Faculté des Sciences & Techniques

Université de Limoges

Master 1ère année

«Remember when, on the Internet, nobody knew who you were?»

Sécurité des Usages TIC et Cryptographie — P-F. Bonnefoi

Version du 4 février 2025

Table des matières

1 Cryptographie : Introduction et définitions 4
La notion de codage de l'information & la cryptographie associée 10
Chiffrement par substitution 11
Cryptanalyse du chiffrement par substitution 14
Chiffrement par transposition 22
Cryptanalyse du chiffrement par transposition 25
2 Cryptographie moderne 28
Chiffrement à clé symétrique 29
Le chiffrement par bloc pour le chiffrement symétrique : chaînage ou non ? 34
Transmission de l'information et entropie 41
Réseau de Feistel : propriétés du xor pour définir un cryptosystème 66
Le chiffrement par flux 73
Les limites de la cryptographie Symétrique 76
Chiffrement à clé asymétrique 79
Courbes elliptiques sur domaine fini 93
Chiffrement asymétrique : application à l'authentification 99
Échange sécurisé : la notion de clé de session 112
L'authentification dynamique ou «vivante» 119
L'authentification d'un document 124
Compression sécurisée de document : fonction de hachage 129

Application des fonctions de hachage cryptographique : l'arbre de Merkle 133
Attaque sur les fonctions de hachage : «*rainbow tables*» 136
MAC,«*Message Authentication Code*» 141
Asymétrique + hachage : la signature électronique 147
3 Quelques notations pour **modéliser** les protocoles 155
Les règles de bonnes pratiques de conception de protocole sécurisé 161
HTTP Basic/Digest Authentication, RFC 2617 165
Logique BAN, «*Burrows–Abadi–Needham*» 171
4 Chiffrement authentifié avec des MACs 177
AEAD, «*Authenticated Encryption with Associated Data*» 188
AES-GCM, «*Galois Counter Mode*» 190
5 La sécurité «électronique» : la signature électronique 197
La signature électronique : aspects juridiques 199
Le problème de l'échange de clé publique 204
6 La PKI, «*Public Key Infrastructure*» un tiers de confiance 206
Le certificat 207
Composition d'une PKI 215
Obtention de certificat 216
Composition d'une PKI : utilisation et vérification du certificat 228
L'Horodatage 241

Introduction

Depuis l'Egypte ancienne, l'homme a voulu pouvoir échanger des informations de facon **confidentielle**.
En grec :
Cryptographie : (κρυπτο – γραφην) écriture cachée / brouillée.

Il existe de nombreux domaines où ce besoin est vital :

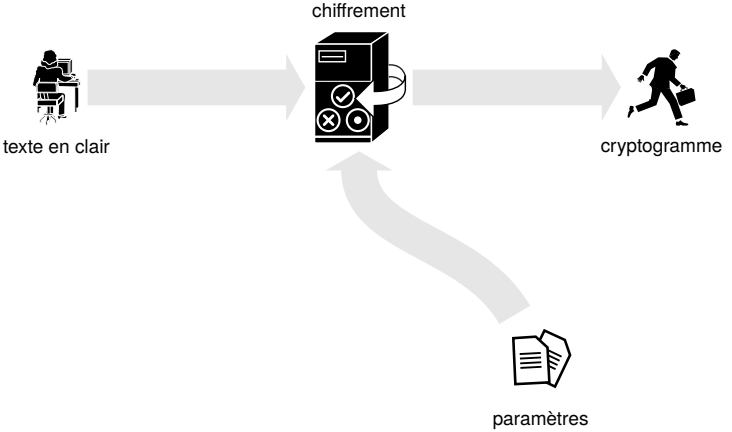
- ▷ **militaire** (sur un champ de bataille ou bien pour protéger l'accès à l'arme atomique) ;
- ▷ **commercial** (protection de secrets industriels) ;
- ▷ **bancaire** (protection des informations liées à une transaction financière) ;
- ▷ **vie privée** (protection des relations entre les personnes) ;
- ▷ **diplomatique** (le fameux «*téléphone rouge*» entre États-Unis et Union soviétique) ;
- ▷ ...

Définitions

Pour assurer la protection des accès à une information, on utilise des techniques de **chiffrement**.
Ces techniques s'appliquent à des **messages** lisibles appelés également «texte en clair».

Le fait de «*cacher*» un message de telle facon à le rendre secret s'appelle **chiffrement**.
La méthode inverse, consistant à retrouver le message original, est appelé **déchiffrement**.

Les messages à chiffrer, appelés «*texte en clair*», sont transformés grâce à une **méthode de chiffrement paramétrable**.
Si la méthode est connue de tous, ce sont les **paramètres** qui constituent la protection : ils servent à chiffrer/déchiffrer le message.



Ce **cryptogramme** est ensuite envoyé à son destinataire.

On appelle **cryptanalyse** les techniques employées pour déchiffrer un cryptogramme, **sans connaître** la méthode et/ou ses paramètres.

Le chiffrement est aussi appelé **cryptographie**.

L'ensemble des techniques de cryptographie et de cryptanalyse est appelé **cryptologie**.

3. Les bases de la cryptographie

a. Vocabulaire

La cryptographie est une discipline consistant à manipuler des données de telle façon que les services suivants puissent être fournis :

Intégrité

Objectif : s'assurer que les données n'ont pas été modifiées sans autorisation.
Remarque : dans les faits, la cryptographie ne s'attache pas vraiment à empêcher une modification de données, mais plutôt à fournir un moyen sûr de détecter une modification malveillante.

Confidentialité

Objectif : ne permettre l'accès aux données qu'aux seules personnes autorisées.

Preuve (authentification et non-répudiation)

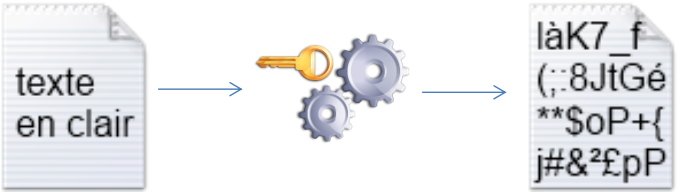
Objectif : fournir un moyen de preuve garantissant la véritable identité des entités ainsi que l'imputation de leurs actions.

3. Les bases de la cryptographie

a. Vocabulaire

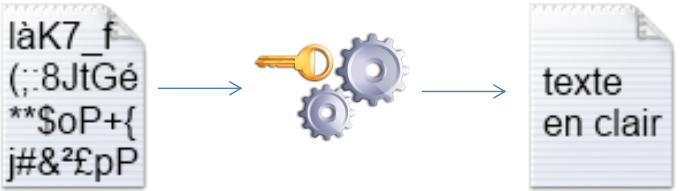
Chiffrer

Transformer une donnée de telle façon qu'elle devienne incompréhensible. Seules les entités autorisées pourront comprendre cette donnée chiffrée.



Déchiffrer

Transformer une donnée précédemment chiffrée pour reconstituer la donnée d'origine. Seules les entités autorisées ont la capacité de procéder à cette action.



Recours à un algorithme et à une clé cryptographique.

La stéganographie ou l'art de la dissimulation

En grec :
Stéganographie : (στεγανο – γραφην) écriture couverte/dissimulée.
Connaissance de l'existence de l'information $\Longrightarrow$ Connaissance de l'information
Cette méthode consiste à **dissimuler l'information à chiffrer** dans une autre information. On appelle cette méthode la «*stéganographie*».
Exemple : utiliser un bit tous les 8 bits dans une image (un bit de poids faible de préférence). L'image est faiblement modifiée et rien ne permet de savoir qu'elle contient un message caché.

Cette méthode peut être utilisé en plus de techniques de **cryptographie avancée** et permet d'en dissimuler l'usage.

Elles peut être utilisée de différentes manières :

- en **associant un groupe de lettres** à un caractère et en composant un texte qui ait un sens pour groupes de lettres, par exemple dans un compte rendu de partie d'échec où chaque coup joué correspond à une lettre du message secret et donne l'illusion d'une partie «*normale*» ;
- le **filigrane** ou «*watermarking*» pour dissimuler une information dans un document pour en permettre l'identification (protection des droits d'auteur) ;
- le **canal de communication caché** ou «*cover channel*» qui permet de diposer d'un véritable canal de communication en détournant l'usage de canaux de communications anodins. Cette technique permet de déjouer l'usage de firewall.
- ...

Exemple : ralentir artificiellement un transfert ftp ou au contraire l'accélérer pour coder un bit à 1 ou à 0, et pouvoir transmettre à un observateur le message qu'il construit.

La **cryptanalyse** reste **difficile** et doit s'appliquer à de **gros volumes** de données à **l'aveugle**.

Un peu d’Histoire...

Au début, il y eut les caractères et l'alphabet...

Historiquement, l'utilisation **d'alphabet** a permis de **coder** chaque mot du langage à partir de mêmes symboles à la différence des **idéogrammes chinois** par exemple.
L'ajout d'un **ordre** sur ces lettres à permis de définir les premières méthodes «*mathématiques*» de chiffrement d'un message constitué de lettres (code César, ROT13…).

Et des méthodes de chiffrement adaptées…

Ces chiffrements partent d'un message contenant des lettres vers un cryptogramme contenant également des lettres.
Ces méthodes se décomposent en deux grandes familles de chiffrement :

- par substitution ;
- par transposition.

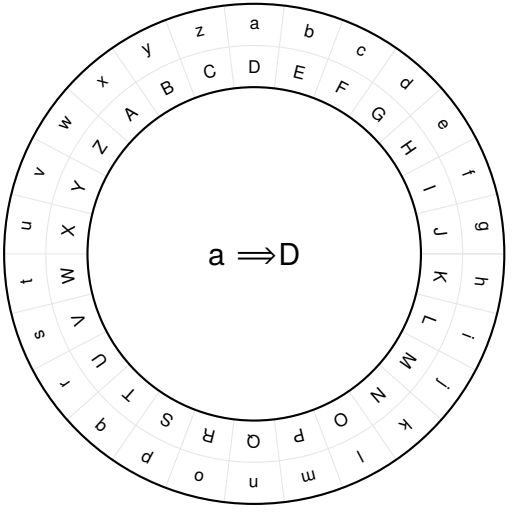
D'autres formes de chiffrement ?

Il existe également d'autres formes comme le **code morse** ou bien les **sémaphores** dans la Marine.
Ce sont des techniques de brouillage.

Chiffrement de César

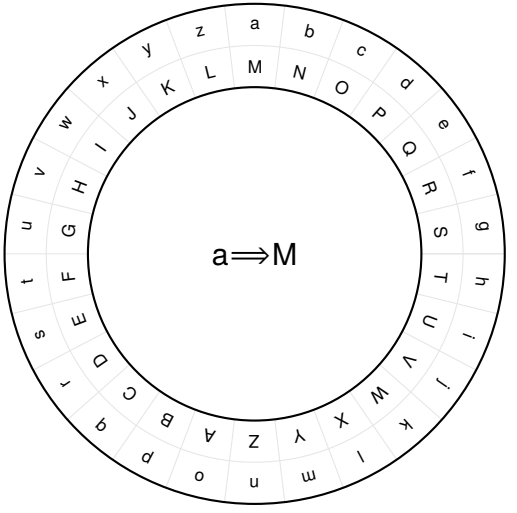
Le texte en clair : «*le petit chaperon se promenait dans la foret*»

chiffrement avec un décalage de 3 :



donne : «*OHSHW LWFKD SHURQ VHSUR PHQDL WGDQV ODIRU HW*»

chiffrement avec un décalage de 12 :



donne : «*XQBQF UFOTM BQDAZ EQBDA YQZMU FPMZE XMRAD QF*»

```
<!-- DEBUT DU SCRIPT MAILTO ANTI SPAM-->
<!-- Script provenant de http://www.toulouse-renaissance.net/c_outils/ -->

<script type="text/javascript">
<!--
var a, s, n;
function Crypt(s) {
  r='';for(i=0;i<s.length;i++)
   {
   n=s.charCodeAt(i);
      if (n>=8364) {n = 128;}
    r += String.fromCharCode( n - 3 ); }
return r;}

a ="pdlowr=";
m='&#64;';
d=unescape(m);

var nom = "protection";
var domaine = "votre_site.com";
var aro = nom + d + domaine;
document.write('<a href='+Crypt(a) + aro + '>');
document.write(aro + '</a>');
// -->
</script>

<!-- FIN DU SCRIPT MAILTO ANTI SPAM-->
```

Chiffrement de César en javascript

Pour chiffrer, chaque lettre est remplacée par celle trois rangs plus loin dans l'alphabet : A devient D, B devient E, ...

Pour déchiffrer, le décalage est effectué dans l'ordre inverse : "pdlowr=" devient "mailto:"

On se protège ainsi des robots qui analysent les pages web à la recherche d'adresse électronique valide.

Et si on veut «*casser*» le chiffrement ?

Cryptanalyse du chiffrement par substitution

Dans le cas de l'utilisation d'un code par substitution, la cryptanalyse ou déchiffrement se fait par l'utilisation de données statistiques :

En anglais :

- les caractères les plus fréquemment utilisés sont : e, t, o, a, n, i…
- les combinaisons de deux lettres (digrammes) les plus fréquentes sont : th, in, er, re, et an.
- les combinaisons de trois lettres (trigrammes) : the, ing, and et ion.

Méthode empirique de cryptanalyse

Il suffit pour retrouver le texte en clair de :

- de rechercher les **caractères**, **digrammes** et **trigrammes** les plus fréquents du texte chiffré ;
- de faire des **suppositions** en les associants à ceux les plus fréquents d'un texte en clair (dans la langue choisi).

Par exemple dans un texte chiffré appartenant à une banque il est probable de trouver des mots tel que financier, montant, solde…

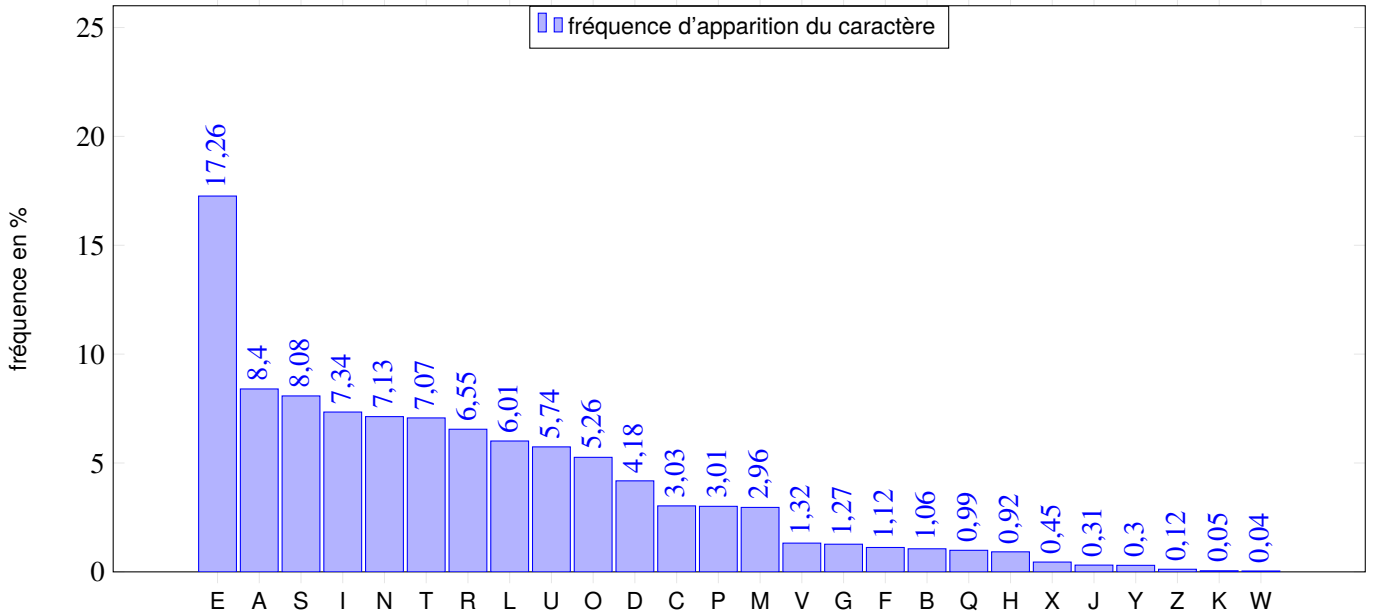
Comment finir la cryptanalyse ?

Si certains mots commencent à émerger du texte chiffré, alors il y a de **fortes probabilités** que le code de chiffrement soit découvert.

Un code par substitution **ne modifie pas** les **propriétés statistiques** des caractères, digrammes et trigrammes substitués.

Il conserve **l'ordre des caractères** du texte en clair, mais masque ces caractères.

Fréquence d'apparition des caractères, bigrammes et trigrammes en français



Les séquences de deux lettres triées suivant les plus fréquentes

Bigrammes	ES	DE	LE	EN	RE	NT	ON	ER	TE	EL	AN	SE	ET	LA	AI	IT	ME	OU	EM	IE
Nombres	3318	2409	2366	2121	1885	1694	1646	1514	1484	1382	1378	1377	1307	1270	1255	1243	1099	1086	1056	1030

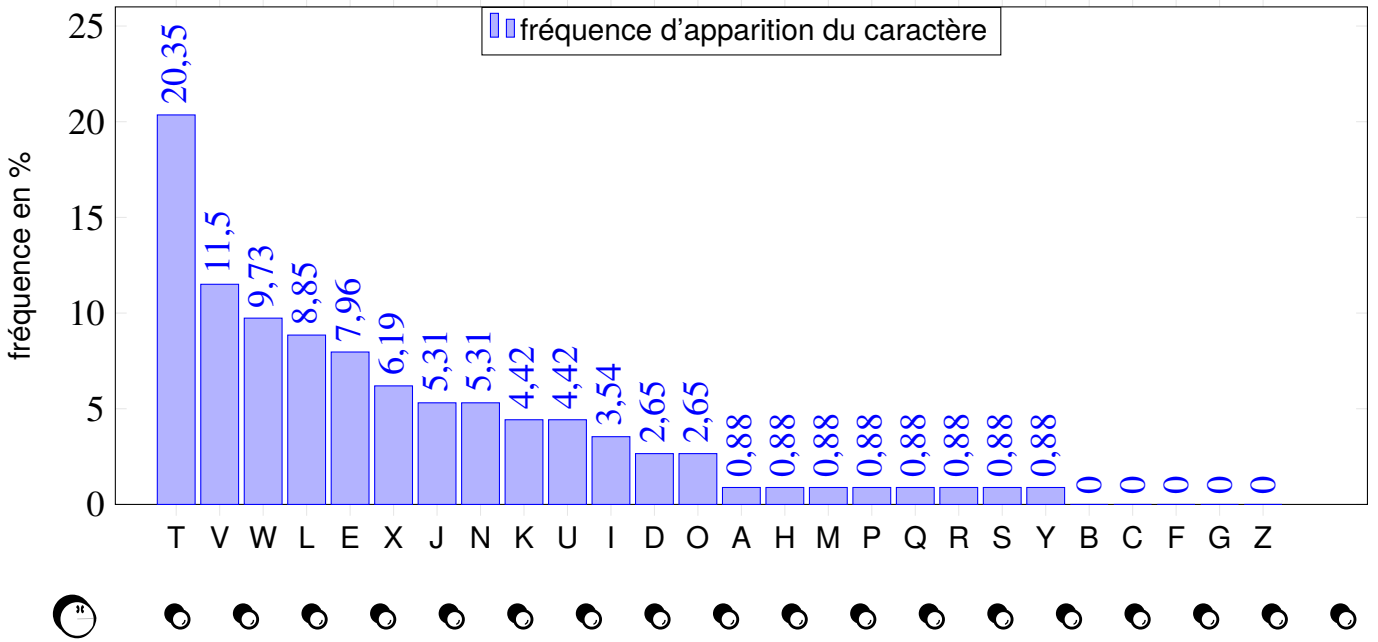
Les séquences de trois lettres triées suivant les plus fréquentes

Trigrammes	ENT	LES	EDE	DES	QUE	AIT	LLE	SDE	ION	EME	ELA	RES	MEN	ESE	DEL	ANT	TIO	PAR	ESD	TDE
Nombres	900	801	630	609	607	542	509	508	477	472	437	432	425	416	404	397	383	360	351	350

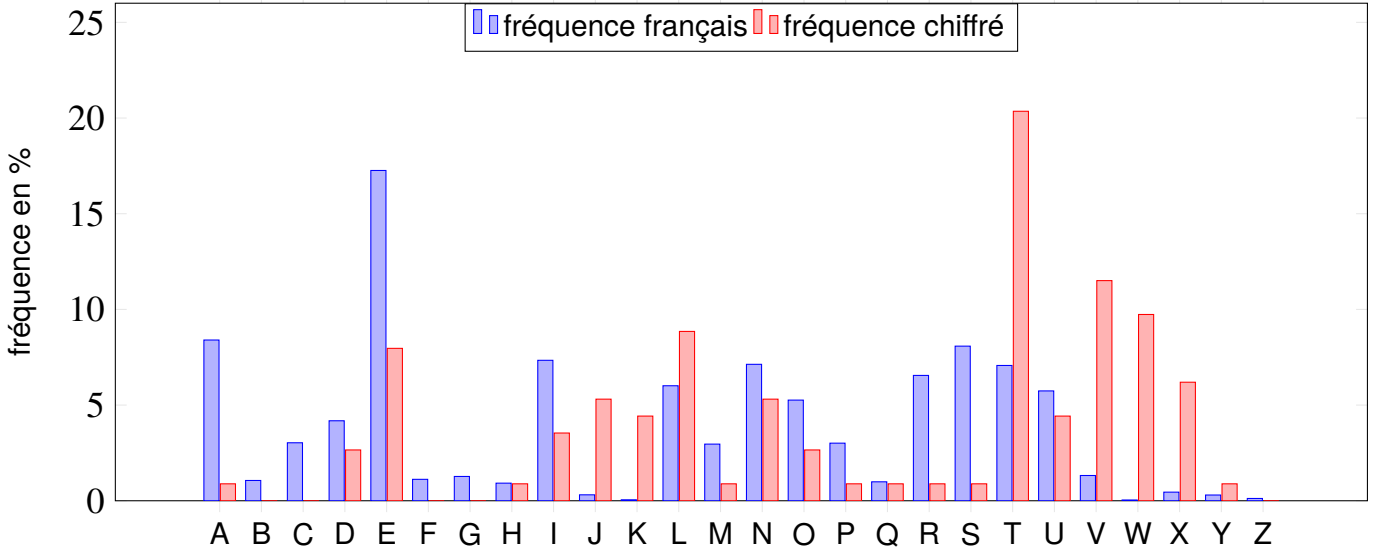
Teste chiffré

JTVMNKKTVLDEVVTLWTWITKTXUTLWJ
ERUTVTWTHDXATLIUNEWV.
JTVIEVWELOWENLVVNOEDJTVLTPTXYT
LWTWUTSNLITTVQXTVXUJXWEJEWTON
KKXLT.

