Lecture 1

Basic notions on Cryptology

UTAS
Sultanate of Oman
February 2023

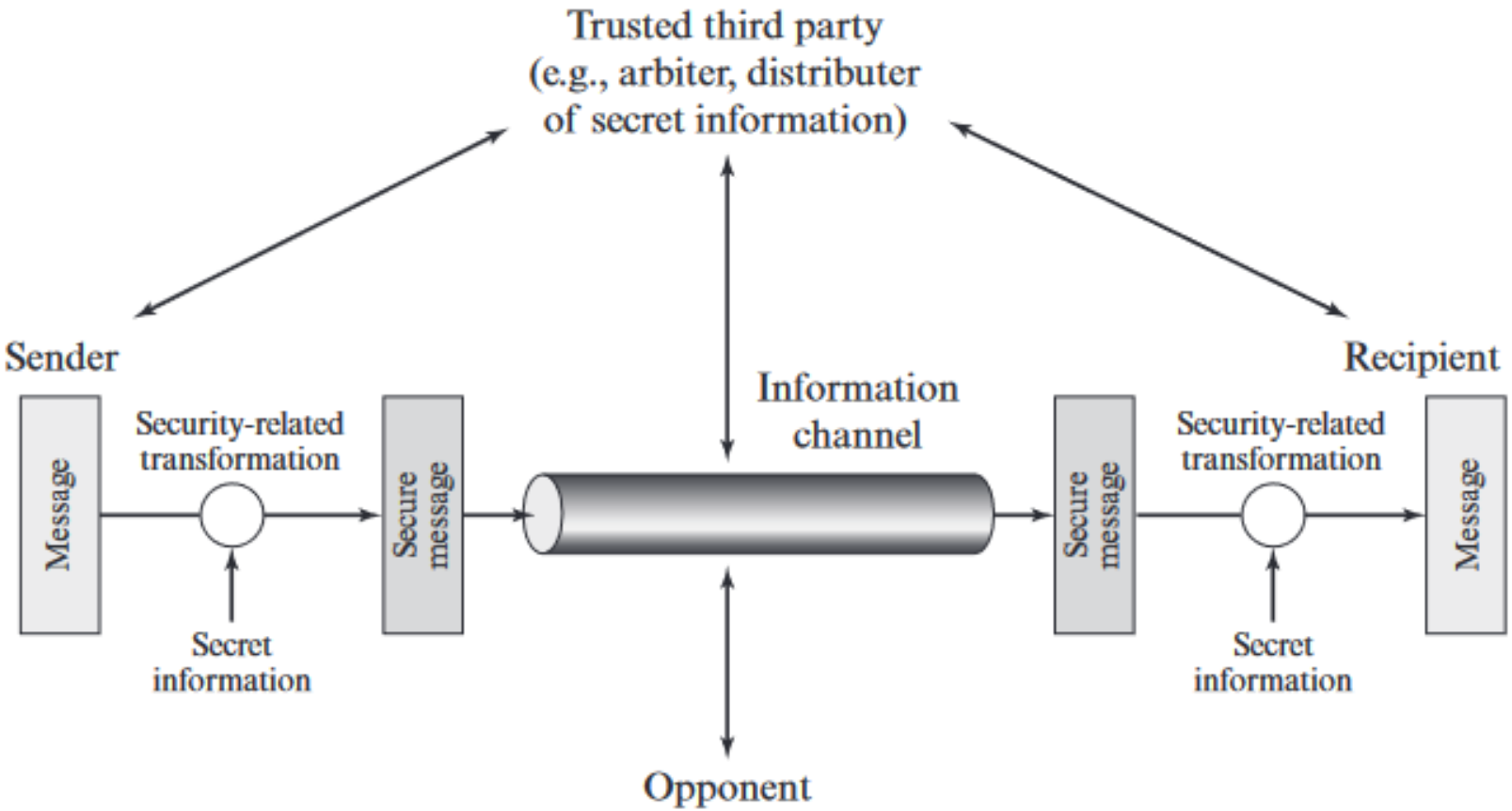
CSSY2201 : Introduction to Cryptography

جامعة التقنية
والعلوم التطبيقية
University of Technology
and Applied Sciences
Salalah صلالة

