IAI TCG CREST

Course: Advanced Cryptography ***Duration: Feb - June, 2023***

Instructor: Dr. Avijit Dutta ***Scribed By: Sajani, Prabal, Mriganka, Hrithik***

Outline: *Cryptology is concerned with the conceptualization, definition, and construction of computing systems that address security concerns. The design of cryptographic systems must be based on firm foundations. This course presents a rigorous and systematic treatment of the foundation issues: defining cryptographic tasks and solving new cryptographic problems using existing and new tools. The focus is given on the basic mathematical tools as well as some new advanced cryptographic tools and the advances of research using those tools.*

Pre-requisites: *Cryptology, Basic Probability and Number Theory.*

Contents

1 Introduction to Turing Machine and Complexity Classes

1.1 Turing Machine

Turing machine (TM) was introduced by Alan Turing as a model of "any possible computation". It consists of an infinite tape with cells, a finite control and a tape-head scanning the content of a cell at any given instant. Depending on the state of the finite control and the letter being scanned by the tape-head, the TM in one move can print a symbol in place of the scanned letter, move the tape-head one cell to the right or left and enters, perhaps, a new state. Initially, the input string w is placed in consecutive cells with all other cells occupied by the blank symbol B. The TM starts at an initial state scanning the first letter of the input string w. If after finitely many moves, the TM enters an accepting state then the TM accepts the input w.

Definition 1. *A Turing Machine $\mathcal{M}$ is a 7 tuple $(\Sigma, \Gamma, Q, q_0, \delta, B, F)$, where*

1. *$\Sigma \subseteq \Gamma$: is the **input alphabet** not containing B,*
2. *Γ: is the **tape alphabet**,*
3. *Q: is the finite set of **states**,*
4. *$q_0 \in Q$: is the **initial state**,*
5. *$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$, is the **transition function**,*
6. *$B \in \Gamma$: is the **blank symbol**,*
7. *$F \subseteq Q$: is the set of **accepting** or **final states**.*

Definition 2. *A string $w \in \Sigma^*$ is said to be accepted by the Turing Machine $\mathcal{M}$ if starting from the initial configuration, in finitely moves, $\mathcal{M}$ enters an accepting state.*

*The **Language accepted by** $\mathcal{M}$ is*

$$\mathcal{L}(\mathcal{M}) = \{w \in \Sigma^* : q_0 w \vdash^* I\}$$

where I is some accepting ID.

1.2 Time Complexity

Suppose $\mathcal{I}$ is an **instance** of a Problem $\mathcal{P}$ of size n and $\mathcal{T}(n)$ is the **time required** to solve the instance $\mathcal{I}$ by an algorithm $\mathcal{A}$, then $\mathcal{T}(n)$ is called the **time complexity** of that algorithm.

If $\mathcal{T}(n)$ is some polynomial of n then we say that the algorithm $\mathcal{A}$ solves the problem $\mathcal{P}$ in polynomial time.

Exercise 1. *Design an Turing Machine $\mathcal{M}$ that accepts all strings whose decimal equivalence is an even number, i.e.,*

$$\mathcal{L}(\mathcal{M}) = \{w \in \{0,1\}^* : w\%2 = 0\}$$

1.3 Complexity Classes

Definition 3 (Polynomial time computable function). *A function*

$$f : {\Sigma_1}^* \longrightarrow {\Sigma_2}^*$$

*is said to be **polynomial time computable** if there is polynomial $p(x)$ and a Turing Machine $\mathcal{M}$ with an output tape such that for every input w of length n, in at most $p(n)$ moves of $\mathcal{M}$, $f(w)$ is obtained on the output tape of $\mathcal{M}$.*

Definition 4 (Decider for a Language)**.** *Let $\mathcal{M}$ be a TM, we call $\mathcal{M}$ is a* ***decider for a language*** *$\mathcal{L}$ if $\mathcal{M}(x) = 1 \iff x \in \mathcal{L}$. We call $\mathcal{M}$ to be a poly-time decider if $\exists$ a positive polynomial $p(\cdot)$ such that $\forall x \in \{0,1\}^*$, $\mathcal{M}$ runs for at most $p(|x|)$ steps, where $|x|$ denotes the size of the i/p string x.*

A Turing Machine $\mathcal{M}$ is said to be **deterministic** if at any point of time there is at most **one move** of the turing machine $\mathcal{M}$. Depending on the **time complexity**, we can divide the languages in the following **complexity classes:**

Definition 5 (**P Class**)**.** *A language $\mathcal{L}$ is said to be in $\mathcal{P}$ class if there exists a deterministic poly-time decider for $\mathcal{L}$.*

Definition 6 (**NP Class**)**.** *A language $\mathcal{L}$ is said to be in $\mathcal{NP}$ class if there exists a two input deterministic poly-time turing machine $\mathcal{M}$ and there exists a polynomial $p(\cdot)$, such that $\forall x \in \mathcal{L}$, $\exists y \in \{0,1\}^*$, $|y| \leq |p(x)|$ and $\mathcal{M}(x, y) = 1$.*

Exercise 2. ***Boolean Satisfiability Problem****:*
Suppose $\phi(x_1, x_2, \cdots, x_n)$ is a boolean expression over n variables. We call that ϕ has a ***satisfying assignment*** *if $\phi(v_1, v_2, \cdots, v_n) = 1$, when $x_i = v_i$, $\forall i = 1(1)n$, $v_i \in \{0,1\}$. Show that, the following language*

$$\mathcal{L} = \{\phi(x_1, x_2, \cdots, x_n) : \phi \text{ has a satisfying assignment}\}$$

is in $\mathcal{NP}$.

Suppose $\mathcal{L}$ be a language. Let, $\mathcal{R}_{\mathcal{L}} \subseteq \{0,1\}^* \times \{0,1\}^*$ be a relation. We say that, $\mathcal{R}_{\mathcal{L}}$ is **poly-time decidable** if $\exists$ a two input deterministic poly-time TM $\mathcal{M}$ such that $\mathcal{M}(x, y) = 1 \iff (x, y) \in \mathcal{R}_{\mathcal{L}}$.

So, in another way we can say that a language $\mathcal{L}$ **is in $\mathcal{NP}$ class**, if $\exists$ **a poly-time decidable relation $\mathcal{R}_{\mathcal{L}}$ such that** $\forall x \in \mathcal{L}$**,** $\exists y$ **with** $|y| \leq p(|x|)$ **and** $(x, y) \in \mathcal{R}_{\mathcal{L}}$. Such a y is called a **witness for membership** of $x \in \mathcal{L}$.

Thus, $\mathcal{NP}$ consists of the set of languages for which there exist short proofs of membership that can be efficiently verified. It is widely believed that $\mathcal{P} \neq \mathcal{NP}$, and resolution of this issue is certainly the most intriguing open problem in computer science. If indeed $\mathcal{P} \neq \mathcal{NP}$, then there exists a language $\mathcal{L} \in \mathcal{NP}$ such that every algorithm recognizing $\mathcal{L}$ will have a super-polynomial running time in the worst case.

Definition 7 (**NP-Complete Class**)**.** *Let, $\mathcal{L}_1$ and $\mathcal{L}_2$ be two languages. $\mathcal{L}_1$ is said to be* ***poly-time reducible*** *to $\mathcal{L}_2$ and we write $\mathcal{L}_1 \leq_p \mathcal{L}_2$ if there is a poly-time computable function $f : \{0,1\}^* \longrightarrow \{0,1\}^*$ such that*

$$w \in \mathcal{L}_1 \iff f(w) \in \mathcal{L}_2$$

Definition 8. *A language $\mathcal{L}$ is said to be $\mathcal{NP}$-Complete if*

1. *$\mathcal{L} \in \mathcal{NP}$,*
2. *$\mathcal{L}' \leq_p \mathcal{L}$, $\forall \mathcal{L}' \in \mathcal{NP}$.*

If only condition (2) holds for any language $\mathcal{L}$ then it is said to be $\mathcal{NP}$-Hard.

Theorem 1. *If $\mathcal{L} \in \mathcal{P}$ and $\mathcal{L}' \leq_p \mathcal{L}$, then $\mathcal{L}' \in \mathcal{P}$.*

Proof. Since, $\mathcal{L}' \leq_p \mathcal{L}$, there is a $p(n)$ time computable function f such that $w \in \mathcal{L}' \leftrightarrow f(w) \in \mathcal{L}$. Let $\mathcal{M}_1$ be a $p(n)$ time bounded TM computing f and $\mathcal{M}_2$ be a $q(n)$ time bounded TM accepting $\mathcal{L}$. Now we shall construct a poly-time bounded TM $\mathcal{M}'$ that accepts $\mathcal{L}'$ as follows.

On input w of length n, $\mathcal{M}'$ first runs $\mathcal{M}_1$ on w to obtain $f(w)$ in at most $p(n)$ steps. $\mathcal{M}'$ then runs $\mathcal{M}_2$ on $f(w)$ and and accepts w iff $\mathcal{M}_2$ accepts $f(w)$. Clearly, $\mathcal{M}'$ accepts $\mathcal{L}'$. Now since

$|f(w)|$ is at most $p(n)$ so, $\mathcal{M}_2$ on $f(w)$ takes at most $q(p(n))$ steps on acceptance. Thus the total time taken by $\mathcal{M}'$ in accepting w is at most $p(n) + q(p(n))$. Hence, $\mathcal{M}'$ is a poly-time bounded TM accepting $\mathcal{L}'$ and so, $\mathcal{L}' \in \mathcal{P}$. □

Theorem 2. *If $\mathcal{L}$ is $\mathcal{NP}$-Complete and $\mathcal{L} \in \mathcal{P}$, then $\mathcal{P} = \mathcal{NP}$.*

Proof. It is obvious that $\mathcal{P} \subset \mathcal{NP}$. Let, $\tilde{\mathcal{L}} \in \mathcal{NP}$. Since $\mathcal{L}$ is $\mathcal{NP}$-Complete, so, $\tilde{\mathcal{L}} \leq_p \mathcal{L}$. Now since $\mathcal{L}$ is in $\mathcal{P}$ so, by Theorem 1 we get that $\tilde{\mathcal{L}} \in \mathcal{P}$. So, $\mathcal{NP} \subset \mathcal{P}$. Hence, $\mathcal{P} = \mathcal{NP}$. □

Theorem 3. *Suppose, $\mathcal{L} \in \mathcal{NP}$ and $\tilde{\mathcal{L}}$ is an $\mathcal{NP}$-Complete language such that $\tilde{\mathcal{L}} \leq_p \mathcal{L}$. Then $\mathcal{L}$ is also $\mathcal{NP}$-Complete.*

Proof. By Definition 8 it is enough to prove that the language $\mathcal{L}$ is $\mathcal{NP}$-Hard. Let, $\mathcal{L}' \in \mathcal{NP}$. Since $\tilde{\mathcal{L}}$ is $\mathcal{NP}$-Complete, so, $\mathcal{L}' \leq_p \tilde{\mathcal{L}}$. Given that, $\tilde{\mathcal{L}} \leq_p \mathcal{L}$. So, by **transitivity** property of '$\leq_p$' we have, $\mathcal{L}' \leq_p \mathcal{L}$. So, $\mathcal{L}$ is $\mathcal{NP}$-Hard. Hence, $\mathcal{L}$ is also $\mathcal{NP}$-Complete. □

Among the languages known to be $\mathcal{NP}$-Complete are *satisfiability* (of propositional formulae), *Graph Colorability*, *Graph Hamiltonicity* etc.

Definition 9 (BPP Class). *A language $\mathcal{L}$ is said to be in $\mathcal{BPP}$ class (Bounded Probabilistic Polynomial time) if $\exists$ a probabilistic polynomial time Turing Machine $\mathcal{M}$ such that the following two holds:*

1. *(**Completeness Error:**) $\forall x \in \mathcal{L}$, $Pr[\mathcal{M}(x) = 1] \geq \frac{2}{3}$,*
2. *(**Soundness Error:**) $\forall x \notin \mathcal{L}$, $Pr[\mathcal{M}(x) = 1] \leq \frac{1}{3}$.*

$\mathcal{BPP}$ is the class of languages that can be recognized by a probabilistic polynomial time Turing Machine (i.e., randomized algorithm). We would like to mention here that the constant 2/3 ot 1/3 in the above definition is not specific. The only thing that we care about that the soundness error should be negligibly small, which can be achieved by the following lemma, called *BPP Amplification Lemma.*

1.4 Amplification Lemma of $\mathcal{BPP}$

Let M be a prob. poly. time TM for a language $\mathcal{L} \in \mathcal{BPP}$ having error prob. is at least $\epsilon < \frac{1}{2}$. Then we can construct a prob. poly. time TM, M_1 for the same language $\mathcal{L}$, having error prob. $2^{-poly(\cdot)}$, for some polynomial $poly(\cdot)$. as follows:

- on i/p x, it does the following:
- Calculate a parameter k (which is to be determined later)
- Run M independently for $2k$ times with the same i/p x
- Record the o/p produced by M
- Let S be the output string of length $2k$
- M_1 will output *accept* if majority of the responses are *accept* else output *reject*

Our goal is to bound the error prob. of M_1. Let the error prob. of M on input x is $\epsilon_x < \epsilon$. Let S contains c correct responses and w wrong responses such that $c + w = 2k$. We call S **Bad** if $c \geq w$. For a given S, such that S contains c correct responses and w wrong responses,

$$Pr[S \text{ will appear}] = \epsilon_x^w (1 - \epsilon_x)^c, \text{ where } \epsilon_x < \epsilon < \tfrac{1}{2} \text{ and } c \leq w$$

Now,

$$\epsilon_x^w(1-\epsilon_x)^c = \epsilon_x^{w-c}\cdot\epsilon_x^c(1-\epsilon_x)^c = \epsilon_x^{w-c}(\epsilon_x(1-\epsilon_x))^c < \epsilon^{w-c}(\epsilon_x(1-\epsilon_x))^c$$

and, since

$$\epsilon_x < \epsilon < \tfrac{1}{2} \implies \epsilon_x+\epsilon < 1 \implies (\epsilon-\epsilon_x)\cdot(\epsilon+\epsilon_x) < (\epsilon-\epsilon_x) \implies \epsilon_x(1-\epsilon_x) < \epsilon(1-\epsilon)$$

we get,

$$\epsilon_x^w(1-\epsilon_x)^c < \epsilon^{w-c}(\epsilon(1-\epsilon)) = \epsilon^w(1-\epsilon)^c$$

Since, $c \leq w$ and $c + w = 2k$,

$$Pr[\text{S is Bad}] < \epsilon^w(1-\epsilon)^c < \epsilon^k(1-\epsilon)^k$$

Objective is to upper bound the error probability of M_1. Define $B = \{S : S \text{ is Bad}\}$.

$$\begin{aligned} Pr[M_1 \text{ fails}] = Pr[X \in B] &= \sum_{S\in B} Pr[X = S] \\ &\leq \sum_{S\in B} \epsilon^k(1-\epsilon)^k \leq 2^{2k}(\epsilon(1-\epsilon))^k = (4\epsilon(1-\epsilon))^k \end{aligned}$$

We want the failure prob. should be upper bounded by $2^{-poly(\cdot)}$. Since, $\epsilon < \frac{1}{2}$, therefore $4\epsilon(1-\epsilon) < \frac{1}{2}$. Hence, $(4\epsilon(1-\epsilon))^k < 2^{-poly(\cdot)}$.

We shall consider as intractable those tasks that cannot be performed by probabilistic polynomial-time machines. However, the adversarial tasks in which we shall be interested ("breaking an encryption scheme", "forging signatures", etc.) can be performed by non-deterministic polynomial-time machines (because the solutions, once found, can be easily tested for validity). Thus, the computational approach to cryptography is *interesting* only if $\mathcal{NP}$ is not contained in $\mathcal{BPP}$ (which certainly implies $\mathcal{P} \neq \mathcal{NP}$). But $\mathcal{NP}$ is not known to contain $\mathcal{BPP}$.

Cryptographic schemes that are guaranteed to be hard to break only in the worst case are useless. A cryptographic scheme must be unbreakable in "most cases" (i.e., "typical cases"), which implies that it will be hard to break on the average. It follows that because we are not able to prove that "worst-case intractability" implies analogous "intractability for the average case" (such a result would be considered a breakthrough in complexity theory), our intractability assumption must concern average-case complexity.

2 Introduction to One-Way Functions

In the previous class we have seen that the necessary condition for the existence of secure modern encryption scheme is that $\mathcal{NP}$ should not be contained in $\mathcal{BPP}$ class, thus $\mathcal{P} \neq \mathcal{NP}$. Although this is the necessary one but not sufficient condition. Suppose that the breaking of some encryption scheme is $\mathcal{NP}$-complete. Then, $\mathcal{P} \neq \mathcal{NP}$ implies that this encryption scheme is hard to break in the worst case, but it does not rule out the possibility that the encryption scheme is easy to break almost always. So we require

(i) the existence of languages which are computationally hard on average.

(ii) the existence of languages such that,

  (a) generating the instance along with the private informations should be poly-time computable.

  (b) the instance should be easy to solve when private informations are given.

(c) the instance should be computationally hard on the average when private informations are not given.

Definition 10 (Negligible Function). *A function $\mu : \mathbb{N} \to \mathbb{R}$ is called **negligible** in n if for every positive polynomial $p(\cdot)$ and all sufficiently large n's, it holds that*

$$\mu(n) < \frac{1}{p(n)}$$

*In other words, a function $\mu : \mathbb{N} \to \mathbb{R}$ is called **Non-negligible function** if $\exists$ a positive polynomial $p(\cdot)$ such that for sufficiently many n's, we have $\mu(n) \geq \frac{1}{p(n)}$.*

2.1 Definitions and Variants of One-Way Functions

In this section, we present definitions and different variants of one-way functions.

Definition 11 (Strong One-Way Functions). *A function $f : \{0,1\}^* \to \{0,1\}^*$ is called **strong one-way** if the following two conditions hold:*

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

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

   where randomness is taken over U_n.

Importance of Auxiliary input 1^n: The function f_{len} defined by $f_{len}(x) = y$ such that y is the binary representation of the length of x i.e. $f_{len}(x) = |x|$. Since $|f_{len}(x)| = log_2|x|$, no algorithm can invert f_{len} on y in time polynomial in $|y|$, because of shortage of printing time of the desired output; yet there exists an obvious algorithm that inverts f_{len} on $y = f_{len}(x)$ in time polynomial in $|x|$. The auxiliary input $1^{|x|}$, provided in conjunction with the input $f(x)$, allows the inverting algorithm to run in time polynomial in the total length of the main input and the desired output.

Trivial Algorithms to Break One-Wayness : It is required that the success probability, defined over the aforementioned probability space, be negligible. To further clarify the condition placed on the success probability, we consider two trivial algorithms to break one-wayness of a function.

Random-Guess Algorithm A: On input $(y, 1^n)$, algorithm A uniformly selects and outputs a string of length n. We note that the success probability of A equals the collision probability of the random variable $f(U_n)$. That is, letting U'_n denote a random variable uniformly distributed over $\{0,1\}^n$ independently of U_n, we have

$$Pr[A'(f(U_n), 1^n) \in f^{-1}(f(U_n))] = Pr[f(U'_n) = f(U_n)] = \textstyle\sum_y Pr[f(U_n) = y]^2 \geq 2^{-n}$$

where the inequality is due to the fact that, for non-negative x_i's summing to 1, the $\sum_i x_i^2$ is minimized when all x_i's are equal. So we can conclude that,

- For any one-way function the success probability is strictly positive.
- For any one-one one-way function the success probability is negligible.
- If f is one-way, then the collision probability of the random variable $f(U_n)$ is negligible.

Fixed-Output Algorithm A: Another trivial algorithm, is one that computes a function that is constant on all inputs of the same length, say 0^n. For every function f,

$$Pr[A'(f(U_n), 1^n) \in f^{-1}(f(U_n))] = Pr[f(0^n) = f(U_n)] = \frac{|f^{-1}(f(0^n)|}{2^n} \geq 2^{-n}$$

In this case also the success probability of A is non-negligible.

Definition 12 (**Weak One-way Function**). *A function $f : \{0,1\}^* \to \{0,1\}^*$ is called **Weakly One-Way** if the following two conditions hold:*

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

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

2.2 Useful Length Conventions

In this section we will describe the existence of weak one-way function and some notion of length preserving function. Then based on that we will prove the existence of strong one-way function.

Definition 13 (**Length-Regular Function**). *A function f is called **Length-Regular** if for every $x, y \in \{0,1\}^*$, if $|x| = |y|$, then $|f(x)| = |f(y)|$.*

Example 1. *Let f be a function defined over $\{0,1\}^*$ and $p(\cdot)$ be a polynomial s.t. $|f(x)| \leq p(|x|)$. The a function f' defined below is length-regular function,*

$$f'(x) := f(x)10^{p(|x|)-|f(x)|}$$

Definition 14. *(Length-Preserving Function)*
*A function f is called **Length-Preserving** if for every $x \in \{0,1\}^*$ it holds that $|f(x)| = |x|$.*

Example 2. *Let f' be a length-regular function and $p(\cdot)$ be a polynomial defined in the above example. We define f'' only on the strings of length $p(n)+1$, for $n \in (N)$, as follows*

$$f''(x'x'') := f'(x'), \text{ where } |x'x''| = p(|x'|) + 1$$

Clearly, f'' is a length-preserving function.

2.3 Weak One-Way Functions Imply Strong Ones

Theorem 4. *Let f be a weak one-way function that is length preserving. We define a function g as follows:*

$$g(p,x) = \begin{cases} (p, f(x)) & \text{if } p \text{ starts with } log_2|x| \text{ zeros} \\ (p, x) & \text{Otherwise} \end{cases}$$

where $|p(x)| = |f(x)|$. Therefore, g is a length-preserving weak one-way function but not a strong one-way function.

Proof. Suppose, g is not a weak one-way function. Then $\exists$ a prob. poly. time algorithm B such that for all positive polynomial $p(\cdot)$,

$$Pr[B(g(U_n)) \in g^{-1}(g(U_n))] \geq 1 - \frac{1}{p(n)} \tag{1}$$

for infinitely many n. Let N denote the set of n's where equation (1) holds. Let A be an algorithm that inverts f, described as follows:

Input to A: $y \in \{0,1\}^*$

- Let $n := |y|$, $l = log_2 n$ and assume that $2n \in N$.
- A will sample $U_{n-l} \xleftarrow{\$} \{0,1\}^{n-l}$
- A will invoke B with $(0^l||U_{n-l}, y)$
- B will return Z
- A will return n bit suffix of Z

Success prob. of A: Define $S_{2n} := \{0^l||\alpha : \alpha \in \{0,1\}^{2n-l}\}$. Then $Pr[U_{2n} \in S_{2n}] = n^{-1}$. Now, let us calculate the success prob. of A as follows:

$$\begin{aligned}
&Pr[A(f(U_n)) \in f^{-1}(f(U_n))] \\
&\geq Pr[B(0^l||U_{n-l}, f(U_n)) \in (0^l||U_{n-l}, f^{-1}(f(U_n)))] \\
&= Pr[B(g(U_{2n})) \in g^{-1}(g(U_{2n}))|U_{2n} \in S_{2n}] \\
&\geq \frac{Pr[B(g(U_{2n})) \in g^{-1}(g(U_{2n}))] - Pr[U_{2n} \notin S_{2n}]}{Pr[U_{2n} \in S_{2n}]} \\
&= n \cdot \left(Pr[B(g(U_{2n})) \in g^{-1}(g(U_{2n}))] - \left(1 - \frac{1}{n}\right)\right) \\
&= 1 - n \cdot (1 - Pr[B(g(U_{2n})) \in g^{-1}(g(U_{2n}))]) \\
&\geq 1 - n\left(1 - 1 + \frac{1}{p(2n)}\right) = 1 - \frac{n}{p(2n)} = 1 - \frac{1}{q(n)}
\end{aligned}$$

where $q(n) = \frac{p(2n)}{n}$. This contradicts our hypothesis that, f is one-way function. Hence, g is weakly one-way function.

Claim: *The above defined function is not strong one-way.*

Proof. Let A be a prob. poly. time algorithm that takes (y, z) as input and returns (y, z). Now, we calculate the success probability of A. To do this, we first define $S_{2n} := \{u||\alpha : u \in \{0,1\}^l \setminus 0^l, \alpha \in \{0,1\}^{2n-l}\}$. Then $Pr[U_{2n} \in S_{2n}] = 1 - \frac{1}{n}$.

$$\begin{aligned}
&Pr[A(g(U_{2n})) \in g^{-1}(g(U_{2n}))] \\
&\geq Pr[A(g(U_{2n})) \in g^{-1}(g(U_{2n}))|U_{2n} \in S_{2n}] \times Pr[U_{2n} \in S_{2n}] \\
&= 1 \times Pr[U_{2n} \in S_{2n}] \\
&= \left(1 - \frac{1}{n}\right) \geq \frac{1}{n},\ \forall n \geq 2
\end{aligned}$$

Hence, the above defined function is **Weak one-way but not Strong one-way function**.

