# Cryptography - Principles –Cryptographie et Sécurité des Communications–

Lionel Morel

Telecommunications - INSA Lyon

Fall-Winter 2022-23

## Context

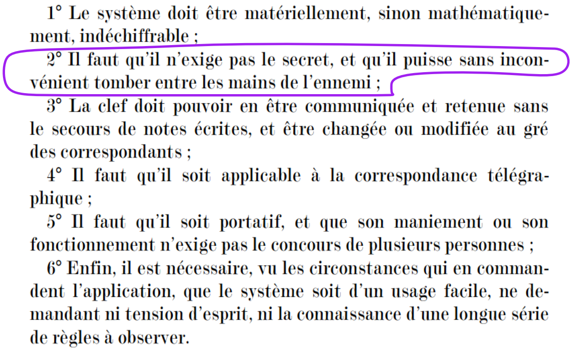
## Previously

- Caesar Cipher
- One-Time Pads
- Enigma
- **Cryptology = Cryptography + Cryptanalysis**

## Today's objectives

- Encryption / Decryption (Confidentiality)
- Verification (Integrity)
- Signature (Authenticity)

## Kerchoffs Principle (in “La Cryptographie Militaire” 1883)



1° Le système doit être matériellement, sinon mathématiquement, indéchiffrable ;

2° Il faut qu'il n'exige pas le secret, et qu'il puisse sans inconvénient tomber entre les mains de l'ennemi ;

3° La clef doit pouvoir en être communiquée et retenue sans le secours de notes écrites, et être changée ou modifiée au gré des correspondants ;

4° Il faut qu'il soit applicable à la correspondance télégraphique ;

5° Il faut qu'il soit portatif, et que son maniement ou son fonctionnement n'exige pas le concours de plusieurs personnes ;

6° Enfin, il est nécessaire, vu les circonstances qui en commandent l'application, que le système soit d'un usage facile, ne demandant ni tension d'esprit, ni la connaissance d'une longue série de règles à observer.

- The adversary knows the system [Shannon]
- $\neq$ Security by Obscurity
- Largely accepted in cryptography
- Can be more widely applied to InfoSec (Information System Security) in general.

## Confusion and Diffusion (Shannon, 1949)

### Confusion

- Each bit in the ciphertext should **depend on several parts of the key**
- Usually implemented using **Substitutions**, aka S-Boxes

### Diffusion

- Encryption/decryptions should imply an **avalanche effet.** Precisely (in the original Shannon description): changing a single bit in the plaintext changes half of the bits in the cipher-text (eg at the block granularity)
- Usually implemented using **Permutations** (P-Boxes)

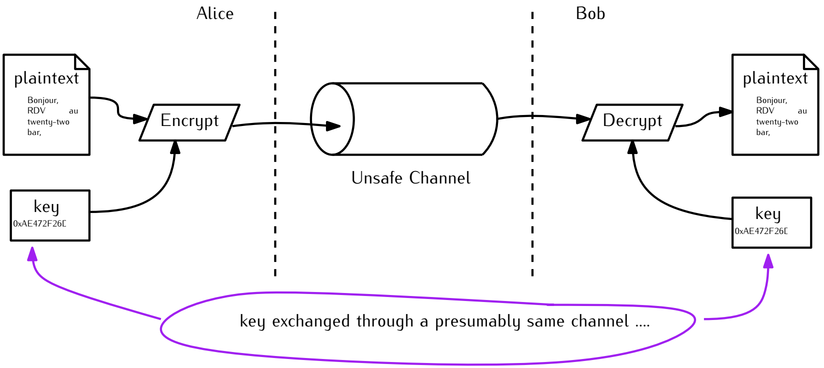
## Precautions

- Use recognized libraries (eg OpenSSL), not your own implementation
- Prefer open-source implementations (easier to identify bugs and backdoors)[1]
- In this class, a lot of simplified versions (same on wikipedia)

[1] https://www.theguardian.com/world/2013/sep/05/nsa-how-to-remain-secure-surveillance

## Symmetric Cryptography

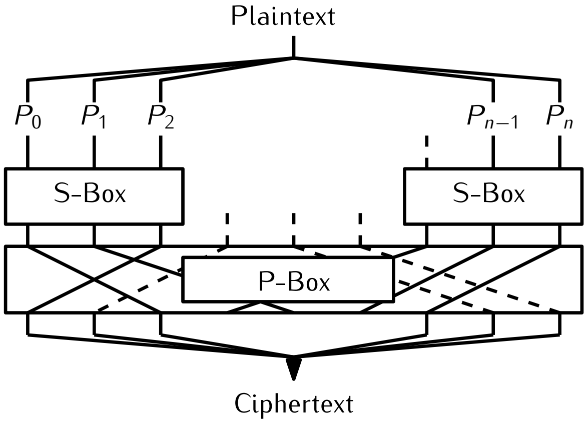
# Symmetric Cryptography - Principles


Alice
Bob
plaintext
Bonjour, RDV au twenty-two bar,
Encrypt
key
0xAE472F26D
Unsafe Channel
Decrypt
plaintext
Bonjour, RDV au twenty-two bar,
key
0xAE472F26D
key exchanged through a presumably same channel ....

