

# Topic 1: Encryption

(For Confidentiality)

1.1: Definition: Encryption/decryption/Keys

1.2: Security Model and Requirement

1.3: Classical ciphers + illustration of attacks

1.4: Modern Ciphers + recommended key length

1.5: Examples of attacks on crypto

1.5.1: Meet-in-the-middle

1.5.2: Padding Oracle & notion of “Oracle”

1.6: Pitfalls in usages and implementations

1.7: Interesting historical facts

# Summary & takeaways (1)

- Encryption is designed for confidentiality. (not necessary provides integrity, although some method (e.g. AES GCM mode) do.)
- Threat model defines types of attacks to be considered. (threat model is useful in security analysis, not just encryption)
  - Attacker's goal: total break → distinguishability
  - Attacker's capability: ciphertext only → plaintext only → encryption oracle → decryption oracle.

Defender wants a method that is secure under most “humble” attacker's goal, and stronger attacker's capability. A system  $S_1$  is more secure than  $S_2$  wrt to the threat model, when for any attack that can be prevented in  $S_2$ , it can also be prevented in  $S_1$
- Notions of “Oracle”.
  - Encryption Oracle, aka CPA (chosen plaintext attack) (oracle's output can be obtained from, e.g. smart card, probing of server)
  - Decryption Oracle. Padding Oracle Attack (know the detailed mechanism). (oracle output can be derived in many real life system)
- Key strength: Quantifying security by equivalence of best-known attack to exhaustive search. (e.g 2048-bit RSA key has key strength of ~128 bits. )
- No known efficient attacks on modern schemes (e.g. AES) under the intended threat models, but there are pitfalls
  - Implementation error: using known insecure crypto, wrong mode, wrong random sources, mishandling of IV.
  - side-channel information attack. (the intended threat model does not consider information available to the attacker that turns out to be feasible)
  - Implicitly require integrity. (the intended threat models does not consider attackers' goal that turns out to be crucial )

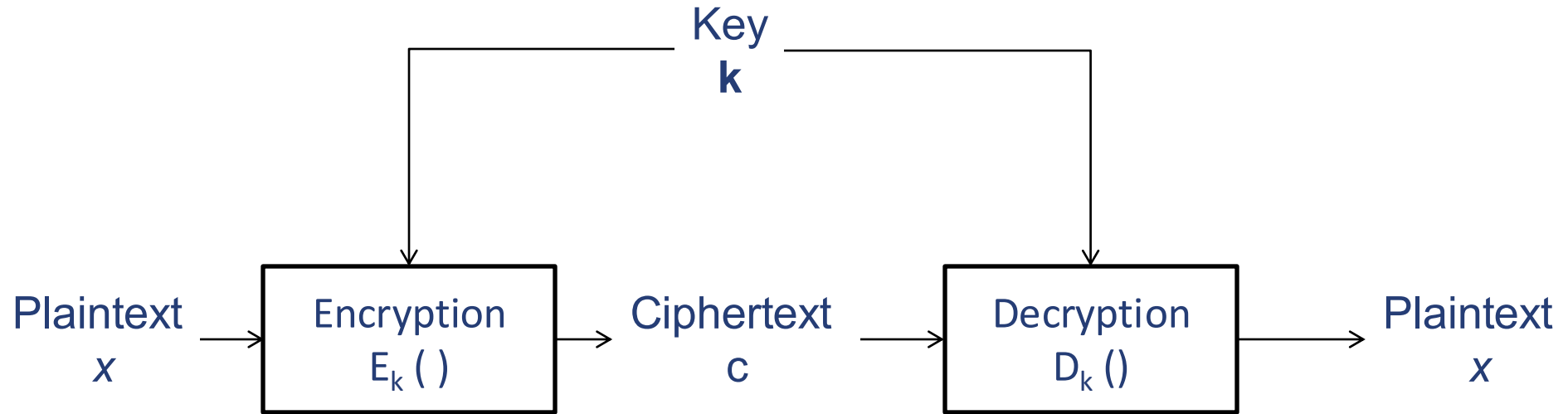
# Summary & takeaways (2)

- Designs of various symmetric key encryption schemes
  - One-time pad. “unbreakable” even if attacker has sufficient time to exhaustively search.
  - Stream Cipher. xor’ing with a “pseudo-random” string.
  - Block Cipher. Mode of operations.
    - CBC: provides some form of integrity. (Secure against CPA. vulnerable to padding oracle attack, BEAST attack, might achieve some forms of integrity. To secure against BEAST, IV needed to be unpredictable (random is more general and will do).) \*USE THIS WITH CAUTION\*
    - ECB: flexible but leak info. Deterministic (no IV involved). \*DO NOT USE\*
    - CTR: stream cipher. \*USE THIS WITH CAUTION\*
    - Secure against CPA; vulnerable to padding oracle attack if padded. No “integrity” at all and easily change (aka malleable). If IV of two ciphertext is the same, leak significant information (in contrast, if IV of two ciphertext under CBC is the same, there are some leakage but not as bad).
    - GCM: Authenticated-Encryption (AE). Achieved both integrity & confidentiality. Secure against Decryption Oracle. Only standardized quite recently and thus not in some legacy systems. \*USE THIS\*
- Crucial role of IV. (need randomness to have indistinguishability)
  - Why? Make the encryption probabilistic. Eg when no IV.
  - Proper implementation

# **1.1 Definitions**

# Encryption (Symmetric key) we will see the asymmetric-key version later.

A *symmetric-key encryption scheme* (also known as *cipher*. We often drop the term “symmetric-key” for simplicity) consists of two algorithms: **encryption** and **decryption**.



A cipher must be correct and secure.

**Correctness:** For any plaintext  $x$  and key  $k$ ,  
$$D_k ( E_k (x) ) = x$$

**Security:** Challenging to define. Definition depend on the **threat models**. Informally, from the ciphertext, the eavesdropper is unable to derive useful information of the key  $k$  or the plaintext  $x$ , even if the eavesdropper can probe the system.

Remarks: Encryption could be probabilistic. That is, for the same  $x$ , there could be different  $c$ 's. Yet they all can be decrypted to the same  $x$ .

# An application scenario

Alice had a large file  $F$  (say info on her bank accounts and financial transactions in Excel). She “encrypted” the file  $F$  using winzip with a password “13j8d7wjnd” and obtained the ciphertext  $C$ . Next, she called Bob to tell him the 10-character password. Subsequently, she sent the ciphertext to Bob via email attachment. Later, Bob received  $C$  and decrypted the ciphertext with the password to obtain the plaintext  $F$ . It is possible that Bob and Alice are a same person.

Anyone, say Eve, who had obtained  $C$ , without knowing the password, was unable to get any information on  $F$ . Although  $C$  indeed contained info of  $F$ , the information was “hidden”. To Eve,  $C$  resembled a sequence of random bits. If Alice sent a truly random string instead of the actual ciphertext, Eve was not able to distinguish the two.

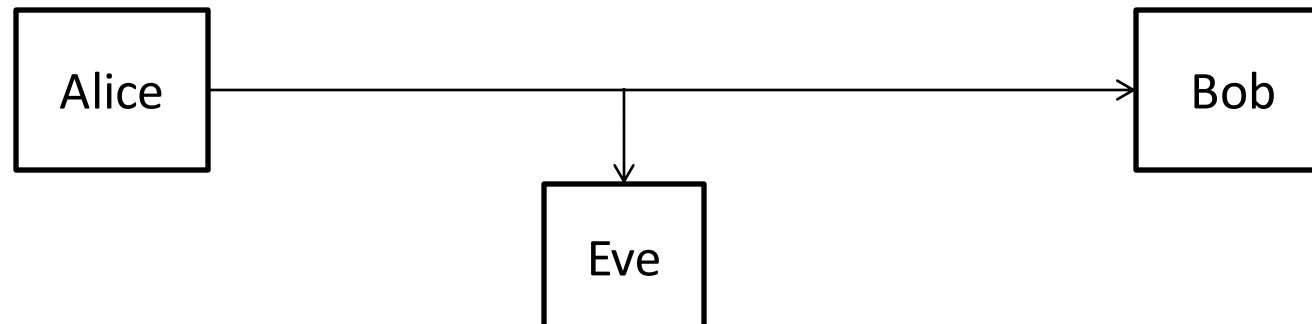


Remark:

- Winzip is **not** an encryption scheme. It is an application that employs standard encryption schemes such as AES.
- We need a method to convert password (human generated) to encryption key. Such method is called KDF, Key Derivation Function. Detail omitted.
- Some zip tools use an insecure crypto ZipCrypto. Do not use. Use those with AES.

# Cryptography

- Cryptography is the study of techniques in securing communication in the presence of attackers who have access to the communication.
- Although cryptography is commonly associated with encryption, there are other primitives such as cryptographic hash, digital signature, etc.
- Terminology: Common placeholders used in cryptography are Alice (usually the originator of message), Bob (usually the recipient), Eve (eavesdropper: can only listen), Mallory (malicious: can listen and modify messages), (see the interesting list in [https://en.wikipedia.org/wiki/Alice\\_and\\_Bob](https://en.wikipedia.org/wiki/Alice_and_Bob))
- Depending on context, Alice may not be a human. She could be the machine that encrypts the message.



## 1.2 Threat Model (aka Attack Model, Attack Scenario, Security model, adversary model, etc)

- Formulate security of encryption by describing the class of attacks it can prevent.
- A class of attacks is described by
  1. Attacker's goal.
  2. Attacker's capability (information, compute resource).




# Threat model

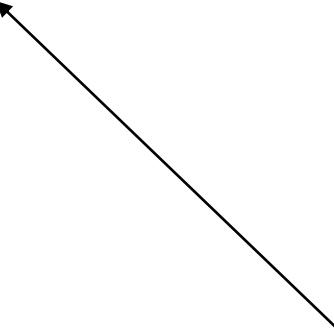
Security of a cryptography system, same for many other security systems, can be described by stating the class of attacks that it can be prevented. A class of attacks is described by:

1. Attacker's goal;
2. Attacker's capability.
  - Information given to the attacker
  - Computing power

*This can be datasets, or services (oracle) access.*



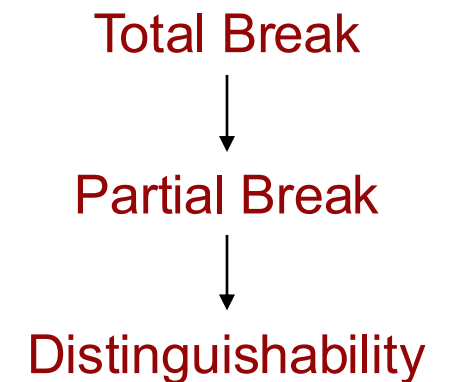
*In this course, we will not get into details of this. We consider attackers who have access to cloud/supercomputer and reasonable number of years (say 10000 years) to run the compute.  
Theoretical formulation use the notion of “polynomial-time” machine.*



# Threat Model: Attacker's goals

- (Total Break). The attacker wants to find the key. We call this goal **total break**.
- (Partial Break). The attacker satisfy with a **partial break**. There are a few definitions for that. For instance, the attacker may want to decrypt a ciphertext but not interested in knowing the secret key, or the attacker may simply want to extract some information about the plaintext. (e.g., whether the plaintext is a JPEG image or an Excel file).
- (Distinguishability) What is the most modest goal? **Distinguishability**. *With some “non-negligible” probability more than  $\frac{1}{2}$ , the attacker can correctly distinguish the ciphertexts of a given plaintext (say, “Y”) from the ciphertext of another given plaintext (say, “N”).* If attacker is unable to distinguish, we call this property **indistinguishability (IND)**.
  - For rigorous definition see the textbook: J. Katz & Y. Lindell, Introduction to Modern Cryptography, 2nd ed.

- Total break is the “most difficult” goal in the sense that, if an attacker can achieve total break, the attacker also can achieve partial break and distinguishability. Distinguishability is the weakest goal.
- We want to design a secure system that can prevent attacker from achieving the weakest goal.



# Threat Models: Attacker's capability

Here are some types of information we assume an attacker has access to.

- **Ciphertext only attack:**

The attacker is given a collection of ciphertext  $c$ . The attacker may know some properties of the plaintext, for e.g. the plaintext is an English sentence. (attacker can't choose the plaintext).

- **Known plaintext attack:**

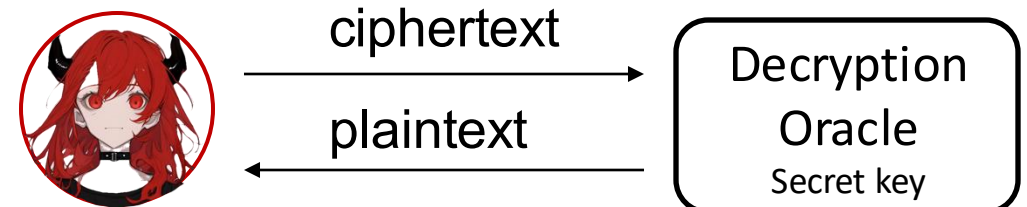
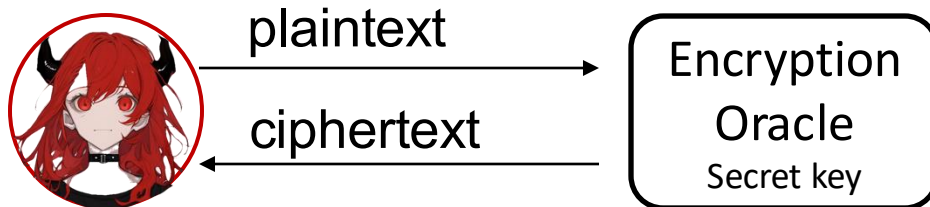
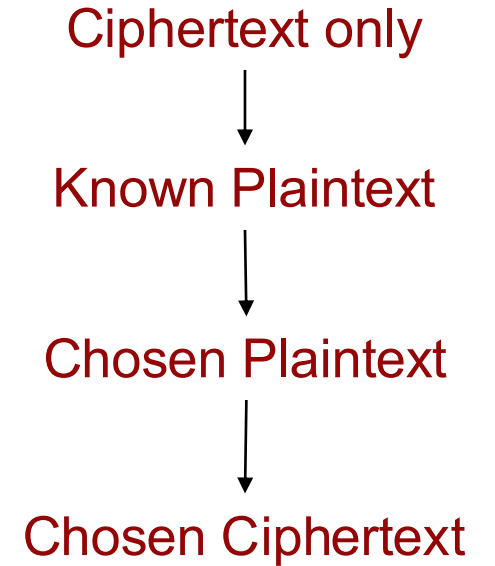
The attacker is given a collection of plaintext  $m$  and their corresponding ciphertext  $c$ . (the attacker can't choose the plaintext.)

- **Chosen plaintext attack (CPA):**

The attacker has access to an **oracle**. The attacker can choose and feed any plaintext  $m$  to the oracle and obtain the corresponding ciphertext  $c$  (all encrypted with the same key). The attacker can access the oracle many times, as long as within the attacker's compute power. He can see the ciphertext and then choose the next input. We call this black-box an **encryption oracle**.

