



# CRYPTOGRAPHIE

3 - ctearrpogorpe.ry.ghf



# VOCABULAIRE

Pour ne pas dire "crypter"

## Message clair / Plaintext

Des données quelconques, lisibles par tous.

## Message chiffré / Cryptogramme / Ciphertext

Un message clair qui a été obfusqué pour le rendre illisible par une tierce partie - tout en restant lisible par les parties de confiance.

## Clé / Secret / Key

Une donnée secrète partagée par les parties de confiance. Nous verrons qu'il existe aussi des clés publiques, qui ne sont donc pas secrètes.

## Chiffre / Cipher

Une méthode d'obtention d'un cryptogramme à partir d'un message clair, en utilisant une clé.

## Chiffrer

Message clair -> Cryptogramme

## Déchiffrer

Cryptogramme -> Message clair



Dire "Crypter" ou "Encrypter" n'est pas français ! <https://chiffrer.info/>





# BASE64 – PAS UN CHIFFRE

... et sa cousine la base32

- Représentation de données non-affichables (ASCII)
- Plus efficace que l'hexa
- Très utilisé dans le web
- Reconnaissable par les = de padding à la fin

## **Plaintext:**

tiens voila mon zob

## **Hex:**

74ab656e7320766f696c61206d6f6e207a6f62

## **Base64:**

dGllbnMgdm9pbGEgbW/9uIHpvYg==



Pas besoin de clé pour retrouver le message



# XOR – UN CHIFFRE

eXclusive OR


## Opération binaire (bit par bit)

- **Très** utilisé en cryptographie

En particulier, quels que soient **P**lain, **K**ey et **C**iphertext (strings, nombres, bits):

- Pour chiffrer  **$P \text{ xor } K = C$**
- Et déchiffrer  **$C \text{ xor } K = P$**
-  Mais aussi  **$C \text{ xor } P = K$**  !

	0	1
0	0	1
1	1	0

 FAIBLE !!! On peut retrouver la clé si on connaît le plaintext



# HASHES – PAS UN CHIFFRE

... Mais importants quand même

## Propriétés d'un hash:

- **Représentation** d'une donnée selon un certain algorithme
- La **même donnée** doit toujours donner le **même hash**
- Modifier **un octet** doit *généralement* **entièrement** modifier le hash
- **Résistance pré-image**  
Il doit être très difficile de retrouver la donnée originale à partir du hash
- **Résistance collisions**  
Il doit être très difficile de trouver deux données qui donnent le même hash

```
~|⇒ echo -ne "zob" | md5sum  
36f3b2748cea3f5714d849889bb4a0c7 -  
~|⇒ echo -ne "zab" | md5sum  
46f6e8fb2d9082a6bffa9070b7aee619c -
```





# HASHS CONNUS

Hashashins Creed

## SHA

Une famille de hash **cryptographiques** reconnue. SHA-256 est le "standard" actuellement.

## MD5

Un hash très utilisé pour les fichiers car très rapide, mais pas approprié pour une utilisation cryptographique.

Même pour MD5, généralement, la meilleure façon de craquer est une attaque par dictionnaire, où le hash est comparé à des hashes connus. Sinon, il faut essayer tous les inputs manuellement (brute-force) jusqu'à retrouver le bon hash.



