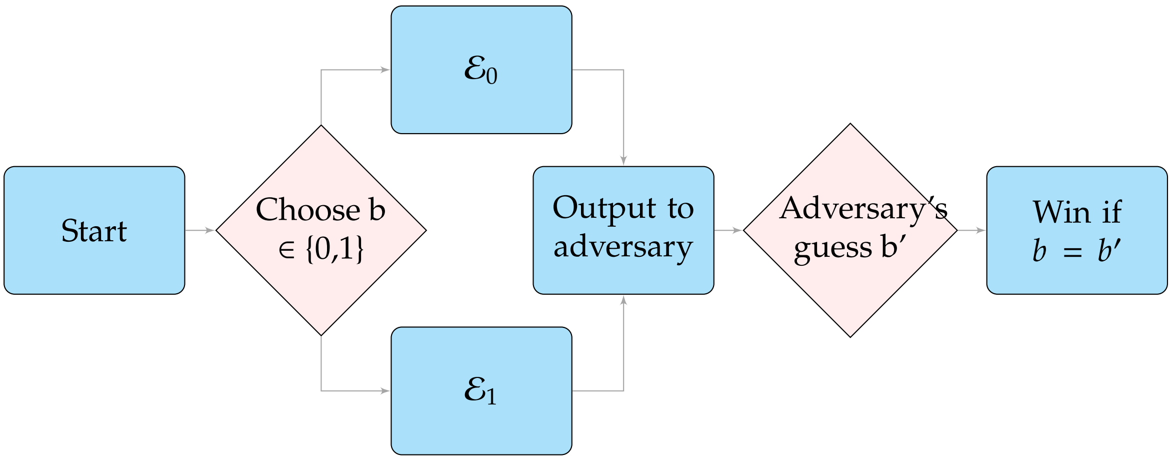
Title

Ji Yong-Hyeon, Kim Dong-Hyeon



Department of Information Security, Cryptology, and Mathematics
College of Science and Technology
Kookmin University

November 8, 2023

Contents

List of Symbols

$\mathtt{0}^\lambda, \mathtt{1}^\lambda$	$\underbrace{\mathtt{0}\mathtt{0}\cdots\mathtt{0}}_{\lambda \text{ times}}, \underbrace{\mathtt{1}\mathtt{1}\cdots\mathtt{1}}_{\lambda \text{ times}}$: λ-bit zero/one sequence
$\mathcal{A} \diamond \mathcal{L}$	The result of **linking** $\mathcal{A}$ to $\mathcal{L}$
$\$$	Randomeness
$\equiv$	Interchangability; Identical

Chapter 1

One-Time Pad & Kerckhoff's Principle

Kerchkhoffs' Principle:

> Design your system to be secure even if the attacker has complete knowledge of all its algorithms.

One-time Pad (OTP)

Construction 1.1. The specific KeyGen, Enc, and Dec algorithms for **one-time pad** are given below:

$\underline{\mathsf{KeyGen}:}$	$\underline{\mathsf{Enc}(k, m \in \{\mathtt{0}, \mathtt{1}\}^\lambda):}$	$\underline{\mathsf{Dec}(k, c \in \{\mathtt{0}, \mathtt{1}\}^\lambda):}$
$k \xleftarrow{\$} \{\mathtt{0}, \mathtt{1}\}^\lambda$	return $k \oplus m$	return $k \oplus c$
return k		

Corectness of OTP

Proposition 1.1.

$$(\forall k, m \in \{\mathtt{0}, \mathtt{1}\}^\lambda) \quad \mathsf{Dec}(k, \mathsf{Enc}(k, m)) = m.$$

Proof. Let $k, m \in \{\mathtt{0}, \mathtt{1}\}^\lambda$ then

$$\begin{aligned}
\mathsf{Dec}(k, \mathsf{Enc}(k, m)) = \mathsf{Dec}(k, k \oplus m) &= k \oplus (k \oplus m) \\
&= (k \oplus k) \oplus m \\
&= \mathtt{0}^\lambda \oplus m \\
&= m.
\end{aligned}$$

□

Remark 1.1 (`Eavesdrop` Algorithm)**.** From Eve's perspective, seeing a ciphertext corresponds to receiving an output from the following algorithm:

$$\begin{array}{|l|}\hline \underline{\texttt{Eavesdrop}(m \in \{\texttt{0},\texttt{1}\}^\lambda)} \\ \quad k \xleftarrow{\$} \{\texttt{0},\texttt{1}\}^\lambda \\ \quad c := k \oplus m \\ \quad \text{return } c \\ \hline \end{array}$$

Theorem 1.2. *Let $m \in \{\texttt{0},\texttt{1}\}^\lambda$. The distribution $\texttt{Eavesdrop}(m)$ is the **uniform distribution** on $\{\texttt{0},\texttt{1}\}^\lambda$. In other words,*

$$m, m' \in \{\texttt{0},\texttt{1}\}^\lambda \implies \text{dist}(\texttt{Eavesdrop}(m)) \sim \text{dist}(\texttt{Eavesdrop}(m')).$$

Proof. content... $\square$

Chapter 2

The Basics of Provable Security

2.1 How to Write a Security Definition

2.1.1 Syntax and Correctness

Encryption Syntax

Definition 2.1. A **symmetric-key encryption (SKE) scheme** consists of the following algorithms:

- KeyGen outputs a key $k = \mathsf{KeyGen}(1^\lambda) \in \mathcal{K}$
- $\mathsf{Enc} : \mathcal{K} \times \mathcal{M} \to \mathcal{C}$
- $\mathsf{Dec} : \mathcal{K} \times \mathcal{C} \to \mathcal{M}$

We call $\mathcal{K}$ the **key space**, $\mathcal{M}$ the **message space**, and $\mathcal{C}$ the **ciphertext space** of the scheme.

Remark 2.1. Note that

- KeyGen is a randomized algorithm[1].
- Enc is a (possibly randomized) algorithm[2].
- Dec is a deterministic algorithm[3].

Remark 2.2. We refer to the entire scheme by a single variable Σ, i.e.,

$$\Sigma = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}).$$

Remark 2.3. We write

$$\begin{array}{ccc} \Sigma.\mathsf{KeyGen}, & \Sigma.\mathsf{Enc}, & \Sigma.\mathsf{Dec}, \\ \Sigma.\mathcal{K}, & \Sigma.\mathcal{M}, & \Sigma.\mathcal{C} \end{array}$$

to refer to its components.