- **Chosen ciphertext attack (CCA2):**

Same as chosen plaintext attack, but here, the attacker chooses the ciphertext and the black-box outputs the plaintext. We call the black-box a **decryption oracle**.

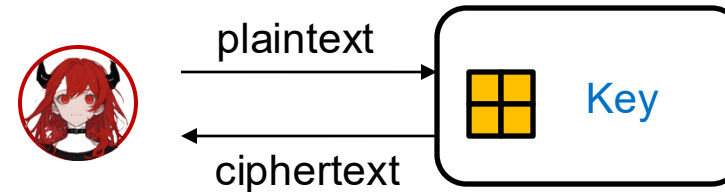


## Is it practical to assume the attacker has known plaintext?

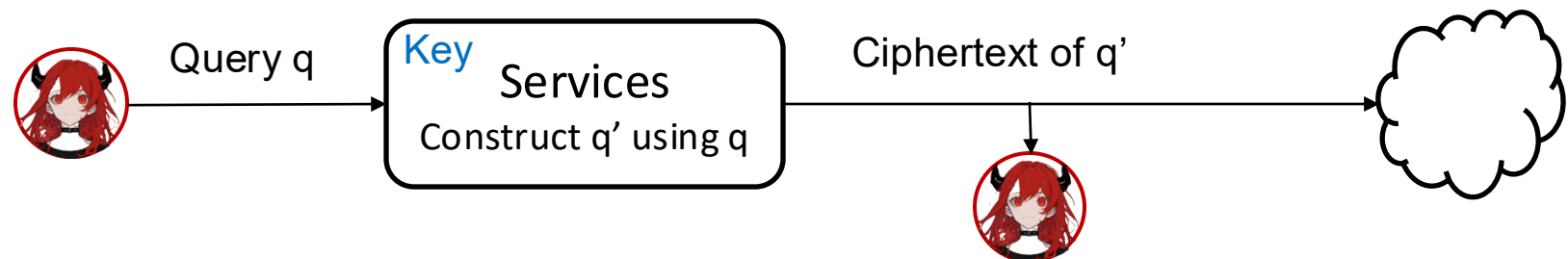
- e.g. Attacker might know the header or part of the plaintext. (e.g. many networks packet header are fixed)

## Is it practical to assume the attacker has access to encryption oracle?

- Eg. Attacker has access to a smartcard. Attacker can query the smartcard to get the ciphertext.

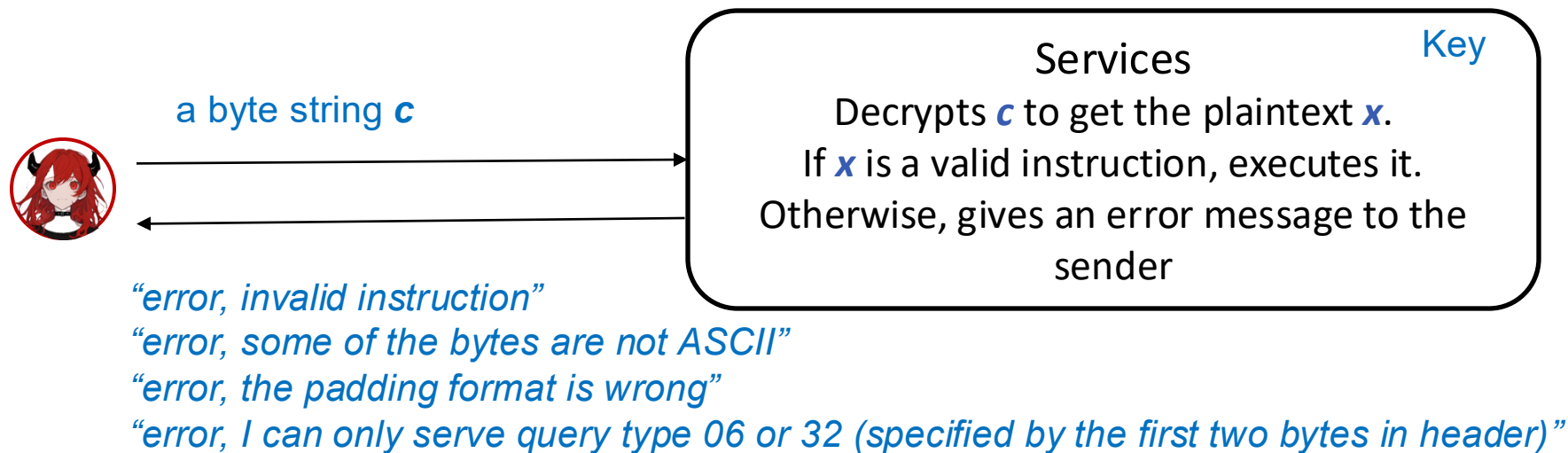


- Eg. Attacker know that after a query  $q$  is sent to a server (e.g. DNS query), the server would construct a query  $q'$  based on  $q$ , encrypt it with a secret key, and then send the ciphertext to another server. Attacker can eavesdrop and obtain the ciphertext.



# What about decryption oracle ?!?!

- Very strange. Isn't it already “GG” if the attacker has a decryption oracle?
- There are practical scenarios where the attacker has access to a weaker form of decryption oracle. We are getting into detail of an example: *Padding Oracle*.
- There could be many different weaker forms, potentially some that the defender is not aware of. But they are all weaker forms of decryption oracle. If a cipher can defend against decryption oracle, then the cipher can defend against all other weaker forms.



- From defender's point of view, we want a cipher that can protect against the attacker with the highest capability.
- It turns out that that if a cipher is secure against CCA2, then it is also secure against CPA.
- Unfortunately (or strangely), many systems employ cipher that is only secure against CPA but not CCA2.

# 1.3 Classical Ciphers

For illustration, we will investigate a few classical ciphers. Classical ciphers are not secure in the computer era. (exception: the “unbreakable” one-time-pad).

(fun to see <http://ciphermachines.com/index>  
for a good listing of classical ciphers and cipher machines used during WWII.)

- 1.3.1. Substitution Cipher
- 1.3.2. Permutation Cipher
- 1.3.3. One time Pad

## **1.3.1 Substitution Cipher**



# Substitution Cipher

- **Plaintext** and **ciphertext**: a string over a set of symbols  $U$ .

E.g.

Let  $U = \{“a”, “b”, “c”, \dots, “z”, “_”\}$ .

e.g. of plaintext: “hello\_world”

- **Key**: a substitution table  $S$ , representing an 1-1 onto function from  $U$  to  $U$ .

E.g.

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	_
g	v	w	b	n	e	f	h	d	a	t	l	u	c	q	m	z	i	r	s	j	x	o	y	k	_	p

$S(“a”) = “g”, S(“b”) = “v”, \dots$

The inverse of  $S$

$S^{-1}(“g”) = “a”, S^{-1}(“v”) = “b”$

- The **key space** is the set of all possible keys. The **key space size** or **size of key space** is the total number of possible keys. The **key size** or **key length** is the number of bits required to represent a key. Here, the key space size is  $(27!)$ , while key size is approximately 94 bits. (obtained by  $\log_2 (27!)$ )

# Substitution cipher: encryption/decryption

**Encryption:** Given a plaintext, which is a string

$$X = x_1 x_2 x_3 \dots x_n$$

and the key  $S$ , outputs the ciphertext

$$E_S(X) = S(x_1) S(x_2) S(x_3) \dots S(x_n)$$

E.g.

plaintext:    h e l l o \_ w o r l d

ciphertext:   h n l l q p o q i l b

**Decryption:** Given a string of ciphertext of length  $n$

$$C = c_1 c_2 c_3 \dots c_n$$

and the key  $S$ , outputs the plaintext

$$D_S(C) = S^{-1}(c_1) S^{-1}(c_2) S^{-1}(c_3) \dots S^{-1}(c_n)$$

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	_
g	v	w	b	n	e	f	h	d	a	t	l	u	c	q	m	z	i	r	s	j	x	o	y	k	_	p

# Exhaustive search (applicable to all encryption schemes)

- The attacker's goal is to find the key or to obtain some information of the plaintext.
- A simple attack is to exhaustively search the keys, i.e. examine all possible keys one by one. Using exhaustive search, eventually (although this might take very long time) the correct key can be found\*.
- So, for a cipher to be secure, exhaustive search must be computationally *infeasible*, e.g. taking millions of years using state-of-the-art supercomputer
- Sophisticated attacks exploit weakness of the encryption scheme so that it can break faster than exhaustive search.

\*: There are "Information theoretic secure" schemes such as one-time-pad that exhaustive search can't succeed.

# Exhaustive search (aka brute-force-search).

- Consider a substitution cipher with table size 27.
- We assume that the attacker knows a ciphertext **C** and the corresponding plaintext **X**. The attacker wants to find the key.

Let **S** be the set of all possible substitution tables. Given **X**, **C**.

1. For each **S** in **S**
  2.     Compute  $X' = D_S(C)$ ;     If  $(X' == X)$  then break;
  3.   end-for
  4.   Display ( "The key is ", **S** );
- The running time depends on the size of the key space **S**.
  - Since a key can be represented by a sequence of 27 symbols. The size of key space is  $27!$   
(This implies that for any representation of the key, the number of bits required is at least  $\log_2(27!) \approx 94$  bits.)
  - Eventually, exhaustive search will find the key. In the worst case, the exhaustive search needs to carry out  $27! \approx 2^{94}$  loops. This is infeasible using current compute power (Tutorial 1).

# Attack: Known-plaintext-attack

- You should have realized that the attacker doesn't need to carry out exhaustive search. Given a plaintext and ciphertext (i.e. "Known plaintext attack"), e.g

plaintext:            h e l l o \_ w o r l d  
ciphertext:          h n l l q p o q i l b

The attacker can figure out the entries in the key

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	_
			b	n			h				l			q			i					o				p

For sufficiently long ciphertext, the full table can be found.

- So, substitution cipher is ***not secure under known plaintext attack***.

# Attack: Ciphertext only attack

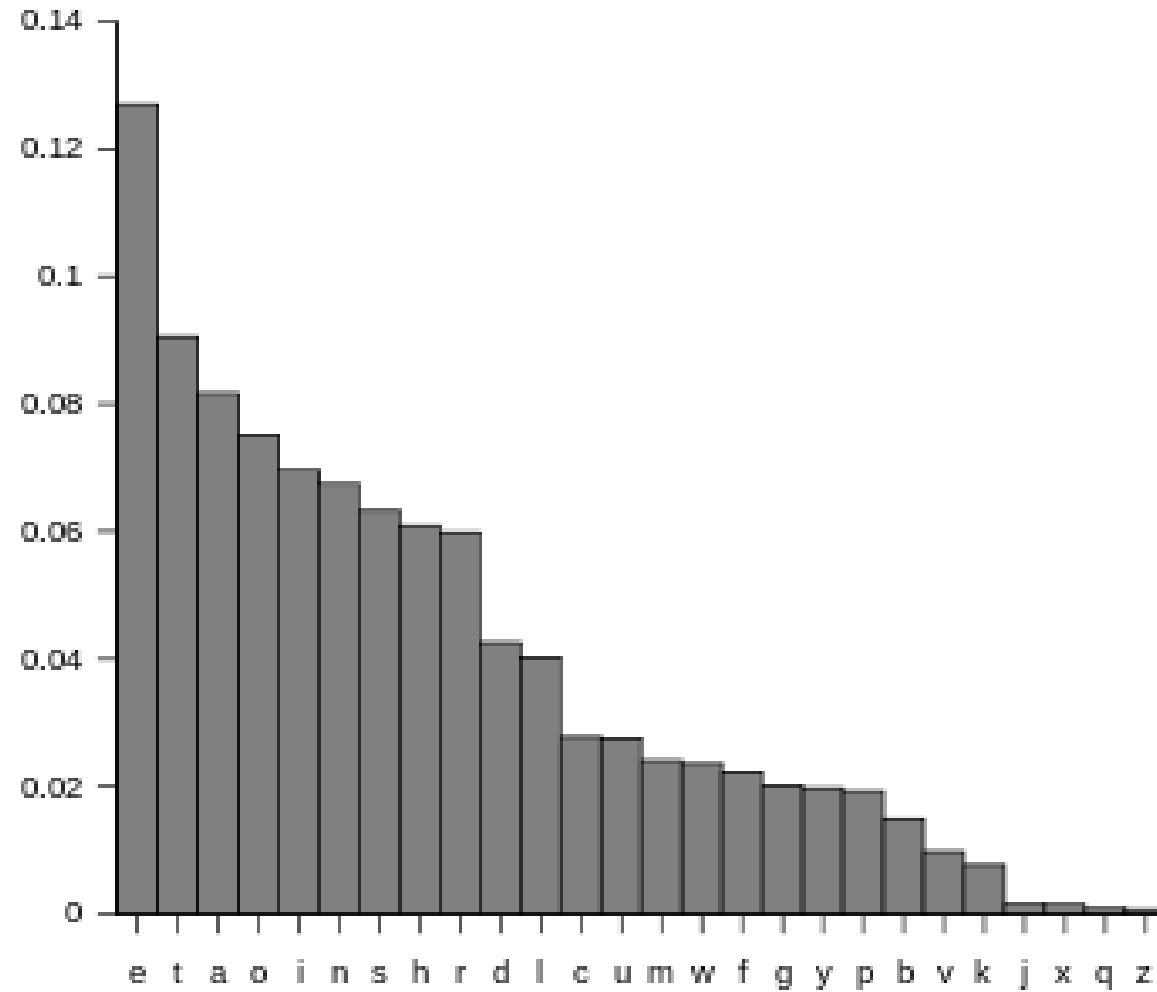
- **Ciphertext only attack:** The attackers have access to ciphertext only (i.e. without the corresponding plaintext).
- Suppose an attacker knows that the plaintext is an English sentence, can he find the key using exhaustive search under ciphertext only attack ? **Yes.**

Let  $\mathcal{S}$  be the set of all possible substitution table. Given  $C$ .

1. For each  $S$  in  $\mathcal{S}$
  2.     Compute  $X = D_S(C)$ ; if  $X$  contains words in the English dictionary, then break;
  3.   end-for
  4.   Display ( "The key is ",  $S$  );
- Likewise, eventually, the exhaustive search will find the key. However, this attack is computationally infeasible.
  - Are there efficient ciphertext only attacks on substitution cipher? **Yes.**

- Substitution cipher is vulnerable to ***frequency analysis*** attack.
- Note that in the `hello_world` example, “o” appears 2 times in the plaintext, whereas the corresponding “q” also appears 2 times in the ciphertext.
- Suppose the plaintexts are English sentences. The frequency of letters used in English is not uniform, for e.g. “e” is more commonly used than “z”. Given a sufficiently long ciphertext (say, around 50 characters), attacker may correctly guess the plaintext by mapping frequent characters in the ciphertext to the frequent character in English. This simple mapping is quite effective.
- Hence, substitution cipher is ***not secure under ciphertext only attack***, when the plaintexts are English sentences.

