# COMPUTER AND NETWORK SECURITY

Edoardo Puglisi

Goals:

- Cryptography: encription, authentication & integrity, digital signatures
- Security protocols: TLS (https), SSH, IPSec
- Firewalls

## INFORMATION SECURITY (AKA INFOSEC)

Safety - prevent unintentional accidents (maybe by people)

Security = prevent malicious activities by people

≠


Infosec is the protection of confidentiality, integrity, availability (C.I.A.), the protection of information and information systems from unauthorized access, use, disclosure, disruption, modification or destruction.

We will cover only confidentiality and integrity.

Security differs from cryptography because it deals most of the times with fraud problems (message modification, user auth.) and not with secrecy of informations (it may use cryptography anyway)

### Encryption

Given an $E$ encryption function with key $K_1$ and a $D$ decryption function with key $K_2$, for every message $m$: $D_{K_2}(E_{K_1}(m)) = m$

It can be simmetric (secret key $K_1 = K_2$) or asymmetric (public key $K_1 \neq K_2$)

Threat: something that is a source of danger

Exploit: software, data, sequence of commands that take advantages of a vulnerability

Within a communication between two user may be an adversary: passive if it only read messages between them, active if can modify it or send new ones as one of the two users (identity steal)

Security goal is to make hard (in computational sense) to find the key to decrypt the message so that no adversary can determine any meaningful information about it.

## CIPHER

- Plaintext = message before encryption
- Ciphertext = message after encryption

Caesar's cipher and substitution cipher are easy to break (also thanks to statistical analysis of language).

Perfect Cipher: plaintext space = $\{0,1\}^n$, $D$ known. Given a ciphertext $C$ the probability that exists $K_2$ such $D_{K_2}(C) = P$ for any plaintext $P$ is equal to the apriori probability that $P$ is the plaintext → ciphertext doesn't reveal any info of plaintext

$P_r[P \mid C] = P_r[P]$

$[P_r[P \mid C] = P_r[P \wedge C] / P_r[C]]$ → definition of conditional probability

$[P_r[P \wedge C] = P_r[P \mid C] P_r[C] = P_r[C \mid P] P_r[P]]$ → Bayes Theorem

## ONE TIME PAD

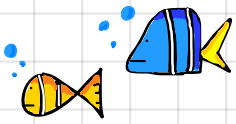
Key $K$ long at least as plain text → $E_K(P) = C = P \oplus K$ $\oplus$ = OR exclusive bit by bit $K$ chosen at random.

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

Key must be used one time only → Key stream.

OTP is a perfect cypher but has issue of key space

Shannon th.: a cypher cannot be perfect if key space size is less than message space size. ($\neq$ key or message length)



## ATTACK MODELS

Eavesdropping: listen to private conversation without consent (passive)

Known Plaintext: attacker knows plain and encrypted text and can reveal secret infos. such as secret keys.

Chosen Plaintext: attackers can choose the plaintext to be encripted so that he can reveal more infos about encryption scheme.

Adaptive chosen plaintext: sequences of plaintext depending on the previous one (encryptions)

Chosen Ciphertext: symmetric to chosen plaintext

Physical Access

"Physical" modification of messages.

## STREAM CIPHER

Define a secret key named seed. Using seed generate a byte stream named Keystream: $i^{th}$ byte can be function of

① only seed → synchronous stream cipher; or

② seed and first i-1 bytes of ciphertext → asynchronous stream cipher

Obtain ciphertext by bitwise xoring plaintext and keystream

Famous stream cipher is RC-4. All are inspired from one time pad.

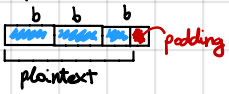
The only thing attackers can easily get is the length → compress plaintext (like zip.)

## BLOCK CIPHER

Simmetric. Plaintext with constant size (h bit) called block. Usually h = 64 or 128 bits. Ciphertext will be a function of P (block) and K (key, also fixed bits) of h bits.

Big files can be divided in multiple blocks producing multiple ciphertext blocks and then unite them.

If file's length isn't a block size multiple we have to deal with the rest → add some padding




Some situation if file's size doesn't reach 1 block length.

Problems:

① Many encryptions

② Need many encryptors (usually hardware) → Need Money! 

DES = well known block cipher, now completely broken.

↳ 56 bits, apparently 64 bit but every last bit of the 8 bytes is a parity bit

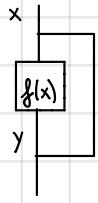

### BROKEN:

$|K| = n$ = Key length → $2^n$ possible keys

If you find a way to use less $2^n$ attemps the algorithm is considered broken but may be still secure.

Rijndael: 128 bit block, 128-256 bit key → Advanced Encryption Standard (AES)

AES:




round key

Every round use a piece of key eg. 128 bits divided in pieces of 32 bit → 4 rounds

Key schedule: algorithm that generate a long key from the starting key.

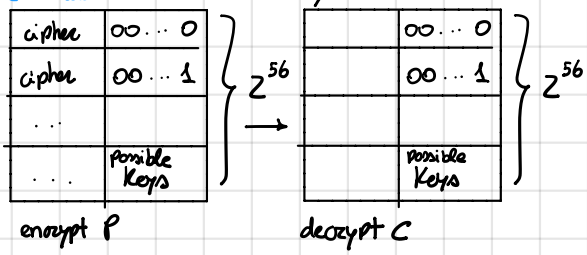
## CIPHER STRENGHTENING

Encrypting multiple times is useless if you use the same key.

What if: plaintext → DES($K_1$) → DES($K_2$) → C  is it the same as a 112 bit encryption (56×2)? NO  Double DES

MEET-in-the-middle ≠ Man in the middle : attacker knows a pair (P, C). Imagine (P,C) was produced by double

DES:




⇒ sort first table by cipher and match with second one → $2^{57}$

3-DES? Combine first 2 encryption → $2^{112}$ size table → harder but possible … in future.

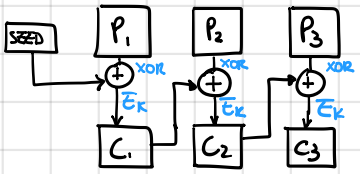
## ELECTRONIC CODE BOOK (ECB)

Separately encrypt each plaintext block IN PARALLEL reducing time (if space ovoible ofc.)

Same thing for decryption. What if attackers change 1 bit of ciphertext (more exposed), data integrity problem?

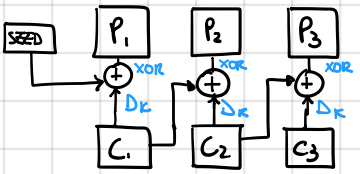
All blocks uneffected by change will be decrypted, the other will be completely useless (gorbage)!

## CBC hoining




Seed = (random) string to start the process.

Change on 1 Cipherblock will effect both its Plaintext and the next one: the second one 1 bit error only, the first one will be completely useless. → error detecting code (CRC)




Encryption can't be done in parallel. Decryption instead can be done in parallel. In most of the cases # decryption > # encry.

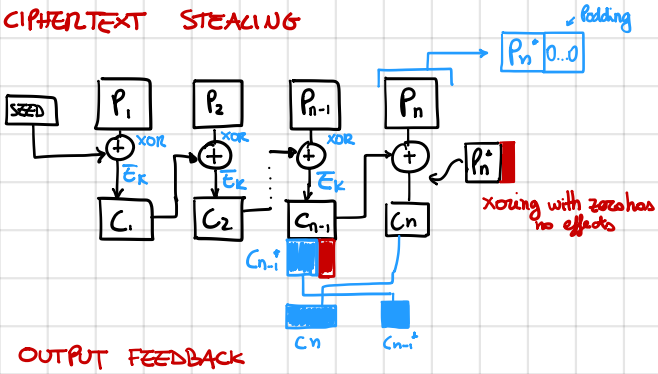
Seed is shared between sender and receiver and NOT encrypted → OPEN security

Encrypting seed is not wrong anyway, just more work to do! Change it every now and then

To encrypt a seed CBC ≃ ECB: for a single block seed, CBC just a little more secure cause need a seed to encrypt it.

CBC acts like an asynchronous stream cipher.

## CIPHERTEXT STEALING




Used to avoid padding. The process for all but two last blocks is the same.

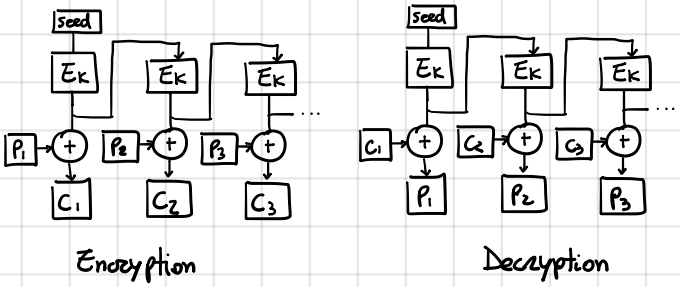
Part of the second-last ciphertext is "stolen" to pad the last plaintext block.

Then swap $C_n$ and $C_{n-1}$ position

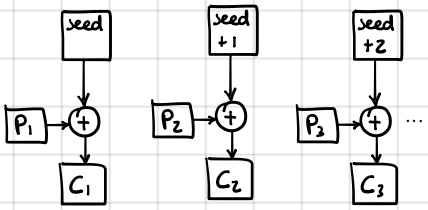
## OUTPUT FEEDBACK

It acts like a synchronous stream cipher. Both encryptor and decryptor schemes use encryptor

Ciphertext modification effect only one plaintext block.



