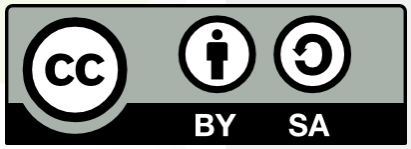
Computer Security

Introduction to Cryptography

Prof. Jean-Noël Colin
jean-noel.colin@unamur.be
Office #306

University of Namur
Computer science faculty

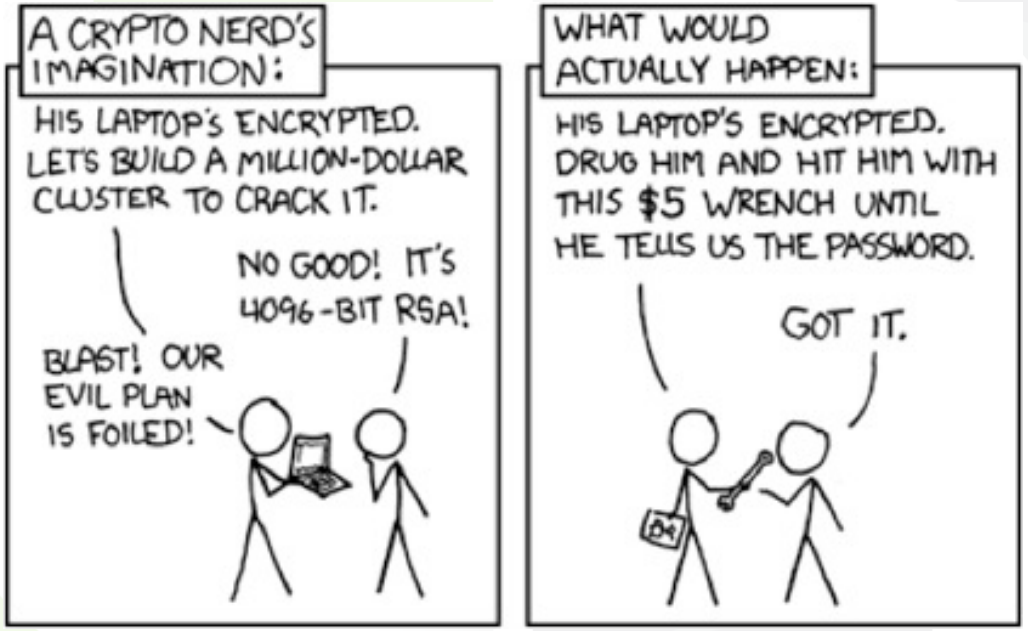
www.unamur.be



Agenda

- Cryptography
  - Introduction
  - Some historical approaches
  - Definitions of security
  - Cryptography and random numbers
  - Symmetric cryptography
  - Key management
  - Asymmetric cryptography
  - Hash functions
  - Message authentication and digital signature
  - Public key infrastructure
  - Applications
  - Conclusion

Introduction


A CRYPTO NERD'S IMAGINATION:
HIS LAPTOP'S ENCRYPTED. LET'S BUILD A MILLION-DOLLAR CLUSTER TO CRACK IT.
NO GOOD! IT'S 4096-BIT RSA!
BLAST! OUR EVIL PLAN IS FOILED!
WHAT WOULD ACTUALLY HAPPEN:
HIS LAPTOP'S ENCRYPTED. DRUG HIM AND HIT HIM WITH THIS $5 WRENCH UNTIL HE TELLS US THE PASSWORD.
GOT IT.

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

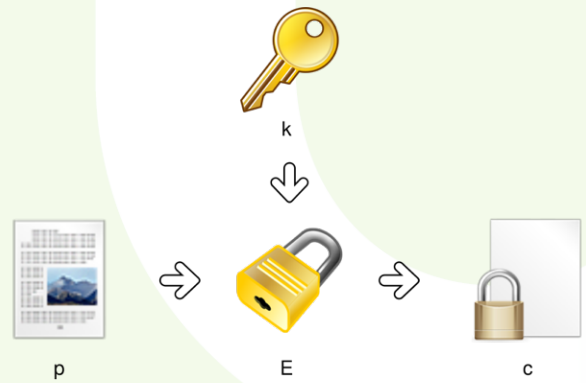
- Notation
  - p : plaintext , taken from set $\mathbb{P}$
  - c : ciphertext, taken from set $\mathbb{C}$
  - k : key, taken from set $\mathbb{K}$, the keyspace
  - E(k,p) : encryption cipher, where $k \in \mathbb{K}$ and $p \in \mathbb{P}$
  - D(k,c) : decryption cipher, where $k \in \mathbb{K}$ and $c \in \mathbb{C}$
- A cryptosystem is made of three primitives
  - KeyGen(): the mechanism to generate a (pair of) key(s)
  - E(k,p): the encryption mechanism (or cipher), where $k \in \mathbb{K}$ and $p \in \mathbb{P}$
  - D(k,c): the decryption mechanism (or cipher), where $k \in \mathbb{K}$ and $c \in \mathbb{C}$

Some vocabulary

- Let me introduce some friends:
  - Alice, Bob, Carol, Dave
  - Eve (eavesdropper), Mallory (malicious)
  - Trent (trusted arbitrator)
  - Walter (warden), Peggy (prover), Victor (verifier)

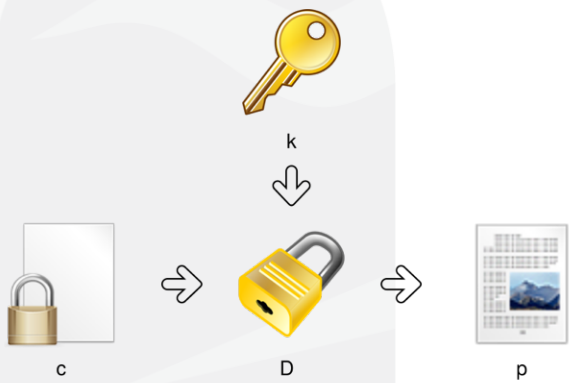
Core principle

Encryption



c = E(k,p)

Decryption



p= D(k,c)

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"

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

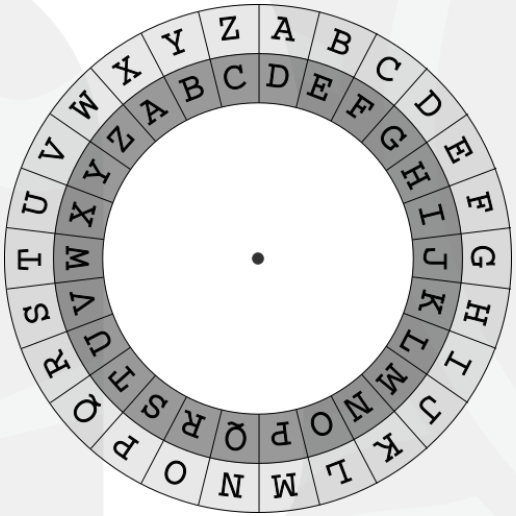
Attack the key

- Key size
  - 40bits → $2^{40} \approx 10^{12}$
  - 56bits → $2^{56} \approx 64.10^{15}$
  - 128, 256 (AES), 2048, 4096 (RSA)
- 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

Some historical approaches

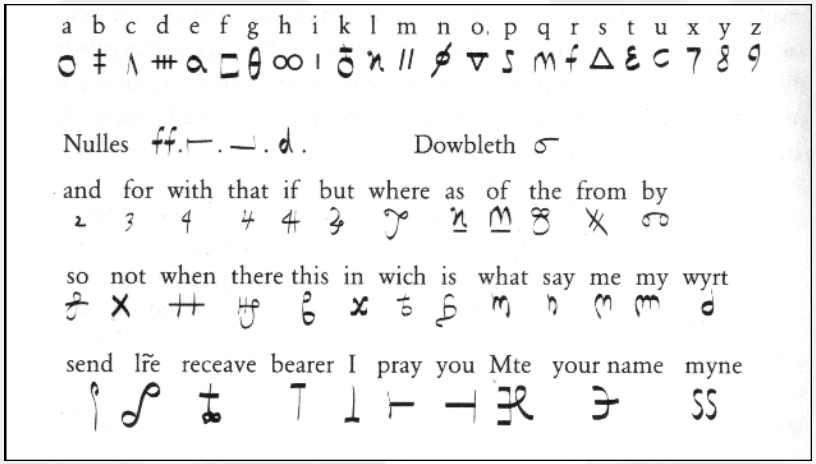
Shift cipher

- Caesar cipher
  - Ave Caesar ⇒ DYH FDHVDU
  - Easy to break (keyspace too small)



Substitution cipher

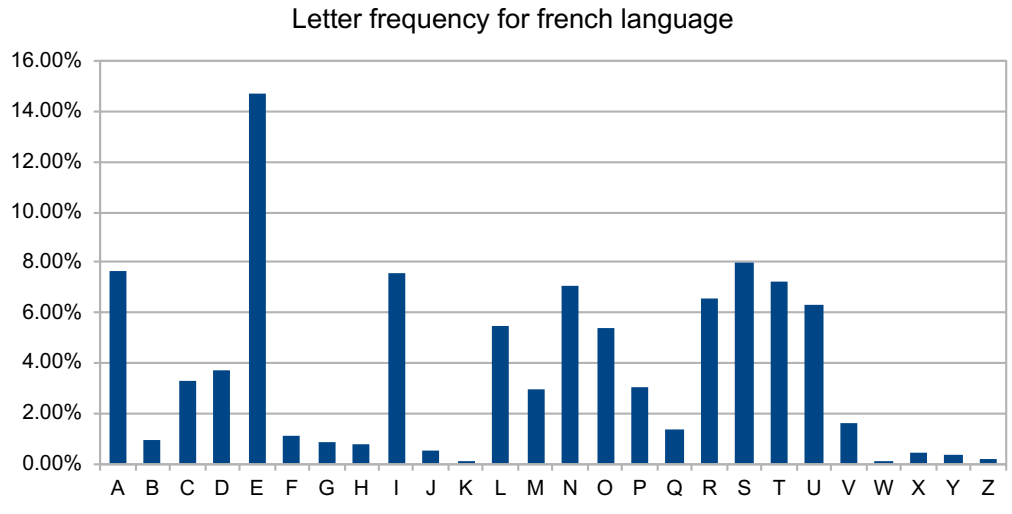
- Principle
  - substitute one letter with another; can also be applied to n-grams
  - based on a substitution alphabet (can be a permutation of the original alphabet)
  - Mono-alphabetic substitution is vulnerable to frequency analysis



Mary Stuart's code (XVI century)

Substitution cipher

- Frequency analysis



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

Training corpus has to be representative; avoid sentences like: "*De Zanzibar à la Zambie et au Zaïre, des zones d'ozone font courir les zèbres en zigzags zinzins.*"

Polyalphabetic substitution

- Vigenère Cipher (Blaise de Vigenère (1523-1596))
  - Polyalphabetic substitution
  - $c_i = p_i + k_i \bmod 26$
  - $c_i = i^{th}$ letter of ciphertext
  - $p_i = i^{th}$ letter of plaintext
  - $k_i = i^{th}$ letter of (repeated) key
  - A=0,B=1,C=2,…Z=25

Vigenère Cipher

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

Keytable

```
 ABCDEFGHIJKLMNOPQRSTUVWXYZ
MNOPQRSTUVWXYZABCDEFGHIJKL
YZABCDEFGHIJKLMNOPQRSTUVWX
PQRSTUVWXYZABCDEFGHIJKLMNO
RSTUVWXYZABCDEFGHIJKLMNOPQ
EFGHIJKLMNOPQRSTUVWXYZABCD
CDEFGHIJKLMNOPQRSTUVWXYZAB
IJKLMNOPQRSTUVWXYZABCDEFGH
OPQRSTUVWXYZABCDEFGHIJKLMN
UVWXYZABCDEFGHIJKLMNOPQRST
STUVWXYZABCDEFGHIJKLMNOPQR
```

