CSCI471/971
Modern Cryptography

Cryptographic Notions

Rupeng Yang

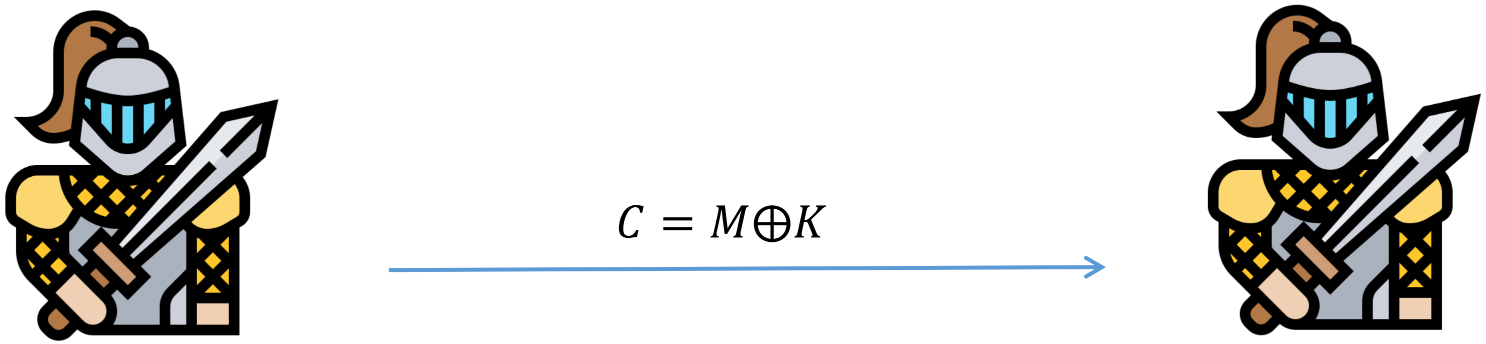
SCIT UOW

RoadMap

- Week 1: The classical cryptography
- Week 2: Towards modern cryptography
- Week 3-12: Modern cryptography

Shannon’s perfect cipher

How to send one bit securely



Alice
M,K

Bob
K

Alice and Bob share a secret bit K that is a random bit.
Alice would like to send one bit M to Bob securely.
She first computes the ciphertext $C = M \oplus K$ and sends C to Bob
Then Bob can recover M by computing $M = C \oplus K$

Is the solution secure? For example, given a ciphertext C=0, can you guess if M=0 or M=1.