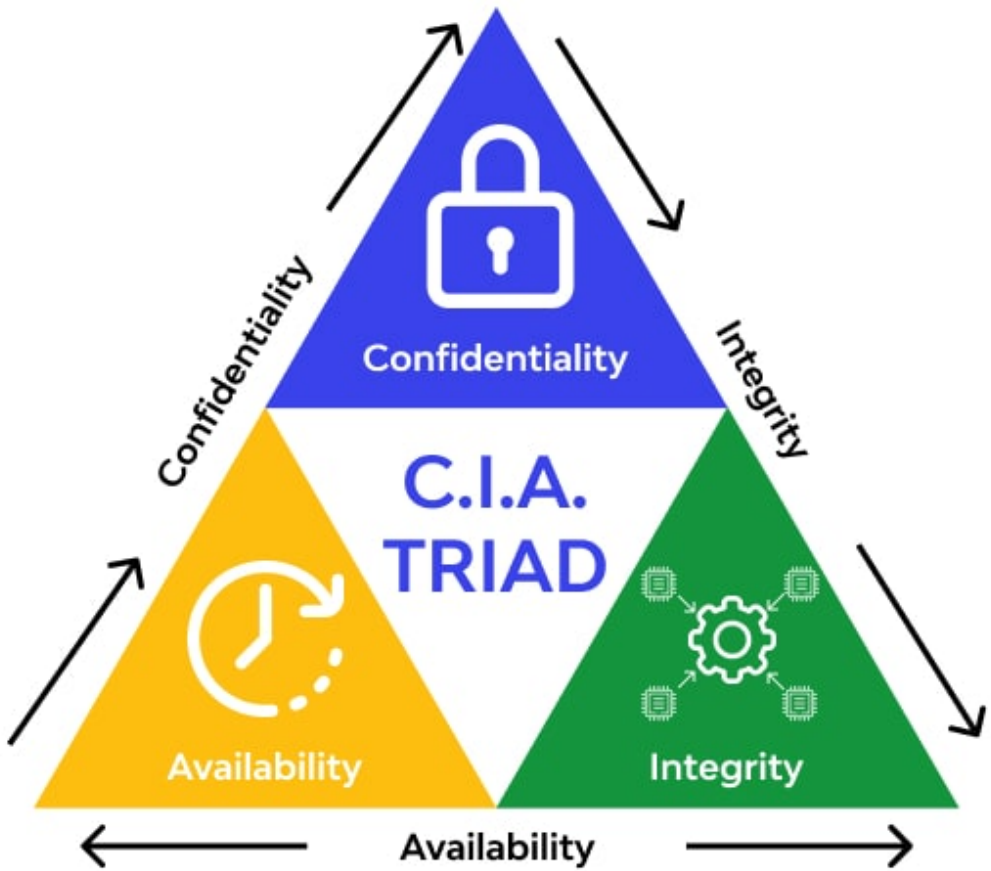
Plan

Plan

Theoretic model of a secure communication



Cryptograhy Objectives : CIA



Others : Authentication, Non-Repudiation

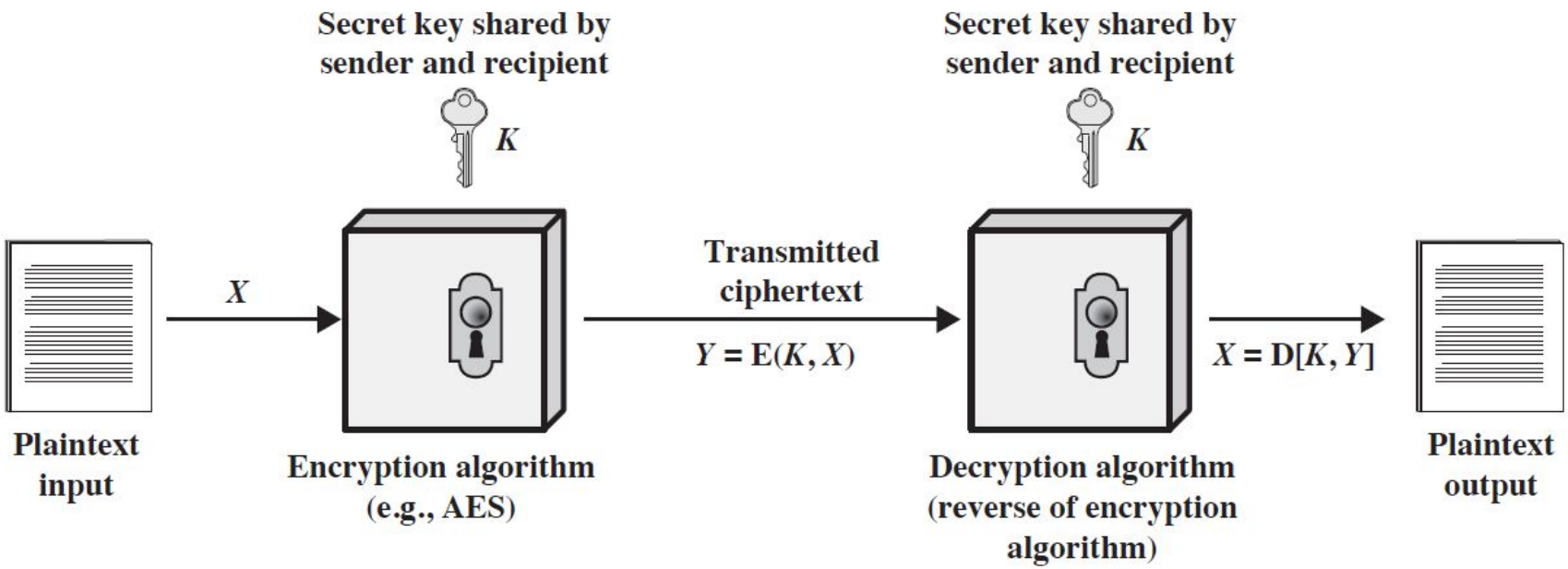
Attacks, services and Mechanisms

- Attack : Any action that compromises information security
- Security Mechanism : A mechanism that is designed to detect, prevent, or recover from a security attack.
- Security Service : A service that improves the security of data processing systems and information transfers. A security service makes use of one or more security mechanisms

Terminology

- Plaintext : original text
- Ciphertext : encrypted text
- Encryption : The process of conversion from plaintext to ciphertext
- Decryption : The process of conversion from ciphertext to plaintext
- Cryptography : Science of secret messages
- Cryptanalysis : Science of breaking secret messages
- Cryptology : science of cryptography & cryptanalysis
- Cryptosystem/cipher : Encryption Algorithm

