

# Sécurité des systèmes informatiques

Éléments de cryptographie

**Prof. Jean-Noël Colin**

Université de Namur

[www.unamur.be](http://www.unamur.be)



# Agenda

## Cryptography

### Introduction

Random numbers

Symmetric cryptography

Key management

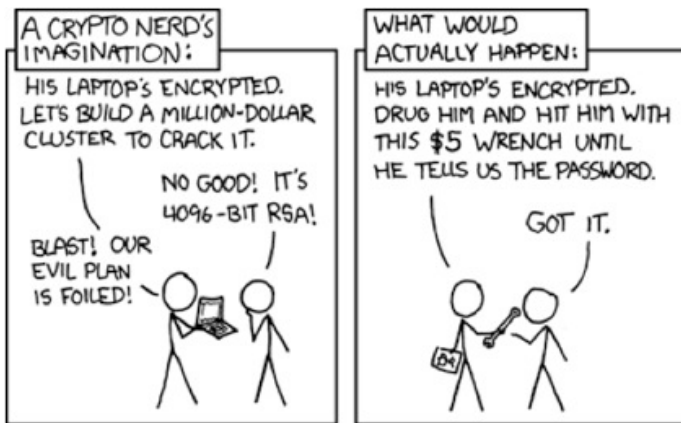
Asymmetric cryptography

Digital signature

Public key infrastructure

Applications

Conclusion



# Some vocabulary

**Encrypt** with the help of an encryption key, transform cleartext data into encrypted data, making it unreadable for anyone not able to decrypt it

**Decrypt** with the help of a decryption key, recover original cleartext data from encrypted text data

**Encryption and decryption** are based on algorithms (cipher)

**Cryptography** science (art) of creating ciphers

**Cryptanalysis** science (art) of breaking (or trying to) ciphers

**Cryptology** study of codes, both creating and breaking them (= cryptography + cryptanalysis)

# Some vocabulary

- Notation

$p$  : plaintext

$c$  : ciphertext

$k$  : key

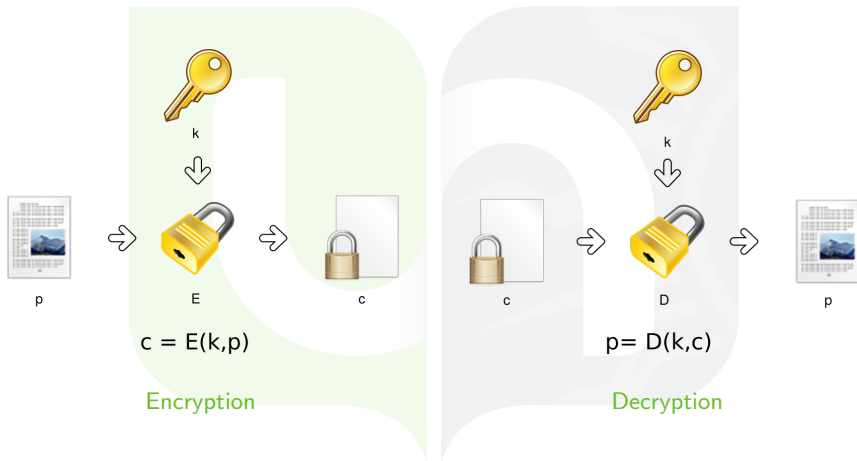
$E$  : encryption cipher

$D$  : decryption cipher

- Let me introduce some friends:

- Alice, Bob, Carol, Dave
- Eve (eavesdropper), Mallory (malicious)
- Trent (trusted arbitrator)
- Walter (warden), Peggy (prover), Victor (verifier)

# Core principle



# Core principle

## Why use keys and not just the cipher?

- is it possible to keep a cipher secret?
- would you trust the cipher author?
- is it possible to sell something that's secret?
- making the cipher public allows for public cryptanalysis and validation
- keys are secret and easier to protect
- one can use multiple keys to reduce the risk
- Kerckhoffs' principle (Journal des Sciences Militaires, 1883)
  - *"A cryptosystem should be secure even if everything about the system, except the key, is public knowledge."*
- reformulated in Shannon's maxim:
  - *"one ought to design systems under the assumption that the enemy will immediately gain full familiarity with them"*

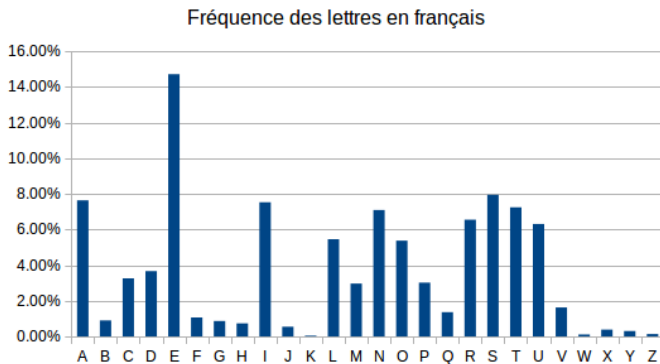
# Caesar Cipher

- Substitution cipher
- Ave Caesar  $\Rightarrow$  DYH FDHVDU
- Can be generalized into mono-alphabetic substitution
- Easy to break (statistical analysis, too small keyspace)

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



# Frequency analysis



Source: <http://www.math93.com/index.php/112-actualites-mathematiques/315-analyse-frequentielle>

Note: *De Zanzibar à la Zambie et au Zaïre, des zones d'ozone font courir les zèbres en zigzags zinzins.*

# Vigenère Cipher

- Blaise de Vigenère (1523-1596)
  - Polyalphabetic substitution
  - $c_i = p_i + k_i \bmod 26$
  - $c_i$  =  $i^{\text{ème}}$  letter of ciphertext
  - $p_i$  =  $i^{\text{ème}}$  letter of plaintext
  - $k_i$  =  $i^{\text{ème}}$  letter of (repeated) key

# Vigenère Cipher

plain text	This book is largely concerned with Hobbits
key	myprecious
plain text	THISBOOKISLARGELYCONCERNEDWITHHOBBITS
key	MYPRECIOUSMYPRECIOUSMYPRECIOUSMYPRECI
ciphertext	FFXJFQWYCKXYGXINGQIFOCGEIFEWNZTMQSMVA

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

Key table

# Enigma



# Enigma



# Enigma

- Invented early 20<sup>th</sup> century, used by the german army during WWII
- Variable substitution through
  - connection panel
    - fixed substitution of 6 letters
  - rotors
    - variable substitution
    - rotation after each letter
  - reflector
- Key defined by
  - connection setup
  - order of rotors
  - initial positioning of rotors
  - $> 10^{16}$  possibilities

# Classes of attacks

- Passive attack: eavesdropping
  - no interaction with communicating parties
  - hard to detect
- Active attack
  - identity spoofing
  - communication tampering (add, modify, remove packets)
  - ex: Man In The Middle – MITM
- Replay attack
  - re-send a previously emitted message
- Side-channel attack
  - attack carried out based on information gathered from the physical implementation of the cryptographic mechanism rather than from its cryptanalysis (power consumption, electromagnetic emissions. . . )
- Timing attack
  - infer information about the cryptographic process from the time it takes

# Attack the key

- Key size
  - 40bits  $\rightarrow 2^{40} \approx 10^{12}$
  - 56bits  $\rightarrow 2^{56} \approx 64 \cdot 10^{15}$
  - 128, 256, ...4096
- Brute force attack
  - on average, explore half of the keyspace
  - the larger the keyspace, the longer it takes to find the key
  - one bit added to the key size doubles the time needed for a brute force attack
- Dictionary attack
  - based on a list of above average likely values



# Attack the algorithm

- Objective: find the key in less operations than a brute force attack
- Classification of attacks
  - ciphertext-only attack: attacker has access to  $c_i$  only
  - known plaintext attack: attacker has access to  $(p_i, c_i)$  pairs
    - ex: brute force attack
  - chosen plaintext attack: attacker can choose  $p_i$  and obtain corresponding  $c_i$ 
    - ex: table lookup
  - chosen ciphertext attack: attacker can choose  $c_i$  and obtain corresponding  $p_i$

# What is a good cipher?

- it should not allow an attacker to find the key
  - not enough!! counter-example:  $E(k, p) = p$
- it should not allow an attacker to recover the complete plain text from the ciphertext
  - not enough!! would you accept a cipher that reveals even just 10% of the plaintext?
- it should not allow an attacker to reveal even a single character of the plaintext
  - not enough!! counter-example:  $\text{len}(E(k, p)) = \text{len}(p)$
- ciphertext must not reveal any information about the plain text
- more formally:  $P[\text{PlainText} = p | \text{CipherText} = c] = P[\text{PlainText} = p]$

# What is a good cipher?

- it should not allow an attacker to find the key
  - not enough!! counter-example:  $E(k, p) = p$
- it should not allow an attacker to recover the complete plain text from the ciphertext
  - not enough!! would you accept a cipher that reveals even just 10% of the plaintext?
- it should not allow an attacker to reveal even a single character of the plaintext
  - not enough!! counter-example:  $\text{len}(E(k, p)) = \text{len}(p)$
- ciphertext must not reveal any information about the plain text
- more formally:  $P[\text{PlainText} = p | \text{CipherText} = c] = P[\text{PlainText} = p]$

# What is a good cipher?

- it should not allow an attacker to find the key
  - not enough!! counter-example:  $E(k, p) = p$
- it should not allow an attacker to recover the complete plain text from the ciphertext
  - not enough!! would you accept a cipher that reveals even just 10% of the plaintext?
- it should not allow an attacker to reveal even a single character of the plaintext
  - not enough!! counter-example:  $\text{len}(E(k, p)) = \text{len}(p)$
- ciphertext must not reveal any information about the plain text
- more formally:  $P[\text{PlainText} = p | \text{CipherText} = c] = P[\text{PlainText} = p]$

# What is a good cipher?

- it should not allow an attacker to find the key
  - not enough!! counter-example:  $E(k, p) = p$
- it should not allow an attacker to recover the complete plain text from the ciphertext
  - not enough!! would you accept a cipher that reveals even just 10% of the plaintext?
- it should not allow an attacker to reveal even a single character of the plaintext
  - not enough!! counter-example:  $\text{len}(E(k, p)) = \text{len}(p)$
- ciphertext must not reveal any information about the plain text
- more formally:  $P[\text{PlainText} = p | \text{CipherText} = c] = P[\text{PlainText} = p]$

# Agenda

## Cryptography

Introduction

**Random numbers**

Symmetric cryptography

Key management

Asymmetric cryptography

Digital signature

Public key infrastructure

Applications

Conclusion

# Cryptography and random numbers

- random numbers are at the basis of many crypto mechanisms
  - key generation, nonces, challenges
- need for pseudo random number generator (PRNG)
  - needs a true random seed, without bias or correlation
    - hardware phenomena: resistor thermal noise, air turbulence in a hard disk...
    - software: clock, keyboard hits, mouse moves, os parameters (load...), mix of several sources
    - example: Intel® Security Driver is a device driver that returns truly random and non-deterministic values from a silicon-based device called the Firmware Hub (FWH), which harnesses resistor thermal noise to generate these values.

