

We now construct $\mathcal{B}$ that inverts g with perfect accuracy. Let e^i be an n-bit string $0 \cdots 010 \cdots 0$, where only the i-th bit is 1, the rest are all 0. On input $f(x)||R$, $\mathcal{B}$ does the following:

for $i = 1$ **to** n **do**
  $x'_i \leftarrow \mathcal{A}(1^{2n}, f(x)||e^i)$
return $x'_1 \cdots x'_n||R$

Observe that $B(x, e^i) = \sum_{j=1}^{n} x_j e^i_j = x_i$. Therefore, the probability that $\mathcal{B}$ inverts a single bit successfully is,

$$\Pr_x\left[\mathcal{A}(1^{2n}, f(x)||e^i) = x_i\right] = \Pr_x\left[\mathcal{A}(1^{2n}, f(x)||e^i) = B(x, e^i)\right] = 1.$$

Hence $\Pr_{x,r}[\mathcal{B}(1^{2n}, g(x,r)) = (x,r)] = 1$.

Simple case. Next moving on to the following more demanding case.

$$\Pr_{x,r}[E] \geq \frac{3}{4} + \epsilon(n),$$

where ϵ is non-negligible. Just like the super simple case, we describe our algorithm of $\mathcal{B}$ for inverting g. On input $f(x)||R$, $\mathcal{B}$ proceeds as follows:

for $i = 1$ **to** n **do**
  for $t = 1$ **to** $T = \frac{n}{2\epsilon(n)^2}$ **do**
    $r \xleftarrow{\$} \{0,1\}^n$
    $x^t_i \leftarrow \mathcal{A}(f(x)||r) \oplus \mathcal{A}(f(x)||r + e^i)$
  $x'_i \leftarrow$ the majority of $\{x^1_i, \cdots, x^T_i\}$
return $x'_1 \cdots x'_n||R$

Correctness of $\mathcal{B}$ given that $\mathcal{A}$ calls output the correct answer follows by observing that $B(x,r) \oplus$

$B(x, r \oplus e^i) = x_i$:

$$\begin{aligned}
&B(x,r) \oplus B(x, r \oplus e^i) \\
=&\sum_j x_j r_j + \sum_j x_j (r_j \oplus e^i_j) \mod 2 \\
=&\sum_{j \neq i} (x_j r_j + x_j r_j) + x_i r_i + x_i (r_i + 1) \mod 2 \\
=&x_i.
\end{aligned}$$

The key technical challenge in proving that $\mathcal{B}$ inverts g with non-negligible probability arises from the fact that the calls to $\mathcal{A}$ made during one execution of $\mathcal{B}$ are not independent. In particular, all calls to $\mathcal{A}$ share the same x and the class $\mathcal{A}(f(x)||r)$ and $\mathcal{A}(f(x)||r + e^i)$ use correlated randomness as well. We solve the first issue by showing that exists a large choices of values of x for which $\mathcal{A}$ still works with large probability. The later issue of lack of independent of $\mathcal{A}(f(x)||r)$ and $\mathcal{A}(f(x)||r+e^i)$ will be solved using a union bound.
Formally, define the set G of "good" x's, which are easy for $\mathcal{A}$ to predict:

$$G := \left\{ x \left| \Pr_r [E] \geq \frac{3}{4} + \frac{\epsilon(n)}{2} \right. \right\}.$$

We start by proving that the size of G is not small. More formally we claim that,

$$\Pr_{x \xleftarrow{\$} \{0,1\}^n} [x \in G] \geq \frac{\epsilon(n)}{2}.$$

Assume, that $\Pr_{x \xleftarrow{\$} \{0,1\}^n}[x \in G] < \frac{\epsilon(n)}{2}$. Then we have the following contradiction:

$$\begin{aligned}
\frac{3}{4} + \epsilon(n) \leq& \Pr_{x,r}[E] \\
=& \Pr_x[x \in G] \Pr_r[E | x \in G] + \Pr_x[x \notin G] \Pr_r[E | x \notin G] \\
<& \frac{\epsilon(n)}{2} \cdot 1 + 1 \cdot \left(\frac{3}{4} + \frac{\epsilon(n)}{2} \right) = \frac{3}{4} + \epsilon(n).
\end{aligned}$$

For and fixed $x \in G$:

$$\begin{aligned}
&\Pr_r \left[\mathcal{A}(f(x), r) \oplus \mathcal{A}(f(x), r + e^i) = x_i \right] \\
=& \Pr_r [\text{Both } \mathcal{A}\text{'s are correct}] + \Pr_r [\text{Both } \mathcal{A}\text{'s are wrong}] \\
\geq& \Pr_r [\text{Both } \mathcal{A}\text{'s are correct}] \\
\geq& 1 - 2 \cdot \Pr_r [\text{Either } \mathcal{A} \text{ is correct}] \\
\geq& 1 - 2 \left(\frac{1}{4} - \frac{\epsilon(n)}{2} \right) = \frac{1}{2} + \epsilon(n).
\end{aligned}$$

Let Y_i^t be the indicator random variable that $x_i^t = x_i$ (namely, $Y_i^t = 1$ with probability $\Pr[x_i^t = x_i]$ and $Y_i^t = 0$ otherwise). Note that $Y_i^1, \cdots, Y_i^T$ are independent and identical random variables,

and for all $t \in \{1, \ldots, T\}$ we have that $\Pr[Y_i^t = 1] = \Pr[x_i^t = x_i] \geq \frac{1}{2} + \epsilon(n)$. Next we argue that majority of $x_i^1, \ldots x_i^T$ coincides with x_i with high probability.

$$\begin{aligned}
\Pr[x_i' \neq x_i] &= \Pr\left[\sum_{t=1}^{T} Y_i^t \leq \frac{T}{2}\right] \\
&= \Pr\left[\sum_{t=1}^{T} Y_i^t - \left(\frac{1}{2} + \epsilon(n)\right) T \leq \frac{T}{2} - \left(\frac{1}{2} + \epsilon(n)\right) T\right] \\
&\leq \Pr\left[\left|\sum_{t=1}^{T} Y_i^t - \left(\frac{1}{2} + \epsilon(n)\right) T\right| \geq \epsilon(n) T\right] \\
&\quad \text{Let } X_1, \cdots, X_m \text{ be i.i.d. random variables taking values 0 or 1. Let } \Pr[X_i = 1] = p. \\
&\quad \text{By Chebyshev's Inequality, } \Pr\left[\left|\sum X_i - pm\right| \geq \delta m\right] \leq \frac{1}{4\delta^2 m}. \\
&\leq \frac{1}{4\epsilon(n)^2 T} = \frac{1}{2n}.
\end{aligned}$$

Then, completing the argument, we have

$$\begin{aligned}
&\Pr_{x,r}[\mathcal{B}(1^{2n}, g(x,r)) = (x,r)] \\
&\geq \Pr_x[x \in G] \Pr[x_1' = x_1, \cdots x_n' = x_n | x \in G] \\
&\geq \frac{\epsilon(n)}{2} \cdot \left(1 - \sum_{i=1}^{n} \Pr[x_i' \neq x_i | x \in G]\right) \\
&\geq \frac{\epsilon(n)}{2} \cdot \left(1 - n \cdot \frac{1}{2n}\right) = \frac{\epsilon(n)}{4}.
\end{aligned}$$

Real Case. Now, we describe the final case where $\Pr_{x,r}[E] \geq \frac{1}{2} + \epsilon(n)$, where $\epsilon(\cdot)$ is a non-negligible function. The key technical challenge in this case is that we cannot make two related calls to $\mathcal{A}$ as was done in the simple case above. However, just using one call to $\mathcal{A}$ seems insufficient. The key idea is to just guess one of those values. Very surprisingly this idea along with careful analysis magically works out. Just like the previous two case we start by describing the algorithm $\mathcal{B}$. On input $f(x)||R$, $\mathcal{B}$ proceeds as follows:

$T = \frac{2n}{\epsilon(n)^2}$
for $\ell = 1$ **to** $\log T$ **do**
  $s_\ell \xleftarrow{\$} \{0,1\}^n$
  $b_\ell \xleftarrow{\$} \{0,1\}$
for $i = 1$ **to** n **do**
  for all $L \subseteq \{1, 2, \cdots, \log T\}$ **do**
    $S_L := \bigoplus_{j \in L} s_j$
    $B_L := \bigoplus_{j \in L} b_j$
    $x_i^L \leftarrow B_L \oplus \mathcal{A}(f(x)||S_L + e^i)$
  $x_i' \leftarrow$ the majority of $\{x_i^{\emptyset}, \cdots, x_i^{[\log T]}\}$
return $x_1' \cdots x_n'||R$

The idea is the following. Let b_ℓ guess the value of $B(x, s_\ell)$, and with probability $\frac{1}{T}$ all the b_ℓ's are correct. In that case, it is easy to see that $B_L = B(x, S_L)$ for every L. If we follow the same argument as above, then it remains to bound the probability that $\mathcal{A}(f(x)||S_L + e^i) = B(x, S_L + e^i)$. However there is a subtle issue. Now the events $Y_i^{\emptyset}, \cdots, Y_i^{[\log T]}$ are not independent any more. But we can still show that they are pairwise independent, and the Chebyshev's Inequality still holds. Now we give the formal proof.
Just as in the simple case, we define the set G as

$$G := \left\{ x \,\middle|\, \Pr_r [E] \geq \frac{1}{2} + \frac{\epsilon(n)}{2} \right\},$$

and with an identical argument we obtain that

$$\Pr_{x \xleftarrow{\$} \{0,1\}^n} [x \in G] \geq \frac{\epsilon(n)}{2}.$$

Correctness of $\mathcal{B}$ follows from the fact in case $b_\ell = B(x, s_\ell)$ for every $\ell \in [\log T]$ then $\forall L \subseteq [\log T]$, it holds that (we use the notation $(s)_k$ to denote the k^{th} bit of s)

$$B(x, S_L) = \sum_{k=1}^{n} x_k \left(\bigoplus_{j \in L} s_j \right)_k = \sum_{k=1}^{n} x_k \sum_{j \in L} (s_j)_k = \sum_{j \in L} \sum_{k=1}^{n} x_k (s_j)_k = \sum_{j \in L} B(x, s_j) = \sum_{j \in L} b_j = B_L.$$

Next given that $b_\ell = B(x, s_\ell), \forall \ell \in [\log T]$ and $x \in G$ we bound the probability,

$$\begin{aligned}
\Pr_r \left[B_L \oplus \mathcal{A}(f(x)||S_L + e^i) = x_i \right] &= \Pr_r \left[B(x, S_L) \oplus \mathcal{A}(f(x)||S_L + e^i) = x_i \right] \\
&= \Pr_r \left[\mathcal{A}(f(x)||S_L + e^i) = B(x, S_L + e^i) \right] \\
&\geq \frac{1}{2} + \frac{\epsilon(n)}{2}.
\end{aligned}$$

For $b_\ell = B(x, s_\ell), \forall \ell \in [\log T]$ and $x \in G$, let Y_i^L be the indicator random variable that $x_i^L = x_i$. Notice that $Y_i^{\emptyset}, \cdots, Y_i^{[\log T]}$ are pairwise independent and $\Pr[Y_i^L = 1] = \Pr[x_i^L = x_i] \geq \frac{1}{2} + \frac{\epsilon(n)}{2}$.

$$\begin{aligned}
\Pr[x_i' \neq x_i] &= \Pr \left[\sum_{L \subseteq [\log T]} Y_i^L \leq \frac{T}{2} \right] \\
&= \Pr \left[\sum_{L \subseteq [\log T]} Y_i^L - \left(\frac{1}{2} + \frac{\epsilon(n)}{2} \right) T \leq \frac{T}{2} - \left(\frac{1}{2} + \frac{\epsilon(n)}{2} \right) T \right] \\
&\leq \Pr \left[\left| \sum_{L \subseteq [\log T]} Y_i^L - \left(\frac{1}{2} + \frac{\epsilon(n)}{2} \right) T \right| \geq \frac{\epsilon(n)}{2} T \right] \\
&\quad \text{(By Theorem 2.2)} \\
&\leq \frac{1}{4 \left(\frac{\epsilon(n)}{2} \right)^2 T} = \frac{1}{2n}.
\end{aligned}$$

Then, completing the proof, we have that

$$\begin{aligned}
&\Pr_{x,r}[\mathcal{B}(1^{2n}, g(x,r)) = (x,r)] \\
&\geq \Pr\left[\forall \ell \in [\log T], b_\ell = B(x, s_\ell)\right] \Pr_x[x \in G] \Pr[x'_1 = x_1, \cdots x'_n = x_n | \forall \ell \in [\log T], b_\ell = B(x, s_\ell), x \in G] \\
&\geq \frac{1}{T} \cdot \frac{\epsilon(n)}{2} \cdot \left(1 - \sum_{i=1}^{n} \Pr[x'_i \neq x_i | \forall \ell \in [\log T], b_\ell = B(x, s_\ell), x \in G]\right) \\
&\geq \frac{\epsilon(n)^2}{2n} \cdot \frac{\epsilon(n)}{2} \cdot \left(1 - n \cdot \frac{1}{2n}\right) = \frac{\epsilon(n)^3}{8n}.
\end{aligned}$$

■

Pairwise Independence and Chebyshev's Inequality. Here, for the sake of completeness, we prove the Chebyshev's Inequality.

Definition 2.2 (Pairwise Independence) *A collection of random variables $\{X_1, \cdots, X_m\}$ is said to be* pairwise independent *if for every pair of random variables $(X_i, X_j), i \neq j$ and every pair of values (v_i, v_j), it holds that*

$$\Pr[X_i = v_i, X_j = v_j] = \Pr[X_i = v_i] \Pr[X_j = v_j].$$

Theorem 2.2 (Chebyshev's Inequality) *Let $X_1, \ldots, X_m$ be pairwise independent and identically distributed binary random variables. In particular, for every $i \in [m]$, $\Pr[X_i = 1] = p$ for some $p \in [0,1]$ and $\Pr[X_i = 0] = 1 - p$. Then it holds that*

$$\Pr\left[\left|\sum_{i=1}^{m} X_i - pm\right| \geq \delta m\right] \leq \frac{1}{4\delta^2 m}.$$

Proof. Let $Y = \sum_i X_i$. Then

$$\begin{aligned}
\Pr\left[\left|\sum_{i=1}^{m} X_i - pm\right| > \delta m\right] &= \Pr\left[\left(\sum_{i=1}^{m} X_i - pm\right)^2 > \delta^2 m^2\right] \\
&\leq \frac{\mathbb{E}\left[|Y - pm|^2\right]}{\delta^2 m^2} \\
&= \frac{\mathrm{Var}(Y)}{\delta^2 m^2}
\end{aligned}$$

Observe that

$$\begin{aligned}
\mathrm{Var}(Y) &= \mathbb{E}\left[Y^2\right] - (\mathbb{E}[Y])^2 \\
&= \sum_{i=1}^{m} \sum_{j=1}^{m} \left(\mathbb{E}\left[X_i X_j\right] - \mathbb{E}\left[X_i\right] \mathbb{E}\left[X_j\right]\right) \\
&\text{By pairwise independence, for } i \neq j, \mathbb{E}\left[X_i X_j\right] = \mathbb{E}\left[X_i\right] \mathbb{E}\left[X_j\right]. \\
&= \sum_{i=1}^{m} \mathbb{E}\left[X_i^2\right] - \mathbb{E}\left[X_i\right]^2 \\
&= mp(1-p).
\end{aligned}$$

Hence

$$\Pr\left[\left|\sum_{i=1}^{m} X_i - pm\right| \geq \delta m\right] \leq \frac{mp(1-p)}{\delta^2 m^2} \leq \frac{1}{\delta^2 m}.$$

■

2.3 Computational Indistinguishability

What should it mean for a computationally bounded adversary to be able to distinguish two distributions from each other? It is tricky to define for a single pair of distributions because the length of the output of a random variable is a constant. Therefore, in order for "computationally bounded" adversaries to make sense, we have to work with infinite families of probability distributions.

Definition 2.3 *An* ensemble *of probability distributions is a sequence of random variables* $\{X_n\}_{n\in\mathbb{N}}$. *Two ensembles of probability distributions* $\{X_n\}_n$ *and* $\{Y_n\}_n$ *(which are samplable in time polynomial in* n*) are said to be* computationally indistinguishable *if for all (non-uniform) PPT machines* $\mathcal{A}$*, the quantities*

$$p(n) := \Pr[\mathcal{A}(1^n, X_n) = 1] = \sum_x \Pr[X_n = x] \Pr[\mathcal{A}(1^n, x) = 1]$$

and

$$q(n) := \Pr[\mathcal{A}(1^n, Y_n) = 1] = \sum_y \Pr[Y_n = y] \Pr[\mathcal{A}(1^n, y) = 1]$$

differ by a negligible amount; i.e. $|p(n) - q(n)|$ *is negligible in* n*. This equivalence is denoted by*

$$\{X_n\}_n \approx \{Y_n\}_n$$

We now prove some properties of computationally indistinguishable ensembles that will be useful later on.

Lemma 2.1 (Sunglass Lemma) *If* $\{X_n\}_n \approx \{Y_n\}_n$ *and* P *is a PPT machine, then*

$$\{P(X_n)\}_n \approx \{P(Y_n)\}_n$$

Proof. Consider an adversary $\mathcal{A}$ that can distinguish $\{P(X_n)\}_n$ from $\{P(Y_n)\}_n$ with non-negligible probability. Then the adversary $\mathcal{A} \circ P$ can distinguish $\{X_n\}_n$ from $\{Y_n\}_n$ with the same non-negligible probability. Since P and $\mathcal{A}$ are both PPT machines, the composition is also a PPT machine. This proves the contrapositive of the lemma. ■

Lemma 2.2 (Hybrid Argument) *For a polynomial* $t : \mathbb{Z}^+ \to \mathbb{Z}^+$ *let the* t*-product of* $\{Z_n\}_n$ *be*

$$\{Z_n^{(1)}, Z_n^{(2)}, \ldots, Z_n^{(t(n))}\}_n$$

where the $Z_n^{(i)}$*s are independent copies of* Z_n*. If*

$$\{X_n\}_n \approx \{Y_n\}_n$$

then

$$\{X_n^{(1)}, \ldots, X_n^{(t)}\}_n \approx \{Y_n^{(1)}, \ldots, Y_n^{(t)}\}_n$$

as well.

Proof. Consider the set of tuple random variables

$$H_n^{(i,t)} = (Y_n^{(1)}, \ldots, Y_n^{(i)}, X_n^{(i+1)}, X_n^{(i+2)}, \ldots, X_n^{(t)})$$

for integers $0 \leq i \leq t$. Assume, for the sake of contradiction, that there is a PPT adversary $\mathcal{A}$ that can distinguish between $\{H_n^{(0,t)}\}_n$ and $\{H_n^{(t,t)}\}_n$ with non-negligible probability difference $r(n)$. Suppose that $\mathcal{A}$ returns 1 with probability ϵ_i when it runs on samples from $H_n^{(i,t)}$. By definition, $|\epsilon_t - \epsilon_0| \geq r(n)$. By the Triangle Inequality and the Pigeonhole Principle, there is some index k for which $|\epsilon_{k+1} - \epsilon_k| \geq r(n)/t$. However, using Sunglass Lemma, note that the computational indistinguishability of X_n and Y_n implies that $\{H_n^{(k,t)}\}_n$ and $\{H_n^{(k+1,t)}\}_n$ are computationally indistinguishable. This is a contradiction. ■

2.4 Pseudorandom Generators

Now, we can define pseudorandom generators, which intuitively generates a polynomial number of bits that are indistinguishable from being uniformly random:

Definition 2.4 *A function $G : \{0,1\}^n \to \{0,1\}^{n+m}$ with $m = poly(n)$ is called a* pseudorandom generator *if*

- *G is computable in polynomial time.*
- *$U_{n+m} \approx G(U_n)$, where U_k denotes the uniform distribution on $\{0,1\}^k$.*

2.4.1 PRG Extension

In this section we show that any pseudorandom generator that produces one bit of randomness can be extended to create a polynomial number of bits of randomness.

Construction 2.1 *Given a PRG $G : \{0,1\}^n \to \{0,1\}^{n+1}$, we construct a new PRG $F : \{0,1\}^n \to \{0,1\}^{n+l}$ as follows (l is polynomial in n).*

(a) Input: $S_0 \xleftarrow{\$} \{0,1\}^n$.

(b) $\forall i \in [l] = \{1, 2, \cdots, l\}$, $(\sigma_i, S_i) := G(S_{i-1})$, where $\sigma_i \in \{0,1\}, S_i \in \{0,1\}^n$.

(c) Output: $\sigma_1 \sigma_2 \cdots \sigma_l S_l$.

Theorem 2.3 *The function F constructed above is a PRG.*

Proof. We prove this by hybrid argument. Define the hybrid H_i as follows.

(a) Input: $S_0 \xleftarrow{\$} \{0,1\}^n$.

(b) $\sigma_1, \sigma_2, \cdots, \sigma_i \xleftarrow{\$} \{0,1\}$, $S_i \leftarrow S_0$.
$\forall j \in \{i+1, i+2, \cdots, l\}$, $(\sigma_j, S_j) := G(S_{j-1})$, where $\sigma_j \in \{0,1\}, S_j \in \{0,1\}^n$.

(c) Output: $\sigma_1 \sigma_2 \cdots \sigma_l S_l$.

Note that $H_0 \equiv F$, and $H_l \equiv U_{n+l}$.

Assume for the sake of contradiction that there exits a non-uniform PPT adversary $\mathcal{A}$ that can distinguish H_0 form H_l. Define $\epsilon_i := \Pr[\mathcal{A}(1^n, H_i) = 1]$ for $i = 0, 1, \cdots, l$. Then there exists a non-negligible function $v(n)$ such that $|\epsilon_0 - \epsilon_l| \geq v(n)$. Since

$$|\epsilon_0 - \epsilon_1| + |\epsilon_1 - \epsilon_2| + \cdots + |\epsilon_{l-1} - \epsilon_l| \geq |\epsilon_0 - \epsilon_l| \geq v(n),$$

there exists $k \in \{0, 1, \cdots, l-1\}$ such that

$$|\epsilon_k - \epsilon_{k+1}| \geq \frac{v(n)}{l}.$$

l is polynomial in n, hence $\frac{v(n)}{l}$ is also a non-negligible function. That is to say, $\mathcal{A}$ can distinguish H_k from H_{k+1}. Then we use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that can distinguish U_{n+1} from $G(U_n)$ (which leads to a contradiction): On input $T \in \{0,1\}^{n+1}$ (T could be either from U_{n+1} or $G(U_n)$), $\mathcal{B}$ proceeds as follows:

- $\sigma_1, \sigma_2, \cdots, \sigma_k \xleftarrow{\$} \{0,1\}$, $(\sigma_{k+1}, S_{k+1}) \leftarrow T$.
- $\forall j \in \{k+2, k+3, \cdots, l\}$, $(\sigma_j, S_j) := G(S_{j-1})$, where $\sigma_j \in \{0,1\}, S_j \in \{0,1\}^n$.
- Output: $\mathcal{A}(1^n, \sigma_1\sigma_2\cdots\sigma_l S_l)$.

First, since $\mathcal{A}$ and G are both PPT computable, $\mathcal{B}$ is also PPT computable.

Second, if $T \leftarrow G(U_n)$, then $\sigma_1\sigma_2\cdots\sigma_l S_l$ is the output of H_k; if $T \xleftarrow{\$} U_{n+1}$, then $\sigma_1\sigma_2\cdots\sigma_l S_l$ is the output of H_{k+1}. Hence

$$\begin{aligned}
&\big|\Pr[\mathcal{B}(1^n, G(U_n)) = 1] - \Pr[\mathcal{B}(1^n, U_{n+1}) = 1]\big| \\
=&\big|\Pr[\mathcal{A}(1^n, H_k) = 1] - \Pr[\mathcal{A}(1^n, H_{k+1}) = 1]\big| \\
=&|\epsilon_k - \epsilon_{k+1}| \geq \frac{v(n)}{l}.
\end{aligned}$$

■

2.4.2 PRG from OWP (One-Way Permutations)

In this section we show how to construct pseudorandom generators under the assumption that one-way permutations exist.

Construction 2.2 *Let $f : \{0,1\}^n \to \{0,1\}^n$ be a OWP. We construct $G : \{0,1\}^{2n} \to \{0,1\}^{2n+1}$ as*

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

where $x, r \in \{0,1\}^n$, and $B(x, r)$ is a hard core bit for the function $g(x, r) = f(x)||r$.

Remark 2.3 *The hard core bit $B(x, r)$ always exists. Recall Theorem 2.1,*

$$B(x, r) = \left(\sum_{i=1}^{n} x_i r_i\right) \mod 2$$

is a hard core bit.

Theorem 2.4 *The G constructed above is a PRG.*

Proof. Assume for the sake of contradiction that G is not PRG. We construct three ensembles of probability distributions:

$$H_0 := G(U_{2n}) = f(x)||r||B(x,r), \text{ where } x, r \xleftarrow{\$} \{0,1\}^n;$$
$$H_1 := f(x)||r||\sigma, \text{ where } x, r \xleftarrow{\$} \{0,1\}^n, \sigma \xleftarrow{\$} \{0,1\};$$
$$H_2 := U_{2n+1}.$$

Since G is not PRG, there exists a non-uniform PPT adversary $\mathcal{A}$ that can distinguish H_0 from H_2. Since f is a permutation, H_1 is uniformly distributed in $\{0,1\}^{2n+1}$, i.e., $H_1 \equiv H_2$. Therefore, $\mathcal{A}$ can distinguish H_0 from H_1, that is, there exists a non-negligible function $v(n)$ satisfying

$$\big|\Pr[\mathcal{A}(H_0) = 1] - \Pr[\mathcal{A}(H_1) = 1]\big| \geq v(n).$$

Next we will construct an adversary $\mathcal{B}$ that "breaks" the hard core bit (which leads to a contradiction). Define a new ensemble of probability distribution

$$H_1' = f(x)||r||(1 - B(x,r)), \text{ where } x, r \xleftarrow{\$} \{0,1\}^n.$$

Then we have

$$\begin{aligned}\Pr[\mathcal{A}(H_1) = 1] =& \Pr[\sigma = B(x,r)] \Pr[A(H_0) = 1] + \Pr[\sigma = 1 - B(x,r)] \Pr[A(H_1') = 1] \\ =& \frac{1}{2} \Pr[A(H_0) = 1] + \frac{1}{2} \Pr[A(H_1') = 1].\end{aligned}$$

Hence

$$\begin{aligned}&\Pr[A(H_1) = 1] - \Pr[A(H_0) = 1] = \frac{1}{2} \Pr[A(H_1') = 1] - \frac{1}{2} \Pr[A(H_0) = 1], \\ &\frac{1}{2} \big|\Pr[A(H_0) = 1] - \Pr[A(H_1') = 1]\big| = |\Pr[A(H_1) = 1] - \Pr[A(H_0) = 1]| \geq v(n), \\ &\big|\Pr[A(H_0) = 1] - \Pr[A(H_1') = 1]\big| \geq 2v(n).\end{aligned}$$

Without loss of generality, we assume that

$$\Pr[A(H_0) = 1] - \Pr[A(H_1') = 1] \geq 2v(n).$$

Then we construct $\mathcal{B}$ as follows:

$$\mathcal{B}(f(x)||r) := \begin{cases} \sigma, & \text{if } \mathcal{A}(f(x)||r||\sigma) = 1 \\ 1 - \sigma, & \text{if } \mathcal{A}(f(x)||r||\sigma) = 0 \end{cases},$$

where $\sigma \xleftarrow{\$} \{0,1\}$. Then we have

$$\begin{aligned}&\Pr[\mathcal{B}(f(x)||r) = B(x,r)] \\ =& \Pr[\sigma = B(x,r)] \Pr[\mathcal{A}(f(x)||r||\sigma) = 1 | \sigma = B(x,r)] + \\ &\Pr[\sigma = 1 - B(x,r)] \Pr[\mathcal{A}(f(x)||r||\sigma) = 0 | \sigma = 1 - B(x,r)] + \\ =& \frac{1}{2}\big(\Pr[\mathcal{A}(f(x)||r||B(x,r)) = 1] + 1 - \Pr[\mathcal{A}(f(x)||r||1 - B(x,r)) = 1]\big) \\ =& \frac{1}{2} + \frac{1}{2}\big(\Pr[A(H_0) = 1] - \Pr[A(H_1') = 1]\big) \\ \geq& \frac{1}{2} + v(n).\end{aligned}$$

Contradiction with the fact that B is a hard core bit. ■

2.5 Pseudorandom Functions

In this section, we first define pseudorandom functions, and then show how to construct a pseudorandom function from a pseudorandom generator.

