Cryptography

Nate Annau

Table of Contents

1. Introduction

Lecture 1 Jan 6

1.1. Cryptographic Systems

1.1.1. Definition: Cryptographic System

A **cryptographic system** is a structure or scheme consisting of a set of algorithms that converts plaintext to ciphertext to encode or decode messages securely.

When constructing a cryptographic system we use the following process:
1) Problem
2) Definition
3) Construction
4) Proof

1.1.2. Definition: Encrypt, Decrypt, Ciphertext, Plaintext

Suppose we have a message M written in **plaintext**, i.e., understandable language. If we **encrypt** the message, we rewrite it in such a way that the data can be recovered (called **decrypting**), but it is difficult to know how to do this without knowing some secret. The encrypted message is called **ciphertext**.

1.1.3. Definition: Private Key Encryption

How do we ensure that an eavesdropper Eve can't intercept the message when Alice sends it to Bob? One way is to **generate** a **secret key** which Alice and Bob share, which Alice uses to encrypt the plaintext and Bob uses to decrypt the ciphertext.

1.1.4. Definition: Kerckhoff's Principle

One might argue that we could just keep an encryption algorithm secret rather than using secret keys. However, this is dangerous if an eavesdropper discovers the scheme. In general, **Kerckhoff's Principle** states that an ideal cryptographic system should be secure even if attackers know the algorithm, which necessarily involves some kind of secret key.

Lecture 2 Jan 8

1.1.5. Definition: Encryption Scheme

An encryption scheme is a cryptographic system with the following three algorithms:

1) Key Generation
   - takes an input λ (in unary)
   - generates a key $k \in \{0,1\}^\lambda$
2) Encryption
   - input: (k, m) where m is the message
   - output: ciphertext c
3) Decryption
   - input: (k, c)
   - output: m

1.1.6. Definition: Probabilistic and Deterministic Algorithm

A **probabilistic algorithm** is an algorithm using some amount of randomness. We can write it $A(x; r)$ where x is the input, r is some randomness, and A is the algorithm.

A **deterministic algorithm** is a probabilistic algorithms that involves no randomness. Notice that deterministic algorithms are a proper subset of probabilistic algorithms.

Note that we can say without loss of generality that the three main algorithms of an encryption scheme are probabilistic.

1.1.7. Remark

We desire that our algorithm has these two properties:

1) **Correctness**, meaning $m = m'$
2) **Security**, meaning given ct and not k, we cannot recover m.

1.1.8. Definition: Polynomial Runtime

An algorithm A runs in polynomial time if $\exists C \in \mathbb{N}$ such that $\forall x$,

$$\frac{\mathrm{RUNTIME}(A(x))}{|x|^C} < \infty.$$

Call the function $\mathrm{RUNTIME}(A(x))$ by $f(|x|)$.

1.1.9. Definition: Polylog Runtime

An algorithm A runs in polylog time if it runs in some polynomial in $\log \lambda$ time.

1.1.10. Remark

1) Notice that $|k| = \lceil \log_2 \lambda \rceil$
2) We want our algorithms to be efficient, which here just means they run in polynomial time.
3) We want our algorithms to be secure against an arbitrarily computationally capable Eve, at least to a realistic degree.
4) We need a better metric for security, since if Eve can recover part of the message the encryption should not be defined as secure. We might instead say that Eve should not be able to recover any information in m, but she still might be able to figure out that the message belongs to some subset of possible plaintext messages, which isn't great either. This leads to a new definition for security.

1.1.11. Definition: Secure Encryption Scheme

Whatever an eavesdropper Eve learns from the ciphertext, she could have generated the same information herself. In particular, if Eve tries to create a probability distribution of possible plaintext given the ciphertext, she must generate the same distribution as a third party with no knowledge of the ciphertext.

1.1.12. Definition: Caesar Cipher

Let m be a single letter in the English alphabet. Let the secret key $k \in \{1, 2, \ldots, 26\}$.

- To encrypt m, shift the letter k places to the right. Call the resulting letter m'.
- To decrypt m', shift the letter k places to the left.

We can visualize this as turning a wheel. More generally, if m is the nth letter in the English alphabet, then the encrypted letter is $m + k \bmod 26$.

If there are many letters, we can apply the substitution for every letter in m. We ignore whitespace and are capitalization invariant.

Notice this cipher can be easily bruteforced.

1.1.13. Example

$\text{Enc}(2, c) = \text{Enc}(k, c) = e$ and $\text{Enc}(z, c) = b$.

1.1.14. Definition: Vigenère Cipher

Similar to the Caesar cipher, except instead of a single key we have a longer key. The k-th element of the key shifts the i-th element of the message.

This is insecure because we can perform a frequency analysis attack. Noting that the English language has commonly occurring patterns, we can start cracking specific words.

1.1.15. Example

Suppose we have key (3, 20)

- Shift T by 3 places $\to$ W
- Shift h by 20 places $\to$ b

Lecture 3 Jan 13

1.1.16. Example: Polynomial Runtime

Suppose $\text{RUNTIME}(A(\lambda)) \geq \lambda$. Thus if we write λ as a bit string, we have $\text{RUNTIME}(A(\lambda)) = f(\log_2(\lambda))$ which implies $f(\log_2(\lambda)) \geq \lambda$. But this implies that f cannot be a polynomial, since in that case f would be polylogarithmic and always dominated by λ.

1.2. One Time Pad

1.2.1. Definition: Exclusive Or

The **Exclusive Or**, or **XOR**, is a binary operation with the following truth table:

In 1	In 2	Out
0	0	0
0	1	1
1	0	1
1	1	0

We use the notation $x \oplus y$ to denote the exclusive or, with $x, y \in \{0,1\}$.

Note if $x = \{0,1\}^n$ and $y = \{0,1\}^n$ then we define $x \oplus y = (x_1 \oplus y_1, ..., x_n \oplus y_n)$.

1.2.2. Example: Probabilistic XOR

Fix a and pick b uniformly at random. (Notationally, we can write $b \xleftarrow{\$} \{0,1\}$).