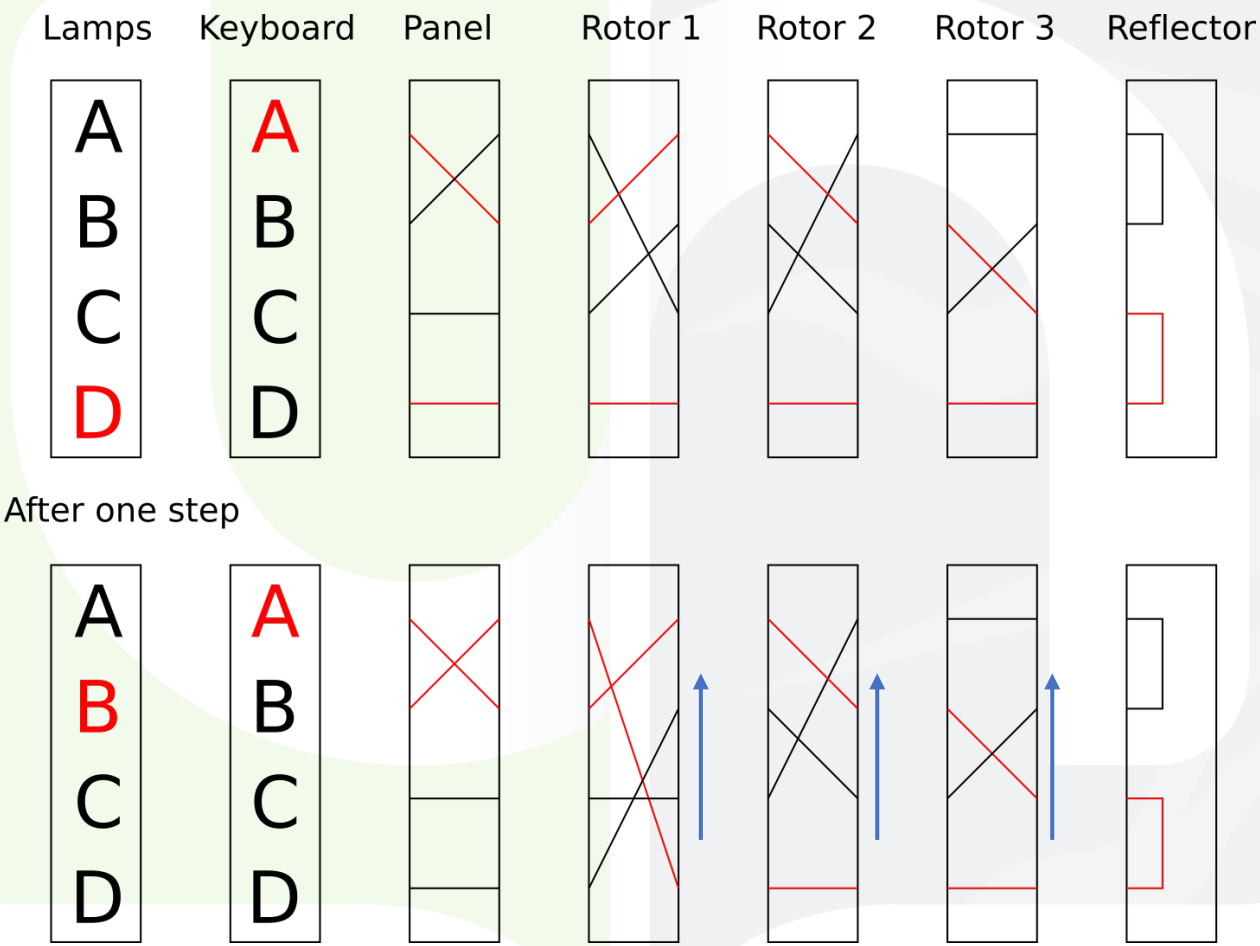
Enigma

Enigma

Enigma

- Invented early 20th century, used by the German army during WWII
- one of the first mechanical encryption machine
- Variable substitution through
  - connection panel
    - fixed substitution of 6 letters
  - rotors
    - variable substitution
    - rotation after each letter
  - reflector

Enigma


Lamps
Keyboard
Panel
Rotor 1
Rotor 2
Rotor 3
Reflector
A
B
C
D
A
B
C
D
After one step
A
B
C
D
A
B
C
D

Enigma

- Key length
  - The key is defined by the following elements
    - Connection setup (6 pairs of connected letters) $\frac{C_{26}^2 C_{24}^2 C_{22}^2 C_{20}^2 C_{18}^2 C_{16}^2}{6!} \approx 100.391.791.500$
    - Selection and order of rotors (3 out of 5):$A_5^3 = \frac{5!}{(5-2)!} = 60$
    - Initial positioning of rotors $26^3 = 17.576$
    - > 10^{16} possibilities
  - Additional security is obtained by
    - Adding more rotors
    - Adding notch and ring to rotors 1..n-1 to specify the letter triggering the move of next rotor

Definitions of security

Different definitions

- A crypto-system is
  - Computationally secure if the best known algorithm to break it requires N operations, where N is such a large number that it would be infeasible to carry them out. Currently, N = 2128 is considered as a good threshold. In other words, a system is computationally secure if it can't be broken by an adversary with bounded capabilities
  - Unconditionally (or perfectly) secure if it can't be broken, regardless of the adversary's capabilities. Also known as perfect secrecy or information-theoretic security

Attack the algorithm

- 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: len(E(k,p)) = len(p)
- Ciphertext must not reveal any information about the plain text

Probabilities

- Assuming X, Y, two random variables
- $\Pr[X \wedge Y]$, the joint probability of X and Y
- $\Pr[X|Y]$, the conditional probability of X given Y
- $\Pr[X \wedge Y] = \Pr[X] \,.\, \Pr[Y|X] = \Pr[Y] \,.\, \Pr[X|Y]$
- $\Pr[X|Y] = \frac{\Pr[X \wedge Y]}{\Pr[Y]}$
- If X and Y are independent
  - $\Pr[X \wedge Y] = \Pr[X] \,.\, \Pr[Y]$
  - $\Pr[X|Y] = \frac{\Pr[X \wedge Y]}{\Pr[Y]} = \frac{\Pr[X].\Pr[Y]}{\Pr[Y]} = \Pr[X]$
- Bayes' theorem
  - $\Pr[X|Y] = \frac{\Pr[X].\Pr[Y|X]}{\Pr[Y]}$

Perfect secrecy

- A crypto-system is *perfectly secure* if the ciphertext reveals no information about the plaintext, which can be formalized by

$$\Pr[P = p | C = c] = \Pr[P = p] \; \forall p \in \mathbb{P}, c \in \mathbb{C}. \qquad (1)$$

- Or equivalently

$$\Pr[C = c | P = p] = \Pr[C = c] \; \forall p \in \mathbb{P}, c \in \mathbb{C}. \qquad (2)$$

- Let's show that (2) ⇒ perfect secrecy

$$\Pr[P = p | C = c] = \frac{\Pr[P = p] \,.\, \Pr[C = c | P = p]}{\Pr[C = c]} \qquad \text{using Bayes' theorem}$$

$$= \frac{\Pr[P = p] \,.\, \Pr[C = c]}{\Pr[C = c]} \qquad \text{using (2)}$$

$$= \Pr[P = p] \qquad \text{thus we have perfect secrecy}$$

Perfect secrecy

- Let's assume $\mathbb{P} = \{a, b, c, d, e\}$ with the following probability distribution

a	b	c	d	e
0.3	0.25	0.1	0.2	0.15

- Let's assume $\mathbb{K} = \{k_1, k_2, k_3, k_4)$ with the following probability distribution

k_1	k_2	k_3	k_4
0.2	0.4	0.3	0.1

- Let's assume the following encryption table

k↓p→	a	b	c	d	e
k_1	1	5	2	3	4
k_2	3	1	2	4	5
k_3	5	1	4	2	3
k_4	5	1	3	4	2

Perfect secrecy

- Let's define $\mathbb{C}(k) = \{E_k(p) : p \in \mathbb{P}\}$, the set of ciphertexts under the key *k*
- One can compute the a priori probabilities of ciphertexts:

$$\Pr[C = c] = \sum_{k:c\in\mathbb{C}(k)} \Pr[K = k] . \Pr[P = p]$$

$$\begin{aligned}
\Pr[C = 1] &= \Pr[K = k_1] . \Pr[P = a] + \Pr[K = k_2] . \Pr[P = b] + \Pr[K = k_3] . \Pr[P = b] + \Pr[K = k_4] . \Pr[P = b] \\
&= 0.26 \\
\Pr[C = 2] &= \Pr[K = k_1] . \Pr[P = c] + \Pr[K = k_2] . \Pr[P = c] + \Pr[K = k_3] . \Pr[P = d] + \Pr[K = k_4] . \Pr[P = e] \\
&= 0.135 \\
\Pr[C = 3] &= \Pr[K = k_1] . \Pr[P = d] + \Pr[K = k_2] . \Pr[P = a] + \Pr[K = k_3] . \Pr[P = e] + \Pr[K = k_4] . \Pr[P = c] \\
&= 0.215 \\
\Pr[C = 4] &= \Pr[K = k_1] . \Pr[P = e] + \Pr[K = k_2] . \Pr[P = d] + \Pr[K = k_3] . \Pr[P = c] + \Pr[K = k_4] . \Pr[P = d] \\
&= 0.16 \\
\Pr[C = 5] &= \Pr[K = k_1] . \Pr[P = b] + \Pr[K = k_2] . \Pr[P = e] + \Pr[K = k_3] . \Pr[P = a] + \Pr[K = k_4] . \Pr[P = a] \\
&= 0.23
\end{aligned}$$

Perfect secrecy

- One can also compute the conditional probabilities of ciphertexts

$$\Pr[C = c | P = p] = \sum_{k:p=D(k,c)} \Pr[K = k]$$

c↓p→	a	b	c	d	e
1	0.2	0.8	0	0	0
2	0	0	0.6	0.3	0.1
3	0.4	0	0.1	0.2	0.3
4	0	0	0.3	0.5	0.2
5	0.4	0.2	0	0	0.4

- For example, knowing that *p=a*, the probability that *c=1* is equal to the probability that $k=k_1$; however, if *p=b*, the key used must be one of k_2,k_3,k_4 to have *c=1,* thus the probability is the sum of 0.4+0.3+0.1

Perfect secrecy

- An adversary is more interested in knowing the probability of a plaintext given a ciphertext, which, following Bayes' theorem, can be computed as

$$\Pr[P = p|C = c] = \frac{\Pr[C = c|P = p]\ \Pr[P = p]}{\Pr[C = c]}$$

- Thus for an adversary knowing the distribution of $\mathbb{K}$ and $\mathbb{P}$, and function E, it possible to compute these conditional probabilities which may reveal information about the plaintext from the ciphertext

Perfect secrecy

- What leaks out of those probabilities?

c↓p→	a	b	c	d	e
1	0.2	0.8	0	0	0
2	0	0	0.6	0.3	0.1
3	0.4	0	0.1	0.2	0.3
4	0	0	0.3	0.5	0.2
5	0.4	0.2	0	0	0.4

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

- If c=1, then p is either a or b,most likely b
- If c=2, then p can't be a or b, and there are equally high chances that it is c or d
- If c=3, then p is certainly not b, and there's a good chance that it's a
- If c=4, then p can't be aorb,andit'slikelythatit'sd
- If c=5, then p can't be c or d, and a is a more likely guess than b or d

Perfect secrecy

- About the sizes of $\mathbb{K}, \mathbb{P}, \mathbb{C}$
  - In a perfectly secure cryptosystem $|\mathbb{K}| \geq |\mathbb{C}| \geq |\mathbb{P}|$
  - Proof
    1. Encryption must be an injective map, thus $|\mathbb{C}| \geq |\mathbb{P}|$
    2. It can be assumed that $\Pr[C = c] > 0, \forall\, c \in \mathbb{C}$; if it was not the case, then $\mathbb{C}$ can be shrunk to those elements that meet this definition
    3. It follows that $\forall\, p \in \mathbb{P}, \Pr[C = c | P = p] = \Pr[C = c] > 0$, by definition of perfect secrecy and (2)
    4. Thus, $\forall\, p \in \mathbb{P}, \forall\, c \in \mathbb{C}, \exists k \in \mathbb{K}: c = E(k, p)$
    5. Hence $|\mathbb{K}| \geq |\mathbb{C}|$

Perfect secrecy: Shannon's theorem

- Consider a cryptosystem $\Pi = (\mathbb{K}, \mathbb{P}, \mathbb{C}, E, D)$ with $|\mathbb{K}| = |\mathbb{C}| = |\mathbb{P}|$
- Π provides perfect secrecy if and only if
  - (1) Every key is used with equal probability $\frac{1}{|\mathbb{K}|}$
  - (2) $\forall p \in \mathbb{P}, \forall c \in \mathbb{C}, \exists! \, k \in \mathbb{K}: c = E(k, p)$

