Chapter 3

Indistinguishability and Pseudo-Randomness

Recall that one main drawback of the One-time pad encryption scheme—and its simple encryption operation $Enc_k(m) = m \oplus k$—is that the key $k$ needs to be as long as the message $m$. A natural approach for making the scheme more efficient would be to start off with a short random key $k$ and then try to use some pseudo-random generator $g$ to expand it into a longer “random-looking” key $k' = g(k)$, and finally use $k'$ as the key in the One-time pad.

Can this be done? We start by noting that there can not exist pseudo-random generators $g$ that on input $k$ generate a perfectly random string $k'$, as this would contradict Shannon’s theorem (show this). However, remember that Shannon’s lower bound relied on the premise that the adversary Eve is computationally unbounded. Thus, if we restrict our attention to efficient adversaries, it might be possible to devise pseudo-random generators that output strings which are “sufficiently” random-looking for our encryption application.

To approach this problem, we must first understand what it means for a string to be “sufficiently random-looking” to a polynomial time adversary. Possible answers include:

- Roughly as many 0 as 1.
- Roughly as many 00 as 11
- Each particular bit is “roughly” unbiased.
- Each sequence of bits occurs with “roughly” the same probability.
Given any prefix, it is hard to guess the next bit.

Given any prefix, it is hard to guess the next sequence.

All of the above answers are examples of specific statistical tests—and many many more such test exist in the literature. For specific simulations, it may be enough to use strings that pass some specific statistical tests. However, for cryptography, we require the use of string that passes all (efficient) statistical tests. At first, it seems quite overwhelming to test a candidate pseudo-random generator against all efficient tests. To do so requires some more abstract concepts which we now introduce.

3.1 Computational Indistinguishability

We introduce the notion of computational indistinguishability to formalize what it means for two probability distributions to “look” the same in the eyes of a computationally bounded adversary. This notion is one of the corner stones of modern cryptography. As our treatment is asymptotic, the actual formalization of this notion considers sequences—called ensembles—of probability distributions (or growing output length).

**Definition 48.1 (Ensembles of Probability Distributions).** ...add

**Definition 48.1. (Computational Indistinguishability).** Let \(\{X_n\}_{n \in \mathbb{N}}\) and \(\{Y_n\}_{n \in \mathbb{N}}\) be ensembles of probability distributions where \(X_n, Y_n\) are probability distributions over \(\{0, 1\}^{l(n)}\) for some polynomial \(l(\cdot)\). We say that \(\{X_n\}_{n \in \mathbb{N}}\) and \(\{Y_n\}_{n \in \mathbb{N}}\) are computationally indistinguishable (abbr. \(\{X_n\}_{n \in \mathbb{N}} \approx \{Y_n\}_{n \in \mathbb{N}}\)) if for all non-uniform PPT \(D\) (called the “distinguisher”), there exists a negligible function \(\epsilon(n)\) such that for all \(n \in \mathbb{N}\)

\[
|\Pr[t \leftarrow X_n, D(t) = 1] - \Pr[t \leftarrow Y_n, D(t) = 1]| < \epsilon(n).
\]

In other words, two (ensembles of) probability distributions are computationally indistinguishable if no efficient distinguisher \(D\) can tell them apart better than with a negligible advantage.

To simplify notation, we say that \(D\) distinguishes the distributions \(X_n\) and \(Y_n\) with probability \(\epsilon\) if

\[
|\Pr[t \leftarrow X_n, D(t) = 1] - \Pr[t \leftarrow Y_n, D(t) = 1]| > \epsilon.
\]

Additionally, we say \(D\) distinguishes the probability ensembles \(\{X_n\}_{n \in \mathbb{N}}\) and \(\{Y_n\}_{n \in \mathbb{N}}\) with probability \(\mu(\cdot)\) if \(\forall n \in \mathbb{N}\), \(D\) distinguishes \(X_n\) and \(Y_n\) with probability \(\mu(n)\).
Properties of Computational Indistinguishability

We highlight some important (and natural) properties of the notion of indistinguishability. This properties will be used over and over again in the remainder of the course.

Closure under efficient operations

The first property formalizes the statement “If two distributions look the same, then they look the same no matter how you process them” (as long as the processing is efficient). More formally, if two distributions are indistinguishable, then they remain indistinguishable also if one applies a p.p.t. computable operation to them.

**Lemma 49.1.** Let \( \{X_n\}_{n \in N}, \{Y_n\}_{n \in N} \) be ensembles of probability distributions where \( X_n, Y_n \) are probability distributions over \( \{0, 1\}^{l(n)} \) for some polynomial \( l(\cdot) \), and let \( M \) be a p.p.t. machine. If \( \{X_n\}_{n \in N} \approx \{Y_n\}_{n \in N} \), then \( \{M(X_n)\}_{n \in N} \approx \{M(Y_n)\}_{n \in N} \).

**Proof.** Suppose there exists a non-uniform p.p.t. \( D \) and non-negligible function \( \mu(n) \) s.t \( D \) distinguishes \( \{M(X_n)\}_{n \in N} \) and \( \{M(Y_n)\}_{n \in N} \) with probability \( \mu(n) \). Then there exists some \( i \in [1, \cdots, m] \) s.t. \( D \) distinguishes \( X_i \) and \( X_{i+1} \) with probability \( \frac{\epsilon}{m} \).

Transitivity - The Hybrid Lemma

We next show that the notion of computational indistinguishability is transitive; namely, if \( \{A_n\}_{n \in N} \approx \{B_n\}_{n \in N} \) and \( \{B_n\}_{n \in N} \approx \{C_n\}_{n \in N} \), then \( \{A_n\}_{n \in N} \approx \{C_n\}_{n \in N} \). In fact, we prove a generalization of this statement which considers \( m = \text{poly}(n) \) distributions.

**Lemma 49.2 (The Hybrid Lemma).** Let \( X^1, X^2, \cdots, X^m \) be a sequence of probability distributions. Assume that the machine \( D \) distinguishes \( X^1 \) and \( X^m \) with probability \( \epsilon \). Then there exists some \( i \in [1, \cdots, m-1] \) s.t. \( D \) distinguishes \( X^i \) and \( X^{i+1} \) with probability \( \frac{\epsilon}{m} \).
Proof. Assume $D$ distinguishes $X_1, X^m$ with probability $\epsilon$. That is,

$$|\Pr[t \leftarrow X_1 : D(t) = 1] - \Pr[t \leftarrow X^m : D(t) = 1]| > \epsilon$$

Let $g_i = \Pr[t \leftarrow X^i : D(t) = 1]$. Thus, $|g_1 - g_m| > \epsilon$. This implies,

$$|g_1 - g_2| + |g_2 - g_3| + \cdots + |g_{m-1} - g_m|$$

$$\geq |g_1 - g_2 + g_2 - g_3 + \cdots + g_{m-1} - g_m|$$

$$= |g_1 - g_m| > \epsilon.$$

Therefore, there must exist $i$ such that $|g_i - g_{i+1}| > \frac{\epsilon}{m}$.

Remark 50.1 (A geometric interpretation). Note that the probability with which $D$ outputs 1 induces a metric space over probability distributions over strings $t$. Given this view the hybrid lemma is just a restatement of the triangle inequality over this metric spaces; in other words, if the distance between two consecutive points—representing probability distributions—is small, then the distance between the extremal points is small too.

Note that because we lose a factor of $m$ when we have a sequence of $m$ distributions, the hybrid lemma can only be used to deduce transitivity when $m$ is polynomially related to the security parameter $n$. (In fact, it is easy to construct a “long” sequence of probability distributions which are all indistinguishable, but where the extremal distributions are distinguishable.)

3.2 Pseudo-randomness

Using the notion of computational indistinguishability, we next turn to defining pseudo-random distributions.

Definition of Pseudo-random Distributions

Let $U_n$ denote the uniform distribution over $\{0, 1\}^n$, i.e, $U_n = \{t \leftarrow \{0, 1\}^n : t\}$. We say that a distribution is pseudo-random if it is indistinguishable from the uniform distribution.

Definition 50.1. (Pseudo-random Ensembles). The probability ensemble $\{X_n\}_{n \in N}$, where $X_n$ is a probability distribution over $\{0, 1\}^{l(n)}$ for some polynomial $l(\cdot)$, is said to be pseudorandom if $\{X_n\}_{n \in N} \approx \{U_{l(n)}\}_{n \in N}$. 
Note that this definition effectively says that a pseudorandom distribution needs to pass all efficiently computable statistical tests that the uniform distribution would have passed; otherwise the statistical test would distinguish the distributions.

Thus, at first sight it might seem very hard to check or prove that a distribution is pseudorandom. As it turns out, there are complete statistical tests; such a test has the property that if a distribution passes only that test, it will also pass all other efficient tests. We proceed to present such a test.

A complete statistical test: The next-bit test

We say that a distribution passes the next-bit test if no efficient adversary can, given any prefix of a sequence sampled from the distribution, predict the next bit in the sequence with probability significantly better than $\frac{1}{2}$ (recall that this was one of the tests originally suggested in the introduction of this chapter).

**Definition 51.1.** An ensemble of probability distributions $\{X_n\}_{n \in \mathbb{N}}$ where $X_n$ is a probability distribution over $\{0, 1\}^{\ell(n)}$ for some polynomial $\ell(n)$ is said to pass the Next-Bit Test if for every non-uniform p.p.t. $A$, there exists a negligible function $\epsilon(n)$ s.t. $\forall n \in \mathbb{N}$ and $\forall i \in [0, \ldots, \ell(n)]$, it holds that

$$\Pr[t \leftarrow X_n : A(1^n, t_1 t_2 \ldots t_i) = t_{i+1}] < \frac{1}{2} + \epsilon(n).$$

Here, $t_i$ denotes the $i$'th bit of $t$.