Considering the set of all functions $f : \{0,1\}^n \to \{0,1\}^n$, there are $(2^n)^{2^n}$ of them. To describe a random function in this set we need $n \cdot 2^n$ bits. Intuitively, a pseudorandom function is one that cannot be distinguished from a random one, but needs much fewer bits (e.g., polynomial in n) to be described. Note that we restrict the distinguisher to only being allowed to ask the function poly(n) times and decide whether it is random or pseudorandom.

2.5.1 Definitions

Definition 2.5 (Function Ensemble) *A* function ensemble *is a sequence of random variables $F_1, F_2, \cdots, F_n, \cdots$ denoted as $\{F_n\}_{n\in\mathbb{N}}$ such that F_n assumes values in the set of functions mapping n-bit input to n-bit output.*

Definition 2.6 (Random Function Ensemble) *We denote a random function ensemble by $\{R_n\}_{n\in\mathbb{N}}$.*

Definition 2.7 (Efficiently Computable Function Ensemble) *A function ensemble is called* efficiently computable *if*

(a) ***Succinct****: $\exists$ a PPT algorithm I and a mapping ϕ from strings to functions such that $\phi(I(1^n))$ and F_n are identically distributed. Note that we can view I as the description of the function.*

(b) ***Efficient****: $\exists$ a poly-time machine V such that $V(i,x) = f_i(x)$ for every $x \in \{0,1\}^n$, where i is in the range of $I(1^n)$, and $f_i = \phi(i)$.*

Definition 2.8 (Pseudorandom Function Ensemble) *A function ensemble $F = \{F_n\}_{n\in\mathbb{N}}$ is* pseudorandom *if for every non-uniform PPT oracle adversary $\mathcal{A}$, there exists a negligible function $\epsilon(n)$ such that*

$$\left|\Pr[\mathcal{A}^{F_n}(1^n) = 1] - \Pr[\mathcal{A}^{R_n}(1^n) = 1]\right| \leq \epsilon(n).$$

Here by saying "oracle" it means that $\mathcal{A}$ has "oracle access" to a function (in our definition, the function is F_n or R_n), and each call to that function costs 1 unit of time.

Note that we will only consider efficiently computable pseudorandom ensembles in the following.

2.5.2 Construction of PRF from PRG

Construction 2.3 *Given a PRG $G : \{0,1\}^n \to \{0,1\}^{2n}$, let $G_0(x)$ be the first n bits of $G(x)$, $G_1(x)$ be the last n bits of $G(x)$. We construct $F^{(K)} : \{0,1\}^n \to \{0,1\}^n$ as follows.*

$$F_n^{(K)}(x_1 x_2 \cdots x_n) := G_{x_n}(G_{x_{n-1}}(\cdots(G_{x_1}(K))\cdots)),$$

where $K \in \{0,1\}^n$ is the key to the pseudorandom function. Here i is an n-bit string, which is the seed of the pseudorandom function.

The construction can be viewed as a binary tree of depth n, as shown in Figure 2.1.

Theorem 2.5 *The function ensemble $\{F_n\}_{n\in\mathbb{N}}$ constructed above is pseudorandom.*

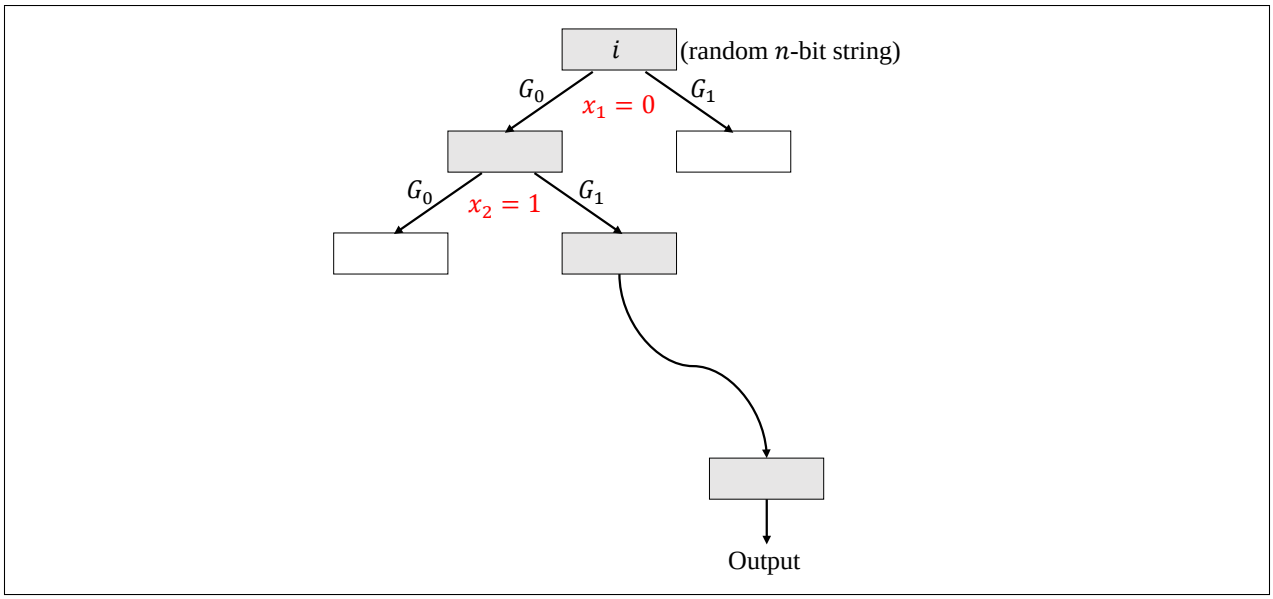


Figure 2.1: View the construction as a binary tree

Proof. Assume for the sake of contradiction that $\{F_n\}_{n\in\mathbb{N}}$ is not PRG. Then there exists a non-uniform PPT oracle adversary $\mathcal{A}$ that can distinguish $\{F_n\}_{n\in\mathbb{N}}$ from $\{R_n\}_{n\in\mathbb{N}}$. Below, via a hybrid argument, we prove that this contradicts the fact that G is a PRG.

Consider the sequence of hybrids H_i for $i \in \{0, 1, \cdots, n\}$ where the hybrid i is defined as follows:

$$H_{n,i}^{(K)}(x_1 x_2 \dots x_n) := G_{x_n}(G_{x_{n-1}}(\cdots(G_{x_{i+1}}(K(x_i x_{i-1} \dots x_1)))\cdots)),$$

where K is a random function from $\{0,1\}^i$ to $\{0,1\}^n$. Intuitively, hybrid H_i corresponds to a binary tree of depth n where the nodes of levels 0 to i correspond to random values and the nodes at levels $i+1$ to n correspond to pseudorandom values. By inspection, observe that hybrids H_0 and H_n are identical to a pseudorandom function and a random function, respectively. There it suffices to prove that hybrids H_i and H_{i+1} are computationally indistinguishable for each $i \in \{0, 1, \cdots, n\}$.

We show that H_i and H_{i+1} are indistinguishable by considering a sequence of sub-hybrids $H_{i,j}$ for $j \in \{0, \dots q_{i+1}\}$, where q_{i+1} is the number of the distinct $i - bit$ prefixes of the queries of $\mathcal{A}$.[1]

We define hybrid $H_{i,j}$ for $j = 0$ to be same as hybrid H_i. Additionally, for $j > 0$ hybrid $H_{i,j}$ is defined to be exactly the same as hybrid $H_{i,j-1}$ except the response provided to the attacker for the j^{th} distinct $i - bit$ prefix query of $\mathcal{A}$. Let this prefix be $x_n^* x_{n-1}^* \dots x_i^*$. Note that in hybrid $H_{i,j-1}$ the children of the node $x_n^* x_{n-1}^* \dots x_i^*$ correspond to two pseudorandom values. In hybrid $H_{i,j}$ we replace these two children with random values. By careful inspection, it follows that hybrid $H_{i,q_{i+1}}$ is actually H_{i+1}. All we are left to prove is that hybrid $H_{i,j}$ and $H_{i,j+1}$ are indistinguishable for the appropriate choices of j and we prove this below.

Now we are ready to construct an adversary $\mathcal{B}$ that distinguishes U_{2n} from $G(U_n)$: On input $T \in \{0,1\}^{2n}$ (T could be either from U_{2n} or $G(U_n)$), construct a full binary tree of depth n that is exactly the same as $H_{i,j}$ except replacing the children of $x_n^* x_{n-1}^* \dots x_i^*$ by the value T. Observe that the only difference between $H_{i,j}$ and $H_{i,j+1}$ is that values corresponding to nodes $x_n^* \dots x_i^* 0$ and $x_n^* \dots x_i^* 1$ are pseudorandom or random respectively. $\mathcal{B}$ uses the value T to generate these two nodes. Hence success in distinguishing hybrids $H_{i,j}$ and $H_{i,j+1}$ provides a successful attack for $\mathcal{B}$ in violating security of the pseudorandom generator. ■

[1] Observe that q_{i+1} for each appropriate choice of i is bounded by the running time of $\mathcal{A}$. Hence, this value is bounded by a polynomial in the security parameter.

Exercises

Exercise 2.1 *Prove or disprove: If f is a one-way function, then the following function $B : \{0,1\}^* \to \{0,1\}$ is a hardcore predicate for f. The function $B(x)$ outputs the inner product modulo 2 of the first $\lfloor |x|/2 \rfloor$ bits of x and the last $\lfloor |x|/2 \rfloor$ bits of x.*

Exercise 2.2 *Let $\phi(n)$ denote the first n digits of $\pi = 3.141592653589\ldots$ after the decimal in binary (π in its binary notation looks like $11.001001000011111101101010100010001000010\ldots$).*

Prove the following: if one-way functions exist, then there exists a one-way function f such that the function $B : \{0,1\}^* \to \{0,1\}$ is not a hard core bit of f. The function $B(x)$ outputs $\langle x, \phi(|x|)\rangle$, where

$$\langle a, b\rangle := \sum_{i=1}^{n} a_i b_i \mod 2$$

for the bit-representation of $a = a_1 a_2 \cdots a_n$ and $b = b_1 b_2 \cdots b_n$.

Exercise 2.3 *If $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ is PRF, then in which of the following cases is $g : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ also a PRF?*

1. $g(K, x) = f(K, f(K, x))$
2. $g(K, x) = f(x, f(K, x))$
3. $g(K, x) = f(K, f(x, K))$

Exercise 2.4 (Puncturable PRFs.) *Puncturable PRFs are PRFs for which a key can be given out such that, it allows evaluation of the PRF on all inputs, except for one designated input.*

A puncturable pseudo-random function F is given by a triple of efficient algorithms (Key_F,$\mathsf{Puncture}_F$, and Eval_F), satisfying the following conditions:

- ***Functionality preserved under puncturing**: For every $x^*, x \in \{0,1\}^n$ such that $x^* \neq x$, we have that:*

$$\Pr[\mathsf{Eval}_F(K, x) = \mathsf{Eval}_F(K_{x^*}, x) : K \leftarrow \mathsf{Key}_F(1^n), K_{x^*} = \mathsf{Puncture}_F(K, x^*)] = 1$$

- ***Pseudorandom at the punctured point**: For every $x^* \in \{0,1\}^n$ we have that for every polysize adversary $\mathcal{A}$ we have that:*

$$|\Pr[\mathcal{A}(K_{x^*}, \mathsf{Eval}_F(K, x^*)) = 1] - \Pr[\mathcal{A}(K_{x^*}, \mathsf{Eval}_F(K, U_n)) = 1]| = \mathsf{negl}(n)$$

where $K \leftarrow \mathsf{Key}_F(1^n)$ and $K_S = \mathsf{Puncture}_F(K, x^*)$. U_n denotes the uniform distribution over n bits.

Prove that: If one-way functions exist, then there exists a puncturable PRF family that maps n bits to n bits.

***Hint:** The GGM tree-based construction of PRFs from a length doubling pseudorandom generator (discussed in class) can be adapted to construct a puncturable PRF. Also note that K and K_{x^*} need not be the same length.*

Chapter 3

Digital Signatures

We introduce digital signatures and present a one-time secure digital signature scheme. We then present collision resistant hashes, and show how they can be used to extend the digital signature scheme to many-time secure.

3.1 Definition

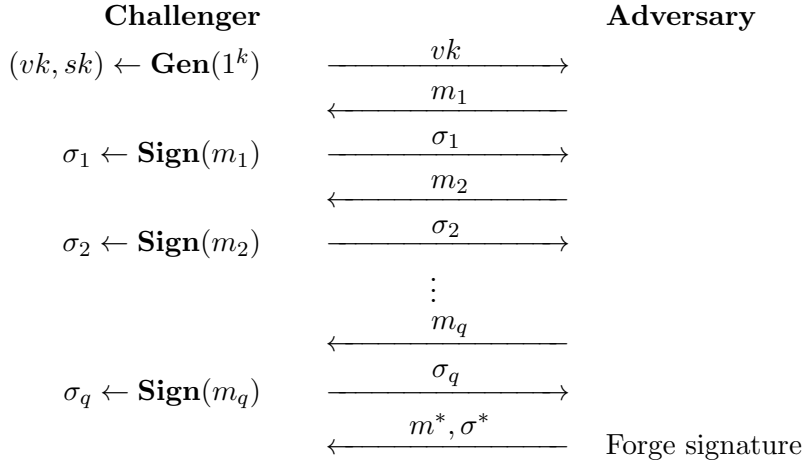
A digital signature scheme consists of a three functions:

1. $\mathbf{Gen}(1^k) \to (vk, sk)$
2. $\mathbf{Sign}(sk, m) \to \sigma$
3. $\mathbf{Verify}(vk, m, \sigma) \to \{0, 1\}$

Correctness. For the correctness of the scheme, we have that $\forall m, vk, sk$ where $vk, sk \leftarrow \mathbf{Gen}(1^k)$,

$$\Pr\left[\sigma \leftarrow \mathbf{Sign}(sk, m) : \mathbf{Verify}(vk, m, \sigma) = 1\right] = 1.$$

Security: CMA (chosen message attack) with existential forger. Here we consider a typical challenger, adversary game where the adversary can choose the message it is forging. We use this game to evaluate a digital signature scheme in Section 3.2.



A wins if $\mathbf{Ver}(vk, m^*, \sigma^*) = 1 \quad$ and $\forall i \in [q], m^* \neq m_i$.

3.2 Digital Signature for q = 1

We now consider a one-time secure digital signature scheme built using one way functions.

- **Gen**(1^k):
  1. Sample $x_i^b \leftarrow \{0,1\}^k, \forall i \in [\ell], b \in \{0,1\}$
  2. $vk = \begin{bmatrix} f(x_1^0) & \dots & f(x_\ell^0) \\ f(x_1^1) & \dots & f(x_\ell^1) \end{bmatrix}$
  3. $sk = \begin{bmatrix} x_1^0 & \dots & x_\ell^0 \\ x_1^1 & \dots & x_\ell^1 \end{bmatrix}$
- **Sign**$(sk, m \in \{0,1\}^\ell)$: $m = \overline{m_1 \dots m_\ell} \rightarrow \sigma = (x_1^{m_1}, \dots, x_\ell^{m_\ell})$
- **Ver**(vk, m, σ) : compute $f(x_{m_i})$ for all bits in the message and compare to vk

To show that this is secure, we can prove that cracking this scheme is equivalent to inverting the one way function. Since the challenger puts $f(x)$ in the verifying key, to forge a signature the attacker needs to break the one way function.

We still have two problems: this scheme is only one-time secure, and the scheme only works for small messages. The latter problem arises because the length of the verifying key is equal to the length of the message.

3.3 Collision Resistant Hash Functions

We can resolve both of these problems by signing a *hash* of the message, rather than the message itself. If this hash is collision resistant, it has the benefit that even if both messages are chosen, it is still extremely difficult to find a colliding hash, to get the same signature, namely the hash function $H : \{0,1\}^* \rightarrow \{0,1\}^k$ has the property that it is hard to find x, x' where $H(x) = H(x')$.

3.3.1 Definition of a family of CRHF

A set of functions

$$H = \{h_i : D_i \rightarrow R_i\}_{i \in I}$$

is a family of collision resistant hash functions if

1. **Gen**$(1^k) \rightarrow i \in I$ (the description of i is poly(k))
2. $|R_i| < |D_i|$
3. $B(i, x) = h_i(x)$ (B is a PPT algorithm)
4. $\forall$ PPT A we have

$$\Pr[i \leftarrow \textbf{Gen}(1^k), (x, x') \leftarrow A(1^k, i) : h_i(x) = h_i(x') \wedge x \neq x'] = neg(k)$$

3.3.2 Discrete log CRHF

Let G be an order p subgroup of Z_q^* where p and q are primes. Let g be the generator of G. Given a random element $h \in G$, it is hard to find x such that $g^x = h$. That is, for every PPT A,

$$\Pr[(G, g, p) \leftarrow \mathbf{Gen}(1^k), x \xleftarrow{\$} Z_p^* : x = A(q, p, g^x)] = neg(k).$$

We can construct a CRHF that halves the size of the input:

1. $(G, g, p) \leftarrow \mathbf{Gen}(1^k)$.
2. $x \leftarrow Z_p^*$.
3. $h = g^x$.
4. $H : \{0,1\}^{2k} \rightarrow \{0,1\}^k$ is defined as $H(x, r) = g^x h^r$ where $x, r \in \{0,1\}^k$.

To reduce an arbitrary size message into the size that can be signed, one can construct a tree of length-halving hashes. Break the message into n $2k$ size chunks, and hash each of those. Concatenate them and rechunk them, yielding $n/2$ $2k$ size chunks. Repeat until the only result is a single k size chunk.

3.4 Multiple-Message Digital Signature

Armed with a way to reduce the message to a fixed size, we can now sign messages that contain verifying keys (previously, the size of the verifying key was dependent on the size of the message, so if the message contained a verifying key we were in trouble).

The signing party can generate a new verifying key and signing key with each new message it sends. The new verifying key is concatenated with the message, and is under the signature. In this way, each key is only used once, but along with the message, a new verifying key is sent to the verifying party.

Exercises

Exercise 3.1 ***Digital signature schemes can be made deterministic.*** *Given a digital signature scheme* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ *for which* Sign *is probabilistic, provide a construction of a digital signature scheme* $(\mathsf{Gen}', \mathsf{Sign}', \mathsf{Verify}')$ *where* Sign' *is deterministic.*

Chapter 4

Public Key Encryption

Public key encryption deals with a setting where there are two parties who wish to communicate a secret message from one of them to the other. Unlike the symmetric setting, in which the two parties share a secret key, the public-key setting has asymmetry in who can decrypt a given message. This allows one party to announce the public key to everyone so messages can be encrypted, but keep the secret key private so no one else can preform decryption.

Definition 4.1 (Public Key Encryption) *A public key crypto-system consists of three algorithms:* Gen, Enc, *and* Dec *with properties as follows:*

1. $\mathsf{Gen}(1^k)$ *outputs a pair of keys* $(\mathsf{pk}, \mathsf{sk})$*; the public and private keys respectively.*
2. $\mathsf{Enc}(\mathsf{pk}, m)$ *encrypts a message* m *under public key* pk.
3. $\mathsf{Dec}(\mathsf{sk}, c)$ *decrypts a ciphertext* c *under secret key* sk.

There are other properties about these algorithms which we will discuss next in order for these algorithms to be useful. The first of these, correctness, ensures that the decryption of an encrypted message returns the original plaintext. The second is the security property, which says that an attacker with access to the encrypted message learns nothing about the plaintext.

Note that the Gen algorithm must be a randomized algorithm. If not, it would always output the same public-private key pair, and would not be very useful. We will later show that Enc must also be a randomized algorithm, or else the security properties will not hold. Finally, Dec may be a randomized algorithm but is not required to be one.

4.1 Correctness

In order for the encryption and decryption to satisfy our intuition of what these algorithms should do, we require that decrypting an encrypted value with the correct keys yields the original message.

Definition 4.2 (Correctness) *An public key algorithm* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is* correct *if*

$$\forall m.\, \Pr[\mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, m)) = m | (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^k)] = 1$$

Some definitions may relax this constraint from being equal to one to being greater than $1 - neg(\cdot)$. However, since this probability is equal to one, we can restate the definition

Lemma 4.1 (Correctness) *An public key algorithm* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is* correct *if*

$$\forall m, \mathsf{pk}, \mathsf{sk}.(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^k) \implies \mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, m)) = m.$$

There is no statement involved about what happens when encrypting or decrypting under the wrong key, nor is there anything about decrypting a malformed ciphertext. It only requires that decryption of an encrypted message under corresponding keys produces the original message.

4.2 Indistinguishability and Semantic Security

Not only must public key encryption be correct, we would also like it to hide the values which are encrypted. There are two different definitions which we will show are identical.

4.2.1 Indistinguishability Security

Our first definition, indistinguishability, states that the ciphertexts obtained from encrypting any message must look identical to those from encrypting any other message. In particular this implies that encryption must be a randomized algorithm.

Definition 4.3 (Indistinguishability) *A public key encryption scheme satisfies the indistinguishability property if the distributions* A_{m_1} *and* A_{m_2} *are computationally indistinguishable for all* $m_1, m_2 \in M$ *such that* $|m_1| = |m_2|$ *where*

$$A_{m_1} = \{\mathsf{pk}, \mathsf{Enc}(\mathsf{pk}, m_1) : (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^k)\}_k$$
$$A_{m_2} = \{\mathsf{pk}, \mathsf{Enc}(\mathsf{pk}, m_2) : (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^k)\}_k$$

It is required that we talk about computational indistinguishability. The encryption of a message must reveal at least something in an information-theoretic setting. It is also required that the sizes of the two messages be equal. It would be very easy to tell the difference between the encryption of a one-bit message and an arbitrarily large message by just comparing their sizes. It is possible to work around this requirement by having an encryption scheme pad messages so that they are all of equal size if an upper bound is known.

4.2.2 Semantic Security

An alternate definition, which we will later prove is identical, is semantic security. This definition intuitively states that given a ciphertext, you should learn nothing about the original message other than it's length.

Definition 4.4 (Semantic Security) *An encryption scheme is semantically secure if there exists a simulator* S *such that the two following processes generate computationally indistinguishable outputs.*

1. $(m, z) \leftarrow M(1^k)$		1. $(m, z) \leftarrow M(1^k)$
2. $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^k)$	$\approx$	2. *Output* $S^A(1^k, z)$
3. *Output* $A(\mathsf{pk}, \mathsf{Enc}(\mathsf{pk}, m), z)$		

Where M *is a machine that randomly samples message from the message space and arbitrary additional information and* A *is the adversary.*

4.2.3 Equivalence of Definitions

Theorem 4.1 (Equivalence of Definitions) *A public key encryption scheme (gen, enc, dec) is semantically secure if and only if it satisfies the indistinguishability property.*

Proof. We begin by proving that semantic security implies indistinguishability. That is, we need to show that for every PPT adversary A, A_{m_1} and A_{m_2} are computationally indistinguishable for every pair of $m_1, m_2 \in M$. We prove by a hybrid argument. In hybrid H_0 let M always output the value m_1, feed into A and output $A(\mathsf{pk}, \mathsf{Enc}(\mathsf{pk}, m_1))$. In hybrid H_1 replace the output by $S^A(1^k)$. In hybrid H_2 let M always output the value m_2, feed into A and output $A(\mathsf{pk}, \mathsf{Enc}(\mathsf{pk}, m_2))$. By semantic security the output of these hybrids are computationally indistinguishable, hence A cannot distinguish A_{m_1} and A_{m_2}.

Now we prove the other direction. The simulator generates a public/secret key pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^k)$, encrypts the message $0^{|m|}$ and feeds into A. It outputs $A(\mathsf{pk}, \mathsf{Enc}(\mathsf{pk}, 0^{|m|}), z)$. By the indistinguishability property, A cannot distinguish $(\mathsf{pk}, \mathsf{Enc}(\mathsf{pk}, 0^{|m|}))$ from $(\mathsf{pk}, \mathsf{Enc}(\mathsf{pk}, m))$, hence the outputs of A and S are computationally indistinguishable. ■

4.3 Public Key Encryption from Trap-Door OWP

We now show how it is possible to create a public key encryption scheme from a trapdoor one way permutation.

Theorem 4.2 *Let (G, F, A) be a trap-door one-way permutation The following is then a public-key encryption scheme:*

1. $\mathsf{Gen}(1^k) = (i, t_i) \leftarrow G(1^k)$.
2. $\mathsf{Enc}(\mathsf{pk}, m) = (F_i(x)||r, B(x, r) \oplus m)$ *where (x, r) are sampled uniformly, and $B(\cdot)$ is a hard-core bit.*
3. $\mathsf{Dec}(\mathsf{sk}, c) = B(A(i, t_i, y), r) \oplus c_0$ *where $c = (y||r, c_0)$.*

Proof. Clearly the encryption scheme is correct, since $\alpha \oplus \alpha \oplus m = m$ for any α.

Now we prove that the scheme is indistinguishable by showing that $\mathsf{Enc}(\mathsf{pk}, 0)$ and $\mathsf{Enc}(\mathsf{pk}, 1)$ are computationally indistinguishable. Note that distinguishing the two distributions is by definition equivalent to guessing the hard-core bit. Hence they are indistinguishable. ■

4.4 Indistinguishability in a Chosen Plaintext Attack

We now define a new security requirement, IND-CPA, which is stronger than indistinguishability or semantic security. Consider the following scheme:

The challenger samples $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^k)$ and gives pk to the attacker. The attacker then replis with two messages (m_0, m_1). The challenger then picks one of these uniformly at random, encrypts it generating $c = \mathsf{Enc}(\mathsf{pk}, m_b)$ and sends it to the attacker. The attacker must then guess which message was encrypted.

Here, the attacker gets to choose two messages which he likes, ask the challenger to encrypt the two messages, and only must be able to distinguish between the two.

Then a scheme is IND-CPA if $\forall A. \Pr[\text{Experiment}^{A(1^k)}] < 1/2 + neg(k)$ where A is a PPT machine. That is, an attacker which can ask the user to encrypt specific messages after learning the

public keys can still not learn anything about the values of the encrypted messages. In particular, there can not be any easy-to-identify weak messages for a given public key.