Perfect secrecy: Shannon's theorem

- Proof part 1: perfect secrecy ⇒ (1) ∧ (2)
- (2)
  - We have already shown that $\forall\, p \in \mathbb{P}, \forall\, c \in \mathbb{C}, \exists k \in \mathbb{K}: c = E(k, p)$
  - We have assumed that $|\mathbb{K}| = |\mathbb{C}|$, thus $|\{\mathrm{E(k, p)}, \mathrm{k} \in \mathbb{K}\}| = |\mathbb{K}|$
  - Hence $\forall p \in \mathbb{P}, \forall c \in \mathbb{C}, \exists!\, k \in \mathbb{K}: c = E(k, p)$
- (1)
  - Let $\mathbb{P} = \{p_i : \forall 1 \leq i \leq n\}$ where $n = |\mathbb{P}|$; let's choose $c \in \mathbb{C}$, and label the keys $k_1, k_2 .. k_n$, such that $c = E(k_i, p_i), \forall 1 \leq i \leq n$

$$\Pr[P = p_i | C = c] = \frac{\Pr[C = c | P = p_i] \Pr[P = p_i]}{\Pr[C = c]}$$ Bayes theorem

$$= \frac{\Pr[K = k_i] \Pr[P = p_i]}{\Pr[C = c]}$$

$$\Pr[K = k_i] = \Pr[C = c]$$ def. of perfect secrecy

  - Perfect secrecy implies that $\Pr[C = c | P = p] = \Pr[C = c | P = p'] = \Pr[C = c]$; hence $\Pr[C = c] = \frac{1}{|\mathbb{C}|} = \frac{1}{|\mathbb{K}|}$
  - Thus all keys are used with equal probability

Perfect secrecy: Shannon's theorem

- Proof part 2: (1) ∧ (2) ⇒ perfect secrecy

$$\Pr[C = c] = \sum_k \Pr[K = k]\Pr[p = D(k, c)]$$

$$= \frac{1}{|\mathbb{K}|} \sum_k \Pr[p = D(k, c)] \qquad \text{thanks to (1)}$$

$$\sum_k \Pr[p = D(k, c)] = 1 \qquad \text{thanks to (2)}$$

$$\Pr[C = c] = \frac{1}{|\mathbb{K}|}$$

- Since $c = E(k, p)$, it follows

$$\Pr[C = c | P = p] = \Pr[K = k] = \frac{1}{|\mathbb{K}|} \qquad \text{(a)}$$

Perfect secrecy: Shannon's theorem

- Proof part 2: (1) ∧ (2) ⇒ perfect secrecy

$$\Pr[P = p|C = c] = \frac{\Pr[P = p]\Pr[C = c|P = p]}{\Pr[C = c]}$$ Bayes' theorem

$$= \frac{\Pr[P = p] \cdot \frac{1}{|\mathbb{K}|}}{\frac{1}{|\mathbb{K}|}}$$ using (a)

$$= \Pr[P = p]$$

We thus have perfect secrecy

Perfect secrecy: Shannon's theorem

- An example: modified shift cipher
  - assume that we use a different random letter to encrypt each letter of the plaintext
  - for a plaintext of length *n*, we thus generate the key as a string of *n* random letters, and encrypt by adding each letter of the plaintext to the corresponding letter of the key module 26, hence $c_i = (p_i + k_i) mod\ 26$
  - In this setup, it is obvious that $|\mathbb{K}| = |\mathbb{C}| = |\mathbb{P}| = 26^n$
  - Each key is uniformly selected at random, with $\Pr[K = k] = \frac{1}{26^n}$
  - For each *p,c* there is one unique key *k* such that $c = E(k, p)$
  - Hence, by Shannon's theorem, this scheme is perfectly secure

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) x0
- 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

Symmetric cryptography

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!

One-time pad (Vernam Cipher)

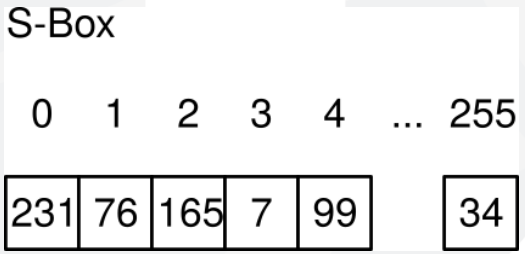
- Based on
  - the one-time usage
  - of a random stream of data (pad)
  - of the same size as the plaintext
- Encryption: $c_i = (p_i \oplus k_i)$
- Decryption: $p_i = (c_i \oplus k_i)$
- One time pad guarantees perfect secrecy if and only if the size of the keyspace is the same as the size of the plaintext space and keys are uniformly distributed in the keyspace (Shannon)
- But: key management complex ⇒ 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:
  - $c = (p \oplus k)$
  - $p = (c \oplus k)$
  - PRNG has to be cryptographically secure and properly initialized

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 ⊕ (xor'ed) with the plaintext (encryption) or ciphertext (decryption)
- Uses a substitution box updated after each use
- 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/decrypt byte per byte ⊕



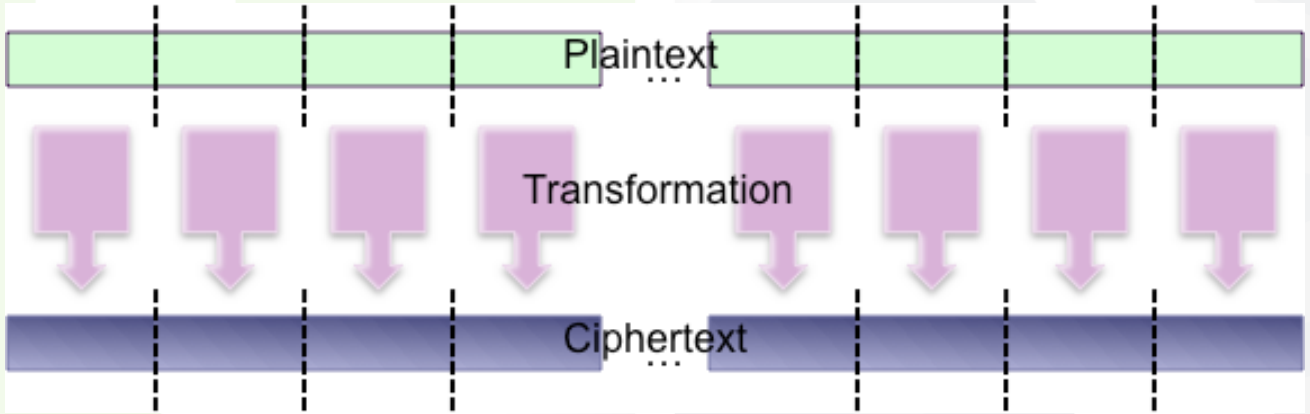
RC4 - Initialization

```
ARRAY_SIZE=256;
key="\x00\x01\x02\x03\x04\x05\x06\x07";
key='\xbe\x24\x43\x5e\x73\xaf\xbc\x7e\x26\x40\xa6\xef\x9d\x47\x27\xe2';
sBox = bytearray();

def init(key):
    print("Initializing RC4");
    sBox[0:]=[];
    for i in range(ARRAY_SIZE):
        sBox.append(i);
    j=0;
    byteKey = key.encode();
    for i in range(ARRAY_SIZE):
        a = sBox[i];
        b = byteKey[i % len(byteKey)];
        j = (a + b + j) % ARRAY_SIZE;
        buf = sBox[i];
        sBox[i] = sBox[j];
        sBox[j] = buf;
```

RC4 – encryption/decryption

```
def encrypt(bytePlainText):
    i = 0;
    j = 0;
    t = 0;
    byteCipherText = bytearray();
    for l in range(len(bytePlainText)):
        i = (i + 1) % ARRAY_SIZE;
        j = (j + sBox[i]) % ARRAY_SIZE;
        buf = sBox[i];
        sBox[i] = sBox[j];
        sBox[j] = buf;
        t = (sBox[i] + sBox[j]) % ARRAY_SIZE;
        byteCipherText.append(bytePlainText[l] ^ sBox[t]);
    return byteCipherText;

def decrypt(ciphertext):
    return(encrypt(ciphertext));
```

Block cipher

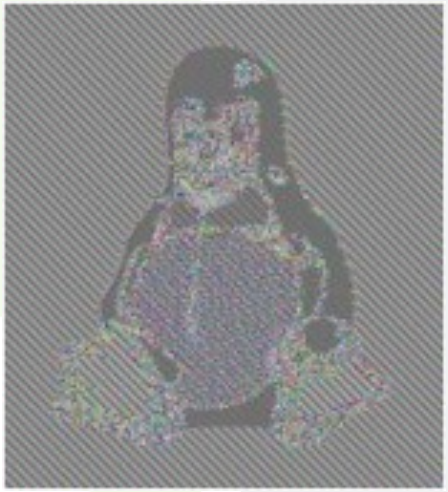


Block cipher

- 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)

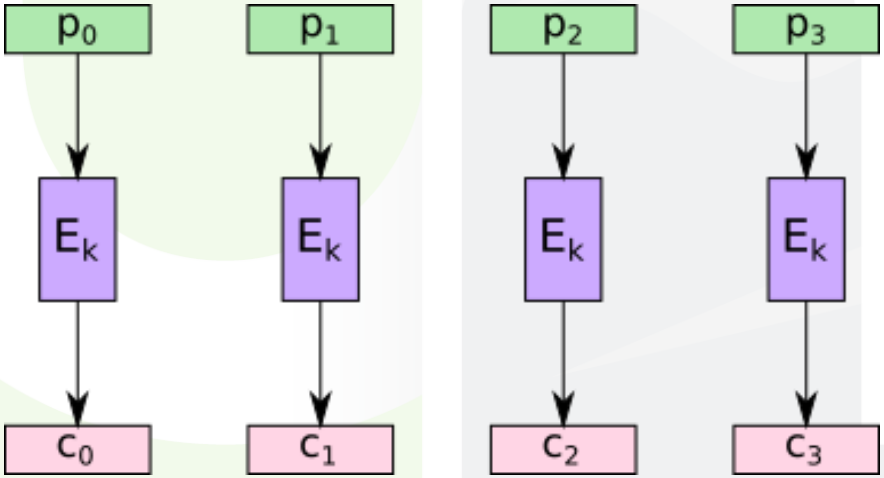
Modes of operation

- When the size of plaintext is larger than the size of a block, is it safe to encrypt all blocks independently?



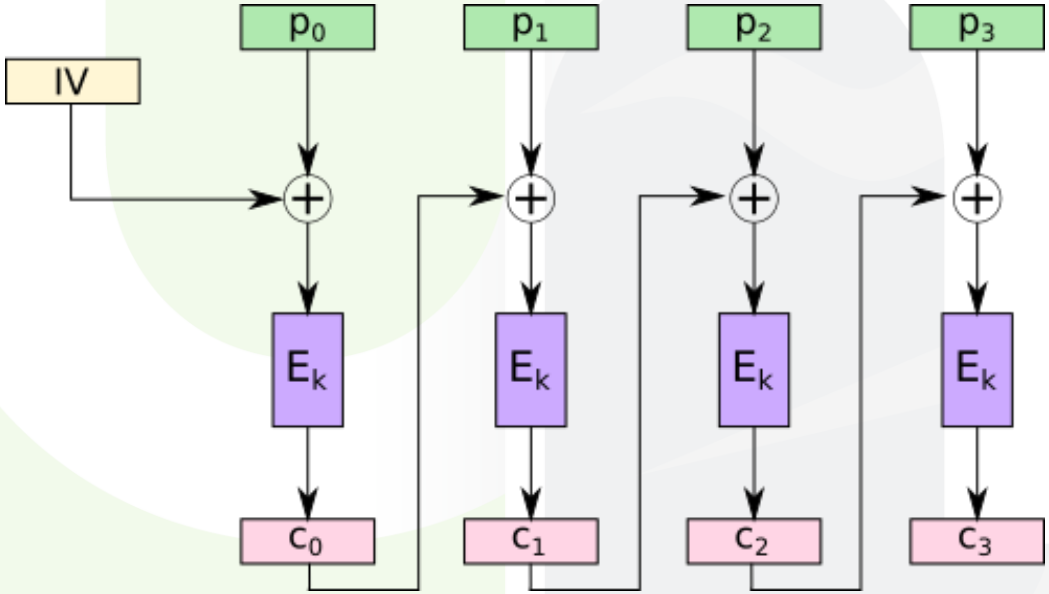
Modes of operation

- ECB – Electronic CodeBook Mode
  - repeating sequences can easily be detected
  - tampering is easy



