Revision notes - CS4236

Ma Hongqiang

January 30, 2019

Contents

1 Classical Cipher System

Definition 1.1 (Syntax of Encryption).

- $\mathcal{M}$: Message space, which a set of legal messages
- `Gen`: Procedure for generating key. It should be a probabilistic algorithm that outputs a key $k \in K$ chosen according to some distribution.
- `Enc`: Procedure for encrypting. Takes as input a key k and message m and outputs a ciphertext c. Denote by $\texttt{Enc}_k(m)$.
- `Dec`: Procedure for decrypting. Takes as input a key k and a ciphertext c and outputs a plaintext m. Denote by $\texttt{Dec}_k(c)$.

Theorem 1.1 (Correctness Requirement).
An encryption scheme must satisfy the correctness requirement:

$$\boxed{\text{For every key } k \text{ output by } \texttt{Gen} \text{ and every message } m \in \mathcal{M}, \text{ we have } \texttt{Dec}_k(\texttt{Enc}_k(m)) = m}$$

Theorem 1.2 (Kerckhoff's Principle).
Security should rely *only* on the secrecy of the key.

Therefore, there should never be "security by obsecurity".

1.1 Classical Cryptography

Definition 1.2 (Caesar's Cipher).
Original Caesar Cipher encrypts messages by shifting the letters of the alphabet 3 places forward.
Problem: There is **no key**.
Keyed shift cipher, which shifts k(key) places forward, is also vulnerable to **brute-force or exhaustive search**.
Also, key space is too small(26).

Theorem 1.3 (Sufficient Key-space Principle).
Any secure encryption scheme must have a key space that is sufficiently large to make an exhaustive-search attack infeasible.

Definition 1.3 (Substitution Cipher).
Substitution cipher encrypts messages by mapping plaintext alphabet to ciphertext alphabet according to a **bijection**.
Problem: Vulnerable to **Frequency analysis** of alphabet or n-gram.

Definition 1.4 (Vigenere Cipher).
Vigenere cipher has a key word, which is repeated until its length is the same as the plaintext. Then the shifting is done according to the key.
Problem:

1. When the length(t) of the key is known,
    - Divide ciphertext into t parts
    - Perform statistical analysis for each part
2. When the length is unknown, but max length T is known
    - Repeat (1) T times

Apart from the breaking algorithm above, it is also vulnerable to the Kasiski's method.

1.2 Principles of Modern Cryptography

Definition 1.5 (Security Definition)**.**
Security Definition is a tuple

1. **Security guarantee**: what the scheme is intended to prevent the attack from doing; and
2. **Threat model**: the capability of the adversary

Typically, the security guarantee is to make it impossible for an attacker to **learn any additional information** aboue the underlying plaintext, whereas the threat model can be either one of the below:

- Ciphertext-only attack: most basic attack
- Known-plaintext attack: attacker learns plaintext/cipphertext pairs
- Chosen-plaintext-attack: attacker obtains plaintext/ciphertext pairs for plaintext of its choice
- Chosen-ciphertext attack: attacker obtains plaintext/ciphertext pairs for ciphertexts of its choice

1.3 Perfectly Secret Encryption

There are two equivalent definition of perfect secrecy.

Definition 1.6 (Perfect Secrecy)**.**
An encryption scheme(`Gen`, `Enc`,`Dec`) with message space $\mathcal{M}$ is **perfectly secret** if for *every* probability distribution over $\mathcal{M}$, every message $m \in \mathcal{M}$, and every ciphertext $c \in \mathcal{C}$ for which $P(C = c) > 0$, we have

$$P(\mathcal{M} = m|\mathcal{C} = c) = P(\mathcal{M} = m)$$

Definition 1.7 (Perfect Secrecy Equivalent Definition)**.**
For every $m, m' \in \mathcal{M}$, and every $c \in \mathcal{C}$, we have

$$P(\texttt{Enc}_K(m) = c) = P(\texttt{Enc}_K(m') = c)$$

The equivalency can be derived by using the lemma below

$$P(\mathcal{C} = c \mid \mathcal{M} = m) = P(\mathcal{C} = c)$$

is equivalent to the perfect secrecy definition.

Definition 1.8 (Perfect Indistinguishability)**.**
Consider an experiment with an adversary $\mathcal{A}$ and an encryption oracle $\Pi = (\texttt{Gen}, \texttt{Enc}, \texttt{Dec})$. The process goes like this:

1. $\mathcal{A}$ sends $m_0, m_1 \in \mathcal{M}$ to Π
2. Π generates key k using `Gen`; picks 0 or 1 with equal chance and set as b; send back $\texttt{Enc}_k(m_b)$.
3. $\mathcal{A}$ outputs the guess of b as b'.

We say that Π is perfectly indistinguishable if for *every* $\mathcal{A}$, it holds that

$$P(b' = b) = \frac{1}{2}$$

It is known that perfect indistinguishability is equivalent to perfect secrecy.

Definition 1.9 (Vernam's One Time Pad)**.**
Fix an integer $l > 0$. The space of $\mathcal{M}, \mathcal{K}, \mathcal{C}$ are all equal to $\{0, 1\}^l$. We define the oracle as

- `Gen`: choose a key from $\mathcal{K}$ with uniform distribution
- `Enc`: $c = k \oplus m$.
- `Dec`: $m = k \oplus c$.

It is known that one time pad has perfect secrecy. However, the problem of perfect secrecy is that

- The key should be at least as long as the message
- The key can be used only once

2 Computational Secrecy

The perfect secrecy is unnecessarily strong, and therefore we relax the constraint from unlimited computational power and perfect indistinguishability to efficient computational power, and a weaker version of indistinguishability, called computational indistinguishability.

Definition 2.1 (Computationally (t, ϵ)-indistinguishable)**.**
Encryption scheme $\Pi = (\texttt{Gen}, \texttt{Enc}, \texttt{Dec})$ is (t, ϵ)-indistinguishable if for every $\mathcal{A}$ running in time at most t, it holds that

$$P(\text{PrivK}^{eav}_{\mathcal{A},\Pi} = 1) \leq \frac{1}{2} + \epsilon$$

The choice of t and ϵ is troublesome. We can define computational indistinguishability in *asymptotic* approach.

Definition 2.2 (Asymptotically Secure)**.**
A scheme is secure if any **probabilistic polynomial-time**(PPT) adversary succeeds in breaking the scheme with at most **negligible probability**.
Here, an algorithm A runs in polynomial time, if there exists a polynomial p such that for every input $x \in \{0, 1\}^*$, the computation of $A(x)$ terminates within at most $p(|x|)$ steps. An algorithm is probabilistic, if it can access an unbiased random bit at each step.
A function is negligible if for every positive polynomial p, there is an N, such that for all integer $n > N$, it holds that $f(n) < \frac{1}{p(n)}$.

Remark:

- suppose f, g is negligible, and p polynomial, then $f + g$ and $f \times p$ are both negligible.
- Both the running time and the probability can be parameterised by a single variable n, which is typically the key length.

2.1 Proof by Reduction

To proof computational security of a scheme requires proving that the scheme cannot be broken by any polynomial-time algorithm, which is not possible now. Therefore, we *need to at least assume* some low-level problems, such as factorisation of large integers, are hard to solve. We can then use these assumptions, which are conjectured to be true, to prove the security of the schemes.

Definition 2.3 (Proof by Reduction)**.**
We **assume** some problem X cannot be solved by any polynomial time algorithm, except with negligible probability. We proceed to **prove** that some cryptographic construction Π is secure. To do this,

1. Fix some efficient, i.e. probabilistic polynomial-time adversary $\mathcal{A}$ attacking Π. Denote the adversary's success probabilistic by $\epsilon(n)$.
2. Construct an efficient adversary $\mathcal{A}'$ that attempts to solve problem X using adversary $\mathcal{A}$ as a subroutine. Therefore, given some instance x of problem X, $\mathcal{A}'$ should simulate, for $\mathcal{A}$, an execution of Π such that

   (a) As far as $\mathcal{A}$ can tell, it is interacting with Π.

   (b) Furthermore, if $\mathcal{A}$ succeeds in breaking the execution of Π that is being simulated by $\mathcal{A}'$, this should allow $\mathcal{A}'$ to solve the instance x given, with at least inverse polynomial probability $\frac{1}{p(n)}$.

3. Point(2) implies that if $\epsilon(n)$ is not negligible, then $\mathcal{A}'$ will solve X with non-negligible probability $\frac{\epsilon(n)}{p(n)}$. Also, $\mathcal{A}'$ is efficient, since it runs a PPT $\mathcal{A}$ as a subroutine. Therefore, we arrive at the conclusion that there is an efficient algorithm solving X with non-negligible probability, which contradicts the initial assumption.

4. Therefore, $\epsilon(n)$ must be negligible, which means that $\mathcal{A}$ is computationally secure.

2.2 Security for Encryption

Before we define the security for encryption, we first define private key encryption scheme.

Definition 2.4 (Private Key Encryption)**.**
A private-key encryption scheme is a PPT algorithm $\Pi = (\texttt{Gen}, \texttt{Enc}, \texttt{Dec})$ such that

- `Gen` outputs a key k with length n.
- `Enc` takes as input a key k and a plaintext message $m \in \{0,1\}^*$, and outputs a ciphertext $c = \texttt{Enc}_k(m)$.
- `Dec` takes as input a key k and a ciphertext c, and outputs a message $m = \texttt{Dec}_k(c)$.

We then define what we called, semantic security.

Definition 2.5 (Semantic Security)**.**
A private key encryption scheme Π is **semantically secure in the presense of an eavesdropper** if any probabilistic polynomial time algorithm $\mathcal{A}$ are infeasible to learn any partial information about the plaintext from the ciphertext.

The exact definition is mathematically complex and is not stated here. However, this definition of semantic security is equivalent to the indistinguishable encryption in the presense of an eavesdropper, whose definition is given below:

Definition 2.6 (Indistinguishable Encryption)**.**
A private-key encryption scheme Π has **indistinguishable encryptions in the presence of an eavesdropper** if for all probabilistic polynomial-time adversaries $\mathcal{A}$, there exists a negligible function **negl** such that

$$P(\mathrm{PrivK}^{\mathrm{eav}}_{\mathcal{A},\Pi}(n)) \leq \frac{1}{2} + \mathbf{negl}(n)$$

for all n, where n is the security parameter.

Note that the experiment now takes in n, which is used to generate the random key k.
We then introduce an equivalent definition as the above, which is used later in some proof.

Definition 2.7 (Equivalent Indistinguishable Encryption).
Define $\text{PrivK}^{\text{eav}}_{\mathcal{A},\Pi}(n,b)$ to be the same as above, except that the fixed bit b is used, rather than being chosen by Π at random. In addition, denote the output bit b' from $\mathcal{A}$ in $\text{PrivK}^{\text{eav}}_{\mathcal{A},\Pi}(n,b)$ by $\text{output}(\text{PrivK}^{\text{eav}}_{\mathcal{A},\Pi}(n,b))$. The following definition essentially states that $\mathcal{A}$ cannot determine whether it is running in experiment $\text{PrivK}^{\text{eav}}_{\mathcal{A},\Pi}(n,0)$ or $\text{PrivK}^{\text{eav}}_{\mathcal{A},\Pi}(n,1)$.