We want to find $a \oplus b$. Note that

$$\begin{aligned}\Pr[a \oplus b = 0] &= \Pr[a \oplus b = 0 \mid b = 0] \cdot \Pr[b = 0] + \Pr[a \oplus b = 1 \mid b = 1] \cdot \Pr[b = 1] \\ &= \frac{1}{2}(\Pr[a = 0] + \Pr[a = 1]) \\ &= \frac{1}{2}\left(\frac{1}{2} + \frac{1}{2}\right) \\ &= \frac{1}{2}.\end{aligned}$$

1.2.3. Definition: One Time Pad (Vernam's Cipher)

- KeyGen(1^n) outputs $k \xleftarrow{\$} \{0,1\}^n$.
- Enc(k, m) takes as input $k, m \in \{0,1\}^n$ and outputs $c = k \oplus m$.
- Dec(k, c) outputs $m = k \oplus c$

1.2.4. *Proposition*

In the One Time Pad, for all $k, m \in \{0,1\}^n$ it holds that $\mathrm{Dec}(k, \mathrm{Enc}(k, m)) = m$

Proof: Note that

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

□

1.2.5. Remark

Let us informally analyze security from Eve's view.

From the viewpoint of Eve, the following happens

- Alice has an n bit message m
- Alice samples some key k uniformly at random from the space of n bit strings and then outputs $c = k \oplus m$

1.2.6. Example: Eve's View

Pr	k	$c = k \oplus 010$
1/8	000	010
1/8	001	011
1/8	010	000
1/8	011	001
1/8	100	110
1/8	101	111
1/8	110	100
1/8	111	101

Note that Eve cannot deduce anything since the probabilities are all equal. In particular, $\forall s \in \{0,1\}^3$, the probability that the ciphertext is s is 1/8.

For all m, the ciphertext is uniformly distributed. But Eve herself can sample from this distribution without knowing the message, which is intuitively what it means for the scheme to be secure.

1.2.7. Definition: Correct Encryption Algorithm

An encryption scheme satisfies **correctness** if for all possible keys k and for all possible messages m, the following holds:

$$\Pr[\mathrm{Dec}(k, \mathrm{Enc}(k, m)) = m] = 1.$$

We use probability because Enc is allowed to be a randomized algorithm.

1.2.8. Remark

Ideal properties for security:
- The secret key should be kept hidden from Eve
- The key is only used to encrypt plaintext
- The ciphertext cannot be decrypted without the key

1.2.9. Definition: Secure Encryption Algorithm

Note: this is only one definition, there are others to follow.

We say that an encryption algorithm is is **one time uniform ciphertext secure** if $\forall m \in \mathcal{M}$ chosen by Eve, the ciphertext is uniformly distributed (over the ciphertext space $\mathcal{C}$), i.e., the following distributions are identical:
1) $\mathcal{D}_1 := \{c := \mathrm{Enc}(k, m); k \leftarrow \mathrm{KeyGen}(1^\lambda)\}$
2) $\mathcal{D}_2 := \left\{c \xleftarrow{\$} \mathcal{C}\right\}$

(Note an insecure scheme is one where these distributions are not the same.)

1.2.10. Example: Secure Encryption Scheme

Is the following secure?
- $\mathrm{KeyGen}(1^n) := k \xleftarrow{\$} \{0,1\}^n$
- $\mathrm{Enc}(k, m) := c = k \wedge m$

It is not because for $m = 0^n$, notice

$$\Pr[c = 0^n \mid \mathcal{D}_1] = 1$$

$$\Pr[c = 0^n \mid \mathcal{D}_2] = \frac{1}{2^n}$$

so the distributions are not equal.

1.2.11. Definition: Alternate Definition for Security

1.2.12. Example

Consider the following interactions between Eve and a challenger:

Eve sends the challenger a message m and Eve sends back c. The goal of the challenger is to prevent Eve from learning anything. The goal of Eve is to know the two distributions.

1) The challenger has a bit $c := \text{Enc}(k, m)$ and $k = \text{KeyGen}(1^n)$
2) The challenger sends a bit selected at random $c \xleftarrow{\$} \mathcal{C}$

An encryption scheme is secure if for any chosen m by Eve, the above two scenarios seem identical to Eve.

Lecture 4 Jan 15

1.2.13. *Lemma*

> One Time Pad encryption scheme satisfies uniform ciphertext security.

Proof: Given a fixed plaintext m and a fixed ciphertext c, we calculate the probability that c is the encryption of m:

$$\Pr[c = \text{Enc}(k, m)] = \Pr[c = m \oplus k] = \Pr[k = m \oplus c] = \frac{1}{2^n}$$

Note that the probability here is over the random choice of $k \in \{0,1\}^n$.

□

1.2.14. Example: Double One Time Pad

Suppose our scheme is like this:

$$\text{KeyGen}(1^n) : k_1 \xleftarrow{\$} \{0,1\}^n, k_2 \xleftarrow{\$} \{0,1\}^n \text{ and output } (k_1, k_2)$$
$$\text{Enc}((k_1, k_2), m) : c_1 = k_1 \oplus m, c_2 = k_2 \oplus c_1 \text{ and output } c_2$$
$$\text{Dec}((k_1, k_2), c) : c_1 = k_2 \oplus c, m = k_1 \oplus c_1 \text{ and output } m.$$

We need to show that for each m, the distributions are identical:
1) $\{c_2 = k_2 \oplus c_1; k \leftarrow \text{KeyGen}(1^n), k_2 \leftarrow \text{KeyGen}(1^n), c_1 = k_1 \oplus m\}$
2) $\left\{c_2 \xleftarrow{\$} \{0,1\}^n\right\}$

Proof: We consider the following set of distributions called hybrids:

$$\mathcal{H}_1 : \{c_2 = k_2 \oplus c_1; k_1 \leftarrow \text{KeyGen}(1^n), k_2 \leftarrow \text{KeyGen}(1^n), c_1 = k_1 \oplus m\}$$
$$\mathcal{H}_2 : \left\{c_2 \xleftarrow{\$} \{0,1\}^n; k_1 \leftarrow \text{KeyGen}(1^n), c_1 = k_1 \oplus m\right\}$$
$$\mathcal{H}_3 : \left\{c_2 \xleftarrow{\$} \{0,1\}^n\right\}$$

Note $\mathcal{H}_1$ and $\mathcal{H}_2$ are the same because one time pad satisfies uniform encryption security and $\mathcal{H}_2$ and $\mathcal{H}_3$ are the same trivially.

□

1.2.15. Definition: One Time Perfect Security

We say that an encryption scheme is one time perfectly secure if $\forall m_0, m_1 \in \mathcal{M}$ chosen by Eve, the following distributions are identical:

$$\mathcal{D}_1 := \{c := \text{Enc}(k, m_0); k \leftarrow \text{KeyGen}(1^n)\}$$
$$\mathcal{D}_2 := \{c := \text{Enc}(k, m_1); k \leftarrow \text{KeyGen}(1^n)\}$$

Thus the ciphertext carries no information from Eve's viewpoint.