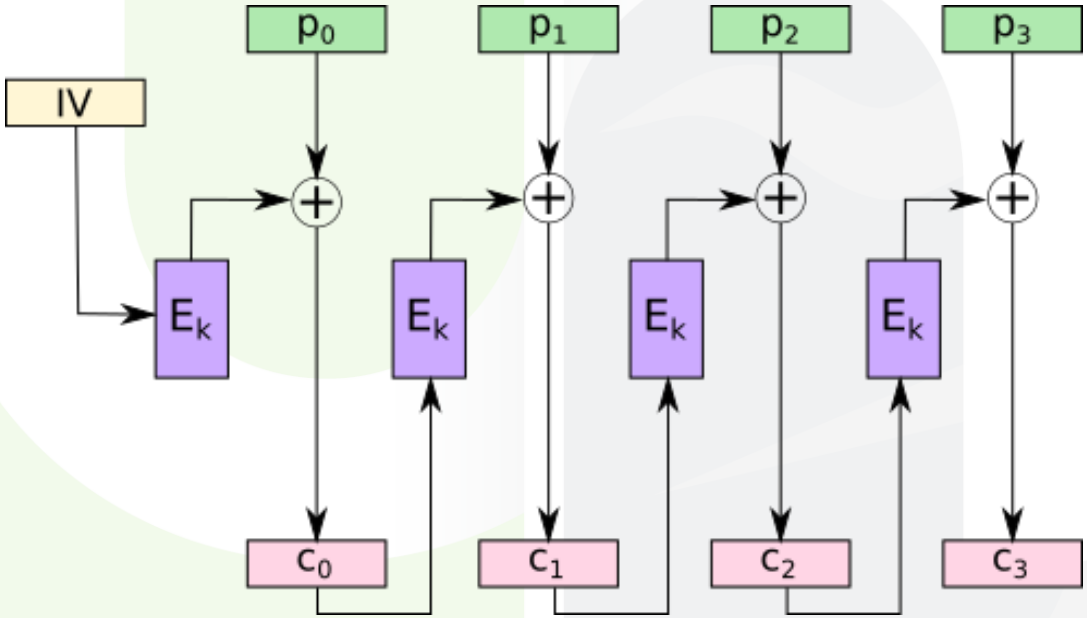
Modes of operation

- CBC – Cipher Block Chaining



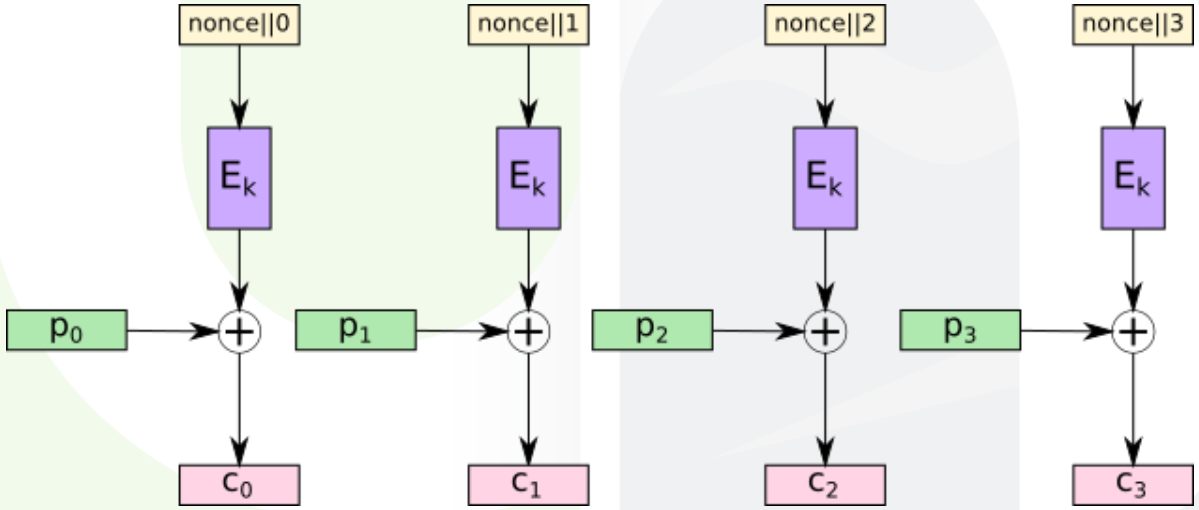
Modes of operation

- CFB – Cipher FeedBack Mode



Modes of operation

- CTR – Counter Mode



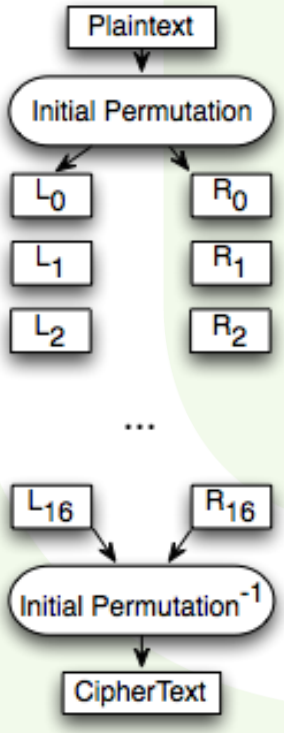
Padding

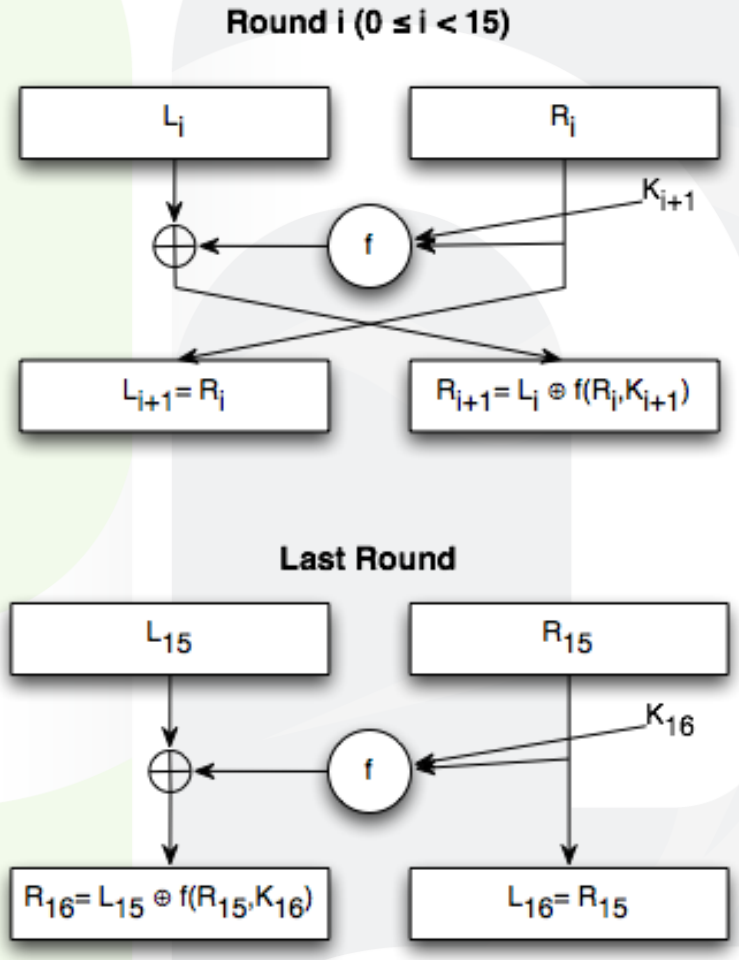
- When the size of plaintext is not a multiple of the block size, how to ‘pad’ data so that the recipient can differentiate between data and padding bytes unambiguously?
- Examples (PKCS5, PKCS7):
  - count number of bytes to add
  - repeat the value in the bytes to pad in the last block
  - recipient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

Digital Encryption Standard - DES

- Origin: US
- 56 bits key
- block cipher (block size: 64bits)
- design objective: make each bit of the ciphertext dependent on every bit of the plaintext and the key as quickly as possible
- based on Feistel structure: use the same function to encrypt and decrypt, so that the function does not have to be invertible
- 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)

