Advanced Computational Complexity

George Voutsadakis[1]

[1]Mathematics and Computer Science
Lake Superior State University

LSSU Math 600

Subsection 1

Perfect Secrecy and its Limitations

The Task of Secure Communication

- The fundamental task of cryptography is **encryption**.
- We focus on the issue of **security** of encryption.
- Encrypting using a **one time pad** provides security, but also suffers from serious limitations.
- Alice wants to send a secret message x, known as the **plaintext**, to Bob, but her adversary Eve is eavesdropping on the communication channel between Alice and Bob.
- Thus, Alice "scrambles" the plaintext x using an **encryption algorithm** E.
- In this way, she obtains a **ciphertext** y, which she sends to Bob.

The Task of Secure Communication (Cont'd)

- Presumably it will be hard or even impossible for Eve to decode the plaintext x from the ciphertext y.
- But Bob will be able to do so using the **decryption algorithm** D.
- Now Bob is seeing the same information that Eve is.
- So, in order to do something that Eve cannot, Bob has to know something that Eve does not.
- In **private key encryption**, we assume that Alice and Bob share some secret string k, known as the **key**, chosen at random.
- Presumably, Alice and Bob met beforehand and agreed on the key k.

A First Attempt

- In summary, the encryption scheme is composed of a pair of algorithms (E, D), such that:
  - Each takes a key and a message, where the key input is written as a subscript;
  - For every key k and plaintext x, $D_k(E_k(x)) = x$.
- This condition says nothing about the security of the scheme.
- E.g., it may be satisfied by the trivial "encryption" that just outputs the plaintext message.
- A first attempt at imposing security might be to declare that a scheme is secure if Eve cannot compute x from $E_k(x)$.
- This may not be sufficient because this does not rule out the possibility of Eve computing some partial information on x.

Example

- Suppose Eve knows that the plaintext is either the message “buy” or “sell”.
- Eve may not be ale to recover the message completely.
- However it is enough for her to learn only the first character of the message.

Shannon's Definition

- Shannon's definition of secure private key encryption ensures Eve does not learn anything about the plaintext from the ciphertext.
- U_n denotes the uniform distribution over $\{0,1\}^n$.

Definition (Perfect Secrecy)

Let (E, D) be an encryption scheme for messages of length m and with a key of length n satisfying, for every key k and plaintext x,

$$D_k(E_k(x)) = x.$$

We say that (E, D) is **perfectly secret** if, for every pair of messages $x, x' \in \{0,1\}^m$, the distributions $E_{U_n}(x)$ and $E_{U_n}(x')$ are identical.

Comments

- In a perfectly secret encryption, the ciphertext that Eve sees always has the same distribution, regardless of the plaintext.
- As a result, Eve gets absolutely no information on the plaintext.
- This condition may seem so strong that it is impossible to satisfy.
- In fact there is a very simple perfectly secret encryption scheme.

Perfectly Secret Encryption and One-Time Pad

The One-Time Pad Scheme

- To encrypt a message $x \in \{0,1\}^n$, we choose a random key $k \in_R \{0,1\}^n$ and encrypt x by simply sending

$$x \oplus k,$$

  where $\oplus$ denotes bitwise XOR or vector addition modulo 2.
- The receiver can recover the message x from $y = x \oplus k$ by XORing y once again with k.

- It is not hard to see that the ciphertext is distributed uniformly regardless of the plaintext message encrypted.
- Hence, the one-time pad is perfectly secret.

Drawback of the One-Time Pad Scheme

- A "one-time pad" must never be reused on another message.
- Suppose two messages x, x' are encoded using the same pad k.
- So Eve has both

$$k \oplus x \quad \text{and} \quad k \oplus x'.$$

- This allows her to compute

$$(k \oplus x) \oplus (k \oplus x') = x \oplus x'.$$

- So Eve obtains some nontrivial information about the messages.
- One can show that no perfectly secret encryption scheme can use a key size shorter than the message size.

Subsection 2

Computational Security, One-Way Functions, Pseudorandomness

Perfect Secrecy in Practice

- A one-time pad does provide perfect secrecy, but it fails utterly as a practical solution:
  - It requires private keys that are as long as the messages;
  - It is unclear such huge keys can be securely exchanged among users.
- Ideally we want to keep the shared secret key fairly small, say a few hundred bits long.
- To allow this, we design encryption schemes that are secure only against eavesdroppers that are efficient (i.e., run in polynomial-time).
- Even with this restriction on the eavesdropper, if $P = NP$, achieving perfect secrecy is impossible with small key sizes.
- In fact, assumptions stronger than $P \neq NP$ will be needed.
- More specifically, the assumption that a one-way function exists.
- Weakening the assumption (ideally to just $P \neq NP$) under which cryptographic schemes can be proven secure is a research problem.

Provable Lack of Security in Encryption

Lemma

Suppose that P = NP. Let (E, D) be any polynomial-time computable encryption scheme satisfying $D_k(E_k(x)) = x$, for every key k and plaintext x, with key shorter than the message. Then, there is a polynomial-time algorithm A, such that, for every input length m, there is a pair of messages $x_0, x_1 \in \{0,1\}^m$ satisfying:

$$\Pr_{\substack{b \in_R \{0,1\} \\ k \in_R \{0,1\}^n}} [A(E_k(x_b)) = b] \geq \frac{3}{4},$$

where $n < m$ denotes the key length for messages of length m.

- Such an algorithm breaks the security of the encryption scheme.
- Recall that a minimal requirement from an encryption is that Eve cannot tell which one of two random messages was encrypted with probability much better than $\frac{1}{2}$.

Proof of the Lack of Security Lemma

- Let (E, D) be an encryption scheme for messages of length m and with key length $n < m$.

  Let $S \subseteq \{0,1\}^*$ denote the support of $E_{U_n}(0^m)$.

  Note that $y \in S$ if and only if $y = E_k(0^m)$, for some k.

  So, if P = NP, then membership in S can be efficiently verified.

  Algorithm A is very simple.

  It receives input y.
  It outputs 0 if $y \in S$.
  It outputs 1, otherwise.

  Claim: Setting $x_0 = 0^m$, there exists some $x_1 \in \{0,1\}^m$, such that

$$\Pr_{\substack{b \in_R \{0,1\} \\ k \in_R \{0,1\}^n}} [A(E_k(x_b)) = b] \geq \frac{3}{4}.$$

Proof of the Lack of Security Lemma (Cont'd)

- Let x be a message.
  Let D_x denote the distribution $E_{U_n}(x)$.
  By the definition of A and since $x_0 = 0^m$, $\Pr[A(D_{x_0}) = 0] = 1$.
  Moreover, we have

$$\begin{aligned}\Pr_{\substack{b\in_R\{0,1\}\\ k\in_R\{0,1\}^n}}[A(E_k(x_b)) = b] &= \tfrac{1}{2}\Pr[A(D_{x_0}) = 0] + \tfrac{1}{2}\Pr[A(D_{x_1}) = 1] \\ &= \tfrac{1}{2} + \tfrac{1}{2}\Pr[A(D_{x_1}) = 1].\end{aligned}$$

  So it suffices to show that there's some $x_1 \in \{0,1\}^m$, such that

$$\Pr[A(D_{x_1}) = 1] \geq \frac{1}{2}.$$

  I.e., it suffices to show that, for some $x_1 \in \{0,1\}^m$,

$$\Pr[D_{x_1} \in S] \leq \frac{1}{2}.$$

Proof of the Lack of Security Lemma (Cont'd)

- Suppose, otherwise, that $\Pr[D_x \in S] > \frac{1}{2}$, for every $x \in \{0,1\}^m$.
  Define
  $$S(x,k) = \begin{cases} 1, & \text{if } E_k(x) \in S, \\ 0, & \text{otherwise.} \end{cases}$$
  Let
  $$T = E_{\substack{x \in_R \{0,1\}^m \\ k \in \{0,1\}^n}}[S(x,k)].$$
  Under our assumption, $T > \frac{1}{2}$.
  Note that, for every fixed key k, the map $x \mapsto E_k(x)$ is one-to-one.
  Hence, it maps at most $2^n \leq \frac{2^m}{2}$ of the x's to a set S of size $\leq 2^n$.
  So, by reversing the order of expectations above,
  $$T = E_{k \in \{0,1\}^n}[E_{x \in \{0,1\}^m}[S(x,k)]] \leq \frac{1}{2}.$$
  This contradicts $\Pr[D_x \in S] > \frac{1}{2}$, for every $x \in \{0,1\}^m$.