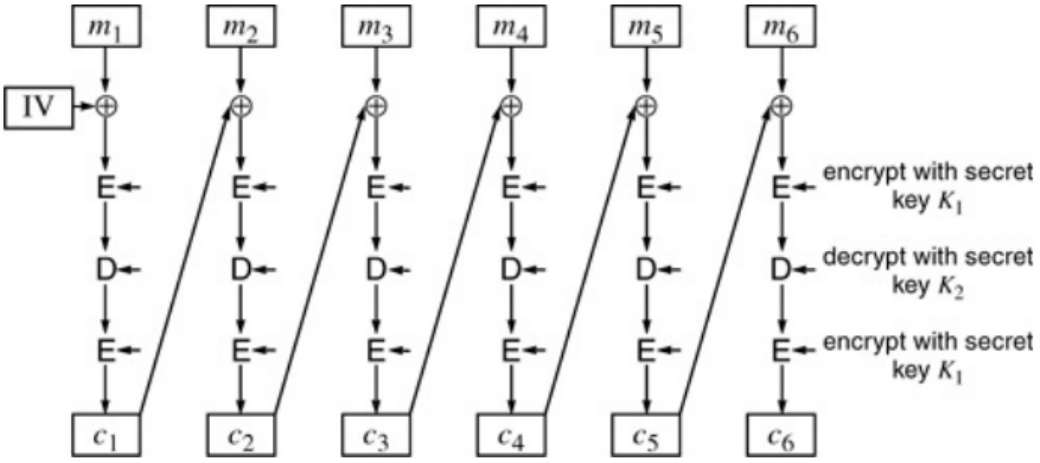
# COUNTER MODE (CTR)



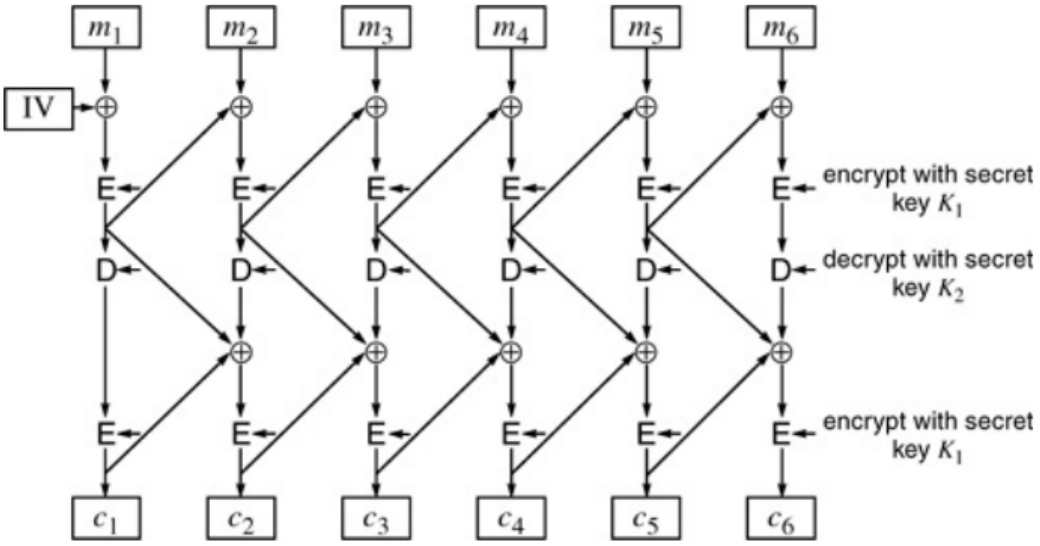

Fix repeated use of same seed like OFB
We can decrypt starting from any block.

Other possible modes are Triple Encoding + CBC = CBC with 3 consequent encryption (and 3 different keys) or Internal / External EDE CBC.

## Outside EDE CBC




## Inside EDE CBC



# GROUPS

A commutative (Abelian) group is a set $G$ with a binary operation $+$ with following properties:

① $\forall a,b \in G \quad a+b \in G \rightarrow$ G closed under +
② $\forall a,b,c \in G \quad (a+b)+c = a+(b+c) \rightarrow$ associative
③ $\forall a,b \in G \quad a+b = b+a \rightarrow$ commutative
④ $\exists 0 \in G \; \forall a \in G \quad a+0 = a \rightarrow$ identity element
⑤ $\forall a \in G \; \exists -a \in G \quad a+(-a) = 0 \rightarrow$ inverse element

} → chang $+, 0, -a$ with $\cdot, 1, 0$ for multiplication

$(H,+)$ is a subset of $(G,+)$ if it is a group and $H \subseteq G$

↳ Lagrange Th.: if $G$ finite and $(H,+)$ subgroup of $(G,+)$ then $|H|$ divides $|G|$

$a \equiv b \pmod n$ (a and b congruent modulo n) if $|a-b|$ is multiple of $n$ i.e. integer division of $a$ and $n$ and band $n$ have same remainder.

$\mathbb{Z}_n$ = quotient set of $n$ classes of equivalence congruent to $0, 1 \dots n-1 \rightarrow -1 \equiv n-1 \pmod n$, $-2 \equiv n-2 \pmod n$ ...

- if $a \equiv b \pmod n \rightarrow a+c \equiv b+c \pmod n$
- if $a \equiv b \pmod n \rightarrow a \cdot c \equiv b \cdot c \pmod n$
- if $a \equiv b \pmod n \rightarrow a^c \equiv b^c \bmod (n)$

## ORDER OF ELEMENTS

$a^n$ denotes $a+a \dots +a$ ($n$ times, $aaa \dots$ if multiplicative operator). $a$ is of order $n$ if $a^n = 0$ and $\forall m < n \; a^m \neq 0$

($a^n = 1$ for multiplicative operator) → all elements of finite group have finite order.

$\mathbb{Z}_m$ = natural numbers mod $n$ = classes of equivalence of congruent integer (mod n)

$\mathbb{Z}_m^*$ = natural numbers mod $n$ prime (co-prime) to $m$ (multiplicative group of $\mathbb{Z}_m$)

$\Phi_m$ = Euler's totient function $= |\mathbb{Z}_m^*|$

→ Euler th.: $\forall a \in \mathbb{Z}_m^* \quad a^{\Phi(m)} = 1 \bmod n \rightarrow a^{k\Phi(m)+1} = a \bmod m \quad k \geq 0$

If $a \in G$ and $\langle a \rangle = \{1, a \dots a^{n-1}\}$ sub-group of $G$ then $a$ = generator of $\langle a \rangle$, $G$ = cyclic group and $a$ = primitive element of $G$

Th.: $\forall p$ prime the multiplicative group of $\mathbb{Z}_p$ ($\mathbb{Z}_p^*$) is cyclic

A set $F$ with two binary operation $+$ and $\cdot$ is called commutative ring with identity if:

① $\forall a,b \in F \quad a+b \in F$
② $\forall a,b,c \in F \quad (a+b)+c = a+(b+c)$
③ $\forall a,b \in F \quad a+b = b+a$
④ $\exists 0 \in F \; \forall a \in F \quad a+0 = a$
⑤ $\forall a \in F \; \exists -a \in F \quad a+(-a) = 0$
⑥ $\forall a,b \in F \quad a \cdot b \in F$
⑦ $\forall a,b,c \in F \quad (a \cdot b) \cdot c = a \cdot (b \cdot c)$
⑧ $\forall a,b \in F \quad a \cdot b = b \cdot a$
⑨ $\exists 0 \in F \; \forall a \in F \quad a+1 = a$
⑤ $\forall a,b,c \in F \quad a \cdot (b+c) = a \cdot b + a \cdot c$

Its called field if we add relation ⑪ $\forall a \neq 0 \in F \; \exists a^{-1} \in F \quad a \cdot a^{-1} = 1 \rightarrow$ each non zero element has a multiplicative inverse

Let $f(x) = a_n \cdot x^n + a_{n-1} x^{n-1} \dots a_1 x + a_0$ over a field $F$ ($a_n, a_{n-1} \dots a_0 \in F$) → $f(x)$ has at most $n$ solutions in $F$

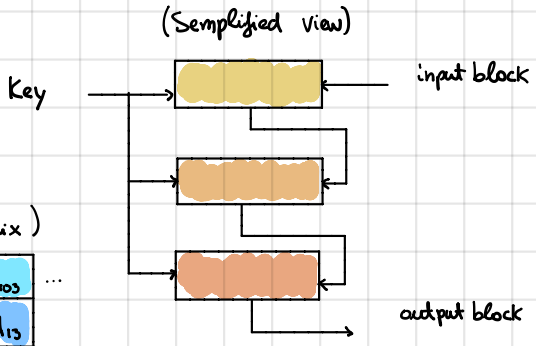
Let $f(x)$ and $g(x) = b_m x^{m-1} \dots b_0 \quad (m \leq n) \rightarrow f(x) = h(x) \cdot g(x) + r(x)$ → $r(x)$ = remainder of $f(x)$ modulo $g(x)$ and its unique.

$(F,+,\cdot)$ is finite field if set $F$ is finite.