Frequency of letters in English text.



from [http://en.wikipedia.org/wiki/Letter\\_frequency](http://en.wikipedia.org/wiki/Letter_frequency)



## **1.3.2 Permutation cipher**

# Permutation Cipher

- Also known as transposition cipher.

The encryption first groups the plaintext into blocks of  $t$  characters, and then applied a secret “permutation” to each block by shuffling the characters.

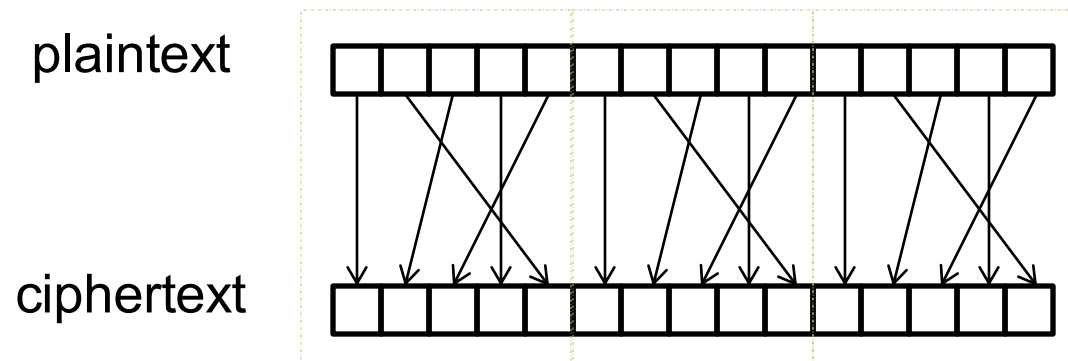
The key is the secret “permutation”, which is an 1-1 onto function  $e$  from  $\{1,2,..,t\}$  to  $\{1,2,...,t\}$ . The size  $t$  could be part of the key, that is,  $t$  is also kept secret. We can write the permutation  $\mathbf{p}$  as a sequence

$$\mathbf{p} = (p_1, p_2, p_3, \dots p_t)$$

which shift the character at position  $i$  to the position  $p_i$ .

Example:

Given the plaintext and the key  $t=5$ ,  $\mathbf{p}=(1,5,2,4,3)$



# Attack

- Permutation cipher fails miserably under known-plaintext attack.

Given a plaintext and a ciphertext, it is very easy to determine the secret key.

m=   a   a   b   b   b   b   a   b   a   b   a   a

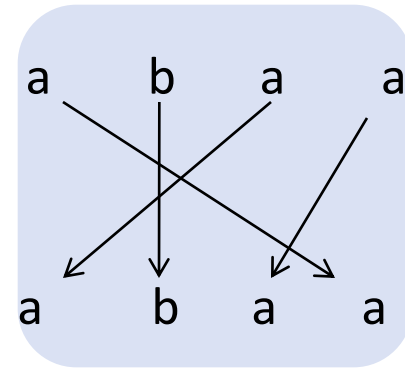
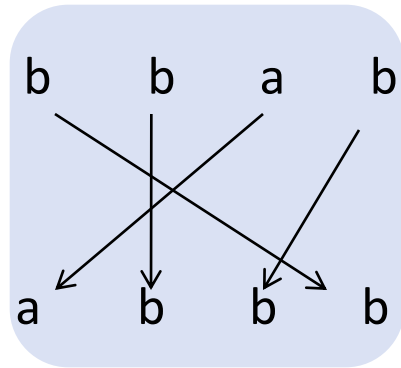
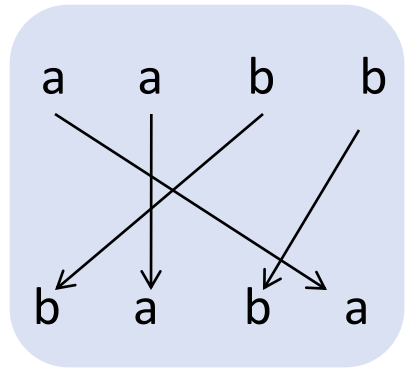
c=   b   a   b   a   a   b   b   b   a   b   a   a

in the above, what is the block size  $t$ ? What is the permutation?

- Permutation cipher is also easily broken under ciphertext only attack if the plaintext is English text.

m=

c=



# Substitution and Permutation cipher

- S and P cipher are not secure.
- Performing substitution twice using two tables does not increase difficulty of attack. It simply reduces to one table (try to see why...). Same for permutation.
- However, by interlacing them, attacks become more difficult. Indeed, many modern encryption scheme (e.g. AES) is designed using rounds of S and P.

## 1.3.3 One Time Pad

XOR operation

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

Some interesting properties:

- Commutative:  $A \oplus B = B \oplus A$
- Associative:  $A \oplus (B \oplus C) = (A \oplus B) \oplus C$
- Identity element:  $A \oplus 0 = A$
- Self-inverse:  $A \oplus A = 0$

$$A \oplus B = (A+B) \bmod 2$$

# One-time-pad

## Encryption:

Given n-bit Plaintext:  $x_1, x_2, \dots, x_n$  and n-bit key:  $k_1, k_2, \dots, k_n$

ciphertext  $C = (x_1 \oplus k_1), (x_2 \oplus k_2), (x_3 \oplus k_3), \dots, (x_n \oplus k_n)$

## Decryption:

Given n-bit ciphertext:  $c_1, c_2, \dots, c_n$  and n-bit key:  $k_1, k_2, \dots, k_n$

plaintext  $X = (c_1 \oplus k_1), (c_2 \oplus k_2), (c_3 \oplus k_3), \dots, (c_n \oplus k_n)$

- The key cannot be re-used. That is, a key can only be used once.  
(Recap that for substitution and permutation cipher, a same key is being used to encrypt multiple plaintexts.)

Due to the above requirement, a 1GB plaintext would need a 1GB key to encrypt.

## E.g. One-time-pad

Encryption: plaintext  $\oplus$  key  $\rightarrow$  ciphertext

Decryption: ciphertext  $\oplus$  key  $\rightarrow$  plaintext

decryption	PlainText	0	0	1	0	1	1	0	encryption
	<b>Key</b>	<b>1</b>	<b>1</b>	<b>0</b>	<b>0</b>	<b>1</b>	<b>1</b>	<b>1</b>	
	Ciphertext	1	1	1	0	0	0	1	

### Correctness (decrypting the ciphertext give back the plaintext):

For any x, k,       $(x \oplus k) \oplus k = x \oplus (k \oplus k) = (x \oplus 0) = x$

ciphertext                                  plaintext



# Security of one-time-pad

- From a pair of ciphertext and plaintext, the attacker can derive the key. However, such key is useless, since it will not be used any more.
- Note that even exhaustive search can't work on one-time-pad. (Suppose we are given a 1Kbytes ciphertext and are told that the plaintext is a jpeg image. By using exhaustive search, can we eventually find the plaintext?)
- In fact, It can be shown that one-time-pad leaks no information of the plaintext, even if the attacker has arbitrary running time. Hence, it is sometime called “unbreakable”.
- CS4236 would study the formulation of “Perfect Secrecy” of one-time-pad.

- The long key renders one-time-pad impractical.
- Nevertheless, it is still relevant in some scenarios.

see <http://ciphermachines.com/otp>

## and the Venona Story (where one-time-pads fails)

(optional:

[https://www.nsa.gov/Portals/70/documents/about/cryptologic-heritage/historical-figures-publications/publications/coldwar/venona\\_story.pdf](https://www.nsa.gov/Portals/70/documents/about/cryptologic-heritage/historical-figures-publications/publications/coldwar/venona_story.pdf) )

<http://ciphermachines.com/otp>



# Intuitively, Perfect Secrecy means:

Optional

Attacker's prior knowledge of the unknown plaintext  $x$ . (before knowing  $y$ )

**Definition:** A cryptosystem has *perfect secrecy* if  
for any distribution  $X$ , for all  $x, y$   
$$\Pr ( X=x \mid Y=y ) = \Pr ( X =x ).$$

Attacker's updated knowledge of the unknown plaintext, after the attacker had seen the ciphertext  $y$ .

for any ciphertext  $y$  and plaintext  $x$ , the chances that an attacker correctly predicts  $x$  before knowing  $y$ , and after knowing  $y$ , are the same.

# 1.4 Modern Ciphers

Modern ciphers generally refer to schemes that use computer to encrypt/decrypt.

E.g. RC4, DES, A5, AES, RSA

Many modern symmetric ciphers employed rounds of substitution (the so called “S-box”) and permutation.

## **1.4.1 DES/Exhaustive Search**

# Modern ciphers

Designs of modern ciphers take into considerations of known-plaintext-attack, frequency analysis and other known attacks. (Usually, they do not consider CCA2. Fortunately, there are method to convert them to be secure under CCA2)

E.g.	DES (Data Encryption Standard, 1977)	<i>broken due to short key</i>
	RC4 (Rivest's Cipher 4, 1987)	<i>broken</i>
	A5/1 (used in GSM, 1987)	<i>broken</i>
	ZipCrypto (used in Zip, 1993?)	<i>broken</i>
	...	
	AES (Advanced Encryption Standard, 2001)	widely analyzed. Believe to be secure

They are “supposed” to be secure so that any successful attack does not perform noticeably better than exhaustive search.

# Exhaustive search and key length

If the key length is 56 bits, there are  $2^{56}$  possible keys. Hence, the exhaustive search needs to “loop” for  $2^{56}$  times in the worst case.

We can quantify the security of an encryption scheme by the length of the key. Consider a scheme **A** with 64-bit keys and a scheme **B** with 54-bit keys. Scheme **A** is more secure w.r.t. exhaustive search. (note that some schemes, e.g. RSA, have known attacks that are more efficient than exhaustively searching all the keys. In those cases, we still want to quantify the security by the equivalent of exhaustive search. For e.g, in the best-known attack on a 2048-bit RSA, roughly  $2^{112}$  searches are required. So, its security is equivalent to 112 bits, and we say that the “2048-bit RSA has key strength of 112 bits”).

How many bits is considered “secure”? (Tutorial 1)

**read** NIST Recommended key length for AES <http://www.keylength.com/en/4/>

Date	Security Strength	Symmetric Algorithms	Factoring Modulus	Discrete Logarithm Key Group		Elliptic Curve	Hash (A)	Hash (B)
Legacy <sup>(1)</sup>	80	2TDEA	1024	160	1024	160	SHA-1 <sup>(2)</sup>	
2019 - 2030	112	(3TDEA) <sup>(3)</sup> AES-128	2048	224	2048	224	SHA-224 SHA-512/224 SHA3-224	
2019 - 2030 & beyond	128	AES-128	3072	256	3072	256	SHA-256 SHA-512/256 SHA3-256	SHA-1 KMAC128
2019 - 2030 & beyond	192	AES-192	7680	384	7680	384	SHA-384 SHA3-384	SHA-224 SHA-512/224 SHA3-224
2019 - 2030 & beyond	256	AES-256	15360	512	15360	512	SHA-512 SHA3-512	SHA-256 SHA-512/256 SHA-384 SHA-512 SHA3-256 SHA3-384 SHA3-512 KMAC256

<http://www.keylength.com/en/4/>

Remark: The possibility of having Quantum Computers complicates requirement of key-length. We will study this in case-studies during Tutorial.

# Exhaustive Search on DES

Key length of DES is 56 bits.

While exhaustive search on 56 bits seemed infeasible in the 70s, very soon, it is possible using distributed computing or specialized chip.

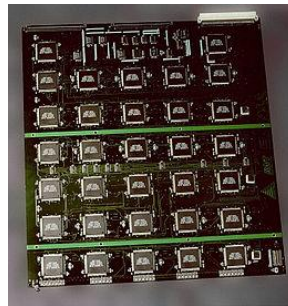
RSA Security hosted a few challenges on DES. (Note: **RSA** is an encryption scheme, **RSA Security** is a company, **RSA Conference** is a well-known conference organized by the company)

**DES Challenge II-1:** The secret message is: "Many hands make light work."

(found in 39 days using distributed computing, early 1998)

**DES Challenge II-2:** The secret message is: "It's time for those 128-, 192-, and 256-bit keys."

(found in 56 hours using specialized hardware, 1998)



EFF's DES cracking machine.

A puzzling question: Why would a standard chose a scheme that can be broken? Believe to be intentional.



# AES

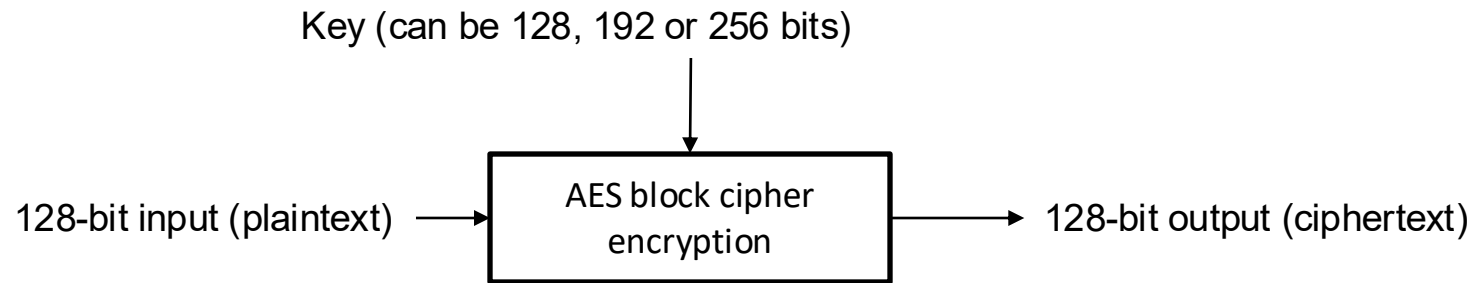
# AES

- In 2000, a new standard for block cipher, AES (Advance Encryption Standard), was proposed by NIST. The selection process was transparent with worldwide involvement.
- NIST called for proposal in 1997 and received 21 submissions by Jun 1998. Many rounds of cryptanalysis on the submissions. In 2000, *Rijndael* was selected as AES.
- Rijndael was invented by Belgian researchers Daemen and Rijmen.
- AES block length is 128, and key length can be 128, 192 or 256 bits.
- Currently, no known attacks on AES. (there are some attacks on the mode-of-operation)
- NSA classifies AES as “Suite B Cryptography”.  
*“**NSA Suite B Cryptography** is a set of cryptographic algorithms promulgated by the National Security Agency as part of its Cryptographic Modernization Program. It is to serve as an interoperable cryptographic base for both unclassified information and most classified information.”*  
*see [https://en.wikipedia.org/wiki/NSA\\_Suite\\_B\\_Cryptography](https://en.wikipedia.org/wiki/NSA_Suite_B_Cryptography)*