A private key encryption scheme Π has indistinguishable encryption in the presence of eavesdropper if for all probabilistic polynomial-time adversaries $\mathcal{A}$, there exists a negligible function **negl** such that

$$|P(\text{output}(\text{PrivK}^{\text{eav}}_{\mathcal{A},\Pi}(n,0))=1)-P(\text{output}(\text{PrivK}^{\text{eav}}_{\mathcal{A},\Pi}(n,1))=1)|\leq \mathbf{negl}(n)$$

2.3 Pseudorandom Generators(PRG)

Definition 2.8 (Pseudorandom Generator).
Let $l(\cdot)$ be a polynomial and let G be a deterministic polynomial-time algorithm such that upon any input $s\in\{0,1\}^n$, algorithm G outputs a string of length $l(n)$.

We say that G is a **pseudorandom generator** if

1. **Expansion**: For every n, it holds that $l(n)>n$.
2. **Pseudorandomness**: For all probabilistic polynomial-time(PPT) distinguishers D, there exists a negligible function **negl** such that

   $$|P(D(r)=1)-P(D(G(s))=1)|\leq \mathbf{negl}(n)$$

   where r is chosen uniformly at random from $\{0,1\}^{l(n)}$, the seed s is chosen uniformly at random from $\{0,1\}^n$.

Here, l is the expansion factor of G.

With PRG, we construct an encryption scheme that has indistinguishable encryptions in the presence of an eavesdropper.

Definition 2.9 (Construction).
Let G be a pseudorandom generator with expansion factor l. Define a private-key encryption scheme for messages of legnth l as follows:

- `Gen`: on input of n, choose $k\leftarrow\{0,1\}^n$ uniformly at random and output it as the key
- `Enc`: on input a key $k\in\{0,1\}^n$ and a message $m\in\{0,1\}A^{l(n)}$, output the ciphertext

  $$c:=G(k)\oplus m$$

- `Dec`: on input a key $k\in\{0,1\}^n$ and a ciphertext $c\in\{0,1\}^{l(n)}$, output the plaintext message

  $$m:=G(k)\oplus c$$

We claim the above construction has indistinguishable encryption in presence of eavesdropper.
Proof Suppose there is a PPT $\mathcal{A}$ such that it violates the definition of indistinguishable encryption, then we can construct a PPT that distinguishes the output of G from a truly random string.
Let $\mathcal{A}$ be a PPT adversary, and define ε as

$$\varepsilon(n) := P[\mathrm{PrivK}^{\mathrm{eav}}_{\mathcal{A},\Pi}(n) = 1] - \frac{1}{2}$$

We use $\mathcal{A}$ to construct a distinguisher D for pseudorandom generator G, such that D can also succeed with probability $\epsilon(n)$.
The distinguisher is given a string w as input, and it is to determine whether w is chosen uniformly at random, or whether w is generated by choosing a random k and computing $w := G(k)$. D emulates the eavesdropping experiment for $\mathcal{A}$ and observes whether $\mathcal{A}$ succeeds or not. If $\mathcal{A}$ succeeds then D guesses that w must have been a pseudorandom string, while if $\mathcal{A}$ does not succeed then D guesses w is a random string. In detail:

Distinguisher D:

D is given as input a string $w \in \{0,1\}^{l(n)}$.

1. Run $\mathcal{A}$ to obtain the pair of messages $m_0, m_1 \in \{0,1\}^{l(n)}$.
2. Choose a random bit $b \leftarrow \{0,1\}$. Set $c := w \oplus m_b$.
3. Give c to $\mathcal{A}$ and obtain output b'. In a sense, we output 1 if $b' = b$ and 0 otherwise.

We also, define a modified encryption scheme $\tilde{\Pi}$ that is exactly one-time pad encryption, i.e. `Gen` can output a completely random key k of length $l(n)$. By perfect secrecy of one-time pad, we have

$$P[\mathrm{PrivK}^{\mathrm{eav}}_{\mathcal{A},\Pi}(n) = 1] = \frac{1}{2}$$

The main observations are:

1. If w is chosedn uniformly at random from $\{0,1\}^{l(n)}$, then the view of $\mathcal{A}$ when run as a subroutine by the distinguisher D is distributed identically to the view of $\mathcal{A}$ in teh experiment $\mathrm{PrivK}^{\mathrm{eav}}_{\mathcal{A},\tilde{\Pi}}(n)$. This is because $\mathcal{A}$ is given a ciphertext $c = w \oplus m_b$ where w is completely random.
2. On the other hand, if $w = G(k)$, where $k \leftarrow \{0,1\}^n$ is chosen uniformly at random, the the view of $\mathcal{A}$ when run as a subroutine by D is distributed identically to the view of $\mathcal{A}$ in experiment $\mathrm{PrivK}^{\mathrm{eav}}_{\mathcal{A},\Pi}(n)$. This is because $\mathcal{A}$ is given a ciphertext $c = w \oplus m_b$ where $w = G(k)$ for a uniformly distributed value $k \in \{0,1\}^n$.

Therefore, if w is chosen uniformly at random

$$P(D(w) = 1) = \frac{1}{2}$$

In contrast, when $w = G(k)$, we have

$$P(D(w)) = P(D(G(k))) = \frac{1}{2} + \varepsilon(n)$$

Therefore,

$$|P(D(w) = 1) - P(D(G(k)) = 1)| = \varepsilon(n)$$

where w is randomly chosen in $\{0,1\}^{l(n)}$ and k randomly chosen in $\{0,1\}^n$. Since G is a pseudorandom generator, ε must be negligible by its definition.

3 CPA Security

Our definition of $\mathrm{PrivK}^{\mathrm{eav}}_{\mathcal{A},\Pi}$, although can be used to prove the security of encryption, has several limitations:

- It only considers ciphertext-only attacks, whereas *real* attackers may have more capabilities
- It only considers single ciphertext observation, whereas *real* attackers may observe more than one ciphertext

From this section onwards, we give more capabilities to adversary.

Definition 3.1 (Chosen Plaintext Attack)**.**
Chosen Plaintext Attack is an attack in which adversary can influence the sender to encrypt message $m_1, \ldots$ with the *same* unknown key k and observe the corresponding ciphertext $c_1, \ldots$.

Definition 3.2 (CPA Security)**.**
Experiment $\mathrm{PrivK}^{\mathrm{cpa}}_{\mathcal{A},\Pi}$ for encryption scheme Π and adversary $\mathcal{A}$ is defined as below:

- $k \leftarrow \mathtt{Gen}(1^n)$.
- $\mathcal{A}$ interacts with $\mathtt{Enc}_k()$.
- $\mathcal{A}$ outputs m_0, m_1. Oracle calculates b randomly chosen from $\{0, 1\}$ and sends $c \leftarrow \mathtt{Enc}_k(m_b)$.
- $\mathcal{A}$ continues interacting with $\mathtt{Enc}_k()$.
- $\mathcal{A}$ outputs b'. $\mathcal{A}$ succeeds if $b = b'$, in which we say $\mathrm{PrivK}^{\mathrm{cpa}}_{\mathcal{A},\Pi} = 1$.

Π is secure against chosen plaintext attacks, i.e. **CPA secure** if for all PPT adversaries $\mathcal{A}$, there is a negligible function **negl** such that

$$P(\mathrm{PrivK}^{\mathrm{cpa}}_{\mathcal{A},\Pi}) \leq \frac{1}{2} + \mathbf{negl}(n)$$

Remark:

- Security against chosen-plaintext attacks is nowadays the **minimal** notion of security an encryption scheme should satisfy.

- CPA-secure encryption must be probablistic.
  It is because if the encryption scheme is deterministic, i.e. $\mathtt{Enc}(m) = c$ for all tries of same m. Then we can just resend m_0 and m_1 to the encryption oracle and determine the ciphertext's corresponding plaintext. This results in a success probability of 1.

3.1 Random Function

Let Func_n denotes all possible functions that maps $\{0,1\}^n$ to $\{0,1\}^n$. Note that, we requires Func_n to be surjective, but not necessarily injective.
There are a total of 2^{n2^n} possible f's.

Definition 3.3 (Keyed Function)**.**
A **keyed function** F is a two-input function $F : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ where the first input is called the **key**, denoted by k and second input called input.

In general, k will be fixed and F_k, a single-input function is of our concern. Essentially, choosing k is the same as choosing a function.
Here,we also assume that F is **length preserving**, so that the key, input and output length of F are all the same.
A keyed function is **efficient** if there is a PPT algorithm that computes $F(k,x)$ given k and x as input.

Definition 3.4 (Pseudorandom Function(PRF))**.**
A pseudorandom function "looks like" a random function. Specifically, F is a pseudorandom function if F_k for key k randomly chosen in $\{0,1\}^n$, is indistinguishable from a random function $f \in \text{Func}_n$. Formally, for all PPT distinguisher D, we need

$$|P_k(D^{F_k(\cdot)} = 1) - P_f(D^{f(\cdot)} = 1)| \leq \mathbf{negl}(n)$$

where k and f are chosen uniformly random.

Remark: D can interact freely with the oracle that uses either a random, or a pseudorandom function. However, since D is PPT, it can only send over polynomial number of queries.

Definition 3.5 (Pseudorandom Permutation(PRP))**.**
Suppose F is an length-preserving, efficient and keyed function. We call F a **keyed permutation** if for every k, the function $F_k(\cdot)$ is one-to-one. Essentially, pseudorandom permutation requires F to be bijective.
An **efficient** keyed permutation is one which there exists a polynomial-time algorithm computing $F_k(x)$ given k and x as well as a polynomial time algorithm computing $F_k^{-1}(x)$ given k and x.
We can F a **pseudorandom permutation**, if F_k for any randomly chosen k, is indistinguishable from a uniform permutation $f \in \text{Perm}_n$.

Here, Perm_n has a size of $(2^n)!$. For large n, PRP is indistinguishable from PRF.

Definition 3.6 (Block Cipher)**.**
Block Cipher is one practical construction designed to be secure instantiations of pseudorandom permutations, with fixed key length and block length.

$$F : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^m$$

where n is key length and m is block length.

3.2 CPA-secure Encryption from PRF

We define an oracle derived from pseudorandom function as below:

- `Gen`: k is randomly chosen from $\{0,1\}^n$.
- $\texttt{Enc}_k(m)$
    - r is randomly chosen from $\{0,1\}^n$.
    - Output $\langle r, c\rangle = \langle r, F_k(r) \oplus m\rangle$.
- $\texttt{Dec}_k(\langle r, c\rangle) = F_k(r) \oplus c$.

This construction is not practical since we require key to be of the length *same* as message length.

3.3 Proof of CPA Security under pseudorandom PRF F

We want to show the above construction has indistinguishable encryptions under a chosen-plaintext attack.

3.4 Modes of Encryption

Definition 3.7 (Electronic Code Book Mode)**.**
Here $\texttt{Enc}_k(m_1, \dots, m_t) = F_k(m_1) \dots F_k(m_t)$

ECB mode is not CPA secure.

Definition 3.8 (Cipher Block Chaining Mode)**.**
When $\texttt{Enc}_k(m_1 \dots m_t)$,

- Choose c_0 randomly from $\{0,1\}^n$, known as initialisation vector
- For $i = 1$ to t, $c_i = F_k(m_i \oplus c_{i-1})$
- Output $c_0 \dots c_t$

CBC mode is CPA secure.
In CBC mode, we require F must be invertible(PRP), and we requires 1 additional ciphertext block to serve IV.
The drawbacks of CBC mode is that it cannot be serializable.