1.2.16. ***Theorem***

> One time uniform ciphertext security $\implies$ one time perfect security

Proof: Consider the following distributions:

$$\mathcal{H}_1 : \{c := \text{Enc}(k, m_0); k \leftarrow \text{KeyGen}(1^n)\}$$
$$\mathcal{H}_2 : \left\{c \xleftarrow{\$} \mathcal{C}\right\}$$
$$\mathcal{H}_3 : \{c := \text{Enc}(k, m_1); k \leftarrow \text{KeyGen}(1^n)\}$$

where $\mathcal{H}_1 \equiv \mathcal{H}_2$ and $\mathcal{H}_2 \equiv \mathcal{H}_3$ both follow from one time uniform ciphertext security.

Thus the one time pad satisfies one time perfect security.

□

1.2.17. Remark

The converse is false. Counterexample:

- $\text{KeyGen}(1^n) : k \xleftarrow{\$} \{0,1\}^n$
- $\text{Enc}(k, m)$: Compute $c' = k \oplus m$ and output $c = c' \| 00$
- $\text{Dec}(k, c)$: Compute $c' = c[0 : n]$ and output $m = k \oplus c'$

Note the distribution is not uniform but it is one time perfectly secure.

1.2.18. Remark

There are some serious issues with one time pad:

- Key must be as long as the plaintext
- A key cannot be used to encrypt more than one plaintext

1.3. Computational Intractability

1.3.1. Remark

A brute force attack for an n bit key should take $O(2^n)$ time.

An attack should be made computationally infeasible for an algorithm to be secure, not necessarily impossible.

1.3.2. Example: Efficient vs Inefficient Algorithms

Efficient algorithms (polynomial time):
1) GCD
2) Arithmetic mod N
3) Inverses mod N
4) Exonentials mod N

Ineffcient algorithms:
1) Factoring integers
2) Discrete logarithm
3) Square roots mod composite N
4) Solving "noisy" linear equations

1.3.3. Remark

If people can guess a key, it's bad even if there's a low probability. Thus we want a new definition of security to follow these two factors:
- Attacks that are expensive as a brute force attack
- Attacks whose success probability is as low as a blind guess attack

Lecture 5 Jan 22

1.3.4. Example

If an adversary's probility f breaking one time perfect security is $\frac{1}{n}$, then if they receive n^2 ciphertexts, that should be able to break n messages on average. This is because the expectation is $\frac{1}{n} \cdot n^2 = n$.

1.3.5. Definition: Negligible Function

A function $\nu(\cdot)$ is **negligible** if for every polynomial $p(\cdot)$, we have

$$\lim_{n\to\infty} p(n)\nu(n) = 0.$$

Alternatively, $\nu(n)$ is negligible if $\forall c \exists n_0$ such that $\forall n > n_0, \nu(n) \leq \frac{1}{n^c}$.

1.3.6. *Proposition*

Let f and g be negligible. Show that $f + g$ is negligible.

Proof: We need to show that $\forall c \exists n_0$ such that $\forall n > n_0, f(n) + g(n) \leq \frac{1}{n^c}$.

Fix $c \in \mathbb{N}$. Since f and g are negligible we know $\exists n_f, n_g$ such that $\forall n > n_f, f(n) \leq \frac{1}{n^{c+1}}$ and $\forall n > n_g, g(n) \leq \frac{1}{n^{c+1}}$.

Then if $n > \max\{n_f, n_g, 2\}$ we have

$$f(n) + g(n) \leq \frac{1}{n^{c+1}} + \frac{1}{n^{c+1}} = \left(\frac{2}{n}\right)\left(\frac{1}{n^c}\right) \leq \frac{1}{n^c}$$

□

1.3.7. *Proposition*

Let ν be a negligible function and p be a polynomial such that $p(n) \geq 0 \forall n > 0$. Show that $\nu(n) \cdot p(n)$ is negligible.

Proof: Since p is a polynomial, we know $\exists n_p, c_p$ such that $\forall n > n_p, p(n) \leq n^{c_p}$. Since ν is negligible, we know that $\exists n_\nu$ corresponding to $c + c_p$ such that $\forall n > n_\nu, \nu(n) \leq \frac{1}{n^{c+c_p}}$. For a given c, let $n_0 = \max\{n_\nu, n_p\}$. Then $\forall n > n_0$ we have

$$\nu(n)p(n) \leq \frac{1}{n^{c+c_p}} \cdot n^{c_p} \leq \frac{1}{n^c}.$$

□

1.3.8. Definition: Ensemble

A sequence $\{X_n\}_{n \in \mathbb{N}}$ is called an **ensemble** if for each $n \in \mathbb{N}$, X_n is a probability distribution over $\{0,1\}^*$.

1.3.9. Example

Suppose Adversary $\mathcal{A}$ tries to guess whether a sample was drawn from two distributions X and Y. If X is the uniform distribution on $\{0,1\}^n$ and Y is a fixed point distribution, then if the adversarial guesses a string of all zeros, it can tell them apart almost all of the time.

Lecture 6 Jan 27

1.3.10. Definition: Probabilistic Polynomial Time (PPT)

Decision problems solvable by a probabilistic Turing machine in polynomial time.

1.3.11. Definition: Computational Indistinguishability

Let $X = \{X_n\}_{n\in\mathbb{N}}$ and $Y = \{Y_n\}_{n\in\mathbb{N}}$ be probability ensembles.

Then X and Y are **computationally indistinguishability** if $\forall$ PPT $\mathcal{A}, \exists \nu(\cdot)$ such that

$$|\Pr[x \leftarrow X_n; \mathcal{A}(1^n, x) = 1] - \Pr[y \leftarrow Y_n; \mathcal{A}(1^n, y) = 1]| \leq \nu(n).$$

where the adversary $\mathcal{A}$ is trying to guess the distribution given the n in the ensemble.

This quantity is called the **advantage** or bias of $\mathcal{A}$.

1.3.12. Example

Let X_1 be a uniform distribution on $\{0,1\}^L$ and Y_1 be a fixed point distribution, so that $\exists y^* \in \{0,1\}^L$ so $\Pr[y^* \leftarrow Y_1] = 1$.

Define $\mathcal{A}(1^n, y)$ so that if it receives y^* it outputs 1, otherwise it outputs 0. Therefore $\Pr[1 \leftarrow \mathcal{A}(1^n, y) : y \leftarrow Y_1] = 1$ but $\Pr[1 \leftarrow \mathcal{A}(1^n, x) : x \leftarrow X_1] = \frac{1}{2^L}$. Now let $\nu(n)$ be a negligible function. Observe $\lim_{n\to\infty} |1 - \frac{1}{2^L}| = 1$ but $\lim_{n\to\infty} \nu(n) = 0$, implying $\exists N$ such that $\forall n > N$, $|1 - \frac{1}{2^L}| > \nu(n)$, showing that X_1 and Y_1 are computationally indistinguishable.