## **1.4.2 Block cipher & Mode-of-Operations**

# Block Cipher

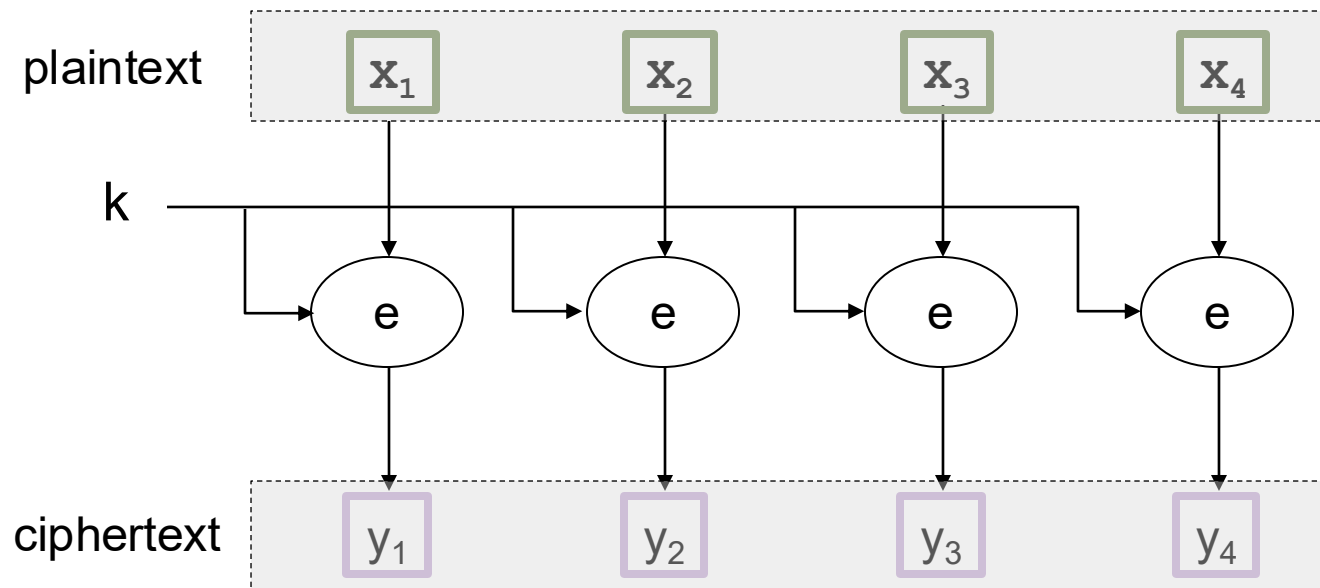
- DES and AES are also known as “Block Cipher”. A Block cipher has fixed size of input/output. E.g. AES is designed for 128 bits (16 bytes) input/output.



- *What?! Only 128 bits! How to encrypt the movie!!!* 🤪
- For large plaintext (say 10 MB), it is first divided into blocks, and the block cipher is then applied. The method of extending encryption from a single block to multiple blocks is not straightforward. It is called *mode-of-operation*.

# Mode-of-operation: ECB mode

- Electronic Code Book naturally is the first to come into our mind.
- It divides plaintext into blocks and then applies block cipher to each block, all with the same key.



- ECB leaks information!!

# ECB

In the following example, the image is divided into blocks and encrypted with the deterministic block cipher using the same key. Since it is deterministic, any two blocks that are the same (for e.g. blocks in the white background) will be encrypted to the same ciphertext.



Plaintext



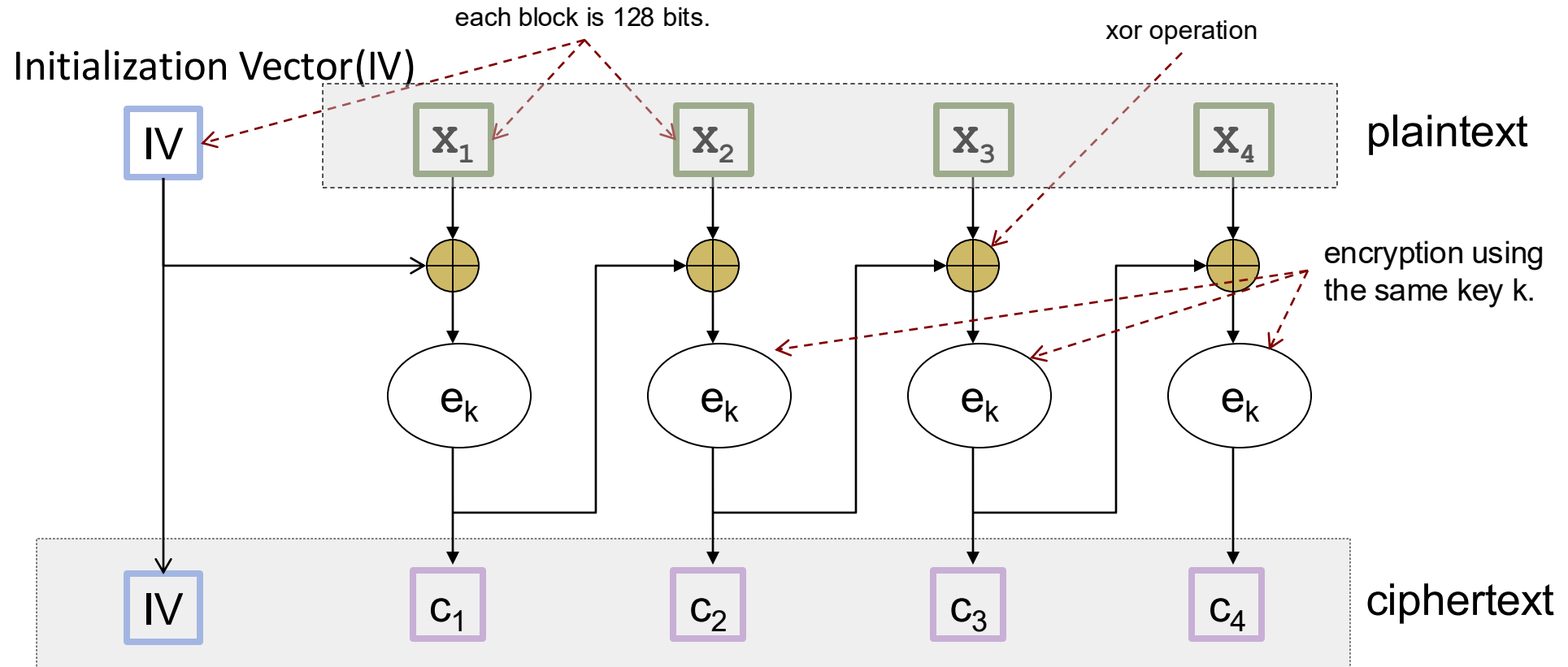
ciphertext

## Remark.

An encryption scheme is “deterministic” in the sense that, the encryption algorithm will always produce the same output (i.e. the ciphertext) when given the same input (i.e. the key and plaintext). In contrast, a “probabilistic” encryption scheme produces different ciphertext even with the same input (key, plaintext).

AES is deterministic. However, if we employ AES with a randomly chosen IV (to be introduced later), then it is probabilistic.

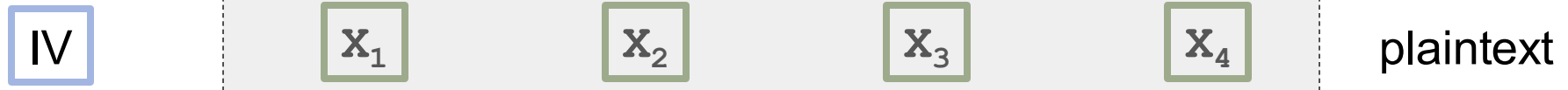
# Mode-of-operation: Cipher Block Chaining (CBC) on AES



- The Initialization Vector (IV) is an arbitrary value chosen during encryption. It must be different in different encryptions. Depending on implementation, IV can be randomly chosen, obtained from a counter, or extracted from other info.

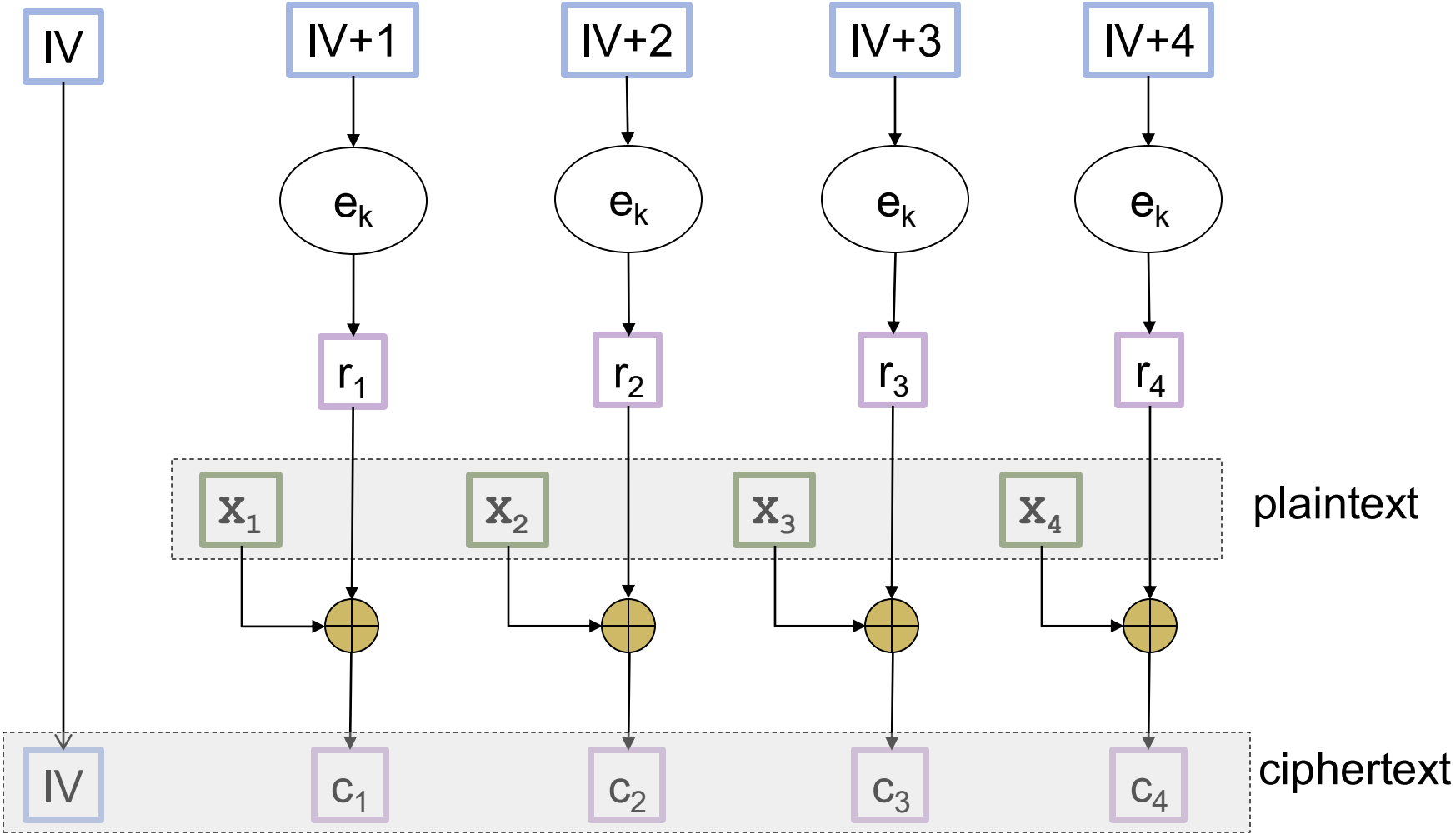
$$y_0 = \text{IV}. \quad c_i = E_k(x_i \oplus c_{i-1}) \quad \text{for } i > 0$$

# Cipher Block Chaining (CBC) decryption



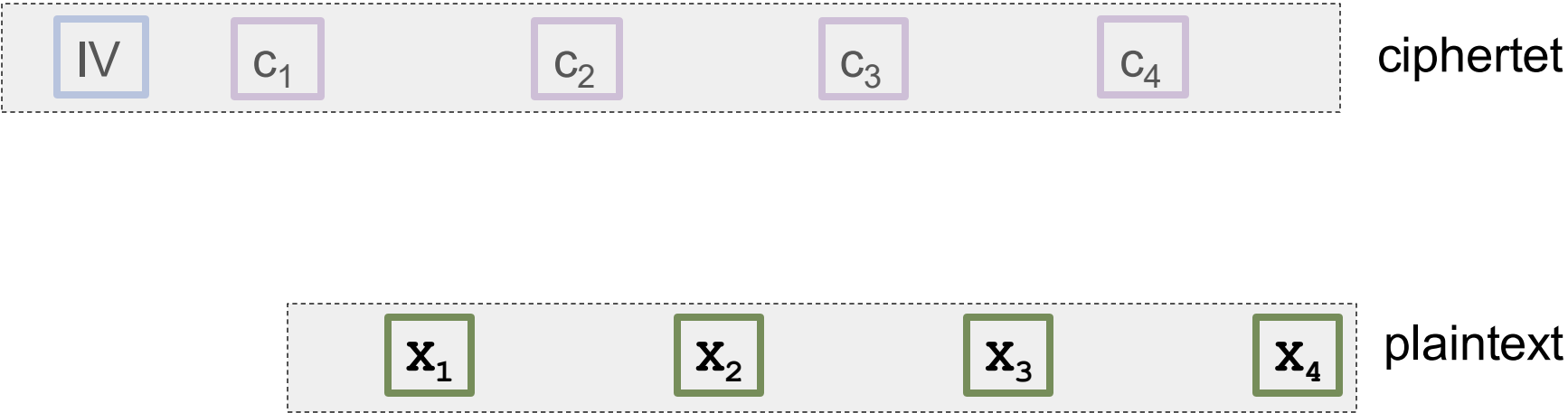


# mode-of-operation: Counter mode (CTR) Encryption



# Counter mode (CTR) decryption

Intentionally left blank



# Programming example

This key can be randomly chosen or set by user (see crypto pitfalls). Of course, if the key is randomly chosen, it must be securely sent to the receiver and/or store in a secure place.

- Python.

(package PyCryptodome <https://pycryptodome.readthedocs.io/en/latest/src/cipher/aes.html>)

(another package PyCrypto)

The IV should be randomly chosen.

```
>>> from Crypto.Cipher import AES
>>>
>>> key = b'Sixteen-byte key'
>>> iv = b'Sixteen-byte IV'
>>> cipher = AES.new(key, AES.MODE_CBC, iv)
>>> c=iv+cipher.encrypt(b'Plaintext of length with multiple of 16 bytes')
```

In Python, to display a byte sequence, we can use...

```
>>> from base64 import *
>>> b16encode(c)
b'5369787465656E206279746520204956B186083256CACCBD1638AF4877FBF2AAFBECEB66FE13C403D7CE8EA04D028E66CA6AE1294FF51C2F363CCC8953137A6A3'
```

## GCM mode (Galois/Counter)

- Construction of this mode is more complicated. Details omitted in this class.
- It is an “*Authenticated-Encryption*”. The ciphertext consists of an extra tag for authentication (AE to be introduced later).
- It is secure in the presence of decryption oracle.