- Encryption, Decryption, Signature and Verification use the same key
- Used implementations are quite efficient.
- A key for each pair of communicating entities

⇒ Rapid explosion in the number of keys

## Symmetric ciphers - Basic Principles



Built as a network of substitution/permutation functions:

- Substitution: replace *n* bits by a pre-determined (but moving) table. Must be one-to-one (to allow reversibility of encryption function)
- Permutation: exchange bits

## Symmetric ciphers - Basic Principles

### Block cipher

- ▶ Treat input as fixed-size blocks (between 64 and 128 bits)
- ➕ More secure
- ➖ Requires padding

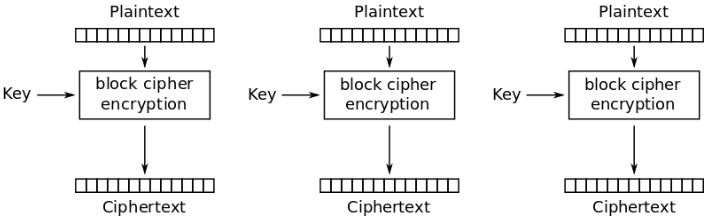
### Stream cipher

- ▶ Treat input one byte at a time
- ▶ The encryption of one byte depends on the current state of the cipher (hence of its history of encryption),
- ➕ fast HW implementation
- ➖ Security less guaranteed

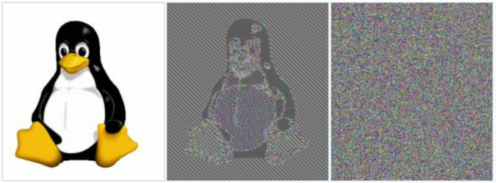
# Symmetric ciphers - Operation Modes

**Electronic Code Book:**

- Message is divided into blocks and each block is encrypted/decrypted separately



- Lacks diffusion

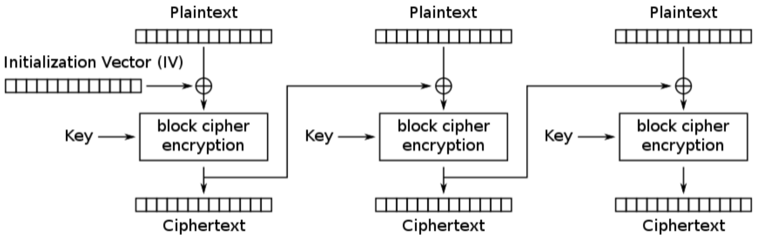


Original image | Encrypted using ECB mode | Modes other than ECB result in pseudo-randomness

## Symmetric ciphers - Operation Modes

**Cipher Block Chaining**

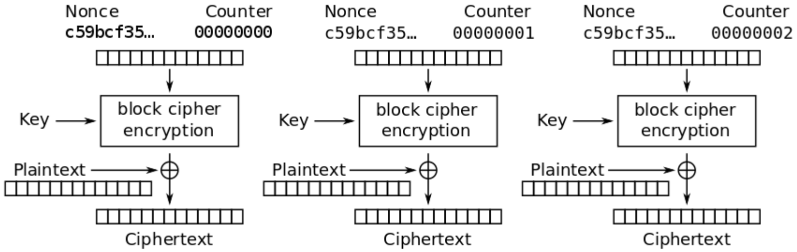
- Initialization Vector to make all cipher message unique