**Remark 51.1.** Note that we provide $A$ with the additional input $1^n$. This is simply allow $A$ to have size and running-time that is polynomial in $n$ and not simply in the (potentially) short prefix $t_0 \ldots t_i$.

**Theorem 51.1 (Completeness of Next-Bit Test).** If a probability ensemble $\{X_n\}_{n \in \mathbb{N}}$ passes the next-bit test then $\{X_n\}_{n \in \mathbb{N}}$ is pseudo-random.

**Proof.** Assume for the sake of contradiction that there exists a nonuniform p.p.t. distinguisher $D$, and a polynomial $p(\cdot)$ such that for infinitely many $n \in \mathbb{N}$, $D$ distinguishes $X_n$ and $U_{\ell(n)}$ with probability $\frac{1}{p(n)}$. We construct a machine $A$ that predicts the next bit of $X_n$ for every such $n$. Define a sequence of hybrid distributions as follows.

$$H^i_n = \{x \leftarrow X_n : u \leftarrow U_{\ell(n)} : x_0 x_1 \ldots x_i u_{i+1} \ldots u_{\ell(n)}\}$$

Note that $H^0_n = U_{\ell(n)}$ and $H^{\ell(n)}_n = X_n$. Thus, $D$ distinguishes between $H^0_n$ and $H^{\ell(n)}_n$ with probability $\frac{1}{p(n)}$. It follows from the hybrid lemma that there
exists some $i \in [\ell(n)]$ such that $D$ distinguishes between $H_n^i$ and $H_{n+1}^i$ with probability $\frac{1}{p(n)\ell(n)}$. That is,

$$\left| \Pr[t \leftarrow H_n^i : D(t) = 1] - \Pr[t \leftarrow H_{n+1}^i : D(t) = 1] \right| > \frac{1}{p(n)\ell(n)}$$

We assume without loss of generality that

$$\Pr[t \leftarrow H_{n+1}^i : D(t) = 1] - \Pr[t \leftarrow H_n^i : D(t) = 1] > \frac{1}{p(n)\ell(n)} \quad (3.1)$$

Note that this is without loss of generality since we could always replace $D$ with $D'() = 1 - D();$ it then must be the case that there exists infinitely many $n$ for which either $D$ or $D'$ works.

Recall, that the only difference between $H_{n+1}^i$ and $H_n^i$ is that in $H_{n+1}^i$, the $(i + 1)$'th bit is $x_{i+1}$, whereas in $H_n^i$ it is $u_{i+1}$. Thus, intuitively, $D$—given only the prefix $x_1 \ldots x_i$—can tell apart $x_{i+1}$ from a uniformly chosen bit. We show how to construct a predictor $A$ that uses such a $D$ to predict $x_{i+1}: A$ on input $(1^n, t_1 t_2 \ldots t_i)$ picks $\ell(n) - i$ random bits $u_{i+1} \ldots u_{\ell(n)} \leftarrow U^{\ell(n)-1}$, and lets $g \leftarrow D(t_1 \ldots t_i u_{i+1} \ldots u_{\ell(n)})$. If $g = 1$, it outputs $u_{i+1}$, otherwise it outputs $\bar{u}_{i+1} = 1 - u_{i+1}$. We show that

$$\Pr[t \leftarrow X_n : A(1^n, t_1 t_2 \ldots t_i) = t_{i+1}] > \frac{1}{2} + \frac{1}{p(n)\ell(n)}.$$

Towards this goal, consider the distribution $\tilde{H}(i)_n$ defined as follows:

$$\tilde{H}_n^i = \{ x \leftarrow X_n : u \leftarrow U^{\ell(n)} : x_0 x_1 \ldots x_{i-1} x_i u_{i+1} \ldots u_{\ell(n)} \}$$

Note that,

$$\Pr[t \leftarrow X_n; A(1^n, t_1 \ldots t_i) = t_{i+1}] = \frac{1}{2} \Pr[t \leftarrow H_{n+1}^i : D(t) = 1] + \frac{1}{2} \Pr[t \leftarrow H_n^i : D(t) \neq 1]$$

$$= \frac{1}{2} \Pr[t \leftarrow H_{n+1}^i : D(t) = 1] - \frac{1}{2} (1 - \Pr[t \leftarrow \tilde{H}_n^i : D(t) = 1])$$

Also note that,

$$\Pr[t \leftarrow H_n^i : D(t) = 1] = \frac{1}{2} \Pr[t \leftarrow H_n^i : D(t) = 1] + \frac{1}{2} \Pr[t \leftarrow \tilde{H}_n^i : D(t) = 1]$$
3.3. PSEUDO-RANDOM GENERATORS

By rearranging the above equation and substituting, we conclude

\[
\Pr [t \leftarrow X_n : A(1^n, t_1 \ldots t_i) = t_{i+1}]
\]

\[
= \frac{1}{2} \Pr [t \leftarrow H_n^{i+1} : D(t) = 1]
\]

\[
- \left( \Pr [t \leftarrow H_n^i : D(t) = 1] - \frac{1}{2} \Pr [t \leftarrow H_n^{i+1} : D(t) = 1] - \frac{1}{2} \right)
\]

\[
= \frac{1}{2} + \Pr [t \leftarrow H_n^{i+1} : D(t) = 1] - \Pr [t \leftarrow H_n^i : D(t) = 1]
\]

\[
> \frac{1}{2} + \frac{1}{p(n)\ell(n)}
\]

where the last inequality follows from Equation 3.1. This concludes the proof the Theorem 51.1.

\[\Box\]

### 3.3 Pseudo-random generators

We now turn to definitions and constructions of pseudo-random generators.

**Definition of a Pseudo-random Generators**

**Definition 53.1.** A function \( G : \{0, 1\}^* \rightarrow \{0, 1\}^* \) is a Pseudo-random Generator (PRG) if the following holds.

1. (efficiency): \( G \) can be computed in PPT.
2. (expansion): \( |G(x)| > |x| \)
3. (pseudo-randomness): The ensemble \( \{x \leftarrow U_n : G(x)\}_{n \in \mathbb{N}} \) is pseudo-random.

**Constructions of Pseudorandom generators.**

**First attempt** The first attempt to provide a general construction of pseudo-random generators was by Adi Shamir. The construction is as follows:

**Definition 53.2 (PRG-Shamir).** Start with a one-way permutation \( f \), then define \( G_{Shamir} \) as follows:

\[
G(s) = f^n(s) \parallel f^{n-1}(s) \parallel \ldots \parallel f(s) \parallel s
\]

(The \( \parallel \) symbol stands for string concatenation.) The basic idea here to iterate a one-way function, then output, in reverse order, all the intermediary valuers. The insight behind the security of the scheme is that given some
prefix of the random sequence, computing the next block is equivalent to inverting the one-way function \( f \).

This scheme leads to unpredictable numbers, but not necessarily pseudo-random \textit{bits}; since some of the output bits of a one-way function may be predictable.

The reason we need \( f \) to be a permutation and not a general one-way function is two-fold. First, we need the domain and range to be the same number of bits. Second, and more importantly, we require that the output of \( f^k(x) \) be uniformly distributed if \( x \) is. If \( f \) is a permutation, this is true, but it need not hold for a general one-way function.

There is an extension of this technique, though, that does what we want.

\textbf{Hard-core bits}

\textbf{Definition 54.1} (Hard-core predicate). A predicate \( h : \{0,1\}^* \rightarrow \{0,1\} \) is a hard-core predicate for \( f(x) \) if \( h \) is efficiently computable given \( x \), and \( \forall \) nonuniform p.p.t. adversaries \( A \), there exists a negligible \( \epsilon \) so that \( \forall k \in \mathbb{N} \)

\[
\Pr[x \rightarrow \{0,1\}^k : A(1^n, f(x)) = h(x)] \leq \frac{1}{2} + \epsilon(n)
\]

In other words, a hard-core predicate for a function can not be computed given the result of the function, but can be computed given the function's input.

The least significant bit of the RSA one-way function is known to be hardcore (under the RSA assumption). Formally, given \( n, e, \) and \( f_{\text{RSA}}(x) = x^e \mod n \), there is no efficient algorithm to predict \( \text{LSB}(x) \) from \( f_{\text{RSA}}(x) \).

Some other examples:

- The function \( \text{half}_n(x) \) which is equal to 1 iff \( 0 \leq x \leq \frac{n}{2} \) is also hardcore for RSA, under the RSA assumption.
- For the exponentiation function \( \mod p \), under the DL assumption, \( \text{half}_{p-1} \) is a hardcore predicate.

\textbf{Constructing the Generator}

Now, how do we construct a PRNG from a one-way permutation? [Blum and Micali]

\textbf{Proposition 54.1}. Let \( f \) be a one-way permutation, and \( h \) a hard-core predicate for \( f \). Then \( G(s) = f(s) \| h(s) \) is a PRNG.
3.3. PSEUDO-RANDOM GENERATORS

Proof. Assume for contradiction that there exists a nonuniform p.p.t. adversary $A$ and polynomial $p(n)$ so that for infinitely many $n$, there exists an $i$ so that $A$ predicts the $i^{th}$ bit with probability $\frac{1}{p(n)}$. The first $n$ bits of $G(s)$ are a permutation of a uniform distribution, and are therefore also distributed uniformly at random. So $A$ must predict bit $n+1$ with advantage $\frac{1}{p(n)}$. Formally,

$$Pr[A(f(s)) = h(s)] > \frac{1}{2} + \frac{1}{p(n)}$$

This contradicts the assumption that $b$ is hard-core for $f$. Therefore, $G$ is a PRNG.