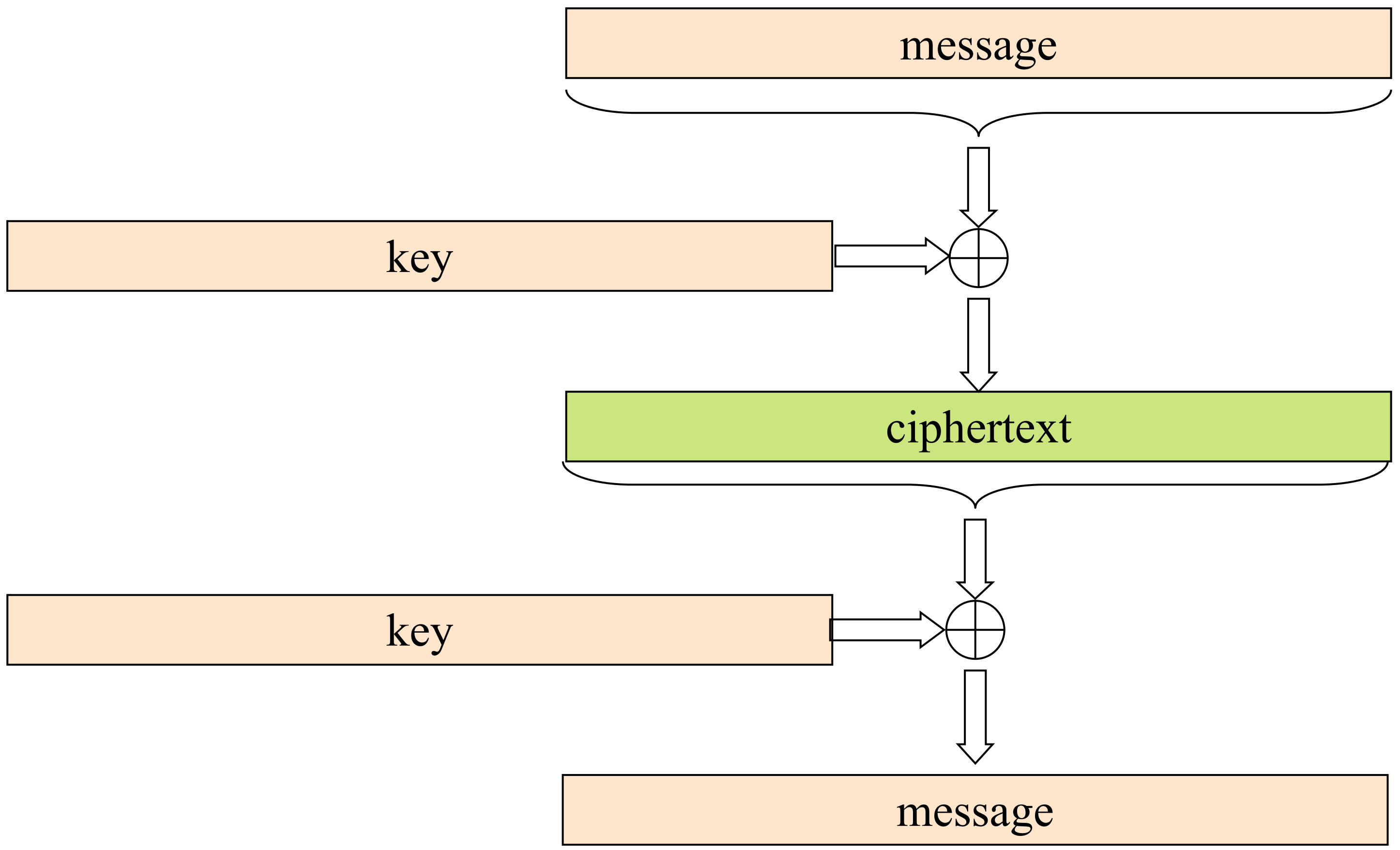
One-time Pad (Vernam Cipher)

- Gilbert Vernam invented a cipher that was extended by Joseph Mauborgne to give a scheme which was later proved to provide *perfect security* by Claude Shannon.

- The cipher is called one-time-pad because the key is written on a long tape and is ***used only once***.

- The cipher does not rely on any assumption and no adversary can do better than simply guessing the message.

