NTNU

Norwegian University of Science and Technology

Cryptography intro (for developers)

TDT4237 2025

OWASP A02:2021
Cryptographic Failures

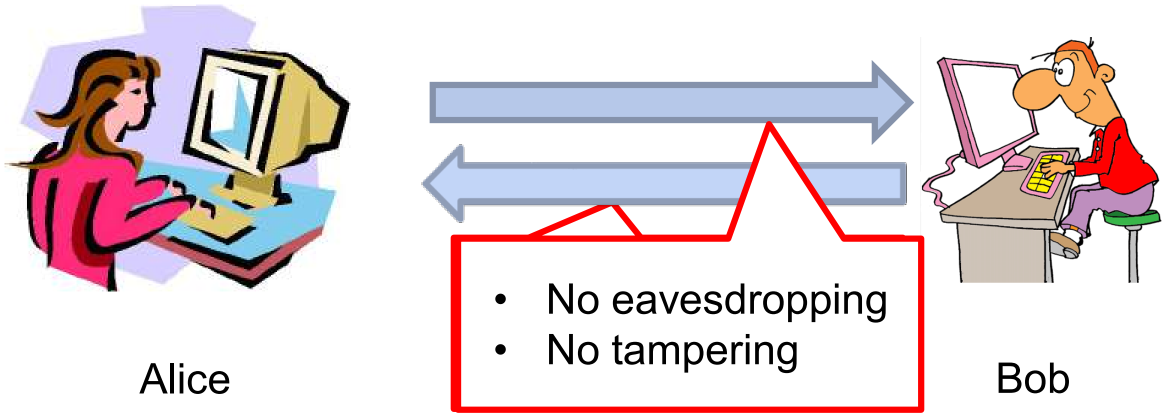
https://xkcd.com/1323/

I'VE DISCOVERED A WAY TO GET COMPUTER SCIENTISTS TO LISTEN TO ANY BORING STORY.

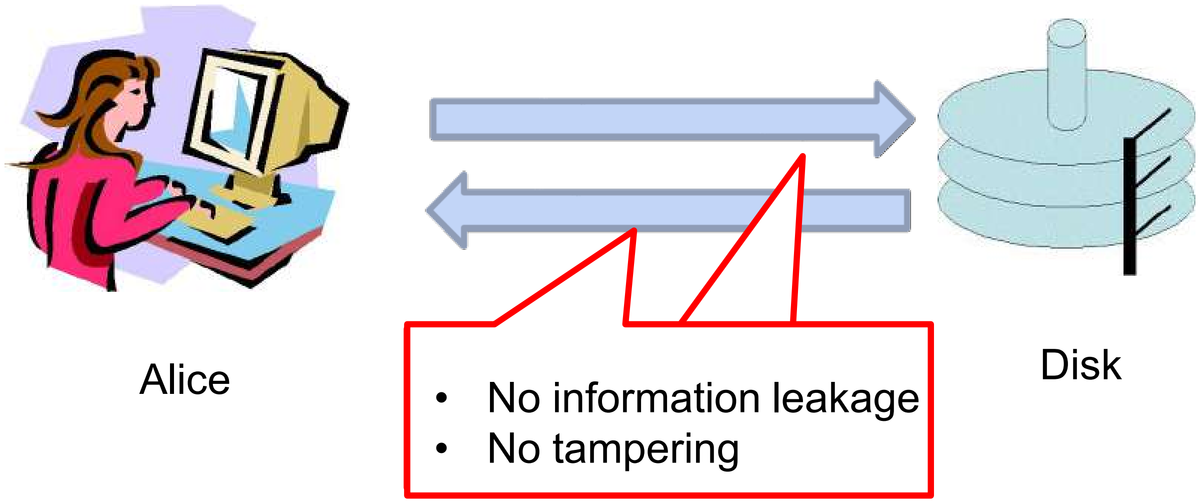
You are expected to learn

- Basic concepts of popular cryptography algorithms and how to use them
  - Ciphers, e.g., AES, 3DES
  - Cryptographic hash function, e.g., MD5, SHA-1, SHA-2
  - Public key cryptography, e.g., RSA, ECDSA
  - A dash of quantum stuff
- Necessary knowledge to understand
  - Unsafe defaults
  - Configure web server and web site
    - E.g.,https://httpd.apache.org/docs/2.4/ssl/ssl_howto.html
  - Make tradeoffs

Secure communication



Secure data storage



Analogous to secure communication:

- Alice today sends a message to Alice tomorrow

Secure communication has two steps

Step 1: Establish a shared secret key through, e.g.,

- Face-to-face meeting
- Trusted courier
- Handshake algorithms

Step 2: Transmit data using the shared secret key

We'll start by introducing algorithms for the second step

Transmit data using a shared secret key

- Confidentiality (No eavesdropping)
  - Substitution cipher
  - Shift cipher
  - The Vigenère cipher
  - One time pad
  - Stream cipher
  - Block cipher
- Integrity (No tampering)
  - ECB
  - HMAC
- Combining confidentiality and integrity

Substitution cipher (Polybios)

S	Y	D	W	Z
R	I	P	U	L
H	C	A	X	F
T	N	O	G	E
B	K	M	Q	V

SS TY TD SW = ?

Shift cipher

`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`

encrypt

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

decrypt

`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`

Shift cipher

- Associate every English letter with a number
  - a with 0; b with 1; …; z with 25
- Choose a letter K (i.e., the encryption key) and associate K to a number ϵ {0, …, 25}
- To encrypt using key K, shift every letter of plaintext by K position (with wraparound)
- To decrypt, do the reverse

Shift cipher (cont')

- To encrypt: **C = (M + K) mod 26**

C: Ciphertext

M: Plaintext

K: Secret key (a single letter that reflects the number of positions to shift)

```
plaintext     helloworld
key           cccccccccc
ciphertext    jgnnqyqtnf
```

- To decrypt: **M = (C - K) mod 26**

Shift cipher is insecure

- Only 26 possible keys
- Try to decrypt ciphertext with every possible key
- Only one generated plaintext “makes sense”

Decryption shift	Candidate plaintext
0	exxegoexsrgi
1	dwwdfndwrqfh
2	cvvcemcvqpeg
3	buubdlbupodf
4	attackatonce
5	zsszbjzsnmbd
6	yrryaiyrmlac
…	
23	haahjrhavujl
24	gzzgiqgzutik
25	fyyfhpfytshj

The Vigenère cipher

- Invented in the 16th century
- Key is a string (e.g., “*cafe*”), not a single letter
- Encrypt: shift each character in the plaintext by the amount dictated by the character of the key (with wraparound)

```
plaintext     tellhimaboutme …
key           cafecafecafeca …
ciphertext    veqpjiredozxoe …
```

- Decrypt: do the reverse

The Vigenère cipher vs. shift cipher

- Vigenère has much more keyspace

  26^n, where n is the length of the key

  n= 3, 26^3 = 17576

  n= 14, 26^{14} = 64509974703297200000

- Brute-force search expensive/impossible

- Believed to be secure for many years …

The Vigenère cipher is insecure*

- Given long enough ciphertext and a short key, the Vigenère method is insecure

- Key Issue: If the key length is n, every n^{th} character is encrypted using the same character in the key

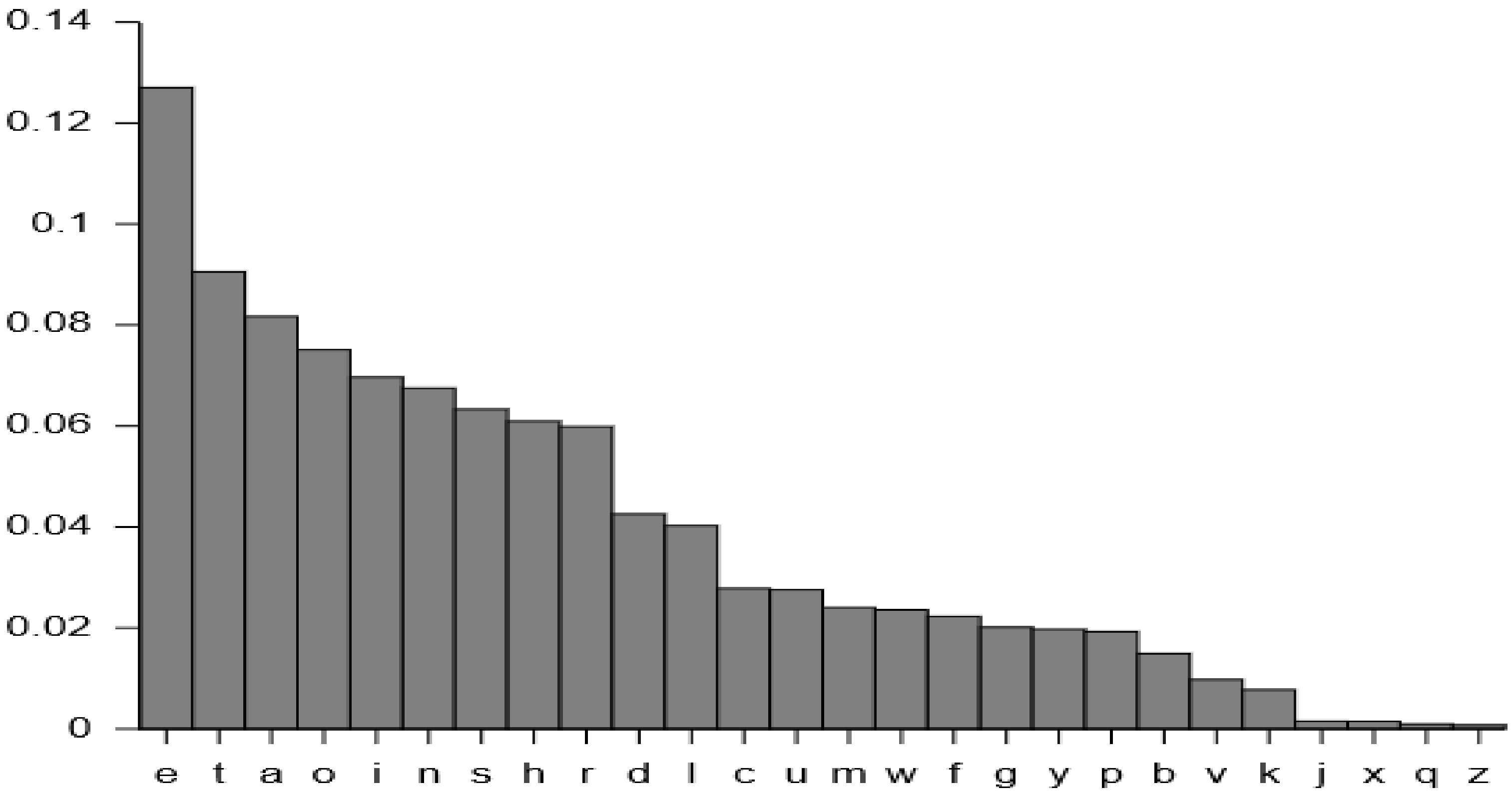
- For example, if the length of the key is 4, we can pick four sequences. All characters in each sequence are encrypted with the same character.
  - 1, 5, 9, …(c)
  - 2, 6, 10, ... (a)
  - 3, 7, 11, ... (f)
  - 4, 8, 12, … (e)