Definition 3.9 (Propagating Cipher Block Chaining Mode)**.**
In the second step, $c_i = F_k(m_i \oplus c_{i-1} \oplus m_{i-1})$.

Definition 3.10 (Counter Mode)**.**
When $\texttt{Enc}_k(m_1 \dots m_t)$,

- ctr is randomly chosen from $\{0,1\}^n$, called IV

- For $i = 1$ to t: $c_i = m_i \oplus F_k(\mathrm{ctr} + i)$.
- Output $c_0, \ldots, c_t$

Here, we only require F to be PRF, instead of PRP. Counter Mode is serializable.

Theorem 3.1 (Conditions for secure IV selection)**.**
We require

- IV to be **unpredictable** by an adversary before encryption.
- IV must change *per message*

One reason of using Propagating CBC mode is that it allows error propagation. Suppose a bit is flipped at c_i,

- Counter mode will have same bit flipped at p_i
- CBC mode will have p_i garbled, p_{i+1} bit flipped
- PCBC mode will have all p_j where $j \geq i$ garbled

Definition 3.11 (Stateful Counter Mode)**.**
In stateful Counter mode, IV is initialised as IV of previous encryption plus the number of blcoks in the previous encryption.

Stateful Counter Mode is CPA secure.

Definition 3.12 (Stateful CBC Mode)**.**
In stateful CBC mode, IV is initialised as the last ciphertext block.

Stateful CBC mode is insecure.

4 Chosen Ciphertext Attack(CCA) Security

Definition 4.1 (Chosen Ciphertext Attack Security)**.**
Define teh experiment $\text{PrivK}^{\text{cca}}_{\mathcal{A},\Pi}$ for encryption scheme Π and adversary $\mathcal{A}$ as below:

1. $k \leftarrow \texttt{Gen}(1^n)$
2. $\mathcal{A}$ interacts with $\texttt{Enc}_k()$ and $\texttt{Dec}_k()$.
3. $\mathcal{A}$ outputs m_0, m_1. Oracle get a random number b from $\{0,1\}$, and sends $c \leftarrow \texttt{Enc}_k(m_b)$.
4. $\mathcal{A}$ continues interacting with $\texttt{Enc}_k()$ and $\texttt{Dec}_k()$, except $\texttt{Dec}_k(c)$.
5. $\mathcal{A}$ outputs b'. $\mathcal{A}$ succeeds if $b = b'$, in which scenario we say $\text{PrivK}^{\text{cca}}_{\mathcal{A},\Pi} = 1$.

Π is secure against chosen-ciphertext attacks if for all PPT adversaries $\mathcal{A}$, there is a negligible function **negl** such that

$$P(\text{PrivK}^{\text{cca}}_{\mathcal{A},\Pi} = 1) \leq \frac{1}{2} + \mathbf{negl}(n)$$

Suppose the adversary sends ciphertext on behalf of users then observe the resposne, sometimes the adversary can carry out **padding oracle attack**.

Definition 4.2 (Non-malleability)**.**
An encryption scheme is **non-malleable** if given $c = \texttt{Dec}_k(m)$, it is computationally infeasible to find $c' = \texttt{Dec}_k(m')$ where m and m' are related.

CCA secure scheme is non-malleable.

4.1 Padding Oracle Attack

Definition 4.3 (Padding Function)**.**
A padding function is a function $PAD : \{0,1\}^* \to (\{0,1\}^L)+$.

Most application requires PAD to be reversible. One example is called CBCPAD, which is a byte-oriented padding function:
Algorithm CBCPAD(m):

```
    let $m_1 \| \cdots \| m_n := m$, where $m_i \in \{0,1\}^8$
    $p \leftarrow b - (n \mod b)$
    return $m \| \underbrace{p \cdots p}_{p\text{times}}$.
```

In this scheme padding will be 01, 0202, 030303, etc. We denote this as $p \times p$

Suppose the padding in the ciphertext is not valid, then usually the encryption scheme will inform the adversary.

Definition 4.4 (Padding Oracle)**.**
Below is the algorithm carried out with padding oracle:
Algorithm $P_K(c)$:

```
    let $c_0 \| \cdots \| c_n := c$, where $c_i \in \{0,1\}^L$
    for $i = 1 \cdots n$ do $m_i \leftarrow f_K^{-1}(c_i) \oplus c_{i-1}$ (Decryption)
    if $m_n$ ends in $p \times p$ for some $p > 0$, return 1
    else return 0
```

We know illustrate an attack where there is only 2 ciphertext block $c_0 \| c_1$.
Since $c_0 \oplus f_K^{-1}(c_1) = m_1$ and $c_0' \oplus f_K^{-1}(c_1) = m_1'$, we have

$$c_0 \oplus m_1 = c_0' \oplus m_1' \Rightarrow m_1 = (c_0 \oplus c_0') \oplus m_1'$$

and we know about $(c_0 \oplus c_0')$.
The aim is to find a $c_0' \| c_1$ such that $P_K(c_0' \| c_1) = 1$. And if c_0' is chosen randomly from $\{0,1\}^L$, then

- with probability $\frac{1}{2^8}$ m_1' ends in 01.
- with probability $\frac{1}{2^{16}}$ m_2' ends in 0202.
- with probability $\frac{1}{2^{8p}}$ m_p ends in $p \times p$.

Therefore, if P_K returns 1, we can safely assume m_1' ends in 01 and therefore, the last byte of m_1 is last byte of $(c_0 \oplus c_0') \oplus 01$.

More formally, after we obtain c_0' with $P_K(c_0' \| c_1) = 1$, we do

- If $P_K(c_0' \oplus 01(00)^{b-1} \mid c_1) = 0$, then m_1' ends in $b \times b$
- If $P_K(c_0' \oplus 01(00)^{b-2} \mid c_1) = 0$, then m_1' ends in $b - 1 \times b - 1$
- $\cdots$
- If $P_K(c_0' \oplus 01(00)^{1} \mid c_1) = 0$, then m_1' ends in 0202
- Else m_1' ends in 01

4.2 Practical Implementation of Block Cipher

For a secure block ciphers, we require

- Efficient Keyed Permutation $F : \{0,1\}^n \times \{0,1\}^l \to \{0,1\}^l$ where n is key length and l block length
- inverse F_k^{-1} also efficient

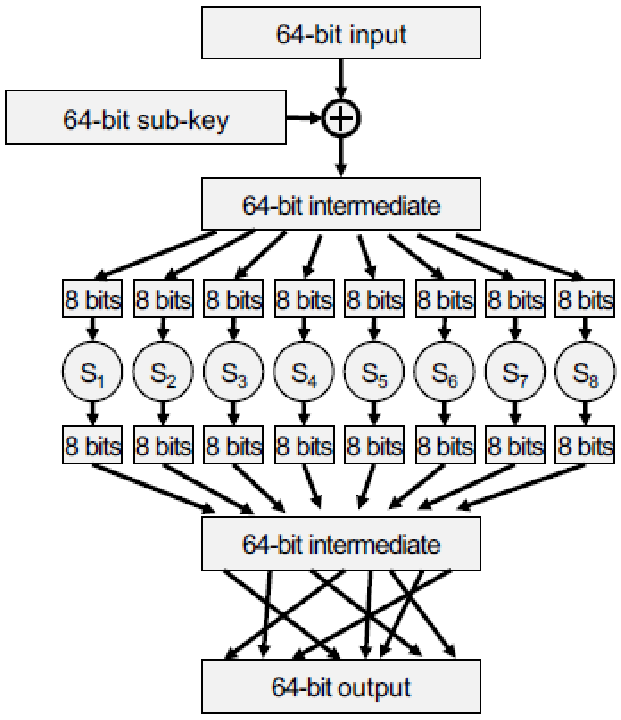
- Pseudorandom permutation

Shannon's confusion-diffusion paradigm gives rise a way to construct a substitution permutation network, by

- Using many smaller random-looking permutations f_i
- Confusion is done via $f_1(x_1)\|\cdots\|f_{16}(x_{16})$
- Diffusion is done by permuting the bits of the above output
- Multiple rounds of confusion-diffusion are required

Definition 4.5 (Substitution Permutation Networks).
Substition-Permutation Network consists of several rounds, and in each round there are three steps. Given the input x, do

1. Key mixing, by setting $x := x \oplus k$ where k is teh sub-key
2. Substitution, by setting $x = S_1(x_1)\|\cdots\|S_8(x_8)$
3. Permutation, by permuting the bits of x to obtain the output of the round.



single round of a substitution-permutation network

For this network, we need to do key scheduling, where round key is derived from master key.

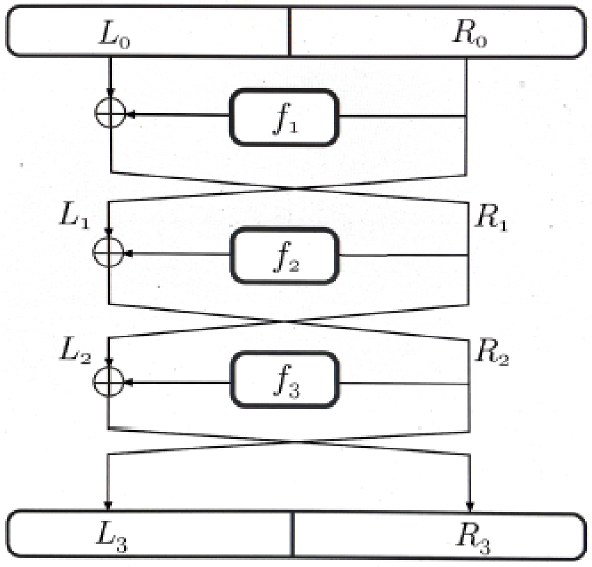
Enough rounds will guarantee an avalanche effect, where small change in the input must affect every bit of the output.
However, this substitution permutation network is usually not invertible. However, we have the feistel network to make use of them to construct a invertible function.

Definition 4.6 (Fiestel Network).
In Fiestel network, we require f_i to be keyed round found from $\{0,1\}^{\frac{l}{2}} \to \{0,1\}^{\frac{l}{2}}$. Then, in teh ith round, we have

$$L_i = R_{i-1} \text{ and } R_i = L_{i-1} \oplus f_i(R_{i-1})$$

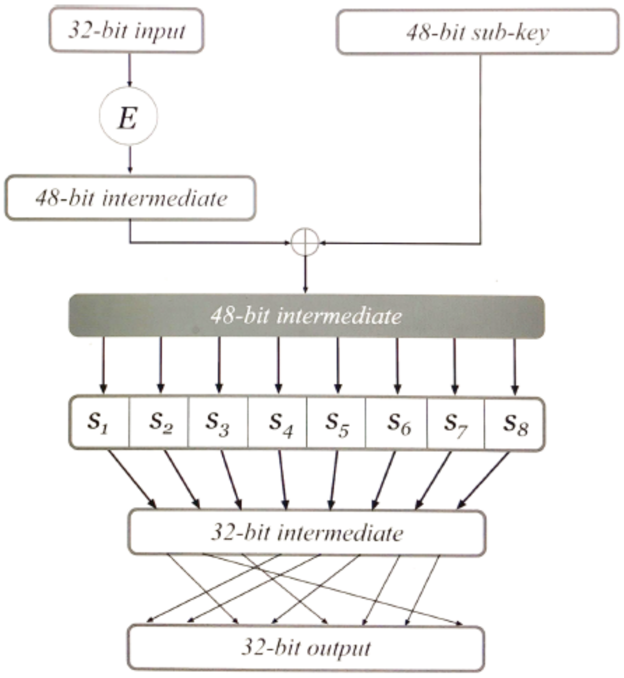
and inverting ith round is

$$R_{i-1} = L_i \text{ and } L_{i-1} = R_i \oplus f_i(R_{i-1})$$



Definition 4.7 (Data Encryption Scheme).
In Data Encryption Scheme, 16 rounds of Feistel netowrk with key length 56 and block length 64 are used. The single round function is illustrated in the figure below. Here, E is a expansion function, and S_i are S boxes from 6 bits to 4 bits, that is non-invertible, publicly known and carefully designed.