1.3.13. Notation

$\{X_n\} \underset{c}{\approx} \{Y_n\}$ means computational indistinguishability.

1.3.14. *Proposition*

If we apply an efficient operation on X and Y, they remain computationally indistinguishable. In other words, $\forall$ nonuniform PPT M,

$$\{X_n\} \underset{c}{\approx} \{Y_n\} \implies \{M(X_n)\} \underset{c}{\approx} \{M(Y_n)\}$$

Proof: Suppose by contradiction that $\{M(X_n)\}$ and $\{M(Y_n)\}$ were computationally distinguishable. Then if $\mathcal{A}$ is the distinguisher, notice $\mathcal{A}(M(X))$ allows us to distinguish X_n and Y_n, a contradiction.

□

1.3.15. *Lemma:* Hybrid Lemma

Let $X^1, \ldots, X^m$ be distribution ensembles for $m = \text{poly}(n)$. If for every $i \in \{1, \ldots, m-1\}, X^i$ and X^{i+1} are computationally indistinguishable, then X^1 and X^m are computationally indistinguishable.

Proof: Transitivity via triangle inequality + apply repeatedly.

□

1.3.16. Definition: Multi-Message Security

For all messages $\{(m_0^i, m_1^i)\}_{i\in[q(n)]}$, where q has a polynomial runtime and n is a security parameter, the following distributions are identical:

- $\left\{\text{Enc}(s, m_0^1), ..., \text{Enc}\left(s, m_0^{q(n)}\right)\right\}$
- $\left\{\text{Enc}(s, m_1^1), ..., \text{Enc}\left(s, m_1^{q(n)}\right)\right\}$

Note this implies that the encryption algorithm must be probabilistic, because the simple attack of letting $m_0^1 = \cdots = m_0^{q(n)}$ would break it otherwise.

2. Pseudorandomness

2.1. Pseudorandom Generators

2.1.1. Computational Problem

Suppose you have n uniformly random bits: $x = x_1 \| \cdots \| x_n$. We want to find a deterministic polynomial time algorithm G such that

- $G(x)$ outputs $n+1$ bits $y = y_1 \| \cdots \| y_{n+1}$
- y looks "as good as" a truly random string $r = r_1$

That is, the following are computationally indistinguishable:

$$\left\{ G(x) : x \xleftarrow{\$} \{0,1\}^n \right\}; \quad \left\{ r : r \xleftarrow{\$} \{0,1\}^{n+1} \right\}$$

2.1.2. Definition: Pseudorandom Generator (PRG)

A deterministic algorithm G is called a **pseudorandom generator (PRG)** if

- G can be computed in polynomial time
- $|G(x)| > |x|$
- $\left\{ G(x) : x \xleftarrow{\$} \{0,1\}^n \right\} \approx_c \left\{ U_{\ell(n)} \right\}$ where $\ell(n) = |G(0^n)|$

The **stretch** of G is defined to be $|G(x)| - |x|$

2.1.3. Definition: Pseudo One Time Pad

We use a PRG with the ideas of one time pad for the following:

- $\text{Gen}(1^n)$:
  - $s \xleftarrow{\$} \{0,1\}^n$
  - outputs $k = s$
- $\text{Enc}(k, x \in \{0,1\}^n)$:
  - output $c = G(s) \oplus x$
- $\text{Dec}(k, c)$:
  - output $m = G(s) \oplus c$

Correctness is clear.

2.1.4. *Proposition*

Pseudo One Time Pad is Secure

Proof: We proceed using the Hybrid Argument. Let $x \in \{0,1\}^m$. Then

$$\mathcal{H}_1 = \left\{ G(S) \oplus X : S \xleftarrow{\$} \{0,1\}^n \right\}$$

$$\mathcal{H}_2 = \left\{ z \oplus X : z \xleftarrow{\$} \{0,1\}^n \right\}$$

$$\mathcal{H}_3 = U_{\{0,1\}^n}$$

Note $\mathcal{H}_1 \underset{c}{\approx} \mathcal{H}_2$ by Proposition 1.3.14. Then $\mathcal{H}_2 \underset{c}{\approx} \mathcal{H}_3$ since one time pad satisfies one time uniform ciphertext security.

□

Lecture 8 Feb 5

2.1.5. Algorithm

We can convert a PRG with a 1 bit stretch G_{one} to a PRG with an m bit stretch G_{poly} by iteratively passing the initial PRG:

$S_0 = S, y = \varepsilon$ (empty string)
for $i = 1$ to m:
  $(x_1 \cdots x_n x_{n+1}) \leftarrow G_{\text{one}}(S_{i-1})$
  y = CONCATENATE(y, x_{n+1})
  $S_i = x_1 \cdots x_n$
OUTPUT y

2.1.6. *Proposition*

The previous construction is secure.

Proof: Assume by contraposition that G_{poly} is not secure. Consider the following hybrids:

$$\mathcal{H}_1 = \left\{ y \leftarrow G_{\text{poly}}(s) : s \xleftarrow{\$} \{0,1\}^n \right\}$$
$$\mathcal{H}_2 = \left\{ y \leftarrow G^{(1)}_{\text{poly}}(s) : s \xleftarrow{\$} \{0,1\}^n \right\}$$
$$\vdots$$
$$\mathcal{H}_{m+1} = \left\{ u \xleftarrow{\$} \{0,1\}^m \right\}$$

Then $\exists i \in \{1, ..., m\}$ such that $\mathcal{H}_i \overset{c}{\not\approx} \mathcal{H}_{i+1}$. Thus there exists a distinguisher D for these two.

REDUCTION(z):

$s_i = z$
$y \xleftarrow{\$} \{0,1\}^i$
for $j = i + 1$ to m:
  $(x_1 \cdots x_n x_{n+1}) \leftarrow G_{\text{one}}(S_{i-1})$
  y = CONCATENATE(y, x_{n+1})
  $S_i = x_1 \cdots x_n$
OUTPUT $D(y)$

Then $\Pr[\text{REDUCTION outputs } 1 \mid z = G_{\text{one}}(s)] = \Pr[D \text{ outputs } 1 \mid y \leftarrow \mathcal{H}_i]$, showing that G_{one} is not secure.