This construction only extends an $n$-bit seed to $n + 1$ bits. This, however is sufficient as we can cleverly iterate this procedure in order to generate a much longer sequence.

Theorem 55.1. If there exists a PRG $f : \{0, 1\}^n \rightarrow \{0, 1\}^{n+1}$ which expands its seed by 1 bit, then there exists another PRG $f'$ which expands its seed to $\ell(n)$ bits for any polynomial $\ell$.

Lemma 55.1. Let $G : \{0, 1\}^n \rightarrow \{0, 1\}^{n+1}$ be a PRG. For some polynomial $\ell$, define $G' : \{0, 1\}^n \rightarrow \{0, 1\}^{\ell(n)}$ as follows:

$$G'(s) = b_1 \ldots b_{\ell(n)}$$

where

$$X_0 \leftarrow s$$

$$X_{i+1} \parallel b_{i+1} \leftarrow G(X_i)$$

Then $G'$ is a PRG.

Proof sketch. A hybrid argument.

Corollary 55.1. Let $f$ be a OWP and $h$ a hard core bit for $f$. Then

$$g(x) = h(x) \parallel h(f(x)) \parallel h(f^{(2)}(x)) \parallel \ldots \parallel h(f^{(n)}(x))$$

is a PRG.

Proof. Let $g'(x) = f(x) \parallel h(x)$. The function $g'$ is a PRG. The result follows by the Lemma above.

Note: The PRG construction above can be computed “on-line.” If we remember $x_i$ we can compute the continuation of the function. The construction also works for collections of OWP:

$$G(r_1, r_2) = h_i(f_i(x)) \parallel h_i(f_i^{(2)}(x)) \parallel \ldots$$

where $r_1$ is used to sample $i$ and $r_2$ is used to sample $x$. 
Concrete examples of PRGs

- **Modular Exponentiation** (Blum-Micali PRG)
  - Use seed to generate $p, g, x$ where $p$ is a prime $> 2$, $g$ is a generator for $\mathbb{Z}_p^*$, and $x \in \mathbb{Z}_p^*$.
  - Output $\text{half}_{p-1}(x) \parallel \text{half}_{p-1}(g^x \mod p) \parallel \text{half}_{p-1}(g^{g^x} \mod p) \parallel \ldots$

- **RSA** (Blum-Micali PRG)
  - Use seed to generate $p, q, e$ where $p, q$ are random $n$-bit primes $p, q$, and $e$ is a random element in $\mathbb{Z}_N^*$ where $N = pq$.
  - Output $\text{LSB}(x) \parallel \text{LSB}(x^e \mod N) \parallel \text{LSB}((x^e)^e \mod N) \parallel \ldots$
    where $\text{LSB}(x)$ is the least significant bit of $x$.

In all the above PRG, we can in fact output $\log k$ bits at each iteration, while still remaining provably secure. Moreover, it is conjectured that it is possible to output $\frac{k}{2}$ bits at each iteration and still remain secure.

### 3.4 *Hard-Core Bits from Any OWF

In the previous section, we showed that if $f$ is a one way permutation and $h$ is a hard-core bit for $f$, then the function

$$G(s) = f(s) \parallel h(s)$$

if a pseudo-random generator. One issue, however, is how to construct a hard-core bit for a given one-way permutation. We have illustrated examples for some of the well-known permutations. Here we will show that every one-way function (and therefore one-way permutation) can be transformed into another one-way function which has a hard-core bit.

**Theorem**

Let $f$ be a OWF. Then $f'(X, r) = f(X), r$ (where $|X| = |r|$) is a OWF and $b(X, r) = \langle X, r \rangle_2 = \Sigma X_i r_i \mod 2$ (inner product mod 2) is a hardcore predicate for $f$.

Here $r$ essentially tells us which bits of $x$ on which to take parity. Note that $f'$ is a OWP if $f$ is a OWP.
Proof. (by reductio ad absurdum) We show that if \( A \), given \( f'(X, r) \) can compute \( b(X, r) \) with probability significantly better than 1/2, then there exists a p.p.t. adversary \( B \) that inverts \( f \).

Our proof takes three steps. In the first step, we prove the theorem for an oversimplified case. In the next step, we take a less simplified case and finally we prove the theorem the general case.

In the oversimplified case, we assume \( A \) always computes \( b \) correctly. And so, we can construct a \( f' \) with an \( r \) such that the first bit is 1 and the other bits are 0. In such a case \( A \) would return the first bit of \( X \). Similarly we can set the second bits of \( r \) to be 1 to obtain the second bit of \( X \). Thus, we have \( B \) given by

\[
B(y): \text{Let } X_i = A(y, e_i) \text{ where } e_i = 000 \ldots 1 \ldots 000 \text{ where the 1 is on the } i^{\text{th}} \text{ position.}
\]

-Output \( X_1, X_2, \ldots, X_n \)

This works, since \( \langle X, e_i \rangle_2 = X_i \)

Now, in the less simplified case, we assume that \( A \), when given random \( y = f(X) \) and random \( r \), computes \( b(X, r) \) w.p. \( \frac{3}{4} + \epsilon \), (\( \epsilon = \frac{1}{\text{poly}(n)} \), \( n \) is the length of \( X \)).

Intuition: we want the attacker to compute \( b \) with a fixed \( X \) and a varying \( r \) so that given enough observations, \( X \) can be computed eventually. The trick is to find the set of good \( X \), for which this will work.

As an attempt to find such \( X \), let \( S = \{X | \text{Pr} [A(f(X), r) = b(X, r)] > \frac{3}{4} + \frac{\epsilon}{2} \} \). It can be shown that \( |S| > \epsilon/2 \).

A simple attack with various \( e_i \) might not work here. More rerandomization is required. Idea: Use linearity of \( \langle a, b \rangle \).

Useful relevant fact: \( \langle a, b \oplus c \rangle = \langle a, b \rangle \oplus \langle a, c \rangle \mod 2 \)

Proof.

\[
\langle a, b \oplus c \rangle = \Sigma a_i (b_i + c_i) \\
= \Sigma a_i b_i + \Sigma a_i c_i \\
= \langle a, b \rangle + \langle a, c \rangle \mod 2
\]

Attacker asks: \( \langle X, r \rangle, \langle X, r + e_1 \rangle \)

and then XOR both to get \( \langle X, e_1 \rangle \) without ever asking for \( e_1 \).

And so, \( B \) inverts \( f \) as follows: \( B(y) : \)

For \( i = 1 \) to \( n \)

1. Pick random \( r \) in \( \{0, 1\}^n \)
2. Let \( r' = e_i \oplus r \)

3. Compute guess for \( X_i \) as \( A(y, r) \oplus A(y, r') \)

4. Repeat \( \text{poly}(1/\epsilon) \) times and let \( X_i \) be majority of guesses.

Finally output \( X_1, \ldots, X_n \).

If we assume \( e_1 \) and \( r + e_1 \) as independent, the proof works fine. However, they are not independent. The proof is still OK though, as can be seen using the union bound:

The proof works because:

- w.p. \( \frac{1}{4} - \frac{\epsilon}{2} \), \( A(y, r) \neq b(X, r) \)
- w.p. \( \frac{1}{4} - \frac{\epsilon}{2} \), \( A(y, r') \neq b(X, r) \)
- by union bound w.p. \( \frac{1}{2} \) both answers of \( A \) are OK.
- Since \( \langle y, r \rangle + \langle y, r' \rangle = \langle y, r \oplus r' \rangle = \langle y, e_i \rangle \), each guess is correct w.p. \( \frac{1}{2} + \epsilon \)
- Since samples are independent, using Chernoff Bound it can be shown that every bit is OK w.h.p.

Now, to the general case. Here, we assume that \( A \), given random \( y = f(X) \), random \( r \) computes \( b(X, r) \) w.p. \( \frac{1}{2} + \epsilon \) (\( \epsilon = \frac{1}{\text{poly}(n)} \)).

Let \( S = \{ X | \Pr [ A(f(X), r) = b(X, r) ] > \frac{1}{2} + \frac{\epsilon}{2} \} \). It again follows that \( |S| > \frac{\epsilon}{2} \).

Assume set access to oracle \( C \) that given \( f(X) \) gives us samples

\[
\langle X, r_1 \rangle, r_1 \\
\vdots \\
\langle X, r_n \rangle, r_n
\]

(where \( r_1, \ldots, r_n \) are independent and random)

We now recall Homework 1, where given an algorithm that computes a correct bit value w.p. greater than \( \frac{1}{2} + \epsilon \), we can run it multiple times and take the majority result, thereby computing the bit w.p. as close to 1 as desired.

From here on, the idea is to eliminate \( C \) from the constructed machine step by step, so that we don’t need an oracle in the final machine \( B \).

Consider the following \( B(y) \):

For \( i = 1 \) to \( n \)
1. \( C(y) \rightarrow (b_1, r_1), \ldots, (b_m, r_m) \)

2. Let \( r'_j = e_i \oplus r_j \)

3. Compute \( g_j = b_j \oplus A(y, r') \)

4. Let \( X_i = majority(g_1, \ldots, g_m) \)

Output \( X_1, \ldots, X_m \)

Each guess \( g_i \) is correct w.p. \( \frac{1}{2} + \frac{\epsilon}{2} = \frac{1}{2} + \epsilon' \). As in HW1, by Chernoff bound, an \( x_i \) is wrong w.p. \( \leq 2^{-\epsilon'^2 m} \) (was \( 2^{-4\epsilon^2 m} \) in the HW). If \( m >> \frac{1}{\epsilon'^2} \), we are OK.