`plaintext`	`tellhimaboutme …`
`key`	`cafecafecafeca …`
`ciphertext`	`veqpjiredozxoe …`

*How to crack vigenere cipher: http://practicalcryptography.com/cryptanalysis/stochastic-searching/cryptanalysis-vigenere-cipher/

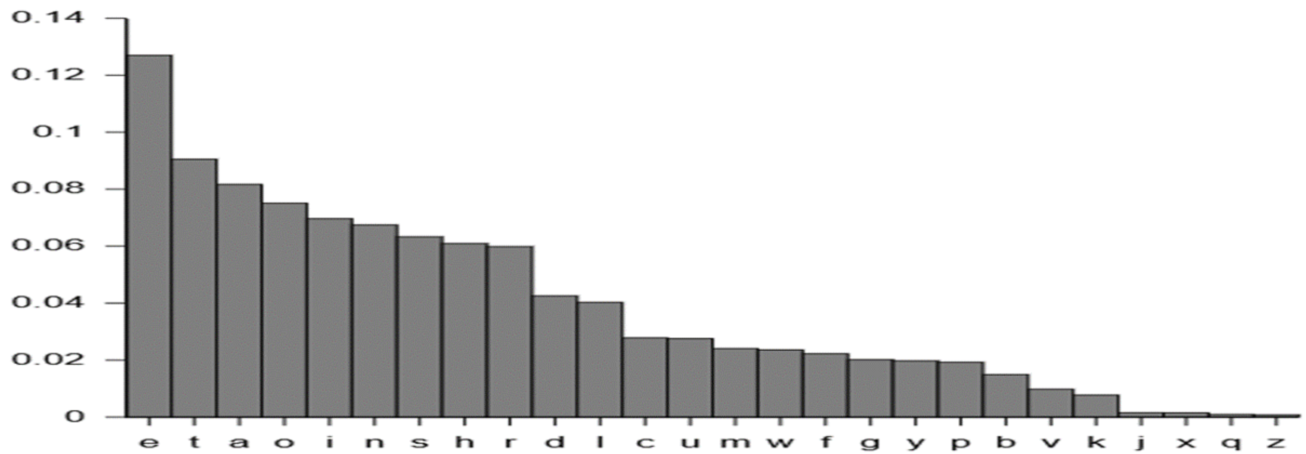
Why Vigenère cipher can be cracked?

English has “letter frequencies”

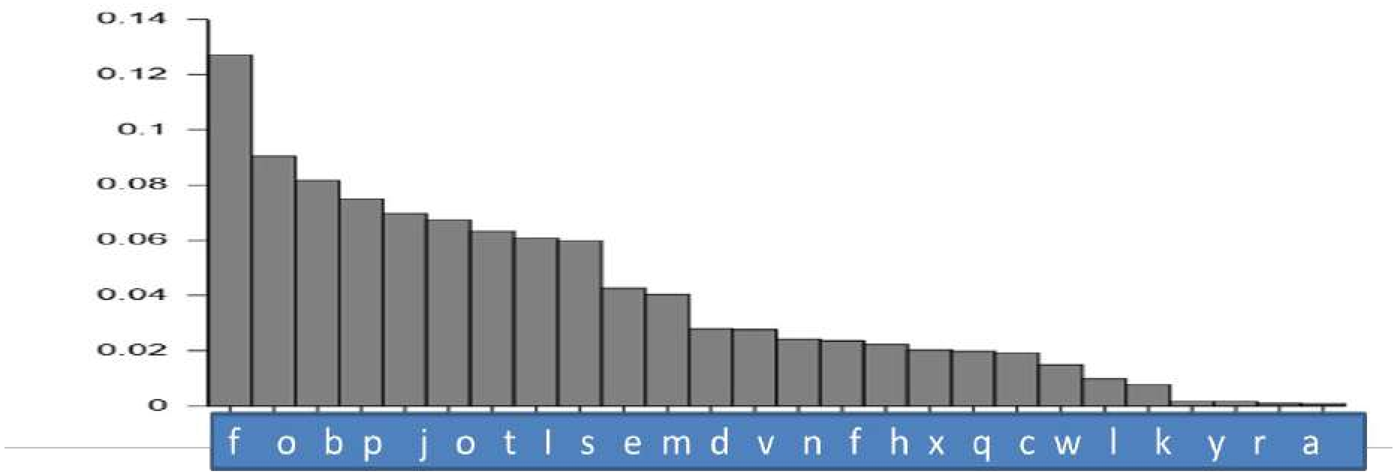


Letter frequencies before and after encryption using the same character are identical

Before encryption



After all letters are encrypted by character "b"



Crack the Vigenère cipher

- Two steps
  - Step 1: brute force the length of the key
  - Step 2: guess each character of the key

Step 1: Brute force the length of the key

- For each guessed key length, extract ciphertext sub-sequences

Ciphertext:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
v	p	t	n	v	f	f	u	n	t	t	h	t	a	r	p

If guessed key length is 2:

Sub-sequence 1 (1, 3, 5, 7...): v t v f n t t r ...

Sub-sequence 2 (2, 4, 6, 8...): p n f u t h a ...

If the guess of the key length is correct, all characters in each sub-sequence should be encrypted by the same character in the key.

Step 1: Brute force the length of the key (cont')

Ciphertext:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
v	p	t	n	v	f	f	u	n	t	t	h	t	a	r	p

If guessed key length is 3:

Sub-sequence 1 (1, 4, 7, 10...): v n f t t p...

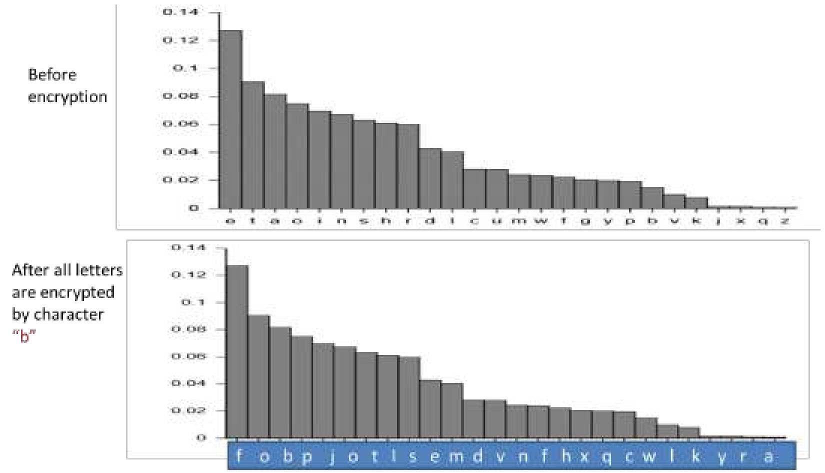
Sub-sequence 2 (2, 5, 8, 11...): p v u t a ...

Sub-sequence 3 (3, 6, 9, 12...): t f n h r ...

Step 1: Brute force the length of the key (cont'')

- How to know if the guess of the length is correct or not?
  - If the length guess is correct and the ciphertext is long enough, the distribution of the "letter frequencies" in the extracted sub-sequences should be similar to the letter of frequencies of English.

Because the letters of an extracted sub-sequence are encrypted by the same character of the key



Step 1: Brute force the length of the key (cont”’)

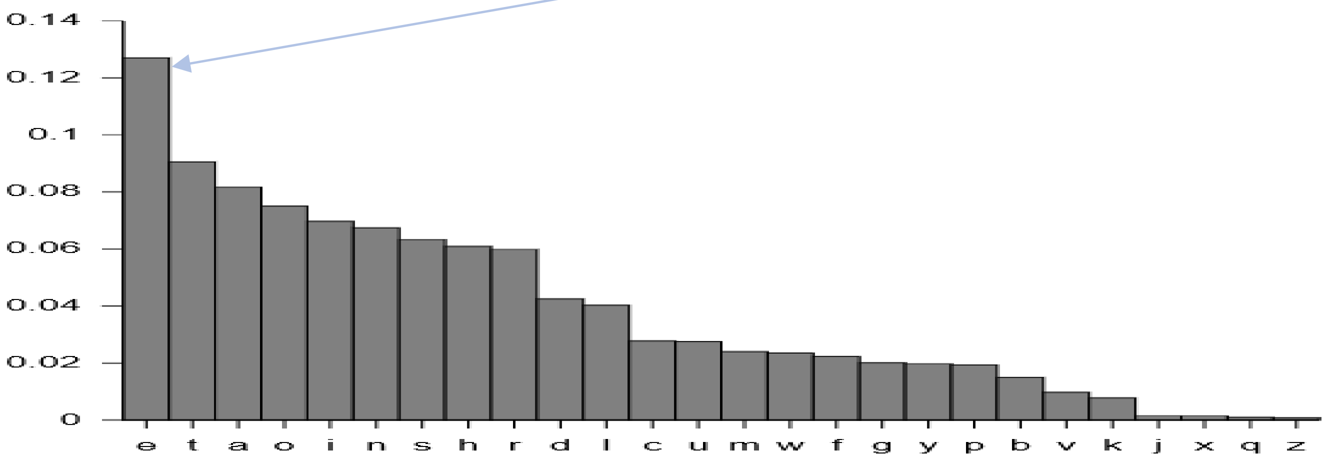
- For each guess of the length of the key
  - Extract ciphertext sub-sequences
  - For each ciphertext sub-sequence
    - Calculate similarities of the “letter of frequencies” of the extracted sub-sequences and “letter of frequencies” of English
    - Calculate the average value of the similarities of all sub-sequences
- Among all the guessed lengths, choose the length with the highest average similarity

Step 1: Brute force the length of the key (cont''')

If attacker guesses key length is 2:	Letter of frequency similarity index
Sub-sequence 1 (1, 3, 5, 7...): v t v f n t t r ...	0.8
Sub-sequence 2 (2, 4, 6, 8...): p n f u t h a p ...	0.9
	Average: 0.85
If attacker guesses key length is 3:	
Sub-sequence 1 (1, 4, 7, 10...): v n f t t p...	0.4
Sub-sequence 2 (2, 5, 8, 11...): p v u t a ...	0.5
Sub-sequence 3 (3, 6, 9, 12...): t f n h r ...	0.3
	Average: 0.4
If attacker guesses key length is 4:	Average: 0.2
...	...
If attacker guesses key length is 16:	Average: 0.3

Step 2: Guess each character of the key

- First, extract sub-sequences based on the guessed key length
- Then, find out the most frequently used character in the sub-sequence.
- For example, if ‘f’ is the most frequently used character in a sequence, ‘f’ is most likely encrypted from the letter ‘e’
- The character in the key to encrypt ‘e’ to ‘f’ is ‘f’ – ‘e’ = ‘b’



Step 2: Guess each character of the key (cont')

Ciphertext	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	v	p	t	n	v	f	f	u	n	t	t	h	t	a	r	p

- If we know from the first step that the key length is 2:

Sub-sequence 1 (1, 3, 5, 7...): v t v f n t t r ...