Negligible Functions

- To simplify notation we adopt

Definition (Negligible Function)

A function $\epsilon : \mathbb{N} \to [0, 1]$ is called **negligible** if

$$\epsilon(n) = n^{-\omega(1)}.$$

I.e., if, for every c and sufficiently large n, $\epsilon(n) < n^{-c}$.

- Negligible functions tend to zero very fast as their input grows.
- It follows that events that happen with negligible probability can be safely ignored in most practical and theoretical settings.

One Way Functions

- Complexity-theoretic conjectures are necessary to prove the security of encryption schemes.
- A one-way function is one that is easy to compute but hard to invert for a polynomial-time algorithm.

Definition (One-Way Functions)

A polynomial-time computable function $f : \{0,1\}^* \to \{0,1\}^*$ is a **one-way function** if, for every probabilistic polynomial-time algorithm A, there is a negligible function $\epsilon : \mathbb{N} \to [0,1]$, such that, for every n,

$$\Pr_{\substack{x \in_R \{0,1\}^n \\ y=f(x)}}[A(y) = x', \text{ such that } f(x') = y] < \epsilon(n).$$

Existence of One Way Functions Conjectured

Conjecture

There exists a one-way function.

- This conjecture implies that P $\neq$ NP.
- There are several examples for functions that no one has yet been able to invert.
- Based on this "evidence", most researchers believe the conjecture is true.

Multiplication

- Simple multiplication turns out to be hard to invert.
- The function that treats its input $x \in \{0,1\}^n$ as describing two $\frac{n}{2}$-bit numbers A and B and outputs $A \cdot B$ is believed to be one way.
- Inverting this function is known as the integer factorization problem.
- Note that it is easy to factor a number N using at most N (or even only $\sqrt{N}$) trial divisions.
- However, N is represented by $\log N$ bits.
- So this trivial algorithm is actually an exponential time algorithm.
- At the moment, no polynomial (i.e., polylog(N)) time algorithm is known for this problem.
- The best known factoring algorithm runs in time $2^{O\left(\log^{1/3} N \sqrt{\log \log N}\right)}$.

Multiplication (Cont'd)

- A standard implementation of a one-way function based on factoring goes as follows.
  - Receive input $x \in \{0,1\}^n$.
  - Treat the x as a random seed.
    Use it to generate two random $n^{1/3}$-bit primes P and Q.
    This can be done by generating random numbers and testing their primality.
  - Then output $P \cdot Q$.
- No efficient factorization algorithm has been found, despite efforts spanning at least two millennia, leading to the conjecture that no such algorithm exists.
- This conjecture is obviously much stronger than the conjecture that P $\neq$ NP or the conjecture that some one-way function exists.

The RSA Function

- The **RSA function** is another very popular candidate for a one-way function.
- Assume that, for every input length n, there exist:
  - An n-bit composite integer N that was generated in some way;
  - Some number e that is coprime to $\varphi(N) = |\mathbb{Z}_N^*|$, where $\mathbb{Z}_N^*$ is the multiplicative group of numbers coprime to N.
- Typically, N would be generated as a product of two $\frac{n}{2}$-long primes.
- e is often set to be simply 3.
- The function $\mathrm{RSA}_{N,e}$ treats its input as a number X in $\mathbb{Z}_N^*$.
- It outputs

$$X^e \pmod{N}.$$

- Since e is coprime to $\varphi(N)$, $\mathrm{RSA}_{N,e}$ is one-to-one on $\mathbb{Z}_N^*$.

The Rabin Function

- An element $X \in \mathbb{Z}_N^*$ is a quadratic residue modulo N if

$$X = W^2 \pmod{N},$$

  for some $W \in \mathbb{Z}_N^*$.
- Let QR_N denote the set of quadratic residues modulo N.
- A candidate one-way function, related to $\mathrm{RSA}_{N,e}$, is the **Rabin function**.
- The input is a number N that is the product of two odd primes P, Q, such that

$$P, Q \equiv 1 \pmod{4}.$$

- It maps $X \in \mathrm{QR}_N$ into

$$X^2 \pmod{N}.$$

- It can be shown that this function is one-to-one on QR_N.

Common Features of RSA and Rabin Functions

- The RSA and the Rabin functions are believed to be hard to invert.
- Inverting them is actually easy if one knows the factorization of N.
- Consider the case of the RSA function.

  The factorization can be used to compute $\varphi(N)$.

  Then a number d, with $d = e^{-1} \pmod{\varphi(N)}$, may be calculated.

  It is not hard to verify that the function $Y^d \pmod{N}$ is the inverse of the function $X^e \pmod{N}$.
- Consider the case of the Rabin function.

  Suppose we know the factorization.

  Then we can use the Chinese Remainder Theorem to reduce the problem of taking a square root modulo N to taking square roots modulo the prime factors of N.

  This can be done in polynomial time.

Trapdoor One-Way Functions

- The RSA and the Rabin functions functions:
  - Are conjectured to be hard to invert;
  - Become easy to invert once we know certain information (i.e., N's factorization).
- For these reasons, they are known as **trapdoor one-way functions**.
- They are crucial to obtaining public key cryptography.
- It it known that inverting Rabin's function is computationally equivalent to factoring N.
- On the other hand, no such equivalence is known for RSA.

Levin's Universal One-way Function

- There is a function $f_{\mathcal{U}}$ that has a curious property:

  If any one-way function exists, then $f_{\mathcal{U}}$ is also a one-way function.
- The function $f_{\mathcal{U}}$ is called a **universal one-way function**.
- The function $f_{\mathcal{U}}$ is defined as follows:
  - Treat the input as a list $x_1, \ldots, x_{\log n}$ of $\frac{n}{\log n}$ bit long strings.
  - Let M_i denote the i-th Turing machine according to some canonical representation.
  - Let $M^t(x)$ be:
    - The output of the Turing machine M on input x, if M uses at most t computational steps on input x;
    - The all-zeroes string $0^{|x|}$, if M uses more than t computational steps on x.
  - Output

$$M_1^{n^2}(x_1), \ldots, M_{\log n}^{n^2}(x_n).$$

Encryption from One-Way Functions

- We show that one-way functions can be used to design secure encryption schemes with keys much shorter than the message length.

Theorem (Encryption from One-Way Function)

Suppose that one-way functions exist. Then for every $c \in \mathbb{N}$, there exists a computationally secure encryption scheme (E, D) using n-length keys for n^c-length messages.

- Of course, for the statement to make sense, we must formally define the term "computationally secure", applied to the encryption scheme (E, D).

Computationally Secure Encryption Schemes

- We must define the term "computationally secure".
- We wish that no partial information about the plaintext to a polynomial-time eavesdropper may be revealed.
- To simplify, we only insist that no individual bit of the plaintext, chosen at random, can be guessed with probability non-negligibly higher than $\frac{1}{2}$.
- We say that a scheme (E, D) using length n keys for length m messages is **computationally secure** if, for every probabilistic polynomial-time A, there is a negligible function $\epsilon : \mathbb{N} \to [0, 1]$, such that

$$\Pr_{\substack{k \in_R \{0,1\}^n \\ x \in_R \{0,1\}^m}} [A(E_k(x)) = (i, b) \text{ such that } x_i = b] \leq \frac{1}{2} + \epsilon(n).$$

Motivating Pseudorandomness

- When is it appropriate to call a string random?
- Kolmogorov provided the following definition:

  A string of length n is **random** if no Turing machine whose description length is $< 0.99n$ outputs this string when started on an empty tape.
- This definition is "right" in some philosophical and technical sense.
- It is not very useful in the complexity setting because checking if a string is random according to this definition is undecidable.
- Statisticians have also attempted definitions that boil down to checking if the string has the "right number" of patterns that one would expect by the laws of statistics (e.g., the number of times 11100 appears as a substring).
- Such definitions are too weak in the cryptographic setting:

  One can find a distribution that passes these statistical tests but still will be completely insecure if used to generate the pad for the one-time pad encryption scheme.