Now, we assume that \( C \) gives us samples \( \langle X, r_1 \rangle, r_1 ; \ldots ; \langle X, r_n \rangle, r_n \) which are random but only pairwise independent. Again, using results from HW1, by Chebyshev’s theorem, each \( X_i \) is wrong w.p. \( \leq \frac{1-4\epsilon'^2}{4m\epsilon'^2} \leq \frac{1}{m\epsilon'} \) (ignoring constants).

By union bound, any of the \( X_i \) is wrong w.p. \( \leq \frac{n}{m\epsilon'} \leq \frac{1}{2} \), when \( m \geq \frac{2n}{\epsilon'} \). Therefore, as long as we have polynomially many samples (precisely \( \frac{2n}{\epsilon'} \) pairwise independent samples), we’d be done.

The question now is: How do we get pairwise independent samples? So, our initial attempt to remove \( C \) would be to pick \( r_1, \ldots, r_m \) on random and guess \( b_1, \ldots, b_m \) randomly. However, \( b_i \) would be correct only w.p. \( 2^{-m} \).

A better attempt is to pick \( \log(m) \) samples \( s_1, \ldots, s_{\log(m)} \) and guessing \( b'_1, \ldots, b'_{\log(m)} \) randomly. Here the guess is correct with probability \( 1/m \).

Now, generate \( r_1, r_2, \ldots, r_{m-1} \) as all possible sums (mod 2) of subsets of \( s_1, \ldots, s_{\log(m)} \), and \( b_1, b_2, \ldots, b_m \) as the corresponding subsets of \( b'_i \). Mathematically

\[
\begin{align*}
  r_i &= \sum_{j \in I_i} s_j \text{ if } j \in I \text{ iff } i_j = 1 \\
  b_i &= \sum_{j \in I_i} b'_j
\end{align*}
\]

In HW1, we showed that these \( r_i \) are pairwise independent samples. Yet w.p. \( 1/m \), all guesses for \( b'_1, \ldots, b'_{\log(m)} \) are correct, which means that \( b_1, \ldots, b_{m-1} \) are also correct.

Thus, for a fraction of \( \epsilon' \) of \( X' \) it holds that w.p. \( 1/m \) we invert w.p. \( 1/2 \). That is \( B(y) \) inverts w.p.

\[
\frac{\epsilon'}{2m} = \frac{\epsilon'^3}{4n} = \frac{(\epsilon/2)^3}{4n} \quad (m = \frac{2n}{\epsilon'^2})
\]
which contradicts the (strong) one-way-ness of $f$.

Yao proved that if OWF exists, then there exists OWF with hard core bits. But this construction is due to Goldreich and Levin[?] and by Charles Rackoff[?].

### 3.5 Pseudo-random Functions (PRFs)

Just like strings, functions can be sampled from the set of all functions. Thus, a natural question is whether one can construct a special family of functions which cannot be distinguished from the set of all functions. In this section, we do exactly that. In particular, we show how pseudo-random generators can be used to construct pseudo-random functions. Let us first define a random function.

**Definition 60.1.** A function $F : \{0, 1\}^n \rightarrow \{0, 1\}^n$ is a random function if it is constructed as follows: For each $x \in \{0, 1\}^n$ pick a random $y \in \{0, 1\}^n$, and let $F(x) = y$.

The description length of a random function is $n2^n$, and there are a total of $2^{n2^n}$ different random functions.

A random function can be simulated in by lazily generating the random values for each input the first time that input is queried.

Let $RF_n$ denote the probability distribution resulting from picking a random function $F : \{0, 1\}^n \rightarrow \{0, 1\}^n$.

**Definition 60.2.**

$$
F = \left\{ f_s : \{0, 1\}^{|s|} \rightarrow \{0, 1\}^n \mid s \in \{0, 1\}^* \right\}
$$

is a family of PRFs if:

- [EASY TO COMPUTE] given $s \in \{0, 1\}^n$ and $x \in \{0, 1\}^n$, can efficiently compute $f_s(x)$.

- [PSEUDO-RANDOM] for all non-uniform PPT “oracle machines” $D$, there exists a negligible function $\epsilon(k)$ such that

$$
\left| \Pr[s \leftarrow \{0, 1\}^k : D^{f_s}(1^n) = 1] - \Pr[F \leftarrow RF_k : D^F(1^n) = 1] \right| \leq \epsilon(k)
$$

Note that the number of PRFs $\{f_s \mid s \in \{0, 1\}^k\}$ is much smaller than the number of random functions of the same length.
3.5. PSEUDO-RANDOM FUNCTIONS (PRFS)

Construction of a PRF

Let

\[ G(x) = G_0(x) \parallel G_1(x) \]

be a length doubling PRG such that \( |G_i(x)| = |x| \). Let

\[ f_s(b_1b_2 \ldots b_n) = G_{b_n}(G_{b_{n-1}}(\ldots G_{b_2}(G_{b_1}(s)) \ldots )) \]

(That is, in an imperative form, \( y := s; \) for \( i \in 1..n, y := G_{b_i}(y); \) output \( y \).)

**Theorem 61.1.** The set \( \{f_s \mid s \in \{0, 1\}^n\} \) is a family of PRFs.

Proof sketch. Use a hybrid argument: assume for contradiction that there exists a PPT \( D \) and polynomial \( p(n) \) such that \( D \) distinguishes \( f_s \) from a random function with probability better than \( p(n) \).

Construct \( HF^n_i \) by picking the first \( i \) “layers” at random and then apply the construction. That is \( HF^n_i \) is a family of functions of the form \( f(b_1 \ldots b_n) = G_{b_n}(G_{b_{n-1}}(\ldots G_{b_1}(r)) \ldots )) \), where \( r \) is drawn from \( U_n \), the uniform random distribution over \( \{0, 1\}^n \).

Note that:

\[ HF^n_1 = \{f_s \mid s \leftarrow \{0, 1\}^n\} \]
\[ HF^n_n = RF_n \]

Using the same proof as the hybrid lemma, there must exist an \( i \) such that \( D \) distinguishes between \( HF^n_i \) and \( HF^n_{i+1} \) with probability \( \frac{1}{p(n)} \) for infinitely many \( n \).

Note that \( HF^n_i \) can be efficiently sampled: first time an \( i \)-prefix is asked, pick a random string, and next time output the same.

Also, the only difference between \( HF^n_i \) and \( HF^n_{i+1} \) is that in \( HF^n_i \) the \( i+1 \) layer is computed as \( G(U_n) \), whereas in \( HF^n_{i+1} \) it is \( U_{2n} \).

Thus, \( D \) is able to distinguish polynomial many samples of \( G(U_n) \) and \( U_{2n} \).

Using an additional hybrid argument, we can contradict the assumption that \( G \) is a PRG, by defining hybrid \( H_i \) to take the first \( i \) bits from \( G(U_n) \), and the rest of the bits from \( U_{2n} \).

Note: A question was asked, can the roles of \( s \) and \( b \) be swapped in the construction of the PRF, that is would

\[ f_s(b) = G_{s_n}(G_{s_{n-1}}(\ldots G_{s_2}(G_{s_1}(b)) \ldots )) \]

also result in a family of PRFs? No, it would not, since the PRGs \( G_0 \) and \( G_1 \) require the input to be a random value. The bit string \( s \) is a random value,
but the bit string \( b = b_1 \ldots b_n \) is picked by the adversary, and thus may not be random.

At this point in the course, we have shown how to define secrecy and how to construct tools such as one-way functions, pseudo-random generators, and pseudo-random functions. We will now use these concepts to build secure encryption schemes whose keys are smaller than the messages one can encrypt.

### 3.6 Secure Encryption Scheme

**Definition 62.1 (Secure Encryption - Indistinguishability).** Let \((\text{Gen}, \text{Enc}, \text{Dec})\) be an encryption scheme over message space \( \mathcal{M} \) and key space \( \mathcal{K} \). The encryption scheme \((\text{Gen}, \text{Enc}, \text{Dec})\) is said to be **single-message secure** if \( \forall \) non-uniform p.p.t \( A \), there exists a negligible function \( \epsilon(n) \) such that \( \forall m_0, m_1 \in \mathcal{M}, |m_0| = |m_1| \) it holds that

\[
\left| \Pr[k \leftarrow \text{Gen}(1^n) : A(\text{Enc}_k(m_0)) = 1] - \Pr[k \leftarrow \text{Gen}(1^n) : A(\text{Enc}_k(m_1)) = 1] \right| \leq \epsilon(n)
\]

The above definition is based on the indistinguishability of the distribution of ciphertexts created by encrypting two different messages. The above definition does not, however, explicitly capture any *a priori* information that an adversary might have. Later in the course, we will see a definition which explicitly captures any *a priori* information that the adversary might have and in fact show that the indistinguishability definition is equivalent to it.

### 3.7 An Encryption Scheme with Short Keys

Recall that we constructed an encryption scheme using one time pads. However, we proved that to be able to prove perfect secrecy, the one time pad needed to be as large as the message. Can we construction encryption schemes using smaller keys which are secure under the new definitions we saw today? The idea is to use pseudorandomness instead of pure randomness. Since we know how to take a small seed and construct a long pseudorandom sequence, we can perform encryption with smaller keys.