Les fréquences calculées sur le texte chiffré



Comparaison des fréquences entre chiffré et clair



Début du déchiffrement

On associe les caractères qui ont la fréquence la plus élevées : T $\Longrightarrow$ e

```
JeVMNKKeVLDEVVeLWeWIeKeXUeLWJERUeVeWeHDXAeLIUNEWVJeVIEVWELOW
ENLVVNOEDJeVLePeXYeLWeWUeSNLIeeVQXeVXUJXWEJEWeONKKXLe
```

```
JeVMNKKeVLDEVVeLWeWIeKeXUeLWJERUeVeWeHDXAeLIUNEWVJeVIEVWELOW
 ENLVVNOEDJeVLePeXYeLWeWUeSNLIeeVQXeVXUJXWEJEWeONKKXLe
```

D'après la table étendue des fréquences de bigrammes, on trouve que les bigramme les plus fréquent, où les lettres sont les mêmes, sont :

- ▷ `ee` : la lettre «e» étant déjà affectée, on ne l'utilisera pas ;
- ▷ `ss` : on va associer V ⟹ s ;
- ▷ `ll` : on pourra tester cette association plus tard.

```
JesMNKKesLDEsseLWeWIeKeXUeLWJERUeseWeHDXAeLIUNEWsJesIEsWELOW
ENLssNOEDJesLePeXYeLWeWUeSNLIeesQXesXUJXWEJEWeONKKXLe
```

Puis avec l'association W ⟹ t :

```
JesMNKKesLDEsseLtetIeKeXUeLtJERUeseteHDXAeLIUNEtsJesIEstELOt
ENLssNOEDJesLePeXYeLtetUeSNLIeesQXesXUJXtEJEteONKKXLe
```

```
JesMNKKesLDEsseLtetIeKeXUeLtJERUeseteHDXAeLIUNEtsJesIEstELOt
 ENLssNOEDJesLePeXYeLtetUeSNLIeesQXesXUJXtEJEteONKKXLe
```

Le trigramme le plus fréquent commençant par «e» et finissant par «t» est «ent», d'où L ⟹ «n» :

```
JesMNKKesnDEssentetIeKeXUentJERUeseteHDXAenIUNEtsJesIEstEnOt
ENnssNOEDJesnePeXYentetUeSNnIeesQXesXUJXtEJEteONKKXne
```

Suite du déchiffrement

```
lesMNmmesnDEssentetIemeurentlERreseteHDuxenIrNEtslesIEstEnOt
ENnssNOEDlesnePeuventetreSNnIeesQuesurlutElEteONmmune
```

Les associations utilisées

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
x	y	z							l	m	n								e	r	s	t	u	v	w

Fin du déchiffrement

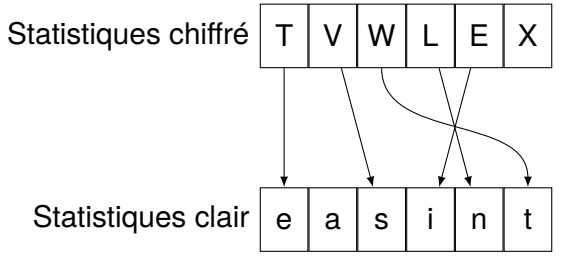
```
leshommesnaissentetdemeurentlibresetegauxendroitslesdistinct
ionssocialesnepeuventetrefondeesquesurlutilitecommune
```

Les associations utilisées

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
x	y	z	a	i	j		g	d	l	m	n	h	o	c	p	q	b	f	e	r	s	t	u	v	w

On remarquera que G⟹k même si ce caractère n'apparait pas dans le texte en clair.

Utilisation des fréquences les plus élevées pour le caractères, bigrammes et trigrammes



- VV $\Longrightarrow$ ss
- KK $\Longrightarrow$ mm
- TLW $\Longrightarrow$ ent

Utilisation d'un dictionnaire

Il est possible de tirer partie d'un **dictionnaire**, en regroupant les lettres en mots, puis en les recherchants dans le dictionnaire.

Dans tous les cas, la cryptanalyse reste un travail **exploratoire**, avec :

⇒la mise en place de nouvelles hypothèses ;

⇒la poursuite du processus de cryptanalyse ;

⇒le retour en arrière pour remettre en cause les dernières hypothèses ;

⇒le recommencement de cette méthode en .

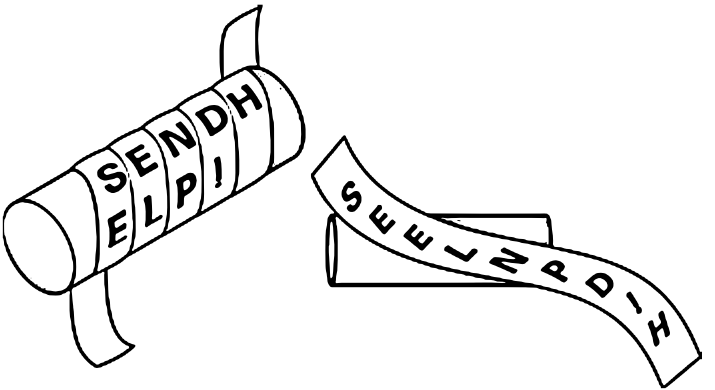
Autre forme de chiffrement

Toutes les lettres du message sont **présentes**, mais dans **un ordre différent**.

C'est un chiffrement de type **anagramme**.

Il utilise le principe mathématique des permutations (par colonne par exemple).

La scytale spartiate (5ème siècle avant notre ère) :



Texte en clair : `LA TRANSPOSITION PERMET EN THEORIE D'AVOIR UN HAUT DEGRE DE SECURITE`

L	R	S	S	I	P	M	E	H	R	D	O	U	A	D	R	E	C	I
A	A	P	I	O	E	E	N	E	I	A	I	N	U	E	E	S	U	T
T	N	O	T	N	R	T	T	O	E	V	R	H	T	G	D	E	R	E

Texte chiffré : `LRSSI PMEHR DOUAD RECIA APIOE ENEIA INUEE SUTTN OTNRT TOEVR HTGDE RE`

Chiffrement par transposition

Les méthodes de chiffrement par transposition consistent à **réarranger** les données à chiffrer de telle façon à les rendre **incompréhensibles**.

En général : **réarranger géométriquement** les données pour les rendre **visuellement** inexploitables.

Par exemple : `Ceci est un texte à chiffrer de la plus haute importance`

Le texte est regroupé en tableau, suivant un nombre de colonnes donné.

C	e	c	i		e	s	t		u
n		t	e	x	t	e		à	
c	h	i	f	f	r	e	r		d
e		l	a		p	l	u	s	
h	a	u	t	e		i	m	p	o
r	t	a	n	c	e				

`Cncehre h atctiluaiefatn…`

Chaque colonne est ensuite copiée l'une après l'autre.

Cassable ?

Cryptanalyse

- Déterminer si une substitution n'a pas été utilisée : une analyse statistique des caractères suffit à déterminer si les caractères ont été substitués (statistiques fréquentielles du texte identiques à celle d'un texte en clair).
- Si ce n'est pas le cas, il y a une **forte probabilité** pour qu'un chiffrement par transposition ait été employé.
- Ensuite, il faut faire une **hypothèse** sur le **nombre de colonnes** utilisées pour réaliser la transposition.

*Les codes de transposition contrairement aux codes par substitution **ne cachent pas** les caractères, mais modifient l'ordre des caractères.*

Et l'ordinateur fut…

L'arrivée des ordinateurs a totalement démodé ces méthodes de chiffrement (*on ne parle plus d'ailleurs de chiffrement car ces méthodes ne résiste pas au traitement informatique*).

La machine **Enigma** utilisée par les nazis a été «*cassée*» par trois cryptographes polonais : Marian Rejewski, Jerzy Różycki et Henryk Zygalski, puis cette méthode a été améliorée par Alan Turing, pionnier de l'informatique.

Il faut attendre les annés 60 pour voir les méthodes de **chiffrement moderne** basées sur l'usage de **clés**.

Combiner Substitution et Transposition

il est possible de faire subir aux caractères du «texte en clair» :

- une substitution ;
- plusieurs opérations de transposition.

Changer les paramètres de ces combinaisons très souvent

l'utilisation des paramètres de chaque opération doit être réduite au chiffrement de quelques messages avant d'être changés pour de nouveaux paramètres.

Combiner les paramètres

Les opérations sont connues, la séquence d'application des opérations est définie par la séquence des paramètres de chaque opération.

La combinaison des différents paramètres des différentes opérations permet de définir un **secret**.

Ce secret permet de réaliser le déchiffrement et assure la sécurité du cryptogramme.
Il est appelé **clé de chiffrement**.

Le but

rendre l'apparence du cryptogramme la plus « aléatoire » possible, c-à-d **éliminer les relations statistiques** des caractères du cryptogramme pour éviter la cryptanalyse :
Transposition + Substitution = Diffusion

L'actualité ?

les chiffrements tels que DES «*Data Encryption System*» et AES «*Advanced Encryption System*» sont utilisés à l'heure actuelle.

Utilisation de la cryptographie moderne pour la sécurité

Ce type de chiffrement repose sur l'utilisation :

- d'un **algorithme public**, connu de tous ;
- d'une **clé**.

Il correspond à la cryptographie moderne, par rapport aux codes par substitution et transposition.

Auparavant, les algorithmes étaient simples mais utilisaient des **clés longues**.
Exemple : un XOR entre le message à transmettre et une clé de même taille suffit à le rendre indéchiffrable...technique du masque jetable
Maintenant, le but est d'utiliser des algorithmes sophistiqués et complexes associés à des **clés courtes**.

Ces algorithmes représente des **investissements à long terme**, c-à-d qu'ils sont employés pendant de nombreuses années jusqu'à ce qu'ils ne puissent plus assurer le même niveau de sécurité.

Il existe **deux sortes** de chiffrement :

- à **clé symétrique** ;
- à **clé asymétrique**.

Rappel de l'hypothèse de base de la cryptographie :
Principe de Kerckhoff – Auguste Kerckhoff, «La cryptographie militaire», février 1883

- *L'opposant connaît le système cryptographique*
- *Toute la sécurité d'un système cryptographique doit reposer sur la clé, et pas sur le système lui-même*

Chiffrement à clé symétrique

Principe

La même clé doit être employée pour chiffrer ou déchiffrer le message : on parle de clé symétrique ou secrète.

Le chiffrement consiste alors à appliquer un algorithme avec la clé secrète sur les données à chiffrer. Le déchiffrement se fait à l'aide de cette **même clé secrète**.

Remarques

La qualité d'un crypto système symétrique se mesure par rapport :

- à des propriétés statistiques des textes chiffrés ;
- à la résistance aux classes d'attaques connues.

En pratique

Tant qu'un crypto système symétrique n'a pas été cassé, il est bon, après il est mauvais !

Chiffrement à clé symétrique

Ce chiffrement repose sur la définition d'une formule mathématique de la forme :

$$\text{Données chiffrées} = F(\text{données, clé})$$

Avec une **fonction inverse** de la forme :

$$\text{Données} = F^{-1}(\text{données chiffrées, clé})$$

Deux types d'algorithmes

- l'algorithme par **bloc** qui prend une longueur spécifiée de données comme entrée, et produit une longueur différente de données chiffrées (exemple : DES, AES...)
- l'algorithme en **flux continu** qui chiffre les données un bit à la fois (exemple : IDEA, CAST, RC4, SKIPjack...).

Avantages et inconvénients d'un crypto-système à clé symétrique

Le principal inconvénient provient de l'échange des **clés** qui doivent rester **secrètes**.
Pour être **totalement sûr** : les chiffrements à clés secrètes doivent utiliser des clés d'une longueur **au moins égale** à celle du message à chiffrer : «*One Time Pad*» ou «Masque Jetable» appliqué à l'aide de l'opération binaire XOR sur les données à chiffrer.
En pratique : les clés ont une taille donnée, **suffisante**.
La plupart des codes utilisés sont relativement **rapides** et peuvent s'appliquer à un **fort débit** de donnée à transmettre.
Il existe des processeurs spécialement concu pour réaliser le chiffrement et le déchiffrement.

Principaux algorithmes utilisés :

- **DES**, «*Data Encryption System*», IBM 1977 ;
- **AES**, «*Advanced Encryption Standard*», 2001 ;
- **IDEA**, «*International Data Encryption Algorithm*», Lai et Massey 1990, brevé expiré en 2012 ;
- **Blowfish**, Bruce Schneier, 1994.

Problème d'assurer la sécurité des clés

Problème de la distribution des clés qui doit se faire par un **canal sûr**.

Un standard de chiffrement

Développé dans les années 70 par IBM, la méthode DES fut adoptée et rendue standard par le gouvernement des Etats Unis.

Il devait répondre à l'époque aux **critères** suivants :

- ▷ avoir un haut niveau de sécurité lié à une clé de petite taille servant au chiffrement et au déchiffrement ;
- ▷ être **compréhensible** ;
- ▷ **ne pas dépendre de la confidentialité** de l'algorithme ;
- ▷ être **adaptable** et **économique** ;
- ▷ être **efficace** et **exportable** ;

Le DES est un standard utilisé depuis plus de 40 ans.

La clé est sur 64bits dont 8 sont utilisés comme calcul de l'intégrité des 56 autres (parité).

La méthode DES utilise des clés d'une taille de 56 bits ce qui la rend de nos jours facile à casser avec les nouvelles technologies de cryptanalyse.

Elle est toujours utilisée pour des petites tâches tel que l'échange de clés de chiffrement dans sa forme étendue **3DES** :

$$\text{chiffré} = C_{K_3}(D_{K_2}(C_{K_1}(\text{texte clair}))) \text{ et texte clair} = D_{K_1}(C_{K_2}(D_{K_3}(\text{chiffré})))$$

où les clés sont choisies :

- ▷ toutes différentes, soient une clé de 168 bits ;
- ▷ K_1 et K_2 sont différentes avec $K_3 = K_1$ soient un clé de 112bits ;
- ▷ toutes identiques $K_1 = K_2 = K_3$ équivalent au DES, ne doit plus être utilisé.

Il a suscité de nombreuses critiques, des suspicions de vulnérabilité à l'attaque de son algorithme, mais n'a pas eu d'alternatives jusqu'à ces dernières années : modifié par la NSA, trafiqué par IBM, ...

AES, «*Advanced Encryption Standard*»

L'AES est un standard de chiffrement symétrique choisi en l'an 2000 par le NIST destiné à remplacer le DES devenu trop faible au regard des attaques actuelles.

L'AES :

- ▷ est un **standard**, libre d'utilisation, sans restriction d'usage ni brevet ;
- ▷ est un algorithme de **chiffrement par blocs** (comme le DES) ;
- ▷ supporte **différentes combinaisons** [longueur de clé]-[longueur de bloc] : 128-128, 192-128 et 256-128 bits

Le choix de cet algorithme répond à de nombreux **critères** tels que :

- □ la **sécurité** ou l'effort requis pour une éventuelle cryptanalyse ;
- □ la **facilité de calcul** : cela entraîne une grande rapidité de traitement ;
- □ les **faibles besoins en ressources** : mémoire très faibles ;
- □ la **flexibilité d'implémentation** : cela inclut une grande variété de plates-formes et d'applications ainsi que des tailles de clés et de blocs supplémentaires (il est possible d'implémenter l'AES aussi bien sous forme logicielle que matérielle, câblé) ;
- □ la **simplicité** : le design de l'AES est relativement simple ;
- □ l'**intégration matérielle** dans l'électronique : intégration dans le processeur.

Et pour chiffrer des
données de taille importante en
chiffrement symétrique ?

Le **chiffrement par bloc** définie la méthode choisie pour chiffrer le message décomposé en **bloc**, c-à-d :

– dans quel ordre ;
– après quelle transformation.

On parlera de mode d'opérations.

Quatre modes définis

Quatre modes sont définis dans FIPS 81, «*Federal Information Processing Standards Publication 81*», (2 décembre 1980) et aussi dans la norme ANSI X3.106-1983.

- **ECB** «*Electronic Code Book*» ;
- **CBC** «*Cipher Block Chaining*» ;
- **CFB** «*Cipher FeedBack*» ;
- **OFB** «*Output FeedBack*».

Il existe de nouveaux modes mieux adaptés aux échanges réseaux comme le mode CTR, «*Counter mode*».

ECB : Electronic Codebook

Mode d'opération normal : il applique l'algorithme au texte clair en transformant normalement chaque bloc de texte clair.

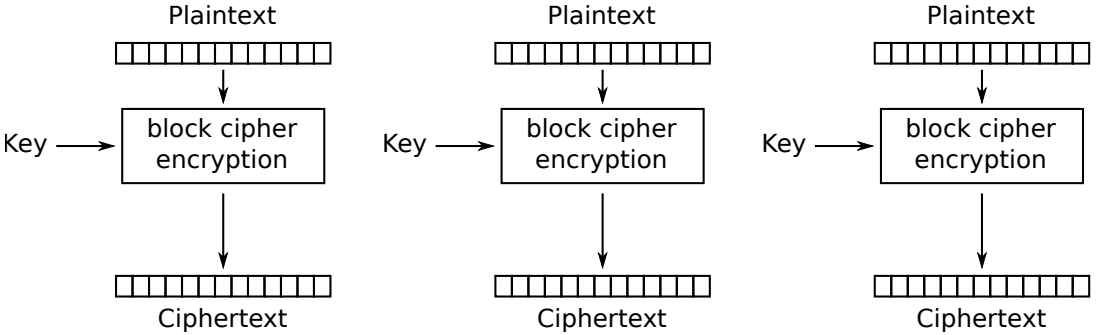
- T[n] = nième bloc de texte en clair.
- C[n] = nième bloc de texte chiffré.
- E(m) = fonction de chiffrement du bloc m.
- D(m) = fonction de déchiffrement du bloc m.
- **Chiffrement** : C[n] = E(T[n])
- **Déchiffrement** : T[n] = D(C[n])
- *T et C sont d'une longueur fixe*.