Pseudorandom Distributions

- In cryptography:
  - Instead of trying to describe what it means for a single string to be "random-looking", we focus on distributions on strings;
  - Instead of focusing on individual tester algorithms as the statisticians did, we say that the distribution has to "look" like the uniformly random distribution to every polynomial-time algorithm.
- Such a distribution is called **pseudorandom**.
- The distinguisher algorithm:
  - Is given a sample string that is drawn from either the uniform distribution or the unknown distribution;
  - Outputs "1" or "0" depending upon whether or not this string looks random to it.
- The distribution is said to be **pseudorandom** if the probability that the polynomial-time algorithm outputs 1 is essentially the same on the two distributions, regardless of which algorithm is used.

Secure Pseudorandom Generators

Definition (Secure Pseudorandom Generator)

Let $G : \{0,1\}^* \to \{0,1\}^*$ be a polynomial-time computable function. Let $\ell : \mathbb{N} \to \mathbb{N}$ 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** $\ell(n)$, if:

- For every $x \in \{0,1\}^*$,
$$|G(x)| = \ell(|x|);$$
- For every probabilistic polynomial-time A, there exists a negligible function $\epsilon : \mathbb{N} \to [0,1]$, such that, for every $n \in \mathbb{N}$,
$$\left|\Pr[A(G(U_n)) = 1] - \Pr[A(U_{\ell(n)}) = 1]\right| < \epsilon(n).$$

Pseudorandom Generators from One-Way Functions

Theorem (Pseudorandom Generators from One-Way Functions)

If one-way functions exist, then for every $c \in \mathbb{N}$, there exists a secure pseudorandom generator with stretch $\ell(n) = n^c$.

- The next section is dedicated to proving the theorem, under the hypothesis that the one-way function is a permutation.
- The definition of a secure pseudorandom generator states that it is infeasible for polynomial-time adversaries to distinguish between a completely random string of length $\ell(n)$ and a string that was generated by applying the generator G to a much shorter random string of length n.
- Using this fact, it may be seen that the preceding theorem implies the existence of computationally secure encryption schemes, subject to the availability of one-way functions.

One-Way Functions to Secure Encryption Schemes

- Suppose that one-way functions exist.
  By the preceding theorem, there exists a secure pseudorandom generator with stretch $\ell(n) = n^c$.
  We want to communicate a message of length n^c.
  We start with a short key of length n.
  We use a a secure pseudorandom generator with stretch n^c.
  We apply the one-time pad encryption with this n^c-length key.
  We claim that a polynomial-time eavesdropper would not be able to tell the difference.
  Assume that there is an adversary A that can predict a bit of the plaintext with probability noticeably larger than $\frac{1}{2}$.
  But such prediction is impossible when the key is truly random.
  We conclude that A can be used to distinguish between a pseudorandom and truly random key.
  This contradicts the security of the generator.

Subsection 3

Pseudorandom Generators from One-Way Permutations

Pseudorandom Generators from One-Way Permutations

- We will prove only the preceding theorem under the proviso that the one-way function is a permutation.

Lemma

Suppose there exists a one-way function $f : \{0,1\}^* \to \{0,1\}^*$, such that:

- f is one-to-one;
- For every $x \in \{0,1\}^*$, $|f(x)| = |x|$.

Then, for every $c \in \mathbb{N}$, there exists a secure pseudorandom generator with stretch n^c.

- The proof incorporates some ideas of independent interest that have found several applications in other areas of computer science.
- Among these are the hybrid argument and the Goldreich-Levin Theorem.

Unpredictability Implies Pseudorandomness

- We provide an alternative characterization of pseudorandom generators.
  - It is a weaker notion and, hence, easier to achieve by explicit constructions.
  - Yao's proof that it is equivalent to the original was surprising.
- Let $G : \{0,1\}^* \to \{0,1\}^*$ be a polynomial-time computable function with stretch $\ell(n)$, i.e., $|G(x)| = \ell(|x|)$, for every $x \in \{0,1\}^*$.
- We call G **unpredictable** if, for every probabilistic polynomial-time B, there is a negligible function $\epsilon : \mathbb{N} \to [0,1]$, such that

$$\Pr_{\substack{x \in_R \{0,1\}^n \\ y = G(x) \\ i \in_R [\ell(n)]}} [B(1^n, y_1, \ldots, y_{i-1}) = y_i] \leq \frac{1}{2} + \epsilon(n).$$

- Predicting the i-th bit given the first $i-1$ bits, where i is a randomly chosen index, is difficult for every polynomial-time algorithm.

Pseudorandomness Implies Unpredictability

- Suppose G is a pseudorandom generator.
- We can see that it is also unpredictable.
- Suppose

$$y_1, \ldots, y_{\ell(n)}$$

  are uniformly chosen bits.
- Then it would be impossible to predict y_i given $y_1, \ldots, y_{i-1}$.
- Assume such a predictor exists, when $y = G(x)$, for a random x.
- Then the predictor can distinguish between $U_{\ell(n)}$ and $G(U_n)$.

Yao's Theorem

Theorem (Unpredictability Implies Pseudorandomness)

Let $\ell : \mathbb{N} \to \mathbb{N}$ be some polynomial-time computable function.
Let $G : \{0,1\}^* \to \{0,1\}^*$ be a polynomial-time computable function such that

$$|G(x)| = \ell(|x|), \quad \text{for every } x \in \{0,1\}^*.$$

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

$$\Pr_{\substack{x \in_R \{0,1\}^n \\ y = G(x) \\ i \in_R [\ell(n)]}} [B(1^n, y_1, \ldots, y_{i-1}) = y_i] \geq \frac{1}{2} + \frac{\epsilon}{\ell(n)}.$$

Proof of Yao's Theorem (Observation)

- We show, first, that the main result follows from the "moreover" part.

  Suppose that G is not a pseudorandom generator.

  Hence, there is some algorithm A and constant c, such that

$$\left|\Pr[A(G(U_n)) = 1] - \Pr[A(U_{\ell(n)}) = 1]\right| \geq n^{-c},$$

  for infinitely many n's.

  Then, we can ensure (changing A to $1 - A$ if necessary), that for infinitely many n's, this relationship holds without the absolute value.

  For every such n, we get a predictor B that succeeds with probability

$$\frac{1}{2} + \frac{n^{-c}}{\ell(n)}.$$

  This contradicts the unpredictability property.

Proof of Yao's Theorem

- We prove the "moreover" part.

  Let A be some probabilistic polynomial-time algorithm that is supposedly more likely to output 1 on inputs from the distribution $G(U_n)$ than on inputs from $U_{\ell(n)}$.

  Our predictor algorithm B is quite simple.
  - Let the input be 1^n, $i \in [\ell(n)]$ and $y_1, \ldots, y_{i-1}$.
  - Choose $z_i, \ldots, z_{\ell(n)}$ independently at random.
  - Compute
    $$a = A(y_1, \ldots, y_{i-1}, z_i, \ldots, z_{\ell(n)}).$$
    - Suppose $a = 1$.
      B surmises its guess for z_i is correct and outputs z_i.
    - Otherwise, it outputs $1 - z_i$.

Correctness of the Algorithm B

- Let $n \in \mathbb{N}$ and $\ell = \ell(n)$.
  Suppose that

$$\Pr[A(G(U_n)) = 1] - \Pr[A(U_{\ell(n)}) = 1] \geq \epsilon.$$

  We show that

$$\Pr_{\substack{x \in_R \{0,1\}^n \\ y = G(x) \\ i \in_R [\ell]}}[B(1^n, y_1, \ldots, y_{i-1}) = y_i] \geq \frac{1}{2} + \frac{\epsilon}{\ell}.$$

  To analyze B's performance, we use a technique called the **hybrid argument**.
  We define ℓ distributions $\mathcal{D}_0, \ldots, \mathcal{D}_\ell$ over $\{0,1\}^\ell$.
  For every i, the distribution $\mathcal{D}_i$ is obtained by:
  - Choosing $x \in_R \{0,1\}^n$.
  - Letting $y = G(x)$;
  - Outputting $y_1, \ldots, y_i, z_{i+1}, \ldots, z_\ell$, where $z_{i+1}, \ldots, z_\ell$ are chosen independently at random in $\{0,1\}$.