One-Time Pad

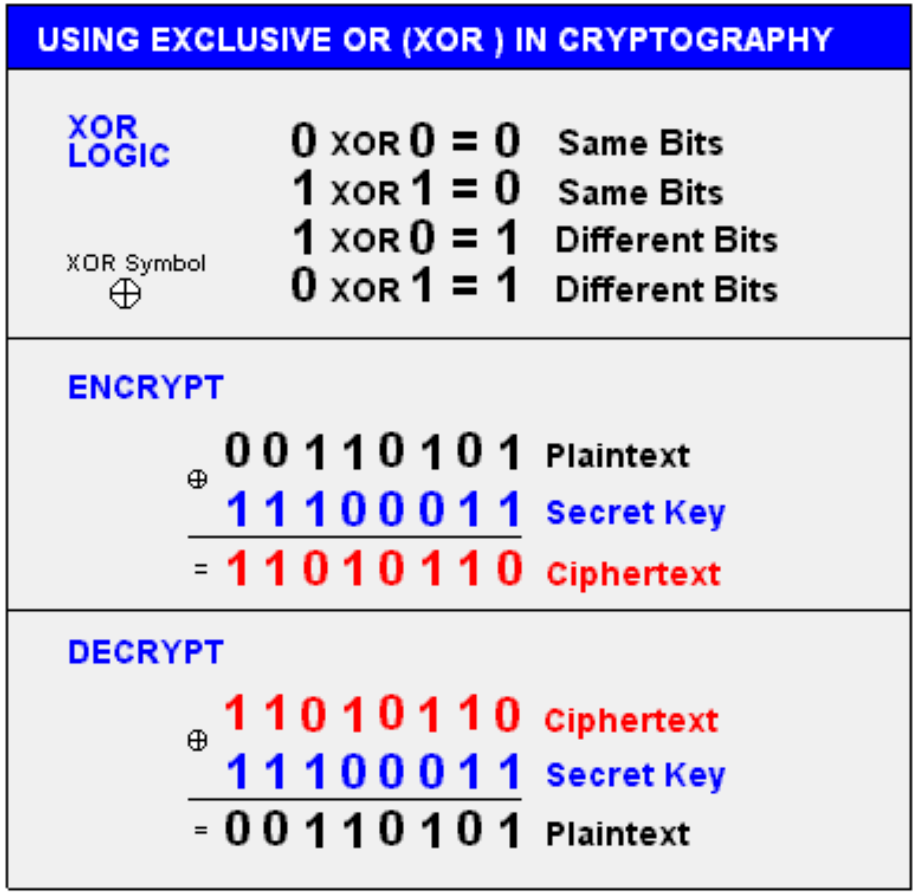


One-Time Pad

- KeyGEN
  - output a random sequence $K_1K_2\dots K_n$ of n bits

- Enc $(X = X_1\dots.X_n, K = K_1\dots K_n) \rightarrow Y = Y = Y_1\dots Y_n$
  - $Y_i = X_i \oplus K_i$

- Dec $(Y = Y_1\dots Y_n, K = K_1\dots K_n) \rightarrow X = X_1\dots.X_n$
  - $X_i = Y_i \oplus K_i$



Perfect Security

We can define perfect security in several different ways:

Def 1:	Def 2:
For every message m in the message space M, and every ciphertext c in the ciphertext space C: $$\mathrm{Pr}[\mathrm{M} = \mathrm{m}] = \mathrm{Pr}[\mathrm{M} = \mathrm{m} \mid \mathrm{C} = \mathrm{c}]$$ That means, knowledge of the ciphertext does not help the attacker to guess the plaintext.	For every pairs of messages m_0, m_1 in the message space M, and every ciphertext c in the ciphertext space C: $$Pr[\ C = c \mid M = m_0] \ = Pr[\ C = c \mid M = m_1]$$ That means, the probability that C = c is the same for M = m_0 or M = m_1.

And they are essentially equivalent.

One-Time Pad offers Perfect Security

- Let (m_0, m_1) be the plaintext pair chosen by the attacker
  - Note that the ciphertext c is returned to the attacker and the attacker's goal is to tell if c is the encryption of m_0 or m_1
- Argument:
  - c can be the encryption of m_0 with the key $k_0 = m_0 \oplus c$
  - c can also be the encryption of m_1 with the key $k_1 = m_1 \oplus c$
  - Since the key is randomly chosen, the probability that the key is k_0 or k_1 is equal
- **In one-time pad, the key can only be used once.**

Necessary Condition for Perfect Security

Theorem (Shannon)
In a system with perfect secrecy the number of keys is at least equal to the number of messages.

- Argument:
  - If the message space is larger than the key space, then given a fixed ciphertext c that encrypts a message m, there exists some message m' that cannot be encrypted to c.
  - Therefore, we have Pr $[C = c \mid M = m] \neq 0$, Pr $[C = c \mid M = m'] = 0$, which contradicts the perfect indistinguishability.
- Implications: If we want perfect secrecy
  - the key size must be large.
  - The key can only be used once.
    - If the key is used for multiple times, the total message length will be larger than the key length.