Problèmes :

▷ il faut un nombre suffisant d'octets de texte en clair avant de commencer (taille du bloc) ;
▷ si on utilise deux fois le **même texte clair** et la **même clé** de chiffrement, le **résultat** du chiffrement sera **identique**.

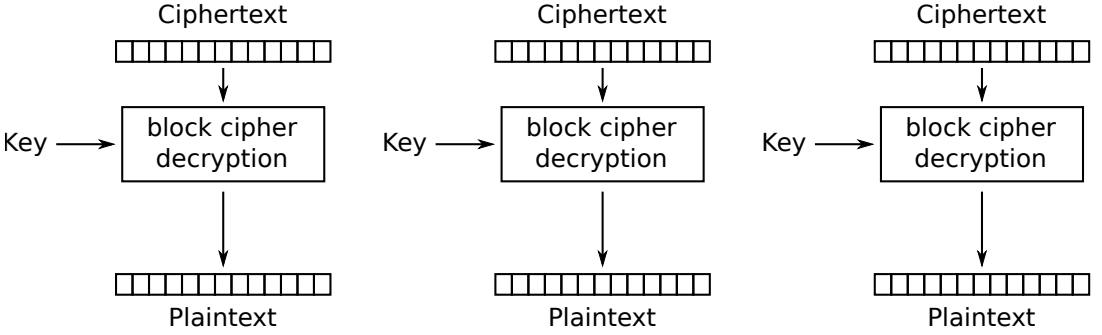
Attaque : en retrouvant un endroit précis dans une transaction et en effectuant une substitution avec les données d'une autre transaction utilisant la même clé de chiffrement.

Chiffrement



Electronic Codebook (ECB) mode encryption

Déchiffrement



Electronic Codebook (ECB) mode decryption

Les limites du mode ECB

https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation####Electronic_Codebook_(ECB)

L'original :

Le chiffré :

⟹ *Mais que se cache-t-il derrière cette image chiffrée...*

CBC : Cipher Block Chaining

C'est un des modes les plus populaires.

Il apporte une solution au premier problème du mode ECB :

- ▷ avant d'être chiffré, l'opération binaire « XOR » est appliquée entre le bloc actuel de texte en clair et le bloc précédent de texte chiffré ;
- ▷ pour le tout premier bloc, un bloc de contenu aléatoire est généré et utilisé, appelé « vecteur d'initialisation » (initialization vector, ou IV).

Ce premier bloc est envoyé tel quel avec le message chiffré.

- □ T[n] = nième bloc de texte en clair.
- □ E(m) = fonction de chiffrement du bloc m.
- □ C[n] = nième bloc de texte chiffré.
- □ D(m) = fonction de déchiffrement du bloc m.
- □ VI = vecteur d'initialisation

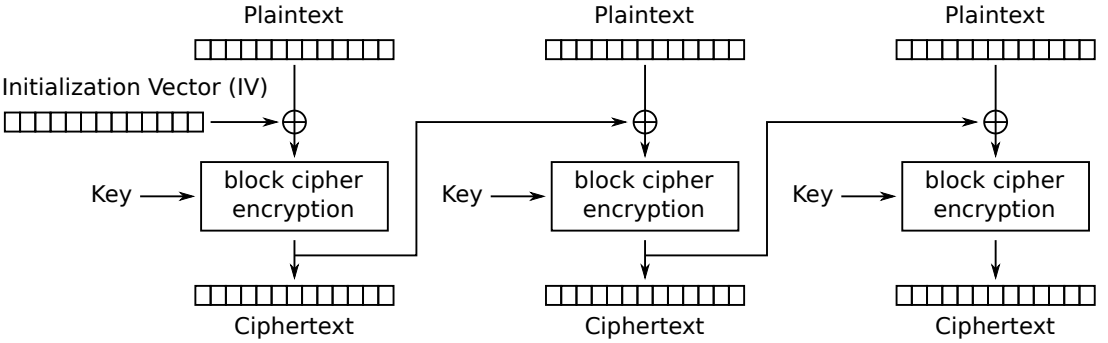
Chiffrement	C[0] = E(T[0] xor VI) C[n] = E(T[n] xor C[n-1]) , si (n > 0)
Déchiffrement	T[0] = D(C[n]) xor VI T[n] = D[C[n]) xor C[n-1] , si (n > 0) *T et C sont d'une longueur fixe*

Protection contre les attaques : il ne faut pas réutiliser le **même Vecteur d'Initialisation** avec le même chiffrement (la **même clé de chiffrement**).

Question : *quand est-ce que le chiffrement ECB et CBC sont identiques ?*

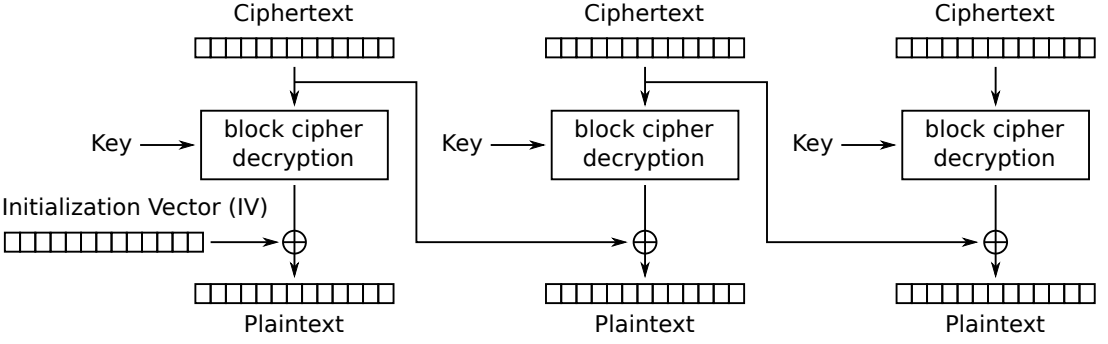
Chiffrement par bloc : le mode CBC

Chiffrement



Cipher Block Chaining (CBC) mode encryption

Déchiffrement

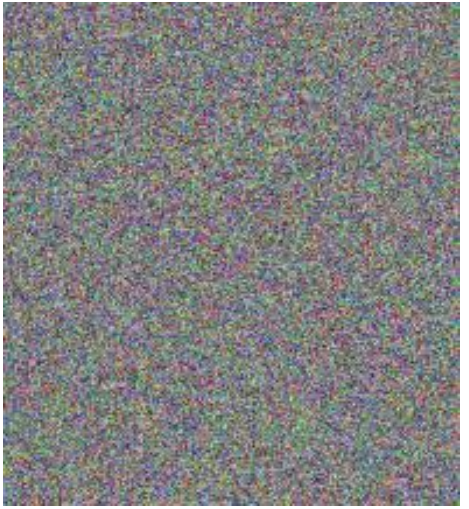


Cipher Block Chaining (CBC) mode decryption

Quand on n'utilise pas ECB...

L'original :

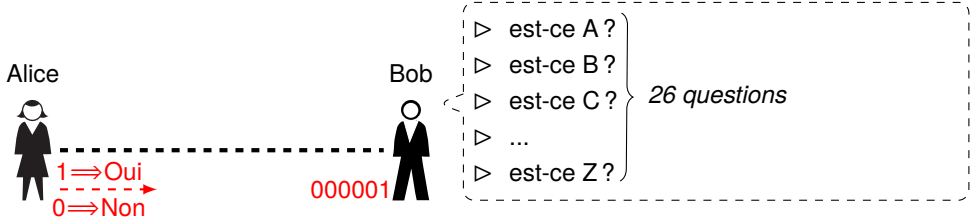
Le chiffré :



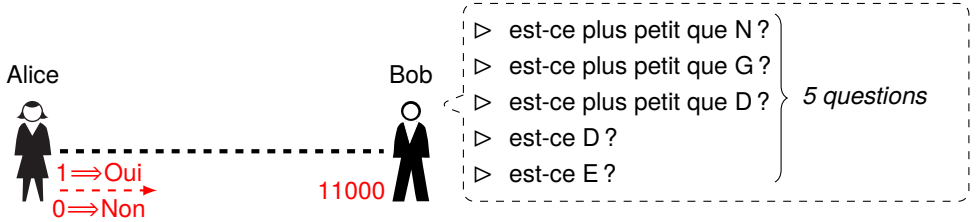
⟹ *On obtient du bruit ou de l'aléas...*

Et si on regardait la notion d'aléas ?
Transmission de l'Information
et
Entropie

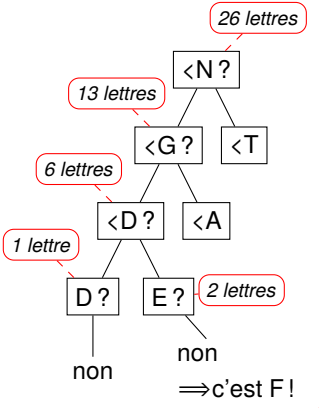
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z



F



F



- ▷ Alice a choisi la lettre F ;
- ▷ Bob lui pose plusieurs questions pour connaître la lettre qu'elle veut transmettre :
  - ◇ methode **naïve** : poser une question pour chaque lettre de l'alphabet à laquelle Alice réponds par 1, oui ou 0, non ;
  - ◇ méthode **optimisée** : découper l'espace de lettres ordonné en deux à chaque question et on cherche le **nombre de questions nécessaires** :
    - ⋆ on a $2^{\text{nbre questions}} = 26$,

⟹ en **moyenne** $\log_2(26) = 4.7$ questions pour une lettre. Pour un mot de 6 lettres : $6 * 4.7 = 28.2$ questions ou «*bits*».

«binary digit»

Imaginons transmettre des mots composés sur un alphabet de 26 lettres :

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
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

Pour construire un mot de 3 lettres, on a : $26 * 26 * 26 = 26^3$ possibilités, c-à-d toutes les permutations possibles des séquences de 3 symboles parmi un choix de 26.

Mesure de la quantité d'information

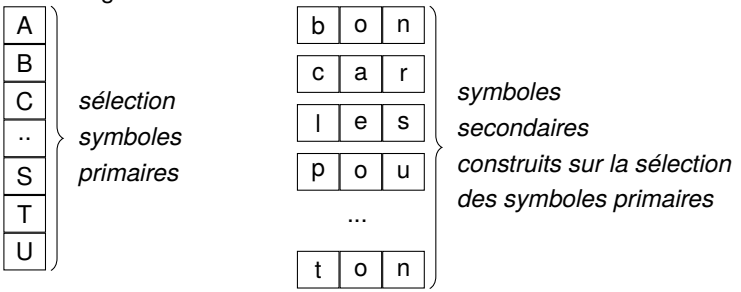
Une **mesure de «*l'information*»**, h, est :

$h = n\log(s) = log(s^n)$ où n est la taille de la **séquence** et s est le nombre de **symboles** utilisés pour la construire.

Pour des mots de 3 lettres, $h = log(26^3) = 41.2$ *(on peut utiliser le logarithme en base 2).*

Mais transmet-on toutes les séquences possibles ?

Pour des mots appartenant à une langue :



Imaginons transmettre **uniquement** les mots «*oui*» et «*non*» :

▷ ils contiennent chacun 3 lettres.

▷ si ce sont les deux seuls mots à transmettre, alors on a que **deux symboles secondaires**.

⟹**La transmission de l'information n'est pas aléatoire, mais un mélange d'éléments prédictibles et de surprises**.

⟹Si une information est **prédictible**, alors il n'est pas nécessaire de la transmettre.

⟹Elle peut être **réduite** si on définit les **symboles** et leurs séquences en **éliminant les redondances**.

Comment réduire du texte à transmettre ? Utilisation d'un dictionnaire

On définit un **dictionnaire** où :

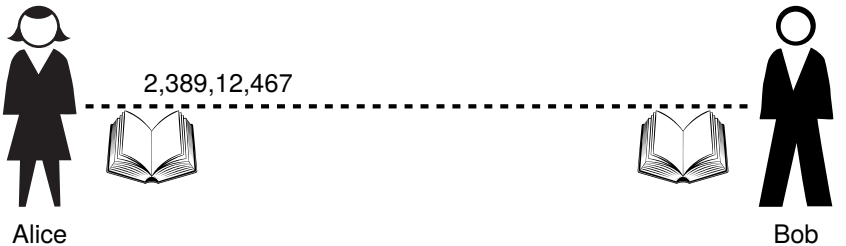
- ▷ chaque mot est associé à une valeur numérique ;
- ▷ les mots les plus utilisés sont associés aux valeurs numériques les plus courtes :

`https://fr.wiktionary.org/wiki/Wiktionnaire:Listes_de_fréquence/wortschatz-fr-1-2000`

1.	de	3.	le	5.	les	7.	en
2.	la	4.	et	6.	des	8.	un
1993.	dizaines	1995.	exactement	1997.	scénario	1999.	émissions
1994.	d'environ	1996.	outil	1998.	coups	2000.	éventuellement

- ▷ «*la sécurité est importante*» ⟹ «*2 389 12 467*»

Pour communiquer Alice et Bob doivent disposer du même dictionnaire :



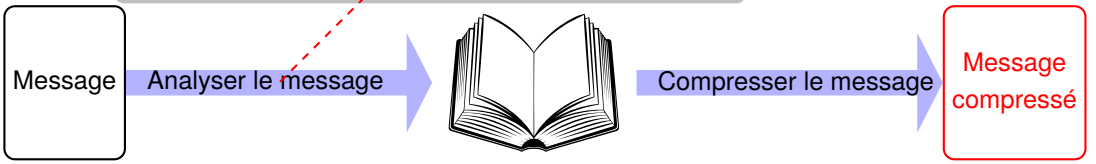
On peut utiliser des **dictionnaires spécialisés** :

- □ Telegraph Dictionary and Seamen's Signal Book
  H K ⟹ «*We must abandon the vessel or place*»
  H S ⟹ «*You have so much abbreviated the words of the message that it is not intelligible.*»
- □ Cotton Telegraph Code :
  Joke ⟹ «*Ship by Memphis & Charleston Railroad, via Norfolk*»
  Kingdom ⟹ «*At what price can you purchase stained, cotton, free from sand or dust, of long, strong staple ?*»

Améliorations : construire le dictionnaire sur les données à transmettre

▷ Comment construire le dictionnaire **en fonction des données** à transmettre ?

Construire le dictionnaire **parfait** pour ce message :
faire correspondre les séquences **les plus fréquentes**
à des séquences de caractères **les plus courtes**



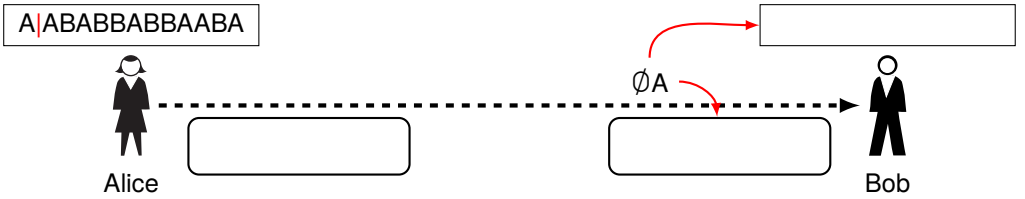
▷ Est-ce intéressant ? Oui, si la **taille** du message compressé et du dictionnaire est **inférieure** au message :

▷ Comment **ne pas transmettre** le dictionnaire ? Construire le dictionnaire :

- ◇ **au fur et à mesure** de la lecture du message ;
- ◇ **simultanément** sur l'émetteur et le récepteur :
  - ⋆ le transmetteur envoi le message compressé en fonction du contenu courant du dictionnaire ;
  - ⋆ le récepteur construit le dictionnaire en réalisant l'opération inverse et recompose le message.

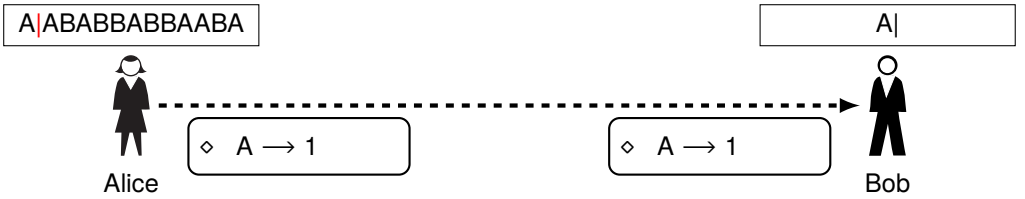
Travail de l'analyseur/compresseur :

▷ lire caractère par caractère et s'arrêter sur une séquence non déja vu :



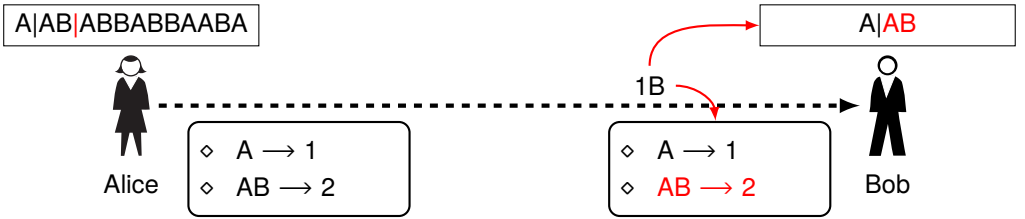
*Alice lit le caractère «*A*» du message.*

Au départ, le dictionnaire est vide $\emptyset$ *et Alice transmet* $\emptyset$*A à Bob.*

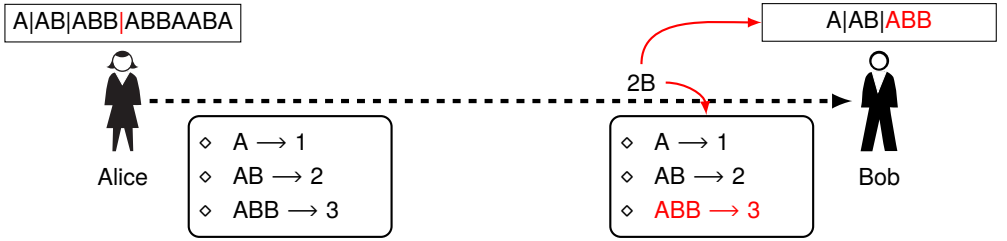


*Alice et Bob rentre dans leur dictionnaire respectif la première association «*A ⟶ 1*».*

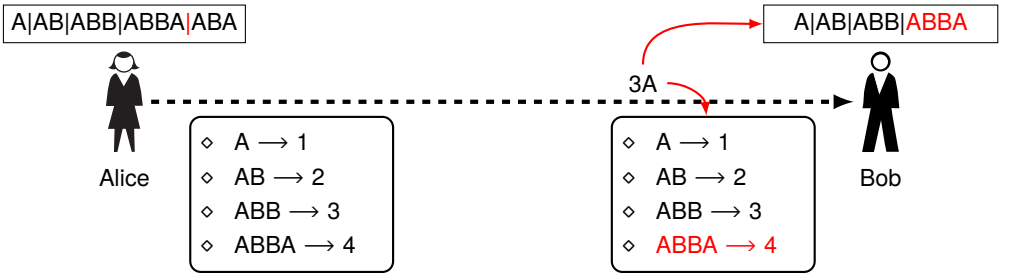
▷ lire tant qu'on trouve **une séquence déjà connue**, puis lire un **caractère supplémentaire** :



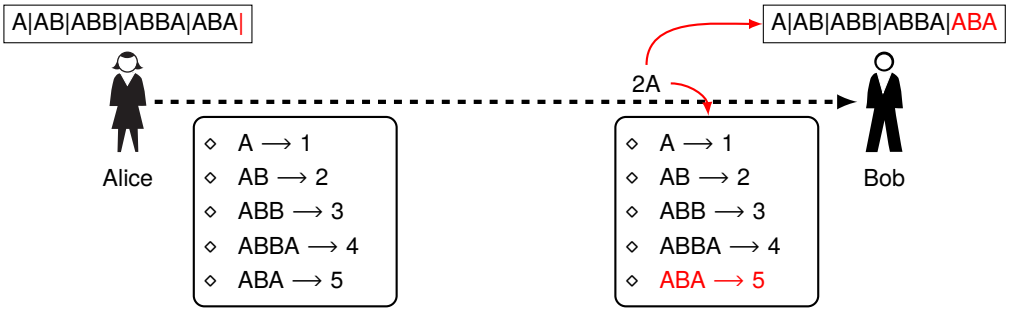
▷ lire tant qu'on trouve une **séquence déjà connue**, puis lire un **caractère supplémentaire** :



▷ lire tant qu'on trouve une **séquence déjà connue**, puis lire un **caractère supplémentaire** :



▷ lire tant qu'on trouve une **séquence déjà connue**, puis lire un **caractère supplémentaire** :



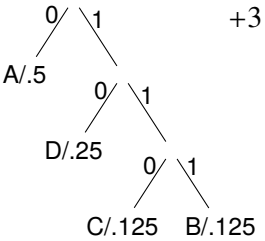
Entropie : $H = \sum_{i=1}^{n} p_i * \log_2(\frac{1}{p_i}) = -\sum_{i=1}^{n} p_i * \log_2(p_i)$ où p est la probabilité d'apparition d'un symbole et $\log_2(\frac{1}{p_i})$ est la **taille de la séquence binaire** correspondant au codage du symbole obtenu de **manière optimale** par rapport à sa **probabilité d'apparition**.

