Definition (Negligible Functions)

A function $\epsilon : \mathbb{N} \rightarrow [0, 1]$ is called negligible function if for every $c$ there exists a large enough $n_0$ such $\epsilon(n) < n^{-c}$ for all $n > n_0$. 

Definition (Pseudorandom Generators)

Let $G : \{0, 1\}^* \rightarrow \{0, 1\}^*$ be a polynomial-time computable function. Let $\ell : \mathbb{N} \rightarrow \mathbb{N}$ be a polynomial-time computable function such that $\ell(n) > n$ for every $n$. We say that $G$ is a secure pseudorandom generator of stretch $\ell(n)$, iff 

$$|G(x)| = \ell(|x|)$$

for every $x \in \{0, 1\}^*$ and for every probabilistic polynomial-time algorithm $A$, there exists a negligible function $\epsilon : \mathbb{N} \rightarrow [0, 1]$ such that 

$$|\Pr(A(G(U_n)) = 1) - \Pr(A(U_\ell(n)) = 1)| < \epsilon(n)$$

This is equivalent to saying that no adversary $A$ can distinguish between a string generated by generator $G$ and a random string of length $\ell(n)$ in polynomial time.
Pseudorandom Generators

Definition (Negligible Functions)
A function $\epsilon : \mathbb{N} \rightarrow [0, 1]$ is called negligible function if for every $c$ there exists a large enough $n_0$ such $\epsilon(n) < n^{-c}$ for all $n > n_0$.

Definition (Pseudorandom Generators)
Let $G : \{0, 1\}^* \rightarrow \{0, 1\}^*$ be a polynomial-time computable function. Let $\ell : \mathbb{N} \rightarrow \mathbb{N}$ be a polynomial-time computable function such that $\ell(n) > n$ for every $n$. We say that $G$ is a secure pseudorandom generator of stretch $\ell(n)$, iff $|G(x)| = \ell(|x|)$ for every $x \in \{0, 1\}^*$ and for every probabilistic polynomial-time algorithm $A$, there exists a negligible function $\epsilon : \mathbb{N} \rightarrow [0, 1]$ such that

$$|\Pr(A(G(U_n)) = 1) - \Pr(A(U_{\ell(n)}) = 1)| < \epsilon(n)$$
Pseudorandom Generators

**Definition (Negligible Functions)**

A function \( \epsilon : \mathbb{N} \rightarrow [0, 1] \) is called negligible function if for every \( c \) there exists a large enough \( n_0 \) such \( \epsilon(n) < n^{-c} \) for all \( n > n_0 \).

**Definition (Pseudorandom Generators)**

Let \( G : \{0, 1\}^* \rightarrow \{0, 1\}^* \) be a polynomial-time computable function. Let \( \ell : \mathbb{N} \rightarrow \mathbb{N} \) be a polynomial-time computable function such that \( \ell(n) > n \) for every \( n \). We say that \( G \) is a secure pseudorandom generator of stretch \( \ell(n) \), iff

\[
|G(x)| = \ell(|x|) \text{ for every } x \in \{0, 1\}^* \text{ and for every probabilistic polynomial-time algorithm } A, \text{ there exists a negligible function } \epsilon : \mathbb{N} \rightarrow [0, 1] \text{ such that }
\]

\[
|Pr(A(G(U_n)) = 1) - Pr(A(U_{\ell(n)}) = 1)| < \epsilon(n)
\]

This is equivalent to saying that no adversary \( A \) can distinguish in between a string generated by generator \( G \) and a random string of length \( \ell(n) \) in polynomial time.
Definition (Unpredictable Functions)

Let $G : \{0, 1\}^* \rightarrow \{0, 1\}^*$ be a polynomial-time computable function with stretch $\ell(n)$. We call that $G$ unpredictable iff for every probabilistic polynomial-time algorithm $B$, there exists a negligible function $\epsilon : \mathbb{N} \rightarrow [0, 1]$ such that

$$Pr(B(1^n, y_1, \ldots, y_{i-1}) = y_i) \leq \frac{1}{2} + \epsilon(n)$$
Definition (Unpredictable Functions)

Let $G : \{0, 1\}^* \rightarrow \{0, 1\}^*$ be a polynomial-time computable function with stretch $\ell(n)$. We call that $G$ unpredictable iff for every probabilistic polynomial-time algorithm $B$, there exists a negligible function $\epsilon : \mathbb{N} \rightarrow [0, 1]$ such that

$$\Pr(B(1^n, y_1, \ldots, y_{i-1}) = y_i) \leq \frac{1}{2} + \epsilon(n)$$

Theorem (Unpredictability implies Pseudorandomness)

For every probabilistic polynomial-time algorithm $A$, there exists a probabilistic polynomial-time algorithm $B$ such that for every $n \in \mathbb{N}$ and $\epsilon(n) > 0$, if $\Pr[A(G(U_n)) = 1] - \Pr[A(U_{\ell(n)}) = 1] \geq \epsilon(n)$, then

$$\Pr[B(1^n, y_1, \ldots, y_{i-1}) = y_i] \geq \frac{1}{2} + \frac{\epsilon(n)}{\ell(n)}$$
Description of predictor algorithm $B$: On input $1^n$, $i \in [\ell(n)]$ and $y_1, \ldots, y_{i-1}$, Algorithm $B$ will choose $z_i, \ldots, z_{\ell(n)}$ independently at random, and compute $a = A(y_1, \ldots, y_{i-1}, z_i, \ldots, z_{\ell(n)})$. If $a = 1$ then $B$ surmises its guess for $z_i$ is correct and outputs $z_i$ otherwise it outputs $1 - z_i$. 
Unpredictability implies Pseudorandomness