Un bon hash cryptographique est un hash **lent** à calculer





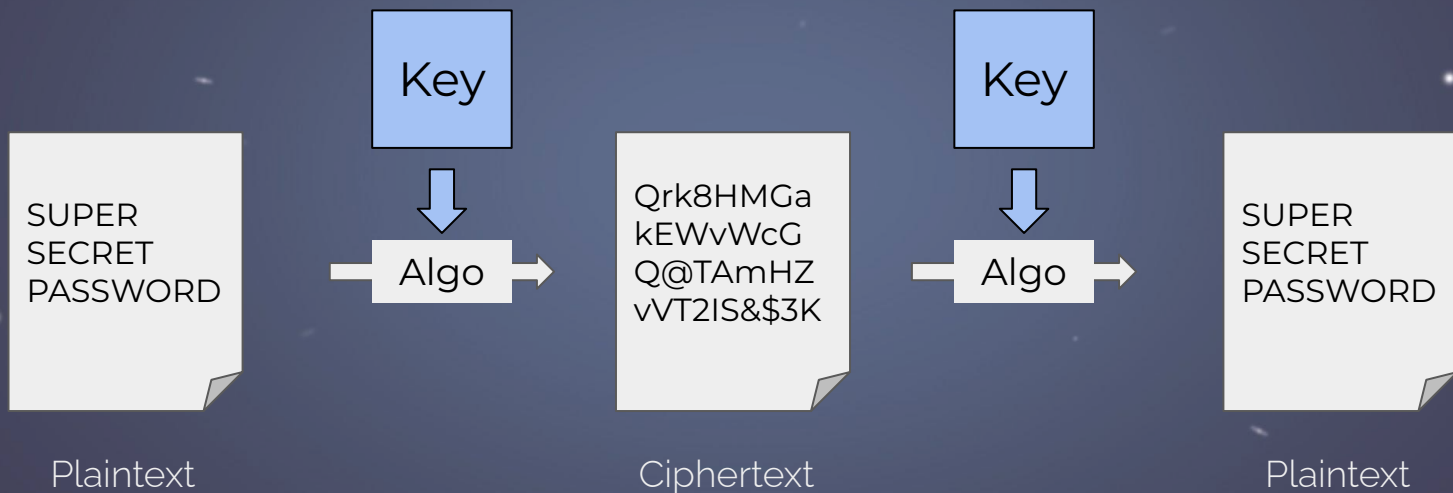
# CHIFFREMENT SYMÉTRIQUE

Une clé pour les gouverner tous



# L'ENVOI D'UN MESSAGE CHIFFRÉ

Pour envoyer ses dick pics tranquille







# L'ENVOI D'UN MESSAGE CHIFFRÉ

Pour envoyer ses dick pics tranquille

Et pour échanger les clés alors ? 🤔



---

# LES PREMIERS CHIFFRES

OLD SCHOOL COOL



# RAIL FENCE

Pas mal pour le jardinage aussi

WE ARE  
DISCOVERED,  
FLEE AT ONCE



W	.	.	.	E	.	.	.	C	.	.	.	R	.	.	.	L	.	.	.	T	.	.	E	
.	E	.	R	.	D	.	S	.	O	.	E	.	E	.	F	.	E	.	A	.	O	.	C	.
.	.	A	.	.	.	I	.	.	.	V	.	.	.	D	.	.	.	E	.	.	.	N	.	.



WECRLTEERDSOEFE  
AOCAVDEN

## Chiffrement par transposition (réorganisation)

- La clé est **un nombre**, le nombre de lignes sur lesquelles on écrit le message (ici, 3).
- Pas très utilisé historiquement.
- Basique car il ne fait que réorganiser les lettres et ne les modifie pas.



Une Scytale



# SUBSTITUTION

Put me in, coach

## Table de substitution

La clé ici est une table de substitution qui fait correspondre chaque caractère à un autre caractère. Par exemple, A devient H, X devient J, etc.

Assez utilisé historiquement, rendu obsolète par les **analyses de fréquence** sur du texte assez large.

### Plaintext:

upon this basis i am going to show you how a bunxh of bright young folks did find a xhampion a man with boys and girls of his own.

### Key:

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

### Cryptogram:

pkji oexn agnxn x gh tjxit oj nejr vjp ejr g apide jy amx teo vjpit yjfcn sxs yxis g degkxji g hgi rxoe ajvn gis txmfn jy exn jri.