Definition 4.5 *A public key encryption scheme (PKE) given by the three efficient procedures (G, E, D) is IND-CPA-secure if no adversary A has a significant advantage in the game represented in Table 4.1.*

Challenger		Adversary
$(\mathsf{pk}, \mathsf{sk}) \leftarrow G(1^k)$	$\xrightarrow{\mathsf{pk}}$	
	$\xleftarrow{m'_0, m'_1}$	
$b \xleftarrow{\$} \{0,1\}$		
$c^* = E(\mathsf{pk}, m'_b)$	$\xrightarrow{c^*}$	
	$\xleftarrow{b'}$	
Output 1 if $(b' = b)$		

Table 4.1: CPA security.

This means that every probabilistic polynomial-time (PPT) adversary A has only a negligible advantage (over a random guess) when guessing which message (m'_0 or m'_1) the challenger used to obtain c^*.

Note that, since the adversary has the public key pk*, he is able to encrypt any polynomial number of plaintexts during the game.*

Theorem 4.3 *IND-CPA is a stronger definition than semantic security.*

Proof. First, observe that IND-CPA is at least as strong as semantic security. An attack which showed an encryption scheme is not semantically secure can be used identically to distinguish the two messages the attacker sends in the IND-CPA protocol.

Now we only need to show that it is stronger. In the following we construct an encryption scheme which is semantically secure, but is not secure under IND-CPA. Assume we have a scheme (G, E, D) which is semantically secure. Now we will construct a new one (G', E', D') which is still semantically secure, but is definitely not IND-CPA.

$$G'(1^k) = ((\mathsf{pk}, x_0, x_1), \mathsf{sk}), \text{ where } (\mathsf{pk}, \mathsf{sk}) \leftarrow G(1^k), x_0, x_1 \leftarrow M(1^k)$$
$$E'((\mathsf{pk}, x_0, x_1), m) = \text{if } m \notin \{x_0, x_1\}, \text{ then } (\mathsf{y}, E(\mathsf{pk}, m)) \text{ else } (\mathsf{n}, m)$$
$$D'(\mathsf{sk}, c) = \text{if } c \text{ starts with } \mathsf{y}, \text{then } D(\mathsf{sk}, E(\mathsf{pk}, m)) \text{ else } m$$

This scheme is still semantically secure. E' is different from E on only two inputs, which is certainly negligible in k. However, this scheme is not IND-CPA. After seeing the public key pair (pk, x_0, x_1), the attacker knows exactly to pick values x_0 and x_1 will be able to distinguish between those two with probability 1. ■

4.5 Chosen Ciphertext Attack for Public Key Encryption

Definition 4.6 *A public key encryption scheme (PKE) given by the three efficient procedures (G, E, D) is IND-CCA1-secure if no adversary A has a significant advantage in the game represented in Table 4.2.*

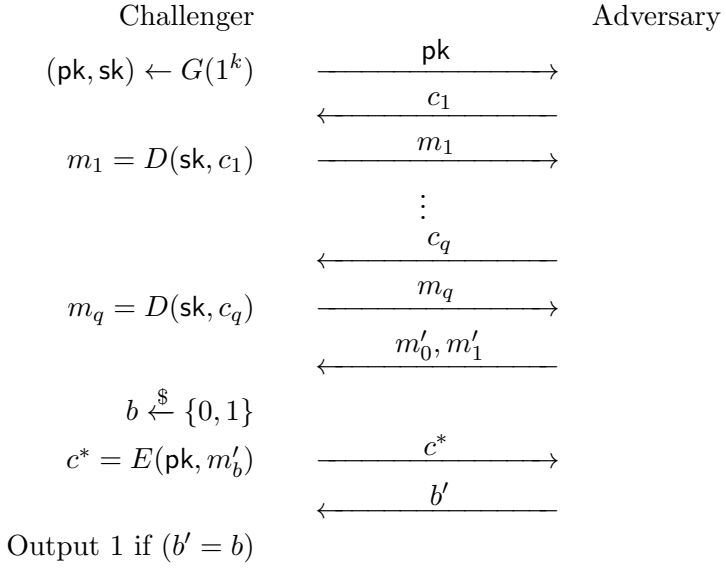


Table 4.2: Non-adaptive CCA security.

In this definition the adversary may send some polynomial number of queries to be decrypted before he receives the challenge ciphertext c^*.

Definition 4.7 *A public key encryption scheme (PKE) given by the three efficient procedures (G, E, D) is IND-CCA2-secure if no adversary A has a significant advantage in the game represented in Table 4.3.*

Note that, in this adaptive version, the adversary is able to send more queries to the challenger even after having seen the challenge ciphertext c^*. The only thing we require is that he does not pass the challenge ciphertext c^* itself for those queries.

Theorem 4.4 *Given an IND-CPA-secure public key encryption scheme (G, E, D), it is possible to construct an IND-CCA1-secure public key encryption scheme (G', E', D').*

Proof. Let $(\mathsf{pk}_1, \mathsf{sk}_1) \leftarrow G(1^k)$, $(\mathsf{pk}_2, \mathsf{sk}_2) \leftarrow G(1^k)$ be two pairs of keys generated by the IND-CPA scheme (G, E, D). We claim that there is a NIZK proof system that is able to prove that c_1 and c_2 are ciphertexts obtained by the encryption of the same message m under the keys pk_1 and pk_2, respectively.

More precisely, we claim that there is a NIZK proof system for the language

$$L = \{(c_1, c_2) \mid \exists r_1, r_2, m \text{ such that } c_1 = E(\mathsf{pk}_1, m; r_1) \text{ and } c_2 = E(\mathsf{pk}_2, m; r_2)\},$$

where $E(\mathsf{pk}_i, m; r_i)$ represents the output of $E(\mathsf{pk}_i, m)$ when the random coin flips of the procedure E are given by r_i.

The language L is clearly in NP, since for every $x = (c_1, c_2) \in L$ there is a witness $w = (r_1, r_2, m)$ that proves that $x \in L$. Given x and w, there is an efficient procedure to verify if w is a witness to the fact that $x \in L$.

By Theorem **??**, there is a NIZK proof system for any language in NP, so our claim holds. Let this NIZK proof system be given by the procedures (K, P, V). We can assume that this is an adaptive NIZK, because it is always possible to construct an adaptive NIZK from a non-adaptive NIZK proof system. Then we define our public key encryption scheme (G', E', D') as follows:

Challenger		Adversary
$(\mathsf{pk}, \mathsf{sk}) = G(1^k)$	$\xrightarrow{\mathsf{pk}}$	
	$\xleftarrow{c_1}$	
$m_1 = D(\mathsf{sk}, c_1)$	$\xrightarrow{m_1}$	
	$\vdots$	
	$\xleftarrow{c_q}$	
$m_q = D(\mathsf{sk}, c_q)$	$\xrightarrow{m_q}$	
	$\xleftarrow{m'_0, m'_1}$	
$b \xleftarrow{\$} \{0,1\}$		
$c^* = E(\mathsf{pk}, m'_b)$	$\xrightarrow{c^*}$	
	$\xleftarrow{c_{q+1}}$	$c_{q+1} \neq c^*$
$m_{q+1} = D(\mathsf{sk}, c_{q+1})$	$\xrightarrow{m_{q+1}}$	
	$\vdots$	
	$\xleftarrow{c_{q+l}}$	$c_{q+l} \neq c^*$
$m_{q+l} = D(\mathsf{sk}, c_{q+l})$	$\xrightarrow{m_{q+l}}$	
	$\xleftarrow{b'}$	
Output 1 if $(b' = b)$		

Table 4.3: Adaptive CCA security.

$G'(1^k)$: $(\mathsf{pk}_1, \mathsf{sk}_1) \leftarrow G(1^k)$
$(\mathsf{pk}_2, \mathsf{sk}_2) \leftarrow G(1^k)$
$\sigma \leftarrow K(1^k)$
let $\mathsf{pk}' = (\mathsf{pk}_1, \mathsf{pk}_2, \sigma)$ and $\mathsf{sk}' = \mathsf{sk}_1$
return $(\mathsf{pk}', \mathsf{sk}')$

$E'(\mathsf{pk}', m)$: let $(\mathsf{pk}_1, \mathsf{pk}_2, \sigma) = \mathsf{pk}'$
$c_1 \leftarrow E(\mathsf{pk}_1, m; r_1)$
$c_2 \leftarrow E(\mathsf{pk}_2, m; r_2)$
let $x = (c_1, c_2)$ // statement to prove
let $w = (r_1, r_2, m)$ // witness for the statement
$\pi \leftarrow P(\sigma, x, w)$
let $c = (c_1, c_2, \pi)$
return c

$D'(\mathsf{pk}', c)$: let $\mathsf{sk}_1 = \mathsf{sk}'$
let $(c_1, c_2, \pi) = c$
let $x = (c_1, c_2)$
if $V(\sigma, x, \pi)$
then return $D(\mathsf{sk}_1, c_1)$
else return $\perp$

Game 0	Game 1
$(\mathsf{pk}_1, \mathsf{sk}_1), (\mathsf{pk}_2, \mathsf{sk}_2) \leftarrow G(1^k)$ $\sigma \leftarrow K(1^k)$ let $\mathsf{pk}' = (\mathsf{pk}_1, \mathsf{pk}_2, \sigma)$ and $\mathsf{sk}' = \mathsf{sk}_1$ $(m_0, m_1) \leftarrow A^{D'(\mathsf{sk}', \cdot)}(\mathsf{pk}')$ $c_1 \leftarrow E(\mathsf{pk}_1, m_0; r_1), c_2 \leftarrow E(\mathsf{pk}_2, m_0; r_2)$ $\pi \leftarrow P(\sigma, (c_1, c_2), (r_1, r_2, m_0))$ $b \leftarrow A(\mathsf{pk}', c_1, c_2, \pi)$	$(\mathsf{pk}_1, \mathsf{sk}_1), (\mathsf{pk}_2, \mathsf{sk}_2) \leftarrow G(1^k)$ $\sigma \leftarrow Sim(1^k)$ let $\mathsf{pk}' = (\mathsf{pk}_1, \mathsf{pk}_2, \sigma)$ and $\mathsf{sk}' = \mathsf{sk}_1$ $(m_0, m_1) \leftarrow A^{D'(\mathsf{sk}', \cdot)}(\mathsf{pk}')$ $c_1 \leftarrow E(\mathsf{pk}_1, m_0; r_1), c_2 \leftarrow E(\mathsf{pk}_2, m_0; r_2)$ $\pi \leftarrow Sim(c_1, c_2)$ $b \leftarrow A(\mathsf{pk}', c_1, c_2, \pi)$
Game 2	Game 2'
$(\mathsf{pk}_1, \mathsf{sk}_1), (\mathsf{pk}_2, \mathsf{sk}_2) \leftarrow G(1^k)$ $\sigma \leftarrow Sim(1^k)$ let $\mathsf{pk}' = (\mathsf{pk}_1, \mathsf{pk}_2, \sigma)$ and $\mathsf{sk}' = \mathsf{sk}_1$ $(m_0, m_1) \leftarrow A^{D'(\mathsf{sk}', \cdot)}(\mathsf{pk}')$ $c_1 \leftarrow E(\mathsf{pk}_1, m_0; r_1), c_2 \leftarrow E(\mathsf{pk}_2, m_1; r_2)$ $\pi \leftarrow Sim(c_1, c_2)$ $b \leftarrow A(\mathsf{pk}', c_1, c_2, \pi)$	$(\mathsf{pk}_1, \mathsf{sk}_1), (\mathsf{pk}_2, \mathsf{sk}_2) \leftarrow G(1^k)$ $\sigma \leftarrow Sim(1^k)$ let $\mathsf{pk}' = (\mathsf{pk}_1, \mathsf{pk}_2, \sigma)$ and $\mathsf{sk}' = \mathsf{sk}_2$ $(m_0, m_1) \leftarrow A^{D'(\mathsf{sk}', \cdot)}(\mathsf{pk}')$ $c_1 \leftarrow E(\mathsf{pk}_1, m_0; r_1), c_2 \leftarrow E(\mathsf{pk}_2, m_1; r_2)$ $\pi \leftarrow Sim(c_1, c_2)$ $b \leftarrow A(\mathsf{pk}', c_1, c_2, \pi)$
Game 3	Game 3'
$(\mathsf{pk}_1, \mathsf{sk}_1), (\mathsf{pk}_2, \mathsf{sk}_2) \leftarrow G(1^k)$ $\sigma \leftarrow Sim(1^k)$ let $\mathsf{pk}' = (\mathsf{pk}_1, \mathsf{pk}_2, \sigma)$ and $\mathsf{sk}' = \mathsf{sk}_2$ $(m_0, m_1) \leftarrow A^{D'(\mathsf{sk}', \cdot)}(\mathsf{pk}')$ $c_1 \leftarrow E(\mathsf{pk}_1, m_1; r_1), c_2 \leftarrow E(\mathsf{pk}_2, m_1; r_2)$ $\pi \leftarrow Sim(c_1, c_2)$ $b \leftarrow A(\mathsf{pk}', c_1, c_2, \pi)$	$(\mathsf{pk}_1, \mathsf{sk}_1), (\mathsf{pk}_2, \mathsf{sk}_2) \leftarrow G(1^k)$ $\sigma \leftarrow Sim(1^k)$ let $\mathsf{pk}' = (\mathsf{pk}_1, \mathsf{pk}_2, \sigma)$ and $\mathsf{sk}' = \mathsf{sk}_1$ $(m_0, m_1) \leftarrow A^{D'(\mathsf{sk}', \cdot)}(\mathsf{pk}')$ $c_1 \leftarrow E(\mathsf{pk}_1, m_1; r_1), c_2 \leftarrow E(\mathsf{pk}_2, m_1; r_2)$ $\pi \leftarrow Sim(c_1, c_2)$ $b \leftarrow A(\mathsf{pk}', c_1, c_2, \pi)$
Game 4	
$(\mathsf{pk}_1, \mathsf{sk}_1), (\mathsf{pk}_2, \mathsf{sk}_2) \leftarrow G(1^k)$ $\sigma \leftarrow K(1^k)$ let $\mathsf{pk}' = (\mathsf{pk}_1, \mathsf{pk}_2, \sigma)$ and $\mathsf{sk}' = \mathsf{sk}_1$ $(m_0, m_1) \leftarrow A^{D'(\mathsf{sk}', \cdot)}(\mathsf{pk}')$ $c_1 \leftarrow E(\mathsf{pk}_1, m_1; r_1), c_2 \leftarrow E(\mathsf{pk}_2, m_1; r_2)$ $\pi \leftarrow P(\sigma, (c_1, c_2), (r_1, r_2, m_1))$ $b \leftarrow A(\mathsf{pk}', c_1, c_2, \pi)$	

Table 4.4: Games used for the hybrid argument.

The correctness of (G', E', D') is easy. If the keys were generated correctly and the messages were encrypted correctly, then π is a valid proof for the fact that c_1 and c_2 encrypt the same message. So the verifier $V(\sigma, x, \pi)$ will output true and the original message m will be obtained by the decryption $D(\mathsf{sk}_1, c_1)$.

Now we want to prove that (G', E', D') is IND-CCA1-secure. Let Sim be a simulator of the NIZK proof system. Consider the games shown in Table 4.4. In these games, the adversary is given a decrypting oracle during the first phase and at the end the adversary should output a bit to distinguish which game he is playing. We now show that the adversary A cannot distinguish

Game 0 and Game 4.

Let FAKE be the event that A submits a query (c_1, c_2, π) for its decryption oracle such that $D(\mathsf{sk}_1, c_1) \neq D(\mathsf{sk}_2, c_2)$ but $V(\sigma, (c_1, c_2), \pi) = 1$. This represents the event where the adversary is able to trick the verifier into returning true on a bad pair of ciphertexts (i.e., a pair not in the language). The probability of event FAKE in Game 0 is negligible because of the soundness of the proof system.

Game 1 differs from Game 0 by the use of a simulator for the random string and proof generation. They are indistinguishable by reduction to the zero-knowledge property of the proof system. In particular, if the adversary could detect the difference, then the adversary could be used to distinguish real proofs from simulated proofs. And the probability of event FAKE in Game 1 is also negligible by the zero-knowledge property.

Game 2 differs from Game 1 in that it obtains ciphertext c_2 from the message m_1. They are indistinguishable by reduction to the IND-CPA property of the underlying PKE, which guarantees that encryptions of m_0 cannot be distinguished from encryptions of m_1. Pr[FAKE] is the same as in Game 1, namely negligible.

Game 2' differs from Game 2 by using the key sk_2 for decryption instead of sk_1. The adversary's view of the two games only differs if the event FAKE occurs, and it happens with negligible probability.

Game 3 differs from Game 2' in that it obtains ciphertext c_1 from the message m_1. They are indistinguishable by reduction to the IND-CPA property of the underlying PKE, which guarantees that encryptions of m_0 cannot be distinguished from encryptions of m_1. Pr[FAKE] stays the same, namely negligible.

Game 3' differs from Game 3 by using the key sk_1 for decryption instead of sk_2. The adversary's view of the two games only differs if the event FAKE occurs, and it happens with negligible probability.

Game 4 differs from Game 3' by the use of the real NIZK proof system instead of a simulator. They are indistinguishable by reduction to the zero-knowledge property of the proof system. ■

4.6 One Time Secure Signature Scheme

Model The scheme consists of the three efficient algorithms ($Setup$, $Sign$, $Verify$). $Setup$ creates verification and signing keys. $Verify$ returns 1 if the signature verifies. The scheme is named one time secure as the attacker is only allowed one query before they must produce a valid message and signature pair. The verification and signing keys are created with the message, the recipient does not know the verification key until they receive it with the message.

$$\begin{aligned}(Vk, Sk) &\xleftarrow{\$} Setup(1^k)\\ \alpha &\leftarrow Sign(Sk, m)\\ \{1, 0\} &\leftarrow Verify(Vk, m, \alpha)\end{aligned}$$

Correctness With honest $Setup$ and $Sign$, $Pr[Verify(Vk, m, Sign(Sk, m)) = 1] = 1$.

Security We base security on the following experiment between a challenger and a PPT adversary.

Challenger		Adversary
$(Vk, Sk) = Setup(1^k)$		
	$\underrightarrow{Vk}$	
		$\underleftarrow{m}$
$\alpha = Sign(m)$		
	$\underrightarrow{\alpha}$	
		$\underleftarrow{m', \alpha'}$
Output 1 if		
$Verify(Vk, m', \alpha') = 1$		
and $(m, \alpha) \neq (m', \alpha')$		
0 otherwise		

We require that an adversary can not forge a signature for a different message.

$$Pr[Output = 1] = neg(k).$$

4.7 CCA2

We now use the one-time signature system $(Setup_s, Sign_s, Verify_s)$, a CCA1 secure scheme ($Gen_1$, Enc_1, Dec_1), and an adaptively secure NIZK proof system ($\mathcal{P}$, $\mathcal{V}$) to create a CCA2 secure scheme ($Setup$, Enc, Dec) using the Dolev-Dwork-Naor (DDN) scheme [**?**].

4.7.1 Model

Setup $Setup(1^k)$ creates $2k$ public keys, random bits σ, and $2k$ private keys.

$$(Pk, Sk) \leftarrow Setup(1^k) \text{ where}$$

$$(Pk_{i,b}, Sk_{i,b}) \leftarrow Gen_1(1^k) \quad \forall i \in [k], b \in \{0,1\}$$
$$\sigma \leftarrow \{0,1\}^{poly(k)}$$

$$Pk = \left(\begin{bmatrix} Pk_{1,0} & Pk_{2,0} & Pk_{3,0} & \cdots & Pk_{k,0} \\ Pk_{1,1} & Pk_{2,1} & Pk_{3,1} & \cdots & Pk_{k,1} \end{bmatrix}, \sigma \right), Sk = \begin{bmatrix} Sk_{1,0} \\ Sk_{1,1} \end{bmatrix}.$$

Enc $Enc(Pk, m)$ uses our one-time signature and CCA1 schemes to encrypt the message k times.

1. Generate verification and signing keys
$$(Vk, Sk) \leftarrow Setup_s(1^k).$$
2. Select one key from each of the k columns of Pk based on the bits of Vk
$$Vk = Vk_1 | Vk_2 | Vk_3 | \cdots | Vk_k$$
$$\begin{bmatrix} Pk_{1,Vk_1} & Pk_{2,,Vk_2} & Pk_{3,,Vk_3} & \cdots & Pk_{k,,Vk_k} \end{bmatrix}.$$
3. Use the k keys and a witness for each key to create k CCA1 encryptions of m
$$w_i \leftarrow \{0,1\}^*$$
$$C_i \leftarrow Enc_1(Pk_{i,Vk_i}, m; w_i).$$

4. Sign the collection of k encryptions C_i along with a proof, π, that each C_i encrypts the same message under the corresponding Pk_{i,Vk_i} public key

$$\pi \leftarrow \mathcal{P}(\sigma, C_1, \cdots, C_k, m, w)$$
$$\alpha \leftarrow Sign_s(Sk, C_1, \cdots, C_k, \pi).$$

5. Output

$$C = (Vk, C_1, \cdots, C_k, \pi, \alpha).$$

Dec If the signature and proof verify, decrypt C_1 using Sk_{1,Vk_1}

$$\begin{aligned} Dec(Vk, C_1, \cdots, C_k, \pi, \alpha) = &\perp \text{ if } Verify_s(Vk, (C_1, \cdots, C_k, \pi), \alpha) = 0, \text{ else} \\ &\perp \text{ if } \mathcal{V}(\sigma, C_1, \cdots, C_n, \pi) = 0, \text{ else} \\ &Dec_1(Sk_{1,Vk_1}, C_1). \end{aligned}$$

4.7.2 Security

The proof is similar to that for CCA1. We start with encrypting m_0, simulate the proof, change all the C_i^* from m_0 to m_1, then return to the honest proof. All while still answering before and after decryption oracle queries. We use the experiment in Table 4.5. For the proof we define two events that might occur.

Table 4.5: CCA2 Indistinguishability Experiment

Phase	Challenger		Adversary $A^{Dec_{Sk}(\cdot), Enc_{Pk}(\cdot)}$
Setup	$(Pk, Sk) \leftarrow Setup(1^k)$	$\underrightarrow{Pk}$	
Oracle Phase 1			$\underleftarrow{C_i} \quad i \in [2, q]$
	$m_i = Dec(Sk, C_i)$	$\underrightarrow{m_i}$	
Challenge			$\underleftarrow{m_0, m_1} \leftarrow A^{Dec_{Sk}(\cdot)}(Pk)$
	$b \xleftarrow{\$} \{0, 1\}$		
	$C^* = Enc(Pk, m_b)$	$\underrightarrow{C^*}$	
Oracle Phase 2			$\underleftarrow{C_i} \quad i \in [q+1, n]$
	$m_i = Dec(Sk, C_i)$	$\underrightarrow{m_i}$	
Response			$\underleftarrow{b'} \leftarrow A^{Dec_{Sk}(\cdot)}(Pk, C^*)$
	Output 1 if $b = b'$, 0 otherwise.		

Forge Event $Forge$ concerns the signature and is defined as the adversary submitting a decryption query d_i that uses our Vk and $Verify(d_i) = 1$, but is different from the cipher text we sent.

$$\begin{aligned}\exists i \in [q'] \,:\, d_i =& (Vk'_i, C'_{1,i}, \cdots, C'_{k,i}, \pi'_i, \alpha'_i) \neq C^* \\ & \text{were } Vk'_i = Vk \\ & \quad Verify(d_i) = 1.\end{aligned}$$

Fake Event $Fake$ concerns the proof and is defined as the adversary submitting a decryption query $d_i : \mathcal{V}(d_i) = 1$ but two of its cipher texts decrypt to different messages.

$$\begin{aligned}\exists i \in [q'] \,:\, d_i =& (Vk_i, C_{1,i}, \cdots, C_{k,i}, \pi_i, \alpha_i) \\ & \text{where } \mathcal{V}(\sigma, C_{1,i}, \cdots, C_{k,i}, \pi_i) = 1 \\ & \quad \text{but } \exists j, l \in [k] \,:\, Dec(Sk_{Vk_{i,j}}, C_{j,i}) \neq Dec(Sk_{Vk_{i,l}}, C_{l,i})\end{aligned}$$

We want to move from

$$C^* = (Vk^*, C_1^*, \cdots, C_k^*, \pi^*, \alpha^*)$$

encrypting m_0 to encrypting m_1 in a manner undetectable to the adversary while continuing to be able to answer our adversaries decryption queries, C_i. We examine a series of experiments with the first encrypting m_0 to the last encrypting m_1 showing that each successive pair is indistinguishable.

H0 H0 is the normal experiment above that encrypts m_0.

$Pr[Forge]$ is negligible in H0. If it were not, we could use our adversary, A, to break the security guarantee of a challenger, C_{1-time}, for the one-time secure signature scheme. We construct a signature scheme adversary, A', that receives Vk from C_{1-time} and passes it to A who returns a forgery attempt, C_i. As $Pr[Forge]$ is non-negligible, by passing C_i to C_{1-time} we have a non-negligible chance of verification, thus breaking the security guarantee of C_{1-time}.

$Pr[Fake]$ is negligible in H0. If it were not, we could with non-negligible probability cause the NIZK $Verify$ to falsely accept a cypher text, C_i, with proof that all $C_{j,i}$ encrypt the same message, when two of them did not.

H1 H1 replaces $\sigma, \mathcal{P}$ from the NIZK proof system with simulations

$$\begin{aligned}\sigma &\leftarrow Sim_1 \\ \pi &\leftarrow Sim_2.\end{aligned}$$

If H0, H1 could be distinguished by a CCA2 adversary A, we could construct an adversary, A', of a NIZK system challenger C'. Sometimes A' creates and sends to A cipher text $C_i = (Vk, C_{1,i,}, \cdots, C_{k,i}, \pi = \mathcal{P}(), \alpha)$, as in H0. Other times A' creates and sends to A cipher text $C_i = (Vk, C_{1,i}, \cdots, C_{k,i}, \pi = Sim_2, \alpha)$, as in H1. If the distribution of A's responses b differ between these two situations then A' has learned something extra from the NIZK challenger C' thus breaking the NIZK system.

$Pr[Forge]$ is negligible in H1. If it were not, we could break the signature scheme just as in H0.

$Pr[Fake]$ is negligible in H1. On every query C_i by A, we can check that each $C_{j,i}$ decrypts to the same message. If $Pr[Fake]$ increased between H0 and H1, we could use this to distinguish between $\mathcal{P}$ and the simulator. As $\mathcal{P}$ is a NIZK proof system, this is a contradiction, thus there is no detectable change in $Pr[Fake]$ between H0 and H1, and so it is still negligible.

H1.5 Samples Vk^* for the encryption challenge in the beginning as part of *Setup*. Note that the Vk^* we create for C^* is separate from the Vk sampled by the adversary when they create C_i decryption requests. Vk^* is generated randomly so generating it early or just before we need it will not affect the distribution of its values. Vk^* is only used in creating our challenge cipher, so changing from creating it just prior to giving the challenge cipher to the adversary, to creating it as some point earlier has no effect on the interaction with the adversary. Thus the distribution the adversary sees is unchanged.

$Pr[Forge]$ and $Pr[Fake]$ do not change as no change is visible to the adversary. Thus they are still negligible.

H1.75 Changes which secret key components we keep so that we know none of the challenge cipher text's secret key components, while still having the secret key components to answer decryption oracle queries. As we now know Vk^* from the start, we can use the secret key components not used by C^* to decrypt the decryption oracle queries.

