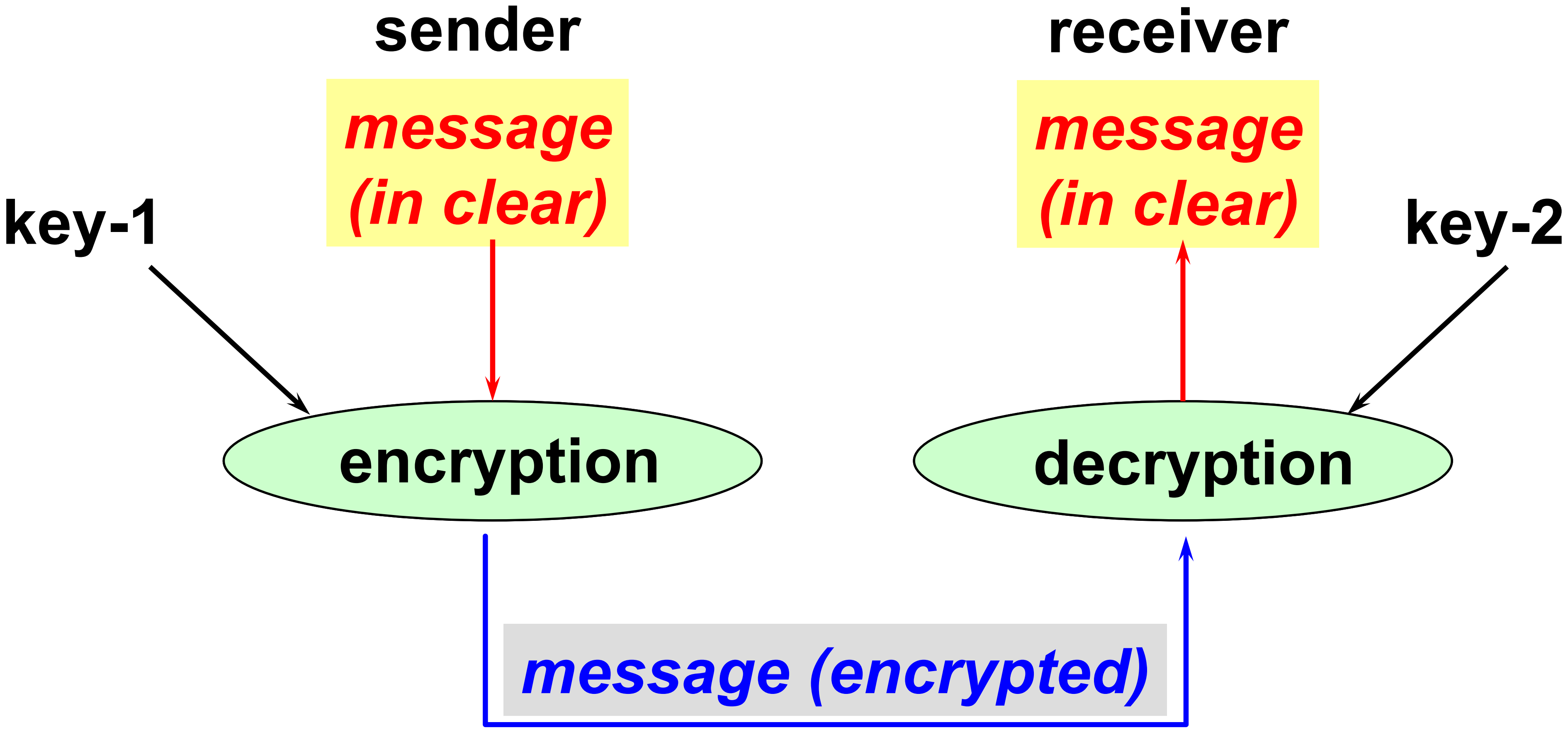
Cryptographic techniques for cybersecurity

Antonio Lioy
< lioy @ polito.it >

Politecnico di Torino
Dip. Automatica e Informatica

Cryptography



Terminology

- **message in clear:**
  - plaintext or cleartext
  - we will refer to it as P
- **encrypted message:**
  - ciphertext
  - we will refer to it as C
  - note that in some countries “encrypted” sounds offensive for religious reasons (cult of dead); in those cases, “enciphered” is preferred

Cryptography's strength (Kerchoffs' principle)

- **if the keys:**
  - are kept secret
  - are managed only by trusted systems
  - are of adequate length
- **then ...**
- **... it has no importance that the encryption and decryption algorithms are kept secret**
- **... on the contrary it is better to make the algorithms public so that they can be widely analysed and their possible weaknesses identified**

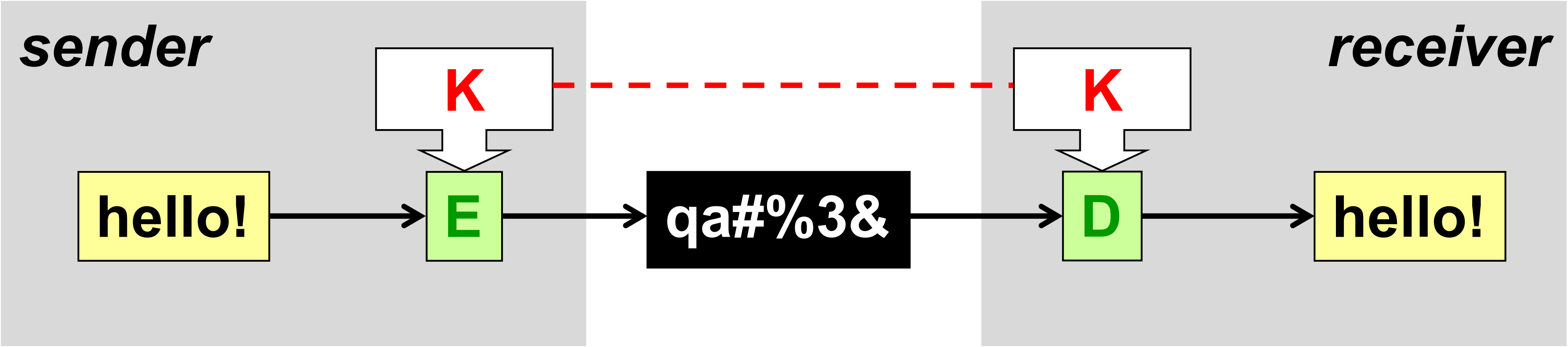
Auguste Kerckhoffs, "La cryptographie militaire", Journal des sciences militaires, vol. IX, pp. 5–38, Janvier 1883, pp. 161–191, Février 1883.

Security through obscurity (STO)

Security trough obscurity
is a thing as bad
with computer systems
as it is with women

Secret key / symmetric cryptography

- **key-1 = key-2 = K**
  - i.e. single key, shared by sender and receiver (only!)
- **low computational load, hence used for data encryption**
- **C = enc (K , P)   or   C = { P } K**
- **P = dec (K , C) = enc^{-1} (K , C)**
- **problem: how to share (securely) the secret key among sender and receiver?**



Some famous symmetric encryption algorithms (block)

name	*key (bit)*	*block (bit)*	*notes*
DES	56	64	obsolete
3-DES (2 keys)	112	64	56…112-bit strength
3-DES (3 keys)	168	64	112-bit strength
IDEA	128	64	famous for PGP
RC2	8-1024	64	usually 64-bit key
Blowfish	32-448	64	usually 128-bit key
CAST	40-128	64	usually 128-bit key
RC5	0-2048	1-256	optimal when B=2W
AES	128-192-256	128	state-of-the-art

The EX-OR (XOR) function

- **ideal "confusion" operator: random input (probability 0 : 1 = 50 : 50%) will generate random output**
- **primitive operation available on all CPU**
- **truth table:**
  - 0 xor 0 = 0
  - 0 xor 1 = 1
  - 1 xor 0 = 1
  - 1 xor 1 = 0

⊕	0	1
0	**0**	**1**
1	**1**	**0**

- **note that it's the inverse of itself:**
  - if A xor B = Z then Z xor B = A (or Z xor A = B)
- **other properties:**
  - A xor 0 = A   A xor 1 = A'   A xor A = 0   A xor A' = 1

DES

- **Data Encryption Standard**
- **standard FIPS 46/2**
- **mode of application standard FIPS 81**
- **56 bits key (+ 8 parity bits) = 64 bits**
- **64 bits data block**
- **designed to be efficient in hardware because it requires:**
  - XOR
  - shift
  - permutation (!)

Triple DES (3DES, TDES)

- repeated application of DES
- uses two or three 56-bit keys
- usually applied in the EDE mode (Encrypt-Decrypt-Encrypt) to achieve compatibility with DES when K1 = K2 = K3
- 3DES with 2 keys (Keq=56 bit if 2^{59}B of memory is available, otherwise Keq=112 bit)

  $C' = enc(K_1, P)$ $C'' = dec(K_2 , C')$ $C = enc(K_1 , C'')$

- 3DES with 3 keys (Keq=112 bit)

  $C' = enc(K_1, P)$ $C'' = dec(K_2, C')$ $C = enc(K_3, C'')$

- standard FIPS 46/3 and ANSI X9.52

Double DES? No, thanks!

- **double application of encryption algorithms is subject to a known-plaintext attack named meet-in-the-middle which allows to decrypt data with at most 2^{N+1} attempts (if the keys are N-bit long)**
- **thus usually the double version of encryption algorithms is never used**
  - the computation time doubles but the effective key length increases by just one bit!
- **note: if the base symmetric algorithm would be a *group,* then it would exist K_3 so that**

  enc(K_2 , enc(K_1 , P)) = enc (K_3 , P)

Meet-in-the-middle attack

- **hypothesis:**
  - N bit keys
  - known P and C such that $C = enc\ (K_2, enc\ (K_1, P))$
- **note:**
  - $\exists$ M such that $M = enc(K_1,P)$ and $C = enc(K_2,M)$
- **actions:**
  - compute 2^N values $X_i = enc\ (K_i, P)$
  - compute 2^N values $Y_j = dec\ (K_j, C)$
  - search those values K_i and K_j such that $X_i = Y_j$
  - "false positives" can be easily discarded if more than one (P,C) couple is available

RC2 and RC4

- **developed by Ron Rivest (RC = Ron's Code)**
- **proprietary algorithms of RSA – but not patented**
  - if used, then problem of security-through-obscurity
- **respectively, 3 and 10 times faster than DES**
- **RC2 is a block algorithm, RC4 is a *stream* one**
- **variable length key**
- **RC2:**
  - published as RFC-2268 (mar 1998)
  - 8 to 1024 bits keys (usually 64 bits)
  - 64 bits data block
- **RC4 still proprietary**
  - … but reverse engineered as ARCFOUR ☺

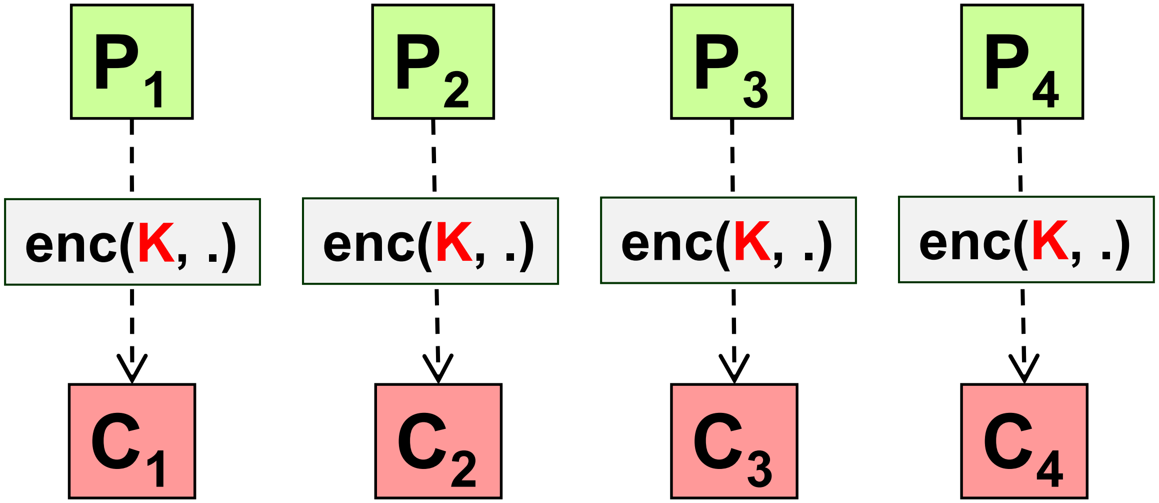
Application of block algorithms

How a block algorithm is applied to a data quantity different from the algorithm's block size?

- **to encrypt data with size > the algorithm's block size:**
  - ECB (Electronic Code Book)
  - CBC (Cipher Block Chaining)
- **to encrypt data with size < the algorithm's block size:**
  - padding
  - CFB (Cipher FeedBack), OFB (Output FeedBack)
  - CTR (Counter mode)

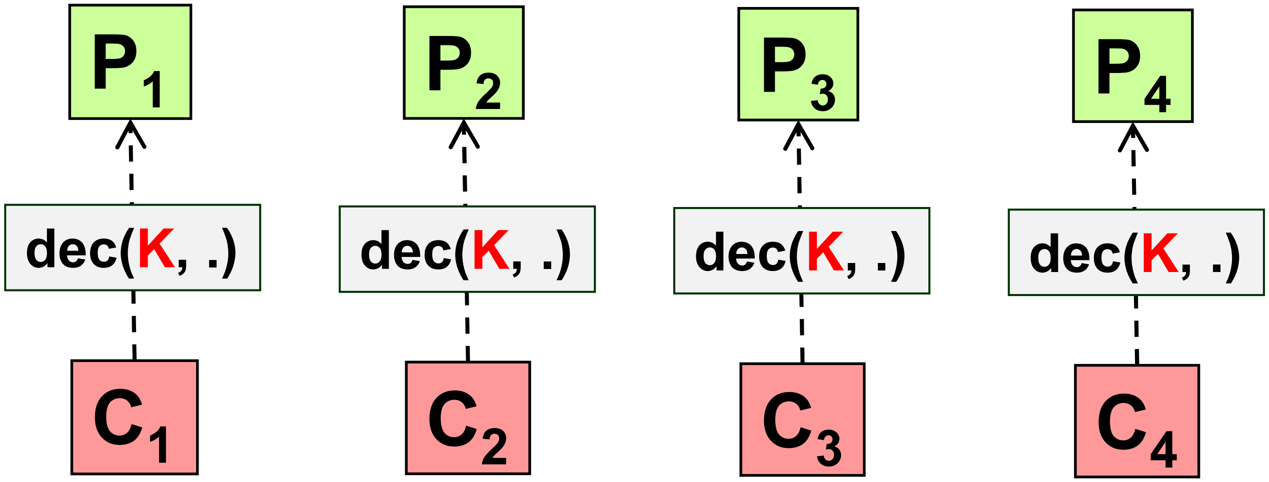
ECB (Electronic Code Book)

- **formula for the n-th block:**
  C_n = enc (K, P_n)
- **NOT to be used on long messages because**
  - swapping of two blocks of ciphertext goes undetected
  - identical blocks generate identical ciphertexts hence it is vulnerable to *known-plaintext* attacks



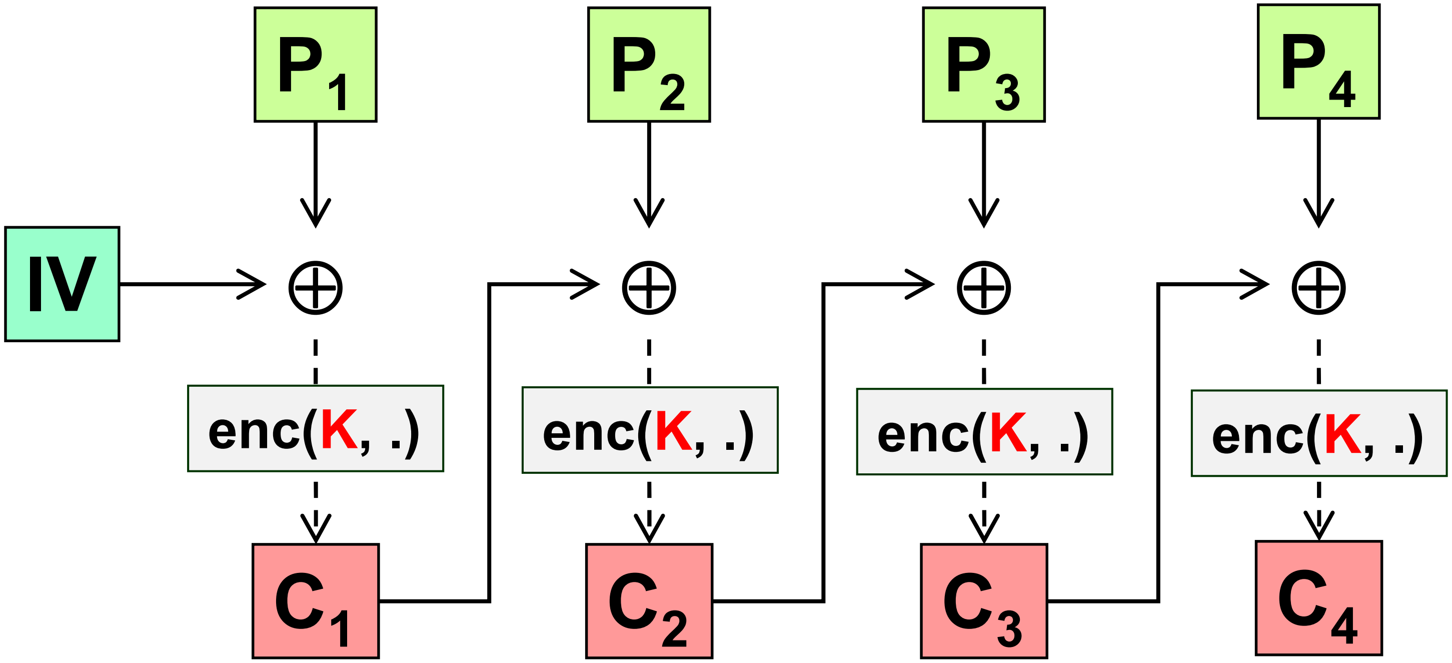
ECB - decrypt

- formula for the n-th block:
  $P_n = enc^{-1} (K, C_n)$
- an error in transmission generates an error at the decryption of one block