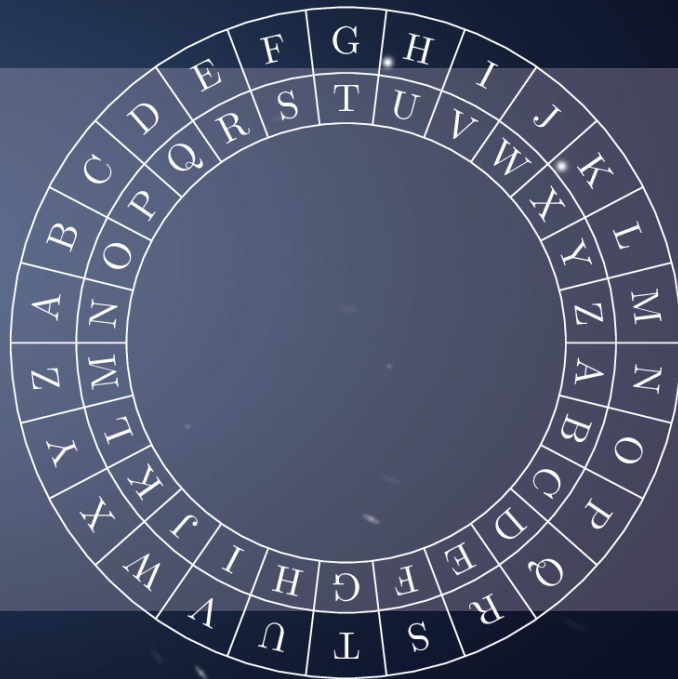
# LE CHIFFRE DE CÉSAR

L'empire contre-attaque

## Chiffrement par décalage (monoalphabétique)

- Une des plus anciennes méthodes
- Chaque lettre est décalée de N lettres dans l'alphabet
- Indéchiffrable pour ces illettrés de barbares
- Aujourd'hui on appelle ça **rot13** (car rot1, rot2...)

La clé ici est **un nombre** entre 1 et 25, on décale de ce nombre dans l'alphabet.





# CHIFFRE DE VIGENÈRE

Un César pour les bonhommes

## Chiffrement par décalage (polyalphabétique)

- Même principe que César mais le nombre de décalage est **variable**
- Décalage donné par une lettre de l'alphabet, A=0, B=1 etc
- Si la clé est trop courte il y a des répétitions (1 = César)

La clé ici est **une phrase** dont chaque lettre décrit une valeur de décalage.

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



# FIN DU MATIN JOUR 1

Go do the project (it's on the first slide)



# CRYPTO MODERNE

Shiny new things





# EN INFORMATIQUE MODERNE

Comment utiliser la clé ?



## Chiffrement de flux

Les données sont chiffrées en continu, octet par octet (par exemple), par la clé.



## Chiffrement par bloc

On divise les données en blocs de  $n$  caractères et on les chiffre bloc par bloc.





---

# Chiffrement par bloc

Block cipher? I hardly knew 'er



# UN CHIFFRE HISTORIQUE SOUS STÉROÏDES

Rien de nouveau sous le soleil

## Le chiffrement par bloc est:

- **Symétrique**, il n'y a donc qu'une clé
- Très, très **rapide**
- Très **employé** aujourd'hui.
- Divisible en des **étapes logiques**, comme un chiffre historique.
- On doit appliquer du padding à la fin d'un input qui ne rentre pas tout pile dans les blocs... Un peu comme un Rail Fence!

Par exemple, des permutations de lignes, de colonnes, de caractères...