[1] An algorithm that makes use of random numbers.
[2] It could operate deterministically or non-deterministically depending on specific conditions or parameters.
[3] An algorithm that does produces the same output for the same input, every time it's run.

SKE Correctness

Definition 2.2. An encryption scheme Σ satisfies **correctness** if

$$(\forall k \in \Sigma.\mathcal{K})\,(\forall m \in \Sigma.\mathcal{M}) \quad \Pr\left[\Sigma.\mathsf{Dec}(k, \Sigma.\mathsf{Enc}(k,m)) = m\right] = 1.$$

Remark 2.4. The definition is written in terms of a probability because Enc is allowed to be a randomized algorithm. In other words, decrypting a ciphertext with the same key that was used for encryption must *always* result in the original plaintext.

Example 2.1. content...

2.1.2 "Real-vs-Random" Style of Security Definition

> "an encryption scheme is a good one if its ciphertexts *look like* random junk to an attacker"

Security definitions always consider **the attacker's view** of the system.

> "an encryption scheme is a good one if its ciphertexts *look like* random junk to an attacker ... when each key is secret and used to encrypt only one plaintext, even when the attacker chooses the plaintexts."

A concise way to express all of these details is to consider **the attacker as a calling program** to the following subroutine:

$$\begin{array}{|l|}\hline \underline{\mathtt{CTXT}(m \in \Sigma.\mathcal{M}):} \\ \quad k \leftarrow \Sigma.\mathsf{KeyGen} \\ \quad c := \Sigma.\mathsf{Enc}(k,m) \\ \quad \text{return } c \\ \hline \end{array}$$

Example 2.2 (One-Time Pad (OTP))**.** a

$$\begin{array}{|l|}\hline \underline{\mathtt{CTXT}(m):} \\ \quad k \leftarrow \{\mathtt{0},\mathtt{1}\}^\lambda \quad \text{//KeyGen of OTP} \\ \quad c := k \oplus m \quad \text{//Enc of OTP} \\ \quad \text{return } c \\ \hline \end{array} \quad \text{vs.} \quad \begin{array}{|l|}\hline \underline{\mathtt{CTXT}(m):} \\ \quad c := \{\mathtt{0},\mathtt{1}\}^\lambda \quad //\mathcal{C} \text{ of OTP} \\ \quad \text{return } c \\ \hline \end{array}$$

> "an encryption scheme is a good one if, when you plug its KeyGen and Enc algorithms into the template of the CTXT subroutine above, the two implementations of CTXT induce identical behavior in every calling program."

2.1.3 “Left-vs-Right” Style of Security Definition

2.2 Formalisms for Security Definition

Library

Definition 2.3. A **library** $\mathcal{L}$ is a collection of subroutines and private/static variables.

Example 2.3. Here is a familiar library and one possible calling program:

$$\begin{array}{|l|}\hline \hfill\mathcal{L}\hfill \\ \hline \underline{\texttt{CTXT}(m):} \\ \quad k \xleftarrow{\$} \{\texttt{0},\texttt{1}\}^\lambda \\ \quad c := k \oplus m \\ \quad \text{return } c \\ \hline \end{array} \quad , \quad \begin{array}{|l|}\hline \hfill\mathcal{A}\hfill \\ \hline m \leftarrow \{\texttt{0},\texttt{1}\}^\lambda \\ c := \texttt{CTXT}(m) \\ \text{return } m \stackrel{?}{=} c \\ \hline \end{array} \quad .$$

Then

$$\Pr\left[\mathcal{A} \diamond \mathcal{L} \Rightarrow \texttt{true}\right] = \frac{1}{2^\lambda}.$$

Interchangeability

Definition 2.4. Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be two libraries that have the same interface. We say that $\mathcal{L}_1$ and $\mathcal{L}_2$ are **interchangeable**, and write $\mathcal{L}_1 \equiv \mathcal{L}_2$, if $\forall \mathcal{A}$:

$$\Pr\left[\mathcal{A} \diamond \mathcal{L}_1 \Rightarrow \texttt{true}\right] = \Pr\left[\mathcal{A} \diamond \mathcal{L}_2 \Rightarrow \texttt{true}\right].$$

One-Time Uniform Ctxts

Definition 2.5. An encryption scheme Σ has **one-time uniform cipher-texts** if

$$\begin{array}{|l|}\hline \hfill\mathcal{L}^{\Sigma}_{\text{ots\$-real}}\hfill \\ \hline \underline{\texttt{CTXT}(m \in \Sigma.\mathcal{M}):} \\ \quad k \leftarrow \Sigma.\mathsf{KeyGen} \\ \quad c \leftarrow \Sigma.\mathsf{Enc}(k,m) \\ \quad \text{return } c \\ \hline \end{array} \equiv \begin{array}{|l|}\hline \hfill\mathcal{L}^{\Sigma}_{\text{ots\$-rand}}\hfill \\ \hline \underline{\texttt{CTXT}(m \in \Sigma.\mathcal{M}):} \\ \quad c \leftarrow \Sigma.\mathcal{C} \\ \quad \text{return } c \\ \hline \end{array}$$

One-Time Secrecy (OTS)

Definition 2.6. One-time secrecy is a property of an encryption scheme where an adversary cannot gain any information about the plaintext message from the ciphertext, even if they know the encryption key was used only once.

$\mathcal{L}^{\Sigma}_{\textsf{ots-L}}$		$\mathcal{L}^{\Sigma}_{\textsf{ots-R}}$
$\texttt{Eve}(m_L, m_R \in \Sigma.\mathcal{M})$:	$\equiv$	$\texttt{Eve}(m_L, m_R \in \Sigma.\mathcal{M})$:
$k \leftarrow \Sigma.\textsf{KeyGen}$		$k \leftarrow \Sigma.\textsf{KeyGen}$
$c \leftarrow \Sigma.\textsf{Enc}(k, m_L)$		$c \leftarrow \Sigma.\textsf{Enc}(k, m_R)$
return c		return c

2.3 How to Demonstrate Insecurity with Attacks

2.4 How to Prove Security with The Hybrid Technique

2.5 How to Compare/Contract Security Definitions

Exercises

Exercise 2.1. In abstract algebra, a (finite) group is a finite set $\mathbb{G}$ of items together with an operator $\otimes$ satisfying the following axioms:

- **Closure:** for all $a, b \in \mathbb{G}$, we have $a \otimes b \in \mathbb{G}$
- **Identity:** there is a special *identity element* $e \in \mathbb{G}$ that satisfies $e \otimes a = a \otimes e = a$ for all $a \in \mathbb{G}$. We typically write "1" rather than e for the identity element.
- **Associativity:** for all $a, b, c \in \mathbb{G}$, we have $(a \otimes b) \otimes c = a \otimes (b \otimes c)$.
- **Inverses:** for all $a \in \mathbb{G}$, there exists an inverse element $b \in \mathbb{G}$ such that $a \otimes b = b \otimes a$ is the identity element of $\mathbb{G}$. We typically write "a^{-1}" for the inverse of a.