More precisely, consider the following encryption scheme. Let \( G(s) \) be a length-doubling pseudo-random generator.

**Theorem 62.1.** \((\text{Gen}, \text{Enc}, \text{Dec})\) is single-message secure.

**Proof.** Consider any two messages \( m_0 \) and \( m_1 \), and a distinguisher \( D \). Consider the following distributions:
3.8 MANY-MESSAGE SECURITY

Algorithm 7: Encryption Scheme for \( n \)-bit messages

\[
\begin{align*}
\text{Gen}(1^n) & : k \leftarrow \mathbb{U}_{n/2} \\
\text{Enc}_k(m) & : \text{Output } m \oplus G(k) \\
\text{Dec}_k(c) & : \text{Output } c \oplus G(k)
\end{align*}
\]

- \( H_1 \) (real with \( m_0 \)) : \{s \leftarrow \text{Gen}(1^n) : m_0 \oplus G(s)\}.
- \( H_2 \) (OTP with \( m_0 \)) : \{r \leftarrow \mathbb{U}_n : m_0 \oplus r\}.
- \( H_3 \) (OTP with \( m_1 \)) : \{r \leftarrow \mathbb{U}_n : m_1 \oplus r\}.
- \( H_4 \) (real with \( m_1 \)) : \{s \leftarrow \text{Gen}(1^n) : m_1 \oplus G(s)\}.

By the perfect secrecy of one-time pad, there is no distinguisher \( D \) that can tell apart \( H_2 \) and \( H_3 \). Since \( G(s) \) is pseudorandom, for every non-uniform p.p.t. distinguisher \( D \), there is a negligible function \( \epsilon \) such that for every \( n \),

\[
\left| \Pr[s \leftarrow \mathbb{U}_{n/2} : D(G(s)) = 1] - \Pr[r \leftarrow \mathbb{U}_n : D(r) = 1] \right| \leq \epsilon(n)
\]

Therefore, since \( f(x) = m \oplus x \) is a p.p.t. computable function, for any \( m \), it holds that

\[
\left| \Pr[s \leftarrow \mathbb{U}_{n/2} : D(m \oplus G(s)) = 1] - \Pr[r \leftarrow \mathbb{U}_n : D(m \oplus r) = 1] \right| \leq \epsilon(n)
\]

Hence, there is no distinguisher which can tell apart \( H_1 \) and \( H_2 \) by more than a negligible function. Also, there is no distinguisher which can tell apart \( H_3 \) and \( H_4 \) by more than a negligible function. Therefore, by the hybrid lemma, there is no distinguisher which can tell apart \( H_1 \) from \( H_4 \) by more than a negligible function.

What we just showed is that for all non-uniform p.p.t distinguisher \( D \), there is a negligible function \( \epsilon \), such that for all messages \( m_1 \in \mathcal{M}, |m_1| = n \) and \( m_2 \in \mathcal{M}, |m_2| = n \),

\[
|\Pr[k \leftarrow \text{Gen}(1^n) : D(\text{Enc}_k(m_1)) = 1] - \Pr[k \leftarrow \text{Gen}(1^n) : D(\text{Enc}_k(m_2)) = 1]| \leq \epsilon(n)
\]

Hence, (Gen, Enc, Dec) is single-message secure. \( \square \)

3.8 Many-message security

Unfortunately, the scheme described in the previous section is not secure if an eavesdropper receives encryptions of many messages. (Show this. In particular, consider the xor of two ciphertexts.) Thus, the previous definitions of
security are insufficient; although they guarantee security for one message, they guarantee nothing when two messages are sent. In this section, we will adapt the definitions to quantify over many messages.

**Definition 64.1 (Secure Encryption, many message indistinguishability).** Let \((\text{Gen}, \text{Enc}, \text{Dec})\) be an encryption scheme over message space \(\mathcal{M}\) and key space \(\mathcal{K}\). Then \((\text{Gen}, \text{Enc}, \text{Dec})\) is said to be many-message secure if \(\forall\) non-uniform p.p.t. \(A\), \(\forall\) polynomial \(q(n)\), \(\exists\) a negligible function \(\epsilon(n)\) such that

\[
\forall m_0, m_1, \ldots, m_q(n), m'_0, m'_1, \ldots, m'_q(n)\text{ for which }|m_0| = \cdots = |m_q(n)| = |m'_0| = \cdots = |m'_q(n)|\text{ it holds that }
\]

\[
\left| \Pr[k \leftarrow \text{Gen}(1^n): A(\text{Enc}_k(m_0), \text{Enc}_k(m_1) \ldots \text{Enc}_k(m_q(n))) = 1] - \Pr[k \leftarrow \text{Gen}(1^n): A(\text{Enc}_k(m'_0), \text{Enc}_k(m'_1) \ldots \text{Enc}_k(m'_q(n))) = 1] \right| \leq \epsilon(n)
\]

The insight is to use a pseudo-random function in order to pick a separate random pad for every message. In order to make decryption possible, the ciphertext contains the input on which the pseudo-random function is evaluated.

**Algorithm 8: Many-message Encryption Scheme**

Assume \(m \in \{0, 1\}^n\) and let \(\{f_k\}\) be a PRF family

- \(\text{Gen}(1^n) : k \leftarrow U_n\)
- \(\text{Enc}_k(m) : \text{Pick } r \leftarrow U_n. \text{Output } (r, m \oplus f_k(r))\)
- \(\text{Dec}_k((r, c)) : \text{Output } c \oplus f_k(r)\)

**Theorem 64.1.** \((\text{Gen}, \text{Enc}, \text{Dec})\) is a many-message secure encryption scheme.

**Proof.** Consider some distinguisher \(D\) and two sequences of messages: \(m_0, m_1, \ldots, m_q(n)\) and \(m'_0, m'_1, \ldots, m'_q(n)\) respectively. We introduce the following sequence of distributions and argue that any two adjacent distributions are computationally indistinguishable.

- \(H_1\) – real execution with \(m_0, m_1, \ldots, m_q(n)\)

\[
\{s \leftarrow \{0, 1\}^n, r_0, \ldots, r_q(n) \leftarrow \{0, 1\}^n : r_0||m_0 \oplus f_s(r_0), \ldots, m_q(n) \oplus f_s(r_q(n))\}
\]

This is precisely what the adversary sees when receiving the encryptions of \(m_0, \ldots, m_q(n)\).
3.9. STRONGER ATTACK MODELS

- $H_2$ – using a truly random function instead of $f$ on $m_0, m_1, \ldots, m_{q(n)}$
  \[ \{ R \leftarrow RF_n; r_0, \ldots, r_{q(n)} \leftarrow \{0,1\}^n : r_0 \oplus R(r_0), \ldots, m_{q(n)} \oplus R(r_{q(n)}) \} \]

- $H_3$ – using one-time pad encryption on $m_0, m_1, \ldots, m_{q(n)}$
  \[ \{ p_0 \ldots p_{q(n)} \leftarrow \{0,1\}^n ; r_0, \ldots, r_{q(n)} \leftarrow \{0,1\}^n : r_0 \oplus p_0, \ldots, m_{q(n)} \oplus p_{q(n)} \} \]

- $H_4$ – using one-time pad encryption on $m'_0, m'_1, \ldots, m'_{q(n)}$
  \[ \{ p_0 \ldots p_{q(n)} \leftarrow \{0,1\}^n ; r_0, \ldots, r_{q(n)} \leftarrow \{0,1\}^n : r_0 \oplus p_0, \ldots, m'_{q(n)} \oplus p_{q(n)} \} \]

- $H_5$ – using a truly random function instead of $f$ on $m'_0, m'_1, \ldots, m'_{q(n)}$
  \[ \{ R \leftarrow \{ \{0,1\}^n \rightarrow \{0,1\}^n \}; r_0, \ldots, r_{q(n)} \leftarrow \{0,1\}^n : r_0 \oplus m'_0 \oplus R(r_0), \ldots, m'_{q(n)} \oplus R(r_{q(n)}) \} \]

- $H_6$ – real execution with $m'_0, m'_1, \ldots, m'_{q(n)}$
  \[ \{ s \leftarrow \{0,1\}^n ; r_0, \ldots, r_{q(n)} \leftarrow \{0,1\}^n : r_0 \oplus m'_0 \oplus f_s(r_0), \ldots, m'_{q(n)} \oplus f_s(r_{q(n)}) \} \]

$D$ can distinguish $H_1$ and $H_2$ with at most negligible probability, otherwise it contradicts the pseudo-randomness properties of $\{f_s\}$. The same argument applies for $H_6$ and $H_5$.

$H_2$ and $H_3$ are “almost” identical except for the case when $\exists i, j$ such that $r_i = r_j$, but this happens with negligible probability, therefore $D$ can distinguish $H_2$ and $H_3$ only with negligible probability. The same argument applies for $H_4$ and $H_5$.

$H_3$ and $H_4$ are identical because of the properties of the one time pad.

By the polyjump lemma, $D$ can distinguish between $H_1$ and $H_6$ with at most negligible probability.

In what follows we shall explore stronger attack scenarios. This far we had Eve listening to the communication channel between Alice and Bob – suppose Eve has greater powers.