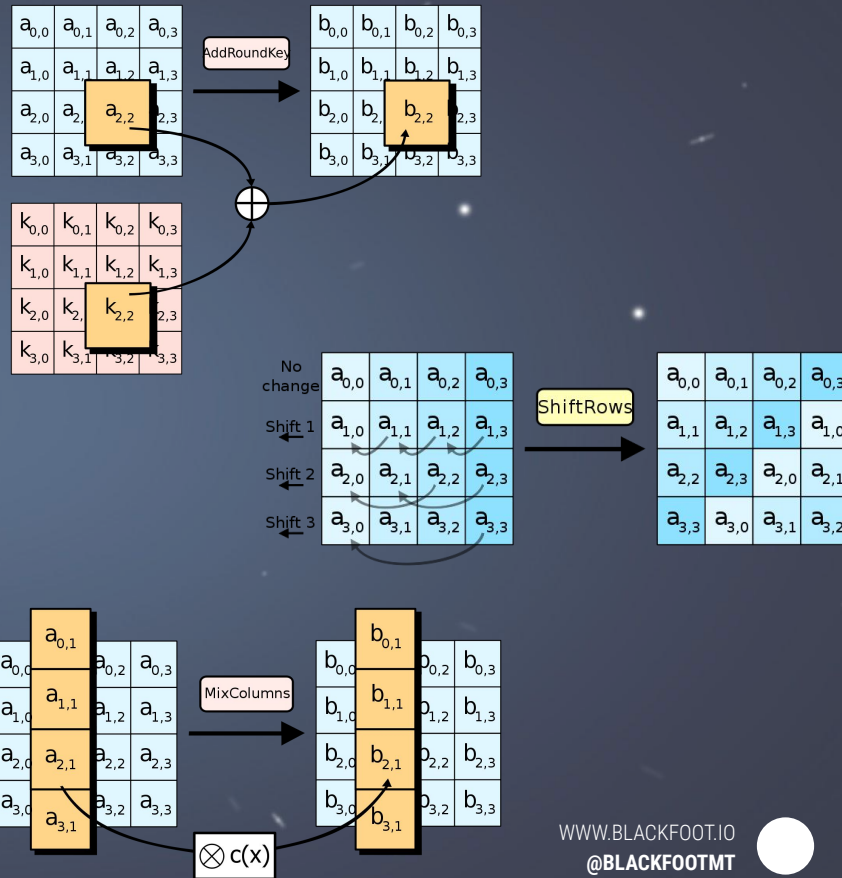
# AES – ADVANCED ENCRYPTION STANDARD

Comment ça marche?

Plusieurs opérations répétées plusieurs fois:

- XOR de la clé et du bloc (*AddRoundKey*)
- Substitution des octets du bloc d'après une table de substitution (*SubBytes*)
- Décalage de rangées (*ShiftRows*)
- Mélange de colonnes (*MixCols*)

Différents algos selon la **taille de la clé**. Le plus utilisé est AES-256.





