

# Cybersecurity



WILEY

# Syllabus 2022

- 1. Information security
- 2. Symmetric encryption
- 3. Data integrity
- 4. Public key encryption
- 5. Digital signatures and PKI
- 6. CSPRNG (Cryptographically Secure Pseudo-Random Number Generators)
- 7. Authentication
- 8. Secret sharing
- 9. Access control
- 10. Secure protocols (tls, ipsec, ssh)
- 11. Firewalls
- 12. Email security
- 13. Application security
- 14. Authentication again
- 15. CTF (Capture The Flag)
  - Coppa
  - d'Amore
  - But log is the official reference

## information security:

- toutes decide next hop, netmask... util for firewall !!
- security on patching: there is a problem, deliver a patch for fixing the problem.
- security by design: when design you have to consider in a explicit way the security issues, parameter that make to deliver a correct product.
- open security: everybody can see the measure you are undertaking, security mostly on secret, but on the strength of algorithm of encryption and so on ... . opposite is security by obscure.

security  $\Rightarrow$  cryptography is necessary not sufficient

$\leftarrow$   
the requirement is something that can exist or not exist, Ex.: a requirement in information security is confidentiality, not always for example for announcement on paper is not need.

CIA (confidentiality, integrity, availability) are the most popular requirement in information security.

safety: we don't want to have any issue, but against incident that are unintentional. (flooding, fire, ...)

security: secure against intentional incident where there is an attack/adversary, a person that try to get something that is not you, host, ...

WE FOCUS ON SECURITY !

• **Confidentiality:** we want that people not authorized to see some data, should not see the data. Typically we use **Encryption**: transforming some information that is in a plain text in a new information that is not understandable, safer text, like a function. It is a requirement, but not always necessary. Management decide if a information is confidential or not, not a choice of IT expert. They only implement that.

• **Integrity:** asking that the information that are you seeing is the same of the information stored in the disk or sent by the network. Information available to receiver is the same of the sender. Cryptography still work

• **Availability:** data should be available to the user always, the most famous attack versus availability is Dos (denial of service) attack, DDoS in distributed service. Availability is relate to safety more than security. (NON TRATTIAMO MOLTO)

there is other requirement like: **non repudial**, you can't say it was not me. (EX. DIGITAL SIGNATURE)

# Cryptography vs Security

Cryptography and Security differ

Cryptography deals with secrecy of information

Most real security deals with problems of fraud:

- Message modifications
- User authentication

Much of security has little to do with encryption however it might use cryptography

Almost invariably, encryption does not live alone without some form of *authentication*

Cryptography is necessary for security but not sufficient.

The operation of changing the identity of a user sending a message, file, ... → this operation done by an attacker is called spoofing, technically is changing the identity of packets that are carrying the communication (identity of the sender).

You can use spoofing for Dos attack, an indirect attack changing the sender.

man in the middle (MitM) very frequent attack where attacker intercept communication so is able to examine your information that you are sending and receive, also can change this information.

### Encryption:

- is a part of cryptography used for CONFIDENTIALITY.
- is changing information by means of a function, that can be inverted (Decryption)
- Encryption can be the function or Algorithm, for function  $E$
- We use encryption / decryption key →  $K$
- if  $K_e = K_d$  symmetrical encryption,  $K_e \neq K_d$  asymmetrical encryption
- $D_{K_2}(E_{K_1}(m)) = m$  always!

# Encryption: definitions

Encryption function (& algorithm):  $E$

Decryption function (& algorithm):  $D$

Encryption key  $k_1$

Decryption key  $k_2$

Message space (usually binary strings)

For every message  $m$ :  
 $D_{k2}(E_{k1}(m)) = m$

- Secret key      (Symmetric)  
 $k_1 = k_2$
- Public key      (Asymmetric)  
 $k_1 \neq k_2$

# Threat & Exploit

threat

- menace, something that is a source of danger

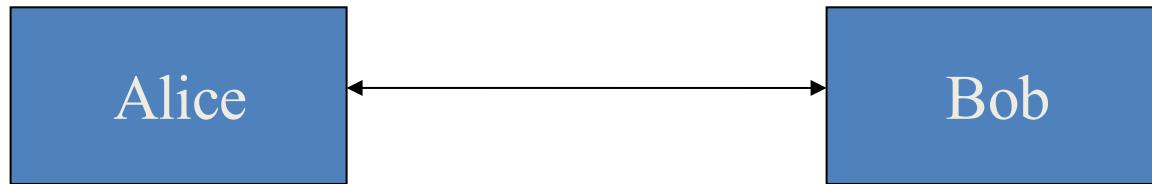
if there is a threat is not true that there is an attack, maybe.

exploit ("achievement", or "accomplishment")

is defining a procedure for attacker to obtain a successful attack.

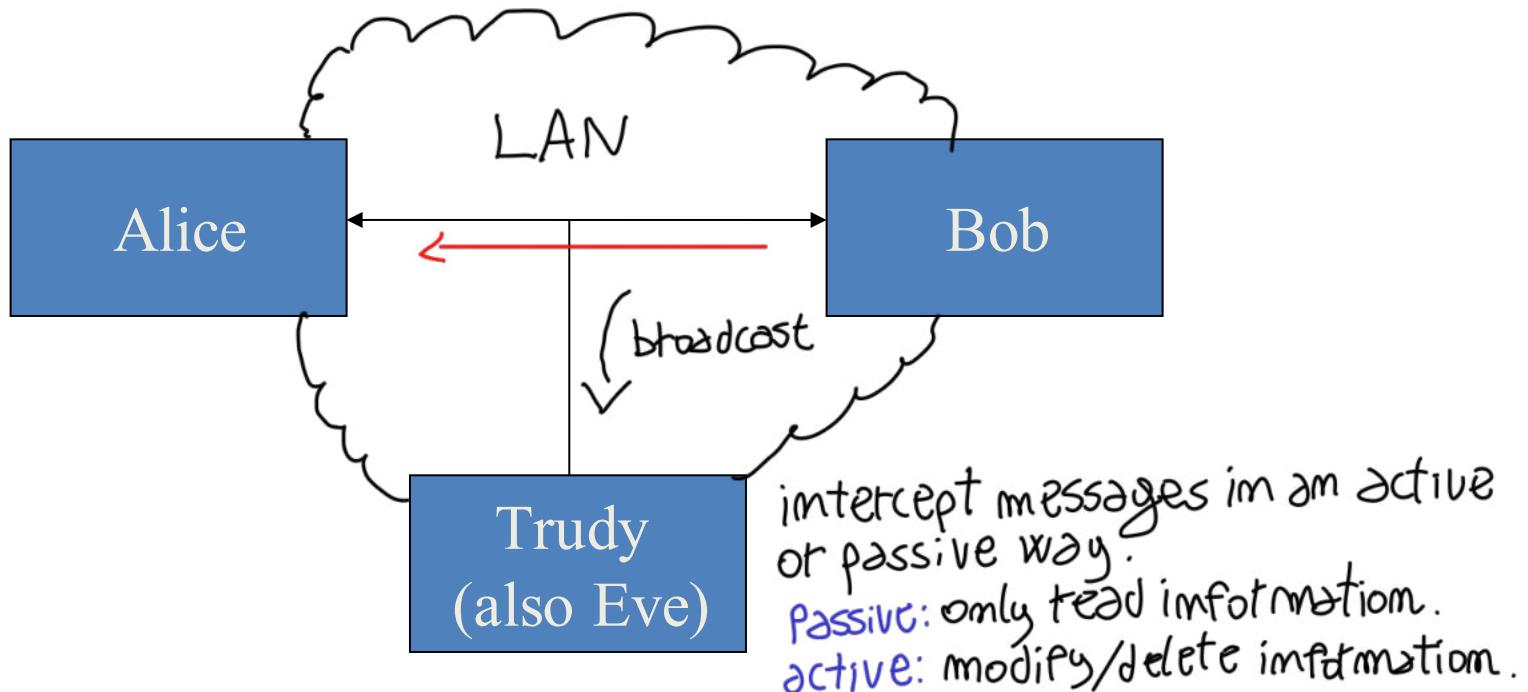
- software, chunk of data, or sequence of commands that take advantage of a vulnerability to cause unintended or unanticipated behavior to occur on computer (e.g., gaining control of a computer system, allowing privilege escalation, denial of service attack etc.)

# Communication Model

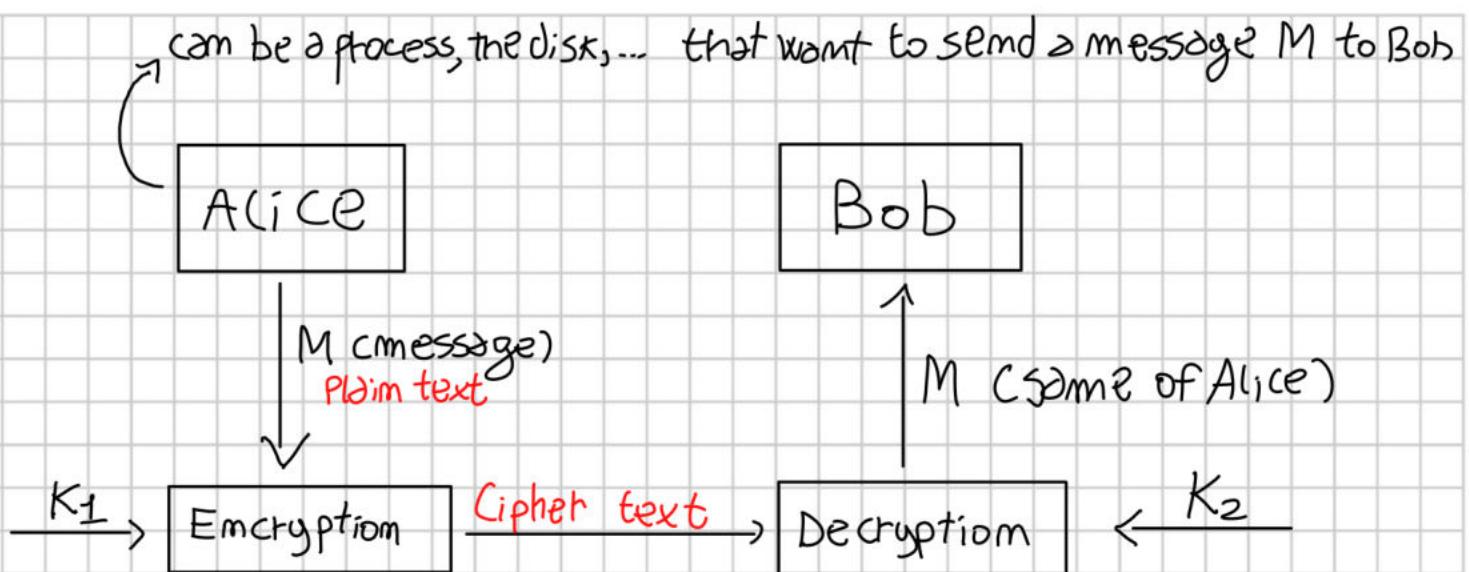


1. Two parties – Alice and Bob
2. Reliable communication line
3. Shared encryption scheme:  $E, D, k_1, k_2$
4. Goal: send a message  $m$  confidentially

# Threat (Attack) Model



4. Goal: send a message  $m$  confidentially



$K$  is the key, is a sequence of random bit, fundamental is the size  $|K|$ , typically is 128 bit. ( $2^{128}$  different keys) normally  $|K_1| = |K_2|$

if  $K_1 = K_2$  we have symmetric Encryption, the simplest and most popular.

M message in practice can be pre-processed.

Cipher text is an apparently random sequence of number, meaning nothing.

Encryption is a good generator of random number.

N.B.: An attacker is satisfied also with partial information! Saver text is easy to intercept with m.i.t.m., for example internal attack is very easy for obtain information, typically in a LAN.

threat like: spoofing, sniffing, Dos, ...

# Adversary

type of interception



## Passive

- reads the exchanged messages  
(no change)

## Active

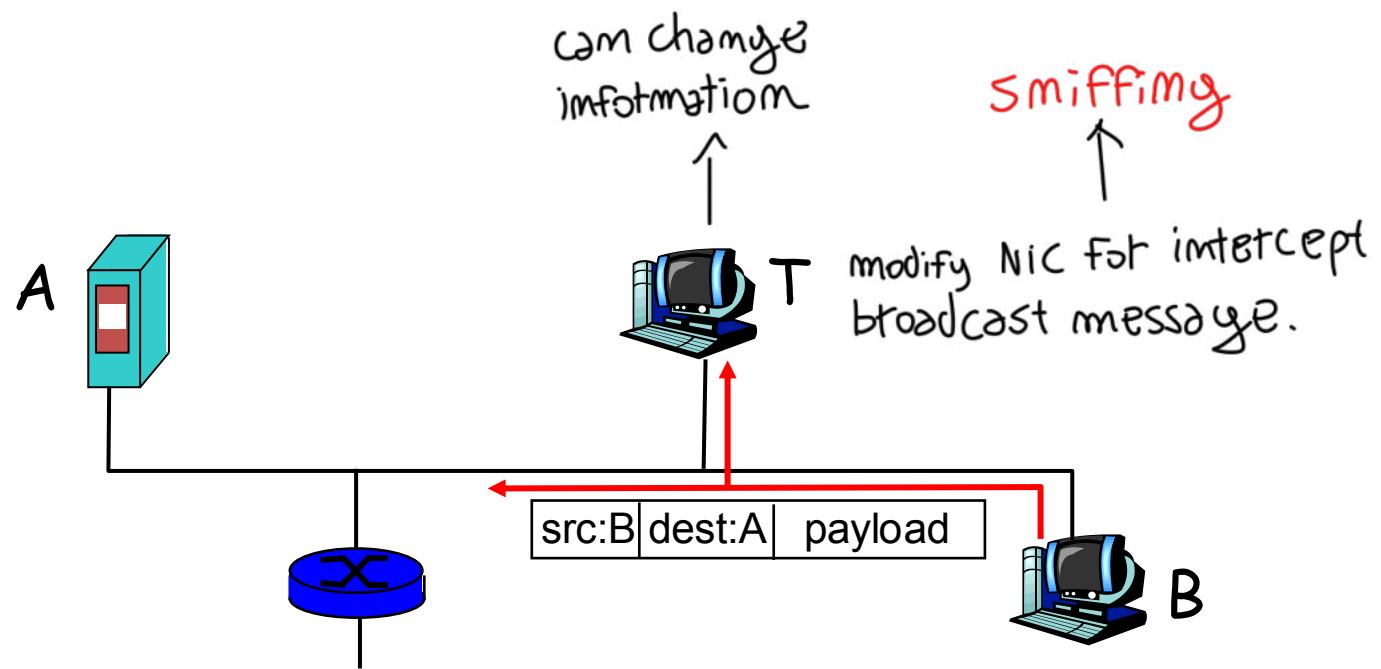
can modify messages sent by Alice or Bob  
can send false (fake) messages claiming  
that they have been sent by someone else  
(Alice or Bob)

# Passive adversary: *packet sniffing*

---

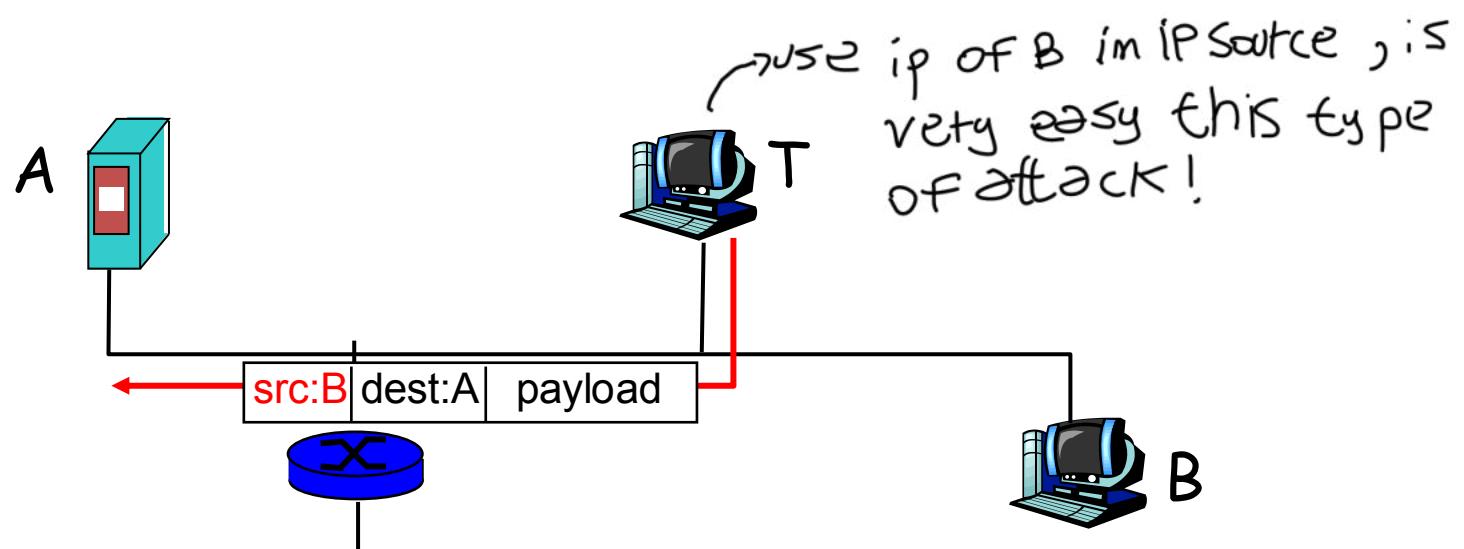
---

Trudy reads all messages exchanged by A and B



# Active adversary: IP spoofing

T is able to *forge* messages that look like messages sent by B (modification of IP header)

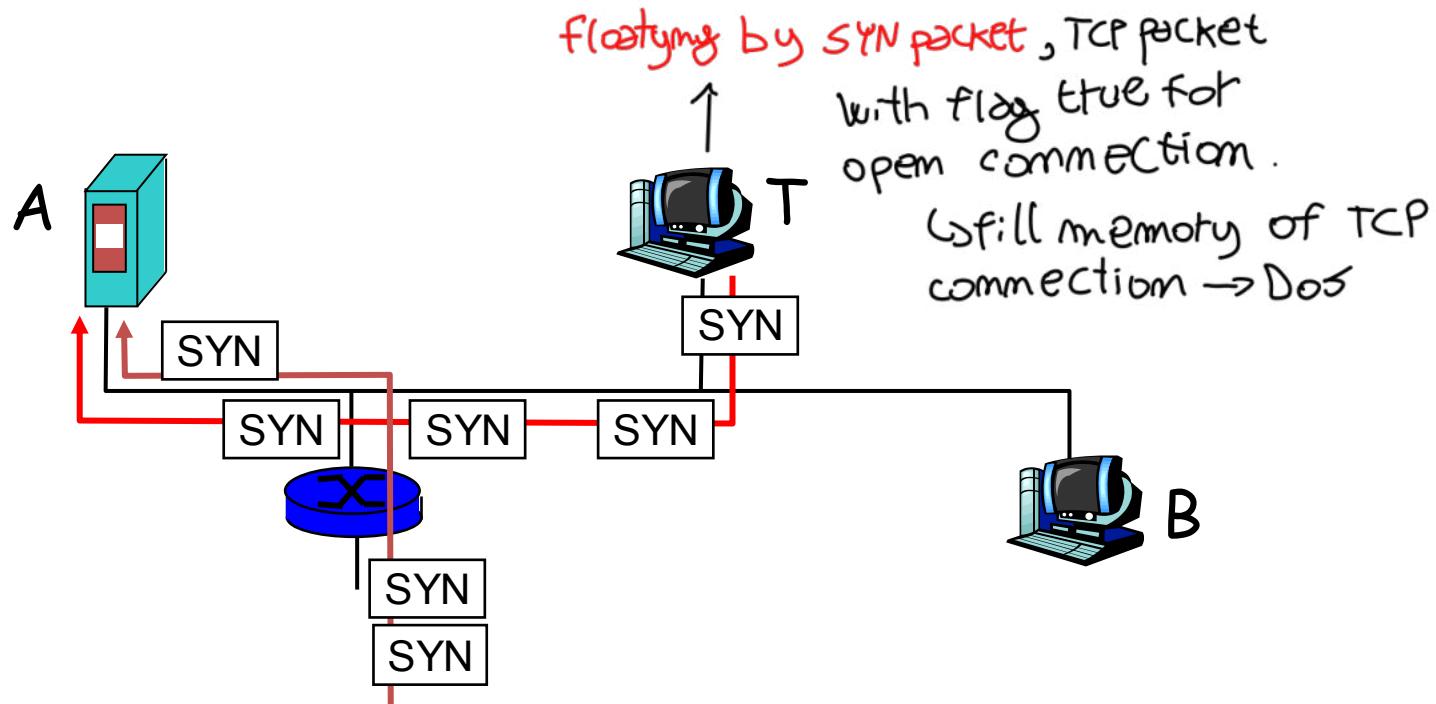


# Security Threat: Denial of Service (DoS)

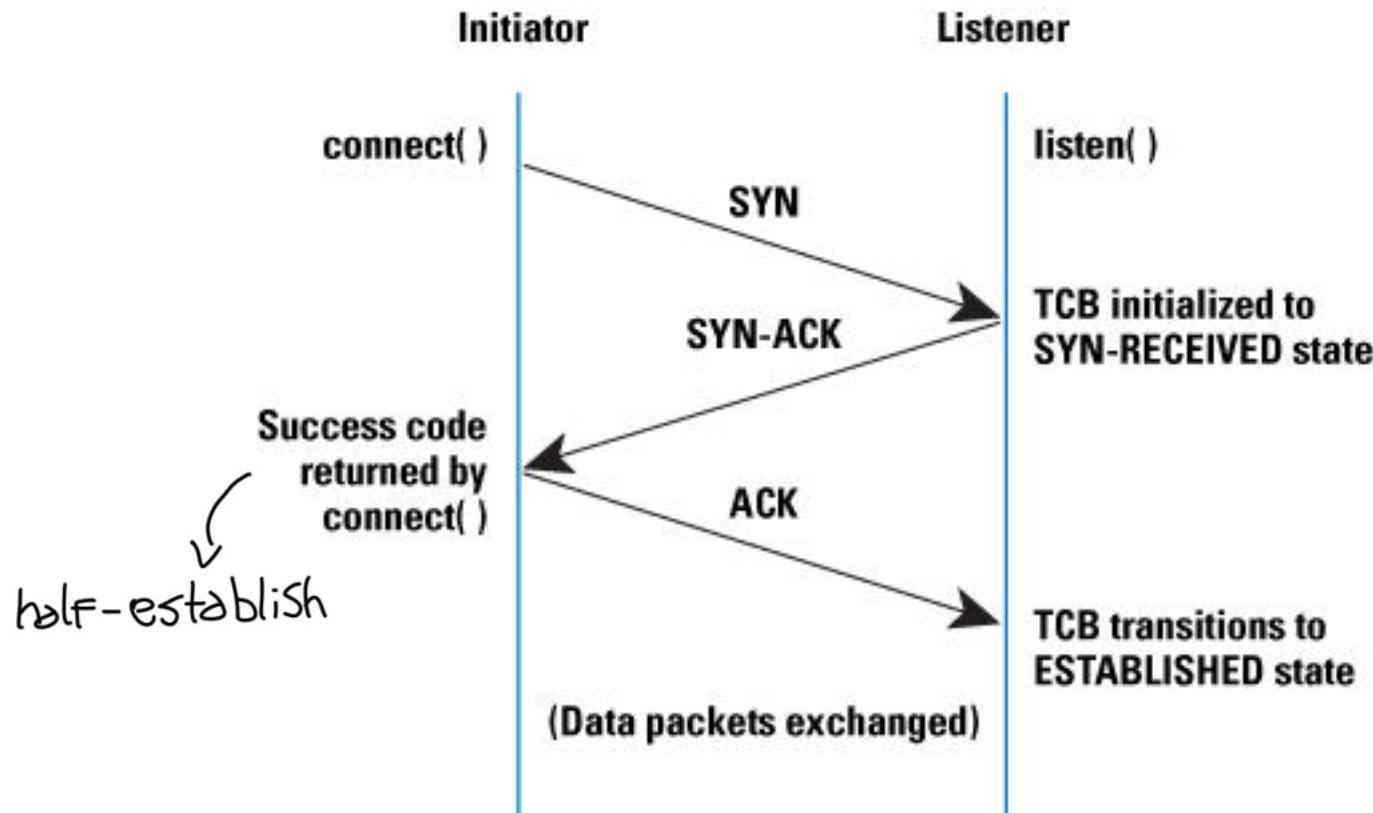
Attackers send many many packets to the attacked host

Distributed attack (DDoS, through infection of unaware computers)

- SYN packets are often used, why?



# TCP three way handshake



**open security**: the algorithm doing encryption and decryption are public, everything is based on the fact that keys are unknown to attacker.

E and D are known, K not known.

**Brute force attack**:

$|K| = l$  size of the key,  $2^l$  different keys

• DES is an old encryption algorithm, is weak by design. use a key  $|K|=56$ ,  $2^{56} \approx$  billion, that for a computer is a small number, using a B.F.A, trying all the combination is question of second to find the right key.

• it is important that  $|K|=l$  is big, but not too much.

• Some problems are **hard** for attacker: there are too many cases.

When K is big the complexity of algorithm is higher we lose efficient, encryption become very long.

The trade-off between  $|K|=l$  and energy consumption now is  $|K|=128/256$  bit.

**asymmetric encryption** is more demanding in term of consumers

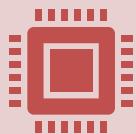
in internet a lot of security tecnics are based on the idea that some operations are very hard for attacker: like in https there is a number big  $N=m \times n$  product of two prime number, finding  $[m,n]$  given N is hard.

# Security goals



If the keys are unknown,  
then

it is hard to obtain even partial information on the message  
it is hard to find the key even if we know clear text



HARD = Computationally hard: it takes long time even if the most powerful computers are available

# Security goals



Possibilities:

No adversary can  
determine message  $M$  (not  
enough)  
  
Plain  
text

No adversary can  
determine some  
information about  $M$  (not  
enough)

No adversary can  
determine any *meaningful*  
information about  $M$  (good)



deterministic algorithm only run  
in exponential time, but probabilistic  
algorithm given the result with  
high probability running in polynomial  
time very often!



*Even in probabilistic sense*

this foul achieve security goal

# Adversarial model

Trudy attempts to discover information about  $M$

in open security  
Trudy knows the algorithms  $E, D$

dealing with binary strings  
Trudy knows the message space

Trudy has at least partial information about  $E_{k1}(M)$

Trudy does not know  $k_1, k_2$

can easily find the ciphertext

fundamentals of cryptography

# Additional definitions



Plaintext – the message prior to encryption  
("attack at dawn", "sell MSFT at 57.5")



Ciphertext – the message after encryption  
("ax4erkjpjepmm", "jhhfoghjklvhgbljhg" )

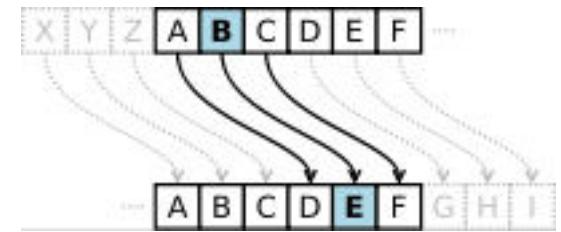


Symmetric key – encryption scheme where  $k_1 = k_2$   
(classical cryptography)

# Examples – bad ciphers

## Shift cipher (Caesar's cipher)

- 26 keys; easy to check them all
- conclusion: large key space required



Shift Letter of alphabet by  
an integer for cryptographys  
the message

## Substitution cipher

- large key space, but...

# Substitution cipher

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

## Example

- plaintext: attack at dawn
- ciphertext: waawoq wa vwmk

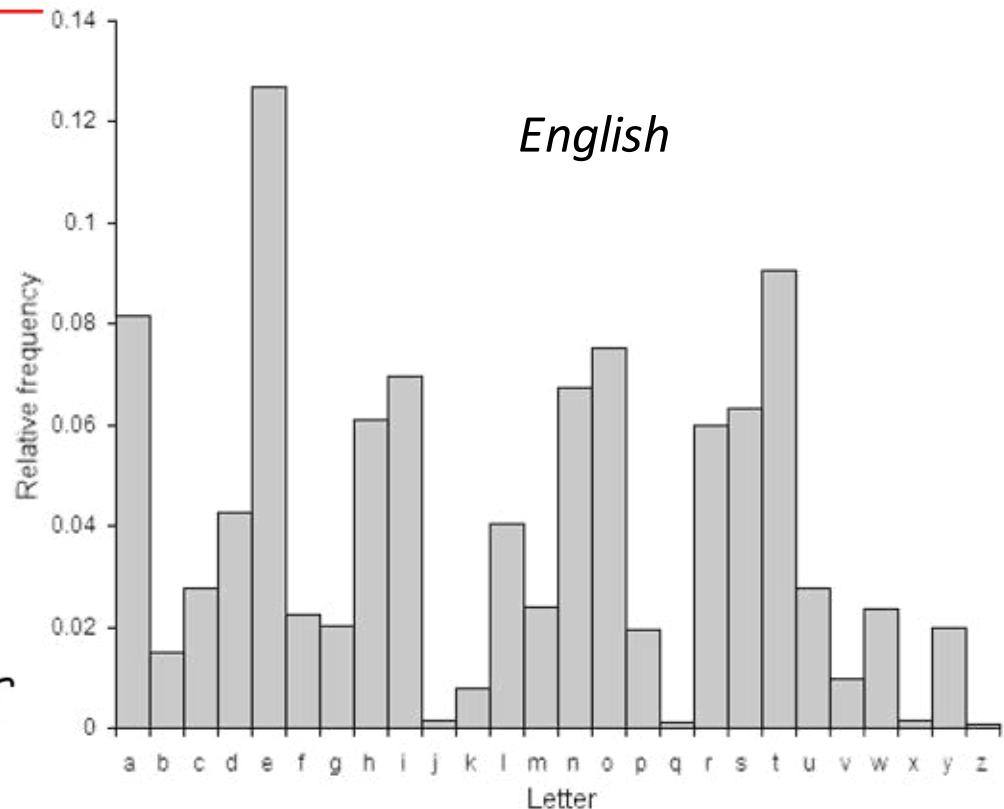
Size of key space:  $26! = 403291461126605635584000000$

$\sim 4.03 \times 10^{26}$  is it large enough? *No because of probability*

# Substitution cipher easily breakable

- in spite of huge size of key space, it is still “easy” to break thru statistical analysis of language (frequency analysis)

Even if the combination is a lot, there are letters more frequent than others in any language.



# Perfect Cipher

encryption algorithm

Plaintext space =  $\{0, 1\}^n$ ,  $D$  known  
only the space of binary strings

Given a ciphertext  $C$  the probability that exists  $k_2$  such that  $D_{k_2}(C) = P$  for any plaintext  $P$  is equal to the apriori probability that  $P$  is the plaintext.

before seeing ciphertext there are some probability  $P$  associated to the plain text, equal to the probability after seeing  $C$ !

In other words: the ciphertext does not reveal any information on the plaintext

$\Pr[P | C]$   $P$  conditioned to  $C$   
 $\Pr[\text{plaintext} = P \mid \text{ciphertext} = C] = \Pr[\text{plaintext} = P]$

in short:  $\Pr[P | C] = \Pr[P]$

Probabilities are over the key space and the plaintext space.

# Conditional probability

$$\Pr[P|C] = \Pr[P \cap C] / \Pr[C] = \Pr[P] \cdot \Pr[C]/\Pr[C] = \Pr[P]$$

$$\Pr[P | C] = \Pr[P \wedge C] / \Pr[C] \text{ (def. of cond. pr.)}$$

and (u)

$$\Pr[P \wedge C] = \Pr[P | C]$$
$$\Pr[C] = \Pr[C | P] \Pr[P]$$

(Th. Bayes)

- if P and C independent:  $\Pr[P \wedge C] = \Pr[P] \Pr[C]$

Hence, in a perfect cipher ( $\Pr[P | C] = \Pr[P]$ ):

$$\Pr[P] \Pr[C] = \Pr[C | P]$$
$$\Pr[P]$$

$$\Pr[C] = \Pr[C | P]$$

because plain text and cipher text are independent

# Example – One Time Pad

$\rightarrow$  is a perfect cipher on these conditions

AKA Vernam Cipher, invented in 1917 and patented in 1919 while *Gilbert Vernam* was working at AT&T

Plaintext space:  $\{0,1\}^n$

very difficult  $|K|$  is the same of Plain text  
Key space:  $\{0,1\}^n$

The scheme is symmetric, key K is chosen at random

$$E_K(P) = C = P \oplus K$$

↑  
XOR  
↓  
SAFER TEXT

$$D_K(C) = C \oplus K = P \oplus K \oplus K = P$$

$P \oplus 0 = P$

$\oplus$  : exclusive OR (bit by bit)

only because have the same space  $P \neq K$  !

## one time pad

plaintext | | bit

$\oplus$        $\oplus \sim \text{bit to bit XOR}$

key stream | |

=

ciphertext | |

→ is a key but very long, at least long as the plain text. 1.

## Properties of XOR:

	$\oplus$
0	0
0	1
1	0
1	0

$$P \oplus K = C$$

$$\text{we know that: } 1 \oplus \emptyset = 1$$

$$\emptyset \oplus \emptyset = \emptyset$$

Xoring with zero don't change the value

if we have this situation with O.T.P.:

$$P_1 \oplus K = C_1 \quad (K \text{ is the same})$$

$$P_2 \oplus K = C_2$$

if P<sub>1</sub> is public:

$$P_1 \oplus C_1 = P_1 \oplus \underbrace{P_1 \oplus P_2}_{\hookrightarrow 1 \oplus 1 = \emptyset} \oplus K = \emptyset \oplus K = K$$

$$\hookrightarrow 1 \oplus 1 = \emptyset$$

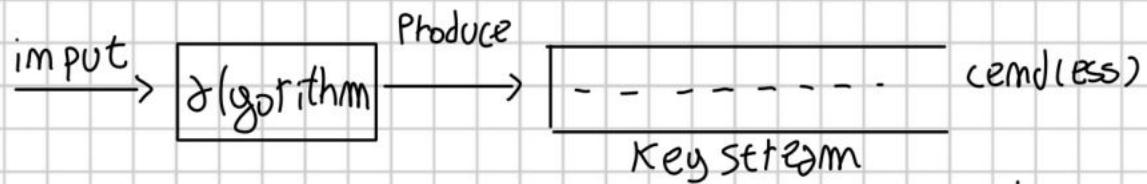
$$\emptyset \oplus \emptyset = \emptyset$$

K can easily be found by attacket.

is extremely dangerous to use the same keystream twice, we should never use again the same keystream, it should be any time different.

2.

- the keystream is somewhat produced for so many encryption, is a sequence of bits with no end,
- never reuse the same keystream.



- Bob and Alice not only must share the algorithm, but also the exact beginning of the usage of keystream
- Sharing a key is a very big problem.
- In open security the algorithm that produce the keystream is public, need to hide the input!
- The input can be a password, that should be secret. If you know the password the keystream can be easily defined.

### Key vs password:

- password by definition should be typeable, there are characters, symbols and so on (UTF encoding, ...). This means that having a password exactly long as a key: % ! # @ .. with some encoding have no sense, you can type a password with strange encoding.
- In key all possible combinations of bits are possible, they have, each bit, the same probability.
- password have less combinations and probabilities not the same

When O.T.P is a perfect cipher?

If you see the cipher text, this not changing what you think about the plain text.

The worst case is a keystream of all ones  $K=1 \dots 1 \dots$ , in this case OTP is not perfect because the safest text is just the complement of plain text. Also  $K=0 \dots 0 \dots 0 \dots$  is bad,  $C=P$  not absolutely perfect.

Keystream must be completely random then we have a good behavior, cipher text will be random with every plain text!

In many cases attacker can't understand if the key is correct or not by the plain text.

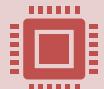
# Pros and Cons



Claim: the one time pad is a perfect cipher. [given a  $k$  bit cipher text every  $k$ -bit plain text has got same probability if key is random]



Problem: size of key space, as show by the following



Theorem (Shannon): A cipher cannot be perfect if the size of its key space is less than the size of its message space.



Why???

**Shannon theorem:** if keystream is shorter than the plaintext, the cipher can't be perfect.

## Proof of Shannon's th.

By contradiction.

number of different keystream  
is less than the number of different plain text

Assume #keys ( $l$ ) < #messages ( $n$ ) and consider ciphertext  $C_0$  s.t.  $\Pr[C_0] > 0$  ( $C_0$  must exist!)

For some key  $K$ , consider  $P = D_K(C_0)$ . There exist at most  $l$  (#keys) such messages (one per each key).

Choose message  $P_0$  s.t. it is not of the form  $D_K(C_0)$  (there exist  $n-l$  such messages)

Hence  $\Pr[C_0 | P_0] = 0$

But in a perfect cipher  $\Pr[C_0 | P_0] = \Pr[C_0] > 0$ . Contradiction.

Length of keystream must be longer than plaintext

# Attack Models

- is a passing attack, only spying, sniffing something on the network teaching other people, is an attack against confidentiality
- **Eavesdropping:** secretly listening to private conversation of others without their consent
  - **Known plaintext:** attacker has samples of both plaintext and its encrypted version (ciphertext) and is at liberty to make use of them to reveal further secret information such as secret keys → by knowing P and C, analyzing the algorithm, can discover information on K, is done by crypto analyst, typically are math person.
  - **Chosen plaintext:** attacker is able to choose arbitrary plaintexts to be encrypted and obtain the corresponding ciphertexts. The goal of the attack is to gain some further information which reduces the security of the encryption scheme. In the worst case, a chosen-plaintext attack could reveal the scheme's secret key
    - ↳ can inspect the result of encryption of chosen plain text, for find information of K. Attacker has access to system that do the encryption.

# Attack Models

- **Adaptive chosen plaintext**: the cryptanalyst makes a series of interactive queries, choosing subsequent plaintexts based on the information from the previous encryptions.
- **Chosen ciphertext**: the cryptanalyst gathers information, at least in part, by choosing a ciphertext and obtaining its decryption under an unknown key.
- **Physical access**  $\hookrightarrow$  is the opposite of chosen plaintext
- “Physical” modification of messages

# Big numbers

- Enalotto: different columns  $622.614.630 = 1.15 \cdot 2^{29}$
- Seconds since the earth exists  $1.38 \cdot 2^{57}$
- Clocks in a century of a 3 GHz computer  $4.05 \cdot 2^{61}$
- Clock in a century of 1000000 2 GHz computers
- 249 bit prime numbers of  $1.8 \cdot 2^{244}$
- Electrons in the universe  $1.8 \cdot 2^{258}$

how much time is needed to break "this" by BRUTE-FORCE?

- You have transform the number of different possibilities in how much time for a computer.

# Cryptography

can't deal with malware,  
cryptography is not sufficient for  
security, are needed different tools



→ typically network communication  
**Cryptography is secure communication and**

**Data integrity:** how to check whether data have been modified

**Digital signature:** how to sign messages

**Authentication:** how to identify users



To use it we must define “good” keys: **random number generation**



To understand it we need Algebra

A single algorithm is a very small part of a protocol, there are hundred algorithms in protocols, are very complex.

OPEN SSL  
↓  
is open source  
↳ old name of TLS a security protocol

# Standards

↳ defining how you will use a solution.

## Textbook vs standards

↳ real solutions are different, there are more details

Robust implementation to attacks

Combination of several tools in one protocol

Kerberos, SSL, IPSEC, PKCS, X509...

↳ is a set of rule defining how the partners should communicate.

# Cryptography vs Security

Cryptography does not guarantee security

- Rules of the thumb Viruses, worms, trojan horses
- Multi level model of security
- Firewalls
- ... (depending on time)

# SECRET KEY: STREAM CIPHERS & BLOCK CIPHERS

## ○ Cybersecurity

two main approaches in practical info security stream ciphers and block cipher

xoring bit by bit keystream with the plain text return ciphertext



# SECRET KEY CRYPTOGRAPHY

Alice and Bob share

- A crypto protocol E
  - A secret key K
  - They communicate using E with key K
  - Adversary knows E, knows some exchanged messages but ignores K
- 

Two approaches:

- Stream Cipher
- Block ciphers

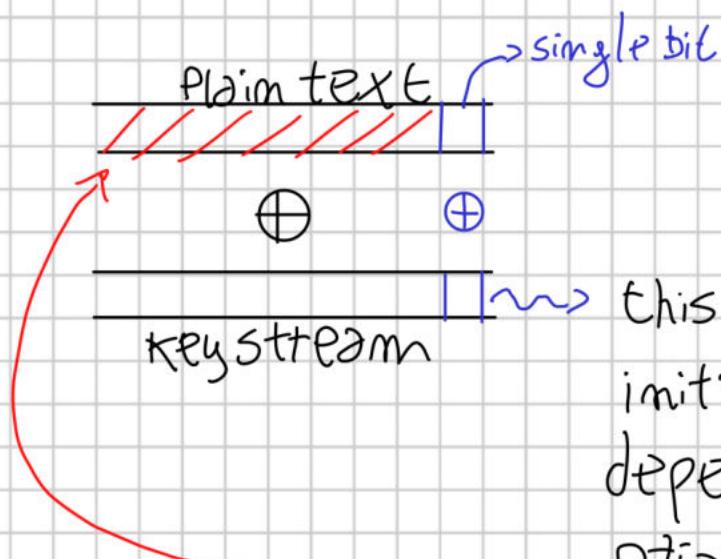
## STREAM CIPHERS

Idea: try to simulate one-time pad

- o define a secret key ("seed") *→ input that make possible to obtain the keystream K*
- o using the seed generate a byte stream (Keystream): i-th byte is function of
  - only key (synchronous stream cipher), or
  - both key and first i-1 bytes of ciphertext (asynchronous stream cipher)
- o obtain ciphertext by bitwise XORing plaintext and keystream

seed; input that make possible to obtain the keystream, there is something like an algorithm that give the keystream infinite. The input is a password ("seed")

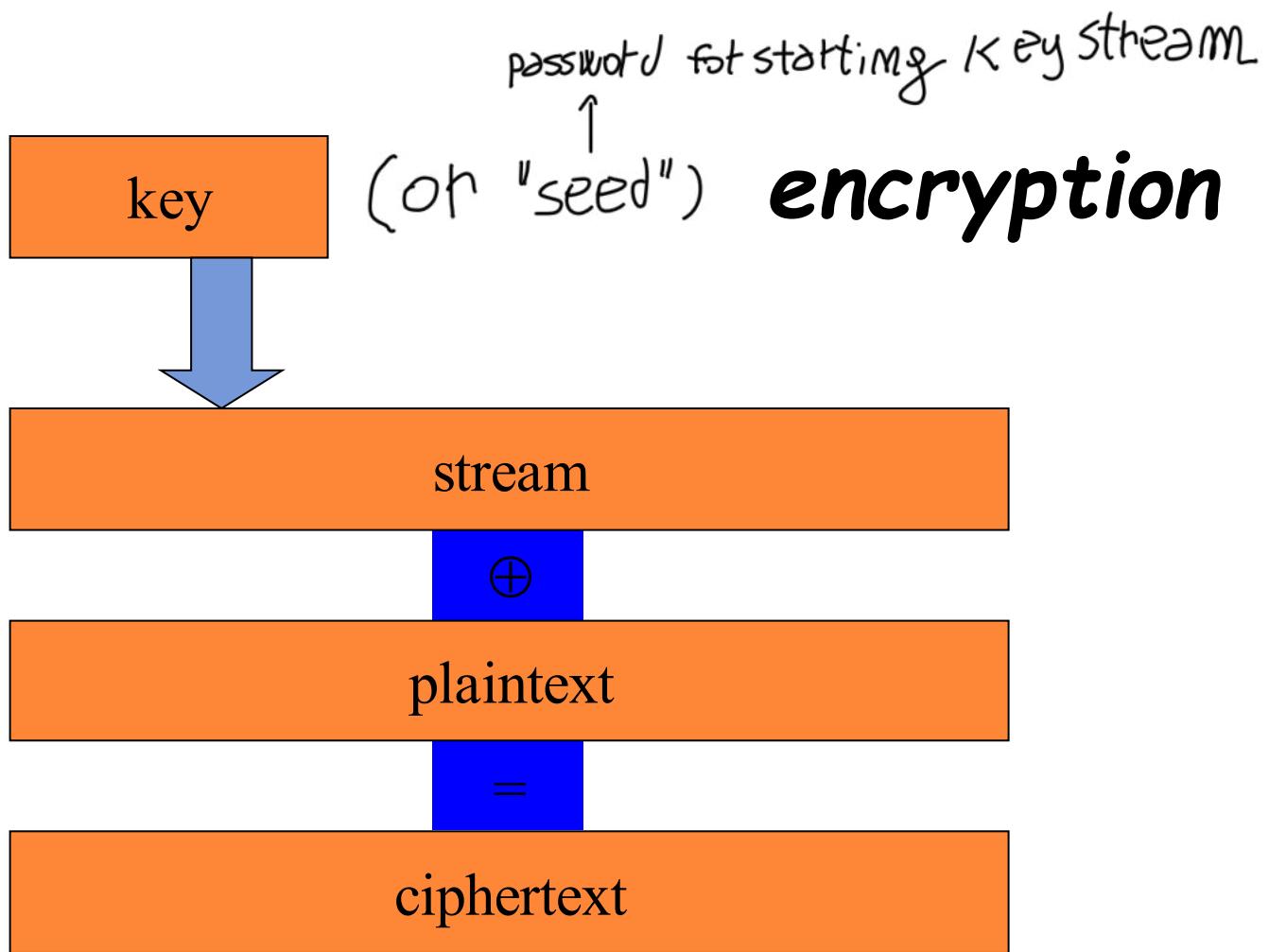
**synchronous stream cipher**: consider a bit of the keystream, if this bit only depending on the initial seed is **synchronous**, while if depend also on what i have already encrypted by means of the algorithm, the previous plaintext is **asynchronous**.



↳ this bit can depend only on the initial seed (synchronous) or can depend on the previous encryption of the plain text (asynchronous)

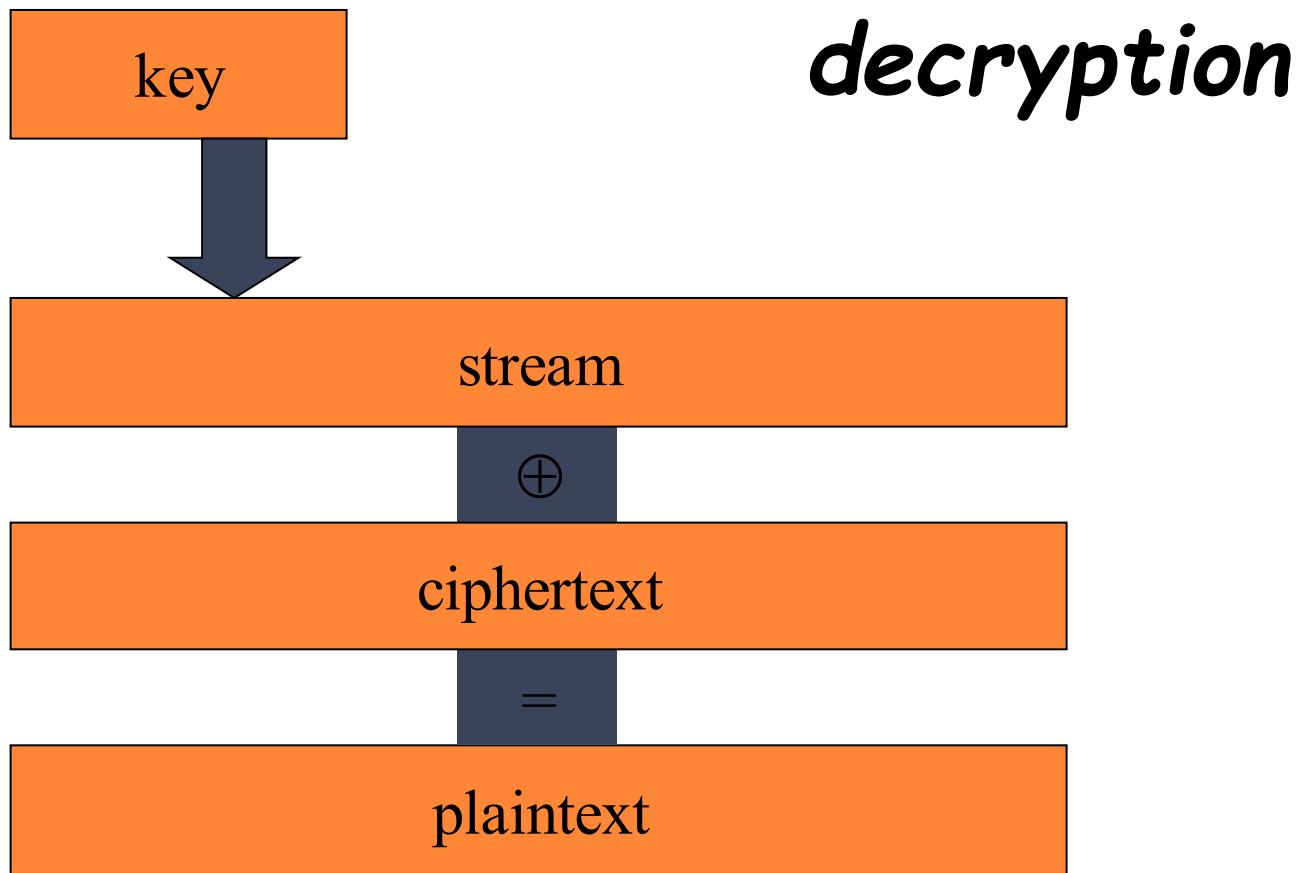
typically an asynchronous stream cipher is more secure, not only depend on a known algorithm but also of the previous Plain text. If the seed is compromised the attacker can produce the same keystream but if the stream cipher is asynchronous the attacker can not obtain the real keystream, he is missing information on previous plaintext.

# SYNCHRONOUS STREAM CIPHER



→ is used also today, not too much

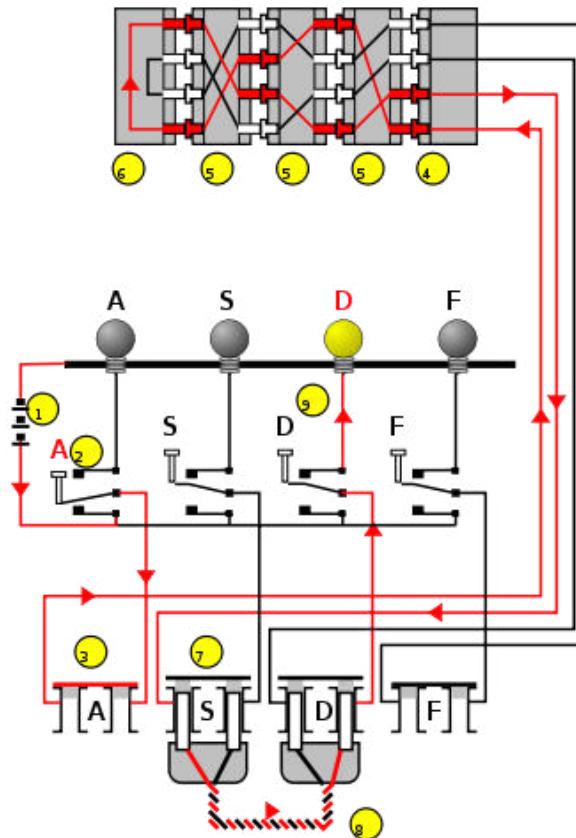
## SYNCHRONOUS STREAM CIPHER



# STREAMS CIPHERS IN PRACTICE

- Many codes before 1940
- Enigma - II world war (Germany)
- A5 - GSM (encryption cell phone-base station)
- WEP - used in Ethernet 802.11 (wireless)
- RC-4 (Ron's Code)

RSA Wim tutting award  
 ↓  
 asymmetric encryption algorithm



Enigma wiring diagram showing current flow.  
 The A key is encoded to the D lamp. D yields A, but A never yields A; this property was due to a patented feature unique to the Enigmas, and could be exploited by cryptanalysts in some situations.

→ example of security by obscurity!

## A5/1 → weak by design

- Stream cipher (1987) used to provide over-the-air communication privacy in the **GSM cellular telephone standard**
- There was a terrific row between the **NATO signal intelligence agencies** in the mid 1980s over whether **GSM encryption should be strong or not**
- Used in Europe and in the United States. A5/2 was a **deliberate weakening of the algorithm** for certain export regions
- Initially kept secret, but the general design was leaked in 1994, and the algorithms were entirely reverse engineered in 1999 by Marc Briceno
  - A number of serious weaknesses in the cipher have been identified

used so much in 90'

RC-4 thy emulate one time pad

Ronald Linn Rivest



- RC: Ron's Code

- (Ron = Ronald Rivest, MIT, born in 1947 in NY state)

- Considered safe: 1987 - 1994 kept secret, after '94 extensively studied

- Good for exporting (complying with US restrictions)

- Easy to program, fast

- Very popular: Lotus Notes, SSL, Wep etc.

- RC4's weak key schedule can give rise to a variety of serious problems

## RC-4 INITIALIZATION

Goal: generate a (pseudo)random permutation of the first 256 natural numbers

1.  $j=0$
2.  $S_0=0, S_1=1, \dots, S_{255}=255$
3. Assume a key of 256 bytes  $k_0, \dots, k_{255}$  (if the key is shorter, repeat)
4. for  $i=0$  to 255 do

1.  $j = (j + S_i + k_i) \bmod 256$
2. exchange  $S_i$  and  $S_j$

*keystream is periodically , but a very large period*

In this way we obtain a permutation of 0, 1, ..., 255, the resulting permutation is a function of the key

*is the seed*  
↑

example of stream cipher

## RC-4 KEY-STREAM GENERATION

Input: permutation S of 0, 1, ..., 255

1.  $i = 0, j = 0$
2. while (true) // we'll not cycle forever, isn't it?
3.  $i = (i + 1) \text{ mod } 256$
4.  $j = (j + S_i) \text{ mod } 256$
5. exchange  $S_i$  and  $S_j$
6.  $t = (S_i + S_j) \text{ mod } 256$
7.  $k = S_t$  // compute XOR *with plain text*  
Keystream

at every iteration compute the XOR between k and next byte of plaintext (or ciphertext)

always talk about symmetric encryption

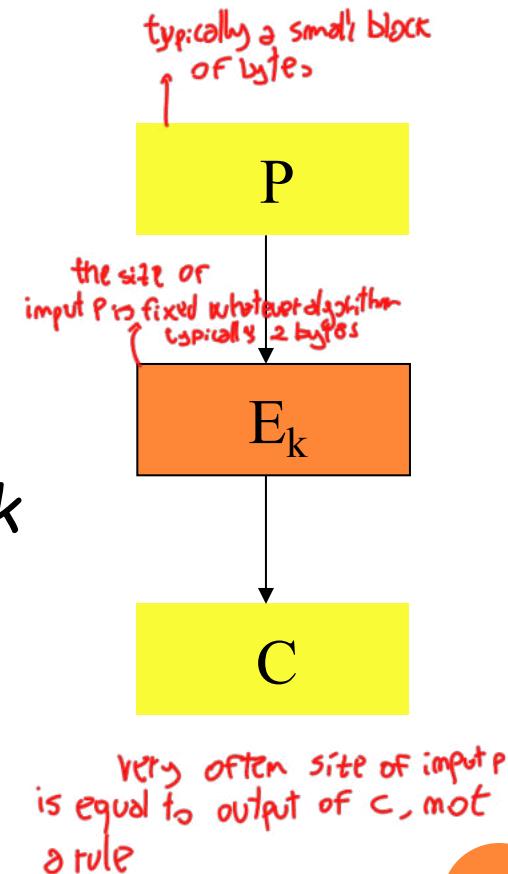
BLOCK CIPHERS → the size of the input is fixed, and very small

Given

- a block  $P$  of text of  $h$  bits ( $h$  fixed)
- a key  $k$  of fixed # of bits

a cryptographic protocol  $E_k$  produces  
a block  $C$  of  $h$  bits, function of  $P$  and  $k$

Note: lengths of both block and key  
(# of bits) are fixed (not necessarily equal)



# REAL WORLD BLOCK CIPHERS

→ weak algorithm by design

- DES, 3-DES - (1976; 64 bit block, 56 bit key)
- RC-2 (1987)
  - designed for exporting cryptography within IBM Lotus Notes
  - 64 bit block, variable key size, vulnerable to an attack using  $2^{34}$  chosen plaintexts
- IDEA (1991)
  - 64 bit block, 128 bit key
  - Strong, only weakened variants have been broken
- Blowfish (1993)
  - 64-bit block size and a variable key length from 32 up to 448 bits
  - Still strong, very very strong algorithm

# REAL WORLD BLOCK CIPHERS

## ○ RC5 (1994)

- variable block size - 32, 64 or 128 bits - key size (0 to 2040 bits) and number of rounds (0 to 255). The original suggested choice of parameters were a block size of 64 bits, a 128-bit key and 12 rounds.
- Distributed.net has brute-forced RC5 messages encrypted with 56-bit and 64-bit keys, and is working on cracking a 72-bit key; as of February 2014, 3.112% of the keyspace has been searched (it was 1.488% at March 2011). At the current rate, it will take approximately 287 years to test every possible remaining key

○ distributed.net (or Distributed Computing Technologies, Inc. or DCTI) is a worldwide distributed computing effort that is attempting to solve large scale problems using otherwise idle CPU or GPU time. It is a non-profit organization

## ○ AES (Rijndael, 2001) Advanced Encryption Standard

- 128 bit block, 128-256 bit key
- Very strong

strongest algorithm  
is open source



# SYMMETRIC BLOCK CIPHERS

## Standard

out

in

DES

AES

Word wide standard

## HISTORIC NOTE

DES (data encryption standard) is a symmetric block cipher using 64 bit blocks and a 56 bit key.

Developed at IBM, approved by the US government (1976) as a standard. Size of key (56 bits) was apparently small enough to allow the NSA (US national security agency) to break it exhaustively even back in 70's.

In the 90's it became clear that DES is too weak for contemporary hardware & algorithmics (Matsui "linear attack", requires only  $2^{43}$  known plaintext/ciphertext pairs; in 1999 Deep Crack and distributed.net break a DES key in 22 hours and 15 minutes)

## LINEAR CRYPTANALYSIS

Wikipedia: based on finding affine approximations to the action of a cipher.

"There are two parts to linear cryptanalysis.

- The first is to construct linear equations relating plaintext, ciphertext and key bits that have a high bias; that is, whose probabilities of holding (over the space of all possible values of their variables) are as close as possible to 0 or 1.
- The second is to use these linear equations in conjunction with known plaintext-ciphertext pairs to derive key bits."

## HISTORIC NOTE (CONT.)

also outside US look on  
↑ these standards, world wide

The US government NIST (National Inst. of standards and technology) announced a call for an advanced encryption standard in 1997.

This was an international open competition. Overall, 15 proposals were made and evaluated, and 6 were finalists. Out of those, a proposal named Rijndael, by Daemen and Rijmen (two Belgians), was chosen in February 2001.

can work with  
different parameters,  
here fixed parameter

AES finalist	positive	negative
Rijndael	86	10
Serpent	59	7
Twofish	31	21
RC6	23	37
MARS	13	84

# AES - ADVANCED ENCRYPTION STANDARD

- Symmetric block cipher (block size: 128 bits)
- Key lengths: 128, 192, or 256 bits
- Approved US standard (2001)
- Finite fields algebra

Galua fields

$$16 \equiv 1 \pmod{15}$$

## CONGRUENCE

- two naturals  $a$  and  $b$  are said to be congruent modulo  $n$  ( $n$  is a positive integer)

$$a \equiv b \pmod{n}$$

if  $|a - b|$  is multiple of  $n$ , or, equivalently, the integer divisions of  $a$  and  $n$  and of  $b$  and  $n$  yield the same remainder

- the congruence relation is reflexive, symmetric and transitive, hence it is an equivalence relation
- the quotient set  $Z_n$  is the set of  $n$  classes of equivalence, congruent to  $0, 1, \dots, n-1$ 
  - $-1 \equiv n - 1 \pmod{n}, -2 \equiv n - 2 \pmod{n}$ , etc.

# NOTATION

- $Z_n = \{[0], [1], [2], \dots, [n-1]\}$ 
  - here  $[i]$  is the equivalence class of integers congruent to  $i \pmod n$
  - for brevity people often write  $Z_n = \{0, 1, 2, \dots, n-1\}$
  - quotient set
- $Z_m^* = \text{natural numbers mod } m \text{ that are relatively prime (co-prime) to } m$   
(multiplicative group of  $Z_m$ )

$$f(x) = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$$

$$f(x) = \lim_{h \rightarrow 0} \frac{(x+h)^2 - x^2}{h}$$

$$= \lim_{h \rightarrow 0} \frac{x^2 + 2xh + h^2 - x^2}{h}$$

$$= \lim_{h \rightarrow 0} \frac{2xh + h^2}{h}$$

$$g'(x) = \lim_{h \rightarrow 0} \frac{g(x+h) - g(x)}{h}$$

$$= \lim_{h \rightarrow 0} h(2x+h)$$

# NOTEWORTHY RESULTS

## Euler theorem (Fermat theorem)

If  $a$  and  $n$  are coprime positive integers (i.e.  $\text{GCD}(a, n) = 1$ ) and  $\varphi(n)$  is the Euler's totient function (how many positive integers not greater than  $n$  are coprime with  $n$ )

$$a^{\varphi(n)} \equiv 1 \pmod{n}$$

1.  $a, n$  are coprime and positive integers don't have common divisor other than 1
2.  $\varphi(n)$  number of positive integers not greater than  $n$ , coprime with  $n$ .

$$\varphi(6) = |\{1, 5\}| = 2 \quad 5^2 \equiv 1 \pmod{6}$$

## Bézout's identity

Let  $a$  and  $b$  be nonzero integers with greatest common divisor  $d$ . Then there exist signed integers  $x$  and  $y$  such that  $ax + by = d$ .

$x$  and  $y$  can be computed by the extended Euclidean algorithm.

(Used for finding the multiplicative inverse, for  $d = 1$ )

multiplicative inverse:

$$q \cdot q^{-1} \equiv 1 \pmod{m}$$

$q, m$  coprime such that  $\rightarrow qx + my = 1$

$qx + my = 1$  Bézout identity

$$qx + my \equiv 1 \pmod{m}$$

$$my \equiv 0 \pmod{m}$$

$$qx \equiv 1 \pmod{m}$$

$$\hookrightarrow q^{-1} = x \text{ and also } b = y^{-1}$$

use OpenSSL

```
Last login: Wed Sep 14 13:04:26 on console
(base) fab@MacBook-Pro-di-Fabrizio ~ % openssl aes-128-cbc -p -in ttt -out ttt.enc
Invalid command '-aes-128-cbc'; type "help" for a list.
(base) fab@MacBook-Pro-di-Fabrizio ~ % openssl aes-128-cbc -p -in ttt -out ttt.enc
enter aes-128-cbc encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
salt=23577EF518677240
key=E91FC2A300DE3D834B0EAC7EA493B98F
iv =C6AB91EC98B42619A5CEFC46B73107CC
(base) fab@MacBook-Pro-di-Fabrizio ~ %
```

Cipher block chain



for encrypt a file: `openssl aes-128-cbc -p -in ttt -out tttenc`

From key derivation function obtain some relevant data: Key, salt, iv (synonymous of seed - initialization vector).

salt is a random string added to password for protect very easy password.  
depending from the password, very hard to reverse.

Attackers use rainbow tables to reverse password:

`crackstation.net` → use this site for reverse encrypted password  
`echo -n "house" | shasum -a256` → encrypt a password  
`hexdump -C ttt.enc` → for see encryption of the file

# CHALLENGE (JUNE EXAM 2017)

compute

$$\begin{array}{c}
 \text{not } \varphi(m) \\
 \uparrow \\
 2^{200} \mod 127 = 2^{126} \cdot 2^{74} = 2^{74} \mod 127 \\
 2^{200} \mod 127 \\
 \hookrightarrow \text{prime number}
 \end{array}$$

using only pen and paper.... using Euler theorem

$$2^{\varphi(127)} \equiv 1 \pmod{127}$$

$$2^{126} \equiv 1 \pmod{127}$$

$$2^{200} \mod 127 = 2^{126} \cdot 2^{74} \mod 127 = 2^{74} \mod 127 = (2^7)^{10} \cdot 2^4 = 16$$

$$2^7 = 128 \Rightarrow 2^7 \mod 127 = 1$$

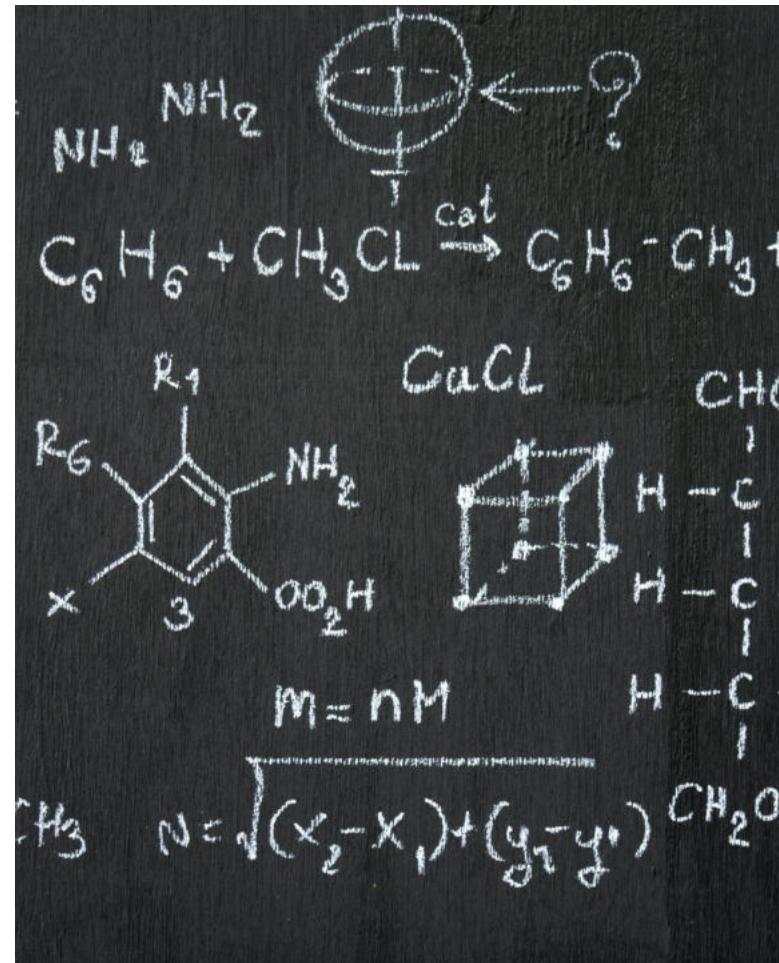
$$(2^7)^{10} \mod 127 = (1)^{10}$$

$$2^{200} \mod 127 = 2^{126} \cdot 2^{74} \mod 127 = \overbrace{2^{126} \mod 127}^1 \cdot \overbrace{(2^7)^{10} \mod 127}^1 \cdot \overbrace{2^4 \mod 127}^{16} = 16$$

# ON THE TOTIENT FUNCTION

from previous definitions:

$\varphi(m)$  = Euler's totient function =  $|Z_m^*|$  = size of the multiplicative group of  $Z_m$



# Galois Fields GF( $p^k$ )

$p$  a prime number

Theorem: For every prime power  $p^k$  ( $k = 1, 2, \dots$ ) there is a unique finite field containing  $p^k$  elements. These fields are denoted by GF( $p^k$ ).

There are no finite fields with other cardinalities.



for us interest case is  $k=2$ , because work with binary

Évariste Galois (1811-1832)

# Polynomials over Finite Fields

Polynomial equations and factorizations in finite fields can be different than over the rationals.

Examples from an XMAPLE session:



```
factor(x^6-1); # over the rationals
(x - 1)(x + 1)(x2 + x + 1)(x2 - x + 1)
Factor(x^6-1) mod 7; # over Z7
(x + 1)(x + 3)(x + 2)(4 + x)(x + 5)(x + 6)
factor(x^4+x^2+x+1); # over the rationals
x4 + x2 + x + 1
Factor(x^4+x^2+x+1) mod 2; # over Z2
(x + 1)(x3 + x2 + 1)
```

# Irreducible Polynomials

A polynomial is **irreducible** in  $GF(p)$  if it does not factor over  $GF(p)$ . Otherwise it is **reducible**.