□

2.4 Functions Defined Only for Some Lengths

Definition 15. *A function $f : \bigcup_{n \in I}\{0,1\}^n \to \{0,1\}^*$ is called a **One Way function** if*

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

$$Pr[A'(f(U_n), 1^n) \in f^{-1}(f(U_n))] \leq \frac{1}{p(n)}$$

Definition 16. *(Successor function) Let $I \subset \mathbb{N}$. We define a function $S_I : \mathbb{N} \to I$, called the* ***Successor function****, defined as follows*

$$S_I(n) := min\{i \in I : i > n\}$$

Definition 17. *I is called* ***polynomial time enumerable*** *if $\exists$ a det. poly. time TM M such that on input n, M returns $1^{S_I(n)}$.*

I is polynomial time enumerable $\implies \forall n$, $S_I(n)$ is polynomially bounded of n, i.e. $S_I(n) \leq poly(n)$.
If I is polynomial enumerable set, then deciding $x \in I$ or not can be done in polynomial time $(x \in I \leftrightarrow S_I(x-1) = x)$.

If we assume that existence of one way function defined over I, then we can provably construct an one way function which is defined for all bit strings of length $n \in \mathbb{N}$. Let f be a function defined over $\bigcup_{n\in I}\{0,1\}^n$, where I is a polynomial time enumerable set. Then $\exists$ a det. poly. time TM M such that on i/p n , it returns $1^{S_I(n)}$.
We define a function g over $\{0,1\}^n$, defined as follows:

$$g(x) = f(x') \text{ where } x' \text{ is the longest prefix of } x \text{ such that } |x'| \in I.$$

Theorem 5. *If f is a strong one way function, then g is a strong one way function.*

Proof. To show that g is easily computable, construct a deterministic TM M as follows,

- On input $x \in \{0,1\}^*$, check whether $S_I(x-1) = x$ or not.

Let B' be the prob. poly. time alg. for inverting g. Now we will construct a prob. poly. time alg. A' that will invert f. Input to A' is $f(U_n) = g(U_m)$, where $n \in I$ and $m \in \{n, n+1, \cdots, S_I(n)-1\}$. A' invokes B' with input $f(U_n) = g(U_m)$ and auxiliary input 1^m.

Algorithm for A':

On input y and 1^n, where y supposedly in the range of $f(U_n)$ and $n \in I$, it does the following:

- it computes $S_I(n)$ and set $k = S_I(n) - n - 1$(for every $i = 1, 2, \cdots, k$, we have $n + i \in I$).
- $i = 0, 1, \cdots, k$ A' invokes B' with i/p $(y, 1^{n+i})$.
- let $z_i \leftarrow B'(y, 1^{n+i})$. If $g(z_i) = y$, then it returns the longest n bit prefix of z_i and abort.

Correctness of the algorithm:

For all $x' \in \{0,1\}^n$, where $n \in I$ and for all x'' such that $|x''| \leq k$, where $k = S_I(n) - n - 1$ $g(x'x'') = f(x')$. If $g(x'x'') = y$, then x' is the input to f. In other words, $f(x') = y$.

Probability Analysis: Since, g is not strong one way so, $\exists$ a polynomial $p(\cdot)$ such that for infinitely many m, B' inverts $g(U_m)$ with prob. at least $\frac{1}{p(m)}$. We need to show the existence of a polynomial $q(\cdot)$ such that for infinitely many $n \in I$,

$$Pr[A(f(U_n), 1^n) \in f^{-1}(f(U_n))] \geq \frac{1}{q(n)}$$

For $m \in \mathbb{N}$, define $l_I(m) := \max\{i \in I : i \leq m\}$. So, $m \geq l_I(m)$ and $m \leq S_I(l_I(m)) - 1$.
Define $\mathcal{M} := \{m : Pr[B(g(U_m), 1^m) \in g^{-1}(g(U_m))] \geq \frac{1}{p(m)}\}$.

Claim 1: Let $m \in \mathbb{N}$ and $n = l_I(m)$. Then

$$Pr[A'(f(U_n), 1^n) \in f^{-1}(f(U_n))] \geq Pr[B'(g(U_m), 1^m) \in g^{-1}(g(U_m))]$$

Hints: Input to A' is $f(x')$ where $x' \in \{0,1\}^n$, $n \in I$ and it obtains $B'(f(x'), 1^t)$ for every $t \in \{n, \cdots, S_I(n) - 1\}$.

Claim 2: There exists a polynomial $q(\cdot)$ such that $m < q(l_I(m))$ for all m.

Hints: Since, I is a poly. time enumerable $\implies$ $\exists$ a polynomial $q(\cdot)$ such that $\forall n \in \mathbb{N}$, $S_I(n) \leq q(n)$. Then for every m we have,

$$m < S_I(l_I(m)) \leq q(l_I(m))$$

Let, $S := \{l_I(m) : m \in \mathcal{M}\}$ is infinite(as, otherwise, for u upper bounding the elements in S we get, $m < q(l_I(m)) \leq q(n)$ for every $m \in \mathcal{M}$, which contradicts the fact that $\mathcal{M}$ is infinite. Therefore, that for every $n = l_I(m) \in S$, the probability that A' inverts f on $f(U_n)$ is at least

$$\frac{1}{p(m)} > \frac{1}{p(q(l_I(m)))} = \frac{1}{p(q(n))} = \frac{1}{poly(n)}$$

where the inequality due to the claim 2. It follows that f is in strong one way , a contradiction. Hence, g is strong one way function. □

To make g length preserving define

$$g(x) := f(x')||x''$$

where $x = x'||x''$ and f is length preserving.

Remarks:

- If f is a weak/strong one way function defined over partial domain, then we can provably construct a weak/strong (resp.) one way function g over arbitrary length strings.
- Additionally, if f is a length preserving function, then g is a length preserving one.
- If f is a length preserving weak one way function, then we can provably construct a length preserving weak one way function that is not strong one way.
- Existence of Weak one way function implies Strong one way function.

Theorem 6. *Let f be a weak one way function which is length preserving. Then, the function g, defined as follows:*

$$g(x_1, x_2, \cdots, x_{t(n)}) := (f(x_1), f(x_2), \cdots, f(x_{t(n)})$$

where each $x_i \in \{0,1\}^n$ and $t(n) = n \cdot p(n)$ for some positive polynomial $p(\cdot)$, is a strong one way function.

Proof. See the proof from section 2.3.1 of [2]. □

We end this section with the following simple problem:

Theorem 7. *Suppose f is a one-way function. We define a function g as follows:*

$$g(x_1, x_2) \stackrel{def}{=} (f(x_1), x_2), \text{ where } |x_1| = |x_2|$$

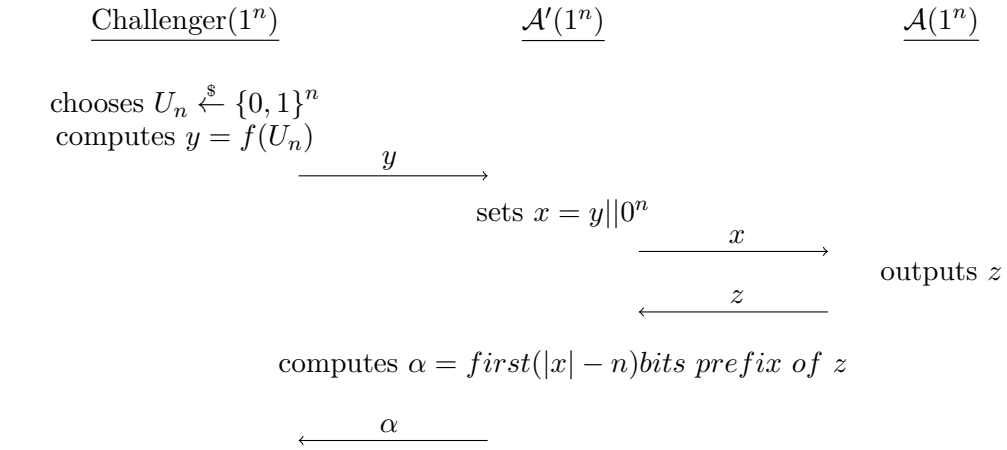
Prove that the function g is one-way.

Proof. Since f is a one-way function, so it is easy to compute for every x. Hence, g is also easy to compute for every x.
WLOG, let f be a strong one-way function and g be not a strong one-way function. So, $\exists$ a

probabilistic polynomial-time adversary $\mathcal{A}$ and a polynomial $p(\cdot)$ and infinitely many n such that

$$Pr[\mathcal{A}(g(U_{2n}), 1^n) \in g^{-1}(g(U_{2n}))] \geq \frac{1}{p(n)}$$

where U_{2n} denotes a random variable uniformly distributed over $\{0,1\}^{2n}$.
We will construct a probabilistic polynomial-time adversary $\mathcal{A}'$ who can invert f with probability at least $non - negl(n)$.



$$\begin{aligned}
&Pr[\mathcal{A}'(f(U_n), 1^n) \in f^{-1}(f(U_n))] \\
&\geq Pr[\mathcal{A}(f(U_n)||0^n, 1^n) \in g^{-1}(f(U_n)||0^n)] \\
&\geq \frac{1}{p(n)}
\end{aligned}$$

□

3 One-Way Functions as Collections

The formulation of one-way functions used till now is suitable for an abstract discussion. However, for describing many natural candidates for one-way functions, the following formulation is more appropriate. Instead of viewing one-way functions as functions operating on an infinite domain (i.e., $\{0,1\}^*$), we consider infinite collections of functions each operating on a finite domain. The functions in the collection share a single evaluating algorithm that when given as input a succinct representation of a function and an element in its domain returns the value of the specified function at the given point.

Definition 18 (Collection of Functions)**.** *A **collection of functions** consists of an infinite set of **indices**, denoted $\bar{I}$, and a corresponding set of finite functions, denoted $\{f_i\}_{i \in \bar{I}}$. That is, for each $i \in \bar{I}$, the domain of the function f_i, denoted D_i, is a finite set.*

We shall be interested only in collections of functions that can be used in cryptographic applications. As hinted earlier, a necessary condition for using a collection of functions is the existence of an efficient function-evaluating algorithm (denoted F) that on input $i \in \bar{I}$ and $x \in D_i$ returns $f_i(x)$.

Definition 19 (Collection of One-Way Functions)**.** *A collection of functions $\{f_i : D_i \to \{0,1\}^*\}_{i \in \bar{I}}$ is called strongly (resp., weakly) **one-way** if there exist three probabilistic poly-time algorithms I, D, and F such that the following two conditions hold:*

1. *Easy to sample and compute:*

   (i) The output distribution of algorithm I on input 1^n is a random variable assigned values in the set $\bar{I} \cap \{0,1\}^n$.

   (ii) The output distribution of the algorithm D on input $i \in \bar{I}$ is a random variable assigned values in D_i.

   (iii) On input $i \in \bar{I}$ and $x \in D_i$, algorithm F always outputs $f_i(x)$.

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

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

   where I_n is a random variable describing the output distribution of the algorithm I on input 1^n, and X_n is a random variable describing the output distribution of algorithm D on input (random variable) I_n.
   (The version for weakly one-way collections is analogous).

We can describe a collection of one-way functions by indicating the corresponding triplet of algorithms. Hence, we can say that a triplet of probabilistic polynomial-time algorithms (I, D, F) constitutes a collection of one-way functions if there exists a collection of functions for which these algorithms satisfy the foregoing two conditions.

- $I \Rightarrow$ Index Sampling Algorithm ($I(1^n) \longrightarrow I_n \in \bar{I_n} \cap \{0,1\}^n$)(Prob. poly-time)
- $D \Rightarrow$ Domain Sampling Algorithm ($D(I_n) \longrightarrow X_{I_n} \in D_{I_n}$)(Prob. poly-time)
- $F \Rightarrow$ Function Evaluating Algorithm ($F(i \in \bar{I_n} \cap \{0,1\}^n, x \in D_i) \longrightarrow f_i(x)$)(Det. poly-time)

3.1 Examples of One-Way Collections

Now we are describing some practical candidates of collection of one-way functions:

- **The RSA Function**: The RSA function is defined by the triplet of algorithms (I_{RSA}, D_{RSA}, F_{RSA}).

  On input 1^n, algorithm I_{RSA} selects uniformly two primes, P and Q, such that $2^{n-1} \leq P < Q < 2^n$, and an integer e such that e is relatively prime to $(P-1)\cdot(Q-1)$. Algorithm I_{RSA} terminates with output (N, e), where $N = P \cdot Q$.

  As for algorithm D_{RSA}, on input (N, e) it selects (almost) uniformly an element x in the set $D_{N,e} = \{1, \cdots, N\}$.

  The output of F_{RSA}, on input $((N, e), x)$, is

$$RSA_{N,e}(x) = x^e \bmod N$$

  It is not known whether or not factoring N can be reduced to inverting $RSA_{N,e}$, and in fact this is a well-known open problem. We remark that the best algorithms known for inverting $RSA_{N,e}$ proceed by (explicitly or implicitly) factoring N. In any case, it is widely believed that the RSA collection is hard to invert.

- **Discrete Logarithms**: Extracting discrete logarithms in finite field is believed to be **computational hard problem** in number theory. The DLP collection of functions, which borrows its name (and its conjectured one-wayness) from the *discrete logarithm problem*, is defined by the triplet of algorithms (I_{DLP}, D_{DLP}, F_{DLP}).

  On input 1^n, algorithm I_{DLP} selects uniformly a prime p, such that $2^{n-1} \leq p < 2^n$, and a primitive element g in the multiplicative group modulo p and outputs (p, g).

Regarding algorithm D_{DLP}, on input (p, g) it selects uniformly a residue modulo $p-1$ say, x.

Algorithm F_{DLP}, on input $((p, g), x)$, outputs

$$DLP_{p,g}(x) = g^x \bmod p$$

Hence, inverting $DLP_{p,g}$ means to extracting the discrete logarithm (to base g) modulo p.

3.2 Trapdoor One-Way Permutations

The formulation of collections of one-way functions is convenient as a starting point to the definition of trapdoor permutations. These are collections of one-way permutations, $\{f_i\}$, with the extra property that f_i is efficiently inverted once it is given as auxiliary input a "trapdoor" for the index i. The trapdoor for index i, denoted $t(i)$ can not be efficiently computed from i, yet one can efficiently generate corresponding pairs $(i, t(i))$.

Definition 20 (Collection of Trapdoor Permutations)**.** *Let $I : 1^* \longrightarrow \{0,1\}^* \times \{0,1\}^*$ be a probabilistic algorithm, and let $I_1(1^n)$ denote the first element of the pair output by $I(1^n)$. A triple of algorithms (I, D, F) is called a* ***collection of strong (resp. weak) trapdoor permutations*** *if the following two conditions hold:*

1. *The algorithms induce a collection of one-way permutations: The triple (I_1, D, F) constitutes a collection of strong (resp, weak) one-way permutations.*
2. *Easy to invert with trapdoor: There exists a deterministic polynomial time algorithm, denoted F^{-1}, such that for every (i, t) in the range of I and for every $x \in D_i$, it holds that $F^{-1}(t, f_i(x)) = x$.*

Example 3 (The RSA Trapdoor)**.** *The RSA collection presented earlier can be easily modified to have the trapdoor property. Algorithm I_{RSA} should be modified so that it outputs both the index (N, e) and the trapdoor (N, d), where d is the multiplicative inverse of e modulo $(p-1)\cdot(q-1)$. The inverting algorithm F_{RSA}^{-1} is identical to the algorithm F_{RSA}, i.e., $F_{RSA}^{-1}((N, d), y) = y^d \ mod \ N$.*

Now, we can easily verify that,

$$F_{RSA}^{-1}((N, d), F_{RSA}((N, e), x)) = F_{RSA}^{-1}((N, d), x^e \ mod \ N) = (x^e)^d \ mod \ N = x \ mod \ N$$

for every x in the multiplicative group modulo N (one can show that the last equality holds for every x, even in case x is not relatively prime to N).

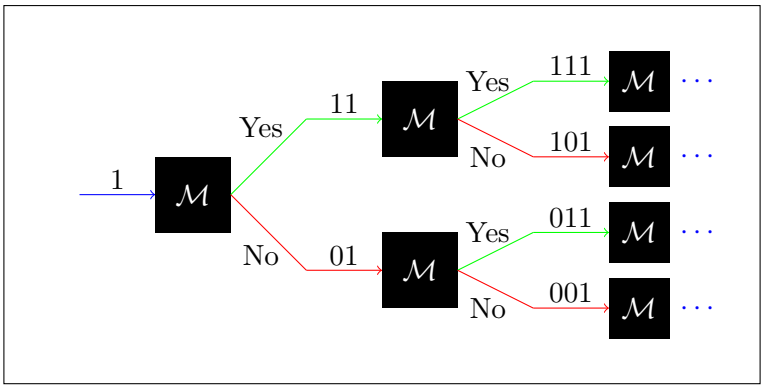
We end this lecture with a very important result of complexity theory and cryptography, which informally says that existence of one way function implies $P \neq NP$. We prove the statement below.

Theorem 8. *If $\mathcal{P} = \mathcal{NP}$, then no one-way function exists.*

Proof. Suppose, $\mathcal{P} = \mathcal{NP}$ and there exists a one-way function say, f. Now we are given 1^n and $y = f(x)$ and we are trying to recover x.
Consider the following language

$$\mathcal{L}_f \stackrel{\text{def}}{=} \{x' : x' \text{ is a prefix of some } x \text{ for which } f(x) = y\}$$

It can be easily shown that the language $\mathcal{L}_f \in \mathcal{NP}$, because x itself is a witness and we can verify it in only polynomial time (since one-way function can be computed in poly-time).
Since $\mathcal{P} = \mathcal{NP}$, so, $\mathcal{L}_f \in \mathcal{P}$ also. Therefore there exists a deterministic poly-time decider for $\mathcal{L}_f$, say $\mathcal{M}$.



Now we uniformly sample a bit b from $\{0,1\}$ and runs $\mathcal{M}$ with input b. So, $\mathcal{M}$ returns either "Yes" or "No". When it returns "Yes" we runs $\mathcal{M}$ with input bb and otherwise we runs $\mathcal{M}$ with input $\bar{b}b$. We runs $\mathcal{M}$, n times in this way increasing input string length by 1 bit. After the n^{th} run if we get "Yes" then the last n bit input string is our required x, otherwise if we get "No", then just flip the last bit of the n bit last input string and that is our required x. So, on input 1^n and $f(x)$ we get back x in polynomial time. Therefore f is not one-way. Hence, no one-way function exists. □

4 Hard-Core Predicates

A function f is one-way implies that given y, it is infeasible to find a pre-image of y under f. This does not mean that it is infeasible to find some partial information about the pre-image of y under f. Specifically, it may be easy to retrieve half of the bits of the pre-image, for example look at Theorem 7 the function g, defined by $g(x_1, x_2) = (f(x_1), x_2)$, where $|x_1| = |x_2|$, is one-way but $g(x_1, x_2)$ reveals half of the bits of the pre-image. The fact that one-way functions do not necessarily hide partial information about their pre-images limits their "direct applicability" to tasks such as secure encryption. Fortunately, assuming the existence of one-way functions, it is possible to construct one-way functions that hide specific partial information about their pre-images (which is easy to compute from the pre-image itself). This partial information can be considered as a "hard-core" of the difficulty of inverting f.

Definition 21 (Hard-Core Predicate). *A polynomial time computable predicate $b : \{0,1\}^* \longrightarrow \{0,1\}$ is called a **hard-core** of a function f if for every probabilistic poly-time algorithm A', every positive polynomial $p(\cdot)$, and all sufficiently large n's,*

$$Pr[A'(f(U_n)) = b(U_n)] < \tfrac{1}{2} + \tfrac{1}{p(n)}$$

Since b itself is polynomial time computable, the failure of efficient algorithms to approximate $b(x)$ from $f(x)$ must be due either to an information loss of f or to the difficulty of inverting f. For example, consider the function g defined as follows:

$$g(\sigma\alpha) \stackrel{\text{def}}{=} \alpha, \text{ where } \sigma \in \{0,1\}, \alpha \in \{0,1\}^*$$

and the predicate $hc(\sigma\alpha) = \sigma$ is a hard-core of the function g. Hence, in this case the fact that hc is a hard-core of the function g is due to the fact that g losses information (specifically, the first bit σ). This types of hard-core predicates are called "**useless hard-core predicate**". We deal with hard-core predicates **that arise due to the computational difficulty of inverting function**.

Proposition 9. *There does not exist any **universal** hard-core predicate.*

Proof. Suppose a predicate $hc : \{0,1\}^* \longrightarrow \{0,1\}$ be hard-core for any one-way function. Let f be a one-way function and we define the function g as follows:

$$g(x) \stackrel{\text{def}}{=} (f(x), hc(x))$$

One can easily prove that g is one-way, but hc is not a hard-core predicate of the one-way function g. □

Remark 1. *For every one-way function, there exists a predicate which is hard-core for that function (Actually it remains open whether this is true).*

4.1 Hard-Core Predicates From One-Way Functions

Theorem 10 (**Goldreich-Levin Theorem**). *Assume one-way functions (resp., permutations) exist. Then there exists a one-way function (resp., permutation) g and a hard-core predicate hc of g.*
Let f be a one-way function. Functions g and hc are constructed as follows: $g(x,r) \stackrel{def}{=} (f(x), r)$, for $|x| = |r|$, and define

$$hc(x,r) \stackrel{def}{=} \bigoplus_{i=1}^{n} x_i \cdot r_i$$

where x_i (resp. r_i) denotes the i^{th} bit of x (resp. r).

Theorem 11. *Let f be a one-way function and define g by $g(x,r) \stackrel{def}{=} (f(x), r)$, where $|x| = |r|$. Define $hc(x,r) \stackrel{def}{=} \bigoplus_{i=1}^{n} x_i \cdot r_i$, where $x = x_1 \cdots x_n$ and $r = r_1 \cdots r_n$. Then hc is a hard-core predicate of g.*

Due to the complexity of the proof, we prove three successively stronger results culminating what is claimed in the theorem.

Proposition 12. *Let f, g, and hc be as in Theorem 8. If there exists a polynomial time algorithm $\mathcal{A}$ such that $\mathcal{A}(f(x), r) = hc(x,r)$ for all n and all $x, r \in \{0,1\}^n$, then there exists a poly-time algorithm $\mathcal{A}'$ such that $\mathcal{A}'(1^n, f(x)) = x$ for all n and all $x \in \{0,1\}^n$.*

Proof. We construct $\mathcal{A}'$ as follows. On input 1^n and y $(= f(x))$, $\mathcal{A}'$ invokes the algorithm $\mathcal{A}$ with input (y, e^i) for $i = 1, \cdots, n$, where e^i denotes the n bit string with 1 in the i^{th} position and 0 everywhere else. Then $\mathcal{A}'$ outputs $x = x_1 \cdots x_n$. Clearly $\mathcal{A}'$ runs in polynomial time.
In the execution of $\mathcal{A}'(1^n, f(\hat{x}))$, the value x_i computed by $\mathcal{A}'$ satisfies

$$x_i = \mathcal{A}(f(\hat{x}), e^i) = hc(\hat{x}, e^i) = \bigoplus_{j=1}^{n} \hat{x}_j \cdot e^i_j = \hat{x}_i.$$

Thus $x_i = \hat{x}_i$, for all i and so $\mathcal{A}'$ outputs the correct inverse $x = \hat{x}$. Since f is one-way, so we can conclude that there is no poly-time algorithm that always correctly computes $hc(x,r)$ from $(f(x), r)$. □

Proposition 13. *Let f, g, and hc be as in Theorem 8. If there exists a probabilistic polynomial time algorithm $\mathcal{A}$ and a polynomial $p(\cdot)$ such that*

$$\Pr_{x,r \leftarrow \{0,1\}^n}[\mathcal{A}(f(x), r) = hc(x,r)] \geq \frac{3}{4} + \frac{1}{p(n)}$$

for infinitely many values of n, then there exists a probabilistic poly-time algorithm $\mathcal{A}'$ such that

$$\Pr_{x \leftarrow \{0,1\}^n}[\mathcal{A}'(1^n, f(x)) \in f^{-1}(f(x))] \geq \frac{1}{4 \cdot p(n)}$$

for infinitely many values of n.

Proof. Algorithm $\mathcal{A}'$ on input 1^n and y works as follows:

1. For $i = 1, \ldots, n$ do:
   - Repeatedly choose a uniform $r \in \{0,1\}^n$ and compute $\mathcal{A}(y, r) \oplus \mathcal{A}(y, r \oplus e^i)$ as an "estimate" for the i^{th} bit of the pre-image of y. After doing this sufficiently many times, let x_i be the "estimate" that occurs a majority of the time.
2. Output $x = x_1 \cdots x_n$.