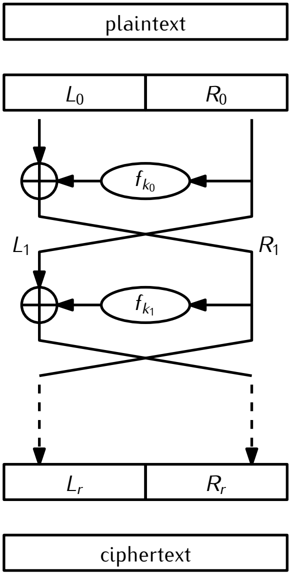
⛔ encryption cannot be parallelized

# Symmetric ciphers - Operation Modes

**CounTeR**



## Feistel

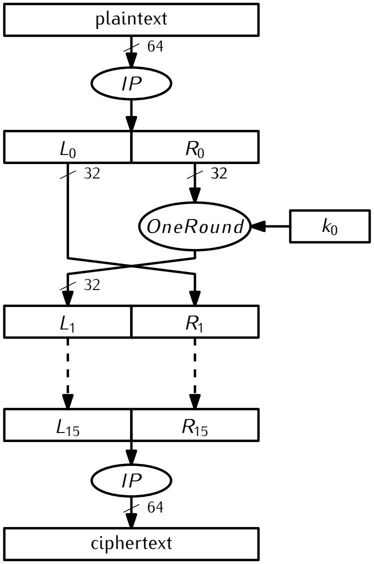


- block cipher
- $r$ rounds
- key $k$ is spilt into $r$ subkeys: $(k_0, \ldots, k_{r-1})$
- plaintext = $(L_0, R_0)$
- $(L_{i+1}, R_{i+1}) = (R_i, L_i \oplus f_{k_i}(R_i))$
- General structure used in all other ciphers

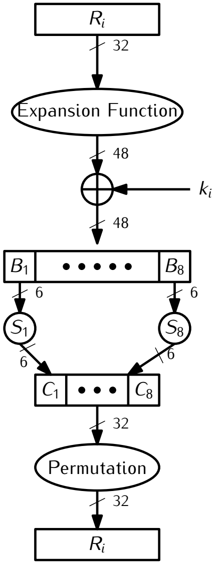
## Symmetric Cryptography - DES

- Expands Feistel algorithm, by introducing:
  - More permutations
  - Substitution Boxes (S-Boxes)
- Designed (and initially published) in 1975.
- Block-cipher

## DES - General Algorithm



## DES - One Round



48-bits subkey obtained through a key-schedule algorithm using the original 64-bits key as input

## DES - Weaknesses and Attacks

- Key size in DES was reduced from 128 bits to 56 bits (after discussions with NSA) “to fit on a single chip”
- Practically cracked (brute-forced) in 1997
- Most practical attack to date: still brute force (ie trying out all possible key in turn).
- Attacks faster than brute-force:
  - Differential cryptanalysis: requires $2^{47}$ chosen plaintexts
  - Linear cryptanalysis: requires $2^{43}$ chosen plaintexts

## Example: Differential Cryptanalysis

**Principle:**

- Choose two plaintexts $x$ and $y$ s.t.[2]:
  $y = x \oplus \Delta_x$
- Compute the corresponding cyphertexts and for each S-Box S:
  - $S(x, k_i)$
  - $S(y, k_i) = S(x \oplus \Delta_x, k_i)$
- Compute difference on S-Boxes:
  - $\Delta_y = S(x \oplus \Delta_x, k_i) \oplus S(x, k_i)$
- Repeat this for many plaintexts and several key hypothesis $k_i, i \in \{0, n\}$
- key $k_j$ that minimizes $\Delta$ is deemed "most probable".

**Limits:**

⛔ In practice requires $2^{47}$ well-chosen plaintext (so that $\Delta_x$ is "not too big")

⛔ Limits: choose the "right" plaintexts

[2] $\oplus$ = "xor"

## Bonus: why $\oplus$ (xor) is "difference"?

| $\oplus$ | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 0 |

Which means $x \oplus y = 1$ iff $x \neq y$

## 3DES

- Standardized in 1998 to compensate for the weaknesses of DES
- DES has a 56-bits key
- 3DES **chains 3 DES together**:
    - Encrypt = Encrypt($k_1$)$\rightarrow$Decrypt($k_2$)$\rightarrow$Encrypt($k_1$)
    - Decrypt = Decrypt($k_1$)$\rightarrow$Encrypt($k_2$)$\rightarrow$Decrypt($k_2$)
    - Key: 112 bits ($k_1|k_2$)
- Developped in parallel of AES (waiting for AES to be defined)

## AES - Advanced Encryption Standard

- Supersedes DES
- Standardized in 2001
- NIST-organized competition with 5 finalists:
  - IBM proposed MARS
  - RSA proposed RC6
  - Serpent by Anderson, Bihman, Knudsen
  - Twofish by Bruce Schneier et al
  - Rijndael, by Daemen and Rijmen
- Rijndael's was elected by community after a thourough international comparative effort (including NSA, companies, academics), based on security, performance (speed, memory usage).
- NB: no-patent allowed (imposed by the NIST)

## AES - Principle[3]

- AES operates on $4 \times 4$ array of 16 bytes, called **the state**

$$\begin{bmatrix} b_0 & b_4 & b_8 & b_{12} \\ b_1 & b_5 & b_9 & b_{13} \\ b_2 & b_6 & b_{10} & b_{14} \\ b_3 & b_7 & b_{11} & b_{15} \end{bmatrix}$$