Define the following encryption scheme in terms of an arbitrary group $(\mathbb{G}, \otimes)$:

$\mathcal{K} = \mathbb{G}$	$\underline{\textsf{KeyGen}:}$	$\underline{\textsf{Enc}(k, m):}$	$\underline{\textsf{Dec}(k, c):}$
$\mathcal{M} = \mathbb{G}$	$k \leftarrow \mathbb{G}$	return $k \otimes m$	??
$\mathcal{C} = \mathbb{G}$	return k		

(a) Prove that $\{0, 1\}^\lambda$ is a group with respect to the xor operator. What is the identity element, and what is the inverse of a value $x \in \{0, 1\}^\lambda$?

(b) Fill in the details of the Dec algorithm and prove (using the group axioms) that the scheme satisfies correctness.

(c) Prove that the scheme satisfies one-time secrecy.

Exercise 2.2. Prove that if an encryption scheme Σ has $|\Sigma.\mathcal{K}| < |\Sigma.\mathcal{M}|$ then it cannot satisfy one-time secrecy.

[Hint: The definition of interchangeability does not place any restriction on the running time of the distinguisher/calling program. Even an exhaustive brute-force attack would be valid]

Solution. content... □

Chapter 3

Cryptography on Intractable Computations

3.1 What Qualifies as a "Computationally Infeasible" Attack?

Polynomial Time

Definition 3.1. A program runs in **polynomial time** if

$$\exists c > 0 : \forall n \geq n_0 : \mathsf{Time}(n) \leq n^c,$$

where Time is the time taken by the algorithm on inputs of size n. n_0 is constant size of the input. That is, there exists a constant $c > 0$ such that for all sufficiently long input strings x with $|x| = n$, the program stops after no more than $O(n^c)$ steps.

Remark 3.1. We see "polynomial-time" as a synonym for "efficient."

Example 3.1. $\gcd(a, b)$ can be computed using $O((\log_2 a)^3)$ bit operation if $a > b$.

Example 3.2.

Efficient algorithm known:	**No known efficient algorithm:**
Computing GCDs	Factoring integers
Arithmetic $\bmod N$	Computing $\phi(N)$ given N
Inverses $\bmod N$	Discrete logarithm
Exponentiation $\bmod N$	Square roots mod composite N

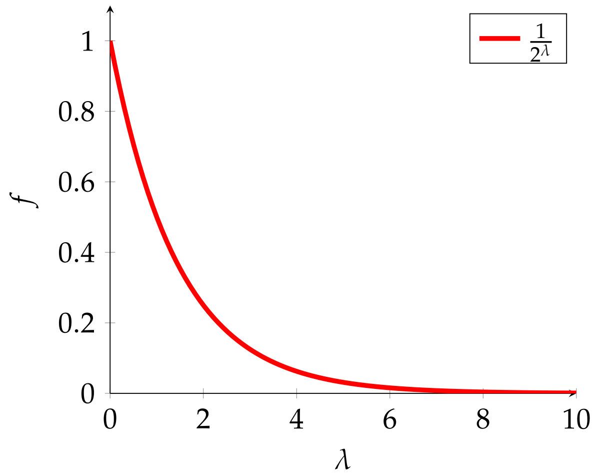
Again, "efficient" means polynomial-time. Furthermore, we only consider polynomial-time algorithms that run on standard, *classical* computers. In fact, all of the problems in the right-hand column *do* have known polynomial-time algorithms on *quantum* computers.

3.2 What Qualifies as a "Negligible" Success Probability?

For a cryptographic system to be considered secure, we often want the success probability of any polynomial-time adversary to be negligible in the security parameter λ.

Idea. $\frac{1}{2^\lambda}$ **approaches zero so fast that no polynomial can "rescue".**

Proof. Assume that $f(\lambda) = \frac{1}{2^\lambda}$.



Consider any polynomial $p(\lambda)$ of degree n, written as:

$$p(\lambda) = a_0 + a_1\lambda + a_2\lambda^2 + \cdots + a_n\lambda^n = \sum_{i=0}^{n} a_i\lambda^i.$$

The product $p(\lambda)$ and $f(\lambda)$ is

$$p(\lambda)f(\lambda) = a_0\frac{1}{2^\lambda} + a_1\frac{\lambda}{2^\lambda} + \cdots + a_n\frac{\lambda^n}{2^\lambda}.$$

We claim that $\lim_{\lambda\to\infty} a_k\frac{\lambda^k}{2^\lambda} = 0$, where $k \in \mathbb{Z}_{\geq 0}$. Let $g(\lambda) = \lambda^k$ and $h(\lambda) = 2^\lambda$. Note that

$$\begin{aligned} h'(\lambda) &= 2^\lambda(\ln 2), \quad & g'(\lambda) &= k\lambda^{k-1} \\ h''(\lambda) &= 2^\lambda(\ln 2)^2, \quad & g''(\lambda) &= k(k-1)\lambda^{k-2} \\ &\vdots \\ h^{(k)}(\lambda) &= 2^\lambda(\ln 2)^k, \quad & g^{(k)}(\lambda) &= k!. \end{aligned}$$

By applying L'Hôpital's Rule k times, we have

$$\lim_{\lambda\to\infty}\frac{\lambda^k}{2^\lambda} = \lim_{\lambda\to\infty}\frac{k!}{2^\lambda(\ln 2)^k} = 0.$$

Thus, $\lim_{\lambda\to\infty} p(\lambda)f(\lambda) = 0$. □

Negligible

Definition 3.2. A function f is **negligible** if,

$$\forall \text{polynomial } p : \lim_{\lambda \to \infty} p(\lambda) f(\lambda) = 0.$$

In other words, a negligible function approaches zero so fast that you can never catch up when mutiplying by a polynomial.

Remark 3.2. As λ (security parameter) gets larger and larger, the product of $p(\lambda)$ (resources or capabilities for an adversary) and $f(\lambda)$ (success probability) approaches 0.

Remark 3.3. A function $f(\lambda)$ is negligible if $\forall p(\lambda) > 0 : \exists \lambda_0 : \lambda > \lambda_0 \Rightarrow \left| f(\lambda) \right| < \frac{1}{p(\lambda)}$.