The most popular letter is t, we guess that t is encrypted from e. The first key character is t - e = p

Sub-sequence 2 (2, 4, 6, 8...): p n f u t h a p ...

The most popular letter is p, we guess that p is encrypted from e. The second key character is p - e = l

So, the key is "pl"

NTNU

One Time Pad (OTP)

Developed during WW1, used in WW2 (and since)

Plaintext (m):	0	1	0	0	1	1	1
Key (k):	1	0	1	0	0	1	1
Ciphertext (c):	1	1	1	0	1	0	0

$\oplus$

$\oplus$ truth table

Inputs		output
0	0	0
0	1	1
1	0	1
1	1	0

Encryption $c = E(m, k) = m \oplus k$

Decryption $D(c, k) = c \oplus k = (m \oplus k) \oplus k = m$

OTP is secure but has limitations

- OTP is proven to be perfectly secure, if used correctly

Perfect secrecy (informal definition): Observing the ciphertext should not change the attacker's knowledge about the distribution of the plaintext.

- Limitations
  - Key must be (at least) as long as the plaintext
  - Key can only be used once

Misuse of OTP

- Use the same key to encrypt two or several messages
- No longer perfectly secure!

- $C_1 = E(m_1, k) = m_1 \oplus k$
- $C_2 = E(m_2, k) = m_2 \oplus k$

- Eavesdropper does:

  $C_1 \oplus C_2 = \mathbf{(}m_1 \oplus k\mathbf{)} \oplus \mathbf{(}m_2 \oplus k\mathbf{)} \longrightarrow m_1 \oplus m_2$

- $m_1 \oplus m_2$ reveals m_1, m_2 information

NTNU

m_1 XOR m_2 reveals m_1, m_2

C1 = m1 ⊕ k 11…

C2 = m2 ⊕ k 10…

C1 ⊕ C2 = m1 ⊕ m2 $\boxed{01}$01 0000

- Letters all begin with 01
- ⊕ two letters gives 00
- Space (0010 0000) character begins with 00
- ⊕ of a letter and the space character gives 01

m1, m2: one is the space character and another is a letter

0101 0000 = 0010 0000 (space) ⊕ 0111 0000 (‘p’)

m1, m2: one is the space character and another is the letter ‘p’

Letter	ASCII Code	Binary
a	097	01100001
b	098	01100010
c	099	01100011
d	100	01100100
e	101	01100101
f	102	01100110
g	103	01100111
h	104	01101000
i	105	01101001
j	106	01101010
k	107	01101011
l	108	01101100
m	109	01101101
n	110	01101110
o	111	01101111
p	112	01110000
q	113	01110001
r	114	01110010
s	115	01110011
t	116	01110100
u	117	01110101
v	118	01110110
w	119	01110111
x	120	01111000
y	121	01111001
z	122	01111010

OTP: No integrity protection

Plain	heilhitler
Key	wclnbtdefj
Cipher	DGTYIBWPJA

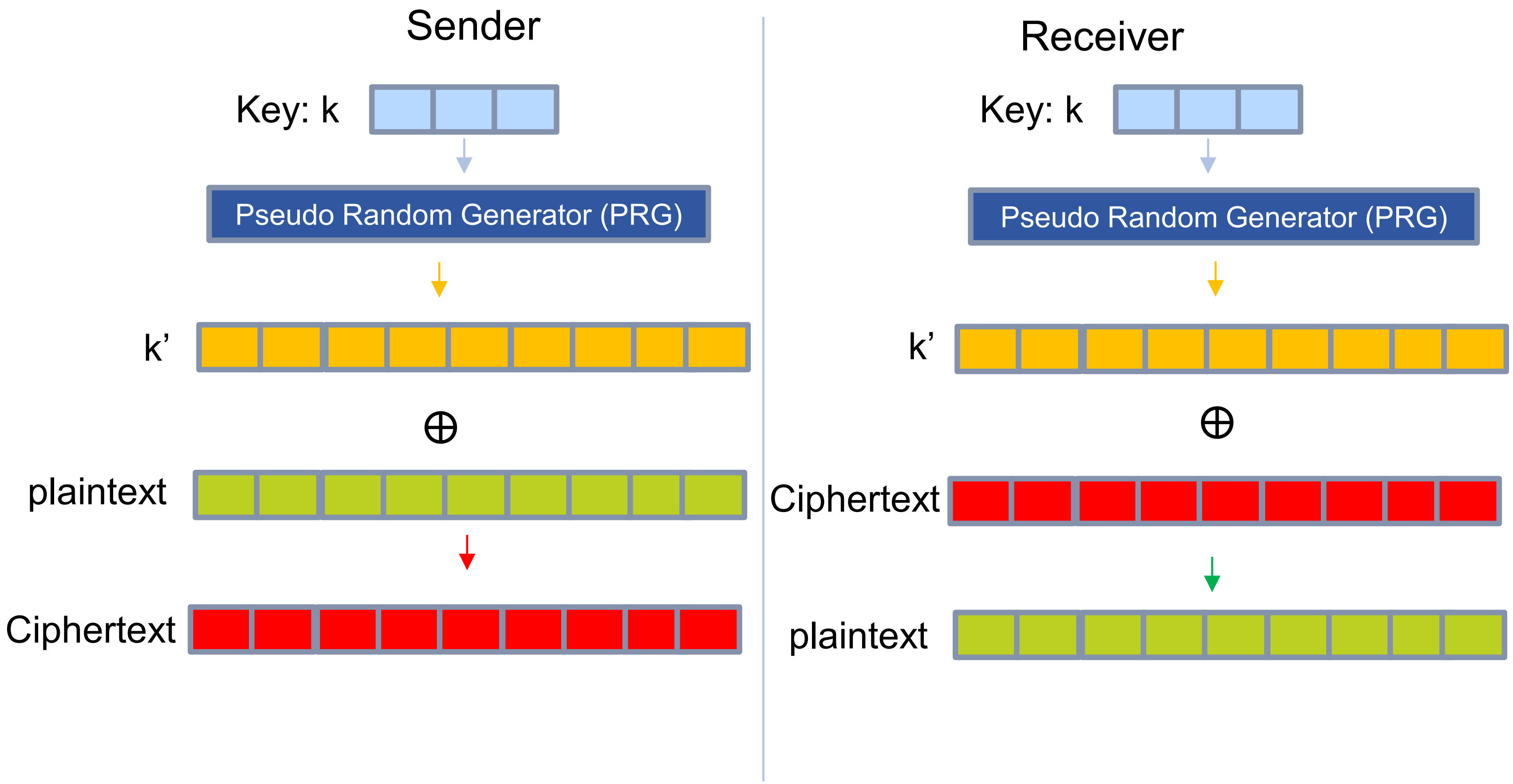
Cipher	DGTYIBWPJA
Key	wggsbtdefj
Plain	hanghitler

Cipher	DCYTIBWPJA
Key	wclnbtdefj
Plain	hanghitler

Stream ciphers

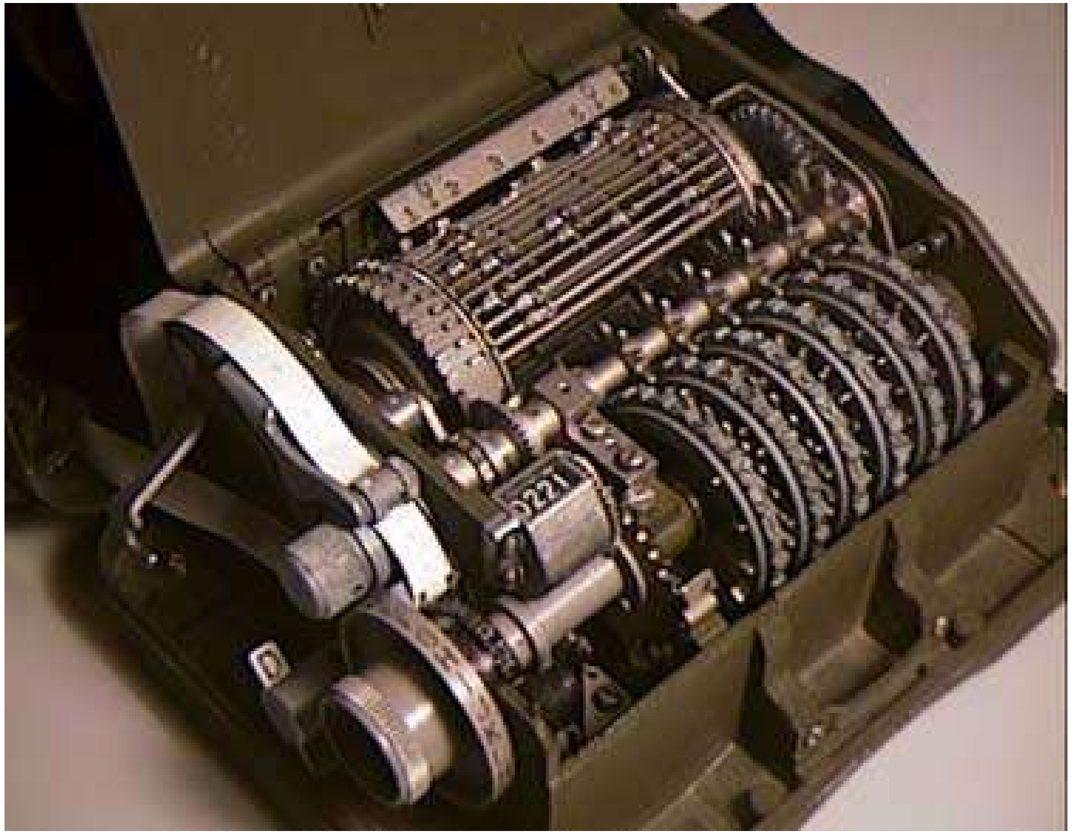
- Address the issue: OTP key is as long as the message
- Idea
  - Use a short key k as seed
  - Generate a pseudo random key k'
  - k' is as long as the message
  - Use k' for OTP encryption and decryption

Stream ciphers (cont’)