Description of predictor algorithm $B$ : On input $1^n$, $i \in [\ell(n)]$ and $y_1, \ldots, y_{i-1}$, Algorithm $B$ will choose $z_i, \ldots, z_{\ell(n)}$ independently at random, and compute $a = A(y_1, \ldots, y_{i-1}, z_i, \ldots, z_{\ell(n)})$. If $a = 1$ then $B$ surmises its guess for $z_i$ is correct and outputs $z_i$ otherwise it outputs $1 - z_i$.

Distribution $D_i$ is defined as : Choose $y \in G(U_n)$ and $z \in \{0, 1\}^{\ell(n)}$ randomly, output $y_1, \ldots, y_{i-1}, z_i, \ldots, z_{\ell(n)}$.
Unpredictability implies Pseudorandomness

Description of predictor algorithm $B$: On input $1^n$, $i \in [\ell(n)]$ and $y_1, \ldots, y_{i-1}$, Algorithm B will choose $z_i, \ldots, z_{\ell(n)}$ independently at random, and compute $a = A(y_1, \ldots, y_{i-1}, z_i, \ldots, z_{\ell(n)})$. If $a = 1$ then $B$ surmises its guess for $z_i$ is correct and outputs $z_i$ otherwise it outputs $1 - z_i$.

Distribution $D_i$ is defined as: Choose $y \in G(U_n)$ and $z \in \{0, 1\}^{\ell(n)}$ randomly, output $y_1, \ldots, y_{i-1}, z_i, \ldots, z_{\ell(n)}$

Let $p_i = Pr[A(D_i) = 1]$. Note $p_{\ell(n)} - p_0 \geq \epsilon(n)$
Unpredictability implies Pseudorandomness

Description of predictor algorithm $B$: On input $1^n$, $i \in [\ell(n)]$ and $y_1, \ldots, y_{i-1}$, Algorithm B will choose $z_i, \ldots, z_{\ell(n)}$ independently at random, and compute $a = A(y_1, \ldots, y_{i-1}, z_i, \ldots, z_{\ell(n)})$. If $a = 1$ then $B$ surmises its guess for $z_i$ is correct and outputs $z_i$ otherwise it outputs $1 - z_i$.

Distribution $D_i$ is defined as: Choose $y \in G(U_n)$ and $z \in \{0, 1\}^{\ell(n)}$ randomly, output $y_1, \ldots, y_{i-1}, z_i, \ldots, z_{\ell(n)}$

Let $p_i = Pr[A(D_i) = 1]$. Note $p_{\ell(n)} - p_0 \geq \epsilon(n)$

This implies $E_i[p_i - p_{i-1}] \geq \frac{\epsilon(n)}{\ell(n)}$
Unpredictability implies Pseudorandomness

Description of predictor algorithm $B$: On input $1^n, i \in [\ell(n)]$ and $y_1, \ldots, y_{i-1}$, Algorithm B will choose $z_i, \ldots, z_{\ell(n)}$ independently at random, and compute $a = A(y_1, \ldots, y_{i-1}, z_i, \ldots, z_{\ell(n)})$. If $a = 1$ then $B$ surmises its guess for $z_i$ is correct and outputs $z_i$ otherwise it outputs $1 - z_i$.

Distribution $D_i$ is defined as: Choose $y \in G(U_n)$ and $z \in \{0, 1\}^{\ell(n)}$ randomly, output $y_1, \ldots, y_{i-1}, z_i, \ldots, z_{\ell(n)}$

Let $p_i = Pr[A(D_i) = 1]$. Note $p_{\ell(n)} - p_0 \geq \epsilon(n)$

This implies $E_i[p_i - p_{i-1}] \geq \frac{\epsilon(n)}{\ell(n)}$

We will prove that for every $i$:

$$Pr_y[B(1^n, y_1, \ldots, y_{i-1}) = y_i] = \frac{1}{2} + p_i - p_{i-1}$$
Unpredictability implies Pseudorandomness

Description of predictor algorithm $B$: On input $1^n$, $i \in [\ell(n)]$ and $y_1, \ldots, y_{i-1}$, Algorithm B will choose $z_i, \ldots, z_{\ell(n)}$ independently at random, and compute $a = A(y_1, \ldots, y_{i-1}, z_i, \ldots, z_{\ell(n)})$. If $a = 1$ then $B$ surmises its guess for $z_i$ is correct and outputs $z_i$ otherwise it outputs $1 - z_i$.