# Pseudo Random Number Generator

- Deterministic algorithm
- Generates a sequence of numbers that look random from a truly random (seed)  $x_0$
- Ex: linear congruential generator
  - $x_n = (ax_{n-1} + b) \bmod m, n \geq 1$ 
    - where  $a, b, m$ : parameters;  $x_0$ : seed
  - predictable
  - cryptographically non secure



# PRNG quality criteria

- Statistical tests to verify the quality of PRNG
  - frequency tests
  - correlation tests
  - FIPS 140-1: SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES
- Crypto-secure PRNG

# Agenda

## Cryptography

Introduction

Random numbers

**Symmetric cryptography**

Key management

Asymmetric cryptography

Digital signature

Public key infrastructure

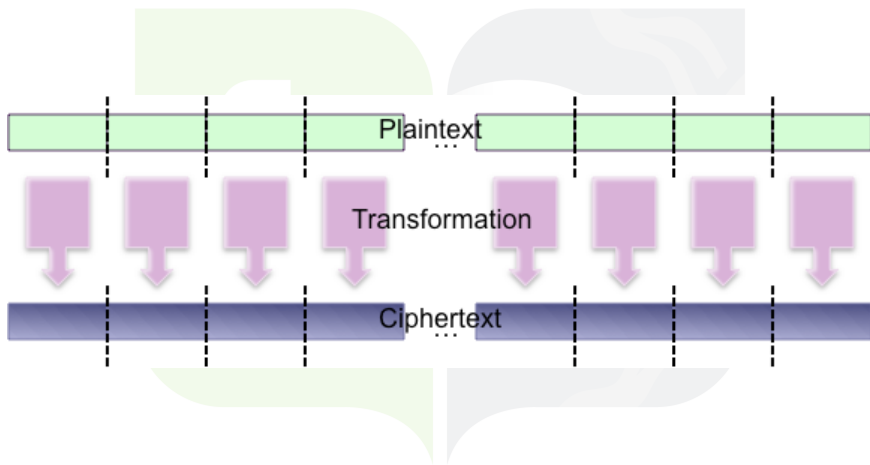
Applications

Conclusion

# Symmetric cryptography

- Aka secret key cryptography
  - $c = E(k,p)$
  - $p = D(k,c)$
  - $k$  is a number
    - sequence of 0 and 1
    - fixed length
  - Security depends on algorithm and key
- Key must be kept secret, but key is shared!
- How to generate the key?
- How to exchange the key?
- How to store the key?
- One key per party!

# Block cipher



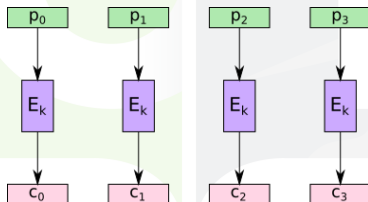
# Block cipher

- Based on product cipher, that combines
  - substitution encryption to achieve *confusion*
  - transposition encryption to achieve *diffusion*
- Processes blocks of data
  - typically 64, 128 or 256 bits
  - size of block in = size of block out
- Multiple rounds, with a different key (round key)
  - objective: reach an optimum in the quality of the secrecy
  - round keys are derived from the main key (key schedule)



# Modes of operation

- When the size of plaintext is larger than the size of a block, how to link the encryption of the various blocks?
- ECB – Electronic CodeBook Mode
  - repeating sequences can easily be detected
  - tampering is easy



# Modes of operation

- ECB – Electronic CodeBook Mode



# Modes of operation

- ECB – Electronic CodeBook Mode





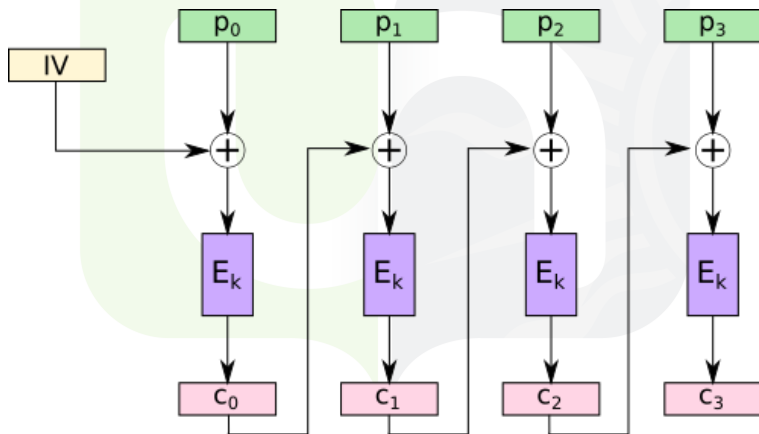
# Modes of operation

- ECB – Electronic CodeBook Mode



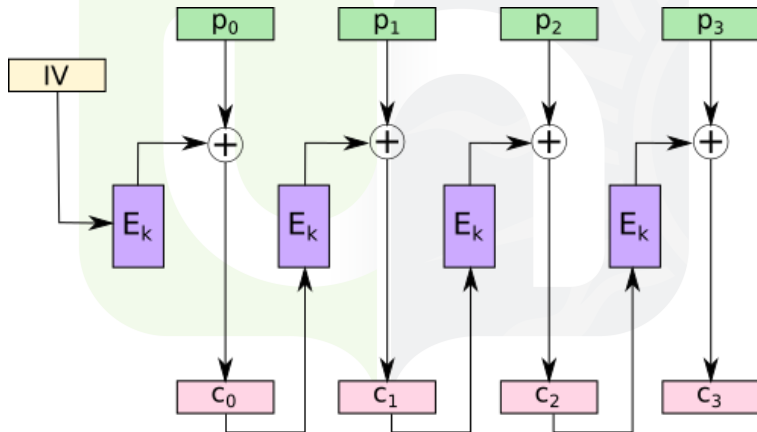
# Modes of operation

- CBC – Cipher Block Chaining



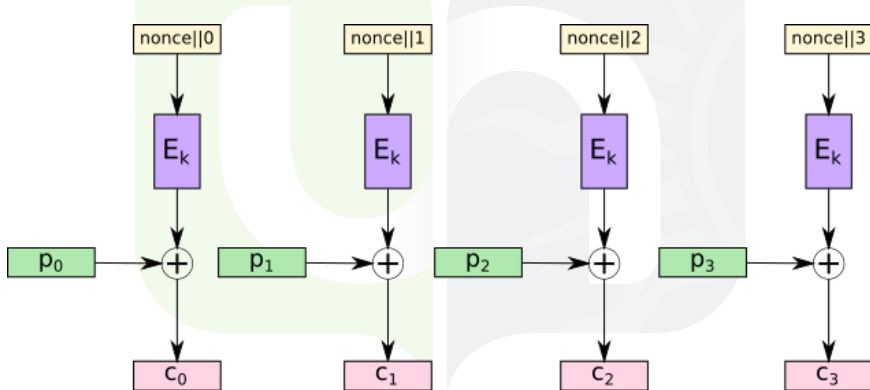
# Modes of operation

- CFB – Cipher FeedBack Mode



# Modes of operation

- CTR – Counter Mode



# Padding

- When the size of plaintext is not a multiple of the block size, how to 'pad' data so that the recipient can differentiate between data and padding bytes unambiguously?
- Examples (PKCS5, PKCS7):
  - count number of bytes to add
  - repeat the value in the bytes to pad in the last block
  - recipient reads the last byte and removes the read number of bytes from the last block

# One-time pad (Vernam Cipher)

- Based on
  - the one-time usage
  - of a random stream of data (pad)
  - of the same size as the plaintext
- Encryption:  $c_i = p_i \oplus k_i$
- Decryption:  $p_i = c_i \oplus k_i$
- One time pad guarantees perfect secrecy if and only if the size of the keyspace is the same as the size of the plaintext space and keys are uniformly distributed in the keyspace (Shannon)
- **But:** key management complex  $\Rightarrow$  infeasible in practice

# Stream cipher

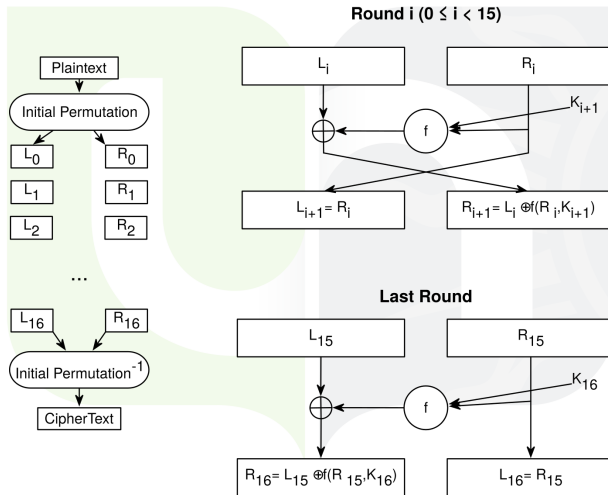
- Inspired from the one-time pad
- PRNG generates random sequence of bits (keystream)
- Encryption/decryption work one bit(byte) at a time:
  - $c = p \oplus pad$
  - $p = c \oplus pad$
- PRNG has to be cryptographically secure and properly initialized

# Digital Encryption Standard – DES

- Origin: US
- 56 bits key
- block cipher (block size: 64bits)
- based on Feistel structure: use the same function to encrypt and decrypt, so that the function does not have to be invertible
- design objective: make each bit of the ciphertext dependent on every bit of the plaintext and the key as quickly as possible
- 16 rounds (assumed to be an optimum)
  - each round uses a 'round key'  $k_i$  derived from the main key by permutation and compression
- Same algorithm to encrypt and decrypt, with keys in reverse order
- Rather slow
- Insecure because of the small key; cracked in 22h in 1999 at the RSA Conference, using a super computer (Deep Crack)

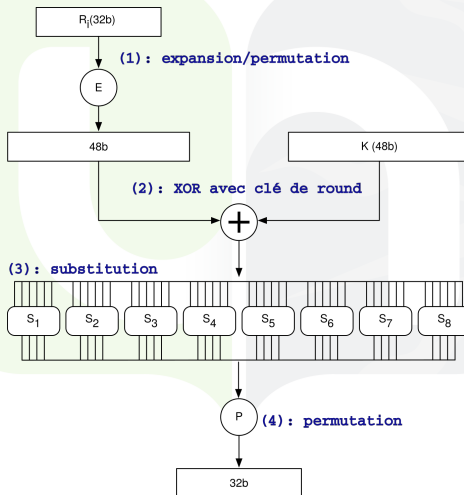


# Digital Encryption Standard – DES



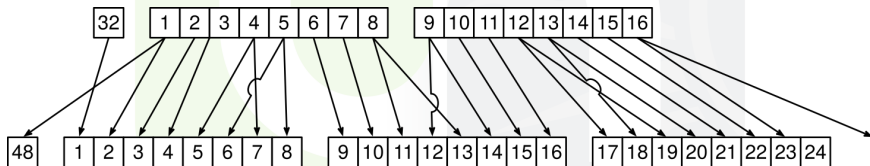
# Digital Encryption Standard – DES

- $f$  function



# Digital Encryption Standard – DES

- f: Expansion/Permutation
  - 'inflates'  $R_i$ ;  $32 \rightarrow 48$  bits



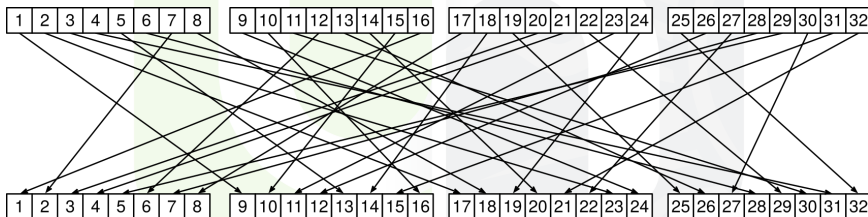
# Digital Encryption Standard – DES

- Substitution: S-Box
  - Split 48 bits into blocks of 6 bits
  - Process  $block_i$  with  $S - Box_i$
  - $S - Box_i$ : 6 bits in, 4 bits out  $\rightarrow$  48 bits in, 32 bits out
    - $b_1 b_2 b_3 b_4 b_5 b_6$
    - $b_1 b_6$  define the row
    - $b_2 b_3 b_4 b_5$  define the column

14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

# Digital Encryption Standard – DES

- Permutation: P-Box



# Digital Encryption Standard – DES

- Triple DES
  - triple DES encryption
  - 3 keys of 56 bits (168 bits)
  - $3DES(k_1, k_2, k_3)$
  - $c = E(k_3, D(k_2, E(k_1, p)))$ 
    - $DES(k, p) = 3DES(k, k, k, p)$
  - 3 times slower than DES!