# Mode of operations mentioned in this lecture.

- ECB
- CBC
- CTR
- GCM

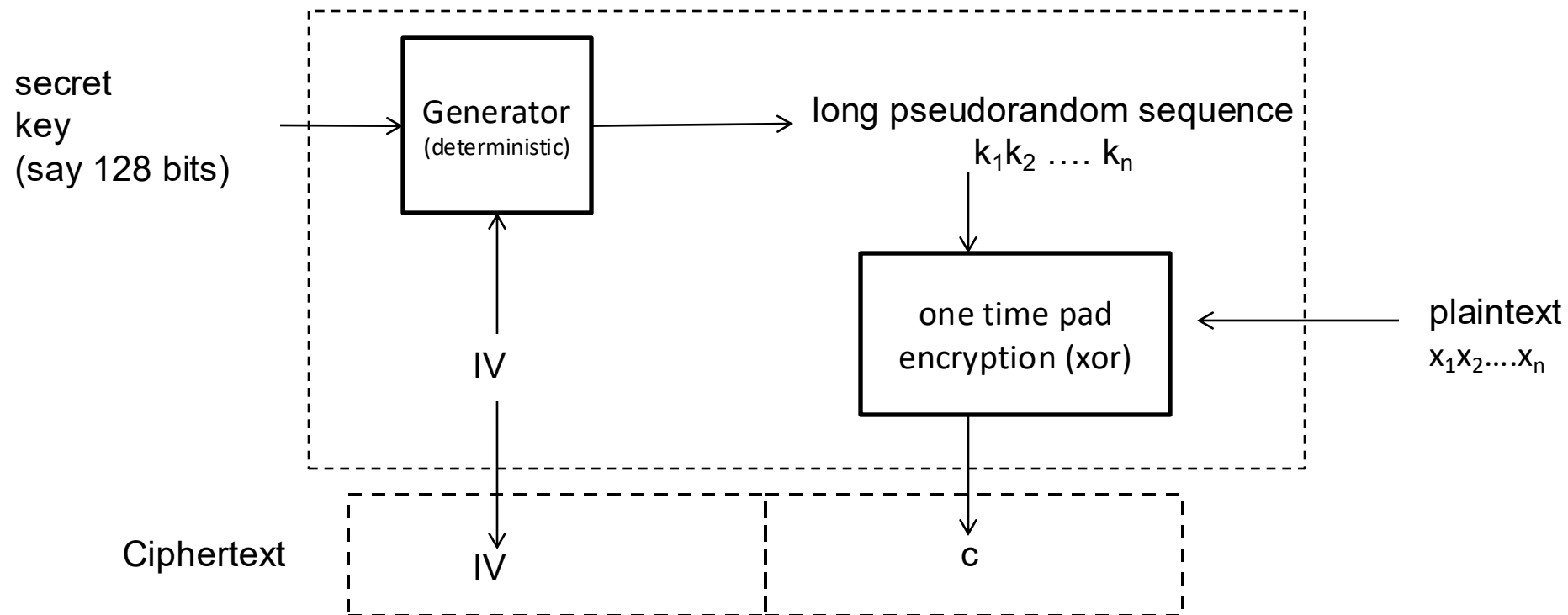
## 1.4.3 Stream Cipher and IVs

CTR mode is a “Stream Cipher”.

# Stream Cipher and one-time-pad

A stream cipher has an Initialization Vector(IV). The IV can be randomly chosen, or from a counter.

- The pseudorandom sequence is generated from the secret key together with the IV. The final ciphertext contains the IV, followed by the output of the one-time-pad encryption.
- For decryption, the IV is extracted from the ciphertext. From the IV and the key, the same pseudorandom sequence can be generated and thus obtain the plaintext.



e.g.

**Encryption:** Given

15-bit Plaintext  $\mathbf{x}$  = 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0  
secret key  $\mathbf{k}$  = 0 1 0 1 ..... 1 1 1 0

Step 1: Randomly generates the initial value  $\mathbf{v}$ ,  
 $\mathbf{v}$  = 0 0 0 1

Step 2: From the secret key  $\mathbf{k}$  and IV, generates a 20-bit sequence  
 $\mathbf{r}$  = 0 1 1 0 1 0 1 0 0 1 0 0 1 1 0

Step 3: outputs  $\mathbf{v}$ , follow by  $\mathbf{r}$  xor  $\mathbf{x}$

$\mathbf{c}$  = 0 0 0 1 0 1 1 0 1 1 0 1 1 0 0 0 1 1 0

**Decryption:** Given the key  $\mathbf{k}$  and the ciphertext  $\mathbf{c}$

Step 1: Extracts  $\mathbf{v}$  from  $\mathbf{c}$

Step 2: From  $\mathbf{k}$  and  $\mathbf{v}$ , generates the long sequence  $\mathbf{r}$ .

Step 3: Performs xor to get the plaintext.



# Why IV? What if the IV is always the same?

Suppose there isn't an IV (or the IV is always set to be a string of 0's)

Consider the situation where the same key is used to encrypt two different plaintexts

$$\mathbf{X} = x_1, x_2, x_3, x_4, x_5 \text{ and}$$

$$\mathbf{Y} = y_1, y_2, y_3, y_4, y_5$$

Further suppose that an attacker eavesdropped and obtained the two corresponding ciphertexts  $\mathbf{U}, \mathbf{V}$ .

The attacker can now compute

$$\mathbf{U} \oplus \mathbf{V} = (\mathbf{X} \oplus \mathbf{K}) \oplus (\mathbf{Y} \oplus \mathbf{K})$$

By associative and commutative property of xor

$$\mathbf{U} \oplus \mathbf{V} = (\mathbf{X} \oplus \mathbf{Y}) \oplus (\mathbf{K} \oplus \mathbf{K}) = \mathbf{X} \oplus \mathbf{Y}.$$

So, from  $\mathbf{U}$  and  $\mathbf{V}$ , the attackers can obtain information about  $\mathbf{X} \oplus \mathbf{Y}$ , i.e. the following sequence

$$(x_1 \oplus y_1), (x_2 \oplus y_2), (x_3 \oplus y_3), (x_4 \oplus y_4), (x_5 \oplus y_5)$$

# What so big deal about revealing $X \oplus Y$ ?

Suppose  $X$  is an  $80 \times 120$  image of an animal. Each pixel is either black or white (0 or 1). The image can be represented as a  $(80 \times 120)$ -bit sequence where each bit corresponds to a pixel.

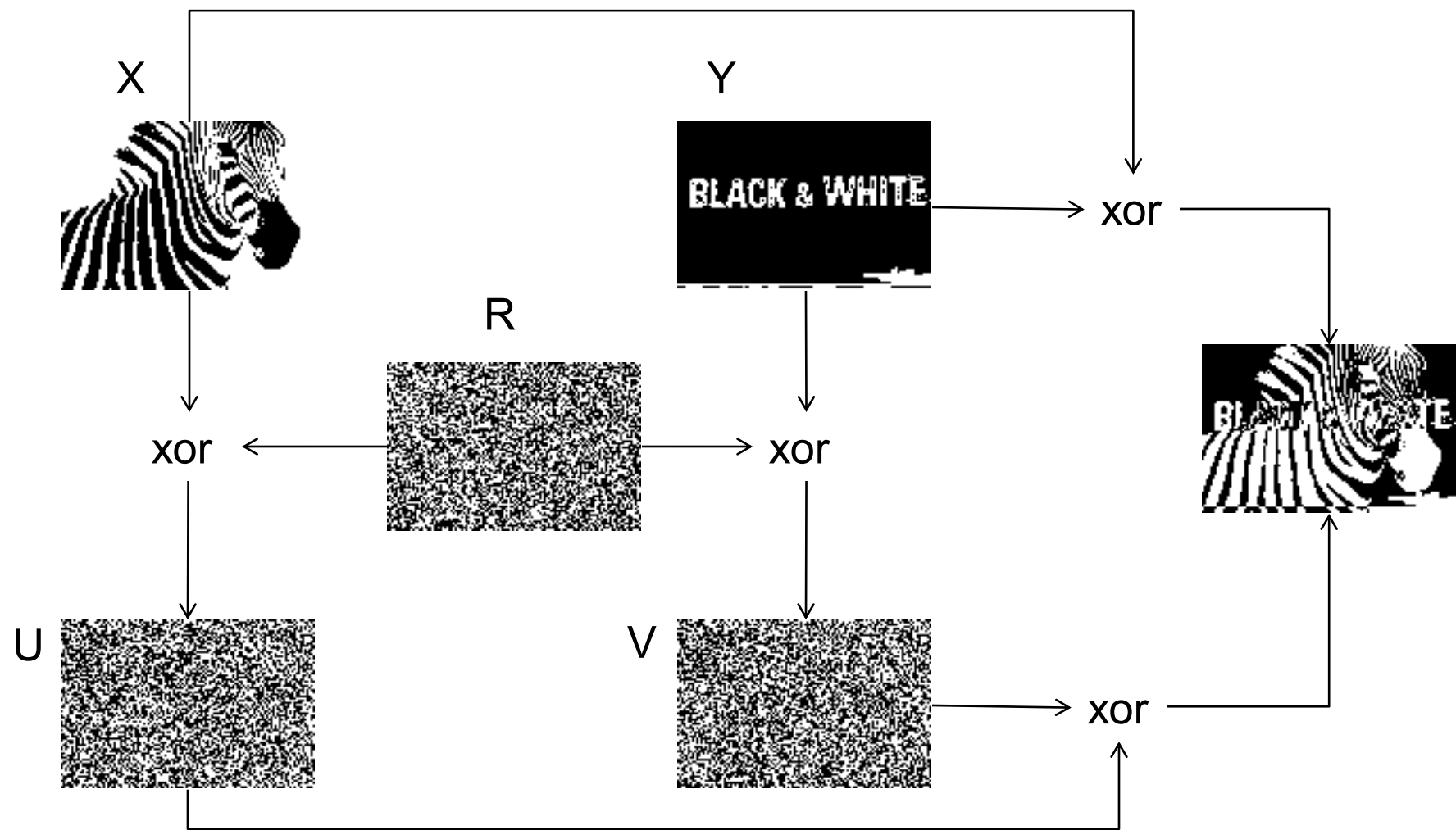
$Y$  is another  $80 \times 120$  pixels image rendering two words, which is similarly represented as a sequence.

Here is  $X \oplus Y$ .

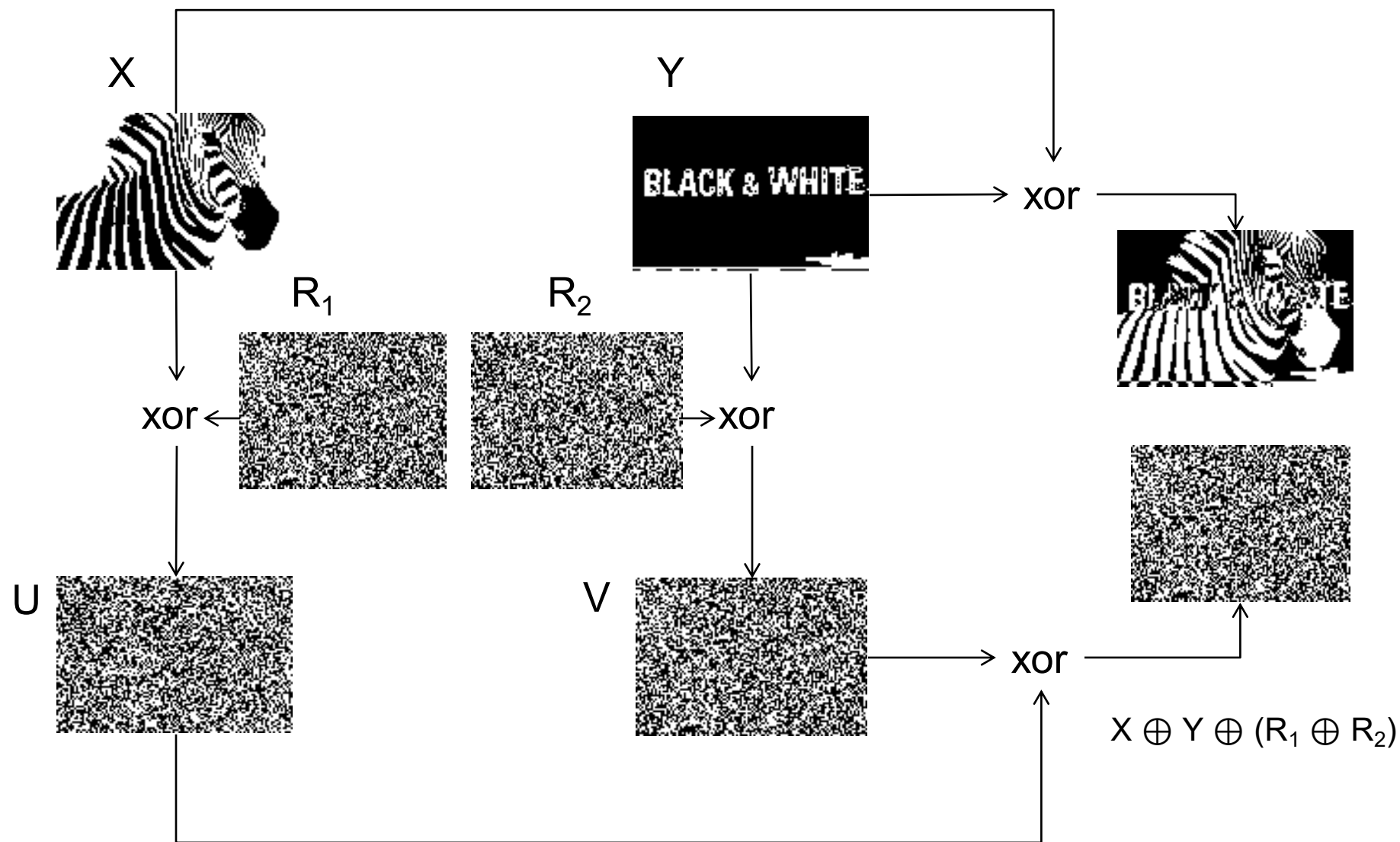


What is  $X$ ,  $Y$ ?

# stream cipher without IV



# stream cipher with IV



# Role of IV

- If the IVs are different in two different instances of encryption, the two pseudorandom sequences will be different.
  - By xor'ing the two ciphertexts would not cancel out the pseudorandom sequences.
  - (Special case when the plaintexts are the same in the two instances): The ciphertexts would be different.
- IV makes an encryption “probabilistic”.
- IV is also needed in CBC mode. The reason is the same. We want the encryption to be non-deterministic, so that two different encryptions of a same plaintext would give two different ciphertexts.