□

2.2. One Way Functions

2.2.1. Definition: One Way Function

$f : \{0,1\}^n \to \{0,1\}^m$ is called **one way** if
1) f is computable in deterministic polynomial time
2) $\forall$ PPT $\mathcal{A}$, $\Pr\left[x' \leftarrow \mathcal{A}(1^n, f(x)) \text{ such that } f(x') = f(x); x \xleftarrow{\$} \{0,1\}^n\right)] \leq \nu(n)$.

I.e., given a function image, the probability that an optimal adversary guesses a preimage right is negligible.

The existence of such a function would prove P $\neq$ NP.

2.2.2. Example: Functions that aren't One Way

1) $f(x) = x_1 \oplus x_2 \oplus \cdots \oplus x_n$ (note $m = 1$).
2) $f(x) = \sum_{i=1}^{n} x_i 2^{x_i}$ (binary representation of x)
3) $f(x) = \sum_{i=1}^{n} 2^{x_i}$

Lecture 9 Feb 10

2.2.3. Example: Finding Prime Factors

Consider a number $N = pq$ where p, q are primes. We want a deterministic algorithm that takes N and returns p and q. Assuming we can check divisibility in an $O(1)$ step, the naive running time is then $O(N)$ (or $O\left(\sqrt{N}\right)$ with a simple optimization). But this is in terms of the size of the input, not the length of the input $\lambda \sim \log N$. Thus this algorithm has an exponential running time.

2.2.4. Definition: One Way Permutation (OWP)

f is a **One Way Permutation** if
- f is one-way
- f is a permutation (nonidentity bijection)

2.2.5. Definition: Predicate

A **predicate** of x is a function $P : S \to \{0, 1\}$ for some set $x \in S$.

2.2.6. Example: Predicate

We could have $P : \mathbb{Z} \to \{0, 1\}$ be defined by "is a multiple of 7".

2.2.7. Definition: Hard-core Predicate

A **hard-core predicate** of a function f is a predicate of x (an input to f) which can be easily computed given the preimage x but is difficult to compute given the image $f(x)$.

This is formalized for an ensemble of predicates h_r as follows:
- $\mathcal{H} = \{h_r\}_{r \in \{0,1\}^n}$ with respect to $f : \{0, 1\}^n \to \{0, 1\}^n$
- $h_r : \{0, 1\}^n \to \{0, 1\}$ is a deterministic poly time algorithm
- $\forall$ PPT $\mathcal{A}, \Pr\left[h_r(x) \leftarrow \mathcal{A}(r, f(x)); r \xleftarrow{\$} \{0, 1\}^n, x \xleftarrow{\$} \{0, 1\}^n\right] \leq \frac{1}{2} + \nu(n)$

A typical example is a hashing function.

2.2.8. *Theorem:* Goldreich-Levin Theorem

If f is a one way function, then hard-core predicates exist.

Proof: Sketch: define $h_r(x)$ by the boolean inner product

$$\langle r, x \rangle = r_1 \wedge x_1 \oplus r_2 \wedge x_2 \oplus \cdots \oplus r_n \wedge x_n = \sum_i r_i x_i.$$

Note we can think of $(\wedge, \oplus)$ as $(\cdot, +)$ over mod 2.

By contradiction, assume an adversary can break the hard-core predicate property. I.e., $\exists$ PPT $\mathcal{A}$ such that when $\mathcal{A}$ is given r and $f(x)$, it can find a u such that $\Pr[u = \langle r, x \rangle] > \frac{1}{2} + \nu(n)$. Then we can show $\exists$ PPT $\mathcal{B}$ such that $\mathcal{B}(f(x)) \to x'$ such that $\Pr[f(x') = f(x)] \geq \frac{1}{\text{poly}}$, contradicting that f is one way.

To do this, first consider an easier case where $\Pr[u = \langle r, x \rangle] = 1$. Then consider $r = 10 \cdots 0$ so that $\langle r, x \rangle$ is the first bit of x. By choosing r's like this we can fully reconstruct x, which is a trivial preimage. The idea of the proof is to iteratively relax this assumption and use a similar technique.

□

2.2.9. Remark

Notice that hard-core predicates may not exist if f is not one way. For example, take the identity function $f(x) = x$. Then we don't lose any information, so trivially a hard-core predicate cannot exist.

2.2.10. *Lemma:* Next-Bit Unpredictability Lemma

Suppose $\mathcal{D}$ is efficiently sampleable. Then $\mathcal{D}$ is pseudorandom if and only if the probability of predicting the ith bit given the first $i - 1$ bits is at most a negligible improvement over random.

I.e., we have that $x \leftarrow \mathcal{D} \implies \Pr[x_i = \mathcal{A}(x_1, \ldots, x_{i-1})] \leq \frac{1}{2} + \nu(n)$ is an equivalent definition to the computational indistinguishability definition of pseudorandomness.

2.2.11. ***Proposition***

Let $f : \{0,1\}^n \to \{0,1\}^n$ be a one way permutation. Given a PRG $G : \{0,1\}^{2n} \to \{0,1\}^{2n+1}$, we have that $G(s\|r) = (f(s)\|r\|h_r(s))$ is secure.

Proof: Proceed by the Hybrid Technique.

$$\mathcal{H}_1 = \left\{ G(s\|r) = y_1 y_2 \cdots y_{2n+1}; s \xleftarrow{\$} \{0,1\}^n, r \xleftarrow{\$} \{0,1\}^n \right\}$$

$$\mathcal{H}_2 = \left\{ G(s\|r) = y_1 y_2 \cdots y_{2n} u_{2n+1}; s \xleftarrow{\$} \{0,1\}^n, r \xleftarrow{\$} \{0,1\}^n, u_{2n+1} \xleftarrow{\$} \{0,1\} \right\},$$

$$\mathcal{H}_3 = \left\{ G(s\|r) = y_1 y_2 \cdots y_{2n-1} u_{2n} u_{2n+1}; s \xleftarrow{\$} \{0,1\}^n, r \xleftarrow{\$} \{0,1\}^n, u_{2n} \xleftarrow{\$} \{0,1\}, u_{2n+1} \xleftarrow{\$} \{0,1\} \right\}$$

$$\vdots$$

$$\mathcal{H}_{2n+1} = \left\{ u_1 \cdots u_{2n+1}; u_i \xleftarrow{\$} \{0,1\} \right\}$$

Notice that $\mathcal{H}_1 = \mathcal{H}_2$ by the Next-Bit Unpredictability Lemma (since the condition is given by the hard-core predicate). Then $\mathcal{H}_3 = \cdots = \mathcal{H}_{2n+1}$ clearly, also following the lemma.