Necessary Condition for Perfect Security

Theorem (Shannon)
In a system with perfect secrecy the number of keys is at least equal to the number of messages.

- Implications: If we want perfect secrecy
  - the key size must be large.
  - The key can only be used once.
    - If the key is used for multiple times, the total message length will be larger than the key length.
- Now, if Alice and Bob have shared a 100-bit secret key in advance, how many bits can they transform securely if they would like to achieve perfect security?
  - Why this is a problem in practice?
- Now, assuming that we have a cryptosystem that is able to encrypt n+1 bits using an n-bit secret key, how many bits can be transformed between Alice and Bob securely, if they have shared a 100-bit secret key in advance?

Cryptosystems with Computational Security

Necessary Condition for Perfect Security

Theorem (Shannon)
In a system with perfect secrecy the number of keys is at least equal to the number of messages.

- Implications: If we want perfect secrecy
  - the key size must be large.
  - The key can only be used once.
    - If the key is used for multiple times, the total message length will be larger than the key length.
- The perfect security guarantees that the plaintext is completely hidden from the ciphertext. This holds even if the attacker has unbounded resources.
- But do we really need security against an unbounded adversary?
  - What if the attack will take the attacker 1000 years?
  - What if the ciphertext can only increase the probability to guess the message by $1/2^{1000}$?
- Can we circumvent the limitation by only considering a practical adversary?

Modern Cryptography

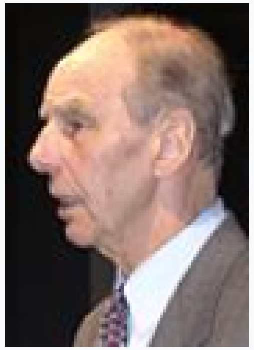
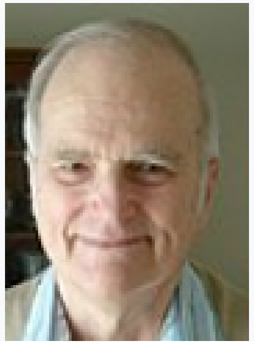
- Cryptography can be broken but it is very hard to break it

- How much hard?

- For example, if cryptography is well-designed, it will take at least 200 years using all computers in the whole world.

- How to describe this formally and How to achieve this?
  - Computational Complexity

1965: paper with Turing Award
"On the Computational Complexity of Algorithms"

1993	Juris Hartmanis		In recognition of their seminal paper which established the foundations for the field of computational complexity theory.[32]
	Richard E. Stearns		

Concepts in Complexity

- Problem
  - A problem asks you to find a solution given an instance.
  - For example, the sorting problem
- Algorithm
  - An algorithm solves a class of specific problems.
  - For example, the heapsort algorithm for the sorting problem
- Costs
  - The resources (e.g., running time, memory) that an algorithm requires.
  - In computational complexity, we consider the costs as a function on the input length of the algorithm.
    - If you have a larger instance (e.g., a longer list to be sorted), the algorithm will typically need more resource.
  - The runtime of the heapsort algorithm is O(n*logn) given list containing n elements.

Big O natation

$f(n) = O(g(n))$ if there exists a positive real number c and a real number x_0 such that

$$f(n) \le cg(n) \text{ for all } n \ge x_0$$

Polynomial time: There are some value k, $f(n) = O(n^k)$

Usually, an algorithm that can solve problem A in polynomial time is called an efficient algorithm.

Big $\boldsymbol{\Omega}$ natation

$f(n) = \Omega(g(n))$ if there exists a positive real number c and a real number x_0 such that

$$f(n) \geq cg(n) \text{ for all } n \geq x_0$$

Exponential time: $f(n) = \Omega(2^n)$

Usually, if all algorithms that can solve problem A are in exponential time, we say the problem is a hard problem.