Exemple :

- l'entropie maximale de données codées sur des octets est proche de 8.
- un texte **littéraire** possède une **faible entropie** (redondance des mots dans les phrases) ;
- la compression optimale de Huffman exploite l'entropie.

$S =$ "$AAAABCDD$", soient $8 * 8 = 64$bits

$H = 1 * .5 + 2 * .25 + 3 * .125$
$+3 * .125 = 1.75$

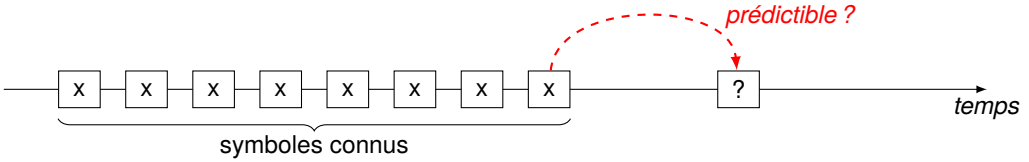


$A \rightarrow 0\ B \rightarrow 111$
$C \rightarrow 110\ D \rightarrow 10$

Compression:
4*1+1*3+1*3+2*2=14bits
ou $8 * H = 8 * 1.75 = 14$

arbre de Huffman

L'entropie mesure la **prédictabilité** d'un symbole connaissant les symboles précédents :



Si les symboles sont **prédictibles** :

▷ il existe des **séquences** qui se **reproduisent** dans les données : de la **redondance**
⟹Il est possible de les **trouver** à l'aide d'une compression par dictionnaire et de les **compresser** ;

▷ l'entropie est **inférieure** à la **mesure d'information** optimale $h = n\log(s) = log(s^n)$ quand n vaut un, c-à-d $\log_2(s)$;

Si les symboles sont **imprédictibles** :

▷ il n'y a **pas de redondance** dans les données
⟹ elles ne sont **pas compressables**.

▷ la mesure d'entropie est **forte** : elle est proche de $h = \log_2(s)$, (pour des octets : proche de $h = 8$) ;

▷ les données présentent des qualités de l'aléas ⟹ utilisation comme **source d'aléas pour la cryptographie**.

Utilisation de la commande `ent`

xterm

La valeur calculée précédemmment

```
$ echo -n 'AAAABCDD' | ent
Entropy = 1.750000 bits per byte.

Optimum compression would reduce the size
of this 8 byte file by 78 percent.

Chi square distribution for 8 samples is 696.00, and randomly
would exceed this value less than 0.01 percent of the times.

Arithmetic mean value of data bytes is 66.1250 (127.5 = random).
Monte Carlo value for Pi is 4.000000000 (error 27.32 percent).
Serial correlation coefficient is 0.533981 (totally uncorrelated = 0.0).
```

Sur une séquence de caractère 0

xterm

Entropie très mauvaise : tout est prédictible !

super compressable parce que super redondant !

```
$ python3 -c "print('0'*100)" | ent
Entropy = 0.080136 bits per byte.

Optimum compression would reduce the size
of this 101 byte file by 98 percent.

Chi square distribution for 101 samples is 25248.07, and randomly
would exceed this value less than 0.01 percent of the times.

Arithmetic mean value of data bytes is 47.6238 (127.5 = random).
Monte Carlo value for Pi is 4.000000000 (error 27.32 percent).
Serial correlation coefficient is -0.010000 (totally uncorrelated = 0.0).
```

Mesurer l'entropie sous Linux

Sur un texte en français : «*20000 lieues sous les mers*» de Jules Vernes

xterm

```
$ wget https://www.gutenberg.org/cache/epub/5097/pg5097.txt
$ ent pg5097.txt
Entropy = 4.637142 bits per byte.

Optimum compression would reduce the size
of this 942654 byte file by 42 percent.

Chi square distribution for 942654 samples is 13540603.37, and randomly
would exceed this value less than 0.01 percent of the times.

Arithmetic mean value of data bytes is 93.5540 (127.5 = random).
Monte Carlo value for Pi is 3.994322413 (error 27.14 percent).
Serial correlation coefficient is 0.206601 (totally uncorrelated = 0.0).
```

Entropie moyenne...texte littéraire

Sur des valeurs dont l'aléas est garanti

xterm

```
$ head -c 1M /dev/urandom > random_values
$ ent random_values
Entropy = 7.999806 bits per byte.

Optimum compression would reduce the size
of this 1048576 byte file by 0 percent.

Chi square distribution for 1048576 samples is 280.76, and randomly
would exceed this value 12.84 percent of the times.

Arithmetic mean value of data bytes is 127.4810 (127.5 = random).
Monte Carlo value for Pi is 3.137386846 (error 0.13 percent).
Serial correlation coefficient is 0.000769 (totally uncorrelated = 0.0).
```

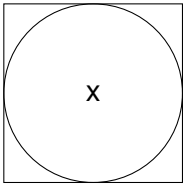
La compression est...nulle ! $\Longrightarrow$ pas de redondance !

C'est quoi cette méthode de «Monte-Carlo» *?*

Le device Linux `/dev/urandom` *fournit des valeurs dont l'aléas est garanti.*

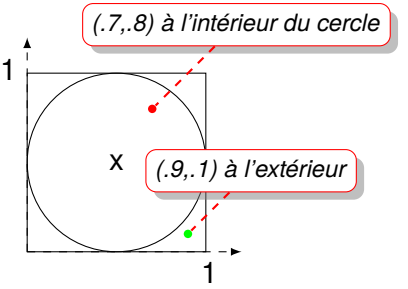
Méthode de Monte-Carlo pour déterminer la valeur de π

Soit un cercle de rayon $r = 0.5$, sa surface est de $\pi * r^2 = \frac{\pi}{4}$:



Ce cercle est inclus dans un carré de côté 1.
La surface du carré est égale à : $(2 * r)^2 = 1 * 1 = 1$.
Le rapport entre la surface du cercle et celle du carré est de $\frac{\pi}{4}$

On choisit au **hasard** des points (x, y) à l'intérieur du carré, c-à-d compris entre $(0, 0)$ et $(1, 1)$:



Pour un point $p = (x, y)$:

▷ p est dans le cercle si la distance entre le centre $(0.5, 0.5)$ du cercle et p est inférieure ou égale à 0.5, c-à-d $\sqrt{(x - .5)^2 + (y - .5)^2} \leq 0.5$

Pour chaque **tirage aléatoire** d'un point p :

▷ on ajoute 1 au nombre total T de points tirés au hasard ;

▷ si le point est à l'intérieur du cercle, alors on incrémente le nombre C de points à l'intérieur du cercle ;

La **probabilité** pour qu'un point choisi **au hasard** appartienne au cercle est $\frac{\text{aire cercle}}{\text{aire carré}} = \frac{\pi r^2}{4r^2} = \frac{\pi}{4}$

Si les points p sont choisis de **manière aléatoire** (distribution uniforme) alors le rapport entre C est T est une **approximation** de notre probabilité, qui tend vers $\frac{\pi}{4}$. Ce qui donne : $\pi = 4\frac{C}{T}$

$\Longrightarrow$Connaissant la valeur de π, on peut juger de **la qualité d'une séquence aléatoire** par l'approximation de π qu'elle fournit (10^6 points donne une approximation correcte de π sur deux décimales).

Et sur un chiffré ?

xterm

```
$ python3 -c 'print("0"*10000)' | openssl enc -aes-128-cbc -e -K 00 -iv 00  | ent
hex string is too short, padding with zero bytes to length
hex string is too short, padding with zero bytes to length
Entropy = 7.982667 bits per byte.

Optimum compression would reduce the size
of this 10016 byte file by 0 percent.

Chi square distribution for 10016 samples is 240.36, and randomly
would exceed this value 73.62 percent of the times.

Arithmetic mean value of data bytes is 127.0628 (127.5 = random).
Monte Carlo value for Pi is 3.209107250 (error 2.15 percent).
Serial correlation coefficient is 0.008944 (totally uncorrelated = 0.0).
```

clé à 00...00

IV à 00...00

$\Longrightarrow$Le chiffrement produit de l'aléas !

xterm

```
python3 -c 'print("0"*1000000)' | openssl enc -aes-128-cbc -e -K 00 -iv 00  | ent
hex string is too short, padding with zero bytes to length
hex string is too short, padding with zero bytes to length
Entropy = 7.999813 bits per byte.

Optimum compression would reduce the size
of this 1000016 byte file by 0 percent.

Chi square distribution for 1000016 samples is 258.70, and randomly
would exceed this value 42.36 percent of the times.

Arithmetic mean value of data bytes is 127.4386 (127.5 = random).
Monte Carlo value for Pi is 3.146331951 (error 0.15 percent).
Serial correlation coefficient is 0.000601 (totally uncorrelated = 0.0).
```

Avec un million de caractères, l'approximation de π devient correcte pour deux décimales...

Utilisation de l'entropie pour retrouver un élément cryptographique

Soit le programme suivant réalisant du chiffrement AES 256, à l'aide de la bibliothèque openSSL :

```
 1 #include <openssl/conf.h>
 2 #include <sys/types.h>
 3 #include <unistd.h>
 4 #include <openssl/evp.h>
 5 #include <openssl/err.h>
 6 #include <string.h>
 7
 8
 9 void handleErrors(void)
10 {
11     ERR_print_errors_fp(stderr);
12     abort();
13 }
14
15 int encrypt(unsigned char *plaintext, int plaintext_len, unsigned char *key,
16             unsigned char *iv, unsigned char *ciphertext)
17 {
18     EVP_CIPHER_CTX *ctx;
19     int len;
20     int ciphertext_len;
21
22     if(!(ctx = EVP_CIPHER_CTX_new())) handleErrors();
23     if(1 != EVP_EncryptInit_ex(ctx, EVP_aes_256_cbc(), NULL, key, iv)) handleErrors();
24
25     if(1 != EVP_EncryptUpdate(ctx, ciphertext, &len, plaintext, plaintext_len)) handleErrors();
26     ciphertext_len = len;
27
28     if(1 != EVP_EncryptFinal_ex(ctx, ciphertext + len, &len)) handleErrors();
29     ciphertext_len += len;
30
31     EVP_CIPHER_CTX_free(ctx);
32
33     return ciphertext_len;
34 }
```

Chiffrement en AES 256 CBC

```
36 int main (void)
37 {
38     /* A 256 bit key */
39     unsigned char *key = (unsigned char *)"01234567890123456789012345678901";
40
41     /* A 128 bit IV */
42     unsigned char *iv = (unsigned char *)"0123456789012345";
43
44     /* Message to be encrypted */
45     unsigned char *plaintext = (unsigned char *)"The quick brown fox jumps over the lazy dog";
46
47     unsigned char ciphertext[128];
48
49     int ciphertext_len;
50
51     ciphertext_len = encrypt (plaintext, strlen ((char *)plaintext), key, iv, ciphertext);
52
53     printf("Ciphertext is:\n");
54     BIO_dump_fp (stdout, (const char *)ciphertext, ciphertext_len);
55
56    printf("PID %d\n",getpid());
57    getchar();
58 }
```

Une clé sur 256 bits, 32 octets.

En mode CBC on a besoin d'un IV

le clair à chiffrer

obtenir le PID

on attends de presser une touche pour quitter

Le Makefile :

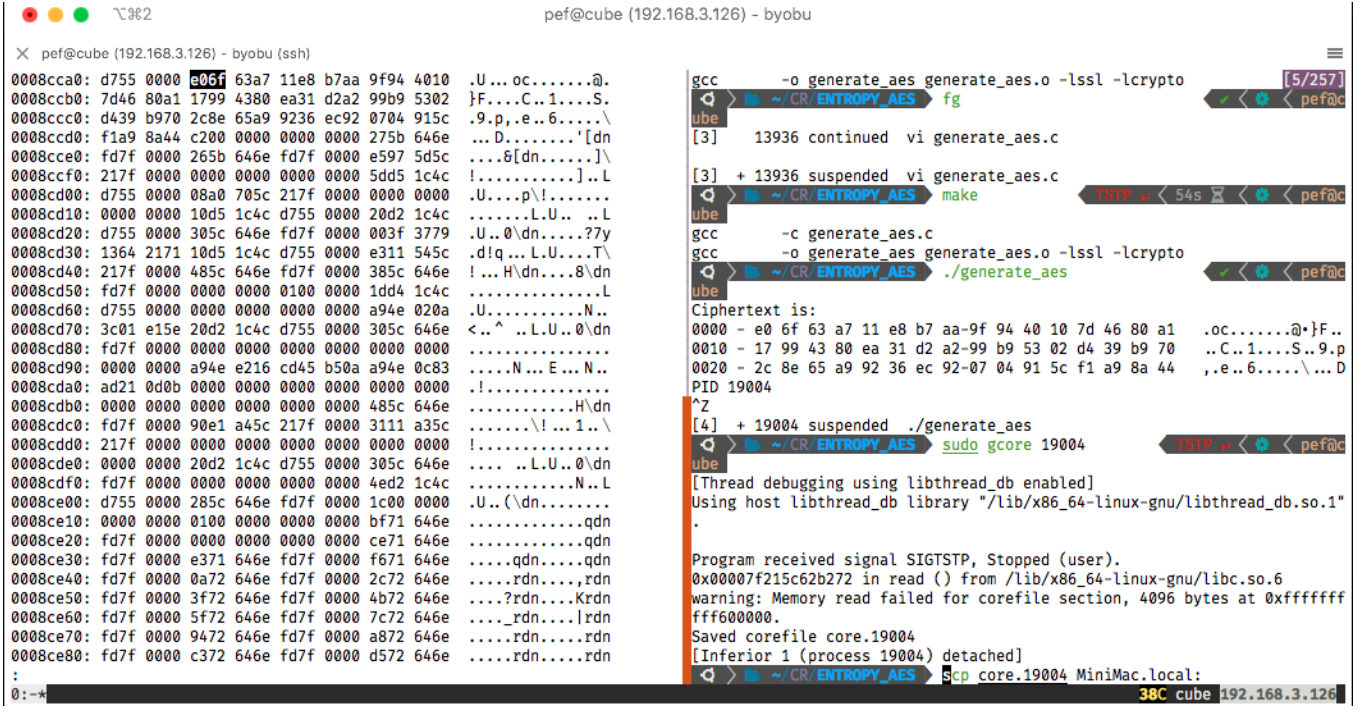
```
CC=gcc
LDFLAGS=-lssl -lcrypto
PROG=generate_aes
OBJS=generate_aes.o

.c.o:
   $(CC) -c $<

all: program

program: $(OBJS)
   $(CC) -o $(PROG) $^ $(LDFLAGS)
```

On exécute le programme et on crée un «*coredump*», c-à-d une copie de son espace mémoire :

```
xterm
make
./generate_aes &
sudo gcore 19004
```

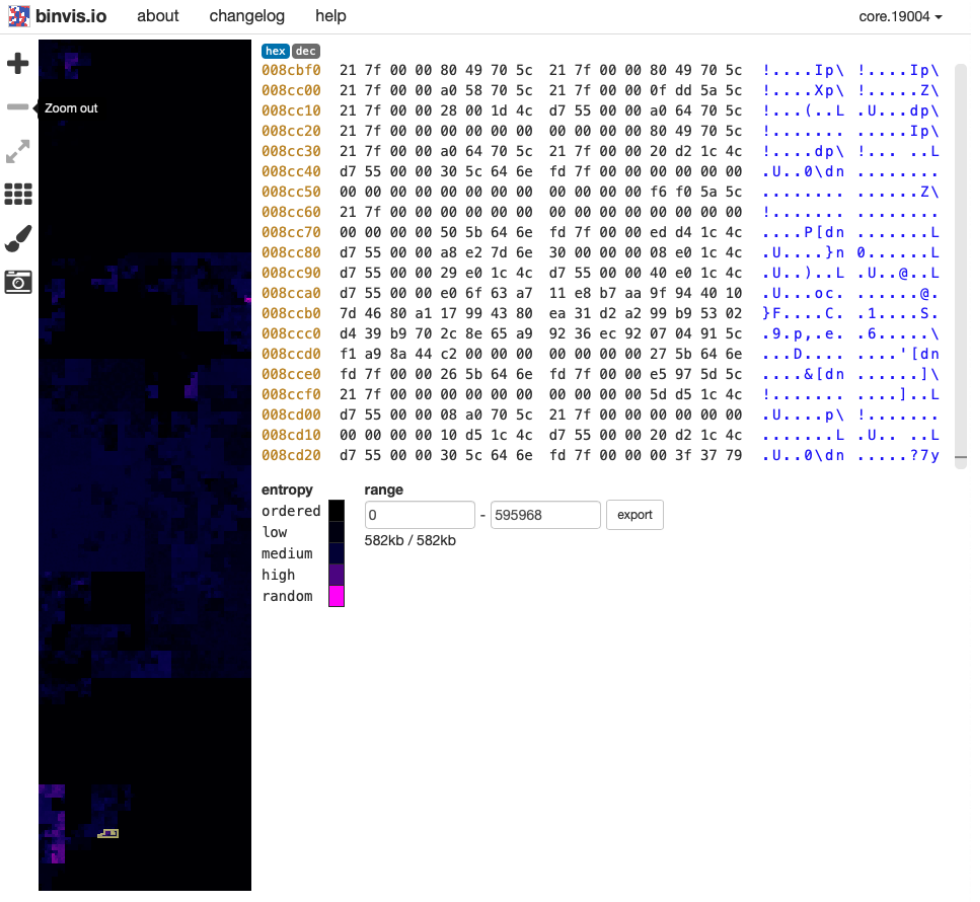
le PID affiché par le programme

À gauche est affiché le contenu du «*coredump*» avec la commande «`xxd`», où on recherche la clé affichée par le programme :



Elle est localisée dans le «*coredump*» vers le décalage `0x8cca0`.