Like OTP, stream cipher key can be used only once

Old stream cipher machine

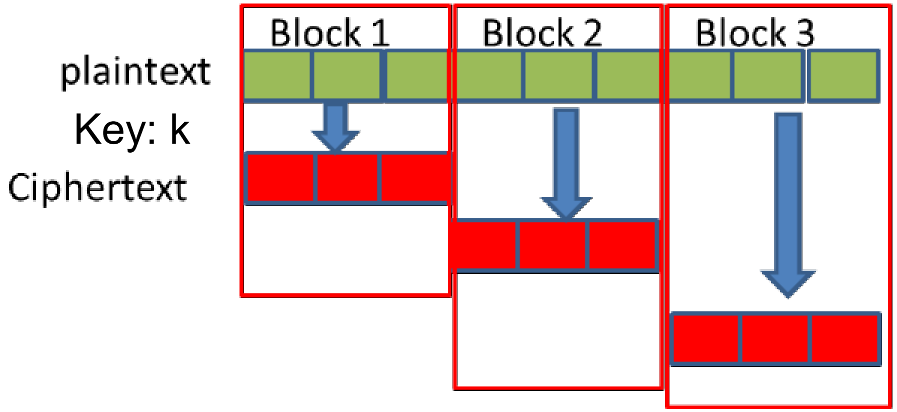


- The Hagelin M-209 is one of many stream cipher machines developed in the 1920s and 30s
- Used by US forces in WW2
- Over 140,000 machines were produced

https://en.wikipedia.org/wiki/M-209

Block cipher

- Another solution to address the OTP long key issue
- Cut the plaintext into small blocks and encrypt/decrypt each block using short keys



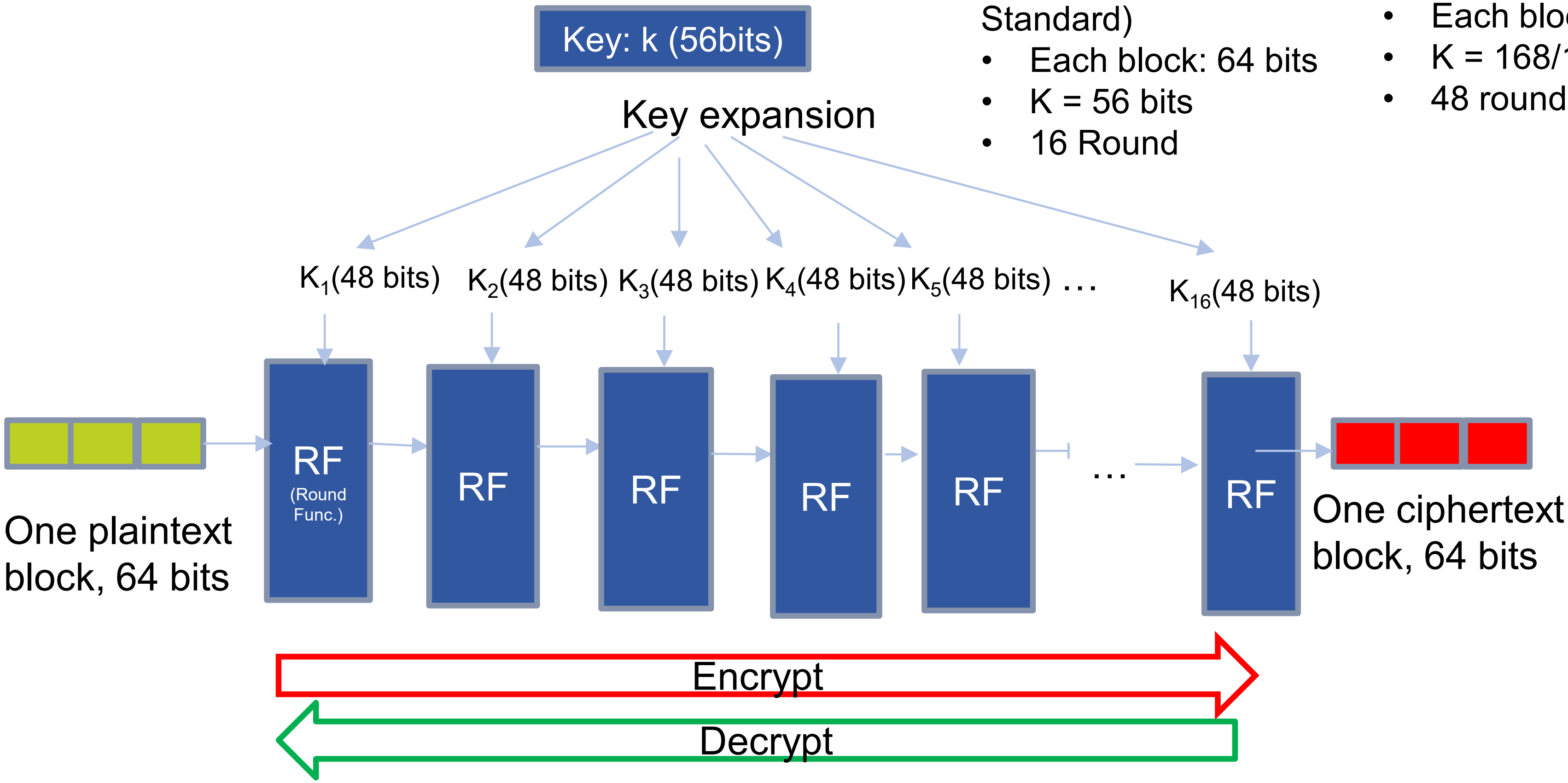
Encrypt / Decrypt each block

DES (Data Encryption Standard)

- Each block: 64 bits
- K = 56 bits
- 16 Round

3DES

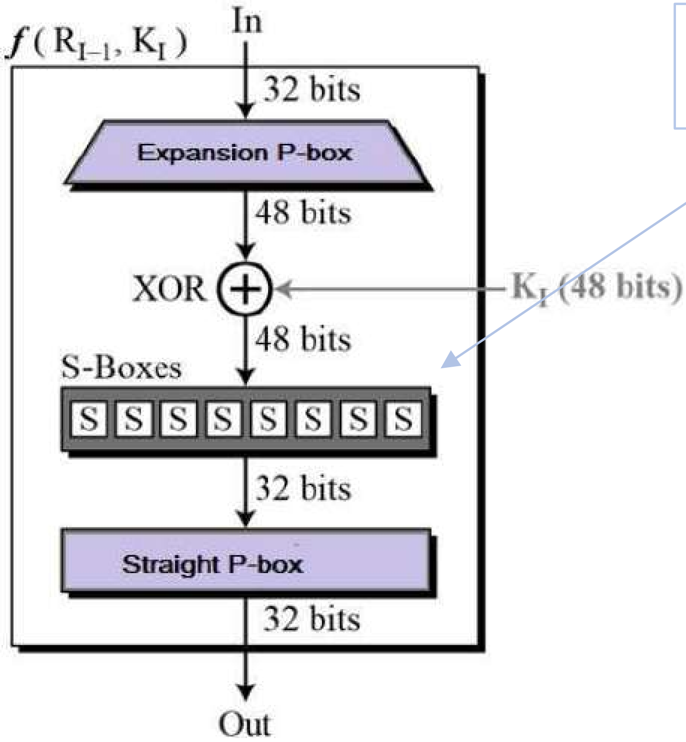
- Each block: 64 bits
- K = 168/112/56 bits
- 48 round



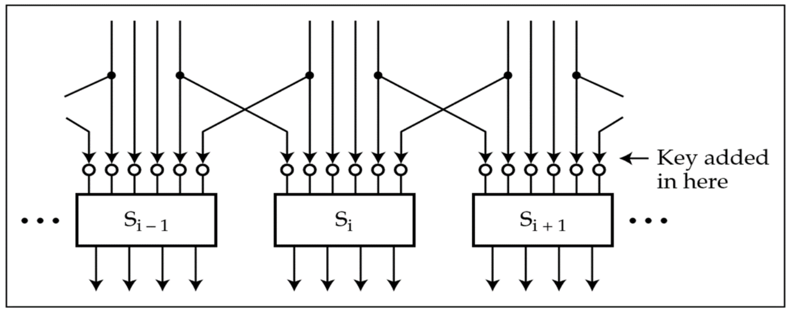
Need to define key expansion and reversible round function (RF)

DES Round Function (RF)*

- DES function applies a 48-bit key to the rightmost 32 bits to produce a 32-bit output. Then, rightmost 32 bits are swapped with leftmost 32 bits



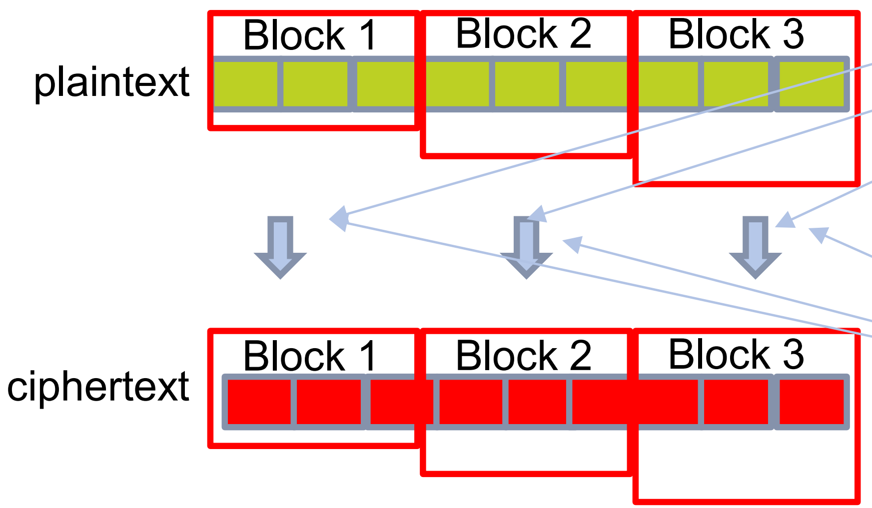
Each S-box takes a six-bit input and provides a four-bit output



*https://www.tutorialspoint.com/cryptography/data_encryption_standard.htm

Wrong block cipher mode of operation

- Mode of operation is the actual way to split messages to blocks and to chain the blocks
- Electronic code book (ECB) mode:



- All blocks use the same key sets generated from identical key and key expansion function
- The round function is also identical