□

2.3. Pseudorandom Functions

Lecture 10 (Jesse transcribed)	Feb 12

2.3.1. Definition: Pseudorandom Function (PRF)

A function

$$F : \{0,1\}^\lambda \times \{0,1\}^n \longrightarrow \{0,1\}^m$$
$$(k, x) \longmapsto y$$

where k is the key and x is the input. F should be computable in deterministic polynomial time.

2.3.2. Definition: Random Function

We have

$$G_{n,m} = \{g : \{0,1\}^n \to \{0,1\}^m\}$$

where $g \xleftarrow{\$} G_{n,m}$.

2.3.3. Notation

$\mathcal{A}^f$ means $\mathcal{A}$ has oracle access to f. This means that given an input x_i, $\mathcal{A}$ knows what $f(x_i)$ is by queries $\{x_1, ..., x_t\}$. Note that querying a function is a polynomial time algorithm.

F is a secure PRF if

$$\left|\Pr\left[1 \leftarrow \mathcal{A}^{F(k,\cdot)} : k \xleftarrow{\$} \{0,1\}^\lambda\right] - \Pr\left[1 \xleftarrow{\$} \mathcal{A}^g : g \xleftarrow{\$} G_{n,m}\right]\right| \leq \nu(\lambda).$$

Since querying a function is polynomial time, we can only query a function polynomial many times since $\mathcal{A}$ is a PPT.

Lecture 11 Feb 19

2.3.4. Example: PRF Encryption Scheme

Let $F : \{0,1\}^\lambda \times \{0,1\}^n \to \{0,1\}^m$ be a PRF.

- Define $\text{Gen}(1^\lambda)$ sampling $k \xleftarrow{\$} \{0,1\}^\lambda$ and then outputting k.
- Define $\text{Enc}(k, x)$ by sampling $r \xleftarrow{\$} \{0,1\}^n$ and then outputting $(r, F(k, r) \oplus x)$.
- Define $\text{Dec}(k, c)$ by decomposing $c = (r, \theta)$ and outputting $\theta \oplus F(k, r)$.

2.3.5. *Proposition*

The previous encryption scheme is correct and satisfies multi-message security.

Proof: We begin by showing correctness. Observe

$$\begin{aligned}\mathrm{Dec}(k, \mathrm{Enc}(k, x)) &= \mathrm{Dec}(k, (r, F(k,r) \oplus x)) \\ &= (F(k,r) \oplus x) \oplus F(k,r) \\ &= x.\end{aligned}$$

Now to show multi-message security, we proceed by the Hybrid Technique.

$\mathcal{H}_1 = \{(\mathrm{Enc}(k, m_0^1), ..., \mathrm{Enc}(k, m_0^t)\}$

$\mathcal{H}_2 = \left\{(r_1, F(k,r_1) \oplus m_0^1), ..., (r_t, F(k,r_t) \oplus m_0^t) : r_1, ..., r_t \xleftarrow{\$} \{0,1\}^n\right\}$

$\mathcal{H}_3 = \left\{\perp \text{ if } \exists i,j \text{ such that } r_i = r_j \wedge i \neq j \text{ else } ((r_1, F(k,r_1) \oplus m_0^1), ..., (r_t, F(k,r_t) \oplus m_0^t)) : r_1, ..., r_t \xleftarrow{\$} \{0,1\}^n\right\}$

$\mathcal{H}_4 = \left\{\perp \text{ if } \exists i,j \text{ such that } r_i = r_j \wedge i \neq j \text{ else } ((r_1, \mathcal{U}_1 \oplus m_0^1), ..., (r_t, \mathcal{U}_t \oplus m_0^t)) : \mathcal{U}_1, ...\mathcal{U}_t \xleftarrow{\$} \{0,1\}^m\right\}$

$\mathcal{H}_5 =$

Define Hybrid 3 by sampling $r_1, ..., r_t \xleftarrow{\$} \{0,1\}^n$, and then if $\exists i,j$ such that $r_i = r_j \wedge i \neq j$ then output "JUNK"; otherwise, output $((r_1, F(k,r_1) \oplus m_0'), ..., (r_t, F(k,r_t) \oplus m_0^t))$.

Define Hybrid 4 to be the same as Hybrid 3 except $\mathcal{U}_1, ...\mathcal{U}_t \xleftarrow{\$} \{0,1\}^m$ and output $((r_1, \mathcal{U}_1 \oplus m_0'), ..., (r_t, \mathcal{U}_t \oplus m_0^t))$. Define Hybrid 5 to be the same as Hybrid 4 except we output $((r_1, \mathcal{U}_1 \oplus m_1'), ..., (r_t, \mathcal{U}_t \oplus m_1^t))$. Define Hybrid 6 to be $((r_1, F(k,r_1) \oplus m_1'), ..., (r_t, F(k,r_t) \oplus m_1^t))$. Define Hybrid 7 to not check if r_i's are distinct. Define Hybrid 8 to be $\{\mathrm{Enc}(k, m_1^1), ..., \mathrm{Enc}(k, m_1^t))\}$.

Now Hybrid 1 $\underset{c}{\approx}$ Hybrid 2 by construction, and Hybrid 2 $\underset{c}{\approx}$ Hybrid 3 (prob of bad event is negligible).

Note that $\Pr[\exists i,j \text{ such that } i \neq j \wedge r_i = r_j] \leq \sum_{i \neq j} \Pr[r_i = r_j] = \binom{t}{2} \cdot \frac{1}{2^n} \leq \frac{t^2}{2^n}$ (we invoked the union bound).

Now Hybrid 4 $\underset{c}{\approx}$ Hybrid 5 follows from Perfect Security of OTP and Hyrbrid Argument. Hybrid 6 $\underset{c}{\approx}$ Hybrid 7 follows from the same reason as $2 = 3$.

Also 7 = 8 follows by construction.

To show 3 = 4, suppose by contradiction there exists a distinguisher $\mathcal{D}$ such that $\mathcal{D}$ distinguishes 3 and 4 with $\frac{1}{\mathrm{poly}}$ advantage. We proceed with a reduction $R^{\mathcal{O}}$. $D \to ((m_0^1, m_1^1), ..., (m_0^t, m_1^t))$ and query $\mathcal{O}$ on $(r_1, ..., r_t)$, giving us $(y_1, ..., y_t)$. Then $\mathcal{D}((r_1, y_1 \oplus m_0^1), ..., (r_t, y_t \oplus m_0^t)) = b$. In case 1, $\theta = F(k, \cdot)$ and the input to $\mathcal{D}$ is Hybrid 3, and in case 2 $\theta = g(\cdot)$, adnd the input to $\mathcal{D}$ is hybrid 4.