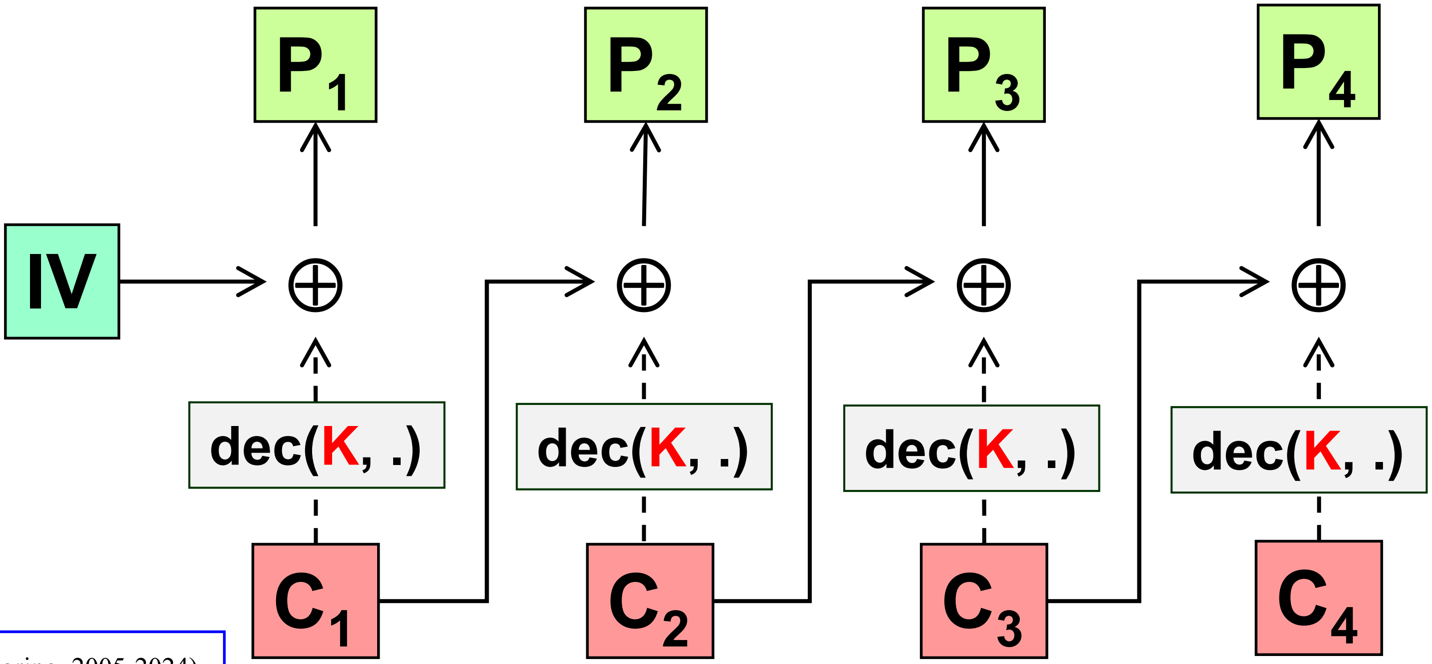
CBC (Cipher Block Chaining)

- formula for the n-th block:
  $C_n = \text{enc} (K, P_n \oplus C_{n-1})$
- requires C_0 = IV (Initialization Vector)



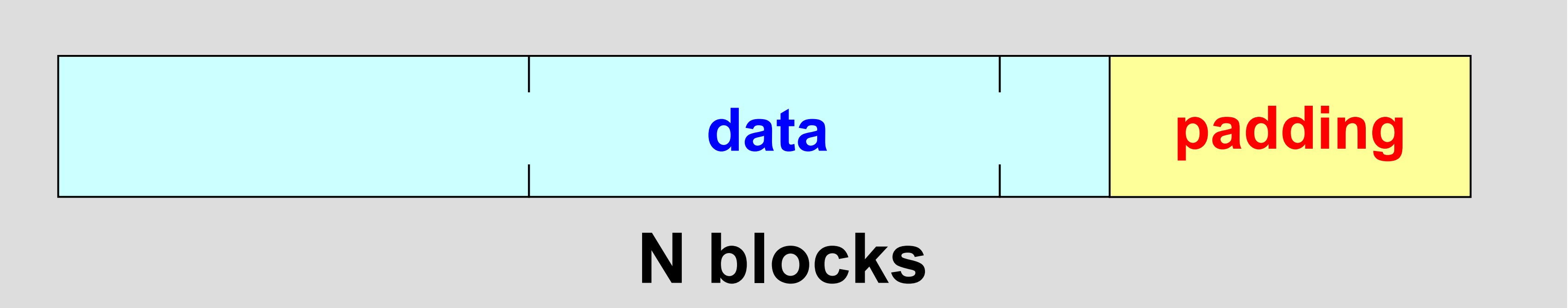
CBC - decryption

- **formula for the n-th block:**
  $P_n = enc^{-1} (K , C_n) \oplus C_{n-1}$
- **requires C_0 (i.e. IV) to be known by the receiver**
  - as plaintext or encrypted
- **one error in transmission generates an error at the decryption of two blocks**



Padding (aligning, filling)

- **size of algorithm’s block B**
- **size of data to process D (not a multiple of B)**
- **add bits until a multiple of B is reached**



- **problems:**
  - transmit/store more data (B) than needed (D)
  - value of padding bits?

Padding techniques

- **(if length is known or it can be obtained – e.g. a C string) add null bytes**
  - … 0x00 0x00 0x00
- **(original DES) one 1 bit followed by many 0**
  - … 1000000
- **one byte with value 128 followed by null bytes**
  - … 0x80 0x00 0x00
- **last byte's value equal to the length of padding**
  - … 0x?? 0x?? 0x03
  - what about the value of the other bytes?

Padding with explicit length (L)

- (Schneier) null bytes:
  - e.g. … 0x00 0x00 0x03
- (SSL/TLS) bytes with value L:
  - e.g. … 0x03 0x03 0x03
- (SSH2) random bytes:
  - e.g. … 0x05 0xF2 0x03
- (IPsec/ESP) progressive number:
  - e.g. … 0x01 0x02 0x03
- bytes with value L-1:
  - e.g. … 0x02 0x02 0x02

Padding – some notes

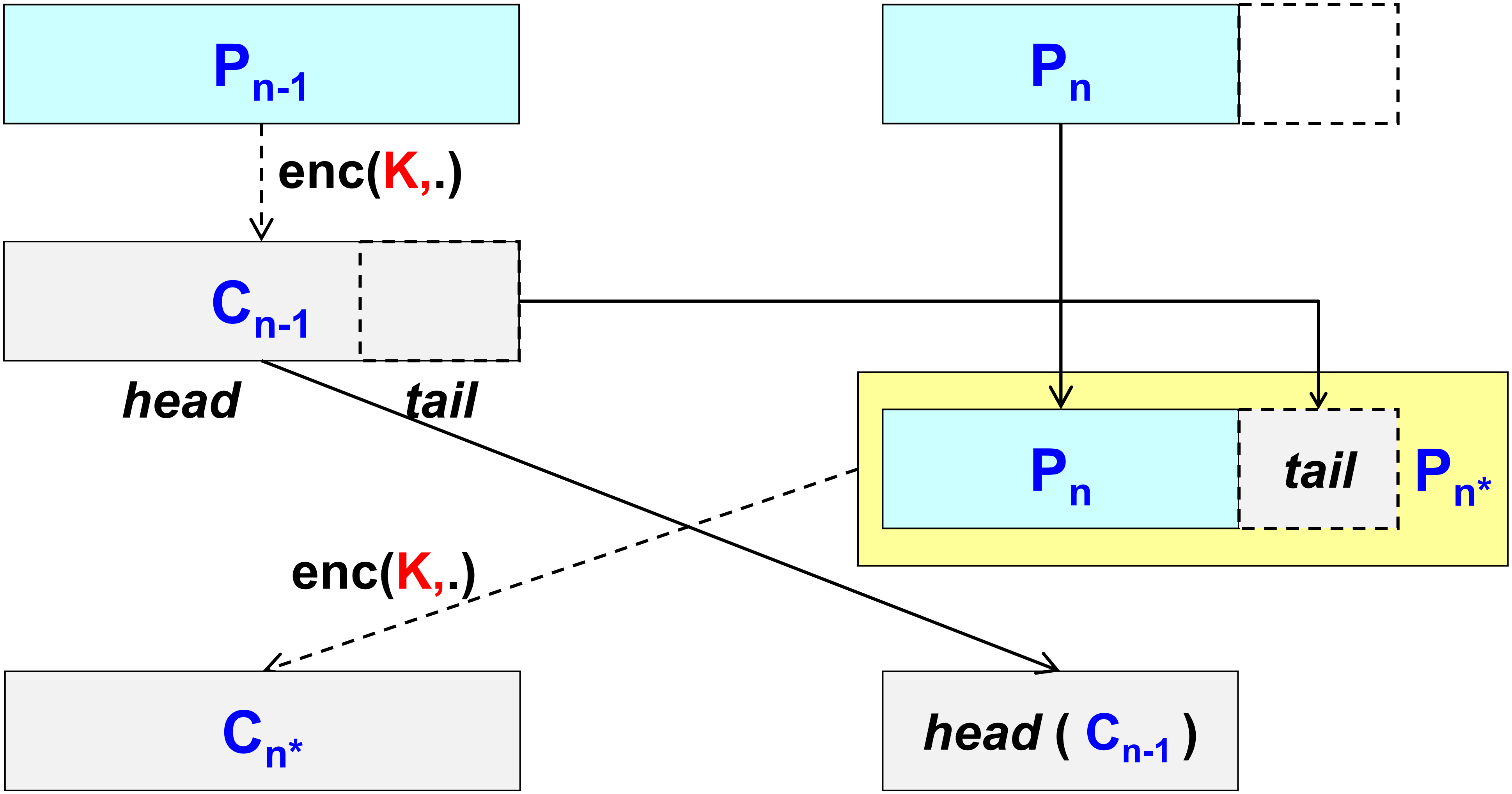
- **some offer (minimal) integrity control**
  - if key is wrong or data is manipulated, then the padding bytes are incoherent (e.g. L>B or wrong padding values)
- **typically applied to large data, on the last fragment resulting from the division in blocks (e.g. for ECB or CBC)**
- **if |D| < |B| we prefer ad-hoc techniques (CFB, OFB, CTR, …)**
- **even if the plaintext is an exact multiple of the block, padding must be added anyhow to avoid errors in the interpretation of the last block**

  $1 \leq L \leq |B|$
- **with SSH2 padding equal data gives different ciphertexts**
- **the padding type for a certain algorithm determines the type of (some) possible attacks**

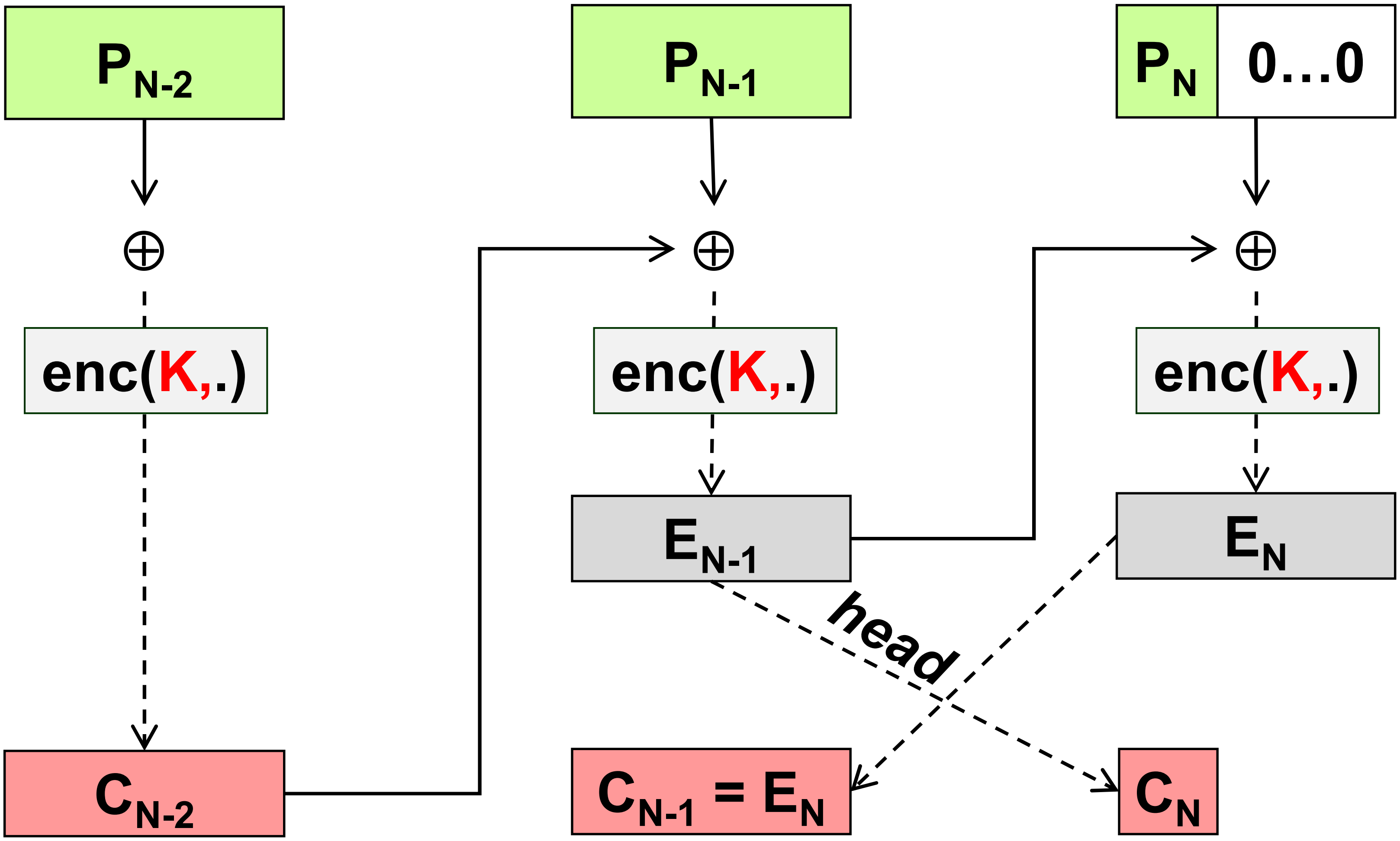
Ciphertext stealing (CTS)

- **CTS permits to use block algorithms without padding**
  - last (partial) block filled with bytes from the second-to-last block
  - these bytes are removed from the second-to-last block (which becomes a partial one)
  - after encryption, exchange the position of the last and second-to-last blocks
- **useful when we cannot increase the size of the data after encryption**
  - very important for storage encryption
- **the computation time slightly increases**

CTS – example with ECB (encryption)

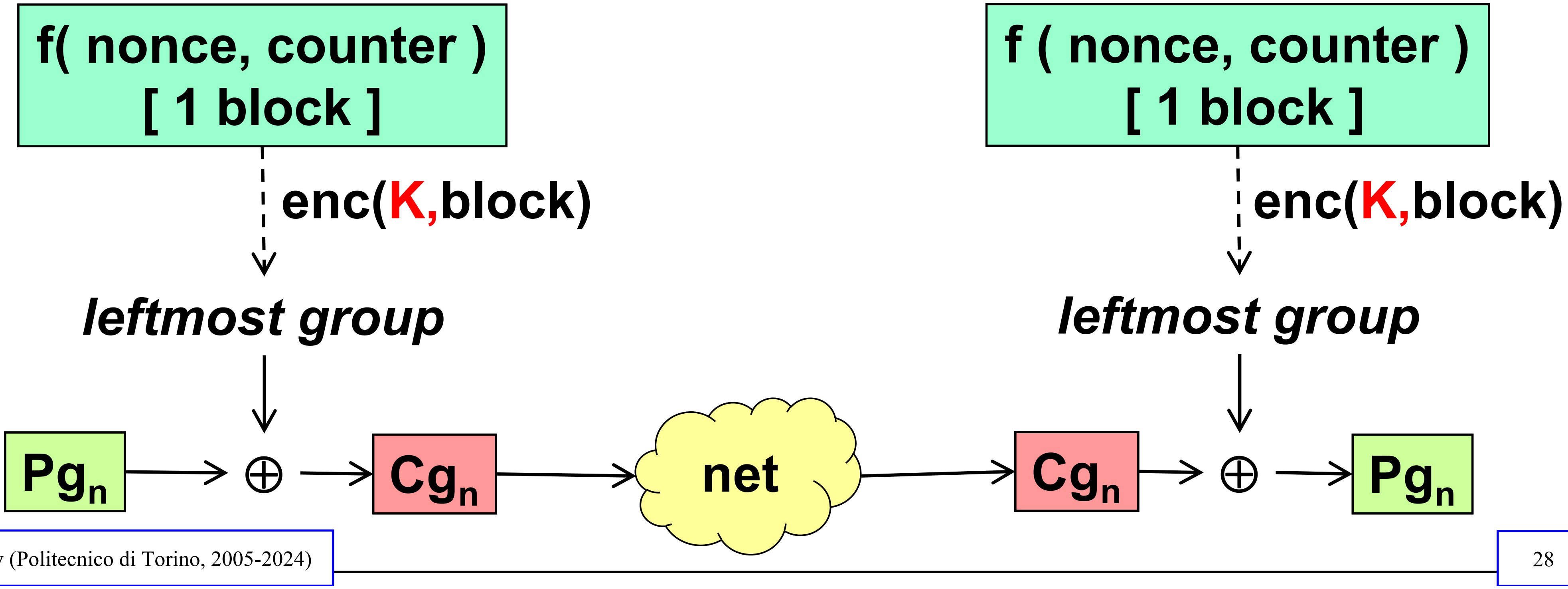


CTS – example with CBC (encryption)



CTR (Counter mode)

- block algo to encipher N bits at a time (a "group", often a byte)
- no padding needed and random direct access to ciphertext
- requires a nonce and a counter, combined together by a suitable function (concatenation, sum, XOR, …)
- one transmission error ~ decryption error only of one group