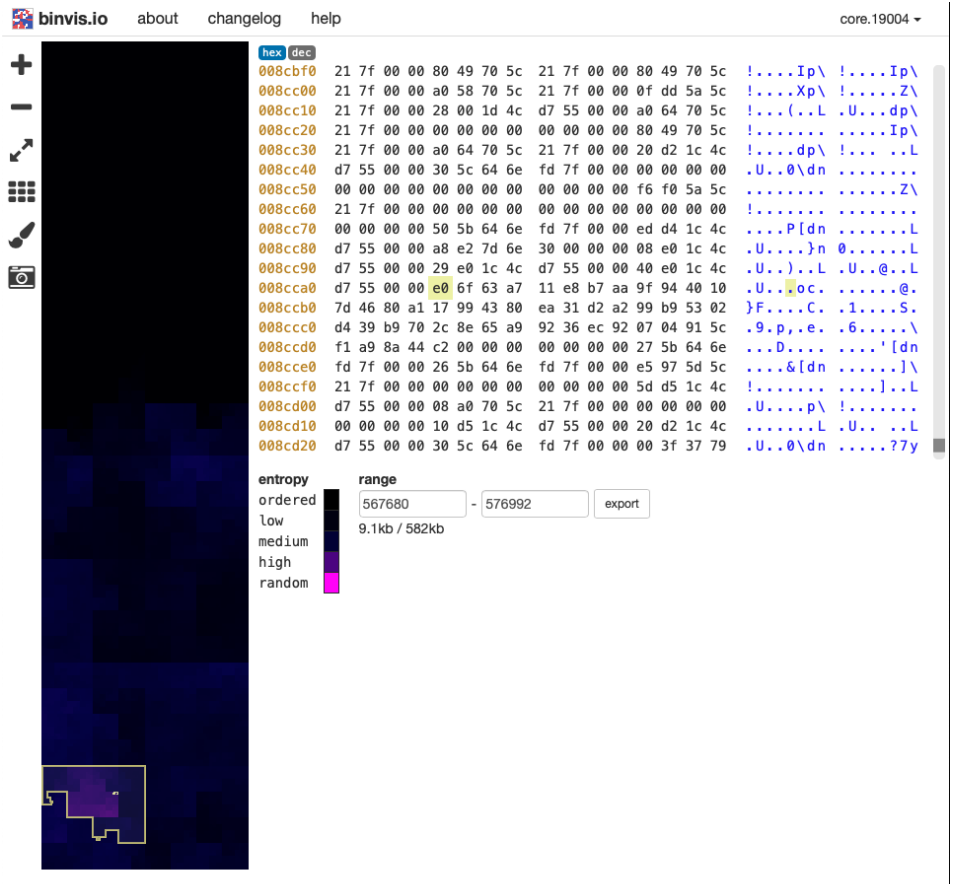
Utilisation de l'entropie pour retrouver un élément cryptographique



Le site **`http://binvis.io/`** permet :

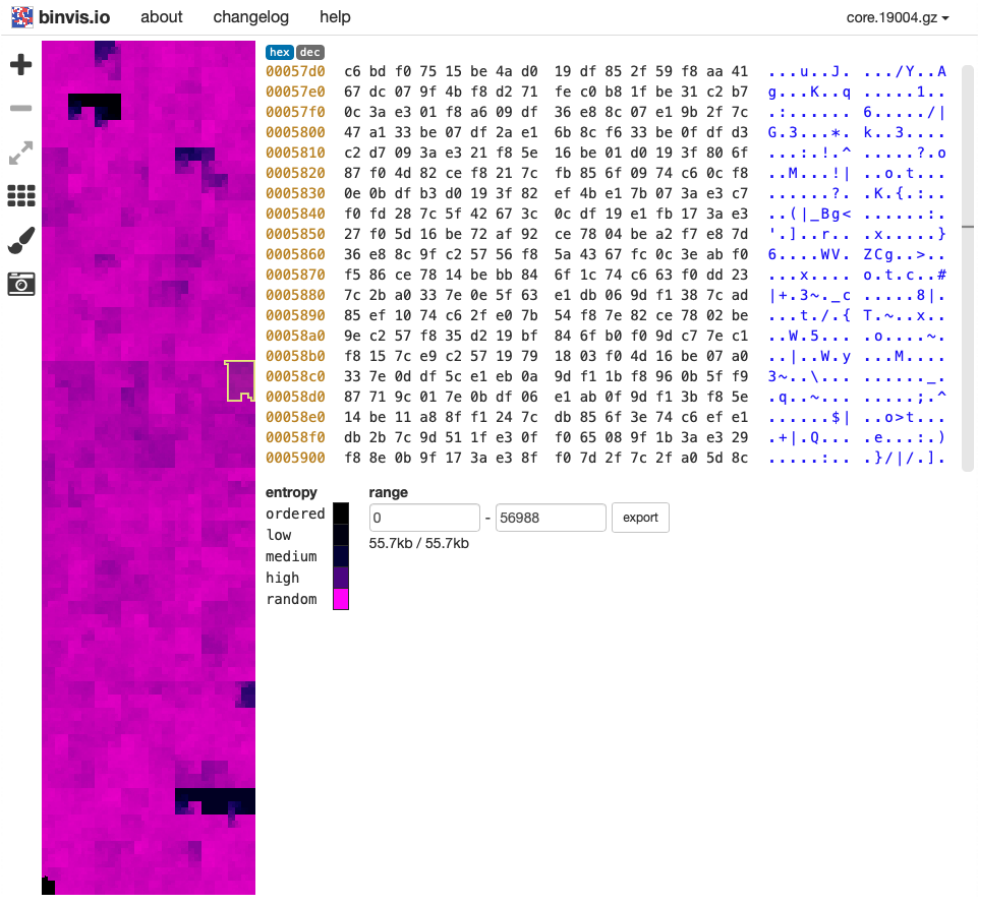
- ▷ de lire un contenu binaire, ici le «*coredump*» du programme `generate_aes` ;
- ▷ d'afficher graphiquement la mesure d'entropie suivant une intensité de rose ;
- ▷ de se déplacer de manière interactive dans le graphique en synchronisation avec un affichage hexadécimal du contenu.

Si on zoom :



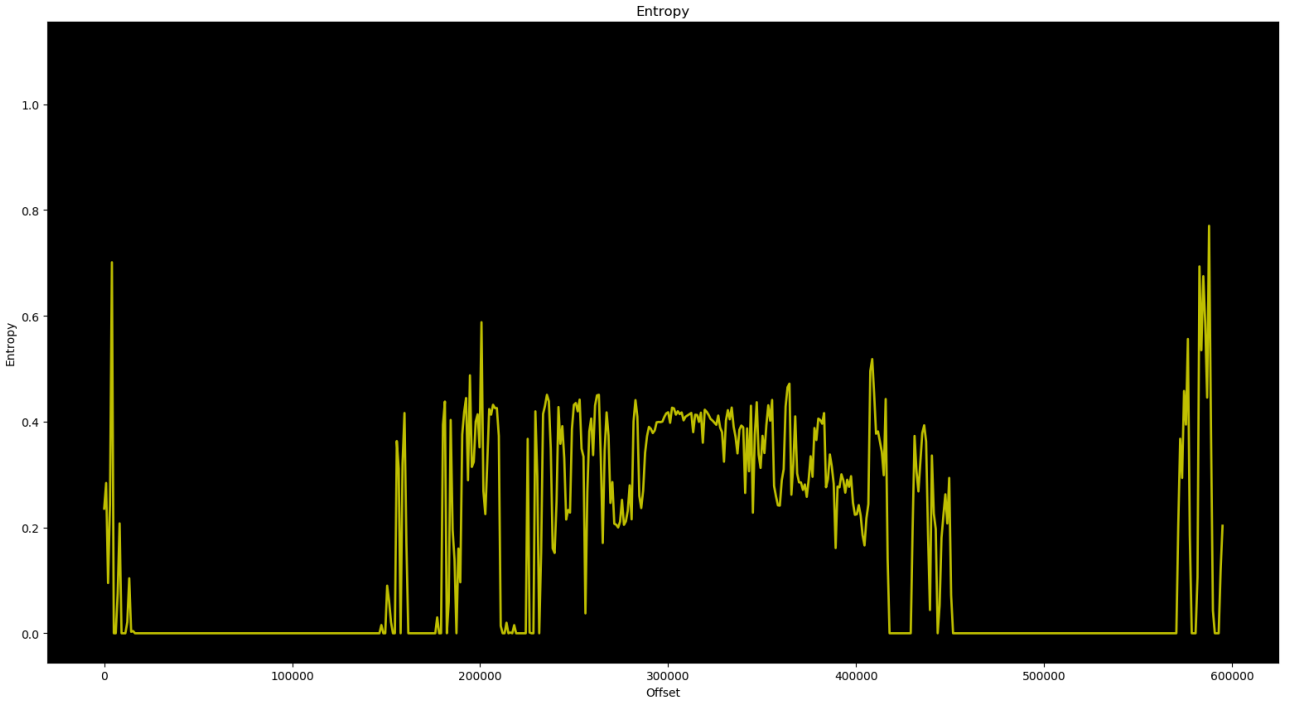
On voit que la mesure d'entropie permet de cerner rapidement les endroits où le chiffrement AES se trouve. Le «*coredump*» a une taille de `595968` octets.

Juste pour s'amuser, si on essai sur une version «*gzipée*» en compression Ziv-Lempel :

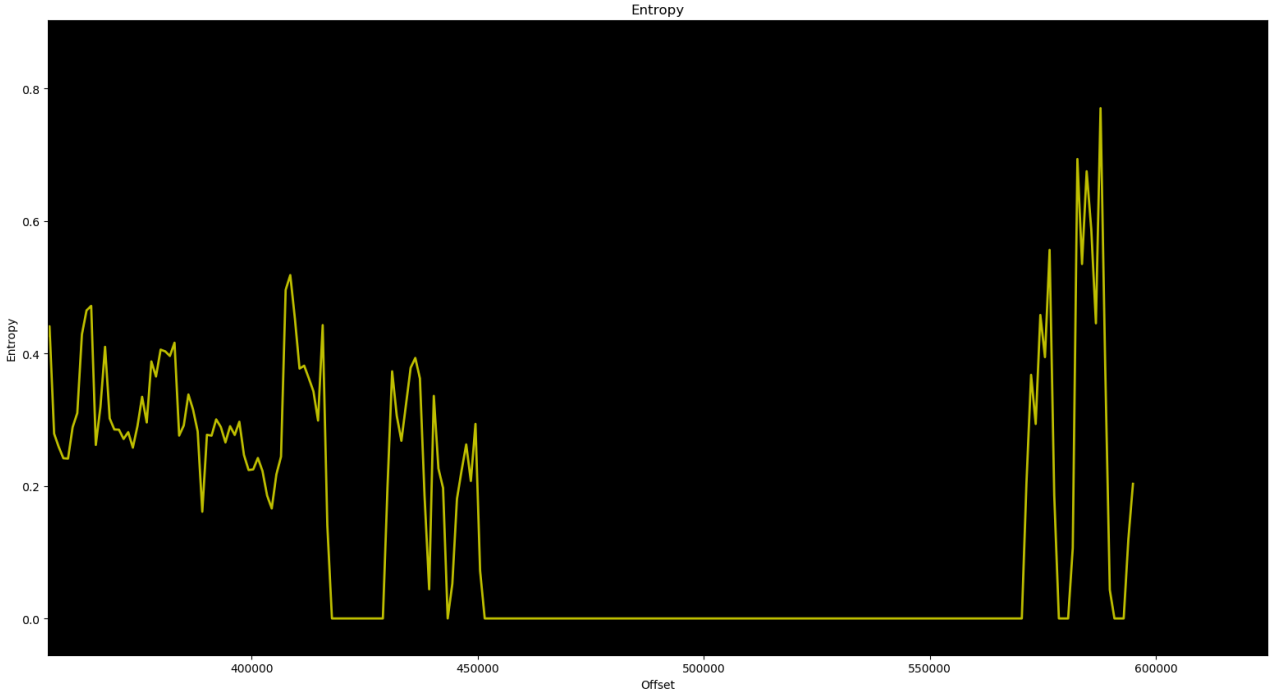


On peut également utiliser l'outil «`binwalk`» :

```
xterm
$ binwalk -E core.19004
```



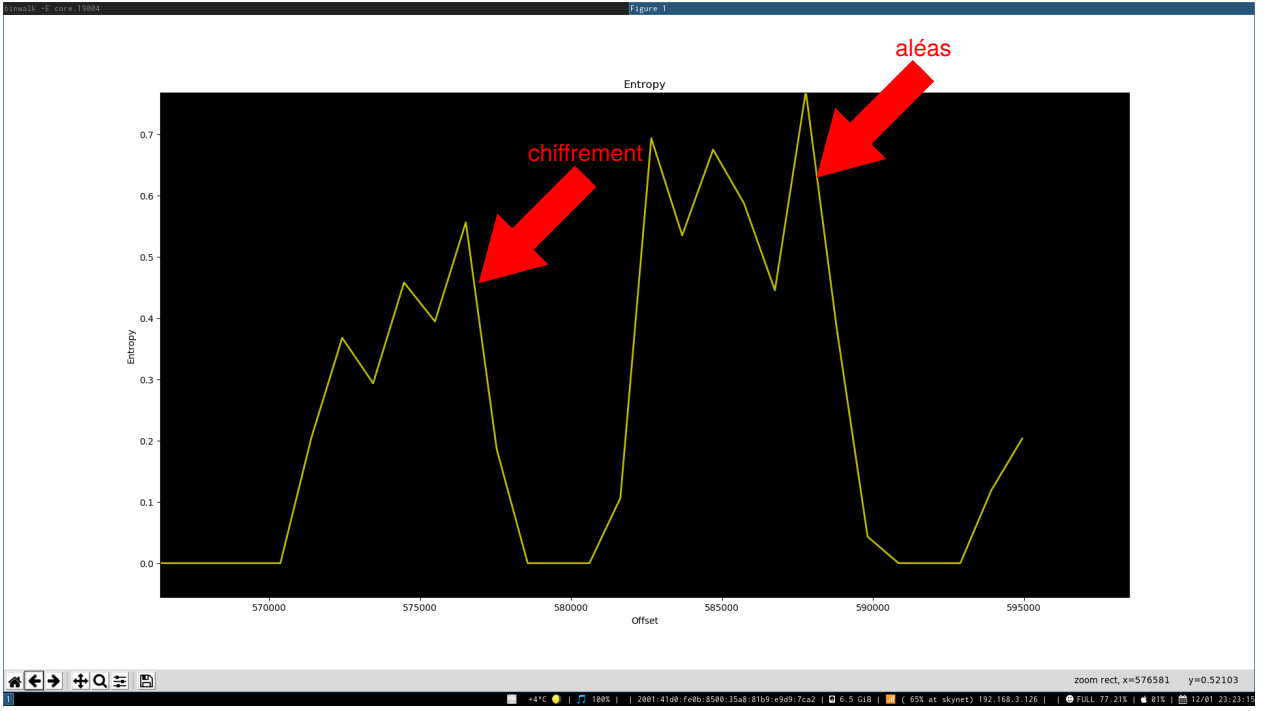
On obtient un graphe de la mesure d'entropie afin d'isoler les endroits où elle est élevée.



On zoom...

L'adresse des données chiffrées est `0x8cca0`, ce qui donne en décimal :

```
xterm
$ printf "%d\n" 0x8cca0
576672
```



Encore un zoom, et on est sur la zone où se trouve le chiffrement AES avec un pic d'entropie vers 576672.

CFB : Cipher Feedback

Le mode qui semble éviter tous les problèmes est le CFB :

- l'opération XOR est appliquée entre le bloc de texte clair et le résultat précédent chiffré à nouveau par la fonction de chiffrement ;
- il offre une grande sécurité.

Pour le premier bloc de texte clair, on génère un vecteur d'initialisation.

- T[n] = nième bloc de texte clair.
- I[n] = nième bloc temporaire
- C[n] = nième bloc de texte chiffré.
- E(m) = fonction de chiffrement du bloc m
- VI = vecteur d'initialisation

Chiffrement

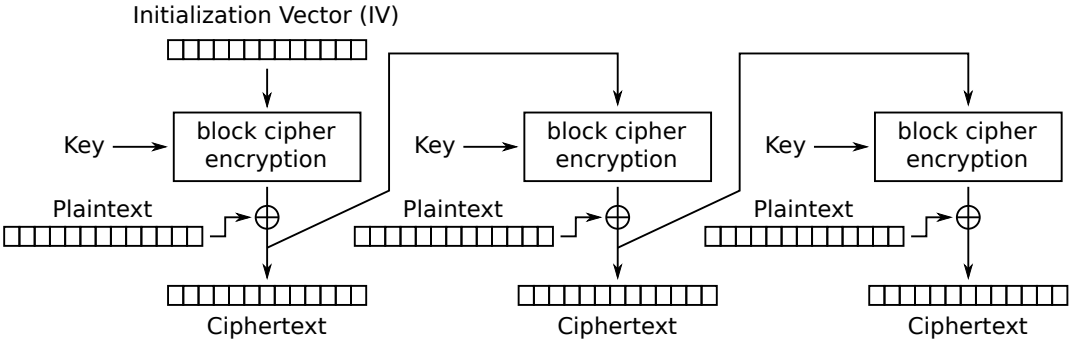
I[0] = VI
I[n] = C[n-1] , si (n > 0)
C[n] = T[n] xor E(I[n])

Déchiffrement

I[0] = VI
I[n] = C[n-1], si (n > 0)
T[n] = C[n] xor E(I[n])

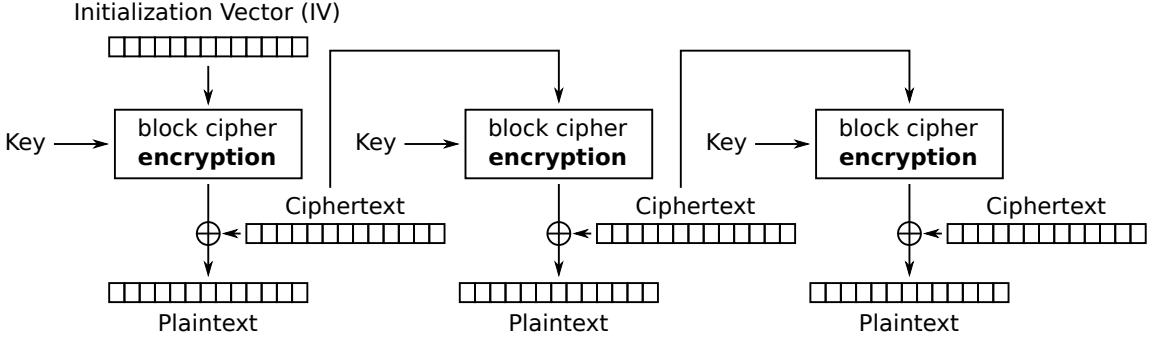
Pour le **chiffrement** et le **déchiffrement** on utilise la **même fonction E(m)**.
Cela permet d'utiliser que la fonction de chiffrement qui peut être **plus rapide** que la fonction de déchiffrement.

Chiffrement



Cipher Feedback (CFB) mode encryption

Déchiffrement



Cipher Feedback (CFB) mode decryption

OFB : Output Feedback

Le mode OFB est une solution aux deux problèmes relatifs au mode ECB :

- ▷ au départ un vecteur d'initialisation est généré ;
- ▷ ce bloc est chiffré successivement et chacun des résultats est utilisé dans l'application de l'opération XOR avec un bloc de texte en clair.

Le vecteur d'initialisation est envoyé tel quel avec le message chiffré.