Consider the complement of Vk^*, $\overline{Vk}^*$, i.e.

$$\begin{aligned} Vk^* &= 011\cdots 1 \\ \overline{Vk}^* &= 100\cdots 0. \end{aligned}$$

Highlighting the corresponding parts of our existing public key and an expanded secret key, we have

$$Pk = \left(\begin{bmatrix} Pk_{1,0} & \overline{Pk_{2,0}} & \overline{Pk_{3,0}} & \cdots & \overline{Pk_{k,0}} \\ \overline{Pk_{1,1}} & Pk_{2,1} & Pk_{3,1} & \cdots & Pk_{k,1} \end{bmatrix}, \sigma \right)$$

$$Sk = \begin{bmatrix} Sk_{1,0} & \overline{Sk_{2,0}} & \overline{Sk_{3,0}} & \cdots & \overline{Sk_{k,0}} \\ \overline{Sk_{1,1}} & Sk_{2,1} & Sk_{3,1} & \cdots & Sk_{k,1} \end{bmatrix}.$$

For the challenge cipher, we will use only the Vk^* components of the Pk and we do not need the Vk^* components of Sk, as we know the value of b. If we were to construct and publish the Pk as before, and change our private Sk to keep only the $\overline{Vk}^*$ components we can still provide and grade the challenge cipher. We now have

$$Pk = \left(\begin{bmatrix} Pk_{1,0} & Pk_{2,0} & Pk_{3,0} & \cdots & Pk_{k,0} \\ Pk_{1,1} & Pk_{2,1} & Pk_{3,1} & \cdots & Pk_{k,1} \end{bmatrix}, \sigma \right)$$

$$Sk = \begin{bmatrix} & \overline{Sk_{2,0}} & \overline{Sk_{3,0}} & \cdots & \overline{Sk_{k,0}} \\ \overline{Sk_{1,1}} & & & \cdots & \end{bmatrix}.$$

The adversary queries to the decryption oracle will have a Vk different from Vk^*. As $Pr[Forge]$ is negligible, at least one bit of the adversaries Vk will have the same value as the corresponding bit in $\overline{Vk}^*$. Thus there is always one $\overline{Vk}^*$ component of Sk that we can use to decrypt the query. As $Pr[Fake]$ is negligible, this provides the same message as in H1.5.

Thus the changes from H1.5 to H1.75 are undetectable to the adversary. With H1.75 we have ensured that when we create and present the challenge cipher we know none of the secret keys necessary to decrypt it.

H2 Changes the challenge cipher from m_0 to m_1. As we do not know the secret keys to decrypt m_0, we can use semantic security to move from m_0 to m_1.

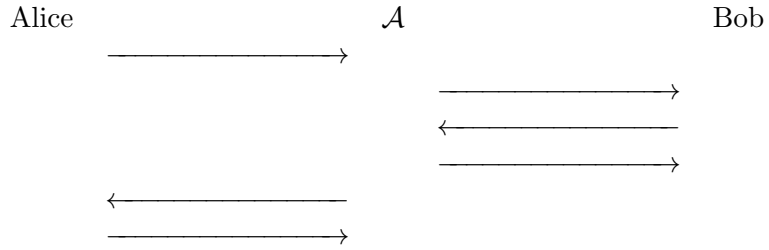
H3 Returns to the real σ, π. As in changing from H0 to H1, if the adversary, A, could detect the change, we could use A to break the underlying NIZK proof system.

4.7.3 Use and Efficiency

This is an example of non-malleable cryptography. An example of its use is in preventing one bidder in an auction from slightly increasing an opponents bid without knowledge of the opponent's unencrypted bid. It is not an efficient construction, but an efficient construction does exist in RSA.

Exercises

Exercise 4.1 *Consider the execution of a two-party protocol in the presence of an adversary that has full control of the communication channel between the two parties, such as omitting, modifying or delaying messages at its choice. This kind of attack is often referred to as* man-in-the-middle *attacks, and protocols that are secure against man-in-the-middle-attacks are said to be* non-malleable.



(a) ***Non-Malleable Public-Key Encryption.*** *We consider a man-in-the-middle adversary that receives a public-key encryption of m under public key pk and tries to "malleate" it into a new encryption also under pk. The adversary is said to have succeeded if he outputs an encryption of a value $\tilde{m}$ such that $\mathcal{R}(m,\tilde{m}) = 1$, where $\mathcal{R} \subseteq \{0,1\}^n \times \{0,1\}^n$ is a polynomial-time computable non-reflexive relation (i.e., $\mathcal{R}(m,m) = 0$). $\mathcal{A}$ also receives an auxiliary input z. We define $\mathsf{min}^{\mathcal{A}}(\mathsf{pk},\mathcal{R},m,z) = 1$ if and only if $\mathcal{A}(\mathsf{pk},z,c^*)$ produces a valid encryption of $\tilde{m}$ such that $R(m,\tilde{m}) = 1$, where c^* is an encryption of m under pk.*

To define security, we consider a stand-alone execution where a simulator $\mathcal{S}$ directly outputs an encryption. $\mathcal{S}(\mathsf{pk},z)$ only receives pk and z as input (and not c^*). We define $\mathsf{sta}^{\mathcal{S}}(\mathsf{pk},\mathcal{R},m,z) = 1$ if and only if $\mathcal{S}$ produces a valid encryption of $\tilde{m}$ under pk such that $\mathcal{R}(m,\tilde{m}) = 1$.

Definition 4.8 (Non-Malleable Public-Key Encryption) *A public-key encryption scheme $(\mathsf{Gen},\mathsf{Enc},\mathsf{Dec})$ is said to be non-malleable if for every PPT man-in-the-middle adversary $\mathcal{A}$, there exists a PPT stand-alone simulator $\mathcal{S}$ such that for every non-reflexive polynomial-time computable relation $\mathcal{R}$, every $m \in \{0,1\}^n$, and every $z \in \{0,1\}^*$, it holds that*

$$\left|\Pr[\mathsf{min}^{\mathcal{A}}(\mathsf{pk},\mathcal{R},m,z) = 1] - \Pr[\mathsf{sta}^{\mathcal{S}}(\mathsf{pk},\mathcal{R},m,z) = 1]\right| \leq \mathsf{negl}(n)$$

where $(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{Gen}(1^n)$.

Prove that a CCA2 secure public-key encryption scheme is also non-malleable.

(b) ***Non-Malleable Commitment.*** *We define non-malleable commitment in a similar way. The man-in-the-middle adversary $\mathcal{A}$ receives a commitment to a value v and attempts to*

commit to a related value $\tilde{v}$. $\mathcal{A}$ succeeds if $\mathcal{R}(v,\tilde{v})=1$. Define $\mathsf{min}^{\mathcal{A}}(\mathcal{R},v,z)=1$ if and only if $\mathcal{A}(c^*,z)$ produces a valid commitment to $\tilde{v}$ such that $R(v,\tilde{v})=1$, where c^* is a commitment of v. In the stand-alone execution $\mathcal{S}$ commits to $\tilde{v}$ directly. Define $\mathsf{sta}^{\mathcal{S}}(\mathcal{R},v,z)=1$ if and only if $\mathcal{S}(z)$ produces a valid commitment to $\tilde{v}$ such that $\mathcal{R}(v,\tilde{v})=1$.

Definition 4.9 (Non-Malleable Commitment) *A commitment scheme is said to be non-malleable if for every PPT man-in-the-middle adversary $\mathcal{A}$, there exists a PPT stand-alone simulator $\mathcal{S}$ such that for every non-reflexive polynomial-time computable relation $\mathcal{R}\subseteq\{0,1\}^n\times\{0,1\}^n$, every $v\in\{0,1\}^n$, and every $z\in\{0,1\}^*$, the following holds.*

$$\left|\Pr[\mathsf{min}^{\mathcal{A}}(\mathcal{R},v,z)=1]-\Pr[\mathsf{sta}^{\mathcal{S}}(\mathcal{R},v,z)=1]\right|\leq\mathsf{negl}(n).$$

Given a CCA2 secure public-key encryption scheme $(\mathsf{Gen},\mathsf{Enc},\mathsf{Dec})$, define the commitment scheme as

$$C(v):=(\mathsf{pk},\mathsf{Enc}(\mathsf{pk},v))$$

where $(\mathsf{pk},\mathsf{sk})\leftarrow\mathsf{Gen}(1^n)$. Show that such a commitment scheme is not necessarily non-malleable.

Chapter 5

Bilinear Maps

We introduce Bilinear Maps and two of its applications: NIKE, Non-Interactive Key Exchange; and IBE, Identity Based Encryption.

5.1 Diffie-Hellman Key Exchange

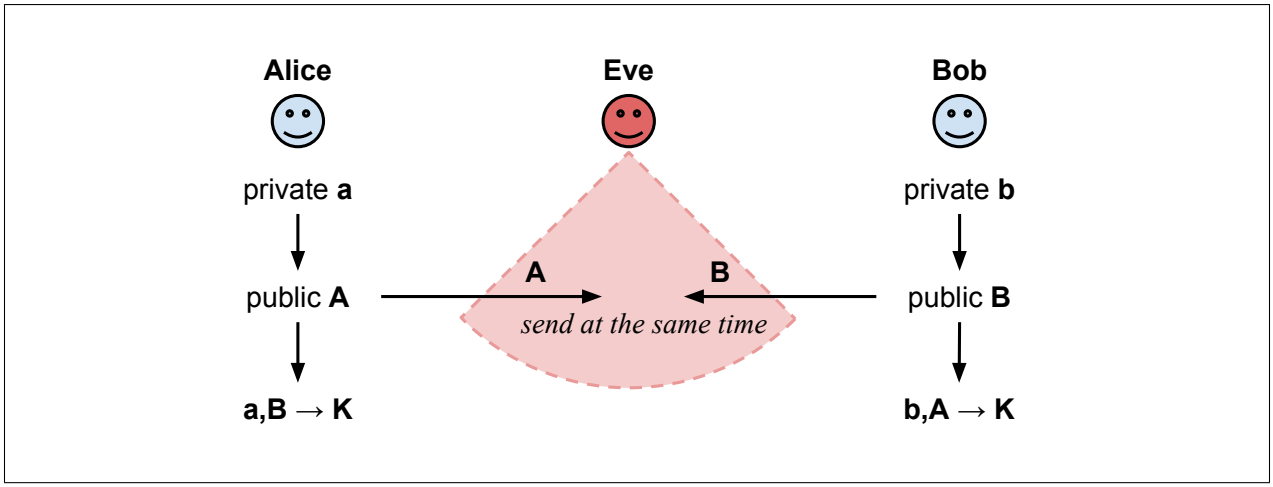


Figure 5.1: Diffie-Hellman Key Exchange

Fig 5.1 illustrates Diffie-Hellman key exchange. Alice and Bob each has a private key (a and b respectively), and they want to build a shared key for symmetric encryption communication. They can only communicate over a insecure link, which is eavesdropped by Eve. So Alice generates a public key A and Bob generates a public key B, and they send their public key to each other at the same time. Then Alice generates the shared key K from a and B, and likewise, Bob generates the shared key K from b and A. And we have $\forall$ PPT Eve, $Pr[k = Eve(A, B)] = neg(k)$, where k is the length of a.

5.1.1 Discussion 1

Assume that $\forall(g, p)$, and $a_1, b_1 \xleftarrow{\$} Z_p^*$, and $a_2, b_2, r \xleftarrow{\$} Z_p^*$, we have $(g^{a_1}, g^{b_1}, g^{a_1 b_1}) \overset{c}{\simeq} (g^{a_2}, g^{b_2}, g^r)$. How to apply this to Diffie-Hellman Key Exchange?

Make $A = g^a$, $B = g^b$, $K = A^b = g^{ab}$, and $K = B^a = g^{ab}$.

5.1.2 Discussion 2

How does Diffie-Hellman Key Exchange imply Public Key Encryption?

Alice $pk = A$, $sk = a$, $Enc(pk, m \in \{0, 1\})$.

Bob $b, r \leftarrow Z_p^*$ $(g^b, mA^b + (1 - m)g^r)$

Alice $Dec(sk, (c_1, c_2))$

$c_1^a \stackrel{?}{=} c_2$

5.2 Bilinear Maps

Definition 5.1 *Bilinear Maps*

Bilinear Maps is (G, P, G_T, g, e)*, where* e *is an efficient function* $G \times G \to G_T$ *such that*

- *if* g *is generator of* G*, then* $e(g, g)$ *is the generator of* G_T*.*

- $\forall a, b \in Z_p$*, we have* $e(g^a, g^b) = e(g, g)^{ab} = e(g^b, g^a)$*.*

5.2.1 Discussion 1

How does Bilinear Maps apply to Diffie-Hellman?

Make $A = g^a$, $B = g^b$, and $T = g^{ab}$, then Diffie-Hellman has $e(A, B) = e(g, T)$.

5.3 Tripartite Diffie-Hellman

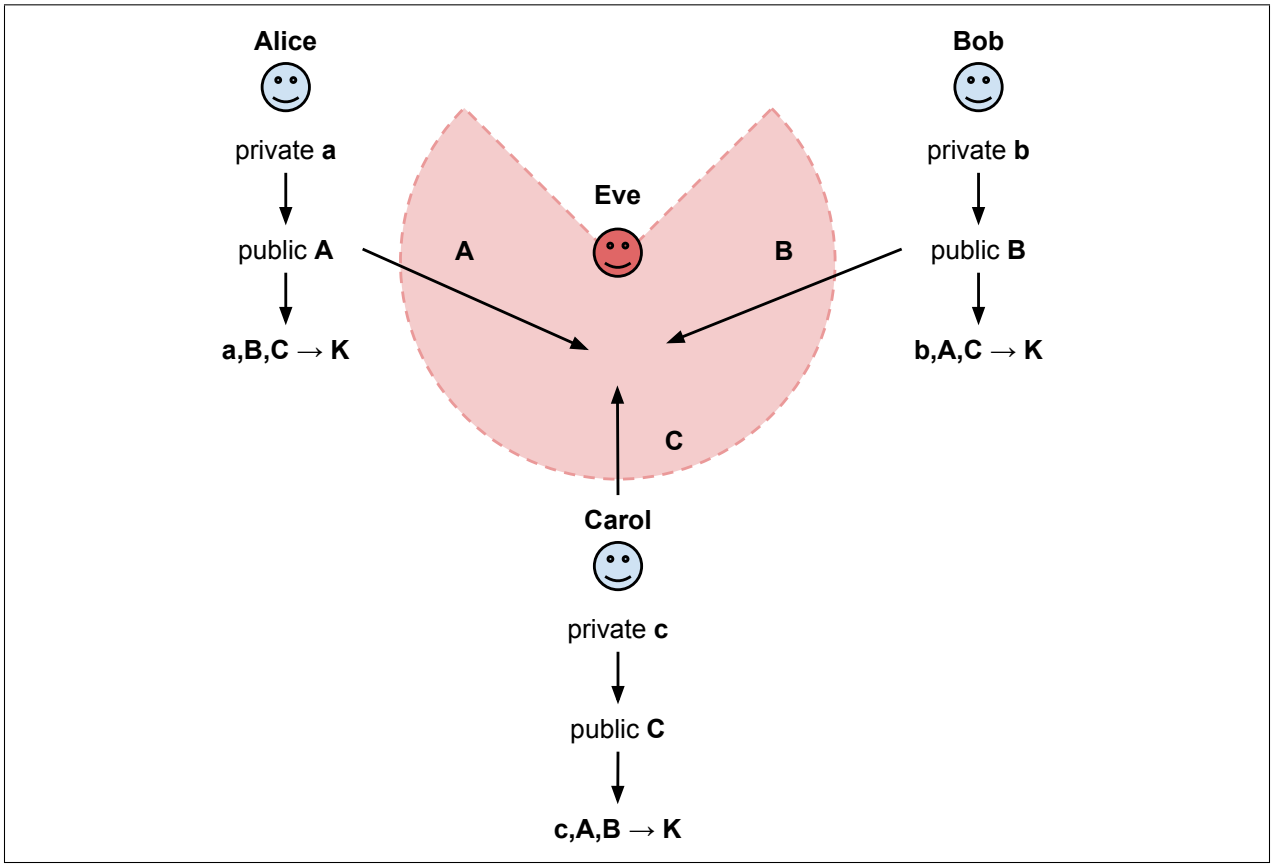


Figure 5.2: Tripartite Diffie-Hellman Key Exchange

Fig 5.3 illustrates Tripartite Diffie-Hellman key exchange. a, b, and c are private key of Alice, Bob, and Carol, respectively. They use g^a, g^b, g^c as public key, and the shared key $K = e(g,g)^{abc}$. Formally, we have

$$a, b, c \xleftarrow{\$} Z_p^*, r \xleftarrow{\$} Z_p^*$$

$$A = g^a, B = g^b, C = g^c$$

$$K = e(g,g)^{abc}$$

5.4 IBE: Identity-Based Encryption

IBE contains four steps: *Setup*, *KeyGen*, *Enc*, and *Dec*. We illustrate it in Figure 5.4. In first step, Key authority get a Master Public Key (MPK) and Master Signing Key (MSK) from $Setup(1^k)$. Then a user with an ID (in this example, "Mike"), sends his ID to the key authority. The key authority generates the Signing Key of Mike with $KeyGen(MSK, ID)$ ans sends it back. Another use, Alice, wants to send an encrypted message to Mike. She only has MPK and Mike's ID. So she encrypts the message with $c = Enc(MPK, ID = Mike, m)$, and sends the encrypted message c to Mike. Mike decodes c with $m = Dec(c, SK_{Mike})$. Notice that Alice never need to know Mike's public key. She only needs to remember MPK and other people's IDs.

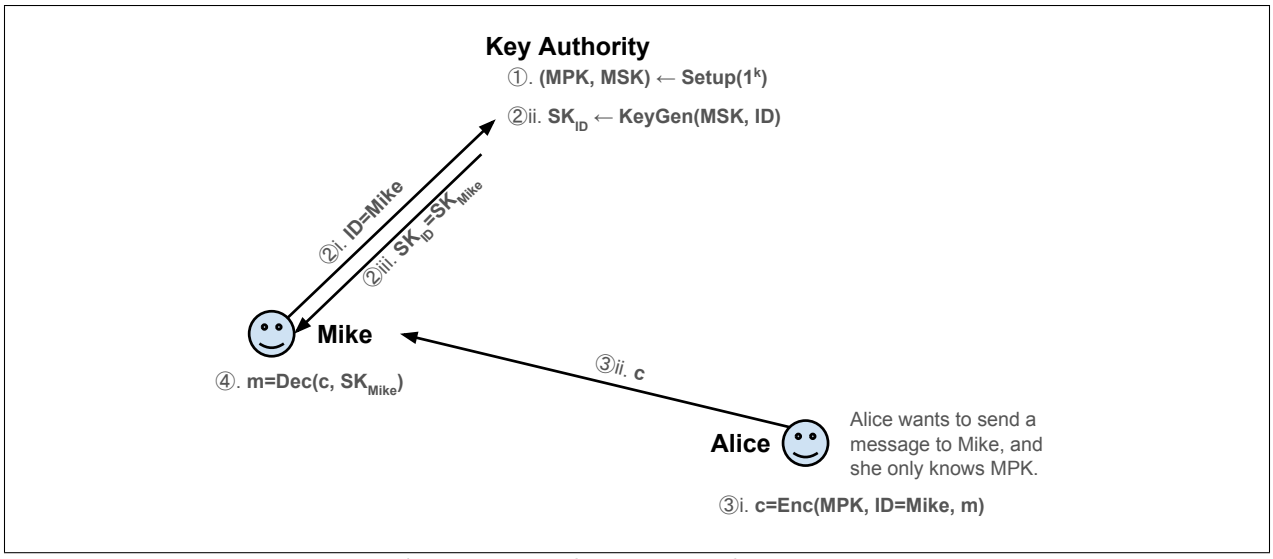


Figure 5.3: Identity-Based Encryption

Formally, we have

$$Pr\left[\begin{array}{c}(MPK, MSK) \leftarrow Setup(1^k),\\ SK_{ID} \leftarrow KeyGen(MSK, ID),\\ c \leftarrow Enc(MPK, ID, m),\\ m \leftarrow Dec(SK_{ID}, c)\end{array}\right] = 1$$

5.4.1 Security Descriptions

We have different security descriptions for IBE, as discussed in this section.

CCA1

Challenger		**Adversary**
$(MPK, MSK) \leftarrow Setup(1^k)$	$\xrightarrow{MPK}$	
	$\xleftarrow{ID_1}$	
$SK_{ID_1} \leftarrow KeyGen(MSK, ID_1)$	$\xrightarrow{SK_{ID_1}}$	
$\vdots$	$\vdots$	$\vdots$
	$\xleftarrow{ID_i}$	
$SK_{ID_i} \leftarrow KeyGen(MSK, ID_i)$	$\xrightarrow{SK_{ID_i}}$	
	$\xleftarrow{ID^*, m_0, m_1}$	$\forall i \in [q], ID^* \neq ID_i$
$b \xleftarrow{\$} \{0,1\}$, $c^* = Enc(MPK, ID^*, m_b)$	$\xrightarrow{c^*}$	
Output 1 if $b' = b$, otherwise 0	$\xleftarrow{b'}$	

CCA2

In CCA2, we allow adversary to send further queries after getting c^*.

Challenger		Adversary
$(MPK, MSK) \leftarrow Setup(1^k)$	$\xrightarrow{MPK}$	
	$\xleftarrow{ID_1}$	
$SK_{ID_1} \leftarrow KeyGen(MSK, ID_1)$	$\xrightarrow{SK_{ID_1}}$	
	$\vdots$	
	$\xleftarrow{ID_i}$	
$SK_{ID_q} \leftarrow KeyGen(MSK, ID_q)$	$\xrightarrow{SK_{ID_q}}$	
	$\xleftarrow{ID^*, m_0, m_1}$	$\forall i \in [q'], ID^* \neq ID_i$
$b \xleftarrow{\$} \{0,1\}$, $c^* = Enc(MPK, ID^*, m_b)$	$\xrightarrow{c^*}$	
	$\xleftarrow{ID_{q+1}}$	
$SK_{ID_{q+1}} \leftarrow KeyGen(MSK, ID_{q+1})$	$\xrightarrow{SK_{ID_{q+1}}}$	
$\vdots$	$\vdots$	$\vdots$
	$\xleftarrow{ID_{q'}}$	
$SK_{ID_{q'}} \leftarrow KeyGen(MSK, ID_{q'})$	$\xrightarrow{SK_{ID_{q'}}}$	
Output 1 if $b' = b$, otherwise 0	$\xleftarrow{b'}$	

Selective Security

In selective security, the adversary sends ID^* before everything.

Challenger		Adversary
	$\xleftarrow{ID^*}$	$\forall i \in [q], ID^* \neq ID_i$
$(MPK, MSK) \leftarrow Setup(1^k)$	$\xrightarrow{MPK}$	
	$\xleftarrow{ID_1}$	
$SK_{ID_1} \leftarrow KeyGen(MSK, ID_1)$	$\xrightarrow{SK_{ID_1}}$	
$\vdots$	$\vdots$	$\vdots$
	$\xleftarrow{ID_q}$	
$SK_{ID_q} \leftarrow KeyGen(MSK, ID_q)$	$\xrightarrow{SK_{ID_q}}$	
	$\xleftarrow{m_0, m_1}$	
$b \xleftarrow{\$} \{0,1\}$, $c^* = Enc(MPK, ID^*, m_b)$	$\xrightarrow{c^*}$	
Output 1 if $b' = b$, otherwise 0	$\xleftarrow{b'}$	

5.4.2 Discussion 1

How does Bilinear Maps apply to IBE?

Given Bilinear Maps: (G, P, G_T, g, e), we have

1. $(G, P, G_T, g, e) \leftarrow Setup(1^k)$
2. $s \leftarrow Z_p^*$, and $H_1 : \{0,1\}^* \rightarrow G$, $H_2 : G_T \rightarrow \{0,1\}^n$
3. $MPK = (G, g^s, H_1, H_2)$, and $MSK = (s)$

Let's look at how we construct each function in IBE.

$KeyGen(s, ID)$**:**

1. Output $SK_{ID} = (H_1(ID))^s$

$Enc(MPK, ID, m)$**:**

1. $r \leftarrow Z_p^*$
2. $c_1 = g^r$
3. $c_2 = m \oplus H_2(e(A, H_1(ID)^r))$, where $A = g^s$
4. Output (c_1, c_2)

$Dec(SK_{ID}, (c_1, c_2))$**:**

1. Get $e(A, H_1(ID)^r) = e(H_1(ID)^s, c_1) = e(SK_{ID}, c_1)$
2. Get $m = c_2 \oplus H_2(e(A, H_1(ID))^r)$

Proof. To prove this, we use a hybrid argument. Assume we have two oracles with exact random functions, denoted as O_{H_1} and O_{H_2}. One can request a random string from them with a query ID. The random strings are denoted as $H_1(ID)$ and $H_2(ID)$, respectively. These two oracles keep track of query IDs and corresponding responses. If a query ID was seen before, they return the exact same response corresponding to it. If not, they generate a random string, correspond the string to the ID, and return the string.

We first define $\mathscr{H}_0$, in which $H_1(ID)$ and $H_2(ID)$ are generated by the oracles. We use the construction described above.

Challenger		**Adversary**
	$\xleftarrow{\mathcal{G}, ID^*}$	$\forall i \in [q], ID^* \neq ID_i$
	$\xleftarrow{g^s}$	
	$\xleftarrow{O_{H_1}}$	
	$\xleftarrow{O_{H_2}}$	
	$\xleftarrow{ID_1}$	
$SK_{ID_1} \leftarrow KeyGen(s, ID_1)$	$\xrightarrow{SK_{ID_1}}$	
$\vdots$	$\vdots$	$\vdots$
	$\xleftarrow{ID_q}$	
$SK_{ID_q} \leftarrow KeyGen(s, ID_q)$	$\xrightarrow{SK_{ID_q}}$	
	$\xleftarrow{m_0, m_1}$	
$b \xleftarrow{\$} \{0,1\}$, $c^* = Enc(MPK, ID^*, m_b)$	$\xrightarrow{c^*}$	
Output 1 if $b' = b$, otherwise 0	$\xleftarrow{b'}$	

Then we discard oracle's H_1, and use $H_1(ID) = g^{\alpha_{ID}}$, where $\alpha_{ID} \leftarrow Z_p^*$. We denote this as $\mathscr{H}_1$.
Then we change SK_{ID} to $SK_{ID} = (H_1(ID))^s = (g^{\alpha_{ID}})^s$. We denote this as $\mathscr{H}_2$.
We have Bilinear Decision Diffie-Hellman (DDH). If $\mathscr{H}_2$ breaks DDH, then $\mathscr{H}_0$ can as well.

In DDH, we have $(g^a, g^b, g^c, e(g,g)^{abc}) \stackrel{c}{\simeq} (g^a, g^b, g^c, e(g,g)^r)$. We denote $A = g^a$, $B = g^b$, $C = g^c$. And in $\mathscr{H}_2$, we have $A = g^s$, $B = H_1(ID^*)$, $C = c_1 = g^r$. And in $c_2 = m \oplus H_2(e(g^s, H_1(ID^*))^r)$, we have $T = H_2(e(g^s, H_1(ID^*))^r) = e(g,g)^{abc}$.