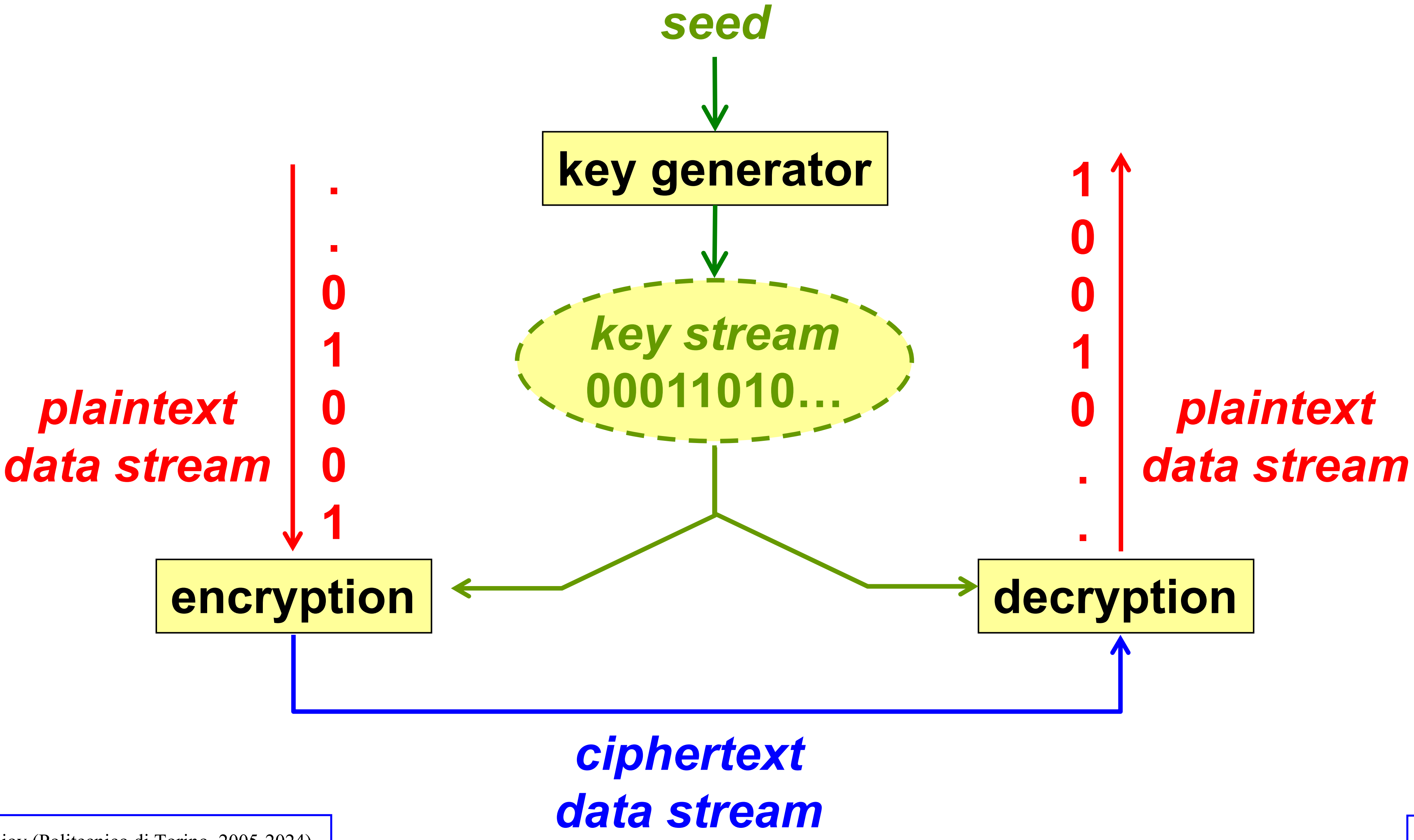
Counter mode – example

- **a long data stream must be enciphered in CTR mode, byte-oriented, with key K, nonce N, and f(.) is just concatenation**
  - note: in this case, N must have size B-1
- **sender:**
  - for (x=0; x<data_size; x++)
    - ctext[x] = ptext[x] xor MSB(enc(K, nonce || x))
- **if the receiver needs to decipher only bytes no. 30 and 31:**
  - ptext[30] = ctext[30] xor MSB(enc(K, nonce || 30))
  - ptext[31] = ctext[31] xor MSB(enc(K, nonce || 31))

Symmetric stream encryption algorithms

- **work on a data stream without requiring the split in blocks**
  - typically operate on one bit or one byte at a time
- **ideal algorithm:**
  - one-time pad (requires a key which is as long as the message to protect!)
- **real algorithms:**
  - use pseudo-random key generators, synchronized between the sender and the receiver
  - (old) RC4, SEAL
  - (modern) Salsa20, ChaCha20
- **note: the byte-oriented CTR mode of a block algorithm may be considered a stream algorithm**

Algorithms of type stream



Salsa20 and ChaCha20

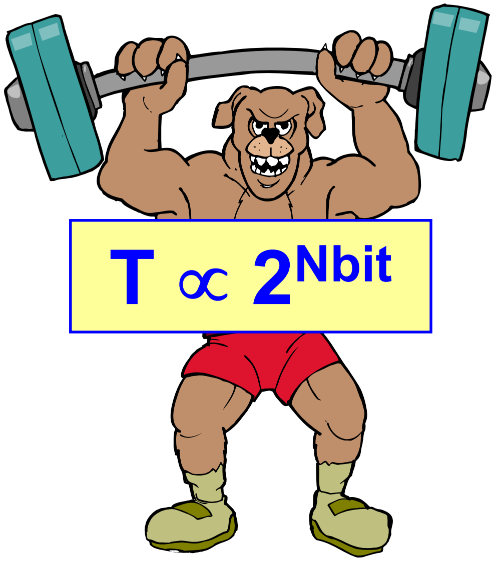
- **symmetric stream algorithms invented by D.J.Bernstein**
  - 128 or 256 bit key
- **ChaCha20 is an improvement of Salsa20**
  - more "diffusion" of bits
  - faster on some architectures
- **base operation: 32-bit ARX (add – rotate – xor)**
- **base function:**
  - f (Key256, Nonce64, Counter64) = 512-bit keystream block
  - this permits O(1) decryption of any block at random
- **Salsa20 performs 20 mixing rounds of the input**
- **Salsa20/12 and Salsa20/8 make 12 and 8 mixing rounds**
  - faster but less secure

ChaCha20 standardization and adoption

- **RFC-8439 (ChaCha20 and Poly1305 for IETF protocols)**
- **IETF has slightly modified the original specification**
  - 96 bit nonce
  - 32 bit block counter
    - note: counter may start from 1 (rather than 0) if the first block is used as an authentication tag (as in AEAD)
  - hence a limit of 256 GB (2^32 64-byte blocks)
  - to overcome this limit, use the original definition by Bernstein
- **used by Google (for Chrome on Android), openssh and various CSPRNG (e.g. /dev/urandom since Linux kernel 4.8)**

Length of secret keys

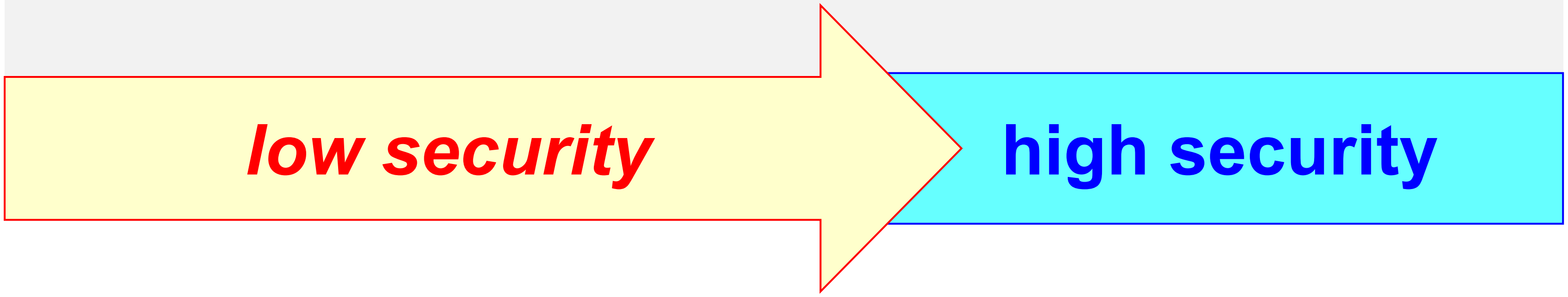
- **if (Kerchoff's conditions):**
  - the encryption algorithm was well designed
  - the keys – Nbit in length – are kept secret
  - the algorithm is executed by a trusted party (e.g. no malware)
- **… then the only possible attack is the brute force (exhaustive) attack which requires a number of trials equal to**



$T \propto 2^{Nbit}$

Length of cryptographic keys

symm	40	80	128	256	…
asymm (FFC, IFC)	512	1024	2048	4096	…
asymm (ECC)	80	160	256	512	…



low security

high security

… proved by theoretical analysis
and experimental “challenges”

AES (Advanced Encryption Standard)

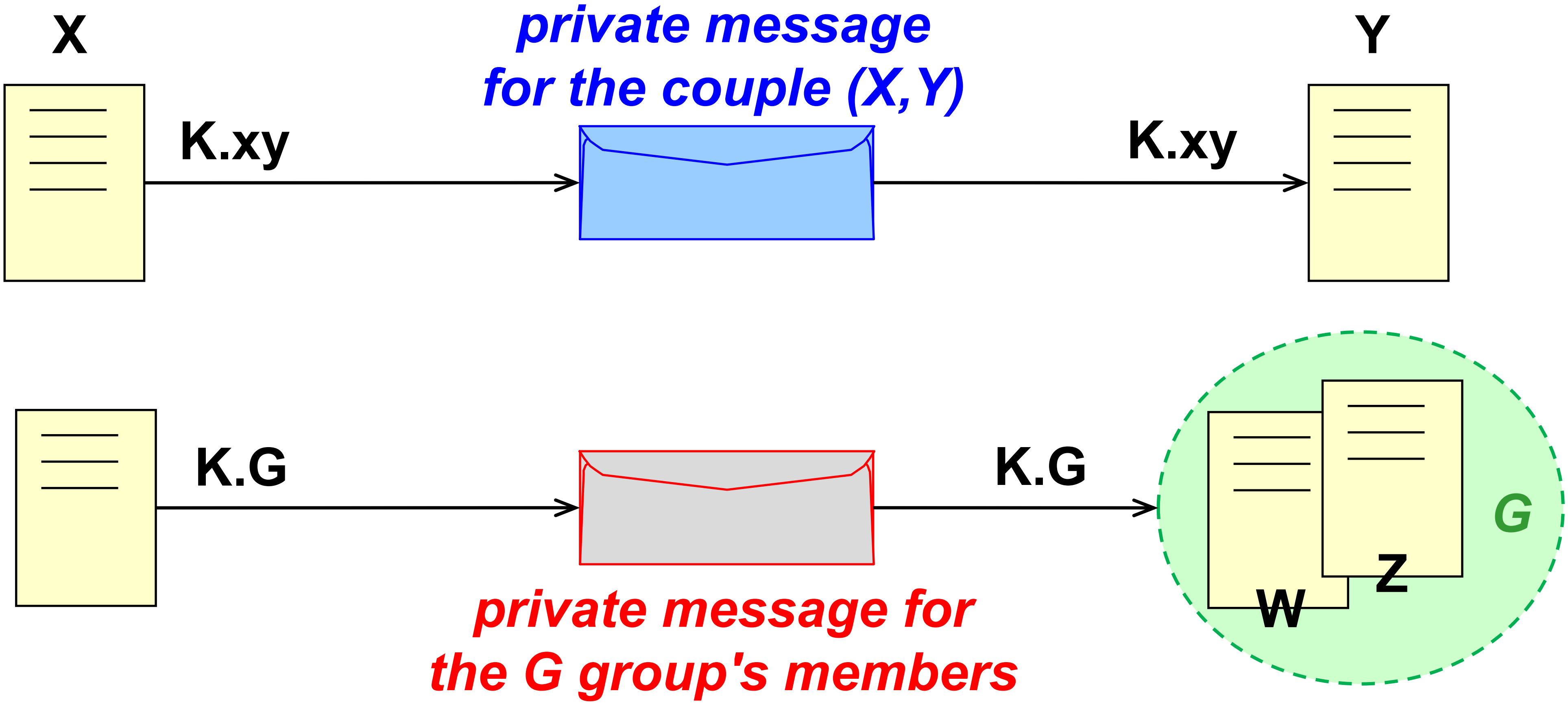
- **public international competition to replace DES**
- **15 candidates**
- **5 finalists (9 august 1999):**
  - MARS (IBM)
  - RC6 (RSA, i.e. Ron Rivest)
  - Rijndael (Joan Daemen, Vincent Rijmen)
  - Serpent (Ross Anderson, Eli Biham, Lars Knudsen)
  - Twofish (Bruce Schneier and others)
- **information about the selection process:**
  http://www.nist.gov/aes

AES = RIJNDAEL

- **2 October 2000**
- **RIJNDAEL chosen as winner**
  - key length 128 / 192 / 256 bit
  - block size 128 bit
- **published in November 2001 as FIPS-197**
- **gradually adopted after 2010 (it took so long because crypto algorithms are like wine: the best ones are those aged for several years …)**

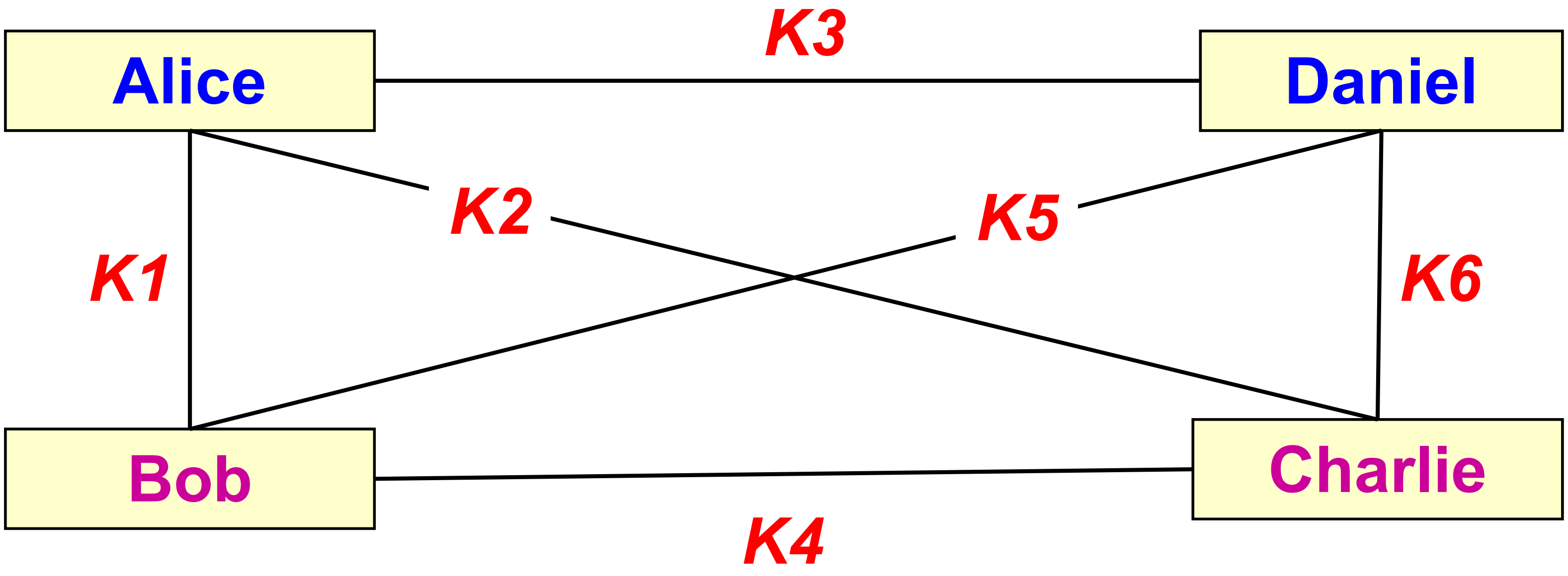
Symmetric encryption

- single and secret key
- one key for each couple (or group!) of users



Key distribution for symmetric cryptography

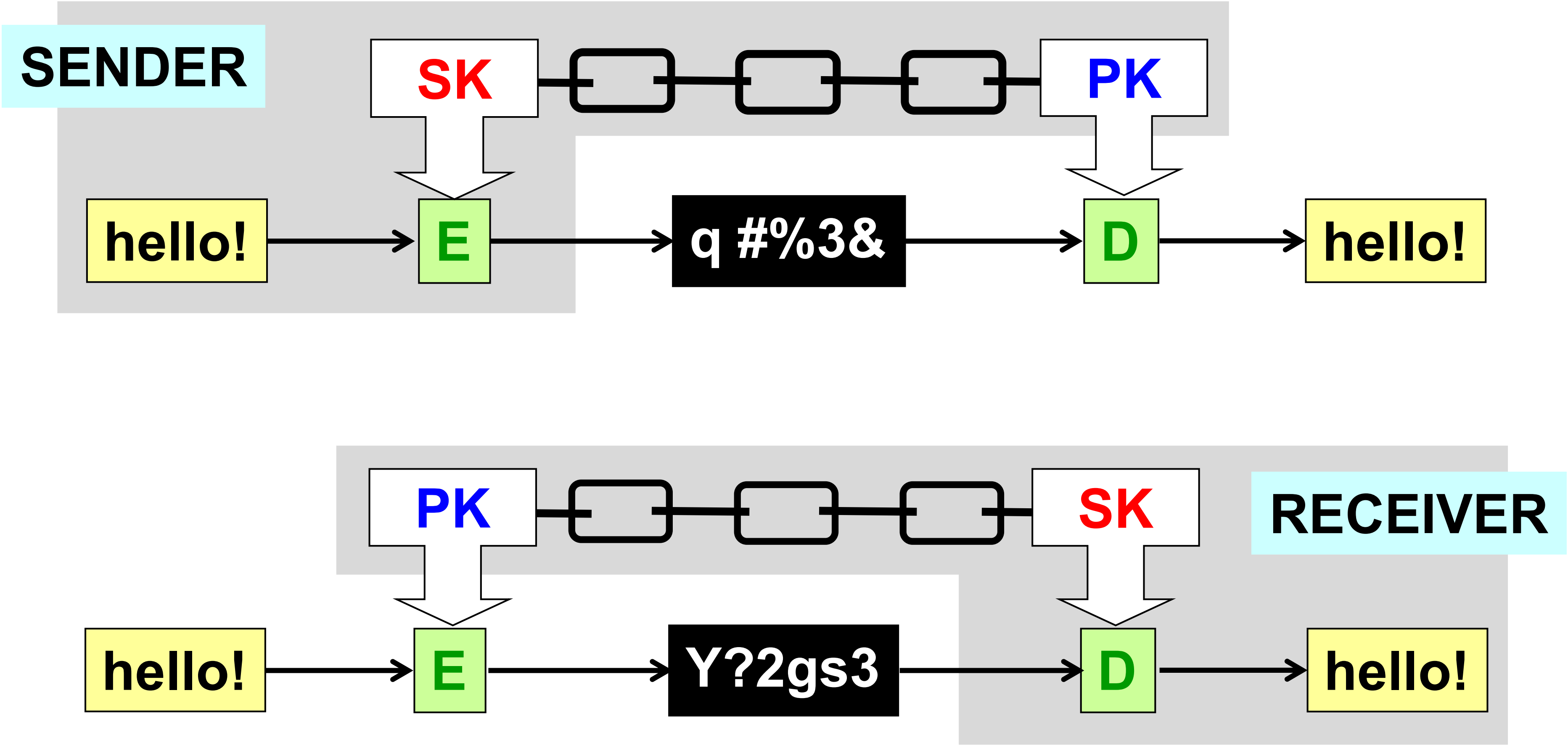
- **for a complete pairwise private communication between N parties N x (N-1) / 2 keys are necessary:**
  - distribution OOB (Out-Of-Band)
  - distribution by means of key exchange algorithms