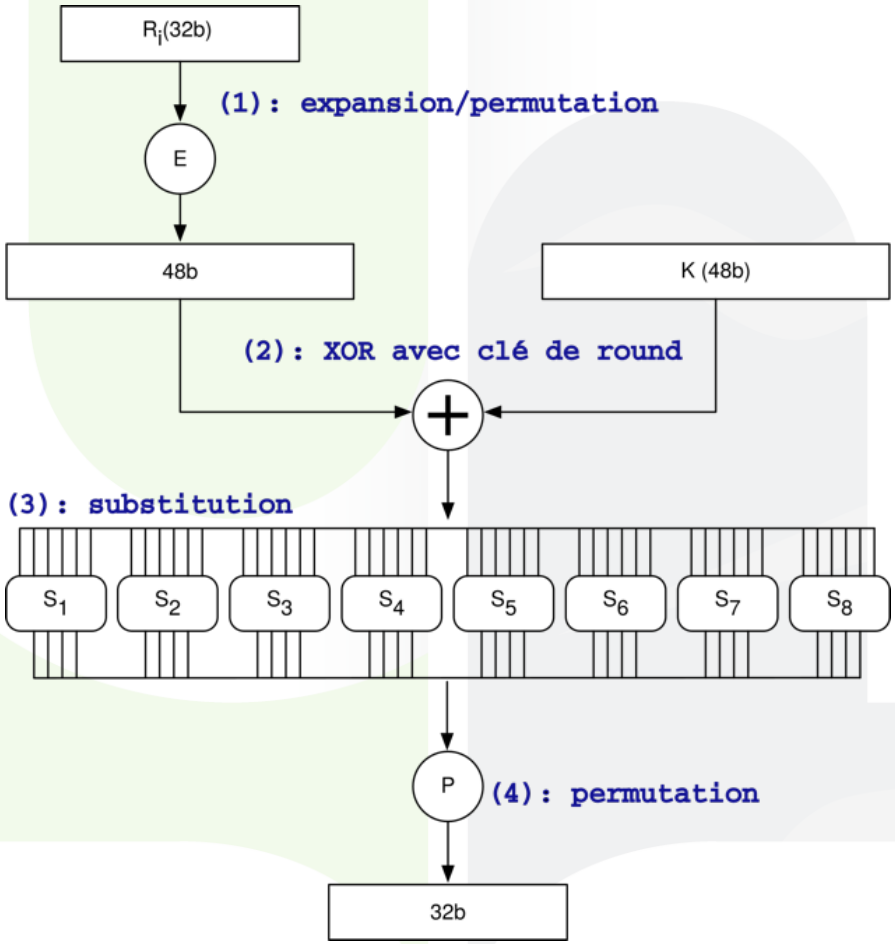
Digital Encryption Standard - DES





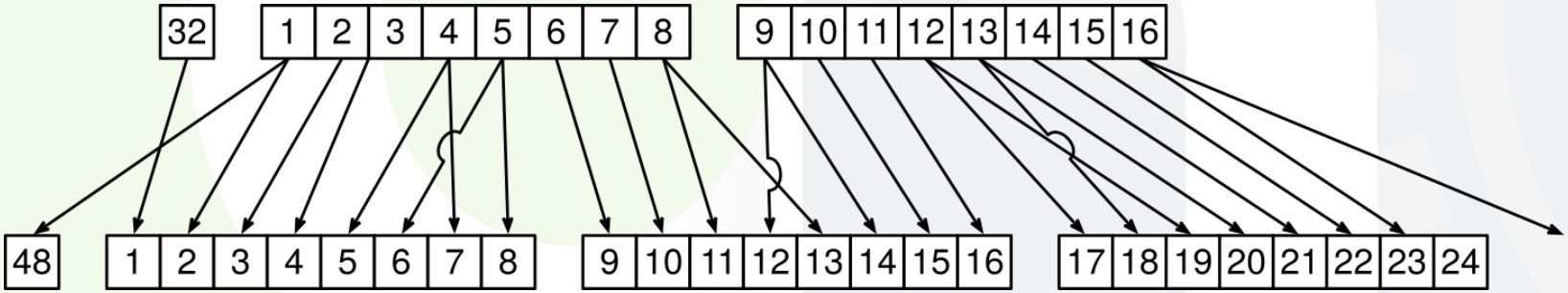
Digital Encryption Standard - DES

- *f* function



Digital Encryption Standard - DES

- *f*: Expansion/Permutation
  - 'inflates' Ri 32 → 48 bits



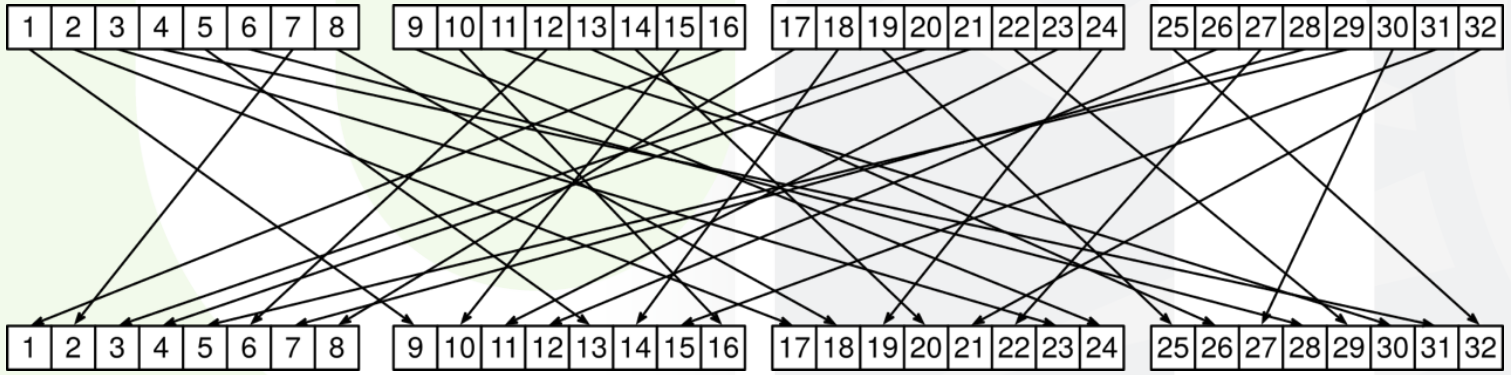
Digital Encryption Standard - DES

- 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 → 48 bits in, 32 bits out
    - $b_1b_2b_3b_4b_5b_6$
    - b_1b_6 define the row
    - $b_2b_3b_4b_5$ 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(k1, k2, k3)
  - c = E(k3, D(k2, E(k1, 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 length: 128, 192 or 256
- Number of rounds: 10, 12 or 14 (depending on key length)
- Demo
- Other block ciphers: RC2, RC5, IDEA, Blowfish

Key management

Key management and distribution

- Key management
  - manage: create, expire, revoke, send, store...
  - threats: loss, theft, compromise, extorsion...
- 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
- ⇒ 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!

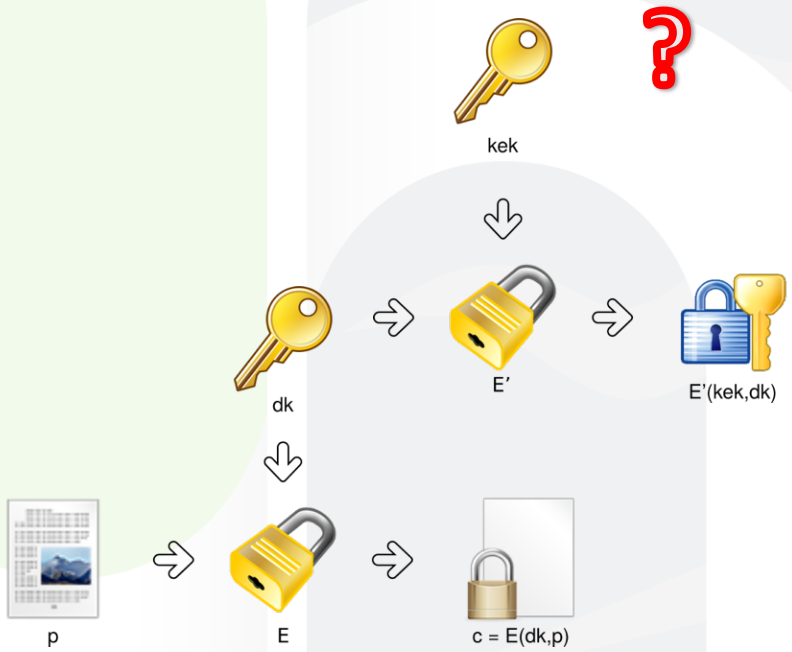
Key escrow

- 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)

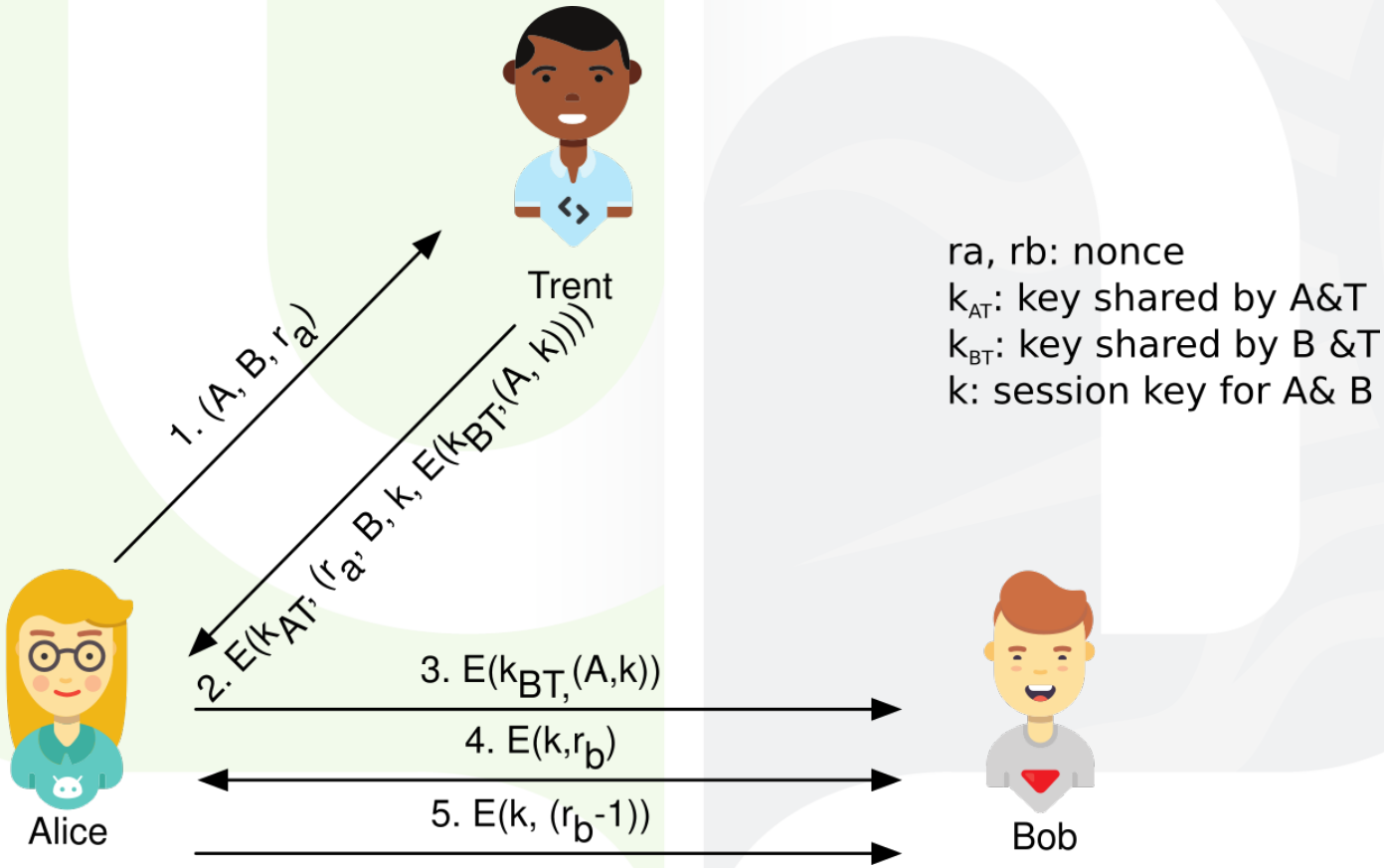


Password based encryption (PBE)

- Encryption
  - generate (dk)
  - enter password
  - generate random salt (s)
  - mix password and s
  - build kek from result of previous step
  - encrypt dk with kek
  - 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) → dk
- Why use a salt?
  - increase entropy
  - avoid pre-computation attacks

Key distribution via trusted third party

- Needham Schroeder



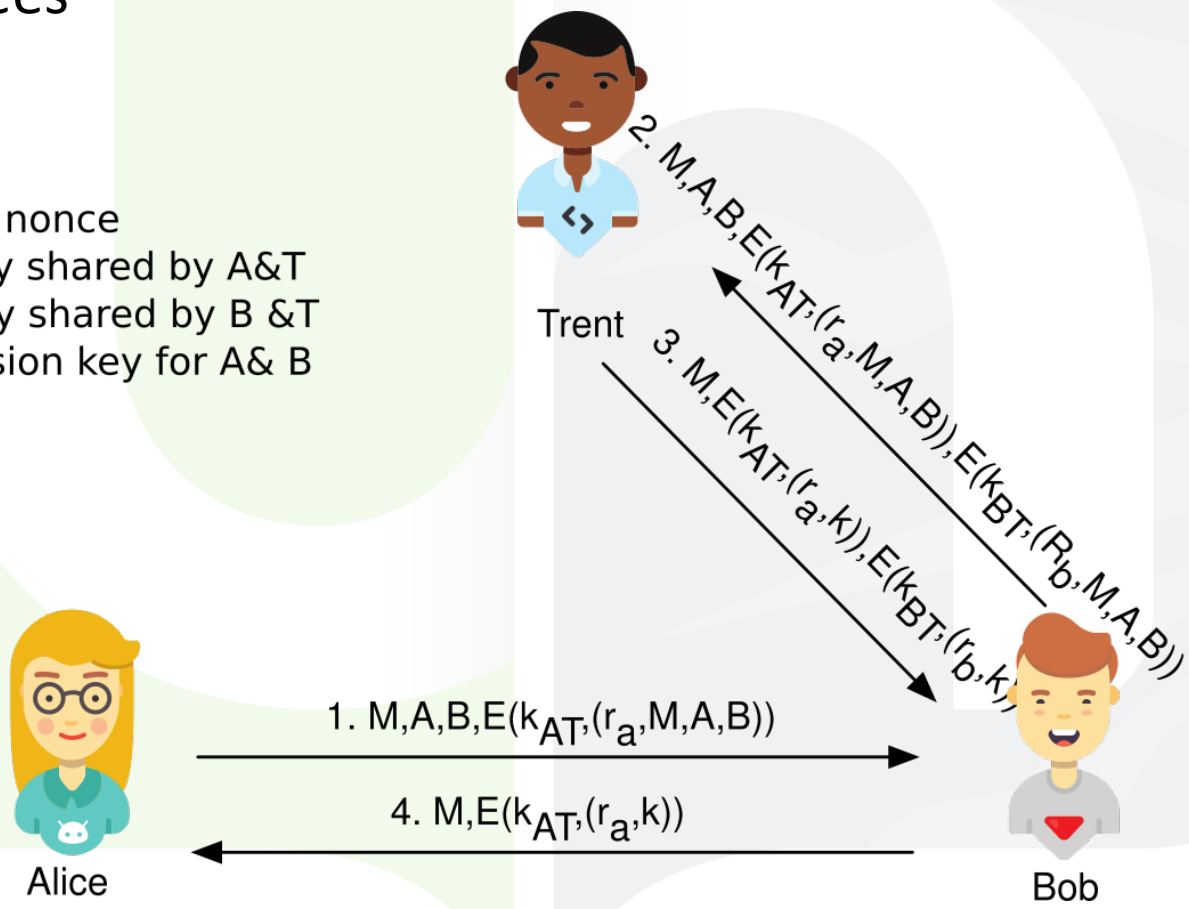
Key distribution via trusted third party

- 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

ra, rb: nonce
k_{AT}: key shared by A&T
k_{BT}: key shared by B &T
k: session key for A& B



Asymmetric cryptography

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

Which combinations are valid?