# Advanced Encryption Standard – AES

- Rijndael (Vincent Rijmen et Joan Daemen)
- Selected by the US government as the DES replacement after a public call
- Block cipher (block size: 128bits)
- Key & block length: 128, 192 or 256
- Demo
- Other block ciphers: RC2, RC5, IDEA, Blowfish

# RC4

- Meant to remain undisclosed, but ended up on the Web
- Stream cipher
- Used for WiFi WEP security
- Variable key length
- Generates random sequences from the key which are  $\oplus$  (xor'ed) with the plaintext (encryption) or ciphertext (decryption)
- Uses a substitution box updated after each use:

S-Box

0 1 2 3 4 ... 255

231	76	165	7	99	34
-----	----	-----	---	----	----



# RC4

- Two steps
  - Initialize S-BOX
    - fill k-vector with the key (repeat if necessary)
    - use k-vector to shuffle S-Box
  - use S-BOX to encrypt byte per byte  $\oplus$
  - decryption works similarly

# RC4 – Initialization

```
1 ARRAY_SIZE=256;
2 key='\xbe\x24\x43\x5e\x73\xaf\xbc\x7e\x26\x40\xa6\xef\x9d\x47\x27\xe2';
3 sBox = bytearray();
4
5 def init(key):
6     print("Initializing_RC4");
7     sBox[0:]=[];
8     for i in range(ARRAY_SIZE):
9         sBox.append(i);
10    j=0;
11    byteKey = key.encode();
12    for i in range(ARRAY_SIZE):
13        a = sBox[i];
14        b = byteKey[i % len(byteKey)];
15        j = (a + b + j) % ARRAY_SIZE;
16        buf = sBox[i];
17        sBox[i] = sBox[j];
18        sBox[j] = buf;
```

# RC4 – encryption/decryption

```
1 def encrypt(bytePlainText):
2     i = 0;
3     j = 0;
4     t = 0;
5     byteCipherText = bytearray();
6     for l in range(len(bytePlainText)):
7         i = (i + 1) % ARRAY_SIZE;
8         j = (j + sBox[i]) % ARRAY_SIZE;
9         buf = sBox[i];
10        sBox[i] = sBox[j];
11        sBox[j] = buf;
12        t = (sBox[i] + sBox[j]) % ARRAY_SIZE;
13        byteCipherText.append(bytePlainText[l] ^ sBox[t]);
14    return byteCipherText;
15
16 def decrypt(ciphertext):
17    return(encrypt(ciphertext));
```

# Agenda

## Cryptography

Introduction

Random numbers

Symmetric cryptography

**Key management**

Asymmetric cryptography

Digital signature

Public key infrastructure

Applications

Conclusion

# Key management and distribution

## Key management

- manage: create, expire, revoke, send, store...
- threats: loss, theft, compromise, extortion...

## Issues

- potentially large number of keys
  - in a network of  $n$  nodes,  $\frac{n(n-1)}{2}$  keys are required for point-to-point communications
  - even worse if multi-party communications
- key has to be kept secret all the time
  - when stored or exchanged
  - easier to steal a key than to break a cipher

# Key lifetime

- The longer the key is used,
  - the greater the chance of compromise
  - the greater the impact of compromise
  - the greater the interest for an attacker
  - the 'easier' the cryptanalysis
- Key lifetime has to be carefully defined
- $\Rightarrow$  tradeoff between the management cost and impact in case of key compromise

# Key generation

- Secure generation method
  - avoid restricted keyspace
- Key must be random
- Key must be of sufficient length
- Key must be renewed periodically (cost ↗)

# Key update and destruction

## Update

- rather than distributing a new key, compute  $k_{t+1} = f(k_t)$
- using a one-way function
- security of  $k_{t+1}$  directly linked to security of  $k_t$

## Destruction

- without destruction, it is still possible to access ciphertext
- → secure destruction
  - documents, files, computer memory...



# Key storage

- Objective: keep keys away from attacker's reach
- Use hardware disconnected devices
  - Smartcards, USB tokens, ring



# Key exchange

- Use Key-Encryption Keys (kek) to encrypt Data Keys (dk)
  - kek distributed manually with a very high level of security, used rarely
  - dk distributed on-demand
- Split key on several channels
- Use a Key Distribution Center – KDC
  - Trusted Third Party
  - can be organized hierarchically
  - single point of failure
  - security concern: KDC knows all the keys!

# Key escrow

- leave a copy of the keys to a trusted third party
- key can be split in different pieces (secret sharing)  $n$ - $m$  threshold system
- use a recovery device that can be passed in case of absence (does not work in case of unexpected absence)

# Key ownership and key compromise

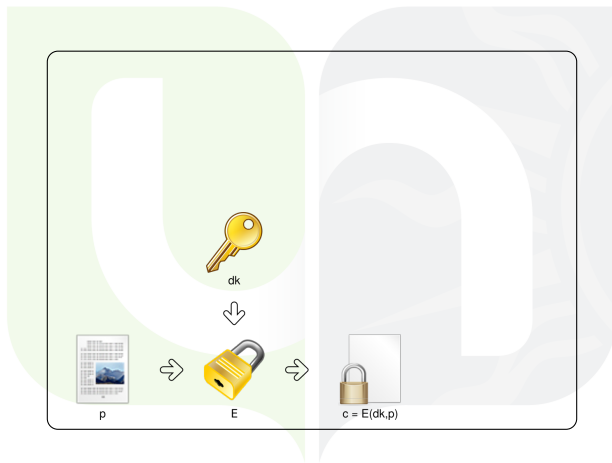
## Ownership

- How can Bob be sure that a key is truly Alice's and not someone else's?
- Trust management: face-to-face exchange, secure channel, KDC...

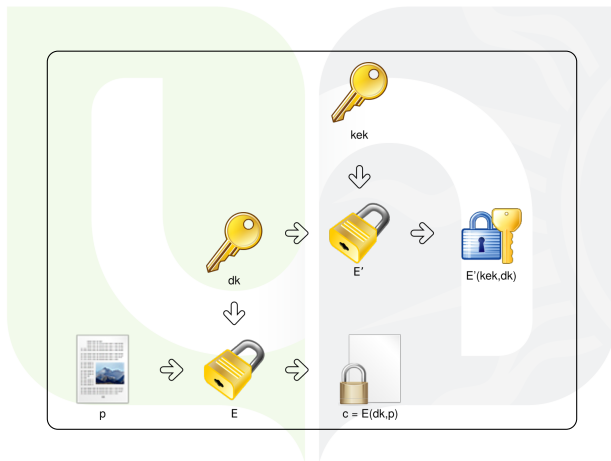
## Key compromise

- How to detect?
- Delay before detection
- Quick propagation to all users of the key
  - communicating parties, KDC...
- Mitigation: use different keys to minimize impact of a key compromise

# Password based encryption (PBE)



# Password based encryption (PBE)



# Password based encryption (PBE)

- Encryption

1. generate (dk)
2. enter password
3. generate random salt (s)
4. mix password and s
5. build kek from result of previous step
6. encrypt dk with kek
7. store s and  $E'(kek, dk)$

- Decryption

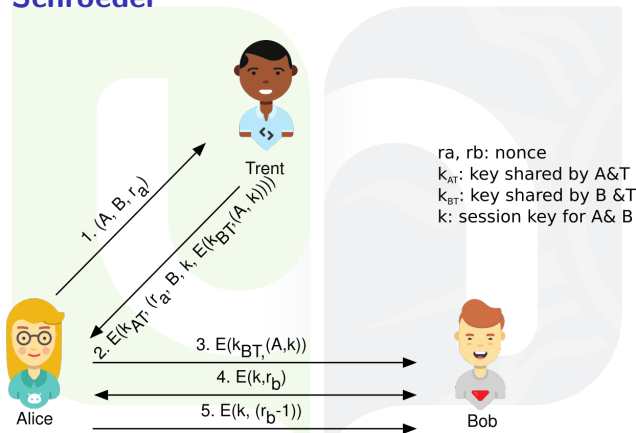
- enter password
- retrieve salt (s)
- mix password and s
- build kek from result of previous step
- use kek to decrypt  $E'(kek, dk) \rightarrow dk$

- Why use a salt?

- increase entropy
- avoid pre-computation attacks

# Key distribution via trusted third party

## Needham Schroeder



Needham-Schroeder algorithm



# Key distribution via trusted third party

## Needham Schroeder

- $r_a$  et  $r_b$  to avoid replay attack
- replay possible if old key is compromised together with message 3
  - use timestamp (requires proper clock synchronization)
- If  $k_{AT}$  is compromised, M can spoof A's identity and get a key to communicate with everyone