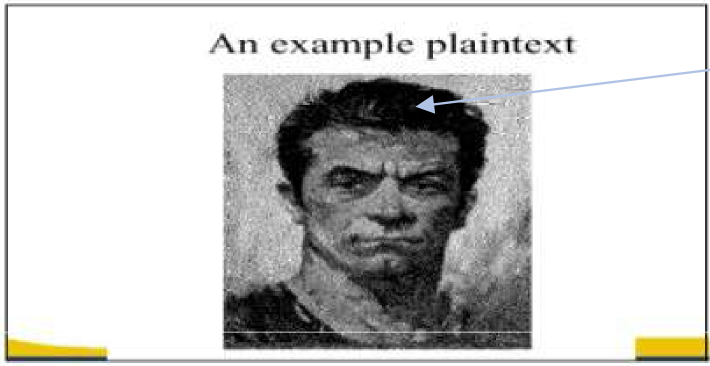
If plaintext in Block 1 = Block 2,

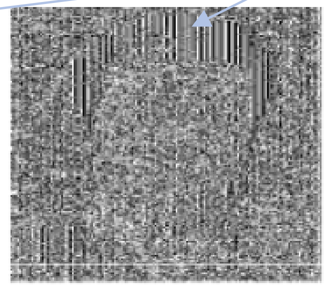
Then ciphertext in Block 1 = Block 2

Encryption using ECB

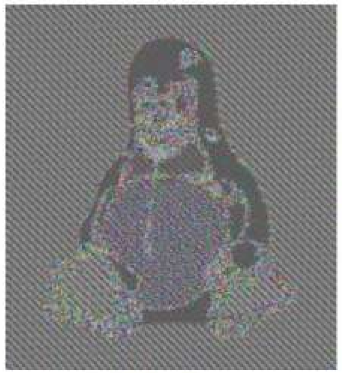
- Can disclose plaintext information

Encrypted using ECB





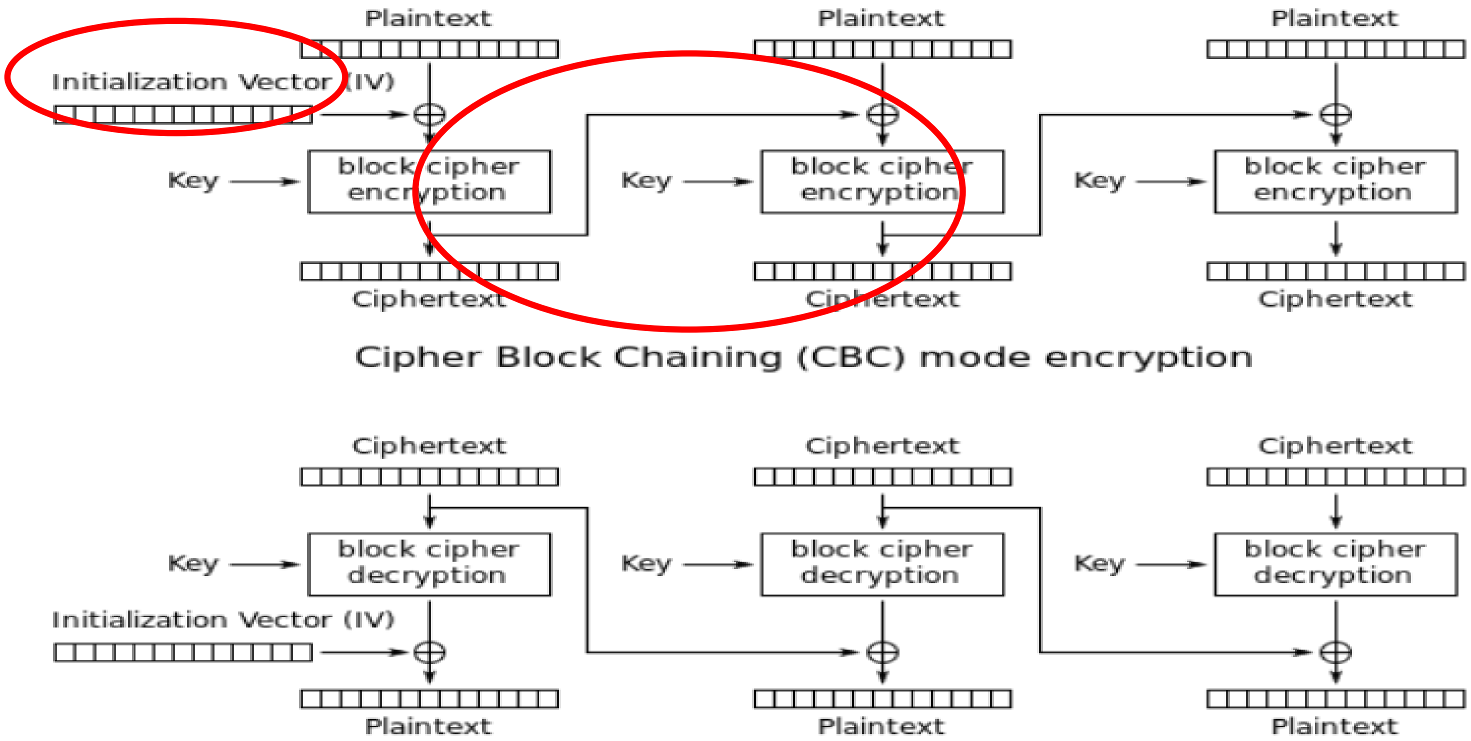
The hair shape of the man can still be seen!



(a) Plaintext (b) ECB ciphertext

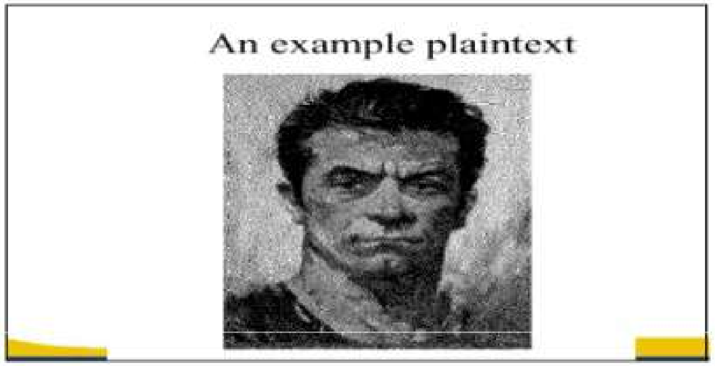
One correct mode of operation

- Cipher Block Chaining (CBC) mode with random Initialization Vector (IV)

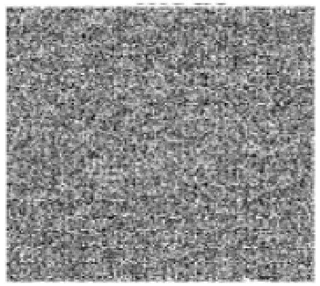


In addition to key, the IV (initialization vector) will be sent to the receiver for decryption

Encryption using Cipher Block Chaining



Encrypted using CBC



Block cipher examples

block ciphers	Creation date	lock size Key size, Rounds	Secure/insecure
DES (Data Encryption Standard)	Early 70’	Block size: 64 bits Key size: 56 bits Rounds: 16	The EFF's DES cracker (Deep Crack) breaks a DES key in 56 hours in 1998
3DES (Triple DES)	1998	Block size: 64 bits Key size: 56, 112, 168 bits Rounds: 48	NIST disallowed it after December 31st 2023. Slow especially in software
AES (Advanced Encryption Standard)	1998	Block size: 128 bits Key size: 128, 192, 256 Rounds: 10, 12, or 14	NIST replaced DES with AES in 2000 AES-128 for secret info. AES-256 for top secret info. Good performance in both software and hardware

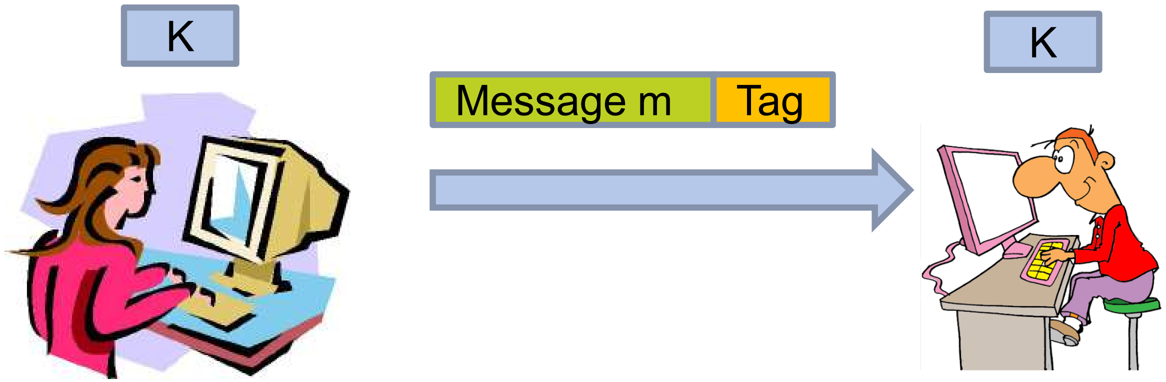
Stream cipher vs. block cipher

Stream cipher	Block cipher
Fast	Slow and requires more memory
Better for cases where the amount of data is either unknown or continuous, e.g., network streams	Better for cases when the amount of data is pre-known, e.g., a file, data fields, request/response protocols
Cannot provide integrity protection	Some can also provide integrity protection

Transmit data using shared secret key

- Confidentiality (No eavesdropping)
  - Substitution cipher
  - Shift cipher
  - The Vigenère cipher
  - One time pad
  - Stream cipher
  - Block cipher
- Integrity (No tampering)
  - ECBC
  - HMAC
- Combining confidentiality and integrity

Integrity: MAC (Message Authentication Code)



Alice

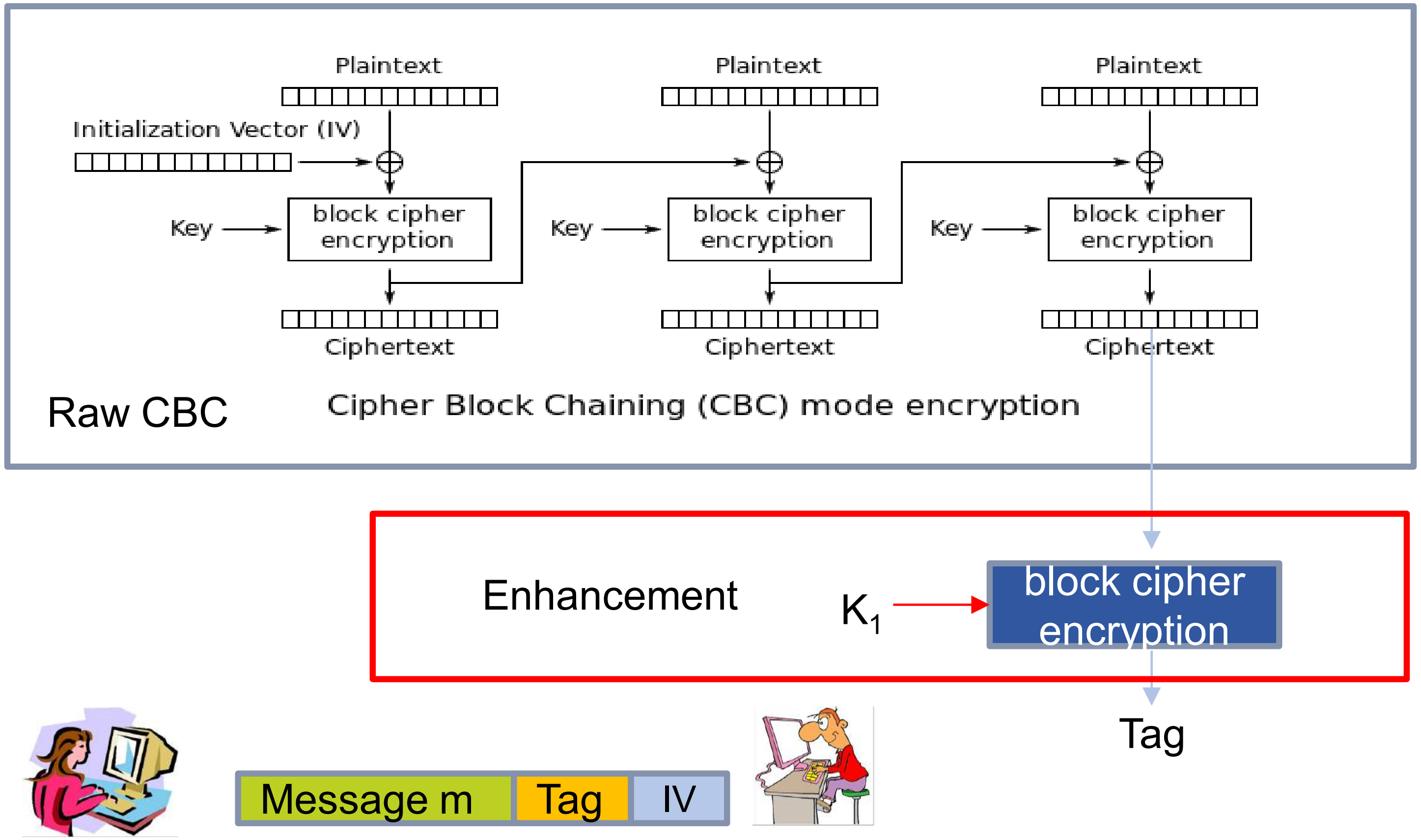
Generate tag:
tag <- G(K, m)

Bob

Verify tag:
V(K, m, tag) = ‘yes’?