GALOIS FIELD $GF(p^k)$: $\forall$ prime power $p^k$ there is a unique finite field containing $p^k$ elements denoted as $GF(p^k)$. No finite fields with other cardinalities

Let $f(x)$ irreducible polynomial of degree $k$ over $\mathbb{Z}_p$ → $GF(p^k)$ can be realized as the set of $k-1$ polynomials over $\mathbb{Z}_p$ with $+$ and $\cdot$ done modulo $f(x)$

# RIJNDAEL - AES




128 bits input / output block → in 4×4 matrix of bytes

The matrix first contains plaintext then "converted" to ciphertext (some matrix)

Key (128, 196, 256..) in a matrix of 4 × n/32 bytes

AES (plaintext, Key)

- Key Expansion = Key expanded to 10 round keys (of key-length)
- AddRoundKey (State, ExpandedKey[i])
- Cycle (i: 0→R) Round (State, ExpandedKey[i])
- FinalRound (State, ExpandedKey[R])

(plaintext matrix)

| | | | |
|---|---|---|---|
| $A_{00}$ | $A_{01}$ | $A_{02}$ | $A_{03}$ |
| $A_{10}$ | $A_{11}$ | $A_{12}$ | $A_{13}$ |
| $A_{20}$ | $A_{21}$ | $A_{22}$ | $A_{23}$ |
| $A_{30}$ | $A_{31}$ | $A_{32}$ | $A_{32}$ |

Every round has 4 steps:

1. Substitution: $A_{i,j} = A_{i,j}^{-1}$ → multiplicative inverse in $GF(2^8)$ highly non linear. If $A_{i,j} = 0$ dont change. Invertible step.
2. Shyft Row: first row no shift, second row 1 shift, third row 2 shift ... . Invertible step.
3. Every column is considered as a polynomial over $GF(2^8)$ → multiply with an invertible polynomial $03x^3 + 01x^2 + 01x + 02 \pmod{x^4+1}$
4. XOR with round key

For algorithm demo: https://www.youtube.com/watch?v=gP4PqVGudtg

# DATA INTEGRITY

(Suppose we don't care anymore of confidentiality). Message received = message sent even with active adversary in the middle.

A = authentication algorithm paired to Verification algorithm V (accept/reject). Key shared

Message $m = (m, A_K(m))$ pair with $A_K(m)$ = authentication tag

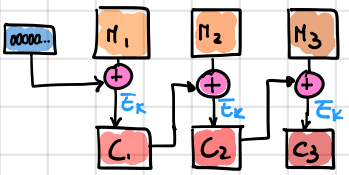
$V_K(m, A_K(m))$ = accept . A is usually called Message Authentication Code → $A_K(m)$ denoted as $MAC_K(m)$ → Verification by executing authentication on $A_K(m)$ and comparing with $MAC_K(m)$

Attacker cant (shouldnt) construct legal pair $(m, MAC_K(m))$ even after observing many legal pairs, output short, MAC function not 1-to-1 (more messages with same auth. tag.

EXISTENTIAL FORGERY: attack that produces a legal pair message - authentication tag even if meaningless for the receiver.

Given n legal pairs $(m_n, MAC_K(m_n))$ find a new legal pair $(m, MAC_K(m))$ efficiently and without negligible probability.

## MAC on CBC




- All zero seed.
- Discard first n-1 blocks
- Send $m_1 \dots m_n$ and $C_n$ as authentication tag $MAC_K(M) = C_n$

Note: we dont need to secret the message ONLY INTEGRITY

Pseudo Random function: function that looks random → if $E_K$ is a pseudo random function then the fixed length CBC MAC is resilient to forgery.

Insecure for variable length messages.

To have confidentiality and data integrity: produce $MAC_{K_1}(M)$ with CBC, produce n cipherblocks with $K_2$ → send cipherblocks and $MAC_{K_1}(M)$

# HASH FUNCTION

Map large domains (messages) to smaller (keys). MAC must be a function of message and key. eg. $MAC_K(m) = h(k||m)$ v $h(k||m||k)$... [|| = concatenation]

Collision Resistancy: $h: D \to R$ is weakly collision resistant if its hard to find $x' \neq x$ such that $h(x') = h(x)$ for $x \in D$. its strongly collision resistant if its hard to find $x, x'$ such that $x' \neq x$ but $h(x') = h(x)$ . STRONG ⇒ WEAK WEAK ⇏ STRONG

Pre-Image Resistancy: given authentication tag is hard to find the message i.e given $h(x)$ is hard to find $x$. Second Pre-Image resistancy if its hard to find a collision to $x$ given $h(x)$

Summary: why hash function is cryptographic secure?

- strong collision resistance: hard to find pair $x \neq x'$ such that $h(x) = h(x')$ ⇒ weak collision resistance
- Pre-image resistance: given $h$ is hard to find ONLY $x$ s.t. $H(x) = h$
- Second Pre-image resistance: given $X$ is hard to find $x' \neq x$ such that $h(x') = h(x)$ (collision)

COLLISION ATTACK: attacker (man in the middle) sends a different message that still match authentication tag.

## BIRTHDAY PARADOX

Given 23 persons 2 of them have same birthday with probability $Pr > 1/2$ → thinking of it as an hash function is not so hard to find a collision

This "paradox" affects ALL hashing function, even the good ones. Anyway generating a message that collides is still difficult.

The probability depends only on range(R) size! → e.g. $(1.1774|R|)^{1/2}$ elements of D → $Pr = 0.5$

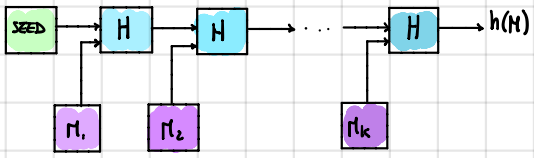
→ BIRTHDAY ATTACK: given $f$ find $x_1 \neq x_2$ such that $f(x_1) = f(x_2)$ → e.g. 64bit hash means $18 \times 10^{19}$ different output → $5.38 \times 10^9$ attempts to generate a collision using brute force ($5.1 \times 10^9$ makes the probability to 0.5) → # of attempts = birthday bound ~ (asintotically) $\sqrt{|R|}$

ATTACK EXAMPLE:

- generate "many" copies of message with small variation like adding spaces N "equivalent" copies.
- generate N copies of the fake message
- pick at random pair from edited and fake messages. if hash has small numbers of bit it will be easy to find a match

Hash function have key (Keyed) or not (Unkeyed)

To deal with longer strings:



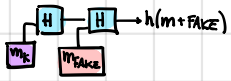

**Merkle - Damgård construction**

$h \neq H$, $H: D \to R$. Using this architecture (and H good) we create a cryptographically secure $h$.

If H is collision resistant then $h$ will be too

**Keyed hashing** = $MAC_K(m)$ = hash of key and message guarantees also **authenticity** cause only the ones with the key can compute the hash

Prepending key to message $(K \| m)$ is a bad choice with previous architecture cause attacker could append a new block to $m_K$ and generate a valid authentication tag without knowing the key.




APPEND KEY: ✗

case: key length = block length : attacker focus on $m - m_{K-1}$ blocks and try to find a colliding message **small vulnerability** (lot of work for attacker)

PREPEND & APPEND : $h(k|m|k)$, no vulnerability found. ✓

Modern hash functions compute more round for each block in which they swap, shuffle and do other satanic things to messages bits.

## HMAC

$HMAC_K(m, h) = h(K \oplus opad \| h(K \oplus ipad \| m))$ opad = outerpad = 0x5c5c... ipad = innerpad = 0x3636... both one-block long

HMAC can be forged only if $h$ can be forged (by attackers)

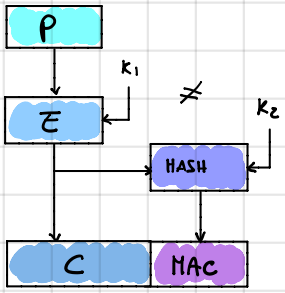
# AUTHENTICATED ENCRYPTION (AE)

Confidentiality + authenticity

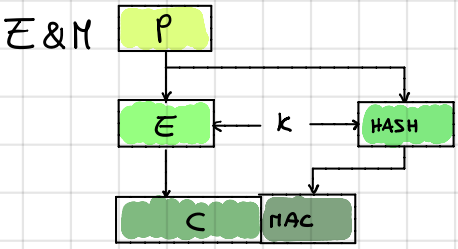
① Encrypt-then-MAC (EtM) ② Encrypt and MAC (E&M) ③ MAC then Encrypt (MtE)

↳ most secure!

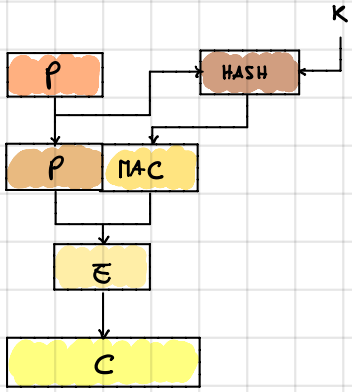
EtM




E&M




MtE




SALT: random string concatenated to (usually) a password to have a more robust password.

# PUBLIC-KEY CRYPTOGRAPHY

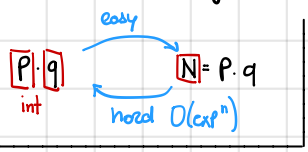
New method. Secret-Key cryptography is still new and good anyway (simmetric / asimmetric)!

The problem with simmetric encryption is that the key must be secretly generated and exchanged a prior in a unsecure channel!

DIFFIE & Hellmon: Split key K in two: $K_E$ used to encrypt, $K_D$ to decrypt, $K_E$ must be public → public-key asymmetric.

The sender will encrypt messages with receiver's public key ($K_E$) that will decrypt with his private key ($K_D$).

The owner of private key can encrypt with his own secret key so everybody can decrypt using the public key. In this way we dont have confidentiality but we gain authentication cause only the owner of private key could have encrypted the message.