# Key distribution via trusted third party

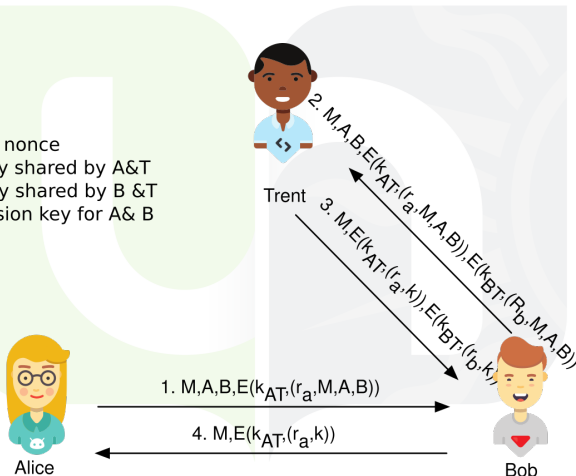
## Otway-Rees

$r_a, r_b$ : nonce

$k_{AT}$ : key shared by A&T

$k_{BT}$ : key shared by B & T

$k$ : session key for A&B



## Otway-Rees algorithm

# Agenda

## Cryptography

Introduction

Random numbers

Symmetric cryptography

Key management

**Asymmetric cryptography**

Digital signature

Public key infrastructure

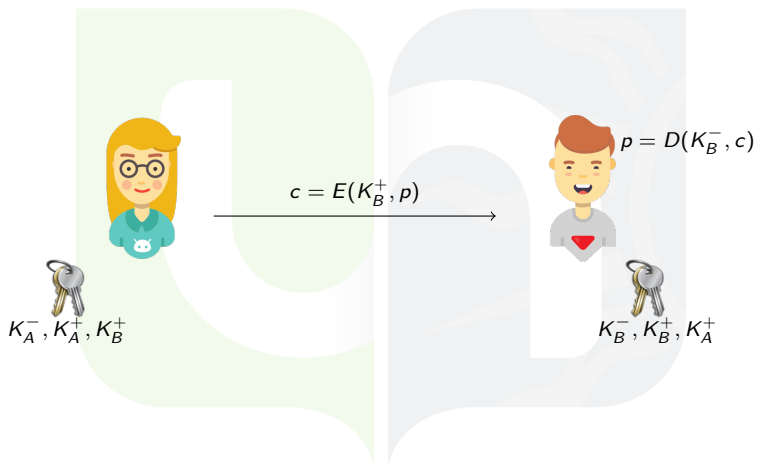
Applications

Conclusion

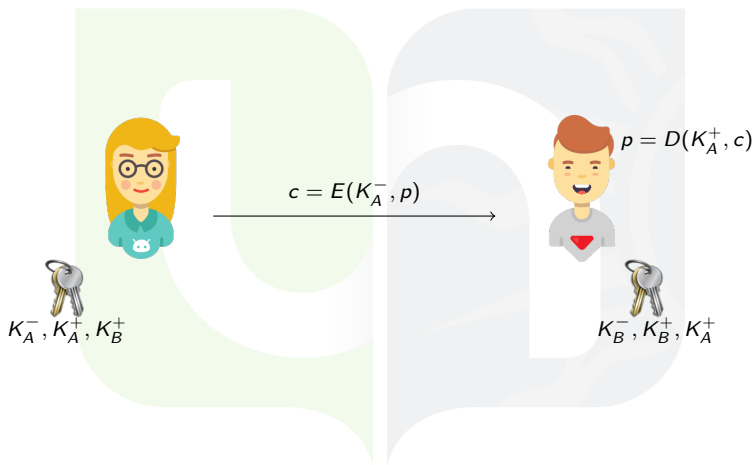
# Asymmetric cryptography

- Aka public key cryptography
- Invented in 1976 by W. Diffie & M. Hellman
- $c = E(k_1, p)$
- $p = D(k_2, c)$
- $k_1 \neq k_2$  but  $k_1$  &  $k_2$  are mathematically related
- What is encrypted with  $k_1$  can only be decrypted with  $k_2$  and vice-versa
- Principle: make one key public ( $K^+$ ) and keep the other secret ( $K^-$ )
- Symmetric crypto manipulates bits
- Asymmetric crypto manipulates numbers

# Core principle



# Core principle



# Core principles

- Based on one-way functions with trap
  - one-way function  $f$ :
    - easy to compute  $y = f(x)$
    - but hard to compute  $x = f^{-1}(y)$
  - trap materialized by  $K^-$
  - computationally hard to invert
  - computationally hard to derive  $K^-$  from  $K^+$
- Based on complex number theory problems
  - large numbers factorization
  - discrete logarithms
  - elliptic curves
- Compared to symmetric crypto
  - + key management and distribution
  - + secret key protection is of the sole responsibility of the key owner
  - performances

# Diffie-Hellman protocol

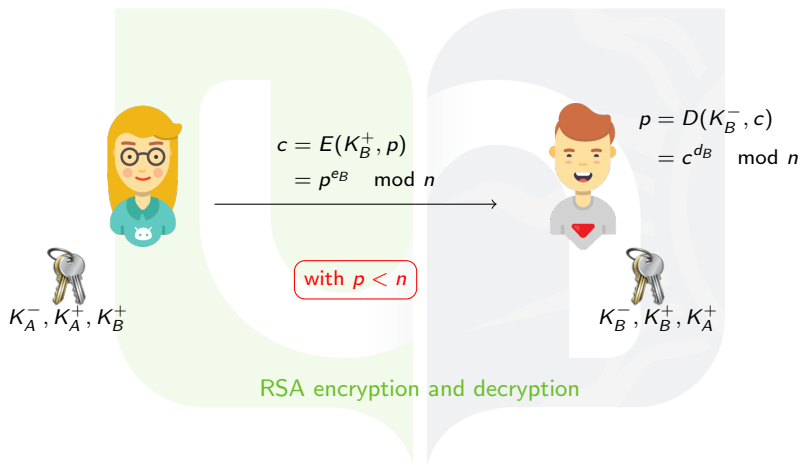
- Key establishment protocol
  - idea: Alice and Bob establish a secret over an insecure communication channel, without ever exposing it
  - how it works:
    1. select  $p$ , a large prime number, and  $g$ , a generator for the cyclic finite group  $G = \mathbb{Z}/p\mathbb{Z}$ ;  $p$  and  $g$  are public
    2. A randomly chooses  $a$  (large int) and computes  $g^a \bmod p$
    3. B randomly chooses  $b$  (large int) and computes  $g^b \bmod p$
    4.  $A \rightarrow B : g^a \bmod p$ , which allows B to compute  $k = (g^a \bmod p)^b \bmod p$
    5.  $B \rightarrow A : g^b \bmod p$ , which allows A to compute  $k = (g^b \bmod p)^a \bmod p$
    6.  $g^{ab} \bmod p$  is the new shared secret
- Security of the algorithm
  - assuming Eve gets  $g^a \bmod p$  and  $g^b \bmod p$ ; to get  $g^{ab} \bmod p$ , she needs to compute  $a$  from  $g^a \bmod p$  (or  $b$  from  $g^b \bmod p$ ), which is known as the discrete logarithm problem, with no easy solution
  - vulnerable to a MITM attack; harder if  $g^a \bmod p$  and  $g^b \bmod p$  are published in a directory



# RSA Algorithm

- Invented by Ron Rivest, Adi Shamir and Leonard Adleman in 1978
- Based on number factorization
- Key generation
  - randomly select two large prime numbers  $p, q$  and compute  $n = p \cdot q$
  - compute  $\varphi(n) = (p - 1)(q - 1)$
  - choose the exponent  $e$  such that
    - $e < n$  and
    - $e$  and  $\varphi(n)$  are relatively prime
  - compute  $d$  such that  $d \cdot e \equiv 1 \pmod{\varphi(n)}$
  - $K^+ = (n, e)$
  - $K^- = (n, d)$
  - typically, length of  $p$  and  $q > 1000$  bits (and  $n$  is twice as long)

# RSA Algorithm



# RSA Algorithm

## Security of the algorithm

- assuming Eve intercepts  $c = m^e \bmod n$ ,  $e$  and  $n$  being public
  - to find  $m$ , Eve needs  $d$
  - Eve knows that  $de = 1 \bmod (p-1)(q-1)$
  - $e$  being public, Eve needs to find  $p$  et  $q$ , which requires to factor  $n$ , with no easy solution
- during the RSA-155 cracking in 1999, it took 290 computers on the Internet and a supercomputer 4 months to factor a 512 bits (155 decimal digits) integer with two large prime factors.

# RSA Algorithm: example

- Let's choose  $p = 11, q = 3 \rightarrow n = p.q = 33$
- $\varphi(n) = (p - 1)(q - 1) = 20$
- let's set  $e = 3$ 
  - $e$  et  $\varphi(n)$  are relatively prime
    - $\text{pgcd}(e, p - 1) = \text{pgcd}(3, 10) = 1$
    - $\text{pgcd}(e, q - 1) = \text{pgcd}(3, 2) = 1$
    - thus  $\text{pgcd}(e, \varphi(n)) = 1$
  - let's find  $d$  such that  $e.d = 1 \pmod{\varphi(n)}$ 
    - $d$  s.t.  $\varphi(n)$  divides  $e.d - 1$
    - $d$  s.t. 20 divides  $3.d - 1$
  - $K^+ = (33, 3)$
  - $K^- = (33, 7)$

# Algorithme RSA: example

- $K^+ = (33, 3)$
- $K^- = (33, 7)$
- let's encrypt  $m = 8$ 
  - $c = m^e \bmod n = 8^3 \bmod 33 = 512 \bmod 33 = 17$
- to decrypt  $c = 17$ 
  - $p' = c^d \bmod n = 17^7 \bmod 33 = 410338673 \bmod 33 = 8$
  - note that  $bc \bmod n = (b \bmod n).(c \bmod n) \bmod n$
- in practice some common values for  $e$  are 3, 17 and 65537 ( $2^{16} + 1$ ) because they make the exponentiation faster

# El Gamal algorithm

- Based on DH and discrete log problem
  - easy to compute  $y = a^x \bmod n$  but hard to find  $x$  knowing that  $a^x \equiv b \bmod n$
- Principle
  1. choose  $n$ , prime and  $g$ , generator of the finite cyclic group  $G = \mathbb{Z}/n\mathbb{Z}$
  2. Alice randomly selects  $K_A^- = a$  in the group and computes  $K_A^+ = g^a \bmod n$ , her public key
  3. Bob randomly selects  $K_B^- = b$  in the group and computes  $K_B^+ = g^b \bmod n$ , his public key
  4. encryption ( $A \rightarrow B$ )
    - $c_1 = K_A^+$
    - $c_2 = m(K_B^+)^a \bmod n = mg^{ab} \bmod n$
    - $A \rightarrow B : (c_1, c_2)$
  5. decryption
    - $m = \frac{c_2}{c_1^b} \bmod n = \frac{mg^{ab} \bmod n}{g^{ab} \bmod n}$

# Agenda

## Cryptography

Introduction

Random numbers

Symmetric cryptography

Key management

Asymmetric cryptography

**Digital signature**

Public key infrastructure

Applications

Conclusion

# Digital signature

- Objectives
  - authentication
  - non-repudiation
- Principles
  - rely on asymmetric crypto
  - private key is known to owner only
  - use private key to sign
- Problem: asymmetric crypto slow



# Message digest

- aka hash, fingerprint
- Idea: from arbitrary data, compute a unique, short fixed length, random looking message digest
- Example: SHA1

```
1 | jnc@enigma ~ $ shasum -
2 | Hello, world!
3 | 09fac8dbfd27bd9b4d23a00eb648aa751789536d -
4 | jnc@enigma ~ $ shasum -
5 | Quousque tandem abutere, Catilina, patientia nostra?
6 | a081ccb6070a9a4cf5137c8bfff6ecad80a6a50d9 -
7 | jnc@enigma ~ $ shasum -
8 | Quousque tandem abutere, Catilina, patientia nostra!
9 | 6be608c1cc7a1f63a04508967d4d98cd0882adcd -
```

# Message digest

- Hash function:  $H$ 
  - one-way function without trap ( $\leftrightarrow$  asymmetric crypto)
  - variable length input
  - fixed length output
  - random appearance
  - given  $p$ , arbitrary data,  $h = H(p)$
  - from  $h$ , it is computationally impossible to find  $p \mid h = H(p)$   
(*pre-image resistance or one-way property*)
  - knowing  $p$ , it is computationally impossible to find  $p' \mid p \neq p' \wedge H(p) = H(p')$  (*second pre-image resistance or weak collision resistance*)
  - it is computationally impossible to find  $p, p' \mid p \neq p' \wedge H(p) = H(p')$  (*strong collision resistance*)
- a tiny change in  $p$  has an impact on the whole  $h$
- $h$  can be used as a unique representation of  $p$

# Message digest

## Collisions

- number of possible input is infinite
- number of possible digests is finite
- collisions are possible ( $\exists p, q \mid p \neq q \wedge H(p) = H(q)$ )
- practically impossible to find

# Message digest

- Examples

- MD2

- invented by Ron Rivest (R de RSA)
    - 128 bit digest ( $2^{128}$  possible digests)
    - weaknesses found
    - collisions found, but not on demand
    - obsolete