## 1.5. Examples of attacks

1.5.1 Triple DES & Meet-in-the-middle

1.5.2 Padding Oracle Attack

- Notions of Oracle in security analysis
- The attack
- Implications

## 1.5.1 Triple DES & Meet-in-the-middle attack

see [http://en.wikipedia.org/wiki/Meet-in-the-middle\\_attack](http://en.wikipedia.org/wiki/Meet-in-the-middle_attack)

## (d) Triple DES

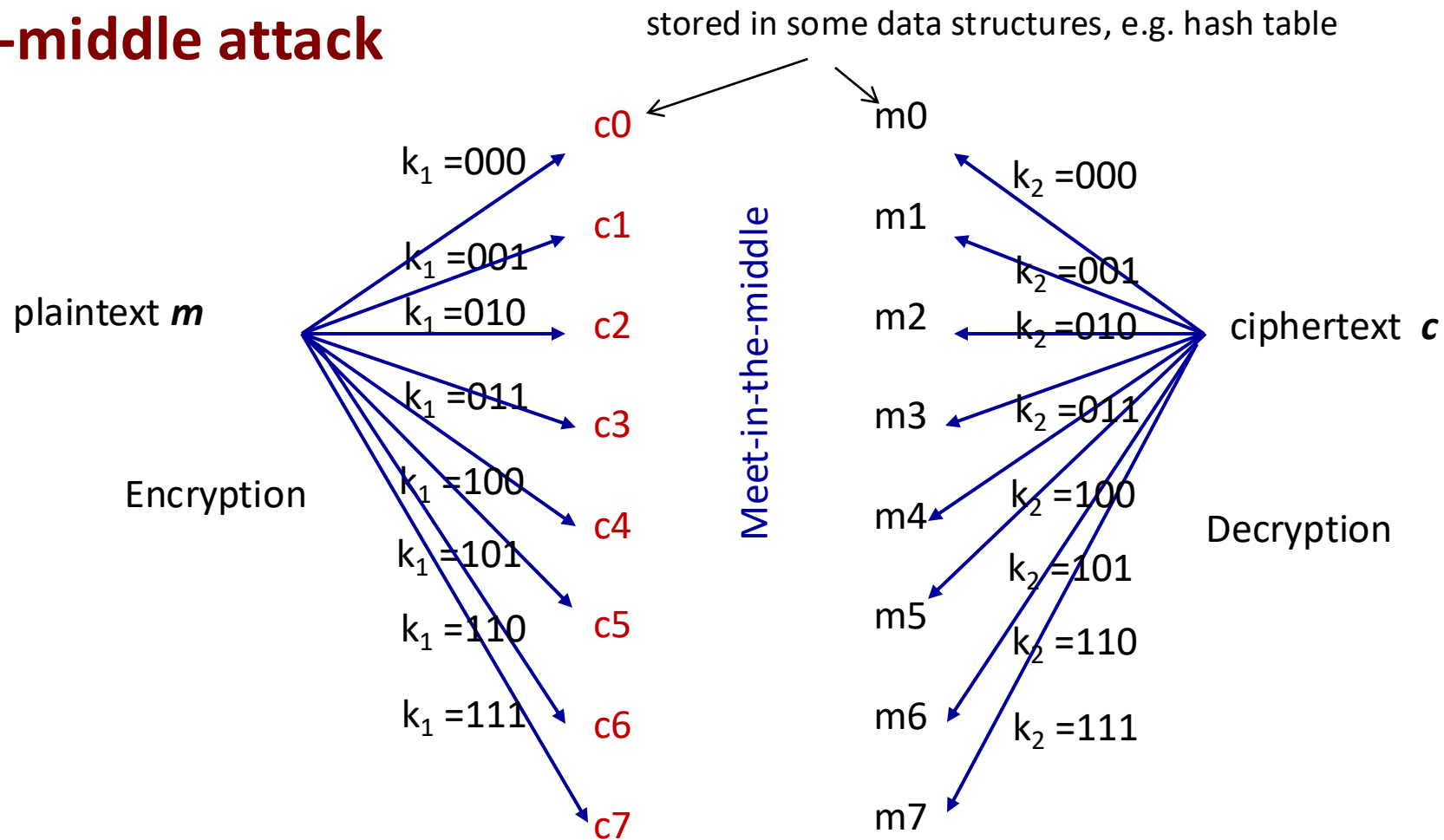
- DES is not secure w.r.t. today computing power. One way to improve it is by multiple encryptions: encrypt the plaintext twice or more, using different keys.
- Let us consider *double* encryption under known plaintext attack. That is, the attacker has a plaintext  $m$  and the corresponding ciphertext  $c$ , and wants to find the two secret keys  $k_1, k_2$ .
- Using exhaustive search, what is the amount of DES encryption/decryption required?  $2^{56+56}$ . So, the key-strength could be 112. Unfortunately, it is much less than that by “meet-in-the-middle” attack.



# Meet-in-the-middle Attack

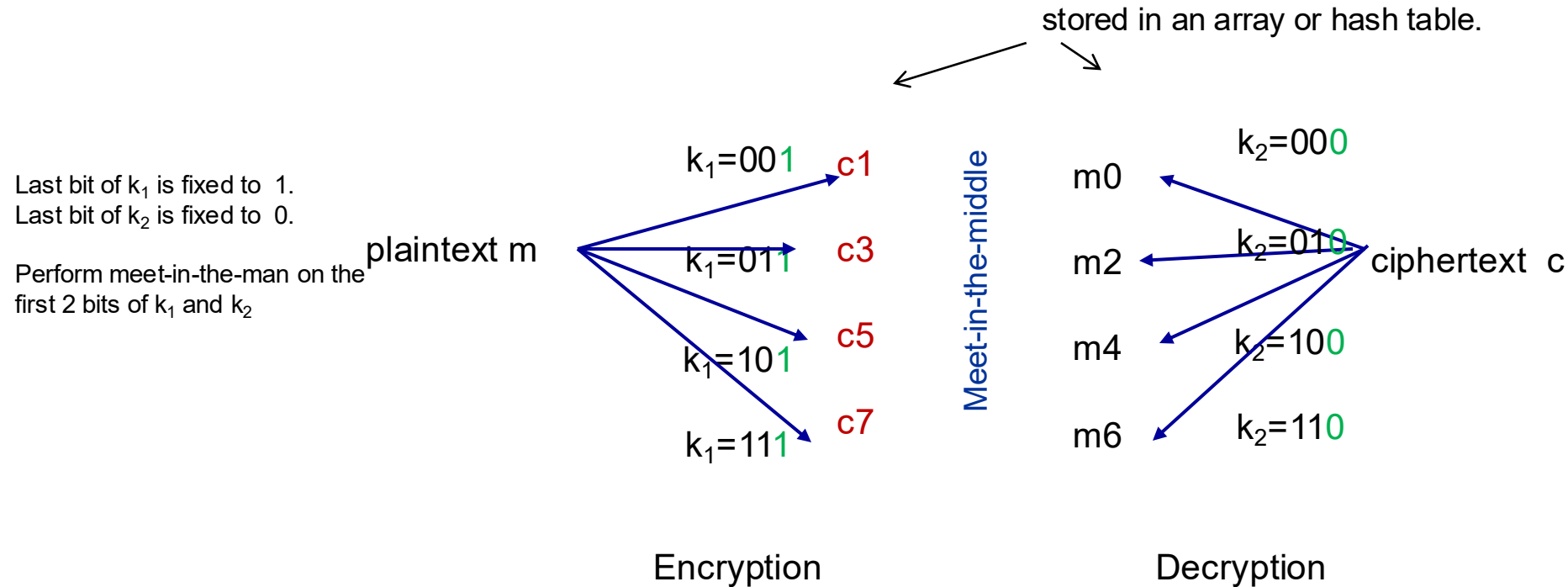
- Not to confuse “meet-in-the-middle” with “man-in-the-middle” attack.
- Introduced by Diffie & Hellman in 1977.
- This is a *known plaintext attack*. We assume attacker has a pair  $(\mathbf{m}, \mathbf{c})$  of plaintext and the corresponding ciphertext. That is, attacker has:  
$$(\mathbf{m}, \quad \mathbf{c} = \text{DES}_{k_1} ( \text{DES}_{k_2} ( \mathbf{m} ) ) )$$
- The attacker’s goal is to find the key, i.e.  $k_1$  and  $k_2$ .

# Meet-in-the-middle attack



- Given  $c$  and  $m$ , find the two keys.
  - Compute two sets  $\mathbf{C}$  and  $\mathbf{M}$ .  $\mathbf{C}$  contains ciphertexts of  $m$  encrypted with all possible keys.  $\mathbf{M}$  contains plaintexts of  $c$  decrypted with all possible keys.
  - Find all common elements (likely to be only one) in  $\mathbf{C}$  and  $\mathbf{M}$ . From the common elements, we can obtain the two keys.
- The above figure, for simplicity, assumes 3-bit keys. Using two keys, there are  $8 \times 8 = 64$  possibilities. However, the attack only perform 8 encryptions and 8 decryptions. In general, for  $k$ -bit keys, it reduces the number of crypto operations to  $2^{k+1}$  using approximately  $2^{k+1}$  units of storage space.

# Tradeoff with time and space



- If  $2^{k+1}$  storage is too much, we can have a tradeoff.
- Given  $m, c$ . For each possible value of the last  $s$  bits of  $k_1$  and last  $s$  bits of  $k_2$ , do the followings:
  - Carries out meet-in-the-middle attack (with last  $s$  bits of  $k_1, k_2$  fixed). If successful, stops the exhaustive search.
- The storage requirement dropped by a factor of  $2^s$ , but the number of cryptographic operations increased by a factor of  $2^{2s}$ .

# 3DES

- Remedy--- Use Triple encryptions, but 2 keys.

a)  $E_{k1} ( E_{k2} ( E_{k1} ( x ) ) )$ .

or

b)  $E_{k1} ( D_{k2} ( E_{k1} ( x ) ) )$ .

Both (a) and (b) are believed to have the same level of security. However, version (b) can be more convenient. By choosing  $K1=K2$ , then it is same as single encryption.

- Other variants 3DES, TDES, TDEA, 2TDES, 3TDES (using 3 keys)

- Meet-in-the-middle take  $\sim 2^{112}$  encryption/decryption. Can it be faster? Interestingly yes. Lucks improved it to  $2^{108}$  DES operations.

[Lucks1998] S. Lucks, *Attacking Triple Encryption*, FSE 1998.

- Triple DES was still in used until recently.
  - VISA seems to support it until Oct 2020  
<https://usa.visa.com/dam/VCOM/global/support-legal/documents/visa-file-exchange-service-key-exchange-key-algorithm.pdf>
  - (3DES was the default encryption in Outlook 2007. See its help page: <http://office.microsoft.com/en-sg/outlook-help/encrypt-messages-HP006369952.aspx>

## 1.5.2 Padding Oracle Attack



<https://images.app.goo.gl/UwpN6vVkBygBEW77A>

# Oracle in security analysis

- Recap that in security analysis, it is important to formulate (1) what information the attackers have; (2) attackers' goals.
- One type of information is obtained via a query-answer system, known as **Oracle**. The attackers can send in queries, and the **Oracle** will output the answer. E.g.
  - Encryption Oracle.** On query a plaintext  $x$ , the oracle outputs the ciphertext  $E_k(x)$  where the key  $k$  is a secret key.
  - Decryption Oracle.** On query a ciphertext  $c$ , the oracle outputs the plaintext  $D_k(c)$  where the key  $k$  is a secret key.
- While encryption oracle make sense, it is not immediately clear why we need to consider attacker with decryption oracle. Padding oracle attack illustrates the need.



# Padding Format

- The block size of AES is 128 bits (or 16 bytes). Suppose the length of the plaintext is 25 bytes, it will be fitted into two blocks, with the remaining 7 bytes “padded” with some values.



- There are many ways to fill in the values. In any case, an important piece of information must be encoded: the number of padded bits. If this info is missing, the receiver will not know the length of the plaintext.
- Next slide describes a padding “standard”.



# Padding - PKCS#7

- PKCS#7 is a padding standard. [https://en.wikipedia.org/wiki/Padding\\_\(cryptography\)#PKCS7](https://en.wikipedia.org/wiki/Padding_(cryptography)#PKCS7)

- The following example is self-explanatory.

*Suppose the block size is 8 bytes, and the last block has 5 bytes (thus 3 extra bytes required), padding is done as follow:*



- In general, the paddings are:

01

02 02

03 03 03

04 04 04 04

....

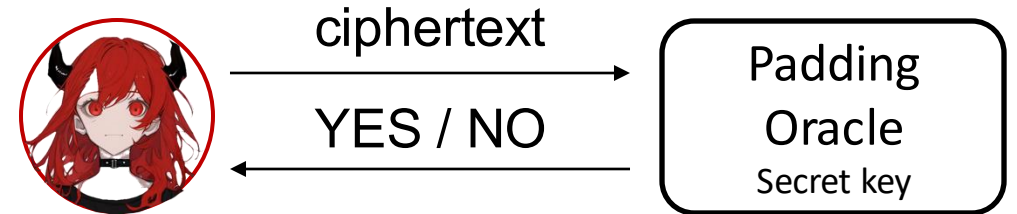
08 08 08 08 08 08 08 08

- If the last block is full, i.e. it has 8 bytes, an extra block is added where each byte is 08 (i.e the block size).