- Key size specifies the number of transformation rounds to convert input plaintext into output ciphertext:
  - 10 rounds for 128-bit keys
  - 12 rounds for 192-bit keys
  - 14 rounds for 256-bit keys

[3] https://en.wikipedia.org/wiki/Advanced_Encryption_Standard

## AES - Algorithm (for 10 rounds)

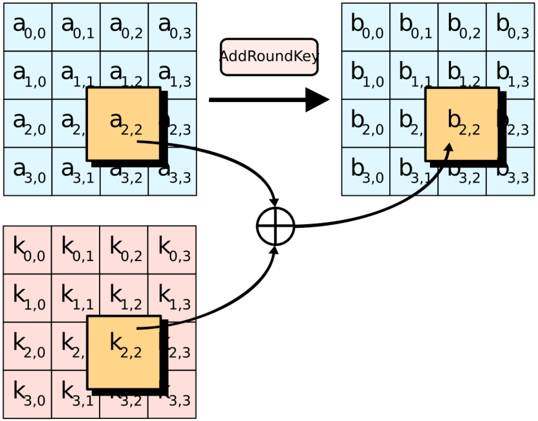
```
void AES_Run_secure(void){
  int i;
  addRoundKey();
  for(i = 0; i < 9; i++){
      subBytes();
      shiftRows();
      mixColumns();
      addRoundKey();
  }
  subBytes();
  shiftRows();
  addRoundKey();
}
```

## AES - Initialization

- **KeyExpansion** — round keys are derived from the cipher key using the AES key schedule. AES requires a separate 128-bit round key block for each round plus one more.
- Initial State = Input plaintext

## AES - Round Key Addition

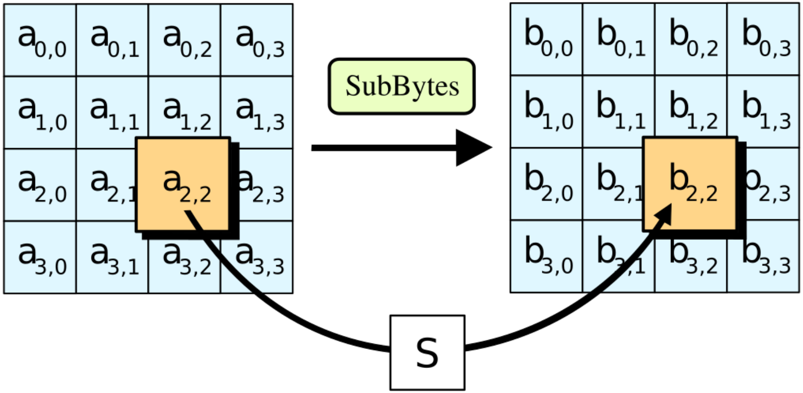
- **AddRoundKey** – each byte of the state is combined with a byte of the round key using bitwise xor.



$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
AddRoundKey
$b_{0,0}$ $b_{0,1}$ $b_{0,2}$ $b_{0,3}$
$b_{1,0}$ $b_{1,1}$ $b_{1,2}$ $b_{1,3}$
$b_{2,0}$ $b_{2,1}$ $b_{2,2}$ $b_{2,3}$
$b_{3,0}$ $b_{3,1}$ $b_{3,2}$ $b_{3,3}$
$k_{0,0}$ $k_{0,1}$ $k_{0,2}$ $k_{0,3}$
$k_{1,0}$ $k_{1,1}$ $k_{1,2}$ $k_{1,3}$
$k_{2,0}$ $k_{2,1}$ $k_{2,2}$ $k_{2,3}$
$k_{3,0}$ $k_{3,1}$ $k_{3,2}$ $k_{3,3}$
$\oplus$

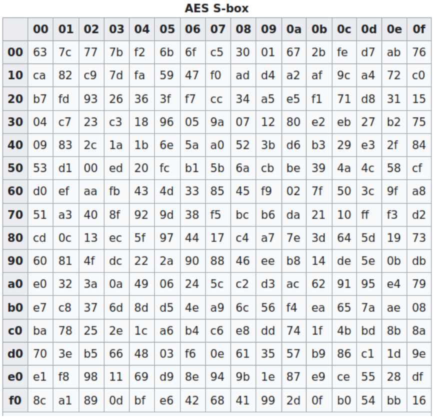
## AES - SubBytes

- SubBytes = a non-linear substitution step where each byte is replaced with another according to a lookup table.
- lookup table = S-box


$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
SubBytes
$b_{0,0}$ $b_{0,1}$ $b_{0,2}$ $b_{0,3}$
$b_{1,0}$ $b_{1,1}$ $b_{1,2}$ $b_{1,3}$
$b_{2,0}$ $b_{2,1}$ $b_{2,2}$ $b_{2,3}$
$b_{3,0}$ $b_{3,1}$ $b_{3,2}$ $b_{3,3}$
S