Correctness of the Algorithm B (Cont'd)

- Note that $\mathcal{D}_0 = U_\ell$, while $\mathcal{D}_\ell = G(U_n)$.
  For every $i \in \{0, \ldots, \ell\}$, define

$$p_i = \Pr[A(\mathcal{D}_i) = 1].$$

  Note that $p_\ell - p_0 \geq \epsilon$.
  Write

$$p_\ell - p_0 = (p_\ell - p_{\ell-1}) + (p_{\ell-1} - p_{\ell-2}) + \cdots + (p_1 - p_0).$$

  So we get that

$$\sum_{i=1}^{\ell} (p_i - p_{i-1}) \geq \epsilon.$$

  That is

$$E_{i \in [\ell]}[p_i - p_{i-1}] \geq \frac{\epsilon}{\ell}.$$

Correctness of the Algorithm B (Cont'd)

- We will prove $\Pr_{\substack{x\in_R\{0,1\}^n \\ y=G(x) \\ i\in_R[\ell]}}[B(1^n, y_1, \ldots, y_{i-1}) = y_i] \geq \frac{1}{2} + \frac{\epsilon}{\ell}$ by showing that for every i,

$$\Pr_{\substack{x\in_R\{0,1\}^n \\ y=G(x)}}[B(1^n, y_1, \ldots, y_{i-1}) = y_i] \geq \frac{1}{2} + (p_i - p_{i-1}).$$

Recall that B:

- Makes a guess z_i for y_i;
- Invokes A to obtain a value a;
- Outputs z_i if $a = 1$ and $1 - z_i$ otherwise.

Thus, B predicts y_i correctly if one of the following occurs:

- $a = 1$ and $y_i = z_i$;
- $a \neq 1$ and $y_i = 1 - z_i$.

The probability this event happens is

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

Correctness of the Algorithm B (Cont'd)

- One can verify that, conditioned on $z_i = y_i$, B invokes A with the distribution $\mathcal{D}_i$.
  So we have
  $$\Pr[a = 1 | z_i = y_i] = p_i.$$
  On the other hand, if we do not condition on z_i, then the distribution B invokes A with the distribution $\mathcal{D}_{i-1}$.
  Hence,
  $$\begin{aligned} p_{i-1} &= \Pr[a = 1] \\ &= \tfrac{1}{2}\Pr[a = 1 | z_i = y_i] + \tfrac{1}{2}\Pr[a = 1 | z_i = 1 - y_i] \\ &= \tfrac{1}{2}p_i + \tfrac{1}{2}\Pr[a = 1 | z_i = 1 - y_i]. \end{aligned}$$
  Plugging this into the last equation of the previous slide, we get that B predicts y_i with probability
  $$\frac{1}{2} + p_i - p_{i-1}.$$

Introducing The Goldreich-Levin Theorem

- Let f be some one-way permutation.
- We need to use f to come up with a pseudorandom generator with arbitrarily large polynomial stretch $\ell(n)$.
- The crucial step is obtaining a pseudorandom generator that extends its input by one bit, i.e., has stretch $\ell(n) = n + 1$.

The Goldreich-Levin Theorem

Theorem (Goldreich-Levin Theorem)

Suppose that $f : \{0,1\} \to \{0,1\}$ is a one-way function such that:

- f is one-to-one;
- $|f(x)| = |x|$, for every $x \in \{0,1\}^*$.

Then, for every probabilistic polynomial-time algorithm A, there is a negligible function $\epsilon : \mathbb{N} \to [0,1]$, such that

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

where $x \odot r$ is defined to be

$$x \odot r = \sum_{i=1}^{n} x_i r_i \pmod 2.$$

Immediate Consequence of the Goldreich-Levin Theorem

- The theorem implies that the function

$$G(x, r) = f(x), r, x \odot r$$

  is a secure pseudorandom generator that extends its input by one bit (mapping $2n$ bits into $2n + 1$ bits).
- Otherwise, there would be a predictor B for this function.
- But f is a permutation over $\{0, 1\}^n$.
- So the first $2n$ bits of $G(U_{2n})$ are completely random and independent.
- Hence, they cannot be predicted from their predecessors with probability better than $\frac{1}{2}$.
- Thus, a predictor for this function would have to succeed at predicting the $(2n + 1)$-st bit from the previous $2n$ bits with probability noticeably larger than $\frac{1}{2}$.
- This exactly amounts to violating the theorem.

Plan of the Proof

- Suppose, for the sake of contradiction, that there is some probabilistic polynomial-time algorithm A that violates the theorem's statement.
- We will use A to show a probabilistic polynomial-time algorithm B that inverts the permutation f.
- This would contradict the assumption that f is one way.

Plan of the Proof (Cont'd)

- Specifically, we will show that if, for some n,

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

  then B will:
  - Run in $O\left(\frac{n^2}{\epsilon^2}\right)$ time;
  - Invert the one-way permutation f on inputs of length n, with probability at least $\Omega(\epsilon)$.
- This means that if A's success probability is more than $\frac{1}{2}+n^{-c}$, for some constant c and infinitely many n's, then B:
  - Runs in polynomial time;
  - Inverts the one-way permutation with probability at least $\Omega(n^{-c})$, for infinitely many n's.

Preparing the Proof

- Let n be such that

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

Then, by a simple averaging argument, for at least an $\frac{\epsilon}{2}$ fraction of the x's,

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

We call such x's **good**.

We show an algorithm B that, with high probability, inverts $f(x)$, for every good x.

Restating, we are given a "black box" that computes an unknown linear function $x \mapsto x \odot r$ for at least $\frac{1}{2} + \frac{\epsilon}{2}$ fraction of r's.

We have to give an efficient algorithm that reconstructs x in time polynomial in $|x|$ and $\frac{1}{\epsilon}$.

Preparing the Proof (Warm-Up)

- As a warm-up, suppose that, for all r,

$$\Pr_r[A(f(x), r) = x \odot r] = 1.$$

Then it is easy to recover x from $f(x)$.
Let e^i be the string defined by

$$e^i_j = \begin{cases} 1, & \text{if } j = i, \\ 0, & \text{if } j \neq i. \end{cases}$$

Run $A(f(x), e^1), \ldots, A(f(x), e^n)$.
Clearly, $x \odot e^i$ is the i-th bit of x.
Hence, by making these n calls to A we can recover x completely.
Of course, this idea breaks down if

$$\Pr_r[A(f(x), r) = x \odot r] < 1.$$

Recovery for Success Probability 0.9

- Suppose that for an $\Omega(\epsilon)$ fraction of x's, we had

$$\Pr_r[A(f(x), r) = x \odot r] \geq 0.9.$$

For such an x, we cannot trust that

$$A(f(x), e^i) = x \odot e^i.$$

It may be that $e^1, \ldots, e^n$ are among the $\frac{2^n}{10}$ strings r on which A answers incorrectly.

Still, there is a simple way to bypass this problem.

Recovery for Success Probability 0.9 (Cont'd)

- Suppose we choose $r \in_R \{0,1\}^n$.

  Then the string $r \oplus e^i$ is also uniformly distributed.

  Hence, by the union bound, the probability that the algorithm A answers incorrectly on either string is

$$\Pr_r[A(f(x), r) \neq x \odot r \text{ or } A(f(x), r \oplus e^i) \neq x \odot (r \oplus e^i)] \leq 0.2.$$

  But

$$x \odot (r \oplus e^i) = (x \odot r) \oplus (x \odot e^i).$$

  This means that, if we choose r at random, and compute

$$z = A(f(x), r) \quad \text{and} \quad z' = A(f(x), r \odot e^i),$$

  then $z \oplus z'$ will be equal to the i-th bit of x with probability ≥ 0.8.

  To obtain every bit of x, we amplify this probability to $1 - \frac{1}{10n}$ by taking majorities.