Since the key length of DES is 56, it is not secure now. Double DES is also not secure due to meet-in-the-middle attack. Triple DES is equivalent to a DES of key length 112 and is still not secure.


32-bit input
48-bit sub-key
E
48-bit intermediate
48-bit intermediate
S_1
S_2
S_3
S_4
S_5
S_6
S_7
S_8
32-bit intermediate
32-bit output

5 Private Key(Symmetric) Cyrptography

Apart from confidentiality, we also concern about **integrity** and **availability**. Under integrity, we have **user authentication**, which concerns the authenticity of sender, and also **message authentication**, which concerns the integrity of message.
Do note that encryption does not ensure integrity:

- CTR-mode encryption: ciphertext bit flip will lead to plaintext bit flip
- CBC-mode encryption: IV bit flip will lead to first message block bit flip
- Perfect secure OTP: Ciphertext bit flip leads to plaintext bit flip

5.1 Message Authentication Code

Message authentication code is used to prevent adversary from modifying a message sent by one party to another, or from injecting a new message. It requires the sharing of key between communicating parties.
Formally, MAC consists of three functions involved with sender and receiver

- `Gen`: key-generation algorithm
- `Mac`: tag-generation algorithm, which
  - Input: a key k and message $m \in \{0,1\}^*$.
  - Output: a tag $t = \texttt{Mac}_k(m)$
- `Vrfy`: verification algorithm, which
  - Input: a key k, and a message m, a tag t
  - Output: bit $b = \texttt{Vrfy}_k(m, t)$, where $b = 1$ indicates valid and 0 indicates invalid

For *canonical* verification where `Mac` is deterministic, receiver's `Vrfy` function will just recompute $\texttt{Mac}_k(m)$ and compare with tag t received.

Definition 5.1 (Existential Unforgeability)**.**
Consider the following **message authentication experiment** $\mathrm{Mac-Forge}_{\mathcal{A},\Pi}(n)$.

- Key k is generated by $\texttt{Gen}(1^n)$.
- $\mathcal{A}$ has oracle access to $\texttt{Mac}_k()$ with Q, the set of all asked queries. A outputs (m, t).
- $\mathcal{A}$ succeeds if and only if
  - $\texttt{Vrfy}_k(m, t) = 1$
  - $m \notin Q$

When $\mathcal{A}$ succeeds, we say $\text{Mac} - \text{Forge}_{\mathcal{A},\Pi}(n) = 1$.
We say a MAC is **existentially unforgeable** under an adaptive chosen-message attack, if for all PPT adversary $\mathcal{A}$, there is a negligible function **negl** such that

$$P(\text{Mac} - \text{Forge}_{\mathcal{A},\Pi}(n) = 1) \leq \mathbf{negl}(n)$$

It is not a "too strong" definition since we want MAC scheme to be application-independent.
Remark: MAC is in general *not* designed to detect **replays**. Replay attack can be prevented via application-dependent schemes such as **sequence number**, or **timestamp**.

Definition 5.2 (Strong MAC)**.**
MAC definition does not allow adversary to forge a different valid tag $t_1' \neq t_1$ on the previously submitted messages. However, adversary may be able to find a valid pair (m_1, t_1').
In strong mac, we consider the following message authentication experiment $\text{Mac} - \text{sForge}_{\mathcal{A},\Pi}(n)$:

- Key k is generated by $\texttt{Gen}(1^n)$
- $\mathcal{A}$ has oracle access to $\texttt{Mac}_k()$ with Q, the set of all asked queries $\underline{\text{pairs}}$. $\mathcal{A}$ outputs (m, t).
- $\mathcal{A}$ succeeds if and only if
  - $\texttt{Vrfy}_k(m, t) = 1$
  - $(m, t) \notin Q$

When $\mathcal{A}$ succeeds, we say $\text{Mac} - \text{sForge}_{\mathcal{A},\Pi} = 1$.
A MAC is a **strong MAC** under an adaptive chosen-message attack, if for all PPT adversary $\mathcal{A}$, there is a negligible function **negl** such that

$$P(\text{Mac} - \text{sForge}_{\mathcal{A},\Pi}(n) = 1) \leq \mathbf{negl}(n)$$

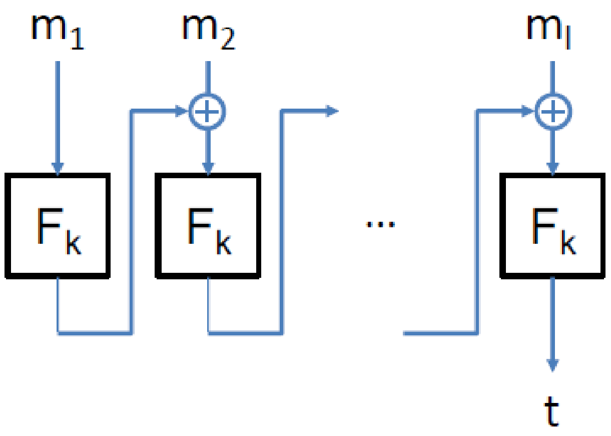
A secure MAC using canonical verification is also strongly secure.

Remark: Authentication enables the receiver to verify origin, but the receiver cannot convince a third party of origin. On the other hand, signature enables the receiver to verify origin and receiver can convince third party of origin as well. The unique property of signature is called **non-repudiation**.

5.2 CBC-MAC

CBC-MAC is the standardized MAC, and is widely used. Here, we define $t_i = F_k(t_{i-1} \oplus m_i)$ and $t_0 = 0^n$. Output t_l as the tag t.
$\texttt{Vrfy}$ will output 1 if and only if $t = \texttt{Mac}_k(m)$.
CBC-MAC is only secure for fixed length messages as **concatenation attack** is possible for arbitrary length CBC-MAC:

- Suppose (m, t) and (m', t') are available
- we construct m'' to be m followed by $t \oplus m_1'$ followed by rest of m'.



- Then the output tag will be t' for m''.

Definition 5.3 (Authenticated Encryption)**.**
A private-key-encryption scheme is an authenticated encryption scheme if it is (1)CCA-secure and (2)unforgeable.

Suppose we have

- $\Pi_E = (\texttt{Enc}, \texttt{Dec})$, a CPA-secure encryption
- $\Pi_M = (\texttt{Mac}, \texttt{Vrfy})$, message authentication code

We may have the following three procedures:

1. Encrypt-and-authenticate: $c \leftarrow \texttt{Enc}_{k_E}(m)$, $t \leftarrow \texttt{Mac}_{k_M}(m)$
   t is deterministic when `Mac` is deterministic and therefore it is not CPA-secure
2. Authenticate-then-encrypt: $t \leftarrow \texttt{Mac}_{k_M}(m)$, $c \leftarrow \texttt{Enc}_{k_E}(m\|\|t)$
   This is not necessarily secure. Consider the CBC-mode-with-padding, there are two error cases, which are "bad pidding" and "MAC failure". The first message may give rise to padding oracle attack.
3. Encrypt-then-authenticate: $c \leftarrow \texttt{Enc}_{k_E}(m)$, $t \leftarrow \texttt{Mac}_{k_M}(c)$
   If Π_E is CPA secure and Π_M is strong MAC, this scheme then is secure authenticated encryption scheme

The important principle here is that each cryptographic primitives should always use independent keys.

5.3 Cryptographic Hash Function

Definition 5.4 (Hash Function)**.**
Hash Function is function that takes input of arbitrary length and compress them into short, fixed-length outputs. Here, evaluation of $H : \{0,1\}^* \rightarrow \{0,1\}^l$ should be efficient and public.

Definition 5.5 (Cryptographic Hash Function).
Cryptographic Hash Function shuold be a hash function that has

- Collision Resistance
- Second Preimage Resistance
- Preimage Resistance

Definition 5.6 (Collision Resistance).
Consider $\Pi = (\texttt{Gen}, H)$ where `Gen` is a key generation algorithm and H a keyed hash function: $H : (s, x) \to \{0,1\}^l$.
Define the collision-finding experiment $\text{Hash-coll}_{\mathcal{A},\Pi}(n)$:

- key s is generated by `Gen`
- Adversary $\mathcal{A}$ is given key s, and will output x, x'.
- $\text{Hash-coll}_{\mathcal{A},\Pi}(n) = 1$ if and only if $x \neq x'$ and $H^s(x) = H^s(x')$.

Π is collision resistant if for all PPT adversary $\mathcal{A}$, there is a **negl** such that

$$P(\text{Hash-coll}_{\mathcal{A},\Pi}(n) = 1) \leq \mathbf{negl}(n)$$

Remark: Cryptographic hash functions in practice are usually **unkeyed**.

Theorem 5.1 (Hash-And-Mac).
Hash-and-Mac uses hash to reduce arbitrary-length message m to a fixed-length digest $H(m)$ and apply a fixed-length MAC, e.g. CBC-MAC.
If MAC is secure and hash is collision resistant, the hash-and-MAC is a secure MAC.

Definition 5.7 (HMAC).
Standardized HMAC is formalized as below:

- `Gen`: two keys s and k
- `Mac`: $t = H^s((k \oplus \mathit{opad}) \| H^s((k \oplus \mathit{ipad}) \| m))$
- `Vrfy`: Output 1 if and only if t equals $\texttt{Mac}_{s,k}(m)$ in a canonical sense.

HMAC is highly efficient and supported by proof of security.

5.4 Attack on Cryptographic Hash Functions

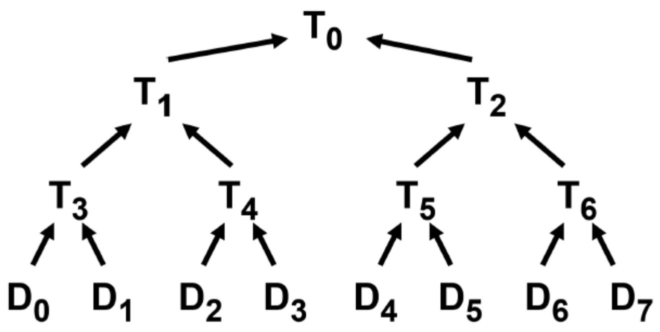
Suppose $H : \{0,1\}^* \to \{0,1\}^l$ is a collision resistant hash function. Then it takes at least $q := 2^l + 1$ hashes to found a collision with probability 1. However, if we want a collision with probability with probability 0.5, it takes roughly $2^{\frac{l}{2}}$ hashes to find a pair of duplicates among them with probability $\frac{1}{2}$.

5.5 Hash Function in Practice

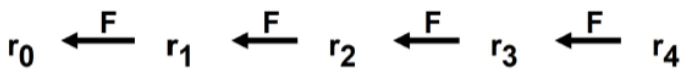
- MD5: 128-bit output. Insecure. Collision found in 2004.
- SHA-1: 160-bit output length. Very common. Collision found in 2017.
- SHA-2: 256/512 bit output length. No known significant weaknesses
- SHA-3: 224,256, 384, 512-bit outputs.