3.9 Stronger Attack Models

So far, we have assumed that the adversary only captures the ciphertext that Alice sends to Bob. In other words, the adversaries attack is a ciphertext only attack. One can imagine, however, a variety of stronger attack models. We list some of these models below:

**Attack models:**
• Ciphertext only attack – this is what we considered so far.
• Known plaintext attack – The adversary may get to see pairs of form \((m_0, \text{Enc}_k(m_0))\) . . .
• Chosen plain text (CPA) – The adversary gets access to an encryption oracle before and after selecting messages.
• Chosen ciphertext attack
  
  **CCA1:** (“Lunch-time attack”) The adversary has access to an encryption oracle and to a decryption oracle before selecting the messages. (due to Naor and Yung)

  **CCA2:** This is just like a CCA1 attack except that the adversary also has access to decryption oracle after selecting the messages. It is not allowed to decrypt the challenge ciphertext however. (introduced by Rackoff and Simon)

Fortunately, all of these attacks can be abstracted and captured by a simple definition which we present below. The different attacks can be captured by allowing the adversary to have oracle-access to a special function which allows it to mount CPA/CCA1/CCA2-type attacks.

**Definition 66.1** (Secure encryption CPA / CCA1 / CCA2). Let \(\Pi = (\text{Gen}, \text{Enc}, \text{Dec})\) be an encryption scheme. Let the random variable \(\text{IND}_{O_1, O_2}^{b}(\Pi, A, n)\) where \(A\) is a non-uniform p.p.t., \(n \in \mathbb{N}\), \(b \in \{0, 1\}\) denote the output of the following experiment:

\[
\text{IND}_{b}^{O_1, O_2}(\Pi, ma, n)
\]

\[
k \leftarrow \text{Gen}(1^n)
\]

\[
m_0, m_1, \text{state} \leftarrow A^{O_1(k)}(1^n)
\]

\[
c \leftarrow \text{Enc}_k(m_b)
\]

Output \(A^{O_2(k)}(c, \text{state})\)

Then we say \(\pi\) is CPA/CCA1/CCA2 secure if \(\forall\) non-uniform p.p.t. PPT \(A\):

\[
\left\{ \text{IND}^{O_1, O_2}_0(\pi, A, n) \right\}_{n \in \mathbb{N}} \approx \left\{ \text{IND}^{O_1, O_2}_1(\pi, A, n) \right\}_{n \in \mathbb{N}}
\]

where \(O_1\) and \(O_2\) are defined as:

<table>
<thead>
<tr>
<th>Type</th>
<th>Oracle Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPA</td>
<td>([\text{Enc}_k; \text{Enc}_k])</td>
</tr>
<tr>
<td>CCA1</td>
<td>([\text{Enc}_k, \text{Dec}_k; \text{Enc}_k])</td>
</tr>
<tr>
<td>CCA2</td>
<td>([\text{Enc}_k, \text{Dec}_k; \text{Enc}_k, \text{Dec}_k])</td>
</tr>
</tbody>
</table>
Additionally, in the case of CCA2 attacks, the decryption oracle returns ⊥ when queried on the challenge ciphertext $c$.

### 3.10 CPA/CCA1 Secure Encryption Scheme

We will now show that the encryption scheme presented in construction 8 satisfies a stronger property than claimed earlier. In particular, we show that it is CCA1 secure (which implies that it is also CPA-secure).

**Theorem 67.1.** $\pi$ in construction 8 is CPA and CCA1 secure.

**Proof.** Consider the encryption scheme $\pi^{RF} = (\text{Gen}^{RF}, \text{Enc}^{RF}, \text{Dec}^{RF})$, which is derived from $\pi$ by replacing PRF $f_k$ in $\pi$ by truly random function. $\pi^{RF}$ is CPA and CCA1 secure. Because the adversary only has access to encryption oracle after chosen $m_0$ and $m_1$. The only chance adversary can differentiate $\text{Enc}_k(m_0) = r_0 \oplus f(r_0)$ and $\text{Enc}_k(m_1) = r_1 \oplus f(r_1)$ is that the encryption oracle happens to have sampled the same $r_0$ or $r_1$ in some previous query, or additionally, in CCA1 attack, the attacker happens to have asked decryption oracle to decrypt ciphertext like $r_0||m$ or $r_1||m$. All cases have only negligible probabilities.

Given $\pi^{RF}$ is CPA and CCA2 secure, then so is $\pi$. Otherwise, if there exists one distinguisher $D$ that can differentiate the experiment results (IND$_0^{\text{Enc}_k;\text{Enc}_k}$ and IND$_1^{\text{Enc}_k;\text{Enc}_k}$ in case of CPA attack, while IND$_0^{\text{Enc}_k,\text{Dec}_k;\text{Enc}_k}$ and IND$_1^{\text{Enc}_k,\text{Dec}_k;\text{Enc}_k}$ in case of CCA1 attack) then we can construct another distinguisher which internally uses $D$ to differentiate PRF from truly random function.

### 3.11 CCA2 Secure Encryption Scheme

However, the encryption scheme $\pi$ is not CCA2 secure. Consider the attack: in experiment IND$_b^{\text{Enc}_k,\text{Dec}_k;\text{Enc}_k,\text{Dec}_k}$, given ciphertext $r||c = \text{Enc}_k(m_b)$, the attacker can ask the decryption oracle to decrypt $r||c + 1$. As this is not the challenge itself, this is allowed. Actually $r||c + 1$ is the ciphertext for message $m_b + 1$, as

$$\text{Enc}_k(m_b + 1) = r||(m_b + 1) \oplus f_k(r) = r||m_b \oplus f_k(r) + 1 = r||c + 1$$

Thus the decryption oracle would reply $m_b + 1$. The adversary can differentiate which message's encryption it is given.

We construct a new encryption scheme that is CCA2 secure. Let $\{f_s\}$ and $\{g_s\}$ be families of PRF on space $\{0, 1\}^{|s|} \rightarrow \{0, 1\}^{|s|}$.

$\pi' = (\text{Gen}', \text{Enc}', \text{Dec}')$:
Algorithm 9: Many-message CCA2-secure Encryption Scheme

Assume \( m \in \{0, 1\}^n \) and let \( \{f_k\} \) be a PRF family

\[
\text{Gen}(1^n) : k_1, k_2 \leftarrow U_n
\]

\[
\text{Enc}_{k_1,k_2}(m) : \text{Pick } r \leftarrow U_n. \text{ Set } c_1 \leftarrow m \oplus f_{k_1}(r). \text{ Output } (r, c_1, f_{k_2}(c))
\]

\[
\text{Dec}_{k_1,k_2}((r, c_1, c_2)) : \text{If } f_{k_2}(c_1) \neq c_2, \text{ output } \bot. \text{ Else output } c_1 \oplus f_{k_1}(r)
\]

Now we show that:

**Theorem 68.0.** \( \pi' \) is CCA2 attack secure.

**Proof.** The main idea is to prove by contradiction. In specific, if there is an CCA2 attack on \( \pi' \), then there is an CPA attack on \( \pi \), which would contradict with the fact that \( \pi \) is CPA secure.

A CCA2 attack on \( \pi' \) is a PPT machine \( A' \), s.t. it can differentiate \( \{\text{IND}_0^{\text{Enc}_k,\text{Dec}_k;\text{Enc}_k,\text{Dec}_k}\} \) and \( \{\text{IND}_1^{\text{Enc}_k,\text{Dec}_k;\text{Enc}_k,\text{Dec}_k}\} \). Visually, it works as that in figure ?? . The attacker \( A' \) needs accesses to the \( \text{Enc}_k \) and \( \text{Dec}_k \) oracles. To built an CPA attack on \( \pi \), we want to construct another machine \( A \) as depicted in figure ?? . To leverage the CCA2 attacker \( A' \), we simulate \( A \) as in figure ?? which internally uses \( A' \).

Formally, the simulator works as follows:

- Whenever \( A' \) asks for an encryption of message \( m \), \( A \) asks its own encryption oracle \( \text{Enc}_{s_1} \) to get \( c_1 = \text{Enc}_{s_1}(m) \). But \( A' \) expects encryption \( c_1||c_2 \), requiring \( s_2 \) to evaluate \( g_{s_2}(c_1) \), which \( A \) has no access to. Thus let \( c_2 \leftarrow \{0, 1\}^n \), and reply \( c_1||c_2 \).

- Whenever \( A' \) asks for a decryption \( c_1||c_2 \). If we previously gave \( A' \) \( c_1||c_2 \) to answer an encryption query of some message \( m \), then reply \( m \), otherwise reply \( \bot \).

- Whenever \( A' \) outputs \( m_0, m_1 \), output \( m_0, m_1 \).

- Upon receiving \( c \), feed \( c||r \), where \( r \leftarrow \{0, 1\}^n \) to \( A' \).

- Finally, output \( A' \)'s output.

Consider encryption scheme \( \pi'^{RF} = (\text{Gen}^{RF}, \text{Enc}^{RF}, \text{Dec}^{RF}) \) which is derived from \( \pi' \) by replacing every appearance of \( g_{s_2} \) with a truly random function.

Note that the simulated \( \text{Enc}' \) is just \( \text{Enc}^{RF} \), and \( \text{Dec}' \) is very similar to \( \text{Dec}^{RF} \). Then \( A' \) inside the simulator is nearly conducting CCA2 attack on
π\textsuperscript{RF} with the only exception when \( A' \) asks an \( c_1||c_2 \) to \( Dec' \) which is not returned by a previous encryption query and is a correct encryption, in which case \( Dec' \) falsely returns \( \perp \). However, this only happens when \( c_2 = f(c_1) \), where \( f \) is the truly random function. Without previous encryption query, the attacker can only guess the correct value of \( f(c_1) \) w.p. \( \frac{1}{2^n} \), which is negligible.