Algorithm B

Algorithm B

1. Choose $r^1, \ldots, r^m$ independently at random from $\{0,1\}^n$ (the number m to be specified shortly).
2. For every $i \in [n]$:
   - Compute the values

$$\begin{array}{ll} z_1 = A(f(x), r^1), & z'_1 = A(f(x), r^1 \odot e^i), \\ \vdots & \vdots \\ z_m = A(f(x), r^m), & z'_m = A(f(x), r^m \odot e^i); \end{array}$$

   - Guess that x_i is the majority value among $\{z_j \oplus z'_j\}_{j \in [m]}$.

Claim: If $m = 200n$, then, for every $i \in [n]$, the majority value will be correct with probability at least $1 - \frac{1}{10n}$.

Hence, B will recover every bit of x with probability at least 0.9.

Proof of the Claim

- To prove the claim, we define the random variable

$$Z_j = \begin{cases} 1, & \text{if } A(f(x), r^j) = x \odot r^j \text{ and } A(f(x), r^j \oplus e^i) = x \odot (r^j \oplus e^i), \\ 0, & \text{otherwise.} \end{cases}$$

Note that the variables $Z_1, \ldots, Z_m$ are independent.
Moreover, by our previous discussion,

$$E[Z_j] \geq 0.8, \quad \text{for every } j.$$

It suffices to show that with probability $1 - \frac{1}{10n}$, more than $\frac{m}{2}$ of the Z_j's are equal to 1.
To rephrase, let

$$Z = Z_1 + \cdots + Z_m.$$

Then it suffices to show that

$$\Pr\left[Z \leq \frac{m}{2}\right] \leq \frac{1}{10n}.$$

Proof of the Claim (Cont'd)

- We know that $E[Z] = \sum_j E[Z_j] \geq 0.8m$.
  So all we need to do is bound

$$\Pr[|Z - E[Z]| \geq 0.3m].$$

  By Chebychev's Inequality,

$$\Pr\left[|Z - E[Z]| \geq k\sqrt{\mathrm{Var}(Z)}\right] \leq \frac{1}{k^2}.$$

  In our case, because the Z_j's are independent 0/1 random variables, $\mathrm{Var}(Z) = \sum_{j=1}^{m} \mathrm{Var}(Z_j)$ and $\mathrm{Var}(Z_j) \leq 1$, for every j.
  So, for $k = 0.3\sqrt{m}$, we get that

$$\Pr[|Z - E[Z]| \geq 0.3m] \leq \frac{1}{(0.3\sqrt{m})^2} \overset{m = 200n}{\leq} \frac{1}{10n}.$$

Recovery for Success Probability $\frac{1}{2} + \frac{\epsilon}{2}$: Idea

- The preceding analysis crucially used the unrealistic assumption that for many x's, $A(f(x), r)$ is correct with probability at least 0.9 over r.
- Once this falls below 0.75, we no longer get any meaningful information by applying the union bound on the events

$$A(f(x), r) = x \odot r \quad \text{and} \quad A(f(x), r \oplus e^i) = x \odot (r \oplus e^i).$$

- In general, our only guarantee is that, if x is good, then this probability is at least $\frac{1}{2} + \frac{\epsilon}{2}$ (which could be much smaller than 0.75).

Recovery for Success Probability $\frac{1}{2} + \frac{\epsilon}{2}$ (Cont'd)

- The crucial insight needed to extend the proof is that all of the above analysis would still carry over even if the strings $r^1, \ldots, r^m$ are only chosen to be pairwise independent as opposed to fully independent.
- The only place where we used independence is to argue that the random variables $Z_1, \ldots, Z_m$ satisfy

$$\text{Var}\left(\sum_j Z_j\right) = \sum_j \text{Var}(Z_j).$$

- This condition holds also for pairwise independent random variables.
- We show how to pick $r^1, \ldots, r^m$ in a pairwise indpendent fashion in such a way that we "know" each $x \odot r^i$ already.
- Despite x being unknown, we can do this thanks to some exhaustive guessing.

Recovery for Success Probability $\frac{1}{2} + \frac{\epsilon}{2}$: Details

- Set k such that $m \leq 2^k - 1$ and proceed as follows.

1. Choose k strings $s^1, \ldots, s^k$ independently at random from $\{0,1\}^n$.
2. For every $j \in [m]$, we associate with j a unique nonempty set

$$T_j \subseteq [k],$$

in some canonical fashion.

We define

$$r^j = \sum_{t \in T_j} s^t \pmod 2.$$

That is, r^j is the bitwise XOR of all the strings among $s^1, \ldots, s^k$ that belong to the j-th set.

Recovery for Success Probability $\frac{1}{2} + \frac{\epsilon}{2}$: Remarks

- It can be shown that the strings $r^1, \ldots, r^m$ are pairwise independent.
- Moreover, for every $x \in \{0,1\}^n$,

$$x \odot r^j = \sum_{t \in T_j} x \odot s^t.$$

- This implies that, if we know the k values $x \odot s^1, \ldots, x \odot s^k$, then we can deduce the m values $x \odot r^1, \ldots, x \odot r^m$.
- This is where exhaustive guessing comes in.
- By construction, $2^k = \mathrm{O}(m)$.
- So we can enumerate over all possible guesses for

$$x \odot s^1, \ldots, x \odot s^k$$

in polynomial time.

Recovery for Success Probability $\frac{1}{2} + \frac{\epsilon}{2}$: The Algorithm

Algorithm B'

Input: $y \in \{0,1\}^n$, where $y = f(x)$, for an unknown x.
We assume that x is "good" and, hence,

$$\Pr_r[A(f(x), r) = x \odot r] \geq \frac{1}{2} + \frac{\epsilon}{2}.$$

(We do not care how B' performs on x's that are not good.)
Operation: Let $m = \frac{200n}{\epsilon^2}$.
Let k be the smallest such that $m \leq 2^k - 1$.
Choose $s^1, \ldots, s^k$ independently at random in $\{0,1\}^n$.
Define $r^1, \ldots, r^m$ as previously.
For every string $w \in \{0,1\}^k$, do the following.

Recovery for Success Probability $\frac{1}{2} + \frac{\epsilon}{2}$ (Cont'd)

Algorithm B' (Cont'd)

- Run the algorithm B under the assumption that

$$x \odot s^t = w_t, \quad \text{for every } t \in [k].$$

That is, for every $i \in [n]$, we compute our guess z_j for $x \odot r^j$ by setting

$$z_j = \sum_{t \in T_j} w_t.$$

We compute the guess z'_j for $x \odot (r^j \oplus e^i)$ as before by setting

$$z'_j = A(y, r^j \oplus e^i).$$

Recovery for Success Probability $\frac{1}{2} + \frac{\epsilon}{2}$ (Cont'd)

Algorithm B' (Cont'd)

- As before, for every $i \in [n]$, our guess for x_i is the majority value among

$$\{z_j \oplus z'_j\}_{j \in [m]}.$$

- We test whether our guess for $x = x_1, \ldots, x_n$ satisfies

$$f(x) = y.$$

If so, we halt and output x.

Recovery for Success Probability $\frac{1}{2} + \frac{\epsilon}{2}$: Analysis

- The analysis is almost identical to the previous case.

  In one of the 2^k iterations, we will guess the correct values $w_1, \ldots, w_k$ for $x \odot s^1, \ldots, x \odot s^k$.

  We will show that, in this particular iteration, for every $i \in [n]$, Algorithm B' guesses x_i correctly with probability at least $1 - \frac{1}{10n}$.

  Indeed, fix some $i \in [n]$.

  Define the random variables $Z_1, \ldots, Z_m$ as before

$$Z_j = \begin{cases} 1, & \text{if } z_j = x \odot r^j \text{ and } z'_j = x \odot (r^j \oplus e^i), \\ 0, & \text{otherwise.} \end{cases}$$

Recovery for Success Probability $\frac{1}{2} + \frac{\epsilon}{2}$ (Cont'd)

- In the iteration where we chose the right values $w_1, \ldots, w_k$, it always holds that

$$z_j = x \odot r^j.$$

Hence, Z_j depends only on the second event.

This event happens with probability at least $\frac{1}{2} + \frac{\epsilon}{2}$.