$$c = E(K_A^-, p)$$
$$c = E(K_A^+, p)$$
$$c = E(K_B^-, p)$$
$$c = E(K_B^+, p)$$

$$p = D(K_A^+, c)$$
$$p = D(K_A^-, c)$$
$$p = D(K_B^+, c)$$
$$p = D(K_B^-, c)$$

K_A^-, K_A^+, K_B^+

K_B^-, K_B^+, K_A^+

Core principle

- 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:
    - 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
    - A randomly chooses a (large int) and computes $g^a mod\ p$
    - B randomly chooses b (large int) and computes $g^b mod\ p$
    - A → B : $g^a mod\ p$, which allows B to compute $k = (g^a mod\ p)^b mod\ p$
    - B → A : $g^b mod\ p$, which allows A to compute $k = (g^b mod\ p)^a mod\ p$
    - $g^{ab} mod\ p$ is the new shared secret
- Security of the algorithm
  - assuming Eve gets $g^a mod\ p$ and $g^b mod\ p$; to get $g^{ab} mod\ p$, she needs to compute a from $g^a mod\ p$ (or b from $g^b mod\ p$), which is known as the discrete logarithm problem, with no easy solution
  - vulnerable to a MITM attack; harder if $g^a mod\ p$ and $g^b mod\ p$ 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.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 $d.e \equiv 1 \bmod \varphi(n)$
  - $K^+ = (n,e)$
  - $K^- = (n,d)$
  - typically, length of p and q > 1000 bits (and n is twice as long)

RSA Algorithm

$c = E(K_B^+, p)$
$c = p^{e_B} \bmod n$

$p = D(K_B^-, c)$
$p = c^{d_B} \bmod n$

c

p<n

K_A^-, K_A^+, K_B^+

K_B^-, K_B^+, K_A^+

RSA Algorithm

- Security of the algorithm
  - assuming Eve intercepts $c = p^{e_B} \; mod \; n$, e and n being public
    - to find m, Eve needs d_B
    - Eve knows that $d.e = 1 \; mod \; (p-1)(q-1)$
    - e being public, Eve needs to find p and 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 → n = p.q = 33
- φ(n) = (p−1)(q−1) = 20
- let's set e = 3
  - e and φ(n) are relatively prime
    - gcd(e,p−1) = gcd(3,10) = 1
    - gcd(e,q−1) = gcd(3,2) = 1
    - thus gcd(e,φ(n)) = 1
  - let's find d such that e.d = 1 mod (φ(n))
    - d s.t. φ(n) divides e.d–1
    - d s.t. 20 divides 3.d −1
  - K+ = (33,3)
  - K− = (33,7)

RSA Algorithm: 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 = ax mod n but hard to find x knowing that ax ≡ b mod n
- Principle
  - choose n, prime and g, generator of the finite cyclic group $G = \mathbb{Z}/p\mathbb{Z}$
  - Alice randomly selects $K_A^- = a$ in the group and computes $K_A^+ = g^a mod\ n$, her public key
  - Bob randomly selects $K_B^- = b$ in the group and computes $K_B^+ = g^b mod\ n$, his public key
  - encryption (A → B)
    - $c_1 = K_A^+$
    - $c_2 = m(K_B^+)^a\ mod\ n = m.g^{ab} mod\ n$
    - A → B : (c_1,c_2)
  - decryption
    - $m = \frac{c_2}{{c_1}^b}\ mod\ n = \frac{m.g^{ab} mod\ n}{g^{ab} mod\ n}$

Hash functions

Message digest

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

```
snoopy:~ jnc$ shasum
Hello world!
47a013e660d408619d894b20806b1d5086aab03b  -
snoopy:~ jnc$ shasum
Quousque tandem abutere, Catilina, patientia nostra?
a081ccb6070a9a4cf5137c8bff6ecad80a6a50d9  -
snoopy:~ jnc$ shasum
Quousque tandem abutere, Catilina, patientia nostra!
6be608c1cc7a1f63a04508967d4d98cd0882adcd  -
```

Message digest

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

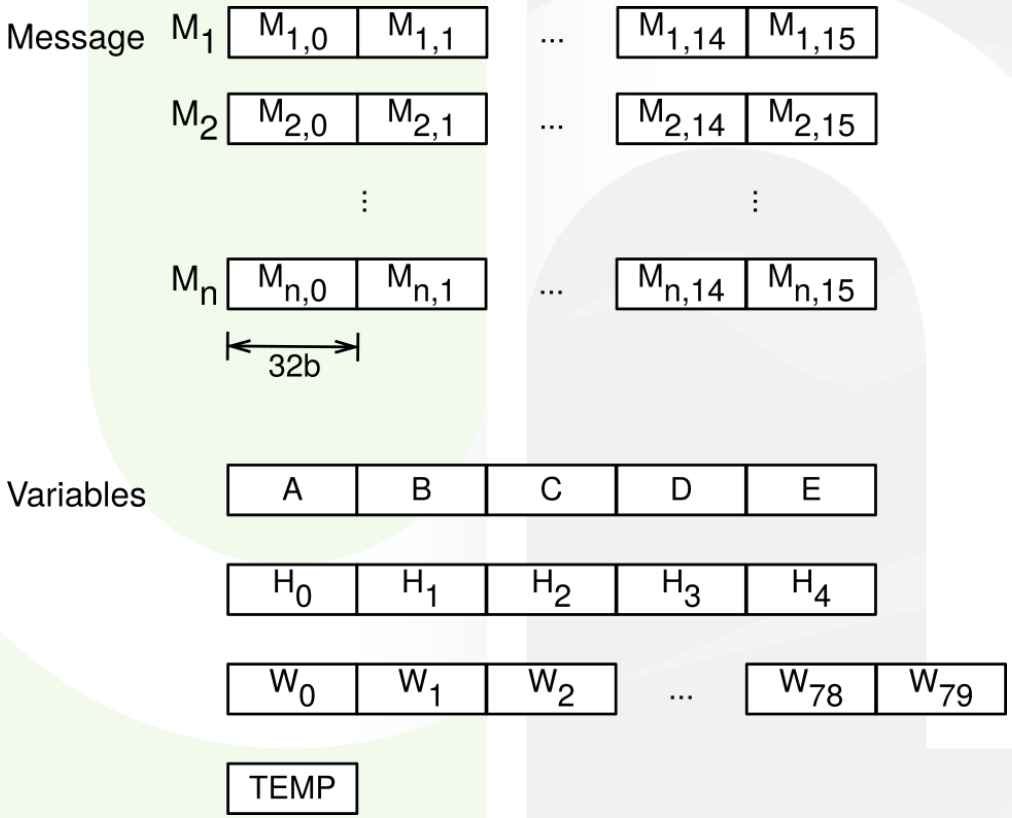
Message digest

- Examples
  - MD2
    - invented by Ron Rivest (R de RSA)
    - 128 bit digest (2128 possible digests)
    - weaknesses found
    - collisions found, but not on demand
    - obsolete
  - MD5
    - invented by Ron Rivest (R from 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.”

SHA-1 Algorithm



SHA-1 Algorithm

(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 $= H_0 H_1 H_2 H_3 H_4)$

SHA-1 Algorithm

- 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 $(60 \leq i \leq 79)$
- Ki Constants
  - 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 $(60 \leq i \leq 79)$