# Padding Oracle attack

The attack model:

- What the attacker have:
  - A ciphertext  $(iv, c)$ . The ciphertext is encrypted using a secret key  $k$ .
  - Access to a Padding Oracle.
- Attacker's goal:
  - The plaintext of  $(iv, c)$
- Padding Oracle:
  - Query: Any ciphertext.
  - Output: YES, if the plaintext is in the correct "padding" format.  
NO, otherwise



# Padding oracle attack on AES CBC mode (AES block is 16 bytes, for ease of presentation, we consider 8-byte block here)

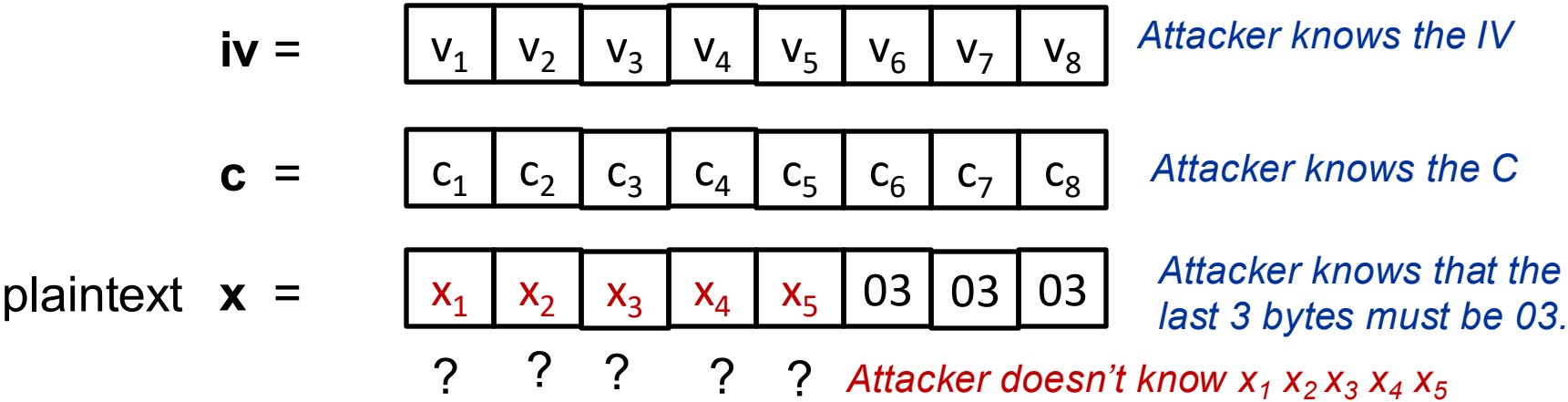
- AES CBC mode is not secure against padding oracle attack.

## Main idea

For ease of illustration, let's assume:

- The block size is 8 bytes;
- The  $c$  is only 1 block;
- The attacker knows that the block is padded with 3 bytes;
- The attacker is only interested in the value of  $x_5$ .

*The attacker knowledge before the attack*

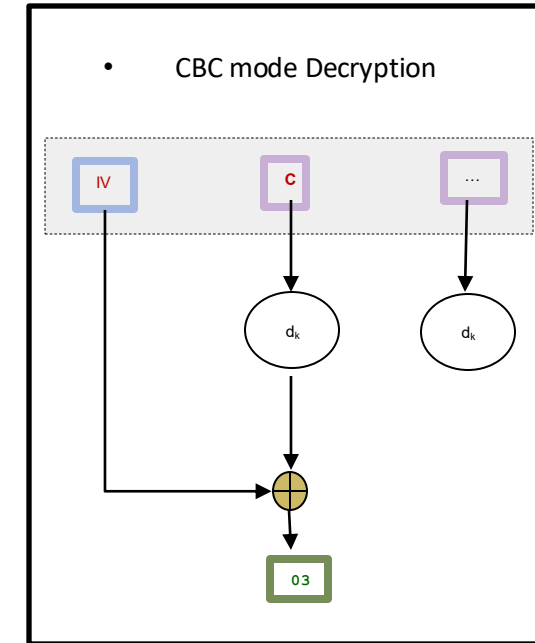
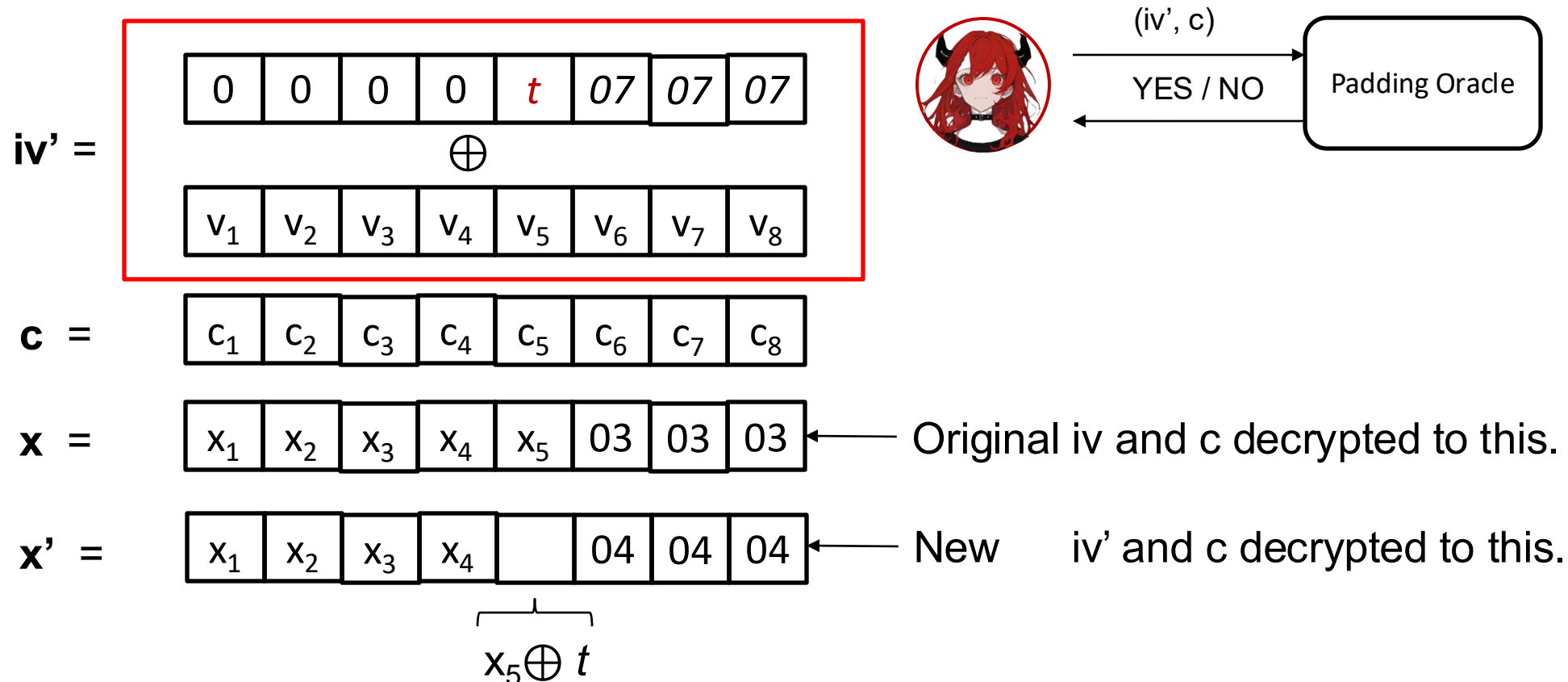


## The attacking steps

3 = 0 1 1  
4 = 1 0 0  
7 = 1 1 1

The attacker carries out these steps. The output is value of  $x_5$ :

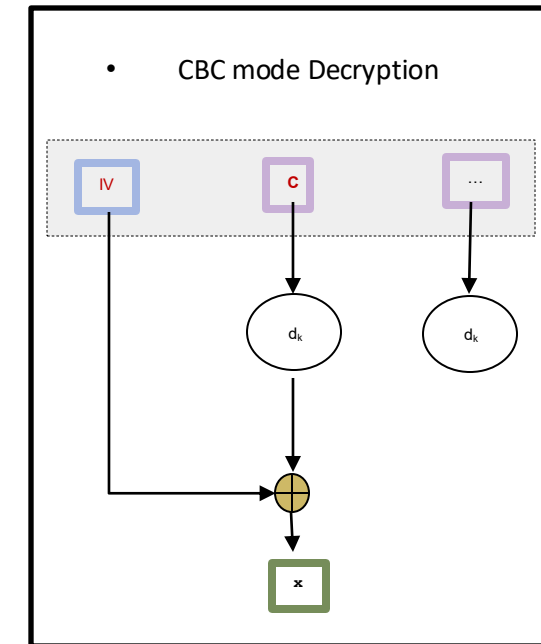
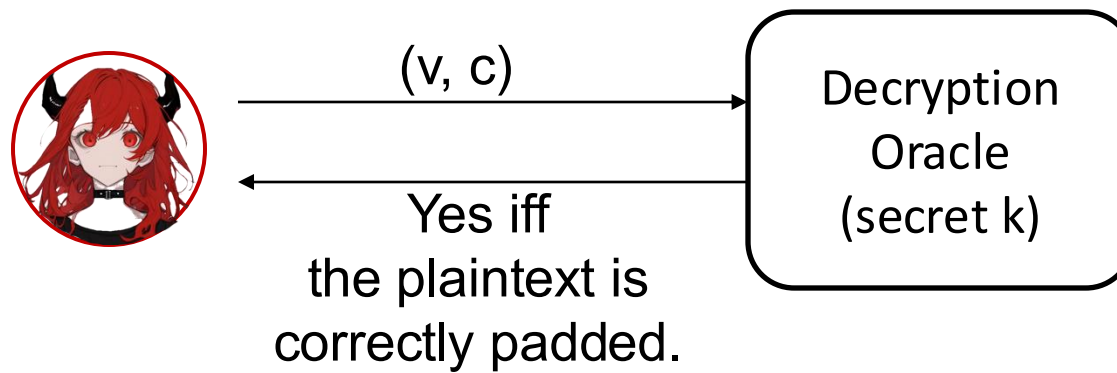
1. Calculates  $iv'$  from  $iv$  by xor the last 4 bytes (see figure), where  $t$  is set to some value, say  $t=0$ .
2. Feeds the padding oracle with  $(iv', c)$ .
3. If the oracle outputs YES, then  $(x_5 \oplus t)$  must be of the value 04. Outputs  $(04 \oplus t)$  and halts. (WHY?)
4. If NO, repeats the process with another value of  $t$ .



Choose another set of  $(iv, c)$  so that the decrypted plaintext changes to 04

# Padding oracle attack

- This algorithm outputs the value of  $x_5$ .
1. For  $t = 00$  to  $FF$       *// hexadecimal representation, i.e. 0 to 255 in decimal*
  2.     Let  $\mathbf{v} = \text{iv} \oplus \begin{bmatrix} 0 & 0 & 0 & 0 & t & 07 & 07 & 07 \end{bmatrix}$
  3.     Sends the two-block query  $(\mathbf{v} \parallel \mathbf{c})$  to **Oracle**.
  4.     If **Oracle** gives **YES**, then outputs  $(04 \oplus t)$
  5. End-for-loop



# Remarks

- We can easily extend the algorithm to find the full plaintext.
- There is still a gap. The algorithm needs to know the initial padding length. Fortunately, it is easy to determine the length (exercise).
- This attack is practical. There are protocols between a client and server which performs this:  
*If the client submits a ciphertext whose plaintext is not padded correctly, the server will reply with an error message.*
- Now, if an attacker obtained a ciphertext, the attacker could interact with the server to get the plaintext.

<https://vulert.com/vuln-db/CVE-2021-29443>

## CVE-2021-29443: Padding Oracle Attack In Jose Npm Library

Affected Package: [jose](#) (Click to see all vulnerabilities of this package)

Summary	19/04/2021	CVE-2021-29443	1.28.1	5.9
	Disclosed on	CVE	Patch	CVSS Score

### Background

The jose npm library is a cryptographic library that provides various cryptographic operations. It is commonly used for handling JSON Web Tokens (JWTs) and performing encryption and decryption operations. The library supports the AES\_CBC\_HMAC\_SHA2 algorithm for encryption and decryption. However, a vulnerability in the library allows for a padding oracle attack, which can lead to unauthorized decryption of data without knowledge of the decryption key.

### Vulnerability Detail

The vulnerability arises from a timing discrepancy in the decryption process of the AES\_CBC\_HMAC\_SHA2 algorithm. Normally, both HMAC tag verification and CBC decryption should be executed during decryption, and if either fails, a decryption error should be thrown. However, due to the timing difference in handling padding errors, an attacker can exploit a padding oracle to decrypt data without knowing the decryption key. This attack requires issuing an average of  $128 \cdot b$  calls to the padding oracle, where  $b$  is the number of bytes in the ciphertext block. The impact of this vulnerability is moderate, as it allows for unauthorized data decryption.

### Workaround

There is no specific workaround mentioned for this vulnerability.

### Conclusion

The jose npm library is vulnerable to a padding oracle attack, which can lead to unauthorized decryption of data. It is crucial to upgrade to the fixed versions of the library (v1.x to ^1.28.1, v2.x to ^2.0.5, and v3.x to ^3.11.4) to mitigate this vulnerability. For further support on vulnerability remediation, contact [DevNack.com](#). For a thorough exploration and assistance in countering such issues, consult the [Vulert Vulnerability Database](#).

# Prevention of padding-oracle attack.

- One method is to deny access to such oracle. Might not be feasible in some applications.
- Changing the padding standard may mitigate the attack. However, there could be other smarter way to attack the new padding.
- CTR mode also vulnerable to padding oracle.
- Padding oracle is a weaker form of Decryption oracle. Schemes that are secure against decryption oracle are also known as IND-CCA2 secure (indistinguishability, adaptive chosen ciphertext attack). GCM, or other “authenticated encryption”, is believed to be IND-CCA2 secure, and thus secure against padding oracle attack.



# 1.6 Cryptography Pitfalls

A secure encryption scheme can be vulnerable if not implemented or adopted properly. This section gives some examples.

1.6.1 – Wrong choice of IV (already discussed)

1.6.2 – Randomness is predictable

1.6.3 – Modify existing or design your own encryption scheme

1.6.4 – Reliance on Obscurity: Kerckhoff's principle.

(already discussed)

- Presence of decryption oracle but use a crypto that is not CCA2 secure.

(to be discussed in another topic)

- Using encryption for integrity (very subtle in some scenario)
- Side Channel Attack

## **1.6.1 Wrong choices of IV. Reusing one-time-pad**

# Wrong choices of IV

Some implementation overlooked IV generation. IV's must not be the same for two different ciphertext.

- E.g. To encrypt a file F, the IV is derived from the filename/meta-data. It is quite common to have files with the same filename/meta-data.

(**Read** **Schneier on Security, Microsoft RC4 Flaw.**

[https://www.schneier.com/blog/archives/2005/01/microsoft\\_rc4\\_f.html](https://www.schneier.com/blog/archives/2005/01/microsoft_rc4_f.html)

<http://eprint.iacr.org/2005/007.pdf> )

- E.g. When using AES under the “CBC mode”, the IV should be unpredictable to prevent a certain type of attack. (So, it is vulnerable to choose IV as 1,2,3,...)

The well-known BEAST attack exploits this.

(optional: <http://resources.infosecinstitute.com/ssl-attacks/> )

# Reusing one-time-pad

- The Verona project is a classic example on such failure.


(optional: [https://www.nsa.gov/about/files/cryptologic\\_heritage/publications/coldwar/venona\\_story.pdf](https://www.nsa.gov/about/files/cryptologic_heritage/publications/coldwar/venona_story.pdf) )

---

95

VERONA

~~TOP SECRET~~



---

USSR

Ref. No: 3/NBE/T1799

Issued: /13/7/1966

Copy No: 201.

FRAGMENTARY PRAGUE TEXT (1948)

From: MOSCOW

To: PRAGUE

No: 36

1 March 48

To MIKES [MIKESH][i]

[3 groups unrecovered] LI...[a]

[62 groups unrecoverable]

meeting with TEREZIE [TEREZIYa][ii] and [2 groups unrecovered].