Note: a key must be shared among Alice and Bob in advance

Block cipher – Enhanced CBC mode



Alice

Generate tag:
tag <- G(K, K_1, m, IV)

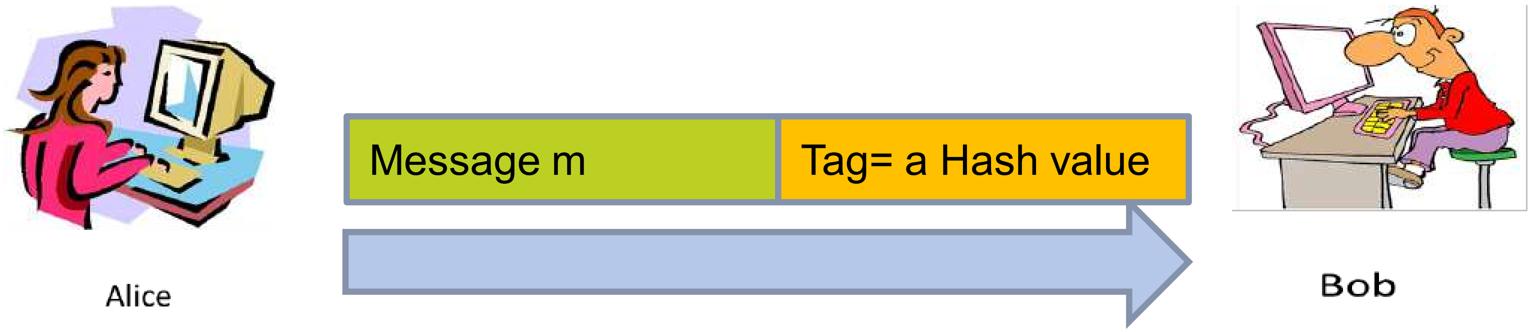
Bob

Verify tag:
V(K, K_1, m, tag, IV) = ‘yes’?

Hash Functions

- A hash function distills a message M down to a hash h(M)
- Desirable properties include:
  1. Preimage resistance – given X, you can’t find M such that h(M) = X
  2. Collision resistance – hard to find M1, M2 such that h(M1) = h(M2)

HASH-MAC (HMAC)



Generate tag:
tag <- G(K, m)

Verify tag:
V(K, m, tag) = ‘yes’?

E.g., Using SHA-256 (output is 256 bits) as Hash
tag = Hash(K $\oplus$ opad, Hash (K $\oplus$ ipad, m))

ipad: Inner pad
opad: Outer pad

MAC function examples

Hash functions	Creation date	Output size	Secure/insecure
		Enhanced CBC MAC	
AES-CMAC	2006	128 bits	Achieves the similar security goal of HMAC
		Hash MAC (HMAC)	
HMAC-MD5	1991	128 bits	Severely compromised lot of collisions have been found
HMAC-SHA1 (Secure Hash Algorithm 1)	1993	160 bits	Is known to be broken
HMAC-SHA2 (Secure Hash Algorithm 2, sometimes called SHA-256)	2001	224, 256, 384, 512 bits	Better security than SHA1
HMAC-SHA3 (Secure Hash Algorithm 3)	2015	Same as above	**Even more secure**

Transmit data using shared secret key

- Confidentiality (No eavesdropping)
  - Substitution cipher
  - Shift cipher
  - The Vigenère method
  - One time pad
  - Stream cipher
  - Block cipher
- Integrity and authenticity (No tampering)
  - ECBC
  - HMAC
- Combining confidentiality and integrity

Strategies to combine confidentiality and integrity

- MAC-then-Encrypt (e.g. TLS)
- Encrypt-and-MAC (e.g. SSH)
- Encrypt-then-MAC (e.g. IPsec)

Encrypt the message to ciphertext and then calculate MAC from the ciphertext

https://crypto.stackexchange.com/questions/202/should-we-mac-then-encrypt-or-encrypt-then-mac

https://medium.com/@c0D3M/lucky-13-attack-explained-dd9a9fd42fa6

Secure communication has two steps

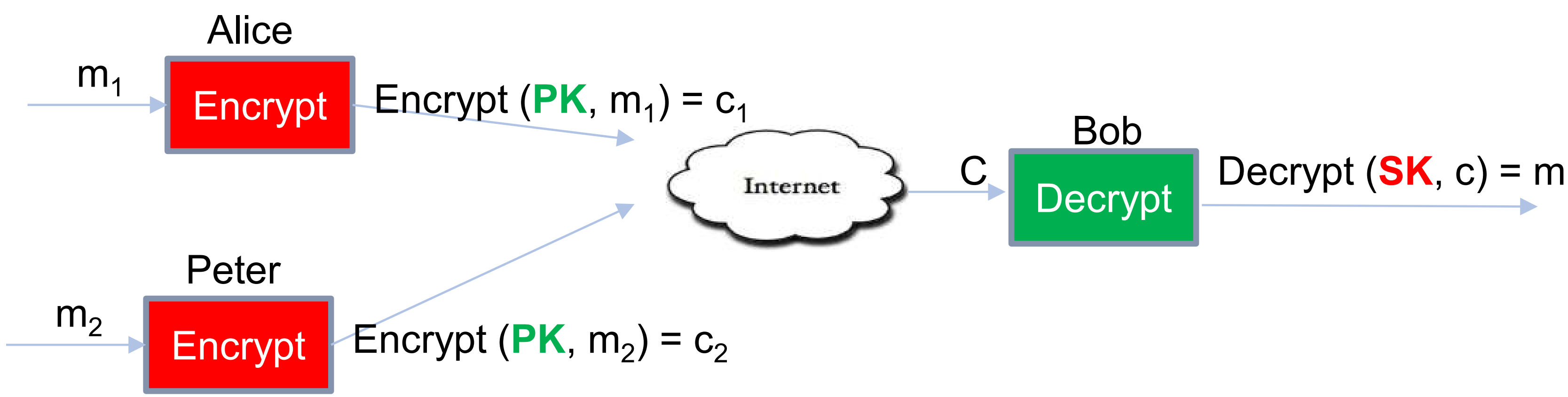
- Step 1: Establish shared secret key through, e.g.,
  - Handshake algorithms
- Step 2: Transmit data using shared secret key

To understand handshake algorithms

- We need to understand
  - Public key encryption
  - Digital signature
  - Certificate Authority (CA)
- SSL (Secure Sockets Layer)/TLS (Transport Layer Security 1.2) handshake

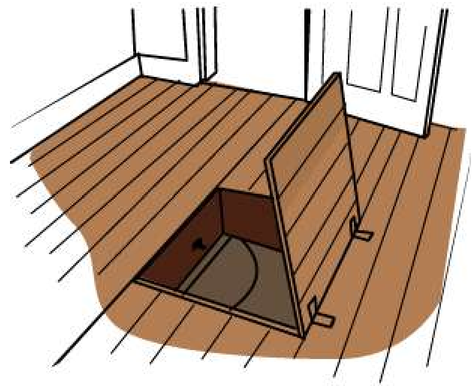
Public key encryption

- Encryption and Decryption use different keys
  - PK – Public Key
  - SK – Secret/Private key



Building blocks of public key encryption

- Algorithm KeyGen: output PK and SK
- One-way trapdoor function F (e.g., RSA, 1977)
  - Computing y = F(PK, x) is easy
  - **One-way**: given y, finding x is difficult
- A function F^{-1}
  - F^{-1} (SK, y) = x



Why is it called trapdoor?
If you have the secret key, suddenly, inverting the function F becomes easy

Digital signature

- Traditional signature
  - Sign document with a pen, stamp or seal
  - Signature is to bind document with author
  - Does not apply to the digital world
    - The attacker can copy & paste Alice's signature from one doc to another
- Digital signature
  - Signature depends on the content of the document

Bob signs the document

- Bob first hashes* document m
- Bob signs the Hash(m) using his secret key SK_{Bob} and F^{-1} (here, F is the trapdoor function)

$\boxed{F^{-1} (SK_{Bob}, \text{Hash}(m))}$ ➡ Signature

Why hash?* https://en.wikipedia.org/wiki/Digital_signature

Alice verifies the signature

- Alice receives document m from Bob
- Alice receives the Signature
- Alice receives Bob's public key PK_{Bob}
- Alice wants to know if the document m is the one signed by Bob

Check if $F(PK_{Bob}, \text{Signature}) = \text{Hash}(m)$?