SHA-1 Algorithm

- 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 = 56bits (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
```

Use digest to check message integrity

$2.\ (p, h)$

$1.\ h = H(p)$

$3.\ h' = H(p)$
if (h==h'), integrity of *p* is verified

Message authentication and digital signature

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

Use digest to check message authenticity

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

Authenticity?

$2.\ (p, mac)$

$1.\ mac = H(f(k,p))$

$3.\ h' = H(f(k,p))$
if (mac==h'), integrity of *p* is verified

Don't use f(k,p)=k||p !
Length extension attack

Use digest to check message authenticity

- Solution 2: digital signature
  - based on asymmetric crypto

$3.\ (p, sig_A)$

1. $h = H(p)$
2. $sig_A = E(K_A^-, h)$

$4.\ h' = D(K_A^+, sig_A)$
$5.\ h'' = H(p)$
if (h'==h''), integrity and authenticity of *p* are verified

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

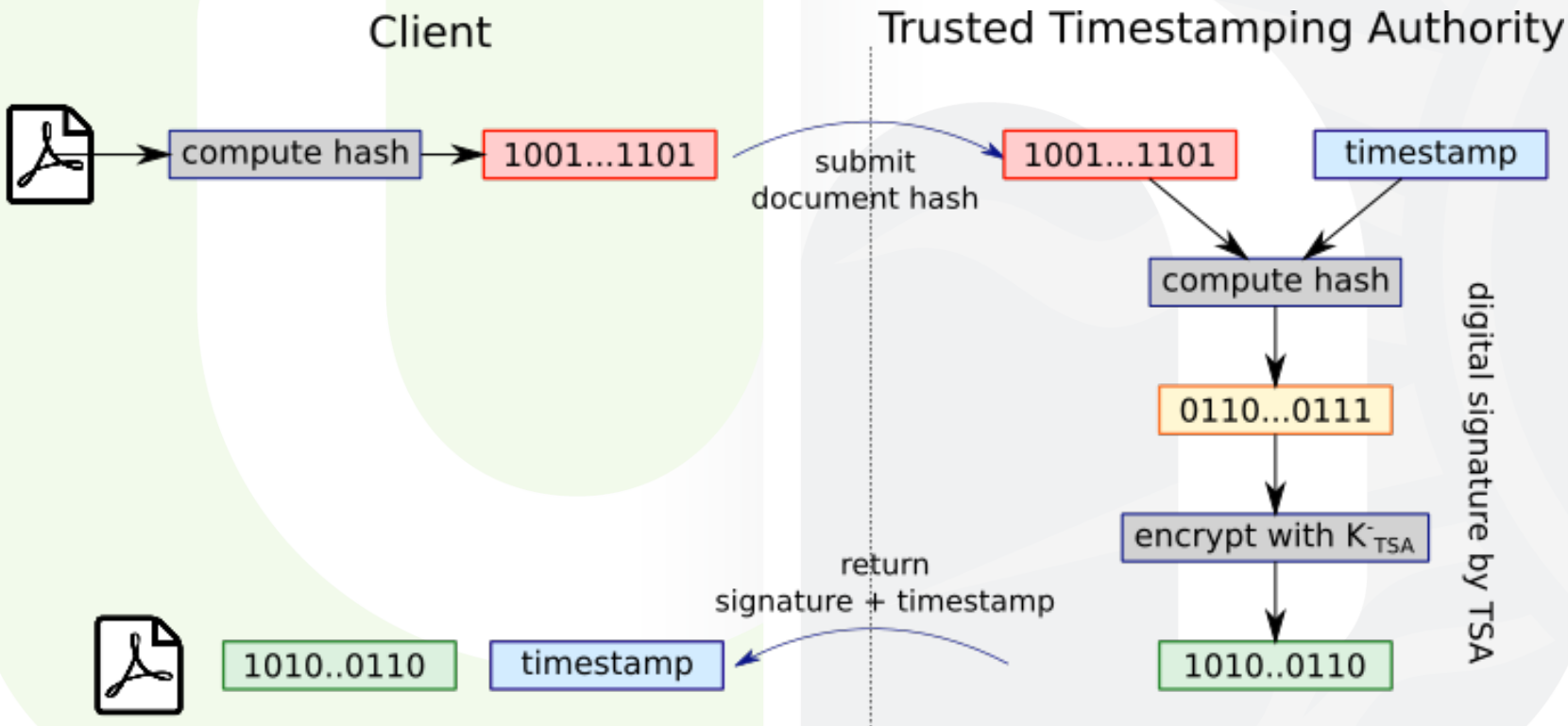
Digital signature

- 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

Advanced usages

- 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 $0 <= i < n$
- 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

Example – trusted timestamping

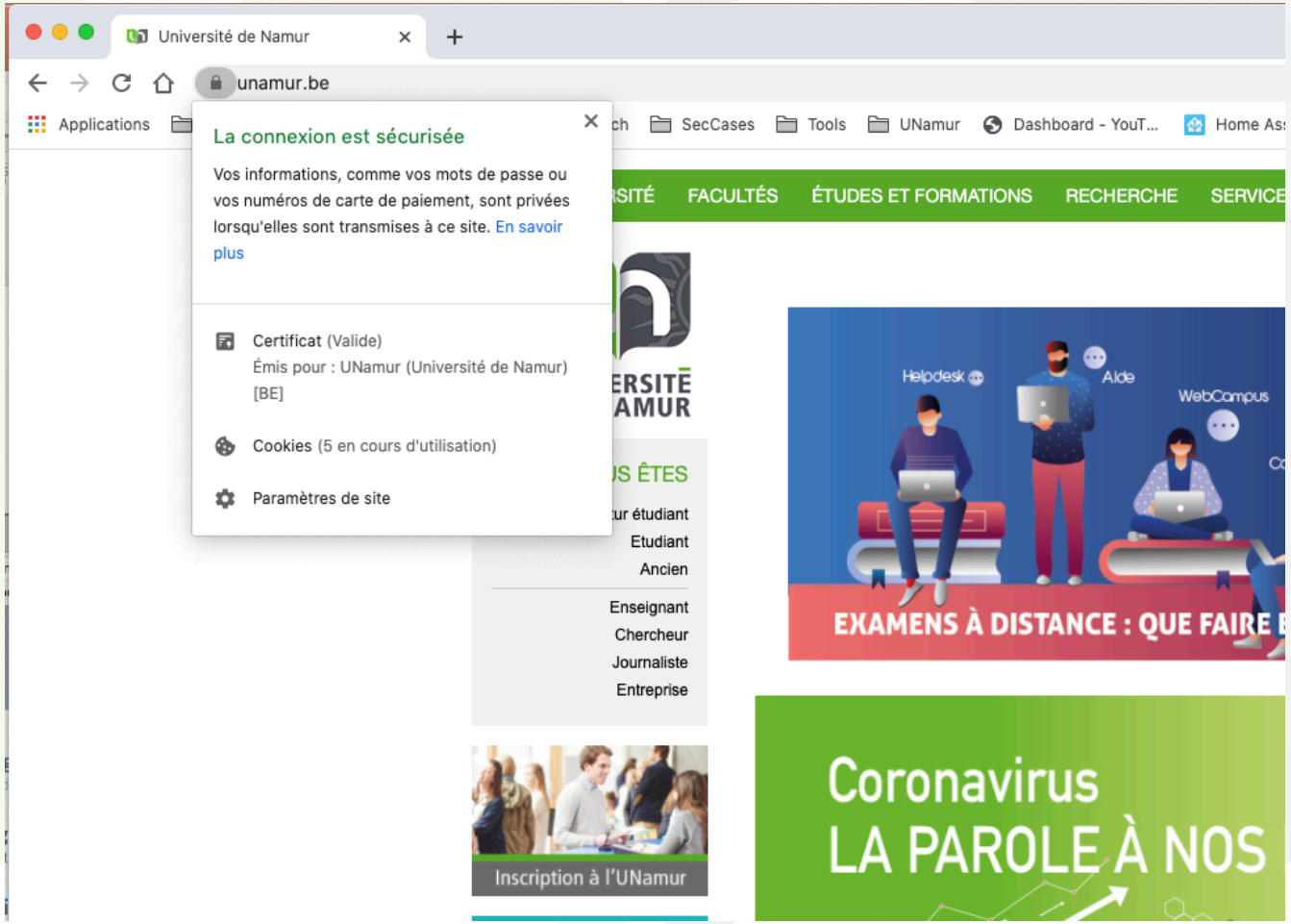


Public Key Infrastructure

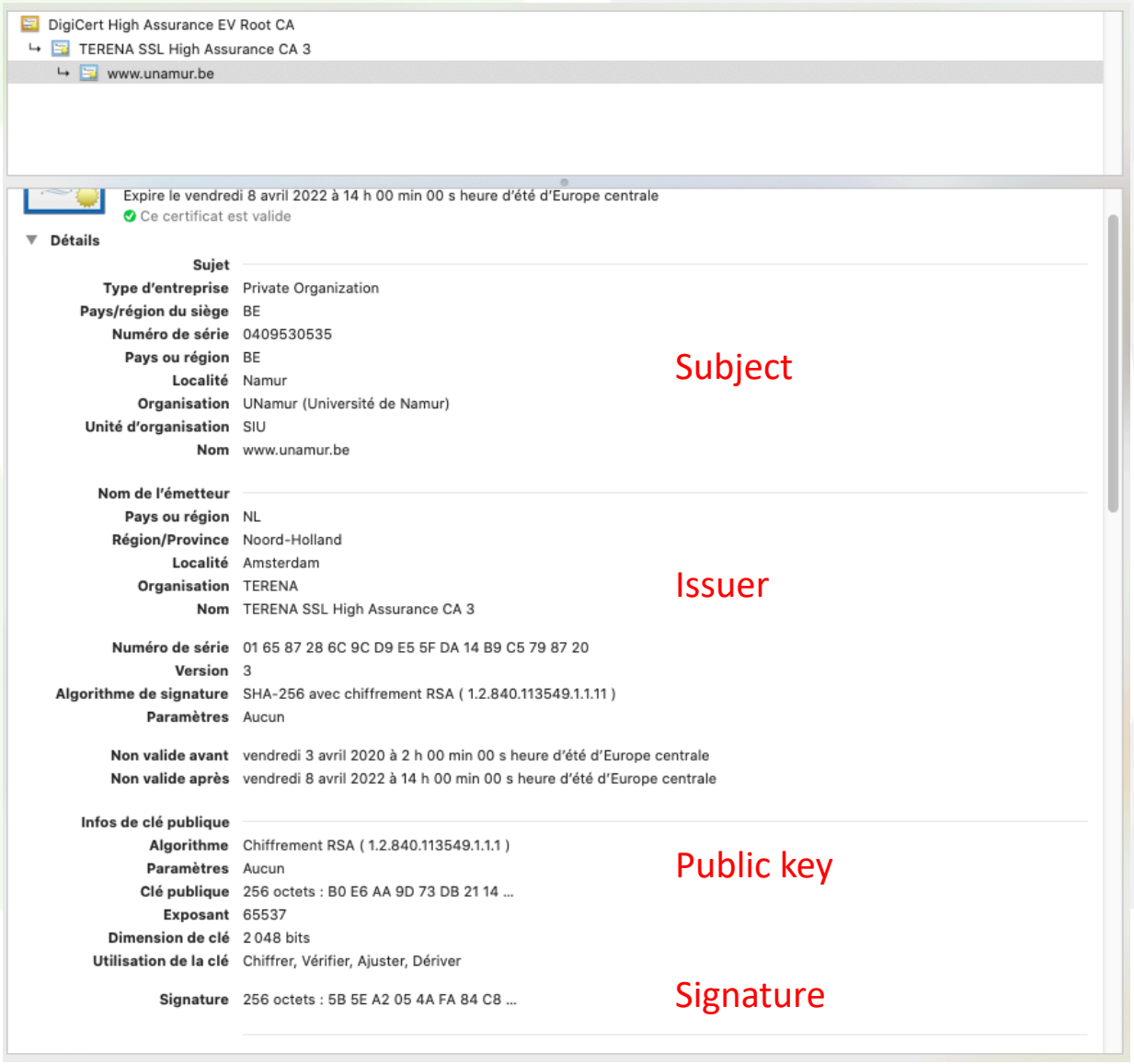
Digital certificate and PKI

- 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


Université de Namur
unamur.be
Applications
SecCases
Tools
UNamur
Dashboard - YouT...
La connexion est sécurisée
Vos informations, comme vos mots de passe ou vos numéros de carte de paiement, sont privées lorsqu'elles sont transmises à ce site. En savoir plus
Certificat (Valide)
Émis pour : UNamur (Université de Namur) [BE]
Cookies (5 en cours d'utilisation)
Paramètres de site
FACULTÉS
ÉTUDES ET FORMATIONS
RECHERCHE
Etudiant
Ancien
Enseignant
Chercheur
Journaliste
Entreprise
Helpdesk
Aide
WebCampus
EXAMENS À DISTANCE : QUE FAIRE
Inscription à l'UNamur
Coronavirus
LA PAROLE À NOS

Example


DigiCert High Assurance EV Root CA
TERENA SSL High Assurance CA 3
www.unamur.be
Expire le vendredi 8 avril 2022 à 14 h 00 min 00 s heure d'été d'Europe centrale
Ce certificat est valide
Détails
Sujet
Type d'entreprise Private Organization
Pays/région du siège BE
Numéro de série 0409530535
Pays ou région BE
Localité Namur
Organisation UNamur (Université de Namur)
Unité d'organisation SIU
Nom www.unamur.be
Subject
Nom de l'émetteur
Pays ou région NL
Région/Province Noord-Holland
Localité Amsterdam
Organisation TERENA
Nom TERENA SSL High Assurance CA 3
Issuer
Numéro de série 01 65 87 28 6C 9C D9 E5 5F DA 14 B9 C5 79 87 20
Version 3
Algorithme de signature SHA-256 avec chiffrement RSA (1.2.840.113549.1.1.11)
Paramètres Aucun
Non valide avant vendredi 3 avril 2020 à 2 h 00 min 00 s heure d'été d'Europe centrale
Non valide après vendredi 8 avril 2022 à 14 h 00 min 00 s heure d'été d'Europe centrale
Infos de clé publique
Algorithme Chiffrement RSA (1.2.840.113549.1.1.1)
Paramètres Aucun
Clé publique 256 octets : B0 E6 AA 9D 73 DB 21 14 ...
Exposant 65537
Dimension de clé 2 048 bits
Utilisation de la clé Chiffrer, Vérifier, Ajuster, Dériver
Public key
Signature 256 octets : 5B 5E A2 05 4A FA 84 C8 ...
Signature

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
  - businessCategory = Private Organization, jurisdictionC = BE, serialNumber = 0409530535, C = BE, L = Namur, O = UNamur (Universit\C3\A9 de Namur), OU = SIU, CN = www.unamur.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

```
Kermit:Desktop jnc$ openssl x509 -inform der -in www.unamur.be.cer -noout -text
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            01:65:87:28:6c:9c:d9:e5:5f:da:14:b9:c5:79:87:20
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=NL, ST=Noord-Holland, L=Amsterdam, O=TERENA, CN=TERENA SSL High
Assurance CA 3
        Validity
            Not Before: Apr  3 00:00:00 2020 GMT
            Not After : Apr  8 12:00:00 2022 GMT
        Subject: businessCategory=Private
Organization/jurisdictionCountryName=BE/serialNumber=0409530535, C=BE, L=Namur,
O=UNamur (Universit\xC3\xA9 de Namur), OU=SIU, CN=www.unamur.be
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:b0:e6:aa:9d:73:db:21:14:c8:40:f8:b5:98:ad:
                    90:9c:23:b9:88:4e:42:86:29:22:65:01:42:c5:58:
                    5a:e0:fc:00:f7:ca:46:37:3b:ea:75:d1:6f:44:87:
                    af:f4:fe:9d:c3:d0:cb:19:31:2a:d5:61:18:07:49:
                 ...
                    be:eb:a9:11:a7:6e:c5:51:3a:eb:96:8a:b8:f5:da:
                    56:75
                Exponent: 65537 (0x10001)
```

Example

```
X509v3 extensions:
            X509v3 Authority Key Identifier:
                keyid:C2:B8:85:D7:E1:B9:13:BD:D1:48:BC:FD:5E:DC:7D:90:42:7A:8A:A9

            X509v3 Subject Key Identifier:
                DB:A4:53:A6:DC:05:32:39:4B:C8:DC:F5:73:BE:1F:DA:A7:07:58:B2
            X509v3 Subject Alternative Name:
                DNS:unamur.be, DNS:www.unamur.be
            X509v3 Key Usage: critical
                Digital Signature, Key Encipherment
            X509v3 Extended Key Usage:
                TLS Web Server Authentication, TLS Web Client Authentication
            X509v3 CRL Distribution Points:

                Full Name:
                  URI:http://crl3.digicert.com/TERENASSLHighAssuranceCA3.crl

                Full Name:
                  URI:http://crl4.digicert.com/TERENASSLHighAssuranceCA3.crl
```

Example

```
 X509v3 Certificate Policies:
                Policy: 2.16.840.1.114412.2.1
                  CPS: https://www.digicert.com/CPS
                Policy: 2.23.140.1.1