No. 1865

DIRECTOR

---

Note: [a] Possibly the beginning of a proper name.

Comments: [i] MIKES: Unidentified; a Czech surname presumably used as a covername.

[ii] TEREZIE: Unidentified; presumably a covername. Also occurs in MOSCOW-PRAGUE No. 37 of 1st March 1948 (3/NBE/T1868).

---

DISTRIBUTION

## **1.6.2 Predictable secret generation**

(tutorial)

- **Scenario 1:**

- You are coding a program for a simulation system, for e.g. to simulate road traffic.
- In the program, you need a sequence of random numbers, for e.g. to decide the speed of the cars.
- How to get these random numbers?

- **Scenario 2:**

- You are coding a program for a security system.
- In the program, you need a random number. For e.g. you need to generate a random number as a temporary secret key.
- How to get these random numbers?

# to be discussed in tutorial

- In Java, what is the different between the following?
  - java.util.Random
  - java.security.SecureRandom
- In C, what is the different between using the following

```
#include <time.h>
#include <stdlib.h>
    srand(time(NULL));
    int r = rand();
```

and a complicated version below?

```
int byte_count = 64;
char data[64];
FILE *fp;
    fp = fopen("/dev/urandom", "r");
    fread(&data, 1, byte_count, fp);
fclose(fp);
```

## **1.6.3 Designing your own cipher**



- Don't design your own crypto or even make slight modification to existing scheme.
- Use well-accepted algorithms. E.g. AES.
- If possible, use well-established library. Don't write code to implement AES.
  - We might implement wrongly.
  - We might implement it insecurely. E.g. buffer overflow vulnerability, and side-channel attack.
- “Don't roll your own crypto”

read <http://security.stackexchange.com/questions/2202/lessons-learned-and-misconceptions-regarding-encryption-and-cryptology/2210#2210>

## **1.6.4 Reliance on Obscurity: Kerckhoffs's Principle**

# Kerckhoffs's principle

A system should be secure even if everything about the system, except the secret key, is public knowledge.

## Security through Obscurity

To hide the design of the system to achieve security.

# Examples (against obscurity)

- RC4 was introduced in 1987 and its algorithm was a trade secret. In 1994, a description of its algorithm was anonymously posted in a mailing group.

<http://en.wikipedia.org/wiki/RC4>

- MIFARE Classic is a contactless smartcard widely used in Europe. It uses a set of proprietary protocols/algorithms. However, they are reverse-engineered in 2007. It turns out that the encryption algorithms are already known to be weak (with 48-bit keys) and breakable.

<http://en.wikipedia.org/wiki/MIFARE>

Presentation (video) by the researcher who reversed-engineered it:

see <http://www.youtube.com/watch?gl=SG&hl=en-GB&v=QJyxUvMGLr0>

(the algo was revealed at 14:00)

# Examples (for obscurity)

- It is not advisable to reveal the computer network structure and settings (for example, location of firewall and the firewall rules), although these are not “secrets”, and many users within the organization may already know the settings.
- Although it is advisable to make the algorithm public, it is not advisable to publish the actual program used in a smart-card. By publishing the program/code, adversary may be able to identify implementation flaw that was previously unaware of or carry out side-channel attacks.
- Usernames are not secret. However, it is not advisable to publish all the usernames.

There is no contradiction.

- In general, obscurity can be used as an addition layer in the ***defense-in-depth*** strategy. It could deter or discourage novice attackers, but ineffective against attackers with high skill and motivation. We cannot rely solely on obscurity for security.

see

<http://technet.microsoft.com/en-us/magazine/2008.06.obscurity.aspx>

[http://en.wikipedia.org/wiki/Security\\_through\\_obscurity](http://en.wikipedia.org/wiki/Security_through_obscurity)

In this course, we always assume that the attackers know the algorithms.

## Others

- We have seen Padding oracle attack. In this attack, the attacker can modify the ciphertext, which can be viewed as compromise of “integrity”.
- Side-channel attack. To be introduced later.

## **1.7 Some historical facts**



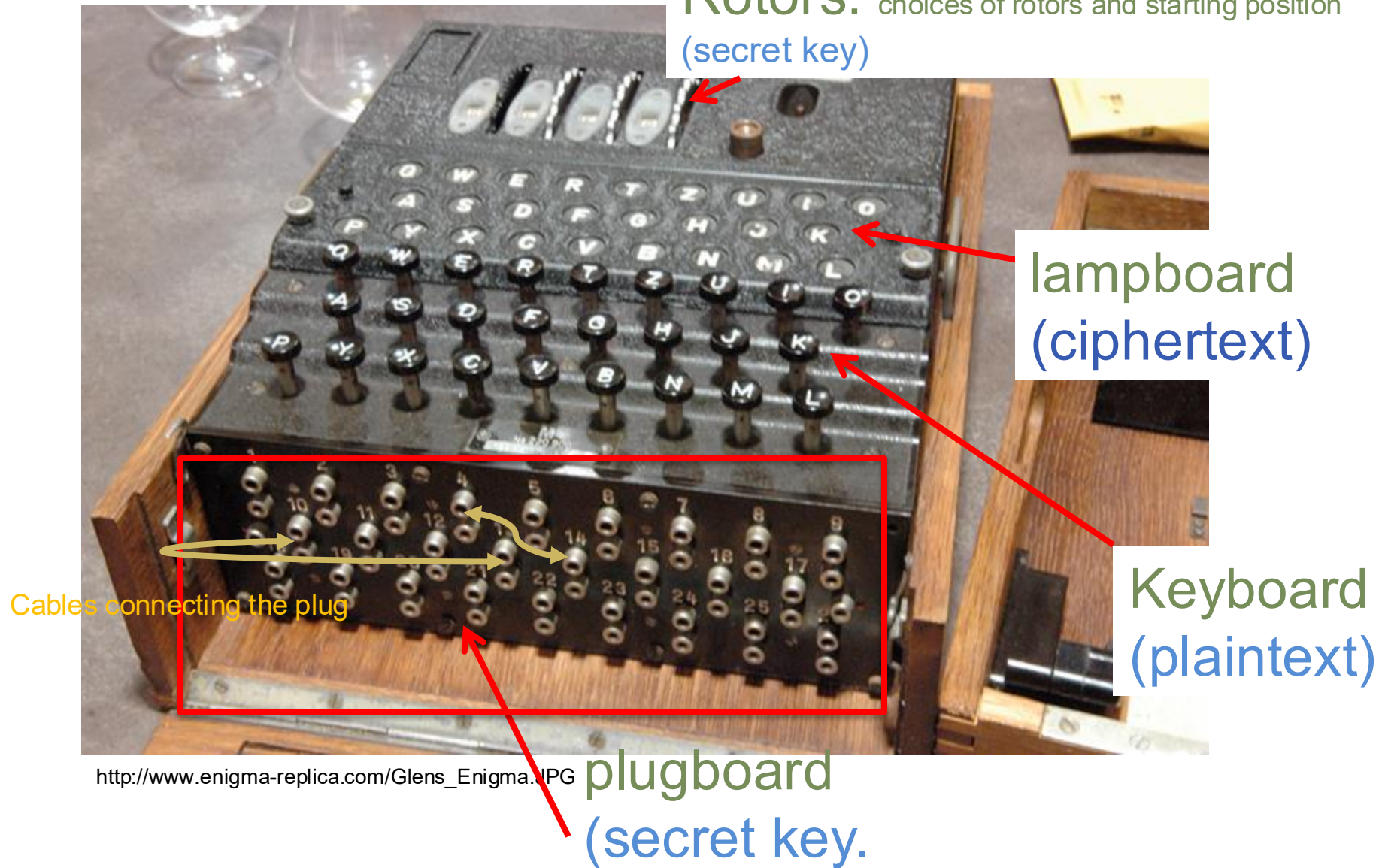
# Cryptography: History

- Cryptography is closely related to warfare and can be traced back to ancient Greece. Its role became significant when information is sent over-the-air. Cryptanalysis is one of the driving forces to the invention of computer (e.g. Colossus computer [https://en.wikipedia.org/wiki/Colossus\\_computer](https://en.wikipedia.org/wiki/Colossus_computer)).
- WWII: Famous encryption machines include the Enigma, and the Bombe (that helped to break Engima).

# Enigma.

“Compute” using Electrical + Mechanical mechanisms

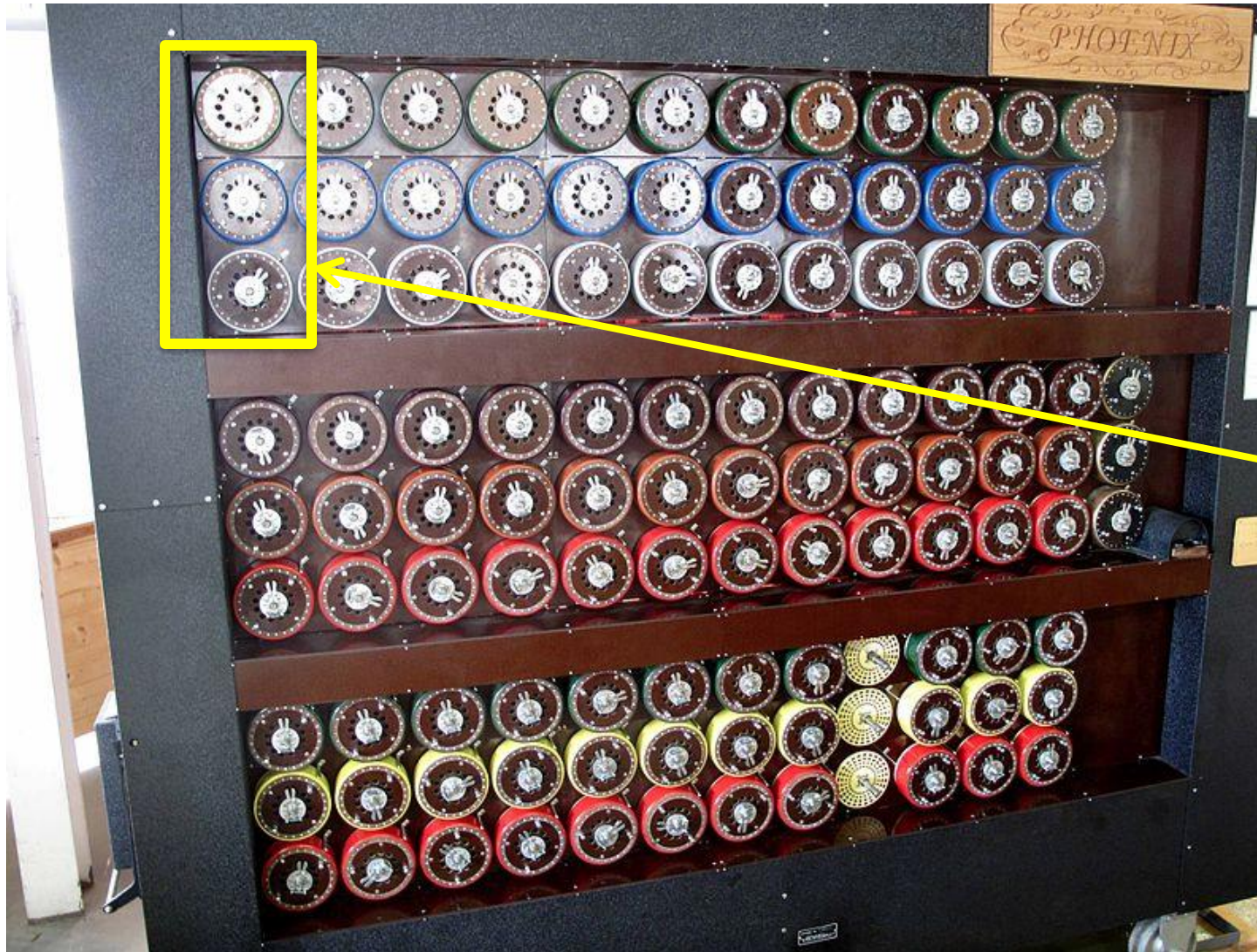
**Rotors:** choices of rotors and starting position  
(secret key)



A vulnerability significantly reduces the search space)



# Working rebuilt bombe at Bletchley Park museum.

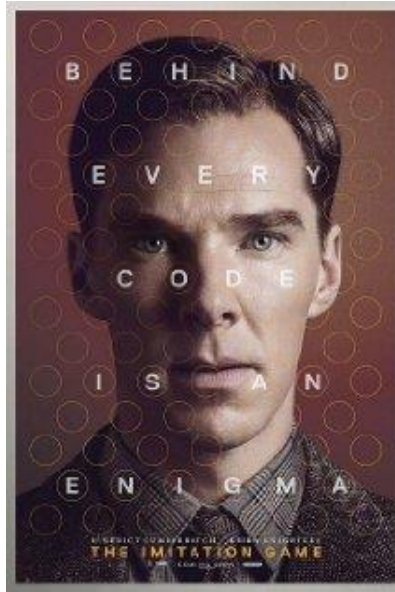


simulates  
the 3  
rotors  
in one  
Enigma  
machine

[http://en.wikipedia.org/wiki/Cryptanalysis\\_of\\_the\\_Enigma#Crib-based\\_decryption](http://en.wikipedia.org/wiki/Cryptanalysis_of_the_Enigma#Crib-based_decryption)

Exhaustive search on choicrotors + some smart trick to search plug cable config.

# Movie about encryption



## The Imitation Game.

During World War II, mathematician Alan Turing tries to crack Enigma with help from fellow mathematicians.

<http://www.imdb.com/title/tt2084970/>



## U-571

Fictional plot on how a U-boat was captured. Actual U-boat captured: U-110, U-505, U-570, U-744, U-1024.

Prize of capturing a U-boat instead of sinking it: Enigma.

[https://en.wikipedia.org/wiki/U-571\\_%28film%29](https://en.wikipedia.org/wiki/U-571_%28film%29)

# Modern Ciphers

## DES

- 1977: DES (Data Encryption Standard), 56 bits.

During cold war, cryptography, in particular DES was considered as “Munition”, and subjected to export control. (Currently, export of certain cryptography products is still controlled by US.)

**Read** the section on Singapore in <http://www.cryptolaw.org/cls2.htm>

- 1998: A DES key broken in 56 hours.

Triple DES (112 bits) is still in used.

- 2001: AES (Advanced Encryption Standard). NIST. 128, 192, 256 bits.

## RC4

- Designed by Ron Rivest (RSA Security) in 1987.
- Initially a trade secret. Algorithm leaked in 1994.
- Used in WEP (for wifi) in 1999. WEP implementation has 40 or 104-bit key. WEP was widely popular.
- 2001: a weakness in how WEP adopts RC4 is published by Fluhrer, Mantin, Shamir.
- 2005: a group from FBI demonstrated the attack.
- Industry switched to WPA2. (with WPA as an intermediate solution).