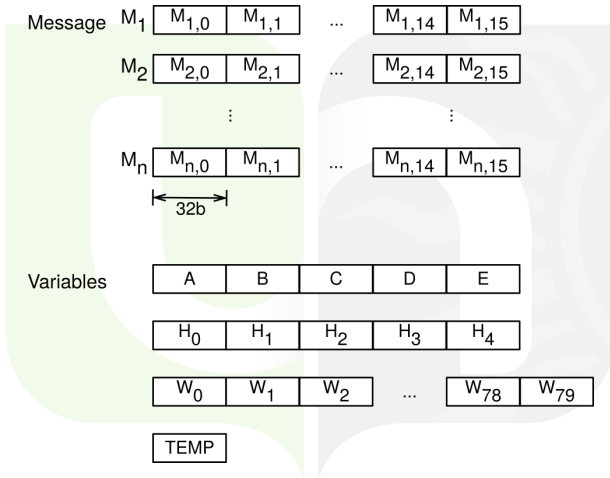
- MD5

- invented by Ron Rivest (R de RSA) in 1991 in response to MD2 weaknesses (MD3 and MD4 were quickly abandoned)
    - 128 bit digest ( $2^{128}$  possible digests)
    - quicker and more resistant than MD2
    - weaknesses found but no collision
    - obsolete

# Message digest

- SHA-1 (1995, NSA, US)
  - FIPS PUB 180-1: Secure Hash Standard
  - message length  $< 2^{64}$  bits
  - digest length: 160 bits
  - processes blocks of 512 bits
  - weaknesses found in 2005 and collisions found (<https://shattered.io/static/shattered.pdf>)
  - obsolete
- SHA-2 (2001)
  - SHA-256, SHA-224, SHA-384, SHA-512
  - resistant to cryptanalysis
- SHA-3 (2012)
  - “SHA-3 is not meant to replace SHA-2, as no significant attack on SHA-2 has been demonstrated. Because of the successful attacks on MD5 and SHA-0 and theoretical attacks on SHA-1 and SHA-2,[5] NIST perceived a need for an alternative, dissimilar cryptographic hash, which became SHA-3.”

# Message digest



## SHA-1 algorithm

# Message digest

*(pad message to a multiple of 512bits)*

**for**  $i = 1 \rightarrow n$  **do**

**for**  $j = 0 \rightarrow 15$  **do**

$W_j \leftarrow M_{i,j}$

**end for**

**for**  $t = 16 \rightarrow 79$  **do**

$W_t \leftarrow S^1(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16})$

**end for**

$A \leftarrow H_0; B \leftarrow H_1; C \leftarrow H_2; D \leftarrow H_3; E \leftarrow H_4;$

**for**  $t = 0 \rightarrow 79$  **do**

$TEMP \leftarrow S^5(A) + f_t(B, C, D) + E + W_t + K_t;$

$E = D; D = C; C = S^{30}(B); B = A; A = TEMP;$

**end for**

$H_0 = H_0 + A; H_1 = H_1 + B; H_2 = H_2 + C; H_3 = H_3 + D; H_4 = H_4 + E;$

**end for**

*(digest =  $H_0H_1H_2H_3H_4$ )*

SHA-1 algorithm

# Message digest

- $S^i$  shift by  $i$  bits
- $f_i$  functions
  - $f_i(B, C, D) = (B \wedge C) \vee (\neg B \wedge D)$ , where  $(0 \leq i \leq 19)$  (conditional function)
  - $f_i(B, C, D) = B \oplus C \oplus D$ , where  $(20 \leq i \leq 39)$  (parity function)
  - $f_i(B, C, D) = (B \wedge C) \vee (B \wedge D) \vee (C \wedge D)$ , where  $(40 \leq i \leq 59)$  (majority function)
  - $f_i(B, C, D) = B \oplus C \oplus D$ , where  $(60 \leq i \leq 79)$
- Constants  $K_i$ 
  - $K_i = 0x5A827999$ , where  $(0 \leq i \leq 19)$
  - $K_i = 0x6ED9EBA1$ , where  $(20 \leq i \leq 39)$
  - $K_i = 0x8F1BBCDC$ , where  $(40 \leq i \leq 59)$
  - $K_i = 0xCA62C1D6$ , where  $(60 \leq i \leq 79)$



# Empreinte numérique

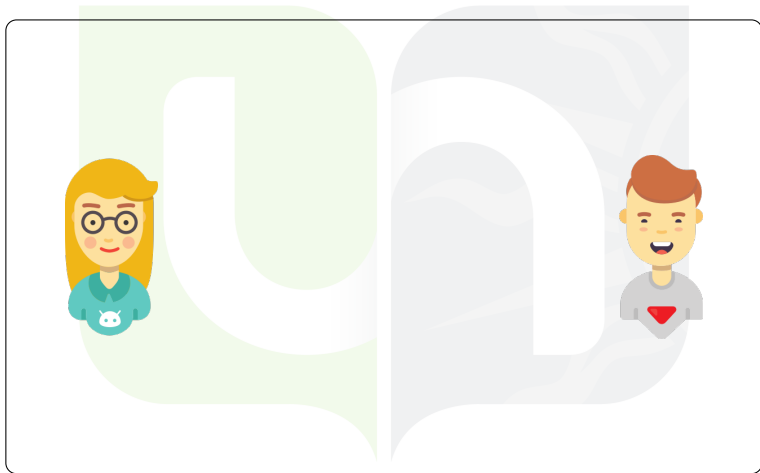
- Padding
  - append '1' to the message
  - store the length of the message in bits in the last 64 bits of the last 512 bits block
  - fill the rest with '0'

Ex.  $l = 56\text{bits}$  (00111000)

00100111	11001100	10100000	10001011	10011100	11110110	01011010	10000000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00111000

Padding in SHA-1

# Use digest to check message integrity



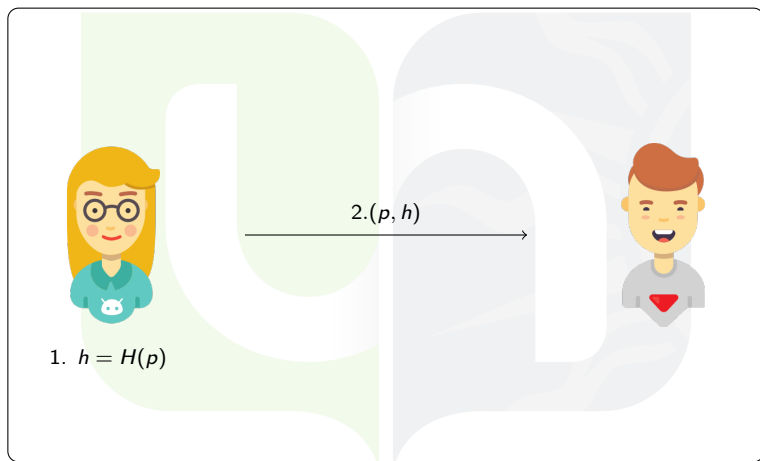
# Use digest to check message integrity



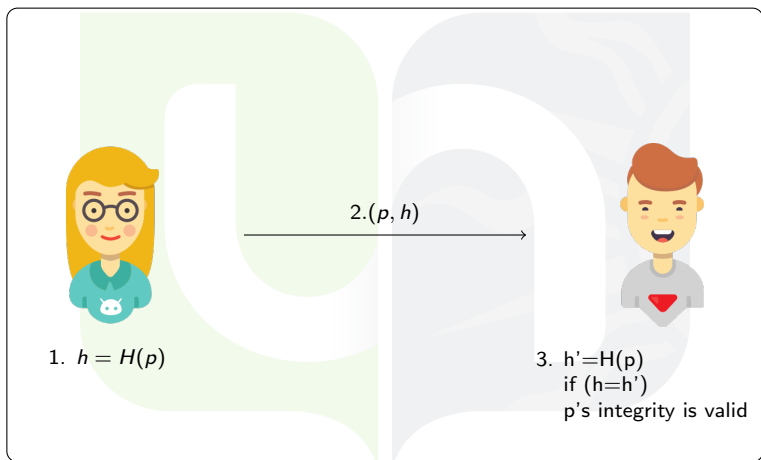
1.  $h = H(p)$



# Use digest to check message integrity

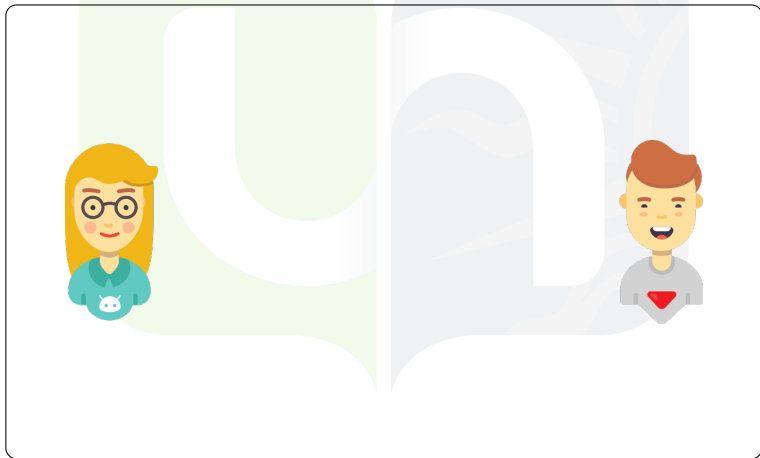


# Use digest to check message integrity



# Use digest to check message authenticity

- Solution 1: HMAC – Hash Message Authentication Code
- based on shared secret  $k$



# Use digest to check message authenticity

- Solution 1: HMAC – Hash Message Authentication Code
- based on shared secret  $k$

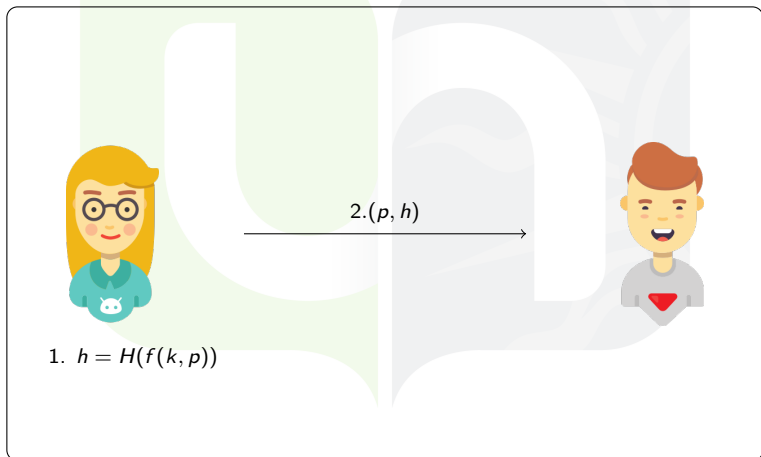


1.  $h = H(f(k, p))$



# Use digest to check message authenticity

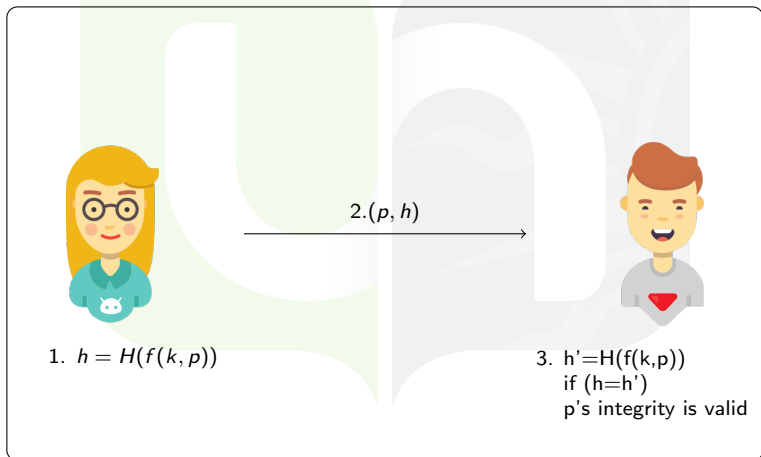
- Solution 1: HMAC – Hash Message Authentication Code
- based on shared secret  $k$