            Authority Information Access:
                OCSP - URI:http://ocsp.digicert.com
                CA Issuers -
URI:http://cacerts.digicert.com/TERENASSLHighAssuranceCA3.crt
            X509v3 Basic Constraints: critical
                CA:FALSE
            1.3.6.1.4.1.11129.2.4.2:
                ...i.g.v.......X......gp
......v.V.../.......D.>.Fv....\....U.......q?........G0E.
?w..N.q.....!...8_b..?...^}.C....!...?2..t.4.QJ.[..........A!..@..u.......q...#...{G8W.
.R....d6.......q?..R.....F0D. ..5.m..*.~....(....9..34.v..H....
?.5....S.9..9.B...U^/.VA..ZA.@..
    Signature Algorithm: sha256WithRSAEncryption
         5b:5e:a2:05:4a:fa:84:c8:83:58:94:3f:fe:a8:19:97:4c:3f:
         ca:7d:af:af:6b:e0:27:38:36:14:f9:27:1a:3c:f7:17:24:29:
         ...
         c3:ce:9a:ea:92:a2:81:7d:32:7f:fe:56:8f:1e:ee:77:63:3e:
         74:e8:ce:62:10:50:aa:ad:b6:ad:e6:07:57:58:5e:b7:37:6b:
         84:05:93:9e
```

Structure of a X.509 certificate

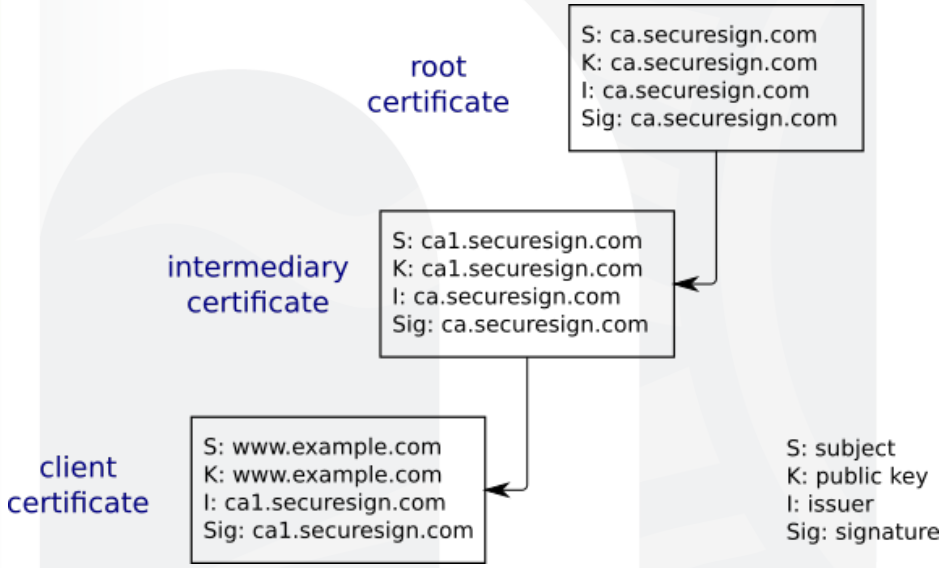
Field	V1	V2	V3
Version	✓	✓	✓
Certificate Serial Number	✓	✓	✓
Signature Algorithm Identifier	✓	✓	✓
Issuer Name	✓	✓	✓
Validity (not before/not after)	✓	✓	✓
Subject Name	✓	✓	✓
Subject Public Key Information	✓	✓	✓
Issuer Unique Identifier		✓	✓
Subject Unique Identifier		✓	✓
Extensions			✓
Signature	✓	✓	✓

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

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
  - Verifier → Prover (subject): $E(K^{+}_{subject}, n)$ where n is a random number (nonce)
  - Prover → 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
  - 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)
  - 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
  - Certificate Directory
  - Key Recovery Server
    - avoid overhead of key/cert creation and distribution in case of lost key

Public Key Infrastructure - PKI

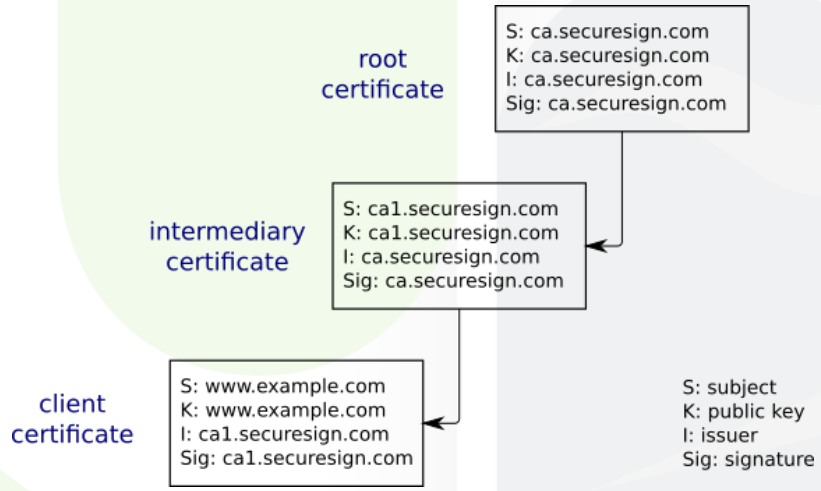
- Certificate revocation
  - because of a compromised or lost key, wrong data in certificate, CA error, disappeared subject...
  - CA maintains a CRL – Certificate Revocation List
  - simple file that contains the list of revoked certs, signed by the CA
  - CRL regularly updated, even if no change, to 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 (or a more generic name that includes 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

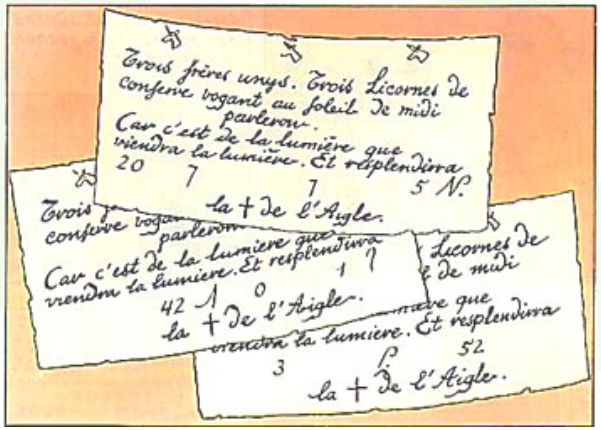
- ANSSI Recommendations
  - Encryption
    - minimal symmetric key size: 100b (< 2020) 128b (> 2020)
    - minimal block size (block cipher): 64b (< 2020) 128b (> 2020)
    - algorithms: DES, 3DES: KO; AES: OK
    - Factorisation (RSA)
      - size of n: 2048b (< 2020) 4096b (> 2020)
      - secret exponent (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

Source: Référentiel Général de Sécurité v2.0. Annexe B1: Mécanismes cryptographiques, ANSSI, 2014

Applications

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⊕R1⊕R2⊕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 ⊕R1⊕R2⊕R3
  - If any of R1, R2, R3, U is lost, p is lost as well (except for T)



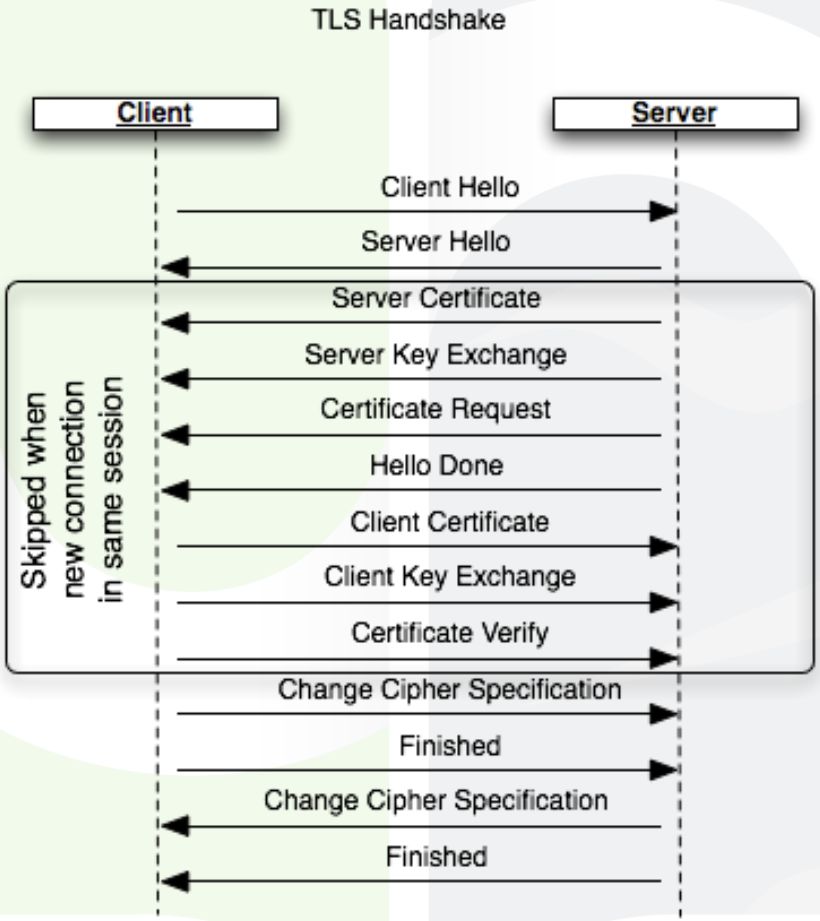
Network security

- 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


TLS Handshake
Client
Server
Client Hello
Server Hello
Server Certificate
Server Key Exchange
Certificate Request
Hello Done
Client Certificate
Client Key Exchange
Certificate Verify
Skipped when new connection in same session
Change Cipher Specification
Finished
Change Cipher Specification
Finished

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

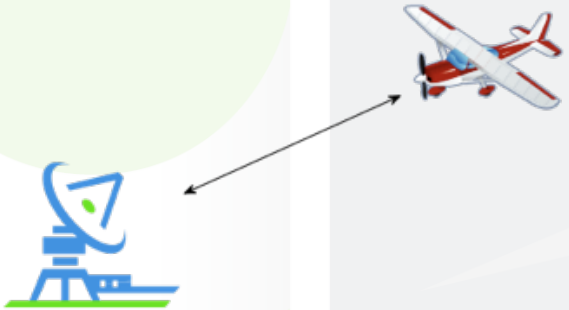
- OpenSSH
  - offers secure versions of old plaintext protocols like ftp, telnet, rsh, rlogin through two tools: ssh and sftp
  - supports strong cryptography (3DES, Blowfish, AES)
  - supports strong authentication, incl. RSA et DSA
  - supports port forwarding (incl. X11)
  - nice to bypass some firewall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

Authentification

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



Authentification

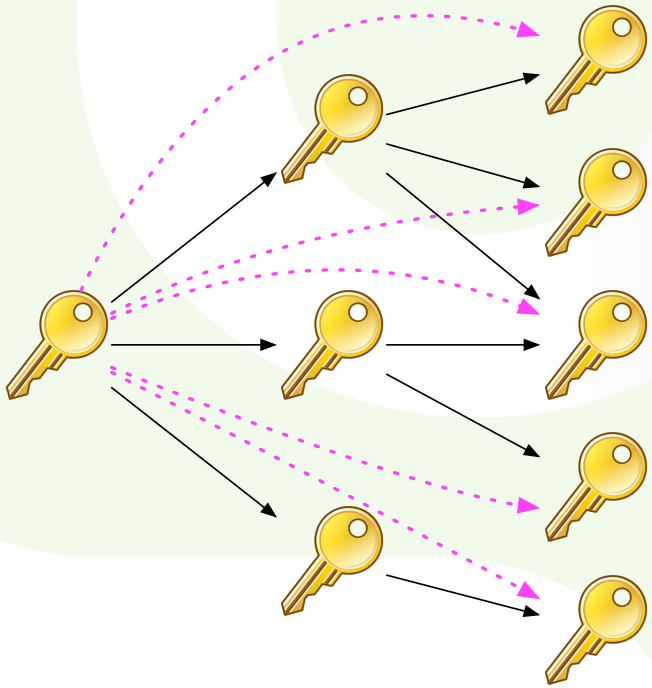
- MITM = “Mig in the middle”
  - Angola vs Namibia war (end of ‘80s)

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 trust(Kj) AND signature(Kj,Ki),THEN trust(Ki)
  - rules can be more elaborated: length of trust chain, number of signatures used to sign the key, different levels of trust



Trust levels
1 = I don't trust or won't say
2 = I do NOT trust
3 = I trust marginally
4 = I trust fully
5 = I trust ultimately

DIRECT TRUST

INDIRECT TRUST

Conclusion

Conclusion

- Cryptography offers strong primitives meeting 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