---

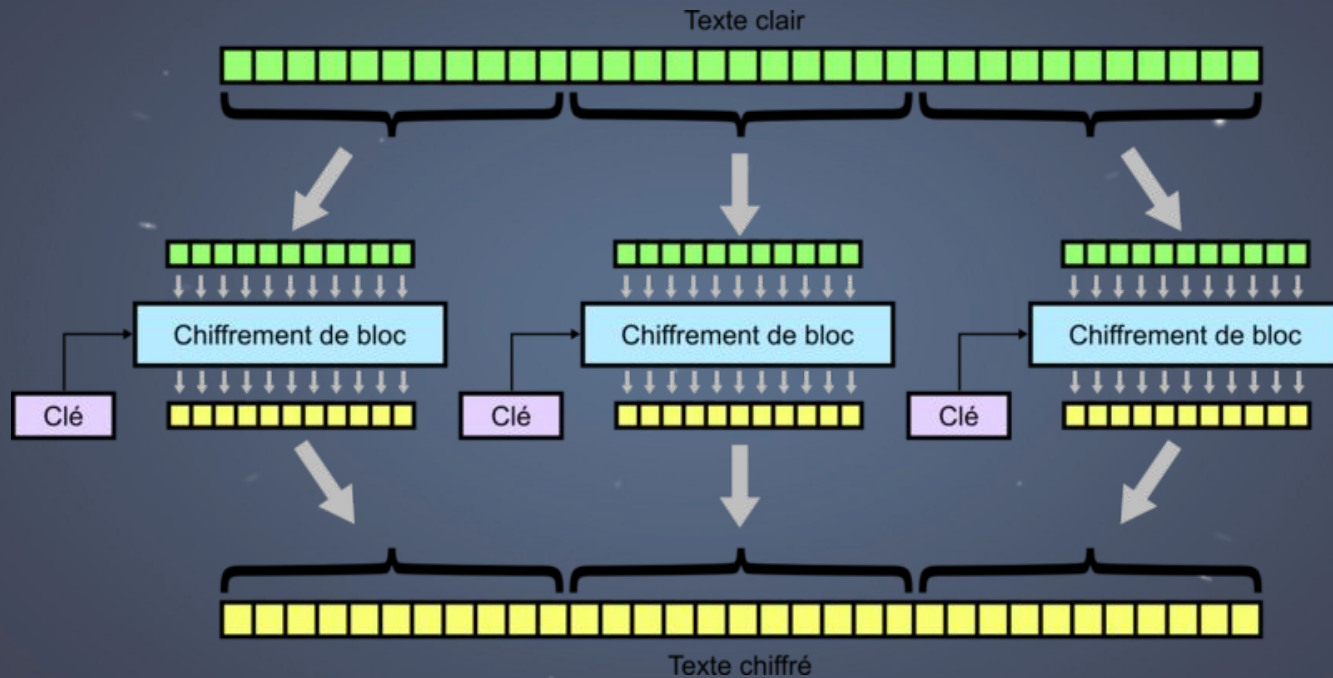
# Les modes d'opération

Tu pensais pas que les blocs c'était simple quand même?



# ECB – ELECTRONIC CODEBOOK

La base





# MAIS...

C'est limitant, ECB!

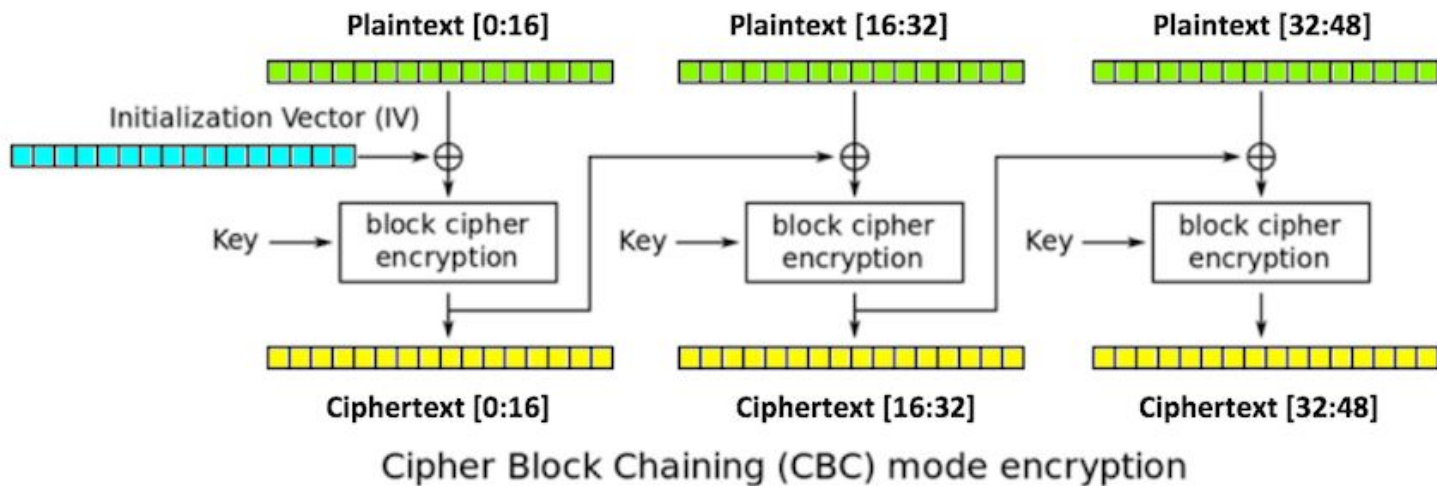
- **Pourquoi chiffrer** les blocs indépendamment avec la même clé ?
- Si on peut déchiffrer un bloc, alors on peut déchiffrer tous les blocs !
- On peut aussi **discerner des patterns** si le message clair est bien plus gros que la clé
- Une des solutions est de **générer** une "clé de round" : une clé unique par bloc
- Mais pourquoi s'arrêter là quand on peut faire encore **mieux** ?
- Quand on peut... Utiliser chaque bloc pour **chiffrer le bloc suivant** ?





# CBC- CIPHER BLOCK CHAINING

Shit gets real





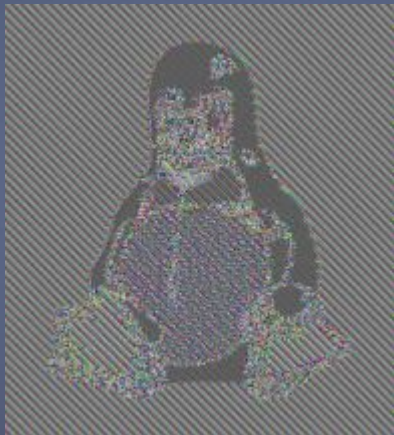


# LES EFFETS DE CBC

Les docteurs le détestent!