Symmetric encryption



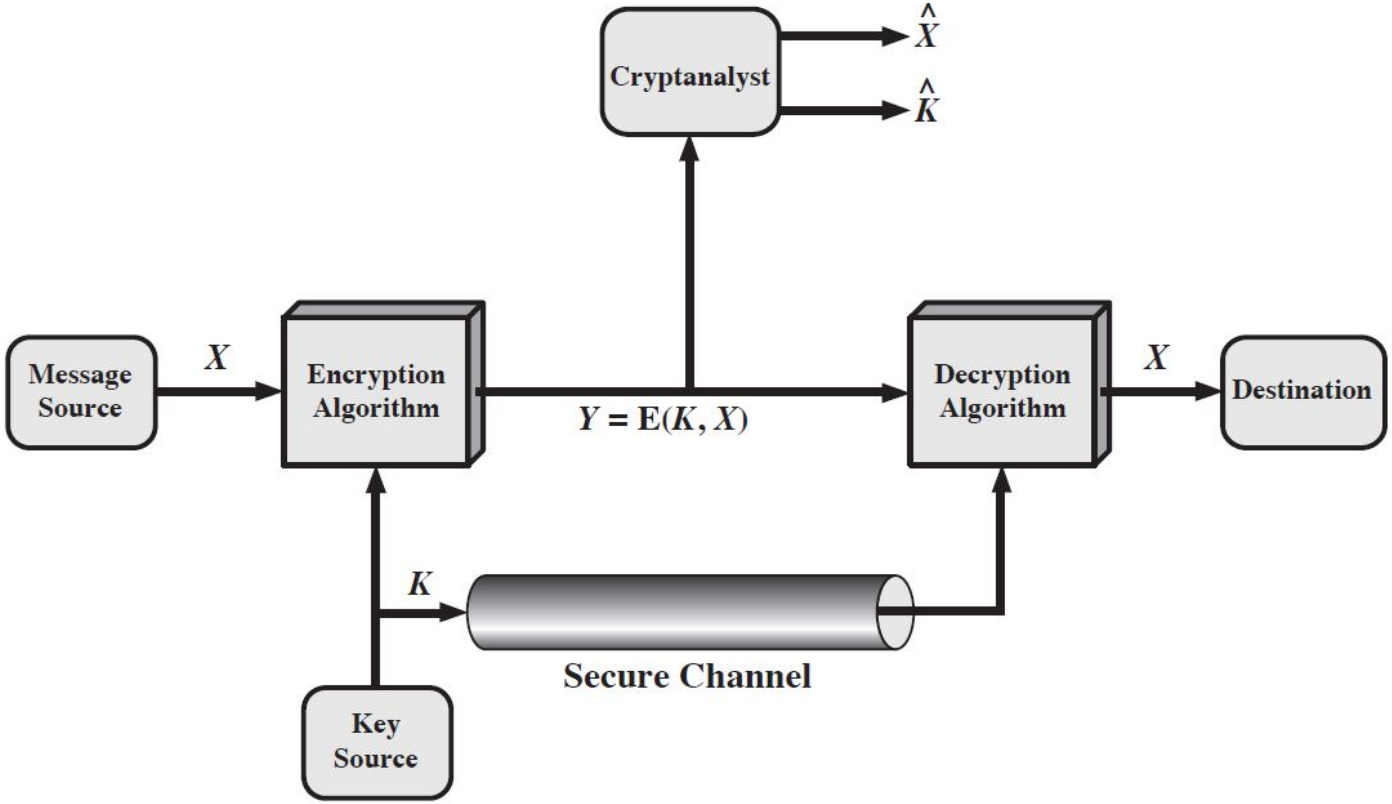
Problems of symmetric cryptography

- symmetric cryptography use only one key for encryption/decryption named (secret key)
- The secret key should be pre-shared between sender and receiver before secret communication.
- if the secret key has been disclosed, then all the communication is compromised.
- does not protect the sender from the receiver which can claim receiving an encrypted message from the sender.
- dose not protect the receiver from someone pretending being the legitimate sender

Plan

Symmetric encryption Model

We introduce the science of breaking secret messages in a symmetric cryptography concept.
here is the model of symmetric cryptography with the presence of a third user (the malicious actor, the hacker, the cryptanalyst)

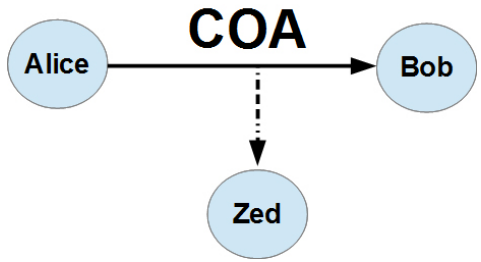


Cryptanalysis

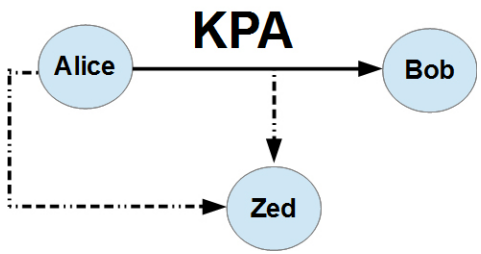
- Objective is to find the secret key not only the plaintext
- brute force attack :
  - Exploit the key length : try all the combinations (on a ciphertext to decrypt it) of the secret key until finding the right one.
  - In average, a hacker needs to try at least half of the key space to break the cryptosystem.
- Cryptanalytic attack : more intelligent, exploits the cryptosystem design vulnerabilities.
- Generally, the cryptanalyst follows 3 steps to break a cryptosystem : (1) Data collection, (2) Analysis, deduction, (3) Exploitation.