Public-key / asymmetric cryptography

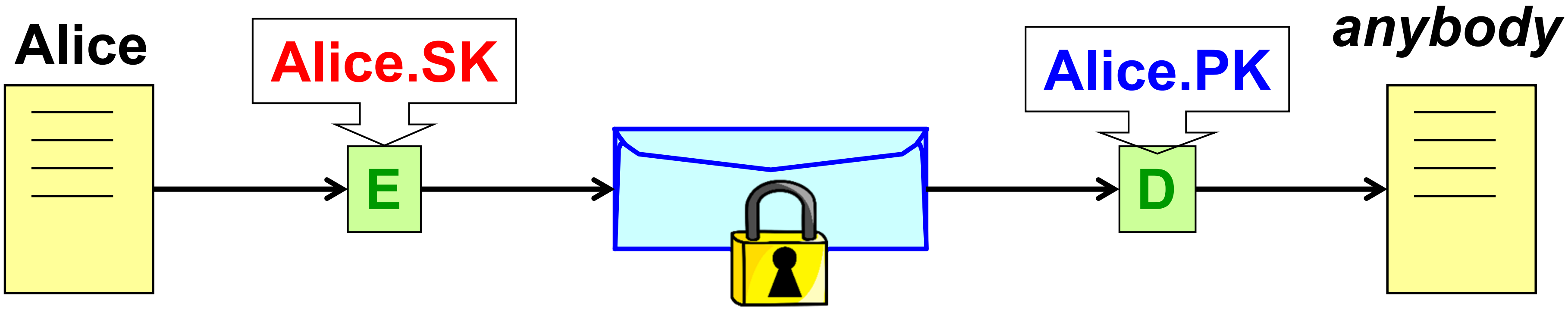
- **key-1 ≠ key-2**
  - hence the term "asymmetric" algorithms
- **keys are not independent, they are generated in pairs: *private key* (SK) + *public key* (PK)**
- **keys in the pair have inverse functionality: data encrypted with one key can be decrypted only by the other one**
- **high computational load**
  - don't use to encrypt large data
- **used to distribute secret keys and to create electronic signatures (with hashing)**
- **main algorithms:**
  - Diffie-Hellman, RSA, DSA, El Gamal, …

Asymmetric cryptography – example



Digital signature

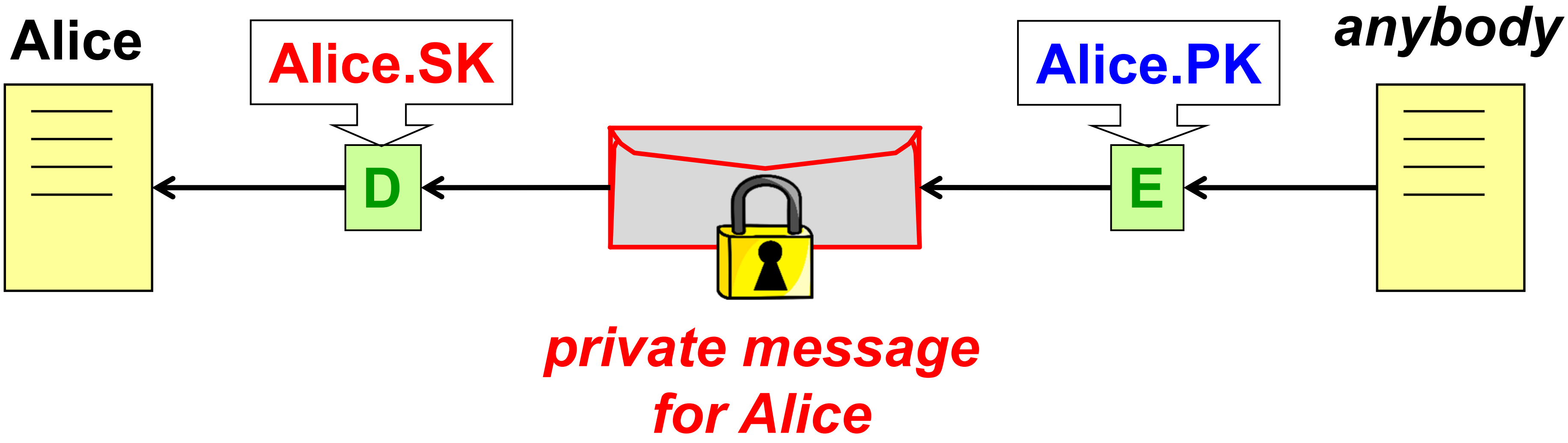
- **digital signature ~ asymmetric encryption of data made with the private key of the author**
  - BEWARE: this is the basic idea, actual dsig is more complex
  - usually, data is not directly encrypted but only its summary (digest)
- **provides data authentication (and integrity)**



public message "signed" by Alice

Confidentiality without shared secrets

- it is possible to generate a secret message for a particular receiver given only its public key



Public-key algorithms

- **RSA (Rivest - Shamir - Adleman)**
  - product of prime numbers (attack = factorization of the result)
  - digital signature and confidentiality w/o shared secrets
  - patented - only in USA - but patent expired on 20-set-2000
- **DSA (Digital Signature Algorithm)**
  - exponentiation (attack = logarithm of the result, unknown base)
  - digital signature only (as it uses a one-way lossy compression function, so original plaintext cannot be recovered)
    - for encryption, use the El-Gamal algorithm
  - standard NIST for DSS (FIPS-186)

RSA – the algorithm (I)

- public module $N = P \times Q$ known to anybody
  P and Q are prime, large, and *secret*
- PHI = (P-1) (Q-1)
- public exponent E such that arbitrarily $1 < E < PHI$ and it is relatively prime with respect to PHI
- private exponent $D = E^{-1} \bmod PHI$
- public key = (N, E)
- private key = (N, D)
- P and Q are deleted, discarded, killed, ...
  - if you come to know P and Q then you can compute D (!!!)
- RSA key size = size of the public module

RSA – the algorithm (II)

- **RSA may cipher/decipher only data whose value is less than the value of the module N**
  - it's a sort of block algorithm, with block size equal to the key size
- **plaintext: $p < N$**
- **encrypt: $c = p^E \bmod N$**
- **decrypt: $p = c^D \bmod N$**
- **the roles of E and D are interchangeable because $(x^D)^E \bmod N = (x^E)^D \bmod N$**
- **note that the complexity of the operations depends upon the number of bits with value 1 in the exponents E and D**

RSA computational optimization (I)

- **usually all public keys have E=3, 17 or 65537 (0x10001, the Fermat number)**
  - the power operation is very easy because these numbers have only two bits set to one
    - (high) speed of the encryption operation
    - (high) speed in the operation of signature verification
  - optimized algorithms for this special case
- **attack: provide a signature made with a key whose exponent has many bits set to one, to generate a high computational load**

RSA computational optimization (II)

- in RSA the operations involving the private key (signing and decrypting) are slow
- the CRT (Chinese Remainder Theorem) makes them faster (4x) thanks to the equivalence
  f(x) mod N ~ f(x) mod P & f(x) mod Q
- it is a different representation of the private key (beware! we don't store P and Q but some derivatives useful in CRT computations)
- standardized in PKCS#1
- attack: makes RSA more susceptible to fault injection attacks

Twinkle (!?)

An Analysis of Shamir’s Factoring Device
Robert D. Silverman
RSA Laboratories
May 3, 1999

At a Eurocrypt rump session, Professor Adi Shamir of the Weizmann Institute announced the design for an unusual piece of hardware. This hardware, called “TWINKLE” (which stands for The Weizmann INstitute Key Locating Engine), is an electro-optical sieving device which will execute sieve-based factoring algorithms approximately two to three orders of magnitude as fast as a conventional fast PC. The announcement only presented a rough design, and there are a number of practical difficulties involved with fabricating the device. It runs at a very high clock rate (10 GHz), must trigger LEDs at precise intervals of time, and uses wafer-scale technology. However, it is my opinion that the device is practical and could be built after some engineering effort is applied to it. Shamir estimates that the device can be fabricated (after the design process is complete) for about $5,000.

Twirl (!!?)

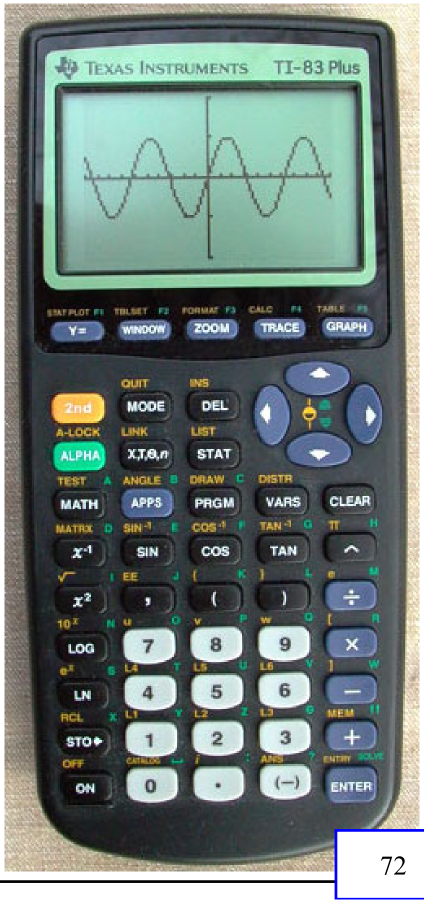
(From Eran Tromer home page, visited October 2017)

TWIRL (The Weizmann Institute Relation Locator) is an electronic device for factoring of large integers. It implements the sieving step of the Number Field Sieve integer factorization algorithm, which is in practice the most expensive step in factorization. TWIRL is more efficient than previous designs by several orders of magnitude, due to high algorithmic parallelization combined with adaptation to technological hardware constraints. Although fairly detailed, the design remains hypothetical since the device has not been actually built. However, projected cost estimates suggest that if TWIRL is built using current VLSI technology, it will be possible to factor 1024-bit integers, and hence to break 1024-bit RSA keys, in 1 year at the cost of a few dozen million US dollars (or significantly less, if several integers are to be factored simultaneously).

Adi Shamir, Eran Tromer, "Factoring Large Numbers with the TWIRL Device", proc. Crypto 2003, LNCS-2729, pp.1-26, Springer-Verlag, 2003

Firmware signature for TI calculators

- **TI83+ graphing calculator**
- **firmware protected by 512-bit RSA signature**
- **signature key factored on July 2009**
  - 73 days of computation
    on a 1.9 GHz dual-core Athlon64 PC
  - on 1999 same computation required
    8000 MIPS-years + Cray C916
- **now all users may autonomously modify the firmware by themselves**

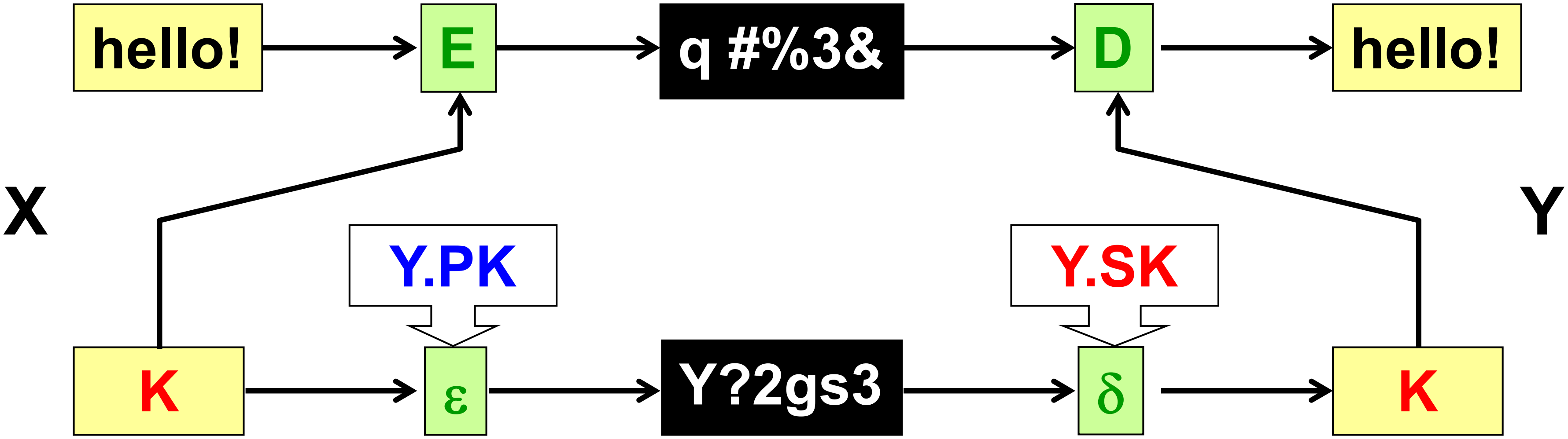


Key distribution for asymmetric cryptography

- **private key never disclosed!**
- **public key distributed as widely as possible**
- **problem: *who guarantees the binding (correspondence) between the public key and the identity of the person?***
- **solution #1: exchange of keys OOB (e.g. key party!)**
- **solution #2: distribution of the public key by means of a specific data structure named public-key certificate (= digital certificate)**
  - format of the certificate?
  - trust in the certificate issuer?

Secret key exchange by asymmetric algorithms

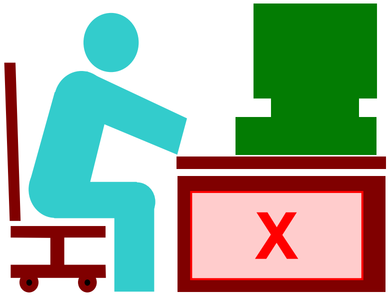
- confidentiality without shared secrets is often used to send the secret key chosen for a symmetric algorithm

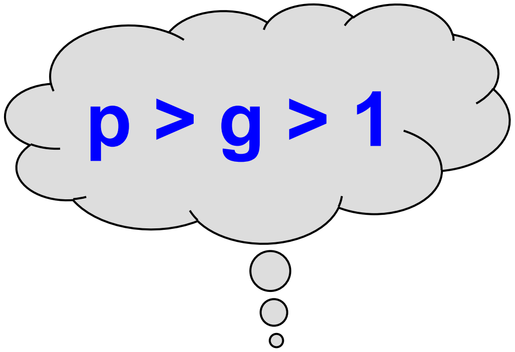


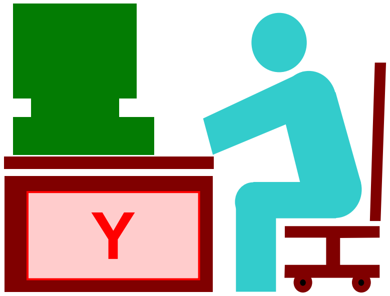
Diffie-Hellman

- A and B choose / agree two public integers p (prime, large) and g (generator, a primitive root modulo p) such that:
  $1 < g < p$ (typically g=2, 3, or 5)
- length of DH key = no. of bits of p
- A arbitrarily chooses an integer x>0 and computes:
  $X = g^x \bmod p$
- B arbitrarily chooses an integer y>0 and computes:
  $Y = g^y \bmod p$
- A and B exchange (publish) X and Y
- A computes $K_A = Y^x \bmod p$
- B computes $K_B = X^y \bmod p$
- but $K_A = K_B = g^{xy} \bmod p$

Diffie-Hellman (DH)

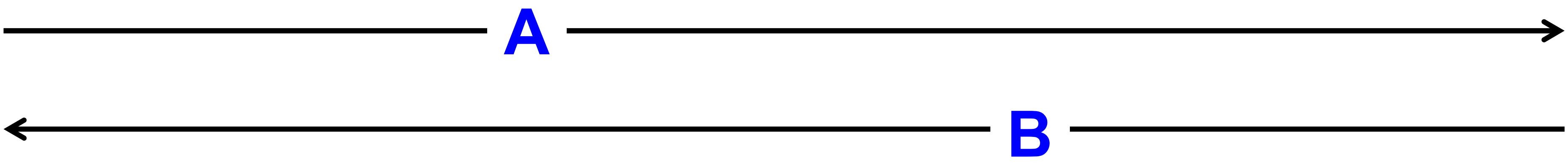






$A = g^X \bmod p$

$B = g^Y \bmod p$



$K_A = B^X \bmod p$

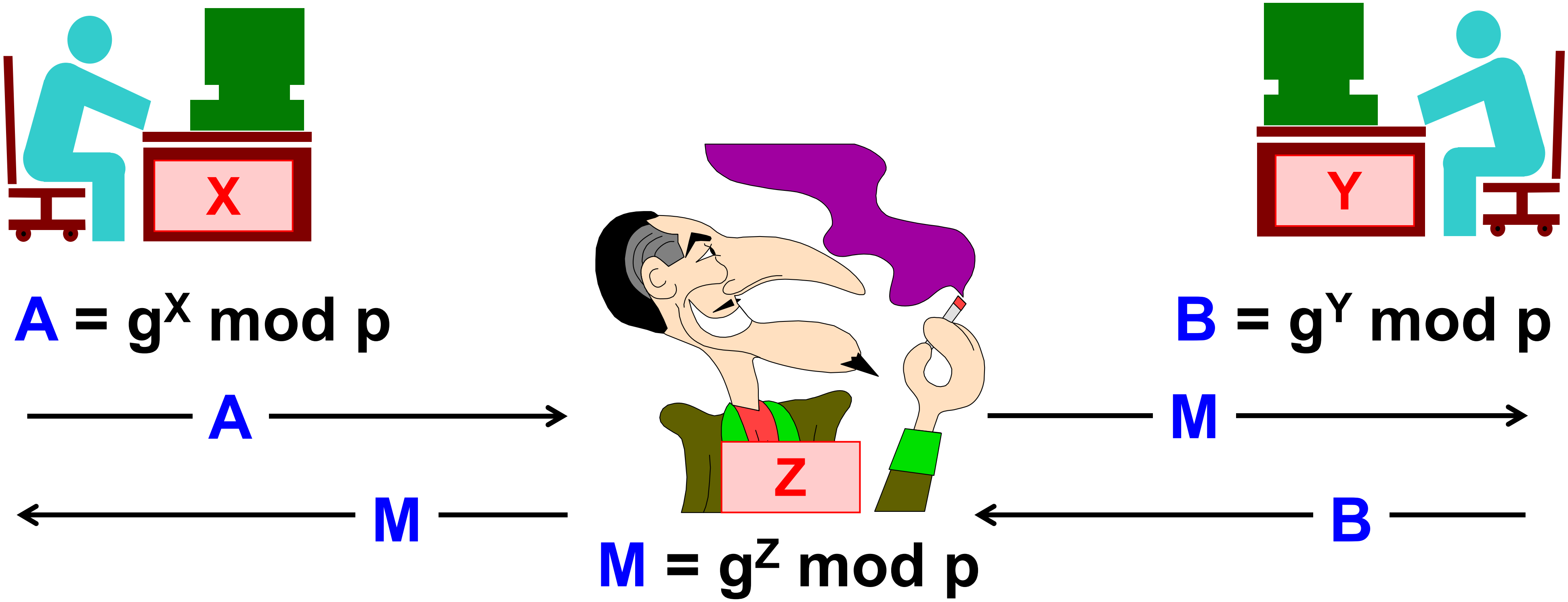
$K_B = A^Y \bmod p$

$$\boxed{K_A = K_B = g^{XY} \bmod p = K_{AB}}$$

Diffie-Hellman

- **first public-key algorithm invented**
- **frequently used to agree on a secret key (*key agreement*)**
- **patented in the USA but the patent expired on 29 April 1997**
- **resistant to the sniffing attack**
- **if the attacker can manipulate the data then it is possible to make a *man-in-the-middle* attack; in this case it requires pre-authentication**
  - certificates for DH keys
  - authenticated DH = MQV (Menezes-Qu-Vanstone) patented by CertiCom