Examples:



MSc  
Vaksemester

Factor  $(x^5 + x^4 + x^3 + x + 1) \pmod{5}$ ;

$$(x + 2)(x^3 + 3x + 2)(x + 4)$$

Factor  $(x^5 + x^4 + x^3 + x + 1) \pmod{2}$ ;

$$x^5 + x^4 + x^3 + x + 1$$

The same polynomial is reducible in  $Z_5$  but irreducible in  $Z_2$ .

# Implementing $GF(p^k)$ arithmetic

Theorem: Let  $f(x)$  be an irreducible polynomial of degree  $k$  over  $Z_p$ .

The finite field  $GF(p^k)$  can be realized as the set of degree  $k-1$  polynomials over  $Z_p$ , with addition and multiplication done modulo  $f(x)$ .

# Example: Implementing $GF(2^5)$

By the theorem, the finite field  $GF(2^5)$  can be realized as the set of degree 4 polynomials over  $\mathbb{Z}_2$ , with addition and multiplication done modulo the irreducible polynomial  $f(x) = x^5 + x^4 + x^3 + x + 1$ .

The coefficients of polynomials over  $\mathbb{Z}_2$  are 0 or 1. So, a degree  $k$  polynomial can be written down by  $k+1$  bits. For example, with  $k=4$ :

$$x^3 + x + 1 \leftrightarrow (0, 1, 0, 1, 1)$$

$$x^4 + x^3 + x + 1 \leftrightarrow (1, 1, 0, 1, 1)$$

# Implementing GF(2<sup>5</sup>)

Addition: bit-wise XOR (since 1+1=0)

addition of the two  
polynomials can be done  
with xor

$$\begin{array}{r} x^3 + x + 1 \\ (0,1,0,1,1) \\ + \\ x^4 + x^3 + x \\ (1,1,0,1,0) \\ \hline \end{array}$$

# Implementing GF(2<sup>5</sup>)

Multiplication: Polynomial multiplication, and then remainder modulo the defining polynomial f(x):

```
> g(x) := (x^4+x^3+x+1) * (x^3+x+1);           (1,1,0,1,1) * (0,1,0,1,1)
      g(x) := (x4 + x3 + x + 1)(x3 + x + 1)
> f(x) := x^5+x^4+x^3+x+1;                      = (1,1,0,0,1)
      f(x) := x5 + x4 + x3 + x + 1
> rem(g(x), f(x), x);
      1 + 3 x4 + x3 + 2 x
> % mod 2;
      1 + x4 + x3
```

→ in practical is small enough

substitution

S-BOX

For small size finite field, a lookup table is the most efficient method for implementing multiplication.

# AES - ADVANCED ENCRYPTION STANDARD

- Symmetric block cipher
- Key lengths: 128, 192, or 256 bits

↳ most popular, big enough to resist brute force attack

## Rationale

- Resistance to all known attacks
- Speed and code compactness
  - good for devices with limited computing power, e.g. smart cards
- Simplicity

→ this is called status of the algorithm and algorithm runs most in place on this status transforming the bytes of the input (Plain text) in the cipher text. The round repeated ten times.

## AES SPECIFICATIONS

$$16 \text{ bytes} = 2^6 = 128 \text{ bits}$$

- Input & output block length: 128 bits.
- State: 128 bits, arranged in a 4-by-4 matrix of bytes.

$A_{0,0}$	$A_{0,1}$	$A_{0,2}$	$A_{0,3}$
$A_{1,0}$	$A_{1,1}$	$A_{1,2}$	$A_{1,3}$
$A_{2,0}$	$A_{2,1}$	$A_{2,2}$	$A_{2,3}$
$A_{3,0}$	$A_{3,1}$	$A_{3,2}$	$A_{3,3}$

Each byte is  
viewed as an  
element in  
 $GF(2^8)$

Input/Output:  $A_{0,0}, A_{1,0}, A_{2,0}, A_{3,0}, A_{0,1}, \dots$

# AES Specifications

- Key length: 128, 196, 256 bits. *→ dynamically added columns during execution*

Cipher Key Layout:  $n = 128, 196, 256$  bits, arranged in a 4-by- $n/32$  matrix of bytes.

$K_{0,0}$	$K_{0,1}$	$K_{0,2}$	$K_{0,3}$	$K_{0,4}$	$K_{0,5}$
$K_{1,0}$	$K_{1,1}$	$K_{1,2}$	$K_{1,3}$	$K_{1,4}$	$K_{1,5}$
$K_{2,0}$	$K_{2,1}$	$K_{2,2}$	$K_{2,3}$	$K_{2,4}$	$K_{2,5}$
$K_{3,0}$	$K_{3,1}$	$K_{3,2}$	$K_{3,3}$	$K_{3,4}$	$K_{3,5}$

Initial layout:  $K_{0,0}, K_{1,0}, K_{2,0}, K_{3,0}, K_{0,1}, \dots$

# AES SPECIFICATIONS

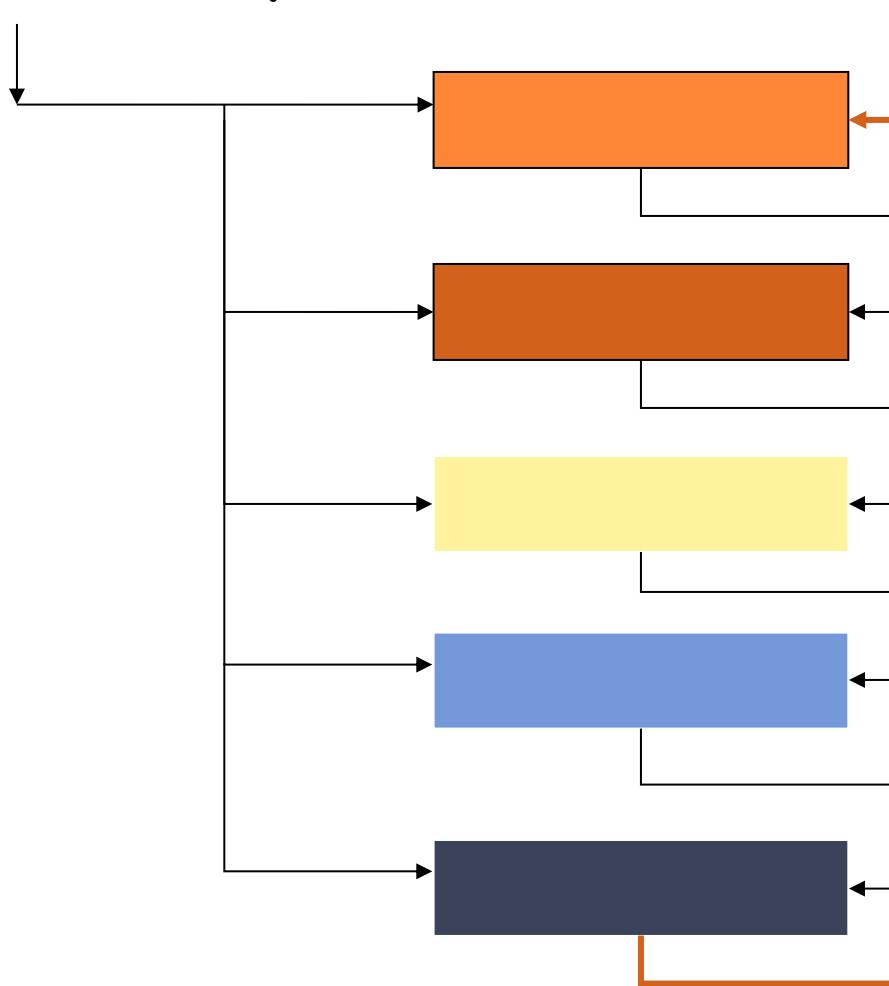
## High level code

### ○ AES(State, Key)

- 1 • KeyExpansion(Key, ExpandKey) XOR state - key  
↑
- 2 • AddRoundKey(State, ExpandKey[0])  
• for ( $i = 1; i < R; i++$ ) do  
    Round(State, ExpandKey[i]); ~> main path  
        running rounds  
        many times
- 3 • FinalRound(State, ExpandKey[R]):  
↳ three steps of the round is also there, almost  
        another round, missing one step  
        nine round plus this ↑

# Encryption: Carried out in rounds

Secret key (128 bits)



input block  
(128 bits)

change secret  
times

output block  
(128 bits)

# Rounds in AES

128 bits AES uses 10 rounds, no shortcuts known for 6 rounds *already enough*

- The secret key is expanded from 128 bits to 10 round keys, 128 bits each.
- Each round changes the state, then XORs the round key. (for longer keys, add one round for every extra 32 bits)

Each rounds complicates things a little.

Overall it seems infeasible to invert without the secret key (but easy given the key).

# AES Specifications: One Round

Transform the state by applying:

$A_{0,0}$	$A_{0,1}$	$A_{0,2}$	$A_{0,3}$
$A_{1,0}$	$A_{1,1}$	$A_{1,2}$	$A_{1,3}$
$A_{2,0}$	$A_{2,1}$	$A_{2,2}$	$A_{2,3}$
$A_{3,0}$	$A_{3,1}$	$A_{3,2}$	$A_{3,3}$

1. Substitution
2. Shift rows
3. Mix columns
4. XOR round key

one round defined by these four steps

all simple operation

# Substitution (S-Box)

→ just a table containing the multiplicative inverse for the first 256 numbers

Substitution operates on every Byte

separately:  $A_{i,j} \leftarrow A_{i,j}^{-1}$

(multiplicative inverse in  $GF(2^8)$ )

which is highly **non linear**)

If  $A_{i,j} = 0$ , don't change  $A_{i,j}$

Clearly, the substitution is invertible.

↳ in order to do  
the decryption

# Cyclic Shift of Rows

$A_{0,0}$	$A_{0,1}$	$A_{0,2}$	$A_{0,3}$
$A_{1,3}$	$A_{1,0}$	$A_{1,1}$	$A_{1,2}$
$A_{2,2}$	$A_{2,3}$	$A_{2,0}$	$A_{2,1}$
$A_{3,1}$	$A_{3,2}$	$A_{3,3}$	$A_{3,0}$

no shift

shift 1 position

shift 2 positions

shift 3 positions

Clearly, the shift is invertible.

# Mixing Columns

Every state column is considered as a  
Polynomial over  $GF(2^8)$

Multiply with an invertible polynomial

$$03x^3 + 01x^2 + 01x + 02 \pmod{x^4 + 1}$$

$$\text{Inv} = 0Bx^3 + 0Dx^2 + 09x + 0E$$

Round: SubBytes(State)

ShiftRows(State)

MixColumns(State)

AddRoundKey(State, ExpandedKey[i])

→ XOR between current state and the key expanded

## KEY EXPANSION

*Schedule operation of key*

- Generate a "different key" per round
- Need a  $4 \times 4$  matrix of values (over  $GF(2^8)$ ) per round
- Based upon a non-linear transformation of the original key.
- Details available: *The Design of Rijndael*, Joan Daemen and Vincent Rijmen, Springer
- animation

# Breaking AES

Not knowing the key

Breaking 1 or 2 rounds is easy.

It is not known how to break 5 rounds.

Breaking the full 10 rounds AES efficiently  
(say 1 year on existing hardware, or in  
less than  $2^{128}$  operations) is considered  
impossible ! (a good, tough challenge...)

most of the protocols use AES for confidentiality

most of the time block cipher have fixed input size of 128 bits, but the file to be encrypted have a bigger size and have not necessarily a size multiple of the block size. the last block often have a smaller size

## BLOCK CIPHER MODES OF OPERATION

- block ciphers operate on blocks of fixed length, often 64 or 128 bits
- because messages may be of any length, and because encrypting the same plaintext under the same key always produces the same output,

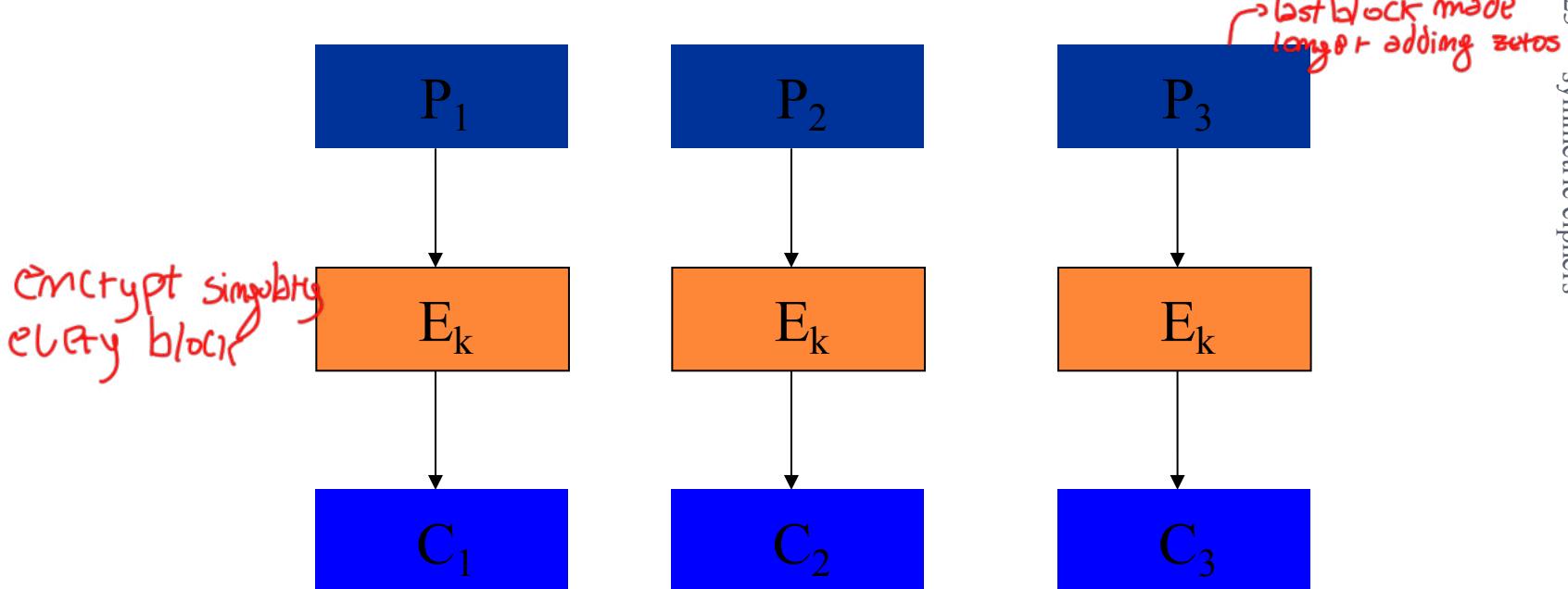
several modes of operation have been invented which allow block ciphers to provide confidentiality for messages of arbitrary length

modes of operation define the way you are considering for solving the problem of handling longer files .

# ECB MODE ENCRYPTION (ELECTRONIC CODE BOOK)

easiest approach

no mech



deeply insecure because for example a man in the middle can easily swap the blocks of cipher text, the decryption will work but the message is corrupted.  
the attacker can easily repeat a block, corrupt information.

encrypt each plaintext block separately

# PROPERTIES OF ECB

- Simple and efficient
- Parallel implementation possible
- Does not conceal plaintext patterns
- Active attacks are possible (plaintext can be easily manipulated by removing, repeating, or interchanging blocks).

attacker not know  
all information, but some

↑ pattern yes

good only when encrypt something

smaller than one block, attacker don't find repetition  
there is not vulnerability. Ex.: encrypt a key  
is small enough to fit in one block

→ cipher text is equal if  
the message is the same

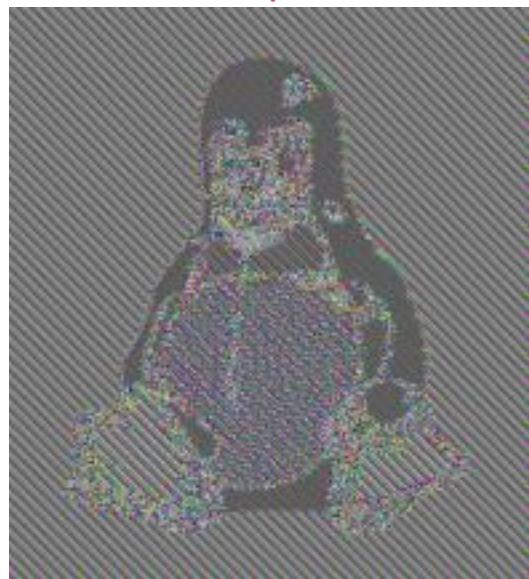
# ECB: PLAINTEXT REPETITIONS

plaintext

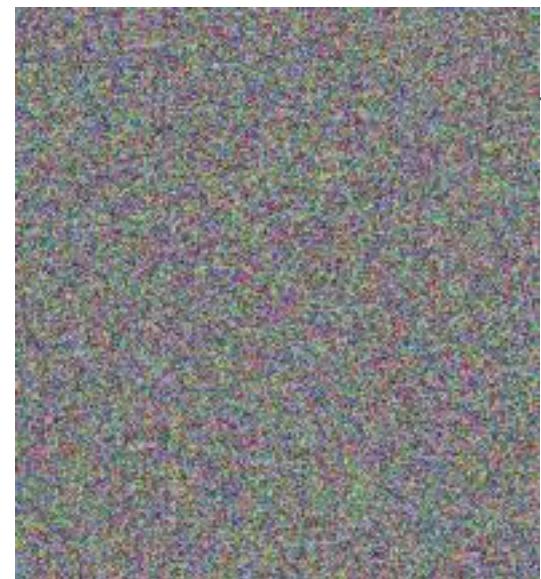


ciphertext ECB

not hide all information  
↑



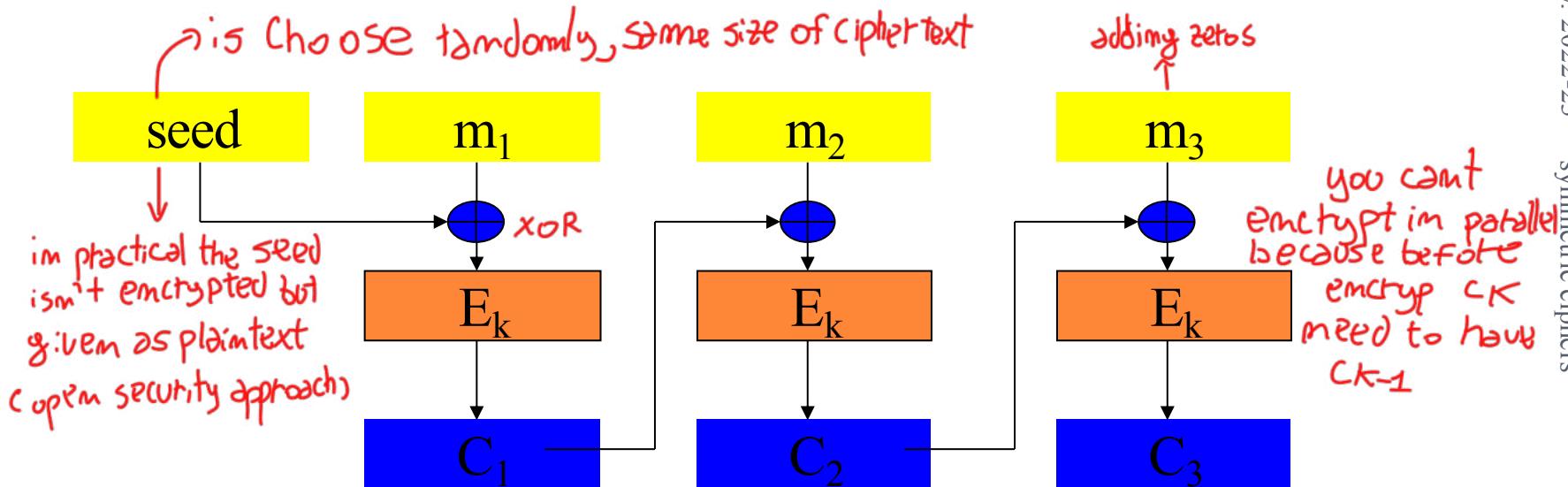
good ciphertext



# CBC (CIPHER BLOCK CHAINING) → very popular

MODE idea to chain blocks

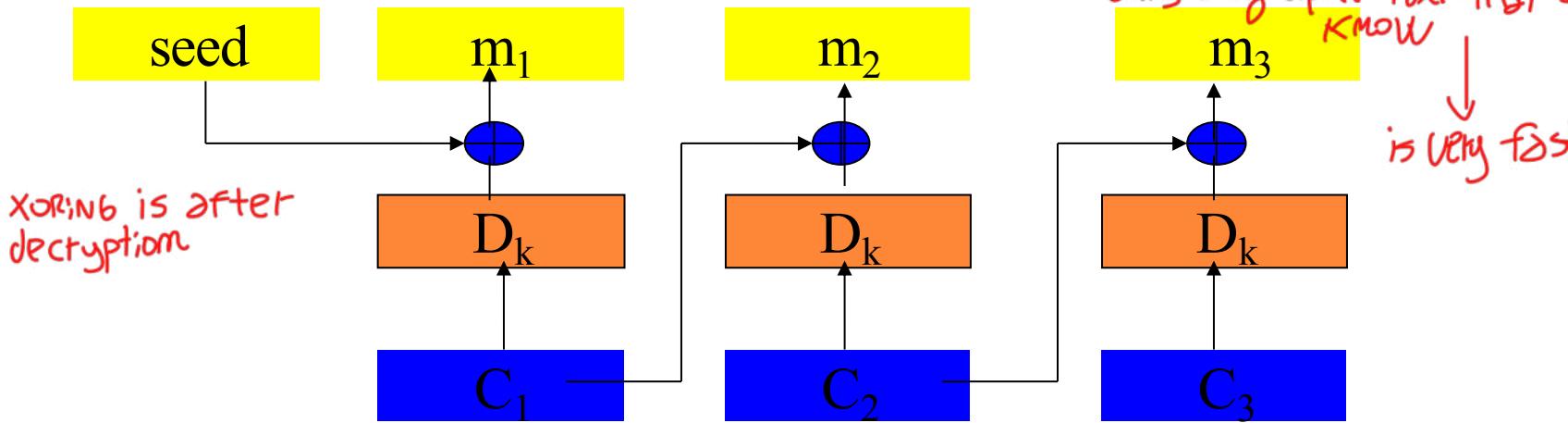
IBM, 1976



- Previous ciphertext is XORed with current plaintext before encrypting current block, depending by the past
- Seed is used to start the process; it can be sent without encryption, XORED with first block
- Seed = 0 safe in most but NOT all cases (e.g. assume the file with salaries is sent once a month, with the same seed we can detect changes in the salaries) therefore a random seed is better

If encrypt same message today and tomorrow using the same key, the sequence of blocks for the cipher text will be different.  
Attacker can't understand cipher text if message is the same

## CBC (CIPHER BLOCK CHAINING): DECRYPTION



### Problem

IF a transmission error changes one bit of  $C_{(i-1)}$ , THEN block  $m_i$  changes in a predictable way (this can be exploited by adversary)

BUT there are unpredictable changes in  $m_{(i-1)}$ :

Solution: always use error detecting codes (for example CRC) to check quality of transmission

## PROPERTIES OF CBC

From external observer you see  
XOR with something, seem  
a stream cipher

key for doing XOR depending by previous block

- Asynchronous stream cipher
- Errors in one ciphertext block propagate
  - a one-bit change to the ciphertext causes complete corruption of the corresponding block of plaintext, and inverts the corresponding bit in the following block of plaintext
- Conceals plaintext patterns
- No parallel implementation known
  - no parallel encryption
  - what about decryption? *yes*
- Plaintext cannot be easily manipulated
- Standard in most systems: SSL, IPsec etc.

## MORE ON CBC

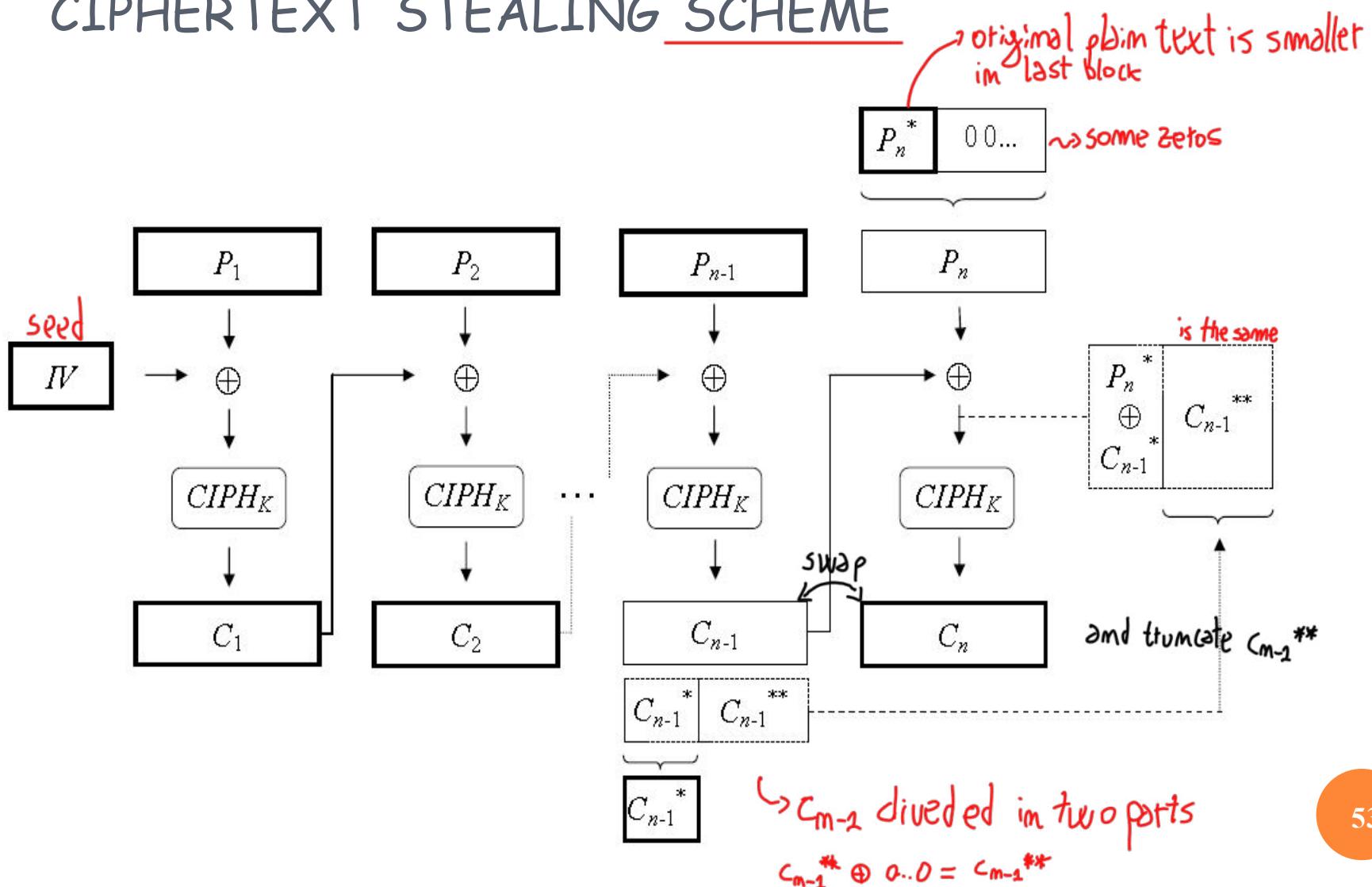
- message must be padded to a multiple of the cipher block size
  - one way to handle this issue is ciphertext stealing
- a plaintext can be recovered from just two adjacent blocks of ciphertext
  - as a consequence, decryption can be parallelized
  - usually a message is encrypted once, but decrypted many times

## CIPHERTEXT STEALING

allow to avoid the padding of zeros in last block

- general method that allows for processing of messages that are not evenly divisible into blocks
  - without resulting in any expansion of the ciphertext
  - at the cost of slightly increased complexity
- consists of altering processing of the last two blocks of plaintext, resulting in a reordered transmission of the last two blocks of ciphertext  
(and no ciphertext expansion)
- suitable for ECB and CBC
- from NIST website  
<http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/ciphertext%20stealing%20proposal.pdf>

# CIPHERTEXT STEALING SCHEME



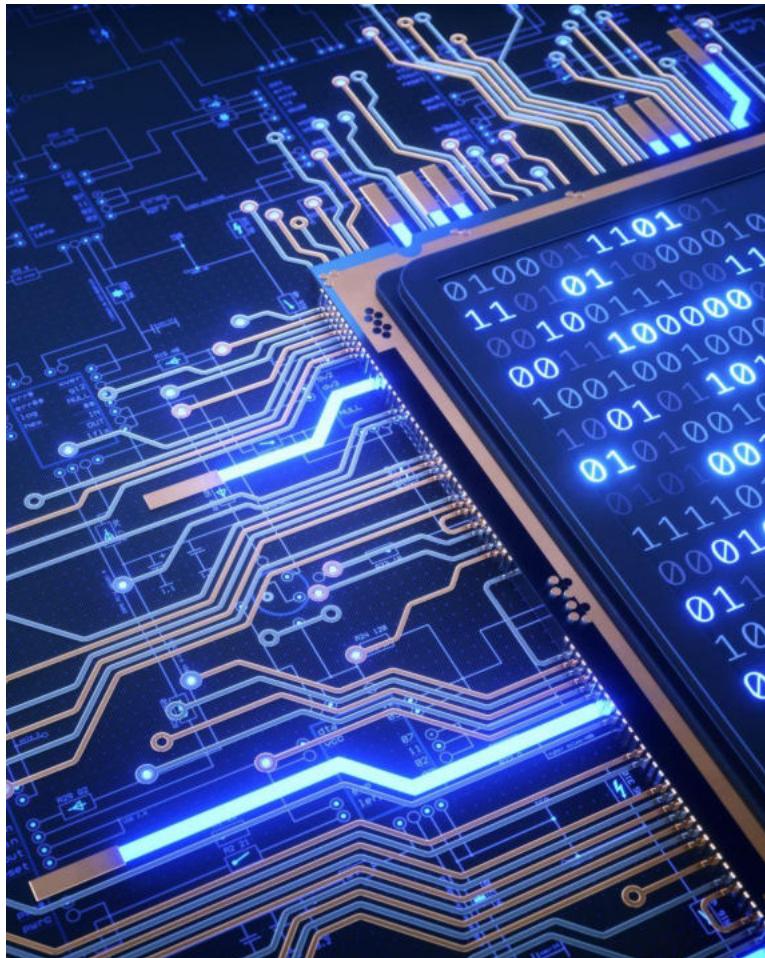
# ENCRYPTION/DECRYPTION

## Encryption procedure

- If the plaintext length is not a multiple of the block size, pad it with enough zero bits until it is.
- Encrypt the plaintext using the Cipher Block Chaining mode.
- Swap the last two ciphertext blocks.
- Truncate the ciphertext to the length of the original plaintext.

## Decryption procedure

- If the ciphertext length is not a multiple of the block size, say it is n bits short, then pad it with the last n bits of the block cipher decryption of the last full ciphertext block.
- Swap the last two ciphertext blocks.
- Decrypt the ciphertext using the Cipher Block Chaining mode.
- Truncate the plaintext to the length of the original ciphertext.

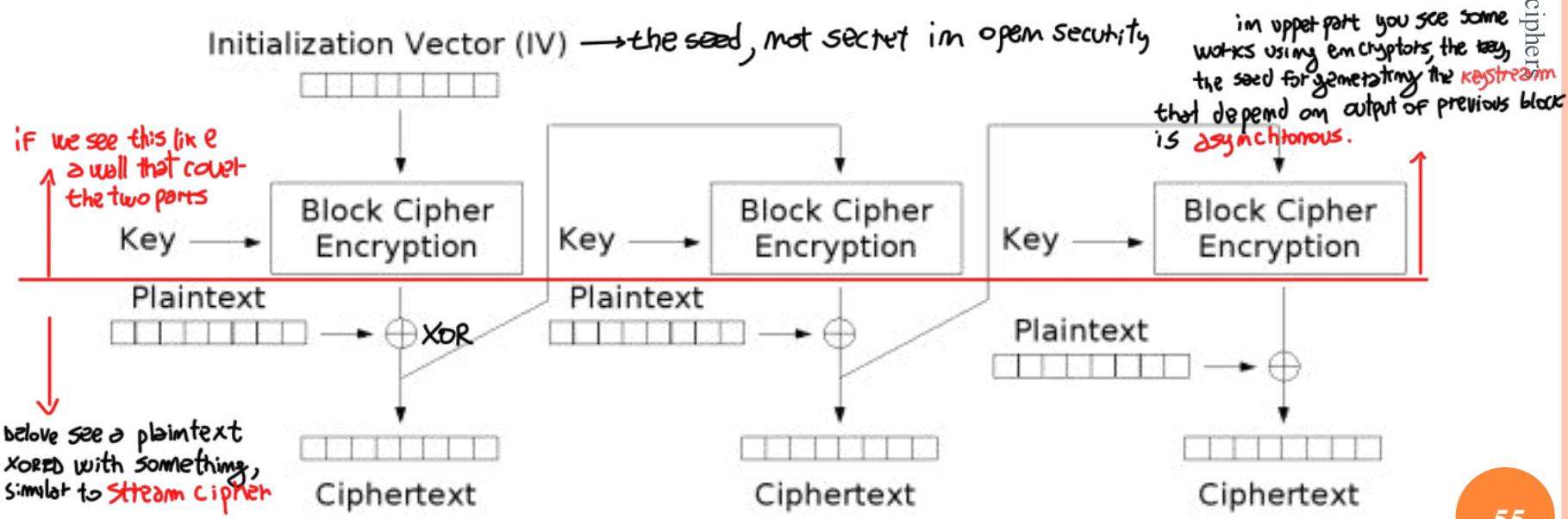


another operation mode very simple

## CIPHER FEEDBACK (CFB)

- like CBC, makes a block cipher into an asynchronous stream cipher

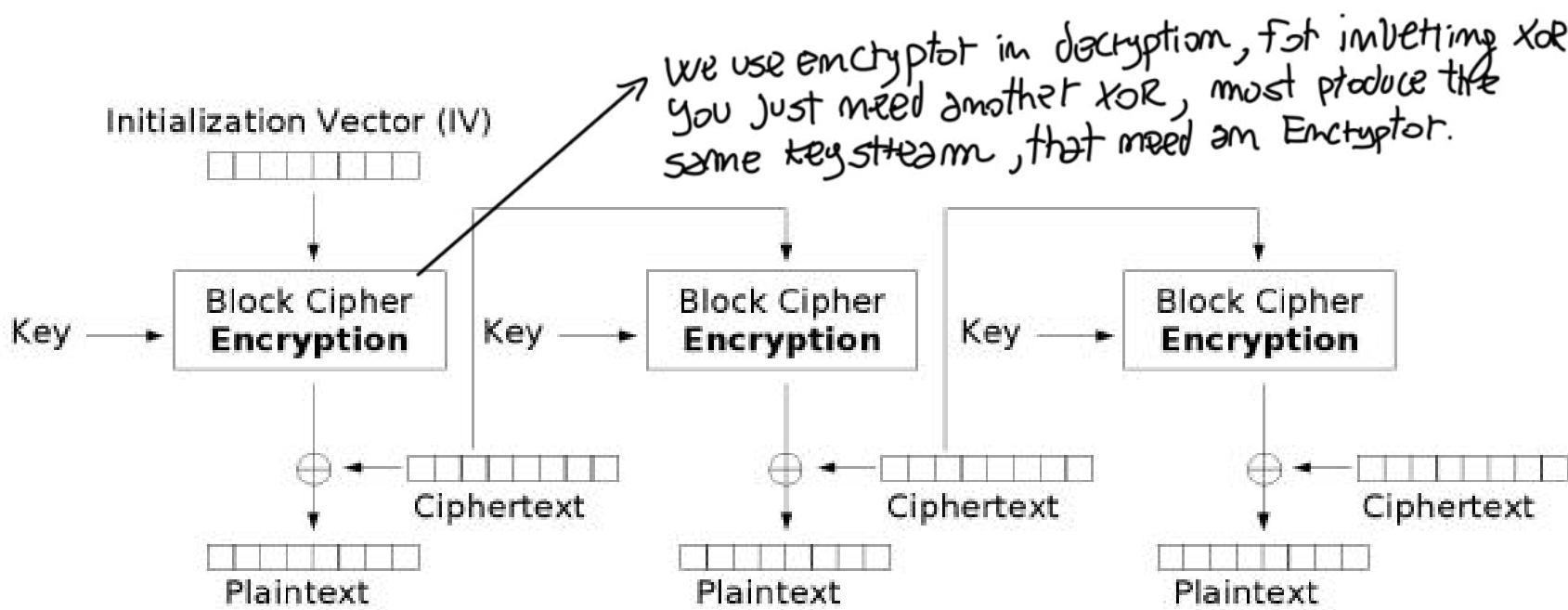
- i.e., supports some re-synchronizing after error, if input to encryptor is given thru a shift-register



Cipher Feedback (CFB) mode encryption

**exam question:** what technic allow to decrypt information using only encryptors? — Show CFB!

## CFB DECRYPTION



Cipher Feedback (CFB) mode decryption

you can parallelize only decryption, but you can't do any pre-processing before know plaintext. Without know P you can't start encrypting

# CFB

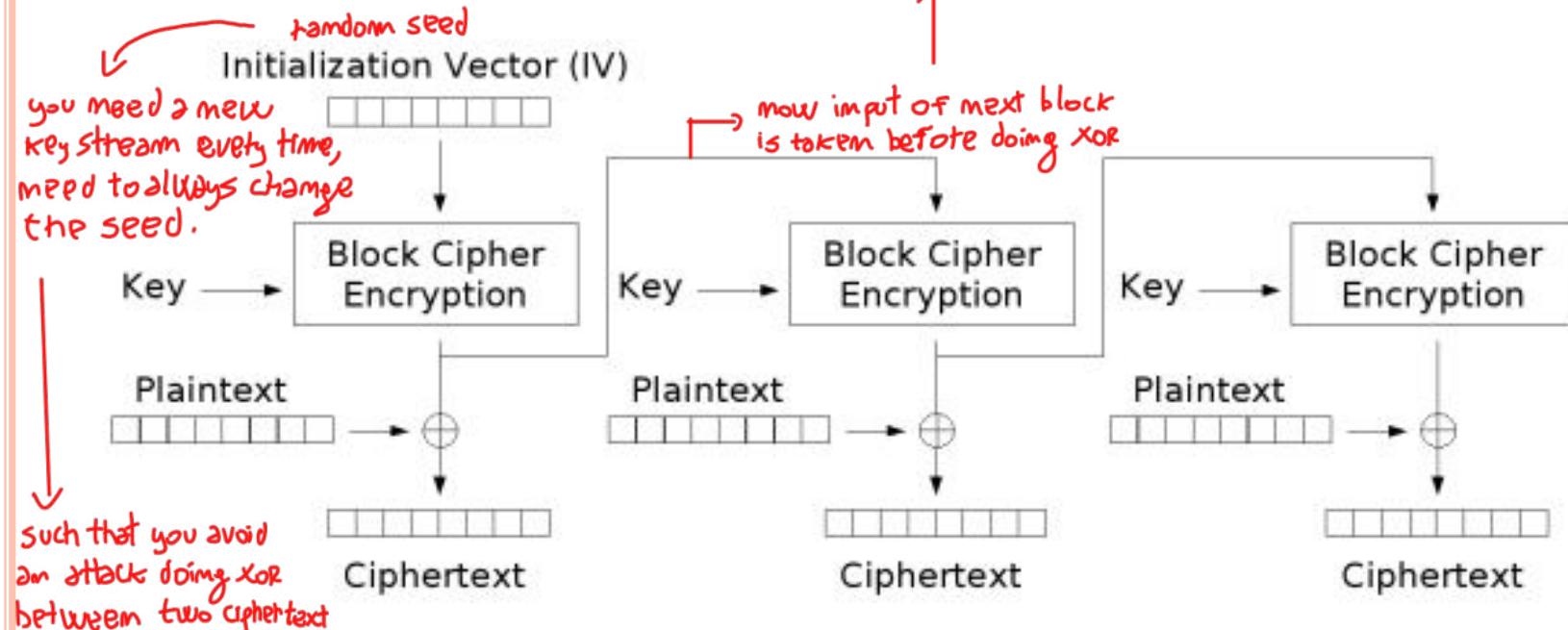
one bit changed in ciphertext propagate to the other

- Like CBC mode, changes in the plaintext propagate forever in the ciphertext, and encryption cannot be parallelized.
  - Also, like CBC, decryption can be parallelized.
- When decrypting, a one-bit change in the ciphertext affects two plaintext blocks: a one-bit change in the corresponding plaintext block, and complete corruption of the following plaintext block. Later plaintext blocks are decrypted normally.
- CFB shares two advantages over CBC mode: the block cipher is only ever used in the encrypting direction, and the message does not need to be padded to a multiple of the cipher block size.

## OFB MODE (OUTPUT FEEDBACK)

- Makes a block cipher into a synchronous stream cipher: it generates keystream blocks, which are then XORed with the plaintext blocks to get the ciphertext.
- Flipping a bit in the ciphertext produces a flipped bit in the plaintext at the same location. This property allows many error correcting codes to function normally even when applied before encryption.

## OFB SCHEME

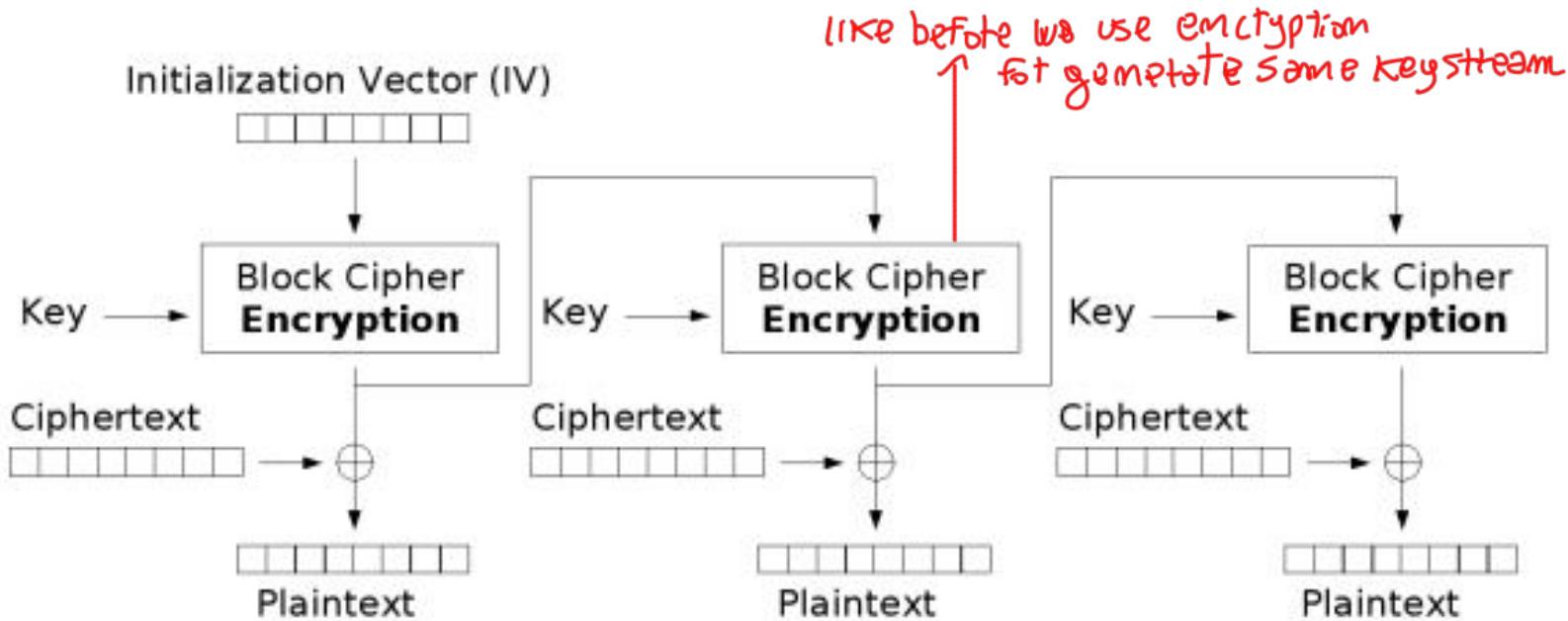


Output Feedback (OFB) mode encryption

An initialization vector IV is used as a "seed" for a sequence of data blocks

this simple scheme is very fast with pre-processing

# OFB DECRYPTION



Output Feedback (OFB) mode decryption

## OFB PROPERTIES

Because of the symmetry of the XOR operation,  
encryption and decryption are exactly the same

$$C_i = P_i \oplus O_i$$

*keystream*

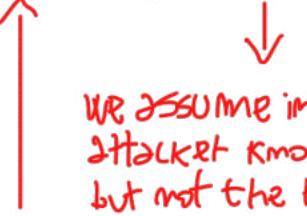
$$P_i = C_i \oplus O_i$$

$$O_i = E_k(O_{i-1})$$

$$O_0 = \text{IV}$$

# OFB MODE

the adversary is able to use a block working doing encryption having the key. Then for having security, you have to rely in secrecy of keystream.

 We assume in O.S. that attacker know algorithm but not the key!

## Discussion

- If  $E_k$  is public (known to the adversary) then initial seed must be encrypted (why?)
- If  $E$  is a cryptographic function that depends on a secret key, then initial seed can be sent in clear (why?) key is secret, seed can be "open"
- Initial seed must be modified for EVERY new message - even if it is protected and unknown to the adversary (in fact if the adv knows a pair message, initial seed then he can encrypt every message - why?)
- Extension: it can be modified in such a way that only  $k$  bits are used to compute the ciphertext ( $k$ -OFB)

## PROPERTIES OF OFB

- Synchronous stream cipher
- Errors in ciphertext do not propagate, one bit change only  
the bit in plain text
- Pre-processing is possible
- Conceals plaintext patterns
- No parallel implementation known
- Active attacks by manipulating plaintext are possible

*last operating mode*

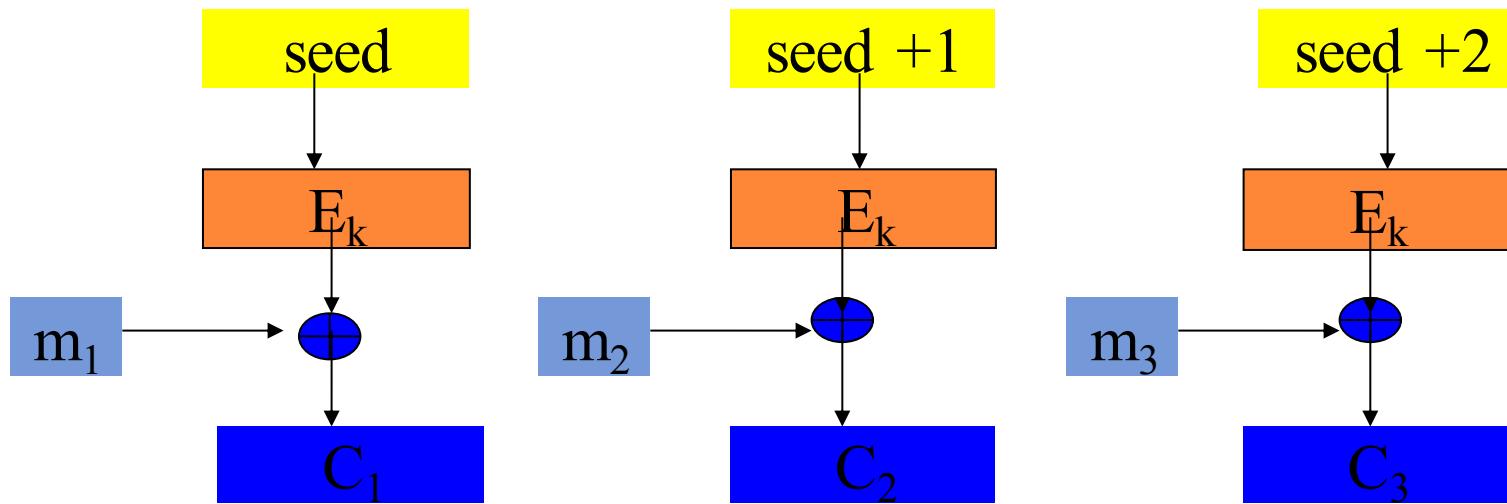
## CTR (COUNTER MODE) *very popular*

- also known as Integer Counter Mode (ICM) and Segmented Integer Counter (SIC) mode
- turns a block cipher into a stream cipher: it generates the next keystream block by encrypting successive values of a "counter"
  - counter can be any function which produces a sequence which is guaranteed not to repeat for a long time, although an actual counter is the simplest and most popular.
- the usage of a simple deterministic input function raised controversial discussions
- has similar characteristics to OFB, but also allows a random access property during decryption
- well suited to operation on a multi-processor machine where blocks can be encrypted in parallel

You choose a random seed, in each block to encrypt you increase the value of the seed, you obtain ciphertext XORing the result of encryption with plaintext

## CTR (COUNTER MODE)

can work in parallel



Similar to OFB

- There are problems in repeated use of same seed (like OFB)
- CTR vs OFB: using CTR you can decrypt the message starting from block  $i$  for any  $i$  (i.e. You do not need to decrypt from the first block as in OFB)

# INITIALIZATION VECTOR (IV) : the seed

- Most modes (except ECB) require an initialization vector, or IV
  - sort of "dummy block" to kick off the process for the first real block, and also to provide some randomization for the process.
  - no need for the IV to be secret, in most cases, but it is important that it is never reused with the same key.
- For CBC and CFB, reusing an IV leaks some information about the first block of plaintext, and about any common prefix shared by the two messages.
- In CBC mode, the IV must, in addition, be unpredictable at encryption time
  - see TLS CBC IV attack
- For OFB and CTR, reusing an IV completely destroys security.

## AES PROPOSED MODES

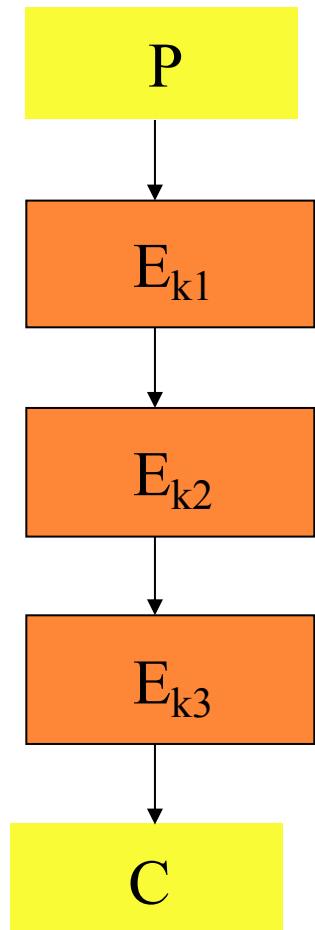
- CTR (Counter) mode (OFB modification): Parallel implementation, offline pre-processing, provable security, simple and efficient
- OCB (Offset Codebook) mode - parallel implementation, offline preprocessing, provable security (under specific assumptions), authenticity
- also see Block Cipher Modes, by NIST  
[http://csrc.nist.gov/groups/ST/toolkit/B  
CM/index.html](http://csrc.nist.gov/groups/ST/toolkit/BCM/index.html) 2001 - 2010

## STRENGTHENING A GIVEN CIPHER

input is xor message  
with another key

- Design multiple key lengths - AES
- Key Whitening - the DES-X idea
  - Key whitening consists of steps that combine the data with portions of the key (most commonly using a simple XOR) before the first round and after the last round of encryption.
  - First use by Ron Rivest for strengthen DES, in 1984
 
$$\text{DES-X}(M) = K_2 \oplus \text{DES}_K(M \oplus K_1)$$
  - apparently key size becomes 184 (= 56 + 64×2), but its strength is 119 ( $|K_1| = |K_2| = 64$ )
- Iterated ciphers - Triple DES (3-DES), triple IDEA and so on

## TRIPLE CIPHER - DIAGRAM



3 time using DES  
with different keys,  
but not equal of triple  
strength but only  
double.

using three keys,  
block cipher

using brute force seem  
that you need  $2^{192}$  cases,  
 $2^{64 \times 3}$ , but with a better  
analysis you get a  
smaller result

↪ algorithm here can  
be a Encryptor or  
also a Decryptor.

# ITERATED CIPHERS

- Plaintext undergoes encryption repeatedly by underlying cipher

- Ideally, each stage uses a different key

- In practice triple cipher is usually

$C = E_{k1}(E_{k2}(E_{k1}(P)))$  [EEE mode] or

$C = E_{k1}(D_{k2}(E_{k1}(P)))$  [EDE mode]

EDE is more common in practice

→ have the security of only two keys, not three but we handle three keys

## TWO OR THREE KEYS

- Sometimes only two keys are used in 3-DES
- Identical key must be at beginning and end
- Legal advantage (export license) due to smaller overall key size
- Used as a KEK (key-encrypting key) in the BPI (Baseline Privacy Interface) protocol which secures the DOCSIS cable modem standard
  - DOCSIS: international standard that permits the addition of high-speed data transfer to an existing Cable TV system. It is employed by many cable television operators to provide Internet access over their existing hybrid fiber coaxial infrastructure

## ADVERSARY'S GOAL

- Final goal: find the secret key
- Partial goals:
  - Reduce the # of possible keys that are used
  - Detect patterns in the text
  - Decode part of the text
  - Modify the ciphertext obtaining a plausible text (even without breaking the cipher; even without knowing which modifications)

→ is very important that the plaintext have a meaning, with the numbers is more difficult

asynchronous encryption usually used for digital signature!

# DOUBLE DES: MEET-IN-THE-MIDDLE ATTACK

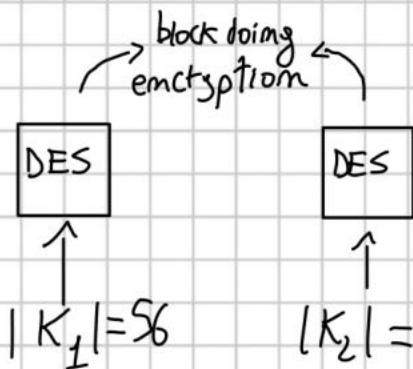
Cipher twice with two different keys? NO!

Meet-in-the-middle attack. Requirements

- Known plaintext/ciphertext pairs ( $P, C$ )
- $2^n$  encryptions +  $2^n$  decryptions (2 keys of  $n$  bit), instead of  $2^{2n}$  brute-force
- $2^n$  memory space
- Idea: try all possible  $2^n$  encryptions of the plaintext and all possible  $2^n$  decryptions of the ciphertext.  
Encryptions stored into a lookup table.
- Check for a pair of keys that transform the plaintext in the ciphertext. Test pair on other pairs plaintext/ciphertext
- Note: the method can be applied to all block codes

## Attack double DES:

Meet in the Middle (MITM) an attack type:



↳ reduce level of multiple encryption by one level

$$2^{56+56} = 2^{112} \text{ number of possible keys}$$

$K_1$  and  $K_2$ , are big enough to contrast brute force.

but exist another attack, the meet in the middle that have the effect to transform  $2^{112}$  to  $2^{56}$ , reduce the keys.

an huge table where compute all possible keys

000...		$\left\{ \begin{array}{l} 10^9 \text{ s for adding one line} \\ 2^{56} \text{ times, all possible keys} \end{array} \right.$
.	.	

adversary have a pair of plaintext and ciphertext ( $P, C$ ), not knowing the key

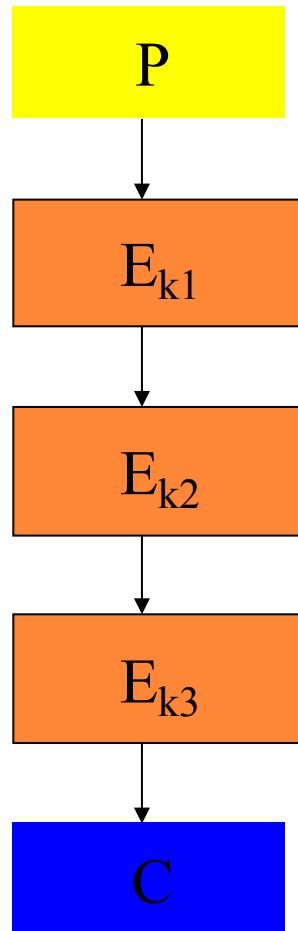
if apply one key to plaintext obtain something and write the result,  $2^{56}$  lines and  $2^{56}$  encryption (very big number). We do this another time with ciphertext for obtain the other key  $K_2$ . Searching in table for a result

$$10^9 \text{ s} \cdot 2^{56} \approx 7200 \text{ years} \approx 2 \text{ years} \quad (\text{a big time!})$$

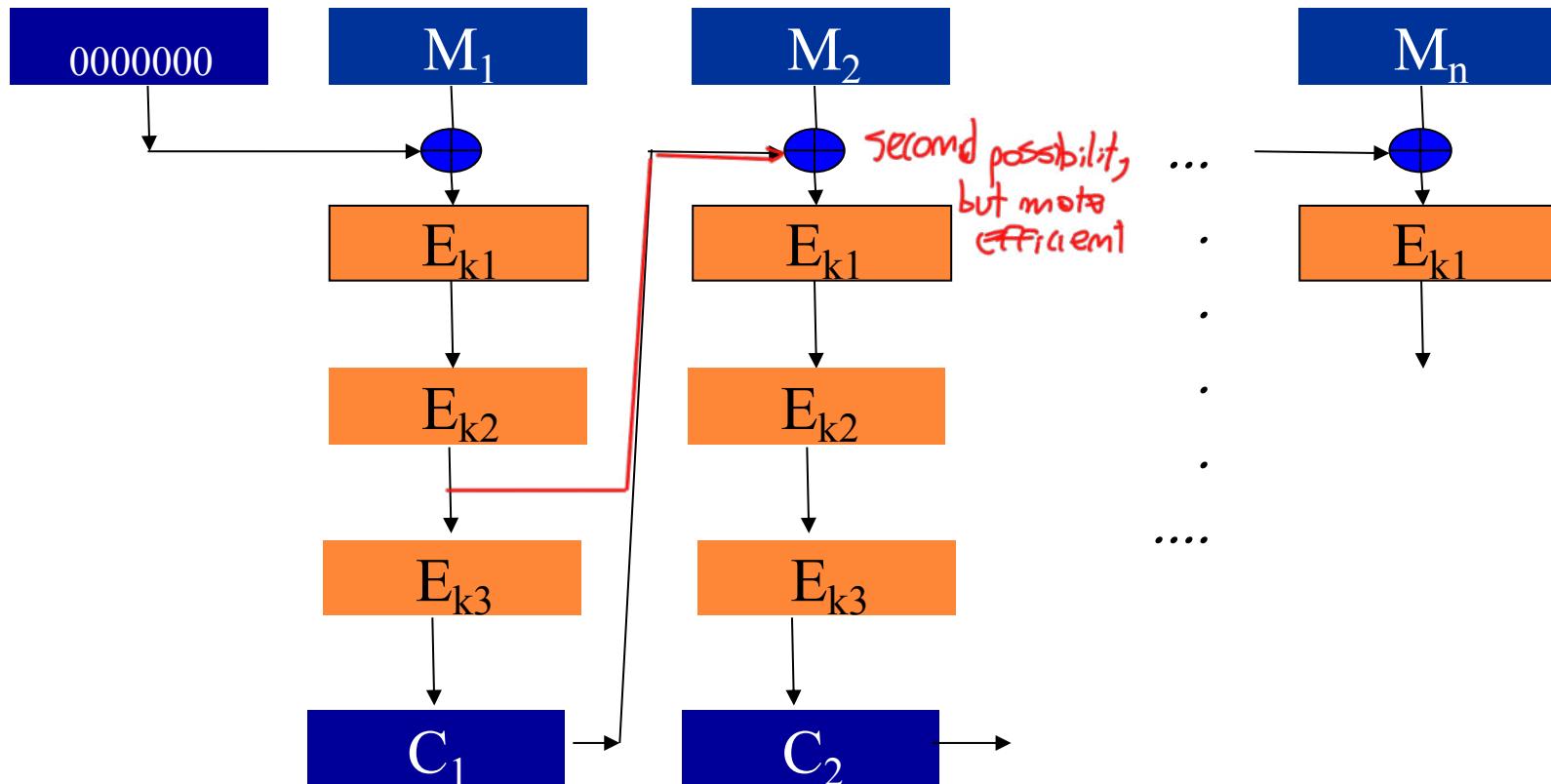
$$2^{56} \text{ generation encryption} + 2^{56} \left( \text{generation decryption} + \overset{\text{cheatable}}{\underset{\uparrow}{\text{searching}}} \right) = 2 \text{ years} + 1 \text{ year} + 2 \text{ years} +$$

$$+ \frac{\text{searching}}{\text{using GPU become possible}} = 5 \text{ years} + \frac{\text{searching}}{\text{using GPU become possible}} \rightarrow \text{days}$$

# TRIPLE ENCODING



# TRIPLE ENCODING AND CBC

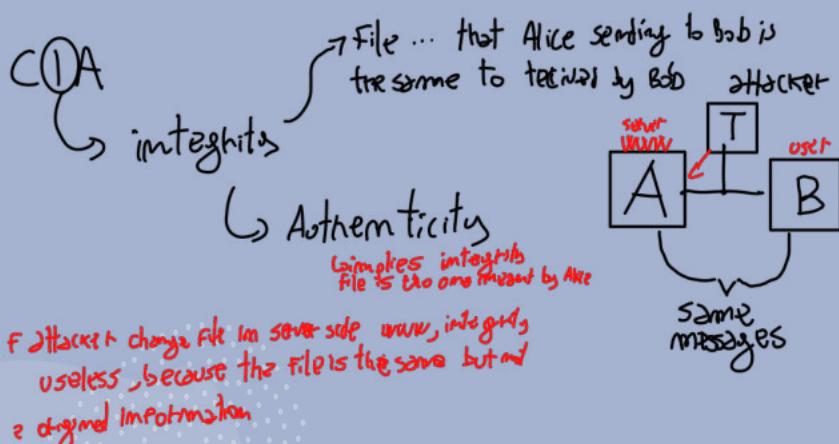


In the picture: External CBC: code (using triple encoding) each block ; then concatenate

Other possibility: Internal CBC (the concatenation is internally made, before 1<sup>st</sup> encryption and after decryption). More secure, but less used

# Data Integrity & Authenticity

## Message Authentication Codes (MACs)

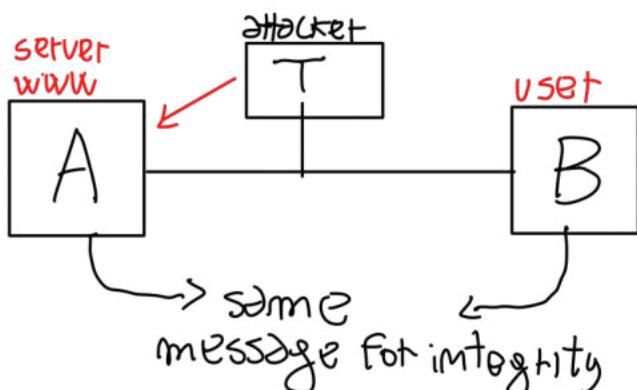


COA

↳ integrity → Authenticity

↳ message that Alice sending to Bob is the same that receive Bob

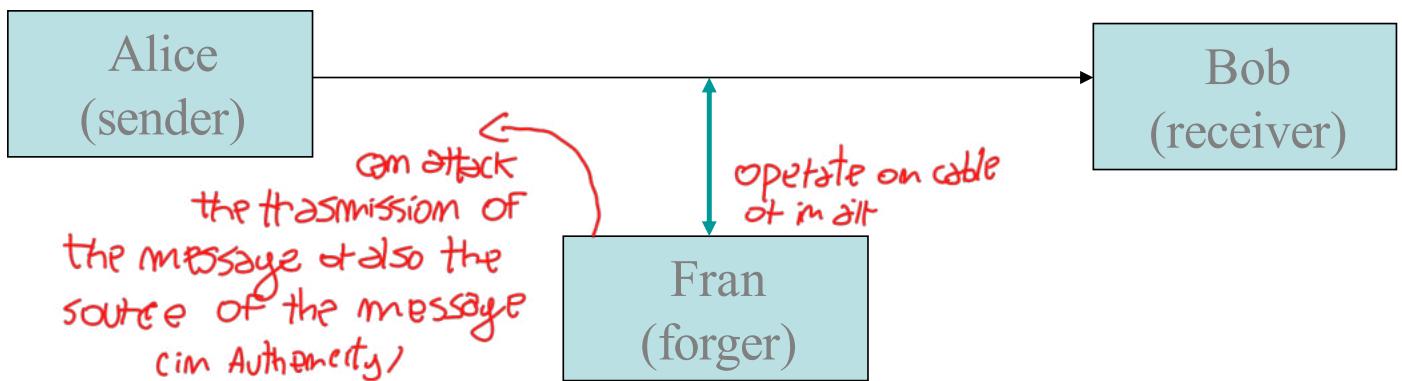
implies integrity not the vice versa



Ex.: web server is attacked by an adversary that changes the file that Bob want download in serverside. The files that arrive is the same but is not the original information. Integrity is useless here, we want also authenticity.

# Goal

Ensure integrity of messages, even in presence of an active adversary who sends own messages.



Remark: Authenticity is orthogonal to secrecy, yet systems often required to provide both.

(Confidentiality is not a requirement for now)

# Definitions

here is  
the authentication of  
the message

algorithm used by  
the sender to provide to the  
recipient additional information  
for do verification

in the web:  
Alice is the web server (sender) and  
Bob is the user, not sharing a key

$m$  and  $A_k(m)$  is sent to different  
channel, for double the effort  
for adversary

a.y. 2022-23

> Understand who is the other part, check whatever  
the other part in our communication is the part we  
believe.

- Authentication algorithm
  - A
- Verification algorithm - V  
("accept"/"reject")
- Authentication key - k
- Message space (usually  
binary strings)
- Every message between  
Alice and Bob is a pair ( $m$ ,  
 $A_k(m)$ )
- $A_k(m)$  is called the  
authentication tag of  $m$

CS #3

3

## Definitions (cont.)

the binary string is small, just 256 bits usually

- Requirement -  
 $V_k(m, A_k(m)) = \text{"accept"}$ 
  - The authentication algorithm is called MAC (Message Authentication Code)
  - $A_k(m)$  is frequently denoted  $MAC_k(m)$  *are equal*
  - Verification is by executing authentication on  $m$  and comparing with  $MAC_k(m)$

the goal of the attacker is to be able to change the message, want to be able to construct a new pair  $(m', A_k(m'))$

# Properties of MAC Functions

attacker intercept a lot of communication pair, but not know the key!

Security requirement – adversary can't construct a new legal pair  $(m, \text{MAC}_k(m))$

even after seeing  $(m_i, \text{MAC}_k(m_i))$  ( $i=1, 2, \dots, n$ )

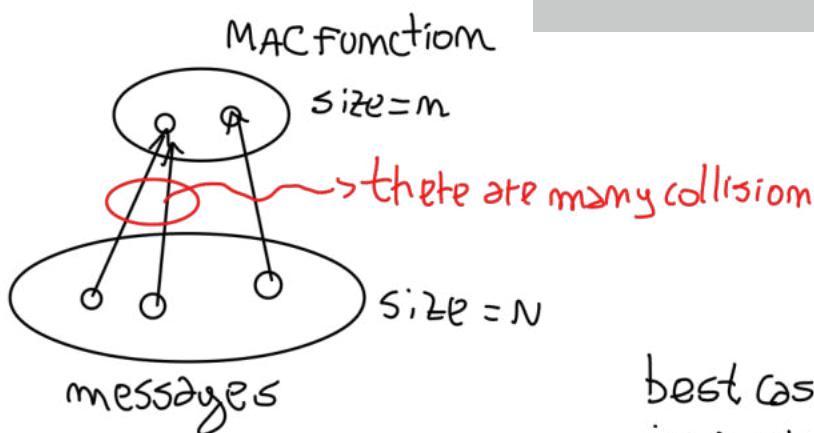
Output should be as short as possible  
↳ a standard length for all output

The MAC function is not 1-to-1

number of messages is much more of MAC function

5 a.y. 2022-23

CS #3



attacker want to replace the message with another message maybe having the same  $\text{MAC}_k$ , there are million of collision!

best case is that the collisions is  $\frac{N}{m}$  in average for every point

find a collision must be the  $\frac{N}{m}$   
(worst case is that every collision in one point)

number of collision is independent of algorithm, is not related to goodness of an algorithm, it make more difficult to find a collision.