■

Chapter 6

Zero-Knowledge Proofs

Traditional Euclidean style proofs allow us to prove veracity of statements to others. However, such proof systems have two shortcomings: (1) the running time of the verifier needs to grow with the length of the proof, and (2) the proof itself needs to be disclosed to the verifier. In this chapter, we will provide methods enabling provers to prove veracity of statements of their choice to verifiers while avoiding the aforementioned limitations. In realizing such methods we will allow the prover and verifier to be probabilistic and also allow them to interact with each other.[1]

6.1 Interactive Proofs

Definition 6.1 (Interactive Proof System) *For a language L we have an interactive proof system if $\exists$ a pair of algorithms (or better, interacting machines) $(\mathcal{P}, \mathcal{V})$, where $\mathcal{V}$ is polynomial in $|x|$, and both can flip coins, such that:*

- *Completeness: $\forall x \in L$*
$$\Pr_{\mathcal{P},\mathcal{V}}[Output_{\mathcal{V}}(\mathcal{P}(x) \leftrightarrow \mathcal{V}(x)) = 1] = 1,$$
- *Soundness: $\forall x \notin L$, $\forall \mathcal{P}^*$*
$$\Pr_{\mathcal{V}}[Output_{\mathcal{V}}(\mathcal{P}^*(x) \leftrightarrow \mathcal{V}(x)) = 1] < neg(|x|),$$

where $Output_{\mathcal{V}}(\mathcal{P}(x) \leftrightarrow \mathcal{V}(x))$ denotes the output of $\mathcal{V}$ in the interaction between $\mathcal{P}$ and $\mathcal{V}$ where both parties get x as input. We stress that $\mathcal{P}$ and $\mathcal{P}^*$ can be computationally unbounded.

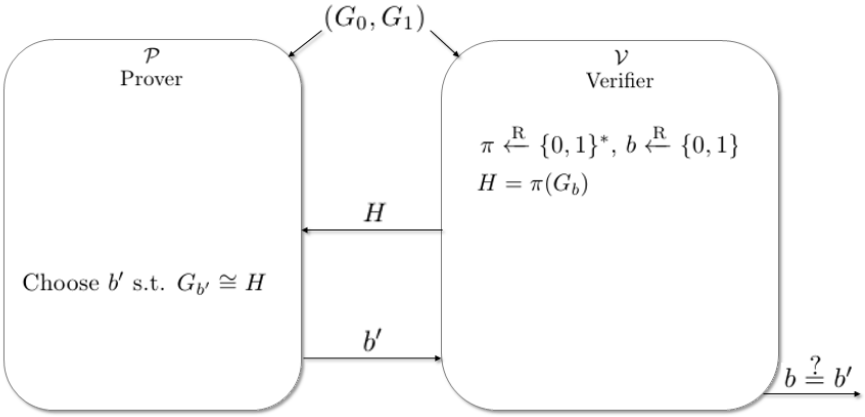
Interactive Proof for Graph Non-Isomorphism (GNI). We say that two graphs G_0 and G_1 are isomorphic, denoted $G_0 \cong G_1$, if $\exists$ an isomorphism $f : V(G_0) \to V(G_1)$ s.t. $(u, v) \in E(G_0)$ iff $(f(u), f(v)) \in E(G_1)$, where $V(G)$ and $E(G)$ are the vertex and edge sets of some graph G. On the other hand, G_0 and G_1 are said to be non-isomorphic, $G_0 \ncong G_1$, if $\nexists$ any such f, and $GNI = \{(G_0, G_1) |\ G_0 \ncong G_1\}$ be the language that consists of pairs of graphs that are not isomorphic.

GNI is not believed to have short proofs so an interactive proof could offer a prover a mechanism to prove to a polynomially bounded verifier that two graphs are non-isomorphic.

The intuition behind a protocol to accomplish the above task is simple. Consider a verifier that randomly rename the vertices of one of the graphs and give it to the prover. Can the prover given the relabeled graph figure out which graph did the verifier start with? If G_0 and G_1 were not isomorphic

[1] Formally, they can be modeled as interactive PPT Turing Machines.

then an unbounded prover can figure this out. However, in case G_0 and G_1 are isomorphic then the distribution resulting form random relabelings of G_0 and G_1 are actually identical. Therefore, even an unbounded prover has no way of distinguishing which graph the verifier started with. So the prover has only a $\frac{1}{2}$ probability of guessing which graph the verifier started with. Note that by repeating this process we can reduce the success probability of a cheating prover to negligible. More formally:



- Completeness: If $(G_0, G_1) \in$ GNI, then the unbounded $\mathcal{P}$ can distinguish isomorphism of G_0 against those of G_1 and can always return the correct b'. Thus, $\mathcal{V}$ will always output 1 for this case.

- Soundness: If $(G_0, G_1) \notin$ GNI, then it is equiprobable that H is a random isomorphism of G_0 as it is G_1 and so $\mathcal{P}$'s guess for b' can be correct only with a probability $\frac{1}{2}$. Repeating this protocol k times means the probability of guessing the correct b' for all k interactions is $\frac{1}{2^k}$. And so the probability of $\mathcal{V}$ outputting 0 (e.g. rejecting $\mathcal{P}$'s proof at the first sign of falter) is $1 - \frac{1}{2^k}$.

6.2 Zero Knowledge Proofs

Definition 6.2 (NP-Verifier) *A language L has an NP-verifier if $\exists$ a verifier $\mathcal{V}$ that is polynomial time in $|x|$ such that:*

- *Completeness: $\forall x \in L,\ \exists$ a proof π s.t. $\mathcal{V}(x, \pi) = 1$*

- *Soundness: $\forall x \notin L\ \forall$ purported proof π we have $\mathcal{V}(x, \pi) = 0$*

That is, the conventional idea of a proof is formalized in terms of what a computer can efficiently verify. So a set of statements considered true (e.g. in a language L) is complete and sound if a proof can be written down that can be "easily" and rigorously verified if and only if a statement is in the language.

Efficient Provers. Unfortunately (fortunately?), there aren't real-life instances of all-powerful provers that we know of. And for cryptography we must make more reasonable assumptions about the provers. In this case we will assume provers are also bounded to be *efficient*.

Previously, if a prover wanted to prove that two graphs, G_0 and G_1 were isomorphic, it would use its all-powerfulness to find the isomorphic mapping between the two graphs and give it to the

verifier to complete the proof. But now, being computationally bounded, the prover is in the same boat as the verifier and can find a proof no better than the verifier can. In order for the prover to be able to prove something that the verifier cannot find out on their own, the prover must have some extra information. If, for example, the prover simply knew the isomorphism between the graphs, this would be the sufficient extra information it needs to enact the proof. That's a rather boring proof though. We have interaction now! Can't we do something fancier?

What if the prover wanted to prove that two graphs were isomorphic but didn't want to fully reveal the isomorphism that they know. If they're lying and don't know an isomorphism is their a way we can exploit them again?

When G_0 and G_1 are isomorphic, the isomorphism between them would be a *witness*, w, to that fact, that can be used in the proof. Unfortunately, the prover is being stubborn and won't just tell us that isomorphism, $w : V(G_0) \to V(G_1)$, that they claim to have. The prover is comfortable however giving us a "scrambled" version, ϕ, of w as long as it doesn't leak any information about their precious w. For example, the prover is willing to divulge $\phi = \pi \circ w$ where π is a privately chosen random permutation of $|V| = |V(G_0)| = |V(G_1)|$ vertices. Since π renames vertices completely randomly, it scrambles what w is doing entirely and ϕ is just a random permutation of $|V|$ elements. At this point, we might be a little annoyed at the prover since we could have just created a random permutation on our own. This might give us an idea on how to gain a little more information however, even though we gained none here:

If we want to be convinced that ϕ really is of the form $\pi \circ w$, thus containing w in its definition, and isn't just a completely random permuation, we can note that if it is of that form then $\phi(G_0) = \pi(w(G_0)) = \pi(G_1)$ (since w being an isomorphism implies that $w(G_0) = G_1$). Note that we started with a mapping on input G_0 and ended with a mapping on input G_1. With an isormphism, one could get from one graph to the other seamlessly; if the prover *really* has the isomorphism it claims to have, then it should have no problem displaying this ability. So, what if we force the prover to give us $H = \pi(G_1)$ just after randomly choosing its π and then let it show us its ability to go from G_1 to G_0 with ease: give us a ϕ so that $\phi(G_0) = \pi(G_1) = H$. The only way the prover can give a mapping that jumps from G_0 to G_1 in such a way is if they know an isomorphism; if the prover could find a ϕ efficiently but did *not* know an isomorphism then they would have been able to see that $\pi^{-1}(\phi(G_0)) = G_1$ and thus have $\pi^{-1} \circ \phi$ as an isomorphism from G_0 to G_1, which would contradict the assumed hardness of finding isomorphisms in the GI problem. So by forcing the prover to give us H as we've defined and to produce a ϕ so that $\phi(G_0) = H$, we've found a way to expose provers that don't really have an isomorphism and we can then be convinced that they really do know w when they pass our test. And the prover didn't directly tell us w, so they may be able to salvage some secrecy!

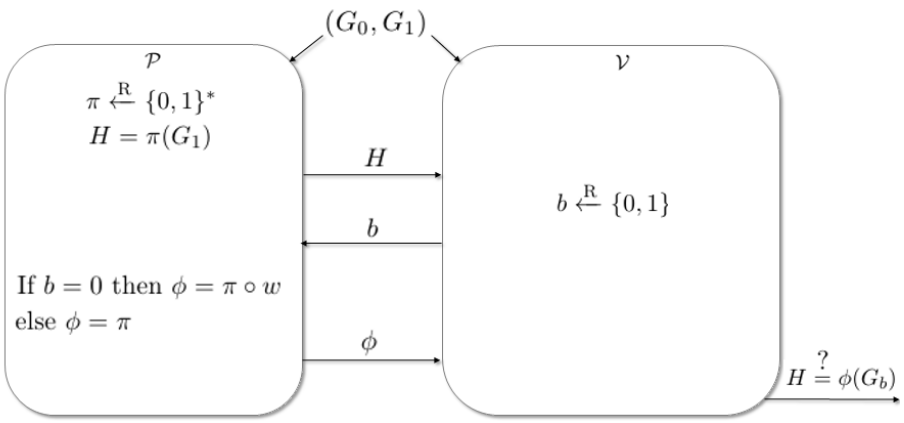
But not everything is airtight about this interaction. Why, for instance, would the prover be willing to provide $H = \pi(G_1)$ when they're trying to divulge as little information as possible? The prover was comfortable giving us ϕ since we could have just simulated the process of getting a completely random permutation of vertices ourselves, but couldn't the additional information of H reveal information about w? At this point, the annoyed feeling may return as we realize that, $H = \pi(G_1) = \pi'(G_0)$, for some π', is just a random isomorphic copy of G_0 *and* G_1 as long as $G_0 \cong G_1$; we could have just chosen a random π', set $H = \pi'(G_0)$, and let $\phi = \pi'$ and would have created our very own random isomorphic copy, H, of G_1 that satisfies our test condition $H = \phi(G_0)$ just like what we got from our interaction with the prover. We couldn't have gained any new information from the prover because we could have run the whole test on our own!

Well, something must be wrong; we couldn't have been convinced of something without gaining *any* new information. Indeed, the test has a hole in it: how can we force the prover to give us $H = \pi(G_1)$ like we asked? If the prover is lying and it knows our test condition is to verify that

$H = \phi(G_0)$, the prover might just cheat and give us $H = \pi(G_0)$ so it doesn't have to use knowledge of w to switch from G_1 to G_0. And, in fact, by doing this and sending $\phi = \pi$, the prover would fool us!

To keep the prover on their toes, though, we can randomly switch whether or not we want H to equal $\phi(G_0)$ or $\phi(G_1)$. If, in our interaction, the prover must first provide their $H = \pi(G_1)$ before we let them know which we want, they then lock themselves into a commitment to either G_0 or G_1 depending on whether they're trying to cheat or not, respectively. They only have a 50% chance of committing to the same case we want on a given round and so, if they don't have w to deftly switch between G_0 and G_1 to always answer correctly, they again have to be an extremely lucky guesser if they're trying to lie.

Again, we've created an interactive scheme that can catch dishonest provers with probability 1-$\frac{1}{2^k}$ and where we always believe honest provers!



- Completeness: If $(G_0, G_1) \in$ GI and $\mathcal{P}$ knows w, then whether $\mathcal{V}$ chooses $b = 0$ or 1, $\mathcal{P}$ can always give the correct ϕ which, by definition, will always result in $H = \phi(G_b)$ and so $\mathcal{V}$ will always output 1.

- Soundness: If $(G_0, G_1) \notin$ GI, then $\mathcal{P}$ can only cheat, as discussed earlier, if the original H it commits to ends up being $\pi(G_b)$ for the b that is randomly chosen at the next step. Since b isn't even chosen yet, this can only happen by chance with probability $\frac{1}{2}$. And so the probability $\mathcal{V}$ outputs 0 is $1 - \frac{1}{2^k}$ for k rounds.

And so, again, we've correctly captured the idea of a proof by having this interaction. But there's a strange feeling that may be lingering around us...

As a verifier, we've seen some things in interacting with the prover. Surely, clever folks like ourselves must be able to glean *some* information about w after seeing enough to thoroughly convince us that the prover knows w. We've first seen H, and we've also seen the random b that we chose, along with ϕ at the end; this is our whole view of information during the interaction. But we're more bewildered than annoyed this time when we realize we could have always just chosen b and ϕ randomly and set $H = \phi(G_b)$ on our own. Again, everything checks out when $G_0 \cong G_1$ and we could have produced everything that we saw during the interaction before it even began. That is, the distribution of the random variable triple (H, b, ϕ) is identical whether it is what we saw from the prover during the interaction or it is yielded from the solitary process we just described. We've just constructed a complete interactive proof system that entirely convinces us of the prover's knowledge of w, yet we could have simulated the whole experience on our own! We couldn't have gain any knowledge about w since we didn't see anything we couldn't have manufactured on own,

yet we are entirely convinced that $(G_0, G_1) \in \text{GI}$ and that $\mathcal{P}$ knows w! And so the prover has proven something to us yet has given us absolutely zero additional knowledge!

This may feel very surprising or as if you've been swindled by a fast talker, and it very much should feel this way; it was certainly an amazing research discovery! But this is true, and it can be made rigorous:

We should first be sure what we want out of this new proof system. We of course want it to be complete and sound so that we accept proofs iff they're true. But we also want the verifier to gain zero knowledge from the interaction; that is, the verifier should have been able to simulate the whole experience on its own without the verifier. Finally, we would also like all witnesses to a true statement to each be sufficient to prove the veracity of that statement and so we let R be the relation s.t. $x \in L$ iff $\exists$ a witness w s.t. $(x, w) \in R$. We can then gather all witness by defining $R(x)$ to be the set of all such witnesses.

Definition 6.3 (Honest Verifier Zero Knowledge Proof [HVZK]) *For a language L we have a (perfect) HVZK proof system w.r.t. witness relation R if $\exists$ an interactive proof system, $(\mathcal{P}, \mathcal{V})$ s.t. $\exists$ a PPT machine $\mathcal{S}$ (called the simulator) s.t. $\forall x \in L$, $\forall w \in R(x)$ the following distributions are identical:*

$$View_{\mathcal{V}}(\mathcal{P}(x, w) \leftrightarrow \mathcal{V}(x))$$

$$\mathcal{S}(x)$$

where $View_{\mathcal{V}}(\mathcal{P}(x, w) \leftrightarrow \mathcal{V}(x))$ is the random coins of $\mathcal{V}$ and all the messages $\mathcal{V}$ saw.

Remark 6.1 *In the above definition, $View_{\mathcal{V}}(\mathcal{P}(x, w) \leftrightarrow \mathcal{V}(x))$ contains both the random coins of $\mathcal{V}$ and all the messages that $\mathcal{V}$ saw, because they together constitute the view of $\mathcal{V}$, and they are correlated. If the random coins of $\mathcal{V}$ are not included in the definition of $View_{\mathcal{V}}(\mathcal{P}(x, w) \leftrightarrow \mathcal{V}(x))$, then even if $\mathcal{S}$ can generate all messages that $\mathcal{V}$ saw with the same distribution as in the real execution, the verifier may still be able to distinguish the two views using its random coins.*

There is an interesting progression of the requirements of a proof system: Completeness, Soundness, and the Zero Knowledge property. Completeness first cares that a prover-verifier pair exist and can capture all true things as a team that works together; they both honestly obey the protocol trying prove true statements. Soundness, however, assumes that the prover is a liar and cares about having a strong enough verifier that can stand up to any type of prover and not be misled. Finally, Zero Knowledge assumes that the verifier is hoping to glean information from the proof to learn the prover's secrets and this requirement makes sure the prover is clever enough that it gives no information away in its proof.

Unlike the soundness' requirment for a verifier to combat *all* malicious provers, HVZK is only concerned with the verifier in the original prover-verifier pair that follows the set protocol. Verifiers that stray from the protocol or cheat, however, are captured in the natural generalization to Zero Knowledge proofs.

Definition 6.4 (Efficient Prover Zero-Knowledge Proof) *We say (P, V) is an efficient prover zero-knowledge proof system for a language L and relation R_L if*

1. *The prover P runs in polynomial time.*
2. *The protocol is* complete. *That is, for every $x \in L$ there exists a witness $w \in R_L(x)$ such that*

$$\Pr[P(x, w) \leftrightarrow V(x) \text{ accepts}] = 1.$$

3. *The protocol is* sound *against unbounded provers. That is, for* $\forall x \notin L$, *we have*

$$\Pr[P^*(x,w) \leftrightarrow V(x) \text{ rejects}] \geq 1/2$$

for any prover P^* *of arbitrary computation power and any witness* w.

4. *There exists an expected polynomial time probabilistic machine* S *(a simulator) such that for all PPT* V^*, *for all* $x \in L, w \in R_L(x), z \in \{0,1\}^*$ *we have*

$$\{View_{V^*}(P(x,w) \leftrightarrow V^*(x,z))\} \simeq_c \{S^{V^*}(x,z)\}$$

The soundness probability can be amplified to be greater than any $1-1/2^k$, for arbitrary $k > 0$, by repeating the proof k times. More precisely, we construct an efficient prover zero-knowledge proof system $(\tilde{P}, \tilde{V})$ which repeats (P, V) independently for k times, and $\tilde{V}$ accepts if and only if V accepts in all the executions.

It is easy to see that $\tilde{P}$ runs in polynomial time and that the protocol is complete. Moreover, it has the following soundness guarantee: for $\forall x \notin L$,

$$\begin{aligned}
&\Pr\left[\tilde{P}^*(x,w) \leftrightarrow \tilde{V}(x) \text{ rejects}\right] \\
=&1 - \Pr\left[\forall 1 \leq i \leq k, P_i^*(x,w) \leftrightarrow V(x) \text{ accepts}\right] \\
=&1 - \prod_{i=1}^{k} \Pr\left[P_i^*(x,w) \leftrightarrow V(x) \text{ accepts}\right] \\
\geq&1 - \frac{1}{2^k}
\end{aligned}$$

for any prover $\tilde{P}^* = (P_1^*, \cdots, P_k^*)$ of arbitrary computation power and any witness w.

Finally, it is zero-knowledge, namely, there exists an expected PPT $\tilde{S}$ such that for all PPT $\tilde{V}^*$, and for all $x \in L, w \in R_L(x), z \in \{0,1\}^*$,

$$\left\{View_{\tilde{V}^*}(\tilde{P}(x,w) \leftrightarrow \tilde{V}^*(x,z))\right\} \simeq_c \left\{\tilde{S}^{\tilde{V}^*}(x,z)\right\}.$$

The construction of $\tilde{S}$ is repeating S for k times. We prove by hybrid argument that the above two distributions are indistinguishable. H_i is defined to be the output of repeating S for the first i executions with $\tilde{V}^*$ and repeating P for the rest $k-i$ executions. Then H_0 is the left distribution and H_k is the right one. Any attacher that can distinguish the above two distributions leads to an attacker that can distinguish H_{i-1} and H_i for some $1 \leq i \leq k$, which violates the zero-knowledge property of the original proof system (P, V).

The order of the quantifiers in item 4 matters. If we quantify over x and w before quantifying over the simulator, then we could hard-code x and w into our simulator. That is, for all $x \in L, w \in R_L(x)$, there exists an expected polynomial time probabilistic machine $S_{x,w}$ such that for all PPT V^* and $z \in \{0,1\}^*$,

$$\{View_{V^*}(P(x,w) \leftrightarrow V^*(x,z))\} \simeq_c \{S_{x,w}^{V^*}(x,z)\}$$

Since we would like our simulator to be universal, this is not acceptable.

If we quantify first over the verifier V^* and then over simulators S, then this variant is considered as *non-black-box zero-knowledge*. Our standard definition is considered as *black-box zero-knowledge*. There also exist variants that use statistical indistinguishability rather than computational indistinguishability.

The z in item 4 is considered as *auxiliary input*. The auxiliary input is crucial for the above argument of soundness amplification.

We will discuss the importance of requiring expected polynomial time in the next section.

6.3 Graph Isomorphism

Recall our protocol for graph isomorphism: the interaction is $P(x, w) \leftrightarrow V(x)$ where x represents graphs $G_0 = (V, E_0)$ and $G_1 = (V, E_1)$ and w represents a permutation π on V such that $\pi(G_0) = G_1$.

1. P samples a random permutation $\sigma : V \to V$ and sends the graph $H = \sigma(G_1)$ to V.
2. V samples a random bit b and sends it to P.
3. If $b = 1$, then P defines a permutation τ to be σ. If $b = 0$, then instead $\tau = \sigma \circ \pi$. P then sends τ to V.
4. V verifies that $\tau(G_b) = H$ and accepts if so.

We will show that this is an efficient prover zero-knowledge proof system. It is clear that if G_0 and G_1 are isomorphic, then this protocol will succeed with probability 1.

For soundness, observe that if G_0 is not isomorphic to G_1, then the graph H that P sends to V in step 1 of the protocol can be isomorphic to at most one of G_0 or G_1. Since V samples a bit b uniformly at random in step 2, then there is a probability of at most $1/2$ that P can produce a valid isomorphism in step 3.

For zero knowledge, consider the following simulator S with input G_0 and G_1 (with vertex set V) and verifier V^*:

1. Guess a bit b uniformly at random.
2. Sample a permutation $\pi : V \to V$ uniformly at random and send $\pi(G_b)$ to V^*.
3. Receive b' form V^*.
4. If $b = b'$, then output $(\pi(G_b), b, \pi)$ and terminate. Otherwise, restart at step 1.

Note that if $G_0 \simeq G_1$, then $\pi(G_b)$ is statistically independent of b because b and π are sampled uniformly. Thus, with probability $1/2$, V^* will output b so on average, two attempts will be needed before S terminates. It follows that S will terminate in *expected* polynomial time.

Since b is sampled uniformly at random, $\pi(G_b)$ is uniformly distributed with all graphs of the form $\sigma(G_1)$ where σ is sampled uniformly at random from permutations on V. Thus, the output $\pi(G_b)$ in our simulator will be identically distributed with the output H in our graph isomorphism protocol.

In step 3 of our graph isomorphism protocol, note that τ is distributed uniformly at random. This is because composing a uniformly random permutation with a fixed permutation will not change its distribution. Thus τ will be identically distributed with π in our simulator. It follows that the transcripts outputted by our simulator will be identically distributed with the transcripts produced by the graph isomorphism protocol.

6.4 Zero-Knowledge for NP

An n-coloring of a graph $G = (A, E)$ is a function $c : A \to \{1, \ldots, n\}$ such that if $(i, j) \in E$, then $c(i) \neq c(j)$. So we want to paint each vertex of a graph a certain color so that the endpoints of any edge are colored differently.

In the graph 3-coloring problem (3COL), we are given a graph and asked if there exists a 3-coloring. In this section, we will provide a computational zero knowledge proof for 3COL. It is a fact that 3COL is NP-complete, so any problem in NP has a polynomial time reduction to 3COL. Thus, by giving a zero knowledge proof for 3COL, we will show that there are zero knowledge proofs for all of NP.

We will first give a high-level description of a zero-knowledge protocol for 3COL. Suppose a prover P wants to convince a verifier V that his graph G is 3-colorable without revealing what the coloring c actually is. If the three colors we use are red, green, and blue, then note that if we colored all the red vertices blue, all the green vertices red, and all the blue vertices green, we would still have a valid 3-coloring. In fact, if ϕ was any permutation on the color set of red, green, and blue, then $\phi \circ c$ would be a valid 3-coloring of G.

P asks V to leave the room and then samples a random permutation ϕ of the three colors. He colors the vertices of G according to $\phi \circ c$, then covers all the vertices with cups. At this point, P invites V back into the room. V is allowed to pick one edge and then uncover the two endpoints of the edge. If the colors on the two endpoints are the same, then V rejects P's claim that the graph is 3-colorable.

If the colors on the two endpoints are different, then V leaves the room again, P samples ϕ randomly, and the process repeats itself. Certainly if G is actually 3-colorable, then V will never reject the claim. If G is not 3-colorable, then there will always be an edge with endpoints that are colored identically and V will eventually uncover such an edge.

Note that V does not gain any information on the coloring because it is masked by a (possibly) different random permutation every time V uncovers an edge. Of course this protocol depends on P not being able to quickly recolor the endpoints of an edge after removing the cups. This is why we need commitment schemes.

6.4.1 Commitment Schemes

We want to construct a protocol between a sender and a receiver where the sender sends a bit to the receiver, but the receiver will not know the value of this bit until the sender chooses to "open" the data that he sent. Of course, this protocol is no good unless the receiver can be sure that the sender was not able to change the value of his bit in between when the receiver first obtained the data and when the sender chose to open it.

Definition 6.5 *A* commitment scheme *is a PPT machine C taking input (b, r) that satisfies two properties:*

- *(perfect binding) For all r, s, we have $C(0, r) \neq C(1, s)$.*
- *(computational hiding) $\{C(0, U_n)\} \simeq_c \{C(1, U_n)\}$*

So for the sender to "open" the data, he just has to send his value of r to the receiver. We say that r is a *decommitment* for $C(x, r)$. Why do we require perfect binding instead of just statistical binding? If there existed even a single pair r, s where $C(0, r) = C(1, s)$, then the sender could cheat. If he wished to reveal a bit value of 0 then he could just offer r and if he wished to reveal a bit value of 1 then he could just offer s.

We can use injective one-way functions to construct commitment schemes.

Theorem 6.1 *If injective one-way functions exist, then so do commitment schemes.*

Proof. We can let f be an injective one-way function. Recall from Lecture 3 that $f'(x,r) := (f(x),r)$ will also be an injective one-way function with hard-core bit $B(x,r) := \langle x,r\rangle$. We claim that $C(b,x,r) := (f'(x,r), b \oplus B(x,r))$ is a commitment scheme.

If $(x,r) \neq (y,s)$ then $C(0,x,r) \neq C(0,y,s)$ because f' is injective. Since $C(0,x,r) = (f'(x,r), B(x,r)) \neq (f'(x,r), \overline{B(x,r)}) = C(1,x,r)$, then C satisfies perfect binding.

Suppose D can distinguish $C(0,U_n)$ from $C(1,U_n)$. Then we can distinguish $B(x,r)$ from $\overline{B(x,r)}$ given $f'(x,r)$ which contradicts the fact that $B(x,r)$ is a hard-core bit for $f'(x,r)$. Thus, C has the computational hiding property. ■

We can extend the definition of commitment schemes to hold for messages longer than a single bit. These commitment schemes will work by taking our commitment schemes for bits and concatenating them together. For the extended definition, we require that for any two messages m_0 and m_1 of the same length, the ensembles $\{C(m_0,U_n)\}$ and $\{C(m_1,U_n)\}$ are computationally indistinguishable.