Proposition 3.1. *Let $c \in \mathbb{Z}$.*

$$\lim_{\lambda \to \infty} \lambda^c f(\lambda) = 0 \implies f \text{ is negligible.}$$

Proof. Suppose that f satisfies $\lim\limits_{\lambda \to \infty} \lambda^c f(\lambda) = 0$ for any $c \in \mathbb{Z}$, and take an arbitrary polynomial p of degree n. Since $\lim\limits_{\lambda \to \infty} \frac{p(\lambda)}{\lambda^{n+1}} = 0$, we have

$$\lim_{\lambda \to \infty} p(\lambda) f(\lambda) = \lim_{\lambda \to \infty} \left[\frac{p(\lambda)}{\lambda^{n+1}} \left(\lambda^{n+1} \cdot f(\lambda) \right) \right] = \left(\lim_{\lambda \to \infty} \frac{p(\lambda)}{\lambda^{n+1}} \right) \left(\lim_{\lambda \to \infty} \lambda^{n+1} f(\lambda) \right) = 0 \cdot 0 = 0.$$

□

Example 3.3. Let $c \in \mathbb{Z}$. Then

$$\lim_{\lambda \to \infty} \lambda^c \frac{1}{2^\lambda} = \lim_{\lambda \to \infty} \frac{(\lambda^c)^{\log_2 2}}{2^\lambda} = \lim_{\lambda \to \infty} \frac{2^{c \log_2 \lambda}}{2^\lambda} = \lim_{\lambda \to \infty} 2^{c \log_2(\lambda) - \lambda} = 0$$

since $c \log_2(\lambda) - \lambda \to -\infty$ as $\lambda \to \infty$. Thus, $1/2^\lambda$ is negligible.

$f \approx g$

Definition 3.3. Let $f, g : \mathbb{N} \to \mathbb{R}$ are real-valued functions. We write $f \approx g$ to mean that $\left| f(\lambda) - g(\lambda) \right|$ is a negligible function.

Remark 3.4. We use the terminology of negligible functions exclusively when discussing probabilities, so the following are common:

$$\begin{aligned} \Pr[X] \approx 0 &\Leftrightarrow \text{"event } X \text{ almost never happens"} \\ \Pr[Y] \approx 1 &\Leftrightarrow \text{"event } Y \text{ almost always happens"} \\ \Pr[A] \approx \Pr[B] &\Leftrightarrow \text{"event } A \text{ and } B \text{ happen with essentially the same probability"} \end{aligned}$$

Additionally, the $\approx$ symbol is *transitive*:

$$\Pr[X] \approx \Pr[Y] \wedge \Pr[Y] \approx \Pr[Z] \implies \Pr[X] \approx \Pr[Z].$$

3.3 Indistinguishability

Indistinguishable (≈)

Definition 3.4. Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be two libraries with a common interface, and let $\mathcal{A}$ is a polynomial-time program that output a single bit. We say that $\mathcal{L}_1$ and $\mathcal{L}_2$ are **indistinguishable**, and write $\mathcal{L}_1 \approx \mathcal{L}_2$, if

$$\Pr[\mathcal{A} \diamond \mathcal{L}_1 \Rightarrow 1] \approx \Pr[\mathcal{A} \diamond \mathcal{L}_2 \Rightarrow 1].$$

Remark 3.5.

(1) We call the quantity

$$\left|\Pr[\mathcal{A} \diamond \mathcal{L}_1 \Rightarrow 1] - \Pr[\mathcal{A} \diamond \mathcal{L}_2 \Rightarrow 1]\right|$$

the **advantage** (or **bias**) of $\mathcal{A}$ in distinguishing $\mathcal{L}_1$ and $\mathcal{L}_2$.

(2) Two libraries are indistinguishable if all polynomial-time calling programs have negligible advantage in distinguishing them.

Example 3.4. Two indistinguishable libraries:

$\mathcal{L}_1$
$\underline{\texttt{Predict}(x)\text{:}}$
$s \xleftarrow{\$} \{\texttt{0}, \texttt{1}\}^\lambda$
return $x \stackrel{?}{=} s$

$\mathcal{L}_2$
$\underline{\texttt{Predict}(x)\text{:}}$
return `false`

The calling program $\mathcal{A}$ repeatedly invokes the '`Predict`' functions and returns '1' if it ever obtains a '`true`' value from the response:

```
A
do q times:
    if Predict(0^λ) = true
        return 1
return 0
```

Then

(1) $\mathcal{L}_2$ can never return `true`, i.e.,

$$\Pr[\mathcal{A} \diamond \mathcal{L}_2 \Rightarrow 1] = 0.$$

(2) $\Pr[\mathcal{A} \diamond \mathcal{L}_2 \Rightarrow 1]$ is surely non-zero.

$$\begin{aligned}\Pr[\mathcal{A}\diamond \Rightarrow 1] &= 1 - \Pr[\mathcal{A} \diamond \mathcal{L}_1 \Rightarrow 0] \\ &= 1 - \left(1 - \frac{1}{2^\lambda}\right)^q.\end{aligned}$$

Using the union bound, we get:

$$\begin{aligned}\Pr[\mathcal{A}\diamond \Rightarrow 1] &\leq \Pr[\text{first call to }\texttt{Predict}\text{ returns }\texttt{true}] \\ &\quad + \Pr[\text{second call to }\texttt{Predict}\text{ returns }\texttt{true}] \\ &\quad + \cdots \\ &= q \cdot \frac{1}{2^\lambda}.\end{aligned}$$

We showed that $\mathcal{A}$ has non-zero advantage, and so $\mathcal{L}_1 \not\equiv \mathcal{L}_2$. We also showed that $\mathcal{A}$ has advantage at most $q/2^\lambda$. Since $\mathcal{A}$ runs in polynomial time, it can only make a polynomial number q of queries to the library, so $q/2^\lambda$ is negligible.

Lemma 3.2.

(1) $\mathcal{L}_1 \equiv \mathcal{L}_2 \implies \mathcal{L}_1 \approx \mathcal{L}_2$.

(2) $\mathcal{L}_1 \approx \mathcal{L}_2 \approx \mathcal{L}_3 \implies \mathcal{L}_1 \approx \mathcal{L}_3$.