Thus, all that is needed is to show that, for $m = \frac{100n}{\epsilon^2}$, if $Z_1, \ldots, Z_m$ are pairwise independent 0/1 random variables, where $E[Z_j] \geq \frac{1}{2} + \frac{\epsilon}{2}$, for every j, then

$$\Pr\left[\sum_j Z_j \leq \frac{m}{2}\right] \leq \frac{1}{10n}.$$

This follows immediately from Chebychev's inequality.

Getting Arbitrarily Large Stretch

- The Goldreich-Levin Theorem provides a secure pseudorandom generator of stretch $\ell(n) = n + 1$.
- We still need a generator with arbitrarily large polynomial stretch.

Theorem

Let f be a one-way permutation and $c \in \mathbb{N}$.
Then the function G that maps $x, r \in \{0,1\}^n$ to

$$r, f^{\ell}(x) \odot r, f^{\ell-1}(x) \odot r, \ldots, f^{1}(x) \odot r, \quad \ell = n^c,$$

is a secure pseudorandom generator of stretch $\ell(2n) = n + n^c$.
Here, f^i denotes the function obtained by applying f repeatedly to the input n times.

- By Yao's Theorem, it suffices to show that individual bits of f are hard to predict.

Getting Arbitrarily Large Stretch (Cont'd)

- Assume, to the contrary, there is a PPT machine A, such that, when $x, r \in \{0,1\}^n$ and $i \in \{1, \ldots, N\}$ are randomly chosen,

$$\Pr[A \text{ predicts } f^i(x) \odot r \text{ given } (r, f^\ell(x) \odot r, f^{N-1}(x) \odot r, \ldots, f^{i+1}(x) \odot r)] \geq \tfrac{1}{2} + \epsilon.$$

We will show a probabilistic polynomial-time algorithm B that, on such n's, will predict $x \odot r$ from $f(x), r$ with probability at least $\frac{1}{2} + \epsilon$.

Thus, if A has nonnegligible success, then B violates the Goldreich-Levin Theorem.

Getting Arbitrarily Large Stretch (Cont'd)

- Algorithm B:
  - Input consist of r and y, such that $y = f(x)$, for some x.
  - Pick $i \in \{1, \ldots, N\}$ randomly.
  - Compute the values
$$f^{\ell-i}(y), \ldots, f(y).$$
  - Output
$$a = A(r, f^{\ell-i-1}(y) \odot r, \ldots, f(y) \odot r, y \odot r).$$

Because f is a permutation, this is exactly the same distribution obtained where we choose $x' \in_R \{0,1\}^n$ and set $x = f^i(x')$.

Hence, A will predict $f^i(x') \odot r$ with probability $\frac{1}{2} + \epsilon$.

Thus, B predicts $x \odot r$ with the same probability.

Subsection 4

Zero Knowledge

The Idea Behind Zero Knowledge

- A proof presents evidence that some statement is true.
- Typically after carefully reading and verifying a proof for some statement, one learns much more than the mere fact that the statement is true.
- Does it always have to be this way?

  Example: Suppose we figured out how to schedule all of the flights of some airline in a way that saves millions of dollars.

  Can we prove to the airline that there exists such a schedule, without actually revealing the schedule?

Example

- Suppose a company has a sensitive building, that only a select group of employees is allowed to enter.
- One way to enforce this is to:
  - Choose two random prime numbers P and Q;
  - Reveal these numbers to the selected employees;
  - Reveal $N = P \cdot Q$ to the guard outside the building.
- The guard will be instructed to let inside only a person demonstrating knowledge of N's factorization.
- Is it possible to demonstrate such knowledge without revealing the factorization?

Zero Knowledge Property

- This is in fact possible using the notion of zero knowledge proof.
- A zero knowledge proofs is a special type of an interactive probabilistic proof systems.
- We have, as usual:
  - The completeness property (prover can convince the verifier to accept with high probability);
  - The soundness property (verifier will reject false statements with high probability).
- In addition, we require the zero knowledge property:
  - Roughly speaking, it requires that the verifier does not learn anything from the interaction apart from the fact that the statement is true.
  - I.e., zero knowledge requires that whatever the verifier learns after participating in a proof for a statement x, she could have computed by herself, without participating in any interaction.

Zero Knowledge Proofs

- We give the formal definition for zero knowledge proofs of NP languages.
- The concept of zero knowledge can be defined also for languages outside NP.

Definition (Zero Knowledge Proofs)

Let L be an NP-language, and let M be a poly time Turing machine, such that

$$x \in L \quad \text{iff} \quad \exists u \in \{0,1\}^{p(|x|)} \; M(x,u) = 1,$$

where p is a polynomial.
A pair P, V of interactive probabilistic polynomial-time algorithms is called a **zero knowledge proof** for L, if the following three conditions hold.

Zero Knowledge Proofs (Cont'd)

Definition (Zero Knowledge Proofs Cont'd)

Completeness: For every $x \in L$ and u a certificate for this fact, i.e., $M(x,u) = 1$,

$$\Pr[\text{out}_V \langle P(x,u), V(x) \rangle] \geq \frac{2}{3},$$

where $\langle P(x,u), V(x) \rangle$ denotes the interaction of P and V in which:

- P gets x, u as input;
- V gets x as input;
- $\text{out}_V I$ denotes the output of V at the end of the interaction I.

Soundness: If $x \notin L$, then, for every strategy P^* and input u,

$$\Pr[\text{out}_V \langle P^*(x,u), V(x) \rangle] \leq \frac{1}{3}$$

Here P need not run in polynomial time.

Zero Knowledge Proofs (Cont'd)

Definition (Zero Knowledge Proofs Cont'd)

Perfect Zero Knowledge: For every probabilistic polynomial-time interactive strategy V^*, there exists an expected probabilistic polynomial-time (stand-alone) algorithm S^*, such that, for every $x \in L$ and u a certificate for this fact,

$$\text{out}_{V^*}\langle P(x,u), V^*(x)\rangle \equiv S^*(x).$$

That is, these two random variables are identically distributed, even though S does not have access to any certificate for x.

This algorithm S^* is called the **simulator** for V^*, as it simulates the outcome of V^*'s interaction with the prover.

Comments

- The zero knowledge condition means that the verifier cannot learn anything new from the interaction, even if she does not follow the protocol but rather uses some other strategy V^*.
- The reason is that she could have learned the same thing by just running the stand-alone algorithm S^* on the publicly known input x.

Statistical and Computational Zero Knowledge

- The perfect zero knowledge condition can be relaxed by requiring that the distributions

$$\text{out}_{V^*}\langle P(x,u), V^*(x)\rangle \quad \text{and} \quad S^*(x)$$

  have instead one of the following:
  - Small statistical distance;
  - Computational indistinguishability.
- The resulting notions are called, respectively:
  - **Statistical zero knowledge**;
  - **Computational zero knowledge**.
- They are central to cryptography and complexity theory.
- The class of languages with statistical zero knowledge proofs, known as SZK, is believed to lie strictly between P and NP.
- In contrast, it is known that, if one-way functions exist, then every NP language has a computational zero knowledge proof.

Revisiting Graph Isomorphism

- We show a perfect zero knowledge proof for the language GI of graph isomorphism.
- GI is in NP and has a trivial proof satisfying completeness and soundness consisting of sending the isomorphism to the verifier.
- Currently, we do not know of a polynomial-time algorithm that can find the isomorphism between two given isomorphic graphs.
- So the proof is not known to be zero knowledge.

Zero-Knowledge Proof for Graph Isomorphism

Zero-Knowledge Proof for Graph Isomorphism

Public input: A pair of graphs G_0, G_1 on n vertices (represented by their adjacency matrices).
Prover's private input: A permutation $\pi : [n] \to [n]$, such that

$$G_1 = \pi(G_0),$$

where $\pi(G)$ denotes the graph obtained by transforming the vertex i into $\pi(i)$ (applying the permutation π to the rows and columns of G's adjacency matrix).

Zero-Knowledge Proof for Graph Isomorphism (Cont'd)

Zero-Knowledge Proof for Graph Isomorphism (Cont'd)

Prover's first message: Prover chooses a random permutation

$$\pi_1 : [n] \to [n].$$

She sends to the verifier the adjacency matrix of $\pi_1(G_1)$.