Prepare the rest of the proof from section 7.3.2 of [4]. □

Proposition 14. *Let f, g, and hc be as in Theorem 8. If there exists a probabilistic polynomial time algorithm $\mathcal{A}$ and a polynomial $p(\cdot)$ such that*

$$\Pr_{x,r \leftarrow \{0,1\}^n}[\mathcal{A}(f(x), r) = hc(x, r)] \geq \frac{1}{2} + \frac{1}{p(n)}$$

for infinitely many values of n, then there exists a probabilistic poly-time algorithm $\mathcal{A}'$ and a polynomial $p'(\cdot)$ such that

$$\Pr_{x \leftarrow \{0,1\}^n}[\mathcal{A}'(1^n, f(x)) \in f^{-1}(f(x))] \geq \frac{1}{p'(n)}$$

for infinitely many values of n.

Proof. **The inversion algorithm $\mathcal{A}'$.** We now provide a full description of an algorithm $\mathcal{A}'$ that receives input $1^n, y$ and tries to compute an inverse of y. The algorithm proceeds as follows:

1. Set $l := \left\lceil log\left(\frac{2n}{\epsilon(n)^2+1}\right)\right\rceil$.
2. Choose uniform, independent $s^1, \ldots, s^l \in \{0,1\}^n$ and $\sigma^1, \ldots, \sigma^l \in \{0,1\}$.
3. For every non-empty subset $I \subseteq \{1, \ldots, l\}$, compute $r^I := \oplus_{i \in I} s^i$ and $\sigma^I := \oplus_{i \in I} \sigma^i$.
4. For $i = 1, \ldots, n$ do:
   (a) For every non-empty subset $I \subseteq \{1, \ldots, l\}$, set ${x_i}^I := \sigma^I \oplus \mathcal{A}(y, r^I \oplus e^i)$.
   (b) Set $x_i := \textbf{majority}_I\{{x_i}^I\}$ (i.e., take the bit that appeared a majority of the time in the previous step).
5. Output $x = x_1 \cdots x_n$.

Now, you can build up rest of the proof from section 7.3.3 of [4]. □

5 Introduction to Pseudorandom Generators

In the previous classes we have discussed about one way function and it's hard core predicate. Now, using that we will introduce another concept, called Pseudorandom Generator. The theory of pseudorandomness is also applied to functions, resulting in the notion of pseudorandom functions, which is a useful tool for many cryptographic applications. This will be a building block for many cryptographic primitive which we will discuss in successive lectures.

Definition 22. ***(Probability Ensemble)*** *Let I be a countable infinite index set. An **Ensemble** over the index set I is a family of random variables $\{X_i\}_{i \in I}$.*

Definition 23. *Suppose, $\{X_n\}_{n \in \mathbb{N}}$ and $\{Y_n\}_{n \in \mathbb{N}}$ are two ensembles. We say that $\{X_n\}_{n \in \mathbb{N}}$ is **Computationally Indistinguishable from Ensemble** $\{Y_n\}_{n \in \mathbb{N}}$, if for every prob. poly time algorithm D, for every positive polynomial $p(\cdot)$ and for all sufficiently large n, it holds that*

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

Definition 24. *(Pseudorandom Ensemble) The ensemble $\{Y_n\}_{n\in\mathbb{N}}$ is called **pseudorandom** if there exists a uniform ensemble $\{U_{l(n)}\}_{n\in\mathbb{N}}$ such that one is computationally indistinguishable form another.*

Here, we stress that $l : \mathbb{N} \to \mathbb{N}$ is a poly time computable function.

Definition 25. *(Pseudorandom Generator) A **Pseudorandom generator** is a deterministic poly time algorithm G satisfying the following two conditions:*

1. ***Expansion:** There exists a function $l : \mathbb{N} \to \mathbb{N}$ such that $l(n) > n$ for all $n \in \mathbb{N}$, and $|G(s)| = l(|s|)$ for all $s \in \{0,1\}^*$.*
2. ***Pseudorandomness:** The ensemble $\{G(U_n)\}_{n\in\mathbb{N}}$ is pseudorandom.*

*The function l is called **length expansion factor** of G.*

5.1 Construction of PRG From One-Way Permutation

Let f be a one way permutation and h_c be its hard core predicate. Then G be a PRG and defined as follows:
$G : \bigcup_{n\geq 0}\{0,1\}^n \to \bigcup_{n\geq 0}\{0,1\}^{l(n)}$ such that $G(s) := (f(s), h_c(s))$, where $|s| = n$, $l(n) = n + 1$

Proof Idea: Let $X = \{X_n\}_{n\in\mathbb{N}}$ where for every $n \in \mathbb{N}$ $X_n = G(s)$ such that $s \xleftarrow{\$} \{0,1\}^n$.
Our goal is to show that X is computationally indistinguishable from $\mathcal{U} = \{U_{n+1}\}_{n\in\mathbb{N}}$, where $U_{n+1} \xleftarrow{\$} \{0,1\}^{n+1}$. Let D be a prob. poly time algorithm and $\exists$ a positive polynomial $p(\cdot)$ such that

$$\Big| \Pr_{s\xleftarrow{\$}\{0,1\}^n}[D(G(s)) = 1] - \Pr_{U_{n+1}\xleftarrow{\$}\{0,1\}^{n+1}}[D(U_{n+1}) = 1]\Big| > \frac{1}{p(n)} \tag{2}$$

holds for infinitely many n.
Our claim is D can also distinguish $(f(s), h_c(s))$ from $(f(s), \bar{h}_c(s))$ i.e.

$$\frac{1}{2} \cdot \Big| \Pr_{s\xleftarrow{\$}\{0,1\}^n}[D(f(s), h_c(s) = 1] - \Pr_{s\xleftarrow{\$}\{0,1\}^n}[D(f(s), \bar{h}_c(s) = 1]\Big| > \frac{1}{p(n)} \tag{3}$$

Since,

$$\Pr_{s\xleftarrow{\$}\{0,1\}^n,\beta\xleftarrow{\$}\{0,1\}}[D(f(s), \beta) = 1] = \frac{1}{2} \cdot Pr[D(f(s), h_c(s) = 1] + \frac{1}{2} \cdot Pr[D(f(s), \bar{h}_c(s) = 1] \tag{4}$$

and,

$$\Pr_{s\xleftarrow{\$}\{0,1\}^n,\beta\xleftarrow{\$}\{0,1\}}[D(f(s), \beta) = 1] = \Pr_{U_{n+1}\xleftarrow{\$}\{0,1\}^{n+1}}[D(U_{n+1}) = 1] \tag{5}$$

So,

$$\begin{aligned}
&\frac{1}{2}\cdot\Big|\Pr_{s\xleftarrow{\$}\{0,1\}^n}[D(f(s),h_c(s)=1]-\Pr_{s\xleftarrow{\$}\{0,1\}^n}[D(f(s),\bar{h}_c(s)=1]\Big| \\
&\qquad=\Big|\frac{1}{2}\cdot\Pr_{s\xleftarrow{\$}\{0,1\}^n}[D(f(s),h_c(s)=1]+\frac{1}{2}\cdot\Pr_{s\xleftarrow{\$}\{0,1\}^n}[D(f(s),h_c(s)=1] \\
&\qquad\qquad\qquad\qquad-\Pr_{s\xleftarrow{\$}\{0,1\}^n,\beta\xleftarrow{\$}\{0,1\}}[D(f(s),\beta)=1]\Big| \\
&\qquad=\Big|\Pr_{s\xleftarrow{\$}\{0,1\}^n}[D(f(s),h_c(s)=1]-\Pr_{s\xleftarrow{\$}\{0,1\}^n,\beta\xleftarrow{\$}\{0,1\}}[D(f(s),\beta)=1]\Big| \\
&\qquad=\Big|\Pr_{s\xleftarrow{\$}\{0,1\}^n}[D(f(s),h_c(s)=1]-\Pr_{U_{n+1}\xleftarrow{\$}\{0,1\}^{n+1}}[D(U_{n+1})=1] \\
&\qquad=\Big|\Pr_{s\xleftarrow{\$}\{0,1\}^n}[D(G(s))=1]-\Pr_{U_{n+1}\xleftarrow{\$}\{0,1\}^{n+1}}[D(U_{n+1})=1]\Big| \\
&\qquad>\frac{1}{p(n)}
\end{aligned}$$

where the first equality comes from Eq. 4, the third equality comes from Eq. 5 and the last inequality comes from Eq. 2.
Now, we construct a prob. poly time algorithm $\mathcal{A}$ that guesses $h_c(x)$ from $f(x)$ as follows:

- **Algorithm $\mathcal{A}$:** Input: $y=f(x)$
  1. Randomly samples a bit $\sigma\xleftarrow{\$}\{0,1\}$.
  2. Invoke D with (y,σ).
  3. If D outputs 1, then it returns σ as $h_c(x)$, else it returns $\bar{\sigma}$ as $h_c(x)$.

- **Success Prob. of A:**

$$\begin{aligned}
&\Pr_{s\xleftarrow{\$}\{0,1\}^n}[A(f(s))=h_c(s)] \\
&\qquad=Pr[A(f(s))=h_c(s)\wedge\sigma=h_c(s)]+Pr[A(f(s))=h_c(s)\wedge\sigma=\bar{h}_c(s)] \\
&\qquad=\frac{1}{2}\cdot\Big(Pr[A(f(s))=h_c(s)|\sigma=h_c(s)]+Pr[A(f(s))=h_c(s)\wedge\sigma=\bar{h}_c(s)]\Big) \\
&\qquad=\frac{1}{2}\cdot\Big(Pr[D(f(s),h_c(s))=1]+Pr[D(f(s),\bar{h}_c(s))=0]\Big) \\
&\qquad=\frac{1}{2}\cdot\Big(Pr[D(f(s),h_c(s))=1]+1-Pr[D(f(s),\bar{h}_c(s))=1]\Big) \\
&\qquad=\frac{1}{2}+\frac{1}{2}\cdot\Big(Pr[D(f(s),h_c(s))=1]-Pr[D(f(s),\bar{h}_c(s))=1]\Big) \\
&\qquad>\frac{1}{2}+\frac{1}{p(n)}
\end{aligned}$$

where the last inequality comes from 3. But this contradicts the theorem's hypothesis by which h_c is a hard-core of f .

5.2 Increasing the Expansion Factor

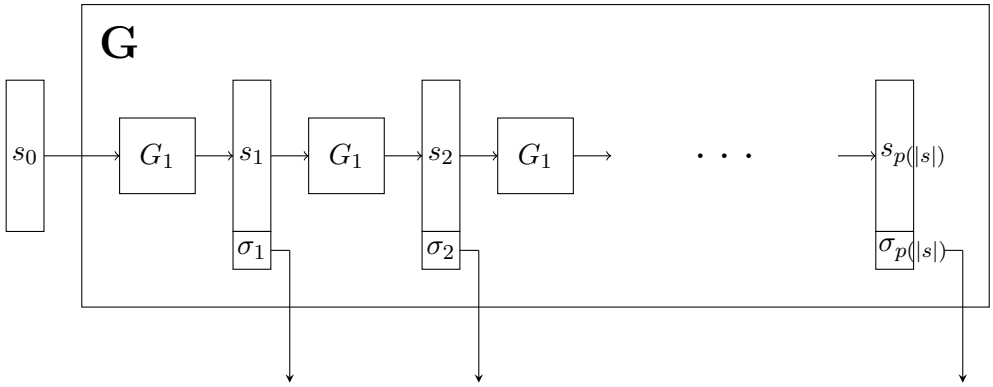
Given a pseudorandom generator G_1 with expansion factor $l_1(n)=n+1$, we construct a pseudorandom generator G with arbitrary polynomial expansion factor as follows:

Construction: Let G_1 be a deterministic polynomial-time algorithm mapping strings of length n into strings of length $n+1$, and let $p(\cdot)$ be a polynomial. Define $G(s) = \sigma_1 \cdots \sigma_{p(|s|)}$, where $s_0 \stackrel{\text{def}}{=} s$, the bit σ_i is the first bit of $G_1(s_{i-1})$, and s_i is the $|s|$-bit long suffix of $G_1(s_{i-1})$ for every $1 \leq i \leq p(|s|)$. That is, on input s, the algorithm G proceeds as follows:

Algorithm

Let $s_0 = s$ and $n = |s|$
for $i = 1$ `to` $p(n)$ **do**
  $\sigma_i s_i \leftarrow G_1(s_{i-1})$, where $\sigma_i \in \{0,1\}$ and $|s_i| = |s_{i-1}|$
Output $\sigma_1 \sigma_2 \cdots \sigma_{p(|s|)}$

The construction is depicted in the below figure:



Theorem 15. *Let G_1, $p(\cdot)$ and G be as in the above construction such that $p(n) > n$. If G_1 is pseudorandom generator, then so is G.*

Proof. See section 3.3.2 of [2].

6 PRG, One-Way Function and Unpredictability

6.1 Relation Between PRG & One-Way Function

Till now we have seen the followings:

One way functions (or permutations) exists
$\Downarrow$
Existence of Hard-Core predicate
$\Downarrow$
Pseudorandom generator exists

Now in this lecture we will prove that, Pseudorandom generators exists $\Rightarrow$ One-Way Function exists.

Theorem 16. *Let G be a PRG with length expansion factor $2n$. Let $f : \{0,1\}^* \to \{0,1\}^*$ be a function defined as:*

$$f(x,y) \stackrel{def}{=} G(x), \forall x, y \text{ such that } |x| = |y| \tag{6}$$

Now if G is a secure PRG, then f is a strongly one-way function.

Proof. Clearly, f is polynomial-time computable. Now we have to show that each probabilistic polynomial-time algorithm can invert f with only negligible success probability. We use a

reducibility argument. Suppose, on the contrary, that f is not strong one-way, i.e., $\mathcal{A}$ is a probabilistic polynomial-time algorithm that for infinitely many n's inverts f on $f(U_{2n})$ with success probability at least $\frac{1}{poly(n)}$. So, for some polynomial $p(\cdot)$ and infinitely many n's

$$Pr[f(\mathcal{A}(f(U_{2n}))) = f(U_{2n})] > \frac{1}{p(n)} \tag{7}$$

Now, we shall construct a probabilistic polynomial-time algorithm $\mathcal{D}$ that distinguishes U_{2n} and $G(U_n)$ on these n's.

Algorithm $\mathcal{D}$:
1. Input $\alpha \in \{0,1\}^{2n}$
2. It invokes the algorithm $\mathcal{A}$ with α
3. It will receive $\beta \leftarrow \mathcal{A}(\alpha)$
4. $\mathcal{D}$ will output 1 if $f(\beta) = \alpha$, and 0 otherwise.

Probability Analysis: Now, from Eq. 6 and Eq. 7 we get,

$$\begin{aligned} Pr[\mathcal{D}(G(U_n)) = 1] &= Pr[f(\mathcal{A}(G(U_n))) = G(U_n)] \\ &= Pr[f(\mathcal{A}(f(U_{2n}))) = f(U_{2n})] \\ &> \frac{1}{p(n)} \end{aligned}$$

On the other hand, by Eq. 6, at most 2^n different $2n$-bit-long strings (i.e., those in the support of $G(U_n)$) have pre-images under f. Hence,

$$Pr[\mathcal{D}(U_{2n}) = 1] = Pr[f(\mathcal{A}(U_{2n})) = U_{2n}] \leq \frac{1}{2^n}$$

So, it follows that for infinitely many n's,

$$Pr[\mathcal{D}(G(U_n)) = 1] - Pr[\mathcal{D}(U_{2n}) = 1] > \frac{1}{p(n)} - \frac{1}{2^n} > \frac{1}{2p(n)}$$

which contradicts the pseudorandomness of G. Hence f is a strong one-way function. □

6.2 Pseudorandomness and Unpredictability

A key property of pseudorandom sequences that is used to justify the use of such sequences in some cryptographic applications is the unpredictability of a sequence. Loosely speaking, a sequence is unpredictable if no efficient algorithm, given a prefix of the sequence, can guess its next bit with a non-negligible advantage over $\frac{1}{2}$.

Definition 26 (Unpredictability)**.** *An ensemble $\{X_n\}_{n\in\mathbb{N}}$ is called **Unpredictable in polynomial time** if for every probabilistic polynomial time algorithm $\mathcal{A}$, every positive polynomial $p(\cdot)$, and all sufficiently large n's,*

$$Pr\left[\mathcal{A}(1^{|X_n|}, X_n) = next_{\mathcal{A}}(X_n)\right] < \frac{1}{2} + \frac{1}{p(n)}$$

where $next_{\mathcal{A}}(x)$ returns the $i+1$ bit of x if on input $(1^{|x|}, x)$ algorithm $\mathcal{A}$ reads only $i < |x|$ of the bits of x and returns a uniformly chosen bit otherwise (i.e., in case $\mathcal{A}$ reads the entire string x).

Theorem 17 (Pseudorandomness versus Unpredictability)**.** *An ensemble $\{X_n\}_{n\in\mathbb{N}}$ is pseudorandom if and only if it is unpredictable in polynomial time.*

Proof. Let us assume that the ensemble $\{X_n\}_{n\in\mathbb{N}}$ is unpredictable in polynomial time. Suppose it is not pseudorandom. So, $\exists$ a ppt algo. $\mathcal{D}$, $\exists$ polynomial $p(\cdot)$ and for infinitely many n's,

$$|Pr[\mathcal{D}(X_n) = 1] - Pr[\mathcal{D}(U_n) = 1]| \geq \frac{1}{p(n)} \tag{8}$$

Now, for each n that satisfies Eq. 8, we define $(n+1)$ hybrids, H_n^i consists of the i^{th} bit long prefix of X_n followed by the $(n-i)$ bit long suffix of U_n.

$$H_n^i = \underbrace{x_1 x_2 \cdots x_i}_{i \text{ bits of } X_n} \overbrace{r_{i+1} r_{i+2} \ldots r_n}^{(n-i) \text{ bits of } U_n}, \quad i = 0, 1, \ldots, n-1, n$$

So, $H_n^0 = U_n$ and $H_n^n = X_n$. Now, from Eq. 8 we get that,

$$\sum_{i=0}^{n-1} \left(Pr[\mathcal{D}(H_n^{i+1}) = 1] - Pr[\mathcal{D}(H_n^i) = 1]\right) \geq \frac{1}{p(n)} \tag{9}$$

Now, we will construct an algorithm which will break the unpredictability of $\{X_n\}$.

Algorithm $\mathcal{A}$: Input: $x = x_1 x_2 \ldots x_n$

- Upon receiving x, select $i \xleftarrow{\$} \{0, 1, \ldots, n-1\}$
- Read first i bits of x
- $r_{i+1}, \ldots, r_n \xleftarrow{\$} \{0, 1\}$
- Invokes $\mathcal{D}$ with $x_1 x_2 \ldots x_i r_{i+1} \ldots r_n$
- Receives b from $\mathcal{D}$
- If $b = 1$ output r_{i+1}, otherwise $\overline{r_{i+1}}$

Now, if $r_{i+1} = x_{i+1}$, then $\mathcal{D}$ is invoked on input distributed identically to H_n^{i+1}, otherwise if $r_{i+1} = \overline{x_{i+1}}$ then let, $\mathcal{D}$ is invoked on input distributed identically, say Z.

Now, by construction of H_n^i, we see that, H_n^i is equals to Z with probability $\frac{1}{2}$ and H_n^{i+1} with probability $\frac{1}{2}$. Since, H_n^i is distributed identically to the distribution obtained by taking $H_n^{i+1} = X^1 X^2 \ldots X^i X^{i+1} R^{i+2} \ldots R^n$ with probability $\frac{1}{2}$ and $Z = X^1 X^2 \ldots X^i \overline{X^{i+1}} R^{i+2} \ldots R^n$ with probability $\frac{1}{2}$, so,

$$\begin{aligned}
Pr[\mathcal{D}(H_n^i) = 1] &= Pr[\mathcal{D}(H_n^i) = 1 \wedge R^{i+1} = X^{i+1}] + Pr[\mathcal{D}(H_n^i) = 1 \wedge R^{i+1} = \overline{X^{i+1}}] \\
&= Pr[\mathcal{D}(H_n^i) = 1 | R^{i+1} = X^{i+1}] \cdot Pr[R^{i+1} = X^{i+1}] \\
&\qquad + Pr[\mathcal{D}(H_n^i) = 1 | R^{i+1} = \overline{X^{i+1}}] \cdot Pr[R^{i+1} = \overline{X^{i+1}}] \\
&= \frac{1}{2} \cdot Pr[\mathcal{D}(H_n^{i+1}) = 1] + \frac{1}{2} \cdot Pr[\mathcal{D}(Z) = 1]
\end{aligned}$$

So, we have,

$$Pr[\mathcal{D}(Z) = 1] = 2 \cdot Pr[\mathcal{D}(H_n^i) = 1] - Pr[\mathcal{D}(H_n^{i+1}) = 1]$$

Now,

$$
\begin{aligned}
&Pr[\mathcal{A}(1^n, X_n) = next_{\mathcal{A}}(X_n)] \\
&\quad = \sum_{i=0}^{n-1} \Big(Pr[\mathcal{A}(1^n, X_n) = next_{\mathcal{A}}(X_n) \wedge I = i] \Big) \\
&\quad = \sum_{i=0}^{n-1} Pr[\mathcal{A}(1^n, X_n) = next_{\mathcal{A}}(X_n) | I = i] \cdot \frac{1}{n} \\
&\quad = \frac{1}{n} \cdot \sum_{i=0}^{n-1} \Big(Pr[\mathcal{D}(X^1 \dots X^i R^{i+1} \dots R^n) = 1 \wedge R^{i+1} = X^{i+1}] \\
&\qquad\qquad + Pr[\mathcal{D}(X^1 \dots X^i R^{i+1} \dots R^n) = 0 \wedge R^{i+1} = \overline{X^{i+1}}] \Big) \\
&\quad = \frac{1}{n} \cdot \sum_{i=0}^{n-1} \Big(Pr[\mathcal{D}(X^1 \dots X^i R^{i+1} \dots R^n) = 1 | R^{i+1} = X^{i+1}] \cdot Pr[R^{i+1} = X^{i+1}] \\
&\qquad\qquad + Pr[\mathcal{D}(X^1 \dots X^i R^{i+1} \dots R^n) = 0 | R^{i+1} = \overline{X^{i+1}}] \cdot Pr[R^{i+1} = \overline{X^{i+1}}] \Big) \\
&\quad = \frac{1}{n} \cdot \sum_{i=0}^{n-1} \Big(\frac{1}{2} \cdot Pr[\mathcal{D}(X^1 \dots X^i X^{i+1} R^{i+2} \dots R^n) = 1] \\
&\qquad\qquad + \frac{1}{2} \cdot Pr[\mathcal{D}(X^1 \dots X^i \overline{X^{i+1}} R^{i+2} \dots R^n) = 0] \Big) \\
&\quad = \frac{1}{2n} \cdot \sum_{i=0}^{n-1} \Big(Pr[\mathcal{D}(X^1 \dots X^i X^{i+1} R^{i+2} \dots R^n) = 1] \\
&\qquad\qquad + 1 - Pr[\mathcal{D}(X^1 \dots X^i \overline{X^{i+1}} R^{i+2} \dots R^n) = 1] \Big) \\
&\quad = \frac{1}{2} + \frac{1}{2n} \cdot \sum_{i=0}^{n-1} \Big(Pr[\mathcal{D}(H_n^{i+1}) = 1] - Pr[\mathcal{D}(Z) = 1] \Big) \\
&\quad = \frac{1}{2} + \frac{1}{2n} \cdot \sum_{i=0}^{n-1} \Big(Pr[\mathcal{D}(H_n^{i+1}) = 1] - 2 \cdot Pr[\mathcal{D}(H_n^i) = 1] + Pr[\mathcal{D}(H_n^{i+1}) = 1] \Big) \\
&\quad = \frac{1}{2} + \frac{1}{2n} \cdot \sum_{i=0}^{n-1} \Big(2 \cdot Pr[\mathcal{D}(H_n^{i+1}) = 1] - 2 \cdot Pr[\mathcal{D}(H_n^i) = 1] \Big) \\
&\quad = \frac{1}{2} + \frac{1}{n} \cdot \sum_{i=0}^{n-1} \Big(Pr[\mathcal{D}(H_n^{i+1}) = 1] - Pr[\mathcal{D}(H_n^i) = 1] \Big) \\
&\quad \geq \frac{1}{2} + \frac{1}{n} \cdot \frac{1}{p(n)}
\end{aligned}
$$

where, the lastc inequality follows from Eq. 9. So, the ensemble $\{X_n\}_{n \in \mathbb{N}}$ is predictable. $\rightarrow\leftarrow$ Hence, **Unpredictability** implies **Pseudorandomness**.