Distribution $D_i$ is defined as: Choose $y \in G(U_n)$ and $z \in \{0, 1\}^{\ell(n)}$ randomly, output $y_1, \ldots, y_{i-1}, z_i, \ldots, z_{\ell}$

Let $p_i = Pr[A(D_i) = 1]$. Note $p_{\ell(n)} - p_0 \geq \epsilon(n)$

This implies $E_i[p_i - p_{i-1}] \geq \frac{\epsilon(n)}{\ell(n)}$

We will prove that for every $i$:

$$Pr_y[B(1^n, y_1, \ldots, y_{i-1}) = y_i] = \frac{1}{2} + p_i - p_{i-1}$$

This can be done by following two equations:

$$Pr_y[B(1^n, y_1, \ldots, y_{i-1}) = y_i] = \frac{1}{2}(Pr[a = 1|z_i = y_i] + 1 - Pr[a = 1|z_i = 1 - y_i])$$
Unpredictability implies Pseudorandomness

Description of predictor algorithm $B$: On input $1^n$, $i \in [\ell(n)]$ and $y_1, \ldots, y_{i-1}$, Algorithm $B$ will choose $z_i, \ldots, z_{\ell(n)}$ independently at random, and compute $a = A(y_1, \ldots, y_{i-1}, z_i, \ldots, z_{\ell(n)})$. If $a = 1$ then $B$ surmises its guess for $z_i$ is correct and outputs $z_i$ otherwise it outputs $1 - z_i$.

Distribution $D_i$ is defined as: Choose $y \in G(U_n)$ and $z \in \{0, 1\}^{\ell(n)}$ randomly, output $y_1, \ldots, y_{i-1}, z_i, \ldots, z_{\ell}$

Let $p_i = Pr[A(D_i) = 1]$. Note $p_{\ell(n)} - p_0 \geq \epsilon(n)$

This implies $E_i[p_i - p_{i-1}] \geq \frac{\epsilon(n)}{\ell(n)}$

We will prove that for every $i$:

$$Pr_y[B(1^n, y_1, \ldots, y_{i-1}) = y_i] = \frac{1}{2} + p_i - p_{i-1}$$

This can be done by following two equations:

$$Pr_y[B(1^n, y_1, \ldots, y_{i-1}) = y_i] = \frac{1}{2}(Pr[a = 1|z_i = y_i] + 1 - Pr[a = 1|z_i = 1 - y_i])$$

$$p_{i-1} = \frac{1}{2}(Pr[a = 1|z_i = y_i] + Pr[a = 1|z_i = 1 - y_i])$$
Unpredictability implies Pseudorandomness

Description of predictor algorithm $B$: On input $1^n$, $i \in [\ell(n)]$ and $y_1, \ldots, y_{i-1}$, Algorithm $B$ will choose $z_i, \ldots, z_{\ell(n)}$ independently at random, and compute $a = A(y_1, \ldots, y_{i-1}, z_i, \ldots, z_{\ell(n)})$. If $a = 1$ then $B$ surmises its guess for $z_i$ is correct and outputs $z_i$ otherwise it outputs $1 - z_i$.

Distribution $D_i$ is defined as: Choose $y \in G(U_n)$ and $z \in \{0, 1\}^{\ell(n)}$ randomly, output $y_1, \ldots, y_{i-1}, z_i, \ldots, z_{\ell(n)}$.

Let $p_i = Pr[A(D_i) = 1]$. Note $p_{\ell(n)} - p_0 \geq \epsilon(n)$

This implies $E_i[p_i - p_{i-1}] \geq \frac{\epsilon(n)}{\ell(n)}$

We will prove that for every $i$:

$$Pr_y[B(1^n, y_1, \ldots, y_{i-1}) = y_i] = \frac{1}{2} + p_i - p_{i-1}$$

This can be done by following two equations:

$$Pr_y[B(1^n, y_1, \ldots, y_{i-1}) = y_i] = \frac{1}{2} (Pr[a = 1|z_i = y_i] + 1 - Pr[a = 1|z_i = 1 - y_i])$$

$$p_{i-1} = \frac{1}{2} (Pr[a = 1|z_i = y_i] + Pr[a = 1|z_i = 1 - y_i])$$

where $p_i = Pr[a = 1|z_i = y_i]$
The Goldreich-Levin Theorem

To get the secure pseudorandom generator with stretch $n^c$, a crucial step is to get secure pseudorandom generator with stretch $l(n) = n + 1$

**Theorem**

Suppose that $f : \{0, 1\}^* \rightarrow \{0, 1\}$ is a one-way function and $|f(x)| = |x|$ for every $x \in \{0, 1\}^*$. Then, for every probabilistic polynomial time algorithm $A$ there is a negligible function $\epsilon : N \rightarrow [0, 1]$ such that

$$
\Pr_{x, r \in \{0, 1\}^n}[A(f(x), r) = x \odot r] \leq \frac{1}{2} + \epsilon(n),
$$

where $x \odot r$ is defined to be $\sum_{i=1}^{n} x_i r_i \pmod{2}$

We produce the generator $G(x, r) = f(x), r, x \odot r$. It clearly extends the input by 1. If it is not secure pseudorandom, by the previous theorem, there will be a predictor $B$

But, $B$ will be an exact contradiction to the Goldreich Levin Theorem.
A trivial case

The proof is by contradiction. We will use $A$ to show a probabilistic polynomial-time algorithm $B$ that inverts the permutation $f$, in contradiction to the assumption that it is one way.
A trivial case

The proof is by contradiction. We will use $A$ to show a probabilistic polynomial-time algorithm $B$ that inverts the permutation $f$, in contradiction to the assumption that it is one way.

Consider the case when

$$Pr_{x,r \in \{0,1\}^n}[A(f(x), r) = x \odot r] = 1$$
A trivial case

- The proof is by contradiction. We will use $A$ to show a probabilistic polynomial-time algorithm $B$ that inverts the permutation $f$, in contradiction to the assumption that it is one way.