Thus we reach that: if \( A' \) breaks CCA2 security of \( \pi\textsuperscript{RF} \), then it can break CPA security of \( \pi \). The premise is true as by assumption \( A' \) breaks CCA2 security of \( \pi' \), and that PRF is indistinguishable from a truly random function.

### 3.12 Non-Malleability

Until this point we have discussed encryptions that prevent a passive attacker from discovering any information about messages that are sent. In some situations, however, we may want to prevent an attacker from creating a new message from a given encryption.

Consider an auction for example. Suppose the Bidder Bob is trying to send a message containing his bid to the Auctioneer Alice. Private key encryption could prevent an attacker Eve from knowing what Bob bids, but if she could construct a message that contained one more than Bob’s bid, then she could win the auction.

We say that an encryption scheme that prevents these kinds of attacks is non-malleable. Informally, if a scheme is non-malleable, then it is impossible to output an encrypted message containing any function of a given encrypted message. Formally, we have the following definition:

**Definition 69.1 (Non-Malleability).** Let \((Gen, Enc, Dec)\) be an encryption scheme. Define the following experiment:

\[
\text{NM}_b(\Pi, ma, n)
\]

\[
k \leftarrow \text{Gen}(1^n)
\]

\[
m_0, m_1, \text{ state } \leftarrow A^{O_1(k)}(1^n)
\]

\[
c \leftarrow \text{Enc}_k(m_b)
\]

\[
c'_1, c'_2, c'_3, \ldots, c'_\ell \leftarrow A^{O_2(k)}(c, \text{ state})
\]

\[
m'_i \leftarrow \begin{cases} 
\perp & \text{if } c_i = c \\
\text{Dec}_k(c'_i) & \text{otherwise}
\end{cases}
\]

Output \((m'_1, m'_2, \ldots, m'_\ell)\)
Then \((Gen, Enc, Dec)\) is non-malleable if for every non-uniform PPT \(A\), and for every non-uniform PPT \(D\), there exists a negligible \(\epsilon\) such that for all \(m_0, m_1 \in \{0, 1\}^n\),

\[
\Pr[\mathcal{D}(NM_0(\Pi, A, n)) = 1] - \Pr[\mathcal{D}(NM_1(\Pi, A, n)) = 1] \leq \epsilon(n)
\]

One non-trivial aspect of this definition is the conversion to \(\perp\) of queries that have already been made (step 4). Clearly without this, the definition would be trivially unsatisfiable, because the attacker could simply “forge” the encryptions that they have already seen by replaying them.

Relation Based Non-Malleability

We chose this definition because it mirrors our definition of secrecy in a satisfying way. However, an earlier and arguably more natural definition can be given by formalizing the intuitive notion that the attacker cannot output an encryption of a message that is related to a given message. For example, we might consider the relation \(R_{\text{next}}(x) = \{x + 1\}\), or the relation \(R_{\text{within-one}}(x) = \{x - 1, x, x + 1\}\). We want to ensure that the encryption of \(x\) doesn’t help the attacker encrypt an element of \(R(x)\). Formally:

**Definition 70.1 (Relation Based Non-Malleability).** We say that an encryption scheme \((Gen, Enc, Dec)\) is relation based non-malleable if for every PPT adversary \(A\) there exists a PPT simulator \(S\) such that for all PPT-recognizable relations \(R\), there exists a negligible \(\epsilon\) such that for all \(m \in \mathcal{M}\) with \(|m| = n\), and for all \(z\), it holds that

\[
\begin{align*}
|\Pr[\mathcal{D}(NM(A(z), m)) \in R(m)] &- \Pr[k \leftarrow Gen(1^n); c \leftarrow S(1^n, z); m' = Dec_k(c) : m' \in R(m)]| < \epsilon \\
\end{align*}
\]

where \(i\) ranges from 1 to a polynomial of \(n\) and \(NM\) is defined as above.

This definition is equivalent to the non-relational definition given above.

**Theorem 70.1.** \((Enc, Dec, Gen)\) is a non-malleable encryption scheme if and only if it is a relation-based non-malleable encryption scheme.

**Proof.** \((\Rightarrow)\) Assume that the scheme is non-malleable by the first definition. For any given adversary \(A\), we need to produce a simulator \(S\) that hits any given relation \(R\) as often as \(A\) does. Let \(S\) be the machine that performs the first 3 steps of \(NM(A(z), m')\) and outputs the sequence of cyphertexts, and let \(\mathcal{D}\) be the distinguisher for the relation \(R\). Then

\[
|\Pr[\mathcal{D}(NM(A(z), m)) \in R(m)] - \Pr[k \leftarrow Gen(1^n); c \leftarrow S(1^n, z); m' = Dec_k(c) : m' \in R(m)]| \leq \epsilon
\]

\[
|\Pr[\mathcal{D}(NM(A(z), m))] - \Pr[\mathcal{D}(NM(A(z), m'))]| \leq \epsilon
\]
3.13. PUBLIC KEY ENCRYPTION

as required.

(⇐) Now, assume that the scheme is relation-based non-malleable. Given
an adversary \( \mathcal{A} \), we know there exists a simulator \( \mathcal{S} \) that outputs related
encryptions as well as \( \mathcal{A} \) does. The relation-based definition tells us that
\( \text{NM}(\mathcal{A}(z), m_0) \approx \text{Dec}(\mathcal{S}()) \) and \( \text{Dec}(\mathcal{S}()) \approx \text{NM}(\mathcal{A}(z), m_1) \). Thus, by
the hybrid lemma, \( \text{NM}(\mathcal{A}(z), m_0) \approx \text{NM}(\mathcal{A}(z), m_1) \) which is the first
definition of non-malleability.

Non-Malleability and Secrecy

Note that non-malleability is a distinct concept from secrecy. For example,
one-time pad is perfectly secret, yet is not non-malleable (since one can eas-
ily produce the encryption of \( a \oplus b \) give then encryption of \( a \), for example).
However, if we consider CCA2 attacks, then the two definitions coincide.

**Theorem 71.1.** An encryption scheme \( \Sigma = (\text{Enc}, \text{Dec}, \text{Gen}) \) is CCA2 secret if and
only if it is CCA2 non-malleable

**Proof sketch.** If \( \Sigma \) is not CCA2 non-malleable, then a CCA2 attacker can
break secrecy by changing the provided encryption into a related encryption,
using the decryption oracle on the related message, and then distinguishing
the unencrypted related messages. Similarly, if \( \Sigma \) is not CCA2 secret, then a
CCA2 attacker can break non-malleability by simply decrypting the cyphertext,
applying a function, and then re-encrypting the modified message.

3.13 Public Key Encryption

So far, our model of communication allows the encryptor and decryptor to
meet in advance and agree on a secret key which they later can use to send
private messages. Ideally, we would like to drop this requirement of meeting
in advance to agree on a secret key. At first, this seems impossible. Certainly
the decryptor of a message needs to use a secret key; otherwise, nothing pre-
vents the eavesdropper from running the same procedure as the decryptor
to recover the message. It also seems like the encryptor needs to use a key
because otherwise the key cannot help to decrypt the cyphertext.

The flaw in this argument is that the encrypter and the decryptor need not
share the same key, and in fact this is how public key cryptography works. We
split the key into a secret decryption key \( sk \) and a public encryption key \( pk \).
The public key is published in a secure repository, where anyone can use it
to encrypt messages. The private key is kept by the recipient so that only she can decrypt messages sent to her.

We define a public key encryption scheme as follows:

**Definition 72.1 (Public Key Encryption Scheme).** A triple \((\text{Gen}, \text{Enc}, \text{Dec})\) is a public key encryption scheme over a message space \(M\) if

1. \((pk, sk) \leftarrow \text{Gen}(1^n)\) is a p.p.t. algorithm that produces a key pair \((pk, sk)\)
2. \(c \leftarrow \text{Enc}_{pk}(m)\) is a p.p.t. algorithm that given \(pk\) and \(m \in M\) produces a cyphertext \(c\)
3. \(m \leftarrow \text{Dec}_{sk}(c)\) is a p.p.t. algorithm that given a ciphertext \(c\) and secret key \(sk\) produces a message \(m \in M \cup \bot\)
4. For all \(m \in M\) and for all \((pk, sk) \leftarrow \text{Gen}(1^n),\)
   \[\text{Dec}_{sk}(\text{Enc}_{pk}(m)) = m\]

We allow the decryption algorithm to produce a special symbol \(\bot\) when the input ciphertext is “undecipherable.” The security property for public-key encryption can be defined using an experiment similar to the ones used in the definition for secure private key encryption. Notice, however, that by giving the adversary the public key, there is no need to provide an encryption oracle. Thus, the oracle only plays a role in the CCA1/2 definitions.