DH: man-in-the-middle attack



$A = g^X \bmod p$

$B = g^Y \bmod p$

$\xrightarrow{\quad A \quad}$ $\xrightarrow{\quad M \quad}$

$\xleftarrow{\quad M \quad}$ $\xleftarrow{\quad B \quad}$

$M = g^Z \bmod p$

$K_A = M^X \bmod p$

$K_B = M^Y \bmod p$

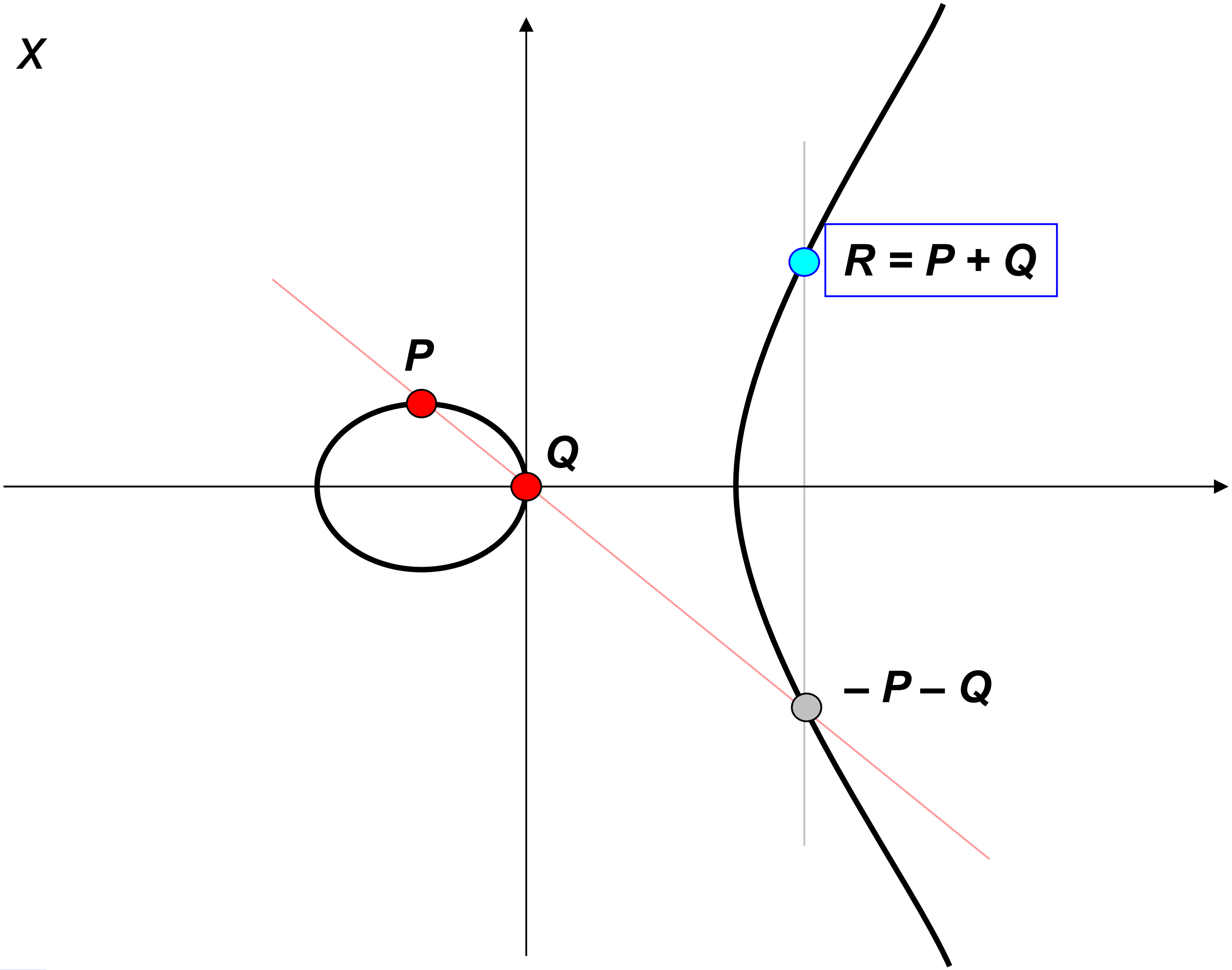
$\boxed{K_{AM} = g^{XZ} \bmod p} \neq \boxed{K_{BM} = g^{YZ} \bmod p}$

Elliptic curve cryptography

- **ECC (Elliptic Curve Cryptosystem)**
- **instead of using modular arithmetic, the operations are executed on the surface of a 2D (elliptic) curve**
- **problem of discrete logarithm on such a curve**
  - more complex than problem in modular arithmetic
  - possible to use shorter keys (about 1/10)
- **digital signature = ECDSA**
- **key agreement = ECDH**
- **authenticated key agreement = ECMQV (patented)**
- **key distribution = ECIES (EC Integrated Encryption Scheme)**

Elliptic curve arithmetics

$y^2 = x^3 - x$



Elliptic curve arithmetics

- **elliptic curve: $y^2 = x^3 + ax + b \quad (\bmod p)$ with $4a^3 + 27b^2 \neq 0$**
- **compute $R = (x, y) = P + Q$ given:**
  - $P = (x_P, y_P)$
  - $Q = (x_Q, y_Q)$
- **$x = \lambda^2 - x_P - x_Q \qquad y = \lambda (xP - x) - yP$**
  - $\lambda = (y_P - y_Q) / (x_P - x_Q)$ if $P \neq Q$ (i.e. P+Q)
  - $\lambda = (3x_P + a) / 2y_P$ if $P = Q$ (i.e. 2P)
- **so we can compute:**
  - addition of two points
  - multiplication of a point by a scalar

EC-Diffie-Hellman

- A and B select the same elliptic curve and a point G of its
- A chooses a random value x and computes:
  X = x G
- B chooses a random value y and computes:
  Y = y G
- A and B exchange (publish) X and Y
- A computes K = x Y
- B computes K' = y X
- but K = K' = x y G
- note: uses only scalar multiplication of a point

Sony PS3 hacking

- PS3 has embedded Linux with loader verifying the binaries' ECDSA signature before execution
- generation of an ECDSA signature requires a random nonce, otherwise the private key can be computed from the signature (!!!)
- … but Sony uses a fixed "random" (!!!)
- consequence: private key computed and distributed world-wide, so that anybody can run his own binaries on her PS3

"Console Hacking 2010 - PS3 Epic Fail" by fail0verflow
http://events.ccc.de/congress/2010/Fahrplan/events/4087.en.html

Who's who in crypto

Adi Shamir

Ron Rivest

Len Adleman

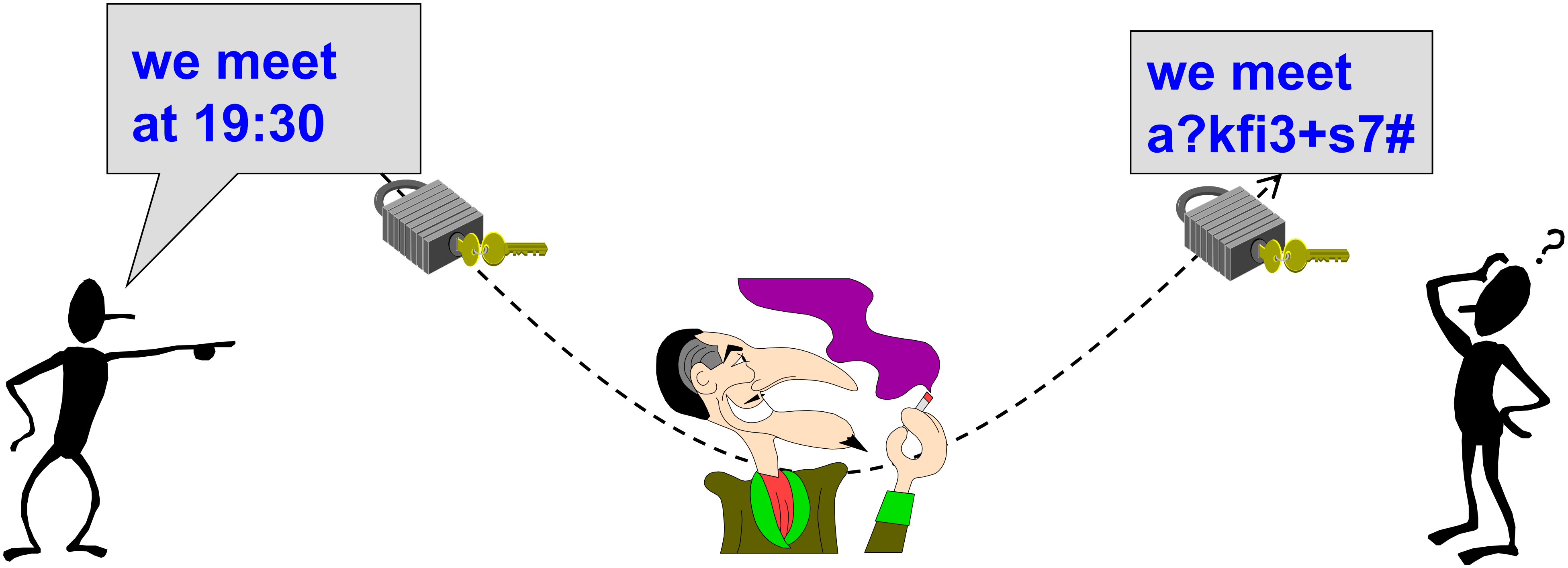
Ralph Merkle

Martin Hellman

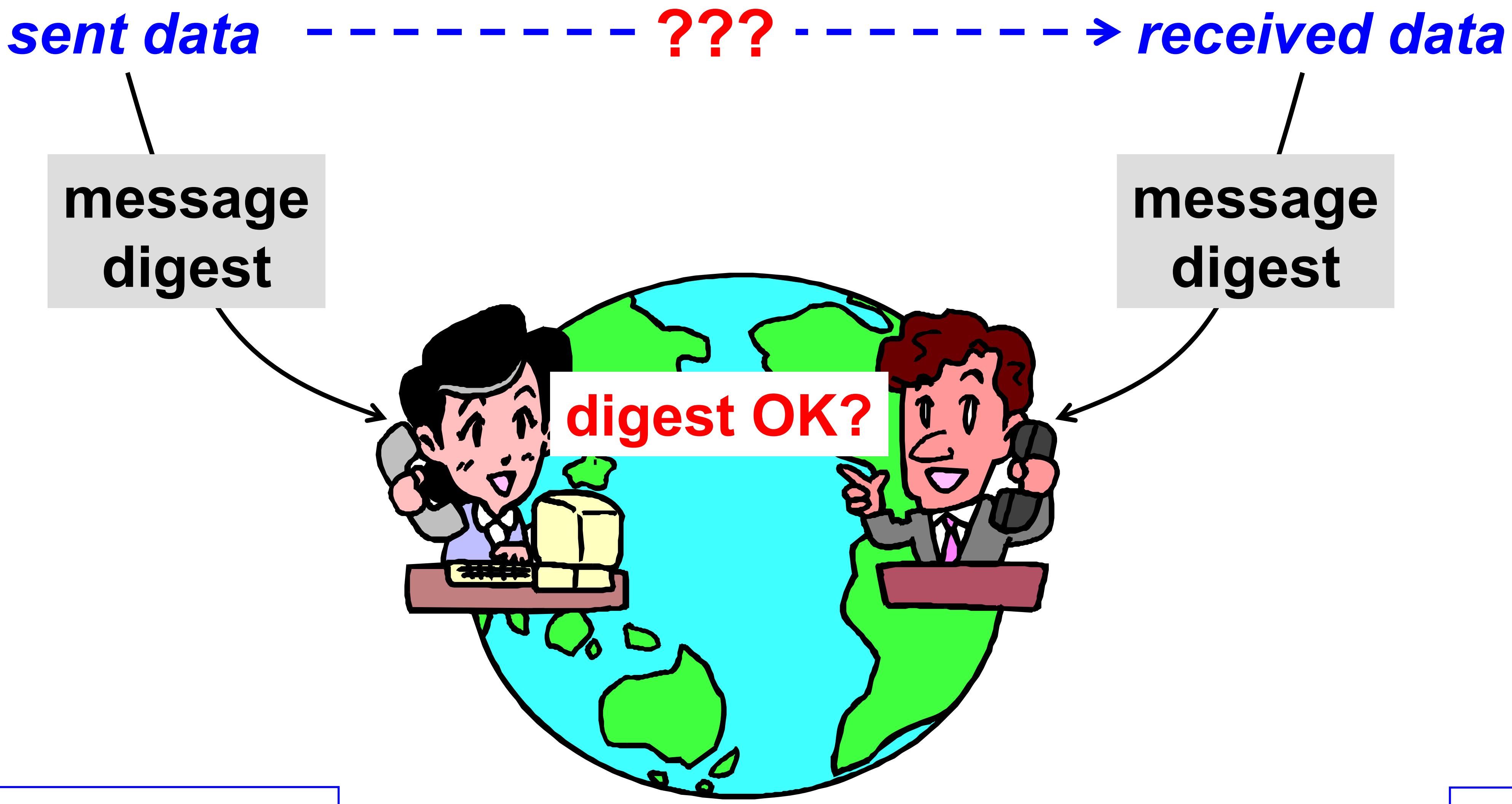
Whit Diffie

Message integrity

- a person that intercepts an encrypted communication cannot read it ...
- ... but can modify it in an unpredictable way!



Digest



sent data - - - - - - - - ??? - - - - - - - - → *received data*

message digest

message digest

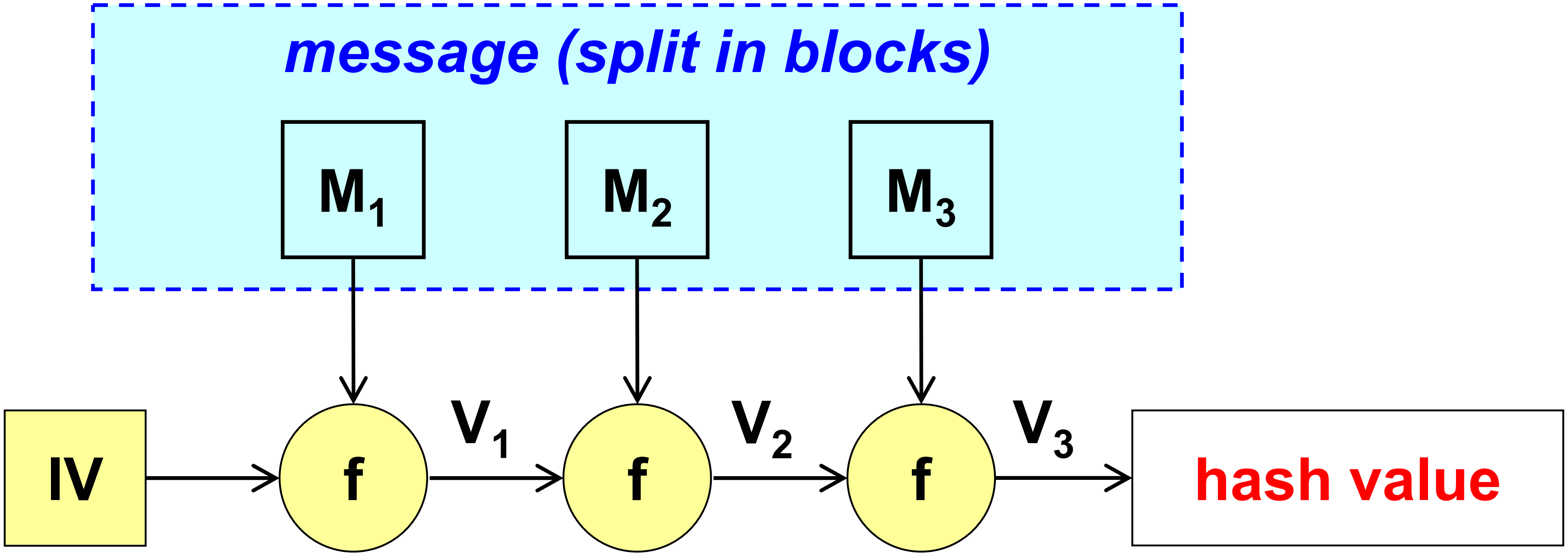
digest OK?

Message digest and hash functions

- **the message digest is a fixed-length "summary" of the message to be protected (of any length)**
- **digest can be calculated in many ways, but usually via a (cryptographic) hash function**
- **the function must be:**
  - fast to compute
    - of course ☺
  - have pre-image resistance
    - difficulty to deduce the input from the output
  - have collision-resistance
    - difficulty to find two inputs producing the same output

Hash functions (dedicated)

- **usually:**
  - split the message M in N blocks $M_1 \ldots M_N$
  - iteratively apply a base function (f)
  - $V_k = f(V_{k-1}, M_k)$ with $V_0 = IV$ and $h = V_N$



Padding in cryptographic hash functions

- **each function defines its own padding strategy**
- **for example, the SHA-1 function has a 512-bit block and uses the following strategy:**
  - block has size B and fragment has size L < 512
  - … append a bit with value 1, followed by how many bits with value 0 as needed
  - … and then insert the size L in the last 64 bit
  - note: one fragment could generate not one but two blocks (when L > 512 – 1 – 1 – 64 = 446)

Cryptographic hash functions

name	*block*	*digest*	*definition*	*notes*
MD2	8 bit	128 bit	RFC-1319	obsolete
MD4	512 bit	128 bit	RFC-1320	obsolete
MD5	512 bit	128 bit	RFC-1321	obsolete
RIPEMD-160	512 bit	160 bit	ISO/IEC 10118-3	old + rare
SHA-1	512 bit	160 bit	FIPS 180-1	sufficient
SHA-224	512 bit	224 bit		
SHA-256	512 bit	256 bit	FIPS 180-2	good
SHA-384	512 bit	384 bit	FIPS 180-3	
SHA-512	512 bit	512 bit		
	SHA-2			
SHA-3	1152-576	224-512	FIPS 202 FIPS 180-4	excellent

SHA-1 broken

February 15, 2005
SHA-1 has been broken. Not a reduced-round version. Not a simplified version. The real thing.
The research team of Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu (mostly from Shandong University in China) have been quietly circulating a paper describing their results:
– collisions in the the full SHA-1 in 2**69 hash operations, much less than the brute-force attack of 2**80 operations based on the hash length.
– collisions in SHA-0 in 2**39 operations.
– collisions in 58-round SHA-1 in 2**33 operations.
This attack builds on previous attacks on SHA-0 and SHA-1, and is a major, major cryptanalytic result. It pretty much puts a bullet into SHA-1 as a hash function for digital signatures (although it doesn't affect applications such as HMAC where collisions aren't important).
The paper isn't generally available yet. At this point I can't tell if the attack is real, but the paper looks good and this is a reputable research team.