- Consider the case when
  \[ \Pr_{x,r \in \{0,1\}^n} [A(f(x), r) = x \odot r] = 1 \]

- Just run $A(f(x), e^1), \ldots, A(f(x), e^n)$ where $e^i$ is the string whose $i^{th}$ coordinate is equal to one and all the other coordinates are zero.
Recovery for Success probability 0.9

- The previous approach will not work. (why?)
The previous approach will not work. (why?)

Observation 1: If we choose \( r \in \{0, 1\}^n \) then the string \( r \oplus e^i \) is also uniformly distributed.

Observation 2: \( \Pr_{r \in \{0, 1\}^n} [A(f(x), r) \neq x \circ r \lor A(f(x), r \oplus e^i) \neq x \circ (r \oplus e^i)] \leq 0.2 \)

\[ z = A(f(x), r) \text{ and } z' = A(f(x), r \oplus e^i), \text{ then } z \oplus z' = x^i \] with probability at least 0.8
Recovery for Success probability 0.9

- The previous approach will not work. (why?)
- Observation 1: If we choose $r \in \{0, 1\}^n$ then the string $r \oplus e^i$ is also uniformly distributed.
- Observation 2:
  \[ Pr_r[A(f(x), r) \neq x \odot r \text{ OR } A(f(x), r \oplus e^i) \neq x \odot (r \oplus e^i)] \leq 0.2 \]
The previous approach will not work. (why?)

Observation 1: If we choose $r \in \mathbb{R} \{0, 1\}^n$ then the string $r \oplus e^i$ is also uniformly distributed.

Observation 2:

\[ Pr_r[A(f(x), r) \neq x \odot r \lor A(f(x), r \oplus e^i) \neq x \odot (r \oplus e^i)] \leq 0.2 \]

If $z = A(f(x), r)$ and $z' = A(f(x), r \oplus e^i)$, then $z \oplus z' = x_i$ with probability at least 0.8.
1. Choose $r_1, r_2, \ldots, r_m$ independently at random from $\{0, 1\}^n$.
2. For every $i \in [n]$:
   - Compute the values $z_1 = A(f(x), r_1), z'_1 = A(f(x), r_1 \oplus e^i), \ldots, z_m = A(f(x), r_m), z'_1 = A(f(x), r_m \oplus e^i)$.
   - Guess that $x_i$ is the majority value among $\{z_j \oplus z'_j\}_{j \in [m]}$. 
Define $Z_j = A(f(x), r_j) = x \odot r_j$ AND $A(f(x), r_j \oplus e^i) = x \odot (r_j \oplus e^i)$

We know, $Z_1, Z_2, .., Z_m$ are independent and $E[Z_j] \geq 0.8$ for all $j \in [m]$
Define $Z_j = A(f(x), r_j) = x \odot r_j$ AND $A(f(x), r_j \oplus e^i) = x \odot (r_j \oplus e^i)$

We know, $Z_1, Z_2, .., Z_m$ are independent and $E[Z_j] \geq 0.8$ for all $j \in [m]

Define $Z = Z_1 + Z_2 + .. + Z_m$. We need to show that $Pr[Z \leq \frac{m}{2}] \leq \frac{1}{10n}$
Define $Z_j = A(f(x), r_j) = x \circ r_j$ AND $A(f(x), r_j \oplus e_i) = x \circ (r_j \oplus e_i)$

We know, $Z_1, Z_2, \ldots, Z_m$ are independent and $E[Z_j] \geq 0.8$ for all $j \in [m]$

Define $Z = Z_1 + Z_2 + \ldots + Z_m$. We need to show that $Pr[Z \leq \frac{m}{2}] \leq \frac{1}{(10n)}$

It suffices to bound $Pr[|Z - E[Z]| \geq 0.3m]$ since $E[Z] \geq 0.8m$
Define $Z_j = A(f(x), r_j) = x \odot r_j$ AND $A(f(x), r_j \oplus e^i) = x \odot (r_j \oplus e^i)$

We know, $Z_1, Z_2, .., Z_m$ are independent and $E[Z_j] \geq 0.8$ for all $j \in [m]$

Define $Z = Z_1 + Z_2 + .. + Z_m$. We need to show that $Pr[Z \leq \frac{m}{2}] \leq \frac{1}{(10n)}$

It suffices to bound $Pr[|Z - E[Z]| \geq 0.3m]$ since $E[Z] \geq 0.8m$

We use Chebychev’s Inequality. Clearly $Var(Z_j) \leq 1$ for all $j$. Thus, $Var(Z) \leq m$.

$$Pr[|Z - E[Z]| \geq 0.3m] \leq \frac{1}{(0.3\sqrt{m})^2}$$

If we take $m = 200n$, then for every $i \in [n]$, the majority value will be correct with probability at least $1 - \frac{1}{(10n)}$
General Case

- The previous approach will work only till 0.75. (why?)
The previous approach will work only till 0.75. (why?)

The previous method works even when we have \( Z_i \)s to be pairwise independent. Because, the only place where we used it is

\[
\text{Var}(\sum_j Z_j) = \sum_j \text{Var}(Z_j).
\]

But, this condition holds even with pairwise independence.
The previous approach will work only till 0.75.(why?)

The previous method works even when we have $Z_i$s to be pairwise independent. Because, the only place where we used it is $\text{Var}(\sum_j Z_j) = \sum_j \text{Var}(Z_j)$. But, this condition holds even with pairwise independence.