Hash Function can be applied in

- Fingerprinting: instead of storing original data, one can stores a short hash digest, such as virus fingerprinting.
- Authenticating a sequence of data values: $D_0, \dots, D_N$. This is done via **merkle hash trees**.
  Merkle has tree makes the data the leaves of the tree and take hash recursively until

  root. Verifier knows T_0, the hash at the root. Verifier can authenticate leaf D_i, if the person can supply D_i with the elements D_j and T_k's required in obtaining T_0 upstream to the root.
- Another way to do so in one-way hash chains. It picks a random r_N with a public

  one-way function F. r_i is calculated by $r_i = F(r_{i+1})$ and r_0 will be made public.
  It is used in reverse order of construction. Authentication of r_i can be efficiently done given r_j with $j > i$.
- Password hashing. Password is stored in terms of hash, since preimage resistance of H makes inverting the hash difficult.
  However, password space is quite limited, and there is possibility for pre-computation. Therefore, we should use
  - Slow hash functions, like bcrypt.

  - Hash password together with a random salt s, i.e. $h = H(\text{passwaord}\|s)$ and store (h, s). The salt increases the difficulty for precomputation.

- Key Derivation. Suppose there is a shared secret with enough entropy but non-uniform distribution. One can use hash function to obtain a uniform distributed secret key.

- Commitment Schemes. Suppose $C(m)$ is the commitment to a value m. We can define $C(m)L = h(r\|m)$ to hide content of m from the public, but also binds the bidder with the value m. Here, r is a length-l random sequence.
  This hash function h requires **preimage resistence**, **collision resistance** and **non-malleability properties**.

6 Cryptographic Hardness Assumptions

Many private-key cryptography are based on assumption that pseudorandom permutation(PRP) exists. The *existence* of PRP can be proved based on assumption that one-way function exist, whereas the existence of one-way function can be proved if factoring problem is hard.

6.1 Mathematics Preliminaries

Theorem 6.1 (Greatest Common Divisor).
There exists a unique $d \in \mathbb{Z}$ such that for any given $a, b \in \mathbb{Z}$, we have

- $d \geq 0$
- $d \mid a$ and $d \mid b$
- for any $c \in \mathbb{Z}$ such that $c \mid a$ and $c \mid b$, it is true that $c \mid d$.

This d is called **greatest common divisor** of a and b.

Definition 6.1 (Relatively Prime). *if* $a, b \in \mathbb{Z}$ *and* $\gcd(a, b) = 1$*, then* a, b *are relatively prime.*

Theorem 6.2. If $a \mid (bc)$ and $d = \gcd(a, b) = 1$, then $a \mid c$.

Definition 6.2 (Modulo).
$a \mod N$ is the remainder of a divided by N. We have $0 \leq a \mod N \leq N - 1$.

Theorem 6.3. $a = b \pmod N \Leftrightarrow a \mod N = b \mod N$.

It is known that integer addition, subtraction, multiplication, division are efficient. It is known that modular addition/subtraction/multiplication/division is efficient. It is known that **modular exponentiation** is efficient.

Theorem 6.4 (Euclid's Algorithm).
$\gcd(a, b) = \gcd(b, r)$ if we have $a = q \cdot b + r$ with $0 \leq r < b$.
Using this, we can find $\gcd(a, b)$ in polynomial time.

Definition 6.3 (Abelian Group).
An abelian group is a set G and a binary operation $\circ$ defined on G such that

- $\forall a, b \in G, a \circ b \in G$.
- $\exists e \in G$ such that $\forall g \in G, e \circ g = g$. e is called identity.
- Every $g \in G$ has inverse $h \in G$ such that $h \circ g = e$. Here the inverse is called left inverse.
- For all $f, g, h \in G$, $f \circ (g \circ h) = (f \circ g) \circ h$. This is associativity.
- For all $g, h \in G$, $g \circ h = h \circ g$. This is commutativity.

We define the **order** of a finite group G by the number of elements in G.

An additive abelian group supports the operation $+$ with an identity denoted by 0. Inverse of g is denoted by $-g$ and group exponentiation is denoted by $m \cdot a$.
An multiplicative abelian group supports the operation $\times$ with an identity denoted by 1. Inverse of g is denoted by g^{-1} and group exponentiation dentoed by a^m.

Theorem 6.5 ($\mathbb{Z}_N$)**.**
$\mathbb{Z}_N$ is a additive group under addition modulo N.

Definition 6.4. *Modular Inverse*
b is invertible modulo N if there eixsts an element b^{-1} such that $b \cdot b^{-1} = 1 \mod N$. It is known that b^{-1} is unique mod N if it exists.

Theorem 6.6 (Existence Condition of Modular Inverse)**.**
b is invertible modulo N if and only if $\gcd(b, N) = 1$.

Theorem 6.7 ($\mathbb{Z}_N^*$)**.**
$\mathbb{Z}_N^*$ is a group with invertible elements between 1 and $N-1$ under multiplication modulo N.

Definition 6.5 (Euler's Totient Function)**.**
$\varphi(N)$ is the Euler's totient function, which measures the number of invertible elements modulo N.
$\varphi(N) = N - 1$ if N prime. If $N = pq$ where p, q are primes, then $\varphi(N) = (p-1)(q-1)$.

Theorem 6.8 (Fermat's Little theorem)**.**
Let G be finite group of order m. Then for any $g \in G$, we have

$$g^m = 1$$

Specifically, for all $z \in \mathbb{Z}_N$, we have $N \cdot a = 0 \mid N$ and for all $z \in \mathbb{Z}_N^*$, we have $a^{\varphi(N)} = 1 \mod N$.

Theorem 6.9. Let G be finite gorup of order m. Then for $g \in G$ and $x \in \mathbb{Z}$, it holds that $g^x = g^{x \mod m}$.

Theorem 6.10. Let G be finite group of order m. We define f_e to be $f_e(g) = g^e$ for any integer e, where $g \in G$. If $\gcd(e, m) = 1$, then f_e is permutation. Also if $d = e^{-1} \mod m$, then f_d is inverse of f_e.

6.2 Factoring and RSA Problem

It is common sense that multiplying two numbers is easy, but factoring a number is hard. Among all numbers, the hardest numbers to factor are those that are the products of two equal-length primes.
The factoring problem is related to RSA problem. RSA problem involves $\mathbb{Z}_N^*$. where $N = p \times q$, p, q primes. The order of $\mathbb{Z}_N^*$ is $\varphi(N) = (p-1)(q-1)$. This order is easy to compute if p, q known and hard to compute otherwise. Therefore, computing $\varphi(N)$ is as hard as

factoring N.
Fix e with $\gcd(e, \varphi(N)) = 1$, then we known exponentiation to the eth power is a permutation of $\mathbb{Z}_N^*$, and there exists a unique d, which satisfies $ed = 1 \mod \varphi(N)$, that gives $(x^d)^e = x \mod N$. We call x^d eth root of x modulo N.
As long as p, q is known or $\varphi(N)$ is known which implies p, q can be computed in polynomial time, then we can compute x^d easily. Otherwise, computing d is as hard as factoring N.

Definition 6.6 (RSA Problem Relative to GenRSA)**.**
We first define a experiment called GenRSA.
GenRSA: On input 1^n, outputs (N, e, d) with $N = pq$, a product of two n-bit primes, and with $ed = 1 \mod \varphi(N)$. Then, the experiment RSA-inv$_{\mathcal{A},\text{GenRSA}}(n)$ is defined as below:

- Compute $(N, e, d) \leftarrow \text{GenRSA}(1^n)$
- Choose uniform $y \in \mathbb{Z}_N^*$.
- Run $\mathcal{A}(N, e, y)$ to get x.
- Experiment evaluates to 1 if $x^e = y \mod N$.

The RSA problem is relative to GenRSA if for all PPT algorithm $\mathcal{A}$, we have $P(\text{RSA-inv}_{\mathcal{A},\text{GenRSA}}(n) = 1) < \textbf{negl}(n)$.

In layman's words, given N and e, it is hard to compute the eth root of a uniform element $y \in \mathbb{Z}_N^*$.

Definition 6.7 (Rough Sketch of Implementation of GenRSA)**.**
To implement GenRSA,

- Generate uniform n-bit primes p, q.
- Set $N = pq$.
- Choose arbitrary e with $\gcd(e, \phi(N)) = 1$.
- Compute $d = e^{-1} \mod \varphi(N)$.
- Output (N, e, d).

It is worth noting that choice of e does not affect hardness of RSA problem, therefore common choices can be $e = 3$ or $e = 2^{16} + 1 = 65537$.

Theorem 6.11 (Hardness of RSA relative to Factoring)**.**
If factoring moduli output by GenRSA is easy, then RSA problem must be easy relative to GenRSA.
However, the RSA problem is not known to be implied by hardness of factoring.

6.3 Cyclic Group, Discrete Logarithm Problem

Definition 6.8 (Cyclic Group)**.**
Let G be a finite group of order m. Let g be an element of G. Consider the set $S = \{g^0, g^1, \ldots\}$. If the set S has m elements, then $G = S$, and we say such G is a cyclic group and such g is a generator of G.

Theorem 6.12 (Prime Order Group is Cyclic)**.**
Any group of prime order is cyclic, and every non-identity element is a generator.

Theorem 6.13. If p is a prime, the $\mathbb{Z}_p^*$ is cyclic.

It is also worth noting, it is easy to do uniform sampling in cyclic group G. Given G is of order m and a generator g, to sample uniformly an element $h \in G$, choose uniformly $x \in \{0, \ldots, m-1\}$ and set $h = g^x$.

Definition 6.9 (Discrete Logarithm Problem)**.**
Let $\mathcal{G}$ be a group-generation algorithm. Specifically, on input 1^n, $\mathcal{G}$ outputs a cyclic group G with order q such that $\|q\| = n$ and a generator g.
For algorithm $\mathcal{A}$, define experiment $\text{Dlog}_{\mathcal{A},\mathcal{G}}(n)$:

- Compute $(G, q, g) \leftarrow \mathcal{G}(1^n)$.
- Choose uniform $h \in G$.
- Run $\mathcal{A}(G, q, g, h)$ to get x.
- And experiment evalutaes to 1 if $g^x = h$.

The discrete-logarithm problem is hard relative to $\mathcal{G}$ if for all PPT algorithm $\mathcal{A}$, $P(\text{Dlog}_{\mathcal{A},\mathcal{G}}(n) = 1) < \textbf{negl}(n)$.

6.4 Diffie-Hellman Problem

Definition 6.10 (Diffie-Hellman Problem)**.**
Here we fix cyclic group G with generator g. Suppose $h_1 = g^x, h_2 = g^y \in G$, then define $\text{DH}_g(h_1, h_2) = DH(g^x, g^y) = g^{xy}$.
For *computational* Diffie-Hellman problem, given g, h_1, h_2, compute $\text{DH}_g(h_1, h_2)$.
For *decisional* Diffie-Hellman problem, given g, h_1, h_2, distinguish the correct $\text{DH}_g(h_1, h_2)$ from a uniform element of G. And the DDH problem is hard relative to $\mathcal{G}$ if for all PPT algorithm $\mathcal{A}$, we have

$$|P[A(G, q, g, g^x, g^y, g^z) = 1] - P[A(G, q, g^x, g^y, g^{xy}) = 1]| \leq \textbf{negl}(n)$$

Theorem 6.14 (Relative Difficulty of Diffie-Hellman Problems)**.**
It is known that, relative to G:

- If DL is easy, so is CDH;
- If CDH is easy, so is DDH.