Verifier's message: Verifier chooses $b \in_R \{0, 1\}$.

He sends b to the prover.

Prover's last message: If $b = 1$, the prover sends π_1 to the verifier.

If $b = 0$, the prover sends $\pi_1 \circ \pi$ to the verifier.

Verifier's check: Let:

- H denote the graph received in the first message;
- π denote the permutation received in the last message.

The verifier accepts if and only if $H = \pi(G_b)$.

Completeness and Soundness of the Protocol

- Completeness: If both the prover and verifier follow the protocol, then the verifier will accept with probability one.
- Soundness: If G_0 and G_1 are not isomorphic, then the verifier will reject with probability at least $\frac{1}{2}$.

  Regardless of the prover's strategy, the graph H that she sends in her first message cannot be isomorphic to both G_0 and G_1.

  There has to exist $b \in \{0, 1\}$, such that H is not isomorphic to G_b.

  But the verifier will choose this value b with probability $\frac{1}{2}$.

  Then, the prover cannot find a permutation π, such that $H = \pi(G_b)$.

  Thus, the verifier rejects.

Zero-Knowledge Property of the Protocol

- Zero-Knowledge: Let V^* be some verifier strategy.

  The simulator S^*:
  - Receives input a pair of graphs G_0, G_1.
  - Chooses $b' \in_R \{0,1\}$, a random permutation π on $[n]$.
  - Computes $H = \pi(G_{b'})$.
  - It then feeds H to the verifier V^* to obtain its message $b \in \{0,1\}^*$.
    - Suppose $b = b'$.
      Then S^* sends π to V^*.
      It outputs whatever V^* outputs.
    - Suppose $b \neq b'$.
      Then the simulator S^* restarts from the beginning.

  S^*'s first message is distributed exactly as the prover's first message, a random graph that is isomorphic to G_0 and G_1.

  So H reveals nothing about the choice of b'.

Zero-Knowledge Property of the Protocol (Cont'd)

- Hence, the probability that $b' = b$ is $\frac{1}{2}$.

  If this happens, then the messages H and π that V^* sees are distributed identically to the distribution of messages that it gets in a real interaction with the prover.

  Now S^* succeeds in getting $b' = b$ with probability $\frac{1}{2}$.

  So the probability that it needs k iterations is 2^{-k}.

  Hence, its expected running time is

$$T(n) \sum_{k=1}^{\infty} 2^{-k} = \mathrm{O}\left(T(n)\right),$$

  where $T(n)$ denotes the running time of V^*.

  Thus, S^* runs in expected probabilistic polynomial-time.

Subsection 5

Some Applications

Introducing Pseudorandom Functions

- Pseudo-random functions are a natural generalization of pseudorandom generators.
- The difference is that the object is a function, whose truth table has exponential size.
- The "blind test" distinguishing algorithm, which runs in polynomial time, only has the ability to ask for the value of the function at any inputs of its choosing.

Pseudorandom Functions

Definition (Pseudorandom Family of Functions)

Let $\{f_k\}_{k\in\{0,1\}^*}$ be a family of functions, such that:

- $f_k : \{0,1\}^{|k|} \to \{0,1\}^{|k|}$, for every $k \in \{0,1\}^*$;
- There is a polynomial-time algorithm that computes $f_k(x)$, given $k \in \{0,1\}^*$ and $x \in \{0,1\}^{|k|}$.

We say that the family is **pseudorandom** if, for every probabilistic polynomial-time oracle Turing machine A, there is a negligible function $\epsilon : \mathbb{N} \to [0,1]$, such that, for every n,

$$\left|\Pr_{k\in_R\{0,1\}^n}\left[A^{f_k(\cdot)}(1^n)=1\right] - \Pr_{g\in_R\mathcal{F}_n}\left[A(g(1^n))=1\right]\right| < \epsilon(n),$$

where $\mathcal{F}_n$ denotes the set of all functions from $\{0,1\}^n$ to $\{0,1\}^n$.

Pseudorandom Functions and Pseudorandom Generators

- Let $\{f_k\}$ be a pseudorandom function family.
- Then, for every polynomial $\ell(n)$, the function G that maps $k \in \{0,1\}^n$ to

$$f_k(1), \ldots, f_k(\ell(n)),$$

where we use some canonical encoding of the numbers $1, \ldots, \ell(n)$ as strings in $\{0,1\}^n$, is a secure pseudorandom generator.

- We also have the following

Theorem

Suppose that there exists a secure pseudorandom generator G with stretch $\ell(n) = 2n$. Then, there exists a pseudorandom function family.

- We prove this next.

The Setup

- Let G be a secure pseudorandom generator, as in the theorem's statement, mapping length-n strings to length-$2n$ strings.

  For every $x \in \{0,1\}^n$, we denote by:
  - $G_0(x)$ the first n bits of $G(x)$;
  - $G_1(x)$ the last n bits of $G(x)$.

  For every $k \in \{0,1\}^n$, we define the function $f_k(\bullet)$ by setting, for all $x \in \{0,1\}^n$,

$$f_k(x) = G_{k_n}(G_{k_{n-1}}(\cdots(G_{k_1}(x))\cdots)).$$

  Note that $f_k(x)$ can be computed by making n invocations of G.

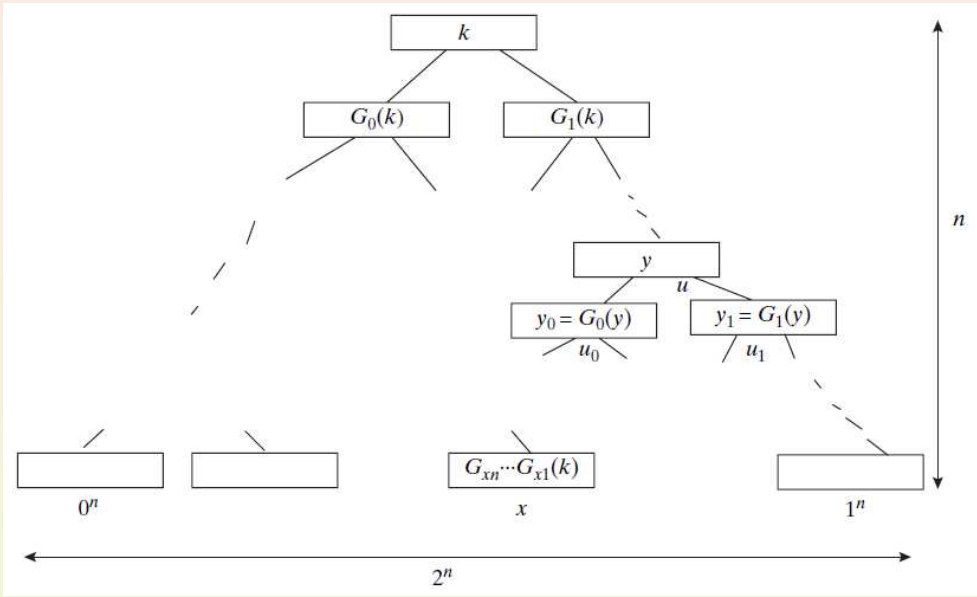
  Hence, it clearly runs in polynomial time.

Graphical Interpretation

- Another way to view f_k is to think of a full depth n binary tree whose root is labeled by k.

  We label the two children of a vertex labeled by y with the values $G_0(y)$ and $G_1(y)$.

  Then $f_k(x)$ denotes the label of the x-th leaf of this tree.



Writing the tree down would take exponential time and space.

However, the label of each leaf can be computed in polynomial time by following the path of length n from the root to the leaf.

Description of the Oracle

- We aim to show that $\{f_k(x)\}$ is pseudorandom.

  We start with a T-time algorithm A that distinguishes between f_{U_n} and a random function with bias ϵ.

  We transform A into a $\text{poly}(n)T$-time algorithm B that distinguishes between U_{2n} and $G(U_n)$ with bias $\frac{\epsilon}{nT}$.

  Assume, without loss of generality, that A makes exactly T queries to its oracle.