# Adversarial Model

- Available Data:
  - The MAC algorithm
  - Known plaintext (pairs  $(m, MAC_k(m))$ ) with a key or not
  - Chosen plaintext (choose  $m$ , get  $MAC_k(m)$ ) \*
- Note: chosen MAC is unrealistic
- Goal: Given n legal pairs  
 $(m_1, MAC_k(m_1)), \dots, (m_n, MAC_k(m_n))$   
find a new legal pair  $(m, MAC_k(m))$

Change the message  
but having the same  $MAC_k(m)$

↳ a collision in message space

\* adversary can choose an arbitrary message and obtain the  $MAC_k(m)$  without knowing the key.

# Adversarial Model

- adversary succeeds even if message Fran forged is "meaningless"
- reasons: hard to predict
  - what has meaning and no meaning in an unknown context
  - how will the receiver react to such successful forgery

great success in when the message forged is not meaningless!

# Efficiency

→ Our algorithm must be robust also versus probabilistic approach. Find a collision must be hard also in probabilistic terms. A expected value must not be polynomial

- **Adversary goal:** given  $n$  legal pairs  $(m_1, MAC_k(m_1)), \dots, (m_n, MAC_k(m_n))$  find a new legal pair  $(m, MAC_k(m))$  efficiently and with non negligible probability.
- If  $n$  is large enough then  $n$  pairs  $(m_i, MAC_k(m_i))$  determine the key  $k$  uniquely (with high prob.).

# MACs Used in Practice

We describe *two approaches for MAC*

- MAC based on CBC Mode Encryption

- uses a block cipher
  - slow

- MAC based on cryptographic hash

functions

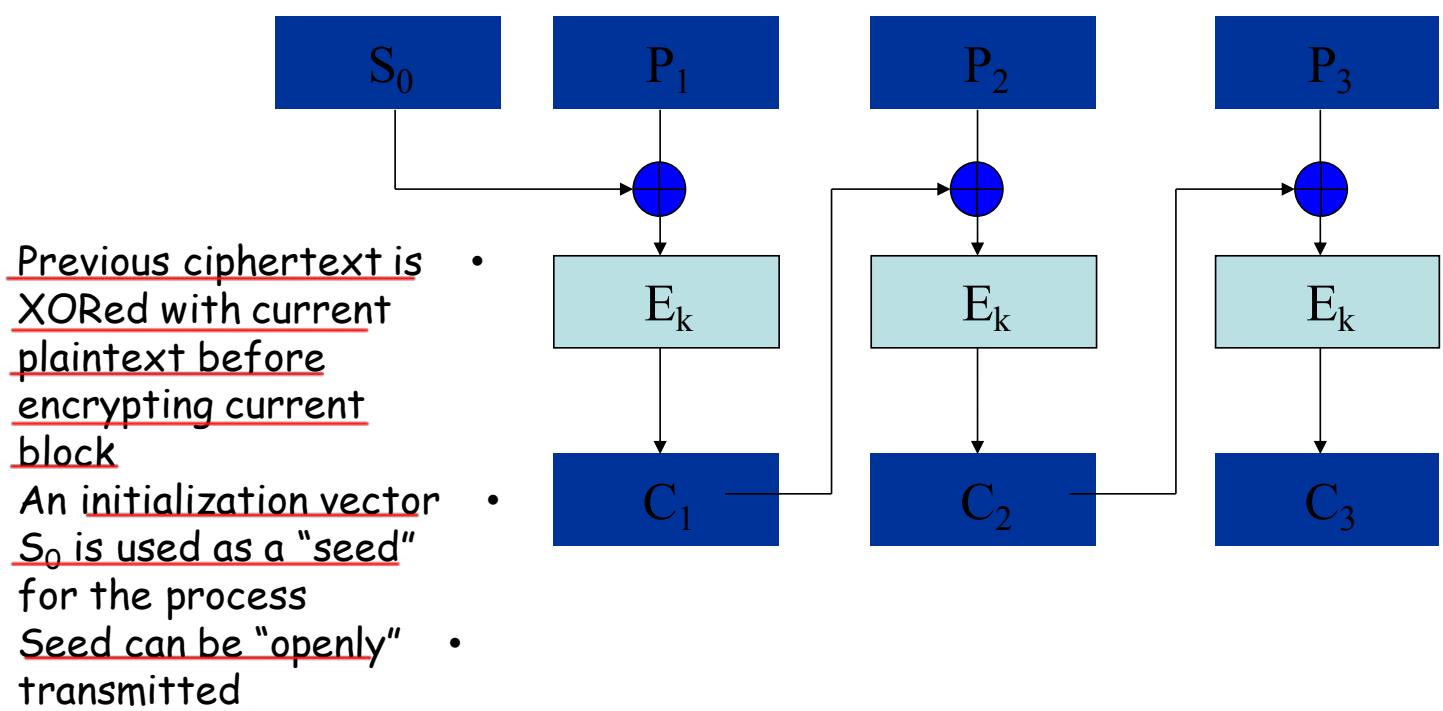
- fast

no restriction on export

↳ more generic  
approach

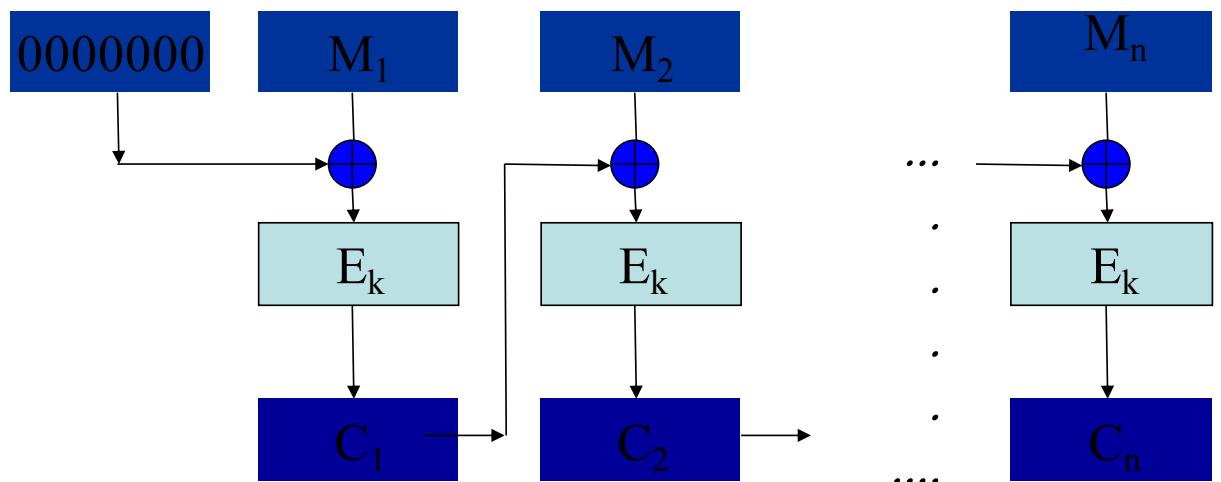
# Reminder: CBC Mode Encryption

## (Cipher Block Chaining)



# CBC Mode MACs

- Start with the all zero seed.
- Given a message consisting of  $n$  blocks  $M_1, M_2, \dots, M_n$ , apply CBC (using the secret key  $k$ ).



- Produce  $n$  "ciphertext" blocks  $C_1, C_2, \dots, C_n$ , discard first  $n-1$ .
- Send  $M_1, M_2, \dots, M_n$  & the authentication tag  $\text{MAC}_k(M) = C_n$ .

last block is the  $\text{MAC}_k(m)$ , of the cipher text.  
for Bob is easy check the tag, know the same  
secret key, redo the same encryption and  
check the result.

research discover a weakness if the length of the message  
is not fixed

# Security of CBC MAC [BKR00]

Contradiction

- Pseudo random function: a function that looks random (to any polynomial time alg.)
- Recall: a good encoding scheme transforms the message in an apparently random string

**Claim:** If  $E_k$  is a pseudo random function, then the fixed length CBC MAC is resilient to forgery. (attacks)

Proof outline: Assume CBC MAC can be forged efficiently. Transform the forging algorithm into an algorithm distinguishing  $E_k$  from random function efficiently. impossible because we assumed that this is acting as a pseudo-random function.

see file BKR2000.pdf (distributed)

**forgery**: act of changing the text of the message that you are sending.

**existential forgery**: attacker is able to provide a message meaning full or meaningless, but have success on the verification test

**selective forgery**: attacker is able not only to find a message with same property, but he can choose the message among several messages.

**universal forgery**: the strongest, adversary is able to use whatever message and find his MAC

# CBC-MAC is insecure for variable-length messages

- CBC-MAC is secure for fixed-length messages, but insecure for variable-length messages
  - if attacker knows correct message-tag pairs  $(m, t)$  and  $(m', t')$  can generate a third (longer) message  $m''$  whose CBC-MAC will also be  $t'$ 
    - XOR first block of  $m'$  with  $t$  and then concatenate  $m$  with this modified  $m'$
    - hence,  $m'' = m \parallel (m'_1 \oplus t) \parallel m'_2 \parallel \dots \parallel m'_x$

$\downarrow$   
 first block of  $m'$  is changed      II: concatenation  
 $m$        $t \oplus m' \oplus t = m'$   
 $\downarrow$        $\rightarrow m'_x$  will be same like  $m$  is  
 $t$       not concatenated

a.y. 2022-23

CS #3

13

The approach uses a secret key  $K$ , that adversary doesn't know. Adversary knows a pair  $(m, t)$  where  $t$  is the  $\text{MAC}_K(m)$ , content of last block of cipher text, knows also another pair  $(m', t')$ .

t and t' are computing with CBC with same E<sub>K</sub> and same key.

Adversary is able to produce a new message  $m''$  that will have for authentication tag  $t'$ .

not strong like selective forgets, this is the easiest

## Combined Secrecy & MAC

- Given a message consisting of n blocks  $M_1, M_2, \dots, M_n$ , apply CBC (using the secret key  $k_1$ ) to produce  $MAC_{k_1}(M)$ .

With this approach  
is easy to obtain  
integrity and  
confidentiality

- Produce n ciphertext blocks  $C_1, C_2, \dots, C_n$  under a different key,  $k_2$ .
- Send  $C_1, C_2, \dots, C_n$  & the authentication tag  $MAC_{k_1}(M)$ .

CS #3

must use two different key  
otherwise we have weakness

a.y. 2022-23

14

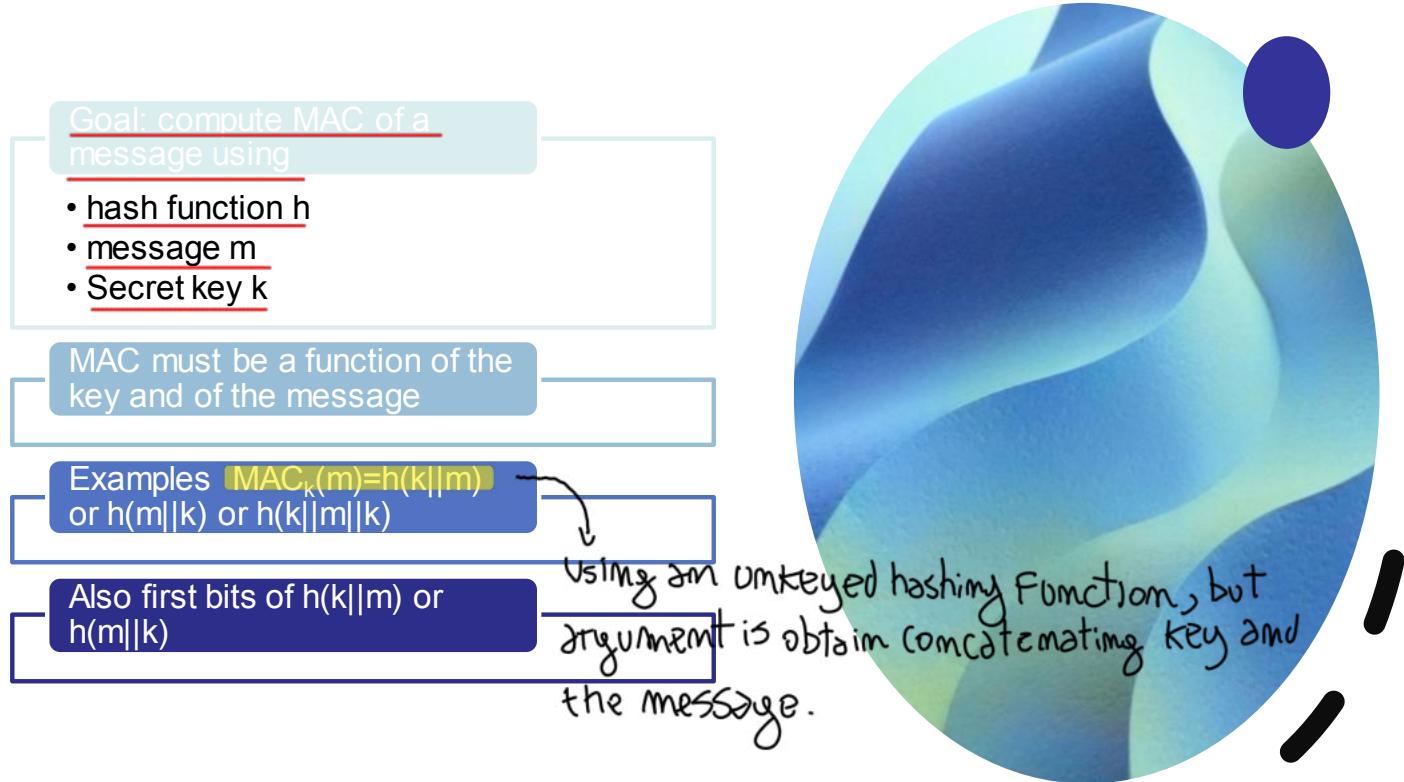
in web we have not a shared key , we use two different web server.

# Hash Functions

- Map large domains to smaller ranges
- Example  $h: \{0, 1, \dots, p^2\} \rightarrow \{0, 1, \dots, p-1\}$  defined by  $h(x) = ax+b \text{ mod } p$
- Used extensively for searching (hash tables)
- Collisions are resolved by several possible means – chaining, double hashing, etc.

We want cryptographical hash function that are a smaller set of hash function.

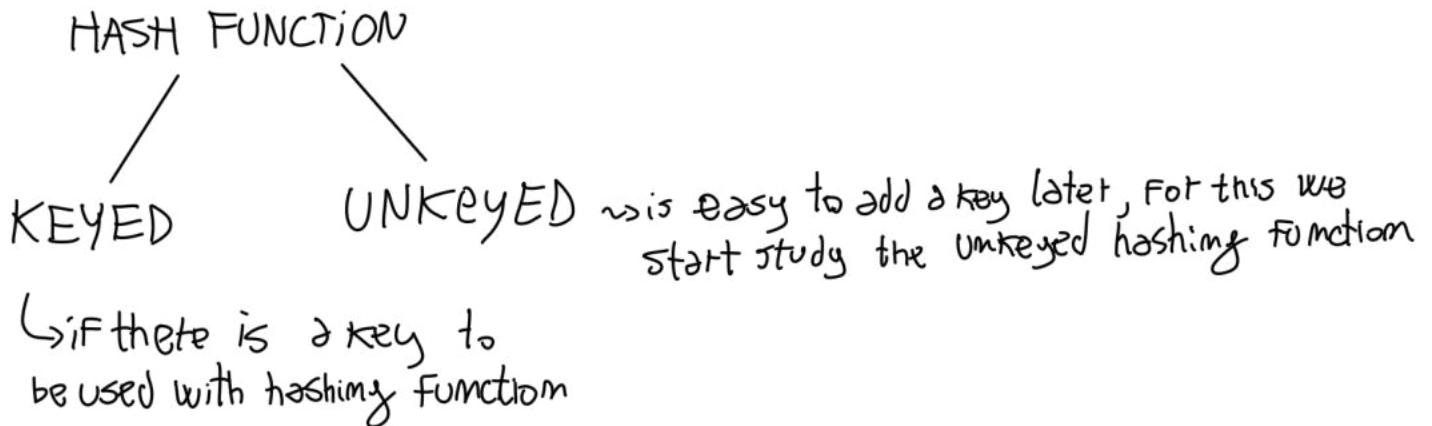
# Hash function and MAC



a.y. 2022-23

CS #3

16



for exam important

hash value =  $\text{fingerprint}, \text{digest}$   
↳ result of hash

# Collision Resistance

↑ DOMAIN      ↑ RANGE

A hash function  $h: D \rightarrow R$  is called weakly collision resistant for  $x \in D$  if it is hard to find  $x' \neq x$  such that  $h(x') = h(x)$   
↳ given  $x$  find the collision  $x'$

A function  $h: D \rightarrow R$  is called strongly collision resistant if it is hard to find  $x, x'$  such that  $x' \neq x$  but  $h(x) = h(x')$

↳ here not given  $x$ , choose what you want

Note: if you find collision then you might be able to find two messages with the same MAC

a.y. 2022-23

CS #3

17

Collision Resistance is necessary for cryptographic hashing function, but not sufficient. It is necessary strongly collision resistant

strongly collision  
resistant  $\xrightarrow{\quad}$  weakly collision  
resistant  
~~not viceversa~~

hard meaning that it is not polynomial, also in the average case.

# strong $\Rightarrow$ weak

asked in the exam

↑ the same

- proof: we show ( $\neg$ weak  $\Rightarrow$   $\neg$ strong)
- given  $h$ , suppose there is poly alg.  $A_h$ :  
 $A_h(x) = x'$  s.t.  $h(x) = h(x')$   $\rightarrow h$  is not weak
- we construct poly alg.  $B_h$  s.t.  
 $B_h() = (x, x')$  s.t.  $h(x) = h(x')$ :
  - randomly choose  $x$
  - return  $(x, A_h(x))$

how to do it

Suppose for contradiction  $h$  is not weak, and  
a.y. 2022-23 exist a poly  $A_h$  for CS #3 find a collision.

18

We build another poly alg.  $b_h$  that produce another pair colliding  
not strongly collision resistance

We find a pair  $(x, A_h(x))$  violating strongly collision resistance

strongly collision resistant is necessary, and also not invertible  
for having a cryptographic hashing function.

not invertible: given result of hash is hard to find the original  
value which have started. Given  $A_h(x)$  it is hard to find  $x$

third condition for be cryptographic is the set of the value (tange) of the  
hashing function should be not too short, long enough

today we know zero cryptographic hashing function!

# The Birthday Paradox

as long all birth-day have the same probability

- If 23 people are chosen at random the probability that two of them have the same birth-day is greater than 0.5
- More generally, let  $h: D \rightarrow R$  be any mapping. If we choose  $1.1774|R|^{1/2}$  elements of  $D$  at random, the probability that two of them are mapped to the same image is 0.5. (mapped with a collision)
  - the expected number of choices before finding the first collision is approximated by  $(\frac{1}{2} \pi |R|)^{1/2}$

probability of a collision only depend on the size of the range  $R$ , not the size  $D$  (messages)

$\hookrightarrow D$  must be much greater of  $R$ , otherwise is easy to do Brute Force CS #3 attack

c. hashing function have collisions, is only hard find them

# Birthday attack

- Given a function  $f$ , find two different inputs  $x_1, x_2$  such that  $f(x_1) = f(x_2)$ . *collision*
- E.g., if a 64-bit hash is used, there are approximately  $1.8 \times 10^{19}$  different outputs. If these are all equally probable, then it would take approximately  $5.38 \times 10^9$  (expected) attempts to generate a collision using brute force ( $5.1 \times 10^9$  is the number that makes probability = 0.5). This value is called **birthday bound**

↪ want big this limit for robustness to  
birthday attack, brute force

# Cryptographic Hash Functions

Cryptographic hash functions are hash functions that are:

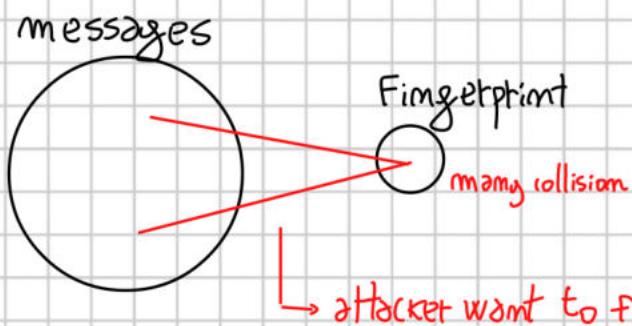
- strongly collision resistant;
  - hard to invert; (given the image it is hard to find the counter-image)
- 
- Notice: No secret key.
  - Should be very fast to compute, yet hard to find colliding pairs (impossible if  $P \neq NP$ ).
  - Usually defined by:
    - Compression function mapping  $n$  bits (e.g., 512) to  $m$  bits (e.g., 160),  $m < n$ .

most famous today is password based key derivation function (bcrypt)

important to use two different channel  
for send The pair, message and hashing function

Birthday bound  $\cong \sqrt{\text{Space size}}$  (independent from type of hashing functions)  
 (BD)  $\hookrightarrow$  of hash value (or fingerprint)

If fingerprint  $| = 160$  (used by SHA1)



$\hookrightarrow$  today the size is too small we have  $2^{160}$  different fingerprints

$\hookrightarrow$  for BD bound  $2^{160} \rightarrow 2^{80}$  that is attackable.

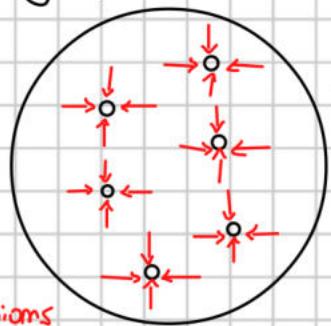
$\hookrightarrow$  attacker want to find a collision

with brute force is needed to generate all possible messages until find a pair that colliding, there are many. But with BD Bound finding a collision (Probability  $\geq \frac{1}{2}$ ) we have explore a space that is  $\sqrt{|tangle|}$ , not important the size of the set of messages, only important size of fingerprints. This size must be not too small, but also not too big.

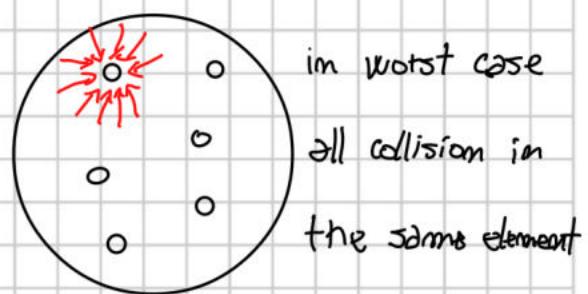
Attacker when finding a collision don't fix a message, but generate different messages until find a pair colliding, meaningless but is important.

For exam: i have a pair  $(m, h(m))$ , assume that  $h$  is a cryptographic hashing function, For finding  $m'$  with same  $h(m)$  we can't use BD bound, you can't decide the colliding messages. You have store messages and check fingerprint for everyone.

fingerprints



good hashing function there are many collision, but in all elements.



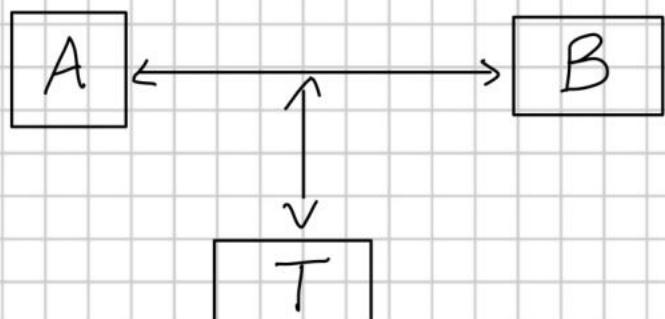
in worst case all collision in the same element

cryptographic hashing functions are not strong against BD bound, are h.f. that have size greater of 160, not invertible and collision resistant.

exam:  $h(x) = x \bmod p$ ,  $p$  is a big number prime have a problem: is very easy to find a collision because every  $x$  collide with  $x+p$ , a cryptographic hashing function must be hard find a collision.

unkeyed hashing function  $h$  can become easily keyed

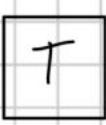
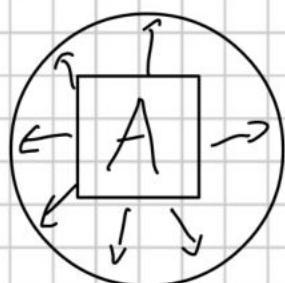
$$h(K || m) = h'_K(m)$$



attacker can listen the communication  
and change information, power of  
MITM

in web hashing value and message must go in two different channels.

also MITM:

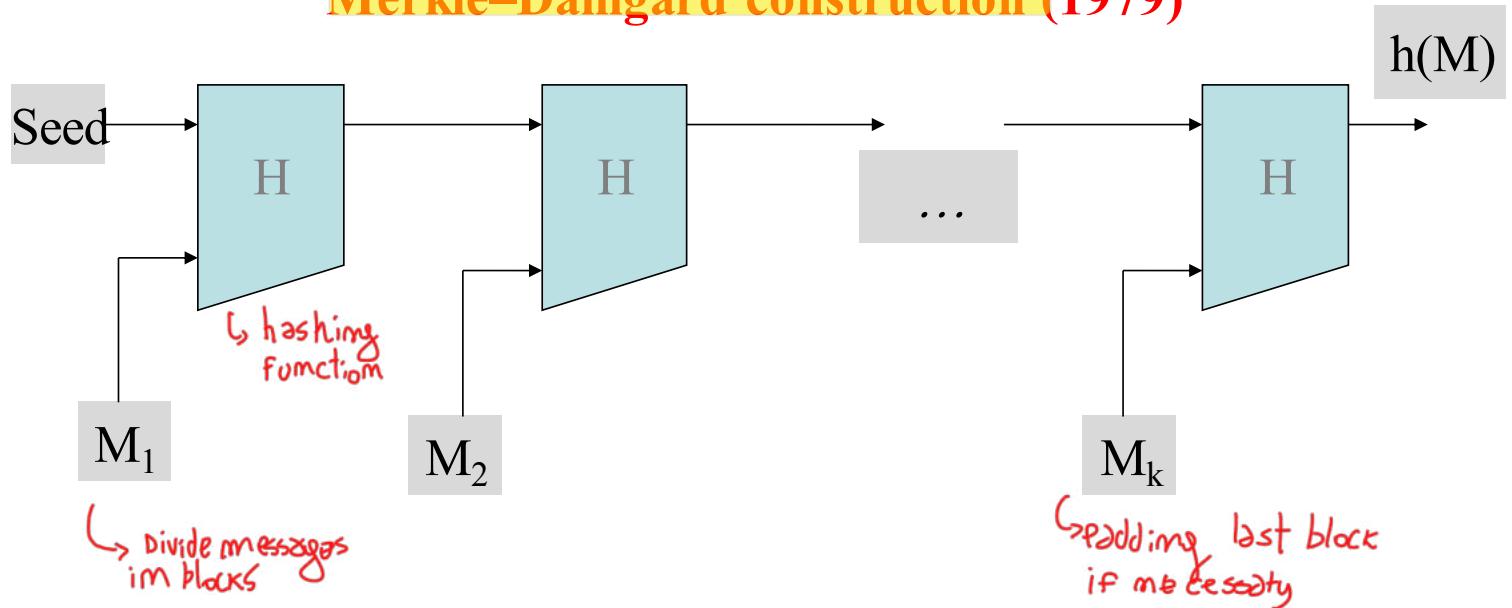


attacker control everything  
that send and receive Alice,  
(EX. control the router)

# Extending to Longer Strings

can handle any length of messages

Merkle-Damgård construction (1979)



$$H : D \rightarrow R \text{ (fixed sets, typically } \{0,1\}^n \text{ and } \{0,1\}^m \text{ )}$$

a.y. 2022-23

CS #3

22

these block hashing function have vulnerability

adversary without knowing the key can add blocks, and obtain result, key influence only first block

## Extending the Domain (cont.)

- The seed is usually constant
- Typically, padding (including text length of original message) is used to ensure a multiple of n.
- **Claim:** if the basic function H is collision resistant, then so is its extension.

# Lengths

a.y. 2022-23

- Input message length should be arbitrary. In practice it is usually up to  $2^{64}$ , which is good enough for all practical purposes.
- Block length is usually 512 bits.
- Output length should be at least 160 bits to prevent *birthday attacks*.

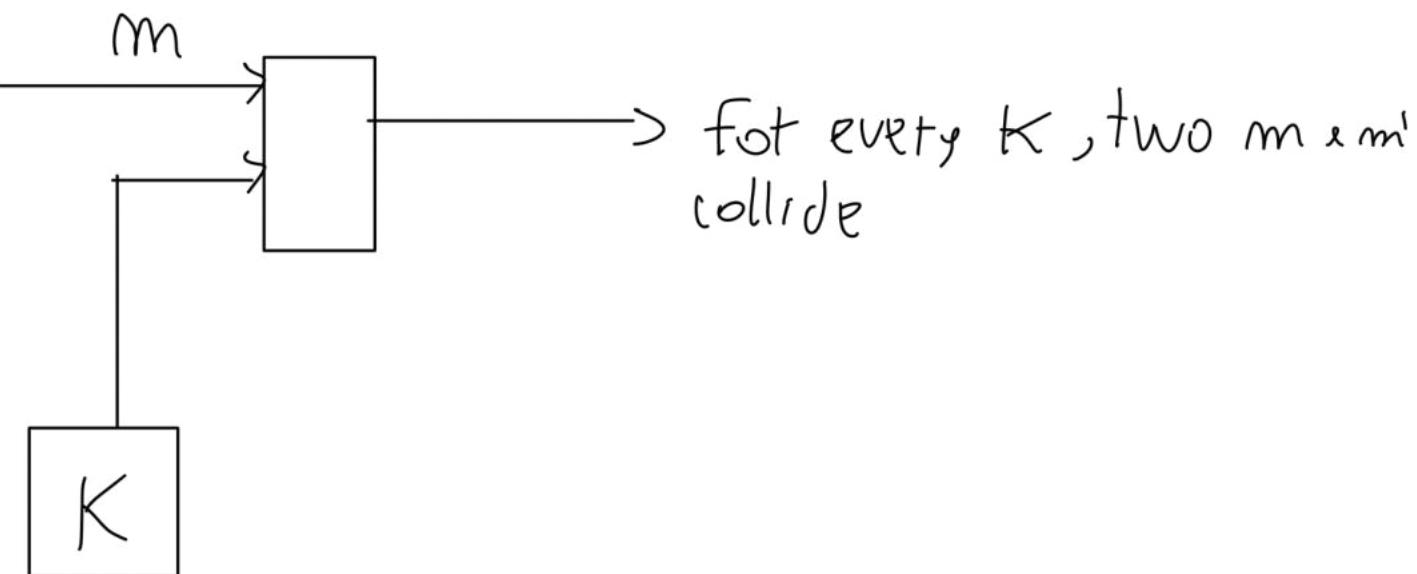
CS #3

24

not good for keyed hashing function

# Basing MACs on Hash Functions

- combine message and secret key, hash and produce MAC (keyed hashing)
  - $\text{MAC}_k(m) = h(k||m)$  KO adv. can add extra bits to message and compute correct MAC (knowledge of  $k$  not necessary, it suffices to add extra  $h$  blocks)
  - $\text{MAC}_k(m) = h(m||k)$  small problem: the adv. can exploit the birthday paradox; in fact assume  $k$  is the last block; then if adv. finds two colliding messages then she knows two messages with the same MAC - for all keys
  - $\text{MAC}_k(m) = h(k||m||k)$ : OK (similar to HMAC)
  - $\text{MAC}_k(m) = \text{first bits (e.g. first half)} \text{ of } h(k||m) \text{ or } h(m||k)$ : OK (adversary is not able to check correctness)



$h(m) \xrightarrow{\text{transform}} f_k(m)$   
In Keyed F.F.

BD attack is valid on  $f_k(m)$ , but attacker not know key

$$h(\underline{\text{"house"} \parallel \text{"meet at midday"}}) = y$$

$f_k$  ↗      ↑  
                |

attacker can find  $h(s) = y$  with some effort of an unkeyed hashing function, a collision. But s normally don't have the same structure and attacker can't check if is valid collision or not, must know that start key is "house". BD attack is possible, but can't check the validity of collision result depending on the key

# Cryptographic Hash Functions

- MD family ("message digest"), MD-4, MD-5: broken
  - SHA-0 [1993] SHA-1 [1995] (secure hash standard, 160 bits)  
([www.itl.nist.gov/fipspubs/fip180-1.htm](http://www.itl.nist.gov/fipspubs/fip180-1.htm)) *most used*
  - RIPE-MD, SHA-2 256, 384 and 512 [2001] (proposed standards, longer digests, use same ideas of SHA-1)
- Idea of SHA-1: divide the message in block
- perform several rounds (say 80) on each block
  - each round mixes changes and shuffles bit of the block
  - at the end what you get looks like a random string
- SHA-3 224, 256, 384, 512 [2012]

SHA is a family of hashing function keyed

a.y. 2022-23

CS #3

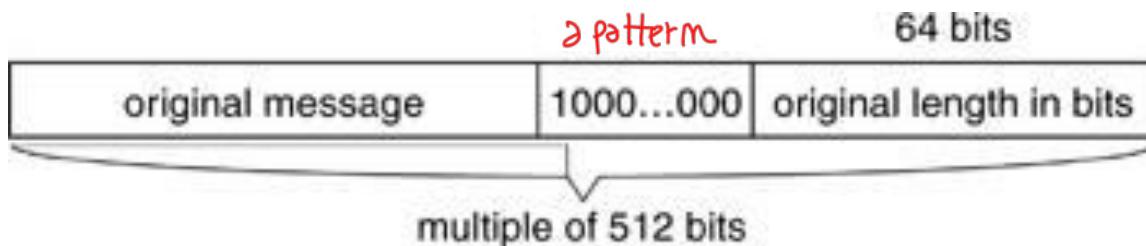
26

↳ 256 is the output of f.f., the fingerprint size

# SHA-1 basics

Mo longer cryptographic

- similar to MD4 & MD5
- $|message| \leq 2^{64}$ ,  $|digest| = 160$ ,  
 $|block| = 512$
- original message is padded



# SHA-1 overview

- The 160-bit message digest consists of five 32-bit words: A, B, C, D, and E.
- Before first stage:  $A = 67452301_{16}$ ,  $B = efcdab89_{16}$ ,  
 $C = 98badcfe_{16}$ ,  $D = 10325476_{16}$ ,  $E = c3d2e1f0_{16}$ .
- After last stage  $A|B|C|D|E$  is message digest



# SHA-1: processing one block

## Block (512 bit, 16 words)

- 80 rounds: each round modifies the buffer (A,B,C,D,E)

## Round:

$$(A, B, C, D, E) \leftarrow$$

$$(E + f(t, B, C, D) + (A \ll 5) + W_t + K_t), A, (B \ll 30), C, D)$$

- t number of round,  $\ll$  denotes left shift
- $f(t, B, C, D)$  is a complicate nonlinear function
- $W_t$  is a 32 bit word obtained by expanding original 16 words into 80 words (using shift and ex-or)
- $K_t$  constants

$$K_t = \lfloor 2^{30} \sqrt{2} \rfloor = 5a827999_{16} \quad (0 \leq t \leq 19)$$

$$K_t = \lfloor 2^{30} \sqrt{3} \rfloor = 6ed9eba1_{16} \quad (20 \leq t \leq 39)$$

$$K_t = \lfloor 2^{30} \sqrt{5} \rfloor = 8f1bbcd\bar{c}_{16} \quad (40 \leq t \leq 59)$$

$$K_t = \lfloor 2^{30} \sqrt{10} \rfloor = ca62c1d6_{16} \quad (60 \leq t \leq 79)$$

# function f

$f(t, B, C, D) =$

- $(B \wedge C) \vee (\neg B \wedge D)$   $(0 \leq t \leq 19)$
- $B \oplus C \oplus D$   $(20 \leq t \leq 39)$
- $(B \wedge C) \vee (B \wedge D) \vee (C \wedge D)$   $(40 \leq t \leq 59)$
- $B \oplus C \oplus D$   $(60 \leq t \leq 79)$

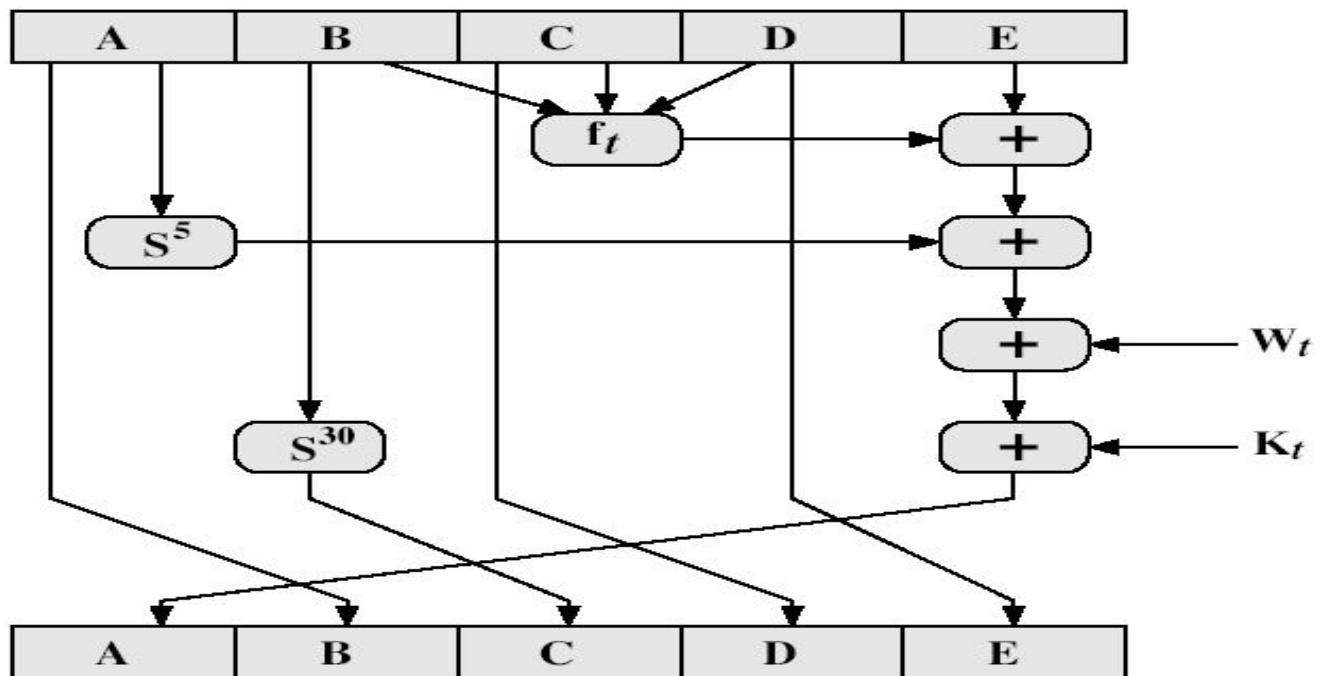
## word $w_t$

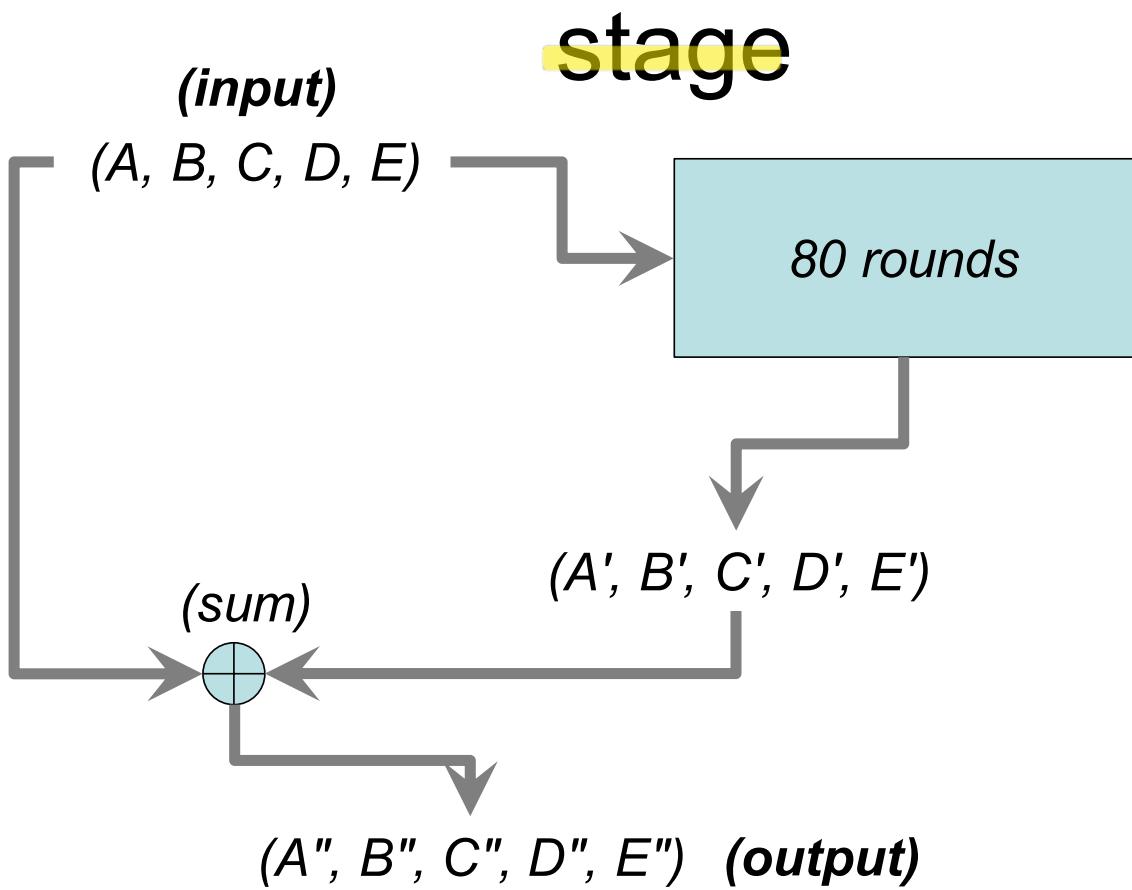
$w_0 \dots w_{15}$  are the original 512 bits

for  $16 \leq t \leq 79$

- $w_t = (w_{t-3} + w_{t-8} + w_{t-14} + w_{t-16}) \ll= 1$
- " $\ll=$ " denotes left bit rotation

# SHA-1: round t





# SHA-1 summary

1. Pad initial message: final length must be  $\equiv 448 \pmod{512}$  bits
2. Last 64 bit are used to denote the message length
3. Initialize buffer of 5 words (160-bit)  
(A,B,C,D,E) (67452301, efcdab89,  
98badcfe, 10325476, c3d2e1f0)
4. Process first block of 16 words (512 bits):
  - 4.1 expand the input block to obtain 80 words block W<sub>0</sub>,W<sub>1</sub>,W<sub>2</sub>,...,W<sub>79</sub> (ex-or and shift on the given 512 bits)
  - 4.2 initialize buffer (A,B,C,D,E)
  - 4.3 update the buffer (A,B,C,D,E): execute 80 rounds  
each round transforms the buffer
  - 4.4 the final value of buffer (H<sub>1</sub> H<sub>2</sub> H<sub>3</sub> H<sub>4</sub> H<sub>5</sub>) is the result
5. Repeat for following blocks using initial buffer (A+H<sub>1</sub>, B+H<sub>2</sub>,...)

# HMAC

Prefeted keyed hashing Function

- Proposed in 1996 [Bellare Canetti Krawczyk]
  - Internet engineering task force RFC 2104
  - FIPS standard (uses a good hash function)
- Receives as input a message  $m$ , a key  $k$  and a hash function  $h$
- Outputs a MAC by:
  - $\text{HMAC}_k(m, h) = h(k \oplus \text{opad} \parallel h(k \oplus \text{ipad} \parallel m))$
  - $\text{opad}$  (outer padding: 0x5c5c5c...5c5c, one-block-long)
  - $\text{ipad}$  (inner padding: 0x363636...3636, one-block-long)
- Theorem [BCK96]: HMAC can be forged if and only if the underlying hash function is broken (collisions found). [Bellare, Canetti, Krawczyk, 1996: <http://cseweb.ucsd.edu/users/mihir/papers/hmac-cb.pdf>]

most used  
 $h$  is SHA-2 256

→ double hashing, there is the message

} chosen by community

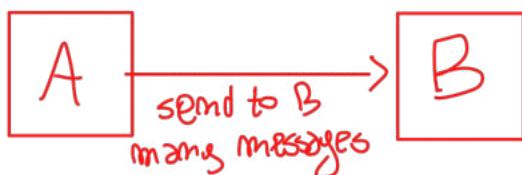
a.y. 2022-23

CS #3

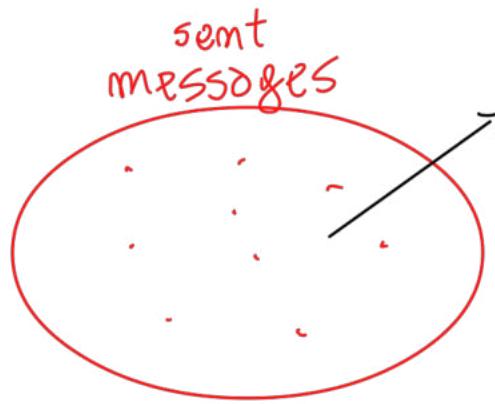
37

$h_k(m)$   
attacker is unknown to adversary

→ when attacker find a collision can't check if is valid, contain the correct key.  
BD attack is very difficult



attacker collect all messages, that contain the same unknown key. These messages



if this number of messages is big enough, the attacker can use the BD bound.  
you try  $2^{n/2}$  messages for have prob  $\geq \frac{1}{2}$  of have a collision (m size of finger print).  
↳ after find a collision in this set the collision is securely valid, share all the same key, but collect this number of messages is unpractical.

# HMAC - Birthday paradox

- Adversary wants to find two messages  $m, m'$  s.t.  $\text{HMAC}_k(m,h) = \text{HMAC}_k(m',h)$ ; Adversary knows IV and  $h$  adv. does not know  $k$
- Birthday paradox holds (you expect to find collisions with  $2^{n/2+1}$  test) but the adv. is not able to check success:
  - Adv. does not know  $k$  hence he cannot generate authentic messages;
  - He must listen  $2^{n/2+1}$  messages obtained with the same key (ex.  $n=128$  at least  $2^{64} + 1$ ) to have prob.  $> 0.5$  of a collision
- Note birthday paradox does not help the adv. also if we use  $\text{hash}(k||m||k)$

# HMAC in Practice

a.y. 2022-23

- FIPS standard
- SSL / TLS
- WTLS (part of WAP stack)
- IPSec:
  - AH
  - ESP

CS #3

39

## Authenticated Encryption (AE)

a.y. 2022-23

CS #3

40

- forms of encryption which simultaneously assures the confidentiality and authenticity of data
- often offered as single primitive in modern APIs
- makes chosen-ciphertext attack less dangerous
  - attacker is not able to choose a ciphertext and present it to the decryption in a proper way

approach of last years, encryption is the same symmetric encryption, but with authenticated allow to obtain confidentiality with data integrity (or authentication)

# typical API

a.y. 2022-23

## Encryption

- Input: plaintext, key, and optional additional plaintext (header) that will be only authenticated
- Output: ciphertext and authentication tag

## Decryption

- Input: ciphertext, key, authentication tag, and optional header (if used in encryption)
- Output: plaintext, or error if authentication tag does not match ciphertext or header

The header is intended to provide authenticity and integrity protection for networking or storage metadata for which confidentiality is unnecessary

CS #3

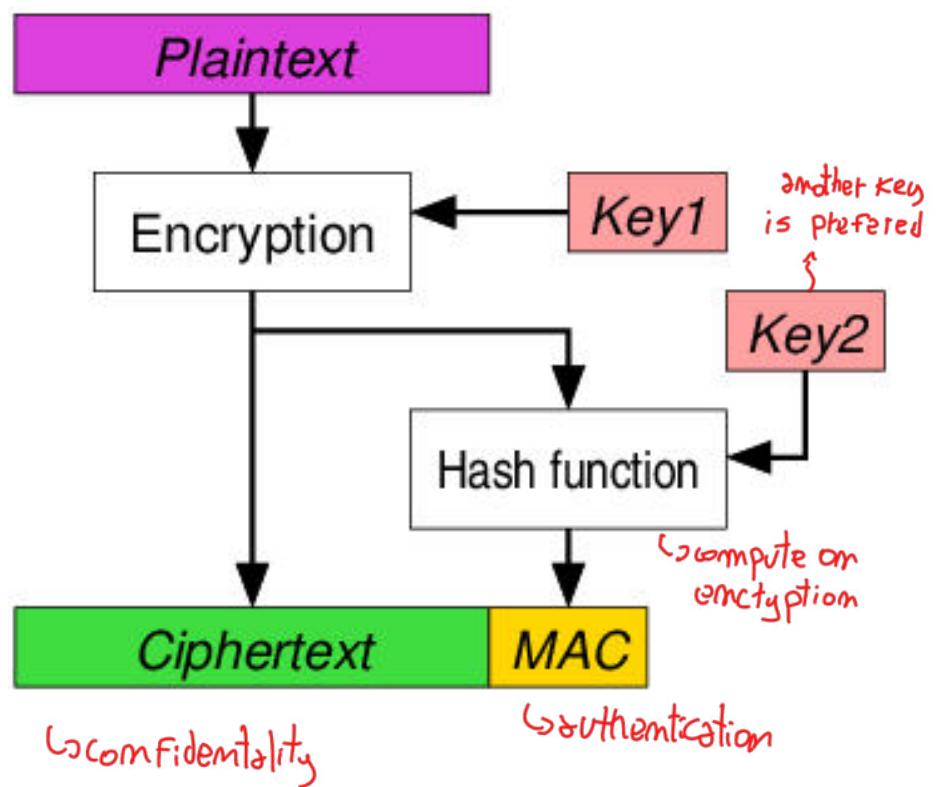
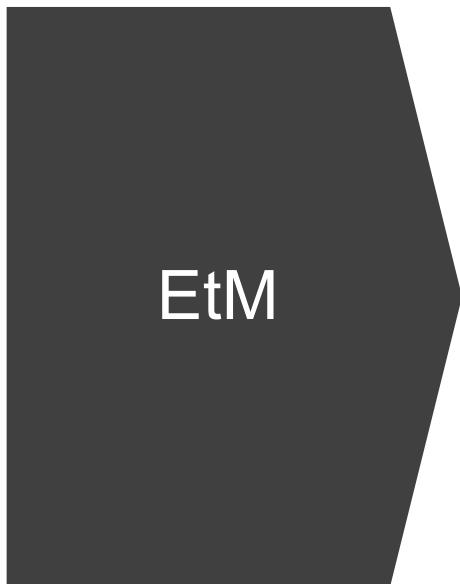
41

don't want encrypt header of a packet, encrypt all packet minus header

## approaches to AE

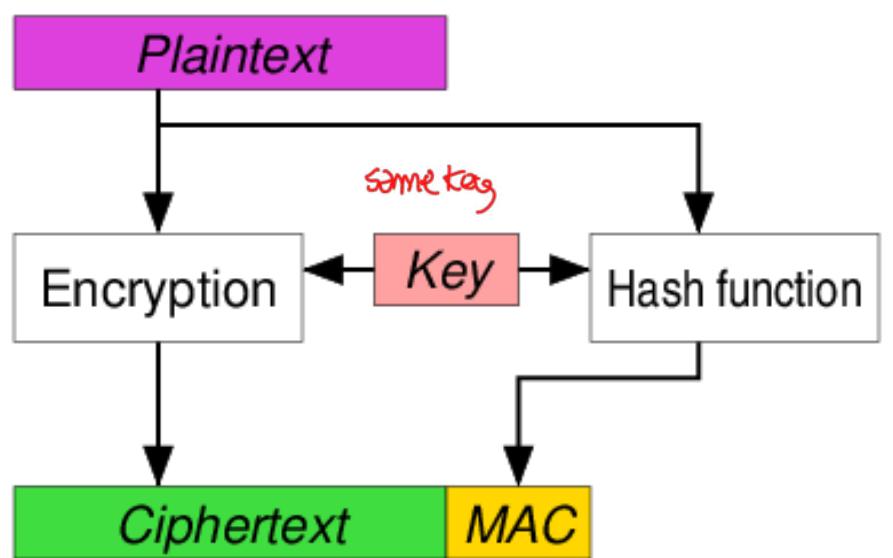
- Encrypt-then-MAC (EtM)
  - as far as we know today, the most secure, *proven theoretical*
- Encrypt-and-MAC (E&M)
  - open problem to prove the security
- MAC-then-Encrypt (MtE)
  - proven to be secure in some specific setting, otherwise open problem

← encrypt them  
MAC



## Encryption and MAC

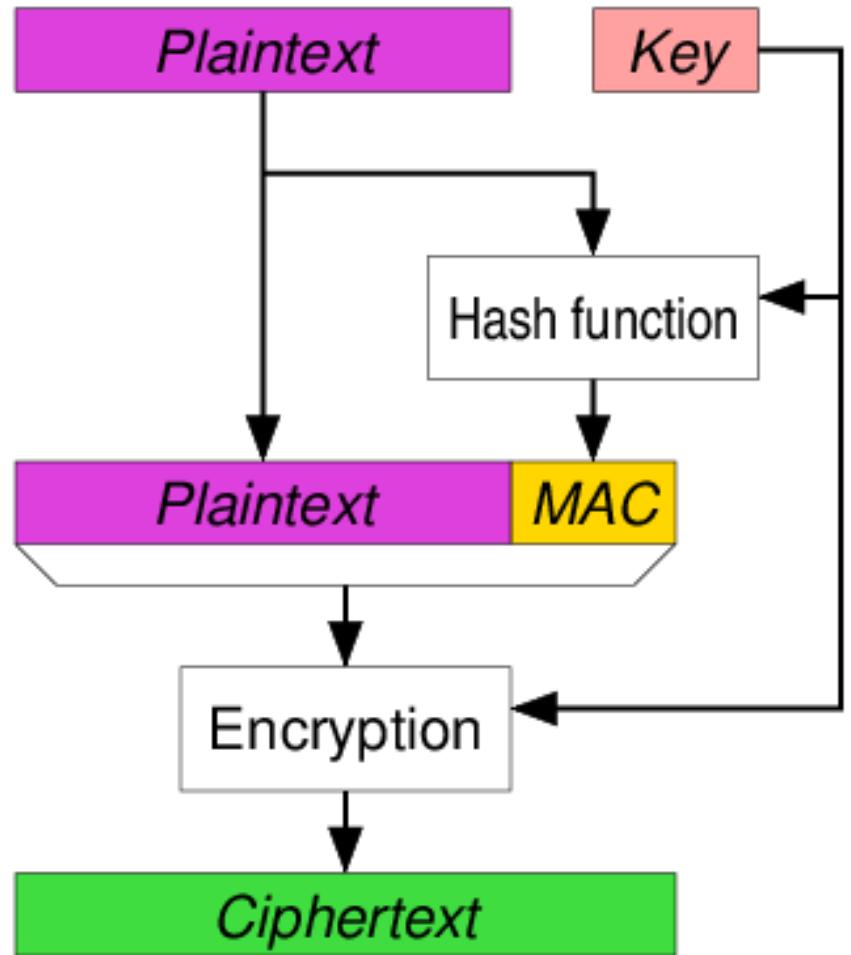
E&M



↳ computed by the plaintext and not like before from ciphertext

MAC then Encrypt

MtE



CS #3

concatenate plaintext  
and MAC ; after encrypt all

a.y. 2022-23

45

for attacker is hard to execute a chosen ciphertext attack, because he choose the ciphertext, but can't check MAC is wrong or not

GCM allowing to have AE

# Authenticated encryption with associated data (AEAD)

- associated data is plaintext not to be encrypted, but only authenticated
  - so that attempts to "cut-and-paste" a valid ciphertext into a different context are detected and rejected.
- required, for example, by network packets
  - header needs integrity, but must be visible
  - payload needs both integrity and confidentiality
  - both need authenticity

Want to defend against spoofing attack,  
Want header is authentic

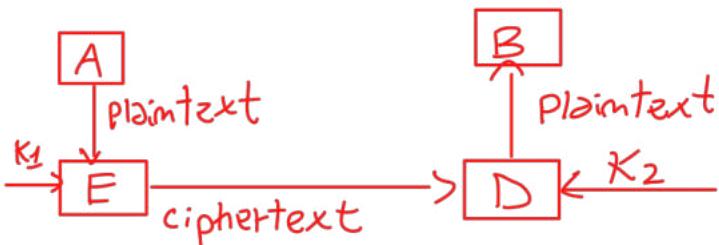
# Public-Key Cryptography



The new era (1976-present)

for obtain confidentiality we use symmetric encryption, when talk about encryption is always symmetric

- Asymmetric encryption is used for other purpose like digital signature, but not for confidentiality.

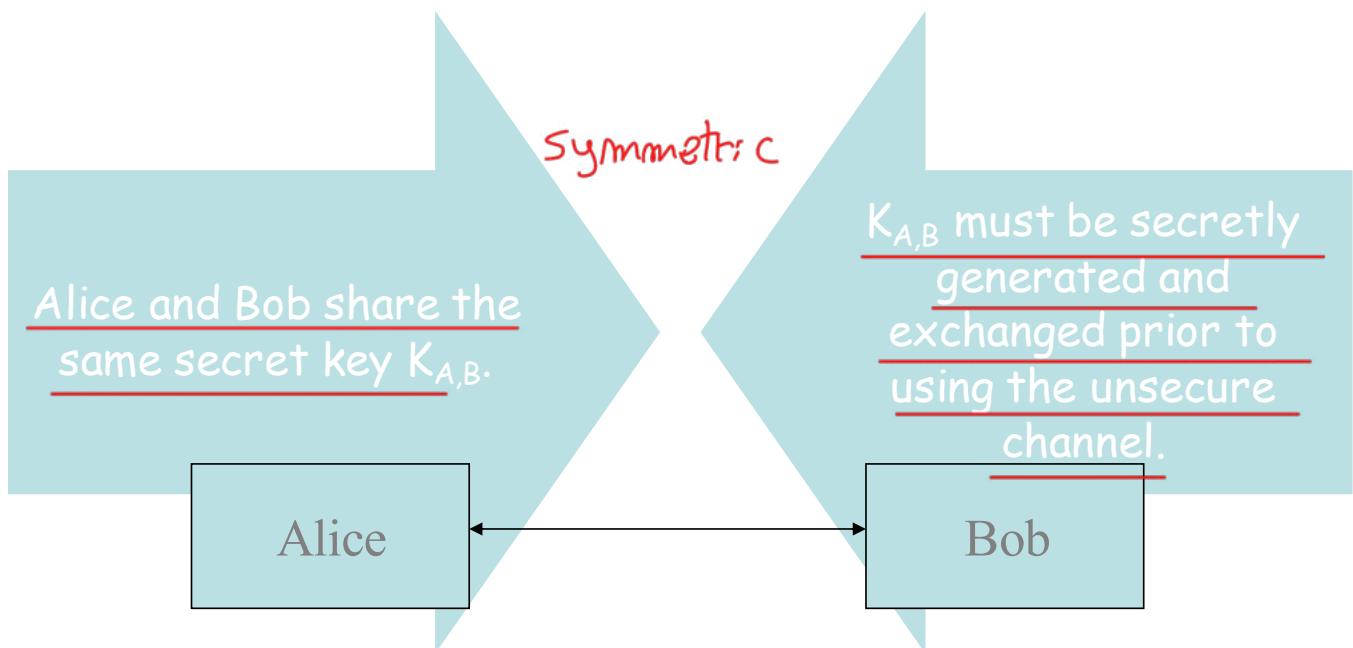


symmetric: if  $K_2 = K_1$ , same key in E and D

asymmetric: if  $K_2 \neq K_1$ , different key in E and D

$\rightarrow K_2 \neq K_1$  in asymmetric, but must be somewhat related  
the problem was studied by Diffie and Hellman, and  
resolved years later with "Public key" approach.  
it is the only approach we know that lead to the  
confidentiality and not repudiation (digital signature)  
prove identity.

# Classical, Symmetric Ciphers



a.y. 2022-23

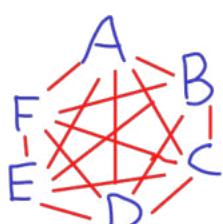
public key crypto

2

Diffie-Hellman = DH

A      B      → number of actors that want to communicate  
F      K      with confidentiality, need encryption, they  
E      C      have to share a key, but it isn't safe to  
D           share the same key. If attacker find the key  
all communication be broken.

must be  $\Rightarrow$  different key in each pair.



we have a complete graph, and want in each edge a different key

$m$  actors, ~~keys~~ keys?  $\rightarrow \frac{m(m-1)}{2}$  different keys

## Diffie and Hellman [1976]

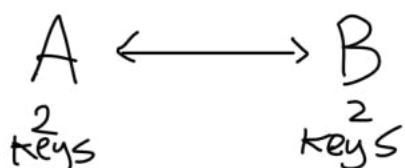
### "New Directions in Cryptography"

Split the Bob's secret key K into two parts:

- $K_E$  to be used for encrypting messages to Bob.
- $K_D$  to be used for decrypting messages by Bob.
- $K_E$  can (must) be made public
  - public key cryptography, asymmetric cryptography



in asymmetric encryption each actors have 2 keys.



We don't want a quadratic number of keys, in "public key" approach we have constant key, 2 keys for actor, number of key grow linear

2 keys:

- PRIVATE KEY → must be secret to other
- PUBLIC KEY → HD want this key public, who ever know this key

Possible relationship between PRIVATE KEY and PUBLIC KEY:

$$D_{\text{private}}(C) = M$$

$$E_{\text{public}}(M) = C$$

$$D_{\text{public}}(C) = M$$

$$D_{\text{private}}(M) = C$$

# "New Directions in Cryptography"

- The Diffie-Hellman paper [IEEE Transactions on Information Theory, vol. IT-22, Nov. 1976] generated lot of interest in crypto research in academia and private industry
  - <http://www-ee.stanford.edu/%7Ehellman/publications/124.pdf>
- Diffie & Hellman produced the revolutionary idea of public key cryptography, but did not have a proposed implementation (this came up two years later with Merkle-Hellman and Rivest-Shamir-Adelman)
- In their '76 paper, Diffie & Hellman did invent a method for key exchange over insecure communication lines, a method that is still in use today

a.y. 2022-23

public key crypto

4

$$\textcircled{1} \quad E_{\text{public}}(M) = C \longrightarrow \text{everybody can do encryption because key is public, also } E \\ D_{\text{private}}(C) = M \longrightarrow \text{only one person can decrypt, private key is known only by one}$$

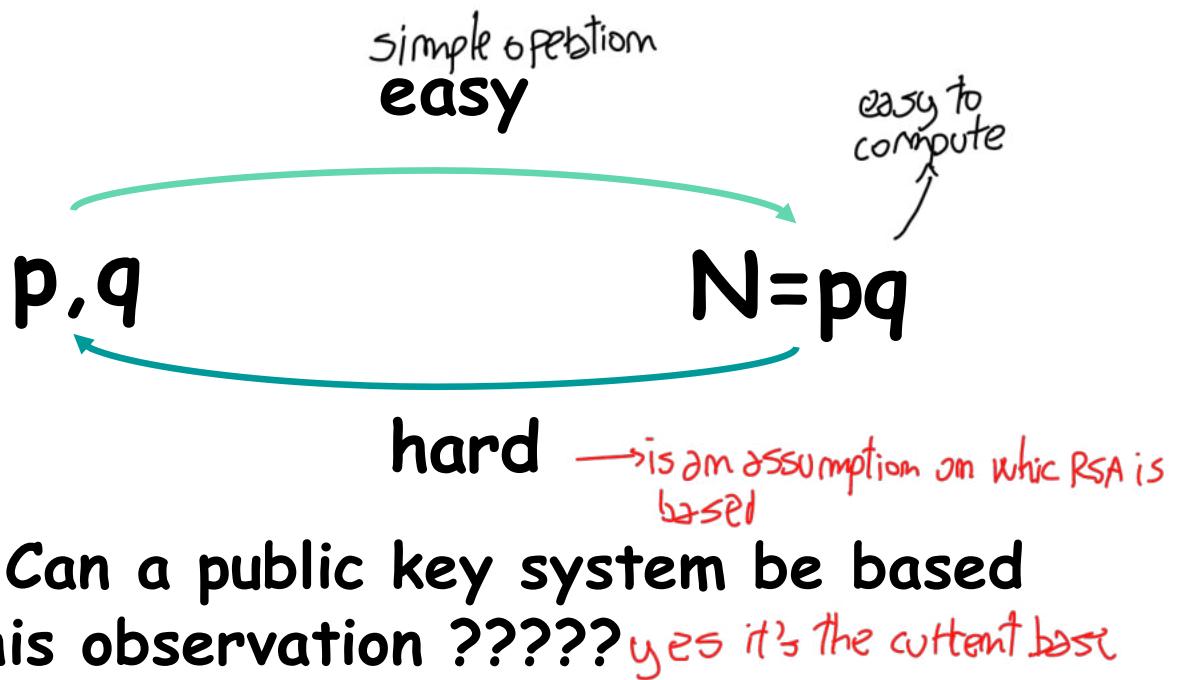
this is the good way of doing "Public key" approach, because everybody can encrypt the message and send the ciphertext, but only the owner of private key can decrypt the message in this way we guarantee confidentiality

$$\textcircled{2} \quad E_{\text{private}}(M) = C \longrightarrow \text{only owner of private key can encrypt} \\ D_{\text{public}}(C) = M \longrightarrow \text{everybody can decrypt the message because all know public key}$$

in this way our message is decrypted by everybody, we loose confidentiality, is very bad way

in asymmetric encryption key is thousand bit longer

## Integer Multiplication & Factoring as a One-Way Function



the approach ① lead to PGP approach, that is a way to obtain confidentiality and authenticity (is open source)

another example of hard problem

## Discrete Log (DL)

log with only  
integers number

- Let  $G$  be a finite cyclic group with  $n$  elements and  $g$  a generator of  $G$
- Let  $y$  be any element in  $G$ ; then it can be written as  $y = g^x$ , for some integer  $x$
- Let  $y = g^x$  and  $x$  the minimal nonnegative integer satisfying the equation.
- The minimal  $x$  s.t.  $y = g^x$  is called the discrete log of  $y$  to base  $g$ .
  - Example:  $y = g^x \pmod{p}$  in the multiplicative group of  $\mathbb{Z}_p$

is given  $y$  and  $g$  Find  $x$  is a hard problem

# Efficient algorithm for exponentiation

# Quick exponentiation (C code)

```
static long fastExp(int base, int exp) {  
    long f = 1; result - initial value  
    long b = base;  
    while(exp > 0) { 0....1 & exp  
        int lsb = 0x1 & exp; (less significant bit)  
        exp >>= 1; shifting the exponent b times  
        if(lsb) f *= b; lsb == 1 compute f....f  
        b *= b;  
    }  
    return f;  
}
```

Add mod  $p$  as needed

## modular exponentiation

$$c \bmod m = (a \cdot b) \bmod m = ((a \bmod m) \cdot (b \bmod m)) \bmod m$$

$$b^e \bmod p = (b \bmod p)^{e \bmod p} \bmod p$$

...

can use  $\bmod p$  on partial results

exponent will have thousand of bits, we use  $\bmod$  for not have huge number

hard mean that doesn't have any polynomial time for invert

# Discrete Log in $Z_p$ : A candidate as One Way Function (OWF)

↳ easy to compute in one direction, but hard to invert  
set of number coprime with p

- Let  $y = g^x \text{ mod } p$  in  $Z_p^*$ , the multiplicative group of  $Z_p$  (set of positive integers less than  $p$  and relatively prime to  $p$ , with multiplication - e.g.,  $Z_{10}^* = \{1, 3, 7, 9\}$ )  
each iteration divided by two
- $g^x \text{ mod } p$  can be computed in  $O(\log x)$  multiplications, each multiplication taking  $O(\log^2 p)$  steps (remind  $g < p$ )
  - also, it turns out  $x$  must be  $< p \Rightarrow$  overall # of steps is  $O(\log^3 p)$
- Standard discrete log is believed to be computationally hard.
- $x \rightarrow g^x \text{ mod } p$  is easy (efficiently computable). we have a poly algorithm
- $g^x \text{ mod } p \rightarrow x$  believed hard (computationally infeasible).
- $x \rightarrow g^x \text{ mod } p$  is believed to be a one-way function.
- This is a computation-based notion.

# One-way functions

$f: \text{mapping binary strings in binary strings}$

A function  $f: \{0, 1\}^* \rightarrow \{0, 1\}^*$  is one-way if  
f can be computed by a polynomial time  
algorithm, but for every randomized  
polynomial time algorithm A, for every  $\sim \rightarrow$  difficult to invert  
polynomial  $p(n)$  and all sufficiently large n

$$\Pr[f(A(f(x))) = f(x)] < 1/p(n)$$

assuming that x is chosen from the uniform  
distribution on  $\{0, 1\}^n$ , and the randomness  
of A.