Proof. content... □

Lemma 3.3. *For any polynomial-time library $\mathcal{L}^*$,*

$$\mathcal{L}_1 \approx \mathcal{L}_2 \implies \mathcal{L}^* \diamond \mathcal{L}_1 \approx \mathcal{L}^* \diamond \mathcal{L}_2.$$

Proof. content... □

Bad-Event Lemma

Lemma 3.4.

$$\left|\Pr[\mathcal{A} \diamond \mathcal{L}_1 \Rightarrow 1] - \Pr[\mathcal{A} \diamond \mathcal{L}_2 \Rightarrow 1]\right| \leq \Pr[\mathcal{A} \diamond \mathcal{L}_1 \textit{ sets bad} = 1].$$

Proof. □

Example 3.5. Consider $\mathcal{L}_1$ and $\mathcal{L}_2$. They are indistinguishable with the following sequence of hybrids:

$\mathcal{L}_1$

Predict(x):
  $s \xleftarrow{\$} \{0, 1\}^\lambda$
  return $x \stackrel{?}{=} s$

$\equiv$

$\mathcal{L}_{\mathsf{hyb}-1}$

bad := 0

Predict(x):
  $s \xleftarrow{\$} \{0, 1\}^\lambda$
  if $x = s$:
    bad := 1
    return true
  return false

$\approx$

$\mathcal{L}_{\mathsf{hyb}-2}$

bad := 0

Predict(x):
  $s \xleftarrow{\$} \{0, 1\}^\lambda$
  if $x = s$:
    bad := 1

  return false

$\equiv$

$\mathcal{L}_2$

Predict(x):
  return false

- $\mathcal{L}_1 \equiv \mathcal{L}_{\mathsf{hyb}-1}$; Without *accessing* the variable "bad", the change can have no effect.
- $\mathcal{L}_{\mathsf{hyb}-1} \approx \mathcal{L}_{\mathsf{hyb}-2}$; By the bad-event lemma,

$$\left|\Pr\left[\mathcal{A} \diamond \mathcal{L}_{\mathsf{hyb}-1} \Rightarrow 1\right] - \Pr\left[\mathcal{A} \diamond \mathcal{L}_{\mathsf{hyb}-2} \Rightarrow 1\right]\right| \leq \Pr\left[\mathcal{A} \diamond \mathcal{L}_{\mathsf{hyb}-1} \text{ sets bad} = 1\right].$$

- $\mathcal{L}_{\mathsf{hyb}-2} \equiv \mathcal{L}_2$; Regardless of input, the subroutine always returns `false`.

Hence

$$\mathcal{L}_1 \equiv \mathcal{L}_{\mathsf{hyb}-1} \approx \mathcal{L}_{\mathsf{hyb}-2} \equiv \mathcal{L}_2 \implies \mathcal{L}_1 \approx \mathcal{L}_2.$$

3.4 Birthday Probabilities & Sampling With/out Replacement

Exercises

4.2. Which of the following are negligible functions in λ? Justify your answers.

$$\frac{1}{2^{\lambda/2}} \quad \frac{1}{2^{\log(\lambda^2)}} \quad \frac{1}{\lambda^{\log(\lambda)}} \quad \frac{1}{\lambda^2} \quad \frac{1}{2^{(\log\lambda)^2}} \quad \frac{1}{(\log\lambda)^2} \quad \frac{1}{\lambda^{1/\lambda}} \quad \frac{1}{\sqrt{\lambda}} \quad \frac{1}{2^{\sqrt{\lambda}}}$$

Solution.

(1) $\frac{1}{2^{\lambda/2}}, \frac{1}{2^{\log(\lambda^2)}}, \frac{1}{\lambda^{\log(\lambda)}}, \frac{1}{\lambda^2}, \frac{1}{2^{(\log\lambda)^2}}, \frac{1}{(\log\lambda)^2}, \frac{1}{\sqrt{\lambda}}, \frac{1}{2^{\sqrt{\lambda}}}$ are negligible functions.

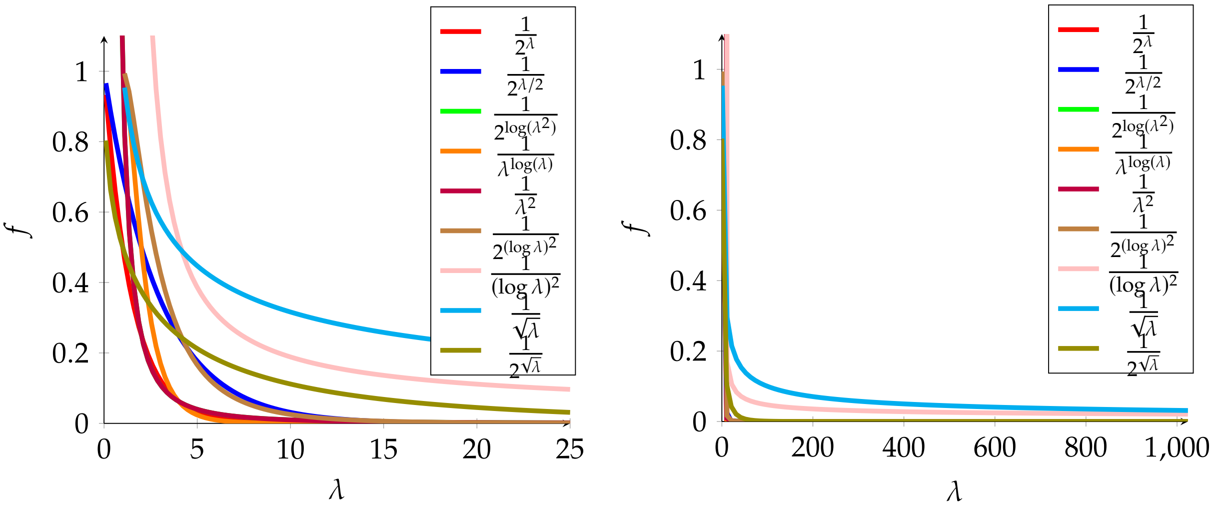


Figure 3.1: Negligible functions.

(2) $\frac{1}{\lambda^{1/\lambda}}$ is non-negligible functions.

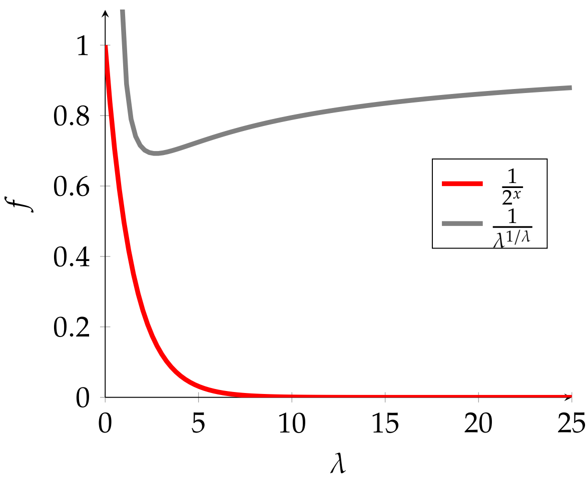


Figure 3.2: Non-negligible functions.