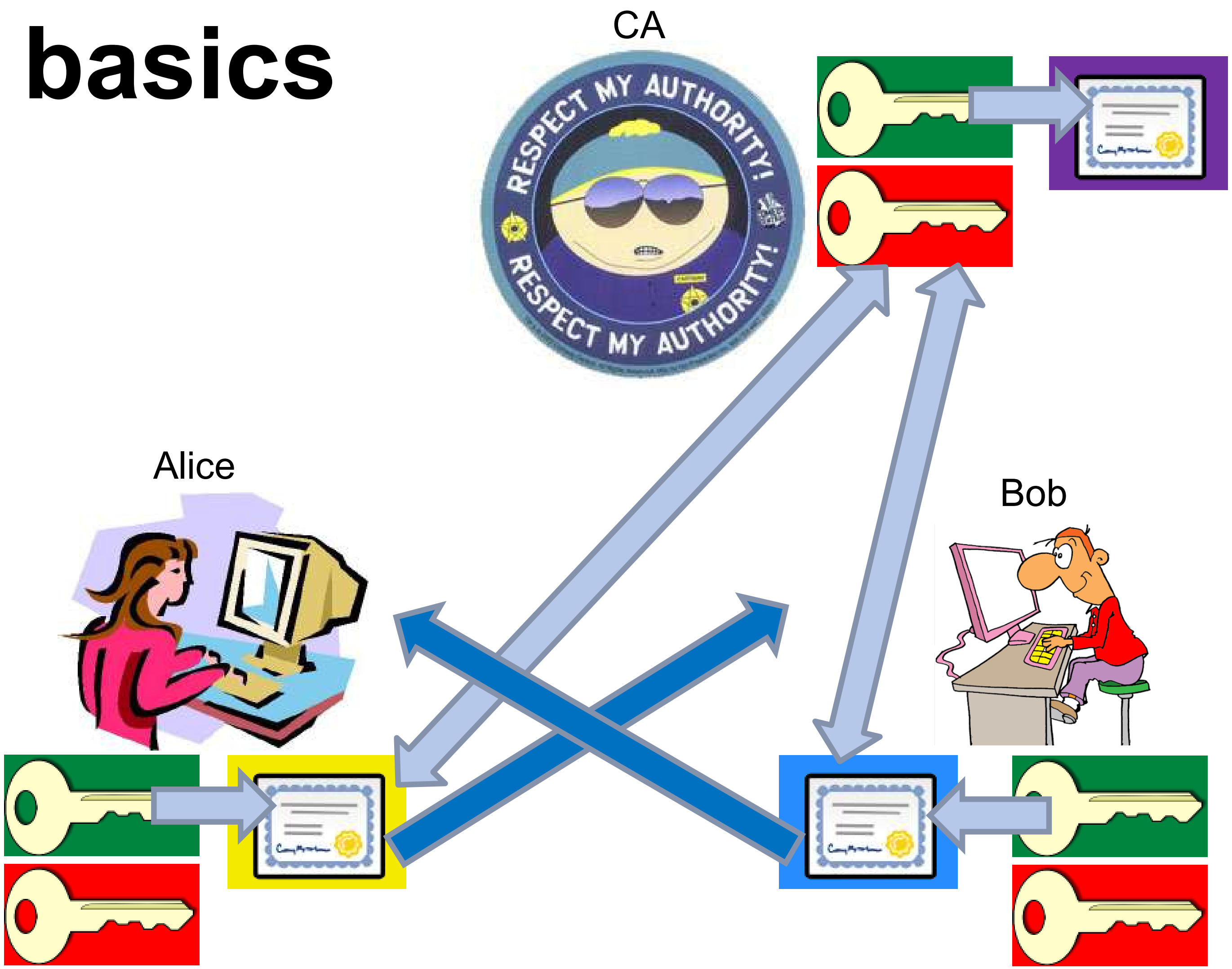
- Yes: the document is signed by Bob
- No: the document is not signed by Bob

Certification from CA

- Next question
  - How to send Bob’s public key to Alice securely?
  - In other words, if Alice receives a public key, how can she know that it is Bob’s public key, not a public key issued by an attacker?

We need help from a third party called Certificate Authority (CA)

PKI basics



A certificate example from CA

```
Certificate:
   Data:
       Version: 1 (0x0)
       Serial Number: 7829 (0x1e95)
       Signature Algorithm: md5WithRSAEncryption
       Issuer: C=ZA, ST=Western Cape, L=Cape Town, O=Thawte Consulting cc,
               OU=Certification Services Division,
               CN=Thawte Server CA/emailAddress=server-certs@thawte.com
       Validity
           Not Before: Jul  9 16:04:02 1998 GMT
           Not After : Jul  9 16:04:02 1999 GMT
       Subject: C=US, ST=Maryland, L=Pasadena, O=Brent Baccala,
                OU=FreeSoft, CN=www.freesoft.org/emailAddress=baccala@freesoft.org
       Subject Public Key Info:
           Public Key Algorithm: rsaEncryption
           RSA Public Key: (1024 bit)
               Modulus (1024 bit):
                   00:b4:31:98:0a:c4:bc:62:c1:88:aa:dc:b0:c8:bb:
                   33:35:19:d5:0c:64:b9:3d:41:b2:96:fc:f3:31:e1:
                   66:36:d0:8e:56:12:44:ba:75:eb:e8:1c:9c:5b:66:
                   70:33:52:14:c9:ec:4f:91:51:70:39:de:53:85:17:
                   16:94:6e:ee:f4:d5:6f:d5:ca:b3:47:5e:1b:0c:7b:
                   c5:cc:2b:6b:c1:90:c3:16:31:0d:bf:7a:c7:47:77:
                   8f:a0:21:c7:4c:d0:16:65:00:c1:0f:d7:b8:80:e3:
                   d2:75:6b:c1:ea:9e:5c:5c:ea:7d:c1:a1:10:bc:b8:
                   e8:35:1c:9e:27:52:7e:41:8f
               Exponent: 65537 (0x10001)
   Signature Algorithm: md5WithRSAEncryption
       93:5f:8f:5f:c5:af:bf:0a:ab:a5:6d:fb:24:5f:b6:59:5d:9d:
       92:2e:4a:1b:8b:ac:7d:99:17:5d:cd:19:f6:ad:ef:63:2f:92:
       ab:2f:4b:cf:0a:13:90:ee:2c:0e:43:03:be:f6:ea:8e:9c:67:
       d0:a2:40:03:f7:ef:6a:15:09:79:a9:46:ed:b7:16:1b:41:72:
       0d:19:aa:ad:dd:9a:df:ab:97:50:65:f5:5e:85:a6:ef:19:d1:
       5a:de:9d:ea:63:cd:cb:cc:6d:5d:01:85:b5:6d:c8:f3:d9:f7:
       8f:0e:fc:ba:1f:34:e9:96:6e:6c:cf:f2:ef:9b:bf:de:b5:22:
       68:9f
```

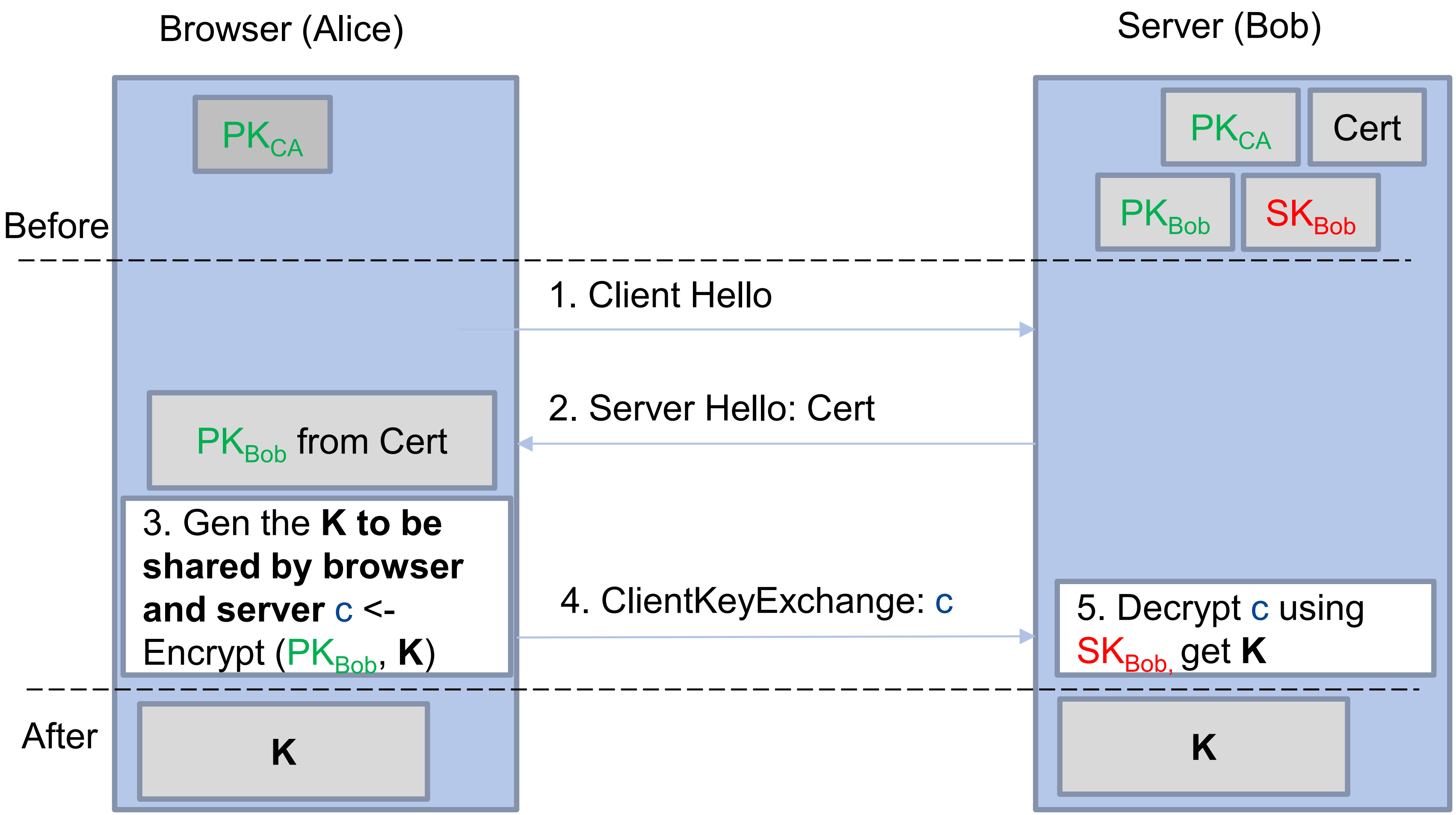
CA

Bob

Bob's Public key

CA's signature

Simplified SSL/TLS 1.2 handshake



TLS 1.2 is insecure

- Can be compromised
  - Raccoon attack

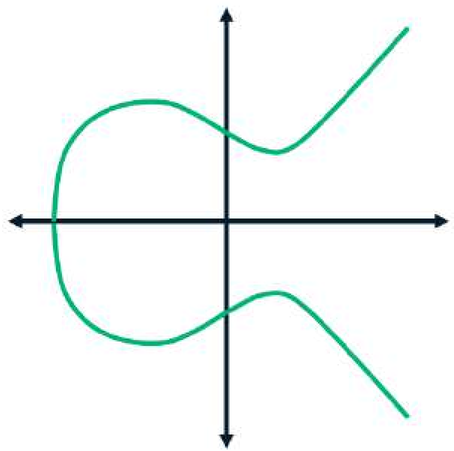
- TLS 1.3* was released in 2018. It should be used whenever possible
  - Faster, better, more secure

* https://www.youtube.com/results?search_query=TLS+1.3

RSA vs ECDSA

- 1977: **R**ivest-**S**hamir-**A**dleman
- 1995: Standardized
- Simple, effective
- Widely used in SSL/TSL, coins, …
- Prime Factorization

- 1985: Koblitz and Miller (ECC)
- 1999: Standardized
- Higher complexity, faster
- Shorter keys (+curve)
- Limited support
- ($y^2 = x^3 + ax + b$)



Quantum technology!