6.4.2 3COL Protocol

Below we describe the protocol $P(x,z) \leftrightarrow V(x)$, where x describes a graph $G = (\{1,\ldots,n\}, E)$ and z describes a 3-coloring c:

1. P picks a random permutation $\pi : \{1,2,3\} \to \{1,2,3\}$ and defines the 3-coloring $\beta := \pi \circ c$ of G. Using a commitment scheme C for the messages $\{1,2,3\}$, P defines $\alpha_i = C(\beta(i), U_n)$ for each $i \in V$. P sends $\alpha_1, \alpha_2, \ldots, \alpha_n$ to V.

2. V uniformly samples an edge $e = (i,j) \in E$ and sends it to P.

3. P opens α_i and α_j.

4. V will accept only if it received valid decommitments for α_i and α_j, and if $\beta(i)$ and $\beta(j)$ are distinct and valid colors.

It is clear that this protocol is PPT. If G is not 3-colorable, then there will be at least a $1/|E|$ probability that V will reject P's claim in step 4. Since $|E| \leq n^2$ we can repeat the protocol polynomially many times to increase the rejection probability to at least $1/2$.

We will now show that this protocol is zero-knowledge. We describe a simulator S below, given a verifier V^*:

1. Sample an edge $e = (i,j) \in E$ uniformly at random.

2. Assign c_i and c_j to have distinct values from $\{1,2,3\}$ and do so uniformly at random. Set $c_k := 1$ for all $k \neq i,j$.

3. Compute n random keys $r_1, \ldots, r_n$ and set $\alpha_i = C(c_i, r_i)$ for all i.

4. Let $e' \in E$ be the response of V^* upon receiving $\alpha_1, \ldots, \alpha_n$.

5. If $e' \neq e$, then terminate and go back to step 1. Otherwise, proceed. If S returns to step 1 more than $2n|E|$ times, then output fail and halt the program.

6. Print $\alpha_1, \ldots, \alpha_n, e$, send r_i and r_j to V^* and then print whatever V^* responds with.

By construction, S will run in polynomial time. However, sometimes it may output a fail message. We will show that this occurs with negligible probability.

Suppose that for infinitely many graphs G, V^* outputs $e' = e$ in step 4 with probability less than $1/2|E|$. If this is true, then it is possible for us to break the commitment scheme C that we use in S. Consider a modified version of S called $\tilde{S}$, where in step 2 we set $c_i = 1$ for all i. Note that in this case, V^* cannot distinguish between any of the edges so the probability that it returns $e' = e$ is $1/|E|$.

If we gave V^* a set of commitments $\alpha_k = C(1, r_k)$ for random keys r_k, then we would be in the setting of $\tilde{S}$. If we gave V^* the commitments α_k but with two of the values set to $C(c, r)$ and $C(c', r')$ where c, c' are distinct random values from $\{1, 2, 3\}$ and r, r' are random keys, then we are in the setting of S. This implies that it possible to distinguish between these two commitment settings with a probability of at least $1/2|E|$ which is non-negligible. It follows that V^* outputs $e' = e$ with probability less than $1/2|E|$ for only finitely many graphs G.

Thus, the probability that S outputs fail in the end is less than $(1 - 1/2|E|)^{2n|E|} < 1/e^n$ which is negligible.

Now we need to argue that the transcripts generated by S are computationally indistinguishable from the transcripts generated by $P \leftrightarrow V^*$. Again, we consider a modified version of S, called S', given a 3-coloring of its input G as auxiliary input. In step 2 of the simulation, S' will choose a random permutation of the colors in its valid 3-coloring for the values of c_i rather than setting all but two values c_i and c_j equal to 1. Note that this is how our protocol between P and V behaves.

Observe that $P \leftrightarrow V^*$ is computationally indistinguishable from S' because S' outputs fail with negligible probability. Thus, it suffices to show that S and S' are computationally indistinguishable. Again, we will suppose otherwise and argue that as a result we can distinguish commitments.

We consider two messages m_0 and m_1 of the same length where m_0 consists of $n - 2$ instances of the message 1 and two committed colors c_i and c_j (for a random edge $(i, j) \in E$) and m_1 consists of a committed random 3-coloring of G (with a random edge $(i, j) \in E$) chosen. Observe that by feeding the former message to V^* we are in the setting of S' and by feeding the latter message to V^* we are in the setting of S. If we could distinguish those two settings, then we could distinguish the commitments for m_0 and m_1. This contradiction completes our argument that our 3-coloring protocol is zero-knowledge.

Exercises

Exercise 6.1 (Leaky ZK proof) *Formally define:*

1. *What it means for an interactive proof (P, V) to be **first-bit leaky** zero-knowledge, where we require that the protocol doesn't leak anything more than the first bit of the witness.*

2. *What it means for an interactive proof (P, V) to be **one-bit leaky** zero-knowledge, where we require that the protocol doesn't leak anything more than one bit that is an arbitrary adversarial chosen function of the witness.*

Exercise 6.2 (Proving OR of two statements) *Give a statistical zero-knowledge proof system $\Pi = (P, V)$ (with efficient prover) for the following language.*

$$L = \left\{ ((G_0, G_1), (G_0', G_1')) \,\middle|\, G_0 \simeq G_1 \bigvee G_0' \simeq G_1' \right\}$$

Caution: *Make sure the verifier doesn't learn which of the two pairs of graphs is isomorphic.*

Exercise 6.3 (ZK implies WI) *Let $L \in NP$ and let (P, V) be an interactive proof system for L. We say that (P, V) is* witness indistinguishable (WI) *if for all PPT V^*, for all $x \in L$, distinct witnesses $w_1, w_2 \in R_L(x)$ and auxiliary input $z \in \{0,1\}^*$, the following two views are computationally indistinguishable:*

$$View_{V^*}\left(P(x, w_1) \leftrightarrow V^*(x, z)\right) \simeq_c View_{V^*}\left(P(x, w_2) \leftrightarrow V^*(x, z)\right).$$

1. *Show that if (P, V) is an efficient prover zero-knowledge proof system, then it is also witness indistinguishable.*

2. *Assume (P, V) is an efficient prover zero-knowledge proof system. We have seen in the exercise that (P, V) is also witness indistinguishable. Define $(\tilde{P}, \tilde{V})$ to repeat (P, V) independently for k times* in parallel *(k is a polynomial), and $\tilde{V}$ accepts if and only if V accepts in all the executions. Prove that $(\tilde{P}, \tilde{V})$ is still witness indistinguishable.*

Chapter 7

Secure Computation

7.1 Introduction

Secure multiparty computation considers the problem of different parties computing a joint function of their separate, private inputs without revealing any extra information about these inputs than that is leaked by just the result of the computation. This setting is well motivated, and captures many different applications. Considering some of these applications will provide intuition about how security should be defined for secure computation:

Voting: Electronic voting can be thought of as a multi party computation between n players: the voters. Their input is their choice $b \in \{0, 1\}$ (we restrict ourselves to the binary choice setting without loss of generality), and the function they wish to compute is the majority function.

Now consider what happens when only one user votes: their input is trivially revealed as the output of the computation. What does privacy of inputs mean in this scenario?

Searchable Encryption: Searchable encryption schemes allow clients to store their data with a server, and subsequently grant servers tokens to conduct specific searches. However, most schemes do not consider access pattern leakage. This leakage tells the server potentially valuable information about the underlying plaintext. How do we model all the different kinds information that is leaked?

From these examples we see that defining security is tricky, with lots of potential edge cases to consider. We want to ensure that no party can learn anything more from the secure computation protocol than it can from just its input and the result of the computation. To formalize this, we adopt the **real/ideal paradigm**.

7.2 Real/Ideal Paradigm

Notation. Suppose there are n parties, and party P_i has access to some data x_i. They are trying to compute some function of their inputs $f(x_1, \ldots, x_n)$. The goal is to do this securely: even if some parties are corrupted, no one should learn more than is strictly necessitated by the computation.

Real World. In the real world, the n parties execute a protocol Π to compute the function f. This protocol can involve multiple rounds of interaction. The real world adversary $\mathcal{A}$ can corrupt arbitrarily many (but not all) parties.

Ideal World. In the ideal world, an angel helps in the computation of f: each party sends their input to the angel and receives the output of the computation $f(x_1, \ldots, x_n)$. Here the ideal world adversary $\mathcal{S}$ can again corrupt arbitrarily many (but not all) parties.

To model malicious adversaries, we need to modify the ideal world model as follows. Some parties are honest, and each honest party P_i simply sends x_i to the angel. The other parties are corrupted and are under control of the adversary $\mathcal{S}$. The adversary chooses an input x_i' for each corrupted party P_i (where possibly $x_i' \neq x_i$) and that party then sends x_i' to the angel. The angel computes a function f of the values she receives (for example, if only party 1 is honest, then the angel computes $f(x_1, x_2', x_3', \ldots, x_n')$) in order to obtain a tuple $(y_1, \ldots, y_n)$. She then sends y_i of corrupted parties to the adversary, who gets to decide whether or not honest parties will receive their response from the angel. The angel obliges. Each honest party P_i then outputs y_i if they receive y_i from the angel and $\perp$ otherwise, and corrupted parties output whatever the adversary tells them to.

Definition of Security. A protocol Π is secure against computationally bounded adversaries if for every PPT adversary $\mathcal{A}$ in the real world, there exists an PPT adversary $\mathcal{S}$ in the ideal world such that for all tuples of bit strings $(x_1, \ldots, x_n)$, we have

$$\text{Real}_{\Pi,\mathcal{A}}(x_1, \ldots x_n) \stackrel{c}{\simeq} \text{Ideal}_{F,\mathcal{S}}(x_1, \ldots, x_n)$$

where the left-hand side denotes the output distribution induced by Π running with $\mathcal{A}$, and the right-hand side denotes the output distribution induced by running the ideal protocol F with $\mathcal{S}$. The ideal protocol is either the original one described for semi-honest adversaries, or the modified one described for malicious adversaries.

Assumptions. We have brushed over some details of the above setting. Below we state these assumptions explicitly:

1. **Communication channel:** We assume that the communication channel between the involved parties is completely insecure, i.e., it does not preserve the privacy of the messages. However, we assume that it is reliable, which means that the adversary can drop messages, but if a message is delivered, then the receiver knows the origin.

2. **Corruption model:** We have different models of how and when the adversary can corrupt parties involved in the protocol:

   - *Static:* The adversary chooses which parties to corrupt before the protocol execution starts, and during the protocol, the malicious parties remain fixed.
   - *Adaptive:* The adversary can corrupt parties dynamically during the protocol execution, but the simulator can do the same.
   - *Mobile:* Parties corrupted by the adversary can be "uncorrupted" at any time during the protocol execution at the adversary's discretion.

3. **Fairness:** The protocols we consider are not "fair", i.e., the adversary can cause corrupted parties to abort arbitrarily. This can mean that one party does not get its share of the output of the computation.

4. **Bounds on corruption:** In some scenarios, we place upper bounds on the number of parties that the adversary can corrupt.

5. **Power of the adversary:** We consider primarily two types of adversaries:

   - *Semi-honest adversaries:* Corrupted parties follow the protocol execution Π honestly, but attempt to learn as much information as they can from the protocol transcript.
   - *Malicious adversaries:* Corrupted parties can deviate arbitrarily from the protocol Π.

6. **Standalone vs. Multiple execution:** In some settings, protocols can be executed in isolation; only one instance of a particular protocol is ever executed at any given time. In other settings, many different protocols can be executed concurrently. This can compromise security.

7.3 Oblivious transfer

Rabin's oblivious transfer sets out to accomplish the following special task of two-party secure computation. The sender has a bit $s \in \{0,1\}$. She places the bit in a box. Then the box reveals the bit to the receiver with probability $1/2$, and reveals $\perp$ to the receiver with probability $1/2$. The sender cannot know whether the receiver received s or $\perp$, and the receiver cannot have any information about s if they receive $\perp$.

7.3.1 1-out-of-2 oblivious transfer

1-out-of-2 oblivious transfer sets out to accomplish the following related task. The sender has two bits $s_0, s_1 \in \{0,1\}$ and the receiver has a bit $c \in \{0,1\}$. The sender places the pair (s_0, s_1) into a box, and the receiver places c into the same box. The box then reveals s_c to the receiver, and reveals $\perp$ to the sender (in order to inform the sender that the receiver has placed his bit c into the box and has been shown s_c). The sender cannot know which of her bits the receiver received, and the receiver cannot know anything about s_{1-c}.

Lemma 7.1 *A system implementing 1-out-of-2 oblivious transfer can be used to implement Rabin's oblivious transfer.*

Proof. The sender has a bit s. She randomly samples a bit $b \in \{0,1\}$ and $r \in \{0,1\}$, and the receiver randomly samples a bit $c \in \{0,1\}$. If $b = 0$, the sender defines $s_0 = s$ and $s_1 = r$, and otherwise, if $b = 1$, she defines $s_0 = r$ and $s_1 = s$. She then places the pair (s_0, s_1) into the 1-out-of-2 oblivious transfer box. The receiver places his bit c into the same box, and then the box reveals s_c to him and $\perp$ to the sender. Notice that if $b = c$, then $s_c = s$, and otherwise $s_c = r$. Once $\perp$ is revealed to the sender, she sends b to the receiver. The recieiver checks whether or not $b = c$. If $b = c$, then he knows that the bit revealed to him was s. Otherwise, he knows that the bit revealed to him was the nonsense bit r and he regards it as $\perp$.

It is easy to see that this procedure satisfies the security requirements of Rabin's oblivious transfer protocol. Indeed, as we saw above, $s_c = s$ if and only if $b = c$, and since the sender knows b, we see that knowledge of whether or not the bit s_c received by the receiver is equal to s is equivalent to knowledge of c, and the security requirements of 1-out-of-2 oblivious transfer prevent the sender from knowing c. Also, if the receiver receives r (or, equivalently, $\perp$), then knowledge of s is knowledge of the bit that was not revealed to him by the box, which is again prevented by the security requirements of 1-out-of-2 oblivious transfer. ■

Lemma 7.2 *A system implementing Rabin's oblivious transfer can be used to implement 1-out-of-2 oblivious transfer.*

Proof sketch. The sender has two bits $s_0, s_1 \in \{0,1\}$ and the receiver has a single bit c. The sender randomly samples $3n$ random bits $x_1, \ldots, x_{3n} \in \{0,1\}$. Each bit is placed into its own a Rabin oblivious transfer box. The ith box then reveals either x_i or else $\perp$ to the receiver. Let

$$S := \{i \in \{1, \ldots, 3n\} : \text{the receiver knows } x_i\}.$$

The receiver picks two sets $I_0, I_1 \subseteq \{1, \ldots, 3n\}$ such that $\#I_0 = \#I_1 = n$, $I_c \subseteq S$ and $I_{1-c} \subseteq \{1, \ldots, 3n\} \setminus S$. This is possible except with probability negligible in n. He then sends the pair (I_0, I_1) to the sender. The sender then computes $t_j = \left(\bigoplus_{i \in I_j} x_i\right) \oplus s_j$ for both $j \in \{0,1\}$ and sends (t_0, t_1) to the receiver.

Notice that the receiver can uncover s_c from t_c since he knows x_i for all $i \in I_c$, but cannot uncover s_{1-c}. One can show that the security requirement of Rabin's oblivious transfer implies that this system satisfies the security requirement necessary for 1-out-of-2 oblivious transfer. ■

We will see below that length-preserving one-way trapdoor permutations can be used to realize 1-out-of-2 oblivious transfer.

Theorem 7.1 *The following protocol realizes 1-out-of-2 oblivious transfer in the presence of computationally bounded and semi-honest adversaries.*

1. *The sender, who has two bits s_0 and s_1, samples a random length-preserving one-way trapdoor permutation (f, f^{-1}) and sends f to the receiver. Let $b(\cdot)$ be a hard-core bit for f.*
2. *The receiver, who has a bit c, randomly samples an n-bit string $x_c \in \{0,1\}^n$ and computes $y_c = f(x_c)$. He then samples another random n-bit string $y_{1-c} \in \{0,1\}^n$, and then sends (y_0, y_1) to the sender.*
3. *The sender computes $x_0 := f^{-1}(y_0)$ and $x_1 := f^{-1}(y_1)$. She computes $b_0 := b(x_0) \oplus s_0$ and $b_1 := b(x_1) \oplus s_1$, and then sends the pair (b_0, b_1) to the receiver.*
4. *The receiver knows c and x_c, and can therefore compute $s_c = b_c \oplus b(x_c)$.*

Proof. Correctness is clear from the protocol. For security, from the sender side, since f is a length-preserving permutation, (y_0, y_1) is statistically indistinguishable from two random strings, hence she can't learn anything about c. From the receiver side, guessing s_{1-c} correctly is equivalent to guessing the hard-core bit for y_{1-c}. ■

7.3.2 1-out-of-4 oblivious transfer

We describe how to implement a 1-out-of-4 OT using 1-out-of-2 OT:

1. The sender, P_1 samples 5 random values $S_i \leftarrow \{0,1\}, i \in \{1, \ldots, 5\}$.
2. P_1 computes

$$\begin{aligned}
\alpha_0 &= S_0 \oplus S_2 \oplus m_0 \\
\alpha_1 &= S_0 \oplus S_3 \oplus m_1 \\
\alpha_2 &= S_1 \oplus S_4 \oplus m_2 \\
\alpha_3 &= S_1 \oplus S_5 \oplus m_3
\end{aligned}$$

It sends these values to P_2.

3. The parties engage in 3 1-out-of-2 Oblivious Transfer protocols for the following messages: (S_0, S_1), (S_2, S_3), (S_4, S_5). THe receiver's input for the first OT is the first choice bit, and for the second and third ones is the second choice bit.

4. The receiver can only decrypt one ciphertext.

7.4 Yao's Two Party Computation Protocol

Yao's Two Party Protocol is a protocol conducted between two parties for computing any function. It is obtained by combining two primitives: a scheme for garbling circuits and oblivious transfer. Informally, the idea is that one party (say P_1) *garbles* a circuit. This involves assigning random labels to each wire of the circuit, including the input and output wires. P_1 then sends the garbled circuit and the labels corresponding to its input wires to P_2. The two parties then engage in an oblivious transfer protocol to transfer the labels corresponding to P_2's inputs to P_2. P_2 then evaluates the garbled circuit using the two sets of input labels to get the result of the computation.

Problem Setup: Let G and W be the gates and wires respectively in $\mathcal{U}$, the universal circuit. Let w_i be the i^{th} input wire, $i \in [n]$, and w_{out} be the output wire.

Definition 7.1 (Garbling Scheme) *A* garbling scheme *is a pair of ppt. algorithms* $(\mathsf{Garble}, \mathsf{Eval})$*:*

- $\mathsf{Garble}(1^\kappa, C) \to \left(\widetilde{C}, \{\mathsf{lab}_{i,b_i}\}_{i\in[n], b_i\in\{0,1\}}\right)$. *The circuit C has n input wires and one output wire.*

- $\mathsf{Eval}(1^\kappa, \widetilde{C}, \{\mathsf{lab}_{i,x_i}\}_{i\in[n]}) \to y$.

It satisfies the following two properties:

Correctness: $\forall\ C, x$*, we have that*

$$\Pr[(\widetilde{C}, \mathsf{lab}_{i,b_i}) \leftarrow \mathsf{Garble}(1^\kappa, C) \wedge y \leftarrow \mathsf{Eval}(1^\kappa, \widetilde{C}, \mathsf{lab}_{i,x_i}) \wedge y = C(x)] = 1$$

Security: $\exists$ *simulator* $\mathcal{S}$ *s.t.* $\forall\ C, x$*, we have that*

$$(\widetilde{C}, \{\mathsf{lab}_{i,x_i}\}) \overset{c}{\approx} \mathcal{S}(1^\kappa, C(x))$$

7.4.1 Construction:

We construct a garbling scheme:

$\mathsf{Garble}(1^\kappa, C)$:
1. For each $w \in W$, sample (k_w^0, k_w^1) as encryption keys.
2. For each $g \in G$ with input wires w_0, w_1, and output wire w_2, set

$$e_g := \left(\mathsf{Enc}_{k_{w_0}^a}\left(\mathsf{Enc}_{k_{w_1}^b}\left(0^\kappa || k_{w_2}^{g(a,b)}\right)\right)\right)_{a,b\in\{0,1\}}$$

3. Set $\widetilde{C} := \left((e_g)_{g\in G}, k_{w_{\mathsf{out}}}^0 \to 0, k_{w_{\mathsf{out}}}^1 \to 1\right)$.
4. Set $\mathsf{lab}_{i,b_i} := k_{w_i}^{b_i}$.
5. Output $(\widetilde{C}, \mathsf{lab}_{i,b_i})$.

$\mathsf{Eval}(1^\kappa, \widetilde{C}, \{\mathsf{lab}_{i,x_i}\}_{i\in[n]})$:
1. For each gate $g \in G$, obtain $\mathsf{lab}_{w_2} \leftarrow \mathsf{Dec}_{\mathsf{lab}_{w_0}}(\mathsf{Dec}_{\mathsf{lab}_{w_1}}(e_g))$.

Figure 7.1: Definition of a garbling scheme

7.4.2 Proof of Security

Proof. We construct a ppt. simulator Sim such that $\forall x \in \{0,1\}^n$:

$$\left\{\widetilde{C}, \mathsf{lab}_{i,x_i}\right\} \overset{c}{\approx} \{\mathsf{Sim}(1^\kappa, C(x))\}$$

$\mathsf{Sim}(1^\kappa, C(x))$:
1. Sample random k_w, k'_w $\forall w \in W$.
2. For each $g \in G$ with input wires w_0, w_1 and output wire w_2, set e_g as
   - $\mathsf{Enc}_{k_{w_0}}(\mathsf{Enc}_{k_{w_1}}(0^\kappa || k_{w_2}))$
   - $\mathsf{Enc}_{k_{w_0}}(\mathsf{Enc}_{k'_{w_1}}(0^\kappa || k_{w_2}))$
   - $\mathsf{Enc}_{k'_{w_0}}(\mathsf{Enc}_{k_{w_1}}(0^\kappa || k_{w_2}))$
   - $\mathsf{Enc}_{k'_{w_0}}(\mathsf{Enc}_{k'_{w_1}}(0^\kappa || k_{w_2}))$
3. Output $\widetilde{C} := \left((e_g)_{g\in G}, k_{w_{\mathsf{out}}} \to C(x), k'_{w_{\mathsf{out}}} \to 1 - C(x)\right)$.

Figure 7.2: Simulator for security of garbled circuits

We prove that the output of this simulator is indistinguishable from the actual view of the circuit evaluator via a series of hybrids:

$$H_0 = \left\{\widetilde{C}, \mathsf{lab}_{i,x_i}\right\} \overset{c}{\approx} H_1 \overset{c}{\approx} \cdots \overset{c}{\approx} H_{T-1} \overset{c}{\approx} \{\mathsf{Sim}(1^\kappa, C(x))\} = H_T$$

We proceed by replacing e_g's gate by gate. The idea is to replace the three non-opened entries with encryptions of the only wire which is correct. Say the input labels are $k_{w_0}^0, k_{w_1}^1$. Originally, e_g consists of

$$\mathsf{Enc}_{k_{w_0}^0}\left(\mathsf{Enc}_{k_{w_1}^0}\left(0^\kappa || k_{w_2}^{g(0,0)}\right)\right)$$
$$\mathsf{Enc}_{k_{w_0}^0}\left(\mathsf{Enc}_{k_{w_1}^1}\left(0^\kappa || k_{w_2}^{g(0,1)}\right)\right)$$
$$\vdots$$

For each $a, b \in \{0,1\}$, we replace the encrypted values with $\mathsf{Enc}_{k_{w_0}^a}\left(\mathsf{Enc}_{k_{w_1}^b}\left(0^\kappa || k_{w_2}^{g(0,1)}\right)\right)$. By the semantic security of the encryption scheme, this new e'_g is indistinguishable from the original e_g. Hence each hybrid is indistinguishable from the previous one. ■

7.5 GMW Protocol

Yao's protocol is limited to two party computation. Goldreich, Micali and Wigderson (GMW) created the first secure multiparty computation protocol. Here we give an informal sketch of the protocol, limiting ourselves to two parties for simplicity of exposition.

Let P_1's input be x_1, and P_2's input be x_2. Each party creates a secret share of their input and sends it to the other party. For example, P_1 samples a random a, and computes $b_1 = a_1 \oplus x_1$. P_2 does the same for x_2. P_1 gets the a shares, and P_2 gets the b shares. Computing NOT, XOR, and AND gates is done as follows:

- NOT gates: Each party simply flips their share of the input bit.
- XOR gates: Since $x_1 \oplus x_2 = (a_1 \oplus b_1) \oplus (a_2 \oplus b_2) = (a_1 \oplus a_2) \oplus (b_1 \oplus b_2)$, each party can simply XOR their shares separately.
- AND gates: $(a_1 \oplus b_1) \cdot (a_2 \oplus b_2) = ((a_1 \cdot a_2) \oplus (b_1 \cdot b_2)) \oplus ((a_1 \cdot b_2) \oplus (a_2 \cdot b_1))$. We see that each party can compute the first parts with the shares they possess. However, to get the remaining, they need to know the other party's shares, which compromises security. So instead, the parties utilize 1-out-of-4 oblivious transfer to compute the AND gate.

  The setup is as follows: P_1 samples a random bit a. This its share of the result of the gate. It computes the following table:

b_1	b_2	b_3
0	0	$(a_1 \cdot a_2) \oplus a$
0	1	$(a_1 \cdot \neg a_2) \oplus a$
1	0	$(\neg a_1 \cdot a_2) \oplus a$
1	1	$(\neg a_1 \cdot \neg a_2) \oplus a$

  The intuition for the entries in the table is that when $b_1 = b_2 = 1$, the AND gate becomes $(a_1 \oplus b_1) \cdot (a_2 \oplus b_2) = (a_1 \oplus 1) \cdot (a_2 \oplus 1) = (\neg a_1) \cdot (\neg a_2)$. The other entries are computed similarly.

  P_1 computes each of the possible values of b_3, and feeds them as input to the OT. P_2 feeds in (b_1, b_2) to the OT, and gets some secret share. Thus both parties possess a share of the correct output.

7.6 Malicious attacker intead of semi-honest attacker

The assumption we had before consisted of a semi-honest attacker instead of a malicious attacker. A malicious attacker does not have to follow the protocol, and may instead alter the original protocol. The main idea here is that we can convert a protocol aimed at semi-honest attackers into one that will work with malicious attackers.

At the beginning of the protocol, we have each party commit to its inputs: Given a commitment protocol *com*, Party 1 produces

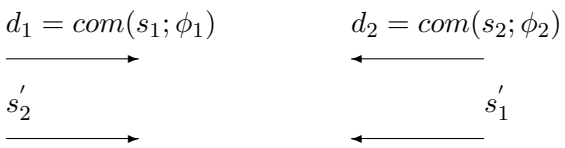
$$c_1 = com(x_1; w_1)$$
$$d_1 = com(r_1; \phi_1)$$

Party 2 produces

$$c_2 = com(x_2; w_2)$$
$$d_2 = com(r_2; \phi_2)$$

We have the following guarantee: $\exists x_i, r_i, w_i, \phi_i$ such that $c_i = com(x_i; w_i) \wedge d_i = com(r_i; \phi_i) \wedge t = \pi(i, \text{transcript}, x_i, r_i)$, where transcript is the set of messages sent in the protocol so far.

Here we have a potential problem. Since both parties are choosing their own random coins, we have to be able to enforce that the coins are *indeed* random. We can solve this by using the following protocol:



We calculate $r_1 = s_1 \oplus s_1'$, and $r_2 = s_2 \oplus s_2'$. As long as one party is picking the random coins honestly, both parties would have truly random coins.