Computational complexity in 5min

- How we can say a problem is easy?
  - Find an algorithm that solves it in polynomial time~

- How we can say a problem is hard??
  - Justify that all algorithms that solve the problem are in exponential time!
    - I have tried for a while but no efficient algorithm has found 😕
    - I have tried 10 yeats, but no efficient algorithm has found 🙁
    - ….
    - Many (including the top) researchers have tried for 50 years, but no efficient algorithm has found 😀
      - For example, the hardness of problems in the class NPC.
      - Some less studied problem like factoring problem, discrete log problem, …
    - We assume that these long-standing problems are hard.
  - Once we have accepted that some problems are hard, we can get more hard problems.

Computational complexity in 5min

- How we can say a problem is easy?
  - Find an algorithm that solves it in polynomial time~

- How we can say a problem is hard??
  - Assume that all algorithms that solve the problem are in exponential time!
  - Once we have accepted that some problems are hard, we can get more hard problems.
    - Let Problem A be the problem assumed to be hard.
    - We can “prove” that B is also hard by:
      - First, suppose that there exists an efficient algorithm F that can solve B.
      - Construct an algorithm F’ that can solve A based on F. Here, we do not have to know how F works.
      - Make a contradiction.
    - The proof above is called a reduction.

Complexity vs Cryptography

- In complexity, we have defined hard problems, i.e., problems that will always cost a lot of resource to solve.
- In cryptography, we require that recovering the plaintext from the ciphertext will always cost a lot of resource.
- We can fit cryptography into computational complexity!

- Problem: Recovering the plaintext from the ciphertext.
- Potentional algorithms: The attackers!
- Problem is hard: All attackers that can recover the plaintext from the ciphertext are in exponential time, i.e., all attackers will spend a lot of resources (e.g., at least 1000 years) to recover the plaintext.

Complexity vs Cryptography

- Problem: Recovering the plaintext from the ciphertext.
- Potentional algorithms: The attackers!
- Problem is hard: All attackers that can recover the plaintext from the ciphertext are in exponential time, i.e., all attackers will spend a lot of resources (e.g., at least 1000 years) to recover the plaintext.

- How to show that a cryptosystem is secure?
  - Try to find an efficient attacker for many years; OR
  - Start with some well-accepted hard problems (assumptions) and make some reductions.
- But for both approaches, we need to define the problem formally first.
  - We need to define a cryptosystem and its security.

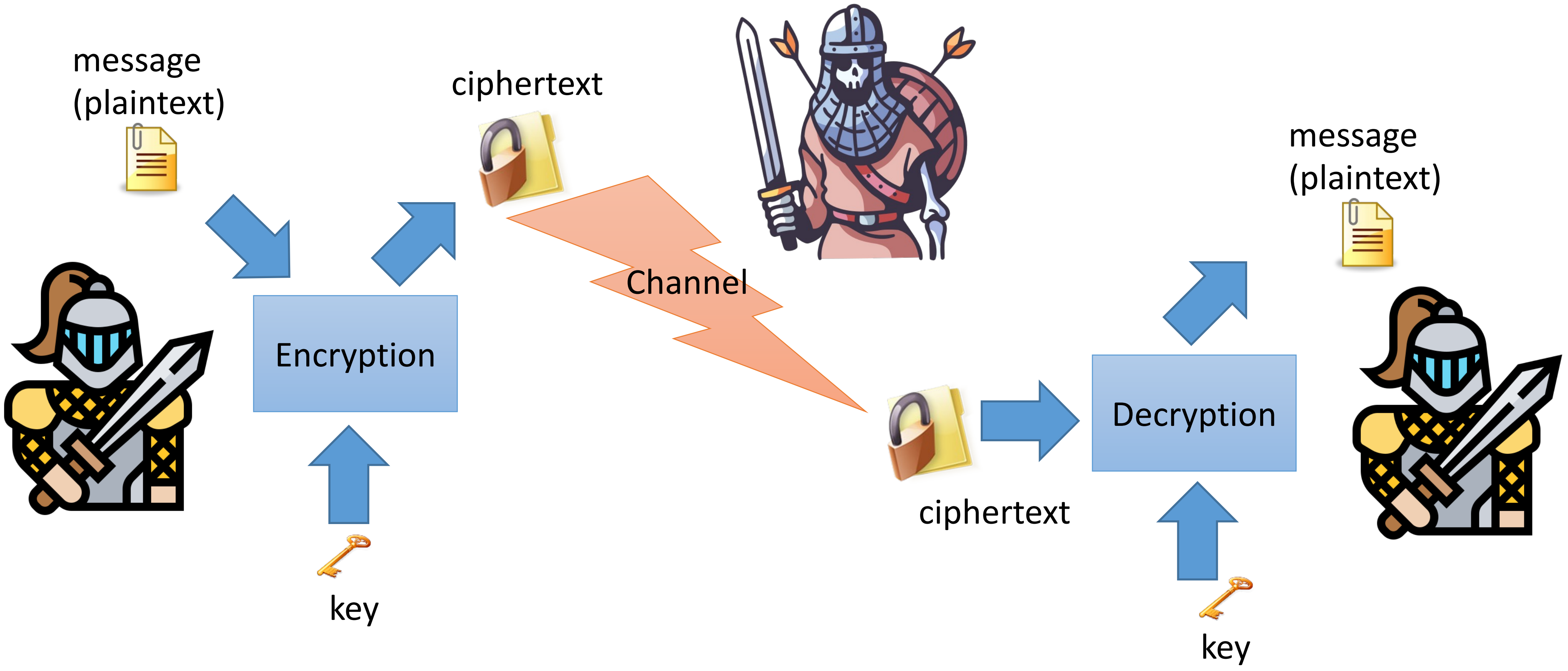
How to define a cryptosystem?