Choose $k$ strings $s_1, s_2, .., s_k$ independently at random from $\{0, 1\}^n$

For every $j \in [m]$ we associate a unique nonempty set $T_j \subset [k]$ and define $r_j = \sum_{t \in T_j} s_t \pmod{2}$.

$r_1, r_2, .., r_m$ are pairwise independent.

for every $x \in \{0, 1\}^n$, $x \odot r_j = \sum_{t \in T_j} x \odot s_t$

We can compute $x \odot r_1, x \odot r_2, .., x \odot r_m$ given $x \odot s_1, .., x \odot s_k$

Thus, we can iterate over polynomial number of possibilities of $x \odot s_1, .., x \odot s_k$
Choose an m. Let \( k \) be the smallest such that \( m \leq 2^k - 1 \). Choose \( s_1, s_2, \ldots, s_k \) randomly in \( \{0,1\}^k \). Compute \( r_1, r_2, \ldots, r_m \) as above. For every string \( w \in \{0,1\}^k \) do:

- Run Algorithm B from above assuming \( x \odot s_t = w_t \)
- Guess \( x_i \) to be majority value among \( \{z_j \oplus z_j'\}_{j\in[m]} \).
- Check if the guess satisfies \( f(x) = y \). If so, halt.
In one of the $2^k$ iterations, we will guess the correct values $w_1, w_2, \ldots, w_k$ for $x \odot s_1, \ldots, x \odot s_k$. 

Here, we have $E[Z_j] \geq \frac{1}{2} + \epsilon$.

We take $m = 100n\epsilon^2$ to get the $1 - \frac{1}{(10n)^\epsilon}$ bound.
In one of the $2^k$ iterations, we will guess the correct values $w_1, w_2, \ldots, w_k$ for $x \odot s_1, \ldots, x \odot s_k$.

In this iteration, we use exactly the same ideas from the previous proof. We define $Z_j$ as it is.

Here, we have $E[Z_j] \geq \frac{1}{2} + \epsilon$.

We take $m = \frac{100n}{\epsilon^2}$ to get the $1 - \frac{1}{(10n)}$ bound.
Zero knowledge proof systems are a subclass of Interactive proof systems (Prover - Verifier) in which the Verifier does not learn anything except the truth value of the statement.

Formally, a language $L$ has a zero knowledge proof system if:

- **Completeness:** If $x \in L$ then the verifier accepts with probability 1.

- **Soundness:** If $x \not\in L$ then the verifier accepts with probability at most $\frac{1}{2}$.

- **Perfect Zero Knowledge:** The verifier does not learn anything new from the interaction. Formally, there exists a simulator $S$ which runs in probabilistic polynomial time and whose distribution is same as that of the transcript of interaction.

Interactive Proof system for Graph Non-isomorphism was Perfect Zero Knowledge.
Zero knowledge proof systems are a subclass of Interactive proof systems (Prover - Verifier) in which the Verifier does not learn anything except the truth value of the statement.

**Definition (Zero Knowledge Proofs)**

Formally, a language \( L \) has a zero knowledge proof system if:

- **Completeness**: If \( x \in L \) then the verifier accepts with probability 1.
- **Soundness**: If \( x \notin L \) then the verifier accepts with probability at most \( \frac{1}{2} \).
Zero Knowledge Proofs

Zero knowledge proof systems are a subclass of Interactive proof systems (Prover - Verifier) in which the Verifier does not learn anything except the truth value of the statement.

Definition (Zero Knowledge Proofs)

Formally, a language $L$ has a zero knowledge proof system if:

- **Completeness**: If $x \in L$ then the verifier accepts with probability 1.
- **Soundness**: If $x \notin L$ then the verifier accepts with probability at most $\frac{1}{2}$.
- **Perfect Zero Knowledge**: The verifier does not learn anything new from the interaction. Formally, there exists a simulator $S$ which runs in probabilistic polynomial time and whose distribution is same as that of the transcript of interaction.
Zero Knowledge Proofs

Zero knowledge proof systems are a subclass of Interactive proof systems (Prover - Verifier) in which the Verifier does not learn anything except the truth value of the statement.

Definition (Zero Knowledge Proofs)

Formally, a language $L$ has a zero knowledge proof system if:

- **Completeness**: If $x \in L$ then the verifier accepts with probability 1.
- **Soundness**: If $x \notin L$ then the verifier accepts with probability at most $\frac{1}{2}$.
- **Perfect Zero Knowledge**: The verifier does not learn anything new from the interaction. Formally, there exists a simulator $S$ which runs in probabilistic polynomial time and whose distribution is same as that of the transcript of interaction.