Furthermore, during the first commitment phase, we want to make sure that the committing party actually knows the value that is being committed to. Thus, we also attach along with the commitment a zero-knowledge proof of knowledge (ZK-PoK) to prove that the committing party knows the value that is being committed to.

7.6.1 Zero-knowledge proof of knowledge (ZK-PoK)

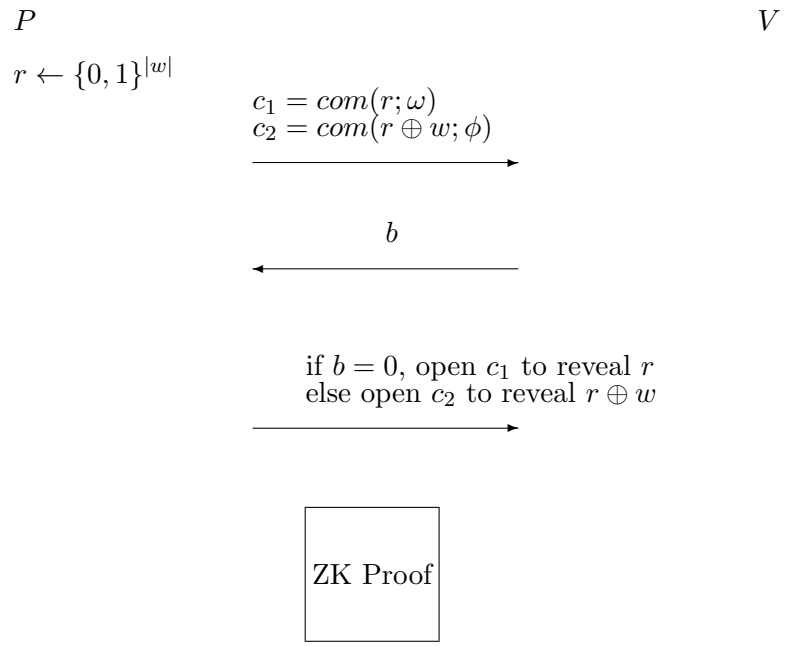
Definition 7.2 (ZK-PoK) *Zero-knowlwedge proof of knowledge (ZK-PoK) is a zero-knowledge proof system (P, V) with the property proof of knowledge with knowledge error κ:*

$\exists$ a PPT E (knowledge extractor) such that $\forall x \in L$ and $\forall P^*$ (possibly unbounded), it holds that if $\Pr[Out_V(P^*(x, w) \leftrightarrow V(x))] > \kappa(x)$, then

$$\Pr[E^{P^*}(x) \in R(x)] \geq \Pr[Out_V(P^* \leftrightarrow V(x))] = 1] - \kappa(x).$$

Here we have L be the language, R be the relation, and $R(x)$ is the set such that $\forall w \in R(x)$, $(x, w) \in R$.

Given a zero-knowledge proof system, we can construct a ZK-PoK system for statement $x \in L$ with witness w as follows:



The last ZK proof proves that $\exists r, w, \omega, \phi$ such that $(x, w) \in R$ and $c_1 = com(r;\omega)$, $c_2 = com(r \oplus w;\phi)$.

Exercises

Exercise 7.1 *Given a (secure against malicious adversaries) two-party secure computation protocol (and nothing else) construct a (secure against malicious adversaries) three-party secure computation protocol.*

Chapter 8

Secure RAM Computation

8.1 Oblivious RAM

Roughly speaking, an ORAM enables executing a RAM program while hiding the access pattern to the memory. ORAM have several fundamental applications. For example, imagine a client has a huge memory/database D. He wants to (encrypt and) store it on the server in such a way that later he can request and get access to a specific location of the database $D[i]$ by communicating with the server without leaking any information of the location i to the server.

Definition 8.1 *An ORAM scheme* $\mathsf{O} = (\mathsf{DGen}, \mathsf{LGen})$ *consists of the following:*

- $\mathsf{DGen}\left(1^{\kappa}, D\right) \rightarrow \left(\tilde{D}, \mathsf{sk}\right)$ *given the security parameter and the initial database outputs an oblivious database and a secret key stored by the client, where* $|\mathsf{sk}| = \mathcal{O}\left(\mathsf{polylog}\left(|D|\right)\right)$.
- $\mathsf{LGen}\left(\mathsf{sk}, \ell_1, \cdots, \ell_T\right) \rightarrow \left(\ell_1, \cdots, \ell_{T'}\right)$ *given memory access locations of* D *outputs memory access locations of* $\tilde{D}$. *Note* $T' = \mathcal{O}\left(T \cdot \mathsf{polylog}\left(|D|\right)\right)$.

Provided $\mathsf{sk}, \tilde{D}\left[\ell_1\right], \cdots, \tilde{D}\left[\ell_{T'}\right]$ *the client is able to recover* $D\left[\ell_1\right], \cdots, D\left[\ell_T\right]$. *Furthermore, for any* $\left(\ell_1, \cdots, \ell_T\right)$ *and* $\left(\ell'_1, \cdots, \ell'_T\right)$ *it satisfies*

$$\mathsf{LGen}\left(\mathsf{sk}, \ell_1, \cdots, \ell_T\right) \approx_s \mathsf{LGen}\left(\mathsf{sk}, \ell'_1, \cdots, \ell'_T\right).$$

In the following we first describe an ORAM construction where the client has storage of $\frac{n}{\alpha}$ for some constant α, where n is the size of D. Then we will show this basic scheme suffices for constructing an ORAM scheme where the client has only $\mathsf{polylog}\left(n\right)$ storage.

A Basic Construction. Assume the client has storage of $\frac{n}{\alpha}$. First DGen splits the database D into $\frac{n}{\alpha}$ blocks, each of size α. Then it samples a "position" for each block uniformly at random from $\left[\frac{n}{\alpha}\right]$, as in Figure 8.1. The position map is stored at the client.

An ORAM tree is then created as in Figure 8.2. It is a binary tree, each node in which is associated with a bucket which stores (at most) K tuples (b, pos, v) where v is the content of block b and pos is the leaf associated with the block b. K is a parameter that will determine the security of the ORAM.

When reading (or writing to) a memory block b, the client first requests the server for the entire path of pos in the ORAM tree, then generates a new random position pos' for b, deletes the old tuple (b, pos, v) from the path, adds to the root a new tuple $(b, pos', v(\text{or } v'))$, and sends the entire path back to the server. After this, there is a *flush* step, in which the client requests for a random path, and pushes each tuple in the path down as far as possible.

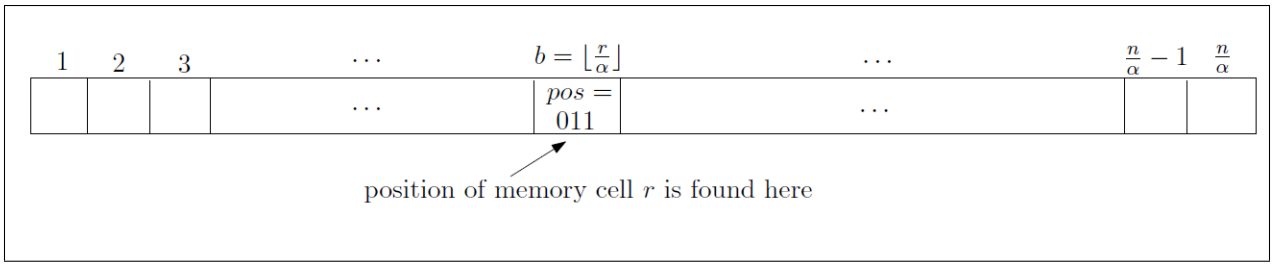


Figure 8.1: Position Map

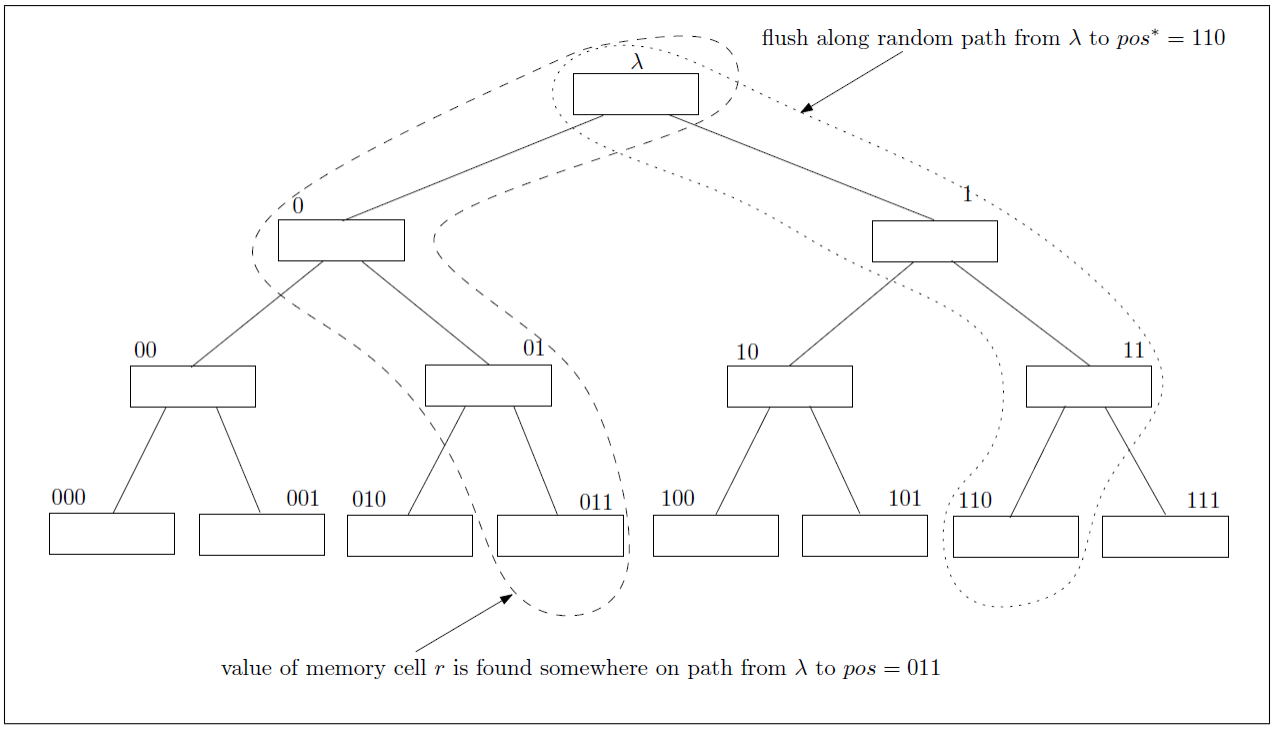


Figure 8.2: The ORAM Tree

Security Proof. It is clear that the access patterns are hidden in the above construction, since every read/write/flush requests a uniformly random path. We only need to argue that the probability of *overflow* (meaning that at any time a node in the ORAM tree contains more than K tuples) is negligible.

Consider a dart game: you have an unbounded number of white and black darts. In each round of the game, you first throw a black dart, and then a white dart; each dart independently hits the bullseye with probability p. You continue the game until at least K darts have hit the bullseye. You "win" if none of darts that hit the bullseye are white. The winning probability is upper bounded by 2^{-K}.

Suppose there is a tree node γ containing more than K tuples at some point of time. Among the K tuples at γ, WLOG assume at least $K/2$ tuples has *pos* with prefix $\gamma||0$. Think of black darts hitting bullseye as assigning a memory block to a leaf *pos* with prefix $\gamma||0$, and white darts hitting bullseye as performing a flushing associated with a leaf *pos* with prefix $\gamma||0$. By the union bound the probability of overflow is upper bounded by $T2^{-K}$.

The Full Scheme. Given an ORAM scheme where the client has storage of size $\frac{n}{\alpha}$ for some constant α, the client can apply ORAM again on the smaller memory of size $\frac{n}{\alpha}$. After $\log(n)$ iterations, the client ends up needing storage of size $\mathsf{polylog}(n)$.

8.2 Introduction to Secure RAM Computation

The goal of secure RAM Computation is to run a program P which accesses a shared database D without revealing its access patterns. We can leverage Oblivious RAM to do this, since ORAM allows us to compile P into $\widetilde{P}$ and D into $\widetilde{D}$ and have $\widetilde{P}$ accesses to $\widetilde{D}$ not reveal P's accesses to D. Given, this we can construct a solution that doesn't hide access patterns, and let ORAM handle hiding access patterns.

We want to avoid direct circuit conversion of the RAM program, since this greatly increases complexity. If a RAM program runs in time $O(T)$, then its circuit runs in $O(T^3)$. If its database accesses are $O(D)$, then the circuit's accesses scale with T and D, because the circuit needs to take D as input. Our goal is to do this in polylog time.

An important thing to keep in mind is that we don't know what has to be read beforehand when working with a RAM program. If we did, then the circuit complexity wouldn't be bad, and we could just use that. The motivating example for the notes can be a binary search on D shared between two parties, where we want to find the solution in polylog time without revealing the query.

RAM Program. A RAM program can be looked at as a series of CPU steps, each which accesses the database (either a read or write). Each CPU step changes the state for the subsequent step.

8.2.1 Secret-Shared Database Construction

One way we can do this is by secret sharing each bit of the database D. A simple way to do this by defining the database as such:

Definition 8.2 $D = \{(i, b_i)\}_{i \in |D|}$

Both parties receive $\{x_i\}$ and $\{y_i\}$ respectively, where $\forall i, x_i \oplus y_i = b_i$.

This construction has high round complexity. From an information theoretic perspective, to have small round complexity, one person must have all data. This is because of the reason mentioned above: we don't know what value to read (we don't know what's in memory to be able to read the next value at a specific CPU step). We'll discuss how to improve this in the next section.

8.2.2 Encrypted Database Construction

To give one person the data, we can have one person own the the encrypted version of the database. The other person can have the key to the database. Formally:

Definition 8.3 *$D_i = Enc_K(b_i)$ given to one user as y_i, and K given as x_i.*

They can still only access the database together. The same protocol as the secret-sharing case applies, just replacing x_i and y_i as mentioned in the definition. However, this doesn't quite work, since we don't really have security here (y_i's owner can figure out D_i since K is sent over by the other person).

Goal: We want to implicitly decrypt without the encrypted database's owner finding out about D_i. We can garble each CPU step separately:

Definition 8.4 $Garble(CPU_i) = \widetilde{CPU_i}$

$\widetilde{CPU_i}$ will output labels for $\widetilde{CPU_{i+1}}$. We create a κ-bit key sampled randomly.

Definition 8.5 $S = \{0,1\}^\kappa$ *randomly sampled.*

In the database, $\forall i$, we define $e_i = PRF_S(i||b_i)$, and store e_is in the database. This PRF is hardcoded into each of the $\widetilde{CPU_i}$s, so each garbled circuit can use it.

Focusing on just $\widetilde{CPU_1}$ and $\widetilde{CPU_2}$, we can how the labels for the circuits are transmitted forward. State labels are known, so $\widetilde{CPU_1}$ can easily provide those to $\widetilde{CPU_2}$. However, l_0 and l_1 need to be emitted based on what b_i is (emit l_{b_i}). To do this, $\widetilde{CPU_1}$ can emit $Enc_{PRF_S(i||0)}(l_0)$ and $Enc_{PRF_S(i||1)}(l_1)$ for l_0 and l_1 respectively.

We can then simulate these garbled circuits to evaluate security. Since garbled circuits require an output value for simulation, and only the last CPU step has an output, we can essentially replace each middle garbled circuit with "garbage" until the last circuit emits. However, a problem is that in garbled circuit simulation, only one label can be known (the one corresponding to l_{b_i}). The other should be indistinguishable from a random string. While this is trivially true for $\widetilde{CPU_1}$, for subsequent steps (say $\widetilde{CPU_2}$), it is not. $PRF_S(i||0)$ and $PRF_S(i||1)$ are known to all circuits, since S is harcoded into each circuit, as mentioned before. We need to expunge one label to be able to simulate.

Note: This is actually a circular argument at the moment. The garbled circuit depends on PRF_S to be random, and PRF_S requires the GC to be secure, since S is hardcoded into the circuit.

8.2.3 Fixed Construction

To fix the problem that everyone knows S, we can have more than one key S. Let's make a tree of keys, where keys are denoted $S_{i,j}$ with i as depth and j as their index within the depth.

In this tree, each child key is encrypted with its parent key (eg. $S_{1,0}$ is encrypted with $S_{0,0}$). If you know a key, you can decrypt its immediate children. The leaf nodes, $S_{d,i}$ for $i \in 0, |D| - 1$, encrypt the bits of the database D. With the root key, you can follow the path down to any leaf and read its corresponding $b_i \in D$.
Now, for every $\widetilde{CPU_i}$ we had before, we create $lg(|D|)$s (we have logarithmically more CPU steps). Each of these logarithmically many CPU steps (shown below) will recreate this path through the key tree.

Each circuit will also replace the key it used with a fresh key (a new randomly sampled key) after it uses it once, and recreate the corresponding ciphertexts. This means that all siblings along the path (in addition to the path) will need to be recreated. Fixing these siblings is logarithmic time. This fixing step maintains the invariant that after each step, the subsequent step has a fresh copy of D as if it was never read.

More Details on Key Generation

The key tree requires that each node encrypt its children under itself. There are two definitions that will help us explain the key tree usage.

Definition 8.6 *$e_{i,j} = \{PRF_{S_{i,j}}(k||(S_{i+1,2j}||S_{i+1,2j+1})_k)\}_{k\in 2\kappa}$, where k is the current key (of length κ) and $e_{i,j}$ is the encryption of the children under k.*

Definition 8.7 *$CPU_{i,j}$ ($i \in \{1...T\}, j \in \{1...d\}$) is a CPU step from the logarithmically expanded CPU step garbled circuits we mentioned before. $CPU_{i,j}$ sends labels to $CPU_{i,j+1}$, unless $j = T$, in which case, it sends labels to $CPU_{i+1,0}$.*

With this in mind, we can pre-generate some keys that we know we will need. Every $CPU_{t,1}$ and $CPU_{t,2}$, $t \in 1...T$ requires a fresh $S_{0,0}$ key, so we can hardcode these into those CPU step circuits beforehand. Similarly, each circuit has two fresh keys encoded, one to encrypt the labels it must pass on to the next circuit and one for which is the fresh encryption key used by its predecessor to encrypt the labels given to it. Now, to understand the process, we'll focus on 2 CPU steps: $CPU_{1,2}$ and $CPU_{1,3}$.

For simplicity, let's assume $S_{1,0}$ encrypts the correct label to send. Then, $CPU_{1,2}$ outputs $c = \{PRF_{S^{new}_{0,0}}(k||(S^{new}_{1,0}||S_{1,1}))_k\}_{k\in 2\kappa}$. For the case where $S_{1,1}$ is the correct key, simply swap $S_{1,0}$ and $S_{1,1}$. Note that in the figure above, $CPU_{1,2}$ has the S^{new}_1 key. This key will become $S^{new}_{1,0}$ or $S^{new}_{1,1}$, depending on which one to send. Similarly, $CPU_{1,3}$ has S^{new}_2 to use to encrypt the labels it sends. It also has S^{new}_1 to decrypt what $CPU_{1,2}$ sends it.

Chapter 9

Witness Encryption

9.1 A Story

Imagine that a billionaire who loves mathematics, would like to award with 1 million dollars the mathematician(s) who will prove the Riemann Hypothesis. Of course, neither does the billionaire know if the Riemann Hypothesis is true, nor if he will be still alive (if and) when a mathematician will come up with a proof. To overcome these couple of problems, the billionaire decides to:

1. Put 1 million dollars in gold in a big treasure chest.
2. Choose an arbitrary place of the world, dig up a hole, and hide the treasure chest.
3. Encrypt the coordinates of the treasure chest in a message so that only the mathematician(s) who can actually prove the Riemann Hypothesis can decrypt it.
4. Publish the ciphertext in every newspaper in the world.

The goal of this lecture is to help the billionaire with step 3. To do so, we will assume for simplicity that the proof is at most 10000 pages long. The latter assumption implies that the language

$$L = \{x \text{ such that } x \text{ is an acceptable Riemann Hypothesis proof}\}$$

is in NP and therefore, using a reduction, we can come up with a circuit C that takes as input x and outputs 1 if x is a proof for the Riemann Hypothesis and 0 otherwise.

Our goal now is to design a pair of PPT machines (Enc, Dec) such that:

1. $\text{Enc}(C, m)$ takes as input the circuit C and $m \in \{0,1\}$ and outputs a ciphertext $e \in \{0,1\}^*$.
2. $\text{Dec}(C, e, w)$ takes as input the circuit C, the cipertext e and a witness $w \in \{0,1\}^*$ and outputs m if if $C(w) = 1$ or $\perp$ otherwise.

and so that they satisfy the following correctness and security requirements:

- **Correctness:** If $\exists w$ such that $C(w) = 1$ then $\text{Dec}(C, e, w)$ outputs m.
- **Security:** If $\nexists w$ such that $C(w) = 1$ then $\text{Enc}(C, 0) \approx^c \text{Enc}(C, 1)$ (where $\approx^c$ means "computationally indistinguishable").

9.2 A Simple Language

As a first example, we show how we can design such an encryption scheme for a simple language. Let G be a group of prime order and g be a generator of the group. For elements $A, B, T \in G$ consider the language $L = \{(a, b) : A = g^a, B = g^b, T = g^{ab}\}$. An encryption scheme for that language with the correctness and security requirements of Section 9.1 is the following:

- **Encryption**(g, A, B, T, G)**:**
  - Choose elements $r_1, r_2 \in \mathbb{Z}_p^*$ uniformly and independently.
  - Let $c_1 = A^{r_1} g^{r_2}$, $c_2 = g^m T^{r_1} B^{r_2}$, where $m \in \{0, 1\}$ is the message we want to encrypt.
  - Output $c = (c_1, c_2)$
- **Decryption**(b)**:**
  - Output $\frac{c_2}{c_1^b}$

Correctness: The correcntess of the above encryption scheme follows from the fact that if there exist $(a, b) \in L$ then:

$$\begin{aligned} \frac{c_2}{c_1^b} &= \frac{g^m T^{r_1} B^{r_2}}{\left(A^{r_1} g^{r_2}\right)^b} \\ &= \frac{g^m \left(g^{ab}\right)^{r_1} \left(g^b\right)^{r_2}}{\left(g^a\right)^{r_1 b} g^{r_2 b}} \\ &= g^m \end{aligned}$$

Since $m \in \{0, 1\}$ and we know g, the value of g^m implies the value of m.

Security: As far as the security of the scheme is concerned, since L is quite simple, we can actually prove that m is information-theoretically hidden. To see this, assume there does not exist $(a, b) \in L$, but an adversary has the power to compute discrete logarithms. In that case, given c_1 and c_2 the adversary could get a system of the form:

$$\begin{aligned} a r_1 + r_2 &= s_1 \\ m + r r_1 + b r_2 &= s_2 \end{aligned}$$

where s_1 and s_2 are the discrete logarithms of c_1 and c_2 respectively (with base g), and $r \neq ab$ is an element of $\mathbb{Z}_p^*$ such that $T = g^r$. Observe now that for each value of m there exist numbers r_1 and r_2 so that the above system has a solution, and thus m is indeed information-theoretically hidden (on the other hand, if we had that $ab = r$ then the equations are linearly dependent).

9.3 An NP Complete Language

In this section we focus on our original goal of designing an encryption for an NP complete language L. Specifically, we will consider the NP-complete problem *exact cover*. Besides that, we introduce the n-Multilinear Decisional Diffie-Hellman (n-MDDH) assumption and the Decisional Multilinear No-Exact-Cover Assumption. The latter will guarantee the security of our construction.

9.3.1 Exact Cover

We are given as input $x = (n, S_1, S_2, \ldots, S_l)$, where n is an integer and each $S_i, i \in [l]$ is a subset of $[n]$, and our goal is to find a subset of indices $T \subseteq [l]$ such that:

1. $\cup_{i \in T} S_i = [n]$ and
2. $\forall i, j \in T$ such that $i \neq j$ we have that $S_i \cap S_j = \emptyset$.

If such a T exists, we say that T is an exact cover of x.

9.3.2 Multilinear Maps

Mutlinear maps is a generalization of bilinear maps (which we have already seen) that will be useful in our construction. Specifically, we assume the existence of a group generator $\mathcal{G}$, which takes as input a security parameter λ and a positive integer n to indicate the number of allowed operations. $\mathcal{G}(1^\lambda, n)$ outputs a sequence of groups $\vec{\mathbb{G}} = (\mathbb{G}_1, \mathbb{G}_2, \ldots, \mathbb{G}_n)$ each of large prime order $P > 2^\lambda$. In addition, we let g_i be a canonical generator of $\mathbb{G}_i$ (and is known from the group's description).

We also assume the existence of a set of bilinear maps $\{e_{i,j} : \mathbb{G}_i \times \mathbb{G}_j \to \mathbb{G}_{i+j} \mid i, j \geq 1; i+j \leq n\}$. The map $e_{i,j}$ satisfies the following relation:

$$e_{i,j}\left(g_i^a, g_j^b\right) = g_{i+j}^{ab} : \forall a, b \in \mathbb{Z}_p \tag{9.1}$$

and we observe that one consequence of this is that $e_{i,j}(g_i, g_j) = g_{i+j}$ for each valid i, j.

9.3.3 The n-MDDH Assumption

The n-Multilinear Decisional Diffie-Hellman (n-MDDH) problem states the following: A challenger runs $\mathcal{G}(1^\lambda, n)$ to generate groups and generators of order p. Then it picks random $s, c_1, \ldots, c_n \in \mathbb{Z}_p$. The assumption then states that given $g = g_1, g^s, g^{c_1}, \ldots, g^{c_n}$ it is hard to distinguish $T = g_n^{s \prod_{j \in [1,n]} c_j}$ from a random group element in G_n, with better than negligible advantage (in security parameter λ).

9.3.4 Decisional Multilinear No-Exact-Cover Assumption

Let $x = (n, S_1, \ldots, S_l)$ be an instance of the exact cover problem that has no solution. Let param $\leftarrow \mathcal{G}(1^{1+n}, n)$ be a description of a multilinear group family with order $p = p(\lambda)$. Let $a_1, a_2, \ldots, a_n, r$ be uniformly random in $\mathbb{Z}_p$. For $i \in [l]$, let $c_i = g_{|S_i|}^{\prod_{j \in S_i} a_j}$. Distiguish between the two distributions:

$$(\text{params}, c_1, \ldots, c_l, g_n^{a_1 a_2 \cdots a_n}) \text{ and } (\text{params}, c_1, \ldots, c_l, g_n^r)$$

The Decisional Multilinear No-Exact-Cover Assumption is that for all adversaries $\mathcal{A}$, there exists a fixed negligible function $\nu(\cdot)$ such that for all instances x with no solution, $\mathcal{A}$'s distinguishing advantage against the Decisional Multilinear No-Exact-Cover Problem for x is at most $\nu(\lambda)$.

9.3.5 The Encryption Scheme

We are now ready to give the description of our encryption scheme.