How to define a cryptosystem?

- A cryptosystem is a set of polynomial-time algorithms to provide security properties.
- What algorithms do we need?
  - How many algorithms do we need?
  - What are inputs and outputs of algorithms?
  - Probabilistic or deterministic?
  - …
- What Properties do we need?
  - Correctness requirements.
  - Security requirements.

- As a concrete example, we will see how to formally define an encryption scheme.

Symmetric-Key Encryption



Symmetric-Key Encryption

Application Scenario:

All users can generate and share keys. When two person Alice and Bob share the same key. One party can encrypt a sensitive message using the key and the other party can decrypt the ciphertext using the same key.

Symmetric-Key Encryption

- KeyGen(λ): Taking as input a security parameter λ, the key generation algorithms returns a key K

- Encrypt(K, M): Taking as input a message M and a key K, the encryption algorithm returns a ciphertext denoted by C.
$$C \longleftarrow \mathrm{Encrypt}(K,M)$$

- Decrypt(K,C): Taking as input a ciphertext C and a key K, the decryption algorithm returns a message M.

- The security parameter λ is denoted by security strength.

Symmetric-Key Encryption: Correctness

- **Correctness**: For all generated K and all C$\leftarrow$Encrypt(K,M), we have

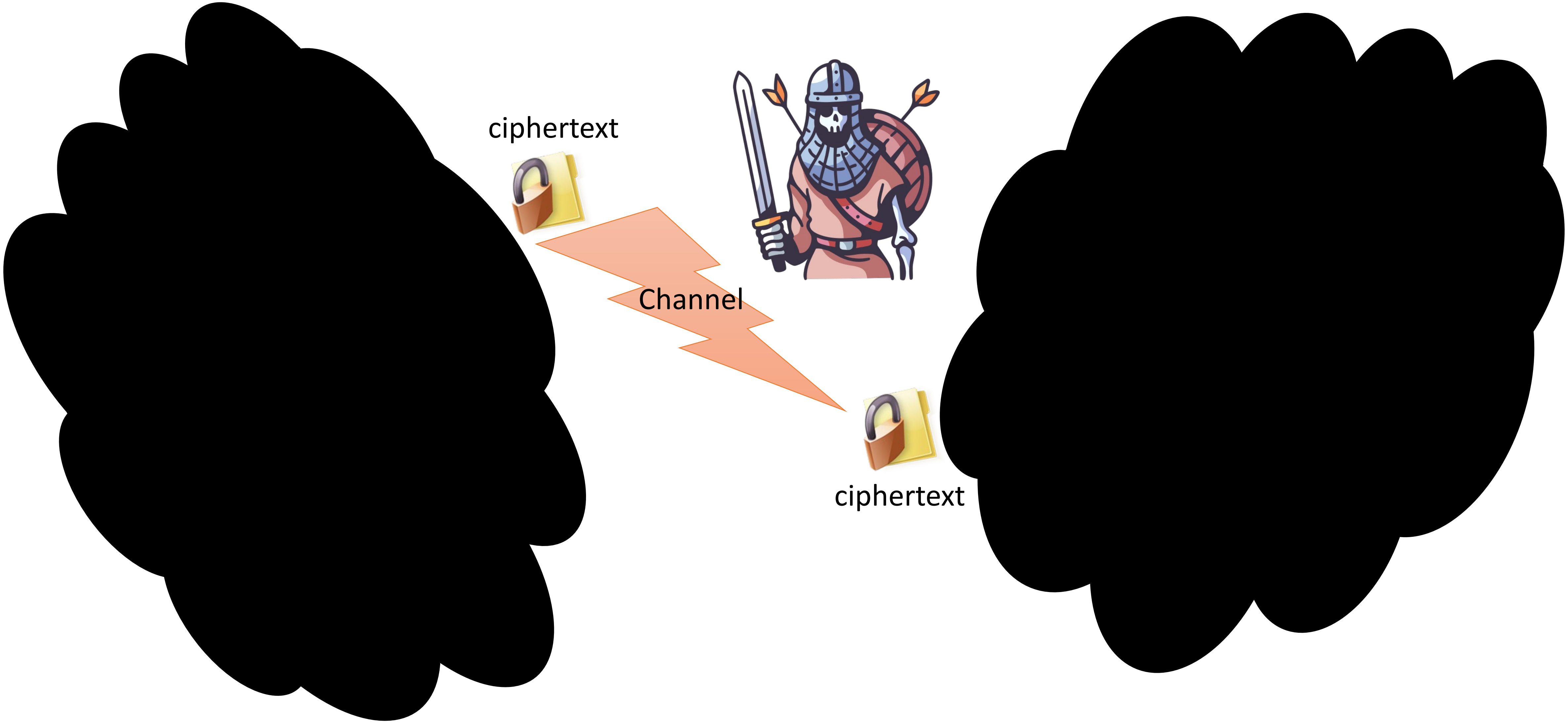
$$\Pr[\text{Decrypt}(K,C)=M]=1$$

- **Correctness (with correctness error)**: For all generated K and all C$\leftarrow$Encrypt(K,M), we have

$$\Pr[\text{Decrypt}(K,C)=M] \geq 1-\text{negl}(\lambda)$$

- We use negl to denote a negligible function, which is smaller than all inverse polynomial.

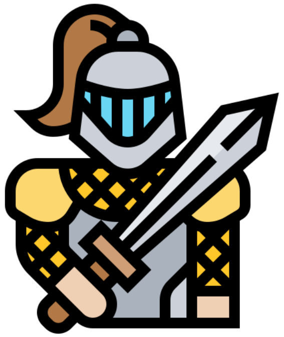
Symmetric-Key Encryption: Security



Symmetric-Key Encryption: Security

Game-Based Security Definition:

Algorithms KeyGen, Enc, Des are public.



Attacker wins if O satisfies some condition!

.
.
.

4. Attacker returns an output O.

An Encryption Scheme is Secure if ***NO*** efficient attacker can win with a good probability.

Efficient attacker: The attacker is a probabilistic polynomial time algorithm.