  We implement an oracle $\mathcal{O}$ to f_{U_n}.
  - Oracle $\mathcal{O}$ labels vertices of the full binary tree of depth n, as needed.
  - Initially, only the root is labeled by a random string k.
  - Suppose a query of A requires the oracle to label the children u_0, u_1 of a vertex v labeled by y.
  - The oracle invokes G on y to obtain $y_0 = G_0(y)$ and $y_1 = G_1(y)$.
  - It then labels u_0, u_1 with y_0, y_1, and deletes the label y of u.

Comments on the Oracle

- Once u_0 and u_1 are labeled, we have no further need for the label of u.

  Following the definition of f_k, the oracle $\mathcal{O}$ answers a query x with the label of the x-th vertex.

  Note that $\mathcal{O}$ invokes the generator G at most Tn times.

  By adding superfluous invocations, we can assume $\mathcal{O}$ invokes the generator exactly Tn times.

Oracles $\mathcal{O}_i$

- For every $i \in \{0, \ldots, Tn\}$, define the oracle $\mathcal{O}_i$ as follows:

  The oracle $\mathcal{O}_i$ follows the operation of $\mathcal{O}$, but for the first i invocations of G, instead of the labels y_0, y_1 of the children of a node labeled y by setting $y_0 = G_0(y)$ and $y_1 = G_1(y)$, the oracle $\mathcal{O}_i$ chooses both y_0 and y_1 independently at random from $\{0,1\}^n$.

Note that $\mathcal{O}_0$ is the same as the oracle $\mathcal{O}$ to f_{U_n}.

On the other hand, $\mathcal{O}_{nT}$ is an oracle to a completely random function.

Let

$$p_i = \Pr\left[A^{\mathcal{O}_i}(1^n) = 1\right].$$

We may assume $p_{Tn} - p_0 \geq \epsilon$.

We deduce, as in the proof of Yao's Theorem, that

$$E_{i \in_R [Tn]}[p_i - p_{i-1}] \geq \frac{\epsilon}{Tn}.$$

Algorithm B

- Algorithm B, distinguishing U_{2n} from $G(U_n)$, operates as follows.
  - Suppose it receives input $y \in \{0,1\}^{2n}$.
  - Choose $i \in_R [Tn]$ and execute A with access to the oracle $\mathcal{O}_{i-1}$, using random values for the first $i-1$ invocations of G.
  - Then, in the i-th invocation use the value y instead of the result of invoking G.
  - In all the rest of the invocations B runs G as usual.
  - At the end, it outputs what A outputs.

  One can verify that, for every choice of i:
  - If the input y is distributed as U_{2n}, then B's output is distributed as $A^{\mathcal{O}_i}(1^n)$;
  - If the input y is distributed as $G(U_n)$, B's output is distributed as $A^{\mathcal{O}_{i-1}}(1^n)$.

Encryption Using Pseudorandom Function Families

- A pseudorandom function family is a way to turn a random string $k \in \{0,1\}^n$ into an implicit description of an exponentially larger "random looking" string.
- This exponentially lager string consists of the table of all values of the function f_k.
- In practice, for secure communication, one often wants to encrypt many messages with the same key.
- Pseudorandom functions allow Alice and Bob to share an "exponentially large one-time pad".

Encryption Using Pseudorandom Functions (Cont'd)

- Alice and Bob can share a key $k \in \{0,1\}^n$ of a pseudorandom function.
- Suppose Alice wants to encrypt a message $x \in \{0,1\}^n$ for Bob.
- Then Alice chooses $r \in_R \{0,1\}^n$.

  She then sends $(r, f_k(r) \oplus x)$.
- Bob knows the key k.

  So Bob can find x.
- On the other hand, to an adversary who does not know the key it looks as if Alice sent two random strings.
- This happens under the proviso that Alice does not choose the same string r to encrypt two different messages.
- However, under the protocol, this can only happen with exponentially small probability.

Authentication Using Pseudorandom Function Families

- Pseudorandom functions are also used for message authentication codes.
  - Suppose Alice and Bob share a key k of a pseudorandom function.
  - Then, when Alice sends a message x to Bob, she can append the value $f_k(x)$ to this message.
  - Bob can verify that the pair (x, y) he receives satisfies $y = f_k(x)$.
  - An adversary Eve that controls the communication line between Alice and Bob cannot change the message x to x' without being detected. The probability that Eve can predict the value of $f_k(x')$ is negligible (after all, a random function is unpredictable).
- Furthermore, pseudorandom function generators have also figured in a very interesting explanation of why current lower bound techniques ave been unable to separate P from NP.

Derandomization

- The existence of pseudorandom generators implies subexponential deterministic algorithms for BPP, often referred to as **derandomization** of BPP.
- If $L \in \mathrm{BPP}$, then for every $\epsilon > 0$, there is a 2^{n^ϵ}-time deterministic algorithm A, such that for every sampleable distribution of inputs $\{X_n\}$, where $X_n \in \{0,1\}^n$,

$$\Pr[A(X_n) = L(X_n)] > 0.99.$$

- The randomness is only over the choice of the inputs, whereas the algorithm A is deterministic.
- The algorithm A works by:
  - Reducing the randomness requirement of the probabilistic algorithm for L to n^ϵ using a pseudorandom generator;
  - Enumerating over all the possible inputs for the pseudorandom generator.

Tossing Coins Over the Phone

- Suppose two parties A and B want to toss a fair random coin over the phone.
- If only one of them actually tosses a coin, there is nothing to prevent him from lying about the result.
- The following "fix" suggests itself.
  - Both players toss a coin and they take the XOR as the shared coin.
  - Even if B does not trust A to use a fair coin, he knows that as long as his bit is random, the XOR is also random.
- Unfortunately, this idea also does not work because the player who reveals his bit first is at a disadvantage.
- The other player could just "adjust" his answer to get the desired final coin toss.

The Bit Commitment Solution

- The following scheme assumes that A and B are polynomial-time Turing machines that cannot invert one-way permutations.
  - A chooses two strings x_A and r_A of length n.
    She sends a message
    $$(f_n(x_A), r_A),$$
    where f_n is a one-way permutation.
  - B selects a random bit b and sends it to A.
  - A reveals x_A.
  - They agree to use the XOR of b and $(x_A \odot r_A)$ as their coin toss.

Correctness of the Bit Commitment Solution

- We can see that no party can cheat.
- B can verify that x_A is the same as in the first message by applying f_n.

  Therefore, A cannot change her mind after learning B's bit.
- A's first message is called a **cryptographic commitment** to the bit $x_A \odot r_A$.
- By the Goldreich-Levin Theorem, B cannot predict $x_A \odot r_A$ from A's first message.

  Thus, she cannot bias her bit according to the choice of $x_A \odot r_A$.

Multiparty Computations

- There are k parties.
- Known to all of them is a polynomial-time computable function

$$f : \{0,1\}^{nk} \to \{0,1\}.$$

- The i-th party holds a string $x_i \in \{0,1\}^n$.
- They wish to compute

$$f(x_1, x_2, \ldots, x_k).$$

- The parties can just exchange their inputs (suitably encrypted if need be so that unauthorized eavesdroppers learn nothing).
- Then each of them can compute f on his or her own.
- This leads to all of them knowing each other's input, which may not be desirable.

Secure Multiparty Computations

- A multiparty protocol for computing f is **secure** if, at the end, no party learns anything new apart from the value

$$f(x_1, x_2, \ldots, x_k).$$

- The formal definition (inspired by zero knowledge) says that whatever a party or a coalition of parties learn during the protocol can be simulated in an ideal setting where they only get to send their inputs to some trusted authority that computes f on these inputs and broadcasts the result.
- Amazingly, there are protocols to achieve this task securely for every number of parties and for every polynomial-time computable f.

Machine Learning

- In machine learning the goal is to learn a succinct function

$$f : \{0,1\}^n \to \{0,1\}$$

  from a sequence $(x_1, f(x_1)), (x_2, f(x_2)), \ldots$, where the x_i's are randomly chosen inputs.
- This is impossible, in general, since a random function has no succinct description.
- Suppose f has a succinct description (e.g., as a small circuit).
- The existence of pseudorandom functions implies that, even though a function may be polynomial-time computable, there is no way to learn it from examples in polynomial time.
- It is possible to extend this impossibility result to more restricted function families such as NC^1.