□

2.3.6. *Proposition*

Given a pseudorandom generator $G : \{0,1\}^\lambda \longrightarrow \{0,1\}^{2\lambda}$, we can always create a pseudorandom function $F : \{0,1\}^\lambda \times \{0,1\}^n \longrightarrow \{0,1\}^\lambda$.

Proof: We begin with the $n = 1$ case. Our input is (k, x) where $k \in \{0,1\}^\lambda$ and $x \in \{0,1\}$. Take $G(k) = (y_0, y_1)$, where y_0 is the first λ bits and y_1 is the last λ characters. Then if $x = 0$, we output y_0 and if $x = 1$, we output y_1.

In the $n = 2$ case, given y_0 and y_1, we invoke the PRG on both of them. I.e., $G(y_0) = (y_{00}, y_{01})$ and $G(y_1) = (y_{10}, y_{11})$ and then choose the subscript corresponding to $x \in \{0,1\}^2$. In general, running G on every output of the previous iteration works. To avoid exponential runtime, just choose the path given by x.

In general, $F(k, x)$ is defined as follows:

$y_\varepsilon = k$
for $i = 1$ to n:
  $G(y_{x^{(i-1)}}) = \left(y_{x^{(i-1)}\|0}, y_{x^{(i-1)}\|1}\right)$
Output y_x

□

2.3.7. *Proposition*

The previous construction is secure.

Proof: First consider the case where $x = 0 \cdots 0$. Notice that since G is secure and we have closure by Proposition 1.3.14,

$$\mathcal{H}_1 = \left\{F(k, 0 \cdots 0) : k \xleftarrow{\$} \{0,1\}^\lambda\right\}$$
$$\mathcal{H}_2 = \left\{y_0 : y_0 \xleftarrow{\$} \{0,1\}^\lambda\right\}$$
$$\vdots$$
$$\mathcal{H}_n = \left\{y \xleftarrow{\$} \{0,1\}^\lambda\right\}$$

The general case follows a similar idea.

□

3. Hash Functions

3.1. Introduction

Lecture 13 Feb 26

3.1.1. Definition: Collision Resistance

Given an adversary $\mathcal{A}$ that wants to find $x, y \in \{0,1\}^n$ such that $h(x) = h(y)$ with $x \neq y$, a hash function has **collision resistance** if $P[\text{ADVERSARY WINS}] = \text{negl}(m)$.

3.1.2. Definition: Hash Function

A **hash function** is a function h_k in the family $\mathcal{H} = \{h_k : \{0,1\}^n \longrightarrow \{0,1\}^m, k \in \{0,1\}^\lambda\}$ where $n \gg m$ and h_k is a deterministic poly time algorithm. A hash function is called **secure** if it has collision resistance.

3.1.3. Definition: Domain Extension

Given $\mathcal{H} = \{h_k : \{0,1\}^{2\lambda} \longrightarrow \{0,1\}^\lambda\}$, we can find a family $G_{\text{poly}} = \{g_k : \{0,1\}^n \longrightarrow \{0,1\}^\lambda$ for $n = \text{poly}(\lambda)$. This is called a **domain extension**. A naive implementation would be to hash the first λ bits and the second λ bits, then hash the result of that with the third λ bits, and iteratively do this until we get down to λ bits.

Formally, $g_k(x_1, ..., x_N)$ is defined by

1 $y = x_1$
2 for $i = 1$ to $N - 1$:
3 | $y = h(y, x_{i+1})$

3.1.4. *Proposition*

> Given that $\mathcal{H}$ is collision resistant, the above construction G is collision resistant.

Proof: By contradiction, suppose G is not collision resistant. In other words, $\Pr[\mathcal{A}(g_k) \rightarrow x^1, x^2 \text{ such that } x^1 \neq x^2 \wedge g_k(x^1) = g_k(x^2)] \geq \text{poly}(\lambda)$. If we run g_k on both x^1 and x^2, we get the same thing. Now consider the $N - 1$ step in g_k: if the inputs are different, we immediately contradict that $\mathcal{H}$ is collision resistant. Thus we can assume they are the same, but now we work backwards to the $N - 2$ step. We can do this inductively so that we eventually get that $x^1 \neq x^2$, also a contradiction.

We could have also thought about this proof as a reduction, where we find an adversary that can always come up with inputs $a \neq b$ such that $h_k(a) = h_k(b)$

□

3.1.5. Definition: Merkle-Damgard Transform

A better way to approach the previous problem is to combine in groups of two: always take adjacent inputs and put them through h_k. The collision resistance proof is similar to the previous proposition.

3.2. Random Oracle Model

Lecture 14 Mar 3

3.2.1. Definition: Random Oracle Model (ROM)

In the **Random Oracle Model (ROM)**, the adversary uses a hash function as a black box, where the hash function behaves like a random function.

3.2.2. Concept: Collision Resistance in ROM

If $h : \{0,1\}^n \to \{0,1\}^m$ with $n > m$ is a hash function (modeled as a random function), then the adversary succeeds if $\mathcal{A}^h \longrightarrow (x, y)$ such that $h(x) = h(y) \wedge x \neq y$.

For an upper bound:
- If $\mathcal{A}$ makes $o\left(\sqrt{2^m}\right)$ queries, then $P[\mathcal{A} \text{ wins}] = \nu(m)$.

For a lower bound:
- If $\mathcal{A}$ makes $O\left(\sqrt{2^m}\right)$ queries, then $P[\mathcal{A} \text{ wins}] \geq \frac{1}{4}$

3.2.3. Concept: Lazy Sampling Technique

In the beginning we have a table of all values in $\{0,1\}^n$, where every one is not known. Now $\mathcal{A}$ makes a query for some $q_1 \in \{0,1\}^n$. Sample $y \xleftarrow{\$} \{0,1\}^m$ and return y to $\mathcal{A}$. Then $\mathcal{A}$ submits $q_2 \in \{0,1\}^n$. If $q_2 = q_1$, return $h(q_1)$; otherwise, sample $y' \xleftarrow{\$} \{0,1\}^n$, update the table, and return y'.

Input	Assigned Output
0...00	X
0...00	X
⋮	⋮
q_1	y
⋮	⋮

This shows that we sample only queries made by the adversary; other queries will not be assigned. This is a convenient model for a hash function.