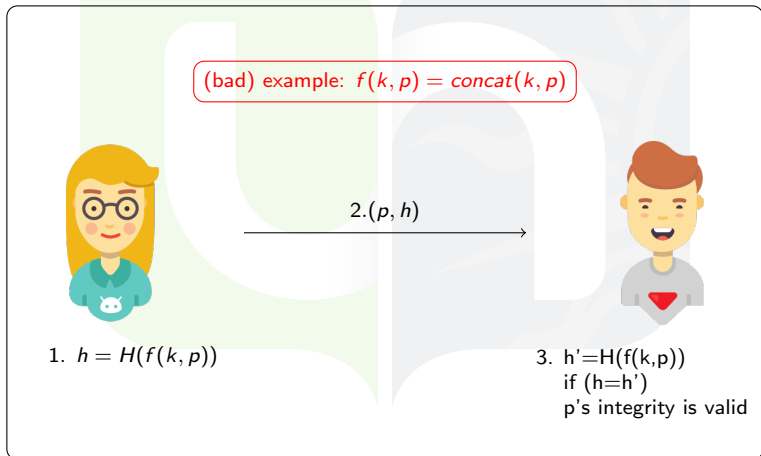
# Use digest to check message authenticity

- Solution 1: HMAC – Hash Message Authentication Code
- based on shared secret  $k$



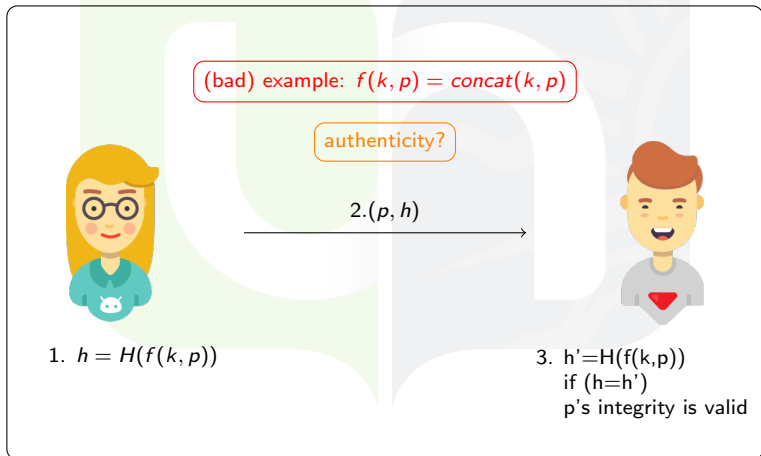
# Use digest to check message authenticity

- Solution 1: HMAC – Hash Message Authentication Code
- based on shared secret  $k$



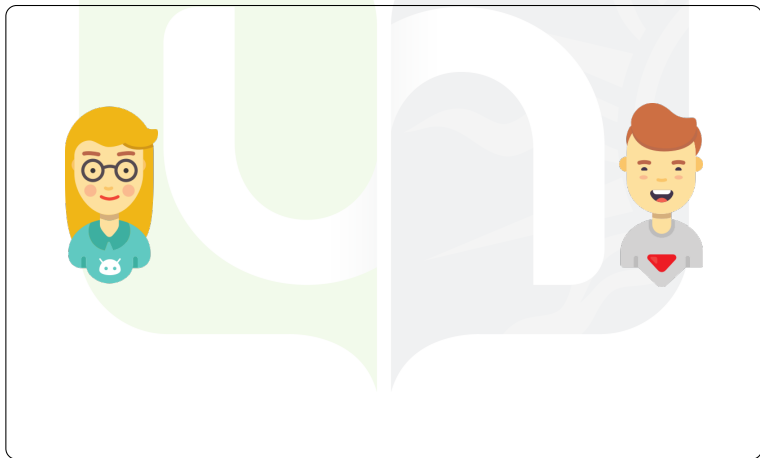
# Use digest to check message authenticity

- Solution 1: HMAC – Hash Message Authentication Code
- based on shared secret  $k$



# Digital signature

- Solution 2: digital signature
- based on asymmetric crypto



# Digital signature

- Solution 2: digital signature
- based on asymmetric crypto

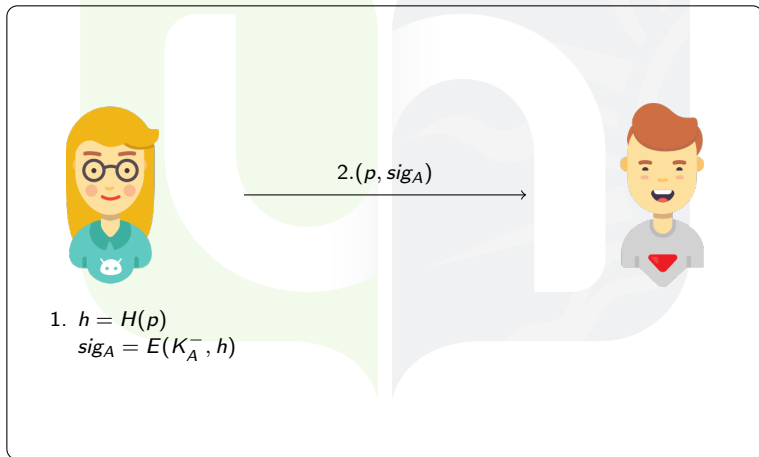


$$\begin{aligned} 1. \quad h &= H(p) \\ \text{sig}_A &= E(K_A^-, h) \end{aligned}$$



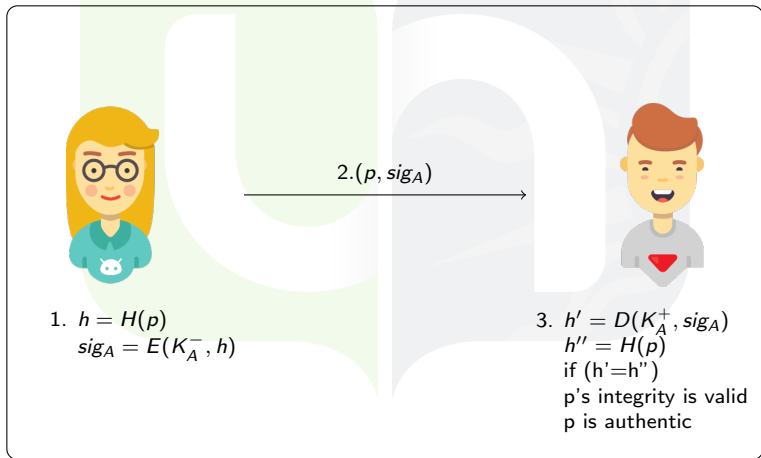
# Digital signature

- Solution 2: digital signature
- based on asymmetric crypto



# Digital signature

- Solution 2: digital signature
- based on asymmetric crypto



# Digital signature

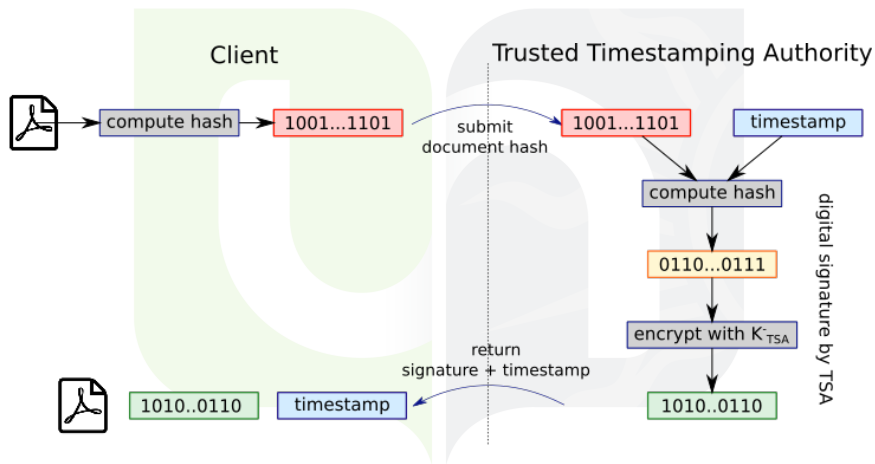
- Digital signature guarantees non-repudiation
- Opposite to handwritten signature, digital signature is not constant:
  - two different messages signed by the same key will have different signatures
  - one message signed with two different keys will have two different signatures
  - it is thus not possible to copy the signature from one message to another one



# Digital signature

- Digital Signature Algorithm – DSA
  - US, 1993
  - based on discrete log problem
  - uses a SHA-1 digest of the text to sign
- Which algorithm to choose?
  - compare security, performance, block and key sizes. . .
  - consider interoperability: RSA, DSA

# Example – trusted timestamping



# Examples

## Multiple signatures

- sequential, with or without defined order
  - $s_0 = S(p)$
  - $s_i = S(p, s_{i-1})$ , for  $0 < i < n$
- parallel, independent signatures
  - $s_i = S(p)$ , for  $0 \leq i < n$

## Group signature

- only members of the group can sign  $p$
- it is possible to validate the group signature, without disclosing the identity of the group member who actually signed
- in case of trouble, the signature can be opened to reveal the identity of the member who signed

# Agenda

## Cryptography

Introduction

Random numbers

Symmetric cryptography

Key management

Asymmetric cryptography

Digital signature

**Public key infrastructure**

Applications

Conclusion

# Digital certificate and PKI

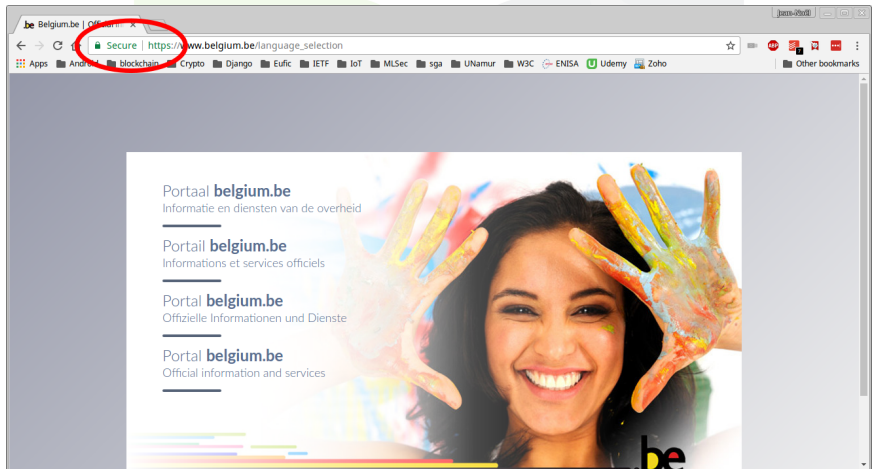
## Objective

- based on **asymmetric cryptography**, establish a provable relationship between a **public key** and the identity of its **owner** (person or system)

## Solution

- digital certificate

# Example



# Example

The screenshot shows a web browser window with the URL <https://www.belgium.be/fr>. A certificate viewer window is open, displaying the details of an SSL Server Certificate for `*.belgium.be`. The certificate has been verified for the following usages:

- Issued To**
  - Common Name (CN): `*.belgium.be`
  - Organization (O): `Federale Overheidsdienst Informatie- en Communicatietechnologie`
  - Organizational Unit (OU): `Fedict`
- Issued By**
  - Common Name (CN): `QuoVadis Europe SSL CA G1`
  - Organization (O): `QuoVadis Trustlink BVBA`
  - Organizational Unit (OU): `<Not Part Of Certificate>`
- Validity Period**
  - Issued On: `Wednesday, July 15, 2015 at 5:27:49 PM`
  - Expires On: `Sunday, July 15, 2018 at 5:37:00 PM`
- Fingerprints**
  - SHA-256 Fingerprint: `E5 87 63 08 82 F4 93 F6 C0 0D 5A 53 59 AC 01 ED 37 16 A8 C6 38 F4 2A 45 19 E1 5D D7 F3 41 80 65 3C D8 91 62 80 1D F7 F5 0A 1F B9 C6 EC B5 DA 96 8D C5 C6 27`
  - SHA-1 Fingerprint: `E5 87 63 08 82 F4 93 F6 C0 0D 5A 53 59 AC 01 ED 37 16 A8 C6 38 F4 2A 45 19 E1 5D D7 F3 41 80 65 3C D8 91 62 80 1D F7 F5 0A 1F B9 C6 EC B5 DA 96 8D C5 C6 27`

# Certificat X.509

- A digital certificate certifies that the **public key** it contains belongs to the **subject** it mentions
- It is issued by a **trusted third party**, a **Certification Authority**, that authenticates the certificate by **digitally signing** it
- It identifies the subject and the issuer using a non-ambiguous notation
  - CN = eservices.minfin.fgov.be, OU = Federal Public Service Finances, O = Service Public Fédéral Finances, L = Schaarbeek, ST = Brussel-Hoofdstad, C = BE
  - C=BE, CN=Jean Colin (Authentication), SN=Colin, GN=Jean Noël/serialNumber=1234567890
- It can be stored in multiple formats: PEM, DER, PFX...



# Example

```
1 Certificate:
2   Data:
3     Version: 3 (0x2)
4     Serial Number:
5       1b:9f:cc:56:f2:b6:47:6c:3f:ac:b0:51:39:cb:38:1c:dd:e4:28:3a
6   Signature Algorithm: sha256WithRSAEncryption
7     Issuer: C=BE, O=QuoVadis Trustlink BVBA, CN=QuoVadis Europe SSL CA G1
8     Validity
9       Not Before: Jul 15 15:27:49 2015 GMT
10      Not After : Jul 15 15:37:00 2018 GMT
11    Subject: C=BE, ST=Brussels Hoofdstedelijk Gewest, L=Brussel, O=Federale
12             Overheidsdienst Informatie- en Communicatietechnologie, OU=Fedict, CN=*.
13             belgium.be
14    Subject Public Key Info:
15      Public Key Algorithm: rsaEncryption
16      Public-Key: (2048 bit)
17      Modulus:
18        00:c5:7c:fc:b2:24:4a:d4:c4:2f:19:8a:1c:4f:af:
19        39:f1:6f:ae:dd:c3:fd:0c:08:87:94:04:15:f1:ed:
20        b9:2e:1f:6f:89:95:1d:fd:d8:09:54:20:c5:73:b5:
21        83:63:9d:3e:2e:95:34:f3:28:c0:03:bd:07:63:23:
22        ef:40:ca:8f:77:5e:7c:13:19:f8:75:2b:86:63:4a:
23        ...
24        e5:f5:bf:e1:a3:2c:63:bf:b1:7c:e1:ab:7f:97:e6:
25        7c:88:be:1d:a4:fa:e6:b3:a3:1a:71:8f:cf:cd:8d:
26        df:85:ef:9b:80:f7:16:7b:cb:18:b0:09:10:ff:75:
27        83:3e:fa:b4:99:bd:67:2d:4f:e6:a9:4f:af:9c:74:
28        08:5a:6f:a7:2f:ad:04:c3:3c:be:5e:7c:84:7d:36:
29        37:e7
30      Exponent: 65537 (0x10001)
```

# Example

```
1 X509v3 extensions:
2   Authority Information Access:
3     CA Issuers - URI:http://trust.quovadisglobal.com/qveussl1.crt
4     OCSP - URI:http://ocsp.quovadisglobal.com
5
6   X509v3 Subject Key Identifier:
7     4C:F0:45:B6:A6:A5:D0:5A:A3:C9:88:B8:5C:BA:62:9C:35:AA:63:6D
8   X509v3 Basic Constraints: critical
9     CA:FALSE
10  X509v3 Authority Key Identifier:
11     keyid:42:D3:87:73:E1:9E:D3:3E:5C:FE:B0:70:BB:CA:DE:21:BA:24:79:36
12
13  X509v3 Certificate Policies:
14     Policy: 1.3.6.1.4.1.8024.0.2.100.1.1
15     CPS: http://www.quovadisglobal.com/repository
16
17  X509v3 CRL Distribution Points:
18
19     Full Name:
20       URI:http://crl.quovadisglobal.com/qveussl1.crl
21
22  X509v3 Key Usage: critical
23     Digital Signature, Key Encipherment
24  X509v3 Extended Key Usage:
25     TLS Web Server Authentication, TLS Web Client Authentication
26  X509v3 Subject Alternative Name:
27     DNS:*.belgium.be, DNS:belgium.be
```

# Example

Signature Algorithm: sha256WithRSAEncryption

```
54:6f:3d:c5:16:2f:f1:68:d7:56:f2:e4:98:c9:47:0c:86:88:
24:e9:38:31:6f:47:a4:92:32:42:0d:10:61:4f:84:00:fd:bd:
b1:f6:0f:2d:31:4d:e6:a0:6e:57:59:79:62:20:a4:e4:cb:bf:
61:68:ae:91:34:ae:82:71:bd:07:82:e1:cb:ee:a6:ac:1e:20:
f6:a7:b9:d3:1a:df:b6:91:bf:55:21:a8:1d:7b:0c:4c:36:3c:
83:f6:97:a7:7f:83:b1:3f:3f:2a:96:38:a1:1b:d1:e9:2f:73:
2a:5e:d4:d0:fe:0a:7c:1c:fd:3b:c6:3d:29:7b:8e:e9:be:c4:
eb:82:83:cd:e9:76:e7:5e:67:1d:f1:0f:ee:36:a0:60:5c:43:
bf:ee:70:29:5e:1f:86:63:1b:7b:8d:35:22:5b:c9:4f:aa:5d:
06:a4:9e:3f:e8:e6:9f:64:2a:24:cd:ef:4e:2e:d5:cc:bb:29:
e6:4e:71:a7:fc:11:2c:78:b1:51:8a:76:42:95:f7:02:fb:33:
0c:22:47:9d:bf:00:aa:1c:2c:7b:cb:98:b8:62:7a:be:9c:07:
05:2f:b2:08:b1:73:b2:14:d4:2d:61:21:27:c3:1b:24:f6:56:
b7:8e:e1:08:4d:68:e6:5a:38:31:cd:96:a5:3a:36:4d:78:f6:
52:38:73:bb:d4:c7:51:fb:07:91:19:6d:e1:86:4c:cc:c9:cb:
00:d1:ad:af:1c:45:32:2c:f9:ba:7b:a0:03:d8:9c:01:37:74:
68:4a:f9:58:fe:1c:28:93:f0:d1:28:01:8c:b9:b5:ca:94:95:
...
4c:11:c8:70:26:4e:19:f2:73:c1:f9:53:84:58:4a:b1:3a:6b:
a3:a7:fa:af:07:16:a5:b0:f4:e0:65:3f:40:49:70:9e:29:16:
ee:73:85:35:6e:4a:4b:60:4e:76:35:b3:36:cf:88:bd:59:a5:
42:fb:66:c9:f7:69:aa:02:95:b6:4c:7c:21:38:42:99:1b:6b:
21:af:3f:57:f7:d3:b9:22
```

# Structure of a X.509 certificate

	v1	v2	v3
Version	✓	✓	✓
Certificate Serial Number	✓	✓	✓
Signature Algorithm Identifier	✓	✓	✓
Issuer Name	✓	✓	✓
Validity (not before/not after)	✓	✓	✓
Subject Name	✓	✓	✓
Subject Public Key Information	✓	✓	✓
Issuer Unique Identifier		✓	✓
Subject Unique Identifier		✓	✓
Extensions			✓
Signature	✓	✓	✓

Content of a X.509 certificat

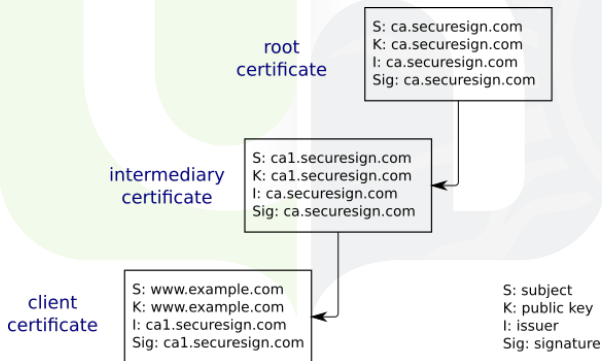
# Structure of a X.509 certificate

Authority Key Identifier	Policy Mappings
Subject Key Identifier	Subject Alternative Name
Key Usage	Issuer Alternative Name
Extended Key Usage	Subject Directory Attributes
CRL Distribution Point	Basic Constraints
Private Key Usage Period	Name Constraints
Certificate Policies	Policy Constraints

Extensions

# Certificate validation (version 1)

- Example: my browser connects to <https://www.example.com>; it receives back a certificate
- to validate the cert signature, it needs the public key of the issuer. . .



# X.509 certificate and authentication

- A certificate does not authenticate its bearer!
- To establish the identity of the subject, one must verify that the subject knows the private key that corresponds to the public key included in the certificate
- for instance, using a challenge/response protocol
  1. Verifier  $\rightarrow$  Prover (subject):  $E(K_{Sujet}^+, n)$  where  $n$  is a random number (nonce)
  2. Prover  $\rightarrow$  Verifier:  $n$

# Public Key Infrastructure – PKI

Certificate management is performed by a set of entities that form a PKI –  
Public Key Infrastructure

## Components of a PKI

### 1 Certificate Authority – CA

- authenticates subjects
- issues, manages, revoke certificates (CSR – Certificate Signing Request)
- makes its public key available (via a certificate)
- signed by whom? by itself! self-signed certificate, root certificate
- in practice, a CA uses a chain of certificates



# Public Key Infrastructure – PKI

## Components of a PKI (cont'd)

### 2 Registration Authority – RA

- sits next to CA
- receives and validates information from subjects
- when needed, generates keys for users, and distributes key storage devices
- handles and authorizes key storing and recovery requests, as well as certificate revocation requests
- multiple RAs for one CA

### 3 Certificate Directory

### 4 Key Recovery Server

- avoid overhead of key/cert creation and distribution in case of lost key

# Public Key Infrastructure – PKI

## Certificate revocation

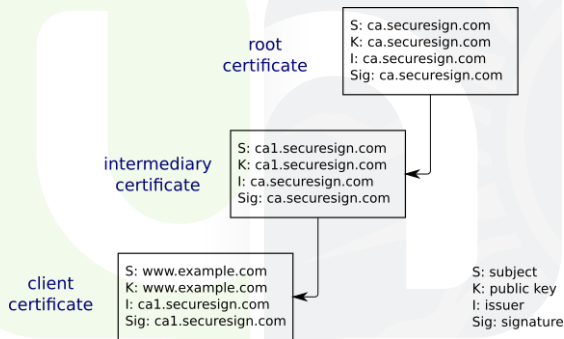
- because of a compromised or lost key, wrong data in certificate, CA error, disappeared subject. . .
- CA maintains a CRL – Certificate Revocation List
- simple file that contains the list of revoked certs, signed by the CA
- CRL regularly updated, even if no change, to provide an updated information
- requires active verification by certificate user (using OCSP for instance)
- for large CA, multiple CRL distribution points