Interactive Proof system for Graph Non-isomorphism was Perfect Zero Knowledge.
Algorithm:

- Public input: A pair of graphs $G_0, G_1$ on $n$ vertices.
Algorithm:

- Public input: A pair of graphs $G_0, G_1$ on $n$ vertices.
- Prover chooses a random permutation $\pi_1 : [n] \rightarrow [n]$ and $b' \in \{0, 1\}$ and sends to the verifier the adjacency matrix of $H = \pi_1(G_{b'}).$
Zero Knowledge Proof for Graph Isomorphism

Algorithm:
- Public input: A pair of graphs \( G_0, G_1 \) on \( n \) vertices.
- Prover chooses a random permutation \( \pi_1 : [n] \rightarrow [n] \) and \( b' \in \{0, 1\} \) and sends to the verifier the adjacency matrix of \( H = \pi_1(G_{b'}) \).
- Verifier randomly chooses \( b \in \{0, 1\} \) and sends to Prover.
- Prover responds with \( \pi_2 = \pi_1 \) if \( b = b' \) and \( \pi_2 = \pi_1 \circ \pi \) if \( b \neq b' \) where \( \pi \) is the isomorphism from \( G_0 \) to \( G_1 \).
- Verifier accepts if \( H = \pi_2(G_b) \).
- If they are isomorphic then the above verifier will always accept (probability = 1).
- If they are non-isomorphic then the above verifier will reject whenever \( b \neq b' \) (probability = \( \frac{1}{2} \)).
Zero Knowledge Proof for Graph Isomorphism

Algorithm:

- Public input: A pair of graphs $G_0, G_1$ on $n$ vertices.
- Prover chooses a random permutation $\pi_1 : [n] \to [n]$ and $b' \in \{0, 1\}$ and sends to the verifier the adjacency matrix of $H = \pi_1(G_{b'})$.
- Verifier randomly chooses $b \in \{0, 1\}$ and sends to Prover.
- Prover responds with $\pi_2 = \pi_1$ if $b = b'$ and $\pi_2 = \pi_1 \circ \pi$ if $b \neq b'$ where $\pi$ is the isomorphism from $G_0$ to $G_1$. 

If they are isomorphic then the above verifier will always accept (probability = 1). If they are non-isomorphic then the above verifier will reject whenever $b \neq b'$ (probability = $\frac{1}{2}$).
Zero Knowledge Proof for Graph Isomorphism

Algorithm:

- Public input: A pair of graphs $G_0$, $G_1$ on $n$ vertices.
- Prover chooses a random permutation $\pi_1 : [n] \rightarrow [n]$ and $b' \in \{0, 1\}$ and sends to the verifier the adjacency matrix of $H = \pi_1(G_{b'})$.
- Verifier randomly chooses $b \in \{0, 1\}$ and sends to Prover.
- Prover responds with $\pi_2 = \pi_1$ if $b = b'$ and $\pi_2 = \pi_1 \circ \pi$ if $b \neq b'$ where $\pi$ is the isomorphism from $G_0$ to $G_1$.
- Verifier accepts if $H = \pi_2(G_b)$.
Algorithm:

- Public input: A pair of graphs $G_0, G_1$ on $n$ vertices.
- Prover chooses a random permutation $\pi_1 : [n] \rightarrow [n]$ and $b' \in \{0, 1\}$ and sends to the verifier the adjacency matrix of $H = \pi_1(G_{b'})$.
- Verifier randomly chooses $b \in \{0, 1\}$ and sends to Prover.
- Prover responds with $\pi_2 = \pi_1$ if $b = b'$ and $\pi_2 = \pi_1 \circ \pi$ if $b \neq b'$ where $\pi$ is the isomorphism from $G_0$ to $G_1$.
- Verifier accepts if $H = \pi_2(G_b)$.

If they are isomorphic then the above verifier will always accept (probability = 1). If they are non-isomorphic then the above verifier will reject whenever $b \neq b'$ (probability = $\frac{1}{2}$).
Let $V$ be some verifier strategy. To show the zero knowledge condition, we use the following simulator $S$: On input a pair of graphs $G_0$, $G_1$, the simulator $S$ chooses $b \in \{0, 1\}$, a random permutation $\pi$ and computes $H = \pi(G_b)$. It then feeds $H$ to the verifier $V$ to obtain its message $b' \in \{0, 1\}$. If $b' = b$ then $S$ sends $\pi$ to $V$ and outputs whatever $V$ outputs.
Let $V$ be some verifier strategy. To show the zero knowledge condition, we use the following simulator $S$: On input a pair of graphs $G_0$, $G_1$, the simulator $S$ chooses $b \in \{0, 1\}$, a random permutation $\pi$ and computes $H = \pi(G_b)$. It then feeds $H$ to the verifier $V$ to obtain its message $b' \in \{0, 1\}$. If $b' = b$ then $S$ sends $\pi$ to $V$ and outputs whatever $V$ outputs. Otherwise the simulator $S$ restarts from the beginning.
Let $V$ be some verifier strategy. To show the zero knowledge condition, we use the following simulator $S$: On input a pair of graphs $G_0$, $G_1$, the simulator $S$ chooses $b \in \{0, 1\}$, a random permutation $\pi$ and computes $H = \pi(G_b)$. It then feeds $H$ to the verifier $V$ to obtain its message $b' \in \{0, 1\}$. If $b' = b$ then $S$ sends $\pi$ to $V$ and outputs whatever $V$ outputs. Otherwise the simulator $S$ restarts from the beginning.

$$E[T(s)] = 2t$$

where $t$ is the time taken for one iteration ($b = b'$)
Theorem

For every $\epsilon > 0$ there exists a $\text{DTIME}(2^{n^\epsilon})$ algorithm for solving $Q \in \text{BPP}$
Applications - Derandomization of BPP

**Theorem**

For every $\epsilon > 0$ there exists a $\text{DTIME}(2^{n^\epsilon})$ algorithm for solving $Q \in \text{BPP}$

**Proof.**

Suppose $Q$ requires $p(n) = n^c$ random bits, then we can solve it by trying all random inputs in $\{0, 1\}^{p(n)}$ and return the majority answer. Hence, For every $Q \in \text{BPP}$ there exists a polynomial $p(n)$ such that there exist an algorithm for solving $Q$ in $\text{DTIME}(2^{p(n)})$.