## AES - SubBytes

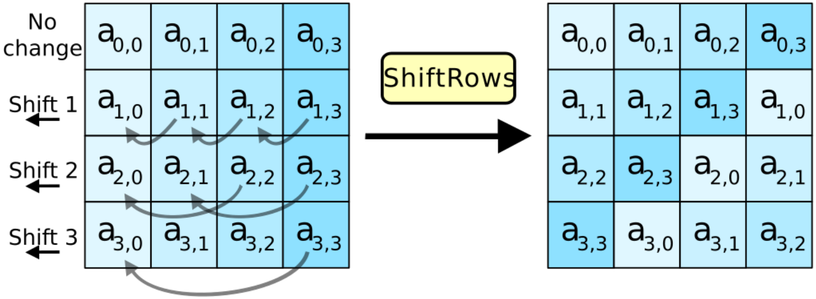
- SubBytes = a non-linear substitution step where each byte is replaced with another according to a lookup table.
- lookup table = S-box


**AES S-box**

| | **00** | **01** | **02** | **03** | **04** | **05** | **06** | **07** | **08** | **09** | **0a** | **0b** | **0c** | **0d** | **0e** | **0f** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **00** | 63 | 7c | 77 | 7b | f2 | 6b | 6f | c5 | 30 | 01 | 67 | 2b | fe | d7 | ab | 76 |
| **10** | ca | 82 | c9 | 7d | fa | 59 | 47 | f0 | ad | d4 | a2 | af | 9c | a4 | 72 | c0 |
| **20** | b7 | fd | 93 | 26 | 36 | 3f | f7 | cc | 34 | a5 | e5 | f1 | 71 | d8 | 31 | 15 |
| **30** | 04 | c7 | 23 | c3 | 18 | 96 | 05 | 9a | 07 | 12 | 80 | e2 | eb | 27 | b2 | 75 |
| **40** | 09 | 83 | 2c | 1a | 1b | 6e | 5a | a0 | 52 | 3b | d6 | b3 | 29 | e3 | 2f | 84 |
| **50** | 53 | d1 | 00 | ed | 20 | fc | b1 | 5b | 6a | cb | be | 39 | 4a | 4c | 58 | cf |
| **60** | d0 | ef | aa | fb | 43 | 4d | 33 | 85 | 45 | f9 | 02 | 7f | 50 | 3c | 9f | a8 |
| **70** | 51 | a3 | 40 | 8f | 92 | 9d | 38 | f5 | bc | b6 | da | 21 | 10 | ff | f3 | d2 |
| **80** | cd | 0c | 13 | ec | 5f | 97 | 44 | 17 | c4 | a7 | 7e | 3d | 64 | 5d | 19 | 73 |
| **90** | 60 | 81 | 4f | dc | 22 | 2a | 90 | 88 | 46 | ee | b8 | 14 | de | 5e | 0b | db |
| **a0** | e0 | 32 | 3a | 0a | 49 | 06 | 24 | 5c | c2 | d3 | ac | 62 | 91 | 95 | e4 | 79 |
| **b0** | e7 | c8 | 37 | 6d | 8d | d5 | 4e | a9 | 6c | 56 | f4 | ea | 65 | 7a | ae | 08 |
| **c0** | ba | 78 | 25 | 2e | 1c | a6 | b4 | c6 | e8 | dd | 74 | 1f | 4b | bd | 8b | 8a |
| **d0** | 70 | 3e | b5 | 66 | 48 | 03 | f6 | 0e | 61 | 35 | 57 | b9 | 86 | c1 | 1d | 9e |
| **e0** | e1 | f8 | 98 | 11 | 69 | d9 | 8e | 94 | 9b | 1e | 87 | e9 | ce | 55 | 28 | df |
| **f0** | 8c | a1 | 89 | 0d | bf | e6 | 42 | 68 | 41 | 99 | 2d | 0f | b0 | 54 | bb | 16 |

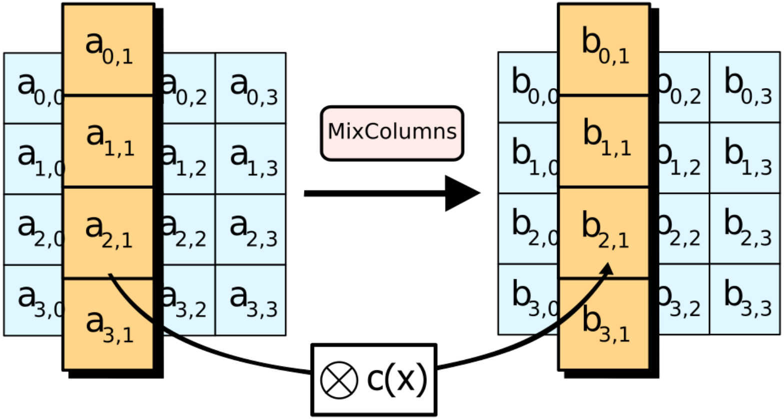
## AES - ShiftRows

ShiftRows = a transposition step where the last three rows of the state are shifted cyclically a certain number of steps.