Conversely let, the ensemble $\{X_n\}_{n \in \mathbb{N}}$ is pseudorandom. Suppose it is not unpredictable, then $\exists$ poly time algorithm $\mathcal{A}$ and $\exists$ polynomial p and for infinitely many n's,

$$Pr[\mathcal{A}(1^n, X_n) = next_{\mathcal{A}}(X_n)] \geq \frac{1}{2} + \frac{1}{p(n)}$$

Then $\mathcal{A}$ can be easily transformed into a distinguisher, denoted $\mathcal{D}$, operating as follows. On input y, the distinguisher invokes $\mathcal{A}$ on input $(1^{|y|}, y)$ and records the number of bits that $\mathcal{A}$ has actually read, as well as $\mathcal{A}$'s prediction for the next bit. In case the prediction is correct, $\mathcal{D}$

outputs 1, and otherwise it outputs 0. Clearly,

$$\begin{aligned} Pr[\mathcal{D}(X_n) = 1] &= Pr[\mathcal{A}(1^n, X_n) = next_{\mathcal{A}}(X_n)] \\ &\geq \frac{1}{2} + \frac{1}{p(n)} \end{aligned}$$

and $Pr[\mathcal{D}(U_n) = 1] = Pr[\mathcal{A}(1^n, U_n) = next_{\mathcal{A}}(U_n)] = \frac{1}{2}$
So,

$$|Pr[\mathcal{D}(X_n) = 1] - Pr[\mathcal{D}(U_n) = 1]| \geq \frac{1}{2} + \frac{1}{p(n)} - \frac{1}{2} = \frac{1}{p(n)}$$

Therefore, $\{X_n\}_{n\in\mathbb{N}}$ is not pseudorandom. $\rightarrow\leftarrow$.
Hence, **Pseudorandomness** implies **Unpredictability**. □

7 Pseudorandom Functions & Pseudorandom Permutations

7.1 Pseudorandom Functions

Loosely speaking, pseudorandom functions are functions that cannot be distinguished from truly random functions by any efficient procedure that can get the values of the functions at arguments of choice.

Definition 27. ***Probability Ensemble:*** *Let I be a countable index set. An **ensemble indexed by** I is a sequence of random variables indexed by I. Namely, any $\mathcal{X} = \{\mathcal{X}_i\}_{i\in I}$, where each X_i is a random variable, is an ensemble indexed by I.*

Definition 28. ***Polynomial-Time Indistinguishability:***

1. *Variant for ensembles indexed by $\mathbb{N}$: Two ensembles, $\mathcal{X} = \{\mathcal{X}_n\}_{n\in\mathbb{N}}$ and $\mathcal{Y} = \{\mathcal{Y}_n\}_{n\in\mathbb{N}}$, are **indistinguishable in polynomial time** if for every probabilistic polynomial-time algorithm $\mathcal{D}$, every positive polynomial $p(\cdot)$, and all sufficiently large n's,*

$$|Pr\left[\mathcal{D}\left(\mathcal{X}_n, 1^n\right) = 1\right] - Pr\left[\mathcal{D}\left(\mathcal{Y}_n, 1^n\right) = 1\right]| < \frac{1}{p(n)}$$

2. *variant for ensembles indexed by a set of strings S: Two ensembles, $X = \{\mathcal{X}_w\}_{w\in S}$ and $Y = \{\mathcal{Y}_w\}_{w\in S}$, are **indistinguishable in polynomial time** if for every probabilistic polynomial-time algorithm $\mathcal{D}$, every positive polynomial $p(\cdot)$, and all sufficiently long $w \in S$,*

$$|Pr[\mathcal{D}(\mathcal{X}_w, w) = 1] - Pr[\mathcal{D}(\mathcal{Y}_w, w) = 1]| < \frac{1}{p(|w|)}$$

We often say computational indistinguishability instead of indistinguishability in polynomialtime.

Definition 29. ***Indistinguishability by Repeated Sampling:*** *Two ensembles, $\mathcal{X} = \{\mathcal{X}_n\}_{n\in\mathbb{N}}$ and $\mathcal{Y} = \{\mathcal{Y}_n\}_{n\in\mathbb{N}}$, are **indistinguishable by polynomial-time sampling** if for every probabilistic polynomial-time algorithm $\mathcal{D}$, every two positive polynomials $m(\cdot)$ and $p(\cdot)$, and all sufficiently large n's,*

$$\left|Pr\left[\mathcal{D}\left(\mathcal{X}_n^{(1)}, \ldots, \mathcal{X}_n^{(m(n))}\right) = 1\right] - Pr\left[\mathcal{D}\left(\mathcal{Y}_n^{(1)}, \ldots, \mathcal{Y}_n^{(m(n))}\right) = 1\right]\right| < \frac{1}{p(n)}$$

where $\mathcal{X}_n^{(1)}$ through $\mathcal{X}_n^{(m(n))}$ and $\mathcal{Y}_n^{(1)}$ through $\mathcal{Y}_n^{(m(n))}$ are independent random variables, with each $\mathcal{X}_n^{(i)}$ identical to $\mathcal{X}_n$ and each $\mathcal{Y}_n^{(i)}$ identical to $\mathcal{Y}_n$.

Definition 30. ***Efficiently Constructible Ensembles:*** *An ensemble $\mathcal{X} = \{\mathcal{X}_n\}_{n\in\mathbb{N}}$ is said to be **polynomial-time-constructible** if there exists a probabilistic polynomial-time algorithm S such that for every n, the random variables $S(1^n)$ and $\mathcal{X}_n$ are identically distributed.*

Theorem 18. *Let $\mathcal{X} = \{\mathcal{X}_n\}_{n\in\mathbb{N}}$ and $\mathcal{Y} = \{\mathcal{Y}_n\}_{n\in\mathbb{N}}$ be two polynomial-time-constructible ensembles, and suppose that $\mathcal{X}$ and $\mathcal{Y}$ are indistinguishable in polynomial time. Then $\mathcal{X}$ and $\mathcal{Y}$ are indistinguishable by polynomial-time sampling.*

Definition 31. ***Function Ensembles:*** *Let $l : \mathbb{N} \to \mathbb{N}$. An **$l$-bit function ensemble** is a sequence $\mathcal{F} = \{\mathcal{F}_n\}_{n\in\mathbb{N}}$ of random variables such that the random variable $\mathcal{F}_n$ assumes values in the set of functions mapping $l(n)$ -bit-long strings to $l(n)$-bit-long strings. The **uniform l-bit function ensemble**, denoted $\mathcal{H} = \{\mathcal{H}_n\}_{\in\mathbb{N}}$, has $\mathcal{H}_n$ uniformly distributed over the set of all functions mapping $l(n)$-bit-long strings to $l(n)$-bit-long strings.*

Definition 32. ***Pseudorandom Function Ensembles:*** *An l-bit function ensemble $\mathcal{F} = \{\mathcal{F}_n\}_{n\in\mathbb{N}}$ is called **pseudorandom** if for every probabilistic polynomial-time oracle machine $\mathcal{M}$, every polynomial $p(\cdot)$, and all sufficiently large n's,*

$$\left|Pr[\mathcal{M}^{\mathcal{F}_n}(1^n) = 1] - Pr[\mathcal{M}^{\mathcal{H}_n}(1^n) = 1]\right| < \frac{1}{p(n)}$$

where $\mathcal{H} = \{\mathcal{H}_n\}_{n\in\mathbb{N}}$ is the uniform l-bit function ensemble.

Suppose domain $D = \{0,1\}^n$, range $R = \{0,1\}^n$
$Func(D,R) = \{f : f : D \to R\}$, $|Func(D,R)| = 2^{n\cdot 2^n}$
Now, every random function T, from D to R is uniquely determined by its action on each point in D. Hence,

$$Pr[T \xleftarrow{\$} Func(D,R)] = Pr[T(x_1) = y_1, T(x_2) = y_2, \cdots, T(x_{2^n}) = y_{2^n}]$$

for any $y_1, y_2, \ldots, y_{2^n} \in \{0,1\}^n$ and $x_1, x_2, \ldots, x_{2^n} \in \{0,1\}^n$ which are all distinct.

Construction: Suppose G is a pseudorandom generator of length expansion factor $2n$. We denote by $G_0(s)$ the $|s|$-bit-long prefix of $G(s)$ and by $G_1(s)$ the $|s|$-bit-long suffix of $G(s)$(i.e., $G(s) = G_0(s)||G_1(s)$). For every $s \in \{0,1\}^n$, we define a function $f : \{0,1\}^n \to \{0,1\}^n$ such that for every $\sigma_1, \sigma_2, \ldots, \sigma_n \in \{0,1\}$,

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

i.e., on input s and $x = \sigma_1\sigma_2\cdots\sigma_n$, the value $f_s(x)$ is computed as follows:

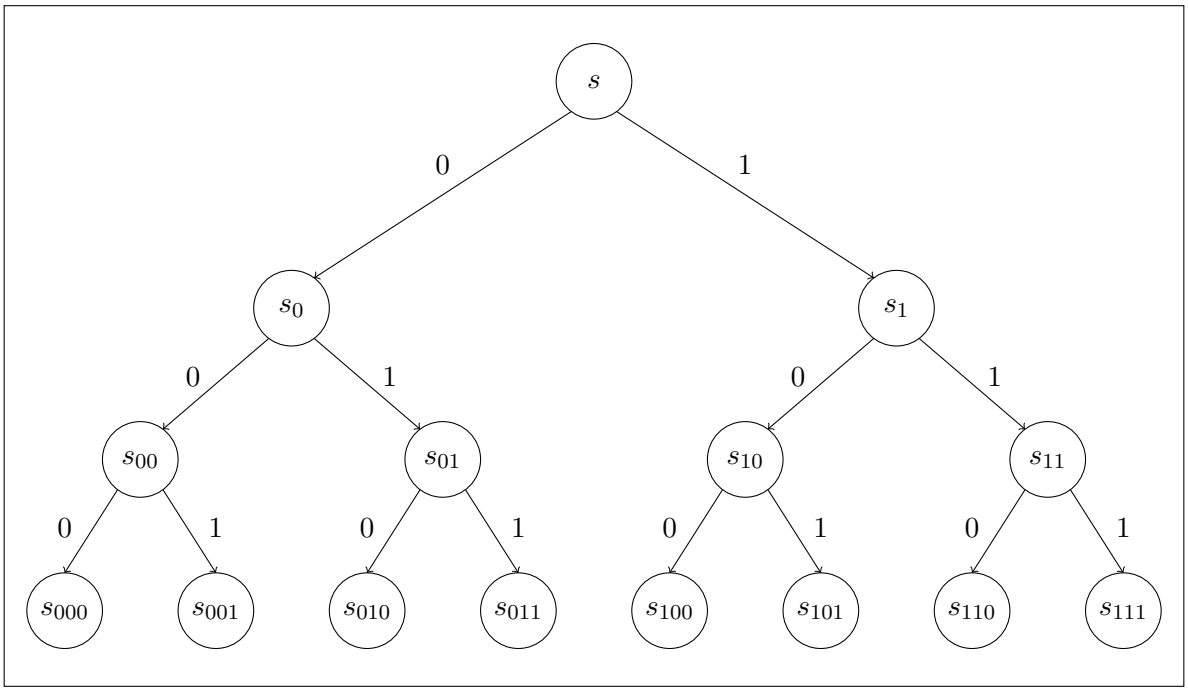
Algorithm

Let $y = s$
for $i = 1$ `to` n **do**
  $y \leftarrow G_{\sigma_i}(y)$
Output y

Let $\mathcal{F}_n$ be a random variable defined by uniformly selecting $s \in \{0,1\}^n$ and setting $\mathcal{F}_n = f_s$. Finally, let $\mathcal{F} = \{\mathcal{F}_n\}_{n\in\mathbb{N}}$ be our function ensemble.

We let $s_\lambda = s$ and $s_{\alpha\sigma} = G_\sigma(s_\alpha)$. The value of $f_s(\sigma_1\sigma_2\cdots\sigma_n) = s_{\sigma_1\sigma_2\cdots\sigma_n}$ is obtained at the leaf reachable from the root (labeled s) by following the path $\sigma_1\sigma_2\cdots\sigma_n$.

Example 4. *For example,* $f_s(001) = s_{001} = G_1(s_{00}) = G_1(G_0(s_0)) = G_1(G_0(G_0(s)))$



Theorem 19. *From the above construction,* $\mathcal{F}$ *is an efficiently computable ensemble of pseudorandom functions.*

Proof. Clearly, the ensemble $\mathcal{F}$ is efficiently computable. To prove that $\mathcal{F}$ is pseudorandom, we use the hybrid technique. The $k-th$ hybrid will be assigned a function that results from uniformly selecting labels for the vertices of the $k-th$ (highest) level of the tree and computing the labels for lower levels as in previous construction. The 0 hybrid will correspond to the random variable $\mathcal{F}_n$, whereas the n hybrid will correspond to the uniform random variable $\mathcal{H}_n$. It will be shown that an efficient oracle machine distinguishing neighboring hybrids can be transformed into an algorithm that distinguishes polynomially many samples of $G(\mathcal{U}_n)$ from polynomially many samples of $\mathcal{U}_{2n}$. Using theorem 18, we derive a contradiction to the hypothesis. Details follows.
For every k, with $0 \le k \le n$, we define a hybrid distribution $\mathcal{H}_n^k$, assigned as values functions $f : \{0,1\}^n \to \{0,1\}^n$, as follows. For every $s_1, s_2, \ldots, s_{2^k} \in \{0,1\}^n$, we define a function $f_{s_1,\ldots,s_{2^k}} : \{0,1\}^n \to \{0,1\}^n$ such that

$$f_{s_1,\ldots,s_{2^k}}(\sigma_1\sigma_2\cdots\sigma_n) := G_{\sigma_n}\left(\cdots\left(G_{\sigma_{k+2}}\left(G_{\sigma_{k+1}}\left(s_{idx(\sigma_k\cdots\sigma_1)}\right)\right)\right)\cdots\right)$$

where $idx(\alpha)$ is the index of α in the standard lexicographic order of binary strings of length $|\alpha|$. namely, $f_{s_1,\ldots,s_{2^k}}(x)$ is computed by first using the k-bit-long prefix of x to determine one of the

s_j's and then using the $(n-k)$-bit-long suffix of x to determine which of the functions G_0 and G_1 to apply at each of the remaining stages. The random variable $\mathcal{H}_n^k$ is uniformly distributed over the $(2^n)^{2^k}$ possible functions (corresponding to all possible choices of $s_1, s_2, \ldots, s_{2^k} \in \{0,1\}^n$). Namely,

$$\mathcal{H}_n^k := f_{\mathcal{U}_n^{(1)},\ldots,\mathcal{U}_n^{(2^k)}}$$

where $\mathcal{U}_n^{(j)}$'s are independent random variables, each uniformly distributed over $\{0,1\}^n$.
At this point it is clear that $\mathcal{H}_n^0$ is identical with $\mathcal{F}_n$, whereas $\mathcal{H}_n^n$ is identical to $\mathcal{H}_n$. Again, as is usual in the hybrid technique, the ability to distinguish the extreme hybrids yields the ability to distinguish a pair of neighboring hybrids. This ability is further transformed so that contradiction to the pesudorandomness of G is reached.
We assume, in contradiction to the theorem, that the function ensemble $\mathcal{F}$ is not pseudorandom. It follows that there exists a probabilistic polynomial-time oracle machine $\mathcal{M}$ and a polynomial $p(\cdot)$ such that for infinitely many n's,

$$\Delta(n) := \left| Pr\left[\mathcal{M}^{\mathcal{F}_n}(1^n) = 1\right] - Pr\left[\mathcal{M}^{\mathcal{H}_n}(1^n) = 1\right] \right| > \frac{1}{p(n)}$$

Let $t(\cdot)$ be a polynomial bounding the running time of $\mathcal{M}(1^n)$ (such a polynomial exists as $\mathcal{M}$ is a polynomial-time machine). It follows that on input 1^n, the oracle machine $\mathcal{M}$ makes at most $t(n)$ queries. Using the machine $\mathcal{M}$, we construct an algorithm $\mathcal{D}$ that distinguishes the $t(\cdot)$-product of the ensemble $\{G(\mathcal{U}_n)\}_{n\in\mathbb{N}}$ from the $t(\cdot)$-product of the ensemble $\{\mathcal{U}_{2n}\}_{n\in\mathbb{N}}$.

Algorithm $\mathcal{D}$: On input $\alpha_1, \ldots, \alpha_t \in \{0,1\}^{2n}$ (with $t = t(n)$), algorithm $\mathcal{D}$ proceeds as follows. First $\mathcal{D}$ selects uniformly $k \in \{0, 1, \ldots, n-1\}$. This random choice, hereafter called the *checkpoint*, is the only random choice made b $\mathcal{D}$ itself. Next algorithm $\mathcal{D}$ invokes the oracle machine $\mathcal{M}$ and answers $\mathcal{M}$'s queries as follows. The first query of machine $\mathcal{M}$, denoted q_1, is answered by

$$G_{\sigma_n}\left(\cdots\left(G_{\sigma_{k+2}}\left(P_{\sigma_{k+1}}(\alpha_1)\right)\right)\cdots\right)$$

where $q_1 = \sigma_1 \cdots \sigma_n$,($\alpha_1$ is the first input string) and $P_0(\alpha)$ (resp. $P_1(\alpha)$) denotes the n-bit prefix of of α(resp. the n-bit suffix of α). In addition, algorithm $\mathcal{D}$ records this query (i.e. q_1). Each subsequent query is answered by first checking to see if its k-bit-long prefix equals the k-bit-long prefix of a previous query. In case the k-bit-long prefix of the current query, denoted q_i, is different from the k-bit-long prefixes of all previous queries, we associate this prefix with a new input string(i.e. α_i). Namely, we answer query q_i by,

$$G_{\sigma_n}\left(\cdots\left(G_{\sigma_{k+2}}\left(P_{\sigma_{k+1}}(\alpha_i)\right)\right)\cdots\right)$$

where $q_i = \sigma_1 \cdots \sigma_n$. In addition, algorithm $\mathcal{D}$ records the current query(i.e. q_i). The other possibility is that the k-bit-long prefix of the i-th query equals the k-bit-long prefix of some previous query. Let j be the smallest integer such that the k-bit-long prefix of the i-th query equals the k-bit-long prefix of the j-th query(by hypothesis $j < i$). Then we record the current query (i.e. q_i), but answer it using the string associated with query q_j(i.e. the input string α_j). Namely, we answer query q_i by

$$G_{\sigma_n}\left(\cdots\left(G_{\sigma_{k+2}}\left(P_{\sigma_{k+1}}(\alpha_j)\right)\right)\cdots\right)$$

where $q_i = \sigma_1 \cdots \sigma_n$. Finally when machine $\mathcal{M}$ halts, algorithm $\mathcal{D}$ halts as well and outputs the same output as $\mathcal{M}$.
Pictorially, algorithm $\mathcal{D}$ answers the first query by first placing the two halves of α_1 in the corresponding children of the tree's vertex reached by following the path from the root corresponding to $\sigma_1 \cdots \sigma_k$. The labels of all vertices in the subtree corresponding to $\sigma_1 \cdots \sigma_k$ are determined by the labels of these two children (as in the construction of $\mathcal{F}$). Subsequent queries

are answered by following the corresponding paths from the root. In case the path does not pass through a $(k+1)$-level vertex that already has a label, we assign this vertex and its sibling a new string. For the sake of simplicity, in case the path of the i-th query requires a new string, we use the i-th input string. In case the path of a new query passes through a $(k+1)$-level vertex that has already been labeled, we use this label to compute the labels of subsequent vertices along this path. We stress that the algorithm does not compute the labels of all vertices in a subtree corresponding to $\sigma_1 \cdots \sigma_k$, but rather computes only the labels of vertices along the paths corresponding to the queries.
Clearly, algorithm $\mathcal{D}$ can be implemented in polynomial time. It is left to evaluate its performance. The key observation is the correspondence between $\mathcal{D}$'s actions on checkpoint k and the hybrids k and $k+1$:

- When the inputs are taken from the $t(n)$-product of $\mathcal{U}_{2n}$(and algorithm $\mathcal{D}$ chooses k as the checkpoint), the invoked machine $\mathcal{M}$ behaves exactly as on the $k+1$ hybrid. This is so because $\mathcal{D}$ places halves of truly random $2n$-bit-long strings at level $k+1$.
- On the other hand, when the inputs are taken from the $t(n)$-product of $G(\mathcal{U}_n)$(and algorithm $\mathcal{D}$ chooses k as the checkpoint), then $\mathcal{M}$ behaves exactly as on the k-th hybrid. Indeed $\mathcal{D}$ does not place the corresponding seeds (generating these pseudorandom strings) at level k, but putting the two halves of the pseudorandom strings at level $k+1$ has exactly the same effect.

Claim: *Let n be an integer, and $t := t(n)$. Let K be a random variable describing the random choice of checkpoint by algorithm $\mathcal{D}$ (on input a t-long sequence of $2n$-bit-long strings). Then for every $k \in \{0, 1, \ldots, n-1\}$,*

$$Pr\left[\mathcal{D}\left(G\left(\mathcal{U}_n^{(1)}\right), \ldots, G\left(\mathcal{U}_n^{(t)}\right)\right) = 1 | K = k\right] = Pr\left[\mathcal{M}^{\mathcal{H}_n^k}(1^n) = 1\right]$$

$$Pr\left[\mathcal{D}\left(\mathcal{U}_{2n}^{(1)}, \ldots, \mathcal{U}_{2n}^{(t)}\right) = 1 | K = k\right] = Pr\left[\mathcal{M}^{\mathcal{H}_n^{k+1}}(1^n) = 1\right]$$

where the $\mathcal{U}_n^{(i)}$'s and $\mathcal{U}_{2n}^{(j)}$'s are independent random variables uniformly distributed over $\{0,1\}^n$ and $\{0,1\}^{2n}$ respectively.

Proof. We start by sketching a proof of the claim for the extremely simple case in which $\mathcal{M}$'s queries are the first t strings(of length n) in lexicographic order. let us further assume, for simplicity, that on input $\alpha_1, \ldots, \alpha_t$, algorithm $\mathcal{D}$ happens to choose checkpoint k such that $t = 2^{k+1}$. In this case the oracle machine $\mathcal{M}$ is invoked on input 1^n and access to the function $f_{s_1 \ldots, s_{2^{k+1}}}$, where $s_{2j-1+\sigma} = P_\sigma(\alpha_j)$, for every $j \leq 2^k$ and $\sigma \in \{0,1\}$. Thus if the inputs to $\mathcal{D}$ are uniformly selected in $\{0,1\}^{2n}$, the $\mathcal{M}$ is invoked with access to the $k+1$ hybrid random variable (since in this case the s_j's are independent uniformly distributed in $\{0,1\}^n$). On the other hand if the inputs to $\mathcal{D}$ are distributed as $G(\mathcal{U}_n)$, then $\mathcal{M}$ is invoked with access to the k-th hybrid random variable (since in this case $f_{s_1,\ldots,s_{2^{k+1}}} = f_{r_1,\ldots,r_{2^k}}$, where the r_j's are seeds corresponding to the α_j's).

For the general case, we consider an alternative way of defining the random variable $\mathcal{H}_n^m$ for every $0 \leq m \leq n$. This alternative way is somewhat similar to the way in which $\mathcal{D}$ answers the queries of the oracle machine $\mathcal{M}$. (We use the symbol m instead of k, since m does not necessarily equal the checkpoint k chosen by algorithm $\mathcal{D}$) This way of defining $\mathcal{H}_n^m$ consists of the interleaving of two random processes, which together first select at random a function $g : \{0,1\}^m \to \{0,1\}^n$ that is later used to determine a function $f : \{0,1\}^n \to \{0,1\}^n$. The first random process denoted ρ, is an arbitrary process that specifies points in the domain of g. The second process denoted ψ, assigns uniformly selected n-bit-long strings to every new point specified by ρ, thus defining the value of g at this point. We stress that in the case ρ specifies

an old point, then the second process does nothing. The process ρ may depend on the history of the two processes and in particular on the values chosen for the previous points. When ρ terminates, the second process selects random values for the remaining undefined points. We stress that the second process is fixed for all possible choices of a process ρ.
We consider a randomized process ρ mapping sequences of n-bit strings to single m-bit strings. We stress that ρ is not necessarily memoryless. Namely, for every fixed sequence $v_1, \ldots, v_i \in \{0,1\}^n$, the random variable $\rho(v_1, \ldots, v_i)$ is distributed over $\{0,1\}^m \cup \{\perp\}$, where $\perp$ is a special symbol denoting termination. A "random" function $g : \{0,1\}^m \to \{0,1\}^n$ is defined by iterating the process ρ with the random process ψ defined next. Process ψ starts with g that is undefined on every point in its domain. At the i-th iteration, ψ lets $p_i := \rho(v_1, \ldots, v_{i-1})$ and assuming $p_i \neq \perp$, sets $v_i := v_j$ if $p_i = p_j$ for some $j < i$ and lets v_i be uniformly distributed in $\{0,1\}^n$, otherwise. In the latter case, ψ gets $g(p_i) := v_i$ (in fact, $g(p_i) = g(p_j) = v_j = v_i$ also in case $p_i = p_j$ for some $j < i$). When ρ terminates (i.e. $\rho(v_1, \ldots, v_T) = \perp$ for some T), process ψ completes the function g(if necessary) by choosing independently and uniformly in $\{0,1\}^n$ values for the points at which g is still undefined.