`http://www.schneier.com/blog/archives/2005/02/sha1_broken.html`

SHA-2

- **as a quick fix after the SHA-1 attack, the SHA-2 family was developed by making the digest longer:**
  - SHA-2 = {SHA-224, SHA-256, SHA-384, SHA-512}
  - SHA-256 uses a 32-bit word
  - SHA-512 uses a 64-bit word
  - SHA-224/-384 are the truncation of SHA-256/-512
- **competition for a new dedicated hash function:**
  - named SHA-3
  - … but based on a completely different design than SHA-1 and SHA-2

Digest length

- **important to avoid *aliasing* (=collisions):**
  - md1 = h(m1)
  - md2 = h(m2)
  - if m1 ≠ m2 then we'd like to have md1 ≠ md2
- **if the algorithm is well designed and generates a digest of N bits, then the probability of aliasing is:**

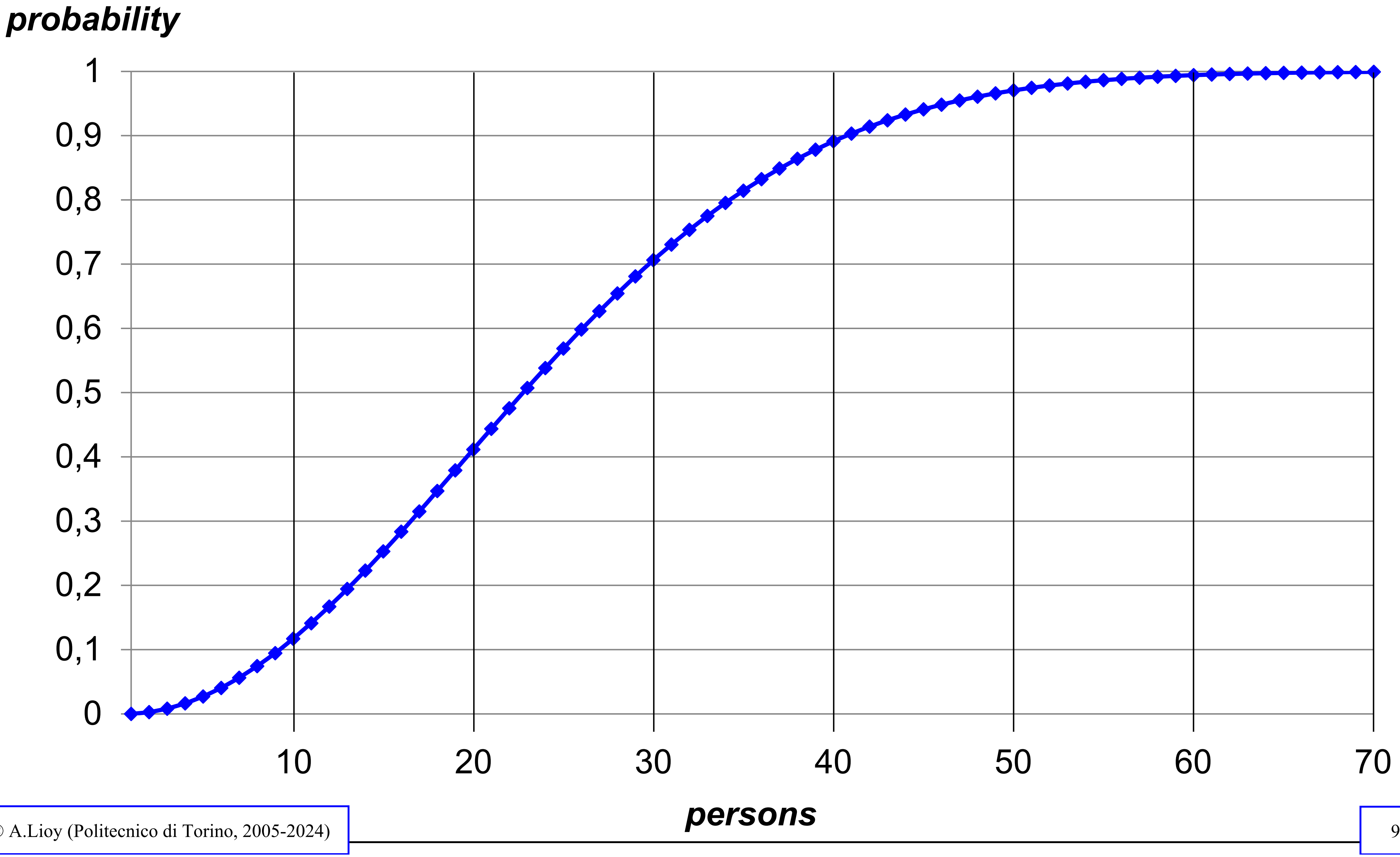
$$P_A \propto 1 / 2^{Nbit}$$

- **thus, digests with many bits are required (because statistical events are involved)**

The birthday paradox

- **if there are at least 23 persons in the same room, then the probability that 2 of them were born in the same day is greater than 50%; with 30 persons the probability is greater than 70%**
- **why? subtract from certainty (1) the probability that the 2nd, 3rd, 4th, … person was not born on the same day of any of the preceding ones**
  - $P(2) = 1 – 364/365$
  - $P(3) = 1 – 364/365 \cdot 363/365$
  - $P(N) = 1 – 364/365 \cdot 363/365 \cdot \ldots \cdot (365–N+1)/365$
  - $P(N) = 1 – [364 \cdot 363 \cdot 362 \cdot (365–N+1)] / 365^{N–1}$

The birthday paradox



The birthday attack

- **a N-bit digest algorithm is insecure when more than 2**(N/2) digests are generated because the probability to have two messages with the same digest is P_A~50%**
- **a cryptosystem is "balanced" when the encryption and digest algorithms have the same resistance:**
  - SHA-256 and SHA-512 have been designed for use respectively with AES-128 and AES-256
  - note: SHA-1 (i.e. SHA-160) matched Skipjack-80

SHA-3

- **candidates: 64 (oct'08) > 51 (dec'08) > 14 (jul'09)**
- **five finalists (dec'10)**
  - BLAKE, Grøstl, JH, Keccak, Skein
- **(2-oct-2012) and now the winner is …**
- **… Keccak (pronounce: “catch-ack”)**
  - authors = G.Bertoni, J.Daemen, G. Van Assche (STM), M.Peeters (NXP)

The NIST team praised the Keccak algorithm for its many admirable qualities, including its elegant design and its ability to run well on many different computing devices. The clarity of Keccak’s construction lends itself to easy analysis (during the competition all submitted algorithms were made available for public examination and criticism), and Keccak has higher performance in hardware implementations than SHA-2 or any of the other finalists.

The SHA-3 family

- **Keccak used to define various hash functions (FIPS-202):**

function	*output*	*block*	*capacity*	*definition*	*strength*
SHA3-224(M)	**224**	**1152**	**448**	**Keccak[448](M || 01, 224)**	**112**
SHA3-256(M)	**256**	**1088**	**512**	**Keccak[512](M || 01, 256)**	**128**
SHA3-384(M)	**384**	**832**	**768**	**Keccak[768](M || 01, 384)**	**192**
SHA3-512(M)	**512**	**576**	**1024**	**Keccak[1024](M || 01, 512)**	**256**
SHAKE128(M,d)	**d**	**1344**	**256**	**Keccak[256](M || 1111, d)**	**min(d/2,128)**
SHAKE256(M,d)	**d**	**1088**	**512**	**Keccak[512](M || 1111, d)**	**min(d/2,256)**

- **and more (NIST SP.800-185):**
  - cSHAKE (customizable domain parameters)
  - KMAC (keyed-digest)
  - TupleHash (hash of a tuple, where sequence matters)
  - ParallelHash (fast hash by exploiting internal CPU parallelism)

KDF (Key Derivation Function)

- **a cryptographic key must be random (each bit has 50% probability to be 0 or 1)**
- **… but users typically insert passwords (or better passphrases) guessable and not random**
- **K = KDF (P, S, I)**
  - P = password or passphrase
  - S = salt (to make K difficult to guess given P)
  - I = no. of iterations of the base function (to slow down the computation and make life complex for attackers)
- **KDF based upon cryptographic hash functions:**
  - PBKDF2 (RFC-2898) uses SHA-1, |S| ≥ 64, I ≥ 1000
  - HKDF (RFC-5869) uses HMAC

PBKDF2

- **RFC-8018, replaces PBKDF1 (limited to keys <= 160 bit)**
- **DK = PBKDF2(PRF, PWD, Salt, C, dkLen)**
  - PRF = pseudorandom function of two parameters with output length hLen (e.g. a keyed HMAC)
  - PWD = password from which a derived key is generated
  - Salt = cryptographic salt
  - C = number of iterations desired
  - dkLen = desired length of the derived key
  - DK = generated derived key = $T_1 \| T_2 \| \ldots \| T_{dkLen/hLen}$ (where each | Ti | = hLen)

PBKDF2 parameters and applications

- (WPA2) DK = PBKDF2(HMAC−SHA1, PSK, SSID, 4096, 256)
- C = 4096 (Kerberos, 2005), 2k (iOS3), 10k (iOS4), 5k (LastPass JS client, 2011), 100k (LastPass server), 600k HMAC-SHA256 and 210k with HMAC-SHA512 (OWASP, 2023)
- (NIST SP800-132, 2018) | S | >= 128

Password Hashing Competition (PHC)

- **PBKDF2 can be attacked because C may be large but this requires only a lot of computation but not a lot of RAM**
  - hence it can be attacked by ASIC or GPU
  - we need to increase the RAM needed for the attack
- **public competition launched in 2013**
- **https://password-hashing.net/**
- **20 July 2015 the winner is**
  - Argon2
  - "special mention also to Catena, Lyra2, yescrypt and Makwa"
- **other alternatives:**
  - Balloon hashing (recommended by NIST SP800-63B, 2021)
  - Oblivious PRF (i.e. two-party on-line computation)

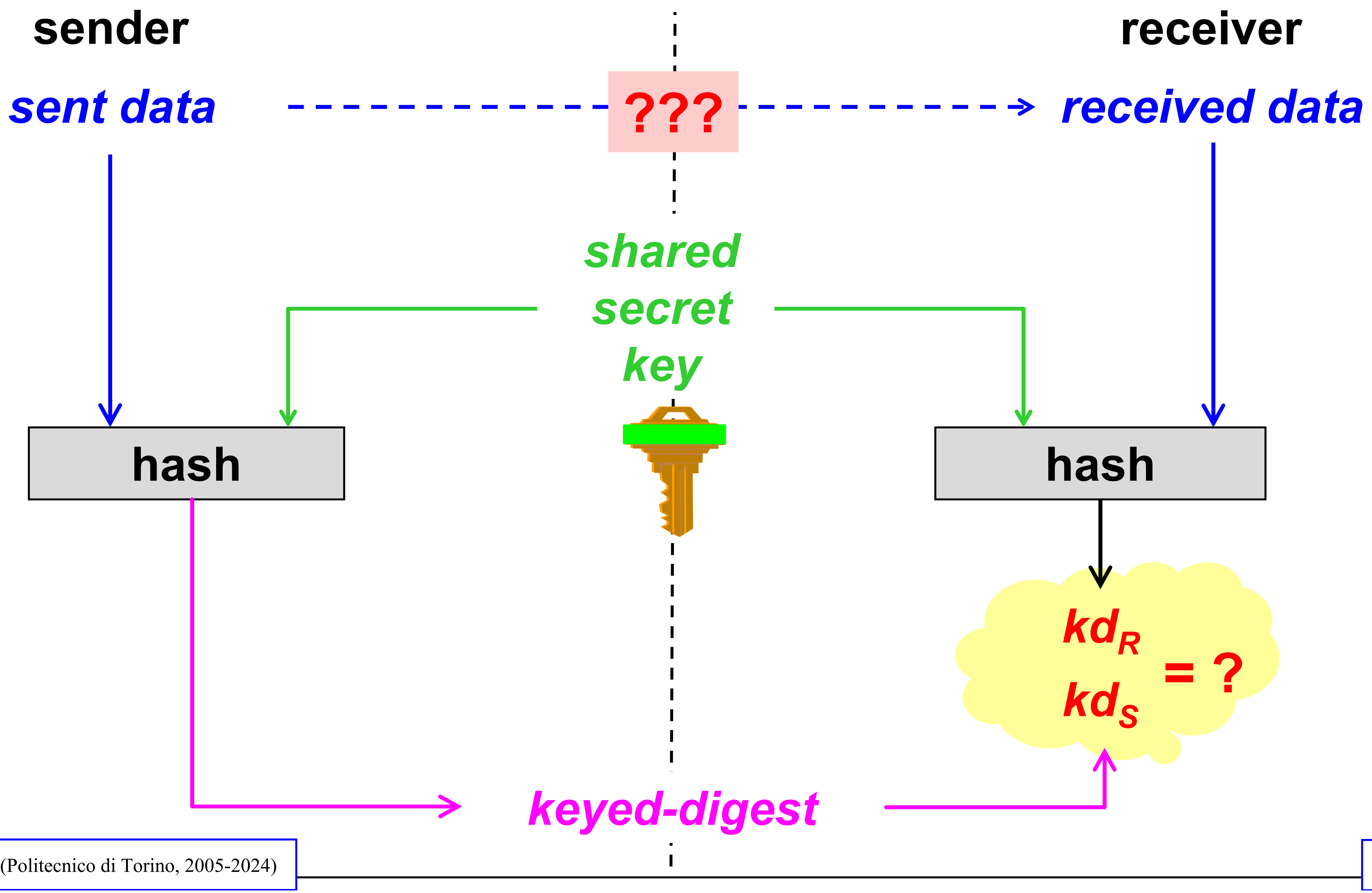
MAC, MIC, MID

- to guarantee the integrity of messages, a code is added to the message:
  MIC (Message Integrity Code)
- often integrity is not useful without authentication, thus the code (ensuring both security properties) is named:
  MAC (Message Authentication Code)
- to avoid replay attacks, a unique identifier can be added to the message:
  MID (Message IDentifier)

Authentication by means of keyed-digest

- **send also a digest calculated not only on data but also on a shared secret key**
  - (sender) d = digest(K, M)
  - (transmission) M || d
  - (receiver) d' = digest(K, M)
  - (verification) if (d == d') then OK else ALARM!
- **only who knows the key can compare the transmitted digest with the digest calculated on the received data**
- **advantages:**
  - only one operation (digest)
  - few additional data
  - authentication + integrity

Keyed-digest



Keyed-digest: HMAC

- **RFC-2104 (also FIPS-198)**
- **base hash function H:**
  - B byte block, L byte output, with B > L
  - e.g. for SHA-256 we have B=64 and L=32
- **definitions:**
  - ipad = 0x36 repeated B times
  - opad = 0x5C repeated B times
  - deprecated keys s.t. | K | < L
  - if | K | > B then K' = H (K) else K' = K
  - if | K' | < B then K' is 0-padded up to B bytes
- **hmac-H = H(K' ⊕ opad || H(K' ⊕ ipad || data))**

CBC-MAC

- exploits a block-oriented symmetric encryption algorithm, in CBC mode with null IV, taking as MAC the last encrypted block
- message M split in N blocks $M_1 \ldots M_N$
- iterations:
  - $V_0 = 0$
  - for (k=1…N) do $V_k = \text{enc}(K, M_k \oplus V_{k-1})$
- cbc-mac = V_N
- DES-based CBC-MAC was the Data Authentication Algorithm (standard FIPS 113, ANSI X9.17)

CBC-MAC insecurity

- **secure only for fixed-length messages**
  - for other cases use CMAC or OMAC
- **possible attack against variable-length messages:**
  - if t = E-cbc-mac(K, M) and t' = E-cbc-mac(K, M')
  - then I can create M'' with t'' = E-cbc-mac(K, M'')'
  - … without knowing K (!!!)
- **operational proof:**
  - if I create M'' = M || (M'_1 xor t) || M'_2 || … || M'_N
  - then I get t'' = E-cbc-mac(K, M'') = t' because xor-ing the first block of M' with t will actually delete its contribution
- **https://en.wikipedia.org/wiki/CBC-MAC**

Integrity and secrecy: how to combine?

- **distinct operations by hypothesis:**
  - secrecy = symmetric encryption with K_1
  - integrity = keyed-digest (MAC) with K_2

- **option 1 – authenticate-and-encrypt (A&E)**
  - enc(K_1 , p) || mac(K_2 , p)
  - must always decrypt before checking integrity (possible DoS attack)
  - may leak info about the plaintext
  - e.g. used by SSH

Integrity and secrecy: how to combine?

- **option 2 – authenticate-then-encrypt (AtE)**
  - enc(K_1 , p || mac(K_2 , p))
  - must always decrypt before checking integrity (possible DoS attack)
  - e.g. used by SSL and TLS
- **option 3 – encrypt-then-authenticate (EtA)**
  - enc(K_1 , p) || mac(K_2 , enc(K_1 , p))
  - can avoid decryption if MAC is wrong
  - e.g. used by IPsec

Integrity and secrecy: security?