No change
Shift 1
Shift 2
Shift 3
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
ShiftRows
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,1}$ $a_{1,2}$ $a_{1,3}$ $a_{1,0}$
$a_{2,2}$ $a_{2,3}$ $a_{2,0}$ $a_{2,1}$
$a_{3,3}$ $a_{3,0}$ $a_{3,1}$ $a_{3,2}$

# AES - MixColumns

- MixColumns = a linear mixing operation which operates on the columns of the state, combining the four bytes in each column. AddRoundKey
- Together with ShiftRows, MixColumns provides diffusion in the cipher.



$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
MixColumns
$\otimes c(x)$
$b_{0,0}$ $b_{0,1}$ $b_{0,2}$ $b_{0,3}$
$b_{1,0}$ $b_{1,1}$ $b_{1,2}$ $b_{1,3}$
$b_{2,0}$ $b_{2,1}$ $b_{2,2}$ $b_{2,3}$
$b_{3,0}$ $b_{3,1}$ $b_{3,2}$ $b_{3,3}$

## AES - One Round

One Round ==

```
subBytes();
shiftRows();
mixColumns();
addRoundKey();
```

- Repeat 9, 11 or 13 rounds
- Plus an extra one without the MixColumns

## AES - Weaknesses and Attacks

### Related-key attacks exists

- $2^{99.5}$ time and space complexity
- btw: age of universe ˜ $2^{70}$
- Anyway totally impractical (because keys are well-chosen to be independant in crypto-systems)

### Side-channel attacks are practical

- 6-7 blocks plaintexts needed

$\Rightarrow$ requires HW protections

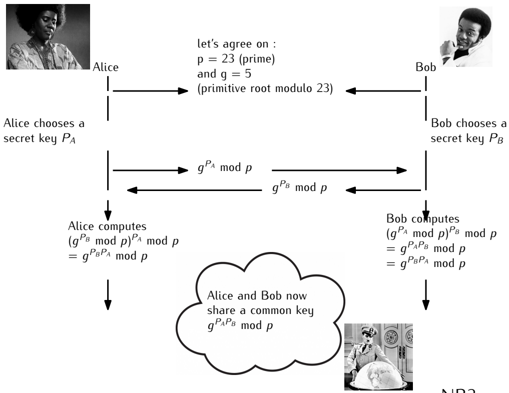
## Symmetric Cryptogaphy - Conclusions

- ➕ Overall very effecient (linear in the size of data to encrypt)
- ➕ Arithmetic/Logical operations are simple: xor.
- ➖ Requires a shared key!
- Solutions to this:
  - Avoid the need for a common key
  - Find a way to securely share a common key

## Key Sharing Problem

- Symmetric cryptography uses same key to encrypt and decrypt
- Problem: how to share this key
- Hypothesis: there is no secure channel to exchange the key

# Diffie-Hellman Key Exchange



Alice

Bob

let's agree on :
p = 23 (prime)
and g = 5
(primitive root modulo 23)

Alice chooses a secret key $P_A$

Bob chooses a secret key $P_B$

$g^{P_A}$ mod $p$

$g^{P_B}$ mod $p$

Alice computes
$(g^{P_B} \bmod p)^{P_A} \bmod p$
$= g^{P_B P_A} \bmod p$

Bob computes
$(g^{P_A} \bmod p)^{P_B} \bmod p$
$= g^{P_A P_B} \bmod p$
$= g^{P_B P_A} \bmod p$

Alice and Bob now share a common key $g^{P_A P_B}$ mod $p$

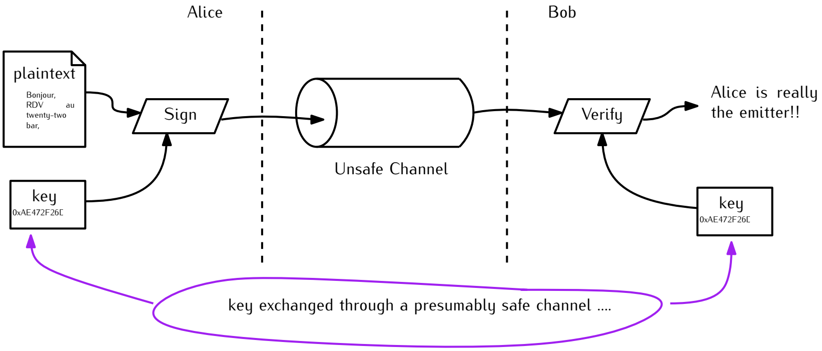
NB1: $g^k mod n$ is the **modular exponentiation** and can be computed quiet efficiently....