Data Collection : Theoretic weaknesses

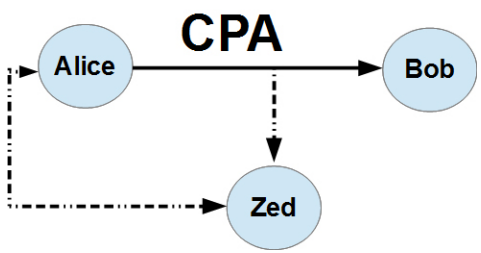
- Observation or action : the hacker is "on-line" connected to the target.
  - Ciphertext-only attack(COA)



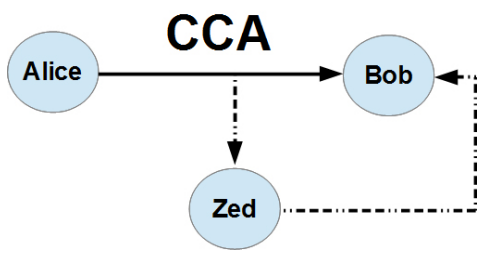
  - Known-plaintext attack(KPA)



  - Chosen-plaintext attack(CPA)

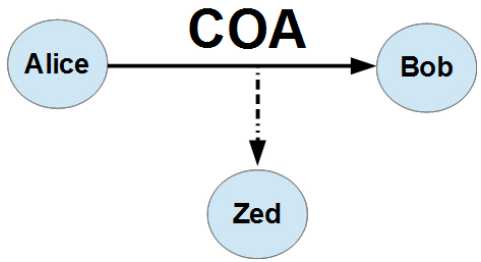


  - Chosen-ciphertext attack(CCA)

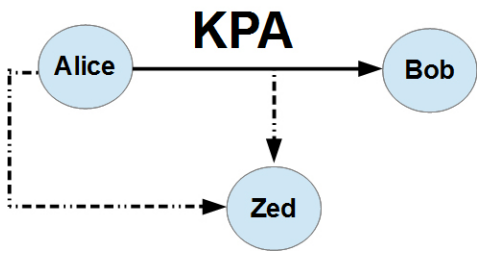


Data Collection : Theoretic weaknesses

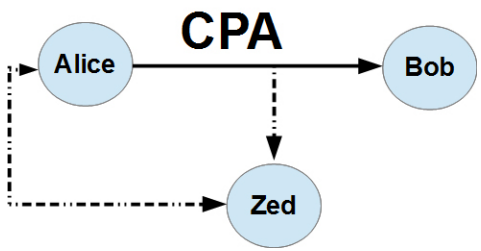
- Observation or action : the hacker is "on-line" connected to the target.
  - Ciphertext-only attack(COA)



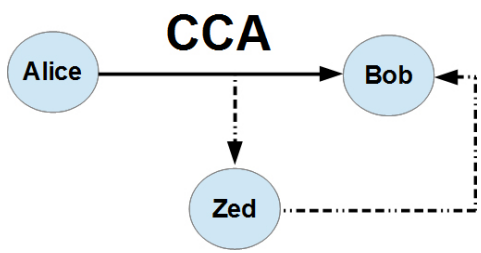
  - Known-plaintext attack(KPA)



  - Chosen-plaintext attack(CPA)

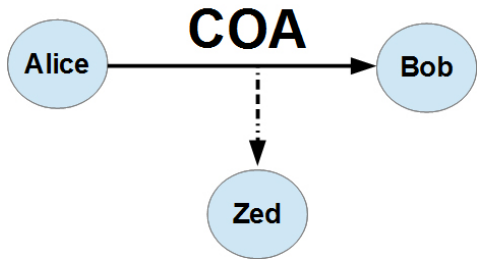


  - Chosen-ciphertext attack(CCA)

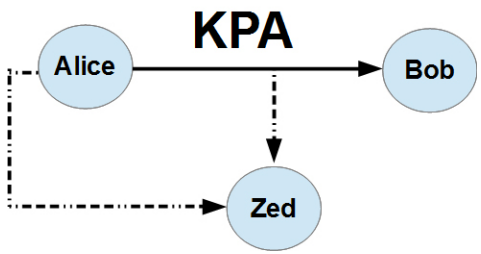


Data Collection : Theoretic weaknesses

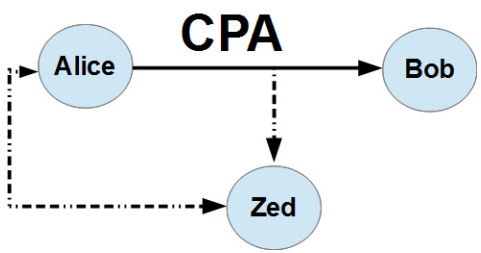
- Observation or action : the hacker is "on-line" connected to the target.
  - Ciphertext-only attack(COA)