By this definition, the function must be  
"hard to invert" in the average-case, rather  
than worst-case sense.

# Do one-way functions exist?

- **open problem**
  - they are conjectured to exist
- **Theorem.** If a one-way function exists, then  $P \neq NP$ 
  - one-way functions are conjectured to exist, as  $P$  is conjectured to be strictly included in  $NP$



# Public Exchange of Keys

- Goal: Two parties (Alice and Bob) who do not share any secret information, perform a protocol and derive the same shared key.
- Eve, who is listening, cannot obtain the new shared key if she has limited computational resources (i.e. in polynomial time)
  - unless  $P = NP$

PsK = pre shared key, exchange a key before communication, that using later

↳ known only to the two parties, but not to others

# Diffie-Hellman Key Exchange

- Public parameters: a prime  $p$ , and an element  $g$  (possibly a generator of the multiplicative group  $\mathbb{Z}_p^*$ )
  - for technical reasons it is worthy choosing  $p$  as a safe prime, i.e., a prime  $p = 2q+1$  where  $q$  is also prime (a Sophie Germain prime)
- Alice chooses  $a$  at random from the interval  $[1..p-1]$  and sends  $g^a \bmod p$  to Bob.
- Bob chooses  $b$  at random from the interval  $[1..p-1]$  and sends  $g^b \bmod p$  to Alice.
- Alice and Bob compute the shared key  $g^{ab} \bmod p$ :  
Bob holds  $b$ , computes  $(g^a)^b \bmod p = g^{ab} \bmod p$ .  
Alice holds  $a$ , computes  $(g^b)^a \bmod p = g^{ab} \bmod p$ . } obtain the same Shared Key, that is a secret

- $a$  and  $b$  are used to be called private key, have the same role.
- This is the same way to build a session key, that is a symmetticky key
  - ↳ communication finish with a common agreement of when one the two parties stop send messages. —> key expire

DH solution is not used for obtain confidentiality, unless what you want to exchange is small, the encryption is still computationally heavy but you will have good practical result for 400 bit, like for symmetrical key. DH solution is used for send symmetrical key to the other part.

# Diffie-Hellman Key Exchange

DH or (ECDH)

Alice

$a, g, p$  → secret number generate, never sent  
is a prime  
↳ fixed

$$A = g^a \text{ mod } p$$

$$K = B^a \text{ mod } p$$

$$g, p, A$$

$$B$$

achieving the same result

Bob

$b$  → secret only Bob know it

$$B = g^b \text{ mod } p$$

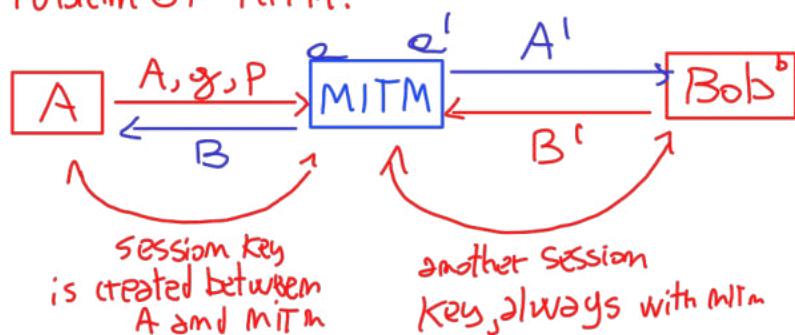
$$K = A^b \text{ mod } p$$

↳ public

$$K = A^b \text{ mod } p = (g^a \text{ mod } p)^b \text{ mod } p = g^{ab} \text{ mod } p = (g^b \text{ mod } p)^a \text{ mod } p = B^a \text{ mod } p$$

Public talk & mt achieve a public key don't know to attacker

Problem of MITM:



A and B think to have a shared public key, but in reality, they are communicating with MITM, they lose confidentiality, and data integrity. There is need of authentication.

must change public key each time  
generate every time as a big prime also p need to be

the leak of a key not impact on next and previous secrecy

# Perfect forward secrecy

→ for exam

## The cryptosystem

- generates random public keys per session for the purposes of key agreement, and
- does not use any sort of deterministic algorithm in doing so

The compromise of a shared key does not compromise former, or subsequent, shared keys

16

a.y. 2022-23

if the shared key of A and B is compromised, the future keys are not compromised, this is the perfect forward secrecy

hash function known  
~~Key:  $K_t = h(\text{seed})$~~

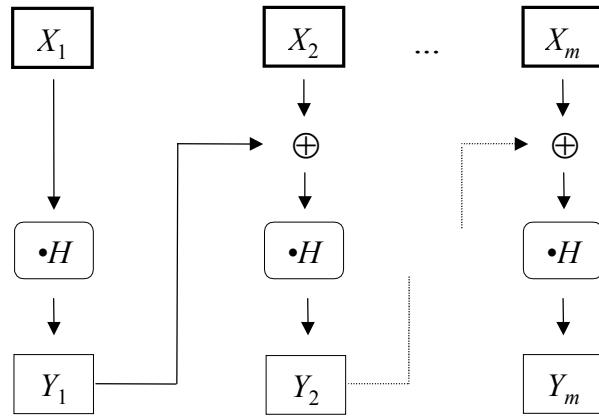
$$K_m = h(K_{m-1}) \dots K_1 = h(\text{seed})$$

not good, no perfect forward secrecy, attacker can predict next keys if leak one key, otherwise is good for generating a key a good hashing function

CMS@diag.unimol.it

1084rs  
hw01 - material subject

↳ same.tex  
same.jpg

Figure 1:  $\text{GHASH}_H(X_1 \parallel X_2 \parallel \dots \parallel X_m) = Y_m$ .

## 6.5 GCTR Function

Algorithm 3 below specifies the GCTR function. The suggested notation does not indicate the choice of the underlying block cipher.

### Algorithm 3: $\text{GCTR}_K(ICI, X)$

*Prerequisites:*

approved block cipher CIPH with a 128-bit block size;  
key  $K$ .

*Input:*

initial counter block  $ICB$ ;  
bit string  $X$ , of arbitrary length.

*Output:*

bit string  $Y$  of bit length  $\text{len}(X)$ .

*Steps:*

1. If  $X$  is the empty string, then return the empty string as  $Y$ .
2. Let  $n = \lceil \text{len}(X)/128 \rceil$ .
3. Let  $X_1, X_2, \dots, X_{n-1}, X_n^*$  denote the unique sequence of bit strings such that
 
$$X = X_1 \parallel X_2 \parallel \dots \parallel X_{n-1} \parallel X_n^*;$$

$$X_1, X_2, \dots, X_{n-1} \text{ are complete blocks.}^2$$
4. Let  $CB_1 = ICB$ .
5. For  $i = 2$  to  $n$ , let  $CB_i = \text{inc}_{32}(CB_{i-1})$ .
6. For  $i = 1$  to  $n - 1$ , let  $Y_i = X_i \oplus \text{CIPH}_K(CB_i)$ .
7. Let  $Y_n^* = X_n^* \oplus \text{MSB}_{\text{len}(X_n^*)}(\text{CIPH}_K(CB_n))$ .

<sup>2</sup> Consequently,  $X_n^*$  is either a complete block or a nonempty partial block, and if  $1 \leq \text{len}(X) \leq 128$ , then  $X = X_1^*$ .

8. Let  $Y = Y_1 \parallel Y_2 \parallel \dots \parallel Y_n^*$ .
9. Return  $Y$ .

In Steps 1 and 2, the input string of arbitrary length is partitioned into a sequence of blocks to the greatest extent possible, so that only the rightmost string in the sequence may be a “partial” block. In Steps 3 and 4, the 32-bit incrementing function is iterated on the initial counter block input to generate a sequence of counter blocks; the input block is the first block of the sequence. In Steps 5 and 6, the block cipher is applied to the counter blocks and the results are XORed with the corresponding blocks (or partial block) of the partition of the input string. In Step 7, the sequence of results is concatenated to form the output.

Figure 2 below illustrates the GCTR function.

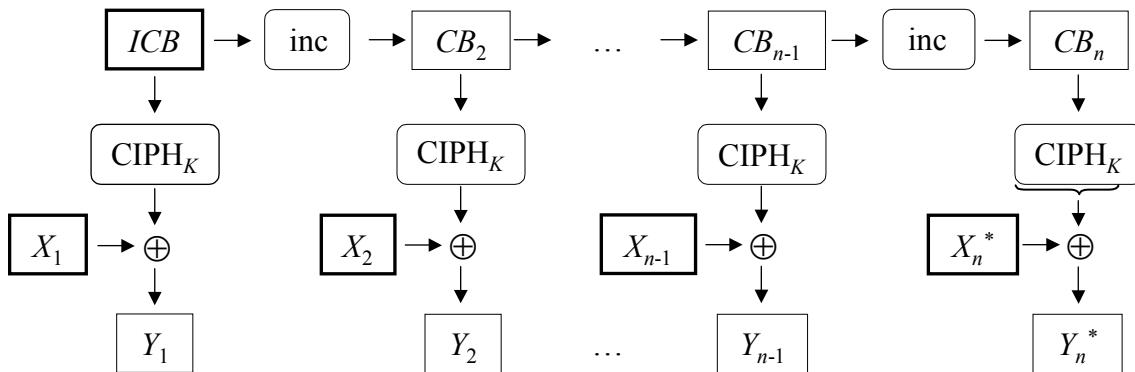


Figure 2:  $\text{GCTR}_K(I\!C\!B, X_1 \parallel X_2 \parallel \dots \parallel X_n^*) = Y_1 \parallel Y_2 \parallel \dots \parallel Y_n^*$ .

## 7 GCM Specification

Algorithms 4 and 5 for the authenticated encryption and authenticated decryption functions of GCM are specified in Secs. 7.1 and 7.2 below. The specifications include the inputs, the outputs, the steps of the algorithm, diagrams, and summaries. The suggested notation does not indicate the choice of the underlying block cipher. The inputs that are typically fixed across many invocations of the function are called the prerequisites; however, some of the prerequisites may also be regarded as (varying) input. The prerequisites and the other inputs shall meet the requirements in Sec. 5.

For both algorithms, equivalent sets of steps that produce the correct output are permitted. For example, in Algorithm 5, the verification of the tag may precede the computation of the plaintext.

### 7.1 Algorithm for the Authenticated Encryption Function

Algorithm 4 below specifies the authenticated encryption function:

Algorithm 4: GCM-AE<sub>K</sub>(IV, P, A)*Prerequisites:*

approved block cipher CIPH with a 128-bit block size;  
key  $K$ ;  
definitions of supported input-output lengths;  
supported tag length  $t$  associated with the key.

*Input:*

initialization vector  $IV$  (whose length is supported);  
plaintext  $P$  (whose length is supported);  
additional authenticated data  $A$  (whose length is supported).

*Output:*

ciphertext  $C$ ;  
authentication tag  $T$ .

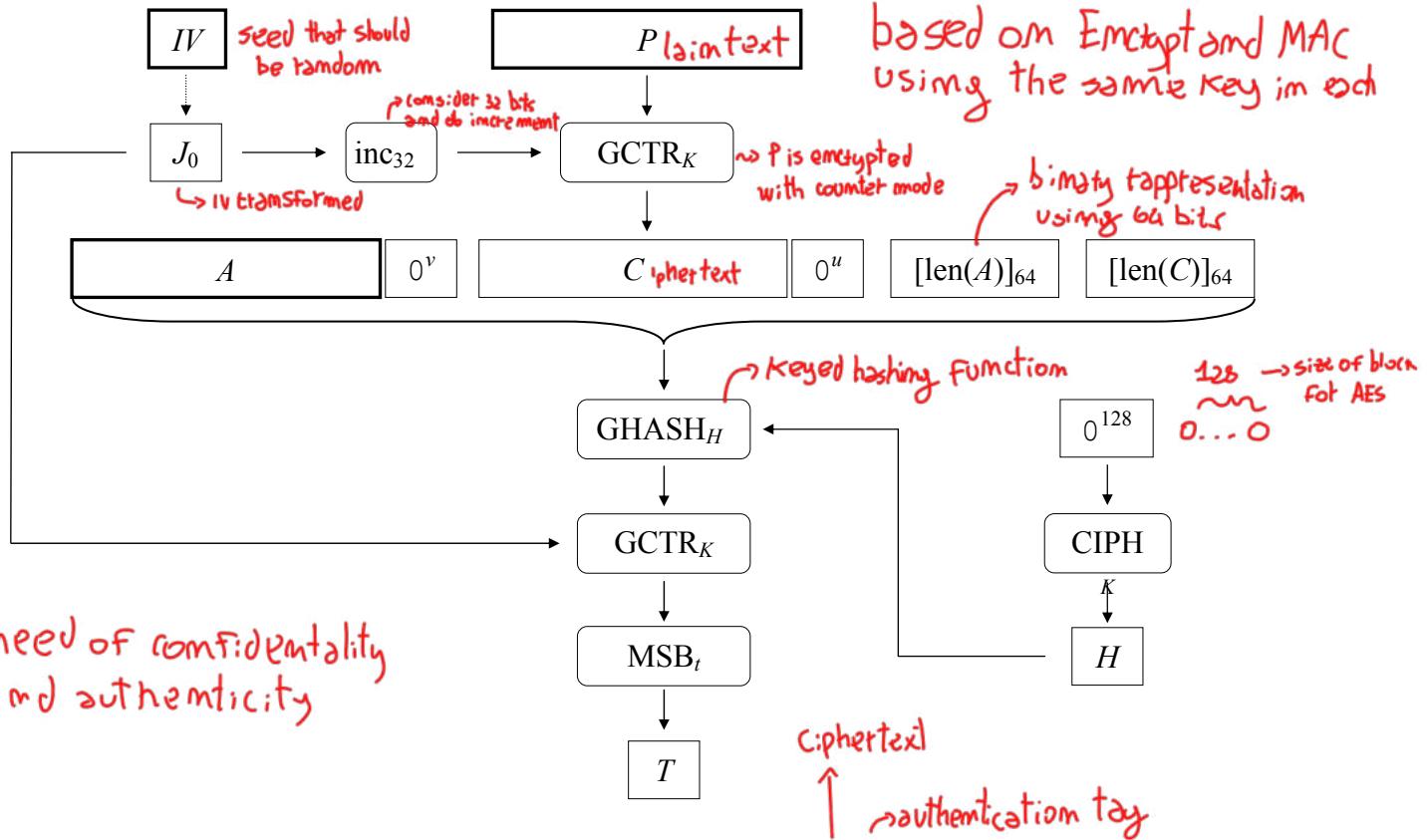
*Steps:*

1. Let  $H = \text{CIPH}_K(0^{128})$ .
2. Define a block,  $J_0$ , as follows:  
If  $\text{len}(IV) = 96$ , then let  $J_0 = IV \parallel 0^{31} \parallel 1$ .  
If  $\text{len}(IV) \neq 96$ , then let  $s = 128 \lceil \text{len}(IV)/128 \rceil - \text{len}(IV)$ , and let  
 $J_0 = \text{GHASH}_H(IV \parallel 0^{s+64} \parallel [\text{len}(IV)]_{64})$ .
3. Let  $C = \text{GCTR}_K(\text{inc}_{32}(J_0), P)$ .
4. Let  $u = 128 \cdot \lceil \text{len}(C)/128 \rceil - \text{len}(C)$  and let  $v = 128 \cdot \lceil \text{len}(A)/128 \rceil - \text{len}(A)$ .
5. Define a block,  $S$ , as follows:  
 $S = \text{GHASH}_H(A \parallel 0^v \parallel C \parallel 0^u \parallel [\text{len}(A)]_{64} \parallel [\text{len}(C)]_{64})$ .
6. Let  $T = \text{MSB}_t(\text{GCTR}_K(J_0, S))$ .
7. Return  $(C, T)$ .

In Step 1, the hash subkey for the GHASH function is generated by applying the block cipher to the “zero” block. In Step 2, the pre-counter block ( $J_0$ ) is generated from the IV. In particular, when the length of the IV is 96 bits, then the padding string  $0^{31} \parallel 1$  is appended to the IV to form the pre-counter block. Otherwise, the IV is padded with the minimum number of ‘0’ bits, possibly none, so that the length of the resulting string is a multiple of 128 bits (the block size); this string in turn is appended with 64 additional ‘0’ bits, followed by the 64-bit representation of the length of the IV, and the GHASH function is applied to the resulting string to form the pre-counter block. In Step 3, the 32-bit incrementing function is applied to the pre-counter block to produce the initial counter block for an invocation of the GCTR function on the plaintext. The output of this invocation of the GCTR function is the ciphertext.

In Steps 4 and 5, the AAD and the ciphertext are each appended with the minimum number of ‘0’ bits, possibly none, so that the bit lengths of the resulting strings are multiples of the block size. The concatenation of these strings is appended with the 64-bit representations of the lengths of the AAD and the ciphertext, and the GHASH function is applied to the result to produce a single output block. In Step 6, this output block is encrypted using the GCTR function

with the pre-counter block that was generated in Step 2, and the result is truncated to the specified tag length to form the authentication tag. The ciphertext and the tag are returned as the output in Step 7.

Figure 3: GCM-AE<sub>K</sub> ( $IV, P, A$ ) = ( $C, T$ ).

The authenticated encryption function is illustrated in Figure 3 above. The determination of  $J_0$  from  $IV$  (Step 2) is not depicted.

[security: https://en.wikipedia.org/wiki/Galois/Counter\\_Mode#Security](https://en.wikipedia.org/wiki/Galois/Counter_Mode#Security)

## 7.2 Algorithm for the Authenticated Decryption Function

Algorithm 5 below specifies the authenticated decryption function:

### Algorithm 5: GCM-AD<sub>K</sub> ( $IV, C, A, T$ )

*Prerequisites:*

approved block cipher CIPH with a 128-bit block size;  
 key  $K$ ;  
 definitions of supported input-output lengths;  
 supported tag length  $t$  associated with the key.

*Input:*

initialization vector  $IV$ ;  
 ciphertext  $C$ ;

# DH Security

- DH is at most as strong as DL in  $Z_p^*$ .
- Formal equivalence unknown, though some partial results known.
- Despite 25 years effort, still considered secure to date.
- Secret integers  $a$  and  $b$  are discarded at the end of the session, hence Diffie-Hellman key exchange achieves **perfect forward secrecy**, because no long-term private keying material exists to be disclosed.
- Computation time is  $O(\log^3 p)$ .

a.y. 2022-23

public key crypto

17

↳ is not secure if use same  $p$  everytime.

Identity based encryption (IBE): no need of certification authorities, is only need one authority that is able to generate keys, you use name, fiscal code, email and generate a public key. → From that the authority generate the corresponding private key.

Attribute based encryption (ABE): instead of identity, we have a set of attribute: age, sex, ...

# Weak Diffie-Hellman and the Logjam Attack

Set user at first ask what version

→ use of TLS, and is able to downgrade version, but attacker can use this feature

use DH and other techniques

**Logjam attack against the TLS protocol.** The Logjam attack allows a man-in-the-middle attacker to downgrade vulnerable TLS connections to 512-bit export-grade cryptography. This allows the attacker to read and modify any data passed over the connection.

**Threats from state-level adversaries.** Millions of HTTPS, SSH, and VPN servers all use the same prime numbers for Diffie-Hellman key exchange. Practitioners believed this was safe as long as new key exchange messages were generated for every connection. However, the first step in the number field sieve—the most efficient algorithm for breaking a Diffie-Hellman connection—is dependent only on this prime. After this first step, an attacker can quickly break individual connections.

<https://weakdh.org/>

<https://weakdh.org/imperfect-forward-secrecy-ccs15.pdf>

# Properties of Key Exchange



Necessary security requirement: the shared secret key is a one-way function of the public and transmitted information.



Necessary "constructive" requirement: an appropriate combination of public and private pieces of information forms the shared secret key efficiently.



DH Key exchange by itself is effective only against a passive adversary. Man-in-the-middle attack is lethal.

# Man-in-the-middle attack

- Man-in-the-middle attack (MITM), or bucket-brigade attack, or sometimes Janus attack
- The attacker makes independent connections with the victims and relays messages between them, making them believe that they are talking directly to each other over a private connection, when in fact the entire conversation is controlled by the attacker.
- The attacker must be able to intercept all messages going between the two victims and inject new ones, which is straightforward in many circumstances.
- A man-in-the-middle attack can succeed only when the attacker can impersonate each endpoint to the satisfaction of the other. Most cryptographic protocols include some form of endpoint authentication specifically to prevent MITM attacks.

# Security Requirements

- Is the one-way relationship between public information and shared private key sufficient?
- A one-way function may leak some bits of its arguments: this is prevented by carefully choosing  $g$  and  $p$ 
  - $g$  = generator,  $p$  = safe prime

# Security Requirements (cont.)

- The full requirement is: given all the communication recorded throughout the protocol, computing any bit of the shared key is hard
- Note that the “any bit” requirement is especially important

public key crypto

# Other DH Systems

- The DH idea can be used with any group structure
- Limitation: groups in which the discrete log can be easily computed are not useful (for example: additive group of  $\mathbb{Z}_p$ )
- Currently useful DH systems: ECDH, the multiplicative group of  $\mathbb{Z}_p$  and elliptic curve systems (see [https://en.wikipedia.org/wiki/Elliptic-curve\\_Diffie-Hellman](https://en.wikipedia.org/wiki/Elliptic-curve_Diffie-Hellman))

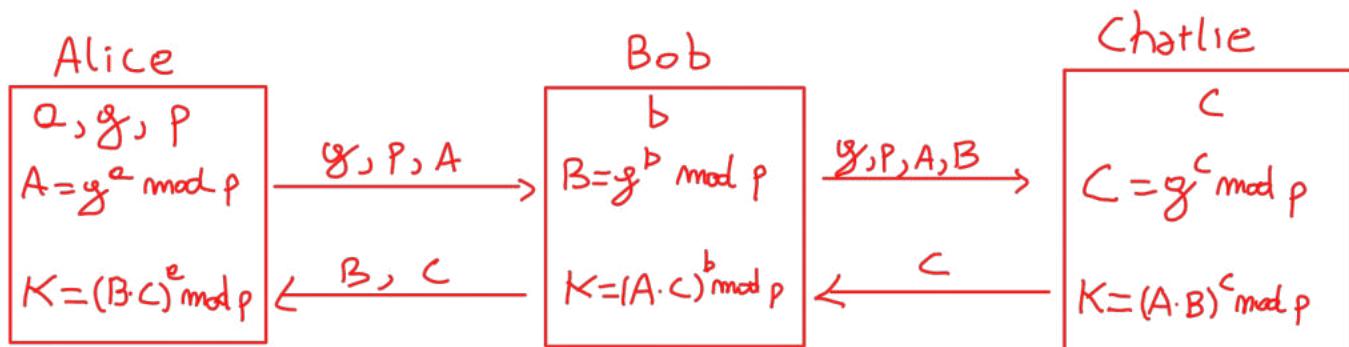
public key crypto

a.y. 2022-23

2  
3

exam: how to generalize the approach of DH for exchanging a shared key between three parties (otm)

→ (a) (b) (c)  $g^{abc} \text{ mod } p$



$$K = g^{abc} \text{ mod } p$$

-

RSA

? Public Key  
CryptoSystem

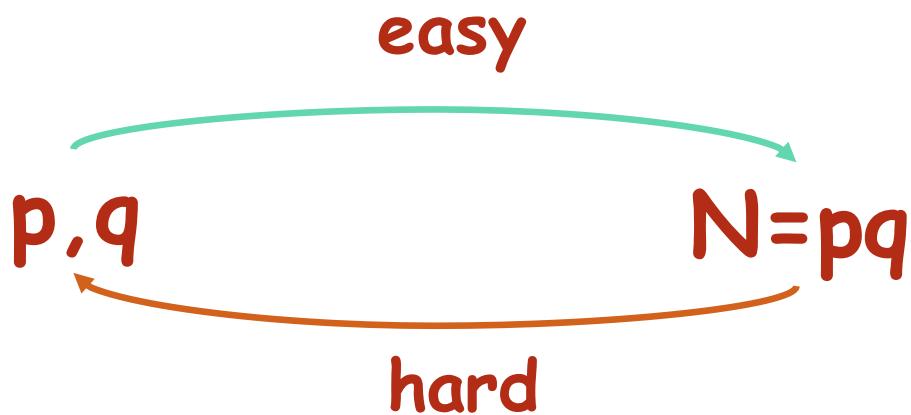


# Diffie and Hellman (76)

## “NEW DIRECTIONS IN CRYPTOGRAPHY”

- Split the Bob's secret key  $K$  to two parts:
  - $K_E$ , to be used for encrypting messages **to Bob**.
  - $K_D$ , to be used for decrypting messages **by Bob**.
- $K_E$  can be made public , *not  $K_D$  for maintain confidentiality*
  - (public key cryptography, asymmetric cryptography)

# INTEGER MULTIPLICATION & FACTORING AS A ONE WAY FUNCTION.



Q.: Can a public key system be based  
on this observation ?????

# EXCERPTS FROM RSA PAPER (CACM, 1978)



*The era of “electronic mail” may soon be upon us; we must ensure that two important properties of the current “paper mail” system are preserved: (a) messages are private, and (b) messages can be signed. We demonstrate in this paper how to build these capabilities into an electronic mail system.*



*At the heart of our proposal is a new encryption method. This method provides an implementation of a “public-key cryptosystem,” an elegant concept invented by Diffie and Hellman. Their article motivated our research, since they presented the concept but not any practical implementation of such system.*

set of numbers that are coprime with p and q, that are prime numbers

## THE MULTIPLICATIVE GROUP $Z_{pq}^*$

Let  $p$  and  $q$  be two large primes. Denote their product  $N = pq$

The multiplicative group  $Z_N^* = Z_{pq}^*$  contains all integers in the range  $[1, pq-1]$  that are relatively prime to both  $p$  and  $q$ . [Prove it]

Since in  $[1, pq-1]$  there are  $(p-1)$  multiples of  $q$  and  $(q-1)$  multiples of  $p$ , the size of the group is  $f(pq) = pq-1-(p-1)-(q-1)=pq-(p+q)+1=(p-1)(q-1)$

so (by Euler's theorem)  
for every  $x \in Z_{pq}^*$ ,  $x^{(p-1)(q-1)} \equiv 1 \pmod{pq} \equiv 1 \pmod{N}$

$$x^{\varphi(m)} \equiv 1 \pmod{m}$$

Want to do exponentiation for encrypt information.

Going to an information to another information that is power made something.

Obviously we want also to decrypt the same information, reversing the exponentiation

## EXPONENTIATION IN $\mathbb{Z}_{pq}^*$

Want to compute exponential value  
for doing encryption

Motivation: We want to exponentiate for encryption.

Note that not all integers in  $\{1, 2, \dots, pq-1\}$  belong to  $\mathbb{Z}_{pq}^*$ . These elements do not necessarily have a unique inverse in  $\mathbb{Z}_{pq}^*$  (in fact multipl. is not a one-to-one mapping)

Let  $e$  be an integer,  $1 < e < (p-1)(q-1)$ .

**Question:** When is exponentiation to the  $e$ -th

power,  $x \rightarrow x^e$ , a one-to-one oper. in  $\mathbb{Z}_{pq}^*$ ?

6

for inverting this function want to know when is one to one.

- when we are doing encryption we need to do operations mod  $N$ , when proving properties we need to use mod  $\varphi(N)$

$p, q$  are prime  
 $(p-1), (q-1)$  are not prime, divisible by 2  
 $\hookrightarrow e$  must be not multiple of 2

the size is  $\varphi(N)$

## EXPONENTIATION IN $\mathbb{Z}_{pq}^*$

$e$  is coprime with  $\varphi(N)$

Claim: If  $e$  is relatively prime to  $(p-1)(q-1)$ ,  $e$  must then  $x \rightarrow x^e$  is a one-to-one op in  $\mathbb{Z}_{pq}^*$  belonging to  $\mathbb{Z}_N^*$

greatest common divisor

Constructive proof: Since  $\gcd(e, (p-1)(q-1)) = 1$ ,  $e$  has a multiplicative inverse mod  $(p-1)(q-1)$ .  $\equiv_{\text{mod } \varphi(N)}$

Denote it by  $d$ , then  $ed = 1 + C(p-1)(q-1)$ ,  $C$  constant.

Let  $y = x^e$ , then  $y^d = (x^e)^d = x^{1+C(p-1)(q-1)} = x^1 x^{C(p-1)(q-1)} = x(x^{(p-1)(q-1)})^C = x \cdot 1 = x$   $(x^e)^d = x$   $d$  is the inverse meaning  $y \rightarrow y^d$  is the inverse of  $x \rightarrow x^e$  QED

- INVERSE is another element of the group such that you compute multiplication and the result will be the unit, element that not change the operand (Ex. 1)
- if  $e$  coprime with  $\varphi(N)$  exist a multiplicative inverse.

$e$  and  $d$  can be translate in demonstration, have same role

$(e, N)$  will be public  $\rightarrow (d, N)$  will private

$\hookrightarrow$  only  $d$  select to advertise

RSA, DSA, DSS most popular cryptosystem

## RSA PUBLIC KEY CRYPTOSYSTEM

- Let  $N = pq$  be the product of two primes *chosen at random*
- Choose  $1 < e < \phi(N)$  such that  $\gcd(e, \phi(N)) = 1$
- Let  $d$  be such that  $de \equiv 1 \pmod{\phi(N)}$  *Ltotque Function*
- The public key is  $(e, N)$
- The private key is  $(d, N)$  *↳ is the real secret*
- Encryption of  $M \in \mathbb{Z}_N$  by  $C = E(M) = M^e \pmod{N}$  *→ 2480 bits or 4960 bits*
- Decryption of  $C \in \mathbb{Z}_N$  by  $M = D(C) = C^d \pmod{N}$

The above-mentioned method should not be confused with the exponentiation technique presented by Diffie and Hellman to solve the key distribution problem".

$C = c$ : cipher text

$M = p$ laintext

$\text{gcd} = \text{greatest common divisor}$

## WHY DECRYPTION WORKS

on message that is decrypted, plain text

- since  $ed \equiv 1 \pmod{\phi}$ , there is integer  $k$  s.t.  
 $ed = 1 + k\phi$ ,  $\phi = (p-1)(q-1)$
- if  $\gcd(m, p) = 1$  then (by Fermat)  $m^{p-1} \equiv 1 \pmod{p}$ .  
    □ raise both sides to the power  $k(q-1)$  and then multiply  
    both sides by  $m$ , and get  
 $m^{1+k(p-1)(q-1)} \equiv m \pmod{p}$   
        ↳  $p$  is prime  
            if  $m < p$   
             $\gcd(m, p) = 1$
- if  $\gcd(m, p) = p$  (no other possibilities!) then the last  
congruence still holds because both sides congruent  
to 0 mod  $p$ , because  $m = jp$ , for some  $j \geq 1$  (hence  $m$  is  
multiple of  $p$ )
- hence, in both cases,  $m^{ed} \equiv m \pmod{p}$
- similarly,  $m^{ed} \equiv m \pmod{q}$
- since  $\gcd(p, q) = 1$ , it follows  $m^{ed} \equiv m \pmod{N}$

↳ step prime numbers

# RSA

- key length is variable
  - the longer, the higher security
  - 4096 bits is common
- block size also variable
  - but  $| \text{plaintext} | \leq | N |$  (in practice, for avoiding weak cases,  $| \text{plaintext} | < | N |$ )
  - $| \text{ciphertext} | = | N |$
- slower than DES

↗ not use for confidentiality

↳ encrypt by private key  
instead, public used only for  
share the key

is battery draining, only  
used for establish a session key

$e=1$  obtain by computing multiplicative inverse of  $d \pmod{\phi(N)}$

## A SMALL EXAMPLE

Two prime

$(p-1)(q-1)$

- Let  $p = 47, q = 59, N = pq = 2773. \phi(N) = 46 \times 58 = 2668.$   $\rightarrow$  Pick  $d,$  if same choose  $e \quad 1 \leq d \leq \phi(N)$
  - Pick  $d = 157$  ( $\gcd(2668, 157) = \gcd(157, 156) = 1$ )  $\rightarrow$  ~~are co-prime~~
  - \*  $\gcd(156, 1) = \gcd(1, 0) = 1,$  then  $157 \times 17 - 2668 = 1,$  so  $e = 17$  is the inverse of  $157 \pmod{2668}$  [see next slide for details]
  - For  $N = 2773$  we can encode two letters per block, using a two-digit number per letter:
  - blank = 00, A = 01, B = 02, ..., Z = 26
  - Message: ITS ALL GREEK TO ME is encoded
  - 0920 1900 0112 1200 0718 0505 1100 2015 0013 0500
- ↳ It's a block encryption, need a mode encryption like CBC

11

$$a \geq b$$

$$\begin{cases} \text{GCD}(a, b) = \text{GCD}(a-b, b) \\ \text{GCD}(a, 1) = 1 \\ \text{GCD}(a, a) = a \end{cases}$$

Ex:  $\text{GCD}(100 \dots 0, 3) = \text{GCD}(99 \dots 97, 3) = \dots$  repetition of subtraction

↳ very slow

\*  $a > b$  FAST way

$$\text{GCD}(a, b) = \text{GCD}(b, a \% b)$$

## COMPUTING THE MULTIPLICATIVE INVERSE

- $p = 47$ ,  $q = 59$ ,  $N = pq = 2773$ ,  $\phi(N) = 46 \times 58 = 2668$ ;  
choose  $d = 157$
- **Bézout's identity:**  $au + bv = d$  ( $a \geq b$ ;  $u, v$  signed integers;  $d$  greatest common divisor of  $a$  and  $b$ ) +  
extended Euclid's alg.
  - if  $a$  and  $b$  coprime then  $d = 1$ ,  $u$  is the multiplicative inverse of  $a$  ( $\text{mod } b$ ) and  $v$  is the multiplicative inverse of  $b$  ( $\text{mod } a$ )
- $-1 \times 2668 + 17 \times 157 = 1$  (Bézout)

# THE EXTENDED EUCLID'S ALGORITHM

- given integer  $x, y$  s.t.  $\text{gcd}(x,y)=1$ , find the multiplicative inverse of  $x \pmod{y}$ 
  - assume wlog  $x \geq y$
- traditional Euclid's alg. for computing  $\text{gcd}(x,y)$  calculates  $r_n = r_{n-2} \% r_{n-1}$ , with  $r_{-2} = x, r_{-1} = y$
- when  $r_n = 0$  gcd has been found, equal to  $r_{n-1}$
- Bézout identity: there exist  $u, v$  s.t.  $ux + vy = 1$
- the multiplicative inverse of  $x \pmod{y}$  is a number  $x^{-1}$  such that  $x^{-1}x \equiv 1 \pmod{y}$ , hence,  $x^{-1}x - 1 = ky$ , for some integer  $k$ , that is  $x^{-1}x - ky = 1$ , or  $x^{-1}x + vy = 1$ , for some integer  $v$  (with  $v = -k$ )

## THE EXTENDED EUCLID'S ALGORITHM (2)

- we can extend Euclid's algorithm to compute signed integers  $u_n$  and  $v_n$  s.t., for each  $n$ :

$$u_n x + v_n y = r_n$$

- (constructive) proof by induction**

- let  $u_{-2} = 1$ ,  $v_{-2} = 0$ ,  $u_{-1} = 0$ ,  $v_{-1} = 1$
- let  $r_n = r_{n-2} - r_{n-1}q_n$ ,  $q_n = \lfloor r_{n-2} / r_{n-1} \rfloor$
- if we set  $u_n = u_{n-2} - q_n u_{n-1}$  and  $v_n = v_{n-2} - q_n v_{n-1}$ , since (by inductive HP)  $r_{n-1} = u_{n-1}x + v_{n-1}y$  and  $r_{n-2} = u_{n-2}x + v_{n-2}y$ , if we compute  $r_n = r_{n-2} - r_{n-1}q_n = = u_{n-2}x + v_{n-2}y - q_n(u_{n-1}x + v_{n-1}y)$  we get  
 $r_n = (u_{n-2} - q_n u_{n-1})x + (v_{n-2} - q_n v_{n-1})y$ , namely  
 $r_n = u_n x + v_n y$       QED

## THE EXTENDED EUCLID'S ALGORITHM (3)

- since we halt the algorithm when  $r_n = 0$  and  $\gcd(x, y) = r_{n-1} = 1$  then:

$$\boxed{\quad \square \quad r_{n-1} = u_{n-1} x + v_{n-1} y = 1}$$

- hence  $x^{-1} = u_{n-1}$  (inverse is unique), and  $y^{-1} = v_{n-1}$

$n$	$r_n$	$q_n$	$u_n$	$v_n$
-2	2668		1	0
-1	157		0	1
0	156	16	1	-16
1	1	1	-1	17
2	0	156	157	-2668

## A SMALL EXAMPLE

- $N = 2773$ ,  $e = 17$  (10001 in binary)
- ITS ALL GREEK TO ME is encoded as

0920 1900 0112 1200 0718 0505 1100 2015 0013  
0500

- First block  $M = 0920$  encrypts to  

$$M^e = M^{17} = (((M^2)^2)^2)^2 \times M = 948 \pmod{2773}$$

*↑ FAST exponentiation algorithm*  
*↑ uses a sequence of squares*  
*↑  $N = p \cdot q$*
- The whole message (10 blocks) is encrypted as  
 0948 2342 1084 1444 2663 2390 0778 0774 0219 1655
- Indeed  $0948^d = 0948^{157} = 948^{1+4+8+16+128} = 920 \pmod{2773}$ , etc.

## QUICK EXPONENTIATION (C CODE)

```
static long fastExp(int base, int exp) {  
    long f = 1;  
    long b = base;  
    while(exp > 0) {  
        int lsb = 0x1 & exp;  
        exp >>= 1;  
        if(lsb) f *= b;  
        b *= b;  
    }  
    return f;  
}
```

Add mod  $p$  as needed

# CONSTRUCTING AN INSTANCE OF RSA

- Alice first picks at random two large primes,  $p$  and  $q$ , big enough (today 2480 bits or 4060 bits)
- Let  $N = p \cdot q$  be the product of  $p$  and  $q$
- Alice then picks at random a large  $d$  that is relatively prime to  $\varphi(N) = (p-1)(q-1)$   
( $\gcd(d, \varphi(N)) = 1$ ) *→ ate coprime*
- Alice computes  $e$  such that  $d \cdot e \equiv 1 \pmod{\varphi(N)}$  *→ compute mult; pl:cate inverse*  
*change always N*
- Alice publishes the public key  $(N, e)$
- Alice keeps the private key  $(N, d)$ , as well as the primes  $p, q$  and the number  $\varphi(N)$ , in a safe place

Use of  $e$  and  $d$  is the same, ate exchangeable

# RSA: IMPLEMENTATION

1. Find  $p$  and  $q$ , two large primes *for generate the key*
  - Random (an adversary should not be able to guess these numbers; they should be different each time a new key is defined)
2. Choose suitable  $e$  to have a fast encryption,  $e$  is not random typically 3 or better  $2^{16} + 1$ 
  - Use exponentiation algorithm based on repeated squaring:
    - ? Compute power of 2, 4, 8, 16, ...
    - ? Compute power  $e$  by using the binary encoding of  $e$  and powers computed in so far (ex.: if  $e = 3$  then 2 multiplications needed; if  $e = 2^{16} + 1$  then 5 multiplications needed)
  - In this way decrypting is generally slower ( $d$  is large)
3. Compute  $d$ :
  - Euclid's extended algorithm

# RSA: IMPLEMENTATION (STEP 1)

1. Find two random primes *very big and random!*

Algorithm:

- randomly choose a random large odd integer
- test if it is a prime

Note:

- Prime number are relatively frequent (in  $[N, 2N]$  there are  $\approx N / \ln N$  primes)
  - prime number theorem ([http://en.wikipedia.org/wiki/Prime\\_number\\_theorem](http://en.wikipedia.org/wiki/Prime_number_theorem)) states that the average gap between prime numbers near  $N$  is roughly  $\ln(N)$
- Hence randomly choosing we expect to find a prime of  $N$  bits every  $\ln N$  attempts

## RSA: IMPLEMENTATION (STEP 2)

### 2. Encrypting algorithm (compute exponentiation)

Compute power by repeated squaring (so computing power of 2, 4, 8, ..) and then executing multiplication (based on binary encoding of the exponent e)

Cost:  $O(\log N)$  operations

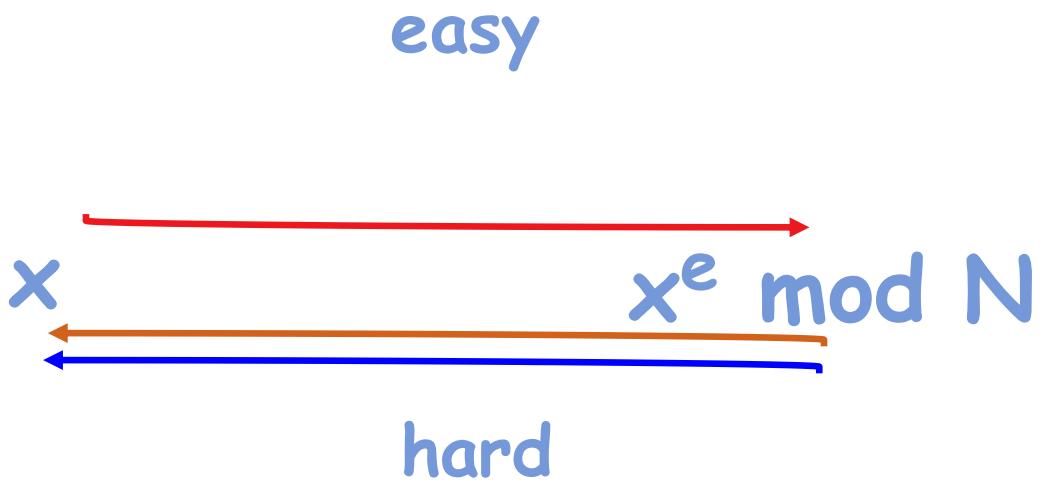
Constant no. of operations if  $e$  is small and its binary representation has few ones

Examples:  $e=3$ , 2 multiplications

$e = 65537 = 2^{16} + 1$ , compute powers  $M^2, M^4, M^8, M^{16}, \dots$   
 $M^{65536}$  and then  $M^{65537} = M^{65536} * M$

(total 16 multiplication)

# RSA AS A ONE WAY TRAPDOOR FUNCTION.



Easy with trapdoor info ( d )

## TRAP-DOOR OWF

Domain      Range

is an information, a binary string  
called  $s$

- **Definition:**  $f: D \rightarrow R$  is a trapdoor one way function if there is a trapdoor  $s$  such that:
  - Without knowledge of  $s$ , the function  $f$  is a one way function
  - Given  $s$ , inverting  $f$  is easy (polynomial)
- Example:  $f_{g,p}(x) = g^x \bmod p$  is **not** a trap-door one way function.
- Example: RSA is a trap-door OWF.

hard for attacker that don't know  $s$ , easy for Bob that know it

# RSA: A COLLECTION OF TRAP-DOOR OWF

- Note: RSA defines a function that depends on the Key
- Definition: Let  $I$  be a set of indices and  $A$  a finite set. A collection of trap door one-way functions is a set of functions  $F$  parameterized by elements in  $I$
- $F$  verifies: For all values  $i$  in  $I$  there exists  $f_i$  such that
  - $f_i$  belongs to  $F$
  - $f_i: D \rightarrow R_i$  is a trap-door one way function
- Idea: we need an algorithm that given a security parameter (the key) selects a function in  $F$  together with a trapdoor information

## ATTACKS ON RSA

↳ have many vulnerability

Recall RSA robustness does not imply it is always robust

1. Factor  $N = pq$ . This is believed hard unless  $p, q$  have some “bad” properties. To avoid such primes, it is recommended to
  - Take  $p, q$  large enough (100 digits each).
  - Make sure  $p, q$  are not too close together.
  - Make sure both  $(p-1), (q-1)$  have large prime factors (to foil Pollard’s rho algorithm)
    - special-purpose integer factorization algorithm (John Pollard, 1975); particularly effective at splitting composite numbers with small factors.
2. Some messages might be easy to decrypt

↳ We need preprocessing messages  
for avoid vulnerability

# RSA AND FACTORING

- ? Fact 1: given  $n, e, p$  and  $q$  it is easy to compute  $d$
- ? Fact 2: given  $n, e$ 
  - if you factor  $n$  then you can compute  $\phi(n)$  and  $d$
- ? Conclusion:
  - If you can factor  $n$  then you break RSA
  - If you invert RSA then you are able to factor  $n$ ?  
**OPEN PROBLEM**

# RSA - ATTACKS

a.j.: 2022-23  
RSA

- ? There are messages easy to decrypt:  
if  $m = 0, 1, n-1$  then  $\text{RSA}(m) = m$

- notice  $e$  must be odd  $\geq 3$ , hence  $(n-1)^e \bmod n = n-1$ ,  
because  $(n-1)^2 \bmod n = 1$

**SOLUT:** rare, use salt

- ? If both  $m$  and  $e$  are small (e.g.  $e = 3$ ) then we  
might have  $m^e < n$ ; hence

$$m^e \bmod n = m^e$$

compute  $e$ th root in arithmetic is easy: adversary  
compute it and finds  $m$

**SOLUT.** Add non zero bytes to avoid small  
messages

## RSA - ATTACKS

Small  $e$  (e.g.,  $e = 3$ )

two plaintext have a relation

? Suppose adversary has two encryptions of similar messages such as

$$c_1 = m^3 \bmod n \text{ and } c_2 = (m+1)^3 \bmod n$$

Therefore

$$m = (c_2 + 2c_1 - 1) / (c_2 - c_1 + 2)$$

? The case  $m$  and  $(am + b)$  is similar (see next slide)

SOLUT. Choose large  $e$

## CHINESE REMAINDER THEOREM

- ? Suppose  $n_1, n_2, \dots, n_k$  are positive integers which are pairwise coprime. Then, for any given integers  $a_1, a_2, \dots, a_k$ , there exists an integer  $x$  solving the system of simultaneous congruencies

$$x \equiv a_i \pmod{n_i} \quad \text{for } i = 1, \dots, k.$$

*$\hookrightarrow n_1, n_2$  coprime and positive*

- ? Furthermore, all solutions  $x$  to this system are congruent modulo the product  $N = n_1 n_2 \dots n_k$ .

# RSA - ATTACKS

Assume  $e = 3$  and then send the same message three times to three different users, with public keys

$$(3, n_1) (3, n_2) (3, n_3)$$

Attack: adv. knows public keys and

$$m^3 \bmod n_1, m^3 \bmod n_2, m^3 \bmod n_3$$

? He can compute (using Chinese remainder theorem)  
 $m^3 \bmod (n_1 \cdot n_2 \cdot n_3)$

? Moreover  $m < n_1, n_2, n_3$ ; hence  $m^3 < n_1 \cdot n_2 \cdot n_3$  and  
 $m^3 \bmod n_1 \cdot n_2 \cdot n_3 = m^3$

? Adv. computes cubic root and gets result

SOLUT. Add random bytes to avoid equal messages

## RSA - ATTACKS

- ? If message space is small, then adv. can test all possible messages ↗ and check matching  
ex.: adv. knows encoding of  $m$  and knows that  $m$  is either  $m_1=10101010$  or  $m_2=01010101$   
adv. encrypts  $m_1$  and  $m_2$  using public key and verifies
- ? SOLUT. Add random string in the message

## RSA - ATTACKS

? If two user have same n (but different e and d) then bad.

→ same p and q

- One user could compute p and q starting from his e, d, n (somewhat tricky, based on Euler's theorem – see for instance "The Handbook of Applied Cryptography", Sect. 8.2.2)
- Then, the user could easily discover the secret key of the other user, given his public key

→ and different keys each time

SOLUT. Each person chooses his own n (the probability two people choose same n is very very low)

# RSA - ATTACKS

? Multiplicative property of RSA:

- If  $M = M_1 * M_2$  then  $(M_1 * M_2)^e \text{ mod } N = ((M_1^e \text{ mod } N) * (M_2^e \text{ mod } N)) \text{ mod } N$

Hence an adversary can proceed using small messages (chosen ciphertext, see next slide)

- Can be generalized if  $M = M_1 * M_2 * \dots * M_k$
- Solut.: padding on short messages

## RSA - CHOSEN CIPHERTEXT ATTACK

Adversary wants to decrypt  $C = M^e \text{ mod } n$

1. adv. computes  $X = (C \cdot 2^e) \text{ mod } n$
2. adv. uses  $X$  as chosen ciphertext and asks the oracle for  $Y = X^d \text{ mod } n$

but....

$$\begin{aligned} X &= (C \text{ mod } n) \cdot (2^e \text{ mod } n) = (M^e \text{ mod } n) \cdot (2^e \text{ mod } n) = \\ &= (2M)^e \text{ mod } n \end{aligned}$$

thus adv. got  $Y = (2M)$

# RSA - ATTACKS

## Implementation attacks

- ? Timing: based on time required to compute  $C^d$
- ? Energy: based on energy required to compute  
(smart card)  $C^d$
- ? Solut.: add random steps in implementation

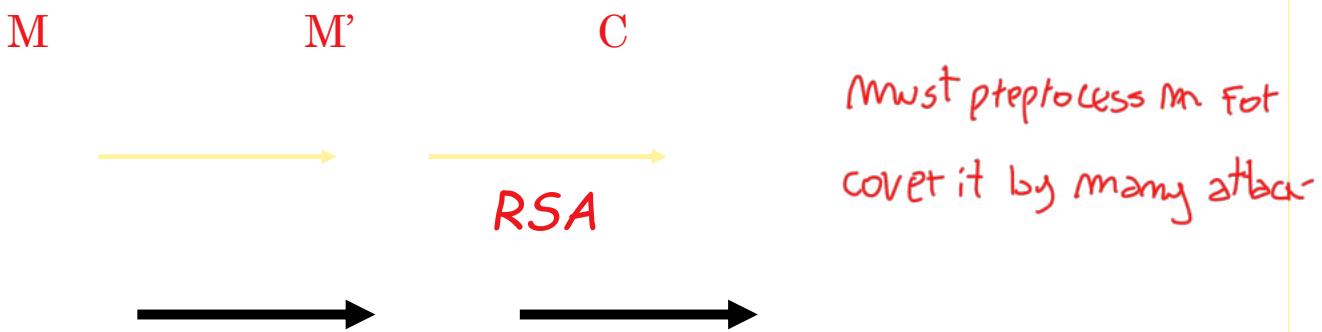
# RSA - ATTACKS : CONCLUSION

Textbook implementation of RSA is NOT safe

- ? It does not verify security criteria
- ? Many attacks

There exists a standard version

- ? Preprocess  $M$  to obtain  $M'$  and apply RSA to  $M'$  (clearly the meaning of  $M \rightarrow M'$  is the same)



## PROPERTIES OF RSA

- ? The requirement  $(e, \phi(n))=1$  is important for uniqueness
- ? Finding  $d$ , given  $p$  and  $q$  is easy. Finding  $d$  given only  $n$  and  $e$  is assumed to be hard (the RSA assumption)
- ? The public exponent  $e$  may be small. Typically, its value is either 3 (problematic) or  $2^{16}+1$
- ? Each encryption involves several modular multiplications. Decryption is longer.

# PUBLIC-KEY CRYPTOGRAPHY STANDARD (PKCS)

Set of standard devised and published by RSA Security;  
many versions with different goals (1-15). See  
Wikipedia for details

PKCS#1: standard to send messages using RSA (byte)

$m = 0 \parallel 2 \parallel$  at least 8 non-zero bytes  $\parallel 0 \parallel M$   
(M original message) ↳ concatenation

- ? first byte 0 implies message is less than N
- ? second byte (= 2) denotes encrypting of a message (1 denotes dig. signature) and implies m is not small
- ? random bytes imply
  - same message sent to many people is always different
  - space of message is large

# PKCS

from the *Handbook of Applied Cryptography*

## 15.3.6 De facto standards

Various security specifications arising through informal processes become de facto standards. This section mentions one such class of specifications: the PKCS suite.

### PKCS specifications

A suite of specifications called *The Public-Key Cryptography Standards* (PKCS) has parts as listed in Table 15.11. The original PKCS #2 and PKCS #4 have been incorporated into PKCS #1. PKCS #11 is referred to as *CRYPTOKI*.

No.	PKCS title
1	RSA encryption standard
3	Diffie-Hellman key-agreement standard
5	Password-based encryption standard
6	Extended-certificate syntax standard
7	Cryptographic message syntax standard
8	Private-key information syntax standard
9	Selected attribute types
10	Certification request syntax standard
11	Cryptographic token interface standard

**Table 15.11:** PKCS specifications.

they are not overlapping  
depend on what are you  
doing RSA

# RSA AND DATA INTEGRITY: OAEP

last idea of  
the standard

- Padding scheme often used together with RSA encryption
  - introduced by Bellare and Rogaway: "Optimal Asymmetric Encryption - How to Encrypt with RSA" 1994, 1995 (<http://cseweb.ucsd.edu/users/mihir/papers/oaep.pdf>)
- OAEP uses a pair of random oracles G and H to process the plaintext prior to asymmetric encryption
  - a random oracle is a mathematical function mapping every possible query to a random response from its output domain.
  - when combined with RSA it is secure under chosen plaintext/ciphertext attacks
- OAEP satisfies the following two goals
  1. Add an element of randomness which can be used to convert a deterministic encryption scheme (e.g., traditional RSA) into a probabilistic scheme.
  2. Prevent partial decryption of ciphertexts (or other information leakage) by ensuring that an adversary cannot recover any portion of the plaintext without being able to invert the trapdoor one-way permutation f.

# OPTIMAL ASYMMETRIC ENCRYPTION PADDING (OAEP)

$n$  = number of bits in RSA modulus

$k_0$  and  $k_1$  = integers fixed by the protocol

$m$  = plaintext message, a  $(n - k_0 - k_1)$ -bit string

$G$  and  $H$  = cryptographic hash functions fixed by the protocol

## To encrypt

? messages are padded with  $k_1$  zeros to be  $n - k_0$  bits in length.

?  $r$  is a random  $k_0$ -bit string

?  $G$  expands the  $k_0$  bits of  $r$  to  $n - k_0$  bits.

?  $X = m00..0 \oplus G(r)$

?  $H$  reduces the  $n - k_0$  bits of  $X$  to  $k_0$  bits.

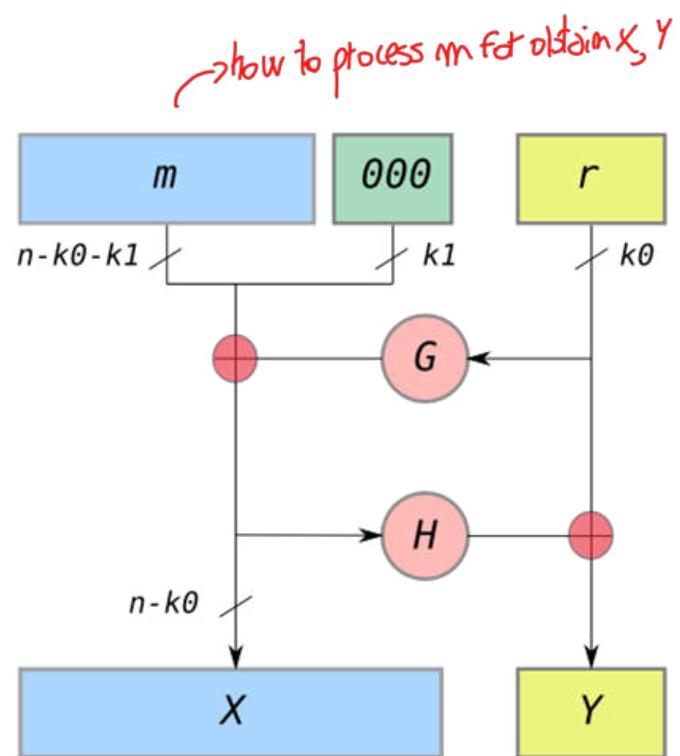
?  $Y = r \oplus H(X)$

? output is  $X \| Y$

## To decrypt

? recover the random string as  $r = Y \oplus H(X)$

? recover the message as  $m00..0 = X \oplus G(r)$



*G and H are identical in PKCS#1 and MGF1 (mask generation function, a hash with programmable output-size)*

Provide some kind of security called all or nothing

→ not get from that schema partial information easily

## OAEP - "ALL-OR-NOTHING" SECURITY

### "all-or-nothing" security

- ? to recover  $m$ , you must recover the entire  $X$  and the entire  $Y$ 
  - $X$  is required to recover  $r$  from  $Y$ , and  $r$  is required to recover  $m$  from  $X$
- ? since any bit of a cryptographic hash completely changes the result, the entire  $X$ , and the entire  $Y$  must both be completely recovered

↳ you can't get partial info

# PKCS#1 TODAY

- ? current version of PKCS#1 is 2.2 (RFC 8017, 2016), based on
  - RSA Encryption Primitive (RSAEP)
  - RSA Decryption Primitive (RSADP)
  - both use the same math with different base/exponent
    - ?  $x^y \bmod N$
- ? according to RFC 8017 two encryption schemes are expected to be supported
  - RSAES-OAEP (required, to be used with new applications)
  - RSAES-PKCS1-v1\_5 (for compatibility with existing applications)

## RSAES-OAEP

- ? given message  $M$ , determine encoded message  $M'$  (octet-stream) by means of OAEP (precedent schema)
- ? determine integer representative  $m$  of  $M'$  *transform m' in a number*
  - use  $m = \text{OS2IP}(M')$  octet-stream to integer primitive
- ? compute ciphertext representative (integer)
  - $c = \text{RSAEP}(N, e, m)$  ( $N, e$ ) is public-key
- ? convert ciphertext representative (integer) to ciphertext (octet-stream)
  - use  $C = \text{I2OSP}(c, |N|)$  integer to octet-stream primitive
- ? decryption is symmetric and is based on  $\text{OAEP}^{-1}$  and  $\text{RSADP}$

# OCTET-STREAMS AND NON-NEGATIVE INTEGERS

- ? OS2IP: *octet-stream to integer primitive*
  - a non-negative integer is constructed by interpreting  $k$  octets (8-bit bytes) as a number in base 256, where each octet represent a digit (in [0, 255])
  - $k$  is the number of bytes for representing  $N$
- ? I2OSP: *integer to octet-stream primitive*
  - an octet-stream (array of  $k$  bytes) is constructed by determining the coefficients of the representation in base 256 of the given integer (each coefficient is an integer in [0, 255] and is stored in one octet)

## RSAES-PKCS1-v1\_5 (old version)

- ? given message  $M$ , determine encoded message  $M'$  (octet-stream) by means of
  - $M' = 0x00 \parallel 0x02 \parallel$  at least 8 non-zero bytes  $\parallel 0x00 \parallel M$
- ? determine integer representative  $m$  of  $M'$ 
  - use  $m = \text{OS2IP}(M')$  octet-stream to integer primitive
- ? compute ciphertext representative (integer)
  - $c = \text{RSAEP}(N, e, m)$  ( $N, e$ ) is public-key
- ? convert ciphertext representative (integer) to ciphertext (octet-stream)
  - use  $C = \text{I2OSP}(c, |N|)$  integer to octet-stream primitive
- ? decryption is symmetric and is based on RSADP and the extraction of  $M$  from  $M'$

49

OS2IP is the name for going to string to a number in RSAES-PkCS

# ELGAMAL ENCRYPTION

- ? Constructed by ElGamal in 1984
- ? Based on DH and consists of three components: key generator, encryption algorithm, decryption algorithm
- ? Alice publishes  $p, g \in \mathbb{Z}_p$  as public parameters
  - $g$  is generator of a cyclic group of order  $p$
- ? Alice chooses  $x$  as a private key and publishes  $g^x \text{ mod } p$  as a public key
  - $x$  chosen at random in  $\{0, 1, \dots, p-1\}$  , private key
- ? Encryption (Bob, for Alice) of  $m \in \mathbb{Z}_p$  by sending  $(g^y \text{ mod } p, mg^{xy} \text{ mod } p)$ 
  - $y$  chosen at random in  $\{0, 1, \dots, p-1\}$
- ? Decryption: Alice computes  $(g^y)^x \text{ mod } p = g^{xy} \text{ mod } p$ , then computes  $(g^{xy})^{-1} \text{ mod } p$  for obtaining  $mg^{xy}(g^{xy})^{-1} \text{ mod } p = m$
- ? Requires two exponentiations per each block transmitted
- ? The involved math is not trivial

# REAL WORLD USAGE

Two words:

Key Exchange

(use RSA (ElGamal) to define a secret key that is used with a faster protocol like AES)

Digital signature: have legal value

ag.it.gov.it

(idea: encrypt by private key)

(, provide: data integrity, authentication, non repudiation

First try: wrong

A  $\xrightarrow{(M, S)} B$

S = signature, similar to authentication tag

( $M, Enc_{\text{private key}}(M)$ ) sent by Alice (let's call it  $(M, R)$ )

|

↓

decrypts by Bob side for verify the signature

↓

use the public key of Alice

$M'$

$M \stackrel{?}{=} M'$

→ yes (accept)

→ No (reject)

$Enc_{\text{priv}}(M)$

Some issue:  $M$  can be a big number (MB, GB), the encryption can be very heavy and in RSA performance are not good, require a lot of time. And we can't estimate length of  $M$ .

no good d.s. have legal  
↑ value

What attacker can do? → existential forgery

Attacker have the possibility to create a pair that is able to have success on both verification. he can create a random file =  $S$  and  $Enc_{\text{public key of Alice}}(S)$ .

Attacker can send this pair  $(S, E)$  to Bob. Bob take  $S$ , encrypt  $S$  by means of public key:  $E_{\text{public key}}(S) = E'$ . Comparing  $E'$  to  $E$  that has been sent, verifying that are the same document. The test is successful.

↓  
authenticity  
digital signature is  
identify a partie in a legal way.

↓  
after putting digital  
signature, they can  
not say, it was not  
me. Unless compromise  
in the system.

Using this approach we get a wrong result, is easily attacked by obtain an existential forgery. This have a legal value don't want this type of forgery.

## A good approach for Digital Signature: OFFICIAL STANDARD

Alice send to Bob a pair  $(M, \sigma)$ .  
 $M$ : document  
 $\sigma$ : signature

instead of encrypt  $M$  that could be very big and battery draining, we encrypt the DIGEST of  $M$ :

This pair:  $(M, \text{Enc}_{\text{Private key}}(H(m)))$

↳ Today  $H(m)$  used is SHA256

$H(m)$  should be a cryptographical hashing function, and UNKEYED  
↳ is not invertible and collision resistant. Here is not possible existential forgery because the attacker could invert  $H(m)$  and is not possible.

Also now the size of the input for  $\text{Enc}$  is smaller, because we encrypt a digest, that are all having the same size, a constant value. Faster operation with smaller input.

Bob verification upon receiving the pair  $(M, \sigma)$ :

1. computes  $\text{Dec}_{\text{Public key (of Alice)}}(\sigma) = h'$

2. computes  $H(m) = h''$

3. comparison  $h' \stackrel{?}{=} h''$  ↗ Yes (accept)  
↓ No (reject)

Private key can be in a SMART CARD that is encrypted.

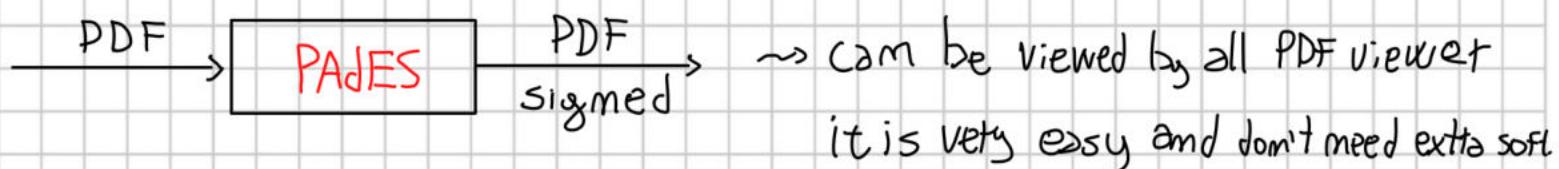
↳ is never stated

• can be also in the computer encrypted, or Remotely stored using a third party

in practice when Alice is sending a digital signed document send one file only containing all information and there are several standard:

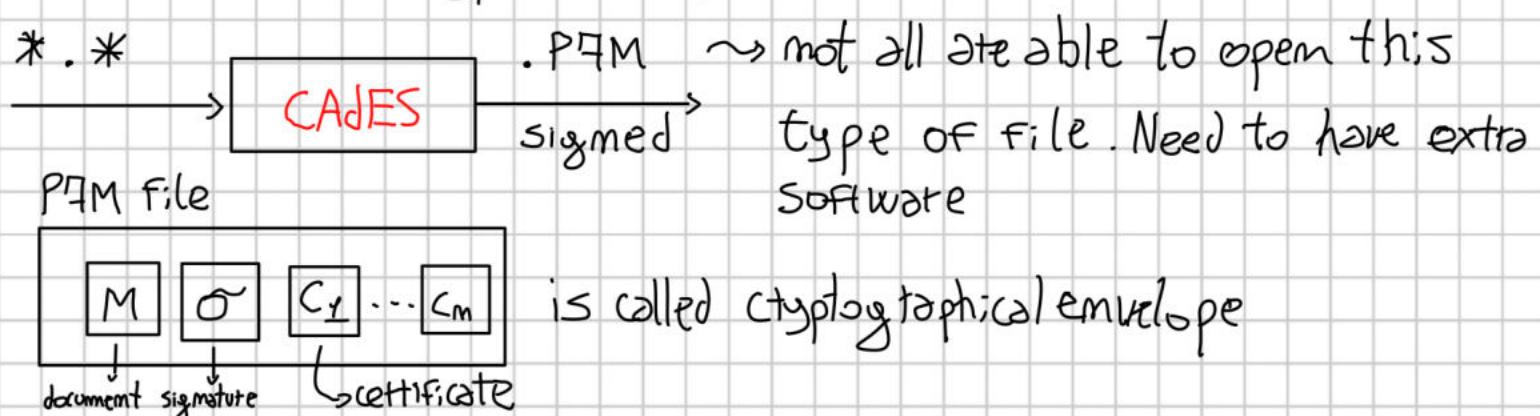
**PAdES**: PDF Advanced Electronic Signature

↳ PDF, this standard is valid only for PDF files.



a time range explain when a digital certificate starting being valid until expiration.

**CAdES**: For all type of file, also PDF



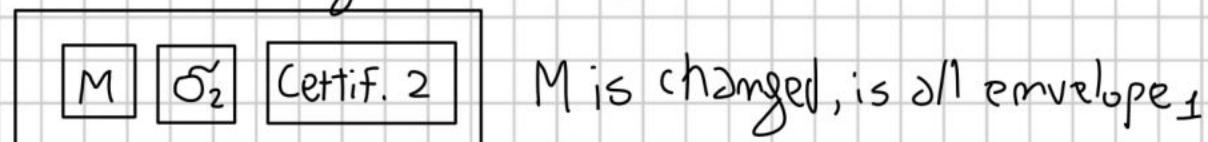
If there are two parties that want to sign a joint document, there are two possibility, in each case one part will sign first and produce a PFM document. The other part can sign document and digital signature of just the document.

In second case will have one document and two signature on the same document and more certificate

Parallel signature →

```
graph LR; M[M] --- O1[O1]; M --- O2[O2]; M --- C1[C1]; M --- C2[C2];
```

Other case sign whatever, like a "matrioska", or **counter signing**



**XADES**: also this for all type of file, X is for XML. This standard is not for humans, but for software system

A strong form of data integrity is data authenticity.

Date authenticity → a very easy solution for obtain data

 ~~not viceversa~~ authenticity is using a KEYED HASHING (like HMAC)  
integrity ↳ is cryptographic

Date authenticity  $\neq$  mom - teleportation (exam question)

if Alice and Bob exchange a document and use a KEYED HASHING for constructing the authentication tag, they are happy. If another person wants to check the originality of the document is real Alice, he can not check, because he is not sharing a key. This approach works only if the environment is only Alice and Bob.

If Alice and Bob start agreeing, a third parties can't check who sent the document.

In digital signature everybody can check the validity of a signature, you don't have to know any private information, you only need a public key given by a certificate authority.

If you have data authenticity you don't have necessarily non-repudiation because Alice and Bob can deny, but non-repudiation implies data authenticity.