DISCRETE LOG: given $y = g^x$ with $x$ minimal non negative integer satisfying the equation. $x$ is called discrete log of $y$ to base $g$. ($g$ generator of $G$ = finite cyclic group of $n$ elements $\wedge$ $y \in G$)

$\longrightarrow y = g^x \bmod p$ in multiplicative group of $\mathbb{Z}_p$

$x \to g^x \bmod p$ is easy with quick exponentiation but hard to reverse ($g^x \bmod p \to x$) $\Rightarrow x \to g^x \bmod p$ is believed to be a one way function!

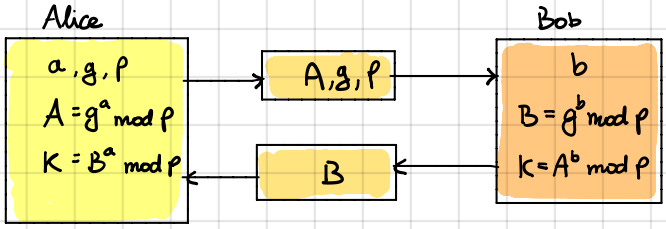
$f: \{0,1\}^* \to \{0,1\}^*$ is one-way if $f$ can be computed in polynomial time, but for every polinomial time algorithm (even randomized) $A$, for every polynomial $p(n)$ and all sufficiently large $n$: $P_r[f(A(f(x))) = f(x)] < 1/p(n) \Rightarrow$ hard to reverse in the average-case!

These fucking functions are just theoretical, conjectured to exist! 😈

PUBLIC EXCHANGE OF KEY: two parties who dont share any secret information, perform a protocol and derive the same shared key.

DIFFIE-HELLMAN KEY EXCHANGE (DH): Alice — BOB

- Public parameters: prime $p$ and an element $g$ (ponibly generator of $\mathbb{Z}_p^*$). $p$ shold be a safe prime → $p = 2q+1$ with $q$ prime too.
- Alice chooses "a" at random in $[1 \dots p-1]$ and sends $g^a \bmod p$ to Bob. "a" secret.
- Bob chooses "b" at random in $[1 \dots p-1]$ and sends $g^b \bmod p$ to Alice. "b" secret
- Both compute $g^{ab} \bmod p$ as shared key. (eg. Alice: $(g^b)^a \bmod p = g^{ab} \bmod p$)



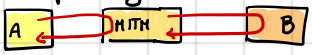

DH can be broken only breaking DL in polynomial time.

DH can be weak if bad implemented

Perfect Forward Secrecy: generate random public key per session for the purpose of key agreement and dont use any sort of deterministic algorithm in doing so. → compromise a shared key doesnt compromise other shared key. Session key and Forward Secrecy

Eg. given $N \to h(N)$ then $h(h(N))$ for next session and so on. Compromising $h(N)$ will compromise only that session.

Non-in-the-middle can succed if there is no authentication




Let $p$ and $q$ be 2 large primes. $N = p \cdot q$

Multiplicative group $\mathbb{Z}_N^* = \mathbb{Z}_{pq}^*$ contains all integers in $[1, pq-1]$ that are prime with $p$ and $q$

In $[1, pq-1]$ there are $(p-1)$ multiples of $q$ and $(q-1)$ multiples of $p$. The size of the group is

$\phi(pq) = pq - 1 - (p-1) - (q-1) = pq - (p+q) + 1 = (p-1)(q-1)$ so by Euler's th. $\forall x \in \mathbb{Z}_{pq}^*$, $x^{(p-1)(q-1)} = 1$

Exponentiation in $\mathbb{Z}_{pq}^*$: we wont to exponentiate for encryption. Not all integers in $[1, pq-1]$ belong to $\mathbb{Z}_{pq}^*$

Given $1 < e < (p-1)(q-1)$ when exponentiation to $e$-th power a one-to-one operation in $\mathbb{Z}_{pq}^*$? e.g. $x \to x^e$ with $x$ = plain text

$\Rightarrow$ if $e$ is relatively prime to $(p-1)(q-1)$ then $x \to x^e$ is a one2one operation in $\mathbb{Z}_{pq}^*$

# RSA Public Key Cryptosystem

- $N = pq$
- Choose $1 < e < \phi(N)$ s.t. $\gcd(e, \phi(N)) = 1$ gcd = greater common divisor
- $d$ s.t. $d \cdot e = 1 \bmod \phi(N)$
- Public key = $(e, N)$
- Private key = $(d, N)$

⇒ Encryption of $M \in \mathbb{Z}_N$ by $C = E(M) = M^e \bmod N$ Note different from DH solution to key distribution
⇒ Decryption of $C \in \mathbb{Z}_N$ by $M = D(C) = C^d \bmod N$

RSA is mostly used to encrypt a secret simmetric key. Not used in practice for long messages.

## EUCLID ALGORITHM → GCD(x,y)

eg $X = 595$ $Y = 408$ $\gcd(595, 408)$:

- $595 = n \cdot 408 + \text{Rest}$ → $= \gcd(408, \text{Rest}) = \gcd(408, 187)$ $n = 1$
- $408 = 2 \cdot 187 + 34$ → $= \gcd(187, 34) =$
- $187 = 5 \cdot 34 + 17$ → $= \gcd(34, 17) =$
- $34 = 2 \cdot 17 + 0$ → $\gcd(17, 0)$ ⇒ $\gcd(595, 408) = 17$

Let $p = 47$ $q = 59$ $N = pq = 2773$ $\phi(N) = 46 \cdot 58 = 2668$ → choose $d = 157$

Bezout's identity: $xu + yv = d$ with $x \geq y$, $u$ and $v$ signed integers and $d = \gcd(x, y)$. $x$ and $y$ are given!

If $x$ and $y$ are coprime → $d = 1$ → $u$ = multiplicative inverse of $x \pmod y$ and $v$ = multiplicative inverse of $y \pmod x$

While calculating gcd with Euclid algorithm we construct the following table:

| n | $q_n$ | $r_n$ | $u_n$ | $v_n$ |
|---|---|---|---|---|
| -2 | / | 797 | 1 | 0 |
| -1 | / | 1047 | 0 | 1 |
| 0 | 0 | 797 | 1 | 0 |
| 1 | 1 | 250 | -1 | 1 |
| 2 | 3 | 47 | 4 | -3 |
| 3 | 5 | 15 | -21 | 16 |
| 4 | 3 | 2 | 67 | -51 |
| 5 | 7 | 1 | -490 | 373 |

Compute multiplicative inverse of 797 mod 1047

$r_{n-2} / r_{n-1} = q_n \cdot r_{n-1} + r_n$ $u_n = u_{n-2} - q_n \cdot u_{n-1}$ $v_n = v_{n-2} - q_n \cdot v_{n-1}$

→ $\left[\frac{797}{1047}\right] = 0$ + rest = 797 $u_n = 1$ $v_n = 0$

→ $\left[\frac{1047}{797}\right] = 1$ + rest = 250 ...

⇒ multiplicative inverse → $1 = -490 \cdot 797 + 373 \cdot 1047$ ⇒ 557 mod 1047

1047 − 490 = 557

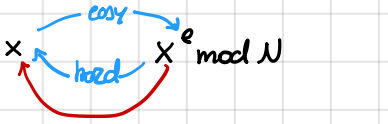
gcd = 1 → multiplicative inverse exists!

We can use this method to calculate gcd even if not equal to 1

## Constructing an instance of RSA

- Alice picks at random coprimes $p$ and $q$ (large) $\rightarrow N = p \cdot q$
- Alice picks at random a large $d$ relatively prime to $\phi(N) = (p-1)(q-1) \rightarrow \gcd(d, \phi(n)) = 1$
- Alice computes $e$ s.t. $d \cdot e = 1 \bmod \phi(n)$
- Alice publish public key $(N, e)$ and keeps secret everything else ($(N, d)$ = private key)

→ Large coprime $p$ and $q$: random, hard to guess. In $[N, 2N]$ there are $\sim N / \ln N$ primes
→ we expect to find a prime of $N$ bits every $\ln N$ attempts.
→ $e$ suitable to have fast encoding: usually the min value avaible → 3




Easy with trapdoor info (d)

$f: D \rightarrow R$ is a trapdoor one way function if there is a trapdoor $s$ s.t.
① without knowledge of $s$, $f$ is a oneway function
② Given $s$, inverting $f$ is easy.

## Problems:

① If $m = 0, 1$ or $N-1 \rightarrow RSA(m) = m \Rightarrow$ zero, use salt
② If $m$ and $e$ small we might have $m^e < N \rightarrow m^e \bmod N = m^e \Rightarrow$ add non zero bytes to avoid small messages

SALTING: add extra bits to original message and then encrypt → how? On decrypt we should be able to distinguish original message from salt.

Robustness of RSA doesn't mean always robust
Attacker can factor $N = pq$ if $p$ and $q$ have bad properties. Recommended to choose them
- large (100 digits)
- not close together
- $(p-1)$ and $(q-1)$ have large prime factors (to foil Pollard's RHO algorithm)

Some messages can be easy to decode (e.g ciphertext = plaintext, particular case)
Note: if we factor $N$ we break RSA.