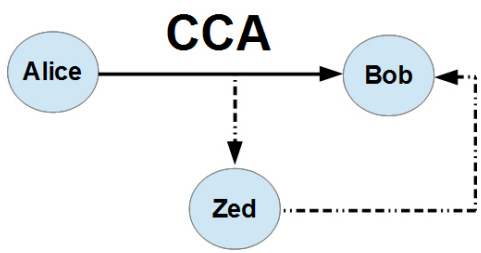
  - Known-plaintext attack(KPA)



  - Chosen-plaintext attack(CPA)

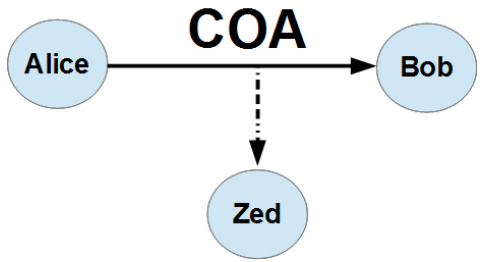


  - Chosen-ciphertext attack(CCA)

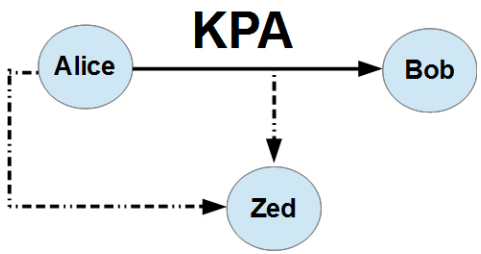


Data Collection : Theoretic weaknesses

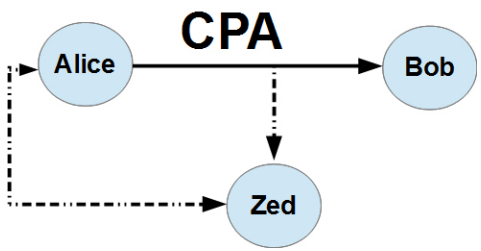
- Observation or action : the hacker is "on-line" connected to the target.
  - Ciphertext-only attack(COA)



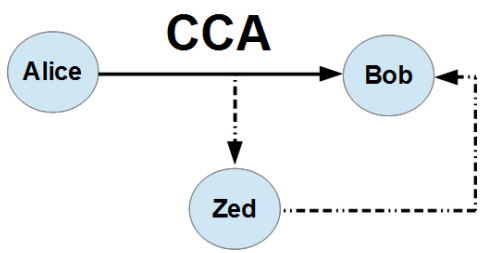
  - Known-plaintext attack(KPA)



  - Chosen-plaintext attack(CPA)



  - Chosen-ciphertext attack(CCA)



Analysis, deduction and Exploit

- "off-line" step : Analysis & Deduction
  - Brute force attack : try all the combinations of the key to find the plaintext from the ciphertext
  - statistical attack : Estimate the occurrence frequency of letters in a text
  - Algebraic attack : try to find equivalent representation of the encryption algorithm to simplify it.
  - Linear cryptanalysis : Linear approximation of the encryption algorithm (which is a non-linear system)
  - Differential cryptanalysis : Study how the plaintexts difference propagate and affect the ciphertext difference to find unbalanced output.
- Exploit : Estimate the decryption key.

Analysis, deduction and Exploit

- "off-line" step : Analysis & Deduction
  - Brute force attack : try all the combinations of the key to find the plaintext from the ciphertext
  - statistical attack : Estimate the occurrence frequency of letters in a text
  - Algebraic attack : try to find equivalent representation of the encryption algorithm to simplify it.
  - Linear cryptanalysis : Linear approximation of the encryption algorithm (which is a non-linear system)
  - Differential cryptanalysis : Study how the plaintexts difference propagate and affect the ciphertext difference to find unbalanced output.
- Exploit : Estimate the decryption key.