□

4.4. Show that when f is negligible, then for every polynomial p, the function $p(\lambda)f(\lambda)$ not only approaches 0, but it is also negligible itself.

Solution. We want to show that

$$p(\lambda)f(\lambda) \text{ is non-negligible} \implies f \text{ is non-negligible.}$$

Suppose that

$$\exists \text{polynomial } q(\lambda) : \lim_{\lambda\to\infty} q(\lambda)p(\lambda)f(\lambda) = c \neq 0.$$

Then p is non-zero polynomial and f is non-zero function, and so

$$\lim_{\lambda\to\infty} q(\lambda) = \frac{c}{\lim_{\lambda\to\infty} p(\lambda)f(\lambda)} = \frac{c}{\text{constant}}.$$

Thus $\lim_{\lambda\to\infty} p(\lambda)f(\lambda)$ cannot be a zero. □

4.8. A deterministic program is one that uses no random choices. Suppose $\mathcal{L}_1$ and $\mathcal{L}_2$ are two deterministic libraries with a common interface. Show that either $\mathcal{L}_1 \equiv \mathcal{L}_2$, or else $\mathcal{L}_1 \& \mathcal{L}_2$ can be distinguished with advantage 1.

Solution. Since both $\mathcal{L}_1$ and $\mathcal{L}_2$ are deterministic libraries, they will always produce the same output for the same input, i.e., either

$$\mathcal{L}_1(x) = \mathcal{L}_2(x) \quad \text{or} \quad \mathcal{L}_1(x) \neq \mathcal{L}_2(x)$$

for any input x.

(i) $(\mathcal{L}_1(x) = \mathcal{L}_2(x))$ Clearly,

$$(\forall \text{input } x : \mathcal{L}_1(x) = \mathcal{L}_2(x)) \implies (\mathcal{L}_1 \equiv \mathcal{L}_2).$$

(ii) $(\mathcal{L}_1(x) \neq \mathcal{L}_2(x))$ Suppose that

$$\exists \text{input } x : \mathcal{L}_1(x) \neq \mathcal{L}_2(x).$$

We construct a adversary $\mathcal{A}$ as follows:

(a) $\left|\Pr[\mathcal{A} \diamond \mathcal{L}_1 \Rightarrow 1] - \Pr[\mathcal{A} \diamond \mathcal{L}_2 \Rightarrow 1]\right| = |1 - 0| = 1.$

(b) $\left|\Pr[\mathcal{A} \diamond \mathcal{L}_1 \Rightarrow 1] - \Pr[\mathcal{A} \diamond \mathcal{L}_2 \Rightarrow 1]\right| = |0 - 1| = 1.$

□

4.12. Suppose you want to enforce password rules so that at least 2^{128} passwords satisfy the rules. How many characters long must the passwords be, in each of these cases?

(a) Passwords consist of lowercase `a` through `z` only.

(b) Passwords consist of lowercase and uppercase letters `a`-`z` and `A`-`Z`.

(c) Passwords consist of lower/uppercase letters and digits `0`-`9`.

(d) Passwords consist of lower/uppercase letters, digits, and any symbol characters that appear on a standard US keyboard (including the space character).

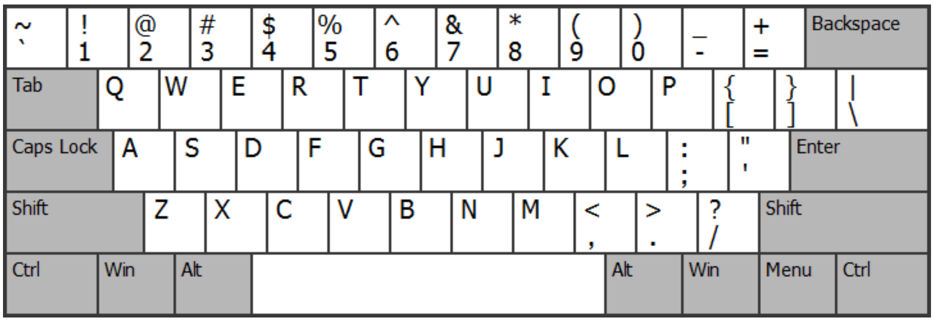


Figure 3.3: Standard US Keyboard (`https://kbd-intl.narod.ru/english/layouts`)

Solution. We want to create a password system that allows for at least 2^{128} (16 bytes) different passwords.

(a) We are only using lowercase letters `a`-`z`, which gives us **26** different possibilities for each character in the password. We need to solve the following equation for n (the length of the password):

$$26^n \geq 2^{128}.$$

Then

$$n \log(26) \geq 128 \log(2) \implies n \geq \frac{128 \cdot \log(2)}{\log(26)} \approx 27.2.$$

(b)

$$52^n \geq 2^{128} \implies n \geq \frac{128 \cdot \log(2)}{\log(52)} \approx 22.4.$$