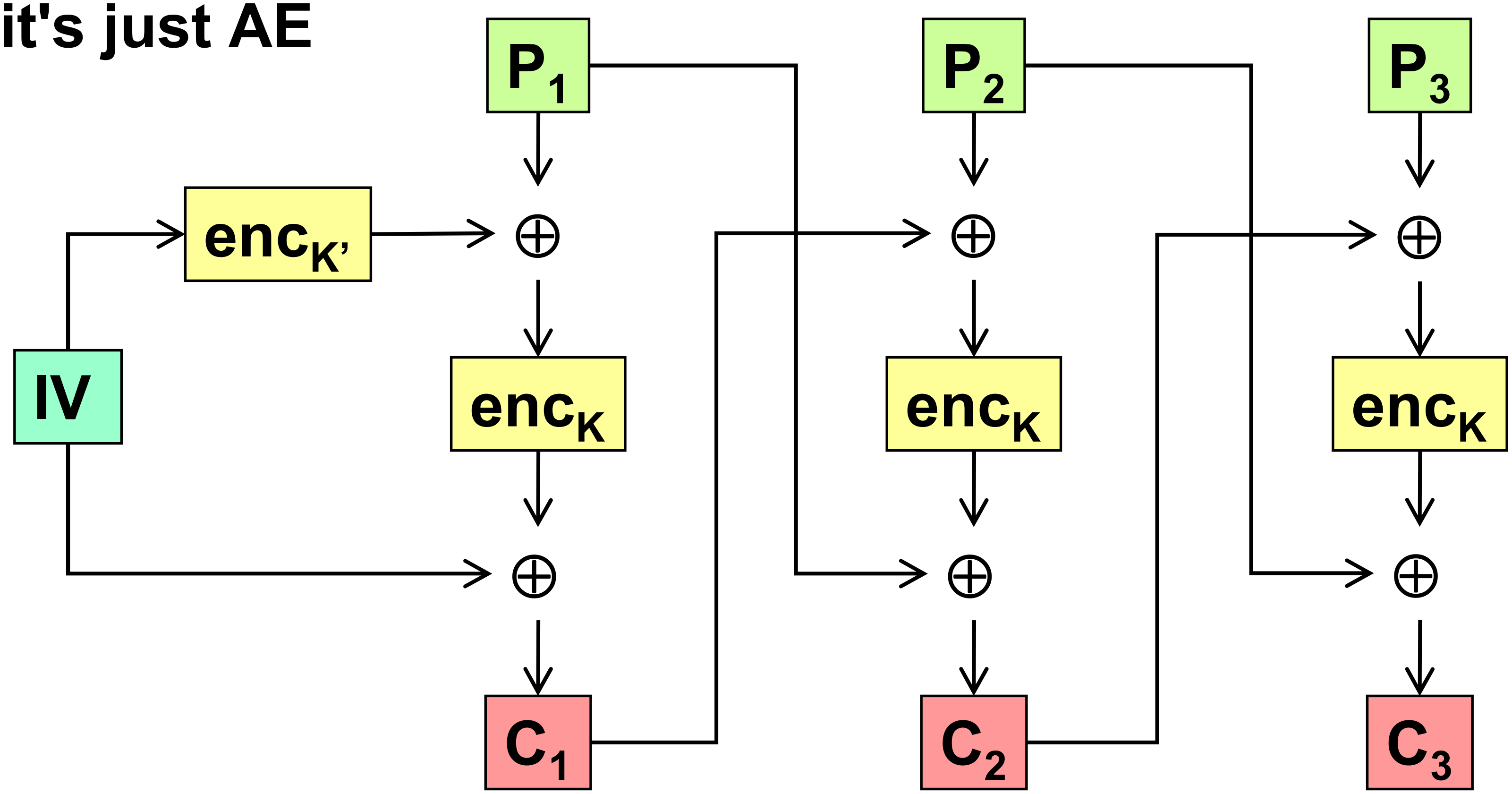
- **improper combination of secure algorithms may lead to … an insecure result!**
- **authenticate-and-encrypt (A&E)**
  - insecure unless performed in a single step
- **authenticate-then-encrypt (AtE)**
  - secure only with CBC or stream encryption
- **encrypt-then-authenticate (EtA)**
  - the most secure mode … but beware of implementation errors
    - always include IV and algorithms in the MAC computation
- **current efforts towards a joint AE algorithm**

Authenticated Encryption (AE)

- **a single operation for privacy and authentication/integrity:**
  - just one key and one algorithm
  - better speed
  - less error likelihood in combining the two functions
- **the normal encryption modes are subject to chosen-ciphertext attacks when used on-line:**
  - the attacker modifies a ciphertext
  - then observes if the receiver signals an error or not (e.g. padding oracle or decryption oracle attack)
  - so we need to verify the integrity of the ciphertext before decrypting it
- **AE = same data for the two properties**

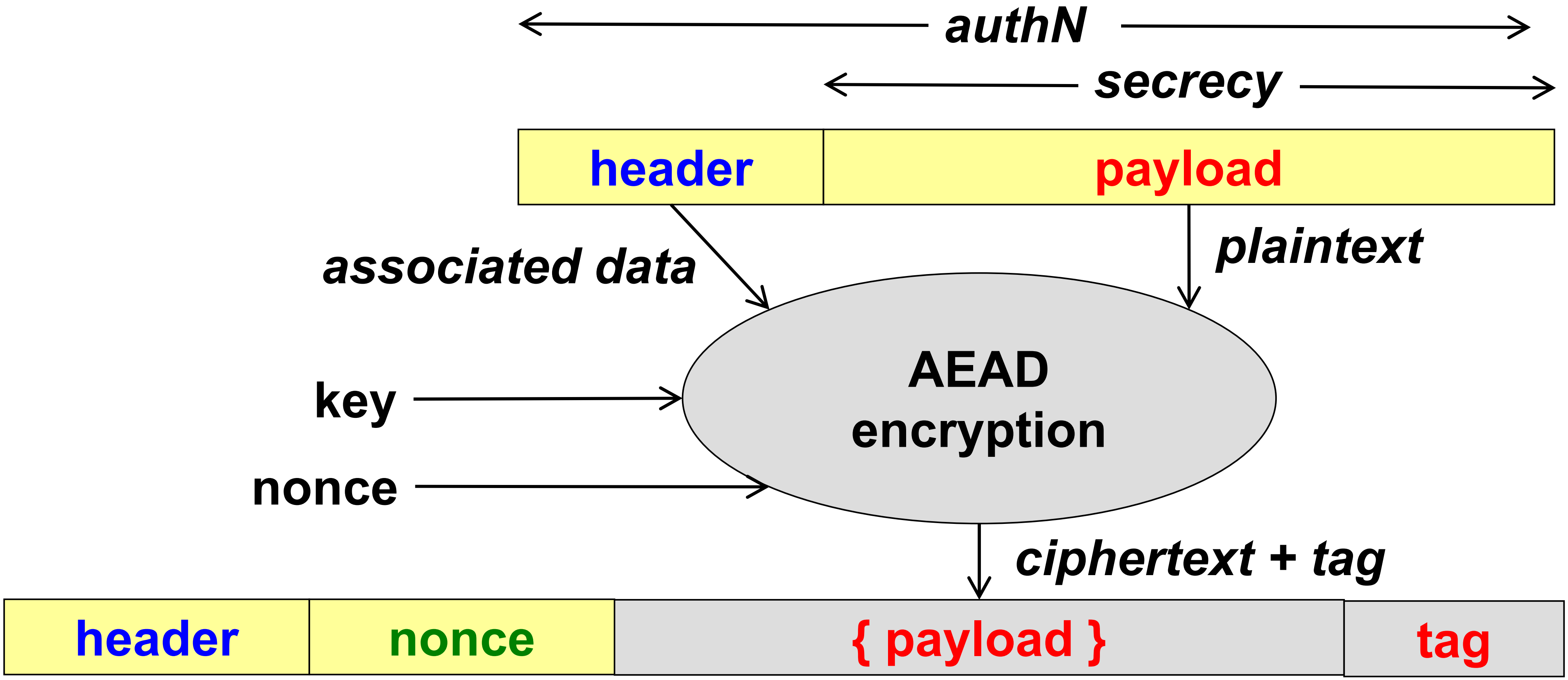
IGE (Infinite Garble Extension)

- error on every block after the mangled one
- easy to add a last control block (e.g. all zeroes, counter of the number of blocks)
- it's just AE



RFC 5116 - Interface and algorithms for authenticated encryption

- Authenticated Encryption with Associated Data (AEAD)



Authenticated encryption: standards

- **ISO/IEC 19772:2009 defines 6 standard modes:**
  - OCB 2.0 (Offset Codebook Mode)
    [single-pass AEAD, patented]
  - AESKW (AES Key Wrap)
  - CCM (CTR mode with CBC-MAC)
  - EAX (Encrypt then Authenticate then X(trans)late) = CTR + OMAC [double-pass AEAD, free]
  - Encrypt-then-MAC
  - GCM (Galois/Counter Mode)
- **other modes exist and are/will be recommended by other bodies (e.g. NIST, IETF)**

GCM as an example of AEAD

- **(C, T) = algo_GCM_enc (K, IV, P, A)**
  - IV size [1 … 2^{64} bits] (96 bits is most efficient)
  - P size [0 … 2^{39} – 256 bits]
  - A (associated authenticated data) size [0 … 2^{64} bits]
  - C has the same size as P
  - T is the authentication tag with size [0 … 128 bits]

- **P = algo_GCM_dec (K, IV, C, A, T)**
  - P is the original plaintext (if authentication is OK)
  - … or a special value FAIL if the authentication check failed

- **GCM defined for encryption algorithms with 128-bit block**

CCM as an example of AEAD

- **CCM = Counter-mode with CBC-MAC**
  - first an authentication tag of (plaintext + associated data) is computed by CBC-MAC
  - … then the plaintext and the tag are (separately) encrypted in CTR mode
- **CCM defined for algorithms with 128-bit block**

Authenticated encryption: applications

- **TLS-1.3 uses GCM and CCM**
- **802.11i uses CCM**
- **ZigBee uses CCM* (=CCM + auth-only + enc-only)**
- **ANSI C12.22 (network transmission of electronic measures, e.g. house power meter) uses EAX’**
  - broken !!!
    Minematsu, Morita and Iwata
    http://eprint.iacr.org/2012/018.pdf
  - EAX had a formal proof of its security … but ANSI sacrificed it for 3-5 less encryption steps and 40 bytes less memory usage (so important for embedded systems?)

Comparison of AE algorithms

- **GCM: the most popular, on-line single-pass AEAD, parallelizable, used by TLS, present in openssl**
  - for encryption, generates ciphertext + authentication tag
  - for decryption, first computes authentication tag and only if it matches the one in input then the ciphertext is decrypted
  - fast on Intel architecture (~4 cycle/byte) using AES-NI for encryption and PCLMULQDQ for the tag
- **OCB 2.0: the fastest one, on-line single-pass AEAD, GPL patented so scarcely used, now free but for military uses**
- **EAX: on-line double-pass AEAD, slow but small (uses just the encryption block) so very good for constrained systems**
- **CCM: off-line double-pass, the slowest one**
- **note: double-pass is 2x slower than one-pass (in software)**

AESKW

- **AES Key Wrap (RFC-3394)**
  - key (secret, private, seed) encryption for storage/transmission
  - key must be multiple of 64 bit
  - generates also a 64-bit authentication tag
- **operations:**
  - KEK = Key Encryption Key
  - CEK = Content Encryption Key
  - {CEK} + tag = aeskw-enc(KEK, CEK)
  - CEK (or FAIL) = aeskw-dec(KEK, {CEK}+tag)
- **why not a normal AE algo?**
  - simple (e.g. no RNG as in GCM, no asymmetric encryption)
  - supports in-place encryption/decryption

NIST lightweight crypto (LWC)

- **https://csrc.nist.gov/Projects/lightweight-cryptography**
- **launched August 2018, call for AEAD lightweight algorithms**
- **round 1 = 56 candidates**
  - NISTIR-8268, Status Report on the First Round of the NIST LWC Standardization Process (October 2019)
- **round 2 = 32 candidates**
  - NISTIR-8369, Status Report on the Second Round of the NIST LWC Standardization Process (July 2021)
- **(march 2021) ten finalists**
  - ASCON, Elephant, GIFT-COFB, Grain128-AEAD, ISAP, Photon-Beetle, Romulus, Sparkle, TinyJambu, Xoodyak
- **(7/2/2023) the winner is … ASCON**
  - usable for encryption, AEAD, hashing

ASCON

- **NOT to replace AES or SHA3, just for lightweight env**
- **for AEAD:**

	key	nonce	tag	rate	capacity	pa	pb
ASCON-128	128	128	128	64	256	12	6
ASCON-128a	128	128	128	128	192	12	8

- **for hashing:**

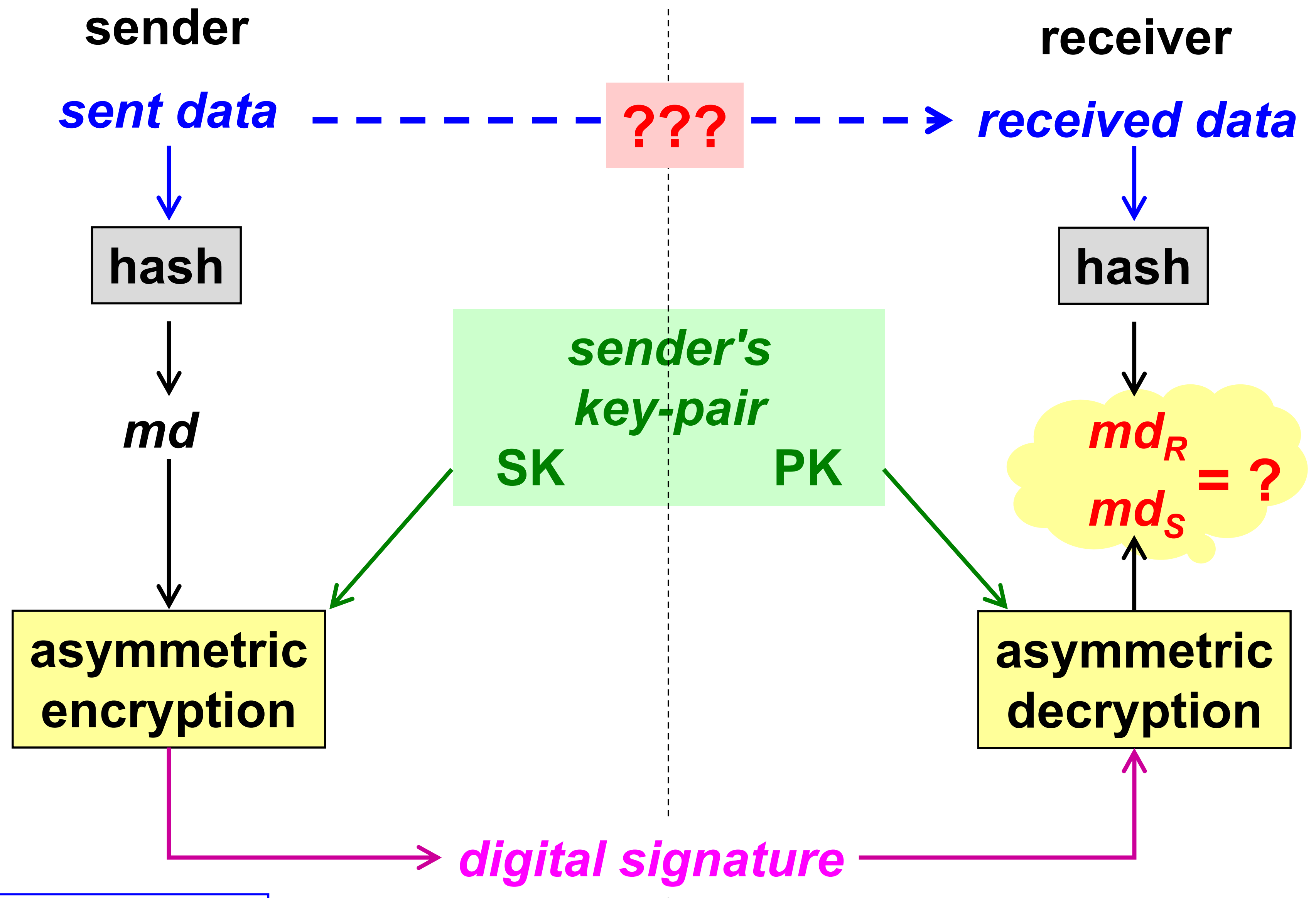
	output	rate	capacity	pa	pb
ASCON-hash	256	64	256	12	12
ASCON-xof	arbitrary	64	256	12	12
ASCON-hasha	256	64	256	12	8
ASCON-xofa	arbitrary	64	256	12	8

- **(pa and pb are two different permutation cycles of Ascon)**

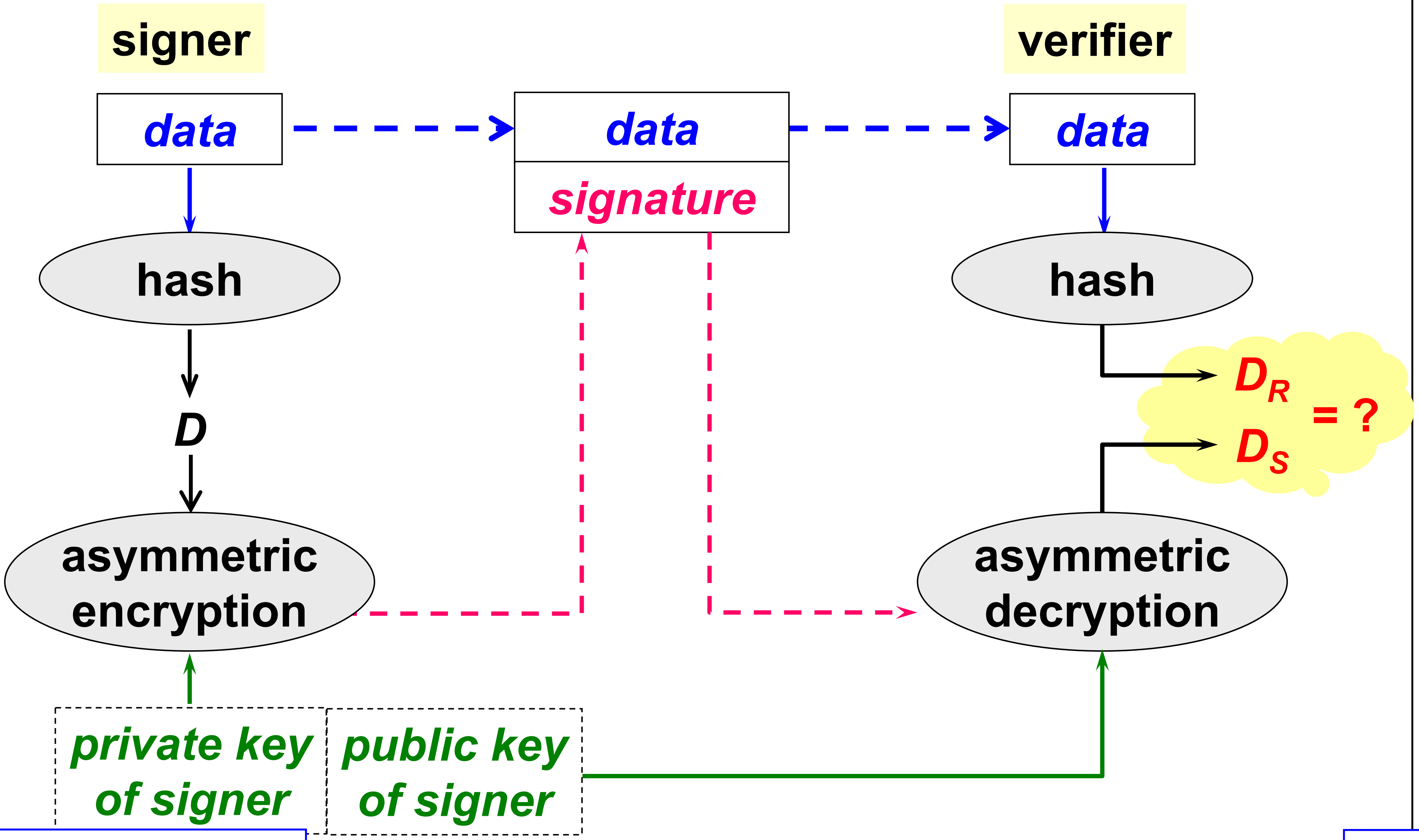
Authentication by digest and asymmetric encryption