- □ T[n] = nième bloc de texte en clair.
- □ C[n] = nième bloc de texte chiffré.
- □ E(m) = fonction de chiffrement du bloc m
- □ I[n] = nième bloc temporaire
- □ R[n] = nième bloc temporaire second
- □ VI = vecteur d'initialisation

Chiffrement

I[0] = VI
I[n] = R[n-1] , si (n > 0)
R[n] = E(I[n])
C[n] = T[n] xor R[n]

Déchiffrement

I[0] = VI
I[n] = R[n-1] , si (n > 0)
R[n] = E(I[n]) T[n] = C[n] ^ R[n]
T et C sont d'une longueur fixe

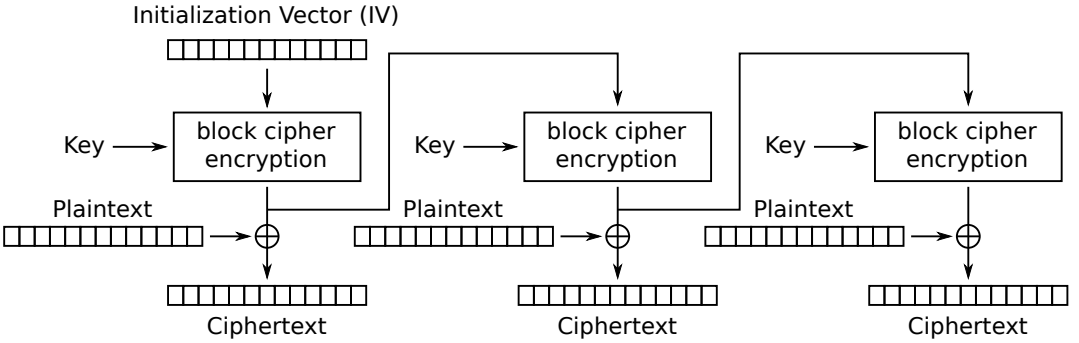
Problèmes :

- ▷ le texte en clair est **seulement** soumis à un XOR.
  Si le **texte clair est connu**, un tout autre texte en clair peut être **substitué** en inversant les bits du texte chiffré de la même manière qu'inverser les bits du texte clair : ***bit-flipping attack***.
- ▷ il existe une possibilité qu'une clé et un vecteur d'initialisation soient choisis tels que les blocs successifs générés puissent se **répéter** sur une courte boucle.

Le mode OFB est souvent utilisé comme PRNG, «*pseudo-random number generator*», c-à-d un pseudo-générateur de nombre aléatoire : génération d'une séquence de valeurs aléatoires **reproductible**.

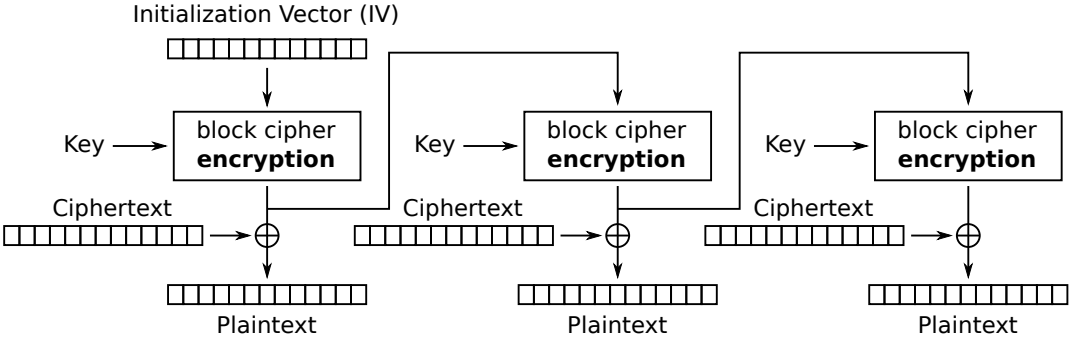
Chiffrement par bloc : le mode OFB

Chiffrement



Output Feedback (OFB) mode encryption

Déchiffrement



Output Feedback (OFB) mode decryption

XOR Calculator

Thanks for using the calculator. View help page.

I. Input: ASCII (base 256)
payer 600

II. Input: ASCII (base 256)
Séquence OFB → 123456789

Calculate XOR

III. Output: hexadecimal (base 16)
41534a514716010809

XOR Calculator

Thanks for using the calculator. View help page.

I. Input: ASCII (base 256)
6

II. Input: ASCII (base 256)
9

Calculate XOR

III. Output: hexadecimal (base 16)
de 6 à 9... → f

XOR Calculator

Thanks for using the calculator. View help page.

I. Input: hexadecimal (base 16)
41534a514716010809

II. Input: hexadecimal (base 16)
Attaque → f0000

Calculate XOR

III. Output: hexadecimal (base 16)
41534a5147160e0809

XOR Calculator

Thanks for using the calculator. View help page.

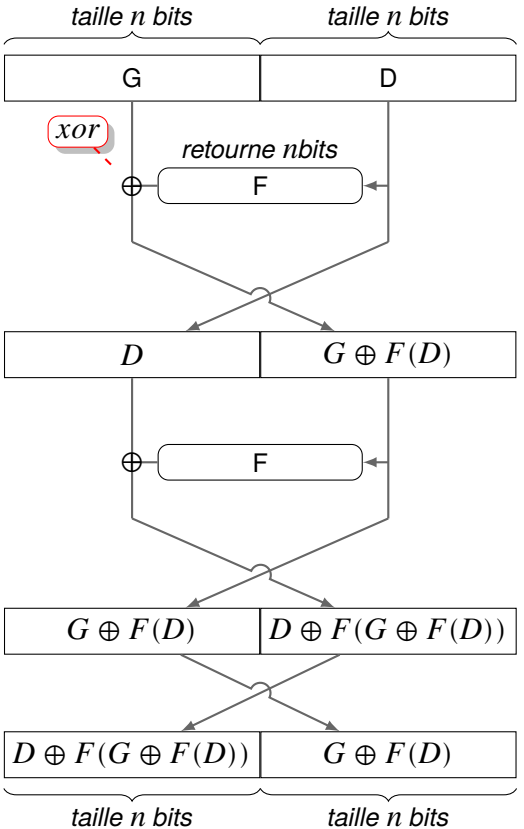
I. Input: hexadecimal (base 16)
41534a5147160e0809

II. Input: ASCII (base 256)
Déchiffrement → 123456789

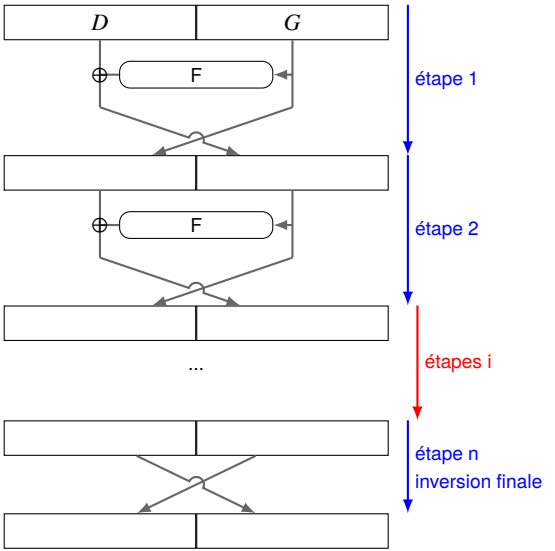
Calculate XOR

III. Output: ASCII (base 256)
Attaque réussie ! → payer 900

Chiffrement & Déchiffement

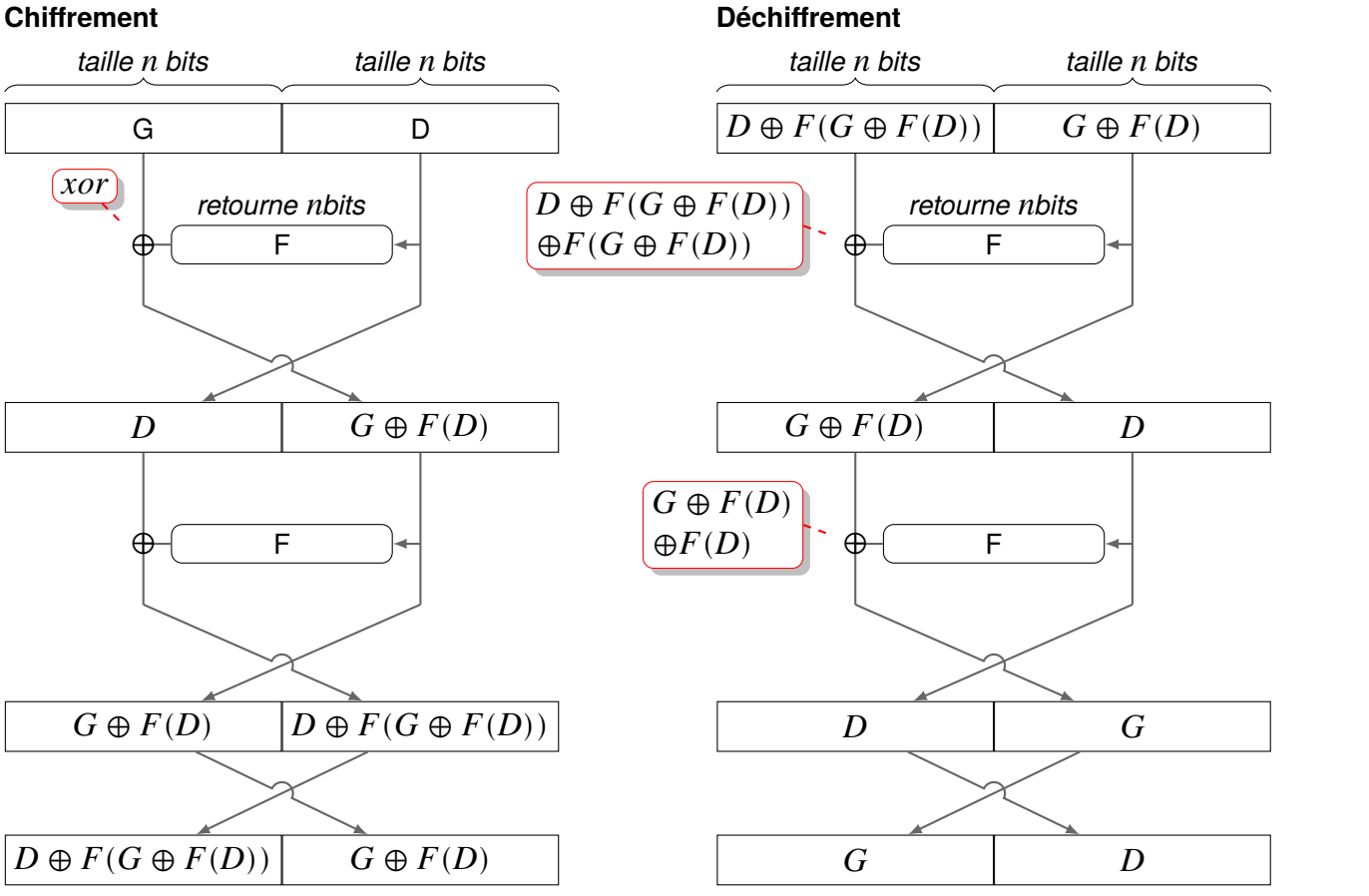


- D et G : séquences de bits de même taille n ;
- F : fonction entrée n bits et sortie n bits
  $F : \{n\ bits\} \mapsto \{n\ bits\}$;
- ◇ chiffrement : application du réseau de Feistel ;
  ◇ déchiffrement : application du **même réseau** de Feistel ;
- le réseau de Feistel est extensible :



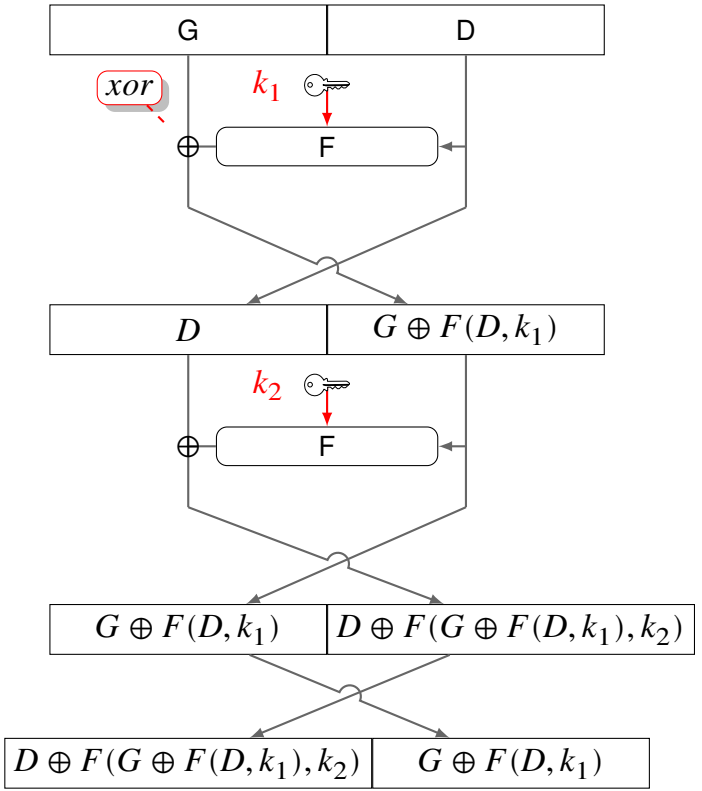
Chaque étape i est appelée «round» ou «tour».
Le réseau de Feistel sert de base au chiffrement DES.

Réseau de Feistel : fonctionnement du chiffrement/déchiffrement

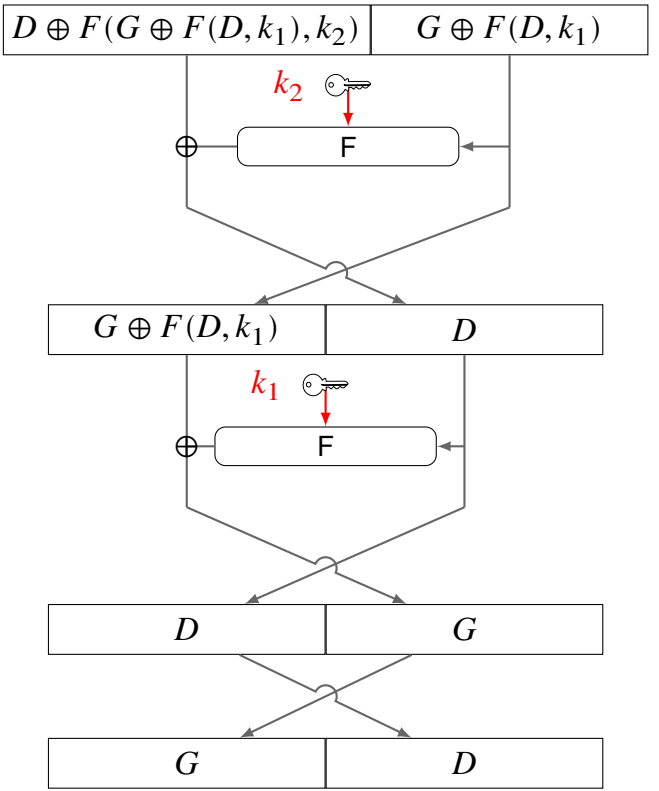


Réseau de Feistel : utilisation du chiffrement symétrique

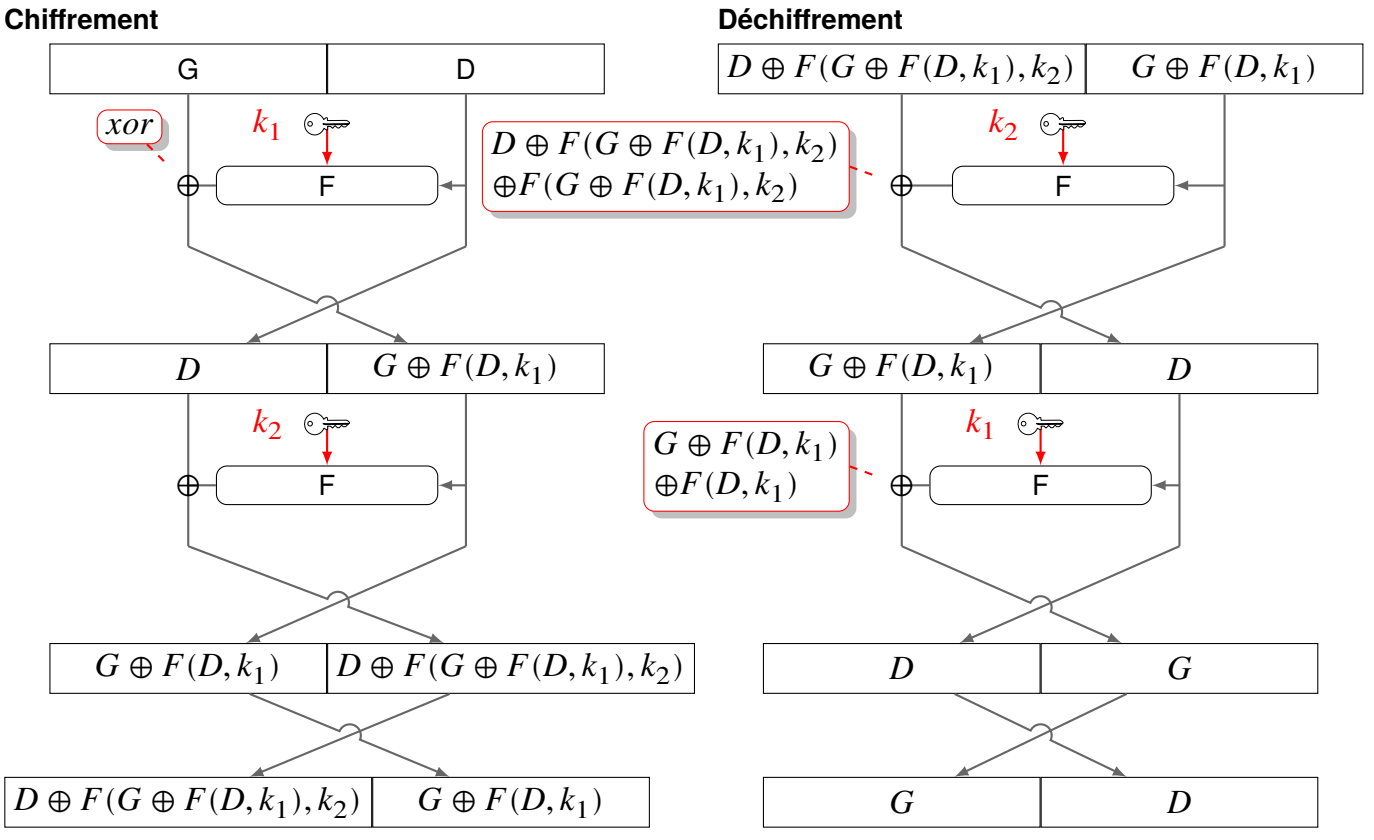
Chiffrement



Déchiffrement



Réseau de Feistel : utilisation du chiffrement symétrique



CTR : counter mode

Ce mode est utilisé lorsque l'on veut envoyer de manière chiffrée des **paquets de données autonomes** qui peuvent être perdus : un paquet perdu doit pouvoir être reconstruit et réémis.

C'est un mode utilisant :

- ▷ un compteur qui est mis à jour à chaque nouveau paquet émis : il peut être connu de tous ;
- ▷ un nonce, «*number used once*» : ce nombre peut être également connu de tous, il doit être unique et associé à une seule séquence de paquets ;
- ▷ un changement du nonce lorsqu'il est nécessaire de réinitialiser le compteur.

Il n'y a pas de lien direct entre le chiffrement d'un bloc et du bloc suivant : le lien est réalisé par une valeur de nonce et de clé de chiffrement commun.