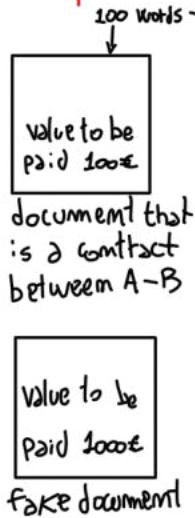
Strongest form of integrity is digital signature because  
have non-repudiation

# Digital signatures- DSA

not used for  
confidence

RSA can be used for confidentiality and also for Digital signature

Example of attack based on birthday bound:  $\rightarrow$  digest must be not too short  
 $2^{56}$  is sufficient long



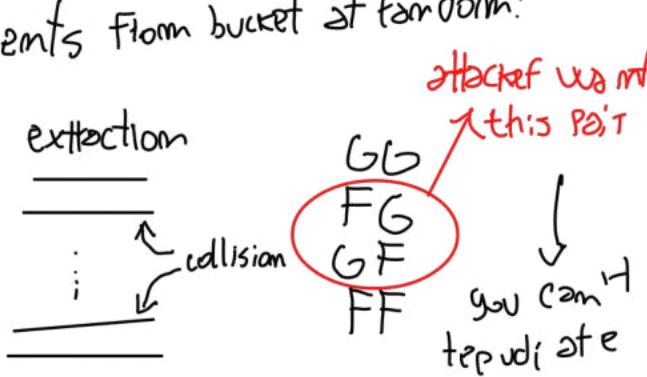
Bob wants to cheat, tries to get 1000€ instead 100€. Bob prepares two documents, better if he does it before signing, identical to the first. Add more spaces between words for changing document but not meaning.  
there are  $2^{100}$  possibilities to generate new document changing the spaces



Bob starts birthday attack, pick documents from bucket at random:

( Bound )  $\sim \sqrt{\text{Space}}$  )

↳ size of digest  
 with  $P > \frac{1}{2}$  find collision, of these  
 only  $\frac{1}{4}$  com turn am attack



# Signatures vs. MACs

Suppose parties A and B share the secret key K. Then M,  $\text{MAC}_K(M)$  convinces A that indeed M originated with B. But in case of dispute A cannot convince a judge that M,  $\text{MAC}_K(M)$  was sent by B, since A could generate it herself.

↳ MACs is only between A and B, using private key. Signature is checked by everyone.

## Problems with "Pure" DH Paradigm

- Easy to forge signatures of random messages even without holding  $D_A$ : Bob picks  $R$  arbitrarily, computes  $S = E_A(R)$ .
- Then the pair  $(S, R)$  is a valid signature of Alice on the "message"  $S$ .
- Therefore, the scheme is subject to existential forgery.

# forgery

ability to create a pair consisting of a message  $m$  and a signature (or MAC)  $\sigma$  that is valid for  $m$ , where  $m$  has not been signed in the past by the legitimate signer

# Existential forgery

- adversary creates any message/signature pair  $(m, \sigma)$ , where  $\sigma$  was not produced by the legitimate signer
- adversary need not have any control over  $m$ ;  $m$  need not have any meaning
- existential forgery is essentially the weakest adversarial goal; therefore the strongest schemes are those which are "existentially unforgeable"

# Selective forgery

- adversary creates a message/signature pair  $(m, \sigma)$  where  $m$  has been chosen by the adversary prior to the attack
- $m$  may be chosen to have interesting mathematical properties with respect to the signature algorithm; however, in selective forgery,  $m$  must be fixed before the start of the attack
- the ability to successfully conduct a selective forgery attack implies the ability to successfully conduct an existential forgery attack

# Universal forgery

adversary creates a  
valid signature  $\sigma$  for  
any given message  $m$

it is the strongest  
ability in forging, and  
it implies the other  
types of forgery

## Problems with "Pure" DH Paradigm

- Consider specifically RSA. Being multiplicative, we have (products mod N)

$$\underline{D_A(M_1 M_2) = D_A(M_1) D_A(M_2)}$$

- If  $M_1 = \text{"I OWE BOB $20"}$  and  $M_2 = \text{"100"}$  then under certain encoding of letters we could get  $M_1 M_2 = \text{"I OWE BOB $20100"}$

## Standard Solution: Hash First

- Let  $E_A$  be Alice's public encryption key, and let  $D_A$  be Alice's private decryption key.
- To sign the message  $M$ , Alice first computes the strings  $y = H(M)$  and  $z = D_A(y)$ . Sends  $(M, z)$  to Bob
- To verify this is indeed Alice's signature, Bob computes the string  $y = E_A(z)$  and checks  $y = H(M)$
- The function  $H$  should be collision resistant, so that cannot find another  $M'$  with  $H(M) = H(M')$

# General Structure: Signature Schemes

Generation of  
private and public  
keys  
(randomized).

Signing (either  
deterministic or  
randomized)

Verification  
(accept/reject) -  
usually  
deterministic.

# Schemes Used in Practice

- RSA
- El-Gamal Signature Scheme (85)
- The DSS (digital signature standard, adopted by NIST in 94 is based on a modification of El-Gamal signature)

## RSA

- Signature: encrypt hash of message using private key
- Only the person who knows the secret key can sign
- Everybody can verify the signature using the public key

Instead of RSA we can use any Public Key cryptographic protocol

# RSA: Public-Key Crypto. Standard (PKCS)

- Signature: code hash of message ("digest") using private key
- PKCS-1: standard encrypt using secret key
- 0||1||at least 8 byte FF base 16|| 0|| specification of used hash function || hash(M)
- (M message to be signed)
  - first byte 0 implies encoded message is less than n
  - second byte (=1) denotes signature (=2 encoding)
  - bytes 11111111 imply encoded message is large
  - specification of used hash function increases security

# El-Gamal Signature Scheme

## [KPS § 6.4.4]

Discrete logarithm  
↑

### Generation

- Pick a prime  $p$  of length 1024 bits such that DL in  $\mathbb{Z}_p^*$  is hard
- Let  $g$  be a generator of  $\mathbb{Z}_p^*$
- Pick  $x$  in  $[2, p-2]$  at random, a secret
- Compute  $y = g^x \text{ mod } p$
- Public key:  $(p, g, y)$
- Private key:  $x$

# El-Gamal Signature Scheme

→ a new pair for every message we want sign  $(t, k)$

Signing M [a per-message public/private key pair  $(r, k)$  is also generated]

- Hash: Let  $m = H(M)$  digest  $M$ ,  $H$  is unkeyed
- Pick  $k$  in  $[1, p-2]$  relatively prime to  $p-1$  at random
- Compute  $r = g^k \text{ mod } p$  → mult; discrete inverse of  $k$
- Compute  $s = (m - rx)k^{-1} \text{ mod } (p-1)$  (\*\*\*)  
 • if  $s$  is zero, restart
- Output signature  $(r, s)$  → signature  
 ↳ Alice send  $(m, (r, s))$

# El-Gamal Signature Scheme

## Verify $M, r, s, p, k$

- Compute  $m = H(M)$
- Accept if  $(0 < r < p) \wedge (0 < s < p-1) \wedge (y^r r^s = g^m) \text{ mod } p$ , else reject
- What's going on?  $\} \text{dgm.}$
- By (\*\*\*)  $s = (m - rx)k^{-1} \text{ mod } p-1$ , so  $sk + rx = m$ . Now  $r = g^k$  so  $r^s = g^{ks}$ , and  $y = g^x$  so  $y^r = g^{rx}$ , implying  $y^r r^s = g^m$

# Digital Signature Standard (DSS)

- NIST, FIPS PUB 186
- DSS uses SHA as hash function and DSA as signature
- DSA inspired by El Gamal

see [KPS § 6.5]

- Let  $p$  be an  $L$  bit prime such that the discrete log problem mod  $p$  is intractable
- Let  $q$  be a 160 bit prime that divides  
 $p - 1: p = j \cdot q + 1 \rightsquigarrow p \text{ and } q \text{ related}$
- Let  $\alpha$  be a  $q$ -th root of 1 modulo  $p$ :  
 $\alpha = 1^{1/q} \bmod p, \text{ or } \alpha^q = 1 \bmod p$

*How do we compute  $\alpha$ ?*

digital signature

asked in exam

## The Digital Signature Algorithm (DSA)

a.y. 2021-22

18

# computing $\alpha$

→ answer of  
exam question

- take a random number  $h$   
s.t.  $1 < h < p - 1$  and compute  
 $g = h^{(p-1)/q} \text{ mod } p = h^j \text{ mod } p$  *j is integer number*
- if  $g = 1$  try a different  $h$ 
  - things would be unsecure
- it holds  $g^q = h^{p-1}$
- by Fermat's theorem  $h^{p-1} \equiv 1 \pmod{p}$ 
  - $p$  is prime and  $h$  is by construction not a multiple of  $p$
- choose  $\alpha = g$

# The Digital Signature Algorithm (DSA)

$p$  prime,  $q$  prime,  $p - 1 = 0 \pmod{q}$ ,  $\alpha = 1^{(1/q)} \pmod{p}$

**Private key:** secret  $s$ , random  $1 \leq s \leq q-1$ .

**Public key:**  $(p, q, \alpha, y = \alpha^s \pmod{p})$

**Signature on message  $M$ :**  $\xrightarrow{\text{used only in that signature, change every time, sub-key}}$

Choose a random  $1 \leq k \leq q-1$ , secret!!

Part I:  $(\alpha^k \pmod{p}) \pmod{q}$   $\xrightarrow{q \text{ smaller def}}$

Part II:  $(\text{SHA}(M) + s(\text{PART I})) k^{-1} \pmod{q}$

**Signature**  $\langle \text{Part I}, \text{Part II} \rangle$

Note that Part I Does not depend on  $M$  (preprocessing)

Part II is fast to compute

# The Digital Signature Algorithm (DSA)

- $p$  prime,  $q$  prime,  $p - 1 = 0 \pmod{q}$ ,  $\alpha = 1^{(1/q)} \pmod{p}$ , Private key: random  $1 \leq s \leq q-1$ . Public key:  $(p, q, \alpha, y = \alpha^s \pmod{p})$ . Signature on message  $M$ :
  - Choose a random  $1 \leq k \leq q-1$ , secret!!
    - Part I:  $(\alpha^k \pmod{p}) \pmod{q}$
    - Part II:  $(\text{SHA}(M) + s \text{ (PART I)}) k^{-1} \pmod{q}$
  - **Verification:**
    - $e_1 = \text{SHA}(M) (\text{PART II})^{-1} \pmod{q}$
    - $e_2 = (\text{PART I}) (\text{PART II})^{-1} \pmod{q}$
    - ACCEPT Signature if
    - $(\alpha^{e_1} y^{e_2} \pmod{p}) \pmod{q} = \text{PART I}$

# Digital Signature-correctness

- Accept if  $(\alpha^{e1} y^{e2} \text{ mod } p) \text{ mod } q = \text{PART I}$ 
  - $e1 = \text{SHA}(M) / (\text{PART II}) \text{ mod } q$
  - $e2 = (\text{PART I}) / (\text{PART II}) \text{ mod } q$
- Proof : 1. definition of PART I and PART II implies
- $\text{SHA}(M) = (-s(\text{PART I}) + k(\text{PART II})) \text{ mod } q$  hence
- $\text{SHA}(M)/(\text{PART II}) + s(\text{PART I})/(\text{PART II}) = k \text{ mod } q$
- 2. Definit. of  $y = \alpha^s \text{ mod } p$  implies  $\alpha^{e1} y^{e2} \text{ mod } p = \alpha^{e1} \alpha^{(se2)} \text{ mod } p$
- $\alpha^{\text{SHA}(M)/(\text{PART II}) + s(\text{PART I})/(\text{PART II})} \text{ mod } p = \alpha^{(k+cq)} \text{ mod } p$
- $\alpha^k \text{ mod } p$  (since  $\alpha^q = 1$ ).
- 3. Execution of  $\text{mod } q$  implies
- $(\alpha^{e1} y^{e2} \text{ mod } p) \text{ mod } q = (\alpha^k \text{ mod } p) \text{ mod } q = \text{PART I}$

# DSS: security [KPS § 6.5.5]

*master secret, fixed for each signature*

Secret key  $s$  is not revealed, and it cannot be forged without knowing it

Use of a random number for signing- not revealed ( $k$ )

- There are no duplicates of the same signature (even if same messages)
- If  $k$  is known, then you can compute  $s \bmod q = s$  ( $s$  is chosen  $< q$ )
  - make  $s$  explicit from PART II
- Two messages signed with same  $k$  can reveal the value  $k$  and therefore  $s \bmod q$ 
  - 2 equations (Part II and Part II'), 2 unknowns ( $s$  and  $k$ )

There exist other sophisticated attacks depending on implementation

# if adversary knows $k$ ...

$$[\text{Part II}] = (\text{SHA}(M) + s [\text{Part I}]) k^{-1} \bmod q$$

$$[\text{Part II}] k = (\text{SHA}(M) + s [\text{Part I}]) \bmod q$$

$$([\text{Part II}] k - \text{SHA}(M)) [\text{Part I}]^{-1} = s \bmod q = s \text{ (since } s < q)$$

then adv knows  $s$

now adv. wants to sign  $M'$  *forgetty*

- Part I =  $(\alpha^k \bmod p) \bmod q$  (independent on  $M'$ )
- Part II =  $((\text{SHA}(M') + s [\text{Part I}]) k^{-1}) \bmod q$

# DSS: efficiency

NONCE  
number to be  
used one time only

- Finding two primes  $p$  and  $q$  such that  $p - 1 = 0 \bmod q$  is not easy and takes time
- $p$  and  $q$  are public: they can be used by many persons
- DSS slower than RSA in signature verification
- DSS and RSA same speed for signing (DSS faster if you use preprocessing)
- DSS requires random numbers: not always easy to generate ↗ very good

# DSS versus RSA

DSS: (+) faster than RSA for signing (preprocessing- suitable for smart card)

(+?) uses random numbers to sign (+)

↳ PArtI don't depend on message

Implementation problems:

- To generate random numbers, you need special hardware (no smartcard);  
    *(We want very good random generator)*
  - pseudo random generator requires memory (no smart card)
  - Random number depending by messages does not allow preprocessing and slow the process
- (+) standard RSA: (+) known since many years and studied - no attacks  
(+) faster in signature verification

# DSA vs RSA

27

DSA: signature only

RSA: signature + key management

DH: Key management ↗

DSA: patent free (RSA patented until 2000)

DSA: short signatures (RSA 5 times longer: 40 vs 200 bytes)

DSA faster

# Timestamping a document

↳ explain exactly when a document is been created

TimeStamping Authority (TSA): guarantees timestamp of a document

Alice (A) wants to timestamp a document

1. A compute hash of document and sends to TSA
2. TSA adds timestamp, computes new hash (of timestamp and received hash) and SIGNS the obtained hash; sends back to A
3. A keeps TSA's signature as a proof
  - Everybody can check the signature
  - TSA does not know Alice's document

digital signature

a.y. 2021-22

- timestamping a digital signature of a normal document is a same thing, no difference.
- even if the certificate is expired if the signature and timestamping are correct, i accept.

openssl genrsa -out private.pem 4096 → bits for generate tsa keys

openssl tsa -in private.pem -certform pem -pubout -out pub.pem

openssl dgst -sha256 -sign private.pem -out data.txt.signature data.txt

openssl dgst -sha256 -verify pub.pem -signature data.txt.signature data.txt

↳ for make digital signature

- Pem is a format for keys

- Pubout for generate public key from the private key

*The generation of random numbers is too important to be left to chance*

---

**Robert Coveyou, Oak Ridge National Laboratory**

# random numbers

---

- ▶ we have seen that in many applications we need random number generators
- ▶ for example, we use random number generator to obtain session keys; to avoid key guessing
  - ▶ if an adversary sees a sequence of session keys, then he/she must NOT be able to guess next session key (even in a probabilistic way)
- ▶ note: good random number generator for cryptography is expensive

# generators of randomness

## two principal methods

- ▶ measure some physical phenomenon that is expected to be random and then compensates for possible biases in the measurement process (**random number generator, RNG**)
- ▶ use algorithms producing long sequences of apparently random results, which are in fact completely determined by an initial value, known as a **seed** or **key** (**pseudorandom number generator, PRNG**)
  - in case of compromise adversary can see state of algorithm (all variables, ecc.)
- ▶ deterministic computations cannot give true random numbers because they are inherently predictable
- ▶ distinguishing a "true" random number from the output of a pseudo-random number generator is a very difficult problem
  - ▶ carefully chosen pseudo-random number generators can be used instead of true random numbers in many applications
  - ▶ rigorous statistical analysis of the output is often needed to have confidence in the algorithm

*If adversary finds a random number can't guess the future  
one, we want perfect forwards secret*

## random number generator

- ▶ use of systems data
- ▶ use of external information ↗ from mature : temperature, ...
- ▶ above data are used to initialize a pseudorandom number generator, a program that:
  - ▶ given an initial seed (random)
  - ▶ obtains a long sequence of numbers
- ▶ if an adversary sees a long sequence of numbers in output it should be computationally very expensive to guess next output number (even in a probabilistic sense)

# initial seed

---

## different sources

- ▶ machine /network

---

- ▶ clock
- ▶ free space on disk
- ▶ number of files on disk
- ▶ info on operating system (I/O queues, buffer state etc.)
- ▶ user information (e.g., windows and size of windows)
- ▶ inter-arrival time of packets

- ▶ user

- ▶ keyboard & mouse timing

## initial seed: how many bits of randomness?

---

- ▶ in many cases, sources provide few bits of randomness (in fact much information can be easily guessed):
    - ▶ clock: we can use only low order digit (e.g., milliseconds: 10 bits of randomness)
    - ▶ day, hour and minute can be easily guessed and are not random
  - ▶ it is advantageous to mix several sources of randomness using crypto functions
-

# common mistakes

---

- ▶ **using small initial seed**
  - ▶ ex.: 16 bits seed give only 65536 different seeds and attacker can try them all
- ▶ **using current time**
  - ▶ if clock granularity is 10 msec then if we approximately know the time (just the hour) there are  
$$60 \text{ (minutes)} \times 60 \text{ (seconds)} \times 100 \text{ (hundredths of a second)} = \\ = 360000 \text{ different choices only!}$$
- ▶ **divulging seed value**
  - ▶ an implementation used the time of day to choose a per-message encryption key; the time of day in this case may have had sufficient granularity, but the problem was that the application included the time of day in the unencrypted header of the message!

# Netscape 1.1 seeding & key generation (1996)

---

```
global variable seed;
```

```
RNG_CreateContext()
(seconds, microseconds) = time of day;
/* Time elapsed since 1970 */
pid = process ID; ppid = parent process ID;
a = mklcpr(microseconds);
b = mklcpr(pid + seconds + (ppid << 12));
seed = MD5(a, b);

mklcpr(x) /* not cryptographically significant */
return ((0xDEECE66D * x + 0x2BBB62DC) >> 1);
```

```
RNG_GenerateRandomBytes()
x = MD5(seed);
seed = seed + 1;
return x;
```

# MD5 (Message-Digest 5, by Ron Rivest, '91)

---

- ▶ widely used cryptographic hash function with a 128-bit hash value
  - ▶ an MD5 hash is typically expressed as a 32-digit hexadecimal number
- ▶ internet standard (RFC 1321)
- ▶ employed in a wide variety of security applications, and is also commonly used to check the integrity of files
- ▶ **not collision resistant**
  - ▶ not suitable for applications like SSL certificates or digital signatures that rely on this property

# about collision weakness of MD5

---

in 1995, collisions were found in the compression function of MD5, and Hans Dobbertin wrote in the RSA Laboratories technical newsletter, "The presented attack does not yet threaten practical applications of MD5, but it comes rather close ... in the future MD5 should no longer be implemented...where a collision-resistant hash function is required."

---

in 2005, researchers were able to create pairs of PostScript documents and X.509 certificates with the same hash. Later that year, Ron Rivest wrote, "MD5 and SHA1 are both clearly broken (in terms of collision-resistance)," and RSA Laboratories wrote that "next-generation products will need to move to new algorithms."

---

on 30 December 2008, a group of researchers announced that they had used MD5 collisions to create an intermediate certificate authority certificate which appeared to be legitimate when checked via its MD5 hash

# assessing randomness in Netscape 1.1

---

- ▶ if attacker does not have an account on the attacked UNIX machine pid and ppid are unknown
- ▶ even though the pid and ppid are 15 bit quantities on most UNIX machines, the sum pid + (ppid << 12) has only 27 bits, not 30
- ▶ if the value of seconds is known, a has only 20 unknown bits, and b has only 27 unknown bits. This leaves, at most, **47 bits of randomness** in the secret key
- ▶ in addition, ppid & pid can be hacked
  - ▶ ppid is often 1 (for example, when the user starts Netscape from an X Window system menu); if not, it is usually just a bit smaller than the pid
  - ▶ mail-transport agent sendmail generates Message-IDs for outgoing mail using its process ID; as a result, sending e-mail to an invalid user on the attacked machine will cause the message to bounce back to the sender; the Message-ID contained in the reply message will tell the sender the last process ID used on that machine. If the user started Netscape relatively recently, and that the machine is not heavily loaded, this will closely approximate Netscape's pid

# random number bug in Debian Linux (2006)

---

- ▶ announced in 2008, affecting Debian + \*Ubuntu
  - ▶ <http://www.debian.org/security/2008/dsa-1571>
- ▶ vulnerability in the OpenSSL package, caused by the removal of the following two lines of code from `md_rand.c`

```
MD_Update(&m,buf,j);
[ .. ]
MD_Update(&m,buf,j); /* purify complains */
```
- ▶ lines removed because they caused Valgrind (OS debugging tool) and Purify (a proprietary debugger, from IBM) to warn about use of uninitialized data in any code linked to OpenSSL
- ▶ instead of mixing in random data for the initial seed, the only used "random" value was the current process ID
  - ▶ on the Linux platform, the default maximum process ID is 32768, resulting in a very small number of seed values being used for all PRNG operations

# BSI evaluation criteria (for pseudo-random generation)

- ▶ Bundesamt für Sicherheit in der Informationstechnik (BSI - in *german*)  
English: Federal Office for Information Security)
- ▶ defines four criteria for quality of deterministic random number generators, named K1, K2, K3 and K4
- ▶ K1 — A sequence of random numbers with a low probability of containing identical consecutive elements (runs). (*a run in a sequence of bits it is a subsequence of consecutive elements all equals*)
- ▶ K2 (see next slide)
- ▶ K3 — It should be impossible for attacker to calculate/guess, from any given sub-sequence, any previous or future values in the sequence, nor any inner state of the generator
- ▶ K4 — It should be impossible for attacker to calculate/guess from an inner state of the generator, any previous numbers in the sequence or any previous inner generator states
- ▶ for cryptographic applications, only generators meeting the K4 standard are acceptable

▶ 15

deterministic random numbers a.y. 2022-23  
↑

- State : *values of all variables of algorithm, that is public for open security.*

## K2 class is a class of criteria

- ▶ the generated sequence of numbers is indistinguishable from "true random" numbers according to specified statistical tests, must pass these tests:
  - ▶ monobit test
    - ▶ equal numbers of ones and zeros in the sequence
  - ▶ poker test
    - ▶ a special instance of the  $\chi^2$  (chi-square) test, a family of tests
  - ▶ runs test
    - ▶ counts the frequency of runs of various lengths
  - ▶ longruns test
    - ▶ checks whether there exists any run of length 34 or greater in 20 000 bits of the sequence
  - ▶ autocorrelation test
- ▶ these requirements are a test of how well a bit sequence:
  - ▶ has zeros and ones equally often;
  - ▶ after a sequence of  $n$  zeros (or ones), the next bit a one (or zero) with probability one-half;
  - ▶ and any selected subsequence contains no information about the next element(s) in the sequence.

▶ 16

random numbers a.y. 2022-23

autocorrelation

cryptographically secure pseudorandom number generators (CSPRNG) *is what we want*

meet requirements of ordinary PRNG, and

► **next-bit test**

- ▶ given the first  $k$  bits of a random sequence, there is no polynomial-time algorithm that can predict the  $(k+1)$ th bit with probability of success better than 50%

► **withstand "state compromise extensions"**

- ▶ if part or all its state has been revealed (or guessed correctly), it should be impossible to reconstruct the stream of random numbers prior to the revelation, *occuted in the past*
- ▶ additionally, if there is an entropy input while running, it should be infeasible to use knowledge of the input's state to predict future conditions of the CSPRNG state, *impossible to obtain unless you are using some extra input during the run of the algorithm*

Simple PRNG is not cryptographical, we want other requirement.

# CSPRGN: statistical definition, other definition

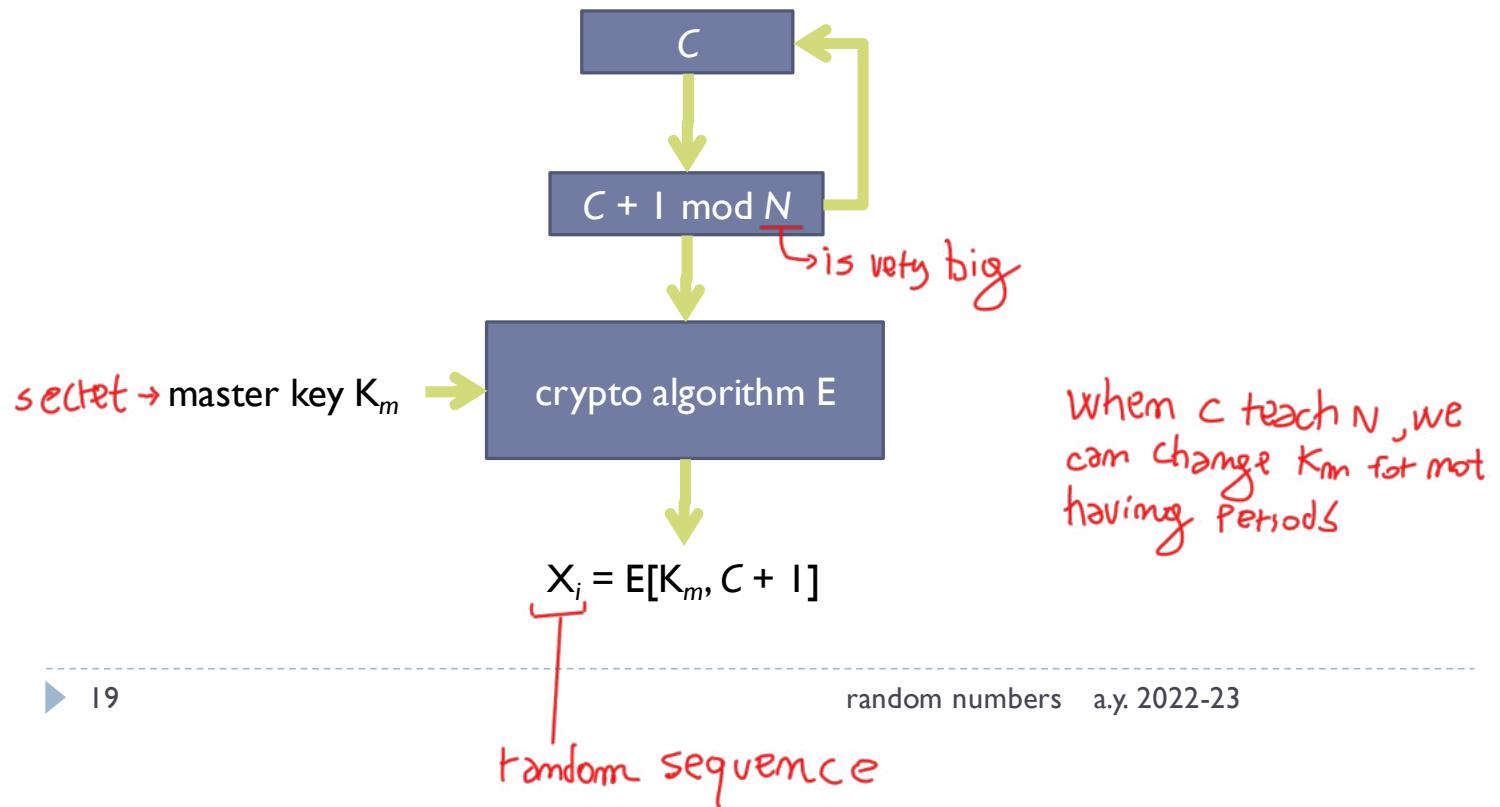
no polynomial time algorithm can distinguish with probability  $> 0.5$  a truly random sequence from a pseudo-random sequence of the generator

open problem

*is there any way to distinguish the output of a high-quality PRNG from a truly random sequence without knowing the algorithm(s) used and the state with which it was initialized?*

# Some good generator: generator "cipher of a counter"

- ▶ **cyclic cryptography**: use counter + cipher
  - ▶ Meyer and Matyas, 1982



## RSA based generator

---

- ▶ prime numbers  $p, q$
- ▶  $n = p \cdot q$
- ▶ integer  $e$  s.t.  $\text{GCD}(e, \varphi(n)) = \text{GCD}(e, (p-1) \cdot (q-1)) = 1$
- ▶  $z = \text{seed}$
- ▶ loop

$z_i = (z_{i-1})^e \bmod n \rightsquigarrow$  next value obtain by previous value

$i = i + 1 \rightsquigarrow i$  increasing in time

output: least significant bit of  $z_i$

*set of standards coming from USA*

## ANSI X9.17 a standard for random numbers

- ▶ strong generator, using 3DES (triple DES, EDE)
- ▶ input: seed s of 64 bit, integer m, triple DES key, D (encoding of date and time)
- ▶ output:  $x_1, x_2, \dots, x_m$  sequence of m strings (64 bits in the whole)

$$Y = 3DES(D)$$

for  $i = 1$  to  $m$  do

$$x_i = 3DES(Y \oplus s)$$

$$s = 3DES(x_i \oplus Y)$$

return  $x_1, x_2, \dots, x_m$

# Blum Blum Shub (CSPRNG)

- ▶ choose  $p, q$  big prime s.t.  $p \equiv q \equiv 3 \pmod{4}$
- ▶  $n = p \cdot q$
- ▶ randomly choose  $s$  s.t.  $\text{GCD}(s, n) = 1$
- ▶ output the sequence of bits  $B_i$

$$X_0 = s^2 \pmod{n}$$

for  $i = 1$  to  $\infty$

$$X_i = (X_{i-1})^2 \pmod{n}$$

$$B_i = X_i \pmod{2} \quad \text{→ last bit of } B_i$$

return  $B_i$

## more on randomness

---

- ▶ subroutines generally provided in programming languages for "random numbers" are not designed to be unguessable, but just designed merely to pass statistical tests
- ▶ hash together all the sources of randomness you can find, e.g., timing of keystrokes, disk seek times, count of packet arrivals, etc.
- ▶ for more reading about randomness, see RFC 1750.

# RFC 1750 (1994)

---

## Randomness Recommendations for Security

Security systems today are built on increasingly strong cryptographic algorithms that foil pattern analysis attempts. However, the security of these systems is dependent on generating secret quantities for passwords, cryptographic keys, and similar quantities. The use of pseudo-random processes to generate secret quantities can result in pseudo-security. The sophisticated attacker of these security systems may find it easier to reproduce the environment that produced the secret quantities, searching the resulting small set of possibilities, than to locate the quantities in the whole of the number space.

Choosing random quantities to foil a resourceful and motivated adversary is surprisingly difficult. This paper points out many pitfalls in using traditional pseudo-random number generation techniques for choosing such quantities. It recommends the use of truly random hardware techniques and shows that the existing hardware on many systems can be used for this purpose. It provides suggestions to ameliorate the problem when a hardware solution is not available. And it gives examples of how large such quantities need to be for some particular applications.

# Authentication



## Motivation

Authentication protocols in different scenarios  
X.509, Kerberos

↳ somewhat related to data integrity . For messages we can provide authentication tag for having authentication.

## Authentication

2

- Alice needs to show her identity to Bob; she needs to get a service, or to access information, or a resource etc.
- Bob needs to be sure of Alice's identity:
  - Who is Alice?
- Trudy tries to impersonate Alice:
  - Once
  - Many times

# Authentication

3

- The process of reliably verifying the identity of someone (or something).
- In human interaction:
  - People who know you can recognize you based on your appearance or voice
  - A guard might authenticate you by comparing you with the picture on your badge
  - A mail order company might accept as authentication the fact that you know the expiration date on your credit card
- Two interesting cases
  - a computer is authenticating another computer
  - a person is using a public workstation that can perform sophisticated operations, but it will not store secrets for every possible user
    - the user's secret must be remembered by the user

# Closed world assumption in authentication

Presumption that what is not currently known to be true, is false

Negation as failure is related to closed world assumption, as it believes false every predicate that cannot be proved to be true

the opposite of the closed world assumption is the open world assumption, stating that lack of knowledge does not imply falsity

true is only what is proved to be true, other is all false

When authentication is failing we consider the failure exactly like a fail authentication

## Closed environment

5

- Authentication in a closed environment (e.g., company, home)
- We use a third-party Carole (trusted server) distributing required information for authentication (before authentication or using secret communication)
- Trudy is a legal user and might use the connection Alice-Bob, usually is a colleague

Usually use a third party for authentication, a trust server. Server shares a secret key with any user

## **Authentication of people**

7

### Use

- What you know (passwords)
- What you have (smart card)
- Who you are (biometric tools)
- Where you are (network address - weak, because of spoofing)

# Password

8

authentication-1

a.y. 2022-23

- Humans:

- Short keys, that possibly allow to obtain long keys
- Sometimes easy to guess

- Computers:

- Long keys
  - Hidden (not stored in clear): either stored encrypted or indirectly
  - One-time password
- a well know threat is the login Trojan horse: the attacker prompts a fake login window that induces the innocent user to prompt the password that is collected by the attacker



attacker give a screen samp of login page to user, where he insert credential, store the values and after that give the original login page.

User believe that have mistake to write the password but it's not the situation



# Biometric

- There are many devices that are used to authenticate:
- Retina examination, fingerprint reader, voice, or based on other characteristics (ex. handwritten signatures, keystroke timing: timing in using the keyboard or the mouse etc.)
- **Accuracy:**
  - Error: false positive and false negative
  - They are used in specific scenarios (sometimes to enforce other methods)

a.y. 2022-23

authentication-1

false positive is the  
big problem!!

10

# Biometric

## Fingerprinting:

- Fake fingerprint, dead people
- Not stable over time: children, people doing manual work (possible damage)

## Voice

- Use of tape; deepfake
- voices are affected: flu and colds; noise in the background, telephone etc



# Biometric

- Keystroke timing:
  - Each person has different speed in typing
  - Typing faster than personal limit is impossible
- Handwritten signature:
  - Poor quality
  - Electronic tablet, for signature + timing

## Authentication scenarios

13

In the following we will consider  
authentication using knowledge of  
a key (public key or secret key,  
possibly stored in a smart card)

1. Users (Alice and Bob) share a secret key
2. Users share a key with Carole, trusted authority (authentication server)
3. Users have a public key

if used a third parties for authentication,  
sharing a private key , a symmetric key.

# Techniques (tools)

14

Users are authenticated using a key

Techniques:

- timestamp *random number used at some time.*
- nonce (or challenge): random number chosen by the person that authenticates to verify the other knows the key
- sequence number in protocols *(used in TCP)*

With nonce using a challenge, expect a certain behavior from the user

# Cryptography vs. authentication

a.y. 2022-23

- Trudy attacks the protocols using non authentic messages (possibly messages that have been exchanged between Alice and Bob in previous application of the protocol)
- Need to guarantee authentication and integrity
- Encryption DOES NOT guarantee authenticity of the message  
*(only confidentiality)*

authentication-1

15

*Assumed*  
PSK - Pre shared key  
**Authentication**  
**by**  
**symmetric key**

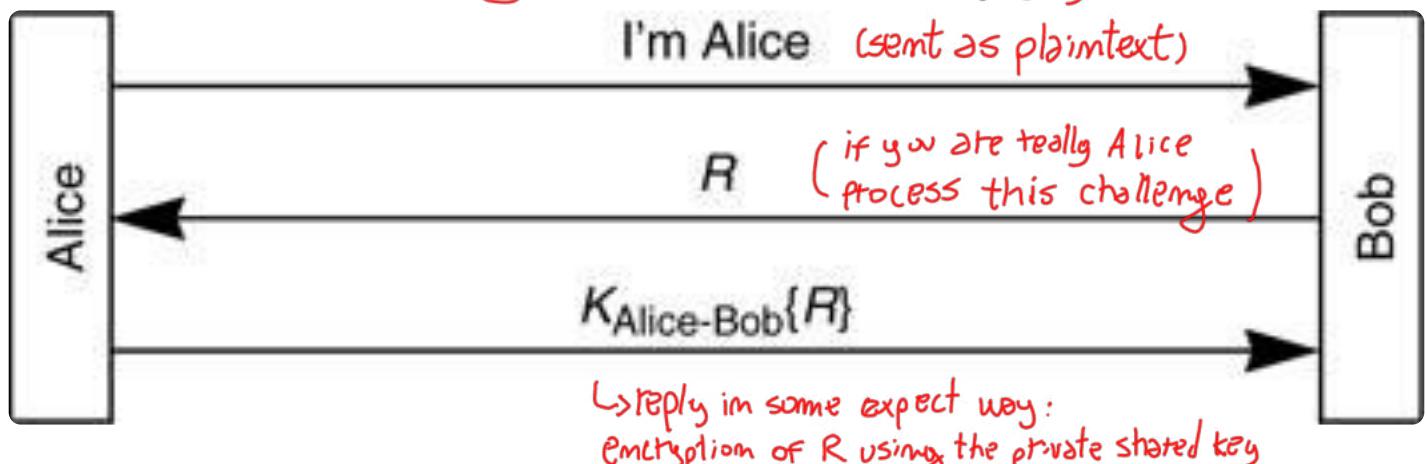
*not using any third party*

Alice and Bob share a secret key  $K_{Alice-Bob}$

$K\{M\}$  denotes M encrypted with secret key K

- One way authentication using nonce (challenge)
- One way authentication using timestamps
- Two ways (mutual) authentication using nonce

note can use hashing Function



challenge/response  
based on shared  
secret

- authentication is not mutual. Bob authenticates Alice, but Alice does not authenticate Bob
- an eavesdropper could mount an off-line password-guessing attack (assuming  $K_{Alice-Bob}$  is derived from a password), knowing R and  $K_{Alice-Bob}\{R\}$
- R is a nonce

a.y. 2022-23

authentication-1

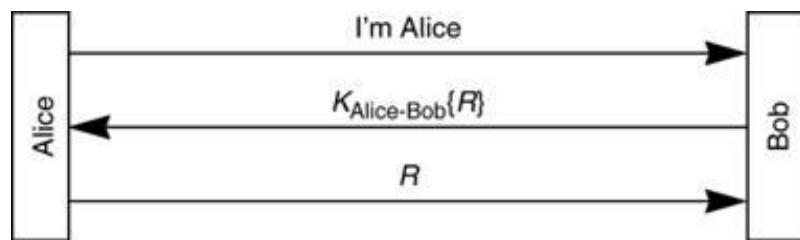
17

next time use a different shared key

Alice asking authentication to Bob, because after Alice sending messages, asking for some services. There will be a communication.

## variant

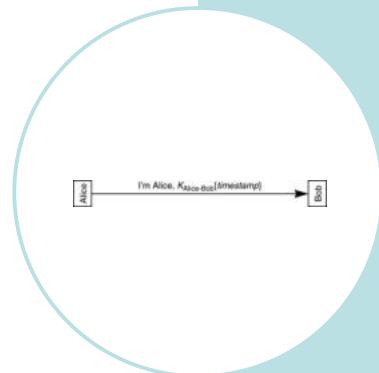
- requires reversible cryptography
- still possible the dictionary attack by eavesdropper
- if  $R$  has limited lifetime (e.g., random number + timestamp) Alice authenticates Bob because only someone knowing  $K_{\text{Alice-Bob}}$  could generate  $K_{\text{Alice-Bob}}\{R\}$ 
  - limited lifetime required to foil replaying of an old  $K_{\text{Alice-Bob}}\{R\}$



here can not use hashing function, must use a reversible cryptography

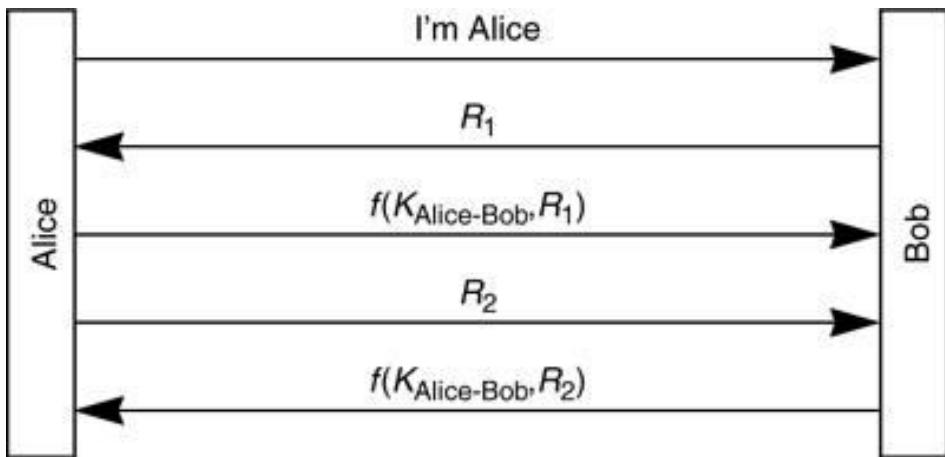
# timestamp based

- Bob and Alice have reasonably synchronized clocks (can be a weakness)
- Alice encrypts the current time, Bob decrypts the result and makes sure the result is acceptable (i.e., within an acceptable clock skew)
- efficient, no intermediate states
- if Bob remembers timestamps until they expire, then no replaying attacks
- if multiple servers with same secret K, then Alice can send  $K\{Bob|timestamp\}$



Nonce can have a lifetime, implementing with a timestamp.  
Requiring synchronization between clocks, attacker can use this fact.

# mutual authentication

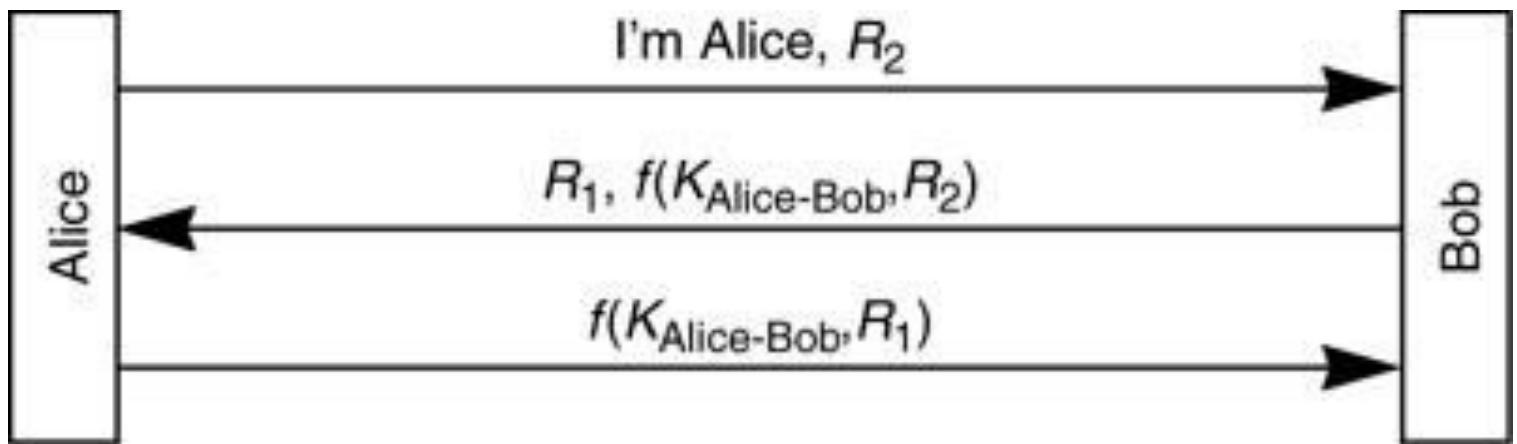


- many messages!
- perhaps a few ones can be saved... but  
schema is attackable

also Alice challenges Bob sending a Nonce

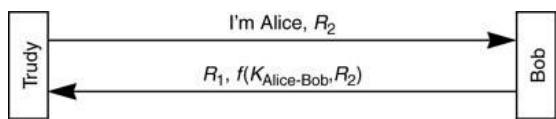
# optimized mutual authentication

save messages but weak to reflection attack

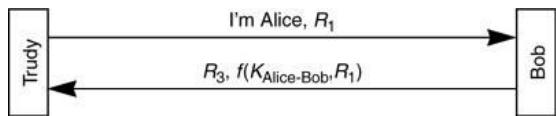


timeout always needed

# reflection attack



another session, attacker use Nounce  $R_1$  of above session, Bob can solve the challenge and reply correctly



Now attacker can use the reply in the original session and pass the authentication challenge

- Trudy wants to impersonate Alice to Bob
- two sessions, the 2<sup>nd</sup> will be incomplete but will allow Trudy to complete the 1<sup>st</sup> one

not used only in this situation, based on the idea of multiple sessions.

attacker (Trudy) behave like Alice with  $R_2$  like Nounce.



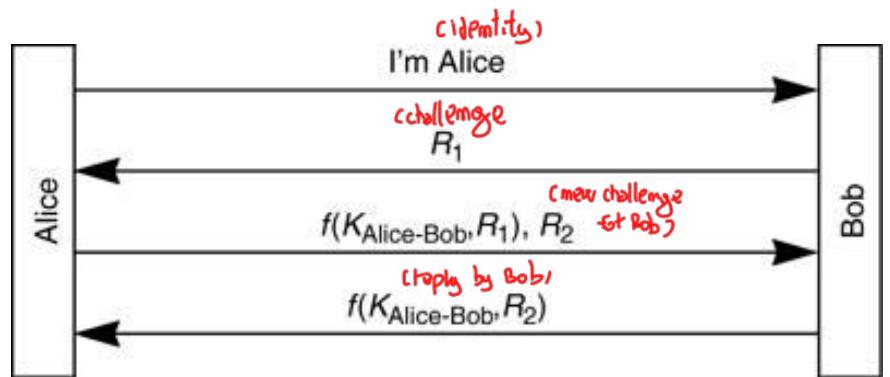
# how to prevent reflection attack?

- use different keys
  - $K_{\text{Alice-Bob}}$  and  $-K_{\text{Alice-Bob}}$ , or  $K_{\text{Alice-Bob}} + 1$
- different challenges, i.e.,  
challenge from initiator  
different from challenge from responder
  - e.g., even number at initiator's side, odd number at responder's side

↳ challenge can't be replaced

# less optimized mutual authentication

- Trudy can mount an off-line password-guessing attack by impersonating Bob's address and tricking Alice into attempting a connection to her



# Trusted party

only in closed environment,  
not possible in all over  
internet. Ex. Enterprise

25

It is not possible that all users share a secret key (quadratic number of keys, each user must have a different key for everyone)

**Scenario:** users share a key - password - with trusted authority C - C authentication server (aka Key Distribution Center (KDC))

→ given asked session key

- A and B share a secret key with C ( $K_{AC}$  e  $K_{BC}$ )
- A and B might not be human users but also entity of the system (e.g., printer, databases etc.)
- **Goals: authenticate A (or A and B);**
  - optional: decide on a session key to be used between A and B for short time

- session key is a random number, low probability that is used before, but attacker in case of compromise store it for later use.
- attacker usually is an user of the system

## Authentication server

26

**Goal:** Alice and Bob must authenticate and choose a secret session key K - used for short time (session)

At the end of the authentication protocol:

1. only A and B know K (besides C - trusted server)
2. Each should be sure of fact 1 above
3. K is a new key (randomly chosen, not used before) session key

## Authentication server - attacker's strength

Trudy (attacker) can

- be a legitimate user of the system  
(share a key with C)
- sniff and spoof messages
- concurrently run more than one session with A, B and C
  - different execution of the protocol can be done interleaved
  - T can convince A and/or B to start a new session with T
- Might know old session keys

## Authentication server - attacker's limits

### Trudy

- is NOT able to guess random numbers chosen by A or B
- does not know keys  $K_{AC}$  and  $K_{BC}$  (in general does not know secret keys of other users)
- is not able to decode in a short time messages encrypted with unknown keys

## Authentication with trusted server

↳ still not perfect, is attackable

trusted parties

A and B share a key with C ( $K_{AC}$  and  $K_{BC}$ , respectively)

1. A sends to C: A, B (I'm Alice want to start a conversation with B)
2. C chooses K - session key - and sends to A: (only to Alice send K)  $K_{AC}(K)$  and  $K_{BC}(K)$  ( $K_{AC}(K)$  is decrypted by A,  $K_{BC}(K)$  not)
3. A decrypts, computes K and sends to B: (only  $K_{AC}(K)$ )  $C, A, K_{BC}(K)$  I'm Alice and this is the key shared with C
4. B decrypts  $K_{BC}(K)$ , finds K and sends to A: K(Hello A, this is B)

# Attack - 1

can intercept all communication start by A: A-C, A-B !

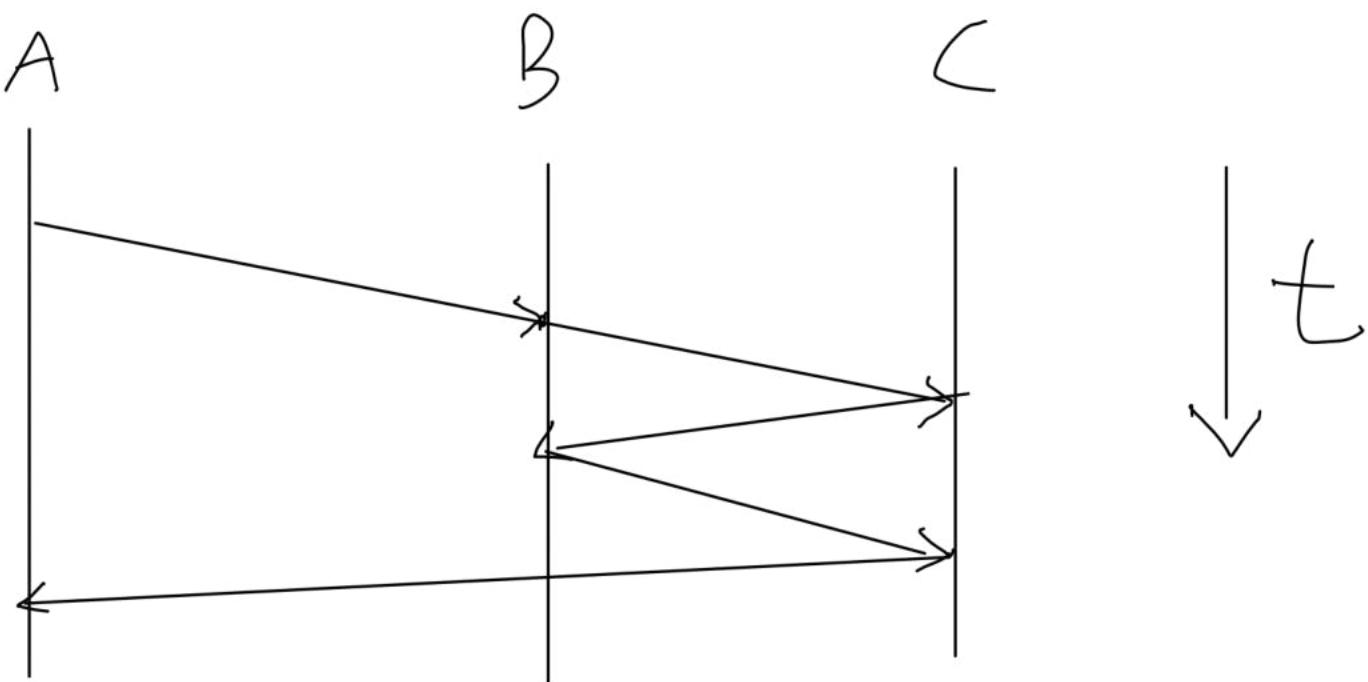
Assume T can sniff, spoof and make MITM attack (T is in the middle of both A to C and A to B communication)

1. A sends to T (instead of A sends to C) : A,B
2. immediately T sends to C: A,T
3. C chooses K and sends to A:  $K_{AC}(K)$  and  $K_{TC}(K)$
4. A sends to B: C, A,  $K_{TC}(K)$  - T intercepts
5. T sends to A K(Hello A, this is B)

Possible modification of step 1 (previous slide):

A sends to C < A,  $K_{AC}(B)$  > (in this way C still knows knows that A wants to talk to B)

for describing protocols, use vertical arrows, where vertical is time:



## Attack - 2

### Modified protocol

1. A sends to C:  $A, K_{AC}(B)$
2. C chooses K, and sends to A:  $K_{AC}(K)$  and  $K_{BC}(K)$
3. A decrypts K and sends to B:  $C, A, K_{BC}(K)$
4. B decrypts K and sends to A : K(Hello A, this is B)

Bob is ciphertext, attacker must decrypt  
to know who is receiver of A.

Note: T does not know that A wants to talk to B

# Attack - 3

mote difficult,  
Probability of attack is lower

Consider modified protocol

1. A sends to C :  $A, K_{AC}(B)$

T gets A's message and sends to C (as A):  $A, K_{AC}(T)$   
(REPLAYS part of some previously exchanged message)

Note: T does not know whom A wants to talk to (not yet)

2. C chooses K, and sends to A :  $K_{AC}(K)$  and  $K_{TC}(K)$

3. A decrypts K and sends to B :  $C, A, K_{TC}(K)$

4. T gets the message and finds that A wants to talk to B; T now acts in place of B and sends to A K(Hello A, this is B)

Note: T knows the identity the person A wants to talk to only at the end of the protocol

the replay attack is an attack against authentication.

# Protocol

## Needham-Schroeder

basis for the Kerberos protocol and aims to establish a session key between two parties on a network, typically to protect further communication

N, nonce: random number

N-S. protocol based on challenge-response :

1. A chooses N (nonce) and sends to C: A, B, N
  2. C chooses K and sends to A:  $K_{AC}(N, K, B, K_{BC}(K, A))$
  3. A decrypts, checks N and B, and sends to B:  $K_{BC}(K, A)$
  4. B decrypts, chooses nonce N' and sends to A:  $K_{BC}(B, N')$
  5. A sends to B  $K_{BC}(A, N' - 1)$   
now B has checked A
- identity of Alice, or Bob  
and the Nonce.
- identity of Alice, or Bob  
and the Nonce.
- identity of Alice, or Bob  
and the Nonce.
- identity of Alice, or Bob  
and the Nonce.
- identity of Alice, or Bob  
and the Nonce.

Note: B does not directly communicate with C

this protocol can be attacked is not perfect

# Attacking N.-S. Protocol

1. A chooses N (nonce) and sends to C: A, B, N
2. C chooses K and sends to A:  $K_{AC}(N, K, B, K_{BC}(K, A))$   
now T acts in place of A
3. A decrypts, checks N and B and sends to B:  $K_{BC}(K, A)$   
T (as A) replays to B:  $K_{BC}(K', A)$ , where K' is an older session key (it is an old message)
4. B decrypts, chooses nonce N' and sends to T (not to A) K' (this is B, N')
5. T sends to B: K' (this is A, N' - 1)

Note:

1. T uses old session key and acts in place of A
2. B is not sure that C is active: there is no direct exchange between B and C

difficult to obtain is  
needed  $\Rightarrow$  Compromised

# Authentication attack (Denning and Sacco, '81)

- two sessions
- assume that Trudy has recorded session 1 and that K is compromised
- after session 2, B is convinced that he shares the secret key K only with A

# session 1, session 2

I(B)

↳ intended that send  
to B

1.  $A \rightarrow C: A, B, N$
2.  $C \rightarrow A: K_{AC}(N, B, K, K_{BC}(K, A))$
3.  $A \rightarrow I(B): K_{BC}(K, A)$

assume that  $K$  is compromised

4.  $I(A) \rightarrow B: K_{BC}(K, A)$  replay
5.  $B \rightarrow I(A): K(N')$
6.  $I(A) \rightarrow B: K(N' - 1)$

B is now convinced that he shares secret key K only with A

# Challenge-response symmetric key

How to guarantee data integrity

- timestamps (a message is valid only in a small time window)
- sequence number (A and B remember sequence number of exchanged messages to avoid replay attacks in which the attacker sends old messages)
- nonce should be used carefully

# Needham-Schroeder Protocol (variant)

use of timestamp

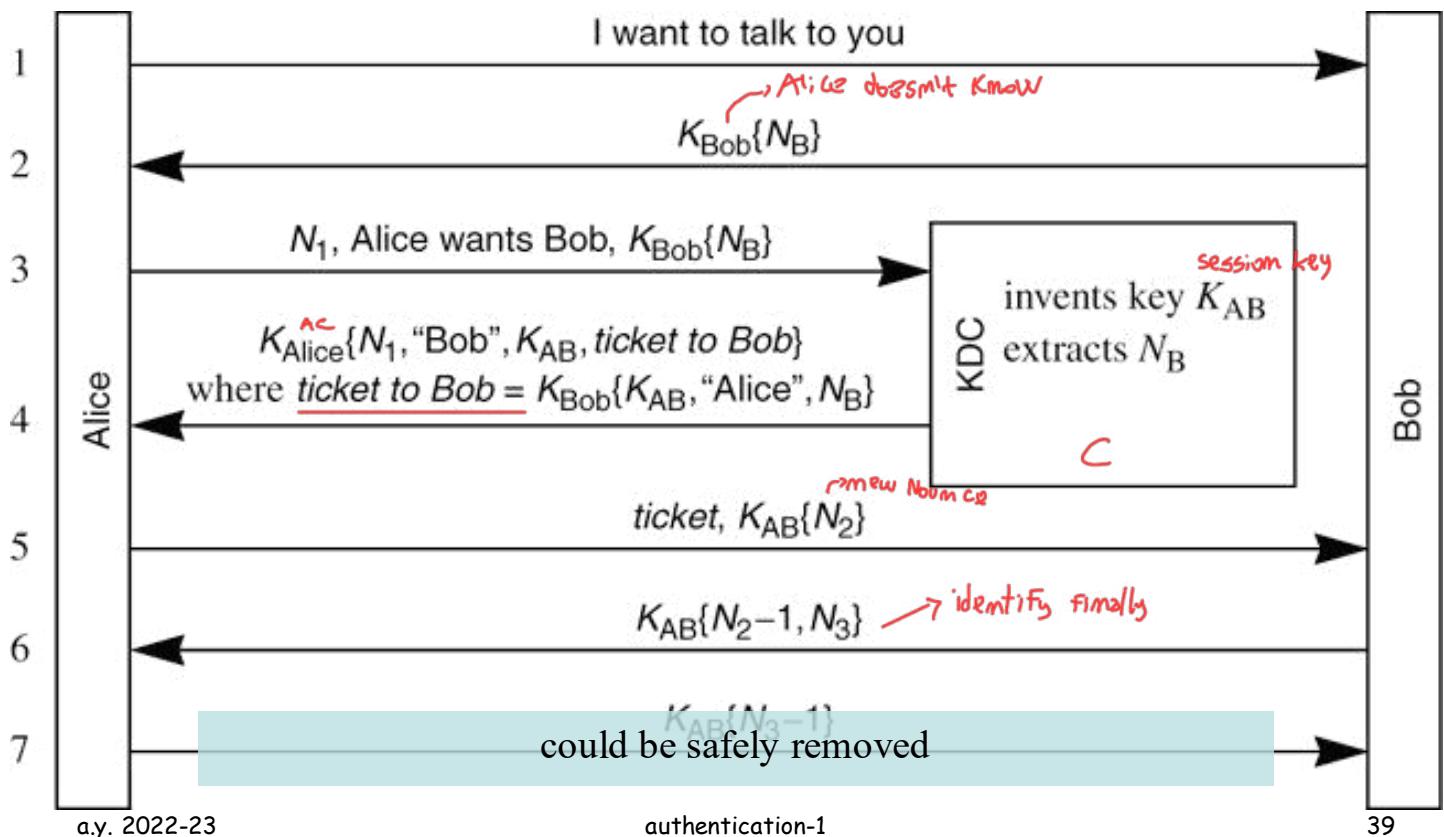
Modified Challenge-Response (nonce and timestamp):

- C exchanges messages with both A and B
  - timestamp  $t$  used only between B and C
  - K session secret key
1. A chooses N and sends to B:  $\langle A, N \rangle$
  2. B chooses  $N'$  and sends to C:  $\langle B, N', K_{BC}(N, A, t) \rangle$
  3. C sends to A:  $\langle K_{AC}(B, N, K, t), K_{BC}(A, K, t), N' \rangle$
  4. A sends to B:  $\langle K_{BC}(A, K, t), K(N') \rangle$

first to Bob

Basis for the Kerberos protocol

# Expanded Needham-Schroeder



a.y. 2022-23

authentication-1

39

not perfect, but very difficult

# Authentication Scenarios

- Users share a password with a trusted authority (authentication servers)

Kerberos

kerberos

1

# Challenge-response Symmetric Key

setver state  
the Nounce for  
a time defined by  
2 timestamps

a.y. 2022-23

for protect from replays

What do we learn from previous attacks?

Timestamps: are valid only in a small-time window

Sequence numbers attached to messages are useful (to avoid replication attacks)

Nonce: we should carefully use them (and we require good random number generator)

kerberos

2

# Kerberos

---

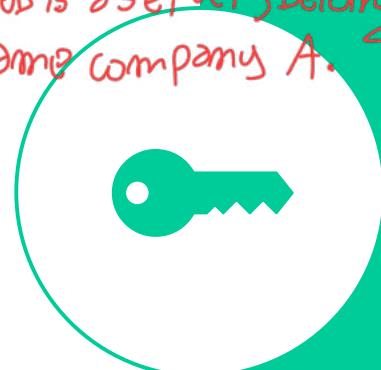
- Kerberos provides authentication in distributed systems
  - Guarantees safe access to network resources (e.g., printer, databases etc.)
  - There is a central authority that allows to reduce the number of passwords that users must memorize
- Reference:
  - proposed by MIT  
<http://web.mit.edu/kerberos/www/dialogue.html>
  - free download in US and Canada (after 2000, in most locations)
  - widespread use (most operating systems)

# Kerberos

most used in LAN where A and B are in same enterprise

- Scenario: A needs to access service provided by B
  - Authentication of A
  - Optional: authentication of B
  - Optional: decide session keys for secret communication and/or authentication
- C is trusted server (authority that shares keys with A and B)
- Idea: use ticket to access services; tickets are valid in each time window, after this expire,
  - We will use something called 'AUTHENTICATOR', in addition to ticket, to provide further authentication to Alice.

Bob is a user, belong to same company A.



# Kerberos

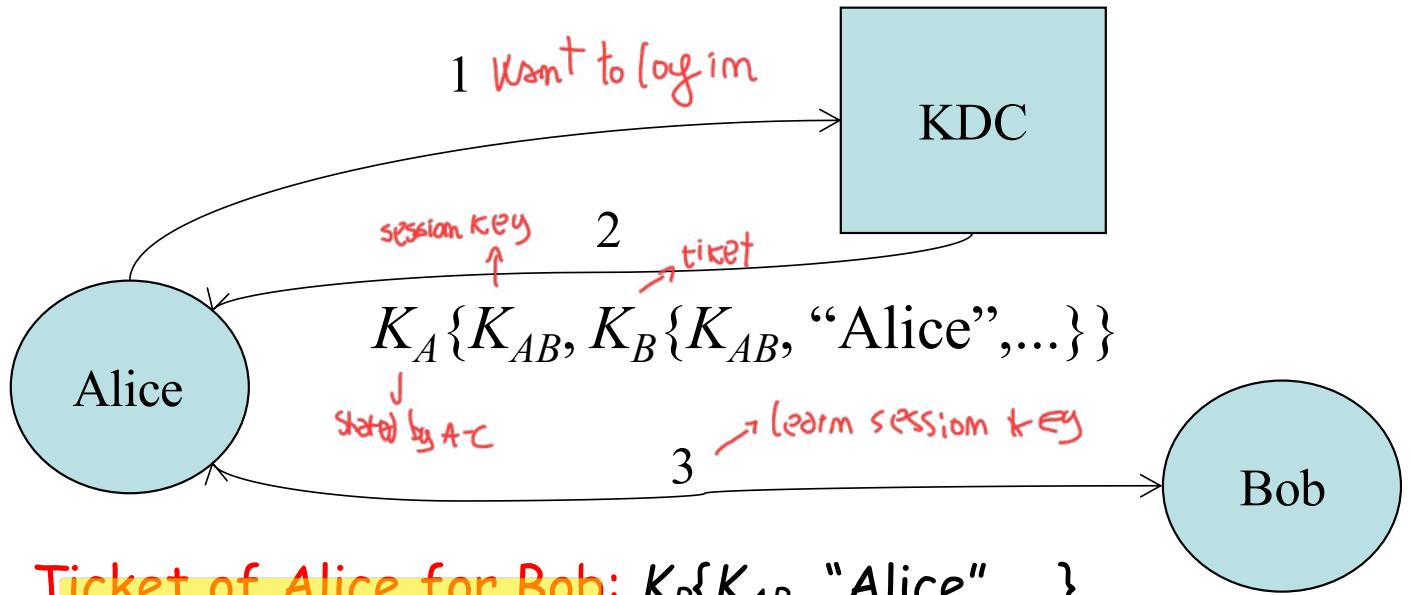
- KDC (Key Distribution Center) is the server (both trusted and physically safe)
- Messages are safe with respect to cryptographic attacks and data integrity (only in last version)
- Kerberos provides security for applications like
  - telnet
  - rtools (rlogin, rcp, rsh)
  - network file systems (NFS/AFS)
  - e-mail
  - printer servers
  - ...

make secure these protocols that normally are not secure.