③ If attacker has two encoding of two similar messages s.t. $c_1 = m^3 \bmod N$ and $c_2 = (m+1)^3 \bmod N$
→ $m = (c_2 + 2c_1 - 1) / (c_2 - c_1 + 2)$ ; or
$m$ and $(am+b)$ similar → more complex but similar to previous one
⇒ choose large $e$ (at least 5)

④ $e = 3$ and send message to three different users with public keys $(3, n_1)$ $(3, n_2)$ $(3, n_3)$
Attacker knows public keys and $m^3 \bmod n_1$, $m^3 \bmod n_2$ $m^3 \bmod n_3$
He can compute with Chinese Remainder Theorem $m^3 \bmod (n_1 \cdot n_2 \cdot n_3)$
$m < n_1, n_2, n_3 \rightarrow m^3 < n_1 \cdot n_2 \cdot n_3$ and $m^3 \bmod n_1 \cdot n_2 \cdot n_3 = m^3 \rightarrow \sqrt[3]{m^3} = m$
⇒ Add random bytes to avoid equal messages

⑤ If message space is small, attacker can try all possible messages
⇒ Add random string in message

⑥ If two users have same $N$ but different $e$ and $d$ → same $p$ and $q$
One of the two could compute $p$ and $q$ from his $e, d, N$ (tricky but possible) then can easily derive secret key of other user given his public key
⇒ Each person choose his own $N$ (prob. to have same $N$ are very very low)

⑦ Multiplicative property of RSA
if $M = M_1 \cdot M_2 \rightarrow (M_1 \cdot M_2)^e \bmod N = ((M_1^e \bmod N) \cdot (M_2^e \bmod N)) \bmod N$ so an attacker can proceed using small messages
⇒ padding on small messages

⑧ Choosen Ciphertext Attack
Attacker wants to decrypt $C = M^e \bmod N$
- computes $X = (C \cdot 2^e) \bmod N$
- uses $X$ as choosen ciphertext and ask oracle for $Y = X^d \bmod N$
but $X = (C \bmod N) \cdot (2^e \bmod N) = (M^e \bmod N) \cdot (2^e \bmod N) = (2M)^e \bmod N$ → he got $Y = 2M$

Attacker T knows $C = M^e \bmod N$, randomly choose $X$ and computes $c' = cX^e \bmod N$ and ask oracle to decode $c'$. T computes $(c')^d = c^d (X^e)^d = MX \bmod N$
⇒ require that $M$ needs to verify some requirements to have the oracle answering

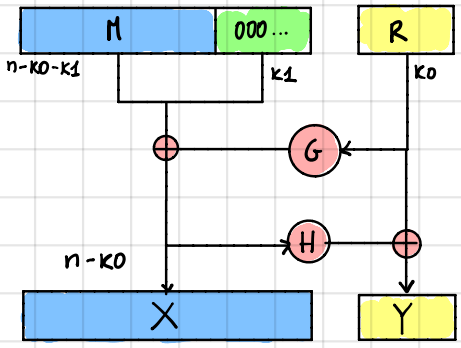
RSA conclusion: given $M$, preprocess it to obtain $M'$ and then apply RSA.

PKCS: public key Cryptography Standard → $m = 0 \| 2 \|$ at least 8 non zero bytes $\| 0 \| M$ ($M$ original message)
0 = message less than $N$ (first byte)
2 = denotes encrypting of a message and $m$ not small (1 denotes encrypting of dig. signature)
random bytes = same message to many people, space message large.
$m$ length ~ $M$ length!

OAEP: padding scheme often used with RSA encryption. = optimal asymmetric encryption padding
It uses a pair of random oracles G and H (hash functions) to process the plaintext prior to asymmetric encryption. The goal is to add randomness used to convert deterministic encryption scheme into a probabilistic one and prevent partial decryption ("all or nothing")

$n$ = # bits in RSA modulus
$k0$ and $k1$ = integers fixed by protocol
$M$ = plaintext, $(n - k0 - k1)$-bit string
G / H = hash functions fixed by protocol

ENCODE:
- message padded with $k1$ zeros → now $n - k0$ length
- $R$ = $n - k0$ bit string
- G expands $k0$ bits of $R$ to $n - k0$ bits
- $X$ = M000... xor G(R)
- H reduces the $n - k0$ bits of $X$ to $k0$ bits
- $Y$ = R xor H(X)
- output $X \| Y$




DECODE
- recover random string $R = Y$ xor $H(X)$
- recover message M000... = X xor G(R)

## RSA Encryption Schema - OAEP

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

OS2IP: a non negative integer is constructed by interpreting $k$ octets = 8-bits bytes as number in base 256. Each octet is a digit (in $[0,255]$). $k$ = # bytes for representing $N$

I2OSP: An octet stream (array of $k$ bytes) is constructed by determining the coefficient of representation in base 256 of the given integer. Each coefficient is an integer in $[0,255]$ and is stored in one octet

## ELGAMAL ENCRYPTION

Based on DH. 3 components: key generator, encryption and decryption algorithms.
Alice publish $p, g \in Z_p$ as public parameters ($g$ generator of cyclic group of order $p$)
Alice chooses $x$ as private key and publish $g^x \bmod p$ as public key ($x$ random in $[0, p-1]$)
Bob to Alice Encryption of $m \in Z_p$ by sending $(g^y \bmod p, mg^{xy} \bmod p)$ ($y$ random in $[0, p-1]$)
Decryption: Alice computes $(g^y)^x \bmod p = g^{xy} \bmod p$, then $(g^{xy})^{-1} \bmod p$ for obtaining $mg^{xy}(g^{xy})^{-1} \bmod p = m$
Requires two exponentiations per each block transmitted
Involved not trivial math.

# DIGITAL SIGNATURES - DSA

HMAC/MAC can be generated by both Alice and Bob → we can have repudiation (Both says that they didnt send the message)

$M \longrightarrow S(M)$

! → identify signer
→ use Public Key Infrastructure

Alice wants to produce a pair $(M, S)$ that everybody can verify.
The signature will identify both the signer and the message !
→ change message/signer = change signature → cant copy signature

## Candidate 1

$M: (M, Enc(M)) = (M, S)$ → with Alice private Key. → Bob receives $(M, S)$:

verify Dec. with public key of Alice and compare result with received M

This method is vulnerable to forgery attack: attacker is able to forge a pair $(M, S)$ that pass the verification.

EXISTENTIAL forgery:

① Generate random file R
② Construct $T = Dec_{Pub(A)}(R)$
③ Send $(T, R)$

Verify ⇒ $Dec_{Pub(A)}(R) = T$ equal to T in $(T, R)$ → ✓

## Candidate 2

$(M, Enc_{Priv(A)}(H(M)))$: use any cryptographically secure hash function unkeyed

Verification upon receival:

$(M, S)$ → compute $H(M)$ ✓ → $Dec_{Pub(A)}(S) = H(M)$ ✓ → compare the two $H(M)$ ✓

For existential forgery the attacker should send $(T', R)$ where $T = H(T')$ × → hard if H cryptosecure.

This is the Official definition of digital signature.

## SELECTIVE forgery

Attacker has the ability to choose M prior to the attack. Selective forgery ⇒ Existential forgery

## UNIVERSAL forgery

Attacker can produce a valid signature for any given message → Selective → Existential

¬ Existential ⇒ ¬ Selective ⇒ ¬ Universal . Preventing Existential forgery we prevent them all.

Practically Alice sends (M, S, certificate) where the certificate guarantee that the public key is associated to Alice. Used schemes are RSA, El-Gamal signature scheme and Digital Signature Standard (modification of El-Gamal)

## RSA

Signature is a code hash of message using private key → only who know private key can sign. Every body can verify using public key.

Public Key Crypto Standard: signature is called digest

$0 \| 1 \|$ at least 8 bytes FF base 16 $\| 0 \|$ specification of used hash function $\|$ hash(M)

- 0: encoded message less than n
- 1: denotes signature (if 2, encoding)
- FF base 16: 11111111 = encoded message large
- specification of used hash function: increase security

SSA: signature scheme with Appendix, the appendix is the signature added to the message.

# ELGAMAL SIGNATURE SCHEME

In addition to private/public key pair, will be generate a second temporary pair for digital signature.

$p$ - prime of 1024 bits which discrete log hard in $\mathbb{Z}_p^*$ $g$ = generator of $\mathbb{Z}_p^*$ $x$ = random $[2, p-2]$ $y = g^x \bmod p$

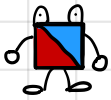
→ Public key = $(p, g, y)$, Private key = $x$ $p$ and $g$ not always in public key but always not secret

Plus for signing $M$:

$m = H(M)$, $K$ = random $[1, p-2]$ relatively prime to $p-1$, $r = g^k \bmod p$ ← not depending on $M$

$s = (m - xr) K^{-1} \bmod (p-1)$ (if $s=0$ restart)

⇒ Signature $= (r, s)$



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

Compute $m = H(M)$. Accept if $(0 < r < p) \wedge (0 < s < p-1) \wedge (y^r r^s = g^m) \bmod p$

# DIGITAL SIGNATURE STANDARD (DSS)

Inspired by Elgamal and uses SHA as hash function and DSA (algorithm) as signature

$p$ = L bit prime s.t. discrete log mod $p$ is hard. $q$ = 160 bit prime divides $p-1$: $p = Jq + 1$

$\alpha$ s.t. $\alpha^q = 1 \bmod p$

↳ $h$ = random number s.t. $1 < h < p-1$ $g = h^{(p-1)/q} \bmod p = h^J \bmod p$ (if $g=1$ try again, too insecure)

$\Rightarrow g^q = h^{p-1}$

Fermat's theorem: $h^{p-1} = 1 \bmod p$ → $p$ is prime → $\alpha = g$

DSA: $p$ prime, $q$ prime, $p - 1 = 0 \bmod q$, $\alpha = 1^{1/q} \bmod p$

Private key = secret $s$ between 1 and $q-1$ random

Public key = $(p, q, \alpha, y = \alpha^s \bmod p)$

Signing $M$:

- choose random $1 \leq k \leq q-1$ secret
- $P_I = (\alpha^k \bmod p) \bmod q$
- $P_{II} = (SHA(M) + s \cdot P_I) K^{-1} \bmod q$

} Signature $(P_I, P_{II})$

$P_I$ independent from $M$,

Verify:

$e_1 = (SHA(M) \cdot P_{II}^{-1}) \bmod p$

$e_2 = (P_I \cdot P_{II}^{-1}) \bmod q$

} Accept if: $(\alpha^{e_1} y^{e_2} \bmod p) \bmod q = P_I$ Proof on slides!