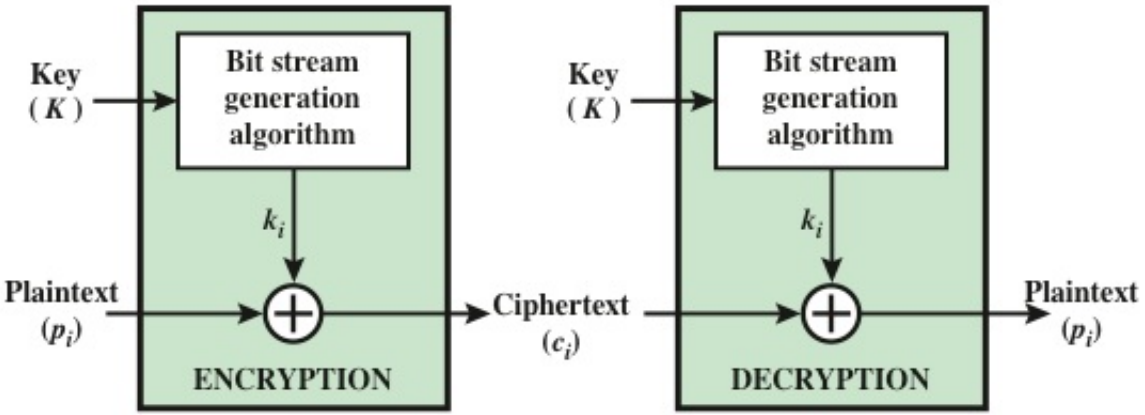
Examples of brute force attack

Key Size (bits)	Number of Alternative Keys	Time required at 1 decryption/μs	Time required at 10^6 decryptions/μs
32	$2^{32} = 4.3 \times 10^9$	2^{31} μs = 35.8 minutes	2.15 milliseconds
56	$2^{56} = 7.2 \times 10^{16}$	2^{55} μs = 1142 years	10.01 hours
128	$2^{128} = 3.4 \times 10^{38}$	2^{127} μs = 5.4×10^{24} years	5.4×10^{18} years
168	$2^{168} = 3.7 \times 10^{50}$	2^{167} μs = 5.9×10^{36} years	5.9×10^{30} years
26 characters (permutation)	$26! = 4 \times 10^{26}$	2×10^{26} μs = 6.4×10^{12} years	6.4×10^6 years

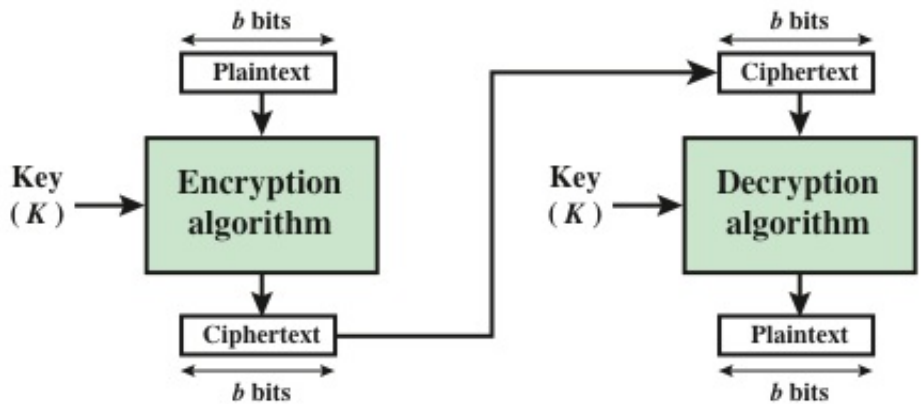
Plan

Block Ciphers and Stream ciphers

- Block Ciphers process the plaintext block by block to be encrypted. and treats the ciphertext block by block too. (ex : DES, AES). -> suitable for strong encryption, encryption of data in rest, encryption of small data amount
- Stream ciphers process the plaintext bit by bit (or byte by byte) to be encrypted. and treat the ciphertext the same way (ex : viginere, vernam) -> suitable for Fast encryption, data in transit encryption, noisy channel like wifi, real-time encryption...



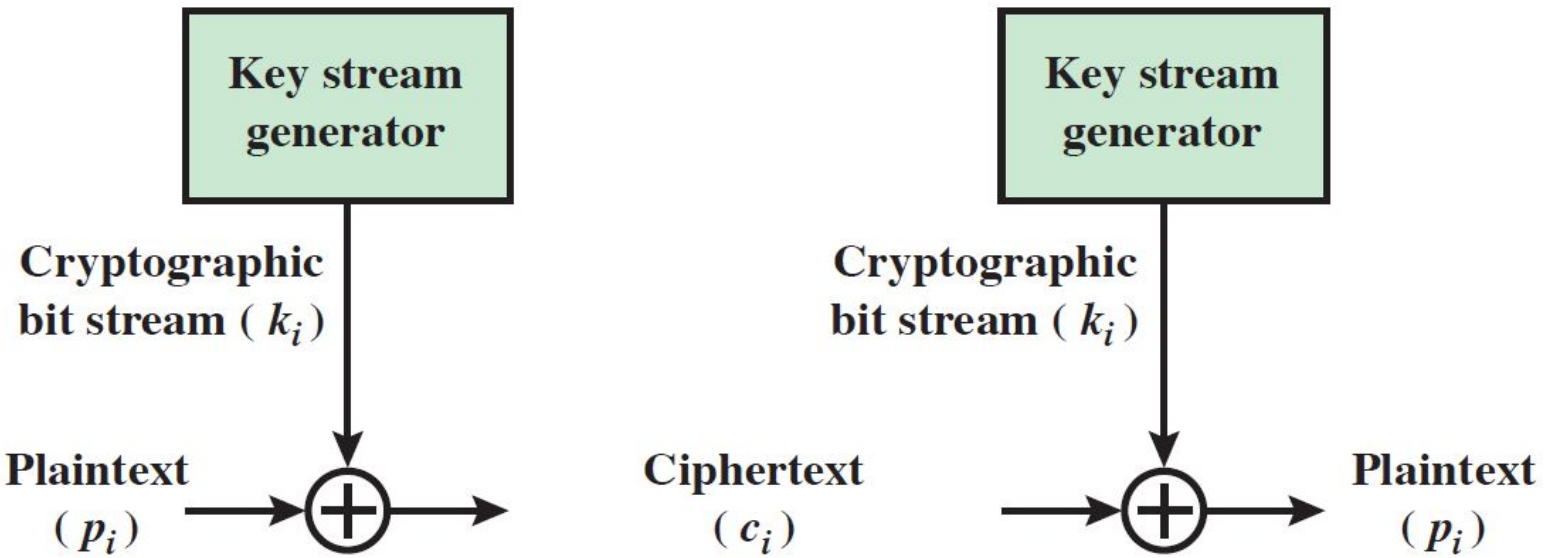
(a) Stream Cipher Using Algorithmic Bit Stream Generator