NB2: $g$ is a primitive root modulo $n$ if $\forall a$ (integer) coprime to $n$, $\exists k$ for which $g^k \equiv a (mod n)$.

NB3: The strength of the scheme comes from the fact that $g^{P_A P_B} mod p = g^{P_B P_A} mod p$ take extremely long times to compute by any known algorithm just from the knowledge of $p$, $g$, $g^{P_A} mod p$, and $g^{P_B} mod p$.

## Hash

## Cryptographic Hash


Alice
Bob
plaintext
Bonjour, RDV au twenty-two bar,
Sign
key
0xAE472F26C
Unsafe Channel
Verify
Alice is really the emitter!!
key
0xAE472F26C
key exchanged through a presumably safe channel ....

- eg Hash-based Message Authentication Code
- Only sender and recipient can sign/verify the message

## Cryptographic Hash - Principle

- Compute a “footprint”
- The message can be of any size, the footprint is of fixed size
- Pseudo-unique identification of message
- Used for:
  - Integrity checks
  - Cryptographic signature
  - PRNG
  - Hashed password storage

## Cryptographic Hash - Good Properties

- **Pre-image resistance**: no one can reverse the hash function (to find input from output)
- **Second pre-image resistance**: unicity of hash. Given an input and the corresponding hash, one cannot find another input with the same hash.
- **Collision-resistance**: no-one can produce two different inputs with the same hash
- **Randomness**

## Cryptographic Hash - today’ state of affairs

### Existing (and used) implementations

- MD5: please don’t use anymore: “cryptographically broken and unsuitable for further use”
- SHA-1: not recommanded anymore (since 2017)
- SHA-2: still not planned for removal
- SHA-3: standardized in 2015

### Current situation

- Hash functions are critical in crypto!
- SHA-2 is still safe but is conceptually close to SHA-1 and might share some weaknesses with it
- SHA-3 considered “as safe” but built completely differently

## Asymmetric Cryptography

## (general) Asymmetric Cryptography

- Each participant $\boldsymbol{u}$ has a pair of keys $(\boldsymbol{Pub_u}, \boldsymbol{Priv_u})$.
- $\boldsymbol{u}$ sends $\boldsymbol{Pub_u}$ to $\boldsymbol{v}$
- $\boldsymbol{v}$ sends $\boldsymbol{Pub_v}$ to $\boldsymbol{u}$
- $\boldsymbol{u}$ can encrypt its messages to $\boldsymbol{v}$ using a combination of $\boldsymbol{Pub_v}$ and $\boldsymbol{Priv_u}$
- $\boldsymbol{v}$ can decrypt messages from $\boldsymbol{u}$ using a combination of $\boldsymbol{Pub_u}$ and $\boldsymbol{Priv_v}$

Note:

- Relies on “hard mathematical problems”:
  - Discrete logarithm
  - Factorization of large numbers
- Usually slow (exponentiation)

## RSA

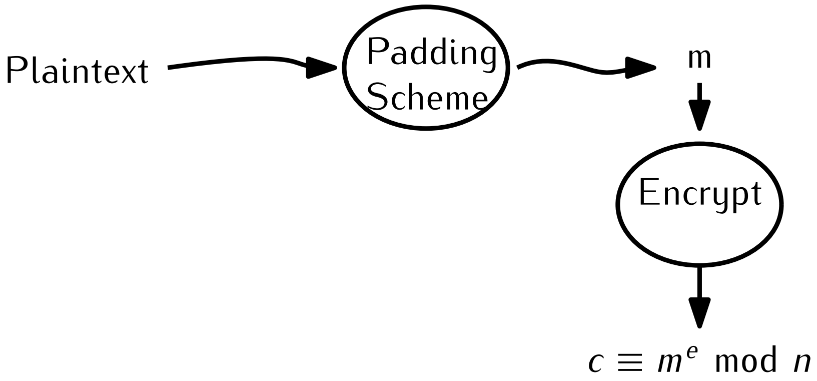
- Invented in 1977 by Rivest, Shamir and Adleman
- MIT Patent in 1983, expired in 2000
- Security based on the difficulty of factorizing large integers