Finding $p$ and $q$ is hard so $p$ and $q$ are generally public

Slower for verification than RSA but faster in signing with preprocessing.

Both admissible.

# TIMESTAMPING DOCUMENT

Timestamping Authority (TSA) guarantees timestamp of a document.

Alice wants to timestamp her doc.

- H(doc) to TSA
- TSA add timestamp, $H(H(doc) + t)$ and signs it, then sends it to Alice
- Alice keeps TSA's signature as proof.

Everybody can verify it, TSA doesn't know Alice's doc, only its hash!

## TRUSTED PARTY

KDC = Key Distribution Center

Not possible to have a secret key for every pair of users.

TRUSTED PARTY: every user share a key with the trusted authority C (authentication server) Note: "user" can be printer, database etc that need authentication.

Eg. A and B share a key with C → $K_{AC}$ and $K_{BC}$

Goal: A and B must authenticate and choose a secret session key K used for short time (session) known only by A, B and C

① A sends to C: $[A, B]$

② C chooses K session key and sends to A: $[K_{AC}(K), K_{BC}(K)]$

③ A decodes and compute K and sends to B: $[C, A, K_{BC}(K)]$

④ B decodes $K_{BC}(K)$ finding K and sends to A $[K(\text{Hello A, Im B})]$ ⟶ now both they have K.


can intercept messages and sustitute C with himself!

1) A: $[A,B]$ to T instead of C
2) T: $[A,T]$ to C
3) C: $[K_{AC}(k), K_{TC}(K)]$
4) A: $[C, A, K_{TC}(K)]$ to B → T intercepts
5) T: $[K(\text{Hello A, Im B})]$ to A ⇒ A thinks T is B ⇒ solve with A: $[A, K_{AC}(B)]$ to C to hide the destination to T

⇓

Still can be attacked: T intercepts at ① and send to C $[A, K_{AC}(T)]$ T doesn't know the real destination.
A will send to B: $[C, A, K_{TC}(K)]$ ⟶ T find that A wants to talk to B and then it acts like B: $[K(\text{Hello A, Im B})]$ to A
Now T knows B.

Protocol Needham - Schroeder: uses "nonce" = random numbers.

1) A chooses N (nonce) and sends $[A, B, N]$ to C
2) C chooses K and sends $[K_{AC}(N, K, B, K_{BC}(K, A))]$ to A
3) A decodes and checks N and B and sends $[K_{BC}(K, A)]$ to B
4) B decodes, chooses N' and sends $K(\text{this is B}, N')$ to A
5) A sends $[K(\text{this is A}, N'-1)]$ to B ⟶ B now has checked A!

⇓

Attack: At point ② T acts as A, at ③ T sends $[K_{BC}(K', A)]$ to B instead of $[K_{BC}(K, A)]$ where K' = old session key.
B responds, T intercepts $[K'(\text{this is B}, N')]$ and T sends $[K'(\text{this is A}, N'-1)]$ to B
Note: T must have K' and B never talk to C!

To guarantee data integrity use timestamps, sequence numbers or nonce (nonce should be used carefully)
Save nonce only for limited time → can't reuse.

KERBEROS: provides authentication in distributed systems
Scenario: A needs to access service of B ⟶ Need authentication of A (optional: auth. B and generate session key), use C ⟶ Idea = use ticket to access services, valid for limited time.
Each user is called principal and has a secret key shared with KDC stored in KDC and encrypted with KDC's key.
Kerberos makes use of "tickets": a message encrypted with destination (B) secret key, containing the key that will be shared between A and B, A's identity eg. $[K_B(K_{AB}, \text{"Alice"}...)]$. Of course the ticket is "generated" by KDC.
B can be a service too. Ticket has a lifetime (duration of permission of access) and timestamp to avoid reuse.

Messages and requests from A to KDC must be encrypted each time with $K_{AC}$ → retype everytime or store it (Never!) → inadequate

⇒ Ticket-granting Ticket (TGT): use a metaticket used to ask the real ticket.

- At first login A receive a session key $S_A$ by KDC, with fixed lifetime (few hours), and other infos.
- Next requests from A to KDC will contain TGT
- No need to store "real" password.

Problems:

- different servers must have synchronized clocks cause of timestamp system to avoid replay attack.
- authenticator does not guarantee data integrity
- many instances of some servers use some master key → replay attack.

Generally there are many instances of some KDC with some DB, with a single one that store infos and then updates others periodically.

In large networks systems, the network is divided in subnetwork called realms. Each of them has different KDC master with some master key. Different KDC in different realms have different users databases. Anyway every user can talk with users in different realms just talking first with the specific KDC of the destination realm.

## AUTHENTICATION USING PUBLIC KEY (Needham - Schroeder)

Idea: sign messages combining timestamp with public key → problem: where to store the key?

⇒ third trusted party C

How? A want to talk with B → tell it to C that will responds with $\langle B, sign\_C(K_{PB}, B), K_{PB} \rangle$

A check the signature, generate Nonce and sends to B $\langle K_{PB}(N, A) \rangle$

Now B wants to check A's identity → $\langle B, A \rangle$ to C → C to B $\langle A, K_{PA}, sign\_C(K_{PB}, A) \rangle$

B check signature, retrieve $K_{PA}$, generate N' and sends to A $\langle K_{PA}(N, N') \rangle$. A check N and sends to B $\langle K_{PB}(N) \rangle$

Attack: T can talk with A, B, C with authentication between A and T (R1) and between T (as A) and B. (R2)

In R1 we have $K_T(N, A)$. In R2 $K_{PB}(N, A)$ and then B sends to T $K_{PA}(N', N)$. T → A $\langle K_{PA}(N', N) \rangle$ in R1 and then A → T $K_T(N')$. In R2 T → B $\langle K_{PB}(N') \rangle$ ⇒ Now BOB thinks he is talking to A!

⇒ Fix: B sends to A $\langle K_{PA}(B, N, N') \rangle$ and not only $\langle K_{PA}(N, N') \rangle$ avoiding T to use it and send it to A.

# PUBLIC KEY INFRASTRUCTURE (PKI)

Certificates are issued by a trusted Certification Authority (CA) for all users in domain. If A wants B's public key it ask to CA that responds with the key signed with its own private key → A must know only one key: CA's publickey.

## X.509

- Serial Number: unique in each CA's domain
- Issuer: the X.500 name of the issuing CA. (country, organization etc)
- Validy: time interval of duration for the certifier. Not valid for signing after expiration, but valid to verify old signatures.
- Subject: X500 name of who own the public key.
- SubjectPublicKeyInfo: algorithm identifier used for the key and subject's public key.
- and many more...

NB. Impossible to falsify. If A and B share some CA they can know each other public key. Otherwise CA form a hierarchy. → 3 levels

Level 1 = root certification authority → self signing

## REVOCATION (before deadline)

Reasons:

① User's secret key not safe anymore or lost
② User is not certified by CA
③ CA's secret key compromised.

After certificate check, a revocation check must be done.

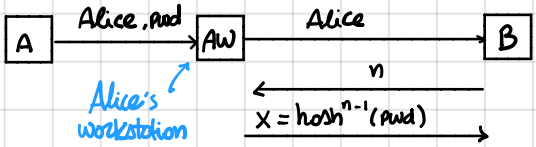
Exp. date → Digital Signature → revocation check

ONLINE CERTIFICATE STATUS PROTOCOL used to get revocation status of X.509 certificates.

# PASSWORD

To avoid eavesdropping and the possibility that an attacker broke the password database we store data obtained from password (hash). User password is attached to salt to avoid dictionary attack and then converted to a 56 bit secret key K and then store $DES_K(DES_K(\dots 25 \text{ times } (data)\dots))$
For authentication we must avoid eavesdropping, REPLY and dictionary attacks → strong protocols: cryptographic authentication with user remembering only personal password

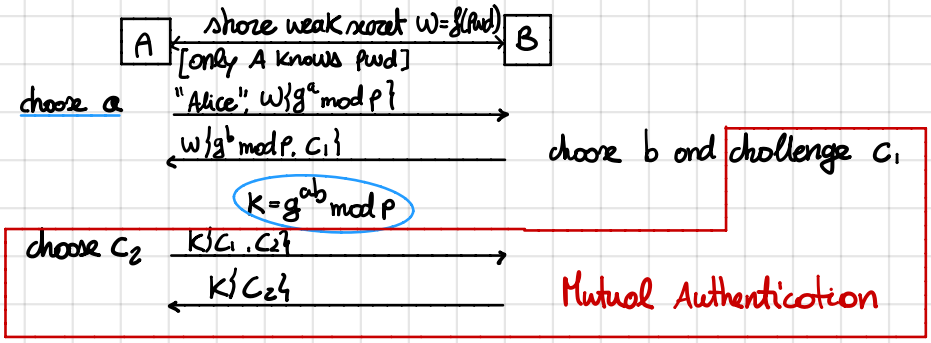
## LAMPORT'S HASH




Bob stores <username, large number n, $h^n$(password)>
Alice's workstation sends $X = h^{n-1}$(password)
Bob computes h(X): if successful, n is decremented, x replaces $h^n$ in Bob's database
In addition we can add salt to password, stored at Bob's setup time communicating to Alice to use the same password with different servers (different salts)
Small n attack: Trudy impersonates Bob and sends $n' < n$ and salt, when she gets the reply she can impersonate Alice until $n = n'$.
Defense: AW shows submitted n to A to verify the approximated range → A needs to remember it.
"Human and paper" env.: if AW is insecure or too dumb to do hashing, A is given a list of all hashing from n and she use them exactly once each. preventing small n attack.