7 Public Key Cryptography

The notion of public-key cryptography involves a pair of keys (pk, sk), where

- pk, the public key is widely disseminated
- sk, the private key is kept secretly

Here, pk is used for encryption and sk is for decryption in encryption scheme; sk is used for signature and pk is for verification in authentication scheme.
Here, we need to assume that pk is disseminated via an authenticated channel. This assumption will be removed in the future.
One consequence is that anyone with pk is able to encrypt message but only the one with sk can decrypt message.
Another thing is that signature using sk will provide non-repudiation property.

7.1 RSA Encryption

Definition 7.1 (Public Key Encryption)**.**
Public key encryption scheme is a triple $(\texttt{Gen}, \texttt{Enc}, \texttt{Dec})$, where

- $\texttt{Gen}$: key-generation algorithm that outputs (pk, sk);
- $\texttt{Enc}$: encryption algorithm $c := \texttt{Enc}_{pk}(m)$
- $\texttt{Dec}$: decryption algorithm $m := \texttt{Dec}_{sk}(c)$.

Since, encryption uses pk which the public has, **CPA** security is the baseline security notion.
Deterministic public-key encryption, such as student RSA, cannot be CPA secure. This is because

- One can determine repeated encryption of a plaintext
- It may allow adversaries to recover plaintext messages when message set is small.

For CCA definition, adversay is allowed to have access to decryption oracle as well.

Definition 7.2 (Textbook RSA)**.**
For textbook RSA, we define the triplet as

- $\texttt{Gen}$: $(N, e, d) \leftarrow \text{GenRSA}(1^n)$. Here, (N, e) is the public key, and (N, d) is private key.
- $\texttt{Enc}$: for $m \in \mathbb{Z}_N^*$, compute $c := m^e \mod N$.
- $\texttt{Dec}$: given $c \in \mathbb{Z}_N^*$, compute $m := c^d \mod N$.

This textbook RSA is insecure, as we can use common-modulus attack.

- Common modulus attack 1

- Common modulus attack 2
  Several users share N, and suppose $\gcd e_1, e_2 = 1$. Afversary can see $c_1 = m^{e_1} \mod N$ and $c_1 = m^{e^2} \mod N$ of the same message m. Since $\gcd(e_1, e_2) = 1$, there exists x, y such that $xe_1 + ye_2 = 1$. Therefore, adversary computes $c_1^x c_2^y = m \mod N$ and gets the plaintext.

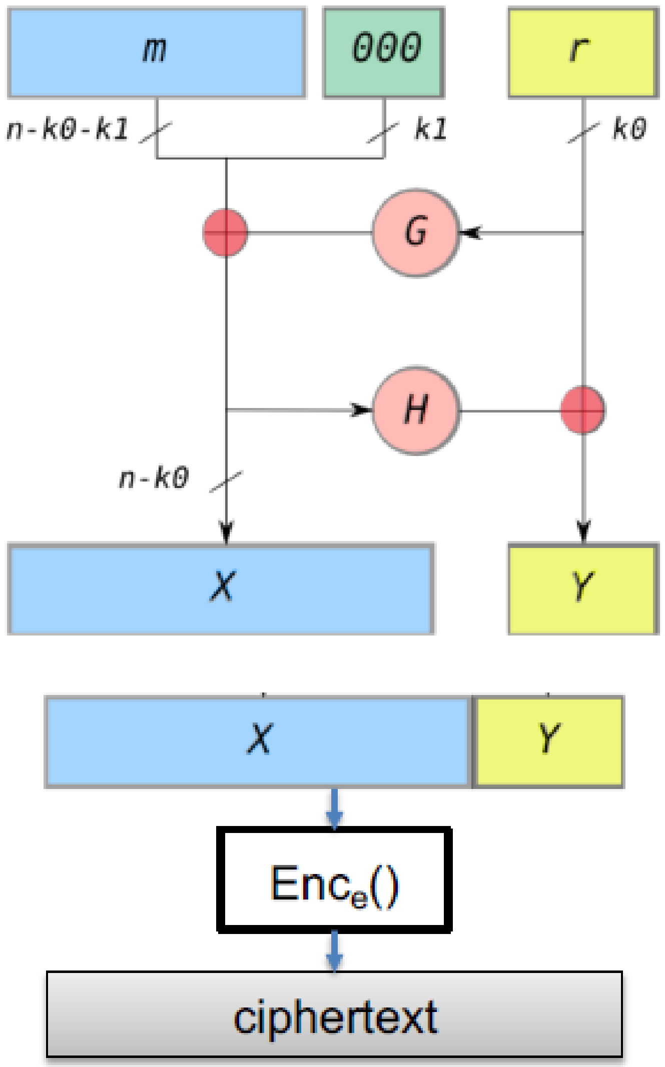
In CCA notion, we can also have CCA attack.

- CCA attack 1
  Adversary obtains a user's ciphertext $c = m^e \mod N$, picks a random $r \in \mathbb{Z}_N^*$ and creates forgery $c' = r^e c \mod N$. It submits c' for decryption, and gets back m'. Then $m = m' r^{-1} \mod N$.

- CCA attack 2
  Adversary obtains a user's ciphertext $c = m^e \mod N$ of a unknown m. It is easy to generate c' that is an encryption of $2m \mod N$, by just setting $c' = 2^e c \mod N$.

Definition 7.3 (Optimal Asymmetric Encryption Padding(OAEP))**.**
This optimal asymmetric encryption padding provides RSA-based CCA-secure encryption. Here, G, H are independent hash functions, r a unique random number per encryption.

Theorem 7.1 (Checking Primality)**.**
We have $a^{p-1} = 1 \mod p$ where $0 < a < p$ if p is a prime. The other direction is not true. However, if $a^{p-1} = 1 \mod p$ holds, and p is long, then propability that p is not a prime is small(100 digit p, 10^{-13} probability). Therefore, to further ensure correctness, try multiple values of a.
Caveat: Carmichael numbers. There exists p such that p not prime and $a^{p-1} = 1 \mod p$ for all choices of $0 < a < p$.
Another useful theorem: if p is odd prime, then $x^2 = 1 \mod p$ has only two solutions, namely $x = 1$ and $x = -1$. So we can use its contrapositive to see whether a number p is not prime.


m
000
r
n-k0-k1
k1
k0
G
H
n-k0
X
Y
X
Y
Enc$_e$()
ciphertext

8 Diffie Hellman and El Gamal

Recall the special cyclic group $\mathbb{Z}_p^*$ where p is a prime. *Not every* element of $\mathbb{Z}_p^*$ is a generator.

Definition 8.1 (Order of Element in $\mathbb{Z}_p^*$)**.**
For any $x \in \mathbb{Z}_p^*$, the order of x is the **smallest** positive integer a such that $x^a = 1 \mod p$.

Theorem 8.1. For all $x \in \mathbb{Z}_p^*$, the order of x divides $p-1$.

Definition 8.2 (Quadratic Residue)**.**
An element $x \in \mathbb{Z}_p^*$ is called a **quadratic residue**(QR) if its square root exists in $\mathbb{Z}_p^*$.

Theorem 8.2. Each element $x \in \mathbb{Z}_p^*$ has either 0 or 2 square roots.
Specially, if $x = a^2 \mod p$, then so is $x = (-a)^2 \mod p$.

Theorem 8.3. Exactly half of the elements in $\mathbb{Z}_p^*$ are quadratic residue.
This is due to the square root-able of g^k, where g generator if and only if k is even.

It is konwn that, if given $x \in \mathbb{Z}_p^*$, deciding whether x is a quadratic residue is computationally **easy**. It can be checked using Legendre symbol.

Theorem 8.4 (Legendre Symbol)**.**
Define Legendre symbol $\mathcal{L}_p(x) = x^{\frac{1}{2}(p-1)} \mod p$. It is known that

$$\mathcal{L}_p(x) = \begin{cases} 1 & \text{if } x \text{ is a QR} \\ -1 & \text{if } x \text{ is not a QR} \\ 0 \text{ if } x = 0 \mod p \end{cases}$$

An useful property is

$$\mathcal{L}_p(xy) = \mathcal{L}_p(x)\mathcal{L}_p(y)$$

Also, computing square roots in $\mathbb{Z}_p^*$ is also computationally feasible. It is known that $O(n^4)$ randomized algorithm exists.
However,

Theorem 8.5. Discrete Logarithm(DL) is in general a computationally **hard** problem in cyclic groups.

8.1 Discrete Logarithm(DL) problem

Recall the Discrete Logarithm problem: let g be a generator of cyclic group. Given $x \in G$ find exponent e such that $x = g^e$.
The DL problem has a similar form of RSA problem:
RSA: given n, e, y, find x such that $x^e = y \mod n$.
DL: given n, x, y, find e such that $x^e = y \mod n$.

8.2 Diffie-Hellman Key Exchange

The aim is to establish a **common secret** key over an **unsecured** channel with presence of **eavesdropper**.

Definition 8.3 (Diffie Hellman Key Exchange)**.**
Suppose Alice and Bob pre-determine a set of **public parameters**: a prime p and a generator g of $\mathbb{Z}_p^*$.

1. Alice chooses a random x and computes $c := g^x \mod p$.
   Alice sends c to Bob.
2. Bob chooses a random y and computes $d = g^y \mod p$. Bob sends d to Alice.
3. Bob computes the secret key $k = c^y \mod p$.
4. Alice computes the secret key $k = d^x \mod p$.

Essentially, the established key $k = g^{xy} \mod p$.

It is obvious that we need to make it hard to determine $k := g^{xy} \mod p$ given g^x, g^y, p and g.

Definition 8.4 (Computational Diffie-Hellman)**.**
Computational Diffie-Hellman Problem(CDH): Let g be generator of a cyclic group G. Given g^a, g^b, find g^{ab}.

CDH is conjectured to be computationally hard in $\mathbb{Z}_p^*$ in general.
If CDH is easy, then adversary of DH key exchange can easily recover the key.

Theorem 8.6. If DL is easy, so is CDH problem.

Definition 8.5 (Decisional Diffie Hellman Problem(DDH))**.**
Let g be a generator of a cyclic group G. Given a 3-tuple (x, y, z) of elements from G, distinguish whether it is generated by process A or B, where

- A: randomly pick a, b, c from G, and outputs $(x := g^a, y := g^b, z := g^c)$.
- B: randomly pick a, b from G, and outputs $(x := g^a, y := g^b, z := g^{ab})$.

Theorem 8.7. If G is to be used for DH key exchange protocol, DDH problem should be hard for this group G.

This is because we want to make the obtained session key to be indistinguishable from random, as it is usually used for private key encryption.

Theorem 8.8 (DDH NOT Hard for $\mathbb{Z}_p^*$)**.**
For the group $\mathbb{Z}_p^*$, DDH is not hard.
Note, if $\mathcal{L}_p(z) = \mathcal{L}_p(x)\mathcal{L}_p(y)$, both A and B are possible; otherwise, only A is possible. Therefore, the chance of distinguishing A from B is non-negligible.
To solve this, we need to pick a, b that are even. Equivalently speaking, we need every element x, y, z for both A and B to be quadratic residue.

Theorem 8.9. DDH is hard for quadratic residues modulo a prime p.

Theorem 8.10 (RSA Key Exchange)**.**
Key exchange in presence of eavesdropper can also be done using RSA.