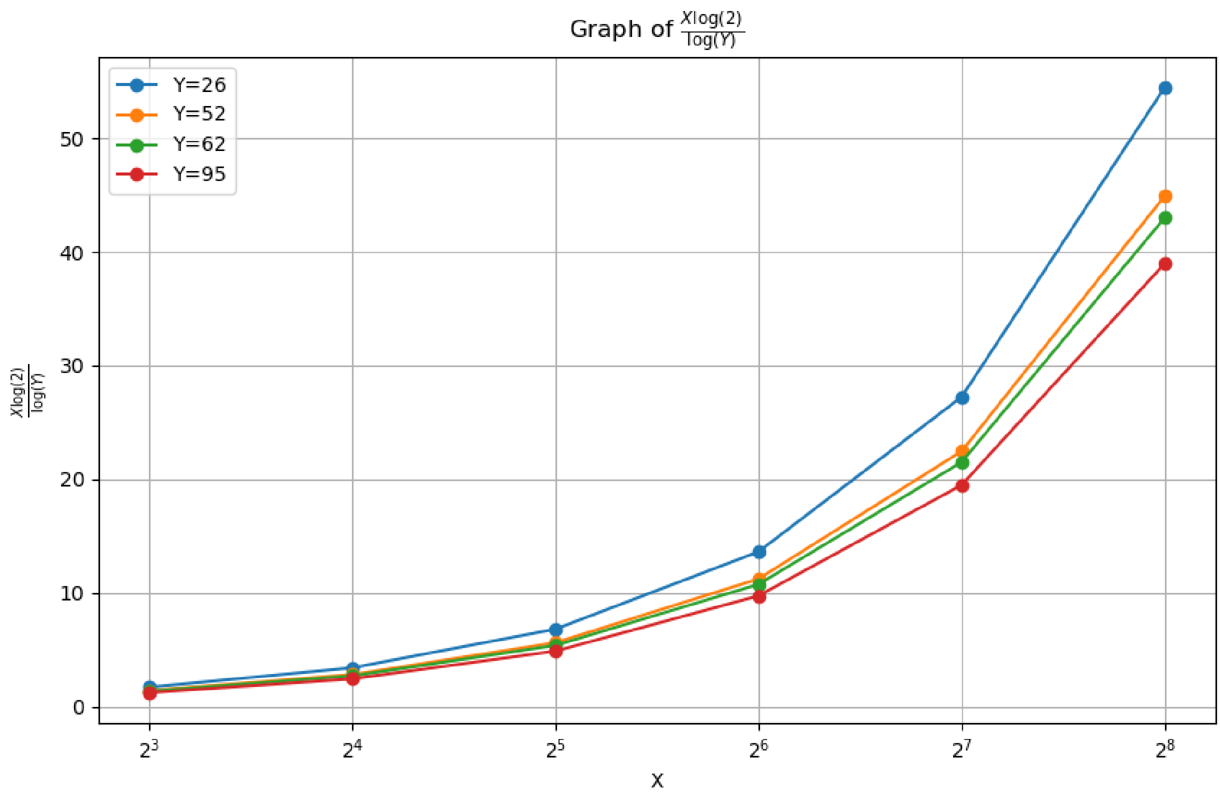
(c)

$$62^n \geq 2^{128} \implies n \geq \frac{128 \cdot \log(2)}{\log(62)} \approx 21.5$$

(d)

$$95^n \geq 2^{128} \implies n \geq \frac{128 \cdot \log(2)}{\log(95)} \approx 19.5$$

□




```
import matplotlib.pyplot as plt

# Given values of X and Y
X_values = [8, 16, 32, 64, 128, 256]
Y_values = [26, 52, 62, 95]

# Initialize a plot
plt.figure(figsize=(10,6))

# Loop through each Y value
for Y in Y_values:
    # Calculate the expression for each X value
    Z = [x * log(2) / log(Y) for x in X_values]

    # Plot the result
    plt.plot(X_values, Z, label='Y=' + str(Y), marker='o')

# Labeling the plot
plt.title(r'Graph of $\frac{X \log(2)}{\log(Y)}$')
plt.xlabel('X')
plt.ylabel(r'$\frac{X \log(2)}{\log(Y)}$')
plt.xscale("log", base=2) # for logarithmic scale on x-axis
plt.legend()
plt.grid(True)
plt.show()
```

Chapter 4

Pseudo-random Generators (PRG)

4.1 Definition

Pseudorandom Generator (PRG)

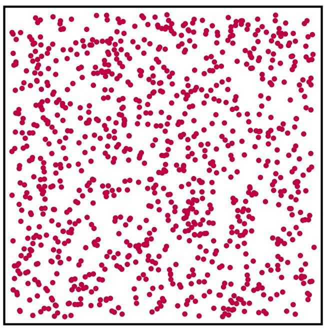
Definition 4.1. A deterministic function $G : \{0,1\}^\lambda \to \{0,1\}^{\lambda+l}$ with $l > 0$ is a **secure pseudorandom generator (PRG)** if $\mathcal{L}^G_{\textsf{PRG-real}} \approx \mathcal{L}^G_{\textsf{PRG-rand}}$, where:

$\mathcal{L}^G_{\textsf{PRG-real}}$
`Query()`:
$\quad s \leftarrow \{0,1\}^\lambda$
$\quad$ return $G(s)$

$\mathcal{L}^G_{\textsf{PRG-rand}}$
`Query()`:
$\quad r \leftarrow \{0,1\}^{\lambda+l}$
$\quad$ return r

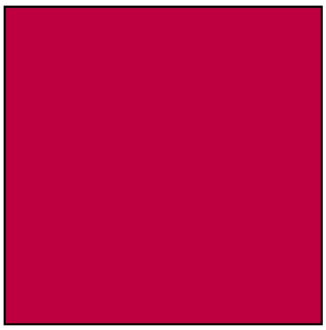
Remark 4.1. The value l is called the **stretch** of the PRG. The input to the PRG is typically called a **seed**.

Remark 4.2. We illustrate the distributions, for a **length doubling** ($l = \lambda$) PRG (not drawn to scale):



$\{0,1\}^\lambda \to \{0,1\}^{2\lambda}$

Pseudorandom dist.



$\{0,1\}^{2\lambda}$

Uniform dist.

Example 4.1 (Length-Doubling PRG)**.** A straightforward approach for the PRG might be to duplicate its input string.

$$\boxed{\begin{array}{l} \underline{G(s):} \\ \quad \text{return } s \,\|\, s \end{array}}$$

We can formalize this observation as an attack against the PRG-security of G:

$$\boxed{\begin{array}{c} \mathcal{A} \\ \hline x \,\|\, y := \texttt{Query}() \\ \text{return } x \stackrel{?}{=} y \end{array}}$$

$$\boxed{\begin{array}{c} \mathcal{A} \\ \hline x \,\|\, y := \texttt{Query}() \\ \text{return } x \stackrel{?}{=} y \end{array}} \diamond \boxed{\begin{array}{l} \quad \mathcal{L}^{G}_{\textsf{PRG-real}} \\ \hline \underline{\texttt{Query}():} \\ \quad s \leftarrow \{\texttt{0},\texttt{1}\}^{\lambda} \\ \quad \text{return } s \,\|\, s \end{array}}$$

$$\Pr[\mathcal{A} \diamond \mathcal{L}^{G}_{\textsf{PRG-real}} \Rightarrow 1] = 1.$$

$$\boxed{\begin{array}{c} \mathcal{A} \\ \hline x \,\|\, y := \texttt{Query}() \\ \text{return } x \stackrel{?}{=} y \end{array}} \diamond \boxed{\begin{array}{l} \quad \mathcal{L}^{G}_{\textsf{PRG-rand}} \\ \hline \underline{\texttt{Query}():} \\ \quad r \leftarrow \{\texttt{0},\texttt{1}\}^{\lambda+l} \\ \quad \text{return } r \end{array}}$$

$$\Pr[\mathcal{A} \diamond \mathcal{L}^{G}_{\textsf{PRG-rand}} \Rightarrow 1] = \frac{1}{2^{\lambda}}.$$