Plaintext



Plaintext -> ECB



Plaintext -> CBC





# ET CBC, C'EST PARFAIT ?

lol



## Plus lent

CBC empêche le parallélisme et ralentit le chiffrement par bloc. Il reste tout de même très rapide.



## HACKERZ!!!!@!

CBC a une faille en particulier, connue et exploitable si un attaquant peut chiffrer les données de façon répétitive :

**l'Oracle Padding Attack.**

D'autres approches comme le Bit Flipping existent.



## ...et l'échange de clés alors?

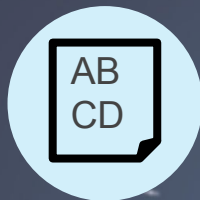
Ah ben oui, avec tout ça on ne sait toujours pas comment on va échanger nos clés, nous...





# L'ENVOI DE LA CLÉ

C'est problématique



## ENVOI EN PHYSIQUE

- En personne ou par courrier
- Problèmes de logistique
- Problèmes de sécurité évidents

ex: Enigma



## CHIFFRER LA CLÉ

- Comment?
- Et surtout, comment on envoie la clé de la clé? Même problème.
- Besoin d'une autre méthode

ex: Utiliser RSA



## LES MATHS

- Génération de clé unique
- Communications publiques
- La clé finale est secrète

ex: Diffie-Hellman



# DIFFIE-HELLMAN

Échange de clé public et sécurisé

## Modulo + Multiplication

Les deux partis (Alice et Bob) se mettent d'accord sur deux nombres arbitraires  $g$  et  $p$ , puis génèrent chacun de leur côté un nombre secret ( $a$  et  $b$ ) et l'utilisent pour générer un nombre public ( $A$  et  $B$ ):

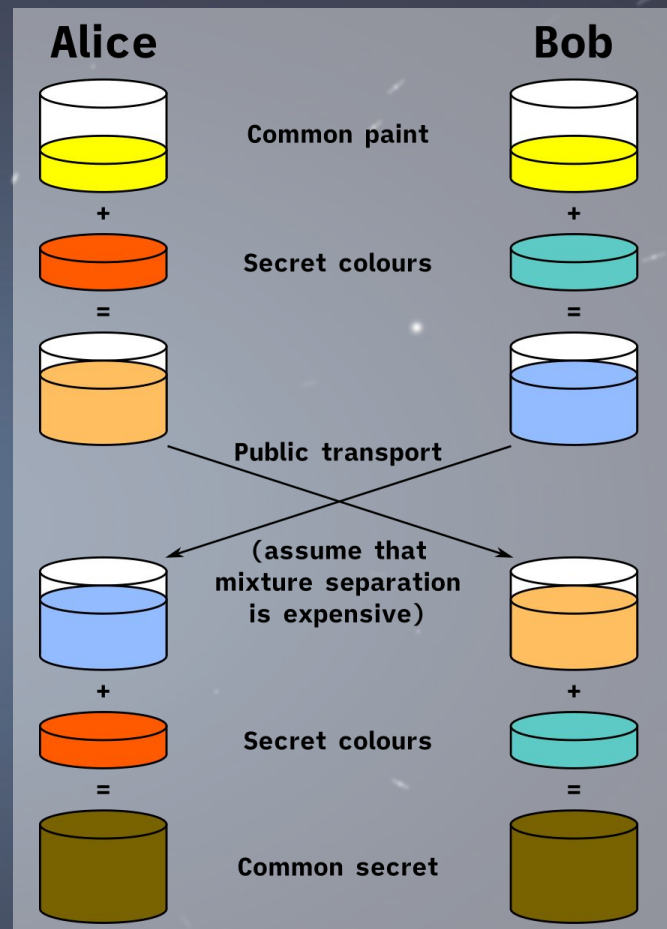
$$A = g^a \pmod{p} \quad B = g^b \pmod{p}$$

Alice envoie  $A$  à Bob, et Bob envoie  $B$  à Alice, et chacun génère la clé en faisant:

$$\text{Pour Alice: } S = B^a \pmod{p}$$

$$\text{Pour Bob: } S = A^b \pmod{p}$$

$S = g^{ab} \pmod{p}$  est très difficile à trouver avec des grands nombres pour un observateur extérieur, même s'il connaît  $g$ ,  $p$ ,  $A$ , et  $B$ .