# Preliminaries

- Each user (also named as principal) has a master secret key with KDC
  - for human users master secret key is derived from password
  - for system resources, keys are defined while configuring the application
- Each principal is registered by the KDC
- All master keys are stored in the KDC database, encrypted with the KDC master key

# Tickets, Alice, Bob, KDC



**Ticket of Alice for Bob:**  $K_B\{K_{AB}, \text{"Alice"}, \dots\}$ ,  
 $K_A$  master key of Alice,  $K_B$  master key of Bob,  
 $K_{AB}$  session key to be used by A and B  
only Bob can decrypt and checks the message

# Tickets

- a ticket is encrypted with the secret key associated with the service
  - ticket basically contains
    - sessionkey
    - username
    - client network address
    - servicename
    - lifetime
    - timestamp
- (lifetime of ticket)*

# Kerberos: simplified version

9

A asks for a ticket TicketB for B

1. A sends to C: A, B, N (N nonce)
2. C sends to A: TicketB,  $K_{AC}(K_{AB}, N, L, B)$   
L = "lifetime of ticket"
3. [A checks N and knows ticket lifetime]  
A sends to B: TicketB,  $K_{AB}(A, t_A)$  [authenticator]
4. [B checks that A's identity in TicketB and in authenticator are the same, time validity of ticket]
5. B sends to A:  $K_{AB}(t_A)$   
[in this way shows knowledge of  $t_A$ ]

TicketB =  $K_{BC}(K_{AB}, A, \text{"lifetime"}, \text{timestamp})$ , N nonce,  $K_{AB}$  session key; "lifetime" = validity of ticket;  $t_A$  timestamp

mainly contain session key and identity of Alice

- if some check fail, communication stop
- every time for using shared key between Alice and Server Alice should type the password, is a problem, we can't store the key,

# Session key and Ticket-granting Ticket (TGT)

- Messages between host and KDC should be protected using the master key (derived from user's password)
  - For each request to the KDC:
    - ✓ user must type the password uncomfortable each time, or
    - ✓ user's password is temporarily stored (to avoid the user the need of retyping)
- all above solutions are inadequate!

insecure automatic reboot of the server  
a human must be present for start  
the key service

new type of ticket : TGT ticket for  
obtain a ticket

# Session key and Ticket-granting Ticket (TGT)

Proposed solution to reduce # of times user types the password and/or master key

at initial login a session key  $S_A$  is derived for Alice by KDC

$S_A$  has a fixed lifetime (e.g., 1 day, 4 hours)

KDC gives Alice a TGT that includes session key  $S_A$  and other useful information to identify Alice (encrypted with KDC's master key)

a.y. 2022-23

kerberos

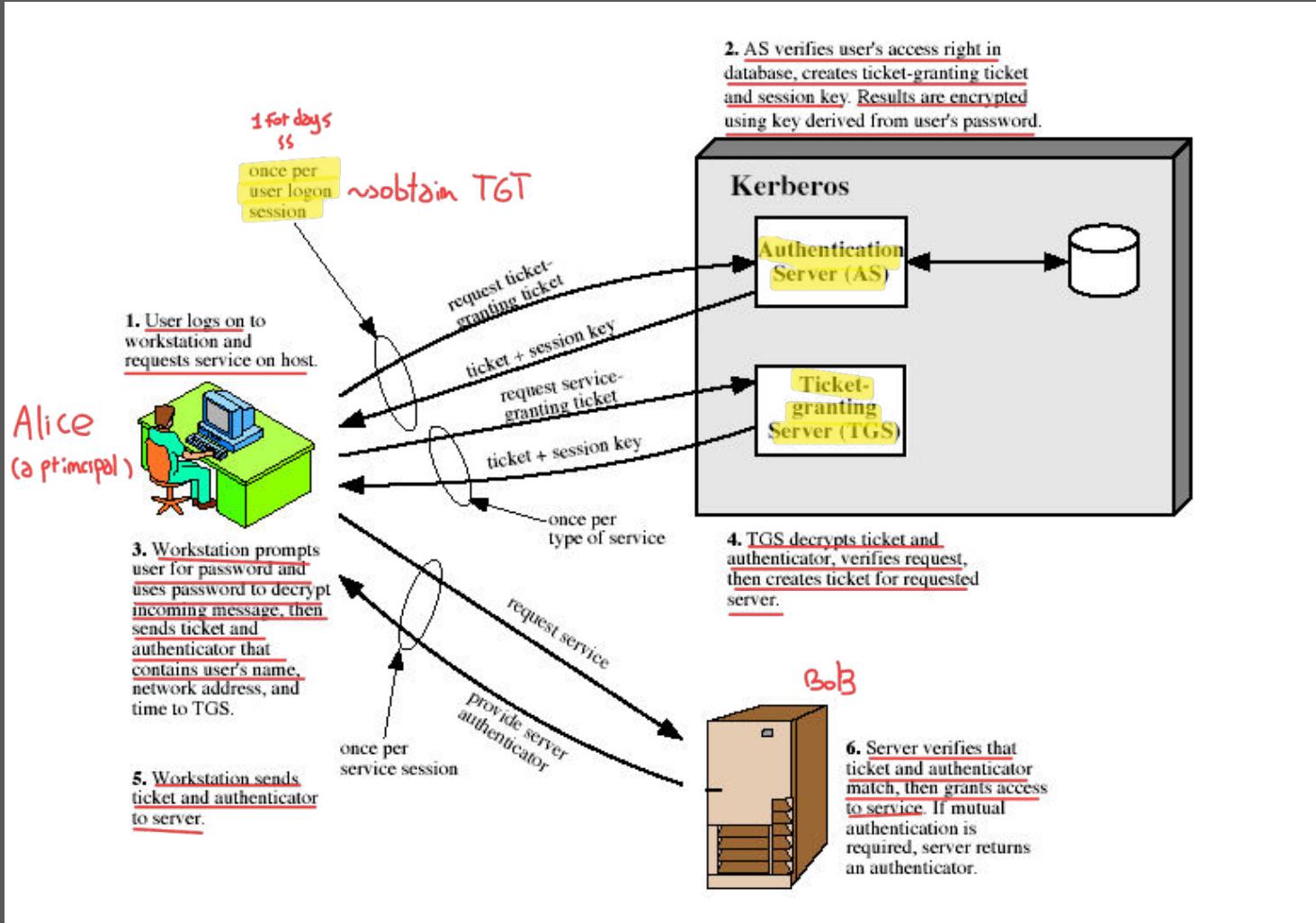
11

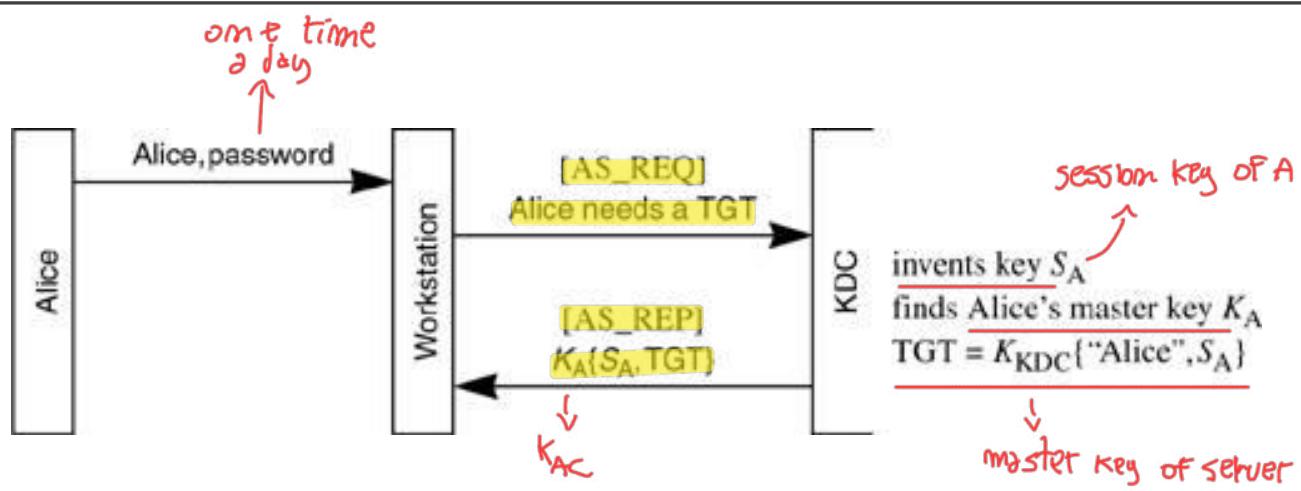
•  $S_A$  is a session key used only for Alice, typically have a long lifetime (~1 day). TGT is a ticket for asking another ticket, is given by C, using  $S_A$ .

# Session key and Ticket-granting Ticket (TGT)

- Subsequent requests from Alice to KDC use TGT in the initial message, instead type & password
- Subsequent tickets provided by KDC for accessing server V are decrypted using  $K_{VC}$
- User provides password only once
- No password is stored

↓  
resolves previous problems

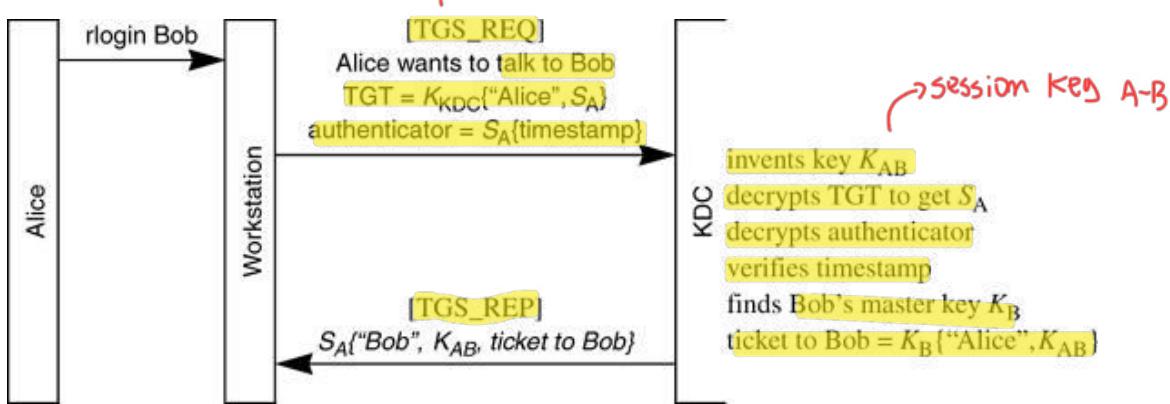




# Login

14

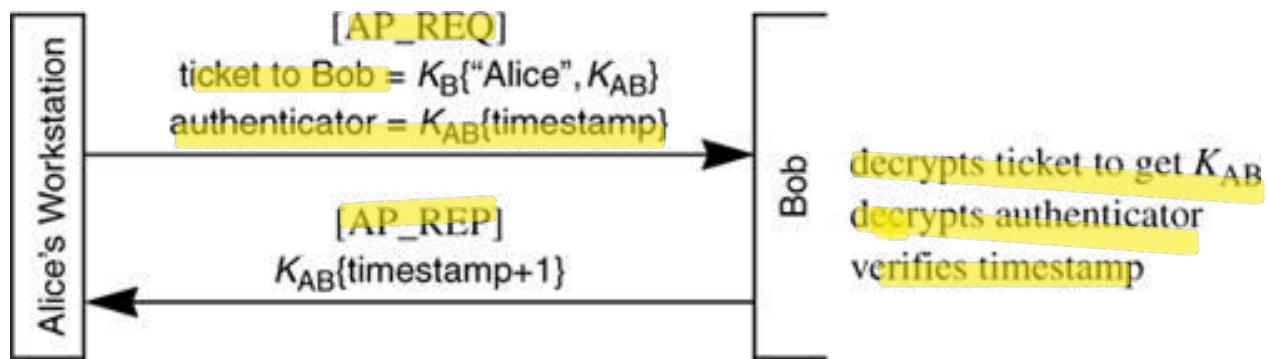
a.y. 2022-23



# Ticket request

15

a.y. 2022-23



# Use of ticket

16

a.y. 2022-23

# Authentication and time synchronization

- Authenticator:  $K_X\{timestamp\}$ 
  - $K_X$  is a session key
- Global Synchronous Clock is required
- Authenticator is used to avoid
  - replay of old messages sent to the same server by the adversary (old messages are eliminated)
  - replay to a server (when there are many servers)
  - Authenticator DOES NOT guarantee data integrity (a MAC is required)
- Vulnerability: many instances of same server all using same master key. Replay attack!
  - how could it be avoided?

*↳ all instances must synchronize*

# KDC and TGS

- KDC and TGS are similar (the same?) why do we need two different entity?
    - Historical reasons
    - One KDC can serve different systems (1 KDC many TGS)
  - multiple copies of KDC, sharing same KDC master key - availability and performance *make sense only when there are many principals*
  - Consistency issues in KDC databases
    - A single KDC stores information concerning principal (safer)
    - Periodically upload information to other KDC
- difficult to have quick synchronization* ↗

# Kerberos - Performance

- KDC stores only TGT and tickets
- Most work is on client
- KDC is involved only at login to provide TGT
- KDC uses only permanent information

# Message types

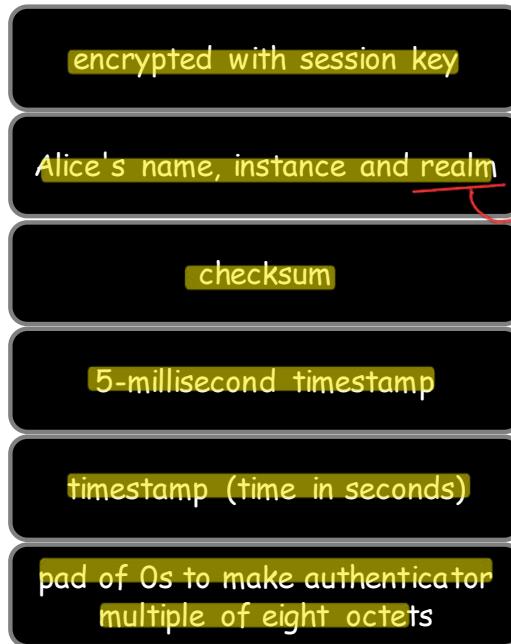
- AS\_REQ
  - Used when asking for the initial TGT.
- AS\_REPLY (also TGS REP)
  - Used to return a ticket, either a TGT or a ticket to some other principal.
- AP\_REQ (also TGS\_REQ)
  - Used to talk to another principal (or the TGS) using a ticket (or a TGT).
- AP\_REQ\_MUTUAL
  - This was intended to be used to talk to another principal and request mutual authentication. In fact, it is never used; instead, applications know whether mutual authentication is expected.
- AS\_ERR
  - Used for the KDC to report why it can't return a ticket or TGT in response to AS\_REQ or TGS\_REQ.
- PRIV
  - This is a message that carries encrypted integrity-protected application data.
- SAFE This
  - is a message that carries integrity-protected application data.
- AP\_ERR
  - Used by an application to report why authentication failed.

# Ticket (Alice, Bob)

encrypted with Bob's key

- Alice's name, instance and realm
- Alice's Network Layer address
- session key for Alice, Bob
- ticket lifetime, units of 5 minutes
- KDC's timestamp when ticket made
- Bob's name and instance
- pad of Os to make ticket length multiple of eight octets

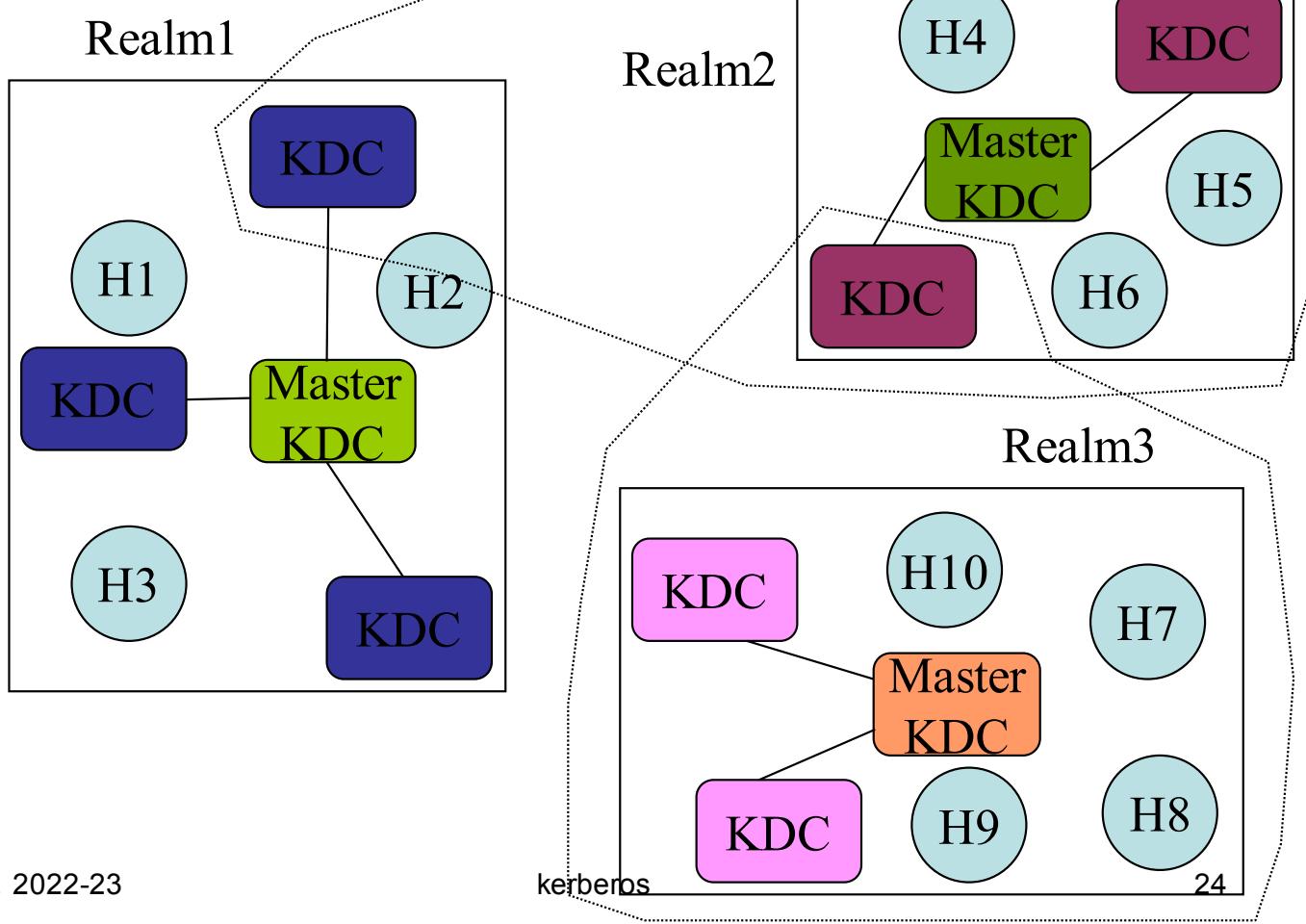
# Authenticator



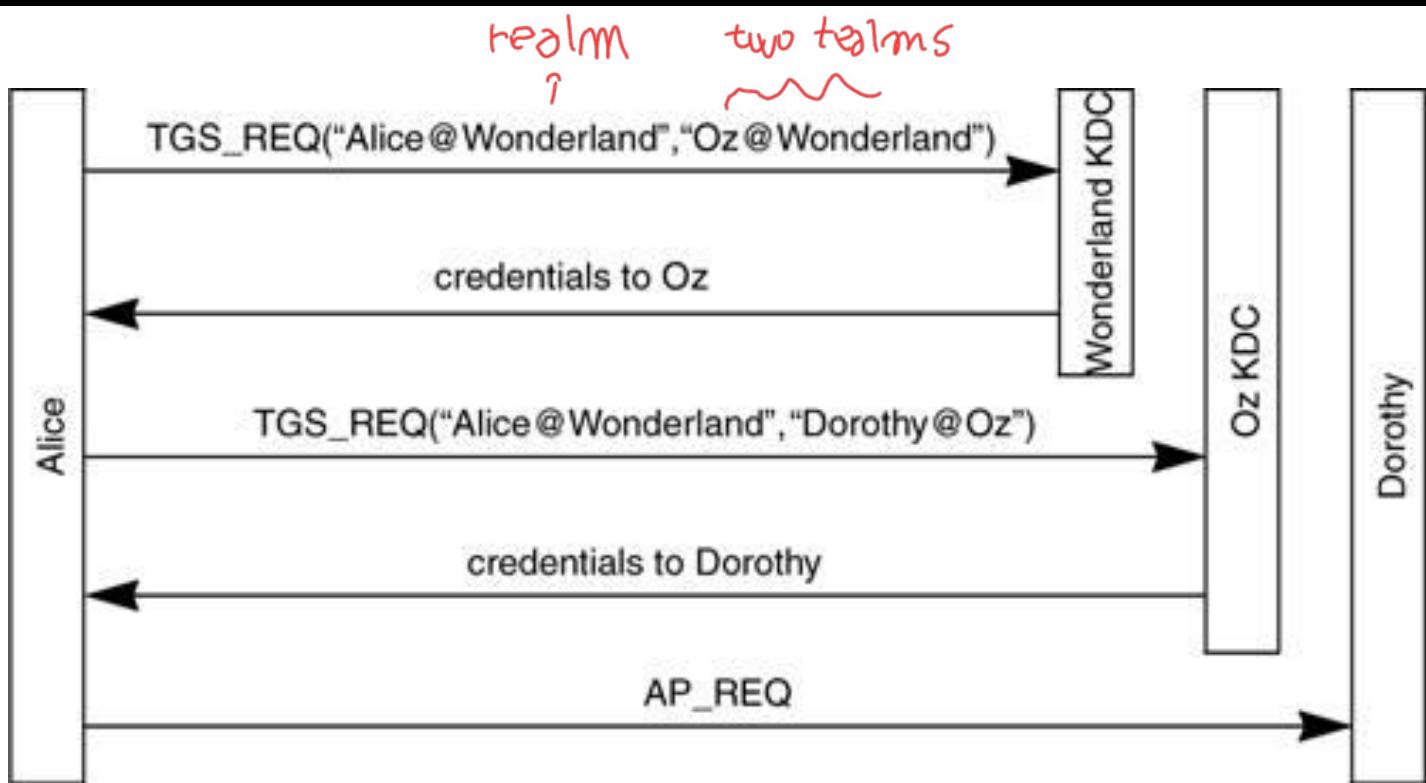
# Kerberos: Realms

- In very large systems security and performance issues suggest to use not only a domain but more (several many KDC)
- REALM
- each realm has a different master KDC
- all KDCs share the same KDC master key
- two KDCs in different realms have different databases of users

# Kerberos V. 4: Realms



# Authentication between realms



# Other features

a.y. 2022-23

## key version numbers

- for supporting changes of master keys

## encryption for privacy and integrity

- DES + Plaintext Cipher Block Chaining

kerberos

26

# Kerberos: version 5

- Same philosophy
- Major changes
- Integrity of messages, authentication using nonce (not only timestamps)
- Flexible encoding: many optional fields,
  - allows future extensions
  - overhead
- Major extensions to the functionality
- Delegation of rights: Alice allows Bob to access:
  - her resources for a specified amount of time
  - a specific subset of her resources
- Renewable tickets: tickets can be used for long time
- More encryption methods (Kerberos designed for DES)
- Hierarchy of realms

## **Authentication based on public keys**

**public keys & X.509**



# Authentication using public key

- **Idea:** use signed messages containing challenges or timestamps; signatures can be verified using public key
- **Problem:** it is important to guarantee knowledge of public key. In fact
  1. Alice wants to be authenticated by Bob;
  2. Let  $K_{PT}$  Trudy's public key
  3. If Trudy convinces B that the public key of A is  $K_{PT}$  then Trudy can be authenticated by Bob as Alice

**Solution:** Public Key Infrastructure: authority that guarantees correctness of public keys.

not perfect, can be attacked

# Authentication using public key

## Needham-Schroeder

7 steps protocol

**K<sub>PX</sub>:** public key of X, Sig\_C digital signature of C (trusted authority guarantees public keys)

### Mutual authentication

1. A to C: <this is A, want to talk to B>
2. C to A: <B, K<sub>PB</sub>, Sig\_C(K<sub>PB</sub>, B)>
3. A checks digital signature of C, generates nonce N and sends to B: K<sub>PB</sub>(N, A)
4. B decrypts (now wants to check A's identity) and sends to C: <B, A>
5. C to B: <A, K<sub>PA</sub>, Sig\_C(K<sub>PA</sub>, A)> *like 2. But with A*
6. B checks C's digital signature, retrieves K<sub>PA</sub>, generates nonce N' and sends to A: K<sub>PA</sub>(N, N')
7. A decrypts, checks N, and sends to B: K<sub>PB</sub>(N')

a.y. 2022-23

pubkey authentication

3

1. Alice → C, I want to talk with Bob

2. C reply to A by sending identity of Bob (B), Public key of Bob K<sub>PB</sub> and digital signature done by server Sig\_C Sig\_M (K<sub>PB</sub>, B)

Sig-C is a technically digital signature, is not valid legally because that's isn't a certification authority, is only given by the trusted server.

4-5. like 1-2 but for B → A

public key is used for checking signature, private key for doing the signature.

# Attack to N.-S. public key

Using reflection attack

- Trudy is a system user that can talk (being authenticated) to A, B & C
- Two interleaved excerpts of the protocol:  
R1: authentication between A and T;  
R2: authentication between T (like A) with B
- Man in the middle attack      ↪ pretend to be Alice
- T must be able to induce A to start an authentication session with T      → not unlikely T is a colleague of Alice
- Steps 1, 2, 4, 5 allow to obtain public keys
- Steps 3, 6, 7 perform authentication

reflection attack is based on doing two different session on a couple of user

# Attack to N.-S. public key

↑ more like a response.

only ask to C to get the public key

Steps 1, 2, 4 e 5 allow to know public keys

We focus on steps 3, 6, 7 of R1 and R2:

a) A-->T : step 3 of R1 sends  $K_{PT}(N, A)$

b) T(like A)-->B: step 3 of R2 sends  $K_{PB}(N, A)$  → T send  $K_{PB}(N, A)$  like he is A

c) B-->T(like A): step 6 of R2 sends  $K_{PA}(N', N)$

d) T-->A: step 6 of R1 sends  $K_{PA}(N', N)$

e) A-->T: step 7 of R1 sends  $K_{PT}(N')$

f) T(like A)-->B: step 7 of R2 sends  $K_{PB}(N')$

B thinks that he is talking to A by sharing secret nonces!

↳ T without know  $K_{PA}$  heach is goal, using reflection pubkey authentication

3-6-7 is focused on authentication.

# Authentication using public key Needham-Schroeder (fixed)

now the precedent attack is not possible

$K_{PX}$ : public key of X,  $Sig_C$  digital signature of C (trusted authority guarantees public keys)

Mutual authentication

1. A to C: <this is A, want to talk to B>
2. C to A: < $B, K_{PB}, Sig_C(K_{PB}, B)$ >
3. A checks digital signature of C, generates nonce N and sends to B:  $K_{PB}(N, A)$
4. B decrypt (now wants to check A's identity) and sends to C: < $B, A$ >
5. C to B: < $A, K_{PA}, Sig_C(K_{PA}, A)$ >
6. B checks C's digital signature, retrieves  $K_{PA}$ , generates nonce  $N'$  and sends to A:  $K_{PA}(B, N, N')$  → add identity of sender
7. A decrypts, checks N, and sends to B:  $K_{PB}(N')$

B, add identity of sender of the message → now T can not reply same message like before, because now A checks that sender is not T but B!

# Why the previous attack fails

We focus on steps 3,6,7 of R1 and R2:

- a) A-->T : step 3 of R1 sends  $K_{PT}(N, A)$
- b) T(like A)-->B: step 3 of R2 sends  $K_{PB}(N, A)$
- c) B-->T(like A): step 6 of R2 sends  $K_{PA}(B, N', N)$
- d) T-->A: EARLIER in step 6 of R1 T sends  $K_{PA}(N', N)$ ; NOW T  
**CANNOT** send  $K_{PA}(B, N', N)$  while is talking to A!!
- e) A-->T: step 7 of R1 sends  $K_{PT}(N')$
- f) T(like A)-->B: step 7 of R2 sends  $K_{PB}(N')$

a.y. 2022-23

pubkey authentication

impossible  
attack with multiple  
communications

# X.509 Authentication standard

X.509 standard for digital certification

- Part of standard known as CCITT X.500
- Defined in 1988 and several times revised (until 2000)
  - version 3
- We need directory of public keys signed by certification authority
- Define authentication protocols (see for instance [Stallings2005], or [https://www.itu.int/rec/dologin\\_pub.asp?lang=e&id=T-REC-X.509-201210-S!PDF-E&type=items](https://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-X.509-201210-S!PDF-E&type=items))
  - One-Way Authentication
  - Two-Way Authentication
  - Three-Way Authentication
- Public key cryptography and digital signatures
  - Algorithms are not part of the standard (why?)

pubkey authentication

↳ they are good for some years,  
attackers have augmented ability,  
algorithm must be changed,  
but not all protocol



we want a standard  
without change it in time,  
algorithms are not included  
because change quickly in time.

## X.509 (one-way) authentication

only one message exchanged between A and B. }

Timestamp  $t_A$

Session key  $K_{AB}$

B's public key  $P_B$

certA: certificate of A's public key, signed by certification authority

Authenticating Alice to Bob. B want check identity of A.

$A \rightarrow B : certA, D_A, Sig_A(D_A)$

$D_A = \langle t_A, B, P_B(K_{AB}) \rangle$

$\hookrightarrow t_A$  timestamp generate by A,  $K_{AB}$  session key generate by A

pubkey authentication

B after check digital signature of A ( $Sig_A$ )

# One-way authentication (discussion)

Single transfer of information from A to B and establishes the following:

1. Identity of A and that message was generated by A
2. That message was intended for B
3. Integrity and originality (not sent multiple times) of message
  - Message includes at least a timestamp  $t_A$  (a nonce could be included too) and identity of B and is signed with A's private key
    - Timestamp consists of (optional) generation time and expiration time. This prevents delayed delivery of messages.
    - Nonce can be used to detect replay attacks. Its value must be unique within the expiration time of the message. B can store nonce until message expires and reject any new messages with same nonce.
  - For authentication, message used simply to present credentials to B
  - Message may also include information, sgnData (not shown)
    - included within signature, guaranteeing authenticity and integrity.
  - Message may also be used to convey session key to B, encrypted with B's public key

## X.509 (two ways) mutual authentication

Mutual authentication:

- $A \rightarrow B$

$\text{cert}_A, D_A, \text{Sig}_A(D_A)$  [ $D_A = \langle t_A, N, B, P_B(k) \rangle$ ]  
(how does A know  $P_B$ ?)

- $B \rightarrow A$

$\text{cert}_B, D_B, \text{Sig}_B(D_B)$  [ $D_B = \langle t_B, N', A, N, P_A(k') \rangle$ ]  
(how does B know  $P_A$ ?)

$t_A, t_B$  = timestamps, to prevent delayed delivery of messages;  $k, k'$  session keys proposed by A and B; use of nonces avoids replay attacks

criticism: in  $D_A$  there is no identity of A - refer to the modified N.S. protocol

need  $P_B$  for start conversation,  
↑ is given before

is obtained by  
the certificate

# X.509 (three-ways) mutual authentication

typically used in bounded environment

Mutual authentication based on nonces, useful for unsynchronised clocks (0 denotes timestamp, optional)

three messages ( $A \rightarrow B, B \rightarrow A, A \rightarrow B$ ):

1.  $A \rightarrow B: \langle \text{cert}_A, D_A, \text{Sig}_A(D_A) \rangle [D_A = \langle 0, N, B, P_B(k) \rangle]$
2.  $B \rightarrow A: \langle \text{cert}_B, D_B, \text{Sig}_B(D_B) \rangle [D_B = \langle 0, N, A, N', P_A(k) \rangle]$
3.  $A \rightarrow B: \langle B, \text{Sig}_A(N, N', B) \rangle$

$\rightarrow$  is given before and only checked after by the certificate B  
 $\hookrightarrow$  given by CERTA  
 $\hookrightarrow$  can check now  $P_B$

Note: step 3 requires digital signature of nonces, making them tied (no replay attacks)

in 1. we can add also identity of A, for more security

pubkey authentication

timestamp optional, need synchronization between clocks

in 3. we add  $\text{Sig}_A$  to guarantee integrity of messages

# Challenge-response: ISO/IEC 9798-3 Mutual authentication (earlier version, bugged)

Another protocol

Why does the following protocol not work?

1. B to A:  $N_B$
2. A to B: certA,  $N_A$ ,  $N_B$ , B,  $Sig_A(N_A, N_B, B)$
3. B to A: certB,  $N_B'$ ,  $N_A$ , A,  $Sig_B(N_B', N_A, A)$  [not predictable by A]

A doesn't know  $N_B'$



"Canadian" attack (from *Protocols for Authentication and Key Establishment*, C. Boyd and A. Mathuria, Springer 2003, p. 112)

1. T(B) to A :  $N_T$
2. A to T(B): certA,  $N_A$ ,  $N_T$ , B,  $Sig_A(N_A, N_T, B)$   
1. T(A) to B:  $N_A$
2. B to T(A): certB,  $N_B$ ,  $N_A$ , A,  $Sig_B(N_B, N_A, A)$
3. T(B) to A: certB,  $N_B$ ,  $N_A$ , A,  $Sig_B(N_B, N_A, A)$ ; T is authenticated!!

A reflection attack

Note: use of  $N_B$  in step 3 (in place of  $N_B'$ ) has the same role as the use of Bob in step 6 of original N.-S. protocol

# PKI: Public Key Infrastructure

need for working with  
public key approach

↳ otherwise  
is easy to  
chat on the  
Pk.

- Certificates are issued by a trusted Certification Authority (CA)
- The CA provides certificates of all users in domain
- When someone wants to know the public key of some user he/she asks the CA
  - CA provides user's public key, signing it by its own private key
- This implies it is sufficient to know one only public key (CA's public key)

Notice:

- If CA is not trusted, its certificates are useless
- Keys are not used forever, they are subject to changes
- Length of key is related to security level

# X.509 Certificates

Certification authority CA guarantees public keys:

Signed Fields	Version
	Certificate serial number
	Signature Algorithm Object Identifier (OID)
	Issuer Distinguished Name (DN)
	Validity period
	Subject (user) Distinguished Name (DN)
	Subject public key information
	Public key Value
	Algorithm Obj. ID (OID)
	Issuer unique identifier (from version 2)
	Subject unique identifier (from version 2)
	Extensions (from version 3)
	Signature on the above fields

pubkey authentication

# X.509 certificate's fields 1

- **VERSION.** There are currently three versions defined, version 1 for which the code is 0, version 2 for which the code is 1, and version 3 for which the code is 2.  
*two different CA can use the same, but unique in the CA.*
- **SERIALNUMBER.** An integer that, together with the issuing CA's name, uniquely identifies this certificate.
- **SIGNATURE.** Specifies the algorithm used to compute the signature on this certificate. It consists of a subfield identifying the algorithm followed by optional parameters for the algorithm.
- **ISSUER.** The X.500 name of the issuing CA.

## X.500 name

- X.500 names look like C=US, O=company name, OU=research, CN=Alice, where C means country, O means organization, OU means organizational unit, and CN is common name.
- There are rules about what types of name components are allowed to be under what others.
  - The encoding uses OIDs (Object IDentifiers) for each of the name component
- There is no standard for displaying X.500 names and different applications display them differently.

## X.509 certificate's fields 2

- **VALIDITY**. This contains two subfields, the time the certificate becomes valid, and the last time for which it is valid.
- **SUBJECT**. The X.500 name of the entity whose key is being certified.
- **SUBJECTPUBLICKEYINFO**. This contains two subfields, an algorithm identifier, and the subject's public key.
- **ISSUERUNIQUEIDENTIFIER**. Optional (permitted only in version 2 and version 3, but deprecated). Uniquely identifies the issuer of this certificate.

# X.509 certificate's fields 3

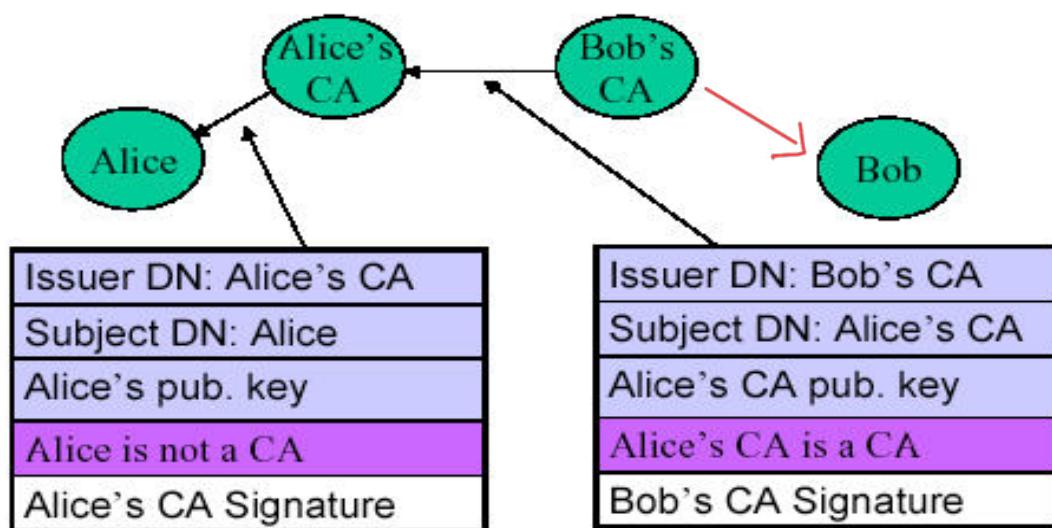
- **SUBJECTUNIQUEIDENTIFIER**. Optional (permitted only in version 2 and version 3, but deprecated). Uniquely identifies the subject of this certificate.
- **ALGORITHMIDENTIFIER**. This repeats the SIGNATURE field. Redundant!
- **EXTENSIONS**. These are only in X.509 version 3. X.509 allows arbitrary extensions, since they are defined by OID.
- **ENCRYPTED**. This field contains the signature on all but the last of the above fields.

## X.509 Certificates

- They can be easily accessed
- Certificates are modified by CA
- Certificates impossible to falsify (RSA > 2000 bit)
- If Alice and Bob share the same CA then they can know each other Public Key
- Otherwise CA form a hierarchy

# Hierarchy of CAs

How Bob gets Alice's certificate if they refer to different CAs?

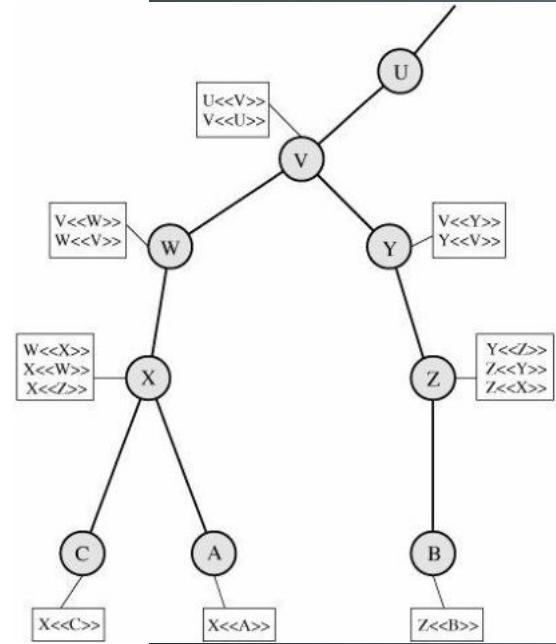


# Hierarchy of CAs

user A can acquire the following certificates from the directory to establish a certification path to B:

X<<W>> W<<V>> V<<Y>> Y<<Z>> Z<<B>>

where A<<B>> means "the certificate of user B has been issued by certification authority A"



# Certificate revocation

→ certificate naturally expired but can be revoked

Certificates are valid for a limited time (CAs want to be paid)

They can be revoked before the deadline:

1. User's secret key is not considered safe anymore
2. User is not certified by CA → For example user leave company
3. CA's secret key is compromised

**CRL: certificate revocation list**

- Must be used before accessing a user

# Certificate revocation

every week or more  
CA send a list of  
revoked certificate  
(not given in real time)

↳ today there is a better approach

Signed fields {	Version of CRL format			
	Signature Algorithm Object Identifier (OID)			
	CRL Issuer Distinguished Name (DN)			
	This update (date/time)			
	Next update (date/time) - optional			
	Subject (user) Distinguished Name (DN)			
	CRL Entry	Certificate Serial Number	Revocation Date	CRL entry extensions
	CRL Entry...	Serial...	Date...	extensions
	....			
	CRL Extensions			
	Signature on the above fields			

## CRL's fields

- **SIGNATURE**. Identical to the SIGNATURE field in certificates, this specifies the algorithm used to compute the signature on this CRL.
- **ISSUER**. Identical to the ISSUER field in certificates, this is the X.500 name of the issuing CA.

# CRL's fields

- **THISUPDATE**. This contains the time the CRL was issued.
- **NEXTUPDATE**. Optional. This contains the time the next CRL is expected to be issued. A reasonable policy is to treat as suspect any certificate issued by a CA whose current CRL has NEXTUPDATE time in the past.

# CRL's fields

- The following three fields repeat together, once for each revoked certificate:
  - **USERCERTIFICATE**. This contains the serial number of the revoked certificate.
  - **REVOCATIONDATE**. This contains the time the certificate was revoked.
  - **CRLENTRYEXTENSIONS**. This contains various optional information such as a reason code for why the certificate was revoked.

# CRL's fields

- **CRLEXTENSIONS**. This contains various optional information.
- **ALGORITHMIDENTIFIER**. As for certificates, this repeats the SIGNATURE field.
- **ENCRYPTED**. This field contains the signature on all but the last of the above fields.

# X.509 Version 3

- In version 3 certificates have much more information:
  - email/URL, possible limitations in the use of the certificate
- Instead of adding fields for every possible new information define extensions
- Extensions:
  - Which kind of extension
  - Specification about the extension

# OCSP today use this protocol

- Online Certificate Status Protocol used for obtaining the revocation status of an X.509 digital certificate (RFC 6960)
  - alternative to CRL, more agile
  - cert. status provided in TLS handshake (OCSP stapling: response on revocation check, signed by legit CA)
  - URL for check provided within the cert. (Certificate Extensions -> Authority Information Access, vers. 3 only)

↳ without see complete list of revocation  
↳ need online connection for this protocol

# PGP: Pretty Good Privacy

↳ open source is OPGP

use a public key and a private key

Trust model for E-mail certif. (Zimmerman)

↳ stored in a trustlist  
selected, choose by user

- There are no trusted CA
- Each user acts like a CA and decides for himself
- Certificates contain email addresses and public keys
- Certificates are signed by one or many users
- If you trust a sufficient number of the users signing a certificate, then you assume the certificate is good
- Each user keeps info on public keys of other users and signatures of these keys - together with trust value of the key

# **Password**

authentication through passwords

- Human beings
  - Short keys; possibly used to generate longer keys
  - Dictionary attack: adversary tries more common keys (easy with a large set of users)
  - Trojan horse
  - Countermeasures: slow login, close after several unsuccessful attempts
- Computers
  - Quality keys (long and not predictable)
  - Hidden: not stored in the clear (encrypted, one time passwords)

# Passwords

## Eavesdropping: adversary is sniffing

- password must not be sent in the clear
- Authentication should be different each time  
(to avoid replay attacks)

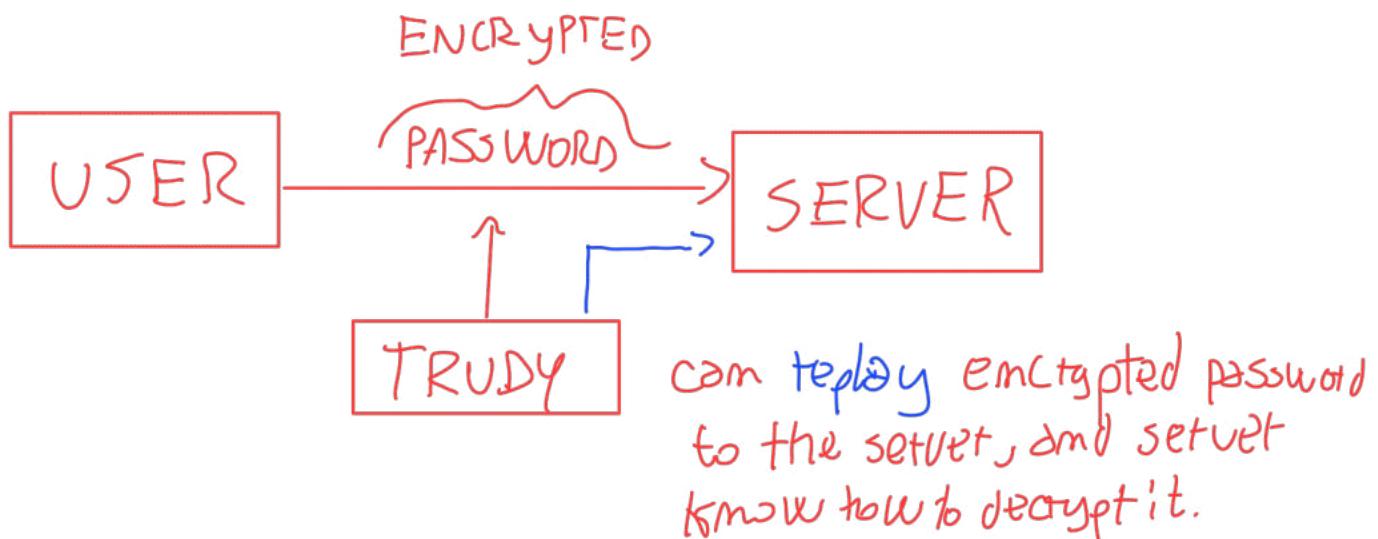
## Store password securely:

- Adversary can access database of passwords:  
encrypt passwords

# Password problems

a.y. 2022-23 passwd-authentication

3



obviously is difficult to realize for attacker because password is encrypted with session key that expire , and also is important to send Nounce and timestamps when send the password for protect user from Sniffing.

Idea: passwords are not stored: data obtained from passwords are stored (use hash)

- user password is first converted to a secret key K (56 bits, obtained by considering the 7-bit ASCII associated with each of the first 8 characters of password - then DES parity added) *(so that is not counted)*
- store  $\text{DES}_K(000\dots0)$ 
  - actually  $\text{DES}_K(\text{DES}_K(\text{DES}_K(\dots\text{DES}_K(000\dots0)\dots)))$  (25 times)
- DES' variant used for making fast DES hardware devices useless

## Unix password hash

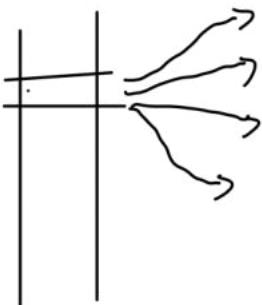
a.y. 2022-23 passwd-authentication

4

/etc/passwd → public for every user  
→ we have dictionary offline attack  
r w - t w - t --

username	salt	ciphertext	...
----------	------	------------	-----

is plaintext



Slowing down dictionary attack with salt.

**Problem:** dictionary attack (users keys are predictable)

- attacker reads password database and there is a high probability that there is at least one user with a weak password
- to increase security use salt (12 bit random number): modify DES through salt and encrypt 000... 0<salt>
- salt is per-user generated and can be stored in the clear
  - salt increases work for attacker because it makes impossible to hash a (guessed) password and check if it matches some user's password, but does not solve the problem of weak users' key

two user - two different attack

## Unix password hash

a.y. 2022-23 passwd-authentication

5

two basic requirements

1. Password should never sent as plaintext (eavesdropping) but always as (changing) ciphertext (play) → related to communication between client and server
2. Servers shouldn't store passwords as plaintext or ciphertext but it's better to store secrets derived by passwords through one way functions
  - in addition passwords should be salted to prevent rainbow tables and slowing down offline dictionary attacks.

Dictionary attack is a problem of human that only use password

Alice wants to authenticate herself to Bob

- send passwd in the clear - eavesdropper!
- do Diffie-Hellman exchange for establishing a secret key to be used for encrypting passwd - Trudy can impersonate Bob!
- use a challenge/response handshake - eavesdropper can carry out dictionary attack
- use strong password protocols

## Alice's authentication

a.y. 2022-23 passwd-authentication

6

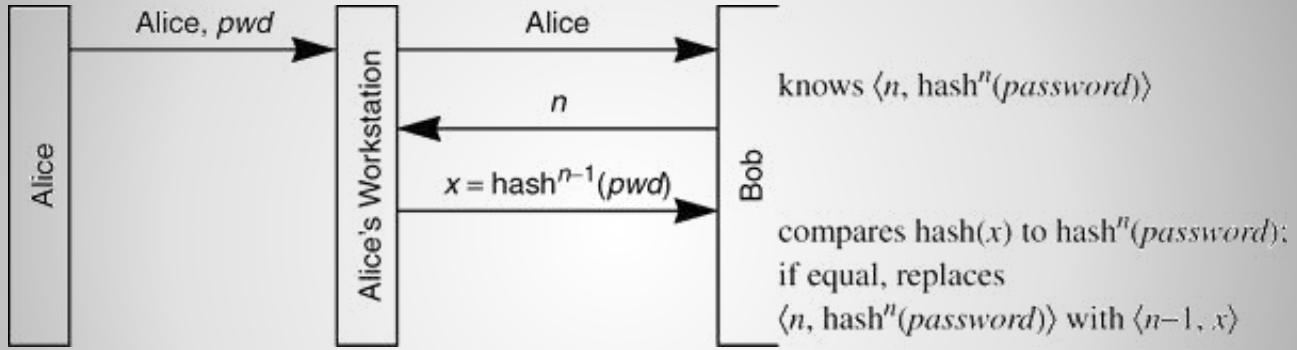
Alice is using a public workstation, can not store anything on it

## Goals:

- Obtaining the benefits of cryptographic authentication with the user being able to remember passwords only
- in particular:
  - no security information is kept at the user's machine (the machine is trusted but not configured)
  - someone impersonating either party will not be able to obtain information for off-line password guessing (online password guessing is not preventable)

## Strong password protocols

- Bob stores  $\langle \text{username}, n, \text{hash}^n(\text{password}) \rangle$ ,  $n$  is a relatively large number, like 1000
- Alice's workstation sends  $x = \text{hash}^{n-1}(\text{password})$
- Bob computes  $h(x)$ : if successful,  $n$  is decremented,  $x$  replaces  $\text{hash}^n(\text{password})$  in Bob's database



- why is sequence of hash transmissions reversed?
  - if you increment instead of decrementing it does NOT work
- safe against eavesdropping, database reading
- no authentication of Bob

OTP - one time password  
size of hpsws is small.

## Lamport's Hash [1981]

a.y. 2022-23 passwd-authentication

8

server store one line for each user:

Alice  $\rightarrow (n, \text{hash}^n \langle \text{password} \rangle)$   $h$  is cryptographical, is not invertible

$\hookrightarrow$  can authenticate only  $n$  times, we decrease. (PROBLEM)

- $h^{n-1}(pwd|salt)$  is used for authentication
- salt is stored at Bob's at setup time, Bob sends salt each time along with n  
 $\leftarrow \text{n, salt}$
- advantages
  - Alice can use the same password with multiple servers, why?
    - if servers use different salts hashes are different!
    - to ensure that the salts are different, servers name are also hashed in
  - easy password reset (when n reaches 1): just change the salt
  - defense against dictionary attacks
    - dictionary attack without the salt: compile  $h^k$  of all the words in the dictionary, for all k's from 1 to 1000; easier to check results in pwd db!

## Salting Lamport's Hash

- **small n attack** MITM → Trudy got  $h^{small\ n}$  (pswtsalt), → can do the remaining hashing and authenticate with Bob
  - when Alice tries to login Trudy impersonates Bob and sends  $n' < n$  and salt, when Trudy gets the reply, she can impersonate Alice until n is decremented to  $n'$
  - **defense:** Alice's workstation shows submitted n to Alice to verify the "approximate" range (Alice must remember it)
- **"human and paper" environment**
  - in case Alice workstation is not trusted or too "dumb" to do hashing
  - Alice is given a list of all hashes starting from 1000, she uses each hash exactly once
    - automatically prevents small n attack
    - string size? 64 bits (~10 characters) is secure enough
- implemented as S/Key and standardized as one-time password system (RFC 1938)

## Lamport's Hash: other properties

**Problem:** dictionary attack if weak keys (i.e., easily guessable) are chosen

### EKE

- Strong w.r.t. dictionary attack
- Mutual authentication
- Define session key

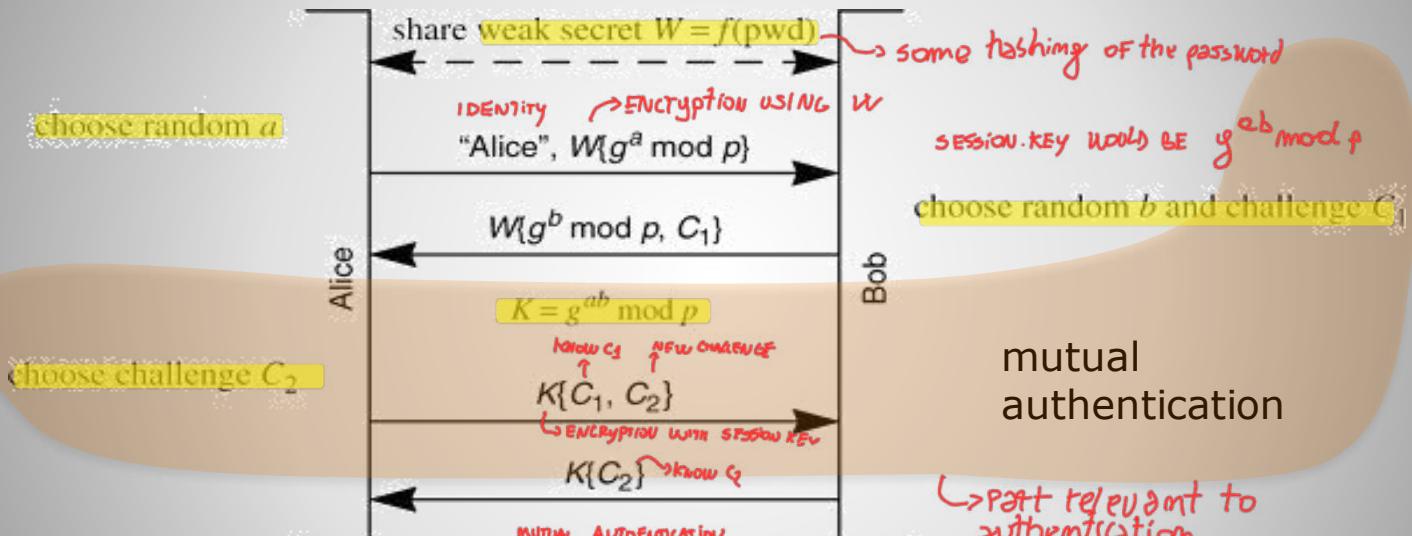
Scenario:

- User and server share a weak secret
- User and server use secret to authenticate and define a session key (Diffie-Hellman)

→ DH was vulnerable to MITM.

## Authentication EKE: Encrypted Key Exchange

pwd = Alice's password; Bob just knows  $W$  weak secret



## EKE basic authentication

EKE is strong w.r.t.

- replay attacks
  - a is changed every time
- dictionary attacks
  - even if the chosen password is weak the choice of random a does not allow the attacker to compute  $g^a$

authentication is strong because uses strong session key k

Note: if the attacker knows the password, then can act in place of A *weakness*

## **EKE: basic properties**

## **SPEKE** (Simple Password Exponential Key Exchange)

- uses  $W$  in place of  $g$  in D.-H. exchange
  - transmits  $W^a \text{ mod } p$  and  $W^b \text{ mod } p$ , session key is  $W^{ab} \text{ mod } p$

## **PDM** (Password Derived Moduli)

- chooses  $p$  depending upon password and uses  $g = 2$

## **EKE variants**

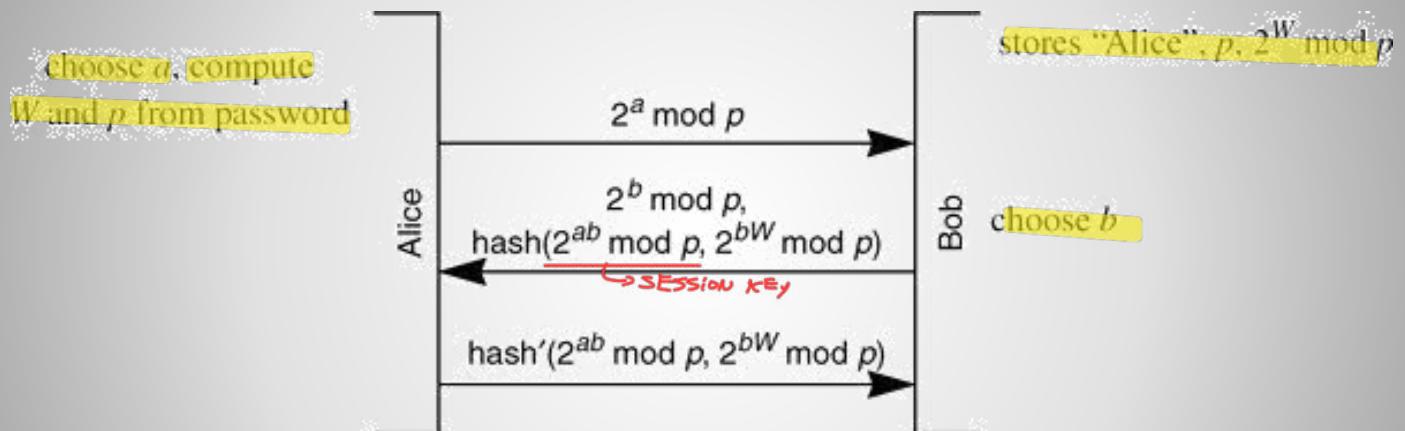
- if Trudy knew  $W$ , could impersonate Alice
- if passwd file stolen, it is possible to do a dictionary attack
  - if successful Trudy could impersonate the user
  - if unsuccessful, knowledge of  $W$  still allows to impersonate Alice
- basic EKE schemes (EKE, SPEKE, and PDM) can be modified to have an augmented property
- the idea is for Bob to store a quantity derived from the password that can be used to verify the password, but Alice's machine is required to know the password (not the derived quantity stored at the server)

↳ like this  $W$  is not enough for authentication, attacker  
will also passworts

## EKE weakness and defense

example for PDM

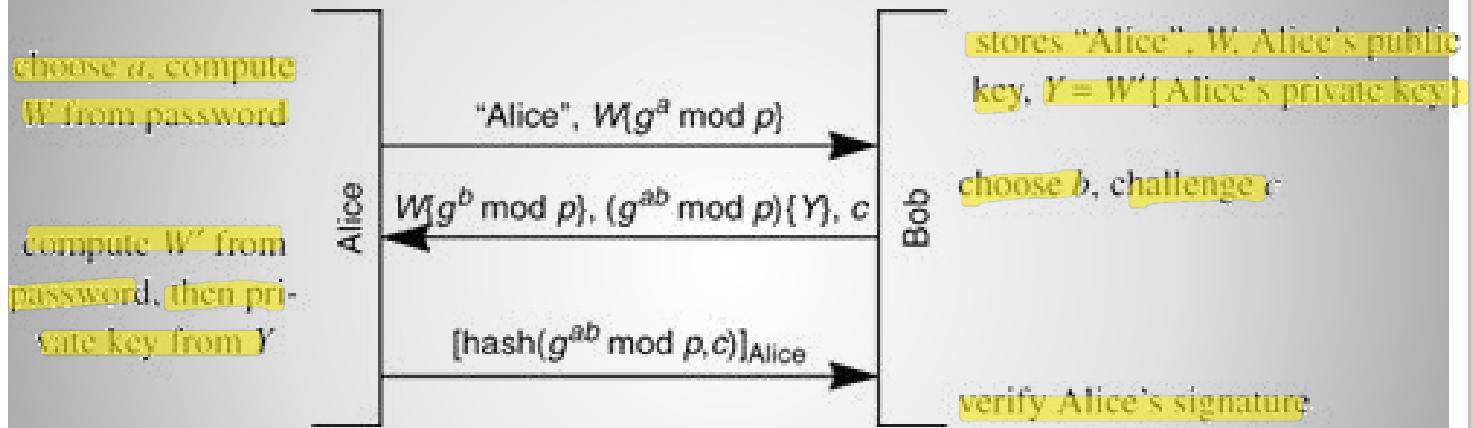
- server stores  $p$  and  $2^W \bmod p$ , where  $W$  is (still) a hash of user's password



## Augmented strong password protocols

- instead of requiring server to do an additional Diffie-Hellman exponentiation, it does an RSA verify operation, which is much less expensive
- this is accomplished by having Bob store, for Alice, an RSA private key encrypted with Alice's password, and the corresponding public key
- this can be done with any of the basic schemes (EKE, SPEKE, or PDM)

## augmentation at higher performance (server side)



## augmented EKE example

- Bob stores Y, which is Alice's private key encrypted with a function of her password
  - different hash of the password than W, or someone that stole the server database would be able to obtain her private key
- Bob also stores Alice's RSA public key corresponding to the encrypted private key
- in message 1, Alice sends the usual first EKE message, consisting of her Diffie-Hellman value encrypted with W
- in message 2, Bob sends his Diffie-Hellman value, along with Y (Alice's encrypted private key), encrypted with the agreed-upon Diffie-Hellman key
- Alice extracts Y by decrypting with  $g^{ab} \bmod p$ , and then decrypts Y with her password to obtain her private key
- in message 3, Alice signs a hash of the Diffie-Hellman key and the challenge c, and Bob verifies her signature using the stored public key
  - this achieves mutual authentication as well as the augmented property

# discussion

a.y. 2022-23 passwd-authentication

19

## EXAM QUESTIONS

### DATA INTEGRITY

→ CAN'T BE USED FOR A CHECK WITH THIRD PARTY

- DEFINE CONCEPT OF DATA INTEGRITY. (WEAKER FORM OF DATA AUTHENTICATION)
- DIFFERENCE WITH DATA AUTHENTICITY (STRONGER FORM PROVIDING PROOF OF IDENTITY OF ORIGINATOR OF THE DOCUMENT)
- CASE WHERE THE INTEGRITY REQUIREMENT IS NOT REQUESTED
  - ↳ NOT EXIST IS ALWAYS IMPORTANT.
- SHOW THAT USING CBC TECHNIQUE FOR PROVIDING A MESSAGE AUTHENTICATION CODE IS FAILING FOR VARIABLE LENGTHS MESSAGES. → SHOW THE ATTACK

### DIGITAL SIGN : Alice is sending to Bob a digitally signed file

- HOW DOES Bob VERIFY THE SIGNATURE. → RSA VERIFICATION OR ALSO DSA
- WHAT CHECKS ARE NECESSARY ON THE DIGITAL CERTIFICATE → WE HAVE TO CHECK THAT THE CERTIFICATE IS NOT BEEN REVOKED USING OSP PROTOCOL. MAIN CHECK IS NO REVOCATION CERTIFICATE, OTHER IS 'WAS THE CERTIFICATE IS BEEN USED WITHIN THE VALIDITY INTERVAL'.
- IS THE DIGITAL SIGNATURE STILL VALID AFTER THE EXPIRATION OF ALICE'S CERTIFICATE
- DIGITAL SIGNATURE WAS GOOD WHEN PUT AND CHECKED, AFTER THE EXPIRATION IS IMPORTANT TO CHECK IF IT WAS VALID WHEN USED, STILL GOOD IN THIS CASE

### HASHING FUNCTIONS

- WHAT IS A PASSWORD BASED KEY DERIVATION FUNCTION?
- WHAT IS A 'RAINBOW TABLE'?
- GIVEN A HASH FUNCTION, IS THE PROPERTY OF BEING CRYPTOGRAPHICALLY RETAINED FOREVER?
  - ↳ A CRYPTOGRAPHICAL HASHING FUNCTION SHOULD SATISFY THAT THE LENGTH OF FINGERPRINT IS LONG ENOUGH IN RESPECT TO THE BOUND OF BIRTHDAY ATTACK, THAT INCREASE IN TIME, ANSWER IS NO LENGTH OF HASHING FUNCTION COULD BECOME TOO SMALL.
- CASE WHERE  $h(m||k)$  IS INSECURE

### SYMMETRIC ENCRYPTION

### FIREWALL



# Cryptography and Network Security

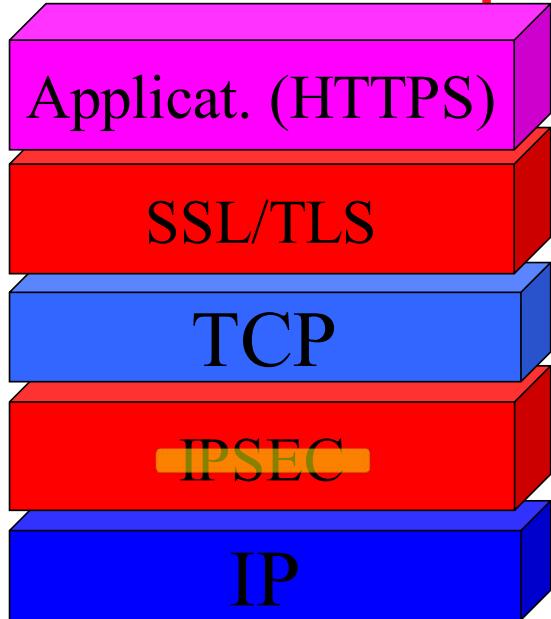
---

## IPSEC

↳ security protocol

Finish talk about basic foundations!

# Security architecture and protocol stack



a.a.2022-23

- Secure applications: PGP, HTTPS, S-HTTP, SFTP, ...
  - or
  - Security down in the protocol stack
  - SSL between TCP and application layer
  - IPSEC between TCP and IP
- ?

ipsec

2

- we can use both IPSEC and TLS, but is not strictly necessary.
- IPSEC if used of not a router must receive a datagram, IPSEC is independent of routing system
- IPSEC is invisible to application, users don't reconfigure application for use IPSEC, same not the for TLS, that is upper TCP

# Why not security on datagrams?

IPSEC is independent by the routing system

Protect IP packets at each hop (there is a shared key among two routers that are connected by a link)

Good: all traffic is encrypted (including IP headers)

Bad:

Cooperation among router is required

Significant computational effort (when a router receives a packet decodes it, then encodes it for next hop)

# IP Security

- there exist several application specific security mechanisms
  - e.g., S/MIME, PGP, Kerberos, SSL/HTTPS
- however, there are security concerns that cut across protocol layers
- it is important to have a security protocol that can be used by all applications
- IP security: security between IP and TCP

# IPSec

- IP Security mechanism provides
  - authentication
  - confidentiality
  - key management

→ Provide ~~data~~ authenticity
- applicable to use over LANs, across public & private WANs & for the Internet
- Very complicated & articulated specification (many docs...)
- was formerly mandatory, then (2011) optional, in IPv6
  - optional in IPv4

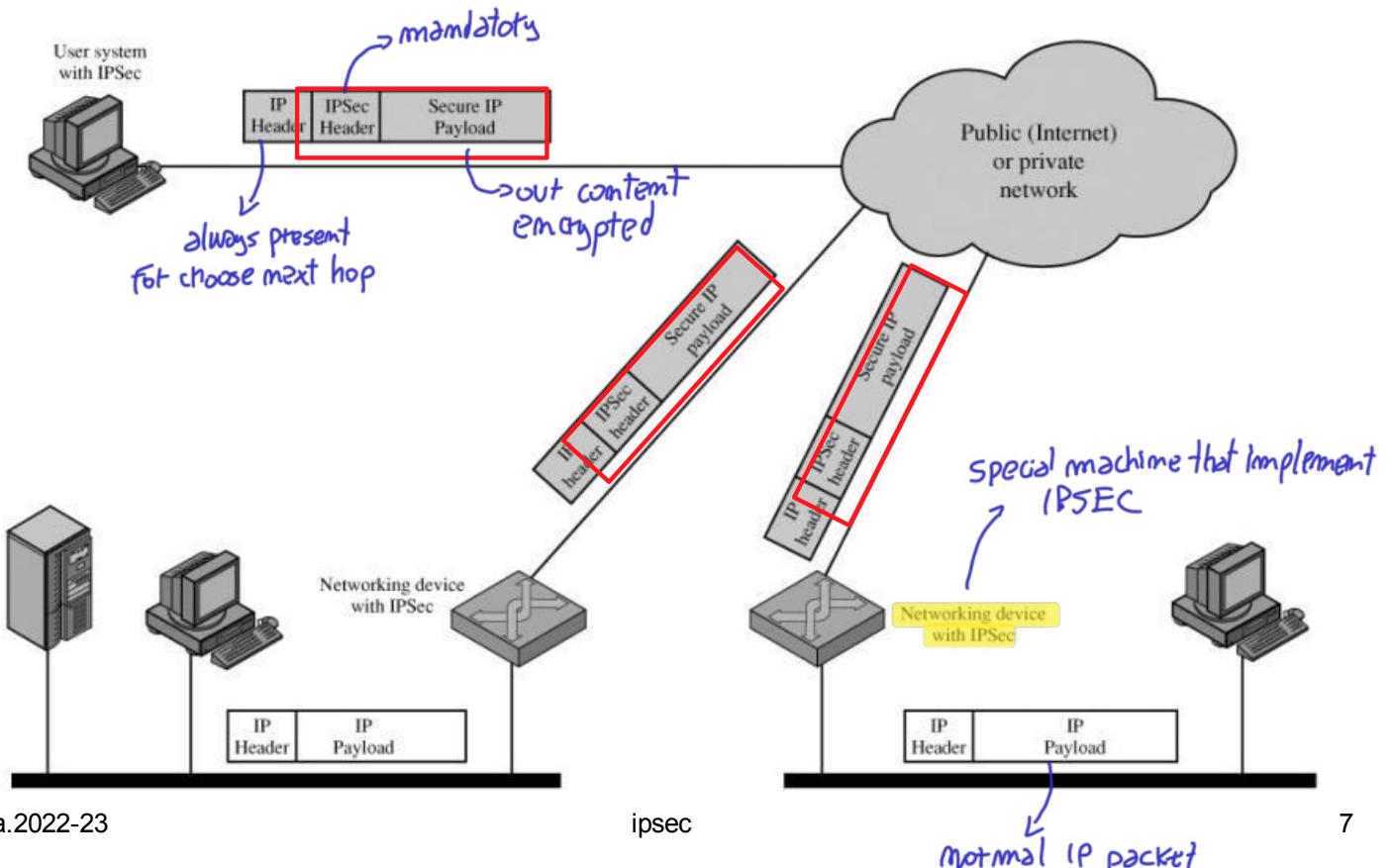
↳ now IPSEC is  
not obligatory

5

a.a.2022-23

touter inspect header and decide next hop,  
always use the IP header, but will be also an IPSEC part in packet  
some machine intermediate will process packet according to IPSEC,  
that have some characteristic like more computationally power

Alice and Bob process the packets according to IPSEC,  
but not all touter elaborate the packet according to IPSEC  
only some intermediator.



a.a.2022-23

7

in LAN we send normal packet without IPSEC while in internet we implement IPSEC.