- $\text{Enc}(x, m)$ takes as input $x = (n, S_1, \ldots, S_l)$ and the message $m \in \{0, 1\}$ and:
  - Samples $a_0, a_1, \ldots, a_n$ uniformly and independently from $\mathbb{Z}_p^*$.
  - $\forall i \in [l]$ let $c_i = g_{|S_i|}^{\prod_{j \in S_j} a_j}$
  - Sample uniformly an element $r \in \mathbb{Z}_p^*$
  - Let $d = d(m)$ be $g_n^{\prod_{j \in [n]} a_j}$ if $m = 1$ or g_n^r if $m = 0$.
  - Output $c = (d, c_1, \ldots, c_l)$
- $\text{Dec}(x, T)$, where $T \subseteq [l]$ is a set of indices, computes $\prod_{i \in T} c_i$ and outputs 1 if the latter value equals to d or 0 otherwise.
- **Correctness:** Assume that T is an exact cover of x. Then, it is not hard to see that:

$$\begin{aligned} \prod_{i \in T} c_i &= \prod_{i \in T} g_{|S_i|}^{\prod_{j \in S_j} a_j} \\ &= g_n^{\prod_{j \in [n]} a_j} \end{aligned}$$

  where we have used (9.1) repeatedly and the fact that T is an exact cover (to show that $\sum_{i \in T} |S_i| = n$ and that $\prod_{i \in T} \prod_{j \in S_i} a_j = \prod_{i \in [n]} a_i$).
- **Security:** Intuitively, the construction is secure, since the only way to make $g_n^{\prod_{j \in [n]} a_i}$ is to find an exact cover of $[n]$. As a matter of fact, observe that if an exact cover does not exist, then for each subset of indices $T'($ such that $\cup_{i \in T'} S_j = [n])$ we have that

$$\sum_{i=1}^{n} |S_i| > n,$$

  which means that $\prod_{i \in T} \prod_{j \in S_i} a_j$ is different than $\prod_{j \in [n]} a_j$. Formally, the security is based on the Decisional Multilinear No-Exact-Cover Assumption.

Chapter 10

Obfuscation

The problem of program obfuscation asks whether one can transform a program (e.g., circuits, Turing machines) to another semantically equivalent program (i.e., having the same input/output behavior), but is otherwise intelligible. It was originally formalized by Barak et al. who constructed a family of circuits that are non-obfuscatable under the most natural virtual black box (VBB) security.

10.1 VBB Obfuscation

As a motivation, recall that in a private-key encryption setting, we have a secret key k, encryption E_k and decryption D_k. A natural candidate for public-key encryption would be to simply release an encryption $E'_k \equiv E_k$ (i.e. E'_k semantically equivalent to E_k, but computationally bounded adversaries would have a hard time figuring out k from E'_k.

Definition 10.1 (Obfuscator of circuits under VBB) *O is an* obfuscator *of circuits if*

1. *Correctness:* $\forall c, O(c) \equiv c$.
2. *Efficiency:* $\forall c, |O(c)| \leq \text{poly}(|c|)$.
3. *VBB:* $\forall A, A$ *is PPT bounded,* $\exists$ S *(also PPT) s.t.* $\forall c$,

$$\left|\Pr\left[A\left(O(c)\right)=1\right]-\Pr\left[S^{c}(1^{|c|})=1\right]\right| \leq \text{negl}(|c|).$$

Similarly we can define it for Turing machines.

Definition 10.2 (Obfuscator of TMs under VBB) *O is an* obfuscator *of Turing machines if*

1. *Correctness:* $\forall M, O(M) \equiv M$.
2. *Efficiency:* $\exists q(\cdot) = \text{poly}(\cdot), \forall M\,(M(x)$ *halts in* t *steps* $\implies O(M)(x)$ *halts in* $q(t)$ *steps*$)$.
3. *VBB: Let* $M'(t,x)$ *be a TM that runs* $M(x)$ *for* t *steps.* $\forall A, A$ *is PPT bounded,* $\exists$ *Sim (also PPT) s.t.* $\forall c$,

$$\left|\Pr\left[A\left(O(M)\right)=1\right]-\Pr\left[S^{M'}(1^{|M'|})=1\right]\right| \leq \text{negl}(\left|M'\right|).$$

Let's show that our candidate PKE from VBB obfuscator O is semantic secure, using a simple hybrid argument.

Proof. Recall the public key $PK = O(E_k)$. Let's assume E_k is a circuit.

$$\begin{aligned}
H_0 :& A(\{(PK, E_k(m_0))\}) \\
H_1 :& S^c(\{E_k(m_0)\}) && \text{by VBB} \\
H_2 :& S^c(\{E_k(m_1)\}) && \text{by semantic security of private key encryption} \\
H_3 :& A(\{(PK, E_k(m_1))\}) && \text{by VBB}
\end{aligned}$$

■

Unfortunately VBB obfuscator for all circuits does not exist. Now we show the impossiblity result of VBB obfuscator.

Theorem 10.1 *Let O be an obfuscator. There exists PPT bounded A, and a family (ensemble) of functions $\{H_n\}$, $\{Z_n\}$ s.t. for every PPT bounded simulator S,*

$$A\left(O(H_n)\right) = 1 \quad \& \quad A\left(O(Z_n)\right) = 0$$
$$\left|\Pr\left[S^{H_n}\left(1^{|H_n|}\right) = 1\right] - \Pr\left[S^{Z_n}\left(1^{|Z_n|}\right) = 1\right]\right| \leq \text{negl}(n).$$

Proof. Let $\alpha, \beta \xleftarrow{\$} \{0,1\}^n$.

We start by constructing $A', C_{\alpha,\beta}, D_{\alpha,\beta}$ s.t.

$$A'\left(O(C_{\alpha,\beta}), O(D_{\alpha,\beta})\right) = 1 \quad \& \quad A'\left(O(Z_n), O(D_{\alpha,\beta})\right) = 0$$
$$\left|\Pr\left[S^{C_{\alpha,\beta}, D_{\alpha,\beta}}(\mathbf{1}) = 1\right] - \Pr\left[S^{Z_n, D_{\alpha,\beta}}(\mathbf{1}) = 1\right]\right| \leq \text{negl}(n).$$

$$C_{\alpha,\beta}(x) = \begin{cases} \beta, & \text{if } x = \alpha, \\ 0^n, & \text{o/w} \end{cases}$$
$$D_{\alpha,\beta}(c) = \begin{cases} 1, & \text{if } c(\alpha) = \beta, \\ 0, & \text{o/w.} \end{cases}$$

Clearly $A'(X, Y) = Y(X)$ works. Now notice that input length to D grows as the size of $O(C)$.

However for Turing machines which can have the same description length, one could combine the two in the following way:

$$F_{\alpha,\beta}(b, x) = \begin{cases} C_{\alpha,\beta}(x), & b = 0 \\ D_{\alpha,\beta}(x), & b = 1 \end{cases}.$$

Let $OF = O(F_{\alpha,\beta})$, $OF_0(x) = OF(0, x)$, similarly for OF_1, then A would be just $A(OF) = OF_1(OF_0)$.

Now assuming OWF exists, specifically we already have priavte-key encryption, we modify D as follows.

$$D_k^{\alpha,\beta}(1, i) = \text{Enc}_k(\alpha_i)$$
$$D_k^{\alpha,\beta}(2, c, d, \odot) = \text{Enc}_k(\text{Dec}_k(c) \odot \text{Dec}_k(d)), \text{where } \odot \text{ is a gate of AND, OR, NOT}$$
$$D_k^{\alpha,\beta}(3, \gamma_1, \cdots, \gamma_n) = \begin{cases} 1, & \forall i, \text{Dec}_k(\gamma_i) = \beta_i, \\ 0, & \text{o/w.} \end{cases}$$

Now the adversary A just simulate $O(C)$ gate by gate with a much smaller $O(D)$, thus we can use the combining tricks as for the Turing machines. ■

10.2 Indistinguishability Obfuscation

Definition 10.3 (Indistinguishability Obfuscator) *A uniform PPT machine $i\mathcal{O}$ is an* indistinguishability obfuscator *for a collection of circuits $\mathcal{C}_\kappa$ if the following conditions hold:*

- Correctness. *For every circuit $C \in \mathcal{C}_\kappa$ and for all inputs x, $C(x) = i\mathcal{O}(C(x))$.*
- Polynomial slowdown. *For every circuit $C \in \mathcal{C}_\kappa$, $|i\mathcal{O}(C)| \leq p(|C|)$ for some polynomial p.*
- Indistinguishability. *For all pairs of circuits $C_1, C_2 \in \mathcal{C}_\kappa$, if $|C_1| = |C_2|$ and $C_1(x) = C_2(x)$ for all inputs x, then $i\mathcal{O}(C_1) \overset{c}{\simeq} i\mathcal{O}(C_2)$. More precisely, there is a negligible function $\nu(k)$ such that for any (possibly non-uniform) PPT A,*

$$\left| \Pr[A(i\mathcal{O}(C_1)) = 1] - \Pr[A(i\mathcal{O}(C_2)) = 1] \right| \leq \nu(k).$$

Proposition 10.1 *Indistinguishability obfuscation implies witness encryption.*

Proof. Recall the witness encryption scheme, with which one could encrypt a message m to an instance x of an NP language L, such that $\mathrm{Dec}\left(x, w, \mathrm{Enc}\left(x, m\right)\right) = \begin{cases} m, \text{if} (x, w) \in L, \\ \perp, \text{o/w} \end{cases}$

Let $C_{x,m}(w)$ be a circuit that on input w, outputs m if and only if $(x, w) \in L$. Now we construct witness encryption as follows: $\mathrm{Enc}\left(x, m\right) = i\mathcal{O}\left(C_{x,m}\right), \mathrm{Dec}\left(x, w, c\right) = c(w)$.

Semantic security follows from the fact that, for $x \notin L$, $C_{x,m}$ is just a circuit that always output $\perp$, and by indistinguishability obfuscation, we could replace it with a constant circuit (padding if necessary), and then change the message, and change the circuit back. ■

Proposition 10.2 *Indistinguishability obfuscation and OWFs imply public key encryption.*

Proof. We'll use a length doubling PRG $F : \{0,1\}^n \to \{0,1\}^{2n}$, together with a witness encryption scheme (E, D). The NP language for the encryption scheme would be the image of F.

$$\begin{aligned} &\mathrm{Gen}(1^n) = (PK = F(s), SK = s), s \overset{\$}{\leftarrow} \{0,1\}^n \\ &\mathrm{Enc}\left(PK, m\right) = E(x = PK, m) \\ &\mathrm{Dec}\left(e, SK = s\right) = D(x = PK, w = s, c = e). \end{aligned}$$

■

Proposition 10.3 *Every best possible obfuscator could be equivalently achieved with an indistinguishability obfuscation (up to padding and computationally bounded).*

Proof. Consider circuit c, the *best possible obfuscated* $BPO(c)$, and c' which is just padding c to the same size of $BPO(c)$. Computationally bounded adversaries cannot distinguish between $i\mathcal{O}\left(c'\right)$ and $i\mathcal{O}\left(BPO(c)\right)$.

Note that doing iO never decreases the "entropy" of a circuit, so $i\mathcal{O}\left(BPO(c)\right)$ is at least as secure as $BPO(c)$. ■

10.3 $i\mathcal{O}$ for Polynomial-sized Circuits

Definition 10.4 (Indistinguishability Obfuscator for $\mathbf{NC}^1$) *Let $\mathcal{C}_\kappa$ be the collection of circuits of size $O(\kappa)$ and depth $O(\log \kappa)$ with respect to gates of bounded fan-in. Then a uniform PPT machine $i\mathcal{O}_{\mathbf{NC}^1}$ is an* indistinguishability obfuscator *for circuit class $\mathbf{NC}^1$ if it is an indistinguishability obfuscator for $\mathcal{C}_\kappa$.*

Given an indistinguishability obfuscator $i\mathcal{O}_{\mathbf{NC}^1}$ for circuit class $\mathbf{NC}^1$, we shall demonstrate how to achieve an indistinguishability obfuscator $i\mathcal{O}$ for all polynomial-sized circuits. The amplification relies on fully homomorphic encryption (FHE).

Definition 10.5 (Homomorphic Encryption) *A* homomorphic encryption scheme *is a tuple of PPT algorithms* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ *as follows:*

- $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is a semantically-secure public-key encryption scheme.*
- $\mathsf{Eval}(\mathsf{pk}, C, e)$ *takes public key* pk*, an arithmetic circuit C, and ciphertext $e = \mathsf{Enc}(\mathsf{pk}, x)$ of some circuit input x, and outputs* $\mathsf{Enc}(\mathsf{pk}, C(x))$.

As an example, the ElGamal encryption scheme is homomorphic over the multiplication function. Consider a cyclic group G of order q and generator g, and let $\mathsf{sk} = a$ and $\mathsf{pk} = g^a$. For ciphertexts $\mathsf{Enc}(\mathsf{pk}, m_1) = (g^{r_1}, g^{ar_1} \cdot m_1)$ and $\mathsf{Enc}(\mathsf{pk}, m_2) = (g^{r_2}, g^{ar_2} \cdot m_2)$, observe that

$$\mathsf{Enc}(\mathsf{pk}, m_1) \cdot \mathsf{Enc}(\mathsf{pk}, m_2) = (g^{r_1+r_2}, g^{a(r_1+r_2)} \cdot m_1 \cdot m_2) = \mathsf{Enc}(\mathsf{pk}, m_1 \cdot m_2)$$

Note that this scheme becomes additively homomorphic by encrypting g^m instead of m.

Definition 10.6 (Fully Homomorphic Encryption) *An encryption scheme is* fully homomorphic *if it is both compact and homomorphic for the class of all arithmetic circuits. Compactness requires that the size of the output of $\mathsf{Eval}(\cdot, \cdot, \cdot)$ is at most polynomial in the security parameter κ.*

10.3.1 Construction

Let $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ be a fully homomorphic encryption scheme. We require that Dec be realizable by a circuit in $\mathbf{NC}^1$. The obfuscation procedure accepts a security parameter κ and a circuit C whose size is at most polynomial in κ.

1. Generate $(\mathsf{pk}_1, \mathsf{sk}_1) \leftarrow \mathsf{Gen}(1^\kappa)$ and $(\mathsf{pk}_2, \mathsf{sk}_2) \leftarrow \mathsf{Gen}(1^\kappa)$.
2. Encrypt C, encoded in canonical form, as $e_1 \leftarrow \mathsf{Enc}(\mathsf{pk}_1, C)$ and $e_2 \leftarrow \mathsf{Enc}(\mathsf{pk}_2, C)$.
3. Output an obfuscation $\sigma = (i\mathcal{O}_{\mathbf{NC}^1}(P), \mathsf{pk}_1, \mathsf{pk}_2, e_1, e_2)$ of program $P_{\mathsf{pk}_1, \mathsf{pk}_2, \mathsf{sk}_1, e_1, e_2}$ as described below.

The evaluation procedure accepts the obfuscation σ and program input x.

1. Let U be a universal circuit that computes $C(x)$ given a circuit description C and input x, and denote by U_x the circuit $U(\cdot, x)$ where x is hard-wired. Let R_1 and R_2 be the circuits which compute $f_1 \leftarrow \mathsf{Eval}(U_x, e_1)$ and $f_2 \leftarrow \mathsf{Eval}(U_x, e_2)$, respectively.
2. Denote by ω_1 and ω_2 the set of all wires in R_1 and R_2, respectively. Compute $\pi_1 : \omega_1 \to \{0, 1\}$ and $\pi_2 : \omega_2 \to \{0, 1\}$, which yield the value of internal wire $w \in \omega_1, \omega_2$ when applying x as the input to R_1 and R_2.

3. Output the result of running $P_{\mathsf{pk}_1,\mathsf{pk}_2,\mathsf{sk}_1,e_1,e_2}(x, f_1, \pi_1, f_2, \pi_2)$.

Program $P_{\mathsf{pk}_1,\mathsf{pk}_2,\mathsf{sk}_1,e_1,e_2}$ has pk_1, pk_2, sk_1, e_1, and e_2 embedded.

1. Check whether $R_1(x) = f_1 \wedge R_2(x) = f_2$. π_1 and π_2 enable this check in logarithmic depth.
2. If the check succeeds, output $\mathsf{Dec}(\mathsf{sk}_1, f_1)$; otherwise output $\perp$.

The use of two key pairs and two encryptions of C, similar to CCA1-secure schemes seen previously, eliminates the virtual black-box requirement for concealing sk_1 within $i\mathcal{O}_{\mathbf{NC}^1}(P_{\mathsf{pk}_1,\mathsf{pk}_2,\mathsf{sk}_1,e_1,e_2})$.

10.3.2 Proof of Security

We prove the indistinguishability property for this construction through a hybrid argument.
Proof. Through the sequence of hybrids, we gradually transform an obfuscation of circuit C_1 into an obfuscation of circuit C_2, with each successor being indistinguishable from its antecedent.

H_0 : This corresponds to an honest execution of $i\mathcal{O}(C_1)$. Recall that $e_1 = \mathsf{Enc}(\mathsf{pk}_1, C_1)$, $e_2 = \mathsf{Enc}(\mathsf{pk}_2, C_1)$, and $\sigma = (i\mathcal{O}_{\mathbf{NC}^1}(P_{\mathsf{pk}_1,\mathsf{pk}_2,\mathsf{sk}_1,e_1,e_2}), \ldots)$.

H_1 : We instead generate $e_2 = \mathsf{Enc}(\mathsf{pk}_2, C_2)$, relying on the semantic security of the underlying fully homomorphic encryption scheme.

H_2 : We alter program $P_{\mathsf{pk}_1,\mathsf{pk}_2,\mathsf{sk}_2,e_1,e_2}$ such that it instead embeds sk_2 and outputs $\mathsf{Dec}(\mathsf{sk}_2, f_2)$. The output of the obfuscation procedure becomes $\sigma = (i\mathcal{O}_{\mathbf{NC}^1}(P_{\mathsf{pk}_1,\mathsf{pk}_2,\mathsf{sk}_2,e_1,e_2}, \ldots)$; we rely on the properties of functional equivalence and indistinguishability of $i\mathcal{O}_{\mathbf{NC}^1}$.

H_3 : We generate $e_1 = \mathsf{Enc}(\mathsf{pk}_1, C_1)$ since sk_1 is now unused, relying again on the semantic security of the fully homomorphic encryption scheme.

H_4 : We revert to the original program $P_{\mathsf{pk}_1,\mathsf{pk}_2,\mathsf{sk}_1,e_1,e_2}$ and arrive at an honest execution of $i\mathcal{O}(C_1)$.

10.4 Identity-Based Encryption

Another use of indistinguishability obfuscation is to realize identity-based encryption (IBE).

Definition 10.7 (Identity-Based Encryption) *An* identity-based encryption scheme *is a tuple of PPT algorithms* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *as follows:*

- $\mathsf{Setup}(1^\kappa)$ *generates and outputs a master public/private key pair* $(\mathsf{mpk}, \mathsf{msk})$.
- $\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id})$ *derives and outputs a secret key* $\mathsf{sk}_{\mathsf{id}}$ *for identity* id.
- $\mathsf{Enc}(\mathsf{mpk}, \mathsf{id}, m)$ *encrypts message* m *under identity* id *and outputs the ciphertext.*
- $\mathsf{Dec}(\mathsf{sk}_{\mathsf{id}}, c)$ *decrypts ciphertext* c *and outputs the corresponding message if* c *is a valid encryption under identity* id*, or* $\perp$ *otherwise.*

We combine an indistinguishability obfuscator $i\mathcal{O}$ with a digital signature scheme $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$.

- Let $\mathsf{Setup} \equiv \mathsf{Gen}$ and $\mathsf{KeyGen} \equiv \mathsf{Sign}$.
- Enc outputs $i\mathcal{O}(P_m)$, where P_m is a program that outputs (embedded) message m if input sk is a secret key for the given id, or $\perp$ otherwise.

- Dec outputs the result of $c(\mathsf{sk}_{\mathsf{id}})$.

However, this requires that we have encryption scheme where the "signatures" do not exist. We therefore investigate an alternative scheme. Let (K, P, V) be a non-interactive zero-knowledge (NIZK) proof system. Denote by $\mathsf{Com}(\cdot; r)$ the commitment algorithm of a non-interactive commitment scheme with explicit random coin r.

- Let σ be a common random string. $\mathsf{Setup}(1^\kappa)$ outputs $(\mathsf{mpk} = (\sigma, c_1, c_2), \mathsf{msk} = r_1)$, where $c_1 = \mathsf{Com}(0; r_1)$ and $c_2 = \mathsf{Com}(0^{|\mathsf{id}|}; r_2)$.
- $\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id})$ produces a proof $\pi = P(\sigma, x_{\mathsf{id}}, s)$ for the following language L: $x \in L$ if there exists s such that
$$\underbrace{c_1 = \mathsf{Com}(0; s)}_{\text{Type I witness}} \vee \underbrace{(c_2 = \mathsf{Com}(\mathsf{id}^*; s) \wedge \mathsf{id}^* \neq \mathsf{id})}_{\text{Type II witness}}$$
- Let $P_{\mathsf{id},m}$ be a program which outputs m if $V(\sigma, x_{\mathsf{id}}, \pi_{\mathsf{id}}) = 1$ or outputs $\perp$ otherwise.
  $\mathsf{Enc}(\mathsf{mpk}, \mathsf{id}, m)$ outputs $i\mathcal{O}(P_{\mathsf{id},m})$.

We briefly sketch the hybrid argument:

$\mathsf{H_0}$: This corresponds to an honest execution as described above.

$\mathsf{H_1}$: We let $c_2 = \mathsf{Com}(\mathsf{id}^*; r_2)$, relying on the hiding property of the commitment scheme.

$\mathsf{H_2}$: We switch to the Type II witness using $\pi_{\mathsf{id}_i} \forall i \in [q]$, corresponding to the queries issued by the adversary during the first phase of the selective-identity security game.

$\mathsf{H_3}$: We let $c_1 = \mathsf{Com}(1; r_1)$.

10.5 Digital Signature Scheme via Indistinguishable Obfuscation

A digital signature scheme can be constructed via indistinguishable obfuscation (iO). A digital signature scheme is made up of $(\mathsf{Setup}, \mathsf{Sign}, \mathsf{Verify})$.

$(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^k)$:
  sk = key of puncturable function and the seed of the PRF F_k
  $\mathsf{vk} = iO(P_k)$ where P_k is the program:
    $P_k(m, \sigma)$:
      for some OWF function f
        return 1 if $f(\sigma) = f(F_k(m))$
        return 0 otherwise

$\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m)$: Output $F_k(m)$.

$\mathsf{Verify}(\mathsf{vk}, m, \sigma)$: Output $P_k(m, \sigma)$.

Our security requirements will be that the adversary does wins the following game negligibly:

Challenger | Adversary

$(\mathsf{vk}, \mathsf{sk}) = \mathsf{Setup}(1^k)$ and picks random m

$\overset{\mathrm{P_k, m}}{\longrightarrow}$

$\overset{\sigma, \mathrm{m}^*}{\longleftarrow}$

Adversary wins game if $\mathsf{Verify}(\mathsf{vk}, m^*, \sigma) = 1$

To prove the security of this system, we use a hybrid argument. H_0 is as above.
H_1: Adjust vk so that $\mathsf{vk} = iO(P_{k,m,\alpha})$ where $\alpha = F_k(m)$ and $P_{k,m,\alpha}$ is the program such that:

$P_{k,m,\alpha}(m^*, \sigma)$:
  for some OWF f
    if $m = m^*$:
      if $f(\sigma) = f(\alpha)$ return 1
      otherwise return 0
    else proceed as P_k from before
      if $f(\sigma) = f(F_k(m^*))$ return 1
      otherwise return 0

Note that this program does not change its output for any value. This is indistinguishable from H_0 by indistinguishability obfuscation.

H_2: Adjust α so that it is a randomly sampled value. The indistinguishability of H_2 and H_1 follows from the security of PRG.
H_3: Adjust the program such that instead of α it relies on some β that is compared instead $f(\alpha)$ in the third line.

Any adversary that can break H_3 non-negligibly can break the OWF f with at the value β.

10.6 Public Key Encryption via Indistinguishable Obfuscation

A public key encryption scheme can be constructed via indistinguishable obfuscation. A public key encryption scheme is made up of (Gen, Enc, Dec). The PRG used below is a length doubling PRG.

$(\mathsf{pk}, \mathsf{sk}) \leftarrow Gen(1^k)$
  sk = key of puncturable function and the seed of the PRF F_k
  $\mathsf{pk} = iO(P_k)$ where P_k is the program:
    $P_k(m, r)$:
      $t = PRG(r)$
      Output $c = (t, F_k(t) \oplus m)$

$Enc(\mathsf{pk}, m)$: Sample r and output $(\mathsf{pk}(m, r))$.

$Dec(\mathsf{sk} = k, c = (c_1, c_2))$: Output $F_k(\mathsf{sk}, c_1) \oplus c_2$.

Our security requirements will be that the adversary does wins the following game negligibly:

Challenger | Adversary

$(\mathsf{pk}, \mathsf{sk}) = Gen(1^k)$ and
Randomly sample b from $\{0,1\}$ and
$c^* = Enc(\mathsf{pk}, b)$ and

$\xrightarrow{\mathrm{P_{k,c^*}}}$

$\xleftarrow{\mathrm{b^*}}$

Adversary wins game if $b = b^*$

To prove the security of this system, we use a hybrid argument. H_0 is as above.
H_1: Adjust pk so that $\mathsf{pk} = iO(P_{k,\alpha,t})$ where $\alpha = F_k(t)$ and $P_{k,\alpha,t}$ is the program such that:
$P_{k,\alpha,t}(m, r)$:
$t^* = PRG(r)$
if $t^* = t$, output $(t^*, \alpha \oplus m)$
else output $(t^*, F_k(t^*) \oplus m)$

Note that this program does not change its output for any value. This is indistinguishable from H_0 by indistinguishability obfuscation.

H_2: Adjust α so that it is a randomly sampled value.
H_3: Adjust the program such that t^* is randomly sampled and is not in the range of the PRG.

Any adversary that can win H_3 can guess a random value non-negligibly.

10.7 Indistinguishable Obfuscation Construction from NC^1 iO

A construction of indistinguishable obfuscation from iO for circuits in NC^1 is as follows:
Let $P_{k,C}(x)$ be the circuit that outputs the garbled circuit $\widetilde{UC}(C, x)$ with randomness $F_k(x)$ which is a punctured (at k) PRF in NC^1
Note that $UC(C, x)$ outputs $C(x)$ (UC is the "universal" circuit)
$iO(C) \rightarrow$ sample k randomly from $\{0,1\}^{|x|}$ and output $iO_{NC^1}(P_{k,C})$ padded to a length l

As before, we use a hybrid argument to show the security for iO.
H_0: $iO(C) = iO_{NC^1}(P_{k,C})$ as above.
$H_{final} = H_{2^n}$: $iO(\mathsf{pk}, c_2)$
$H_1 \cdots H_i$: Create a program $Q_{k,c_1,c_2,i}(x)$ and obfuscate it.
$Q_{k,c_1,c_2,i}(x)$:
Sample k randomly
if $x \geq i$, return $P_{k,c_1}(x)$
else , return $P_{k,c_2}(x)$

Note that H_i and H_{i+1} are indistinguishable for any value other than $x = i$.
$H_{i,1}$ (between H_i and H_{i+1}): Create a program $Q_{k,c_1,c_2,i,\alpha}(x)$, where $\alpha = Q_{k,c_1,c_2,i}(x)$ and obfuscate it.
$Q_{k,c_1,c_2,i,\alpha}(x)$:
Sample k randomly

if $x = i$, return α
else , return $Q_{k,c_1,c_2,i}(x)$

$H_{i,2}$: Replace α with a random β using fresh coins
$H_{i,3}$: Create the $c_2(x)$ value using fresh coins
$H_{i,4}$: Create the $c_2(x)$ value using $F_k(x)$
$H_{i,5}$: Finish the migration to $Q_{k,c_1,c_2,i+1}$

Note that at H_{final}, the circuit being obfuscated is completely changed from c_1 to c_2.