Once a function $g : \{0,1\}^m \to \{0,1\}^n$ is totally defined, we define a function $f^g : \{0,1\}^n \to \{0,1\}^n$ by

$$f^g(\sigma_1\sigma_2\cdots\sigma_n) := G_{\sigma_n}\left(\cdots\left(G_{\sigma_{m+2}}\left(G_{\sigma_{m+1}}\left(g(\sigma_m\cdots\sigma_1)\right)\right)\right)\cdots\right)$$

We can easily verify that f^g equals $f_{g(0^m),\ldots,g(1^m)}$. Also we can verify that the preceding random process yields a function g that is uniformly distributed over the set of all possible functions mapping m-bit strings to n-bit strings. It follows that the previously described random process yields a result that is distributed identically to the random variable $\mathcal{H}_n^m$.

Suppose now that the checkpoint chosen by $\mathcal{D}$ equals k and that $\mathcal{D}$'s inputs are independently and uniformly selected in $\{0,1\}^{2n}$. In this case the way in which $\mathcal{D}$ answers $\mathcal{M}$'s queries can be viewed as placing independently and uniformly selected n-bit strings as the labels of the $(k+1)$-level vertices. It follows that the way in which $\mathcal{D}$ answers $\mathcal{M}$'s queries corresponds to the previously described process with $m = k+1$ (with $\mathcal{M}$ playing the role of ρ and $\mathcal{D}$ playing role of ψ). Hence in this case, $\mathcal{M}$ is invoked with access to the $k+1$ hybrid random variable.

Suppose on the other hand, that (again the checkpoint chosen by $\mathcal{D}$ equals k and that) $\mathcal{D}$'s inputs are independently selected so that each is distributed identically to $G(\mathcal{U}_n)$. In this case the way in which $\mathcal{D}$ answers $\mathcal{M}$'s queries can be viewed as placing independently and uniformly selected n-bit strings as the labels of the k-level vertices. It follows that the way in which $\mathcal{D}$ answers $\mathcal{M}$'s queries corresponds to the previously described process with $m = k$. Hence, in this case $\mathcal{M}$ is invoked with the access t the k-th hybrid random variable.

Combining claim 7.1 and $\Delta(n) = Pr\left[\mathcal{M}^{\mathcal{H}_n^0}(1^n) = 1\right] - Pr\left[\mathcal{M}^{\mathcal{H}_n^k}(1^n) = 1\right]$, it follows that

$$\begin{aligned}
&Pr\left[\mathcal{D}\left(G\left(\mathcal{U}_n^{(1)}\right),\ldots,G\left(\mathcal{U}_n^{(t)}\right)\right) = 1\right] - Pr\left[\mathcal{D}\left(\mathcal{U}_{2n}^{(1)},\ldots,\mathcal{U}_{2n}^{(t)}\right) = 1\right] \\
&= \left(\frac{1}{n}\sum_{k=0}^{n-1} Pr\left[\mathcal{M}^{\mathcal{H}_n^k}(1^n) = 1\right]\right) - \left(\frac{1}{n}\sum_{k=0}^{n-1} Pr\left[\mathcal{M}^{\mathcal{H}_n^{k+1}}(1^n) = 1\right]\right) \\
&= \frac{\Delta(n)}{n}
\end{aligned}$$

which by the contradiction hypothesis, is greater than $\frac{1}{n \cdot p(n)}$ for infinitely many n's. So it follows that $\mathcal{D}$ distinguishes polynomially many samples of $G(\mathcal{U}_n)$ from polynomially many samples of $\mathcal{U}_{2n}$. Using theorem 18, we derive a contradiction to the hypothesis that G is a pseudorandom generator and the current theorem follows. □

Corollary: If there exists one-way functions, then pseudorandom functions exist as well.

Exercise 3. *Pseudorandom functions exist if and only if pseudorandom generators exist.*

7.2 Pseudorandom Permutations

In this section we present definitions and constructions for pseudorandom permutations. Clearly, pseudorandom permutations (over huge domains) can be used instead of pseudorandom functions in any efficient application, yet pseudorandom permutations offer the extra advantage of having unique pre-images.

$\text{Perm}(n) \coloneqq \{P\colon P \text{ is a permutation over } \{0,1\}^n\}$, therefore, $|\text{Perm}(n)| = (2^n)!$
A random permutation π is a uniform random variable over $\text{Perm}(n)$.

Definition 33 (**Permutation Ensembles**). *A **permutation ensemble** is a sequence $P = \{P_n\}_{n\in\mathbb{N}}$ of random variables such that the random variable P_n assumes values in the set of permutations mapping n-bit long strings to n-bit long strings. The **uniform permutation ensemble**, denoted $K = \{K_n\}_{n\in\mathbb{N}}$, has K_n uniformly distributed over the set of all permutations mapping n-bit long strings to n-bit long strings.*

Every permutation ensemble is a function ensemble. Hence the definition of an efficiently computable permutation ensemble is obvious (i.e., it is derived from the definition of an efficiently computable function ensemble). Pseudorandom permutations are defined as computationally indistinguishable from the uniform permutation ensemble.

Definition 34 (**Pseudorandom Permutation Ensembles**). *A permutation ensemble $P = \{P_n\}_{n\in\mathbb{N}}$ is called **pseudorandom** if for every probabilistic polynomial-time oracle machine M, every polynomial $p(\cdot)$, and all sufficiently large n's,*

$$\left|Pr[M^{P_n}(1^n) = 1] - |Pr[M^{K_n}(1^n) = 1]\right| < \frac{1}{p(n)}$$

where $K = \{K_n\}_{n\in\mathbb{N}}$ is the uniform permutation ensemble.

Alternatively, a keyed permutation family $E\colon \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$ such that every function in the family $\{E_k\colon \{0,1\}^n \to \{0,1\}^n\}_{k\in\mathcal{K}}$ is a bijective function, is called a pseudorandom permutation family if for every probabilistic poly-time algorithm $\mathcal{D}$, for every positive polynomial $p(\cdot)$ and for all sufficiently large n's

$$\text{Adv}_E^{PRP}(\mathcal{D}) = \left|Pr[k \xleftarrow{\$} \mathcal{K}\colon \mathcal{D}^{E_k(\cdot)} = 1] - Pr[\pi \xleftarrow{\$} \text{Perm}(n)\colon \mathcal{D}^{\pi(\cdot)} = 1]\right| \leq \frac{1}{p(n)}$$

Definition 35 (**Strong Pseudorandom Permutations**). *A permutation ensemble $P = \{P_n\}_{n\in\mathbb{N}}$ is called **strongly pseudorandom** if for every probabilistic polynomial-time oracle machine M, every polynomial $p(\cdot)$, and all sufficiently large n's,*

$$\left|Pr[M^{P_n,P_n^{-1}}(1^n) = 1] - |Pr[M^{K_n,K_n^{-1}}(1^n) = 1]\right| < \frac{1}{p(n)}$$

where $M^{f,g}$ denotes the execution of machine M when given access to the oracles f and g.

8 Statistical Distance and Distinguishing Advantage

8.1 Statistical Distance

Statistical distance of two random variable are defined as follows:

Definition 36. *Suppose X and Y are two discrete random variables defined over a finite sample space Ω. Then the **statistical distance** of X and Y, denoted by $\Delta(X,Y)$ is defined as:*

$$\Delta(X,Y) := \max_{\mathcal{T}\subseteq\Omega} \big|Pr[X \in \mathcal{T}] - Pr[Y \in \mathcal{T}]\big|$$

Equivalently *it can be defined as:*

$$\Delta(X,Y) := \tfrac{1}{2} \sum_{\omega\in\Omega} \big|Pr[X=\omega] - Pr[Y=\omega]\big|$$

The equivalence of above two definition can be established as follows:

Proof. Let $\mathcal{T}$ be any arbitrary subset of Ω. Then

$$\begin{aligned}
&2\times(Pr[X \in \mathcal{T}] - Pr[Y \in \mathcal{T}]) \\
&\quad = 2 \times Pr[X \in \mathcal{T}] - 2 \times Pr[Y \in \mathcal{T}] \\
&\quad = (Pr[X \in \mathcal{T}] - Pr[Y \in \mathcal{T}]) - (Pr[X \notin \mathcal{T}] - Pr[Y \notin \mathcal{T}]) \\
&\quad = \sum_{\omega\in\mathcal{T}} (Pr[X=\omega] - Pr[Y=\omega]) - \sum_{\omega\notin\mathcal{T}} (Pr[X=\omega] - Pr[Y=\omega]) \\
&\quad \leq \sum_{\omega\in\Omega} \big|Pr[X=\omega] - Pr[Y=\omega]\big|
\end{aligned}$$

Since the above relation holds for any arbitrary subset $\mathcal{T}$ of Ω, it also holds for which $(Pr[X \in \mathcal{T}] - Pr[Y \in \mathcal{T}])$ is maximum. □

Now we discuss about five important lemmas of statistical distance.

Lemma 1. *Let X and Y be two discrete random variables defined over a finite sample space Ω. Let $T = \{\omega \in \Omega : Pr[X=\omega] \geq Pr[Y=\omega]\}$. Then $\Delta(X,Y) = Pr[X \in T] - Pr[Y \in T]$.*

Proof. Given that, $T = \{\omega \in \Omega : Pr[X=\omega] \geq Pr[Y=\omega]\}$.
We know that,

$$\begin{aligned}
\Delta(X,Y) &= \frac{1}{2} \sum_{\omega\in\Omega} |Pr[X=\omega] - Pr[Y=\omega]| \\
&= \frac{1}{2} \left\{ \sum_{\omega\in T} |Pr[X=\omega] - Pr[Y=\omega]| + \sum_{\omega\in T^c} |Pr[X=\omega] - Pr[Y=\omega]| \right\} \\
&= \frac{1}{2} \left\{ \sum_{\omega\in T} (Pr[X=\omega] - Pr[Y=\omega]) + \sum_{\omega\in T^c} (Pr[Y=\omega] - Pr[X=\omega]) \right\} \\
&= \frac{1}{2} \left\{ \sum_{\omega\in T} (Pr[X=\omega] - Pr[Y=\omega]) + \sum_{\omega\in\Omega} (Pr[Y=\omega \wedge \omega \in T^c] - Pr[X=\omega \wedge \omega \in T^c]) \right\} \\
&= \frac{1}{2} \left\{ \sum_{\omega\in T} (Pr[X=\omega] - Pr[Y=\omega]) \right\} \\
&\quad + \frac{1}{2} \left\{ \sum_{\omega\in T^c} (Pr[Y=\omega] - Pr[Y=\omega \wedge \omega \in T] - Pr[X=\omega] + Pr[X=\omega] + Pr[X=\omega \wedge \omega \in T]) \right\} \\
&= \frac{1}{2} (Pr[X \in T] - Pr[Y \in T]) \\
&\quad + \frac{1}{2} \left\{ \sum_{\omega\in\Omega} Pr[Y=\omega] - \sum_{\omega\in T} Pr[Y=\omega] - \sum_{\omega\in\Omega} Pr[X=\omega] + \sum_{\omega\in T} Pr[X=\omega] \right\} \\
&= \frac{1}{2} \{Pr[X \in T] - Pr[Y \in T] + 1 - Pr[Y \in T] - 1 + Pr[X \in T]\} \\
&= Pr[X \in T] - Pr[Y \in T]
\end{aligned}$$

The 4^{th} equality follows from $Pr(A \cap B) = Pr(A) - Pr(A \cap B^c)$ and 7^{th} equality follows from $\sum_{\omega \in \Omega} Pr[Y = \omega] = \sum_{\omega \in \Omega} Pr[X = \omega] = 1$. □

Lemma 2. *Let X and Y be two discrete random variables defined over a finite sample space Ω. Let Z be a discrete random variable defined over $\mathcal{Z}$. Then $\Delta(X, Y) \leq \Delta((X, Z), (Y, Z))$.*

Proof. Use joint probability distribution to prove the above inequality. □

Exercise 4. *State the condition when does the equality of above lemma hold.*

Answer: If Z is independent from both X & Y then equality holds.

Lemma 3. *Let X and Y be two discrete random variables defined over a finite sample space Ω. Let $f : \Omega \to \Omega'$. Then $\Delta(X, Y) \geq \Delta(f(X), f(Y))$.*

Proof. Same as lemma 6, replacing $\mathcal{A}$ with f. □

Lemma 4. *Let X and Y be two discrete random variables such that $X \leftarrow\!\!\$\ \Omega$ and $Y \leftarrow\!\!\$\ \Omega' \subseteq \Omega$. Then $\Delta(X, Y) = \left(1 - \frac{|\Omega'|}{|\Omega|}\right)$*

Proof. $Pr[X = \omega] = \frac{1}{|\Omega|}$ and $Pr[Y = \omega] = \frac{1}{|\Omega'|}$, i.e., $Pr[Y = \omega] = 0$, whenever $\omega \in \Omega \setminus \Omega'$.
We know that,

$$
\begin{aligned}
\Delta(X, Y) &= \frac{1}{2} \sum_{\omega \in \Omega} |Pr[X = \omega] - Pr[Y = \omega]| \\
&= \frac{1}{2} \left\{ \sum_{\omega \in \Omega'} |Pr[X = \omega] - Pr[Y = \omega]| + \sum_{\omega \in \Omega \setminus \Omega'} |Pr[X = \omega] - Pr[Y = \omega]| \right\} \\
&= \frac{1}{2} \left\{ \sum_{\omega \in \Omega'} |Pr[X = \omega] - Pr[Y = \omega]| + \sum_{\omega \in \Omega \setminus \Omega'} |Pr[X = \omega] - 0| \right\} \\
&= \frac{1}{2} \left\{ \sum_{\omega \in \Omega'} \left| \frac{1}{|\Omega|} - \frac{1}{|\Omega'|} \right| + \sum_{\omega \in \Omega \setminus \Omega'} \frac{1}{|\Omega|} \right\} \\
&= \frac{1}{2} \left\{ \sum_{\omega \in \Omega'} \left(\frac{1}{|\Omega'|} - \frac{1}{|\Omega|} \right) + \frac{|\Omega \setminus \Omega'|}{|\Omega|} \right\} \\
&= \frac{1}{2} \left(\frac{|\Omega'|}{|\Omega'|} - \frac{|\Omega'|}{|\Omega|} + \frac{|\Omega \setminus \Omega'|}{|\Omega|} \right) \\
&= \frac{1}{2} \left(1 - \frac{|\Omega'|}{|\Omega|} + \frac{|\Omega|}{|\Omega|} - \frac{|\Omega'|}{|\Omega|} \right) \\
&= \frac{1}{2} \left(1 - 2 \cdot \frac{|\Omega'|}{|\Omega|} + 1 \right) \\
&= 1 - \frac{|\Omega'|}{|\Omega|}
\end{aligned}
$$

The 5^{th} equality follows from $\Omega' \subseteq \Omega \implies |\Omega'| \leq |\Omega|$ and 7^{th} equality follows from $A \subseteq B \implies |A \setminus B| = |A| - |B|$. □

Lemma 5 (H-coefficient technique). *Let X and Y be two discrete random variables defined over a finite sample space Ω. $\forall \omega \in \Omega_0 \subseteq \Omega, Pr[X = \omega] \geq (1 - \epsilon) \cdot Pr[Y = \omega]$, for some $\epsilon \in [0, 1]$. Then $\Delta(X, Y) \leq \epsilon + Pr[Y \notin \Omega_0]$.*

Proof. Let $T = \{\omega \in \Omega : Pr[Y = \omega] \geq Pr[X = \omega]\}$. Then,

$$\begin{aligned}
\Delta(X, Y) &= Pr[Y \in T] - Pr[X \in T] \\
&= \sum_{\omega \in T} (Pr[Y = \omega] - Pr[X = \omega]) \\
&= \sum_{\omega \in T \cap \Omega_0} (Pr[Y = \omega] - Pr[X = \omega]) + \sum_{\omega \in T \cap \Omega_0^c} (Pr[Y = \omega] - Pr[X = \omega]) \\
&\leq \epsilon \cdot \sum_{\omega \in T \cap \Omega_0} Pr[Y = \omega] + \sum_{\omega \in T \setminus \Omega_0} Pr[Y = \omega] \\
&\leq \epsilon + \sum_{\omega \notin \Omega_0} Pr[Y = \omega] \text{ (since, } T \setminus \Omega_0 \subseteq \Omega_0^c) \\
&= \epsilon + Pr[Y \notin \Omega_0]
\end{aligned}$$

□

8.2 Distinguishing Advantage

Before **Distinguishing Advantage**, we shall discuss about **Probabilistic Algorithm**.

8.2.1 Probabilistic Algorithm

A probabilistic algorithm $\mathcal{A}$ is an algorithm that takes two inputs, a random input r and a real input x and it outputs $y \leftarrow \mathcal{A}(r; x)$. r is called the random string of fixed length which is not provided externally, instead the algorithm chooses the random string internally from some random source, called coin space $\mathcal{C}$. Random string is used to make the random choices that $\mathcal{A}$ needs to make during the computation on x. Usually we run $\mathcal{A}$ with a uniform random string and the output y of $\mathcal{A}$ is also a random variable, $\mathcal{A}(x)$.
The probabilistic algorithm $\mathcal{A}$ can be viewed in the following way:
On input $x, \mathcal{A}$ chooses $r \leftarrow^{\$} \mathcal{C}$ which is independent of the input distribution. Then it calls some deterministic algorithm $\mathcal{A}_{det}$ with input r and x. If $\mathcal{A}$ is a randomized algorithm using randomness from $\mathcal{C}$ then we use $\mathcal{A}(x)$ to denote the random variable defined as follows:

- Sample $r \xleftarrow{\$} \mathcal{C}$
- Compute $y = \mathcal{A}_{det}(r; x)$
- Output y

We write the output of $\mathcal{A}$, on input x as $y \leftarrow \mathcal{A}(x)$.
To define the probability distribution of random variable $\mathcal{A}(x)$, let y be an element in the output space of the algorithm $\mathcal{A}$.

$$Pr[\mathcal{A}(x) = y] = \frac{\#\{r : \mathcal{A}_{det}(r; x) = y\}}{|\ \mathcal{C}\ |} \tag{10}$$

where x is a fixed element in the input space of $\mathcal{A}$ and the probability is taken over r.

Variant: When the input to the probabilistic algorithm is a random variable.
If X is a random variable over the input space of $\mathcal{A}$, then $\mathcal{A}(X)$ is also a random variable over the output space of $\mathcal{A}$ defined by:

- Sample $x \xleftarrow{\mu_1} \mathcal{X}$
- Sample $y \xleftarrow{\mu_2} \mathcal{A}(x)$
- Output y

Since $\mathcal{A}(x)$ is a random variable, we can talk about the statistical distance of $\mathcal{A}(X_0)$ and $\mathcal{A}(X_1)$ where X_0 and X_1 are two random variables.

Definition 37. *Let X_0 and X_1 are two random variables defined over $\mathcal{X}$. Let $\mathcal{A}$ be a probabilistic polynomial time algorithm having the input space $\mathcal{X}$ and the output space $\mathcal{Y}$. Then*

$$\Delta(\mathcal{A}(X_0), \mathcal{A}(X_1)) = \frac{1}{2}\sum_{y\in\mathcal{Y}}\big|Pr[\mathcal{A}(X_0) = y] - Pr[\mathcal{A}(X_1) = y]\big| \tag{11}$$

and equivalently

$$\Delta(\mathcal{A}(X_0), \mathcal{A}(X_1)) = \max_{\mathcal{B}\subset\mathcal{Y}}\big|Pr[\mathcal{A}(X_0) \in \mathcal{B}] - Pr[\mathcal{A}(X_1) \in \mathcal{B}]\big| \tag{12}$$

Now we prove the following lemma that says statistical distance of output of a probabilistic algorithm cannot be more than the statistical distance of its input.

Lemma 6. *Let X_0 and X_1 are two random variables defined over $\mathcal{X}$. Let $\mathcal{A}$ be a probabilistic polynomial time algorithm having the input space $\mathcal{X}$ and the output space $\mathcal{Y}$. Then, $\Delta(\mathcal{A}(X_0), \mathcal{A}(X_1)) \leq \Delta(X_0, X_1)$*

Proof. We consider an arbitrary set B $\subset \mathcal{Y}$ and for $r \in \mathcal{C}$, we consider the set $\mathcal{X}_r = x \in \mathcal{X} : \mathcal{A}_{det}(r; x) \in B \subseteq \mathcal{X}$. Therefore,

$$\begin{aligned}
&\big|Pr[\mathcal{A}(X_0) \in B] - Pr[\mathcal{A}(X_1) \in B]\big| \\
&= \big|Pr[\mathcal{A}_{det}(R, X_0) \in B] - Pr[\mathcal{A}_{det}(R, X_1) \in B]\big| \\
&= \big|\sum_{r\in\mathcal{C}} Pr[\mathcal{A}_{det}(R, X_0) \in B, R = r] - \sum_{r\in\mathcal{C}} Pr[\mathcal{A}_{det}(R, X_1) \in B, R = r]\big| \\
&= \big|\sum_{r\in\mathcal{C}} Pr[R = r](Pr[\mathcal{A}_{det}(R, X_0) \in B] - Pr[\mathcal{A}_{det}(R, X_1) \in B]\big| \\
&= \big|\sum_{r\in\mathcal{C}} Pr[R = r](Pr[X_0 \in \mathcal{X}_r] - Pr[X_1 \in \mathcal{X}_r])\big|
\end{aligned}$$

Let r_0 be the element of $\mathcal{C}$ such that $\big|Pr[X_0 \in \mathcal{X}_{r_0}] - Pr[X_1 \in \mathcal{X}_{r_0}]\big|$ is maximum.Then,

$$\begin{aligned}
&\big|\sum_{r\in\mathcal{C}} Pr[R = r](Pr[X_0 \in \mathcal{X}_r] - Pr[X_1 \in \mathcal{X}_r])\big| \\
&\leq \big|Pr[X_0 \in \mathcal{X}_{r_0}] - Pr[X_1 \in \mathcal{X}_{r_0}]\big| \\
&\leq \Delta(X_0, X_1)
\end{aligned}$$

We have shown that for any B $\subset \mathcal{Y}$,

$$\big|Pr[\mathcal{A}(X_0) \in B] - Pr[\mathcal{A}(X_1) \in B]\big| \leq \Delta(X_0, X_1)$$

Hence it also holds for that subset $\tilde{B} \subset \mathcal{Y}$ for which $\big|Pr[\mathcal{A}(X_0) \in \tilde{B}] - Pr[\mathcal{A}(X_1) \in \tilde{B}]\big|$ is maximum, which is $\Delta(\mathcal{A}(X_0), \mathcal{A}(X_1))$. Therefore,

$$\Delta(\mathcal{A}(X_0), \mathcal{A}(X_1)) \leq \Delta(X_0, X_1).$$

□

8.2.2 Distinguishing Advantage Of Algorithm

Let $\mathcal{A}$ be a probabilistic algorithm and X_0, X_1 be two discrete random variables. To measure how well $\mathcal{A}$ can distinguish between these two samples. For these we play a game defined as follows:

- Sample $b \xleftarrow{\$} \{0,1\}$
- Compute $x \leftarrow X_b$
- Output $b' \leftarrow \mathcal{A}(x)$

We think of b' as guess of $\mathcal{A}$ at b. A fixed guess $b' = 0$ will allow to be correct with probability $\frac{1}{2}$. This is trivial and so we are interested in how much better than $\frac{1}{2}\mathcal{A}$ distinguishes X_0 from X_1, which is written as $\big|Pr[b' = b] - \frac{1}{2}\big|$, which always lies between $[0, \frac{1}{2}]$. To normalize the above definition we just multiply it by 2, i.e

$$\mathbf{Adv}(\mathcal{A}) = 2\big|Pr[b' = b] - \tfrac{1}{2}\big|.$$

We define equivalently the advantage of an algorithm as follows:

Definition 38. *Let X_0 and X_1 be two random variables and $\mathcal{A}$ be any probabilistic algorithm outputting $b' \in \{0,1\}$. The advantage of $\mathcal{A}$ in distinguishing X_0 and X_1 is*

$$\mathbf{Adv}(\mathcal{A}) = \big|Pr[\mathcal{A}(X_0) = 1] - Pr[\mathcal{A}(X_1) = 1]\big|.$$

Such an algorithm $\mathcal{A}$ is called a distinguisher.

To show the equivalence of two definitions let us compute as follows:

$$\begin{aligned}
Pr[b' = b] &= Pr[b' = 0 \wedge b = 0] + Pr[b' = 1 \wedge b = 1] \\
&= \frac{1}{2} \cdot Pr[b' = 0 \big| b = 0] + \frac{1}{2} \cdot Pr[b' = 1 \big| b = 1] \\
&= \frac{1}{2} \cdot Pr[\mathcal{A}(X_0) = 0] + \frac{1}{2} \cdot Pr[\mathcal{A}(X_1) = 1] \\
&= \frac{1}{2} - \frac{1}{2} \cdot Pr[\mathcal{A}(X_0) = 1] + \frac{1}{2} \cdot Pr[\mathcal{A}(X_1) = 1]
\end{aligned}$$

Therefore,

$$\begin{aligned}
\mathbf{Adv}(\mathcal{A}) &= 2 \cdot \left| Pr[b' = b] - \frac{1}{2} \right| \\
&= 2 \cdot \left| \frac{1}{2} \cdot Pr[\mathcal{A}(X_1) = 1] - \frac{1}{2} \cdot Pr[\mathcal{A}(X_0) = 1] \right| \\
&= \big| Pr[\mathcal{A}(X_1) = 1] - Pr[\mathcal{A}(X_0) = 1] \big| \\
&= \big| Pr[\mathcal{A}(X_0) = 1] - Pr[\mathcal{A}(X_1) = 1] \big|.
\end{aligned}$$

Lemma 7. *Let X_0 and X_1 be two discrete random variables defined over $\mathcal{X}$ and $\mathcal{A}$ be a distinguisher defined over the input space $\mathcal{X}$. Then $\boldsymbol{Adv}(\mathcal{A}) = \Delta(\mathcal{A}(X_0), \mathcal{A}(X_1))$.*

Proof.