Symmetric-Key Encryption: Security

- The game-based definition is described by a game between an adversary and a challenger.
  - The challenger represents the secret key owners of the system.
  - The adversary is the attacker trying to break the cryptosystem.
- In a game-based definition, we only consider abstract attacks from the adversary, which focus on what information can be learned by the adversary rather than how the information can be learned.
- When defining the security, we need to consider
  - The adversary's capabilities:
    - What information can be learned by the adversary and When the adversary can learn the information.
    - This is described by allowing the adversary to make some queries to the challenger.
    - The challenger must answer the queries honestly.
  - The adversary's security goal:
    - How the adversary wins the game.

Symmetric-Key Encryption: Security

What is the adversary's security goal?

- One-Wayness: Given a ciphertetx C*=Enc(K,M*) generated by the challenger, the adversary is going to compute its plaintext M*.

This observation is straightforward but .....

Symmetric-Key Encryption: Security

What is the adversary's security goal?

- Semantic-Security : Given a ciphertetx C*=Enc(K,M*) generated by the challenger, the adversary's goal is to learn any information about M*.

This seems a reasonable goal, but how to define it?

Symmetric-Key Encryption: Security

What is the adversary's security goal?

- Indistinguishability : Given a ciphertext C* and two messages M_0 and M_1 where C*=Enc(K,M_b), the adversary is going to compute b from {0,1}.

Question: Who chose M_0 and M_1?

This security goal is equivalent to the semantic-security in most cases!

Symmetric-Key Encryption

What are the capabilities of the adversary?

- Ciphertext-Only Attack:

The adversary knows some ciphertexts.

- Known-Plaintext Attack:

The adversary knows some plaintext-ciphertext pairs.

- Chosen-Plaintext Attack:

The adversary can choose any plaintext to know its ciphertext.

- Chosen-Ciphertext Attack:

The adversary can choose any plaintext to know its ciphertext.
The adversary can choose any ciphertxt to know its plaintext.

Symmetric-Key Encryption: IND-CPA Security

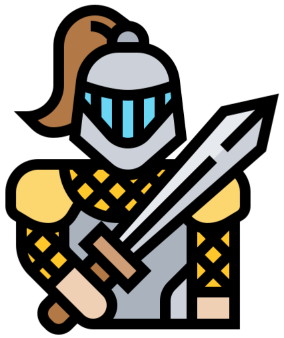
1. Run KeyGen to get k

Algorithms KeyGen, Enc, Des are public.

2.1 Attacker sends m to challenger

2.2 Run Enc(k, m) to get c

2.3 Send c to the Attacker



3.1 Attacker sends m_0, m_1 to challenger

3.2 Randomly choose a bit b, run Enc(k, m_b) to get c*

3.3 Send c* to the Attacker

We assume that ($|m_0|=|m_1|$)

4.1 Attacker sends m to challenger

4.2 Run Enc(k, m) to get c

4.3 Send c to the Attacker

5. Attacker returns a guess bit b'

Attacker wins if b = b'!

An Encryption Scheme is Secure if ***NO*** efficient attacker can win with a probability of ½+1/poly(λ). Alternatively, that means,

Security Model of Symmetric-Key Encryption (IND-CPA)

<u>Setup:</u> The challenger chooses a random key K.

<u>Phase 1:</u> The adversary can choose any M for encryption queries and learns the encrypted result.

<u>Challenge:</u> The adversary can choose any two different messages M_0 and M_1. The challenger chooses a random b and computes the challenge ciphertext CT*=Enc(M_b, K), which is given to the adversary.

<u>Phase 2:</u> The adversary can choose any M for encryption queries.

<u>Guess:</u> The adversary returns the guess b' and wins if b'=b.

We say that the encryption is secure if no P.P.T adversary can win with a probability of ½+1/poly(λ).

Summary

- One-time pad
  - Construction
  - Security
  - Limitations
- Computational Security*
  - Computational Complexity
  - Reduction
  - Modern Cryptography – An Overview
- Definition
  - Syntax
  - Correctness
  - Security
    - Game-based definition
    - The adversary's capabilities
    - Security goals
    - The security definition