1. Alice randomly chooses the public key and a parameter of RSA, i.e., N and e, and sends them to Bob but keeps the private key secret.
2. Bob randomly picks a session key k, encrypts it using public key and sends the ciphertext to Alice. That is, Bob sends
$$c = k^e \mod N$$
3. Alice uses her private key d to encrypt the ciphertext c and obtain k.

There are two major differences of DH and RSA:

1. RSA-based key exchange does not achieve forward secrecy.
    - RSA-based key exchange, if private key d for Alice is compromised in future, all previous session keys can be recovered.
    - However, in DH-based key exchange, compromise of Alice's private key x in the future does not reveal previous session keys since Bob's key y also contribute to session key generation.
2. In RSA-based key exchange, Bob selects a session key.
    - If it is weak, e.g., not pseudorandom, it can be problematic.
    - In DH-based key exchange, both Alice and Bob contribute to the key generation and thus such vulnerability has less impact.

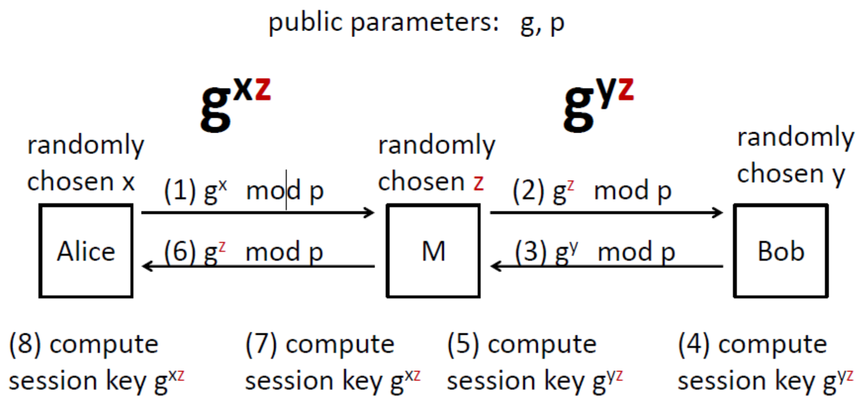
8.3 Man-In-The-Middle(MITM) Attack Against DH Key Exchange

Imagine Mallory M is a malicious attacker in between Alice and Bob that can modify messages. M can carry out the MITM attack by sending his public key g^z to both parties, so that 2 different session keys are established between Mallory and Alice and Bob respectively.

8.4 ElGamal Cryptosystem

In ElGamal Cryptosystem, any party will have

- a private key x, and
- a public key g, which is the generator g of a cyclic group , and $h := g^x$.



Encryption of message m:

$$E(m) = \langle mh^r, g^r \rangle$$

where m must be an element in the cyclic group and r is randonly chosen.
Decryption:

$$D(\langle a, b \rangle) = ab^{-x}$$

Theorem 8.11 (Security of ElGamal)**.**

- If attacker can solve **DL**, he can calculate x from g, h. Then attacker can decrypt any message.
- If attacker can solve **CDH**, he can find plaintext, given g, h and ciphertext.
  This is because one can compute h^r easily, which allows one to compute m.
- If attacker can solve **DDH**, then cryptosystem is not semantically secure, or equivalently speaking, not CPA secure.
  First note, DDH is easy in $\mathbb{Z}_p^*$. Also, ElGamal encryption is probabilistic.
  To be semantically secure, the following problem has to be difficult: the adversary knows that plaintext m is either g or 1, given public key and ciphertext, $h = g^x, \langle b := mg^{rx}, c := g^r \rangle$, the adversary wants to determine the plaintext.
  Adversay can use DDH to check whether h, b, c are consistent. If consistent, message $m = 1$. Otherwise $m = g$.

Therefore, to achieve semantic security, one should choose a group where DDH is hard, and use that to construct ElGamal cryptosystem.
For example, pick $g \in QR_p \subset \mathbb{Z}_p^*$. Take group generated by g as the underlying group in ElGamal.

Note, ElGamal in QR_p is CPA secure.
Elliptic Curve Cryptography(ECC) is basically ElGamal with another choice of cyclic group.

9 Digital Signature Algorithm

Signature scheme usually admits the has-and-sign pattern. Where we have $m \to h(m) \to s(h(m))$, the cryptographic hash function h reduces the size of message m to a small, fixed-sized digest. Then we use s and `vrfy` to sign or verify, which is more costly than hash function h.

Definition 9.1 (Digital Signature Algorithm)**.**
DSA is based on ElGamal's signature scheme. It requies two primes p, q, where p is 512,1024,2048 or 3072 bits and q 160, 224, 256 bits.
We use SHA-1, SHA-2 to hash the message.
Let us take p to be 1024 bit and q 160 bits. The size of signature is 3320 bits as a result.
The public key is $p(1024), q(160), g(1024), y(1024)$ and private key $x(160)$.
We require

- p, q to be two primes such that
  - $q \mid p-1$ and $q^2 \mid p-1$.
- g a generator in $\mathbb{Z}_p^*$ having order q.
- Let $h : \{0,1\}^* \to \mathbb{Z}_q$ a collision resistant hash function.
- Set private key be a *randomly chosen* $x \in \mathbb{Z}_q^*$.
- Set p, q, g public, together with $y := g^x \mod p$.

For `sign`, given a message $m \in \{0,1\}^*$,

- Choose $k \in \mathbb{Z}_q^*$ uniformly at random
- Compute $r = (g^k \mod p) \mod q$.
- Compute $s = (h(m) + x \cdot r) \cdot k^{-1} \mod q$
- the signature for m is (r, s).

For `Vrfy`, given a message $m \in \{0,1\}^*$ and a signature (r, s),

- Compute $u_1 := h(m) \cdot s^{-1} \mod q$
- Compute $u_2 := r \cdot s^{-1} \mod q$
- Output `YES` if and only if $r = ((g^{u_1} y^{y_2} \mod p) \mod q)$.

Remark:

1. Signing can be very fast with precomputation, since k, k^{-1} and r can be precomputed before seeing m.
2. Extra care must be taken in choosing k in implementing dsa. It must have high entropy, be unpredictable and unique.

9.1 Public Key Infrastructure(PKI)

Definition 9.2 (Digital Certificate)**.**
Digital certificate is a signature that binds an entity to some public key.
For eaxmple, $\text{cert}_{C\to A}$ is a certificate for A's public key issued by C. In the certificate, we include $\texttt{sign}_{sk_C}(A\text{'s key is } pk_A)$ in the certificate.

If B knows C's key pk_C and **trust** C. Then B can believe that pk_A is indeed A's key.
In real life, B can be treated as the certificate authority, which keeps the pairings of entity and their public key and creates certificate signed by B's private key.
There are two PKI models:

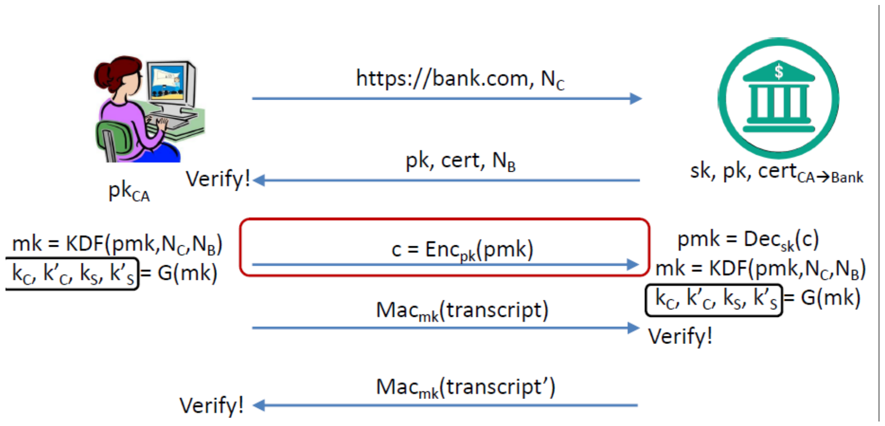
1. **Single CA**: everybody in the system trusts the single CA. The public key of the CA pk_{CA} is installed on everyone's machine over authenticated channel.
2. **Multiple CA**: Multiple CAs issue certificate, so as to prevent single point of failure. However, the entire PKI is as insecure as the *weakest* CA.
3. **Delegated CAs**: CA can delegate its ability to issue certificates, so that CA's are arranged in a hierarchical structure, with the root CA as the root of the hierarchy.

There are several scenarios where we need to invalidate certificates, for example

- Expiration, which can be done by adding an expiration date in the CA and signed
- Revocation, which can be done by adding a serial number to the CA and use a published **Certificate Revocation List** to invalidate.

9.2 Transport Layer Security

Transport Layer Security is used in `HTTPS` connections. The TLS handshake is outlined in the diagram below. For the boxed part, the key exchange can be done in different key



https://bank.com, N_C
pk, cert, N_B
Verify!
pk_{CA}
sk, pk, $\text{cert}_{CA\to Bank}$
mk = KDF(pmk,N_C,N_B)
k_C, k'_C, k_S, k'_S = G(mk)
c = Enc_{pk}(pmk)
pmk = Dec_{sk}(c)
mk = KDF(pmk,N_C,N_B)
k_C, k'_C, k_S, k'_S = G(mk)
Mac_{mk}(transcript)
Verify!
Mac_{mk}(transcript')
Verify!

exchange protocol.

- **RSA**: no forward secrecy

- **Fixed Diffie Hellman**: S has fixed DH key x_s; S's DH public key g^{x_s} is included in the certificate. The certificate include bank's public key and DH key. For C, it also uses x_C and g^{x_C} so that C and S can execute standard DH protocol.
- **Ephemeral Diffie-Hellman**: Both S and C create fresh secret keys for new sessions. S' DH public key g^{x_S} is signed by S's RSA private key. Then C and S carry standard DH protocol to generate a fresh *pmk*.

For record-layer protocol, since parties now share k_C, k'_C, k_S, k'_S, these key can be used for encryption and authentication. It prevents reflection attacks. Sequence numbers can be used to prevent replay attacks.

10 Digital Cash and Shamir's Secret Sharing

For today's digital cash, we hope that we can achieve

- Only a previously withdrawn coin can be deposited
- Coins must be impossible to deposit twice
- Shop accept withdrawn coins
- Pay is done anonymously
- Vendor is assured Bank will accept a payment received from the buyer

Currently, we cannot acheive anonymous payment, since it requires

- the bank be impossible to trace a coin
- us to protect honest customer's privacy AND preventing misuse

Definition 10.1 (Blind Signature)**.**
Blind signature requires the signer to sign a message *without* seeing its content. Later when message and signature are revealed, the signer should not be able to link the signature with the corresponding signing transaction. However, the signature needs to be verifiable by everyone.

We introduce Schnorr's identification protocol, which is the fundamental of Schnorr's blind signature.

Theorem 10.1 (Schnorr's Identification Protocol)**.**
Let p, q be large primes that satisfies

$$q \mid p - 1$$

Let G_q be the subgroup of order q in $\mathbb{Z}_p^*$. Let g be a generator of G_q. (A special choice of p, q, G_q is actually $G_q = QR_p$, and $p = 2q + 1$)
Let $H : \{0,1\}^* \to \mathbb{Z}_p^*$ be a collision-resistant hash.
We set the **prover's** secret key x to be $x \in \mathbb{Z}_q$. Its public key is a tuple:

$$(p, q, g, y := g^x)$$

The Schnorr's identification protocol wants to achieve that the veritifier is convinced that prover has the private key x corresponding to y without learning x.:

- Prover chooses $r \in \mathbb{Z}_q$, and sends $a := g^r$ to verifier
- Verifier chooses $c \in \mathbb{Z}_q$, and sends it to prover
- Prover computes $b = r - cx$ and sends it to verifier.
- Verifier accepts the proof if $a = g^b y^c$. Otherwise reject.