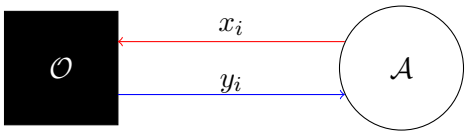
$$\begin{aligned}
&\Delta(\mathcal{A}(X_0), \mathcal{A}(X_1)) \\
&\quad = \frac{1}{2} \cdot \sum_{y \in \mathcal{Y}} \big| Pr[\mathcal{A}(X_0) = y] - Pr[\mathcal{A}(X_1) = y] \big| \\
&\quad = \frac{1}{2} \cdot \big[\big| Pr[\mathcal{A}(X_0) = 0] - Pr[\mathcal{A}(X_1) = 0] \big| + \big| Pr[\mathcal{A}(X_0) = 1] - Pr[\mathcal{A}(X_1) = 1] \big| \big] \\
&\quad = \frac{1}{2} \cdot \big[\big| 1 - Pr[\mathcal{A}(X_0) = 1] - 1 + Pr[\mathcal{A}(X_1) = 1] \big| + \big| Pr[\mathcal{A}(X_0) = 1] - Pr[\mathcal{A}(X_1) = 1] \big| \big] \\
&\quad = \frac{1}{2} \times 2 \cdot \big| Pr[\mathcal{A}(X_0) = 1] - Pr[\mathcal{A}(X_1) = 1] \big| \\
&\quad = \big| Pr[\mathcal{A}(X_0) = 1] - Pr[\mathcal{A}(X_1) = 1] \big| \\
&\quad = \mathbf{Adv}(\mathcal{A})
\end{aligned}$$

□

Since for any probabilistic polynomial time algorithm $\mathcal{A}$, $\Delta(\mathcal{A}(X_0), \mathcal{A}(X_1)) \leq \Delta(X_0, X_1)$, so we can conclude that $\mathbf{Adv}(\mathcal{A}) \leq \Delta(X_0, X_1)$.

9 Oracle Algorithm and Its Advantage

In this lecture we will discuss about oracle algorithm and its advantage and then we will see a construction of PRP and its PRP advantage.
Suppose $Func(D, R) = \{f : D \to R\}$. We consider two random variables $\mathcal{F}$ and $\mathcal{G}$ over $Func(D, R)$. Let $\mathcal{F} \leftarrow \mu_1 Func(D, R)$ and $\mathcal{G} \leftarrow \mu_2 Func(D, R)$. We consider an oracle algorithm $\mathcal{A}$ that has access to oracle $\mathcal{O}$. We also assume that $\mathcal{A}$ is probabilistic.



1. $\mathcal{A}$ will randomly sample $r \xleftarrow{\$} C$
2. It generates its first query $x_1 = \phi_1(r)$
3. $\mathcal{A}$ will make the next query $x_2 = \phi_2(r, y_1)$, $\phi_2 : C \times R \to D$
4. In general for i^{th} query

$$x_i = \phi_i(r, y_1, \ldots, y_{i-1}), \phi_i : C \times R^{i-1} \to D$$

5. At the q^{th} step $x_q = \phi_q(r, y_1, \ldots, y_{q-1})$
6. Finally it invokes a deterministic algorithm $\mathcal{A}_{out}(r, y_1, \ldots, y_q)$, which will output either 0 or 1.

An adaptive adversary $\mathcal{A}$ is a tuple $(\phi_1, \phi_2, \ldots, \phi_q, \mathcal{A}_{out})$. Now,

$$\mathbf{Adv}^{\mathcal{F}}_{\mathcal{G}}(\mathcal{A}) = \big| Pr[\mathcal{F} \leftarrow \mu_1, \mathcal{A}^{\mathcal{F}} = 1] - Pr[\mathcal{G} \leftarrow \mu_2, \mathcal{A}^{\mathcal{G}} = 1] \big|$$

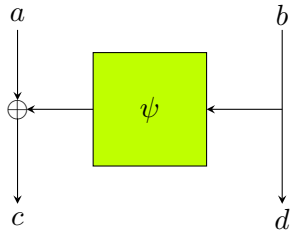
Note that, if r varies then $(y_1, \ldots, y_q)$ varies. If we fix an $r \in C$ then the conditional view $(y_1, \ldots, y_q)$ conditioned on r is denoted as $F[r; (\phi_1, \ldots, \phi_q)]$ or $G[r; (\phi_1, \ldots, \phi_q)]$. Therefore $\mathcal{A}_{out}$

distinguishes two family of random variables $\{F[r;(\phi_1,\ldots,\phi_q)]\}_{r\in C}$ and $\{G[r;(\phi_1,\ldots,\phi_q)]\}_{r\in C}$

$$\begin{aligned}\mathbf{Adv}^{\mathcal{F}}_{\mathcal{G}}(\mathcal{A}) &= Pr\Big[\mathcal{A}_{out}\big(R,F[R;(\phi_1,\ldots,\phi_q)]\big)=1\Big] - Pr\Big[\mathcal{A}_{out}\big(R,G[R;(\phi_1,\ldots,\phi_q)]\big)=1\Big]\\ &= \sum_{r\in C} Pr[R=r]\cdot \mathbf{Adv}_{\mathcal{A}_{out}}\Big(F[r;(\phi_1,\ldots,\phi_q)],G[r;(\phi_1,\ldots,\phi_q)]\Big)\\ &\leq \sum_{r\in C} Pr[R=r]\cdot \Delta\Big(F[r;(\phi_1,\ldots,\phi_q)],G[r;(\phi_1,\ldots,\phi_q)]\Big)\end{aligned}$$

9.1 1-Round Feistel Cipher

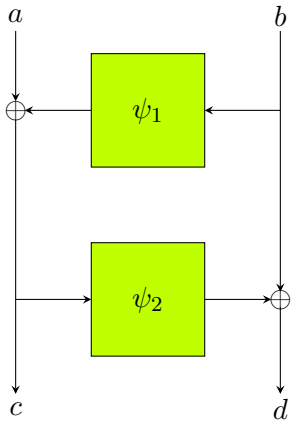
Now we will discuss a PRP construction and its PRP advantage.
Question: Look at the following sturcture. If ψ is a random function over $\{0,1\}^n$, then is it a secure PRP?



Ans. No, it is not a secure PRP. We get that, $c = a \oplus \psi(b)$ and $d = b$. Suppose we first query with (a, b) and we get (c, d) and check whether $d = b$, if it is, then we output 1 i.e., we interacts with the given system otherwise it will be an random permutation.

9.2 2-Round Feistel Cipher

Question: If ψ_1 and ψ_2 are random functions over $\{0,1\}^n$, then is the following construction a secure PRP?



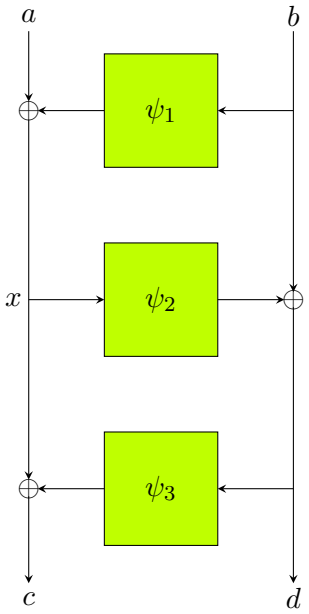
Ans. No, it is not a secure PRP. First we will check that it is a permutation. We get, $c = a \oplus \psi_1(b)$ and $d = b \oplus \psi_2(a \oplus \psi_1(b))$. If $(c_1, d_1) = (c_2, d_2)$ then $c_1 = c_2$ and $d_1 = d_2$, now

$$d_1 = d_2 \Rightarrow b_1 \oplus \psi_2(c_1) = b_2 \oplus \psi_2(c_2) \Rightarrow b_1 = b_2 \Rightarrow a_1 = a_2$$

So, it is a 1-1 function over finite domain, hence it is a permutation.
Suppose we first query with (a, b) and get (c_1, d_1) then we query with (a', b) and get (c_2, d_2). Now we will check whether $c_1 \oplus c_2 = a \oplus a'$, if it is then we output 1 i.e., we interacts with the given system otherwise it will be an random permutation.

9.3 3-Round Feistel Cipher

Now the question is can we get secured PRP extending one more round? Ok, let's go for one more round:



This is well known 3-round Luby-Rackoff [5] scheme, denoted LR3, is defined as $\psi_3 \circ \psi_2 \circ \psi_1$. Here ψ_1, ψ_2 and ψ_3 are random functions over $\{0,1\}^n$. Yes, it is a secure PRP. First we will check that extending one more round still it is a permutation or not. So, we get $c = \psi_3(\psi_2(a \oplus \psi_1(b)) \oplus b) \oplus a \oplus \psi_1(b)$ and $d = \psi_2(a \oplus \psi_1(b)) \oplus b$. Now if $(c_1, d_1) = (c_2, d_2)$ then $c_1 = c_2$ and $d_1 = d_2$, now

$$\begin{aligned} c_1 = c_2 &\Rightarrow \psi_3(d_1) \oplus a_1 \oplus \psi_1(b_1) = \psi_3(d_1) \oplus a_2 \oplus \psi_1(b_2) \\ &\Rightarrow a_1 \oplus \psi_1(b_1) = a_2 \oplus \psi_1(b_2) \end{aligned}$$

and then

$$\begin{aligned} d_1 = d_2 &\Rightarrow \psi_2(a_1 \oplus \psi_1(b_1)) \oplus b_1 = \psi_2(a_2 \oplus \psi_1(b_2)) \oplus b_2 \\ &\Rightarrow b_1 = b_2 \\ &\Rightarrow a_1 = a_2 \end{aligned}$$

Lemma 8. *For $t \leq n$ we have*

$$\mathbf{Adv}^{prp}_{LR3}(q) \leq \frac{q^2}{2^n} + \frac{q^2}{2^{2n}}$$

Proof. We apply the H-coefficient technique to prove this lemma.
Analysis of Bad Transcripts: For input (a, b) and output (c, d), let the 1- round and 2-round outputs be (x, b) and (x, d). Let $\mathbf{F}$ be the response system corresponding to LR3 and Π be the system corresponding to a random permutation. The transcript random variable τ is defined as the tuple (A^q, B^q, C^q, D^q). We say that a transcript (a^q, b^q, c^q, d^q) is bad if d^q has a colliding pair, i.e., for two distinct queries i and j, $d_i = d_j$. Let, $\Theta = \{\tau : \exists P, P(a_i, b_i) = (c_i, d_i), \forall i \in \{1, \ldots, q\}\}$, this set is called the set of all attainable transcripts. Now we can define,

$$\Theta_{Bad} = \{\tau : \exists i \neq j \in [q], d_i = d_j\}.$$

Therefore, $\Theta_{Good} = \Theta \setminus \Theta_{Bad}$. Now,

$$Pr[\mathbf{F} \in \Theta_{Bad}] \leq \frac{q(q-1)(2^n - 1)}{2(2^{2n} - 1)} \leq \frac{q^2}{2^{n+1}} \tag{13}$$

Analysis of Good Transcripts: Fix a good transcript $\tau = (a^q, b^q, c^q, d^q)$. We say that a function $f \in Func$ is bad, denoted $f \in Func_{Bad}$ if for some distinct $i, j \in [q], f(b_i) \oplus a_i = f(b_j) \oplus a_j$, otherwise we say it is good. So, for a uniform random f the probability of f being bad is bounded to at most $\frac{q^2}{2^{n+1}}$. Let $Func_{Good} = Func \setminus Func_{Bad}$. So we have,

$$\begin{aligned} Pr[\mathbf{F}(a^q, b^q) = (c^q, d^q)] &\geq Pr[\psi_1 \in Func_{Good}] \times Pr[\psi_2(x^q) = b^q \oplus d^q, \psi_3(d^q) = x^q \oplus c^q | \psi_1] \\ &\geq \left(1 - \frac{q^2}{2^{n+1}}\right) \times \frac{1}{2^{2nq}} \\ &\geq \left(1 - \frac{q^2}{2^{n+1}} - \frac{q^2}{2^{2n}}\right) \times \frac{1}{(2^{2n})_q} \end{aligned}$$

Now, $Pr[\Pi(a^q, b^q) = (c^q, d^q)] = \frac{1}{(2^{2n})_q}$, so we get,

$$\frac{Pr[\mathbf{F}(a^q, b^q) = (c^q, d^q)]}{Pr[\Pi(a^q, b^q) = (c^q, d^q)]} \geq \left(1 - \frac{q^2}{2^{n+1}} - \frac{q^2}{2^{2n}}\right) \tag{14}$$

Now from Eq. 13 and Eq. 14 using Lemma 5 we get,

$$\mathbf{Adv}^{prp}_{LR3}(q) \leq \frac{q^2}{2^{n+1}} + \frac{q^2}{2^{n+1}} + \frac{q^2}{2^{2n}} = \frac{q^2}{2^n} + \frac{q^2}{2^{2n}}.$$

□

■ **Now, We are showing that the above construction, LR3 is not a secure SPRP.**

The attack algorithm $\mathcal{A}$ is the following:

- Encryption query with (a, b) and obtains (c, d).
- Encryption query with (a', b) and obtains (c', d').
- Decryption query with $(c^* = c' \oplus a \oplus a', d^* = d')$ and obtains (a^*, b^*).
- If $b^* = d \oplus d' \oplus b$, then output 1, else output 0.

So,

$$\begin{aligned} \mathbf{Adv}^{sprp}_{LR3}(q) &= \left|Pr\left[\mathcal{A}^{LR3} = 1\right] - Pr\left[\mathcal{A}^{\Pi} = 1\right]\right| \\ &= \left|Pr\left[LR3(c^*, d^*) = (a^*, b^*)\right] - Pr\left[\Pi(c^*, d^*) = (a^*, b^*)\right]\right| \\ &= \left|\Pr_{LR3}\left[b^* = d \oplus d' \oplus b\right] - \Pr_{\Pi}\left[b^* = d \oplus d' \oplus b\right]\right| \\ &= \left|1 - \frac{1}{2^n}\right| \end{aligned}$$

This is not negligible. Hence $LR3$ is not a secure SPRP (Strong Pseudorandom Permutation).

Remark 2. *It can be proved that, extending one more round in the above construction, i.e., LR4 is a secure SPRP.*

10 Interactive Proof Systems

In this section we introduce the notion of an interactive proof system and present a non-trivial example of such a system (specifically to claims of the form "the following two graphs are not isomorphic"). The presentation is directed toward the introduction of zero-knowledge interactive proofs. Interactive proof systems are interesting for their own sake and have important complexity-theoretic applications.

10.1 ITM: Definitions

Definition 39. (**An Interactive Machine**)

- An **interactive Turing machine** (ITM) is a (deterministic) multi-tape Turing machine. The tapes are:
  - a read-only input tape,
  - a read-only random tape,
  - a read-and-write work tape,
  - a write-only output tape,
  - a pair of communication tapes (One communication tape is read-only, and the other is write-only),
  - a read-and-write switch tape consisting of a single cell.
- Each ITM is associated a single bit $\sigma \in \{0,1\}$, called its identity. An ITM is said to be active, in a configuration, if the content of its switch tape equals the machine's identity. Otherwise the machine is said to be idle. While being idle, the state of the machine, the locations of its heads on the various tapes, and the contents of the writeable tapes of the ITM are not modified.
- The content of the input tape is called input, the content of the random tape is called random input, and the content of the output tape at termination is called output. The content written on the write-only communication tape during a (time) period in which the machine is active is called the message sent at that period. Likewise, the content read from the read-only communication tape during an active period is called the message received (at that period).
  (Without loss of generality, the machine movements on both communication tapes are in only one direction, e.g., from left to right)

Definition 40 (Linked Pair of ITM)**.** *A Pair of ITM is called* ***Linked Pair of ITM*** *if it satisfies the followings:*

- *Their read-only input tape coincide (they must work on same input),*
- *Their switch tape coincide,*
- *R/O communication tape of one machine is the W/O communication tape of the other machine, and vice versa.*
- *All the other tapes of both machines (i.e., the random tape, the work tape, and the output tape) are distinct.*

In a linked pair of ITM, if a machine (A) halts while the content of the switch tape holding its identity, we say that, "both the machines ave halted". The content of the output tape of the machine A is the overall output of the computation.

Definition 41 (Joint Computation of a Linked Pair of ITM)**.** *The joint computation of a linked pair of ITMs, on a common input x, is a sequence of pairs representing the local configurations of both machines. That is, each pair consists of two strings, each representing the local configuration of one of the machines. In each such pair of local configurations, one machine (not necessarily the same one) is active, while the other machine is idle. The first pair in the sequence consists of initial configurations corresponding to the common input x, with the content of the switch tape set to zero.*

Notation: Let A and B be a linked pair of ITMs, and suppose that all possible interactions of A and B on each common input terminate in a finite number of steps. We denote by $\langle A, B\rangle(x)$ the random variable representing the (local) output of B when interacting with machine A on common input x, when the random input to each machine is uniformly and independently chosen. (Indeed, this definition is asymmetric, since it considers only B's output.)

Definition 42 (The Complexity of an Interactive Machine)**.** *Let A be an ITM. A has time complexity $t : \mathbb{N} \longrightarrow \mathbb{R}$ if for every interactive machine B, and for every x, when A interacts with B on common input x, then B halts within $t(|x|)$ steps, regardless of the content of the random tape of A and B.*
In particular if t is a polynomial, then we say A has a polynomial time complexity or A is a polynomial time interactive turing machine.

Definition 43 (Interactive Proof System)**.** *A pair of interactive machine $\langle P, V\rangle$ is called an interactive proof system for a language $\mathcal{L}$, if the followings are satisfied:*

(i) V is polynomial time ITM,

(ii) For every $x \in \mathcal{L}$ it hoklds that:

- *Completeness: For every $x \in \mathcal{L}$,*
$$Pr[\langle P, V\rangle(x) = 1] \geq \tfrac{2}{3}$$
- *Soundness: For every $x \notin \mathcal{L}$, for every possible prover P^*,*
$$Pr[\langle P^*, V\rangle(x) = 1] \leq \tfrac{1}{3}$$

Every language in $\mathcal{NP}$ has an interactive proof system. Specifically, let $\mathcal{L} \in \mathcal{NP}$, and let $R_{\mathcal{L}}$ be a witness relation associated with the language $\mathcal{L}$. Then an interactive proof for the language $\mathcal{L}$ consists of a prover that on input $x \in \mathcal{L}$ sends a witness y (as before), and a verifier that upon receiving y (on common input x) outputs 1 if $|y| =$poly$(|x|)$ and $(x, y) \in R_{\mathcal{L}}$ (and outputs 0 otherwise). Clearly, when interacting with the prescribed prover, this verifier will always accept inputs in the language. On the other hand, no matter what a cheating "prover" does, this verifier will never accept inputs not in the language.

Definition 44 (The Class $\mathcal{IP}$)**.** *The class $\mathcal{IP}$ consists of all languages having interactive proof systems.*

By the above discussion, $\mathcal{NP} \subseteq \mathcal{IP}$. Because languages in $\mathcal{BPP}$ can be viewed as each having a verifier that decides on membership without any interaction, it follows that $\mathcal{BPP} \cup \mathcal{NP} \subseteq \mathcal{IP}$.

10.2 An Example (Graph Non-Isomorphism in $\mathcal{IP}$)

Let us first define the Language: Two graphs, $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are called *isomorphic* if there exists a one-one and onto mapping π, from the vertex set V_1 to the vertex set V_2 such that $(u, v) \in E_1$ iff $(\pi(u), \pi(v)) \in E_2$. The mapping π is called the *isomorphism* between the graphs. The set of pairs of non isomorphic graphs is denoted by ***GNI***.

■ **Construction (An Interactive Proof System for Graph Non Isomorphism):**

- Common input: A pair of two graphs, $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. Suppose without loss of generality, that $V_1 = \{1, 2, \ldots, |V_1|\}$, and similarly for V_2.
- Verifier's first step ($V1$): The verifier selects at random one of the two input graphs and sends to the prover a random isomorphic copy of this graph. Namely, the verifier selects uniformly $\sigma \in \{1, 2\}$ and a random permutation π from the set of permutations over the

vertex set V_σ. The verifier constructs a graph with vertex set V_σ and edge set

$$F \stackrel{\text{def}}{=} \{(\pi(u), \pi(v)) : (u, v) \in E_\sigma\}$$

and sends (V_σ, F) to the prover.

- Motivating remark: If the input graphs are non-isomorphic, as the prover claims, then the prover should be able to distinguish (not necessarily by an efficient procedure) isomorphic copies of one graph from isomorphic copies of the other graph. However, if the input graphs are isomorphic, then a random isomorphic copy of one graph will be distributed identically to a random isomorphic copy of the other graph.
- Prover's first step ($P1$): Upon receiving a graph $G' = (V', E')$ from the verifier, the prover finds $\tau \in \{1, 2\}$ such that the graph G' is isomorphic to the input graph G_τ. (If both $\tau = 1$ and $\tau = 2$ satisfy the condition, then τ is selected arbitrarily. In case no $\tau \in \{1, 2\}$ satisfies the condition, τ is set to 0.) The prover sends τ to the verifier.
- Verifier's second step ($V2$): If the message τ received from the prover equals σ (chosen in Step $V1$), then the verifier outputs 1 (i.e., accepts the common input). Otherwise the verifier outputs 0 (i.e., rejects the common input).

This verifier program is easily implemented in probabilistic polynomial time. We do not know of a probabilistic polynomial-time implementation of the prover's program, but this is not required. We shall now show that the foregoing pair of interactive machines constitutes an interactive proof system (in the general sense) for the language ***GNI*** (Graph Non-Isomorphism).

Proposition 20. *The language **GNI** is in the class IP. Furthermore, the programs specified in the above Construction constitute a generalized interactive proof system for **GNI**, with completeness bound 1 and soundness bound $\frac{1}{2}$. Namely:*

1. *If G_1 and G_2 are not isomorphic (i.e., $(G_1, G_2) \in$ **GNI**), then the verifier always accepts (when interacting with the prover).*
2. *If G_1 and G_2 are isomorphic (i.e., $(G_1, G2) \notin$ **GNI**), then no matter with what machine the verifier interacts, it rejects the input with probability at least $\frac{1}{2}$.*

Proof. It is clear that if $(G_1, G_2) \in$ ***GNI***, then no graph can be isomorphic to both G_1 and G_2. It follows that there exists a unique τ such that the graph G' (received by the prover in Step $P1$) is isomorphic to the input graph G_τ. Hence, τ found by the prover in Step $P1$ always equals σ chosen in Step $V1$. So, *1.* follows.
Now, if $G_1, G_2 \notin$ ***GNI***, then the graph G' is isomorphic to both input graphs. Furthermore, we shall show that in this case the graph G' yields no information about σ, and consequently no machine can (on input G_1, G_2 and G') set τ such that it will equal σ with probability greater than $\frac{1}{2}$.
Let π be a permutation on the vertex set of a graph $G = (V, E)$. We denote by $\pi(G)$ the graph with vertex set V and edge set $\{(\pi(u), \pi(v)) : (u, v) \in E\}$. Let ξ be a random variable uniformly distributed over $\{1, 2\}$, and let Π be a random variable uniformly distributed over the set of permutations on V. We stress that these two random variables are independent. We are interested in the distribution of the random variable $\Pi(G_\xi)$. We are going to show that although $\Pi(G_\xi)$ is determined by the random variables Π and ξ, the random variables ξ and $\Pi(G_\xi)$ are statistically independent. In fact, we show the following:
Claim: If $G_1, G_2 \notin$ *GNI*, then for every graph G' that is isomorphic to G_1 (and G_2), it holds that

$$Pr[\xi = 1 | \Pi(G_\xi) = G'] = Pr[\xi = 2 | \Pi(G_\xi) = G'] = \frac{1}{2}$$

Proof: Let, $S_1 \stackrel{\text{def}}{=} \{\pi : \pi(G_1) = G'\}$ and $S_2 \stackrel{\text{def}}{=} \{\pi : \pi(G_2) = G'\}$. Since, there is a 1-1 and

onto correspondence between the set S_1 and S_2 (the correspondence is given by the isomorphism between the graphs G_1 and G_2), so they are of same cardinality. Hence,

$$\begin{aligned} Pr[\Pi(G_\xi) = G' | \xi = 1] &= Pr[\Pi(G_1) = G'] \\ &= Pr[\Pi \in S_1] \\ &= Pr[\Pi \in S_2] \\ &= Pr[\Pi(G_\xi) = G' | \xi = 2] \end{aligned}$$

Now using Bayes' rule, the claim follows.