**Definition 72.2 (Secure Public Key Encryption).** Let \(\Pi = (\text{Gen}, \text{Enc}, \text{Dec})\) be a public key encryption. Let the random variable \(\text{Ind}_b(\Pi, \mathcal{A}, n)\) where \(\mathcal{A}\) is a non-uniform p.p.t. adversary, \(n \in \mathbb{N}\), and \(b \in \{0, 1\}\) denote the output of the following experiment:

\[
\begin{align*}
\text{Ind}_b(\Pi, \mathcal{A}, n) \\
(pk, sk) & \leftarrow \text{Gen}(1^n) \\
m_0, m_1, \text{state} & \leftarrow A_{O_1}(sk)(1^n, pk) \\
c & \leftarrow \text{Enc}_{pk}(m_b) \\
\text{Output} & A_{O_2}(k)(c, \text{state})
\end{align*}
\]

We say that \(\Pi\) is CPA/CCA1/CCA2 secure if for all non-uniform p.p.t. \(\mathcal{A}\), the following two distributions are computationally indistinguishable:

\[
\{\text{Ind}_0(\Pi, \mathcal{A}, n)\}_{n \in \mathbb{N}} \approx \{\text{Ind}_1(\Pi, \mathcal{A}, n)\}_{n \in \mathbb{N}}
\]

The oracles \(O_1, O_2\) are defined as follows:
3.13. PUBLIC KEY ENCRYPTION

- CPA \([., .]\)
- CCA1 \([\text{Dec}, .]\)
- CCA2 \([\text{Dec}, \text{Dec}^*]\)

where \(\text{Dec}^*\) answers all queries except for the challenge ciphertext \(c\).

With these definitions, there are some immediate impossibility results:

**Perfect secrecy** Perfect secrecy is not possible (even for small message spaces) since an unbounded adversary could simply encrypt every message in \(\mathcal{M}\) with every random string and compare with the challenge ciphertext to learn the underlying message.

**Deterministic encryption** It is also impossible to have a deterministic encryption algorithm because otherwise an adversary could simply encrypt and compare the encryption of \(m_0\) with the challenge ciphertext to distinguish the two experiments.

In addition, it is a straightforward exercise to show that single-message security implies many-message security.

**Constructing a PK encryption system**

Trapdoor permutations seem to fit the requirements for a public key cryptosystem. We could let the public key be the index \(i\) of the function to apply, and the private key be the trapdoor \(t\). Then we might consider \(\text{Enc}(m, i) = f_i(m)\), and \(\text{Dec}(c, i, t) = f_i^{-1}(c)\). This makes it easy to encrypt, and easy to decrypt with the public key, and hard to decrypt without. Using the RSA function defined in Theorem 41.1, this construction yields the commonly used RSA cryptosystem.

However, according to our definition, this construction does not yield a secure encryption scheme. In particular, it is deterministic, so it is subject to comparison attacks. A better scheme (for single-bit messages) is to let \(\text{Enc}(x, i) = \{r \leftarrow \{0, 1\}^n : \langle f_i(r), b(r) \oplus m \rangle\}\) where \(b\) is a hardcore bit for \(f\). This scheme is secure, because distinguishing encryptions of 0 and 1 is essentially the same as recognizing the hardcore bit of a one-way permutation, which we have argued is infeasible.

**Theorem 73.1.** If one-way trapdoor permutations exist, then scheme 11 is a secure public-key encryption system for the message space \(\mathcal{M} = \{0, 1\}\).

**Proof:** To show that \((\text{Gen}, \text{Enc}, \text{Dec})\) is a secure Public-Key Encryption system, we must show that

\[
\{(pk, sk) \leftarrow \text{Gen}(1^n) : \text{Enc}_{pk}(0)\} \approx \{(pk, sk) \leftarrow \text{Gen}(1^n) : \text{Enc}_{pk}(1)\}
\]
Algorithm 10: 1-Bit Secure Public Key Encryption

Let \((f_i, f_i^{-1})_{i \in I}\) be a family of one-way trapdoor permutations and let \(b_i\) be the hard-core bit corresponding to \(f_i\). Let \(Gen_T\) be the p.p.t that samples a trapdoor permutation index from \(I\).

\[
\begin{align*}
\text{Gen}(1^n) : (f_i, f_i^{-1}) &\leftarrow Gen(1^n). \text{ Output } (pk, sk) \leftarrow ((f_i, b_i), (b_i, f_i^{-1})). \\
\text{Enc}_{pk}(m) : \text{ Pick } r &\leftarrow \{0,1\}^n. \text{ Output } (f_i(r), b_i(r) \oplus m). \\
\text{Dec}_{sk}(c_1, c_2) : \text{ Compute } r &\leftarrow f_i^{-1}(c_1). \text{ Output } b_i(r) \oplus c_2.
\end{align*}
\]

We define the following distributions.

\[
\begin{align*}
\text{Ind}_0 &= \{(pk, sk) \leftarrow \text{Gen}(1^n) : \text{Enc}_{pk}(0)\} \\
&= \{(pk, sk) \leftarrow \text{Gen}(1^n) : r \leftarrow \{0,1\}^n : (f_{pk}(r), b(r) \oplus 0)\} \\
\text{Ind}_1 &= \{(pk, sk) \leftarrow \text{Gen}(1^n) : r \leftarrow \{0,1\}^n : (f_{pk}(r), b(r) \oplus 1)\} \\
\text{H}_1 &= \{(pk, sk) \leftarrow \text{Gen}(1^n) : r \leftarrow \{0,1\}^n : (f_{pk}(r), b(r))\} \\
\text{H}_2 &= \{(pk, sk) \leftarrow \text{Gen}(1^n) : r \leftarrow \{0,1\}^n ; r' \leftarrow \{0,1\} : (f_{pk}(r), r')\} \\
\text{H}_3 &= \{(pk, sk) \leftarrow \text{Gen}(1^n) ; r \leftarrow \{0,1\}^n ; r' \leftarrow \{0,1\} : (f_{pk}(r), r' \oplus 1)\}
\end{align*}
\]

The proof is by the hybrid argument. We first argue that \(\text{Ind}_0 = \text{H}_1\): This follows trivially because \(b(r) \oplus 0 = b(r)\). Next, we claim that \(\text{H}_1 \approx \text{H}_2\). This follows from the definition of hard-core bits. Now \(\text{H}_2 = \text{H}_3\) because \(\{r' \leftarrow \{0,1\} : r'\} = \{r' \leftarrow \{0,1\} : r' \oplus 1\}\) is the uniform distribution over \(\{0,1\}\). Finally, by the definition of hard-core bits, we have \(\text{H}_3 \approx \text{Ind}_1\). Therefore, by the Hybrid Lemma, we have that \(\text{Ind}_0 \approx \text{Ind}_1\). \(\square\)

3.14 El-Gamal Public Key Encryption scheme

The El-Gamal public key encryption scheme is a popular and simple public key encryption scheme that is far more efficient than the one just presented. However, this efficiency requires us to make a new complexity assumption called the Decisional Diffie-Hellman Assumption (DDH).

**Conjecture 74.1** (Decisional Diffie-Hellman assumption (DDH)). *For all p.p.t \(A\), the following two distributions are computationally indistinguishable*

\[
\begin{align*}
\{p \leftarrow \Pi_n, y \leftarrow \text{Gen}_p, a, b \leftarrow \mathbb{Z}_q : p, y^a, y^b, y^{ab}\}_n \\
\approx \{p \leftarrow \Pi_n, y \leftarrow \text{Gen}_p, a, b, z \leftarrow \mathbb{Z}_q : p, y^a, y^b, y^z\}_n
\end{align*}
\]
Notice that this assumption implies the discrete-log assumption 40.1 since after solving the discrete log twice on the first two components, it is easy to distinguish whether the third component is $y^{ab}$ or not. Again, with this assumption it is easy to see that the above key-exchange scheme is indistinguishably secure.

We now construct a Public Key Encryption scheme based on the DDH assumption.

**Algorithm 11: El-Gamal Secure Public Key Encryption**

\begin{align*}
\text{Gen}(1^n): & \text{ Pick a safe prime } p = 2q + 1 \text{ of length } n. \text{ Choose } h \text{ a generator from } QR_p. \text{ Choose } a \leftarrow \mathbb{Z}_q. \text{ Output the public key } pk \text{ as } \\
& pk \leftarrow (p, h, h^a \mod p) \text{ and } sk \leftarrow (p, h, a). \\
\text{Enc}_{pk}(m): & \text{ Choose } b \leftarrow \mathbb{Z}_q. \text{ Output } (h^b, h^{ab} \cdot m \mod p). \\
\text{Dec}_{sk}(c = (c_1, c_2)): & \text{ Output } c_2/c_1^a \mod p.
\end{align*}

This is secure assuming the DDH assumption because $h^{ab}$ is indistinguishable from a random element and hence $h^{ab} \cdot m$ is indistinguishable from a random element too.

**Theorem 75.1.** Under the DDH Assumption, the El-Gamal encryption scheme is CPA-secure.

**Proof.**

Notice that instead of working in $\mathbb{Z}_p^*$, the scheme only works with elements of the subgroup $QR_p$ (even though the operations will be performed $\mod p$). We further restrict our primes $p$ to be of the form $2q + 1$ where $q$ is prime too. Such primes are $p$ are called **safe primes** and the corresponding $q$ is called a **Sophie Germain prime**. This allows for an easy method to test whether an element $h$ is a generator.

### 3.15 A Note on Complexity Assumptions

Throughout this semester, we have built a hierarchy of constructions. At the bottom of this hierarchy are computationally difficult problems such as one-way functions, one-way permutations, and trapdoor permutations. Our efficient constructions of these objects were further based on specific number-theoretic assumptions, including factoring, RSA, discrete log, and decisional Diffie-Hellman.
Using these hard problems, we constructed several primitives: pseudo-random generators, pseudorandom functions, and private-key encryption schemes. Although our constructions were usually based on one-way permutations, it is possible to construct these using one-way functions. Further, one-way functions are a minimal assumption, because the existence of any of these primitives implies the existence of one-way functions.

Public-key encryption schemes are noticeably absent from the list of primitives above. Although we did construct two schemes, it is unknown how to base such a construction on one-way functions. Moreover, it is known to be impossible to create a black-box construction from one-way functions.