For every $\epsilon$ there exist a Pseudorandom Generator $G$ such that its stretch is $\ell(n) = n^{c/e}$. Now instead of trying all possibilities of $x \in \{0, 1\}^{p(n)}$ just try all $G(x)$ for $x \in \{0, 1\}^{n^\epsilon}$ and return the majority answer. This should be same as the earlier answer as otherwise this would give us a way to distinguish in between Pseudorandom and random distributions.
Bit Commitment and Coin Tossing over Phone

Problems:

- How can two parties $A$ and $B$ toss a fair random coin over the phone?

Algorithm:

1. $A$ selects strings $x_A$ and $r_A$ of length $n$ and sends $(f_n(x_A), r_A)$ to $B$, where $f_n$ is a one-way permutation. Committed bit is $x_A \otimes r_A$.

2. $B$ selects a bit $b \in \{0, 1\}$ randomly and sends it to $A$.

3. $A$ sends $x_A$ to $B$ and $B$ confirms it by checking the value of $f_n(x_A)$.

$b \oplus (x_A \otimes r_A)$ can be taken as the coin toss.

After the first step $B$ cannot guess $x_A \otimes r_A$ as that would allow it to distinguish between pseudorandom and random distributions. Also $A$ cannot change $x_A$ as $f_n(x_A)$ is fixed and changing it would imply inverting one-way functions.
Problems:

- How can two parties $A$ and $B$ toss a fair random coin over the phone?
- How can $A$ commit to a bit and reveal it at a later point such that $B$ is sure that it is the original bit?
Bit Commitment and Coin Tossing over Phone

Problems:

- How can two parties $A$ and $B$ toss a fair random coin over the phone?
- How can $A$ commit to a bit and reveal it at a later point such that $B$ is sure that it is the original bit?

Algorithm:

- $A$ selects strings $x_A$ and $r_A$ of length $n$ and sends $(f_n(x_A), r_A)$ to $B$, where $f_n$ is a one way permutation. Committed bit is $x_A \odot r_A$. 

$B$ selects a bit $b \in \{0, 1\}$ randomly and sends it to $A$. $A$ sends $x_A$ to $B$ and $B$ confirms it by checking value of $f_n(x_A)$.

$b \oplus (x_A \odot r_A)$ can be taken as the coin toss.

After the first step $B$ cannot guess $x_A \odot r_A$ as that would allow it to differentiate in between pseudorandom and random distributions. Also $A$ cannot change $x_A$ as $f_n(x_A)$ is fixed and changing it would imply inverting one-way functions.
Bit Commitment and Coin Tossing over Phone

Problems:
- How can two parties $A$ and $B$ toss a fair random coin over the phone?
- How can $A$ commit to a bit and reveal it at a later point such that $B$ is sure that it is the original bit?

Algorithm:
- $A$ selects strings $x_A$ and $r_A$ of length $n$ and sends $(f_n(x_A), r_A)$ to $B$, where $f_n$ is a one way permutation. Committed bit is $x_A \odot r_A$.
- $B$ selects a bit $b \in \{0, 1\}$ randomly and sends it to $A$. 

$b \oplus (x_a \odot r_A)$ can be taken as the coin toss. After the first step $B$ cannot guess $x_A \odot r_A$ as that would allow it to differentiate in between pseudorandom and random distributions. Also $A$ cannot change $x_A$ as $f_n(x_A)$ is fixed and changing it would imply inverting one-way functions.
Problems:
- How can two parties $A$ and $B$ toss a fair random coin over the phone?
- How can $A$ commit to a bit and reveal it at a later point such that $B$ is sure that it is the original bit?

Algorithm:
- $A$ selects strings $x_A$ and $r_A$ of length $n$ and sends $(f_n(x_A), r_A)$ to $B$, where $f_n$ is a one way permutation. Committed bit is $x_A \odot r_A$.
- $B$ selects a bit $b \in \{0, 1\}$ randomly and sends it to $A$.
- $A$ sends $x_A$ to $B$ and $B$ confirms it by checking value of $f_n(x_A)$. $b \oplus (x_a \odot r_A)$ can be taken as the coin toss.
Bit Commitment and Coin Tossing over Phone

Problems:

- How can two parties $A$ and $B$ toss a fair random coin over the phone?
- How can $A$ commit to a bit and reveal it at a later point such that $B$ is sure that it is the original bit?

Algorithm:

- $A$ selects strings $x_A$ and $r_A$ of length $n$ and sends $(f_n(x_A), r_A)$ to $B$, where $f_n$ is a one way permutation. Committed bit is $x_A \odot r_A$.
- $B$ selects a bit $b \in \{0, 1\}$ randomly and sends it to $A$.
- $A$ sends $x_A$ to $B$ and $B$ confirms it by checking value of $f_n(x_A)$. $b \oplus (x_a \odot r_A)$ can be taken as the coin toss.

After the first step $B$ cannot guess $x_A \odot r_A$ as that would allow it to differentiate in between pseudorandom and random distributions.
Bit Commitment and Coin Tossing over Phone

Problems:
- How can two parties \( A \) and \( B \) toss a fair random coin over the phone?
- How can \( A \) commit to a bit and reveal it at a later point such that \( B \) is sure that it is the original bit?