The above claim says that for every pair (G_1, G_2) of isomorphic graphs, the random variable $\Pi(G_\xi)$ yields no information on ξ, and so no prover can fool the verifier into accepting with probability greater than $\frac{1}{2}$. Specifically, let R be an arbitrary randomized process (representing a possible cheating-prover strategy that depends on (G_1, G_2) that given the verifier's message in step V_1 tries to guess the value of ξ. Then, $R(\Pi(G_\xi)) = \xi$ represents the event in which the verifier accepts, and we have

$$Pr[R(\Pi(G_\xi)) = \xi] = \sum_{G'} Pr[\Pi(G_\xi) = G'] \cdot Pr[R(G') = \xi | \Pi(G_\xi) = G']$$

Now using the above claim we have,

$$\begin{aligned} Pr[R(G') = \xi | \Pi(G_\xi) = G'] &= \sum_v Pr[R(G') = v \& \xi = v | \Pi(G_\xi) = G'] \\ &= \sum_v Pr[R(G') = v] \cdot Pr[\xi = v | \Pi(G_\xi) = G'] \\ &= \sum_{v \in \{1,2\}} Pr[R(G') = v] \cdot Pr[\xi = v] \\ &= \frac{Pr[R(G') \in \{1,2\}]}{2} \\ &\leq \frac{1}{2} \end{aligned}$$

with equality in case R always outputs an element in the set $\{1, 2\}$. Part 2 of the proposition follows. □

Remark 3. *Which languages have interactive proof systems?*

It turns out that every language in $\mathcal{PSPACE}$ has an interactive proof system. In fact, $\mathcal{IP}$ equals $\mathcal{PSPACE}$. We comment that $\mathcal{PSPACE}$ is believed to be much larger than $\mathcal{NP}$; in particular, co$\mathcal{NP} \subseteq \mathcal{PSPACE}$, whereas it is commonly believed that co$\mathcal{NP} \neq \mathcal{NP}$. Also, because $\mathcal{PSPACE}$ is closed under complementation, so is $\mathcal{IP}$.

11 Zero-Knowledge Proofs

On the upcoming lectures we will try to understand a new proof technique, called the Zero-Knowledge proof.

11.1 Perfect Zero-knowledge Proof System

In this section we will introduce the notion of **zero-knowledge interactive proof system** and present a non-trivial example of such system.

Definition 45. *Let $\langle \mathcal{P}, \mathcal{V} \rangle$ be a linked pair of **ITM**. We call $\langle \mathcal{P}, \mathcal{V} \rangle$ to be a (perfect) **zero-knowledge proof system** for a language $\mathcal{L}$, if the following two conditions are satisfied:*

1. $\langle \mathcal{P}, \mathcal{V} \rangle$ *be an* ***interactive proof system*** *for language* $\mathcal{L}$.
2. *for every polynomial-time interactive Turing machine* $\mathcal{V}^*$, $\exists$ *a* probabilistic polynomial-time Turing machine $\mathcal{M}^*$ such that for every $x \in \mathcal{L}$,

$$\langle \mathcal{P}, \mathcal{V}^* \rangle(x) \cong \mathcal{M}^*(x)$$

i.e.

$$Pr[\langle \mathcal{P}, \mathcal{V}^* \rangle(x) = b] = Pr[\mathcal{M}^*(x) = b]$$

Alternative Definition of Perfect Zero-Knowledge

Definition 46. $\langle \mathcal{P}, \mathcal{V} \rangle$ *be a* ***PZK*** *proof system for a language* $\mathcal{L}$*, if it satisfies the following two conditions:*

1. $\langle \mathcal{P}, \mathcal{V} \rangle$ *is an* ***interactive proof system*** *for* $\mathcal{L}$.
2. *for every polynomial-time* ***ITM*** $\mathcal{V}^*$, $\exists$ *a* probabilistic polynomial-time Turing machine $\mathcal{M}^*$ such that the following two conditions hold-
   - $Pr[\mathcal{M}^*(x) = \perp] \leq \frac{1}{2}$ for every $x \in \mathcal{L}$.
   - for every α, it holds that

$$Pr[\langle \mathcal{P}, \mathcal{V}^* \rangle(x) = \alpha] = Pr[\mathcal{M}^*(x) = \alpha | \mathcal{M}^*(x) \neq \perp]$$

   for every $x \in \mathcal{L}$.

11.2 Computational Zero-knowledge Proof System

Definition 47. *Let* $\langle \mathcal{P}, \mathcal{V} \rangle$ *be an* ***interactive proof system*** *for some language* $\mathcal{L}$*. We say that* $\langle \mathcal{P}, \mathcal{V} \rangle$ ***computational zero-knowledge*** *if for every* probabilistic polynomial-time interactive machine $\mathcal{V}^*$ there exists a probabilistic polynomial-time algorithm $\mathcal{M}^*$ such that the following two ensembles are computationally indistinguishable:

1. $\{\langle \mathcal{P}, \mathcal{V}^* \rangle(x)\}_{x \in \mathcal{L}}$
2. $\{\mathcal{M}^*(x)\}_{x \in \mathcal{L}}$

Alternative Definition of Computational Zero-Knowledge

Definition 48. $\langle \mathcal{P}, \mathcal{V} \rangle$ *be a* ***CZK*** *proof system for a language* $\mathcal{L}$*, if*

- $\langle \mathcal{P}, \mathcal{V} \rangle$ *is an* ***interactive proof system*** *for* $\mathcal{L}$.
- *for every polynomial time* ***ITM*** $\mathcal{V}^*$, $\exists$ *a* probabilistic polynomial-time Turing machine $\mathcal{M}^*$ such that the following two conditions hold:
  1. for every $x \in \mathcal{L}$, $Pr[\mathcal{M}^*(x) = \perp] \leq \frac{1}{2}$.
  2. for every $x \in \mathcal{L}$, $\{\langle \mathcal{P}, \mathcal{V}^* \rangle(x)\}_{x \in \mathcal{L}} \approx_C \{\mathcal{M}^*(x) | \mathcal{M}^*(x) \neq \perp\}_{x \in \mathcal{L}}$

11.3 Working Definition of Zero-knowledge Proof

Definition 49. $\langle \mathcal{P}, \mathcal{V} \rangle$ *be a interactive proof system for a language* $\mathcal{L}$*. We denote by* $\text{view}^{\mathcal{P}}_{\mathcal{V}^*}(x)$ *a random variable describing the content of the random tape of* $\mathcal{V}^*$ *and the messages* $\mathcal{V}^*$ *receives from* $\mathcal{P}$ *during a joint computation on common input* x*. We say that* $(\mathcal{P}, \mathcal{V})$ ***zero-knowledge*** *if for every* probabilistic polynomial-time interactive machine $\mathcal{V}^*$ there exists a probabilistic polynomial-time algorithm M^* such that the ensembles $\{\text{view}^{\mathcal{P}}_{\mathcal{V}^*}(x)\}_{x \in \mathcal{L}}$ and $\{\mathcal{M}^*(x)\}_{x \in \mathcal{L}}$ are **computationally indistinguishable**.

We can modify the definitions of **PZK** & **CZK** by replacing $\langle \mathcal{P}, \mathcal{V}^* \rangle(x)$ with $\text{view}^{\mathcal{P}}_{\mathcal{V}^*}(x)$.

11.4 Almost-Perfect (Statistical) Zero-knowledge

Definition 50. *Let $\langle \mathcal{P}, \mathcal{V} \rangle$ be an interactive proof system for a language $\mathcal{L}$. We say that $\langle \mathcal{P}, \mathcal{V} \rangle$ is **almost-perfect zero-knowledge(statistical zero-knowledge)** if for every* probabilistic polynomial-time interactive machine $\mathcal{V}^*$ there exists a probabilistic polynomial-time algorithm $\mathcal{M}^*$ such that the following two ensembles are statistically close as functions of $|x|$:

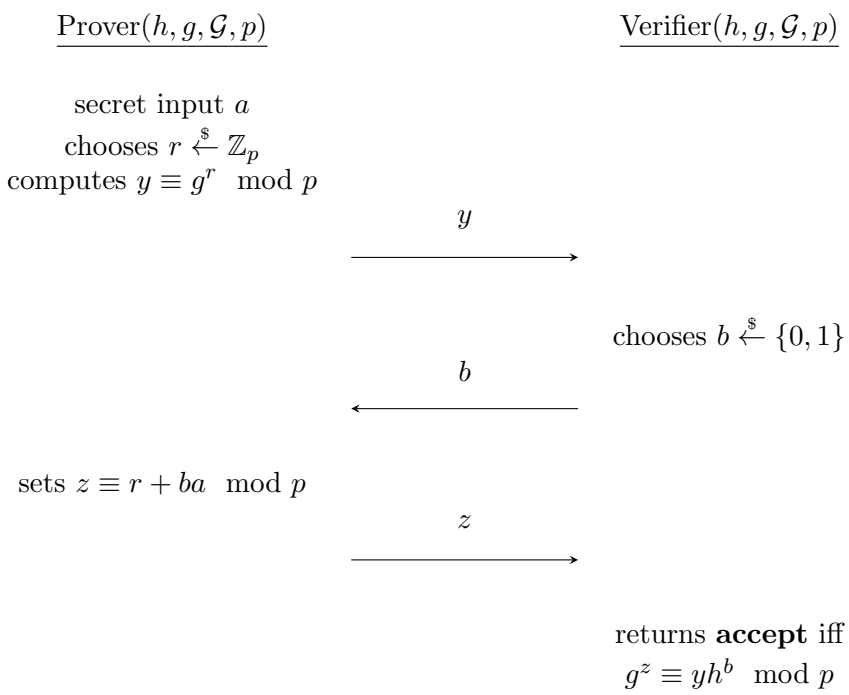
1. $\{\langle \mathcal{P}, \mathcal{V}^* \rangle(x)\}_{x \in \mathcal{L}}$
2. $\{\mathcal{M}^*(x)\}_{x \in \mathcal{L}}$

i.e.

$$\Delta(\text{view}^{\mathcal{P}}_{\mathcal{V}^*}(x), \mathcal{M}^*(x) | \mathcal{M}^*(x) \neq \perp) \leq negl(|x|)$$

11.5 Zero-Knowledge Proof for Discrete Log

- **Language:** $\mathcal{L} = \{(h, g, p) : \exists\ a \in \{1, \ldots, p-1\}, h = g^a \mod p, \mathcal{G} = \langle g \rangle\}$



- **Completeness:** If prover $\mathcal{P}$ knows the secret key a, then for $b = 0$, he sends $z = r$ and for $b = 1$, he sends $z = r + a$. Now,

$$\begin{aligned}
&Pr[\text{verifier}(\mathcal{V}) \textbf{ accepts}] \\
&= Pr[g^z \equiv yh^b \mod p] \\
&= Pr[(g^z \equiv yh^b \mod p) \wedge (b = 0)] + Pr[(g^z \equiv yh^b \mod p) \wedge (b = 1)] \\
&= Pr[(g^z \equiv yh^b \mod p) | (b = 0)] \cdot Pr[b = 0] + Pr[(g^z \equiv yh^b \mod p) | (b = 1)] \cdot Pr[b = 1] \\
&= Pr[g^r \equiv y \mod p] \cdot \frac{1}{2} + Pr[g^{r+a} \equiv yh \mod p] \cdot \frac{1}{2} \\
&= 1 \cdot \frac{1}{2} + 1 \cdot \frac{1}{2} = 1
\end{aligned}$$

- **Soundness:** Let $\mathcal{P}^*$ be a cheating prover and $h \notin \mathcal{G}$. In some strategy $\mathcal{P}^*$ needs to create the transcript (y, b, z) while interacting with $\mathcal{V}$. There are two cases that can be occurred: (*Note that:* $z \in \mathbb{Z}_p$)

- **Case 1:** y has a discrete log, i.e. $y \equiv g^r \mod p$ for some $r \in \{1, \cdots, p-1\}$. For $b = 0$,

$$Pr[(g^z \equiv yh^b \mod p) \wedge (b = 0)] \leq \frac{1}{2}$$

For $b = 1$, $z = \mathcal{P}^*(1)$ be the last message of the prover $\mathcal{P}^*$. If the prover wants to win, then it should hold that

$$g^z \equiv yh^b \mod p \iff g^z \equiv yh \mod p \iff g^z \equiv g^r h \mod p \iff h \equiv g^{z-r}$$

So, $h \in \mathcal{G}$, which arises a contradiction.Hence,

$$Pr[(g^z \equiv yh^b \mod p) \wedge (b = 1)] = 0$$

Thus,

$$Pr[\text{verifier}(\mathcal{V}) \textbf{ accepts}] \leq \frac{1}{2}$$

- **Case 2:** y has no discrete log. For $b = 1$,

$$Pr[(g^z \equiv yh^b \mod p) \wedge (b = 1)] \leq \frac{1}{2}$$

For $b = 0$, $z = \mathcal{P}^*(0)$ be the last message of the prover $\mathcal{P}^*$. If $\mathcal{P}^*$ has to win then,

$$g^z \equiv y \mod p$$

So, $y \in \mathcal{G}$, which arises a contradiction. Hence,

$$Pr[(g^z \equiv yh^b \mod p) \wedge (b = 0)] = 0$$

Thus,

$$Pr[\text{verifier}(\mathcal{V}) \textbf{ accepts}] \leq \frac{1}{2}$$

- **Zero-Knowledge Proof:** We will try to construct a simulator who can output same transcript(with identical distribution) of honest $\mathcal{P}$, without knowing the secret input 'a'.
Simulator- Input $(h, g, \mathcal{G}, p)$
  - $z \xleftarrow{\$} \mathbb{Z}_p$
  - $b' \xleftarrow{\$} \{0, 1\}$
  - if $b' = 0$, then $y \equiv g^z \mod p$
    if $b' = 1$, then $y \equiv g^z h^{-1} \mod p$
  - invokes $\mathcal{V}^*$ on input y to obtain a bit b.
  - if $b' = b$, output $\langle y, z \rangle$, else output $\perp$.

Claim: In both cases $b' = 0$ & $b' = 1$, the element y is a random element of $\mathcal{G}$.

Proof. for $b' = 0$, $y \equiv g^z \mod p$, where z is chosen uniformly at random from $\mathbb{Z}_p$ so y is random over $\mathcal{G}$. For $b' = 1$, $y \equiv g^z h^{-1} \mod p$, where $h \equiv g^a \mod p$, for some $a \in \{1, \cdots, p-1\}$, so y is random over $\mathcal{G}$. □

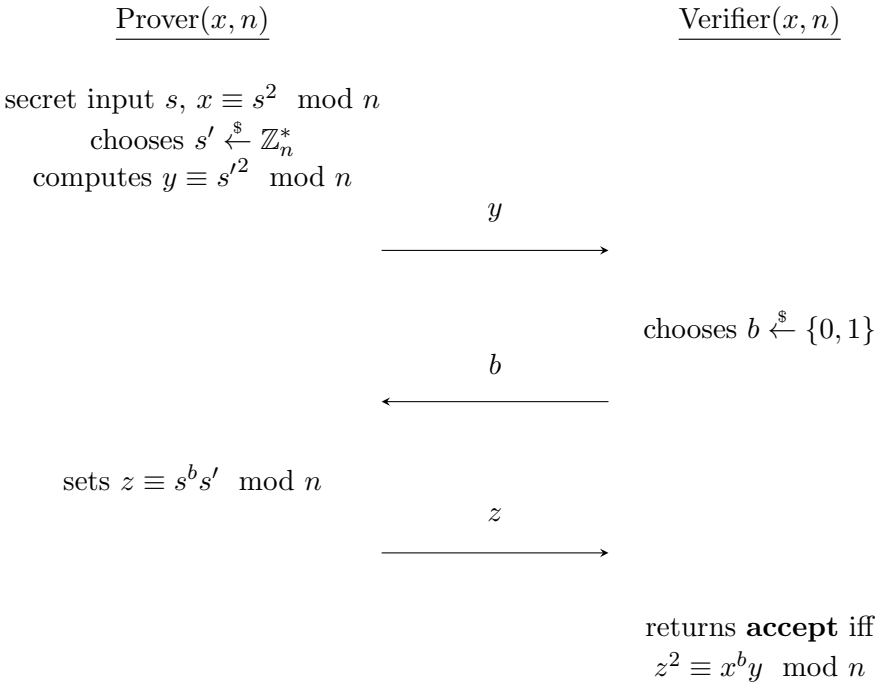
Thus, transcript of the simulator $\langle y, z\rangle$ follows identical distribution with the transcript of the honest prover $\mathcal{P}$. From this above claim, we can conclude that whatever b' is chosen y is independent of b'. For any verfier $\mathcal{V}^*$, output b depends only on its initial inputs and y, i.e. $b = \mathcal{V}^*(y)$. So, b is independent of b'. Therefore,

$$Pr[b = b'] = \frac{1}{2}$$

This means that if we run the procedure for k-steps, we will halt with very high $(1 - 2^{-k})$ probability.

11.6 Zero-Knowledge Proof for Quadratic Residue

- **Language:** $\mathcal{L} = \{(x, n) : x \text{ is a quadratic residue } \mod n,\ x \in \mathbb{Z}_n^*\}$



- **Completeness:** If prover $\mathcal{P}$ knows the secret key s, then for $b = 0$, he sends $z = s'$ and for $b = 1$, he sends $z = ss'$. Now

$$\begin{aligned}
&Pr[\text{verifier}(\mathcal{V}) \textbf{ accepts}] \\
&= Pr[z^2 \equiv x^b y \mod n] \\
&= Pr[(z^2 \equiv x^b y \mod n) \wedge (b = 0)] + Pr[(z^2 \equiv x^b y \mod n) \wedge (b = 1)] \\
&= Pr[(z^2 \equiv x^b y \mod n) | (b = 0)] \cdot Pr[b = 0] \\
&\qquad + Pr[(z^2 \equiv x^b y \mod n) | (b = 1)] \cdot Pr[b = 1] \\
&= Pr[z^2 \equiv y \mod n] \cdot \frac{1}{2} + Pr[z^2 \equiv xy \mod n] \cdot \frac{1}{2} \\
&= 1 \cdot \frac{1}{2} + 1 \cdot \frac{1}{2} = 1
\end{aligned}$$

- **Soundness:** Let $\mathcal{P}^*$ be a cheating prover and $x \notin \mathcal{QR}_n$. In some strategy $\mathcal{P}^*$ needs to create the transcript (y, b, z) while interacting with $\mathcal{V}$. There are two cases that can be occurred: (*Note that:* $z \in \mathbb{Z}_n^*$)
  - **Case 1:** $y \in \mathcal{QR}_n$, i.e. $y \equiv u^2 \mod n$ for some $u \in \mathbb{Z}_n^*$. For $b = 0$,

$$Pr[(z^2 \equiv x^b y \mod n) \wedge (b = 0)] \leq \frac{1}{2}$$

For $b = 1$, $z = \mathcal{P}^*(1)$ be the last message of the prover $\mathcal{P}^*$. If the prover wants to win, then it should hold that

$$z^2 \equiv x^b y \mod n \iff z^2 \equiv xy \mod n \iff z^2 \equiv xu^2 \mod n \iff x \equiv x^2 u^{-2}$$

So, $x \in \mathcal{QR}_n$, which arises a contradiction.Hence,

$$Pr[(z^2 \equiv x^b y \mod n) \wedge (b = 1)] = 0$$

Thus,

$$Pr[\text{verifier}(\mathcal{V}) \textbf{ accepts}] \leq \frac{1}{2}$$

– **Case 2:** $y \notin \mathcal{QR}_n$. For $b = 1$,

$$Pr[(z^2 \equiv x^b y \mod n) \wedge (b = 1)] \leq \frac{1}{2}$$

For $b = 0$, $z = \mathcal{P}^*(0)$ be the last message of the prover $\mathcal{P}^*$. If $\mathcal{P}^*$ has to win then,

$$z^2 \equiv y \mod n$$

So, $y \in \mathcal{QR}_n$, which arises a contradiction. Hence,

$$Pr[(z^2 \equiv x^b y \mod n) \wedge (b = 0)] = 0$$

Thus,

$$Pr[\text{verifier}(\mathcal{V}) \textbf{ accepts}] \leq \frac{1}{2}$$

- **Zero-Knowledge Proof:** We will try to construct a simulator who can output same transcript(with identical distribution) of honest $\mathcal{P}$, without knowing the secret input 's'.
Simulator- Input (x, n)
  - $z \xleftarrow{\$} \mathbb{Z}_n^*$
  - $b' \xleftarrow{\$} \{0, 1\}$
  - if $b' = 0$, then $y \equiv z^2 \mod n$
    if $b' = 1$, then $y \equiv z^2 x^{-1} \mod n$
  - invokes $\mathcal{V}^*$ on input y to obtain a bit b.
  - if $b' = b$, output $\langle y, z \rangle$, else output $\perp$.

Claim: In both cases $b' = 0$ & $b' = 1$, the element y is a random element of $\mathbb{Z}_n^*$.

Proof. for $b' = 0$, $y \equiv z^2 \mod n$, where z is chosen uniformly at random from $\mathbb{Z}_n^*$ so y is random over $\mathbb{Z}_n^*$. For $b' = 1$, $y \equiv z^2 x^{-1} \mod n$, where $x \equiv s^2 \mod n$, for some $u \in \mathbb{Z}_n^*$, so y is random over $\mathbb{Z}_n^*$. □

Thus, transcript of the simulator $\langle y, z \rangle$ follows identical distribution with the transcript of the honest prover $\mathcal{P}$. From this above claim, we can conclude that whatever b' is chosen y is independent of b'. For any verfier $\mathcal{V}^*$, output b depends only on its initial inputs and y, i.e. $b = \mathcal{V}^*(y)$. So, b is independent of b'. Therefore,

$$Pr[b = b'] = \frac{1}{2}$$

This means that if we run the procedure for k-steps, we will halt with very high $(1 - 2^{-k})$ probability.

11.7 Zero-Knowledge Proof for Graph Isomorphism

- **Language:** $\mathcal{L} = \{(G_1, G_2) : G_1 \cong G_2\}$

Prover(G_1, G_2)		Verifier(G_1, G_2)
secret input ϕ, $G_1 \cong_\phi G_2$ chooses $\Pi \xleftarrow{\$} \text{Perm}(V_2)$ computes $y = \Pi(G_2)$		
	$y \longrightarrow$	
		chooses $b \xleftarrow{\$} \{1, 2\}$
	$\longleftarrow b$	
sets $z = \Pi \circ \phi^b \mod 2$		
	$z \longrightarrow$	
		returns **accept** iff $z(G_b) = y$

- **Completeness:** If prover $\mathcal{P}$ knows the secret key ϕ, then for $b = 1$, he sends $z = \Pi \circ \phi$ and for $b = 2$, he sends $z = \Pi$. Now

$$
\begin{aligned}
&Pr[\text{verifier}(\mathcal{V}) \textbf{ accepts}] \\
&= Pr[z(G_b) = y] \\
&= Pr[(z(G_b) = y) \wedge (b = 1)] + Pr[(z(G_b) = y) \wedge (b = 2)] \\
&= Pr[(z(G_b) = y) | (b = 1)] \cdot Pr[b = 1] + Pr[(z(G_b) = y) | (b = 2)] \cdot Pr[b = 2] \\
&= Pr[z(G_1) = y] \cdot \frac{1}{2} + Pr[z(G_2) = y] \cdot \frac{1}{2} \\
&= Pr[\Pi \circ \phi(G_1) = y] \cdot \frac{1}{2} + Pr[\Pi(G_2) = y] \cdot \frac{1}{2} \\
&= 1 \cdot \frac{1}{2} + 1 \cdot \frac{1}{2} = 1
\end{aligned}
$$

- **Soundness:** Let $\mathcal{P}^*$ be a cheating prover and $G_1 \ncong G_2$. In some strategy $\mathcal{P}^*$ needs to create the transcript (y, b, z) while interacting with $\mathcal{V}$. There are two cases that can be occurred: (*Note that:* $z \in \text{Perm}(V_2)$)
  - **Case 1:** $y \cong G_2$, i.e. $y = \Psi(G_2)$ for some $\Psi \in \text{Perm}(V_2)$. For $b = 2$,

$$Pr[(z(G_b) = y) \wedge (b = 2)] \leq \frac{1}{2}$$

    For $b = 1$, $z = \mathcal{P}^*(1)$ be the last message of the prover $\mathcal{P}^*$. If the prover wants to win, then it should hold that

$$z(G_b) = y \iff z(G_1) = y \iff z(G_1) = \Psi(G_2) \iff G_1 = z^{-1} \circ \Psi(G_2)$$

    So, $G_1 \cong G_2$, which arises a contradiction.Hence,

$$Pr[(z(G_b) = y) \wedge (b = 1)] = 0$$

Thus,

$$Pr[\text{verifier}(\mathcal{V}) \textbf{ accepts}] \leq \frac{1}{2}$$

– **Case 2:** $y \ncong G_2$. For $b = 1$,

$$Pr[(z(G_b) = y) \wedge (b = 1)] \leq \frac{1}{2}$$

For $b = 2$, $z = \mathcal{P}^*(2)$ be the last message of the prover $\mathcal{P}^*$. If $\mathcal{P}^*$ has to win then,

$$z(G_2) = y$$

So, $y \cong G_2$, which arises a contradiction. Hence,

$$Pr[(z(G_b) = y) \wedge (b = 2)] = 0$$

Thus,

$$Pr[\text{verifier}(\mathcal{V}) \textbf{ accepts}] \leq \frac{1}{2}$$

- **Zero-Knowledge Proof:** We will try to construct a simulator who can output same transcript(with identical distribution) of honest $\mathcal{P}$, without knowing the secret input 'ϕ'.
Simulator- Input (G_1, G_2)
  - $z \xleftarrow{\$} \text{Perm}(V_2)$
  - $b' \xleftarrow{\$} \{1, 2\}$
  - if $b' = 1$, then $y = z(G_1)$
    if $b' = 2$, then $y = z(G_2)$
  - invokes $\mathcal{V}^*$ on input y to obtain a bit b.
  - if $b' = b$, output $\langle y, z \rangle$, else output $\perp$.