## RSA - Key generation

- Choose $\boldsymbol{p}$ and $\boldsymbol{q}$, two prime numbers: random, kept secret
- Compute $\boldsymbol{n} = \boldsymbol{pq}$
- Compute $\lambda(\boldsymbol{n})$,
  - $\lambda(\boldsymbol{n}) = \boldsymbol{lcm}(\lambda(\boldsymbol{p}), \lambda(\boldsymbol{q}))$
  - $= \boldsymbol{lcm}(\boldsymbol{p} - 1, \boldsymbol{q} - 1)$
  - $= \frac{\boldsymbol{pq}}{\boldsymbol{gcd}(\boldsymbol{p},\boldsymbol{q})}$ ... (gcd obtained with Euclid algorithm)
- Choose $\boldsymbol{e}$ s.t.:
  - $1 < \boldsymbol{e} < \lambda(\boldsymbol{n})$
  - $\boldsymbol{gdc}(\boldsymbol{e}, \lambda(\boldsymbol{n})) = 1$
- Compute $\boldsymbol{d} = \boldsymbol{e}^{-1} \textbf{ mod } \lambda(\boldsymbol{n})$
  - $\boldsymbol{d}$ is the “private key exponent”

$\boldsymbol{Pub} = (\boldsymbol{e}, \boldsymbol{n})$ $\boldsymbol{Priv} = (\boldsymbol{d}, \boldsymbol{n})$

## RSA - Encryption



## RSA - Decryption

$$\begin{tikzcd}
c \arrow[r, squiggly] & \boxed{\text{Decrypt}} \arrow[r, squiggly] & m \equiv c^d \bmod n
\end{tikzcd}$$

## RSA - Example

1. $p = 61$ and $q = 53$
2. n = pq = 3233
3. $\lambda(n) = lcm(p - 1, q - 1)$
4. $= \lambda(3233) = lcm(60, 52) = 780$
5. Choose $1 < e < 780$ (coprime to 780), eg $e = 17$
6. $d = e^{-1}$ mod $\lambda(n)$
7. $= 413$ (as $1 = 17 * 413$ mod 780)
8. $\boxed{\text{Public key = } (e = 17, n = 3233)}$
9. $\boxed{\text{Private key = } (d = 413, m = 3233)}$
10. $c(m) = m^{17}$ mod 3233
11. $m(c) = c^{413} mod 3233$
12. $m = 65 \rightarrow c = 65^{17}$ mod $3233 = 2790$
13. $2790 \rightarrow m = 2790^{413}$ mod $3233 = 65$

## RSA - Properties & Limitations

- ➕ Finding $d$ requires factorizing $n$ (if finding $p$ and $q$ s.t. $n = p * q$: proven difficult (for $p$ and $q$ large)
- ➖ Implementation is tricky : good PRNG, acceptable $e$
- ➖ Relies on exponentiation which is **expensive** :

$$x^y = \underbrace{x * x * \ldots * x}_{y \text{ times}}$$

  - Requires a (fast) multiplier
  - $y$ is big (if you want security)
  - Way more expensive than xor !

## Key management

### The key distribution problem

- To encrypt a message or check a signature, Alice needs Bob's public key
- Otherwise, it may encrypt a message thinking only Bob will read it, but maybe Charlie can read it instead
- How can she get this public key in a secure manner?
- Hard problem, no perfect solution

Note: Using the right key guarantees Bob **is** Bob, but not that Bob is honnest ...

### Existing solutions

- Hierarchical certification authorities
- Web of trust (eg PGP)
- Direct exchange of keys

## Hybrid Cryptography

# Comparing Symmetric / Asymmetric cryptogaphy

## Symmetric cryptography

- 1 key per pair of participants ($n^2$ keys)
- Fast: simple operations, easy to implement in HW

## Asymmetric cryptography

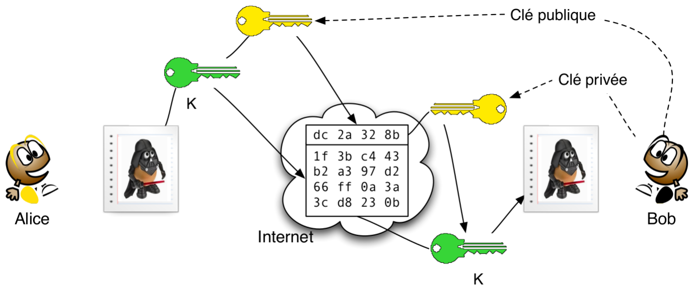
- 1 pair of key per participant ($2n$ keys)
- Slow: complex operations, eg exponentiations

## Hybrid cryptography

- Alice encrypts a symmetric key with the public key of Bob
- Alice encrypts the message with the symmetric key

$\Rightarrow$ Best of both worlds

## The “best of both worlds”



- Alice encrypts message with Symm key $k$
- Alice encrypts $k$ with Bob’s public key
- Bob decrypts $k$ with his private key
- Bob decrypts message with $k$

## Next time

- Cryptographic protocols
- Public Key Authorities
- PGP