(b) Block Cipher

Stream cipher : Vernam cipher

- use a key as lomg as the plaintext
- invented by an AT&T enginnner Gilbert Vernam in 1918



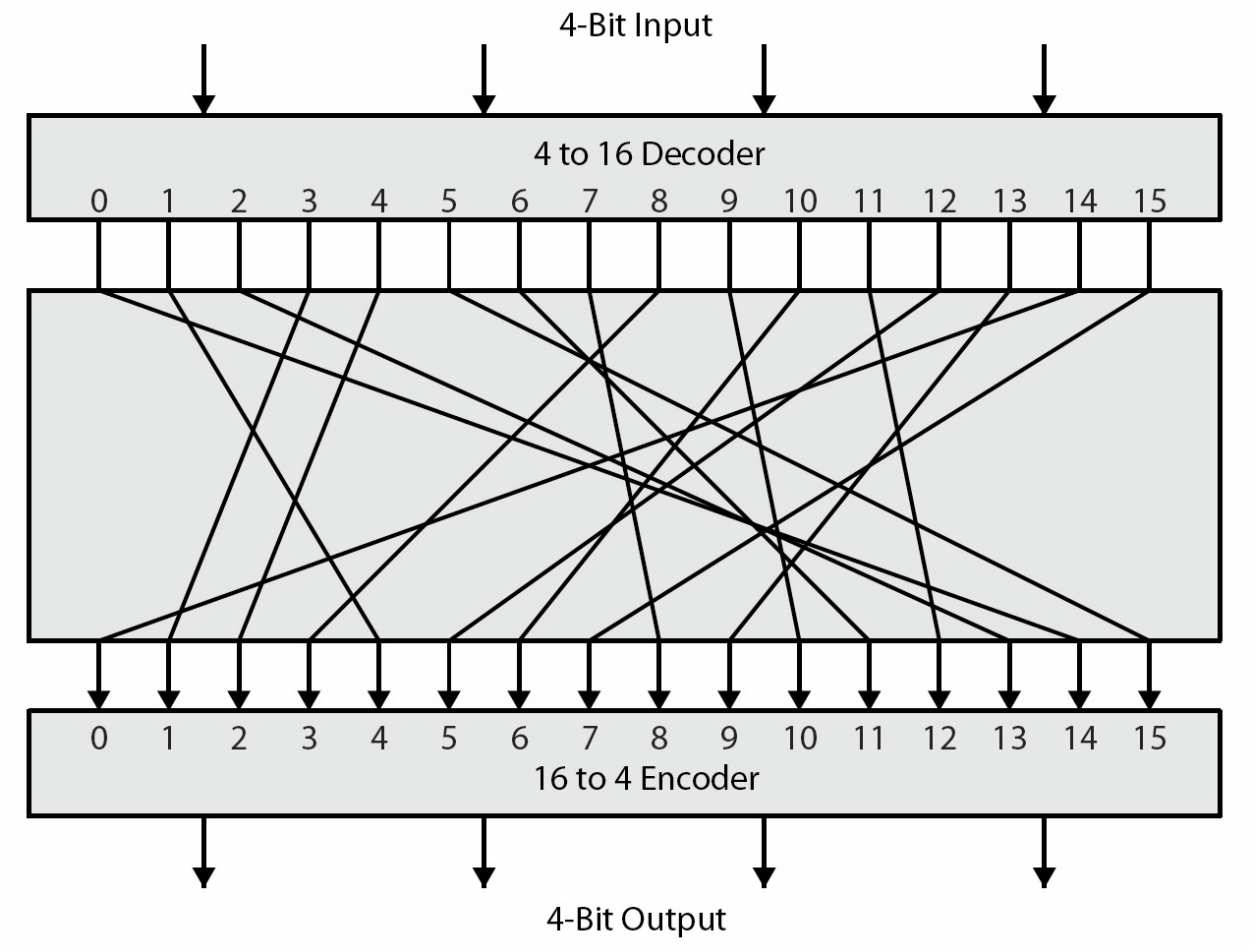
reversible and irreversible functions

- A block cipher alg takes n bits of plaintext and turns it into n bits of ciphertext
- there are 2^n possible combinations of plaintext
- Encryption must be reversible
- each plaintext block produces a different ciphertext block (bijectivity)
- there are 2^n possible transformations

Reversible Mapping	
Plaintext	Ciphertext
00	11
01	10
10	00
11	01

Irreversible Mapping	
Plaintext	Ciphertext
00	11
01	10
10	01
11	01

model of a block cipher : 4-bit length word



Look-up tables of the bloc cipher example

Plaintext	Ciphertext
0000	1110
0001	0100
0010	1101
0011	0001
0100	0010
0101	1111
0110	1011
0111	1000
1000	0011
1001	1010
1010	0110
1011	1100
1100	0101
1101	1001
1110	0000
1111	0111

Ciphertext	Plaintext
0000	1110
0001	0011
0010	0100
0011	1000
0100	0001
0101	1100
0110	1010
0111	1111
1000	0111
1001	1101
1010	1001
1011	0110
1100	1011
1101	0010
1110	0000
1111	0101