↳ logical lan

# Benefits of IPSec



a firewall/router provides strong security to all traffic crossing the perimeter



is resistant to bypass



is below transport layer, hence transparent to applications



can be transparent to end users (allows to realize Virtual Private Networks)



can provide security for individual users if desired

a.a.2022-23

ipsec

8

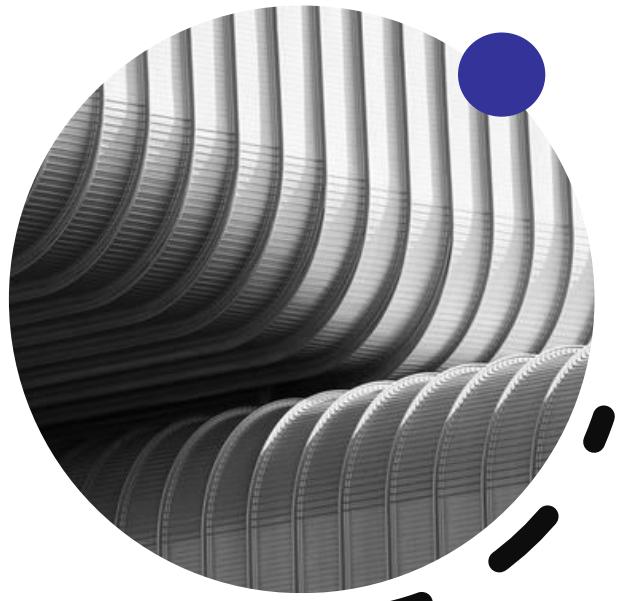
VPN SERVER is a commercial organization offering a connection using some secure protocol, you connect to this VPN SERVER and other service that offer a VPN SERVER is to be a proxy, your connection send packet from your home that go through this protected connection, like a tunnel, and VPN will be deliver the packet to correct destination. Receiver don't see you as sender!

two level IPSEC if you want that extract data is protected, when packet arrive to gateway one level is removed, but one level is there.

# Practical applications of IPsec



a.a.2022-23



10

- A typical attack to a company is to attack other company working with main company that are less secure. A way to prevent this is ask to the smaller company to implement IPSEC
- INTRANET is a private internet working using the same protocol of internet, EXTRANET is the part of company network that is show outside people, exposed on internet.

# security features

- implemented as extension headers that follow the main IP header

- Authentication Header (AH) is the extension header for authentication
- Encapsulating Security Payload (ESP) is the extension header for encryption

→ first sub protocol

↳ second sub protocol

12

a.a.2022-23

IPSEC USE some sub protocol, three main protocol are:  
one for authentication, one for confidentiality and one  
for key exchange and maintain.  
All this protocol are important

- AH protocol that provide authentication
- ESP protocol that provide confidentiality

two variant : one that provide  
encryption and one that  
provide also authentication  
most → Home

# services provided by AH and ESP protocols

16

two variant of ESP

	AH	ESP (encryption only)	ESP (encryption plus authentication)
Access control	✓	✓	✓
Connectionless integrity	✓		✓
Data origin authentication	✓		✓
Rejection of replayed packets	✓	✓	✓
Confidentiality		✓	✓
Limited traffic flow confidentiality		✓	✓

For integrity must consider TCP, that provide ipsec handshake before communication

a.a.2022-23

- For ESP, two cases:  
with and without the authentication option
- Both AH and ESP are vehicles for access control, based on the distribution of cryptographic keys and the management of traffic flows relative to these security protocols

↳ most secure way is use AH with encryption  
ESP, using these configuration achieve more security

Traffic flow attack: attacker based on header of IP packet can understand type of traffic of a user, if you want to prevent must need more security on header of IP.

# Security Associations

17

- A security association (SA) is a one-way relationship between sender & receiver that affords security for traffic flow
  - logical group of security parameters, that ease the sharing of information to another entity
- There is a database of Security Associations (SADB)
  - according to RFC 2401, each interface for which IPsec is enabled requires nominally separate inbound vs. outbound databases, because of the directionality
- SA identified by 3 main parameters:
  - Security Parameters Index (SPI)
  - IP Destination Address
  - Security Protocol Identifier (AH or ESP)

ipsec

a.a.2022-23

SA : association that allow security on the traffic between sender and receiver, two parties must share a key , need additional parameters

Firewall communication is bidirectional , like https need to specify type of security it characteristic for outgoing packets but also incoming one.

# SA, continued 1

18

ipsec

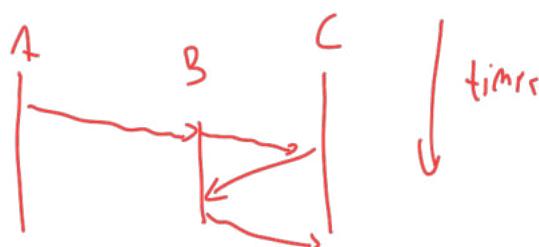
Consider just 3 participants:

Alice	100	→ salary does not know by others
Bob	200	
Charlie	300	
	3	=

generate 3 random numbers: Alice 50 80 130 → 100  
 Bob 100 0 100 → 200  
 Charlie 250 110 160 → 300

one number is kept and other two distributed:  
 (at random)

design 3 lines:



A	B	C
50	130	-80
0	400	100
110	250	250

infringement  
Kept by Alice

$$\begin{aligned}
 \text{sum: } & -60 \quad 400 \quad 290 \\
 & \downarrow + \downarrow \quad \downarrow + \downarrow \\
 \frac{-60 + 400 + 290}{3} & = \frac{630}{3} = \\
 & = 230
 \end{aligned}$$

# SA, continued

## 2

19

ipsec

- For multicast, SA is provided for the group, and is duplicated across all authorized receivers of the group.
- There may be more than one SA for a group, using different SPIs, thereby allowing multiple levels and sets of security within a group.
- Note that the relevant standard does not describe how the association is chosen and duplicated across the group; it is assumed that a responsible party will have made the choice.

a.a.2022-23

# SA's parameters

20

ipsec

## Other information

- **Sequence Number Counter**
  - 32-bit value used to generate the Sequence Number field in AH or ESP headers
- **Sequence Counter Overflow**  $\leadsto$  higher possibility of number of SN
  - flag indicating whether overflow of the Sequence Number Counter should generate an auditable event and prevent further transmission of packets on this SA
- **Anti-Replay Window**  $\leadsto$  for prevent replay attack
  - used to determine whether an inbound AH or ESP packet is a replay

a.a.2022-23

# SA's parameters

21

- **AH Information**
  - Authentication algorithm, keys, key lifetimes, and related parameters being used with AH
- **ESP Information**
  - Encryption and authentication algorithm, keys, initialization values, key lifetimes, and related parameters being used with ESP

# SA's parameters

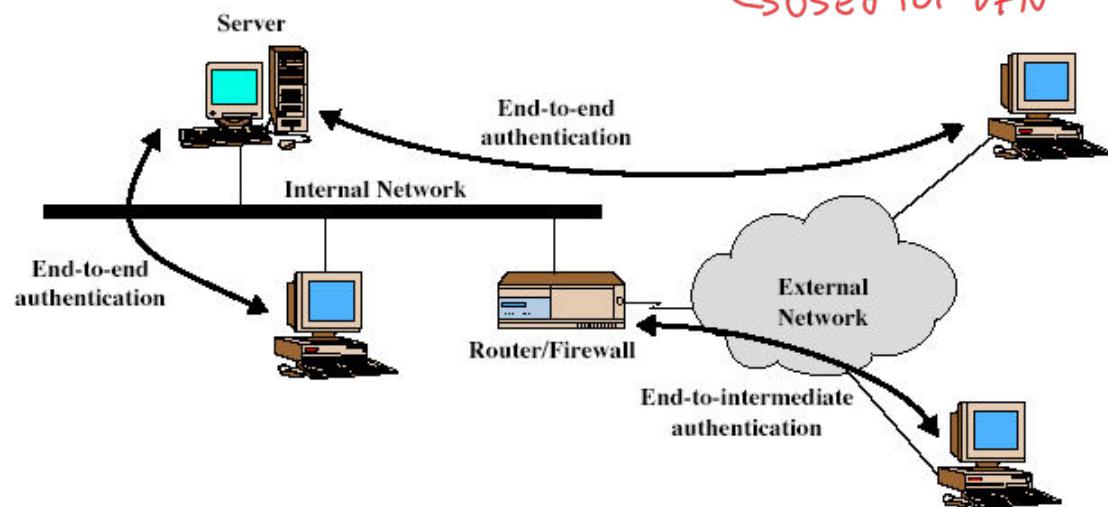
22

- Lifetime of This Security Association
  - A time interval or byte count after which an SA must be replaced with a new SA (and new SPI) or terminated, plus an indication of which of these actions should occur
- IPSec Protocol Mode
  - Tunnel, transport, or wildcard *important*
- Path MTU
  - Any observed path maximum transmission unit (**maximum size of a packet that can be transmitted without fragmentation**) and aging variables

# Transport & Tunnel Modes

↳ more secure, but heavier

↳ used for VPN



a.a.2022-23

ipsec

26

**Fragmentation:** problem of packet that is divided in many packet that are smaller, upon receiving of all you can reconstruct original packet.

**Fragmentation attack:** by changing meta information of a fragment about of number of fragment of order is possible to make OS crash  
↳ you ask to OS to fragment of information and after fake information

# Transport mode summary

- Transport mode: original IP header not touched; IPsec information added between IP header and packet body
  - IP header | IPsec | [ packet ]  
  protected → by the protocol
  - Most logical when IPsec used end-to-end

IPSEC produce new datagram provide the security parameter configured

## Transport mode

28

- Used for host-to-host communications
- Only payload (the data you transfer) of IP packet is encrypted and/or authenticated
- Routing is intact, since the IP header is neither modified nor encrypted
  - however, when the authentication header is used, the IP addresses cannot be translated (NAT), as this will invalidate the hash value

# Tunnel mode summary

- Tunnel mode: keep original IP packet intact but protect it; add new header information outside
  - New IP header | IPsec | [ old IP header | packet ]
    - encrypted
    - authenticated
  - Can be used when IPSec is applied at intermediate point along path (e.g., for firewall-to-firewall traffic)
    - Treat the link as a secure tunnel
  - Results in slightly longer packet

a.a.2022-23

ipsec

30

- In tunnel mode we change original header with a newer one
- used usually when are connecting two different LAN or in VPN
- we achieve protection of all datagram but very demanding and also longer packet. Best protection.

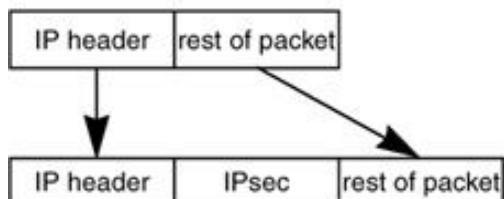
# Tunnel mode

31

- The entire IP packet (data and IP header) is encrypted and/or authenticated. It is then encapsulated into a new IP packet with a new IP header.
- Tunnel mode is used to create Virtual Private Networks (VPN) for network-to-network communications (e.g., between routers to link sites), host-to-network communications (e.g., remote user access), and host-to-host communications (e.g., private chat)

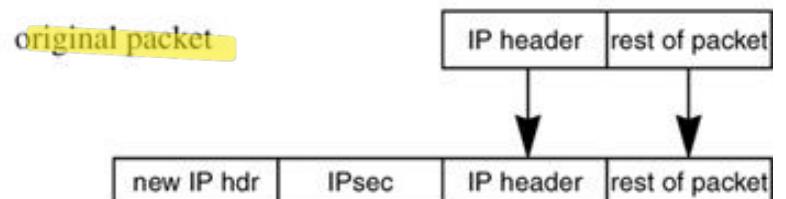
# Transport & tunnel modes

Transport Mode



a.a.2022-23

Tunnel Mode



ipsec

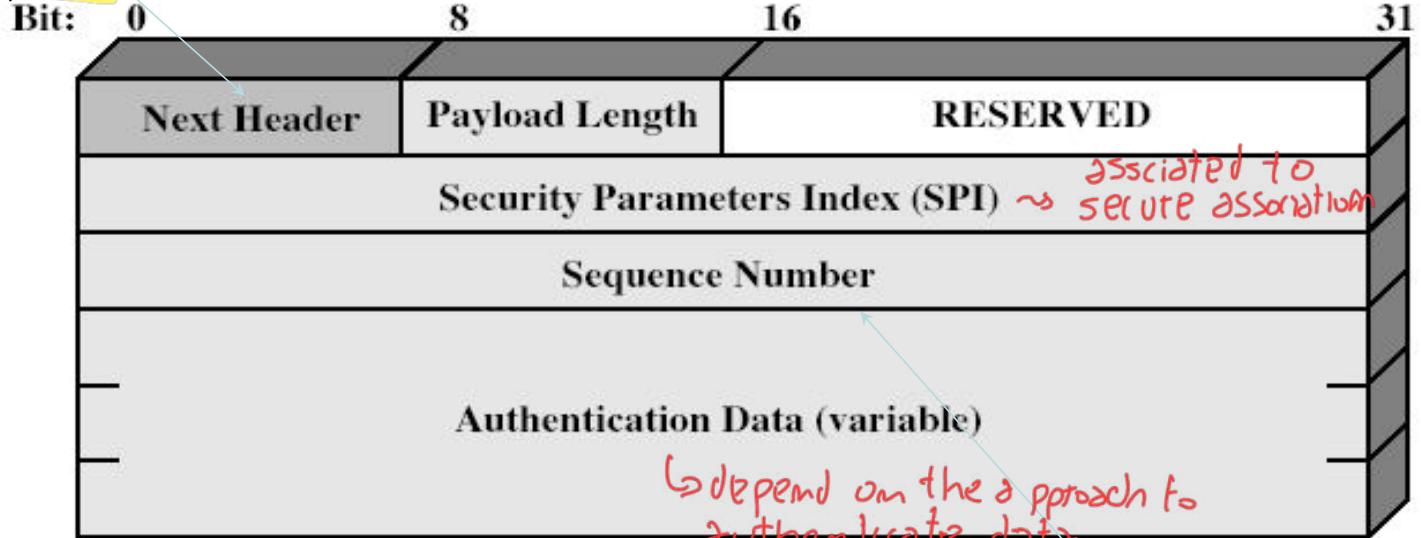
32

*new IP Header*  
Payload is consist hete  
OF old IP header and old PL  
OF PACKET

# Authentication Header

higher level protocol,

e.g., TCP



a.a.2022-23

ipsec

not restarting

35

always use key hashing function

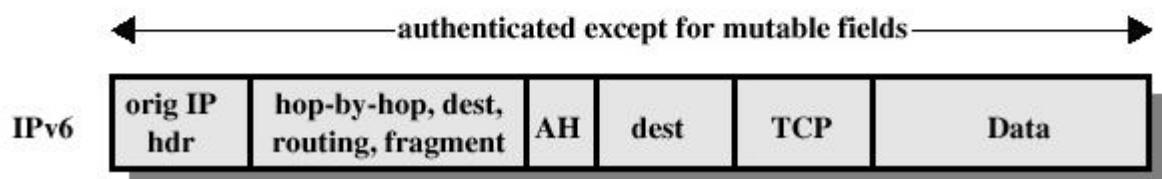
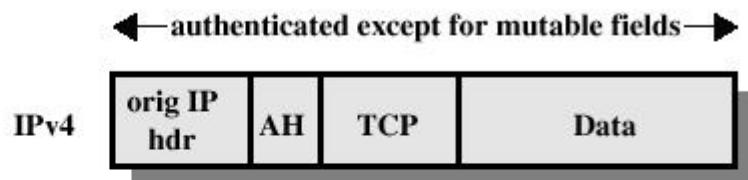
# AH protocol

36

- AH protects the IP payload and all header fields of an IP datagram except for mutable fields
  - In IPv4, mutable (and therefore unauthenticated) IP header fields include TOS, Flags, Fragment Offset, TTL and Header Checksum.
- AH operates directly on top of IP, using IP protocol number 51

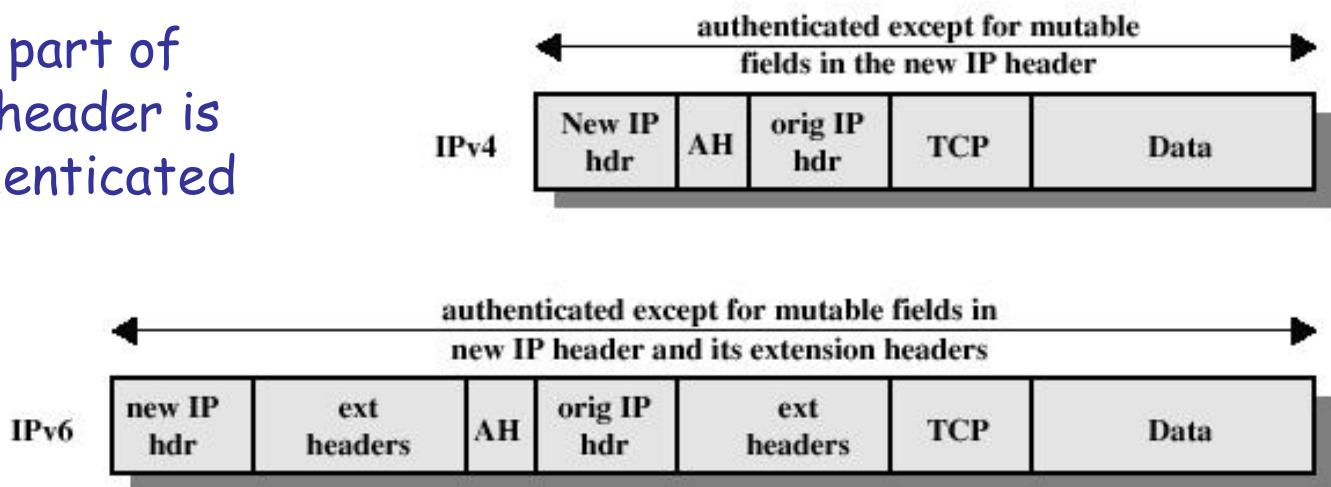
# Authentication Header (AH): transport mode

only part of  
the header is  
authenticated



# Authentication Header (AH): tunnel mode

only part of  
the header is  
authenticated



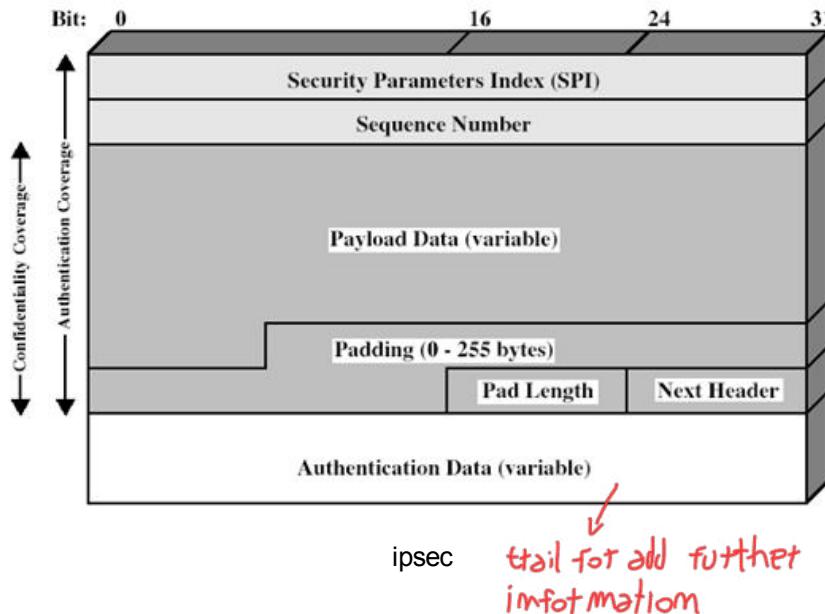
# Encapsulating Security Payload (ESP)

39

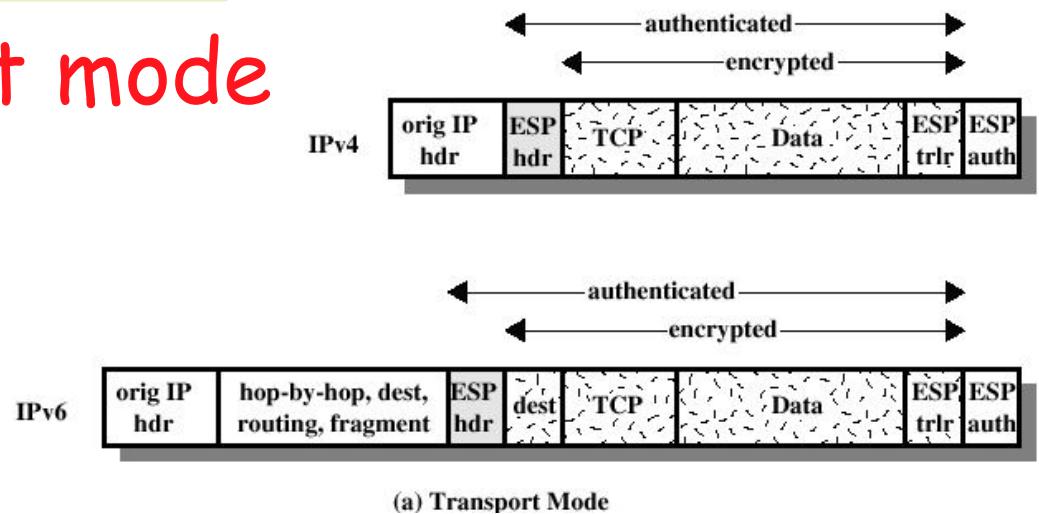
- provides message content confidentiality & limited traffic flow confidentiality
- can optionally provide the same authentication services as AH
- supports range of ciphers, modes, padding
  - AES, DES, Triple-DES, Blowfish etc
  - CBC most common
  - padding to meet blocksize of the packet
  - HMAC (same as AH)

# Encapsulating Security Payload

ESP

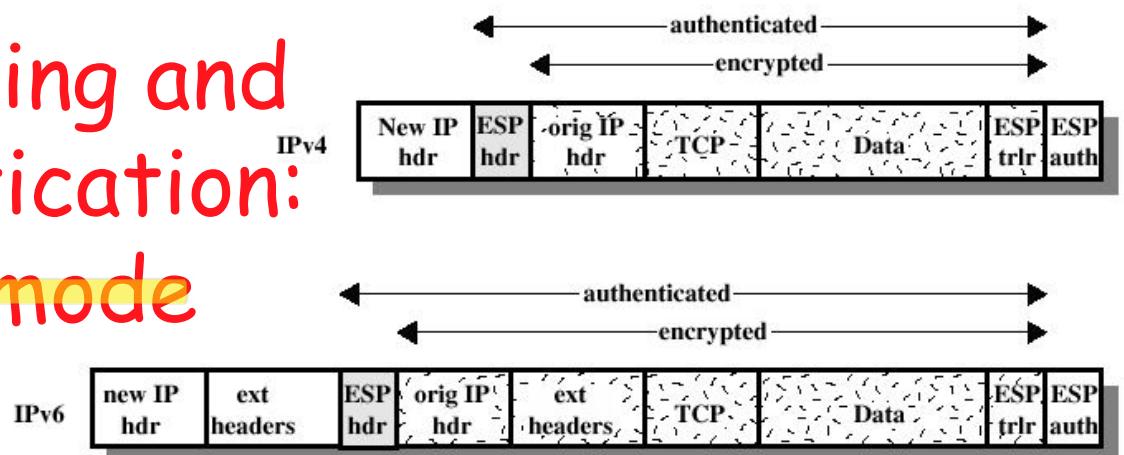


# ESP - encoding and authentication: Transport mode



difference between ESP with authentication and AH is:  
AH is stronger, is authenticated also the part of header that  
is not changed, with ESP header is ignored.

# ESP - encrypting and authentication: Tunnel mode



(b) Tunnel Mode

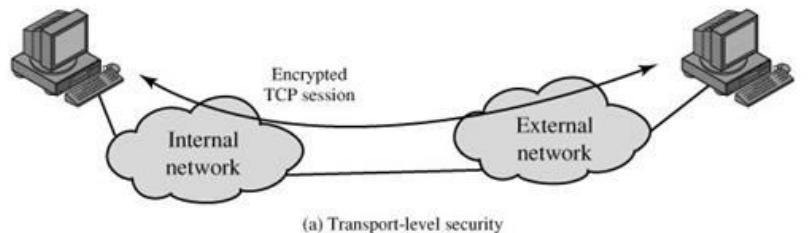


## Transport vs Tunnel Mode ESP

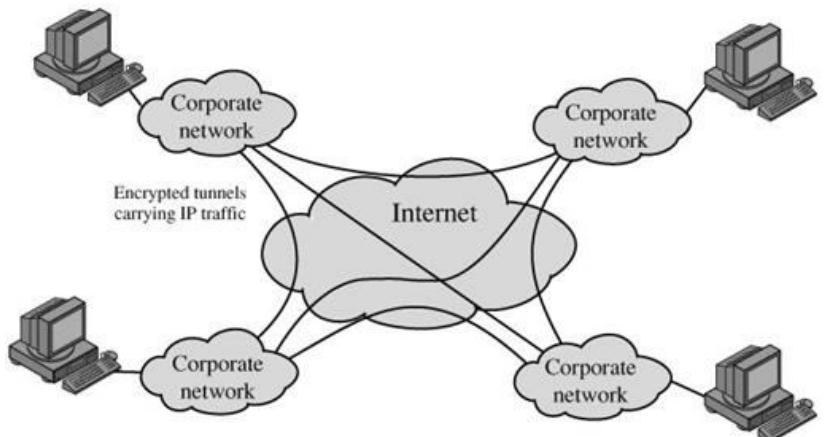
- transport mode is used to encrypt & optionally authenticate IP data
  - data protected but header left in clear
  - adversary can try traffic analysis
  - good for host-to-host traffic
- tunnel mode encrypts entire IP packet
  - add new header for next hop
  - slow
  - good for VPNs (Virtual Private Networks, gateway to gateway security)

# Transport vs Tunnel Mode ESP

ipsec



(a) Transport-level security



(b) A virtual private network via tunnel mode

# What about authentication first?

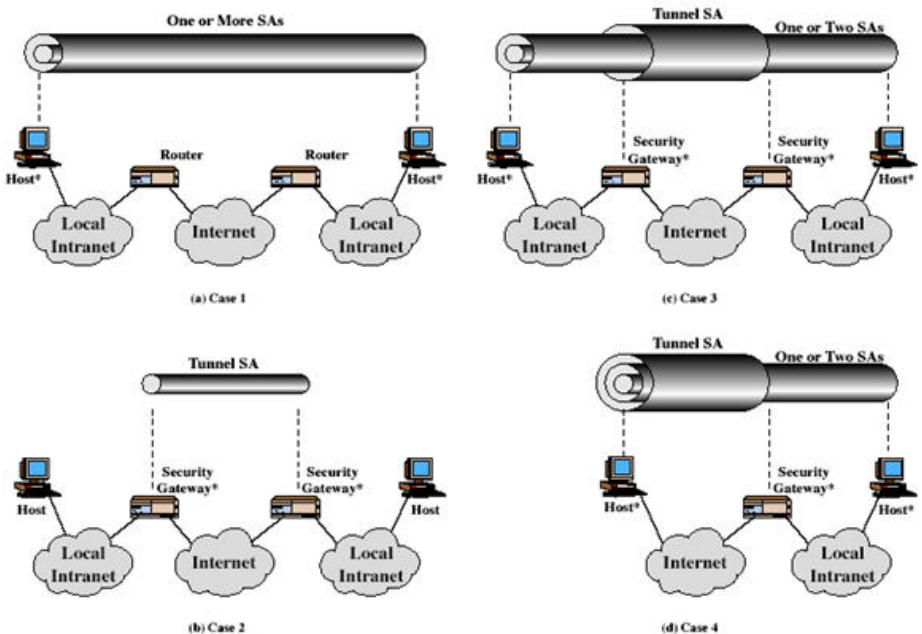
authentication prior to encryption might be preferable for two reasons

- since authentication data are protected by encryption, it is impossible to alter authentication data without decryption
- if message is stored as plaintext, then verifying authentication data requires re-encryption

50

# Combining Security Associations

support of IPsec is at least provide support for these four cases:

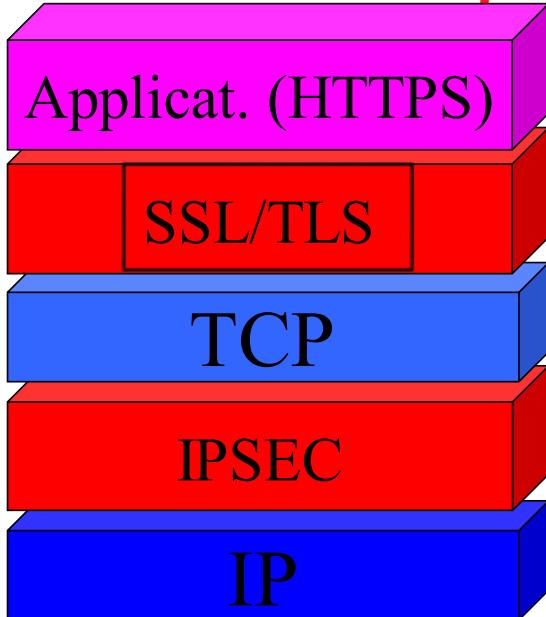


↳ add more level  
of security LAN-to-LAN

↳ another level  
if only one host  
is in company

- 1) Host-to-Host security association
- 2) LAN-to-LAN tunnel security association because it's transmit information over internet

# Security architecture and protocol stack



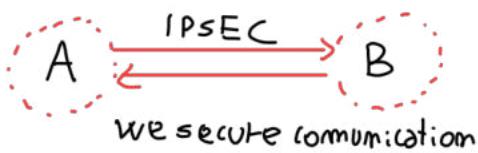
a.y. 2022-23

TSL

2

- Secure applications: PGP, HTTPS, S-HTTP, SFTP, ...
- or
- Security down in the protocol stack
  - SSL between TCP and application layer
  - IPSEC between TCP and IP
- ?

We use an open source version OpenSSL.

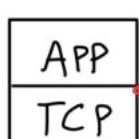


most popular configuration of IPSEC is the one where all packets sent by Alice to Bob and all the replies are captured and secured by IPSEC.

These mean that we can have several applications talking between A and B (File server, web server, ...) all these application are impacted these choice that make secure the communication. Application is not relevant in IPSEC, obviously there are more advanced configuration.

↳ IPSEC SECURE HOST TO HOST Communication

In TLS is different:



APP ask services to TCP, like deliver packets to a port.  
in TCP we elaborate number of ports.

We have a number of ports 0-65535, from 0 to 1023 are STANDARD, and other ports 1024-65535 are client applications.

When a client open a connection with some server, is choosing a number of a port, server listen on a standard port number 22, 25 - MAIL SMTP  
21 - FTP  
23 - TELNET  
...

# SSL/TLS intro

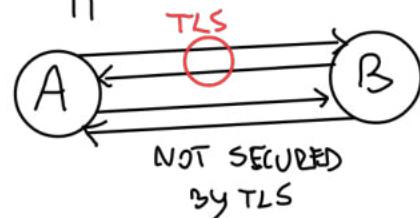
- Transport Layer Security (TLS) and its predecessor, Secure Sockets Layer (SSL), are cryptographic protocols that provide security for communications over networks
- TLS and SSL encrypt the segments of network connections at the Transport Layer end-to-end

a.y. 2022-23

TSL

3

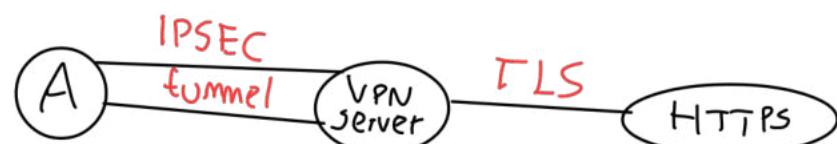
In case of TLS, the applications not see directly TCP but TLS now, whatever should be asked to TCP now it should be asked to TLS: TLS provide an END TO END SECURE CONNECTION that is selective with application



We have some conversation SECURED by TLS, but on other parts we can have not secured conversation.

And we case where for the same conversation we use both IPSEC and TLS: like VPN

↳ secured by IPSEC in tunnel mode the connection between A and VPN, if A visit a website the VPN use TLS for HTTPS. Packets exiting from A are processed with both.



# SSL/TLS intro, continued

## 2

- SSL/TLS allow client/server applications to communicate across a network in a way designed to prevent eavesdropping, tampering and message forgery
- provide endpoint authentication and communications confidentiality over the Internet using cryptography
  - RSA security with 1024- and 2048-bit strengths
- current version TLS 1.3 (RFC 8446 by Mozilla)
  - SSL 1, 2, 3 broken
  - TLS 1.0 having several weakness
  - TLS 1.1 fair
  - TLS 1.2 still good

↳ best choice TLS 1.3

- When client connect to a server in previous handshake, server choose the most recent version supported to client, but is not a good choice back word compatibility.
- Data integrity is always requested, TLS providing confidentiality and data integrity.

TLS cover

## main threats addressed

sniff network and  
get messages

- **eavesdropping** = the act of secretly listening to private conversation
- **tampering** = the act of altering something secretly or improperly
- **message forgery** = sending of a message to deceive the recipient as to whom the real sender is

TAMPERING ; is changing the content without authorization

FORGERY : we create a message

# SSL/TLS intro, continued

3

- In typical end-user/browser usage, TLS authentication is unilateral: only server is authenticated, but not vice versa
- TLS also supports mutual authentication
  - if partners diligently scrutinize identity information

→ Asking credentials to client, TLS is possible used for client authentication

↳ not very used, mutual authentication

extended certification = EV

↓ validation

# SSL/TLS intro, continued

4

- Mutual authentication requires that the TLS client-side also holds a certificate (which is not usually the case in the end-user/browser scenario)
  - Unless TLS-PSK, the Secure Remote Password (SRP) protocol, or some other protocol is used that can provide strong mutual authentication in the absence of certificates



# SSL/TLS intro, continued 5

---

TLS involves three basic phases:

1. Peer negotiation for algorithm support
2. Key exchange and authentication
3. Symmetric cipher encryption and message authentication

# SSL/TLS intro, continued

## 6

a.y. 2022-23

During first phase, client and server negotiate cipher suites, which determine ciphers to be used, key exchange and authentication algorithms, as well as message authentication codes (MACs)

- key exchange and authentication algorithms are typically public key algorithms, or, as in TLS-PSK, pre-shared keys (PSKs) could be used
- message authentication codes are made up from cryptographic hash functions using the HMAC construction for TLS, and a non-standard pseudorandom function for SSL.

TSL

TLS - PSK

↳ preshared key  
for do authentication  
with HMAC

10

# SSL/TLS: typical algorithms

- **For key exchange:** RSA, Diffie-Hellman, ECDH (Elliptic Curve Diffie-Hellman), SRP (Secure Remote Password protocol), PSK
- **For authentication:** RSA, DSA, ECDSA (Elliptic Curve Digital Signature Algorithm)
- **Symmetric ciphers:** RC4, Triple DES, AES, IDEA, DES, or Camellia. In older versions of SSL, RC2 was also used.
- **For cryptographic hash function:** HMAC-MD5 or HMAC-SHA are used for TLS, MD5 and SHA for SSL, while older versions of SSL also used MD2 and MD4.

## SLL/TLS and digital certificates

The key information and certificates necessary for TLS are handled in the form of X.509 certificates, which define required fields and data formats.

optionally client send the the digital certificate.

stateful : memory of the past

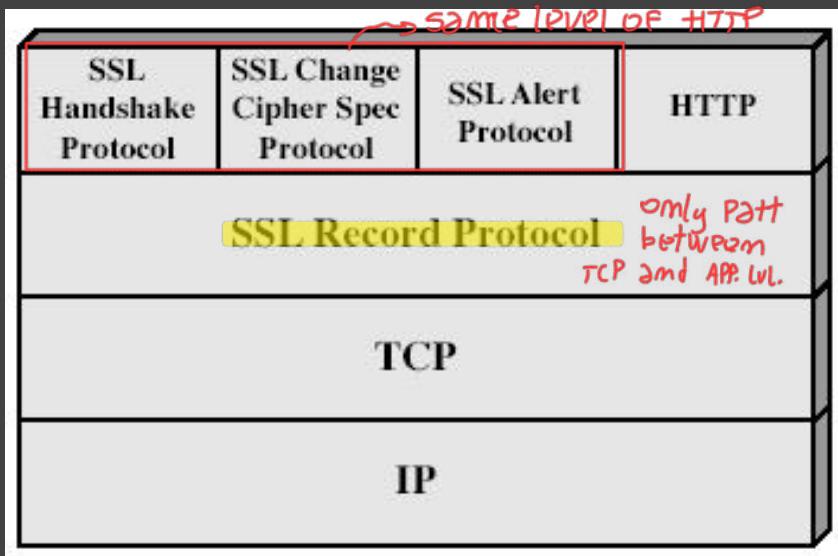
# how SSL/TLS works 1/5

1. Client and server negotiate a stateful connection by using a handshaking procedure. During handshake, client and server agree on various parameters used to establish connection's security
2. Handshake begins when client connects to TLS-enabled server requesting a secure connection and presents a list of supported ciphers and hash functions
3. From this list, server picks the strongest cipher and hash function that it also supports and notifies client of the decision

# how SSL/TLS works 2/5

4. Server sends back its identification in the form of a digital certificate X.509
5. Client may contact the CA and confirm that the certificate is authentic and not revoked before proceeding
  - modern browsers support Extended Validation certificates and can use the Online Certificate Status Protocol (OCSP)

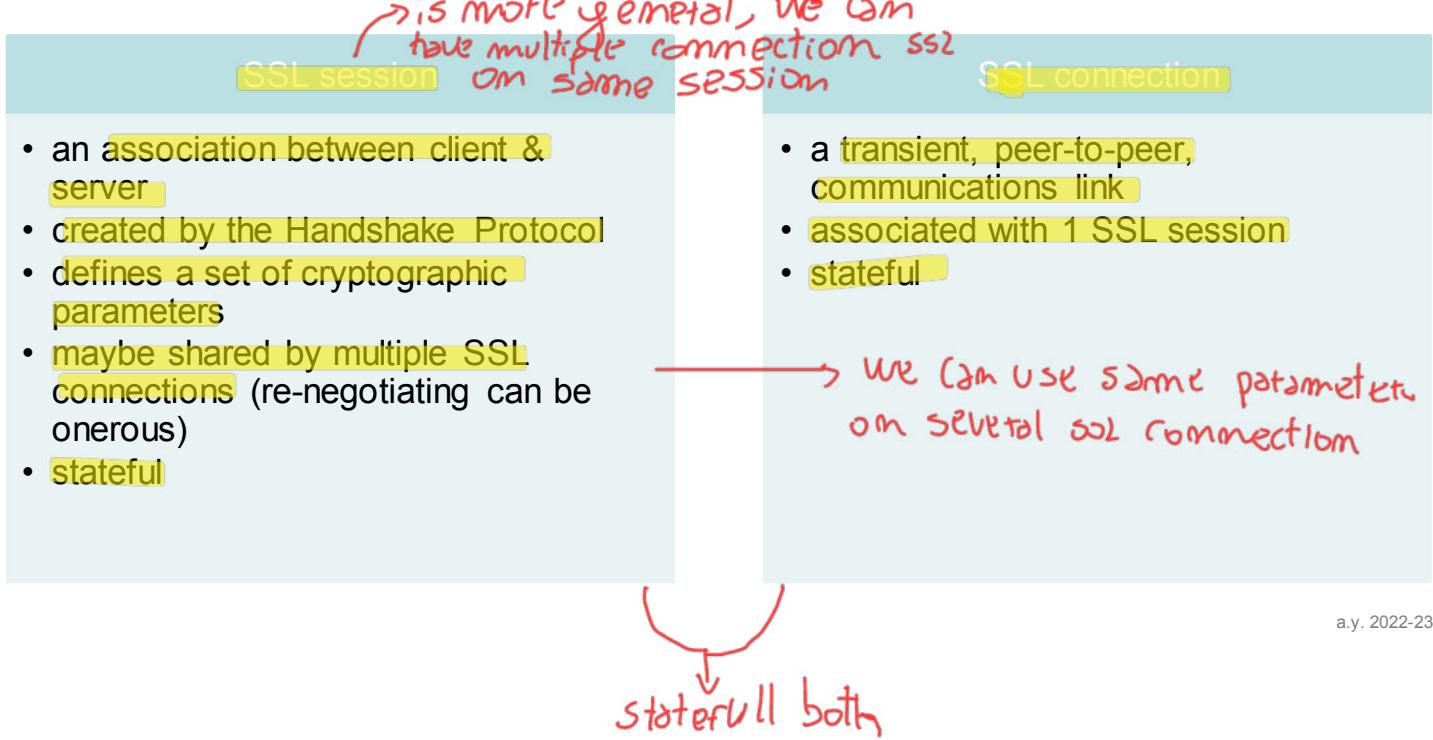
always check for revocation



## SSL Architecture

SSL have 2 part above  
TCP and one at application  
Level.

# SSL Architecture



# sessions and connections

- between any pair of parties there may be multiple secure connections
  - there may also be multiple simultaneous sessions between parties, but this feature is not used in practice
- several states associated with each session
  - once a session is established, there is a current operating state for both read and write (i.e., receive and send)
  - during Handshake Protocol, pending read and write states are created
  - after conclusion of Handshake Protocol, the pending states become the current states

# parameters defining session state

- Session identifier  
arbitrary byte sequence chosen by the server to identify an active or resumable session state
- Peer certificate  
X509.v3 certificate of the peer. This element of the state may be null
- Compression method  
The algorithm used to compress data prior to encryption.

# parameters defining session state

- **Cipher spec**  
Specifies the bulk data encryption algorithm (such as null, DES, etc.) and hash algorithm (such as MD5 or SHA-1) used for MAC calculation. It also defines cryptographic attributes such as the hash\_size.
- **Master secret**  
48-byte secret shared between the client and server.
- **Is resumable**  
flag indicating whether the session can be used to initiate new connections.

# parameters defining connection state

- Server and client random  
Byte sequences that are chosen by the server and client for each connection.
- Server write MAC secret  
The secret key used in MAC operations on data sent by the server
- Client write MAC secret  
The secret key used in MAC operations on data sent by the client.
- Server write key  
The conventional encryption key for data encrypted by the server and decrypted by the client

for obtain keys and Nonces

↑ IV, ...

Server and client have different keys,  
also different for encryption

# parameters defining connection state

- Client write key  
The conventional encryption key for data encrypted by the client and decrypted by the server.
- Initialization vectors  
When a block cipher in CBC mode is used, an initialization vector (IV) is maintained for each key. This field is first initialized by the SSL Handshake Protocol. Thereafter the final ciphertext block from each record is preserved for use as the IV with the following record

# parameters defining connection state

- Sequence numbers

Each party maintains separate sequence numbers for transmitted and received messages for each connection. When a party sends or receives a change cipher spec message, the appropriate sequence number is set to zero. Sequence numbers may not exceed  $2^{64}-1$ .

↳ Prevent replay attack

After restart  
connection or session

# SSL Record Protocol

two main services

- confidentiality

- using symmetric encryption with a shared secret key defined by Handshake Protocol

- IDEA, RC2-40, DES-40, DES, 3DES, Fortezza, RC4-40, RC4-128

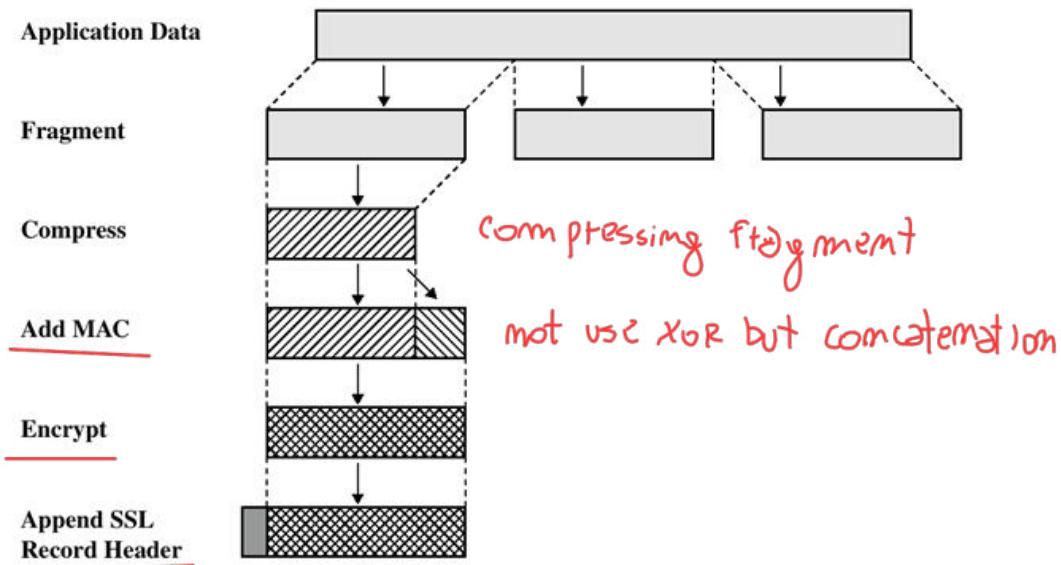
- message is compressed before encryption

- message integrity

- using a MAC with shared secret key

- similar to HMAC but with different padding

# SSL - Record Protocol



# Authentication: ~~MAC~~

---

Like HMAC (uses concatenation instead of EXOR)

```
Hash(MAC_secret_key || pad2  
    || hash(MAC_secret_key || pad1 || seqNum ||  
    SSLcompressed.type ||  
    SSLcompressed.length ||  
    SSLcompressed.fragment))
```

- pad1=0x36 repeated 48 times (MD5); 40 times (SHA-1)
- pad2=0x5C repeated ...
- SSLcompressed.type = high level protocol used to process segment

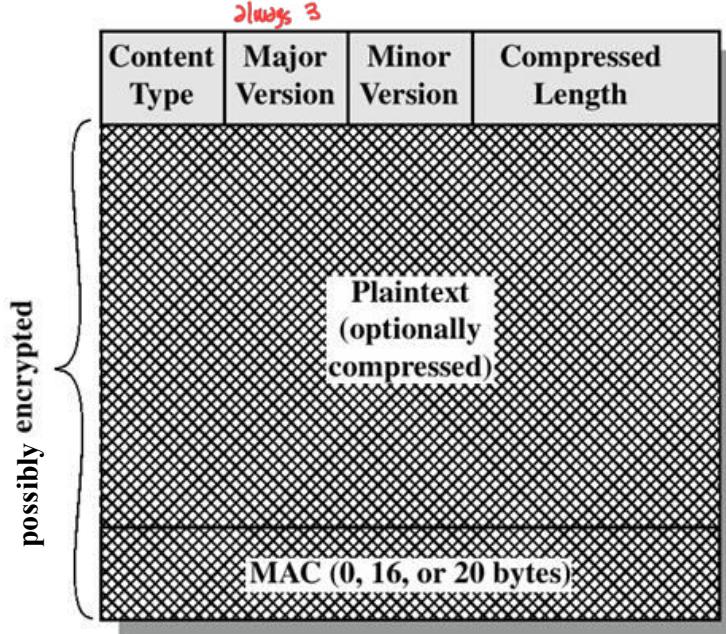
# Encryption encoding methods

↳ typically we  
use AES.

↳ not very usually SMART CARD

- segment into blocks of  $2^{14} = 16\,384$  bytes
- compression (optional):
  - must be lossy and must guarantee to reduce pack size
  - default in SSLv3: no compression
- MAC computation (see previous slide)
  - on compressed data
- several (symmetric) encryption methods:
  - block ciphers: IDEA (128), RC2-40, DES-40, DES (56), 3DES (168),
  - Stream Cipher: RC4-40, RC4-128
  - Smart card: Fortezza

# SSL - record



# fields

32

- Content Type (8 bits)
  - The higher layer protocol used to process the enclosed fragment (`change_cipher_spec`, `alert`, `handshake`, and `application_data`). The first three are the SSL-specific protocols; no distinction is made among the various applications (e.g., HTTP) that might use SSL)
- Major Version (8 bits)
  - Indicates major version of SSL in use. For SSLv3 and TLS1.2, the value is 3
- Minor Version (8 bits)
  - Indicates minor version in use. For SSLv3, the value is 0; for TLS1.2 it is 3
- Compressed Length (16 bits)
  - The length in bytes of the plaintext fragment (or compressed fragment if compression is used).

## Versions

Major version	Minor version	Version type
3	0	SSL 3.0
3	1	TLS 1.0
3	2	TLS 1.1
3	3	TLS 1.2

# SSL - Payload

→ depend on packet type you want to send

1 byte

1
---

1 byte

Type
------

3 bytes

Length
--------

0 bytes

Content
---------

↳ important material

(a) Change Cipher Spec Protocol

↳ signal for starting use encryption

(c) Handshake Protocol

1 byte 1 byte

Level	Alert
-------	-------

1 byte

OpaqueContent

(b) Alert Protocol

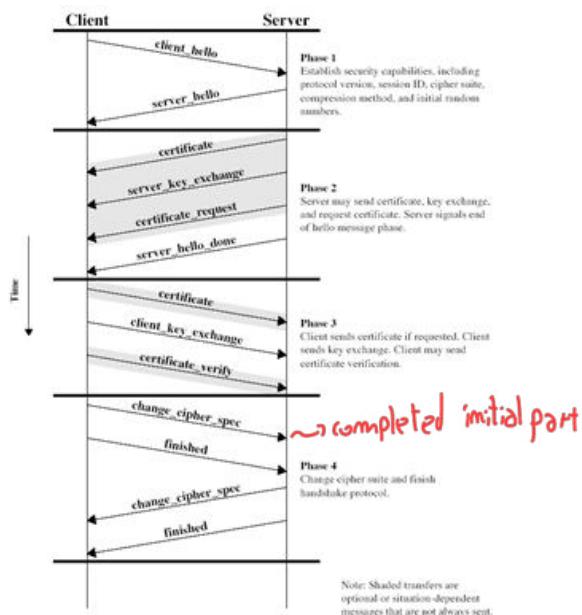
(d) Other Upper-Layer Protocol (e.g., HTTP)

# ~~SSL~~ ~~Change~~ ~~Cipher~~ ~~Spec~~ Protocol

34

- one of 3 SSL specific protocols which use the SSL Record protocol
- ~~a single message~~
- to ~~cause the pending state to be copied into the current state, which updates the cipher suite to be used on this connection~~
- usually ~~sent just after handshaking~~

# SSL Handshake Protocol



a.y. 2022-23

TSL

37

is very long, for this we want avoid to do new session for each connection

# Handshake protocol

## 4 steps

1. Hello: determine security capabilities
2. Server sends certificate, asks for certificate and starts exchange session keys
3. Client sends certificate and continues exchanges of keys
4. End of handshake protocol: encoded methods changes

Note: some requests are optional  
clear separation between handshake and the rest (to avoid attacks)

For negotiation of  
the keys they can use

## Fixed Diffie- Hellman

Diffie-Hellman key exchange in which server's certificate contains Diffie-Hellman public parameters signed by the certificate authority (CA)

Client provides its Diffie-Hellman public key parameters either in a certificate, if client authentication is required, or in a key exchange message

This method results in a fixed secret key between two peers, based on the Diffie-Hellman calculation using the fixed public keys

Fix parameters, writing them on a digital certificate

# Ephemeral Diffie- Hellman

Used to create ephemeral (temporary, one-time) secret keys. In this case, the Diffie-Hellman public keys are exchanged, signed using the sender's private RSA or DSS key.

The receiver can use the corresponding public key to verify the signature. Certificates are used to authenticate the public keys.

This would appear to be the most secure of the three Diffie-Hellman options because it results in a temporary, authenticated key.

using general information changed every time

# Anonymous Diffie- Hellman

44

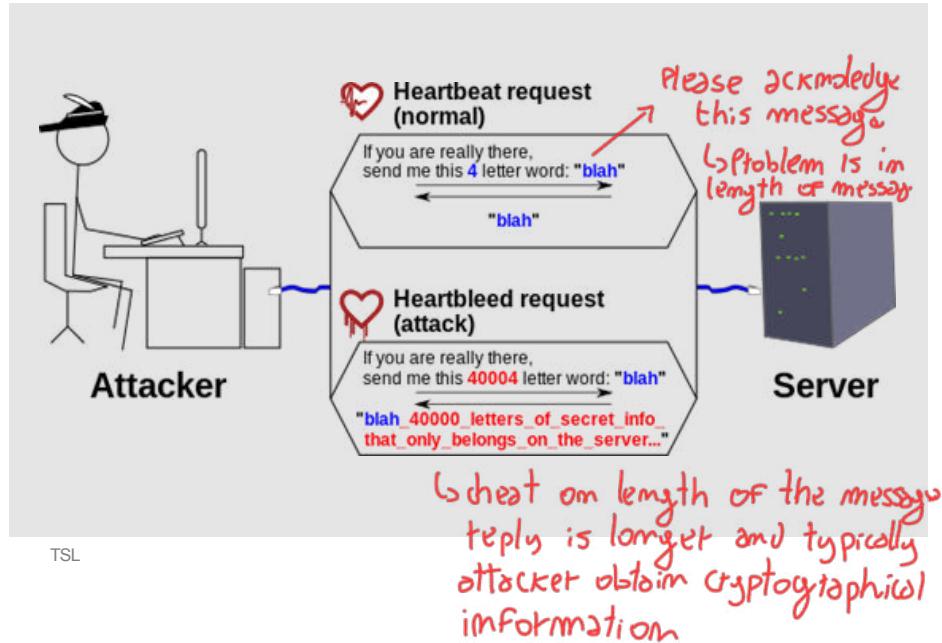
- The base Diffie-Hellman algorithm is used, with no authentication.
- Each side sends its public Diffie-Hellman parameters to the other, with no authentication. This approach is vulnerable to man-in-the-middle attacks, in which the attacker conducts anonymous Diffie-Hellman with both parties.

Not used weak to MITM

# SSL/TLS: heartbeat & heartbleed

very famous vulnerability

- severe vulnerability in OpenSSL allowing attackers obtaining huge amounts of data from the "secure" server, by directly accessing to its memory
- according to some sources it was known since 2012
- fixed in April 2014



# SECURE SHELL (SSH)

→ not mean the protocol but an environment that allows to connect to a remote host provides a terminal

→ is commercial; is popular the open source version

CYBERSECURITY  
PROF. F. D'AMORE



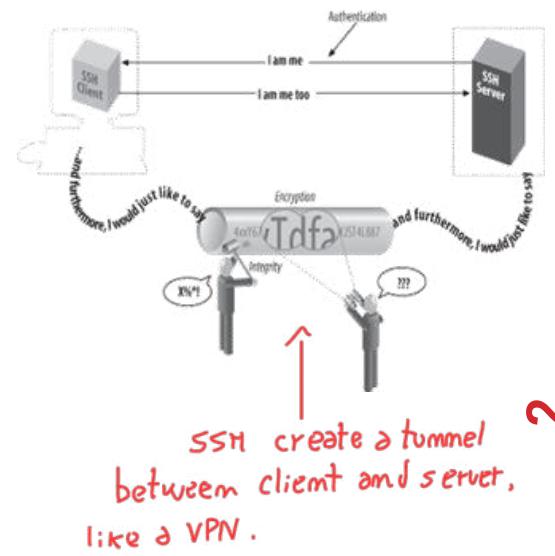
- Using services directly of TCP, is above this level SECURE SHELL, you can connect to a remote host using a secure shell.
- SSH can be the environment of the protocol on what is based.

PORT 20-21  
|

- old protocol FTP is been replaced to secure FTP, that is based on secure shell that works on port = 22.

# THE SSH PROTOCOL

- Network protocol that allows data to be exchanged using a secure channel between two networked devices
- The channel is made secure through
  - One- or two-ways authentication
  - Data confidentiality (encryption)
  - Data integrity
  - Authenticity , full data integrity



a.y.2022-23  
ssh

When use two way authentication typically ssh ask for username and password , traditional credentials

↳ we have a VPN based on IPSEC, TLS and SSH

# SSH ALLOWS TO

login to a shell on a remote host (replacing Telnet and rlogin)

execute a single command on a remote host (replacing rsh)

securely transfer files → better than HTTPS, less overhead

back up, copy and mirror files efficiently and securely (in combination with rsync)

forward or tunnel a TCP/IP port

arrange a full-fledged encrypted VPN

- OpenSSH only

forward X from a remote host (possible through multiple intermediate hosts)

browse the Web through an encrypted proxy connection with SSH clients that support the SOCKS protocol

securely mount a directory on a remote server as a filesystem on a local computer using SSHFS (SSH filesystem)

a.y.2022-23

ssh

3

Forward TCP/IP port is of two type: 1. we have a machine not able to connect to internet, you can use a local machine that is the only with the access

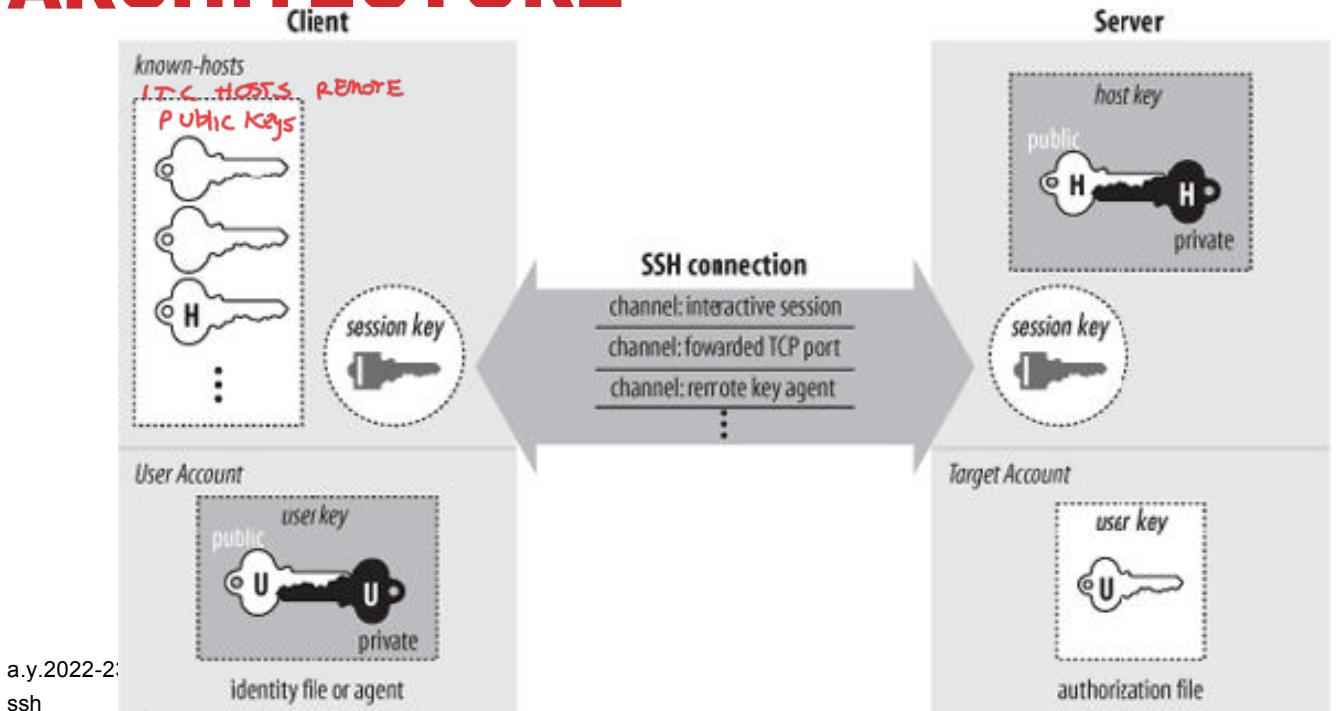
. SCS is commercial company that own SSH

↳ OpenSSH have less powerful feature.

# SSH BASIC

## ARCHITECTURE

• For security need cryptography but not sufficient



→ there is one file  
ITC host that contain  
list of known host )  
of who you trust the  
public key .

↳ there are a list of account  
allow to connect to the  
service, a session key for  
each client can be used

# SSH ARCHITECTURE

- **actually divided into four major pieces**
  - formally described as four separate protocols in different IETF documents
  - in principle independent of one another
- **Four Protocols**
- **SSH Transport Layer Protocol (SSH-TRANS)**
- **SSH Authentication Protocol (SSH-AUTH)**
- **SSH Connection Protocol (SSH-CONN)**
- **SSH File Transfer Protocol (SSH-SFTP)**

application software (e.g., ssh, sshd, scp, sftp, sftp-server)		
SSH Authentication Protocol [SSH-AUTH]	SSH Connection Protocol [SSH-CONN]	SSH File Transfer Protocol [SSH-SFTP]
client authentication publickey hostbased password <i>gssapi</i> <i>gssapi-with-mic</i> <i>external-keyx</i> <i>keyboard-interactive</i>	channel multiplexing pseudo-terminals flow control signal propagation remote program execution authentication agent forwarding TCP port and X forwarding terminal handling subsystems	remote filesystem access file transfer
<b>SSH Transport Protocol [SSH-TRANS]</b>		
algorithm negotiation session key exchange session ID server authentication privacy integrity data compression	<p style="text-align: center;">↓</p> <p>only protocol on TCP</p>	
TCP (or other transparent, reliable, duplex byte-oriented connection)		

other don't interact with TCP, but only with SSH-TRANS.

a.y.2022-23

ssh

# SSH SUMMARY

server listening on port 21

- **SSH-TRANS**
  - providing initial connection, server authentication, basic encryption and integrity services
  - after establishing SSH-TRANS connection, client has single, secure, full-duplex byte stream to authenticated peer
- **SSH-AUTH over the SSH-TRANS connection to authenticate client**
  - multiple authentication mechanisms may be used
  - it *requires* only one method: public key with the DSS algorithm
  - it further defines two more methods: password and hostbased
  - a number of other methods have been defined in various Internet-Drafts, and some of them have gained wide acceptance
- **after authentication SSH clients invoke the SSH-CONN protocol**
  - a variety of services over the single pipe provided by SSH-TRANS
  - support of multiple interactive and noninteractive sessions
    - multiplexing several streams (or *channels*) over the underlying connection
    - managing X, TCP, and agent forwarding
    - propagating signals across the connection (such as SIGINT, when a user types ^C to interrupt a process);
    - terminal handling, data compression, remote program execution etc.
- **finally, an application may use SSH-SFTP over an SSH-CONN channel to provide file-transfer and remote filesystem manipulation functions**

a.y.2022-23

ssh

→ use auth-conn

# SSH-TRANS (RFC 4253)

## Transport layer

- Handles initial key exchange, server authentication and sets up encryption, compression and integrity verification
- Exposes to upper layer an interface for sending/receiving plaintext packets with sizes of up to 32,768 bytes each (more can be allowed by the implementation)
- Transport layer also arranges for key re-exchange, usually after 1 GB of data or after 1 hour

# SSH-AUTH (RFC 4252)

User authentication layer

Somewhat some of  
these method are not  
implemented

- Handles client authentication and provides a number of authentication methods
- Authentication is client-driven. Widely used user authentication methods include the following:
  - password: straightforward password authentication; not implemented by all programs
  - publickey: public key-based authentication, usually supporting at least DSA or RSA keypairs, with other implementations also supporting X.509 certificates
  - keyboard-interactive (RFC 4256): server sends one or more prompts to enter information and the client displays them and sends back responses keyed-in by the user (e.g., one-time passwords)
  - GSSAPI: extensible scheme to perform SSH authentication using external mechanisms such as Kerberos 5 or NTLM, providing single sign on capability to SSH sessions
    - application programming interface for programs to access security services
    - usually implemented by commercial SSH implementations for use in organizations, though OpenSSH does have a working GSSAPI implementation

a.y.2022-23

ssh

# SSH-CONN (RFC 4254)

## Connection layer

- This layer defines the concept of channels, channel requests and global requests using which SSH services are provided
- A single SSH connection can host multiple channels simultaneously, each transferring data in both directions.
- Channel requests are used to relay out-of-band channel specific data, such as the changed size of a terminal window or the exit code of a server-side process.
- The SSH client requests a server-side port to be forwarded using a global request. Standard channel types include:
  - shell for terminal shells, SFTP and exec requests (including SCP transfers)
  - direct-TCP/IP for client-to-server forwarded connections
  - forwarded-TCP/IP for server-to-client forwarded connections

a.y.2022-23

ssh

# SSH-2 ALGORITHMS

now supported in future  
will be change

**Key establishment through Diffie-Hellman key exchange**

- Variety of groups supported

**Server authentication via RSA or DSS signatures on nonces (and other fields)**

**HMAC for MAC algorithm**

**3DES, RC4, or AES**

**Pseudo-random function for key derivation**

# **SSH (OPENSSH) BASIC CLIENT USAGE**

**ssh [options] <userid>@<hostaddress> [command]**

**some options**

- v [-v [-v]] enables verbose modes**
- C enables gzip compression**
- 1,-2 forces version 1, 2**
- 4,-6 forces addresses for IPv4, IPv6**

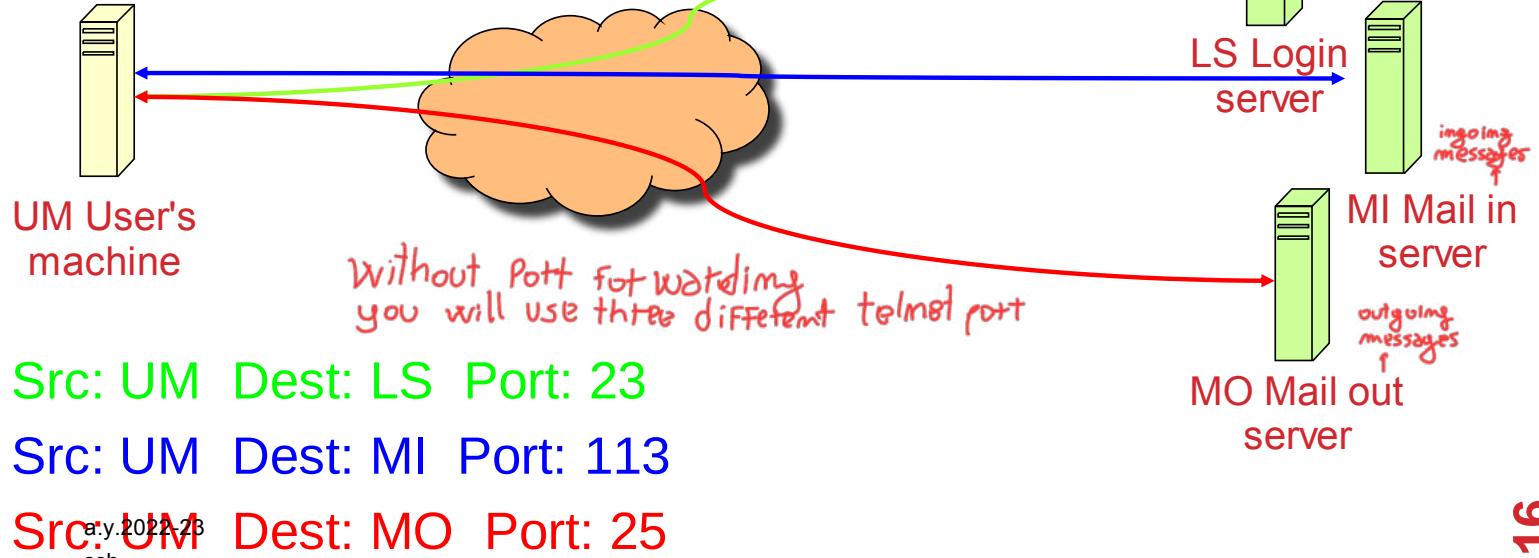
*↳ if you want use ssh for execute one command*

**if just ssh <userid>@<hostaddress> then an interactive terminal session is established**

a.y.2022-23  
ssh

# SSH PORT FORWARDING

Without SSH or port forwarding.



# SSH PORT FORWARDING

Recall: TCP port number ‘identifies’ application.

SSH on local machine: (client)

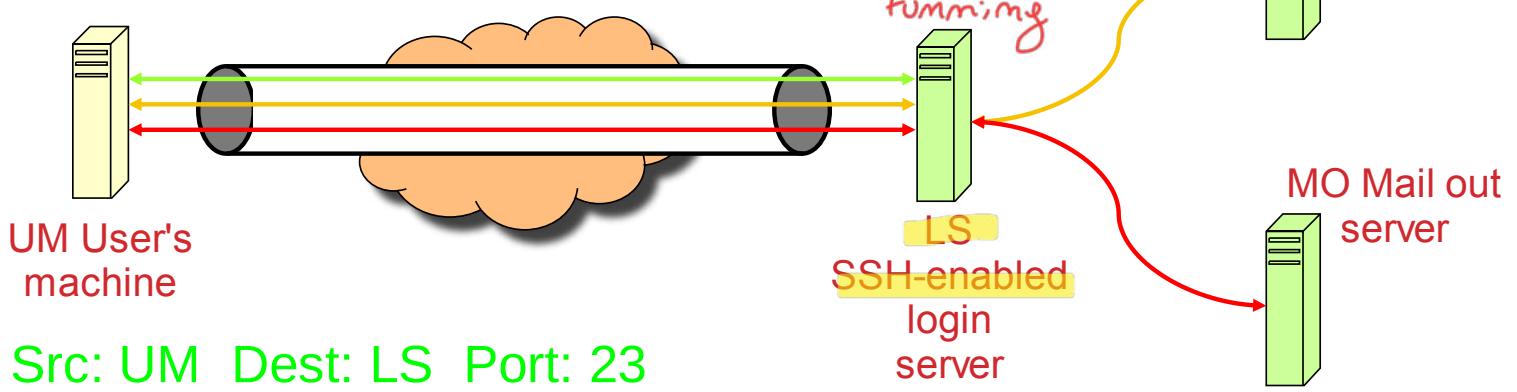
- Intercepts traffic bound for server.
- Translates standard TCP port numbers.
  - E.g. port 113 → port 5113.
- Sends packets to SSH-enabled server through SSH secure channel.