Theorem 10.2 (Schnorr's Signature)**.**
The signer will have the same setting as identification protocol. Upon signing the message m, he will produce a pair (c, b) where

$$c := h(m\|a)$$

and

$$b = r - cx$$

The signature can be verrified by checking $c = h(m\|g^b y^c)$.

This allows the signature scheme to be non-interactive. This scheme relies on the hardness of discrete logarithms of elements of G_q.
Suppose we have observed a valid transcript (a', c', b') of a Schnorr's identity protocol execution. Consider the transform below:

- $u, v, w \in \mathbb{Z}_q$.
- $a := a'^u g^v y^w$
- $c = uc' + w$
- $b = ub' + v$

The (a, c, b) will be another accepting transcript of Schnorr's identity protocol. With this observation, we can devise Schnorr's blind signature.

Theorem 10.3 (Schnorr's Blind Signature)**.**
Let the signer's setting to remain the same.

- Signer picks a $r' \in \mathbb{Z}_q$ and sends $a' := g^{r'}$ to verifier.
- Verifier initialize $u, v, w \in \mathbb{Z}_q$, compute $a := a'^u g^v y^w$. It let $c = h(m\|a)$ and $c' = (c - w)u^{-1}$. It sends c' to signer.
- Signer returns $b' := r' - c'x$ to verifier.
- Verifier computes $a' := g^{b'} y^{c'}$ and $b := ub' + v$. It verifies whether (c, b) is valid Schnorr's signature.

This is good since signer does not learn m, (c, b). Also, verifier does not learn x. However, anyone can verify (c, b).

Theorem 10.4 (RSA BBlind Signature)**.**
Let Signer's secret key be (N, d) and public key be (N, e).
The verifier will pick $r \in \mathbb{Z}_N^*$ and compute $m' = H(m)r^e \mod N$. Verifier sends m' to signer.
The signer will compute $x' := (m')^d \mod N$ and sends x' to verifier.
Verifier will compute $x = r^{-1}x' \mod N$, which is the ordinary RSA signature.

10.1 Splitting Secret Information

We hope the secret can be split in an information-theoretic way.

Definition 10.2 (Threshold Secret Sharing Scheme)**.**
Let t, w be positive integers where $t \leq w$. A (t, w)-threshold scheme is a method of sharing a secret K among a set of w participants, in such a way that any t participants can compute the value of K but no group of $t-1$ participants can do so.
Let $P_1, \ldots, P_w$ be participants, and let the data held by P_i be s_i. We assume that there is a dealer who computes the *shares* from K and securely sends the shares to the participants.

Theorem 10.5 ((t, w)-Shamir Threshold Scheme)**.**
Initialization Phase: Dealer chooses a sufficiently large prime p, where $p \geq w$ and sends it to all participants. Each participant is assigned a distinct index, from 1 to w. Let P_i be the participant assigned to index i.
Share Distribution:

- Given the secret K to be shared, the dealer randomly chooses $t-1$ elements $a_1, \ldots, a_{t-1}$ from $\mathbb{Z}_p$.
- Let $f(x) = (a_{t-1}x^{t-1} + \cdots + a_1 x + K) \mod p$
- The dealer computes $s_i := f(i)$ for $i = 1, \ldots, w$ and **securely** sends s_i to P_i.

One can use the Vandermonde Matrix to recover the secret K.
However, here we need a honest dealer. In case the dealer may not be honest, we need a **verifiable secret sharing scheme**, where each share of secret is attached with auxiliary data to help others verify whether the shares are correct.

Theorem 10.6 (Feldman's (t, w)-VSS scheme)**.**
Initialization Phase: Dealer choose a cyclic group G of prime order p such that discrete log is difficult, and a generator g of G. Dealer broadcasts p, q, g.
Share Distribution:

- Given the secret K to be shared, the dealer randomly chooses $t-1$ elements in $\mathbb{Z}_p$ dentoed as $a_1, \ldots, a_{t-1}$.
- Let $f(x) = a_{t-1}x^{t-1} + \cdots + a_1 x + K$.
- The dealer computes $s_i = f(i)$ and securely sends s_i to P_i.
- In addition, dealer broadcasts $c_j = g^{a_j} \mod q$ for $j = 0, \cdots, t-1$.

Reconstruction is the same. For vertification, participants should check whether s_i is consistent via c_j's by doing:

$$g^{s_i} = (c_{t-1})^{i^{t-1}} \cdots (c_1)^i \cdot c_0 \mod q$$

We can understand this scheme by treating c_i as a_i's digest.

This scheme depends on the security of discrete log. Therefore, it is not infomation theoretically secure. This poses a problem if entropy of K is low.
To fix this, we can use a semantically secure homomorphic PKC as the hash, e.g., Paillier Cryptosystem.

10.2 Distributed Ledger

10.3 Crypto Applications

10.4 Commitment Schemes

One commitment scheme can be achieved using SHA-256. Suppose they want to exchange a single bit of content. Then,

- Alice randomly picks r_A, a 511 bits sequence, and her choice of a. Similarly, Bob picks r_B and his choice of b.
- Alice computes $c_A := SHA(r_A \| a)$; Bob computes $c_B := SHA(r_B \| b)$.
- Alice sends to Bob c_A and Bob sends Alice c_B. (*exchange commitment*)
- Alice sends Bob a, r_Al Bob sends Alice b, r_B.(*open commitment*)
- Alice and Bob verify c_A and c_B respectively. The final random bit is $a \oplus b$.

The binding property of the commitment scheme, says that one cannot change the commitment. Binding can be achieved as long as the commitment scheme is 2nd pre-image resistant. The hiding property of the commitment scheme, says that one cannot determine the other people's committed value. This will be achieved if the commitment scheme is pseudorandom.

Definition 10.3 (Commitment Scheme).
A commitment scheme involves 3 entities: Alice, BOb and a trusted party T. Alice is known as the **prover** whereas Bob is the **verifier**.
There are three phases: setup, commit, open:

- Setup: trusted T helps Alice and Bob establish a common public key.
- Commit: Alice commits her choice
- Open: Alice opens the commitment to convince Bob that she does not change her mind.

There are two security requirements: binding and hiding.

Definition 10.4 (Binding).
Binding means that Alice cannot change the committed value, so that the value revealed is the original choice of Alice.
There are two levels of security:

- **Unconditionally secure**: Even if Alice has arbitrary computing resources and time, she is unable to change the commited value
- **Conditionally secure**: Alice is unable to change the value, assuming certain problems are computationally difficult to solve

Definition 10.5 (Hiding).
Hiding means Bob is unable to determine the value from Alice's commitment(confidentiality).
There are two levels of security:

- **Unconditionally secure**: Even if Bob has arbitrary computing resource and time, he is unable to determine the committed value.
- **Conditionally secure**: Bob is unable to determine the value, assuming certain problems are computationally difficult to solve.

Theorem 10.7. It is **not possible** to have a scheme that is unconditionally secure for both binding and hiding.

One example is an RSA-based unconditionally binding scheme.

1. Setup: T choose N and sends (e, N) to Alice and Bob.
2. Commit: To commit a bit a, Alice randomly pick an *non-zero* number r from $\mathbb{Z}_N$, such that its *least significant bit* is a. The commitment is
$$c = r^e \mod N$$
3. Open: Alice reveal a.

Here, Alice is unconditioanlly bound with a, since there does not exist another s such that $c = s^e \mod N$.
Also, Bob is not able to determine a from c, assuming RSA is hard.
Another example is the unconditionally hiding scheme is ElGamal based.

1. T picks a prime p, an element g with order $p-1$. T picks a random x and computes $h = g^x \mod p$. T sends (p, h, g) to Alice and Bob. Alice does not know x.
2. To commit a, Alice picks a random r from $\mathbb{Z}_p$ and computes the commitment
$$c = g^r h^a \mod p$$
3. To open, Alice reveals a and r. To check, Bob reencrypt.

This achieves unconditionally hiding since whether $a = 0$ or 1, the corresponding r can be equally likely chosen. However, this is conditionally binding, since if Alice can find
$$c = g^r h^0 = g^s h^1 \mod p$$
she can break this scheme. However, this requires Alice to break discrete log.
There are other desired properties specific to some scenarios, for example:

- Alice has commitment c_1, c_2, c_3, and Alice need to convince $c_3 = c_1 + c_2$.
- Alice has commitment c_1, c_2 and wants to convince Bob that $c_1 \neq c_2$.

Definition 10.6 (Homomorphic Commitment).
Suppose c_1, c_2 are commitment of b_1, b_2, say

- $c_1 = g^{r_1} h^{b_1} \mod p$
- $c_2 = g^{r_2} h^{b_2} \mod p$

Then we have

$$c_1 c_2 = g^{r_1 + r_2} h^{b_1 + b_2} \mod p$$

which is the commitment of $b_1 + b_2 \mod p - 1$.

10.5 Zero Knowledge Proof

Definition 10.7 (Zero-Knowledge Proof)**.**
A protocol is zero-knowledge if it can be simulated in the following way: for any polynomial-time verifier V, there exists a polynomial-time algorithm which can produce, without interacting with the prover P, a trace that is indistinguishable from the communication between V and P.

One example is the following, using square root.
Let $y = s^2 \mod N$. Bob and Alice both know y. Alice claims that she knows s, which is the square root of y.

- Alice chooses a random r_1, and she finds r_2 such that $r_1 r_2 = s \mod N$.
- She sends $x_1 := r_1^2 \mod N$ and $x_2 : r_2^2 \mod N$ to Bob.
- Bob can verify $x_1 x_2 = y \mod N$. Bob sends the challenge c, which can be 1 or 2.
- Alice sends r_c.
- Bob accepts if r_c is indeed the square root of x_c.

We can verify easily the completeness and soundness of the scheme. Completeness means if Alice knows s, Bob will always accepts. Soundness means if Alice does not know s, it can only make Bob accept with probability $0.5 + \mathbf{negl}(n)$.
Another zero-knowledge proof of committed value is non-zero is listed below:

- $c := g^r h^a \mod p$
- Alice wants to show $a \neq 0$, without revealing r and a.
- Alice randomly picks $s, t \in \{1, 2, \ldots, p-1\}$ where $\gcd(s, p-1) = 1$.
- Alice sends to Bob $X := c^s g^t \mod p$
- Bob randomly picks $C \in \{0, 1\}$ and sends to Alice
- Alice sends (s, t) if $C = 0$, or $(sa \mod (p-1), (sr + t) \mod (p-1))$ if $C = 1$.

10.6 Anonymous Communication and Mixnets

Internet is designed as a public network, so machine on your LAN may see your traffic, whereas routers see all traffic. Encryption does not hide identities, since it does not hide routing information.
Basic Mix design requires any meesage sender to encrypt the message M as

$$\{r_i, \{r_j, M\}_{pk(D)}, D\}_{pk(mix)}$$

where r_i, r_j are random numbers, and D is the receiver.
To allow anonymous return address, M shoudl include $\{K_1, A\}_{pk(mix)}$ and K_2 where K_2 is a fresh public key.