# STORING PASSWORDS

Plain text bad

## Best Practice

How you store user passwords is one of the most important parts of being a web developer. Here's some guidelines:

- A cryptographically sound hashing function should be used, like **SHA256**.
- Each user password should be **salted** with a randomly-generated string before being hashed.
- Ideally, a **secret key** stored on the filesystem and loaded at boot should be used as well for extra security.

Here we store PW in the database.

**hash:** PW = SHA256(password)

**salt:** salt = rand\_str()  
PW = SHA256(password + salt)

**salt+sk:** // sk is a global  
salt = rand\_str()  
PW = SHA256(password + salt + sk)



# How Passwords Are Cracked

Oh no



## Brute Force

Try every single combination along with the salt until the hash or a collision is found.

**Effectiveness:** Only doable with very weak passwords (only numbers and letters). Very slow.

**Solution:** Strong hashing algorithm, strong passwords, secret key.



## Lookup / [Rainbow Tables](#)

Find the hash in a pre-computed list of hashes. Cracks multiple passwords at a time if users share passwords.

**Effectiveness:** Lookup table is **almost instant** on weak passwords, rainbow table works great on long, strong passwords.

**Solution:** Salting passwords, secret key.



## Dictionary Attack

Try common passwords and hash them using the salt.

**Effectiveness:** Great against common passwords, still relatively slow.

**Solution:** Strong passwords, secret key.



# LA SUITE AU JOUR 2 !

Now to the challenges



# CRYPTO ASYMÉTRIQUE

Jésus, mais pour la crypto





# DEUX CLÉS

À DEUX, C'EST MIEUX

PLAINTEXT  
MESSAGE



CLÉ PUBLIQUE

1]  
FQ?E??r.?b?  
?&y?hR?????)  
,????@??  
[%O?!?k?u??  
?qV????(h



CLÉ PRIVÉE

PLAINTEXT  
MESSAGE



# DESCRIPTION DES CLÉS



## LA CLÉ PUBLIQUE (PK)

- **Chiffre** un message mais ne peut le déchiffrer
- Partagée avec **tout le monde**
- **Très difficile** de retrouver la SK
- Peut **identifier** un message signé par la SK



## LA CLÉ PRIVÉE / SECRÈTE (SK)

- **Déchiffre** un message chiffré par la PK
- Gardée **secrète**
- **Identifie** le propriétaire
- Peut **signer** un message pour identifier l'auteur



---

# PRINCIPES DE RSA

YOUPI C'EST L'HEURE DES MATHS



# CLÉ PUBLIQUE

**Clé publique:** 2 nombres  
**e** (Exponent) et **N** (Modulus)

- Le plaintext est encodé en un **nombre M** (+ padding)
- Le **chiffrement** de **M** avec la clé publique (**e**, **N**) donne le ciphertext **C**

"abcdef"  $\longrightarrow$  **M**

$$\mathbf{C} = \mathbf{M}^{\mathbf{e}} \% \mathbf{N}$$



**Clé privée:** un nombre, **d**

- **Déchiffrement** du ciphertext **C** avec la clé privée **d** donne le nombre original **M**
- Le plain text est le nombre **M**

$$M = C^d \% N$$

**M**  $\longrightarrow$  "abcdef"



# RELATION DES CLÉS

C'EST EN GROS POUR UNE RAISON

$$M = (M^{e^d}) \% N$$



---

# GÉNÉRATION DE CLÉS

ET OUI JAMIE!