Quantum technology in brief

Quantum
Computing

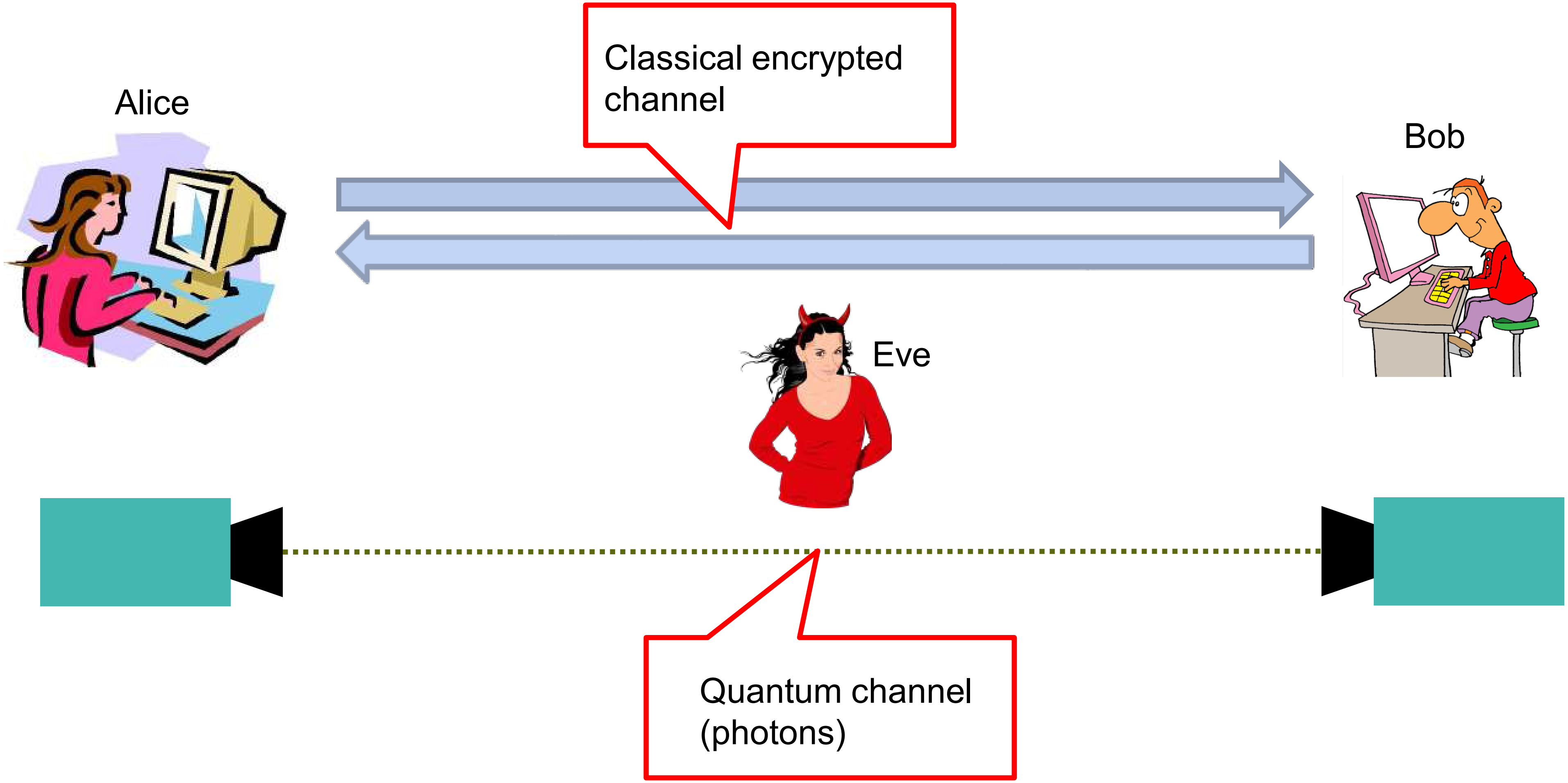
Quantum
Sensing

Quantum
Communication

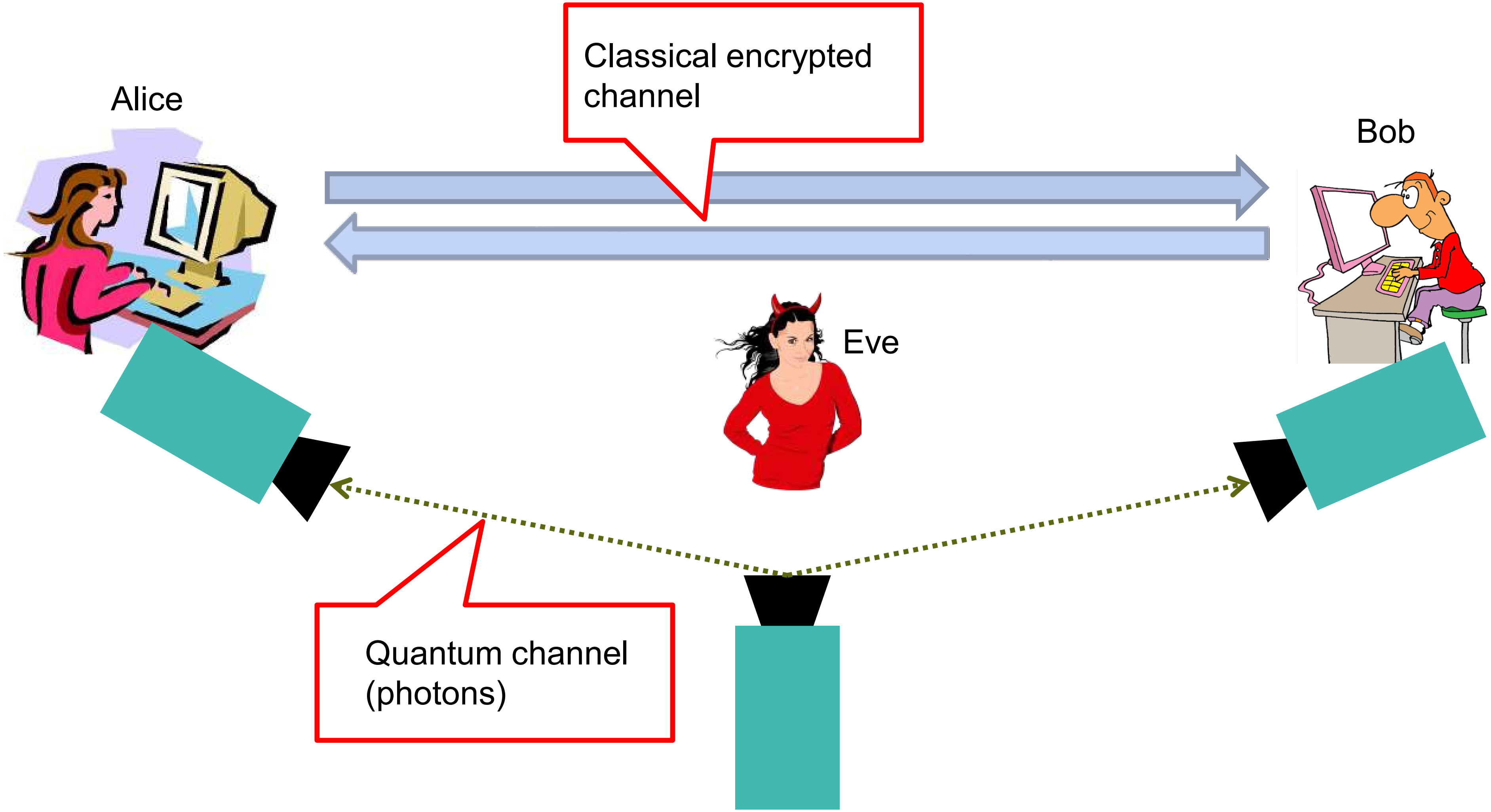
Post-Quantum
Cryptography

…not quantum tech

Quantum key distribution



Quantum key distribution



Post-Quantum Cryptography (PQC)

- Aka quantum {-safe, -secure, -resistant, -proof} cryptography

- FIPS 203, ML-KEM (Module-Lattice-Based Key-Encapsulation Mechanism)
  - General purpose encryption, small keys
- FIPS 204, ML-DSA (Module-Lattice-Based Digital Signature Algorithm)
  - Digital signatures
- FIPS 205, SLH-DSA (Stateless Hash-Based Digital Signature Algorithm)
  - Backup for FIPS 204
- (FIPS 206, FN-DSA (Fast-Fourier Transform over NTRU-Lattice-Based Digital Signature Algorithm))
  - Not released yet

Cryptography

- Is
  - The basis for many security mechanisms
  - Reliable when implemented and used properly
- Is not
  - The solution to all security problems (e.g., SQL injection)
- Cryptography is something you should NOT try to invent yourself

NTNU

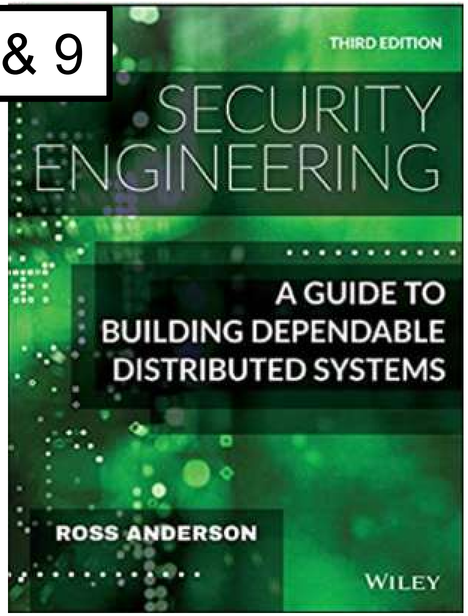
Kerckhoff's principle

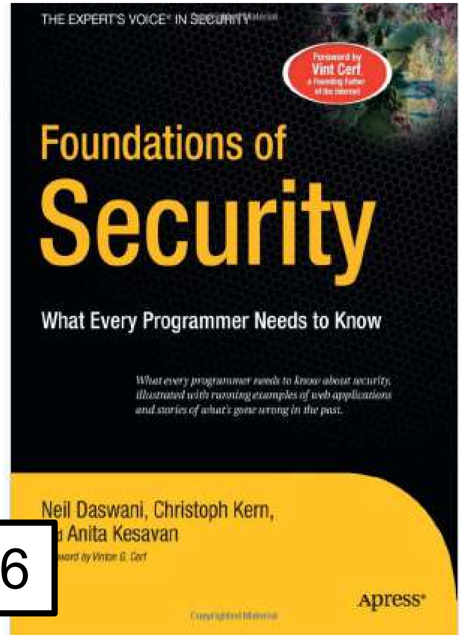
- The encryption algorithm is open
  - The only secret is the key
  - The key must be chosen at random, kept secret
- Because
  - Easier to change key than to change the algorithm
  - Standardization and public validation

Next week

- Authorization and multi-level security
- Authentication and single sign-on
- Control hijacking attacks

Ch 6 & 9





Ch 6