SSH-enabled server:

- Receives traffic.
- Re-translates port numbers.
  - E.g. port 5113 → port 113.
- Forwards traffic to appropriate server using internal network.

# SSH PORT FORWARDING

With SSH and port forwarding:



Src: UM Dest: LS Port: 23

Src: UM Dest: MI Port: 113

Src: UM Dest: LS Port: 5113

Src: <sub>a.y.2022-23</sub> LS Dest: MI Port: 113

Src: UM Dest: MO Port: 25

Src: UM Dest: LS Port: 5025

Src: LS Dest: MO Port: 25

# SSH PORT FORWARDING

## EXAMPLE

how to connect to a “hidden” hostB, same network as the SSH server hostA

ssh -L portC:hostB:portB user@hostA

The given portC on client is to be forwarded to the given hostB and portB on the remote side, through hostA (hostB should be reachable from hostA). Whenever a connection is made to portC, the connection is forwarded over the secure channel, and a connection is made to portB of hostB from the remote machine (localhost). Port forwardings can also be specified in the configuration file.

Only the superuser can forward privileged ports.

Example

ssh -L 54321:10.0.2.92:22 damore@linuxserv.dis.uniroma1.it

makes it available at port 54321 of localhost port 22 of the (unreachable) host 10.0.2.92, providing a tunnel between (localhost, 54321) and (10.0.2.92, 22)

option -N disables execution of remote commands (forwarding only)

ssh -N -L 54321:10.0.2.92:22 ...

option -f runs the ssh command in the background

ssh -f -N -L 54321:10.0.2.92:22 ...

good for two-hops sshfs!(now:sshfs -p 54321 user@localhost:mountdir)

using -R instead of -L switches the roles of client and server

a.y.2022-23

ssh

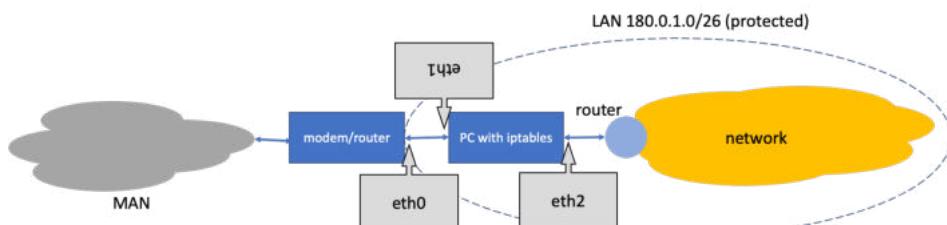
## Cybersecurity/CNS exam

14th July 2022

Syllabus 2021-22

Please write in a large and understandable handwriting, using a pen. Any pencil portions will be skipped. If necessary, use capital or block letters.

1. Write the name of the exam, first name, surname, matriculation and number of homeworks done in the top line of the first page.
2. Define *weak collision resistance*. Is it needed for making a hashing function of cryptographic quality? What is the difference with *second preimage resistance*? [3pt]
3. Why do we want hashing *fingerprints* longer than 160 bits? [3pt]
4. Compare *keyed hashing functions* to *digital signatures*: pros and cons. [3pt]
5. Explain how *authenticity* is a strengthening of *integrity*. [3pt]
6. What are the differences between *integrity verification* and *digital signature verification*? Discuss in depth. (Superficial approaches will be penalised). [3pt]
7. Define Shamir's secret sharing technique  $(n, k)$ , where  $n$  is the number of participants and  $1 < k \leq n$ , where  $k$  is the threshold. Given three points, will these be sufficient for  $k = 3$ ? Discuss. [3pt]
8. Public keys checking.
  - 8.1. Define OCSP. How is it better than CRLs? [2pt]
  - 8.2. Why does OCSP cause privacy issues? [2pt]
9. RSA: What happens if we exchange public and private keys? [2pt]
10. Firewalls
  - 10.1. What is the difference between packet filtering and session filtering? [2pt]
  - 10.2. You are a home user with a strong Internet connection, however the modem/router - offering only wired connection - is not providing any firewall/NAT. Carefully check the figure.



Default iptables policy is ALLOW and you are asked to protect ALL the LAN blocking bidirectional http/https traffic to/from site 140.11.201.23. Write the corresponding iptables rules. [4pt]

- 10.3. Can a session filtering firewall help confidentiality? (It cannot guarantee it, but some policies may help). Discuss. [2pt]

**2.** Weak collision resistance is the fact that your have as input a document, hashing function is knowned, you are computing in a easy way the fingerprint of the document. The resistance ask that is hard to find another document that is different from first one with same fingerprint. It almost the same thing of second preimage resistance, the only difference is that you can compute the hashing and have already the result. The S.p.t. ask that is hard to find a document that have the same fingerprint (don't look of the first document, only on the second document). For cryptographic hashing function we require that is hard to find a collision (not require few collision), weak resistance is necessary but not sufficient, we need strong collision resistance (find two document collision, not given one), also need an enough lenght of fingerprint because it possible to execute birthday attack (today 256 bit)

**3.** We want a fingerprint longer to 160 bit for prevent the birthday attack.

**4.** have a Keyed hashing function is good for date integrity and date authenticity, having a key is great for the two property. It will guarantee that Alice and Bob can identify themself with the shared key, only between the two parts (it is a secret key). With this approach there isn't no reputation because in front of a third party, Alice can't say it was Bob and Bob

can't say the reverse, third party will not able to check the truth. In some case is needed the no repudiation and is used the DIGITAL SIGNATURE, that guarantee data integrity because it use an unkeyed cryptographic hashing (just U.C.H guarantee data integrity but not data authenticity), if Alice keep the private key in a safe key, finding the encryption using the private key will mean it was Alice, there is NO REPUDIATION. Digital signature provide no repudiation (PRO) but for having it we need a public key infrastructure that is costly.

5. Authenticity is an addition to integrity, that is the sum of data integrity plus you can say who is the author, no absolute definition. Additional part is knowing identity of sender of the message.

6. in integrity verification (hashing), we have that Alice send a message to Bob, the message is a pair that is message and MAC, Bob consider the message and the hashing that is known, the result must be the same fingerprint, the hashing can be UNKEYED or KEYED, is the same. In digital signature there is an additional step, Bob should compare the received fingerprint with the computed one, in D.S. Bob could use RSA, compute the fingerprint by de-crypting the

digital signature using the public key getting the fingerprint, the rest is same to before, only difference is in the Not.

7. Technique of Shamir based on polynomial interpolation. With three points you can describe a curve of degree two that miss one point for  $K = 3$ , it is not sufficient.

8. OCSP is the approach of using the digital certificate, that is providing in some fields, on extension ( $\geq$  Version 3), the link for obtaining online information on revocation of certificates, better than CRL that is updated everytime, there is a time interval where you don't have fresh informations. For very recent information you should use OCSP. The certificate revocation list it is just a document, with a format, where there is the serial number and date of revocation, for each revocation. Not good for privacy OCSP, if all website are secured by TLS protocol, it means that for each website the person will asking the OCSP test, so the CA will be able to know the history of website visited by user. In OCSP-stapling is not client ask the OCSP but the server.

9. If we exchange public and private key in RSA. RSA by construction is made in a way an exchange arbitrary the private and public. The way in

which encryption and decryption work is the same, if the same mathematical part. If we define a place for public and private, it is a problem we exchange the role, the no repudiation it is no longer valid, for the policy.

10.1 packet filtering is completed without memory, is stateless, session filtering have some memory, very limited memory, TCP session, not about application. Useful for enforce some more strong rule for allow packets or not.

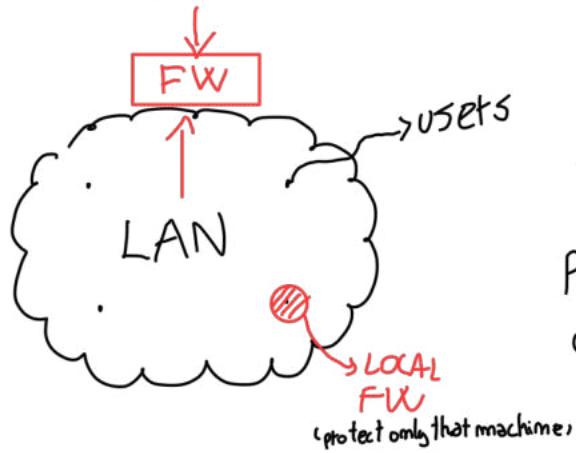
10.3 A session filtering is not sufficient for guarantee confidentiality, but it can help because you can restrict the set of users allowed to contact some destination, preventing accessing to some data in the machine protected by firewall

10.2 you should specify ip of destination, the netmask, that protocol is TCP (-p TCP) and the ports for the two protocols HTTP/HTTPS 80 and 443. You should block bidirectional traffic, target would be -j BLOCK, two rules for incoming and outgoing, the interface chain is FORWARD because it is not a personal firewall. For a personal is INPUT and OUTPUT. It nearly always need bidirectional rules. (Wrong -p SSH because is application protocol, only until TCP protocol)



## FIREWALL

We have in SW or and HW. Solution in HW is less flexible but quicker, in SW is more flexible but slower.



LAN belongs to an organization that want to protect the network using several policies given by organization.  
ORG (Policies).

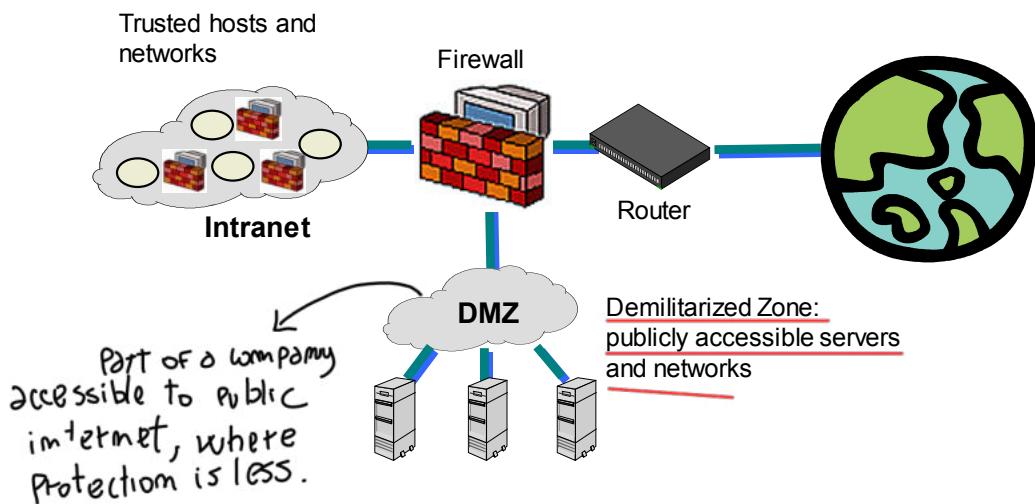
↳ should be given by management, but not by IT people.

First way for protect this LAN is a FIREWALL, that is a machine that is protecting all LAN, controlling packet outgoing and incoming in the LAN. All network traffic pass through to the FIREWALL. FW has been configured with some policies. FW inspect the packets that are trying to pass and decide if each packet can pass or not. Packets not approved are simply dropped, without informing sender.

The FW not protect against a lot of threats, for example attack from another local user, for these people implement a local firewall called PERSONAL FIREWALL that can be implemented like FW, but the difference the traffic captured is only that is incoming and outgoing the machine. FW more general.

# Firewalls

- Idea: separate local network from the Internet



a.y. 2022-23

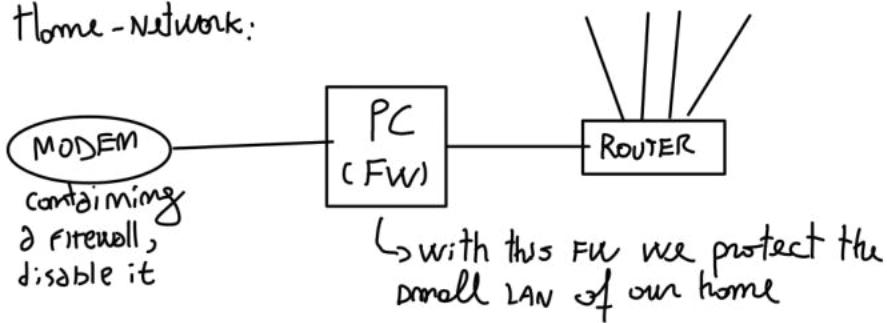
firewalls

2

two main reason for use a **PERSONAL FIREWALL** are:

- protection against inside.
- stricter rules.

Home-Network:



↳ with this FW we protect the small LAN of our home

TWO GENERAL APPROACH:

- BLACK LISTING** → is to prepare a list of forbidden actions, sites, locations, networks points (IP, TCP) that should be blocked.
- WHITE LISTING**

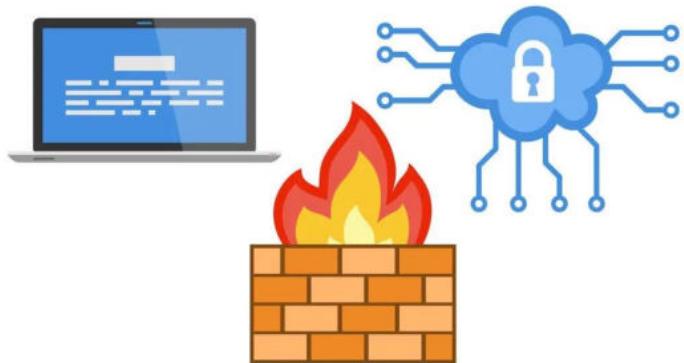
↳ opposite since whatever is blocked, any kind of packets, except only packet from some defined point are allowed

↳ more restricted, but more secure

# Firewall

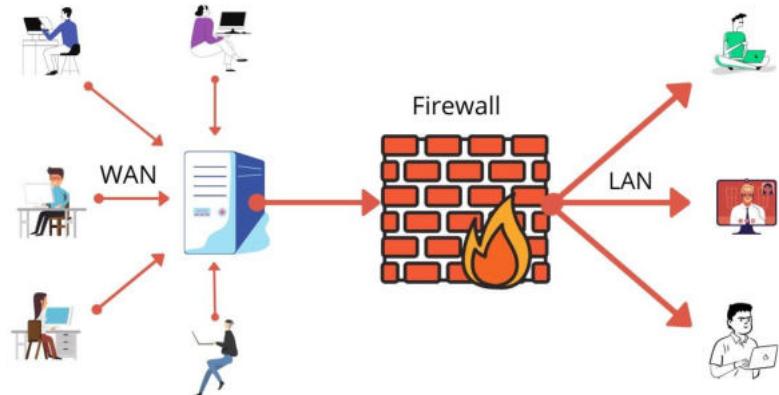
Firewall controls and monitors network traffic

- Most cases: a firewall links an internal network to the external world (public internet)
  - Limits the inbound and outbound traffic
  - Only authorized traffic passes the firewall
  - Hides the internal network to the external world
  - Controls and monitors accesses to service
- On end-user machines
  - “Personal firewall”
- Should be immune to attacks



# Firewall

- Does not protect with respect to attacks that pass the firewall
- Does not protect from attacks originated within the network to be protected
- Is not able to avoid/block all possible viruses and worms (too many, dependent on specific characteristics of the Operating Systems)



# Firewall Types

implemented within router

Packet- or session-filtering router (Packet filter)

- filtering is done by inspecting headers (and payloads, in some cases)
- usually stateless, sometimes stateful

Proxy gateway

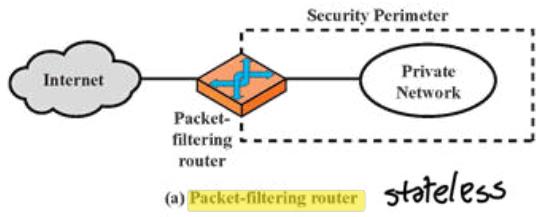
- All incoming traffic is directed to firewall, all outgoing traffic appears to come from firewall
- Application-level: separate proxy for each application
  - Different proxies for SMTP (email), HTTP, FTP, etc.
  - Filtering rules are application-specific
- Circuit-level: application-independent, "transparent"

Personal firewall with application-specific rules

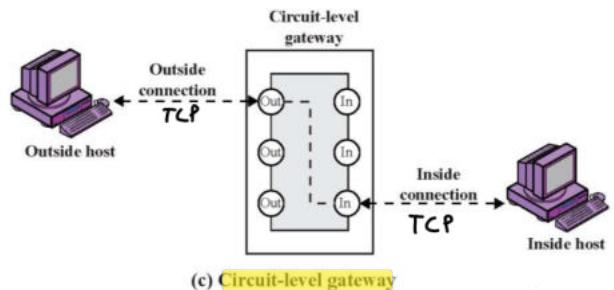
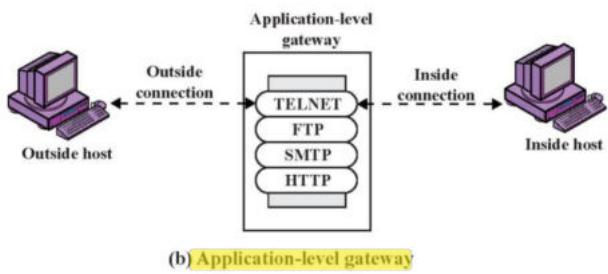
- E.g., no outbound telnet connections from email client

Packet inspect inside application context

# Firewall Types



NOT VERY USED



also this application level,  
but use TCP level, allow or disallow  
TCP connection, inspect all messages  
in session

a.y. 2022-23

firewalls

in packet-filtering each packet it is inspected alone, there is also session-filtering that is a little statefull, remember packets send on the TCP connection.

# Packet Filtering accept of drop

For each packet, firewall decides whether to allow it to proceed

Decision must be made on per-packet basis

• Stateless; cannot examine packet's context (TCP connection, application to which it belongs, etc.)

To decide, use information available in the packet

IP source and destination addresses, ports

Protocol identifier (TCP, UDP, ICMP, etc.)

TCP flags (SYN, ACK, RST, PSH, FIN)

ICMP message type

Filtering rules are based on pattern-matching

Default rule: accept/reject

a.y. 2022-23

pattern are checked  
in a sequential way, if first don't match pass to second and ecc.

firewalls

7

# Packet Filtering Examples

*internal network*

A	action	ourhost	port	theirhost	port	comment
	block	*	*	SPIGOT	*	we don't trust these people
	allow	OUR-GW	25	*	*	connection to our SMTP port <i>all</i>

B	action	ourhost	port	theirhost	port	comment
	block	*	*	*	*	default

→ block all, whitelist  
(implementation, new rule above this line)

C	action	ourhost	port	theirhost	port	comment
	allow	*	*	*	25	connection to their SMTP port

D	action	src	port	dest	port	flags	comment
	allow	{our hosts}	*	*	25		our packets to their SMTP port
	allow	*	25	*	*	ACK	their replies

E	action	src	port	dest	port	flags	comment
	allow	{our hosts}	*	*	*		our outgoing calls
	allow	*	*	*	*	ACK	replies to our calls
	allow	*	*	*	>1024		traffic to nonservers

# FTP Packet Filter

The following filtering rules allow a user to FTP from any IP address to the FTP server at 172.168.10.12

```
access-list 100 permit tcp any gt 1023 host 172.168.10.12 eq 21
access-list 100 permit tcp any gt 1023 host 172.168.10.12 eq 20
! Allows packets from any client to the FTP control and data ports
access-list 101 permit tcp host 172.168.10.12 eq 21 any gt 1023
access-list 101 permit tcp host 172.168.10.12 eq 20 any gt 1023
! Allows the FTP server to send packets back to any IP address with TCP ports > 1023
```

```
interface Ethernet 0
```

```
access-list 100 in ! Apply the first rule to inbound traffic
access-list 101 out ! Apply the second rule to outbound traffic
!
```

Anything not explicitly permitted  
by the access list is denied!

a.y. 2022-23

firewalls

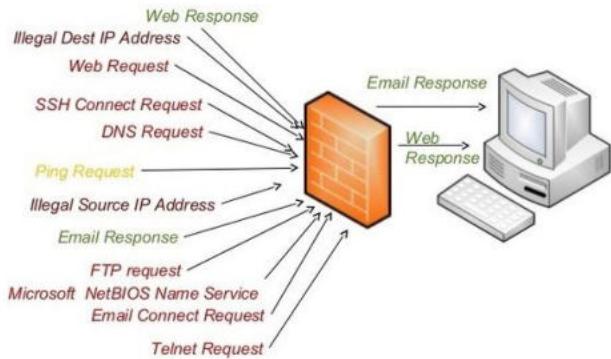
9

- rule 100 and 101 are implemented
- rule for outgoing and incoming packets

# Weaknesses of Packet Filters

- Do not prevent application-specific attacks
  - For example, if there is a buffer overflow in URL decoding routine, firewall will not block an attack string
- No user authentication mechanisms
  - ... except (spoofable) address-based authentication
  - Firewalls don't have any upper-level functionality
- Vulnerable to TCP/IP attacks such as spoofing
  - Solution: list of addresses for each interface (packets with internal addresses shouldn't come from outside)
- Security breaches due to misconfiguration

## Packet Filter Firewall



# Fragmentation Attacks

A fragmentation attack uses two or more pcks such that each pck passes the firewall; BUT when the pcks are assembled (and it is possible to check TCP header) they form a pck that should be dropped. Examples

Two ack pack assembled from SYN pck (TCP request)

Split ICMP message into two fragments, the assembled message is too large

- Buffer overflow, OS crash

Fragment a URL or FTP "put" command

- Firewall needs to understand application-specific commands to catch

IP fragments overlap

- some operating systems do not properly handle that

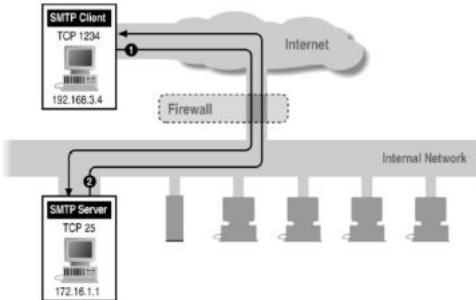
excessive number of incomplete fragmented datagrams

- denial of service attack or an attempt to bypass security measures

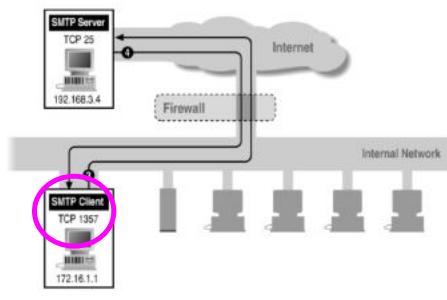
# Limitation of Stateless Filtering

- In TCP connections, ports with numbers less than 1024 are permanently assigned to servers
  - 20, 21 for ftp, 23 for telnet, 25 for smtp, 80 for http...
- Clients use ports numbered from 1024 to 16383
  - They must be available for clients to receive responses
- What should a firewall do if it sees, say, an incoming request to some client's port 1234?
  - It must allow it: this could be a server's response in a previously established connection...
  - ...OR it could be malicious traffic
  - Can't tell without keeping state for each connection

# Example: Variable Port Use



Inbound SMTP



Outbound SMTP

# Session Filtering

packet filtering  
+  
knowledge of TCP connection

- Decision is made separately for each packet, but in the context of a connection
  - If new connection, then check against security policy
  - If existing connection, then look it up in the table and update the table, if necessary
    - Only allow incoming traffic to a high-numbered port if there is an established connection to that port
- Hard to filter stateless protocols (UDP) and ICMP
- Typical filter: deny everything that's not allowed
  - Must be careful filtering out service traffic such as ICMP
- Filters can be bypassed with IP tunneling

# Example: Connection State Table

Source Address	Source Port	Destination Address	Destination Port	Connection State
192.168.1.100	1030	210.9.88.29	80	Established
192.168.1.102	1031	216.32.42.123	80	Established
192.168.1.101	1033	173.66.32.122	25	Established
192.168.1.106	1035	177.231.32.12	79	Established
223.43.21.231	1990	192.168.1.6	80	Established
219.22.123.32	2112	192.168.1.6	80	Established
210.99.212.18	3321	192.168.1.6	80	Established
24.102.32.23	1025	192.168.1.6	80	Established
223.212.212	1046	192.168.1.6	80	Established

rule can be append at begin of  
the list of end, or to a specific  
position

## iptables

sudo iptables -nvL

- **Iptables** is used to set up, maintain, and inspect the tables of IPv4 packet filter rules in the Linux kernel.
- Several different tables may be defined. Each table contains several built-in chains and may also contain user-defined chains.
- **chain** = list of rules which can match a set of packets
  - each rule specifies criteria for a packet and an associated target, namely what to do with a packet that matches the pattern

a.y. 2022-23

firewalls

16

the list of rules are sequential, if doesn't match a packet, we go to next rule, end of the chain we apply default one

→ here if no matching we use a default policy  
that can be accept or discard

→ 3e  
packet match  
third rule

target can be accept or discard  
or also a user define:

EXAMPLE



EXAMPLE subroutine



if packet match a  
rule go to another  
user chain

4e  
go to successive  
rule

↓  
if no matching happen the  
default is to return packet

# tables

normally there exist a few standard tables

- filter (default)
- nat
- mangle
- raw

each table contains built-in chains and may contain user-defined chains

# Built-in chains

5 built in chains  
for us only three

- **PREROUTING:** Packets will enter this chain before a routing decision is made.
- **INPUT:** Packet is going to be locally delivered.
- **FORWARD:** All packets that have been routed and were not for local delivery will traverse this chain.
- **OUTPUT:** Packets sent from the machine itself will be visiting this chain.
- **POSTROUTING:** Routing decision has been made. Packets enter this chain just before handing them off to the hardware.
- Built-in chains have a *policy*, for example **DROP**, which is applied to the packet if it reaches the end of the chain.

a.y. 2022-23

firewalls

18

- PRF/POST ROUTING IS RELATED TO NETWORKING
- WHITE LIST MOST SECURE BUT NOT FLEXIBLE → my decision can be changed over time
- my built in chain have 2 policies: accept or discard

# targets

- each rule specifies criteria for a packet and a target
- if the packet does not match a rule, next rule in chain is then examined
- if it matches, then the next rule is specified by the value of the target *reject packet*
- targets: accept, drop, queue, return, or name of a user-defined chain

↳ can be only in a user define chain

↳ all content of packet is handled by a special process, we don't use firewalls

# standard targets

- **accept** = let the packet through
- **drop** = drop the packet on the floor
- **queue** = pass the packet to userspace (what happens depends on a queue handler, included in all modern linux kernels)
- **return** = stop traversing this chain and resume at the next rule in the previous (calling) chain

conversations are bidirectional  
need two rules for in  
and out

## tables, again

- filter & three built chains: INPUT, FORWARD, OUTPUT
  - default table, contains the built-in chains INPUT (for packets destined to local sockets), FORWARD (for packets being routed through the box), and OUTPUT (for locally-generated packets)
- nat
  - Network Address Translation occurs before routing. Facilitates the transformation of the destination IP address to be compatible with the firewall's routing table. Used with NAT of the destination IP address,
  - It consists of three built-ins: PREROUTING (for altering packets as soon as they come in), OUTPUT (for altering locally-generated packets before routing), and POSTROUTING (for altering packets as they are about to go out)

a.y. 2022-23

firewalls

21

## Personal firewall vs firewall of all network

INPUT/OUTPUT  
typically

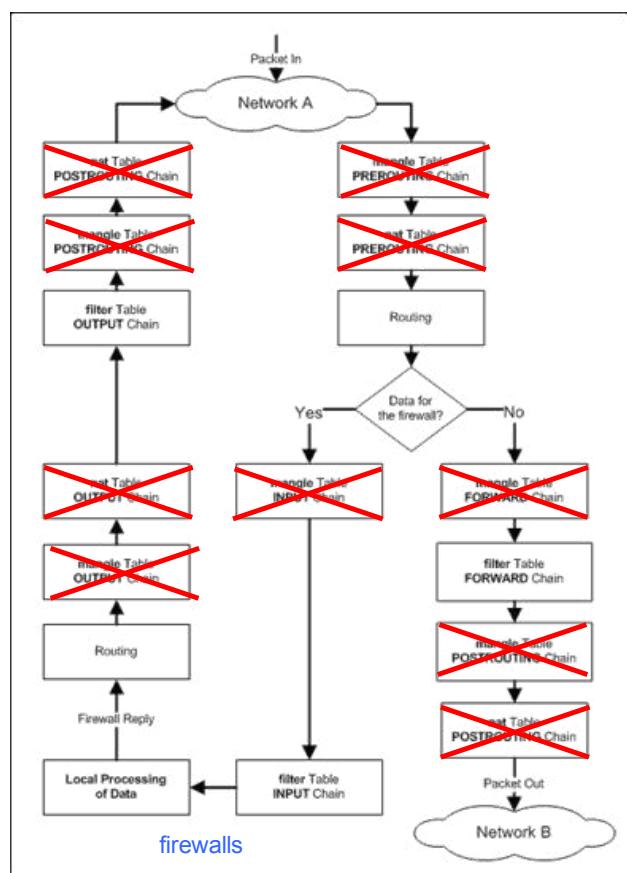
FORWARD : for packet that are not  
to be sent to final destination to the fw,  
but must pass for fw.

# tables, again 2

- **mangle**
  - TCP header modification (modification of the TCP packet quality of service bits before routing occurs)
  - built-in chains: PREROUTING (for altering incoming packets before routing), OUTPUT (for altering locally-generated packets before routing), INPUT (for packets coming into the box itself), FORWARD (for altering packets being routed through the box), and POSTROUTING (for altering packets as they are about to go out)
- **raw**
  - mainly used for configuring exemptions from connection tracking

# Iptables Packet Flow Diagram

a.y. 2022-23



only part related to firewalling

# iptables extended modules

↪ can't act at application level (TCP and below OK!)

- iptables can use extended packet matching modules (regular expression, -- ) can't use -p HTTP or -p SSL!
- two ways: implicitly (when -p is specified) or explicitly (with the -m option, followed by the matching module name)
- after these, various extra command line options become available, depending on the specific module.
- -m state [!] --state st
  - where st in {INVALID, ESTABLISHED, NEW, RELATED}
    - a new connection has state related if it's a new connection and if is related to an already established connection

a.y. 2022-23

firewalls

24

! for negation, "is not matching"

- state : work on state name of TCP connection

# extended modules

module name	info
account	accounts traffic for all hosts in defined network/netmask
addrtype	matches packets based on their address type (UNSPEC, UNICAST, LOCAL, BROADCAST, ANYCAST, MULTICAST...)
connbytes	matches by how many bytes or packets a connection have transferred so far, or by average bytes per packet
connlimit	allows to restrict the number of parallel TCP connections to a server per client IP address (or address block)
connrate	module matches the current transfer rate in a connection

# extended modules

module name	info
conntrack	allows access to connection tracking information (more than the "state" match)
hashlimit	gives you the ability to express '1000 packets per second for every host in 192.168.0.0/16' or '100 packets per second for every service of 192.168.1.1' with a single iptables rule
icmp	<u>allows specification of the ICMP type</u> <i>for tum Ping</i> <i>ECHO PING /REPLY</i>
iprange	matches on a given arbitrary range of IPv4 addresses
length	matches the length of a packet against a specific value or range of values

# extended modules

module name	info
mac	matches source MAC address. It must be of the form XX:XX:XX:XX:XX:XX. Note that this only makes sense for packets coming from an Ethernet device and entering the PREROUTING, FORWARD or INPUT chains
<b>multiport</b>	matches a set of source or destination ports. Up to 15 ports can be specified. A port range (port:port) counts as two ports. It can only be used in conjunction with -p tcp or -p udp
nth	matches every 'n'th packet
owner	matches various characteristics of the packet creator, for locally-generated packets. It is only valid in the OUTPUT chain, and even this some packets (such as ICMP ping responses) may have no owner, and hence never match

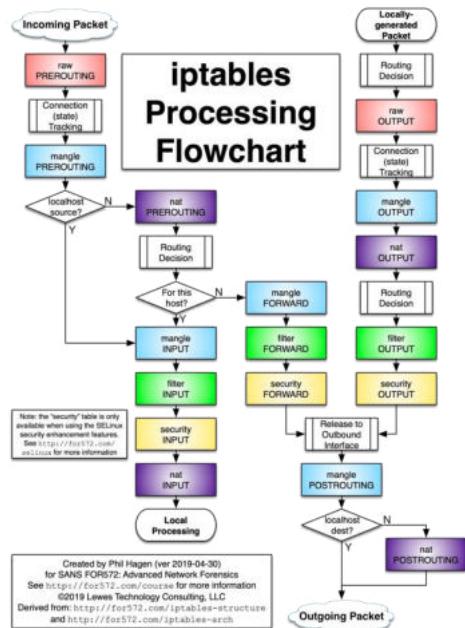
# extended modules

module name	info
psd	attempts to detect TCP and UDP port scans. This match was derived from Solar Designer's scanlogd
quota	Implements network quotas by decrementing a byte counter with each packet
random	randomly matches a certain percentage of all packets
state	allows access to the connection tracking state for this packet
<b>tcp</b>	extensions are loaded if '--protocol tcp' is specified
time	matches if the packet arrival time/date is within a given range
ttl	matches the time to live field in the IP header
<b>udp</b>	loaded if '--protocol udp' is specified

many other modules!

# iptables options

2019



type of options

- COMMANDS

- `-A (--append) chain rule-specification`

- `-L (--list) [chain]`

- PARAMETERS

- `[!] -p (--protocol) protocol`

- `[!] -s (--source) address[/mask]`

- `[!] -i (--in-interface) name`

- OTHER

- `-n (--numeric)` without at beginning

- `-j (--jump) target`

see man iptables for more

a.y. 2022-23

firewalls

29

-A to add a rule to a chain  
 catdb is turbulent in chain

-I 1 new rule be number 1

# Firewall: packet filter

always add sudo ...



Rules:

`IPTABLES -t TABLE -A CHAIN -[I|O] IFACE -s x.y.z.w -d a.b.c.d -p PROT -m state --state STATE -j ACTION`

Rules use

PACKET ADDRESS (TABLE) = nat | filter | ...

ORIGIN OF CONNECTION/PACK. = INPUT (I) | OUTPUT (O) | FORWARD (F) | ...

NETWORK INTERFACE (IFACE) = eth0 | eth1 | ppp0 (network adapter)

PROTOCOL (PROT) = tcp | icmp | udp .....

STATE OF THE CONNECTION (STATE) = NEW | ESTABLISHED | RELATED .....

BASED ON THE RULES THERE IS ONE ACTION

ACTION ON THE PACKET = DROP | ACCEPT | REJECT | DNAT | SNAT .....

# Firewall: examples

-i eth0 = packet incoming to eth0 is dropped

Assume eth0 interface of a router to public Internet

- Block all incoming traffic

iptables -A FORWARD -i eth0 -j DROP

Note: packets are discarded with no reply to the sender; in this way the firewall does not protect against flooding attacks and does not provide information for attacks based on "port scanning"

- Accept pck from outside if they refer to a TCP connection started within the network

iptables -A FORWARD -i eth0 -m state  
--state ESTABLISHED -j

ACCEPT

Note state "ESTABLISHED" allows to decide whether the connection originated from the inside or the outside; ESTABLISHED information is stored in the IPTABLES;

## example 1

- Allow firewall to accept TCP packets for routing when they enter on interface eth0 from any IP address and are destined for an IP address of 192.168.1.58 that is reachable via interface eth1. The source port is in the range 1024 to 65535 and the destination port is port 80 (www/http)

```
iptables -A FORWARD -s 0/0 -i eth0 -d  
192.168.1.58 -o eth1 -p TCP --sport 1024:65535 -  
-dport 80 -j ACCEPT
```

-i input

-o output

firewalls -s source IP

-d destination IP

--sport source port

--dport destination port

## example 2

- allow the firewall to send ICMP echo-requests (pings) and in turn accept the expected ICMP echo-replies

```
iptables -A OUTPUT -p icmp --icmp-type echo-request -j ACCEPT
```

```
iptables -A INPUT -p icmp --icmp-type echo-reply -j ACCEPT
```

## example 3

- accept at most 1 ping/second

```
iptables -A INPUT -p icmp --icmp-type echo-request -m limit --limit 1/s -i eth0 -j ACCEPT
```

- limiting the acceptance of TCP segments with the SYN bit set to no more than five per second

```
iptables -A INPUT -p tcp --syn -m limit --limit 5/s -i eth0 -j ACCEPT
```

## example 4

- Allow the firewall to accept TCP packets to be routed when they enter on interface eth0 from any IP address destined for IP address of 192.168.1.58 that is reachable via interface eth1. The source port is in the range 1024 to 65535 and the destination ports are port 80 (www/http) and 443 (https).
- The return packets from 192.168.1.58 are allowed to be accepted too. Instead of stating the source and destination ports, you can simply allow packets related to established connections using the -m state and --state ESTABLISHED options.

```
iptables -A FORWARD -s 0/0 -i eth0 -d 192.168.1.58 -o eth1 -p TCP --sport 1024:65535 -m multiport --dports 80,443 -j ACCEPT
```

```
iptables -A FORWARD -d 0/0 -o eth0 -s 192.168.1.58 -i eth1 -p TCP -m state --state ESTABLISHED -j ACCEPT
```

## example 5

- allow DNS access from/to firewall

```
iptables -A OUTPUT -p udp -o eth0 --dport 53 --  
sport 1024:65535 -j ACCEPT
```

```
iptables -A INPUT -p udp -i eth0 --sport 53 --  
dport 1024:65535 -j ACCEPT
```

## example 6

- allow www & ssh access to firewall

```
iptables -A OUTPUT -o eth0 -m state --state  
ESTABLISHED,RELATED -j ACCEPT  
iptables -A INPUT -p tcp -i eth0 --dport 22 --  
sport 1024:65535 -m state --state NEW -j ACCEPT  
iptables -A INPUT -p tcp -i eth0 --dport 80 --  
sport 1024:65535 -m state --state NEW -j ACCEPT
```

a.y. 2022-23

firewalls

37

- sudo iptables -vL for show iptables
- sudo iptables -P INPUT DROP ~> for drop packets at defaults  
OUTPUT FORWARD
- sudo iptables -A INPUT -i lo -j ACCEPT  
OUTPUT

For allow DNS must:

- sudo iptables -A INPUT -p UDP --sport 53 -j ACCEPT  
OUTPUT --dport
- sudo iptables -A INPUT -p TCP --sport 53 -j ACCEPT  
OUTPUT --dport

must find ip of site now:

- nslookup www.tutotrs.com => 52.222.130

Final: incoming and outgoing

- sudo iptables -A INPUT -s 52.222.130.0/24 -p TCP -j ACCEPT  
OUTPUT -d

## example 7

- **allow firewall to access the Internet**
  - enables a user on the firewall to use a Web browser to surf the Internet. HTTP traffic uses TCP port 80, and HTTPS uses port 443

```
iptables -A OUTPUT -j ACCEPT -m state --state  
NEW,ESTABLISHED,RELATED -o eth0 -p tcp -m  
multiport --dports 80,443 --sport 1024:65535
```

```
iptables -A INPUT -j ACCEPT -m state --state  
ESTABLISHED,RELATED -i eth0 -p tcp
```

## example 8

- allow home Network to access firewall
  - in the example, eth1 is directly connected to a home network using IP addresses from the 192.168.1.0 network. All traffic between this network and the firewall is simplistically assumed to be trusted and allowed.
- `iptables -A INPUT -j ACCEPT -p all -s 192.168.1.0/24 -i eth1`
- `iptables -A OUTPUT -j ACCEPT -p all -d 192.168.1.0/24 -o eth1`

IPT -A INPUT -i eth0 -s  
192.168.1.0/24 -p all -j ACCEPT

IPTAB -A DEFRT -o eth1 -d  
192.168.1.0/24 -p all -j ACCEPT

## example 9

- allow loopback

```
iptables -A INPUT -i lo -j ACCEPT  
iptables -A OUTPUT -o lo -j ACCEPT
```

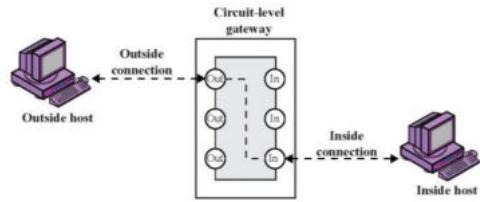
# iptables administration

- a new rule immediately applies
  - no need to restart iptables
- changes are lost at reboot
  - good to insert iptables configuration in the boot sequence
- useful commands (need sudoer)
  - iptables-save > iptables.dat
  - iptables-restore < iptables.dat
- good HOWTO:  
<https://www.netfilter.org/documentation/HOWTO/packet-filtering-HOWTO.txt>

# Firewalls: other approaches

- **Application level**
  - use a specific application
  - fully accesses protocols
    - user requests service
    - request is accepted/denied according to defined rules
    - accepted requests are served
  - needs a proxy server for each service!
- **Circuit level**
  - establishes two TCP connections
  - security enforced by limiting the authorized connections

# Circuit-Level Gateway

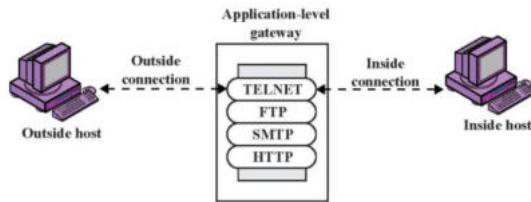


- Splices and relays two TCP connections

- Does not examine the contents of TCP segments; less control than application-level gateway
- checks validity of TCP connections against a table of allowed connections, before a session can be opened
- valid session on the base of dest/src addr/ports, time of day, protocol, user and password.
- Once session is allowed, no further checks !

- Client applications must be adapted for SOCKETS
  - “Universal” interface to circuit-level gateways
- For lower overhead, application-level proxy on inbound, circuit-level on outbound (trusted users)

# Application-Level Gateway



- Splices and relays application-specific connections
  - Example: Web browser proxy
  - Big overhead, but can log and audit all activity
- Can support user-to-gateway authentication
  - Log into the proxy server with username and password
- Can use filtering rules
- Need separate proxy for each application

# Comparison

	Performance	Modify client application	Defends against attacks
Packet filter	Best	No	Worst
Session filter		No	
Circuit-level gateway		Yes (SOCKS)	
Application-level gateway	Worst	Yes	Best

# other firewalls' operations

in addition to control in/out traffic firewalls

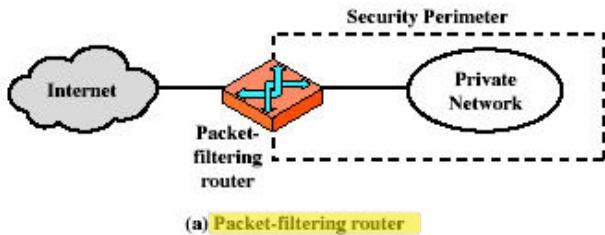
- can control band use
- hide information on internal network

## Fun with Outbound

[From "The Art of Intrusion", available [here](#)]

Guess	Guess CEO's password and log into his laptop
Try	Try to download hacking tools with FTP •Oops! Personal firewall on laptop pops up a warning every time FTP tries to connect to the Internet •Kill firewall before CEO notices
Use	Use Internet Explorer object instead •Most firewalls permit Internet Explorer to connect to the Internet
Get	<i>Get crackin'...</i>

# Firewall: where to place it



- We need servers of the network to be protected should be accessible from outside
- Solution: allow traffic for specific applications to enter (i.e., open specific ports for applications: 25 for smtp, 80 for http, ...)

BUT

- Software applications can have bugs that are exploited by the attacker
- Hacker can take control of servers bypassing the firewall

## Bastion Host



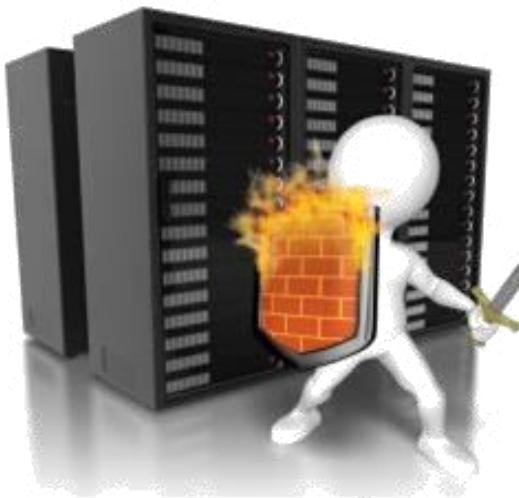
Bastion host is a hardened system implementing application-level gateway behind packet filter

- Trustable operating systems: run few applications and all non-essential services are turned off
- Application-specific proxies for supported services
  - Each proxy supports only a subset of application's commands, traffic is logged and audited (to analyze attacks), disk access restricted, runs as a non-privileged user in a separate directory (independent of others)
- Support for user authentication

All traffic flows through bastion host

- Packet router allows external packets to enter only if their destination is bastion host, and internal packets to leave only if their origin is bastion host

## Bastion Host



unique host that is reachable from the Internet

massively protected host

secure operating system (hardened or trusted)

no unneeded software, no compilers & interpreters

proxy server in a insulated environment (chrooting)

read-only file system

process checker

integrity file system checker

small number of services and no user accounts

untrusted services have been removed

saving & control of logs

source-routing disabled

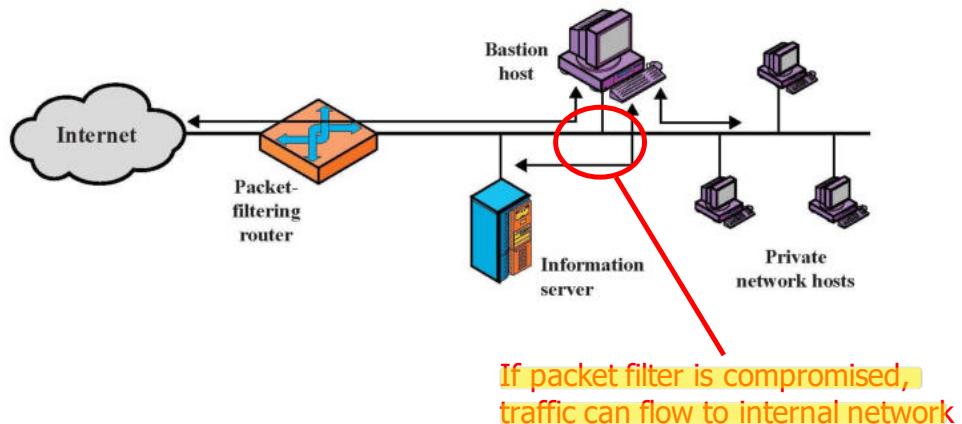
# Firewall: where to place it

## DeMilitarized Zone (DMZ)

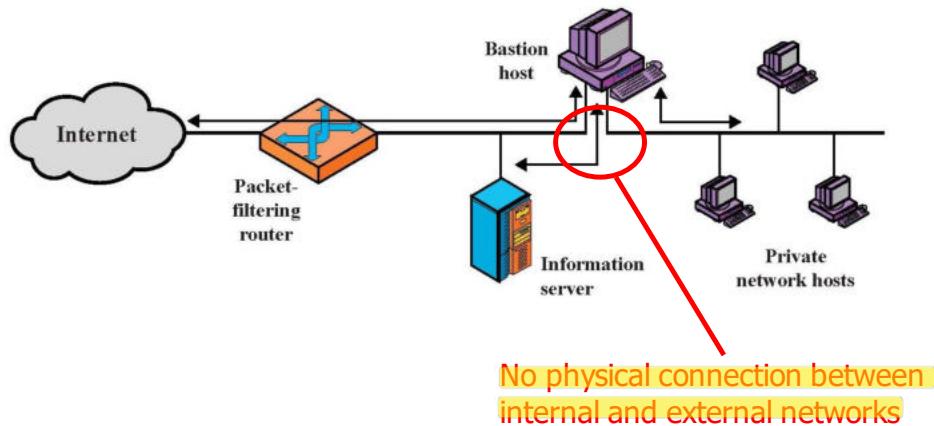
- Servers that should be reachable from the outside are placed in a special area DMZ
- External connections/users can reach these servers but cannot reach the internal network because it is blocked by the Bastion host
- External connections/users that do not access these servers are dropped
- There can be several levels

Note: great attention should be dedicated to the traffic entering the DMZ: if a hacker controls the bastion host, he can enter the internal LAN

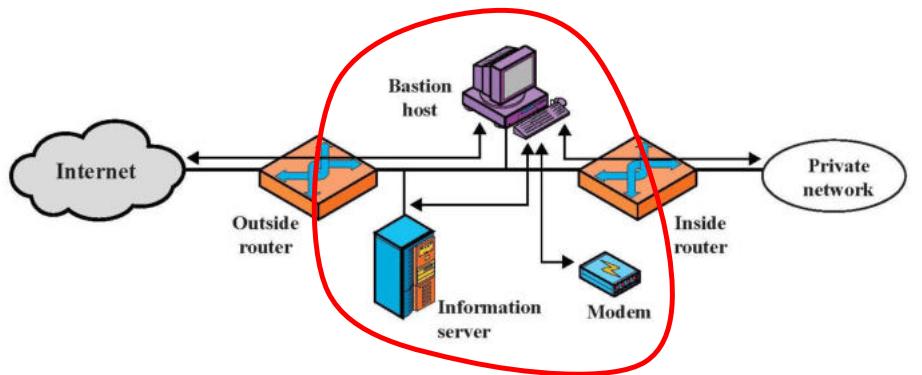
# Single-Homed Bastion Host



# Dual-Homed Bastion Host



# Screened Subnet



Only the screened subnet is visible  
to the external network;  
internal network is invisible

# Protecting Addresses and Routes

- Hide IP addresses of hosts on internal network
  - Only services that are intended to be accessed from outside need to reveal their IP addresses
  - Keep other addresses secret to make spoofing harder
- Use NAT (network address translation) to map addresses in packet headers to internal addresses
  - 1-to-1 or N-to-1 mapping
- Filter route announcements
  - No need to advertise routes to internal hosts
  - Prevent attacker from advertising that the shortest route to an internal host lies through him

# General Problems with Firewalls

- Interfere with networked applications
- Don't solve the real problems
  - Buggy software (think buffer overflow exploits)
  - Bad protocol design (think WEP in 802.11b)
- Generally, don't prevent denial of service
- Limited prevention of insider attacks
- Increasing complexity and potential for misconfiguration

# Shamir's secret sharing

Course of Cybersecurity

Prof. Fabrizio d'Amore

# sharing a secret



- assume a **secret  $S$**  is given
  - password, code, PIN, passphrase, any string...
- **goal:** sharing  $S$  with  $n$  subjects by consigning some data (fragment) to each of them
  - none of them knows  $S$
  - they can reconstruct  $S$  (only) by "joining" the fragments they hold
- applications: boards of directors, nuclear weapons control, shutdown sequences, joint bank accounts, consensus etc.
  - all authorised members must agree
- can be *easily implemented* in an **information-theoretically secure mode**
  - cannot be broken even if adversary has infinite computing power

# sharing $S$ with $n$ subjects



Assume:

$S$  is given as a sequence of bits (unsigned integer)

$n \geq 2$

Algorithm (uses xor operation  $\wedge$ )

- randomly generate fragments (nonces)  $s_1, \dots, s_{n-1}$

- set  $s_n = S \wedge s_1 \wedge s_2 \wedge \dots \wedge s_{n-1}$

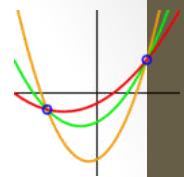
hence  $S = s_1 \wedge s_2 \wedge \dots \wedge s_n$

*↳ last fragment xor-ing  
other  $n-1$  and secret*

*$n-1$  random number, fragments  
that will be sent*

- $S$  can be reconstructed by xorring the  $n$  fragments
- If attacker knows  $n' < n$  fragments he cannot reconstruct  $S$  (not enough information)
- Knowing  $n' < n$  fragments does not provide more information than knowing one fragment
- Information-theoretically secure

*→ with  $n-1$  fragment is  
not possible  
to reconstruct  
 $S$*



# Shamir secret sharing (SSS)

## threshold scheme

*Threshold less than  $m$ ,  $k < m$*

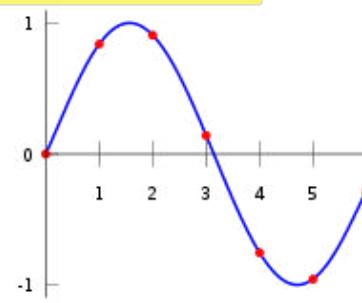
- given a secret  $S$  and a pair  $(k, n)$ , with  $1 < k \leq n$ , find  $n$  data fragments  $s_1, s_2, \dots, s_n$  such that
  - given any  $m \geq k$  fragments it is possible to reconstruct  $S$
  - $m < k$  fragments are not sufficient for reconstructing  $S$
  - reconstruction attempt from  $k-1$  fragments is not easier than reconstruction attempt from 1 fragment
- requirement: information-theoretically secure**

case  $k = n$ : easily solved by xoring nonces (see previous slide)

# SSS: ingredients for general case

## Ingredients

- mod arithmetic and finite fields
- polynomial interpolation



- Polynomial interpolation is the interpolation of a given data set by a polynomial and is based on the following unisolvence theorem
- Theorem. Given  $r > 1$  points of  $\mathbf{R}^2$  there exists a unique polynomial of degree  $r-1$  going exactly through the  $r$  points
  - Theorem also holds for polynomials defined over Galois fields

# SSS: generation of fragments

- let  $p$  be a prime ( $p > S, p > n$ ) bigger them secrets and participants
- randomly choose  $k-1$  integers in  $[0, p)$ :  $a_1, a_2, \dots, a_{k-1}$ ; let be  $a_0 = S$
- consider polynomial  $P(x) = a_{k-1}x^{k-1} + a_{k-2}x^{k-2} + \dots + a_1x + a_0 \pmod{p}$
- let be  $s_i = P(i)$ , for  $i = 1, 2, \dots, n$

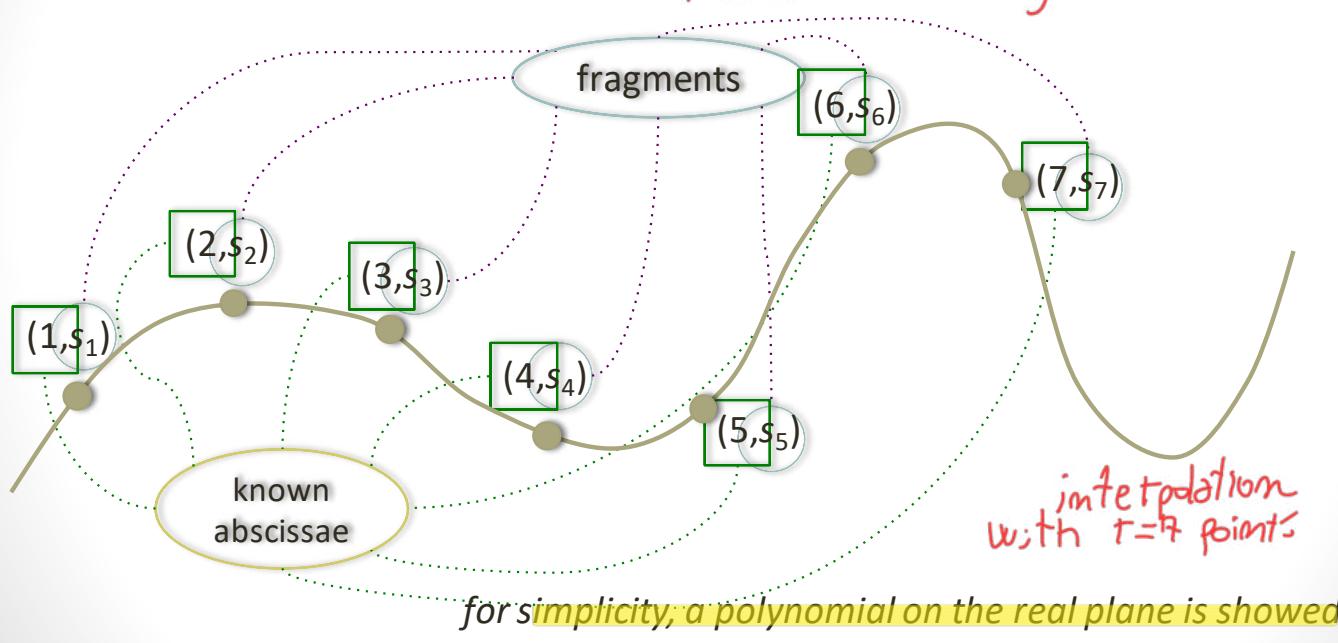
by construction it holds  $P(0) = S \quad a_0 \pmod{p} = S \quad (p > S)$

After discarding  $P(x)$  and  $S$ , only the  $n$  points  $(i, s_i)$  are known

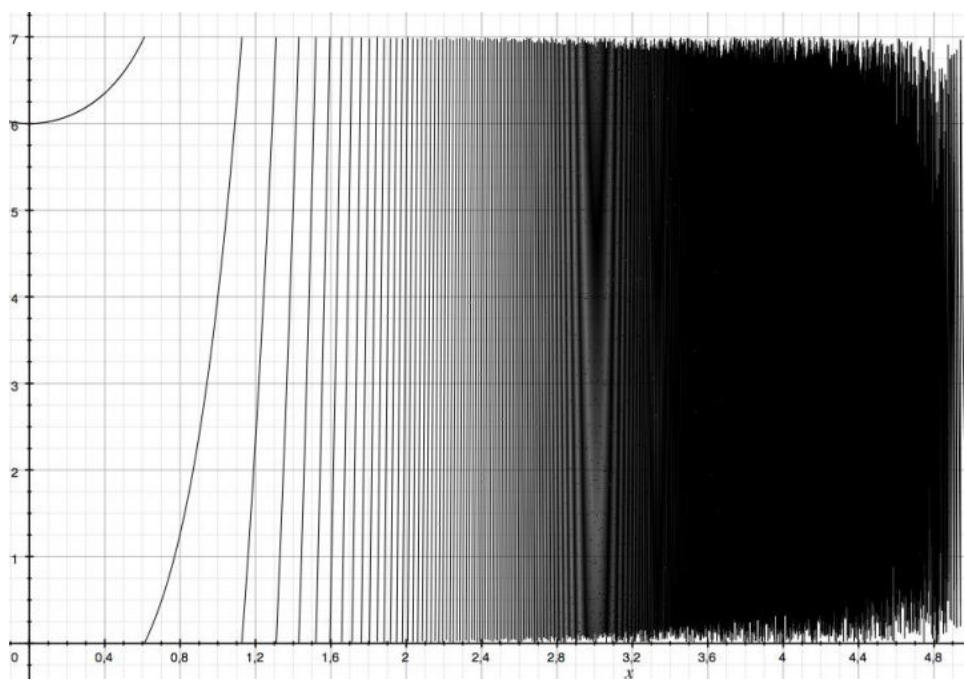
↳ with  $P(x)$  you can reconstruct  $S$ ! must discard

# SSS: the interpolating polynomial

$s_i$ : value of the polynomial,  
 $p(x)$  and  $s$  is destroyed.



$$y = (3x^5 + 2x^2 + 6) \bmod 7$$



# SSS: reconstructing S



- Given  $k$  fragments  $s_{i1}, s_{i2}, \dots, s_{ik}$  find the degree  $k-1$  polynomial going through  $(i_1, s_{i1}), (i_2, s_{i2}), \dots, (i_k, s_{ik})$
- Use for instance the Lagrange formula (polynomial denoted by  $L$ )

Given a set of  $k+1$  data points

$$(x_0, y_0), \dots, (x_j, y_j), \dots, (x_k, y_k)$$

where no two  $x_j$  are the same, the interpolation polynomial in the Lagrange form is a linear combination

$$L(x) := \sum_{j=0}^k y_j \ell_j(x) \quad (\text{Wikipedia})$$

of Lagrange basis polynomials

$$\ell_j(x) := \prod_{\substack{0 \leq m \leq k \\ m \neq j}} \frac{x - x_m}{x_j - x_m} = \frac{(x - x_0)}{(x_j - x_0)} \cdots \frac{(x - x_{j-1})}{(x_j - x_{j-1})} \frac{(x - x_{j+1})}{(x_j - x_{j+1})} \cdots \frac{(x - x_k)}{(x_j - x_k)},$$

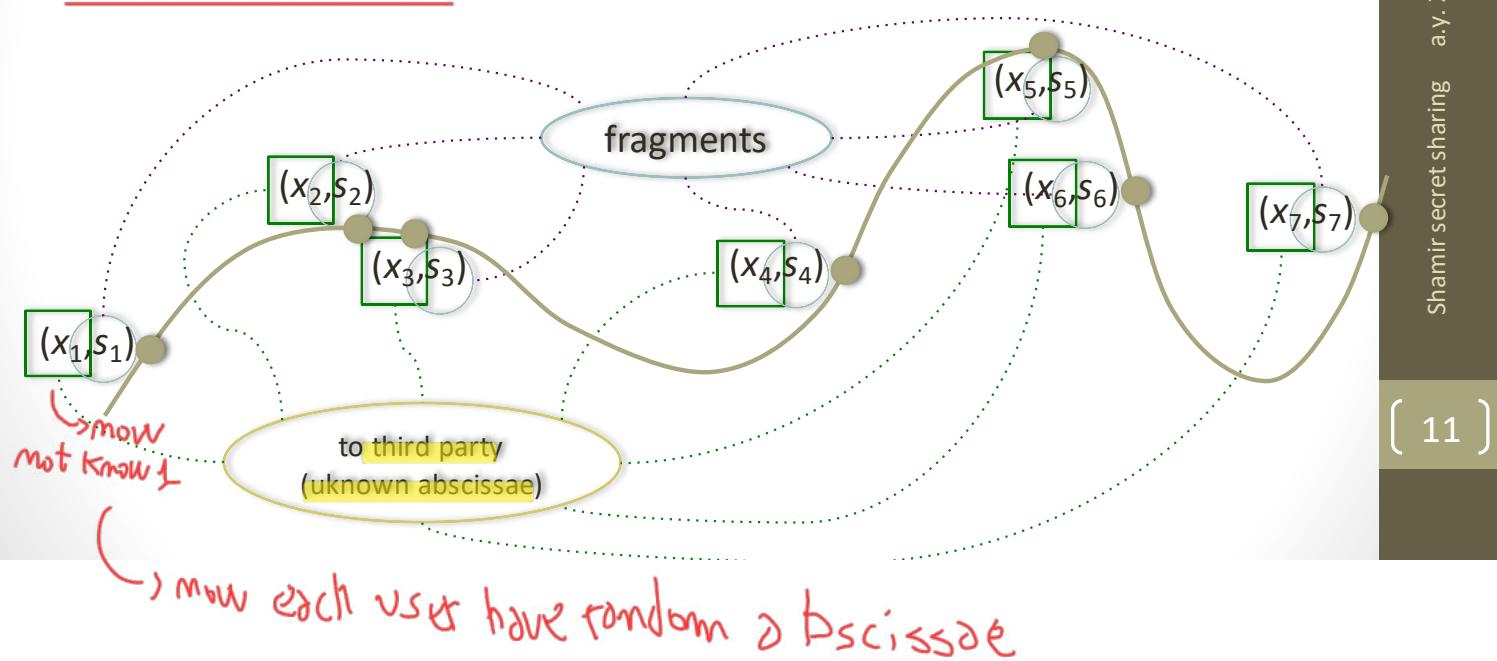
- Then  $S = L(0)$

# properties of SSS

- size of fragments and of secret are upper bounded by size of  $p$   
 $(|s_i|, |S| < |p|)$
- if  $k$  is kept fixed fragments can be dynamically added/deleted without affecting the other fragments
- it is straightforward to generate a new set of fragments: randomly build a new polynomial
- we can assign higher weights to members by giving them more than one fragment

# introducing a third party

- use unknown abscissae, given to a trusted third party, for additional services



# third party: extra services

- gives evidence of reconstruction and identifies the contributors
  - crystal safe-box metaphor
- can recognise possible cheaters (if stores hashes of fragments)
- maintains at least same security as traditional approach
  - if compromised does not reveal the secret

# Cybersecurity/CNS exam

8th September 2022 - Syllabus 2021-22 - 120 minutes

Please write in a large and understandable handwriting, using a pen. Pencil portions will be skipped. If necessary, use capital or block letters.

0. Write the name of the exam, first name, surname, matriculation and number of homeworks done in the top line of the first page. (I will save time).

## 1. Collision resistance

- 1.1. Define strong and weak collision resistance. [2pt]
- 1.2. Why does the strong one imply the weak one? [2pt]
- 1.3. Consider the mod s function (% s), where s is a large random prime number. Why isn't it cryptographic? Provide all reasons. [2pt]

## 2. RSA

- 2.1. When is RSA normally used for confidentiality and why? [1pt]
- 2.2. What are the keys (private and public) used for RSA and what relationship binds them? [2pt]
- 2.3. What is OS2IP? Describe it in detail. [3pt]

## 3. Authentication

- 3.1. Define SPEKE and all parameters/options that occur in it. [2.5pt]
- 3.2. In a web application, authentication is made by requesting username and password to the user, who connects via https. Design the details of authentication, clarifying how the server checks the user's credentials. [3.5pt]

4. Explain the difference between a key and a password. [1pt]

5. A VPN is based on IPSec tunnelling. What partial (meta)information is available to an eavesdropper that intercepts the packets? [2pt]

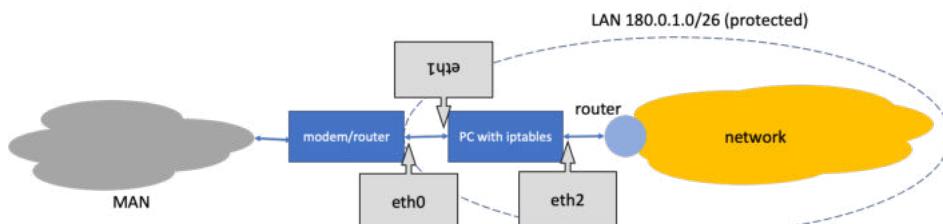
6. Describe strengths and limits of inserting a timestamp within a message of some cryptographic protocol. [2pt]

## 7. Timestamp Authority

- 7.1. What is a Timestamp Authority (TSA) and what function does it perform? [1pt]
- 7.2. Illustrate a way of timestamping a digitally signed message. [2pt]
- 7.3. Illustrate a way of timestamping a message without digital signature but with integrity. [2pt]

## 8. Firewalls

- 8.1. What is the difference between a personal firewall and a perimeter firewall? [1pt]
- 8.2. You are a home user with a strong Internet connection, however the modem/router - offering only wired connection - is not providing any firewall/NAT. Carefully check the figure.



Default iptables policy is ALLOW and you are asked to block all connections to the pc running iptables (and vice versa) except those initiated inside the LAN. Write the corresponding iptables rules. [3pt]

## Collision resistance

1.1 again (often in exam)

1.2 we demonstrate  $T\text{weak} \Rightarrow T\text{strong}$  using contradiction.

1.3 mod S function is not cryptographic because we don't know how large (bits) is S, at least 256 bits for SHA. Given a fingerprint it is easy to find a lot of different numbers having some fingerprint, given a number A it collide with  $A + kS$ . Not weak collision resistant.

## RSA

2.1 RSA is used for confidentiality only for key exchange because it is one block only. The computational effort is big, but not for encrypt a large file.

2.2 In RSA define e and d, one is the multiplicative inverse of other mod p.

2.3 OS2IP (octet stream to integer primitive) is approach for convert a stream of byte to an integer. Used in RSA, because in practice you have a string and want a number.

## Authentication

3.2. Server storing passwords that are hashed and salted, for preventing rainbow tables. Salt is plaintext. Encrypt of password in a file is bad!!!

4. Password is typeable on a keyboard, the key is not. If password and key have same length

there are more combinations in case of keys.  
Keys are stronger versus brute force attacks.

5. Tunneling makes the datagram becoming in payload of a new datagram, we construct a new datagram and there is an encrypted part, IPSEC header and a IP header. Looking at IP header the passive attacker will be able to understand where the packet is sent, the identity of VPN server, where the packet will go after arriving at VPN server it is not visible to attacker, it is contained in encrypted part. Just be able to obtain VPN server that will get the packet.

6. Insert a timestamp mean define a timestamp and a time interval where timestamp is still valid, need a Nounce for be robust against attack, like replay attacks.