  Claim: In both cases $b' = 0$ & $b' = 1$, the element y is a random element of $\mathbb{Z}_n^*$.

  Proof. for $b' = 1$, $y = z(G_1)$, where z is chosen uniformly at random from $\text{Perm}(V_1)$, so y is random over $iso(G_1) = iso(G_2)$. For $b' = 2$, $y = z(G_2)$,where z is chosen uniformly at random from $\text{Perm}(V_2)$, so y is random over $iso(G_2)$.
  Note that: Here we consider V_1 and V_2 are same set. We can consider this as $G_1 \cong G_2$.
  $iso(G)$:= set of isomorphic graphs of G

  □

  Thus, transcript of the simulator $\langle y, z \rangle$ follows identical distribution with the transcript of the honest prover $\mathcal{P}$. From this above claim, we can conclude that whatever b' is chosen y is independent of b'. For any verfier $\mathcal{V}^*$, output b depends only on its initial inputs and y, i.e. $b = \mathcal{V}^*(y)$. So, b is independent of b'. Therefore,

$$Pr[b = b'] = \frac{1}{2}$$

This means that if we run the procedure for k-steps, we will halt with very high $(1 - 2^{-k})$ probability.

12 Bit Commitment Scheme

Now onward our main aim is to proof that *every $\mathcal{NP}$ problem has a Zero knowledge proof, if One-way function exists.* To prove this statement we need to develop some idea on *Commitment Scheme.*

12.1 Commitment Scheme using Hash function

Any Commitment Scheme consists of two operations- Hiding & Revealing. One simple way of committing a message m is to use a collision-free Hash function H. Sender will choose m from the message space and compute $y = H(m)$ and sends it to the receiver. Since H is collision-free so it is pre-image resistant also, i.e. having $H(m)$ one can get back m with at most negligible probability. So it maintains the *Hiding property*. On the other hand on the *revealing phase* sender can send m to the receiver and receiver can verify that whether $H(m) = y$ or not. Since H is collision-free so receiver can verify with at least non-negligible probability. Note that here we consider that receiver is a *probabilistic polynomial-time Turing machine*.

12.2 Definition of Bit-Commitment Scheme

A **Bit-Commitment Scheme** is a pair of *probabilistic polynomial-time interactive machines, denoted* $(\mathcal{S}, \mathcal{R})$ *(for sender and receiver)*, satisfying the following:

- **Input specification:** *The* common input is an integer n presented id unary (serving as the security parameter).
- **Secrecy (or hiding):** *The receiver (even when deviating arbitrarily from the protocol) cannot distinguish a commitment to* 0 *from a commitment to* 1. *Namely, for every* probabilistic polynomial-time machine $\mathcal{R}^*$ interacting with S, the probability ensembles describing the output of $\mathcal{R}^*$ in the two cases, namely $\{\langle \mathcal{S}(0), \mathcal{R}^* \rangle (1^n)\}_{n \in \mathbb{N}}$ and $\{\langle \mathcal{S}(1), \mathcal{R}^* \rangle (1^n)\}_{n \in \mathbb{N}}$, are computationally indistinguishable.
- **Unambiguity (or binding):** *Preliminaries to the requirement:*
  1. **Receiver's view:** *A* ***receiver's view*** *of an interaction with the sender, denoted* $(r, \bar{m})$*, consists of the random coins used by the receiver* (r) *and the sequence of messages received from the sender* $(\bar{m})$.
  2. **Possible σ-Commitment:** *We say that a receiver's view (of such interaction),* $(r, \bar{m})$*, is a* ***possible σ-commitment*** *if there exists a string* s *such that* $\bar{m}$ *describes the messages received by* $\mathcal{R}$ *when* $\mathcal{R}$ *uses local coins* r *and interacts with machine* S *that uses local coins* s *and has inputs* $(\sigma, 1^n)$.
  (Using the notation of Definition 49, we say that $(r, \bar{m})$ *is a* possible σ-commitment if $(r, \bar{m}) = \text{view}^{\mathcal{S}(\sigma, 1^n, s)}_{\mathcal{R}(1^n, r)}$.)

Sender		Receiver
Random tape (s)	$\xrightarrow{m_1}$	Random tape (r)
Input tape(n)	$\longleftarrow$	Input tape(n)
Auxiliary input(σ)	$\xrightarrow{m_2}$	

  $\bar{m} = m_1 || m_2 || \cdots || m_r$
  $(r, \bar{m})$ *is an* ***admissible tuple*** *for committing bit* σ. *Fix a* σ,

$$Pr[(R, \bar{M}) = (r, \bar{m})] = \frac{|\{(r, \bar{m}) = \text{view}^{\mathcal{S}(\sigma, 1^n, s)}_{\mathcal{R}^*(1^n, r)}\}|}{|\{R\}|}$$

  3. **Ambiguity:** *We say that the receiver's view* $(r, \bar{m})$ *is* ***ambiguous*** *if it is both a possible* 0*-commitment and a possible* 1*-commitment. Therefore,* $(r, \bar{m})$ *is* ***ambiguous*** *if*

$$(r, \bar{m}) = \text{view}^{\mathcal{S}(0, 1^n, s_0)}_{\mathcal{R}^*(1^n, r)} = \text{view}^{\mathcal{S}(1, 1^n, s_1)}_{\mathcal{R}^*(1^n, r)}$$

A commitment scheme has to satisfy the binding property, i.e., each$(r, \bar{m})$ has to be an unambiguous receiver's view except with a negligible fraction.

Thus **Hiding** of a Bit Commitment Scheme is **computational** & **Binding** is **information-theoretic**.

12.3 Examples of Bit-Commitment Scheme

Construction: *Let $f : \{0,1\}^* \to \{0,1\}^*$ be a 1-1 one-way function, and let $b : \{0,1\}^* \to \{0,1\}$ be a hard-core predicate of f.*

- **Commit phase:** *To commit to value $v \in \{0,1\}$ (using security parameter n), the sender uniformly selects $s \in \{0,1\}^n$ and sends the pair $(\alpha, \sigma) \equiv (f(s), b(s) \oplus v)$ to the receiver.*
- **Reveal phase:** *In the reveal phase, the sender reveals the bit v and the string s used in the commit phase. The receiver accepts the value v if $f(s) = \alpha$ and $b(s) \oplus v = \sigma$, where (α, σ) is the receiver's view of the commit phase.*

Exercise 5. *Prove that the above construction constitutes a bit-commitment scheme.*

Exercise 6. *Can you design a bit-commitment scheme using an one-way function (not necessarily 1-1) ?*

Answer: *The answer is* ***Yes****.*

Hint: *We can construct a pseudo random generator $G : \{0,1\}^n \to \{0,1\}^{3n}$. Suppose assume that G has the following property:*

$$\{G(s) : s \in \{0,1\}^n\} \cap \{G(s) \oplus 1^{3n} : s \in \{0,1\}^n\} = \phi$$

Think about it, we will discuss it in the next class.

13 Constructions of BCS

We will continue our discussion from the last class.

13.1 Bit-commitment scheme using one-way function

According to our previous discussions, we can construct a pseudo random generator $G : \{0,1\}^n \to \{0,1\}^{3n}$. Suppose assume that G has the following property:

$$\{G(s) : s \in \{0,1\}^n\} \cap \{G(s) \oplus 1^{3n} : s \in \{0,1\}^n\} = \phi$$

- **Commit phase:**

Sender(1^n)		Receiver(1^n)
chooses a bit $b \in \{0,1\}$		
chooses $s \xleftarrow{\$} \{0,1\}^n$		
if $b = 0$ sets $\alpha = G(s)$		
if $b = 1$ sets $\alpha = G(s) \oplus 1^{3n}$		
	$\xrightarrow{\alpha}$	
		stores α

- **Reveal phase:** *Sender sends s, used in the commit phase and receiver accepts the bit 0 if $G(s) = \alpha$, and accepts the bit 1 if $G(s) \oplus 1^{3n} = \alpha$.*

Exercise 7. *Check that the above described construction consists a bit-commitment scheme.*

But the above given property of G may not always hold for any pseudorandom generator.

Definition 51. *A string* $r \in \{0,1\}^{3n}$ *is called* ***'good'*** *if*

$$\{G(s) : s \in \{0,1\}^n\} \cap \{G(s) \oplus r : s \in \{0,1\}^n\} = \phi$$

otherwise we call that r *is* ***'bad'****.*

Claim 1. *Probability of selecting* 'good' r is at least non-negligible.

Proof. r is **'bad'** $\iff \exists\ s_1\ \&\ s_2 \in \{0,1\}^n$ such that $G(s_1) = G(s_2) \oplus r \iff r = G(s_1) \oplus G(s_2)$. So number of **'bad'** r is at most $\binom{2^n}{2} < 2^{2n}$. So number of **'good'** r is at least $(2^{3n} - 2^{2n})$. Hence choosing a **'good'** r from $\{0,1\}^{3n}$ is

$$\underset{r \leftarrow \{0,1\}^{3n}}{Pr}[r \text{ is } \mathbf{good}] \geq \left(1 - \frac{1}{2^n}\right)$$

□

13.2 Bit-commitment under General Assumptions

Construction: *Let* $G : \{0,1\}^* \to \{0,1\}^*$ *be a pseudorandom generator such that* $|G(s)| = 3.|s|$ *for all* $s \in \{0,1\}^*$.

- **Commit phase:**

Sender(1^n)		Receiver(1^n)
		selects $r \xleftarrow{\$} \{0,1\}^{3n}$
	$\xleftarrow{\quad r \quad}$	
chooses a bit $b \in \{0,1\}$		
chooses $s \xleftarrow{\$} \{0,1\}^n$		
if $b = 0$ sets $\alpha = G(s)$		
if $b = 1$ sets $\alpha = G(s) \oplus r$	$\xrightarrow{\quad \alpha \quad}$	
		stores α

- **Reveal phase:** *Sender sends* s *which was used in the commit phase. Receiver accepts the bit* 0 *if* $G(s) = \alpha$ *and accepts the bit* 1 *if* $G(s) \oplus r = \alpha$.

Exercise 8. *Use above claim 1 & property of pseudo random generator to show that the above described construction consists a bit-commitment scheme.*

Exercise 9. *What will happen if the sender choose r? - Explain.*

Proof. If the sender chooses r then he can chooses r such that $r \in \mathsf{bad}$, which was described in claim 1, i.e., $\exists\ s_1 \neq s_2 \in \{0,1\}^n$ such that $G(s_1) = G(s_2) \oplus r$. So sender can use s_1 and bit 0 in commit phase and use s_2 in reveal phase to convince the receiver that in commit phase he chooses the bit 1. Receiver will always convince since $G(s_1) = G(s_2) \oplus r$. □

Exercise 10. *How can you extend the above BCS protocol for committing a bit-string of length "2"?*

Proof. **Method 1:** One way we can extend the above BCS protocol by bit-by-bit composition. **Method 2:**Let $G : \{0,1\}^* \to \{0,1\}^*$ be a pseudorandom generator such that $|G(s)| = 5.|s|$ for all $s \in \{0,1\}^*$.

Consider the finite field $\mathcal{F}$ of cardinality 2^{5n}. The elements of this field are naturally represented as $5n$-bit string. Moreover, both the addition and subtraction in such representation coincide with XOR-operation $\oplus$ (because the field has characteristic 2). Let $\cdot$ now denote multiplication, and interpret every string $m \in \{0,1\}^2$ as $0^{5n-2} \circ m \in \{0,1\}^{5n}$, so we can view m as a member of $\mathcal{F}$. Now we can directly extend our scheme as follows:

- **Commit phase:**

Sender(1^n)		Receiver(1^n)
		selects $r \xleftarrow{\$} \{0,1\}^{5n}$
	$\xleftarrow{\quad r \quad}$	
chooses a bit $b \in \{0,1\}$		
chooses $s \xleftarrow{\$} \{0,1\}^n$		
for $m \in \{0,1\}^2$,		
$\alpha = G(s) \oplus (m \cdot r)$	$\xrightarrow{\quad \alpha \quad}$	
		stores α

- **Reveal phase:** *Sender sends s and m which was used in the commit phase. Receiver accepts m if and only if $\alpha = G(s) \oplus (m \cdot r)$ and otherwise returns $\perp$.*

Hiding can be proven easily, $G(s)$ plus any fixed string looks pseudorandom. As for binding in order for $G(s_0) \oplus (m_0 \cdot r) = G(s_1) \oplus (m_1 \cdot r)$, where $m_0 \neq m_1$, we must have

$$r = (G(s_0) \oplus G(S_1)) \cdot (m_0 \oplus m_1)^{-1}$$

where the inverse is taken over the field $\mathcal{F}$. There are at most $\binom{2^n}{2}\binom{2^2}{2}$ values for the quantity $(G(s_0) \oplus G(S_1)) \cdot (m_0 \oplus m_1)^{-1}$, so random $r \in \{0,1\}^{5n}$ can be of the above form with probability at most $\frac{\binom{2^n}{2}\binom{2^2}{2}}{2^{5n}} < \frac{2^{2n}2^4}{2^{5n}} \leq \frac{1}{2^{3n-4}} = \mathsf{negl}(n)$.

□

For further reading:

- Polynomial Commitment Scheme
- Vector Commitment Scheme

14 ZKP for any $\mathcal{NP}$ Languages

In this lecture we will give a construction of ZKP of any $\mathcal{NP}$-language using BCS. First we will construct a ZKP for a $\mathcal{NP}$-Complete problem called *"Graph 3-Coloring problem"*. Using this construction we will try to construct ZKP for any $\mathcal{NP}$-language.

14.1 Zero Knowledge Proof for G3C

Language of "Graph 3-Coloring problem":

$$\mathcal{L} = \{\langle G \rangle : \exists\ \phi : V(G) \to \{1,2,3\},\ \phi(u) \neq \phi(v) \iff (u,v) \in E(G)\}$$

Zero Knowledge Protocol:
Let, G be a graph on n vertices and define $V = \{v_1, v_2, \ldots, v_n\}$ be the set of vertices & $E = \{e_{i,j} : \exists$ edge between $v_i, v_j\}$ and ϕ be a proper 3-coloring of the graph G.

Prover(G, ϕ)		Verifier(G)
chooses $\psi \xleftarrow{\$} \text{Perm}(\{1,2,3\})$		
computes $\psi \circ \phi(v)\ \forall\ v \in V(G)$		
commits $\psi \circ \phi(v)$ for each $v \in V(G)$ let $C_v = C_{S_v}(\psi \circ \phi(v))$		
		picks an edge $(u, v) \xleftarrow{\$} E(G)$
	$\xleftarrow{(u,v)}$	
	$\xrightarrow{(S_u, S_v, \psi \circ \phi(u), \psi \circ \phi(v))}$	
		checks $\psi \circ \phi(u) = \psi \circ \phi(v)$ if so it immediately **'aborts'**
		otherwise, it returns **'Accept'** if $C_u = C_{S_u}(\psi \circ \phi(u))$ & $C_v = C_{S_v}(\psi \circ \phi(v))$ else returns **'Reject'**

Theorem 21. *The above protocol satisfies* ***completeness, soundness*** *with* $\frac{1}{|E|}$*,* ***and zero-knowledge.***

Proof. We will prove the theorem part by part.

- **Completeness:** If witness ϕ provides a valid 3-coloring of the graph G. Then the Prover can commit to the colors such that regardless of what edge the Verifier chooses the Verifier will see that $C_u \neq C_v$ and will returns **'Accept'.**
- **Soundness:** We need to show that if ϕ provides an invalid 3-coloring of G, then the

  $$Pr[\text{ Verifier returns accept}] \leq negl(n)$$

  Since ϕ is an invalid 3-coloring of G, then there exists edge (u, v) such that $C_u = C_v$. Thus

  $$Pr[\text{Verifier returns reject}] \geq Pr[\text{Verifier picks } (u, v)] = \frac{1}{|E|}$$

  We can just repeat the protocol sequentially to improve the soundness.
  After repeating the protocol for k times where $k \gg E$ then

  $$Pr[\text{ Verifier returns accept}] \leq \left(1 - \frac{1}{|E|}\right)^k \leq \left(1 - \frac{1}{|E|}\right)^{kE} \leq e^{-k} \leq negl(n)$$

- **Zero-Knowledge:** To prove Zero-Knowledge we construct a simulator $\mathcal{S}$ which has the code of V^*, the Verifier, and works as follows:
  - **Step 1:** Chooses random $e'_{i,j} = (v'_i, v'_j)$ and commits to different random colors for C'_i, C'_j. For all other vertices, v_k where $k \neq i, j$. Let $C_k = 0$, the zero string.
  - **Step 2:** Sends first message to V^* and gets $e_{i,j}$ from V^*.
  - **Step 3:** If $e_{i,j} = e'_{i,j}$ opens $C_i\prime, C_j\prime$. Else goes to Step 1.

Let $\mathcal{H}_0$ be the description of the protocol, $\mathcal{H}_3$ be the description of simulator $\mathcal{S}$. Using hybrid lemma we will show that these two transcripts are **indistinguishable**.

- $\mathcal{H}_0$ Algorithm has the correct witness ϕ and code of V^*. $\mathcal{S}_0$ acts as an honest Prover and interacts with V^* which means :
  - **Step 1:** Commits colors of vertices and computes first message honestly with witness ϕ.
  - **Step 2:** Gets $e_{i,j}$.
  - **Step 3:** Opens C_i, C_j and if $C_i \neq C_j$ returns **accept**, else returns **reject**.
    Outputs the transcript τ_0. τ_0 has the same distribution as in the real protocol.
- $\mathcal{H}_1$ Algorithm $\mathcal{S}_1$ has the correct witness ϕ and code of V^*. $\mathcal{S}_1$ guesses a random edge $e'_{i,j}$. $\mathcal{S}_1$ acts as an honest Prover and interacts with V^* which means :
  - **Step 1:** Commits colors of vertices and computes first message honestly with witness ϕ.
  - **Step 2:** Gets $e_{i,j}$.
  - **Step 3:** Now if $e_{i,j} \neq e'_{i,j}$ goes to Step 1. Else opens C_i, C_j and if $C_i \neq C_j$ returns **accept**, else returns **reject**.
    Outputs the transcript τ_1.
- $\mathcal{H}_2$ Algorithm $\mathcal{S}_2$ has the correctness witness ϕ and code of V^*. $\mathcal{S}_2$ guesses a random edge $e'_{i,j}$. $\mathcal{S}_2$ computes the first message using $e'_{i,j}$ which means :
  - **Step 1:** $\mathcal{S}_2$ commits the coloring of every vertex to be zero for all C'_k where $k \neq i, j$. C'_i, C'_j are still computed honestly using ϕ.
  - **Step 2:** Gets $e_{i,j}$.
  - **Step 3:** Now if $e_{i,j} \neq e'_{i,j}$ goes to Step 1. Else opens C'_i, C'_j and if $C'_i \neq C'_j$ returns **accept**, else returns **reject**.
    Outputs the transcript τ_2.
- $\mathcal{H}_3$ Algorithm $\mathcal{S}$ is the simulator with code of V^*. $\mathcal{S}_2$ guesses a random edge $e'_{i,j}$.
  - **Step 1:** $\mathcal{S}$ commits the coloring of every vertex to be zero for all C'_k where $k \neq i, j$. C'_i, C'_j are computed randomly.
  - **Step 2:** Gets $e_{i,j}$.
  - **Step 3:** Now if $e_{i,j} \neq e'_{i,j}$ goes to Step 1. Else opens C'_i, C'_j and if $C'_i \neq C'_j$ returns **accept**, else returns **reject**.
    Outputs the transcript τ_3.

Exercise 11. *Prove that $\mathcal{H}_i$ & $\mathcal{H}_{i+1}$ are indistinguishable $\forall\ i \in \{0, 1, 2\}$.*

□

14.2 Construction of ZKP for any $\mathcal{NP}$ Languages

We will show that, if one-way function exists , then any $\mathcal{L} \in \mathcal{NP}$ has a **Zero-Knowledge** proof.

Before presenting zero-knowledge proof systems for every language in $\mathcal{NP}$, let us recall some conventions and facts concerning $\mathcal{NP}$. We first recall that every language $\mathcal{L} \in \mathcal{NP}$ is *characterized* by a binary relation $R_{\mathcal{L}}$ satisfying the following :

- There exists a polynomial $p(\cdot)$ such that for every $(x, y) \in R_{\mathcal{L}}$, it holds that $|y| \leq p(|x|)$.
- There exists a polynomial-time algorithm for deciding membership in $R_{\mathcal{L}}$.

- $\mathcal{L} = \{x \in \{0,1\}^* : \exists\ y \in \{0,1\}^* \text{ s.t. } (x,y) \in R_{\mathcal{L}}\}$

Since $G3C$ is $\mathcal{NP}$-complete, we know that $\mathcal{L}$ is polynomial-time-reducible to $G3C$. Namely. there exists a polynomial-time-computable function f such that $x \in \mathcal{L}$ iff $f_{\mathcal{L}}(x) \in G3C$. $f_{\mathcal{L}}$ has the following additional property :

- *There exists a polynomial-time-computable function, denoted $g_{\mathcal{L}}$, such that for every $(x,y) \in R_{\mathcal{L}}$, it holds that $g_{\mathcal{L}}(x,y)$ is a 3-coloring of $f_{\mathcal{L}}(x)$.*

Construction (A Zero-Knowledge Proof for a Language $\mathcal{L} \in \mathcal{NP}$):
For any string $x \in \mathcal{L}$,

Prover(x)	Verifier(x)
Secret witness y	
Computes $H = f_{\mathcal{L}}(x)$	Computes $H = f_{\mathcal{L}}(x)$
Computes $\psi = g_{\mathcal{L}}(x,y)$	

Prover and Verifier invoke the ZKP for $G3C$

Proposition 22. *Suppose that the sub-protocol used in the last step of this **Construction** is indeed an auxiliary-input zero-knowledge proof for $G3C$. Then **Construction** constitutes an auxiliary-input zero-knowledge proof for $\mathcal{L}$.*

Example 5. *Let v denote the value to which $\mathcal{S}$ commits, let s denote the randomness it uses in the commitment phase, and let $c := C_s(v)$ be the resulting commitment. Suppose that $\mathcal{S}$ wants to prove to $\mathcal{R}$ that c is a commitment to a value greater than u, i.e. $\mathcal{S}$ wants to prove that $\exists\ v$ and s such that $c = C_s(v)$ and $v > u$, where c and u are known to $\mathcal{R}$.*

$$\mathcal{L} := \{(c,u) : \exists\ v,s : c = C_s(v) \text{ and } v > u\}$$

$\mathcal{S}$ knows the witness (v,s). Clearly, $\mathcal{L} \in \mathcal{NP}$ with $\mathcal{NP}$-witness for $(c,u) \in \mathcal{L}$ is a pair (v,s). So we can construct a ZKP for this $\mathcal{L}$.

Theorem 23. *Any $\mathcal{L} \in \mathcal{IP}$ has a Zero-knowledge proof(w.r.t. public coin), a.k.a. **'Arthur-Merlin proof System'**.*

References

[1] R. Barua. Formal languages and automata theory iv. URL https://www.tcgcrest.org/wp-content/uploads/2023/05/Automata_4.pdf.

[2] O. Goldreich. *The Foundations of Cryptography - Volume 1.* Cambridge University Press, 2001. ISBN 9780511546891.

[3] A. Jha and M. Nandi. A survey on applications of h-technique: Revisiting security analysis of prp and prf. *Entropy*, 24(4):462, 2022.

[4] J. Katz and Y. Lindell. *Introduction to modern cryptography.* CRC press, 2020.

[5] M. Luby and C. Rackoff. How to construct pseudo-random permutations from pseudo-random functions (abstract). volume 17, page 447, 08 1985. ISBN 978-3-540-16463-0. doi: 10.1007/3-540-39799-X_34.