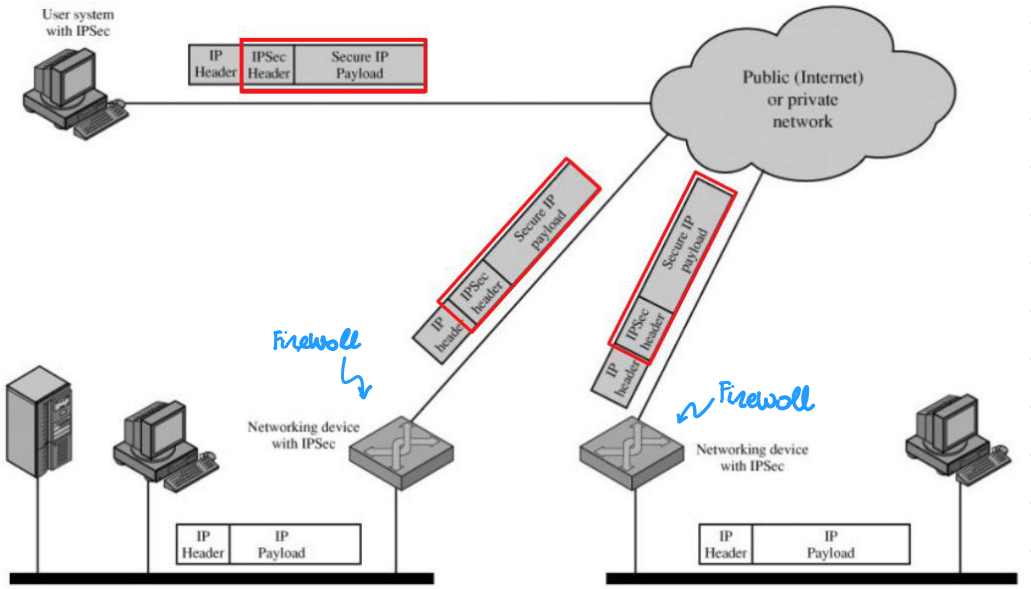
## EKE: encrypted Key exchange

To avoid dictionary attacks if weak keys are choosen. It guarantees mutual authentication with session keys.



# IPSEC

IP security: provide security between IP and TCP. Security protocol that can be used by all applications. Provides authentication, confidentiality and key management. Applicable over LANs, public and private WLANs and internet. Very complicated.




Using IPSEC a firewall/router can provide security to all traffic crossing the border, resistant to bypass. and can be used even on individual users. Works below transport layer → transparent to applications. Main feature is encrypting and/or authenticating traffic at IP level. securing all distributed applications.
It is implemented as additional header:
Authentication header (AH): header for authentication
Encapsulating Security Payload (ESP): header for encryption.

- Access controll: prevents unauthorized use of a resource
- Connectionless integrity: detect edit of individual IP datagram
- Data origin authentication: verify identity of data's source
- Rejection of replayed packets: partial sequence integrity
- Confidentiality (encryption)
- limited traffic flow confidentiality: TFC = concealing source and destination addresses, message length or frequency of communication.

| | AH | ESP (no auth.) | ESP (with auth.) |
|---|---|---|---|
| Access control | ✔ | ✔ | ✔ |
| Connectionless integrity | ✔ | | ✔ |
| Data origin authentication | ✔ | | ✔ |
| Rejection of replayed packets | ✔ | ✔ | ✔ |
| Confidentiality | | ✔ | ✔ |
| Limited traffic flow confidentiality | | ✔ | ✔ |

# SECURITY ASSOCIATIONS

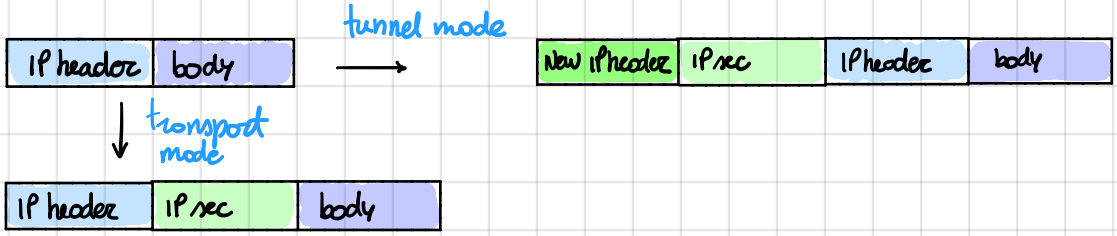
SA are one-way relationship between sender and receiver. There are databases for SAs (SADB) dividing inbound and outbound connections for each interface with IPSec enabled.
SA is denoted by: Security Parameters Index (SPI), IP Destination Address and Security Protocol Identifier (AH / ESP).
For every bidirectional connection we have a pair of SAs. SPI is a sort of TCP port number that allow reciver to select in which SA will process a packet. For multicast, SA is provided for the group and duplicated for all authorized receiver. May be multiple SA for single group to add multiple layer of security.
SA Selector: filter traffic in order to map it into a particular SA of SPDB.
SPDB is a database that specify the policies that determine the disposition of all IP traffic even if not IPsec while SADB contain SAs parameters (eg. encryption key)
SAs can work in transport or tunnel mode according to the fact that the authentication is end2end or end2 intermediate (eg router)

## TRANSPORT MODE

Original IP header untouched, IPsec infos added between IP header and body. Used for host-to-host communications. Routing intact since only payload is encrypted and/or authenticated

## TUNNEL MODE

Keep original IP packet intact but protect it adding new header. → New IP packet. Usually used to create VPN.




| | Transport Mode SA | Tunnel Mode SA |
|---|---|---|
| **AH** | Authenticates IP payload and selected portions of IP header and IPv6 extension headers. | Authenticates entire inner IP packet (inner header plus IP payload) plus selected portions of outer IP header and outer IPv6 extension headers. |
| **ESP** | Encrypts IP payload and any IPv6 extension headers following the ESP header. | Encrypts entire inner IP packet. |
| **ESP with Authentication** | Encrypts IP payload and any IPv6 extension headers following the ESP header. Authenticates IP payload but not IP header. | Encrypts entire inner IP packet. Authenticates inner IP packet. |

As we said SAs can implement both AH and ESP in two ways:

- transport adjacency: apply more than one protocol to same IP packet without tunneling, only one level of combination.
- iterated tunneling: apply multiple layers of security with multiple tunneling.

The two methods can be combined.

In order to have both encryption and authentication there are 3 ways.

1) ESP with authentication

User first applies ESP and then append authentication data field.

(1a) transport mode: auth. and encr. to IP payload but IP header is not protected ] authentication after encryption

(1b) tunnel mode: entire inner IP packet is protected, all IP packet authenticated. ]

2) Transport adjacency:

Still authentication after encryption. Uses two bundled transport SAs: ESP (inner) and AH (outer)
Encryption applied to IP payload and resulting packet is IP header + ESP. Then AH is applied in transport mode → auth. covers ESP + original IP header

3) Transport-Tunnel Bundle

Authentication before encryption: authentication data encrypted → impossible to alter them
+ if message stored as plaintext then verifying authentication data doesn't require re-encryption.

Inner AH on transport mode and outer ESP in tunnel mode: authentication applied to IP payload + IP header, then all processed in ESP tunnel mode, resulting in an authenticated inner packet encrypted + new IP header.

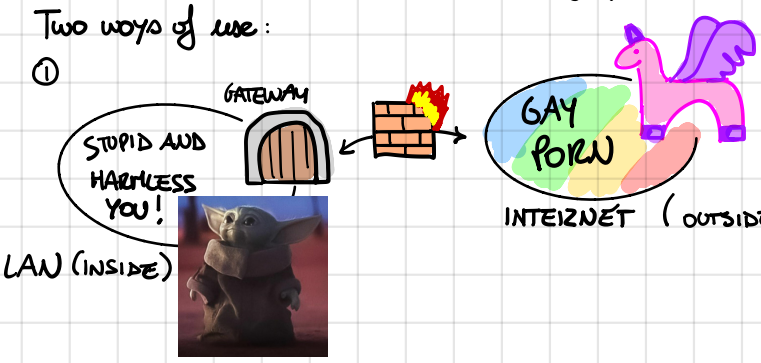
# How SSL/TSL 🔒 works

Under Application level and over TCP to be used with HTTP → HTTPS.
Offers interface to securely send packets (delivery guaranteed by TCP).
Handshaking between client and server, server picks strongest cipher and hash function available for both and notifies client of the decision. → Timeconsuming → Many webmasters dont use TLS for this reason (wrong)
Server sends its identification (X.509 certificate) and client can check with CA.
Client generate session key creating a random number (RN) encrypted with server public key (PbK) the only one who can decrypt it with its private key (PvK).
Client knows PbK and RN and server knows PvK and RN: from RN both can encrypt and decrypt.
If one of previous steps fails → connection closed.
Typically authentication is unilateral (only server) even though is possible to have bidirectional (require certificate in server side too)
Session ≠ connection. A session (made by handshaking) can have multiple (peer-2-peer) connections which are persistent in the session.
Between any pair of parties there may be multiple secure connections.