# GÉNÉRATION DE PAIRE DE CLÉS

POUR LA DONNER À TES 3 AMIS

- Génération de 2 grands nombres premiers, **p** et **q**
- Génération de **N** =  $p \cdot q$
- Calcul de **λ**, le plus petit multiple commun entre (p-1) et (q-1)
- On choisit un nombre premier **e** plus petit que **λ** et co-premier à **λ** (65537 en général)
- On calcule **d**, l'**inverse modulaire** de **e** par **λ**

$$p \cdot q$$

$$N = p \cdot q$$

$$\lambda = \text{lcm}(p-1, q-1)$$

$$1 < e < \lambda$$

$$(e \cdot d) \% \lambda = 1$$





---

# **LIMITATIONS**

COMMENT ÇA, PAS TOUT-PUISSANT ?



# LIMITES

BOF BOF

- Calcul **lent** par rapport à AES
- On est limités sur la taille du message : **245B** pour une clé de **2048B** !
- Cela vient de  $C = M^e \% N$



# ET SI... RSA ONLY ?

BOF BOF AGAIN...

- There can be substantial weaknesses in how the data is split and then rebuilt. There is no well-studied standard for that.
- Each chunk, when encrypted, grows a bit (with a 2048-bit key, the 245 bytes of data become 256 bytes); when processing large amounts of data, the size overhead becomes significant
- Decryption of a large message may become intolerably expensive



RSA maximum bytes to encrypt, comparison to AES in terms of security?



# RSA POUR LA CLÉ, AES POUR CHIFFRER

ET LES POULES SERONT BIEN GARDÉES

- **RSA** pour l'échange de clés
- **AES** pour chiffrer en quantité



---

# VULNÉRABILITÉS

LA PARTIE FUN



# FACTORISATION DE CLÉ PUBLIQUE

PARCE QUE FUCK LES RÈGLES

RSA repose sur le fait que factoriser **N** (trouver **p** et **q** à partir de **N**) est **extrêmement difficile**.

Donc pas évident, mais faisable dans dans quelques cas:

- **N** est très petit, on peut juste **bruteforce**
- Les facteurs de **N** sont **connus** (<http://factordb.com/>)
- $d$  est très petit comparé à **N**, factorisation par **fractions continues**
- Mauvaise génération de **p** et **q**



# ATTAQUE SUR PETIT EXPOSANT PUBLIC

J'AI PAS TROUVÉ DE MOYEN D'EXPLIQUER ÇA GRAPHIQUEMENT

Si plusieurs personnes choisissent un petit exposant public ( $e=3$ ) et que le même message  $M$  est envoyé à ces personnes, générant 3 ciphers ( $C_1$ ,  $C_2$ ,  $C_3$ ), on a donc :

$$M^3 = C_1 \% N_1 = C_2 \% N_2 = C_3 \% N_3$$

On peut calculer  $M^3 \% N_1.N_2.N_3$  grâce au **théorème des restes chinois**.  
Mais comme  $M$  est inférieur à  $N_1$ ,  $N_2$  et  $N_3$ ,

$$M^3 < N_1.N_2.N_3 \text{ et donc } M^3 \% N_1.N_2.N_3 == M^3$$

Et c'est gagné, on a trouvé  $M$  !



Chiffrement :  $C = M^e \% N$ , ce qui équivaut à  $M^e = C \% N$



# NOTES

- La majorité de ces attaques sont sur le "textbook RSA", c.a.d sur RSA sans padding appliqué.
- La factorisation difficile, la base de RSA, n'a **jamais été prouvée** mathématiquement.
- RSA reste utilisé surtout pour échanger des clés symétriques qui sont ensuite utilisées pour s'envoyer des données





---

# Demo

Soyez attentifs



# LIENS / OUTILS UTILES

## Why RSA works

Un article qui va en profondeur sur les mathématiques impliquées dans RSA

<http://doctrina.org/Why-RSA-Works-Three-Fundamental-Questions-Answered.html>

## Numpy

Une lib python qui simplifie la manipulation et les calculs sur les grands nombres.

## Sagemath

Un outil qui peut faire des calculs de grands nombres très facilement, et qui utilise Python

<http://www.sagemath.org/>

## RootMe

Exos de sécurité variés, extrêmement utile pour s'améliorer

<https://www.root-me.org/>

## CryptoHack

Exos de crypto, très bonne qualité

<https://cryptohack.org/>



# That's all folks!

Now go have fun!