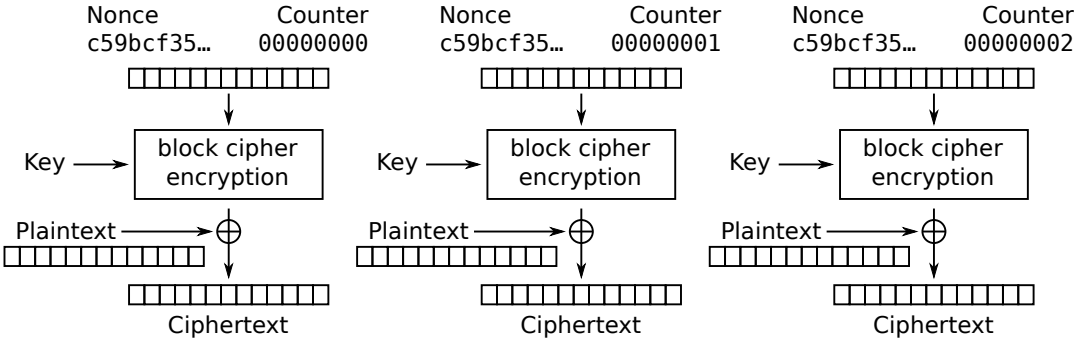
- ☐ T[n] = nième bloc de texte en clair.
- ☐ C[n] = nième bloc de texte chiffré.
- ☐ E(m) = fonction de chiffrement
- ☐ I[n] = nième bloc temporaire
- ☐ Nonce une valeur utilisée une seule fois (aléatoire)
- ☐ Cmpt numéro du paquet

Chiffrement
I[n] = E(Nonce,Cmpt)
C[n] = T[n] xor I[n]

Déchiffrement
I[n] = E(Nonce,Cmpt)
T[n] = C[n] xor I[n]
T et C sont d'une longueur fixe

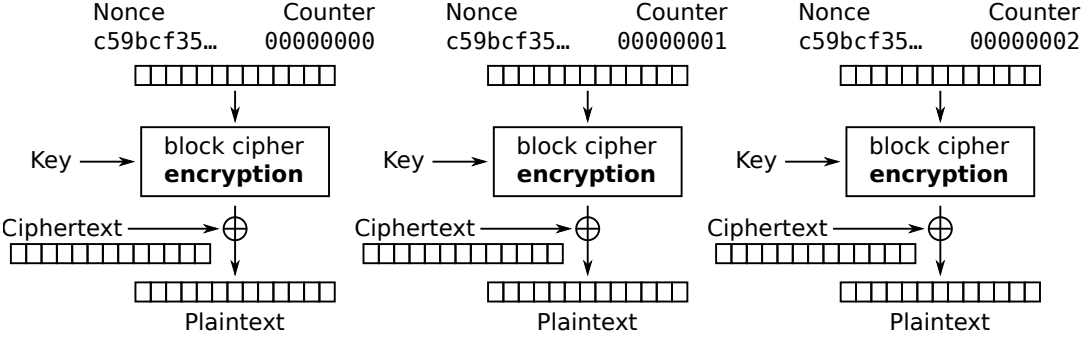
Chiffrement par bloc : le mode CTR

Chiffrement



Counter (CTR) mode encryption

Déchiffrement



Counter (CTR) mode decryption

Et le chiffrement «à la volée»
ou bien de données en continue
(comme un flux de communication) ?

Définition

Les algorithmes de chiffrement par flux, «*stream ciphers*», peuvent être vu comme des algorithmes de chiffrement par bloc où le bloc a une dimension unitaire (1 bit, 1 octet...) ou relativement petite.

Avantages :

- la méthode de chiffrement **peut être changée à chaque symbole** du texte clair ;
- ils sont **extrêmement rapides** ;
- ils ne **propagent pas les erreurs** (diffusion) dans un environnement où les erreurs sont fréquentes ;
- ils sont utilisables lorsque l'information ne peut être traitée qu'avec de **petites quantités de symboles à la fois** (par exemple si l'équipement n'a pas de mémoire physique ou une mémoire tampon très limitée).

Fonctionnement

Ils appliquent de simples transformations selon un *keystream* utilisé (opération xor par exemple).

Le keystream est une **séquence de bits** utilisée en tant que clé qui est **générée aléatoirement** par un algorithme (*keystream generator*) et dont les paramètres sont échangés de manière sécurisée.

Propriétés : Avec un keystream choisi **aléatoirement** et **utilisé qu'une seule fois**, le texte chiffré est **très sécurisé**.

La génération du keystream peut être :

- ▷ **indépendante** du texte en clair et du texte chiffré, appelée chiffrement de **flux synchrone**, «*synchronous stream cipher*» ;
- ▷ **dépendante**, «*self-synchronizing stream cipher*».

*Les chiffrements de flux les plus répandus sont **synchrones**.*

Algorithmes les plus connus

LFSR, «*Linear Feedback Shift Register*», rapide mais vulnérable à l'heure actuelle.

RC4, inventé par Ron Rivest en 87 (société RSA), utilisé dans le protocole SSL et Oracle Secure SQL.

SEAL, «*Software-optimized Encryption Algorithm*», Don Coppersmith et Phillip Rogaway en 93 (IBM), plus rapide que RC4.

Quels sont les «*stream-ciphers*» utilisés ?

En 2015, des chercheurs de l'Université de Louvain démontrent l'attaque NOMORE, «*Numerous Occurrence MOnitoring & Recovery Exploit*» sur RC-4 valable pour TLS et WPA-TKIP :

- ▷ l'attaque sur TLS permet en 75 heures de déchiffrer un cookie de sécurité ;
- ▷ l'attaque sur WPA-TKIP permet, au bout d'une heure, de déchiffrer et d'injecter des paquets dans le réseau.

⟹ RC-4 **disparait de TLS**, la RFC 7465 «*Prohibiting RC4 Cipher Suites*» l'interdit. TLS v1.3 utilise AES-GCM, AES-CCM et ChaCha20 combiné avec le MAC Poly1305 (RFC7539).

⟹ RC-4 **disparait** du WiFi, WPA utilise Counter Mode CBC-MAC, «*Counter Mode Cipher Block Chaining Message Authentication Code Protocol*» appelé également CCMP, «*CCM mode Protocol*» basé sur AES.

Et si on demande à un serveur Web ce qu'il supporte en TLS ?

xterm

```
depth=2 O = Digital Signature Trust Co., CN = DST Root CA X3
verify return:1
depth=1 C = US, O = Let's Encrypt, CN = Let's Encrypt Authority X3
verify return:1
depth=0 CN = p-fb.net
verify return:1
New, TLSv1/SSLv3, Cipher is ECDHE-RSA-AES256-GCM-SHA384
    Cipher    : ECDHE-RSA-AES256-GCM-SHA384
```

Le «*new kid on the block*»

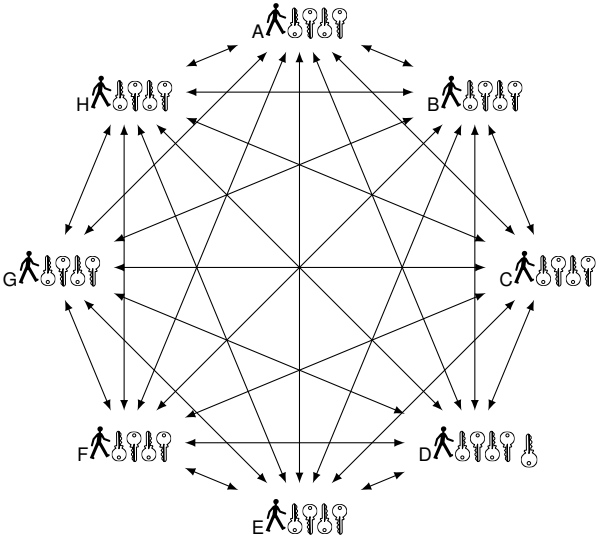
- ☐ utilisation de **AES-CTR** ;
- ☐ **ChaCha20** de Daniel J. Bernstein proposé en 2008 est un PRNG. Il est souvent associé à Poly1305 proposé en 2004.

Les limites du chiffrement symétrique

La multiplication des clés

Pour établir un canal de communication entre deux individus :

- Il faut qu'il soit chiffré avec une clé partagée entre les deux individus ;
- Il est ainsi confidentiel pour ceux qui ne possède pas la clé de chiffrement.



Pour que deux canaux de communications soient indépendants l'un de l'autre, c-à-d qu'une personne accède à l'un mais pas à l'autre, il faut que ces deux canaux utilisent des **clés différentes**.

Il est possible qu'un des interlocuteurs connaissent plusieurs clés utilisées dans différents canaux le reliant à des utilisateurs différents.

Exemple l'utilisateur D possède une clé pour chaque lien (avec H, G, F, E et C).

Problème comment échanger toutes ces clés ?

Pas d'intégrité et d'identification de l'auteur

Si Alice, Bob et Cédric partage le même lien de communication alors ils partagent la même clé de chiffrement symétrique.

Bob

clé secrète

Cédric

Alice

clé secrète

clé secrète

Message

message chiffré par Bob

message chiffré par Cédric

Message
(modifié par Cédric)

❶ Bob chiffre le message à destination d'Alice

❷ Cédric intercepte le message, le modifie, et le chiffre à nouveau avec la clé secrète

Problème

Chacun peut intercepter et modifier les messages qui s'échangent.

Et l’asymétrique alors ?

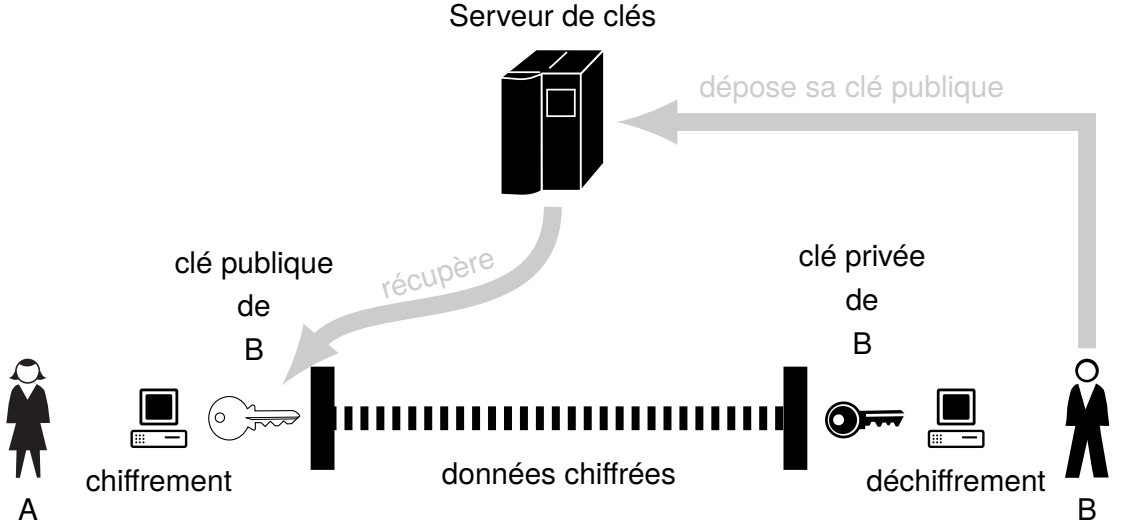
Chiffrement à clé asymétrique

Principe

Il utilise :

- une **clé publique** connue de tous ;
- une **clé privée** connue seulement du destinataire du cryptogramme.

Ces chiffrements à «clé publique» *ont été découverts par James Ellis (Angleterre) en 1969 et par Whitfield Diffie (Etats unis) en 1975.*



L'idée de la conception de tels algorithmes revient à Diffie et Hellman en 1976.

Échange par réseau

Un objectif de la cryptographie est de permettre à deux personnes, **Alice** et **Bob**, de communiquer au travers d'un canal peu sûr (téléphone, réseau informatique ou autre), sans qu'un opposant **Cédric**, puisse comprendre ce qui est échangé.

Scénario type

- Alice souhaite transmettre à Bob un ensemble de données (texte, nombres, …).
- Alice transforme ces informations par un **procédé de chiffrement** en utilisant la **clé publique** de Bob ;
- Alice envoi le texte chiffré au travers du canal de communication ;
- Cédric, *qui espionne peut-être le canal*, ne peut **reconstituer l'information**, contrairement à Bob qui dispose de la **clé privée** pour déchiffrer le cryptogramme.

Construction des clés

Les utilisateurs (A et B) choisissent une **clé aléatoire** dont ils sont seuls connaisseurs (il s'agit de la clé privée).

A partir de cette clé, ils **déduisent** chacun automatiquement par un algorithme la **clé publique**.

Les utilisateurs **s'échangent** cette clé publique au travers d'un canal **non sécurisé**.

Chiffrement d'un message

Lorsqu'un utilisateur désire **envoyer un message** à un autre utilisateur, il lui suffit de **chiffrer** le message à envoyer au moyen de la **clé publique** du destinataire (*qu'il trouvera par exemple dans un serveur de clés tel qu'un annuaire ou bien en signature d'un courrier électronique*).

Le destinataire sera en mesure de **déchiffrer** le message à l'aide de sa clé privée (**qu'il est seul à connaître**).

Rapports entre les clés

La recherche de la clé privée à partir de la clé publique revient à résoudre un **problème mathématique notoirement très compliqué**, c-à-d demandant un **grand nombre d'opérations** et beaucoup de mémoire pour effectuer les calculs $\Longrightarrow$ infaisable !

Par exemple dans RSA, l'algorithme le plus utilisé actuellement, la déduction de la clé privée à partir de la clé publique revient à résoudre un problème de factorisation de grand nombre que lequel travaille les mathématiciens depuis plus de 2000 ans !

Le choix des clés doit être fait de la manière la plus **imprédictible possible** : éviter les mots du dictionnaire, nombres **pseudo-aléatoires** à germe de génération difficile à deviner, *etc.*

Chiffrement à clé asymétrique

Il repose sur la connaissance d'une fonction mathématique **unidirectionnelle**, «*one-way function*», munie d'une **trappe** «*one-way trapdoor function*».

*Une fonction unidirectionnelle est une fonction $y = f(x)$ telle que, si l'on connaît la valeur y, on doit **chercher séquentiellement** la valeur x car il n'existe pas de fonction inverse directement calculable de la fonction f.*

On dit que cette fonction est munie d'une **trappe**, s'il existe une fonction $x = g(y, z)$ telle que l'on connaît z, il est facile de calculer x à partir de y.

Exemple de scénario d'échange

Bob veux recevoir des messages codés d'Alice, il souhaite que ces messages soient indéchiffrables pour Oscar qui a accès à leurs échanges :

- ▷ Bob et Alice connaissent la fonction unidirectionnelle f;
- ▷ Bob fournit à Alice sa «clé publique» c.

f et c peuvent être connus de tout le monde : ils sont connus d'Oscar.

Alice chiffre le message M en utilisant l'algorithme f et la clé c : ceci fournit un texte T chiffré ayant les apparences d'une séquence de valeurs choisies au hasard :

$$T = f(M, c) \quad .$$

Comme f est une fonction unidirectionnelle, Oscar est incapable de reconstituer le message même s'il connaît l'algorithme f, la clé publique c et le texte T.

Bob, lui, possède la «clé privée» z qui est absolument secrète.
z ouvre la trappe de la fonction f et permet de déchiffrer le message en appliquant la fonction g au triplet (T, c, z) :

$$M = g(T, c, z) \quad .$$

Bob peut lire le contenu du message envoyé par Alice !

Des clé et des cadenas

Alice :

- ▷ crée une **clé aléatoire** (la clé privée) ;
- ▷ puis fabrique un **grand nombre de cadenas** (clé publique) qu'elle met à disposition dans un casier accessible par tous (le casier joue le rôle de canal de communication non sécurisé).

Bob :

- ▷ prend un cadenas (ouvert) ;
- ▷ ferme une valise contenant le document qu'il souhaite envoyer à Alice ;

- ▷ envoi la valise à Alice, propriétaire de la clé publique (le cadenas).

Cette dernière pourra ouvrir le cadenas et la valise avec sa clé privée.

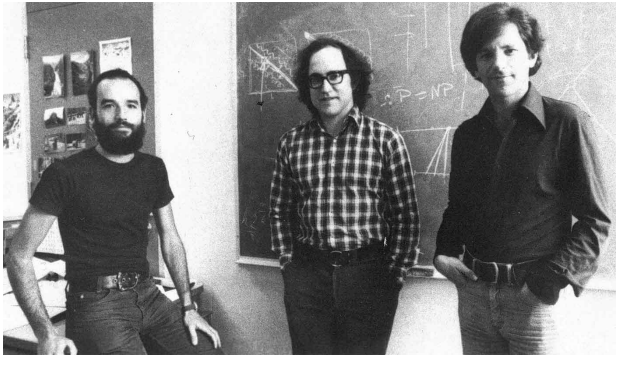
Les contraintes pour un tel algorithme

Il faut trouver un couple de fonctions f (fonction unidirectionnelle) et g (fonction de «*backdoor*») :
C'est un problème mathématique difficile !
Au départ, le système à clé publique n'a d'abord été qu'une idée dont la faisabilité restait à démontrer.

Des algorithmes ont été proposés par des mathématiciens

Un des premiers algorithmes proposé repose sur la **factorisation du produit de deux grands nombres entiers**.
Suivant la taille des nombres, cette factorisation peut demander un temps de calcul de plusieurs années : le problème est résolu !

Cet algorithme a été proposé par Rivest, Shamir et Adleman en 1977, ce qui a donné naissance à RSA.



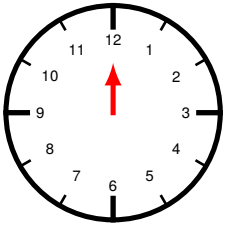
L'idée générale est la suivante :

- la fonction f est l'**exponentiation modulaire** ;
- la **clé publique** est la valeur c utilisée en combinaison avec le produit n de deux grands nombres entiers ;
- la **clé privée** est la valeur z ;
- g consiste en la **factorisation** de n.

Seul Bob, qui connaît z et g peut déchiffrer le message chiffré.

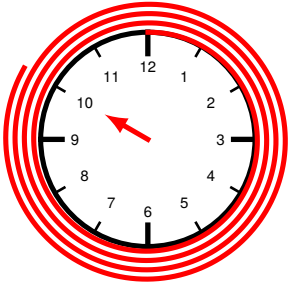
Les nombres modulaires : $x \bmod p$

Il est 0h ou 12h :



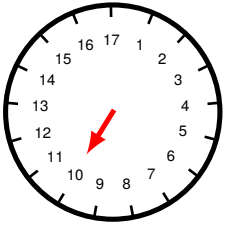
Et si on avance de 46h ?

on tourne de 46h autour du cadran...
On fait 3 tours : $3 * 12 = 36$
plus $46 - 36 = 10$ *h*



Ce qui fait $46 \bmod 12$? *10h !*

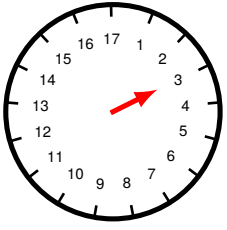
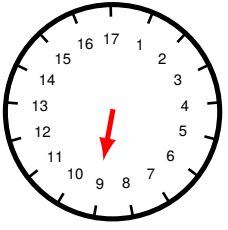
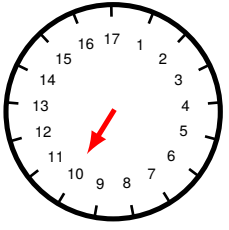
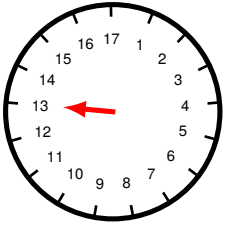
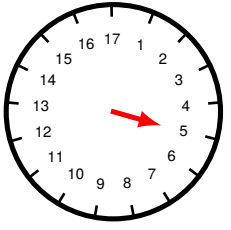
Et si on utilise un **nombre premier**, comme 17, à la place de 12 ?



et 3, la «*racine primitive*» modulo 17, c-à-d n'ayant pas de facteur en commun...

Les résultats de l'exponentiation modulaire :

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	...
$3^n \bmod 17$	3	9	10	13	5	15	11	16	14	8	7	4	12	2	6	1	...