4.2 Shorter Keys in One-Time-Secret Encryption

One-time Pad (OTP)

Construction 4.1. The **one-time pad** are given below:

$\mathcal{K} = \{0,1\}^\lambda$ $\mathcal{M} = \{0,1\}^\lambda$ $\mathcal{C} = \{0,1\}^\lambda$	KeyGen : $k \xleftarrow{\$} \mathcal{K}$ return k	Enc($k, m \in \{0,1\}^\lambda$) : return $k \oplus m$	Dec($k, c \in \{0,1\}^\lambda$) : return $k \oplus c$

Pseudo-OTP

Construction 4.2. Let $G : \{0,1\}^\lambda \to \{0,1\}^{\lambda+l}$ be a PRG, and define the following:

$\mathcal{K} = \{0,1\}^\lambda$ $\mathcal{M} = \{0,1\}^{\lambda+l}$ $\mathcal{C} = \{0,1\}^{\lambda+l}$	KeyGen : $k \xleftarrow{\$} \mathcal{K}$ return k	Enc(k, m) : return $G(k) \oplus m$	Dec(k, c) : return $G(k) \oplus c$

Computational One-Time Secrecy

Definition 4.2. An encryption scheme Σ has **(computational) one-time secrecy** if $\mathcal{L}^\Sigma_{\text{ots-1}} \approx \mathcal{L}^\Sigma_{\text{ots-2}}$. That is, if for all polynomial-time distinguishers $\mathcal{A}$, we have

$$\Pr\left[\mathcal{A} \diamond \mathcal{L}^\Sigma_{\text{ots-1}} \Rightarrow 1\right] \approx \Pr\left[\mathcal{A} \diamond \mathcal{L}^\Sigma_{\text{ots-2}} \Rightarrow 1\right].$$

Remark 4.3. Σ has **one-time secrecy** if

$\mathcal{L}^\Sigma_{\text{ots-1}}$		$\mathcal{L}^\Sigma_{\text{ots-2}}$
Eve($m_1, m_2 \in \Sigma.\mathcal{M}$): $k \leftarrow \Sigma$.KeyGen $c \leftarrow \Sigma$.Enc(k, m_1) return c	$\equiv$	Eve($m_1, m_2 \in \Sigma.\mathcal{M}$): $k \leftarrow \Sigma$.KeyGen $c \leftarrow \Sigma$.Enc(k, m_2) return c

Theorem 4.1. *Let* pOTP *denote* **Construction 4.2**. *If one constructs the* pOTP *utilizing a secure pseudorandom generator* G, *then* pOTP *has computational one-time secrecy.*

Proof. We must show that

$$\mathcal{L}^{\mathsf{pOTP}}_{\mathsf{ots-1}} \approx \mathcal{L}^{\mathsf{pOTP}}_{\mathsf{ots-2}}.$$

We will show that a sequence of hybrid libraries satisfying the following:

$$\mathcal{L}^{\mathsf{pOTP}}_{\mathsf{ots-1}} \equiv \mathcal{L}_{\mathsf{hyp-1}} \approx \mathcal{L}_{\mathsf{hyp-2}} \equiv \mathcal{L}_{\mathsf{hyp-3}} \equiv \mathcal{L}_{\mathsf{hyp-4}} \equiv \mathcal{L}_{\mathsf{hyp-5}} \approx \mathcal{L}_{\mathsf{hyp-6}} \equiv \mathcal{L}^{\mathsf{pOTP}}_{\mathsf{ots-2}}.$$

$\mathcal{L}^{\mathsf{pOTP}}_{\mathsf{ots-1}}$:

$\mathcal{L}^{\mathsf{pOTP}}_{\mathsf{ots-1}}$
```
Eve(m1, m2 ∈ {0,1}^{λ+l}):
    k ← {0,1}^λ
    c := G(k) ⊕ m1
    return c
```

$\mathcal{L}_{\mathsf{hyp-1}}$:

```
Eve(m1, m2):
    z ← Query()
    c := z ⊕ m1
    return c
```
$\diamond$ $\mathcal{L}^{G}_{\mathsf{PRG-real}}$
```
Query():
    s ← {0,1}^λ
    return G(s)
```

$\mathcal{L}_{\mathsf{hyp-2}}$:

```
Eve(m1, m2):
    z ← Query()
    c := z ⊕ m1
    return c
```
$\diamond$ $\mathcal{L}^{G}_{\mathsf{PRG-rand}}$
```
Query():
    r ← {0,1}^{λ+l}
    return r
```

$\mathcal{L}_{\mathsf{hyp-3}}$:

$\mathcal{L}^{\mathsf{OTP}}_{\mathsf{ots-1}}$
```
Eve(m1, m2):
    z ← {0,1}^{λ+l}
    c : z ⊕ m1
    return c
```

$\mathcal{L}^{\mathsf{pOTP}}_{\mathsf{ots-2}}$:

$\mathcal{L}^{\mathsf{pOTP}}_{\mathsf{ots-2}}$
```
Eve(m1, m2 ∈ {0,1}^{λ+l}):
    k ← {0,1}^λ
    c := G(k) ⊕ m2
    return c
```

$\mathcal{L}_{\mathsf{hyp-6}}$:

```
Eve(m1, m2):
    z ← Query()
    c := z ⊕ m2
    return c
```
$\diamond$ $\mathcal{L}^{G}_{\mathsf{PRG-real}}$
```
Query():
    s ← {0,1}^λ
    return G(s)
```

$\mathcal{L}_{\mathsf{hyp-5}}$:

```
Eve(m1, m2):
    z ← Query()
    c := z ⊕ m2
    return c
```
$\diamond$ $\mathcal{L}^{G}_{\mathsf{PRG-rand}}$
```
Query():
    r ← {0,1}^{λ+l}
    return r
```

$\mathcal{L}_{\mathsf{hyp-4}}$:

$\mathcal{L}^{\mathsf{OTP}}_{\mathsf{ots-2}}$
```
Eve(m1, m2):
    z ← {0,1}^{λ+l}
    c := G(k) ⊕ m2
    return c
```

$\square$

Bibliography

[1] M. Rosulek, *The Joy of Cryptography*, [Online]. Available: https://joyofcryptography.com