Plan

ASCII encoding

- ASCII, abbreviated from American Standard Code for Information Interchange, is a character-encoding standard for electronic communication. ASCII codes represent text in computers, telecommunications equipment, and other devices. Most modern character-encoding schemes are based on ASCII, although they support many additional characters.
- In ASCII encoding, each letter is converted to one byte. Look at the following examples :

$$A = 65\ or\ 0b01000001$$

$$B = 66\ or\ 0b01000010$$

$$C = 67\ or\ 0b01000011$$

$$ABC = 0b01000001\ 0b01000010\ 0b01000011$$

Base64 Encoding Text

- Base64, also known as privacy enhanced electronic mail (PEM), is the encoding that converts binary data into a textual format ; it can be passed through communication channels where text can be handled in a safe environment. PEM is primarily used in the email encryption process. To use the functions included in the Base64 module, you will need to import the library in your code. Base64 offers a decode and encode module that both accepts input and provides output.
- To break ASCII encoding into Base64-encoded text, each sequence of six bits encodes to a single character.

Base64 table

Index	Binary	Char	Index	Binary	Char	Index	Binary	Char	Index	Binary	Char
0	000000	A	16	010000	Q	32	100000	g	48	110000	w
1	000001	B	17	010001	R	33	100001	h	49	110001	x
2	000010	C	18	010010	S	34	100010	i	50	110010	y
3	000011	D	19	010011	T	35	100011	j	51	110011	z
4	000100	E	20	010100	U	36	100100	k	52	110100	0
5	000101	F	21	010101	V	37	100101	l	53	110101	1
6	000110	G	22	010110	W	38	100110	m	54	110110	2
7	000111	H	23	010111	X	39	100111	n	55	110111	3
8	001000	I	24	011000	Y	40	101000	o	56	111000	4
9	001001	J	25	011001	Z	41	101001	p	57	111001	5
10	001010	K	26	011010	a	42	101010	q	58	111010	6
11	001011	L	27	011011	b	43	101011	r	59	111011	7
12	001100	M	28	011100	c	44	101100	s	60	111100	8
13	001101	N	29	011101	d	45	101101	t	61	111101	9
14	001110	O	30	011110	e	46	101110	u	62	111110	+
15	001111	P	31	011111	f	47	101111	v	63	111111	/
Padding		=									

- Examine the 24 bits from the previous section :

$$0b01000001\ 0b01000010\ 0b01000011$$

- Break the line into 6-bit groups :

$$0b010000\ 010100\ 001001\ 000011$$

- When you convert the four groups to decimal, you will see that they are equal to the following :

$$16\ 20\ 9\ 3$$

- You now convert the numbers to Base64 :

$$QUJD$$

- Therefore, when you encode "ABC" to Base64, you should end up with QUJD, as shown here :

```
>>> import base64
>>> value = 'ABC'.encode()
>>> print(base64.b64encode(value))
b'QUJD'
```

- In the event that the text cannot be broken down into groups of six, you will see the padding character, which is shown using the equal sign =. If the example had four bytes : for example 'ABCD', then the output : 'QUJDRA==' :

```
In [4]: import base64

In [5]: value='ABCD'.encode()

In [6]: print(base64.b64encode(value))
b'QUJDRA=='
```

utf-8

- UTF-8 is a variable-width character encoding used for electronic communication. Defined by the Unicode Standard, the name is derived from Unicode (or Universal Coded Character Set) Transformation Format 8-bit.
- UTF-8 is capable of encoding all 1,112,064 valid character code points in Unicode using one to four one-byte (8-bit) code units. Code points with lower numerical values, which tend to occur more frequently, are encoded using fewer bytes. It was designed for backward compatibility with ASCII : the first 128 characters of Unicode, which correspond one-to-one with ASCII, are encoded using a single byte with the same binary value as ASCII, so that valid ASCII text is valid UTF-8 encoded Unicode as well.

utf-8

- When dealing with displaying ciphertexts and hashes (digests), the output is binary (raw bytes). when you try to display this what you get :

```
>>>import hashlib
>>>plaintext_password = b'password'
>>>hashed = hashlib.md5(plaintext_password).digest()
>>>print(hashed)


b"_M\xcc;Z\xa7e\xd6\x1d\x83'\xde\xb8\x82\xcf\x99"
```

- to be readable, convert it to hexadecimal or base64 like follows :

```
>>>import hashlib
>>>plaintext_password = b'password'
>>>hashedHEX = hashlib.md5(plaintext_password).hexdigest()
>>>print(hashedHEX)

5f4dcc3b5aa765d61d8327deb882cf99
```

utf-8

- or simply produce the binary digest and use *hex* and *base*64.*b*64*encode* functions to produce the hexadecimal and base64 representation of the digest :

```
import hashlib
import base64
plaintext_password = b'password'
hashed = hashlib.md5(plaintext_password).digest()
print("binary digest =", hashed)
hhex=hashed.hex()
print("hexadecimal digest =", hhex)
encoded = base64.b64encode(hashed)
print("base64 digest =", encoded)
```

- result will be the same hash displayed in binary, hexadecimal, and base64 encoding.

```
~/practical-crypto-course$ python lect2.py
binary digest = b"_M\xcc;Z\xa7e\xd6\x1d\x83'\xde\xb8\x82\xcf\x99"
hexadecimal digest = 5f4dcc3b5aa765d61d8327deb882cf99
base64 digest = b'X03MO1qnZdYdgyfeuILPmQ=='
```