3.2.4. *Proposition*

The upper bound on the adversary's success in finding a hash collision is correct. I.e., if $\mathcal{A}$ makes a series of t queries $\{q_1, ..., q_t\}$ using lazy sampling, if $t = o\left(\sqrt{2^m}\right)$, we have $P[\mathcal{A} \text{ wins}] = \nu(m)$.

Proof: $\mathcal{A}$ makes a series of t queries $\{q_1, ..., q_t\}$ using lazy sampling.

Case 1:

- For $\mathcal{A}$'s output (x, y), $x \notin \{q_1, ..., q_t\}$ or $y \notin \{q_1, ..., q_t\}$. Then $P[\mathcal{A} \text{ succeeds}] = \frac{1}{2^m}$ since without loss of generality we can consider x as fixed in $\{q_t, ..., q_t\}$ and y as an unassigned string, and then y is assigned randomly.

Case 2:

- For $\mathcal{A}$'s output (x, y), $x \in \{q_1, ..., q_t\} \wedge y \in \{q_1, ..., q_t\}$ Now

$$P[\mathcal{A} \text{ succeeds}] = \Pr\left[\exists i, j : h(q_i) = h(q_j)\right] \leq \frac{\binom{t}{2}}{2^m} \leq \frac{t^2}{2^m} = \frac{o(2^m)}{2^m} = \nu(m).$$

□

3.2.5. *Proposition*

The lower bound is correct. That is, if we can make $t = O\left(\sqrt{2^m}\right)$ queries (This is the **birthday attack.**)

Proof: Given $\mathcal{A}^h$, we query $\{q_1, ..., q_t\}$ distinct where $q_i \xleftarrow{\$} \{0,1\}^n$. If $h(q_i) = h(q_j)$ then output (q_i, q_j); otherwise, fail.

We claim $P[\mathcal{A} \text{ wins}] \leq \frac{1}{4}$. Observe

$$\begin{aligned}\Pr[\nexists \text{ collision}] &= \Pr[h(q_t) \notin \{h(q_1), ..., h(q_{t-1})\} : \nexists \text{ collision in } \{q_1, ..., q_{t-1}\}] \cdot P[\nexists \text{ collision in } \{q_1, ..., q_{t-1}\}] \\ &= \prod_{i=1}^{t-1}\left(1 - \frac{i}{2^m}\right) \leq \frac{3}{4}.\end{aligned}$$

□

3.3. Public Key Encryption

Lecture 15	Mar 5

3.3.1. Concept: Public Key Encryption

Suppose Alice and Bob have public keys pk_{ALICE} and pk_{BOB} respectively. They also each have a secret key sk_{ALICE} and sk_{BOB}.

To send messages to Bob, Alice uses Bob's private key via $\text{Enc}(\text{pk}_{\text{BOB}}, m)$; Bob uses Alice's public key via $\text{Enc}(\text{pk}_{\text{ALICE}}, m)$.

In public key encryption, anyone can encrypt messages (unlike private key encryption).

3.3.2. Definition: Chosen Plaintext Attack (CPA) Security

Consider a PPT adversary $\mathcal{A}$ that has knowledge of the public key. Suppose Alice and Bob want to send a bit $b \in \{0, 1\}$, and the adversary guesses the bit is b' based on $\text{Enc}(\text{pk}, m)$.

Our public key encryption algorithm has **chosen plaintext security** if

$$|\Pr[b' = 1 \mid b = 0] - \Pr[b' = 1 \mid b = 1]| \leq \nu(n).$$

3.3.3. Definition: Multi-message CPA Security

Suppose the challenger gives the adversary the public key, the adversary queries with multiple messages $(m_0^1, m_1^1), ..., (m_0^q, m_1^q)$, and the adversary receives the encrypted messages. If the adversary is unable to guess the original message, it has **multi-message CPA Security**.

3.3.4. *Proposition*

> CPA Security $\implies$ multi-message CPA Security

Proof: We proceed by the Hybrid Technique:

$$\begin{aligned}
\mathcal{H}_1 &= \{\text{Enc}(\text{pk}, m_0^1), \text{Enc}(\text{pk}, m_0^2), ..., \text{Enc}(\text{pk}, m_0^q)\} \\
\mathcal{H}_2 &= \{\text{Enc}(\text{pk}, m_1^1), \text{Enc}(\text{pk}, m_0^2), ..., \text{Enc}(\text{pk}, m_0^q)\} \\
\mathcal{H}_3 &= \{\text{Enc}(\text{pk}, m_1^1), \text{Enc}(\text{pk}, m_1^2), ..., \text{Enc}(\text{pk}, m_0^q)\} \\
&\vdots \\
\mathcal{H}_q &= \{\text{Enc}(\text{pk}, m_1^1), \text{Enc}(\text{pk}, m_1^2), ..., \text{Enc}(\text{pk}, m_1^q)\}
\end{aligned}$$

To prove these hybrids are computationally indistinguishable, consider Hybrids 1 and 2 and proceed by reduction. Suppose there exists a distinguisher $\mathcal{D}$ that can distinguish these two hybrids. Then $\mathcal{D}$, with access to the public key pk, sends (m_0^1, m_1^1) to the challenger, receiving a ciphertext. But then everything else is the same, so this breaks CPA security.

□

3.3.5. Definition: Factoring Assumption

If a challenger samples n bit primes (pq) and sets $N = pq$, then any PPT adversary can guess p or q only with negligible probability.

3.3.6. *Theorem:* (Weak) Prime Number Theorem

> For a fixed n, the number of n bit integers that are prime is at least $\frac{1}{3n}$.

3.3.7. Algorithm: GenPrime(1^n)

```
for i = 1 to 3n^2
    sample p' ←$ {0,1}^{n-1}
    set p = 1 ‖ p'
    check if p is prime (highly nontrivial)
    if so: output p
```

Now notice

$$\Pr[\text{GenPrime}(1^n) \to \text{Fail}] = \Pr([\text{1 iteration fails}])^{3n^2} < \left(1 - \frac{1}{3n}\right)^{3n^2} \approx \frac{1}{e^n}$$

which is negligible. We noted that $\left(1 - \frac{1}{x}\right)^x \approx \frac{1}{e}$ for large x.

3.3.8. Algorithm: GenModulus(1^n)

```
p ← GenPrime(1^n)
q ← GenPrime(1^n)
if p = q:
    | output "FAIL"
else
    | output (p · q, p, q)
```

Thus we can reformulate the Factoring Assumption as GenModulus(1^n) giving (N, p, q).

3.3.9. Example

Define $f(x, y)$ by if x and y Given $f(x, y) = pq$ where (p, q) are primes sampled from x and y respectively, there is no guarantee