Algorithm:
- \( A \) selects strings \( x_A \) and \( r_A \) of length \( n \) and sends \((f_n(x_A), r_A)\) to \( B \), where \( f_n \) is a one way permutation. Committed bit is \( x_A \odot r_A \).
- \( B \) selects a bit \( b \in \{0, 1\} \) randomly and sends it to \( A \).
- \( A \) sends \( x_A \) to \( B \) and \( B \) confirms it by checking value of \( f_n(x_A) \).
  - \( b \oplus (x_a \odot r_A) \) can be taken as the coin toss.

After the first step \( B \) cannot guess \( x_A \odot r_A \) as that would allow it to differentiate in between pseudorandom and random distributions.
Also \( A \) cannot change \( x_A \) as \( f_n(x_A) \) is fixed and changing it would imply inverting one-way functions.
One-Way Functions and Zero Knowledge Proofs

Theorem
If one way functions exist then there exist Zero Knowledge Proofs for all NP Problems.

Proof.
Proving the result for one NP-complete problem is enough.

Algorithm for 3-colourability:
The Prover finds a 3-colouring ($\psi(v)$ : $v \in V$) and a random permutation $\pi$ over \{1, 2, 3\} and commits to $\phi(v) = \pi(\psi(v))$ and sends (after applying the commitment algorithm) to Verifier.

Verifier responds with two vertices $u, v$ choosen randomly such that $(u, v) \in E$.
Prover sends $\phi(u)$ and $\phi(v)$.
Verifier checks that they are correct and different and if so, accepts.

The probability of rejection in non 3-colourable graph is atleast $1/|E|$ and can be further reduced by repeating the algorithm.
Theorem

If one way functions exist then there exist Zero Knowledge Proofs for all NP Problems.
Theorem

If one way functions exist then there exist Zero Knowledge Proofs for all NP Problems.

Proof.

Proving the result for one NP-complete problem is enough.
One-Way Functions and Zero Knowledge Proofs

Theorem

If one way functions exist then there exist Zero Knowledge Proofs for all NP Problems.

Proof.

Proving the result for one NP-complete problem is enough.
Algorithm for 3-colourability:

- The Prover finds a 3-colouring \( \psi(v) : v \in V \) and a random permutation \( \pi \) over \{1, 2, 3\} and commits to \( \phi(v) = \pi(\psi(v)) \) and sends (after applying the commitment algorithm) to Verifier.
If one way functions exist then there exist Zero Knowledge Proofs for all NP Problems.

Proof.

Proving the result for one NP-complete problem is enough.

Algorithm for 3-colourability:

- The Prover finds a 3-colouring \( \psi(v) : v \in V \) and a random permutation \( \pi \) over \( \{1, 2, 3\} \) and commits to \( \phi(v) = \pi(\psi(v)) \) and sends (after applying the commitment algorithm) to Verifier.

- Verifier responds with two vertices \( u, v \) choosen randomly such that \( (u, v) \in E \).
Theorem

If one way functions exist then there exist Zero Knowledge Proofs for all NP Problems.

Proof.

Proving the result for one NP-complete problem is enough.

Algorithm for 3-colourability:

- The Prover finds a 3-colouring \( \psi(v) : v \in V \) and a random permutation \( \pi \) over \( \{1, 2, 3\} \) and commits to \( \phi(v) = \pi(\psi(v)) \) and sends (after applying the commitment algorithm) to Verifier.
- Verifier responds with two vertices \( u, v \) chosen randomly such that \( (u, v) \in E \).
- Prover sends \( \phi(u) \) and \( \phi(v) \).
Theorem

If one way functions exist then there exist Zero Knowledge Proofs for all NP Problems.

Proof.

Proving the result for one NP-complete problem is enough.

Algorithm for 3-colourability:

1. The Prover finds a 3-colouring \( (\psi(v) : v \in V) \) and a random permutation \( \pi \) over \( \{1, 2, 3\} \) and commits to \( \phi(v) = \pi(\psi(v)) \) and sends (after applying the commitment algorithm) to Verifier.
2. Verifier responds with two vertices \( u, v \) chosen randomly such that \( (u, v) \in E \).
3. Prover sends \( \phi(u) \) and \( \phi(v) \).
4. Verifier checks that they are correct and different and if so, accepts.

The probability of rejection in a non-3-colourable graph is at least \( \frac{1}{|E|} \) and can be further reduced by repeating the algorithm.
Theorem

If one way functions exist then there exist Zero Knowledge Proofs for all NP Problems.

Proof.

Proving the result for one NP-complete problem is enough.

Algorithm for 3-colourability:

- The Prover finds a 3-colouring \( \psi(v) : v \in V \) and a random permutation \( \pi \) over \{1, 2, 3\} and commits to \( \phi(v) = \pi(\psi(v)) \) and sends (after applying the commitment algorithm) to Verifier.
- Verifier responds with two vertices \( u, v \) chosen randomly such that \( (u, v) \in E \).
- Prover sends \( \phi(u) \) and \( \phi(v) \).
- Verifier checks that they are correct and different and if so, accepts.

The probability of rejection in non 3-colourable graph is at least \( \frac{1}{|E|} \) and can be further reduced by repeating the algorithm.