# FIREWALLS

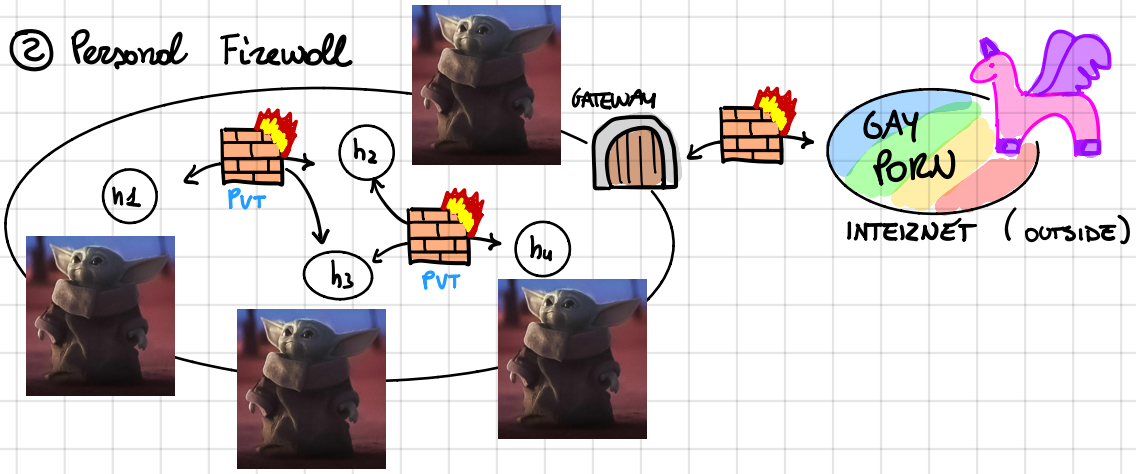
Separate local network from internet. It is an additional security system (hardware or software) that can be used with or without cryptography etc.

Two ways of use:

① 


Firewall works as a packet filter.
- Stateless → without memory.
- Statefull → connection filtering: store info about connections (associated with packets)

Firewalls can work at application level analyzing HTTP messages (FIREWALL PROXY)

② Personal Firewall 


## IPTABLES

Used (in Linux) to maintain tables of IPv4 packet filter rules. These table contain the so called chain, list of rules which can match set of packets

Example of built-in chains are INPUT, OUTPUT (for personal firewalls mostly) and FORWARD (for perimeter firewall)

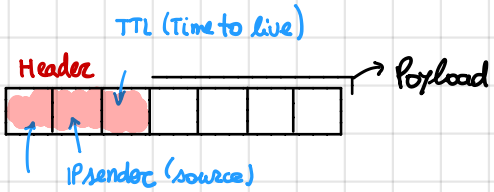
Examples:

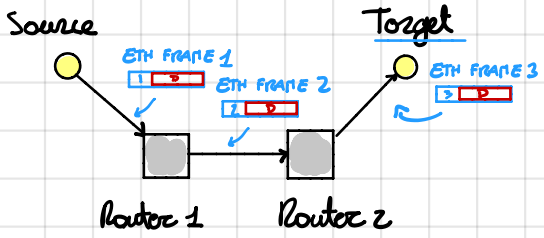
1) iptables -A FORWARD -i eth0 -J DROP
   - -A: append the following rule to table
   - FORWARD: specify that we are using rules for perimeter firewall and not for local delivery
   - -i: input interface (eg eth0)
   - -J: target
   - DROP: discard packet without notification

→ Block all INCOMING traffic.

2) iptables -A FORWARD -i eth0 -m state --state ESTABLISHED -J ACCEPT
   - -m state: add module state
   - --state ESTABLISHED: specify state module settings
   - ACCEPT: accept packets

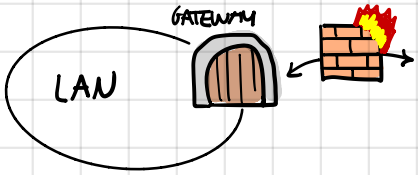
→ Accept packets from outside if they refer to a TCP connection started within the network (ESTABLISHED means that)



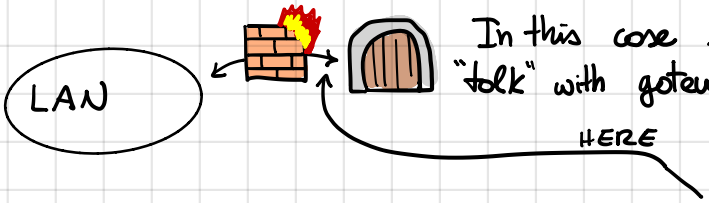

target: public IP number of target (not routers between)

Datagram is inserted in ethernet frame which has two new source and target to move the datagram across routers. (Source – R1, R1 – R2, R2 – Target)




Gateway has to convert private ip numbers (of machines in lan) into public IP → Natting

According to its position, firewall will use private or public IPs




In this case firewall will use private ip of single machine to "talk" with gateway.

In case of attack we want to block all packets incoming:

IPTABLES -I 1 FORWARD -i eth0 -J DROP

IPTABLES -I 2 FORWARD -o eth0 -J DROP

↳ important to put at beginning both

SUICIDE RULE: IPTABLES -I input -dport 22 -J DROP → need to access manually to firewall no more SSH

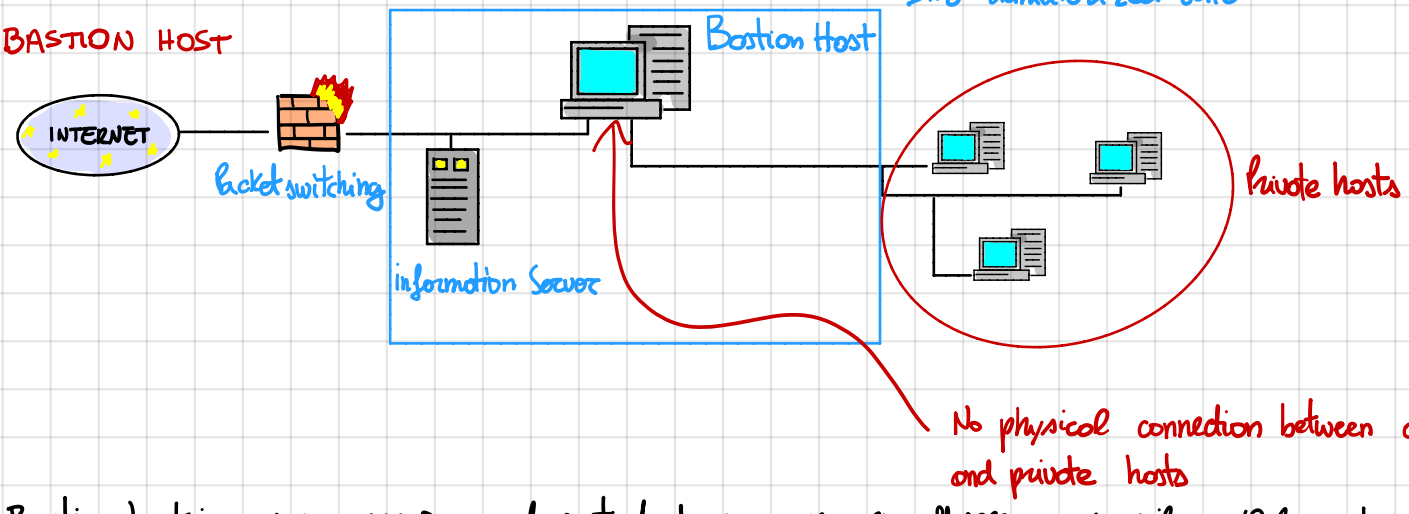
↳ # omitted = 1

As we said, firewalls can work at application level and today is the most used/powerfull approach.

APPLICATION LEVEL GATEWAY (firewall): it works as proxy for different protocols, adding new one is complex though. A firewall doesn't ensure confidentiality anyway!

DMZ: demilitorized zone

BASTION HOST




Bastion host is super secure and protected, no unnecessary software or compilers. Only read system. Traces and integrity checker

A similar solution is to add an other firewall between bastion host and private hosts: SCREENED SUBNET

# IPTABLES SCHEMA

General structure: iptables <command> <chain> <table> rule specification (<match> <target>)

↳ iptables -A POSTROUTING -t nat -o eth0 -j MASQUERADE

Each chain is a list of rules that can match a set of packets. What to do with that matched packet is called target.

Important commands:

- iptables [-t table] -A chain rule-specification : append rule to chain
- iptables [-t table] -I chain rulenum rule-specification : append rule to chain in position rulenum
  Note: rule match is done in sequence: rulenum=1 first, rulenum=2 ... etc.
- iptables [-t table] -N chain : create a new chain

Targets:

if packets match, then next rule is specified by the target which can be the name of an user-defined chain or one of following value:

- ACCEPT: let the packet pass, no more rule checked
- DROP: cancel the packets

Note: -t table command is omitted in this course = we only use one default table

Chains:

- INPUT: for packets destinated to local sockets (to firewall)
- FORWARD: for packets being routed through the box
- OUTPUT: for locally-generated packets (from firewall)

Match:

- -p protocol : the protocol of the rule / packet to check. Eg. UDP, TCP etc
  - --sport : add source port match
  - --dport : add destination port match
- -s address : match on source of packets. Address can be network name, host name, IP (with/without mask)
- -d address: match on destination, same as source.
- -i interface: match the name of interface via which packet was received (only INPUT and FORWARD)
- -o interface: match the name of interface via which packet will be sent (only OUTPUT and FORWARD)

State:

allow connection tracking.

- -m state state : state is a comma separated list of state to match
  - ESTABLISHED: the packet is associated with a connection which has seen packets in both directions
  - NEW: the packet has started the connection