- **send also a digest (encrypted with the private key of the sender)**
  - (signer S) H = enc(S.SK, hash(M))
  - (transmission) M || H
  - (verifier V) X = dec(S.PK, H)
  - (verification) if (X == hash(M)) then OK else ALARM!
- **those who know the public key can compare the transmitted digest with the digest calculated on the received data**
- **DIGITAL SIGNATURE !!!**

Digital signature



Signature creation and verification



Authentication and integrity: analysis

- **by means of a shared secret:**
  - useful only for the receiver
  - cannot be used as a proof without disclosing the secret key
  - not useful for non repudiation
- **by means of asymmetric encryption:**
  - being slow it is applied to the digest only
  - can be used as a formal proof
  - can be used for non repudiation
  - = *digital signature*

Digital vs. handwritten signature

- **digital signature = authentication + integrity**

- **handwritten signature = authentication**

- **thus the digital signature is better, because it is tightly bound to the data**

- **note: each user does not have a digital signature but a private key, which can be used to generate an infinite number of digital signatures (one for each different document)**

Public key certificate

> "A data structure used to securely bind a public key to some attributes"

- typically it binds a key to an identity ... but other associations are possible too (e.g. IP address)
- digitally signed by the issuer: the Certification Authority (**CA**)
- limited lifetime
- can be revoked on request both by the user and the issuer

Formats for public key certificates

- **X.509:**
  - v1, v2 (ISO)
  - v3 (ISO + IETF)
- **non X.509:**
  - PGP
  - SPKI (IETF)
- **PKCS#6:**
  - RSA, partly compatible with X.509
  - obsolete

Structure of a X.509 certificate

version	2
serial number	1231
signature algorithm	RSA with MD5, 1024
issuer	C=IT, O=Polito, OU=CA
validity	1/1/97 - 31/12/97

subject	C=IT, O=Polito,
	CN=Antonio Lioy
	Email=lioy@polito.it

subjectPublicKeyInfo	RSA, 1024, xx...x
CA digital signature	yy...y

PKI (Public-Key Infrastructure)

- is the infrastructure ...
- technical and administrative ...
- put in place for the creation, distribution and revocation of public key certificates

Certificate revocation

- **any certificate can be revoked before its expiration date:**
  - on request by the owner (subject)
  - autonomously by the creator (issuer)
- **when a signature is verified, the receiver must check that the certificate was valid at signature time**
- **this kind of check is the responsibility of the receiver (relying party, RP)**

Revocation mechanisms

- **CRL (Certificate Revocation List)**
  - list of revoked certificates
  - signed by the CA or by a delegated party
  - certificate validity since it was issued
- **OCSP (On-line Certificate Status Protocol)**
  - response containing information about the certificate status
  - signed by the server
  - certificate validity at the current time

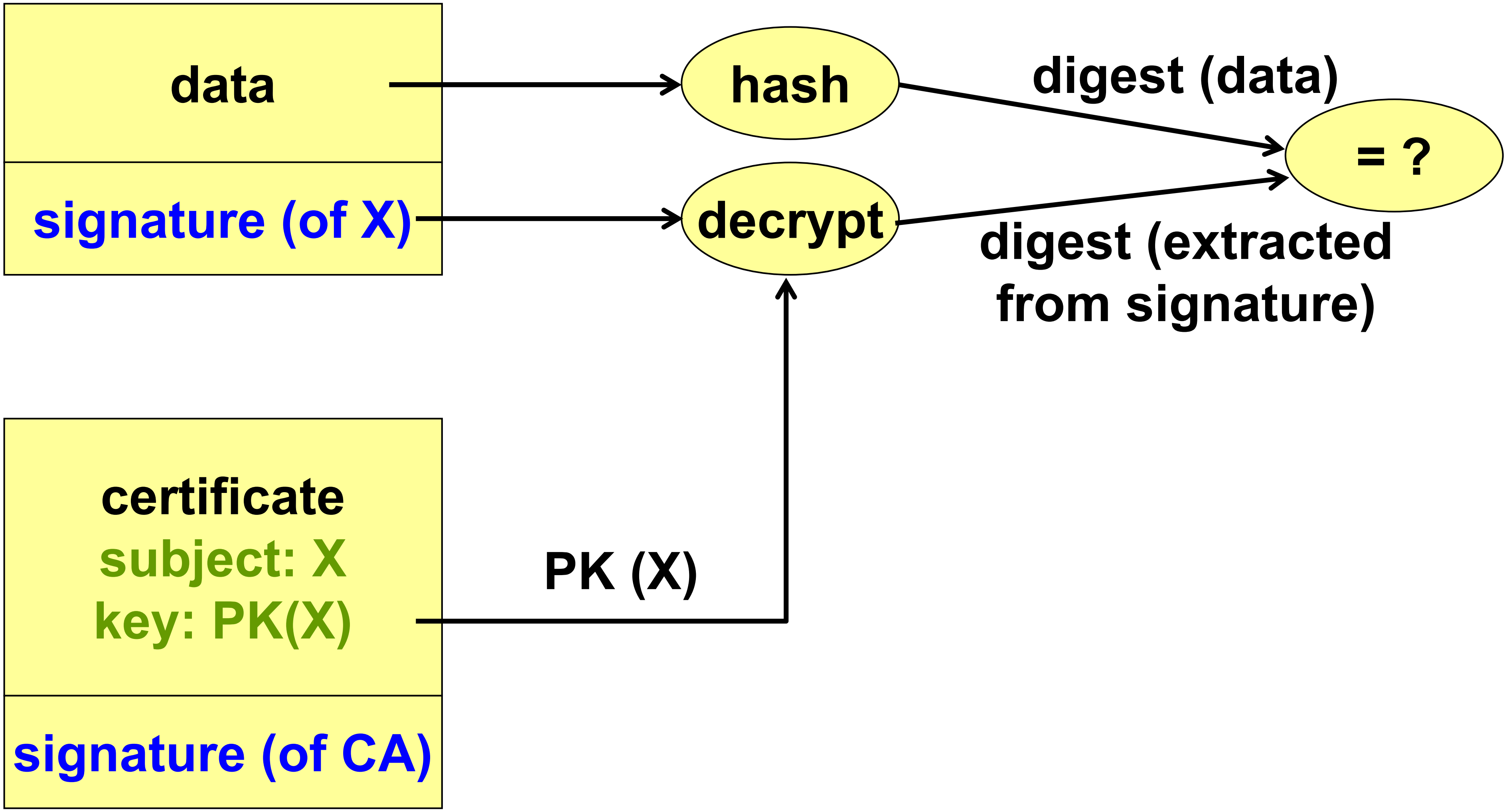
Structure of a X.509 CRL

version	1
signature algorithm	RSA with MD5, 1024
issuer	C=IT, O=Polito, OU=CA
thisUpdate	15/10/2000 17:30:00
userCertificate revocationDate	1496 13/10/2000 15:56:00
userCertificate revocationDate	1574 4/6/1999 23:58:00
CA digital signature	yy...y

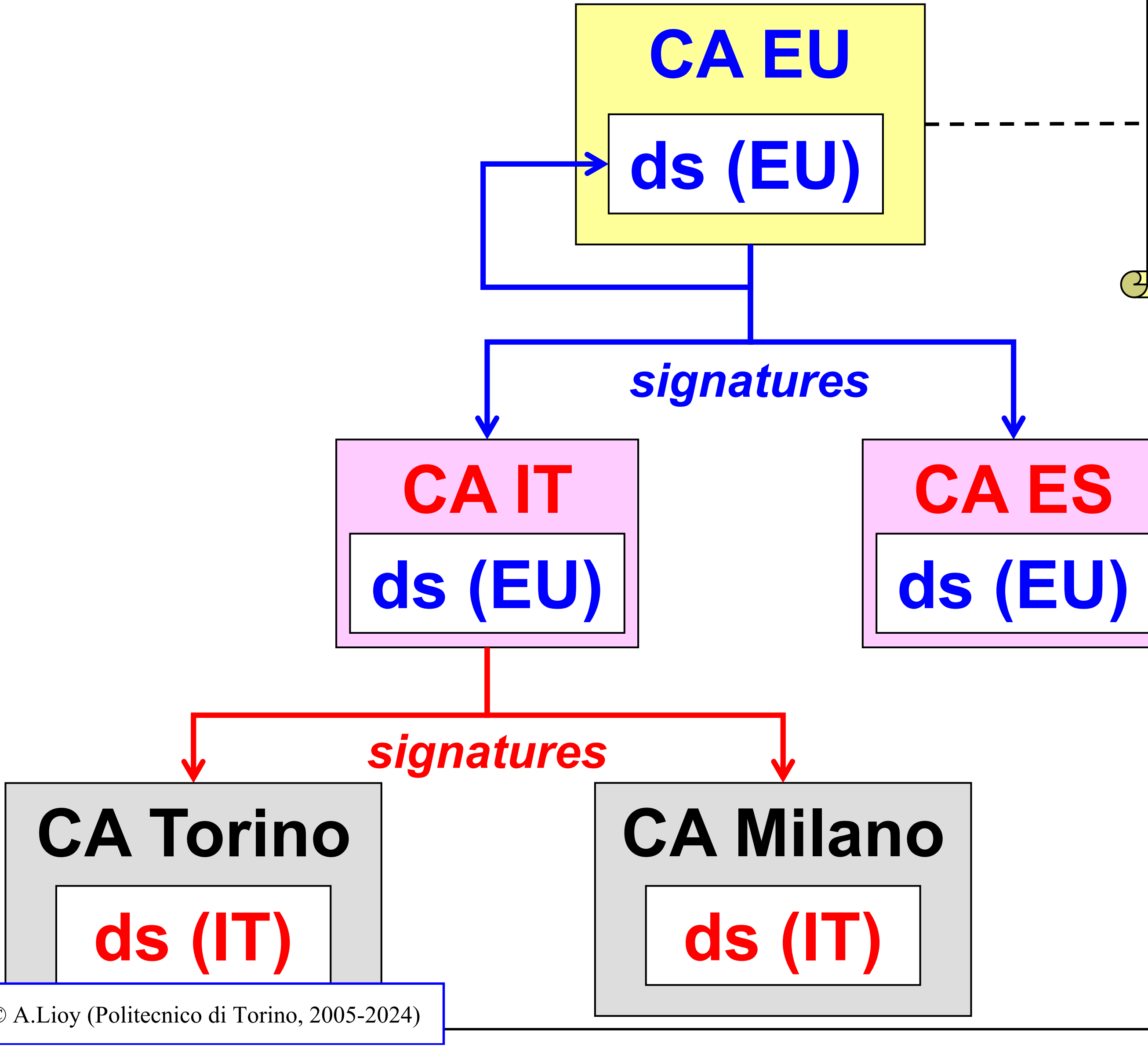
Verification of a signature / certificate

- how to verify that a public-key certificate (signed by CA1) is authentic?
- … the public-key certificate of CA1 is required (which will be signed by CA2)
- how to verify the last one ?
- … the public-key certificate of CA2 is required (which will be signed by CA3)
- … and so on …
- it becomes thus necessary to have an infrastructure (hierarchical?) for certification and distribution of public-key certificates – and the respective revocation information

Verification of a signature / certificate



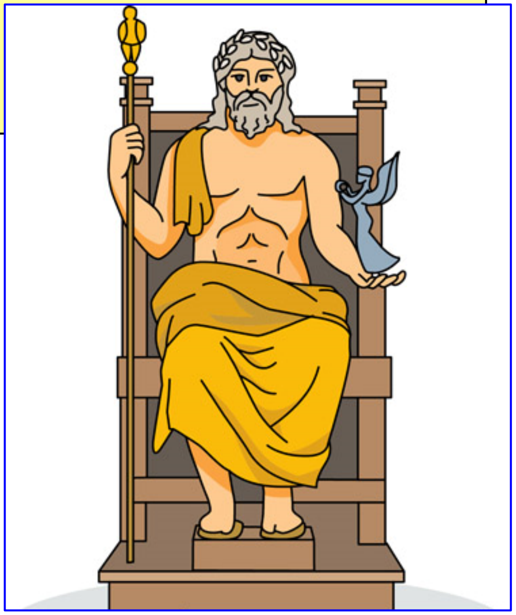
Certification hierarchy (chain of trust)



TLCA = Top-Level CA

(Trusted) Root CA

"God"



Performance

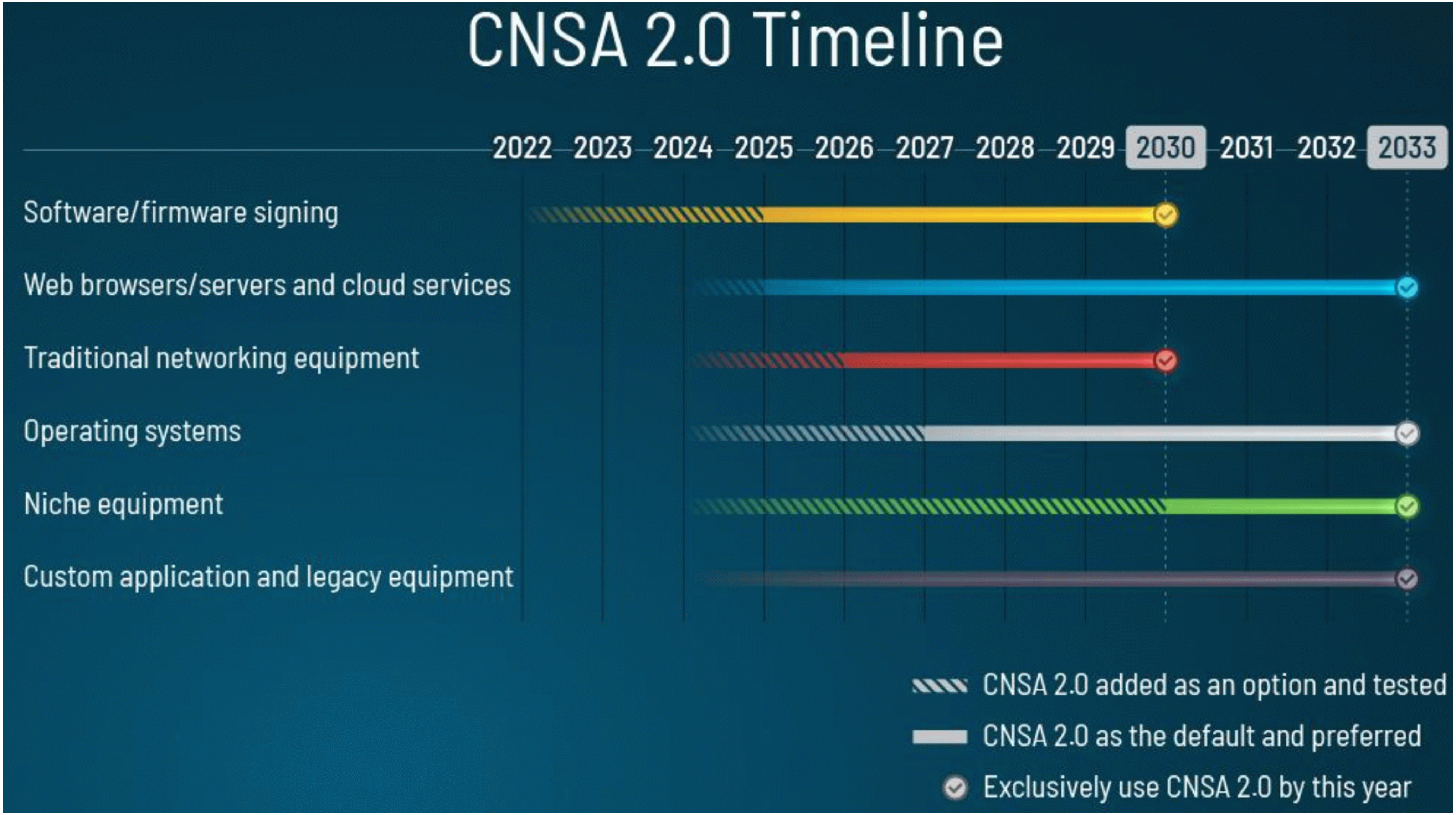
- **cryptographic performance does not depend on RAM but on CPU (architecture and instruction set) and cache size**
- **performance is not a problem on clients (except when not very powerful, e.g. IoT, or they are overloaded by local applications)**
- **performance can become a problem on the servers and/or on the network nodes (e.g. router):**
  - use cryptographic accelerators (HSM, Hardware Security Module)
    - special-purpose accelerators tied to a protocol (e.g. TLS, IPsec) or generic ones (e.g. AES, RSA)

CNSA (2018)

- **Commercial National Security Algorithm Suite**
- **includes the following algorithms:**
  - (symmetric encryption) AES-256
    - (flow) mode CTR (low bandwidth) or GCM (high bandwidth)
  - (hash) SHA-384
  - (key agreement) ECDH e ECMQV
  - (digital signature) ECDSA
  - EC on the curve P-384
- **for legacy systems**
  - (key agreement) DH-3072
  - (key exchange, digital signature) RSA-3072
  - new quantum-resistant algorithms expected by 2022

CNSA 2.0 (2022)

- **Commercial National Security Algorithm Suite, version 2**
- **includes the following algorithms:**
  - (symmetric encryption) AES-256
    - (flow) mode CTR (low bandwidth) or GCM (high bandwidth)
  - (hash) SHA-384 or SHA-512
  - (key agreement) CRYSTALS-Kyber w/ level V parameters
  - (digital signature) CRYSTALS-Dilithium w/ level V parameters
  - (digital signature of firmware and software)
    - all NIST SP 800-208 algorithms (LMS, XMSS)
    - suggested LMS SHA256/192


CNSA 2.0 Timeline
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
Software/firmware signing
Web browsers/servers and cloud services
Traditional networking equipment
Operating systems
Niche equipment
Custom application and legacy equipment
▨ CNSA 2.0 added as an option and tested
▬ CNSA 2.0 as the default and preferred
✓ Exclusively use CNSA 2.0 by this year

Bibliography

- B.Schneier: “Applied cryptography”
- A.Menezes, P. van Oorschot, S.Vanstone: “Handbook of Applied Cryptography”
- W.Stallings:
  “Cryptography and network security”
- C.P.Pfleeger, S.Pfleeger:
  “Security in computing”
- S.Garfinkel, G.Spafford:
  “Practical Unix and Internet security”
- W.R.Cheswick, S.M.Bellovin:
  “Firewalls and Internet security” (2nd ed.)
- R.Anderson, "Security Engineering",
  http://www.cl.cam.ac.uk/~rja14/book.html

Bibliography (in Italian)

- W.Stallings
  “Sicurezza delle reti - applicazioni e standard”
  Addison-Wesley Italia, 2004
- C.Pfleeger, S.Pfleeger
  “Sicurezza in informatica”
  Pearson Education Italia, 2004
- Fugini, Maio, Plebani
  “Sicurezza dei sistemi informativi”
  Apogeo
- S.Singh
  “Codici e segreti”
  BUR saggi