# Certificate validation (version 2)

- Example: my browser connects to <https://www.google.com>; it receives back a certificate
- It check the certificate validity by verifying that:
  - the subject of the cert is [www.google.com](https://www.google.com) (or a more generic name that includes [www.google.com](https://www.google.com))
  - the cert has not expired
  - the cert conforms with the specified usage
  - the cert has been signed with the public key of the issuer
  - the cert has not been revoked
- To validate the signature of the cert, my browser needs the certificate of the cert issuer, that contains the public key that corresponds to the private key used to sign the certificate; this last certificate will need to be checked according to the same procedure
  - ⇒ Pull vs Push: server sends the whole certificate chains vs the browser has to collect all certificates separately

# Trust models

- CA hierarchy
  - trust in one node gives trust in all node's descendants



- Cross-certification
  - non-hierarchical trust
  - establish trust between separate domains

# Trust? Really?

Axel ARNBAK, Hadi ASGHARI, Michel VAN EETEN, and Nico VAN EIJK.  
*Security Collapse in the HTTPS Market.* ACM Queue 12, 8, pages 30  
(August 2014), 14 pages. DOI:  
<http://dx.doi.org/10.1145/2668152.2673311>

# Classes of certificates

- Class 1: simple verification (email or domain name)
- Class 2: remote verification of subject's identity (ex: photocopy of ID)
- Class 3: face to face verification of subject's identity
- Qualified certificate: as defined by EU EIDAS regulation, certificates issued by *qualified* authorities, i.e. authorities that conform with regulation requirements

# Sizes and algorithms

## ANSSI Recommendations<sup>1</sup>

- Encryption
  - minimal symmetric key size: 100b (< 2020) 128b (> 2020)
  - minimal block size (block cipher): 64b (< 2020) 128b (> 2020)
  - algorithms: DES, 3DES: KO; AES: OK
  - Factorisation (RSA)
    - size of  $n$ : 2048b (< 2020) 4096b (> 2020)
    - secret exponent ( $d$ ): same size as  $n$
    - public exponent ( $e$ ):  $> 2^{16}$
- Digest
  - digest size: 200b (< 2020)
  - algorithms: SHA-1: NOK; SHA-256: OK
- Avoid shared keys at all cost

<sup>1</sup> Source: Référentiel Général de Sécurité v2.0. Annexe B1: Mécanismes cryptographiques, ANSSI, 2014

# Agenda

## Cryptography

Introduction

Random numbers

Symmetric cryptography

Key management

Asymmetric cryptography

Digital signature

Public key infrastructure

## Applications

Conclusion

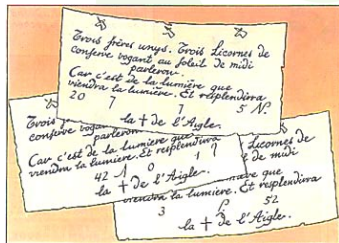


# Applications

## • Secret splitting

- Assuming T has a secret p he wants to share between A, B, C & D without revealing it
- T generates 3 random strings R1, R2 & R3
- T computes  $U = p \oplus R1 \oplus R2 \oplus R3$
- T gives R1 to A, R2 to B, R3 to C, U to D
- A, B, D & C can together assemble their parts to rebuild p:  

$$p = U \oplus R1 \oplus R2 \oplus R3$$
- If any of R1, R2, R3, U is lost, p is lost as well (except for T)



# Network security

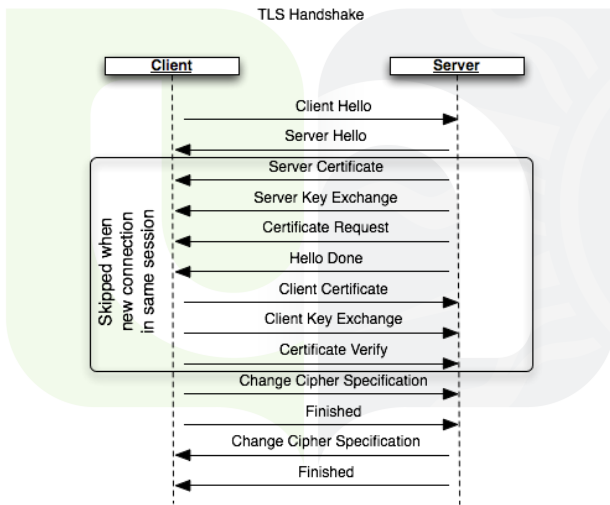
- IPSec – enriches IP traffic to ensure confidentiality, integrity and authenticity
- IKE – Internet Key Exchange
- SSL/TLS – Secure Socket Layer/Transport Layer Security
  - client-server model
  - relies on PKI and certificates
  - adds encryption on top of plain sockets to guarantee confidentiality, integrity and authenticity
  - supports server-only or mutual authentication
  - various versions: SSLv2, SSLv3 and SSLv3.1 (TLS v1)
  - supports many ciphers
  - independent of the upper layer protocol (SMTP, HTTP, POP, IMAP, LDAP...)

# Network security

## SSL/TLS

- 4 types of messages
  - Handshake: session parameters negotiation
  - Alert: error management
  - Change cipher spec: change in security parameters
  - Application Data
- Session and connection
  - security parameters negotiated at the session level
  - individual connections established within a single session
- Authentication takes place at the session level
- Connection keys derived from the master key

# SSL/TLS



# OpenSSL

- Generic toolbox that supports a wide range of crypto primitives:
  - key management for many protocols
  - key encoding in various formats
  - digest algorithms
  - digital signature computation and validation
  - full pki solution (X.509 certificates, CSRs and CRLs)
  - client/server to test SSL/TLS endpoints
  - S/MIME operations
  - secure timestamping

# OpenSSH

- OpenSSH

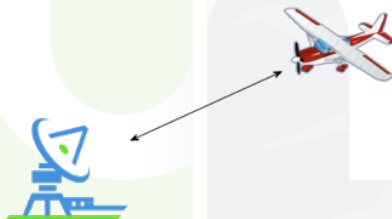
- offers secure versions of old plaintext protocols like ftp, telnet, rsh, rlogin through two tools: ssh and sftp
- supports strong cryptography (3DES, Blowfish, AES)
- supports strong authentication, incl. RSA et DSA
- supports port forwarding (incl. X11)
- nice to bypass some firewall protections

# XML security

- XML Signature (xml-dsig)
  - digital signature of XML documents
  - enveloping vs enveloped vs detached signature
  - signature element
  - supports various signature algorithms
    - DSA-SHA1
    - RSA-SHA1
- XML Encryption (xml-enc)
  - encryption of XML documents
  - supports various ciphers
- Similar RFC for JSON

# Authentication

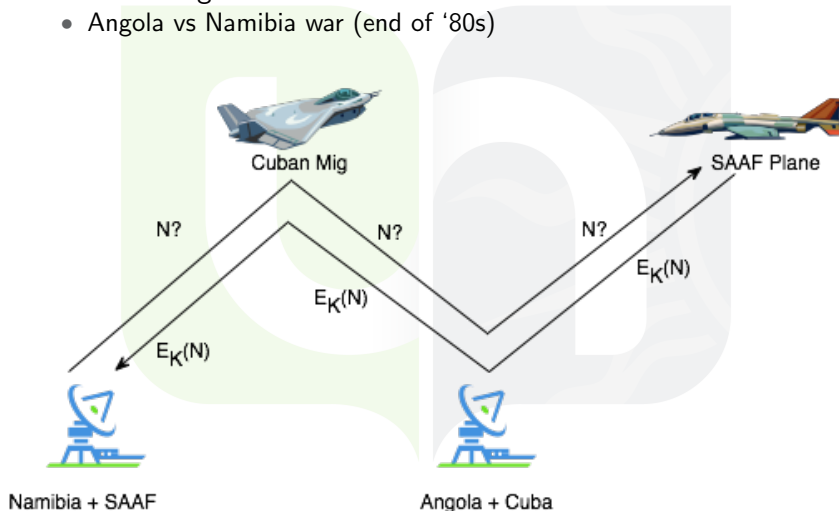
- IFF – Identification Friend or Foe
  - based on challenge/response protocol





# Authentication

- MITM = “Mig in the middle”
  - Angola vs Namibia war (end of '80s)



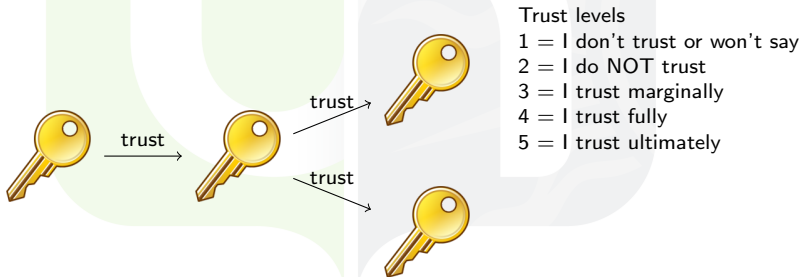
# GnuPG – Gnu Privacy Guard

- Open source implementation of OpenPGP standard
- Encryption of data and communication based on public key cryptography
- Key management, incl. revocation
- Public key repositories
- Each user has a pair of primary keys, plus pairs of secondary keys
- No certificate

# GnuPG – Gnu Privacy Guard

- Web of Trust

- IF  $\text{trust}(K_j)$  AND  $\text{signature}(K_j, K_i)$ , THEN  $\text{trust}(K_i)$
- rules can be more elaborated: length of trust chain, number of signatures used to sign the key, different levels of trust



# Agenda

## Cryptography

Introduction

Random numbers

Symmetric cryptography

Key management

Asymmetric cryptography

Digital signature

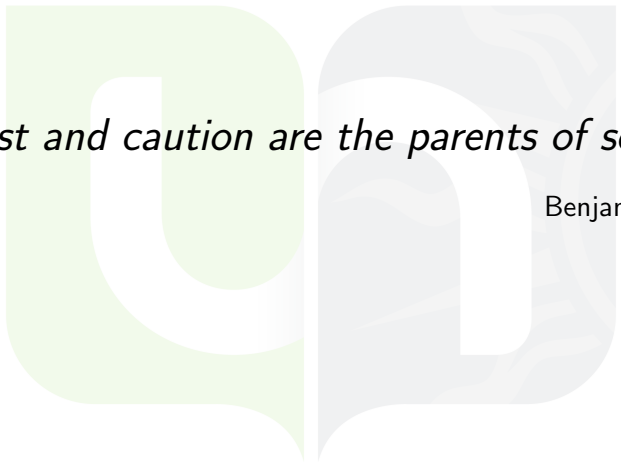
Public key infrastructure

Applications

Conclusion

# Conclusion

- Cryptography offers strong primitives providing security criteria (confidentiality, integrity, authenticity, non repudiation)
- It is hard to design crypto tools and protocols
- Do not reinvent the wheel, use existing and proven approaches
- Technology evolves, cryptanalysts work hard, thus re-assess the security of your crypto tools



*“Distrust and caution are the parents of security”*

Benjamin Franklin