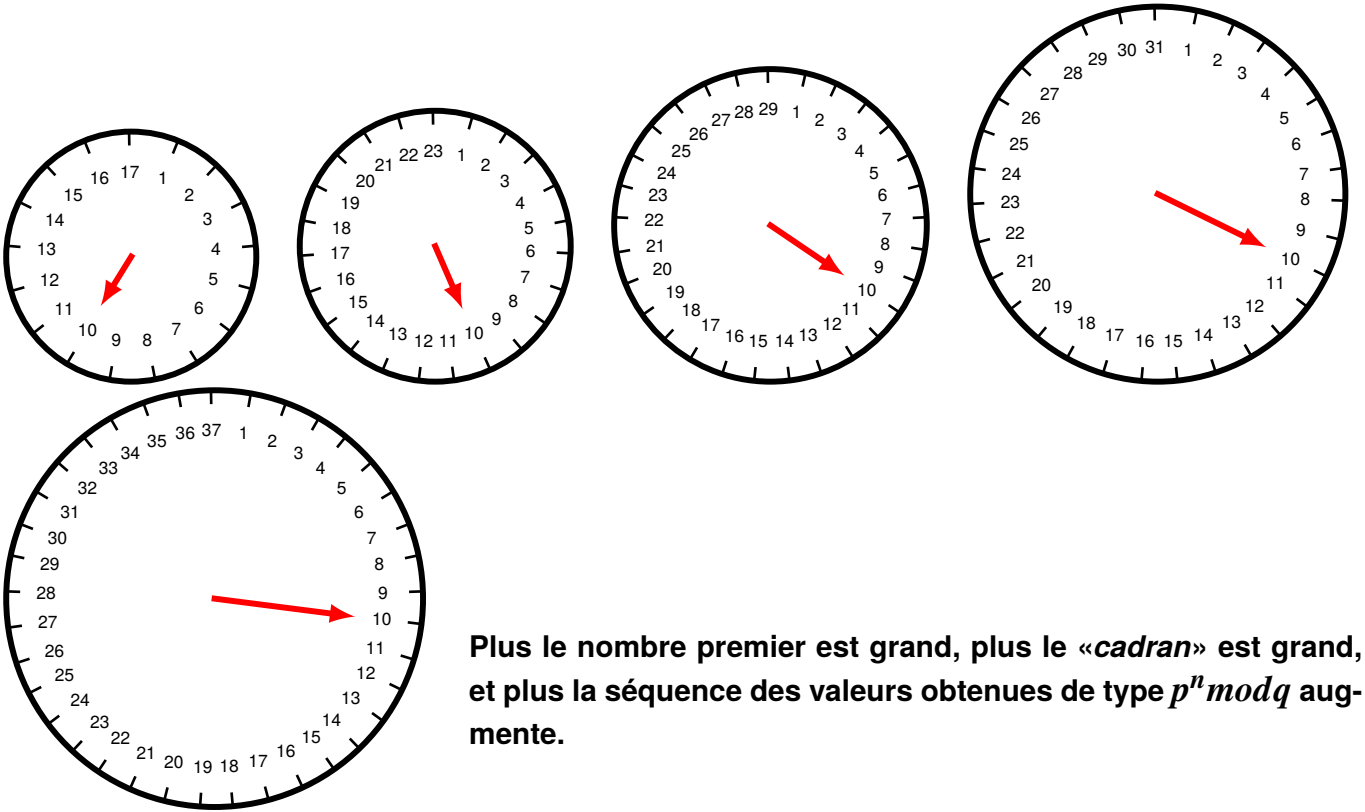






...

Les valeurs sont **également distribuées** autour du cadran...ce qui donne l'impression qu'elles sont **aléatoires**.
La **procédure inverse** qui permet de passer de 12, par exemple, à la valeur de x telle que : $3^x \bmod 17 = 12$ est **dure** !

$\Longrightarrow$ ***pourquoi ne pas s'en inspirer pour définir un crypto-système ?***

Pour différents nombres premiers : 17, 23, 29, 31, 37, ...

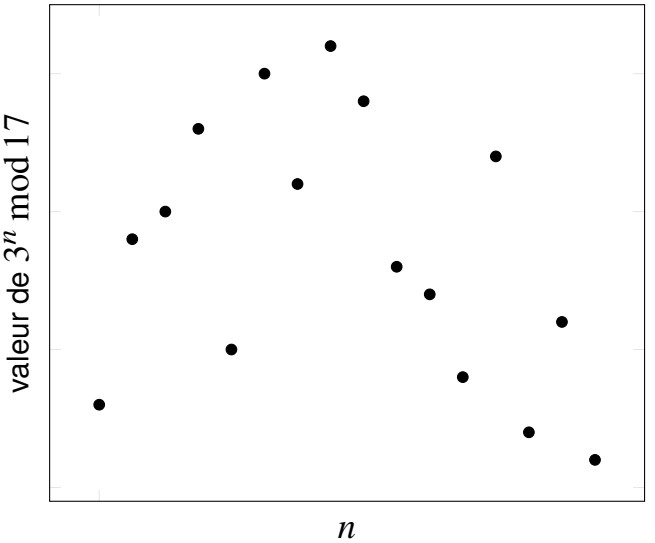


Plus le nombre premier est grand, plus le «*cadran*» est grand, et plus la séquence des valeurs obtenues de type $p^n mod q$ augmente.

On vérifie le «*caractère aléatoire*» des résultats de l'exponentiation modulaire :

On recommence

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	...
$3^n \bmod 17$	3	9	10	13	5	15	11	16	14	8	7	4	12	2	6	1	3	9	10	...



▷ Pourquoi la **procédure inverse** qui permet de passer de 12, par exemple, à la valeur de x telle que : $3^x \bmod 17 = 12$ est-elle **dure** ?

⟹ parce-qu'il faut **essayer toutes les valeurs possibles** avant de trouver la bonne, ce qui peut prendre beaucoup de temps !

Crypto-système : recherche d'un problème difficile à résoudre

$3^{29} \bmod 17$ --- facile ---> 12

$3^{?} \bmod 17$ <--- difficile --- 12

Problème du «*logarithme discret*» :
$\Longrightarrow$ pour trouver l'exposant, il faut **essayer les différentes valeurs possibles** !

Est-ce trouvable en cherchant ?

Pour des valeurs petites, comme 17 oui... mais si on utlise des nombre plus grands comme :

```
41 699 392 957 415 060 122 123 251 743 926 118 047 280 755 942 727 859 562 221 144 422 770 879 634
528 234 687 327 545 978 537 253 643 190 435 384 223 451 790 600 743 186 710 706 948 084 596 275 495
353 648 241 532 947 688 879 507 515 517 193 627 711 579 879 475 350 089 816 821 087 217 733 022 854
019 999 144 696 637 566 134 923 661 835 848 181 145 153 671 156 026 923 341 312 533 527 164 598 580
084 075 991
```

ce nombre premier a été choisi sur 1024bits.

Il faut **beaucoup, beaucoup de temps** pour y arriver ! *Suivant la taille, plusieurs années avec des ordinateurs puissants !*

Et si on essaye d'aller plus loin ?

La fonction $\Phi(n)$ calcule le nombre d'entiers inférieur à n qui ne partagent pas de diviseur supérieur à 1 avec n.
Exemple : $\Phi(8) = 4$, *car* $\boxed{1}, 2, \boxed{3}, 4, \boxed{5}, 6, \boxed{7}$, *il y en a 4 ayant cette propriété.*

On peut arriver à :

- $\Phi(n) = n - 1$ si n est premier, sinon Φ(n) est **long à calculer** (il faut **énumérer** les différents entiers) ;
- $\Phi(a * b) = \Phi(a) * \Phi(b)$;
- si on choisit **deux nombres premiers** p_1 et p_2 et on calcule $n = p_1 * p_2$,
  on a $\Phi(n) = \Phi(p_1) * \Phi(p_2) = (p_1 - 1) * (p_2 - 1)$
- Théorème d'Euler : $m^{k*\Phi(n)+1} = m \bmod n$
  On essaye de trouver $e * d = k * \Phi(n) + 1$, soit $d = \frac{k*\Phi(n)+1}{e}$,
  $\Longrightarrow$ d'où notre **cryptosystème** : $\boxed{\text{message}^e \bmod n = \text{chiffré}}$ et $\boxed{\text{chiffré}^d \bmod n = \text{message}}$

Chiffrement

Alice prépare ses valeurs :

$p_1 = 53$

$p_2 = 59$ (*le module*)

$n = 53 * 59 = 3127$

$\Phi(n) = 52 * 58 = 3016$

$e = 3$

$d = \frac{2*(3016)+1}{3} = 2011$

Alice **partage** avec Bob :

$n = 3127$

$e = 3$ *clé publique*

et conserve **secrétement** :

$n = 3127$

$d = 2011$ *clé privée*

Bob **veut envoyer un message** à Alice :

$m = 89$ où 89 correspond à une lettre de l'alphabet par exemple ;

Il calcule :

$m^e \bmod n \Longrightarrow 89^3 \bmod 3127 = 1394$

et transmets à Alice la valeur $m' = 1394$

Déchiffrement

Alice reçoit $m' = 1394$.

Elle calcule :

$1394^{2011} \bmod 3127 = 89$

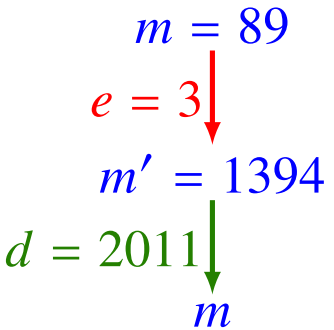
et retrouve le message m de Bob !

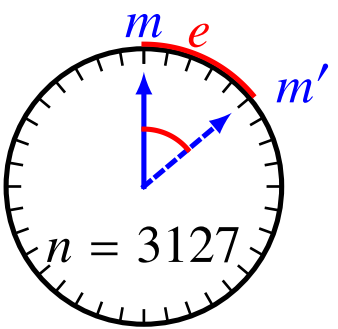
Un attaquant obtenant les valeurs

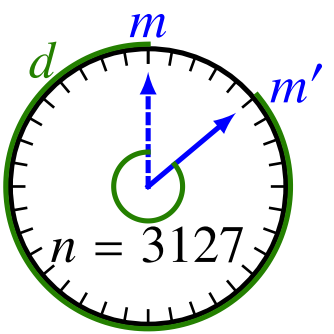
$n = 3127, 1394$ *et* $e = 3$

doit calculer $\Phi(3127)$ *pour trouver* d

Ce qui lui prendrait trop de temps pour n *très grand !*



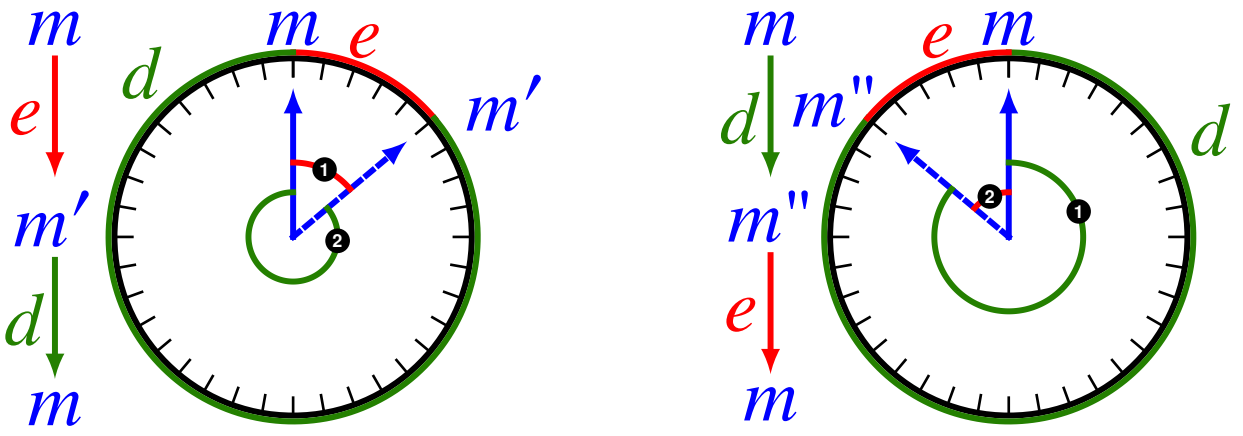




Chiffrement ? Déchiffrement ? Ce ne serait pas la même chose ?

On peut **permuter** l'usage de e et d...

$$(m^e)^d \bmod n = m^{e*d} \bmod n = m^{d*e} \bmod n = (m^d)^e = m$$



On peut créer un message m'' qui **peut être «(dé)chiffrer» avec la clé publique** (e et n) en le message original m mais qui ne peut être **créer qu'avec la clé privée** (d et n) !

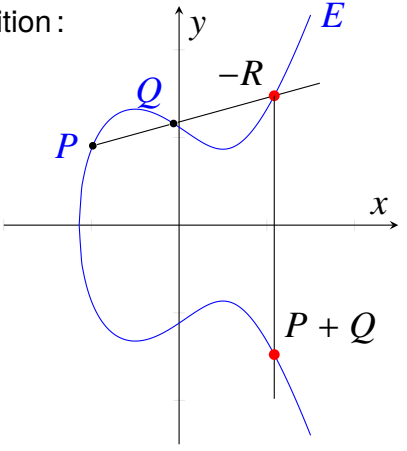
$\Longrightarrow$ Seule la personne **ayant la clé privée** peut chiffrer un message choisi **déchiffrable par la clé publique** associée...

$\Longrightarrow$ On peut **authentifier** la personne ! C-à-d **garantir son identité**, car elle est la **seule à posséder** cette clé privée !

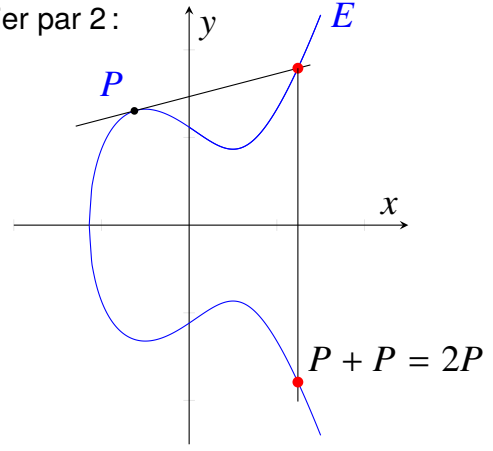
Et les «*Courbes elliptiques ?*»

Courbes elliptiques : $y^2 = x^3 + ax + b$

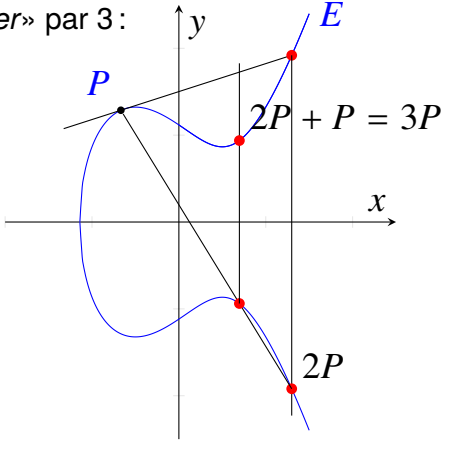
Addition :



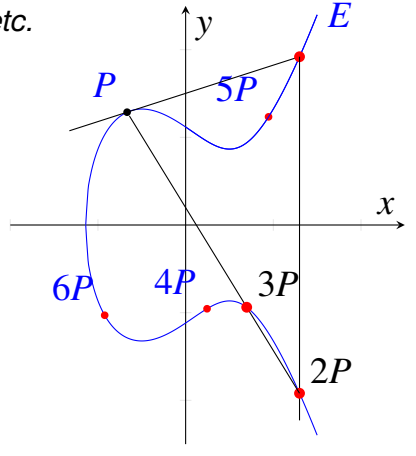
Multiplier par 2 :



«*Multiplier*» par 3 :

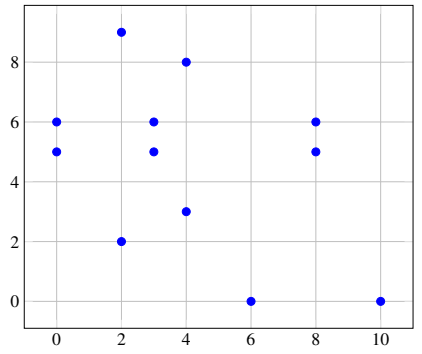


etc.

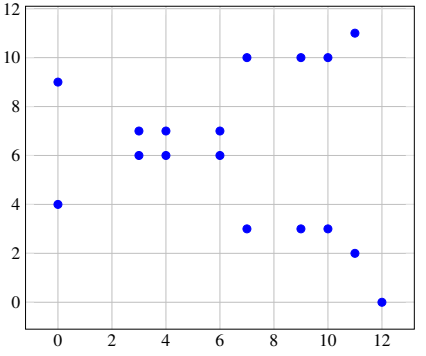


- ▷ on n'utilise que des **valeurs entières** positives pour les coordonnées des points sur la courbe ;
- ▷ on utilise le **modulo**, p, pour «*replier*» le domaine de ces points dans un **espace réduit**
  $\Longrightarrow$ on obtient un **ensemble fini** de points ;

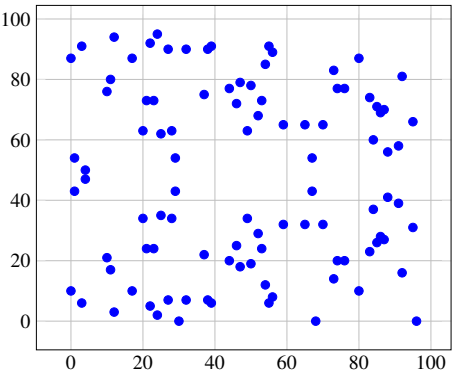
$y^2 \equiv x^3 + 2x + 3 \bmod 11$



$y^2 \equiv x^3 + 2x + 3 \bmod 13$



$y^2 \equiv x^3 + 2x + 3 \bmod 97$



On observe que :

- ▷ plus la taille du module p est **grande**, plus le domaine $\mathbb{F}_p$ est **important** ;
- ▷ la distribution des points semble **aléatoire** et recouvre la surface de manière **régulière** ;
- ▷ Notion de **groupe** :
  - ◇ les points appartenant à la courbe elliptique définie sur $\mathbb{F}_p$ définissent un **groupe** :
    - ⋆ **l'addition** d'un point avec un autre et la **multiplication scalaire** donnent, chacune, un point **appartenant** à cet ensemble ;
  - ◇ le **nombre de points** appartenant à ce groupe définit **l'ordre** du groupe ;
    Exemple : pour $y^2 \equiv x^3 + 2x + 3 \bmod 11$ *on dénombre 12 points, et 17 points pour* $y^2 \equiv x^3 + 2x + 3 \bmod 13$.

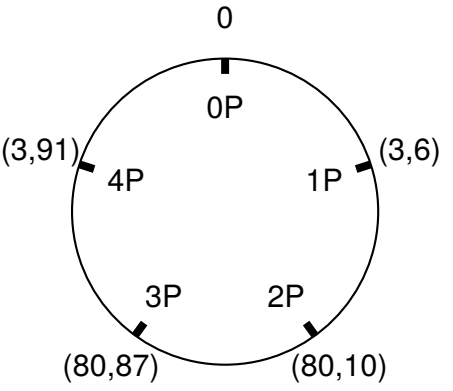
▷ en utilisant un **nombre premier** comme module, on obtient des propriétés intéressantes :

◇ pour un point P, on observe des **cycles** :

Pour la courbe $y^2 \equiv x^3 + 2x + 3 \bmod 97$ définie sur le domaine fini $\mathbb{F}_p$ où $p = 97$:

⋆ $0P = 0$
⋆ $1P = (3, 6)$
⋆ $2P = (80, 10)$
⋆ $3P = (80, 87)$
⋆ $4P = (3, 91)$
⋆ $5P = 0$
⋆ $6P = (3, 6)$

...



On remarque que :

⋆ on a définit un **sous groupe** de notre domaine $\mathbb{F}_p$: les autres points de la courbe elliptique n'apparaissent pas dans le cycle ;
⋆ on observe que $5kP = 0$, $(5k + 1)P = P$, $(5k + 2)P = 2P$, ..., $(5k + 4)P = 4P$ et $(5k + 5)P = 0$

$\Longrightarrow$ on déduit que $kP = (k \bmod 5)P$; P est appelé «*générateur*» du sous-groupe, 5 est **l'ordre** du sous-groupe.

$\Longrightarrow$ Et si on choisissait pour notre problème connaissant P et Q, peut on trouver k tel que $Q = kP$ en utilisant un sous-groupe comme vu plus haut ?

$\Longrightarrow$ on arrive au problème du «*logarithme discret*» car l'ensemble des points du sous-group est **fini**.

$\Longrightarrow$ dans RSA on utilisait l'**exponentiation modulaire**, en ECC on utilise la **multiplication scalaire** sur un domaine fini (sous-groupe cyclique) ;

$\Longrightarrow$ le **problème est plus dur** dans ECC que dans RSA $\Longrightarrow$ la **taille des clés peut être plus petite** pour une **difficulté similaire** !

Un ensemble d'opérations :

▷ on peut **additionner** des points ;

▷ on peut **multiplier** un point par un entier :
- ◇ Multiplier par 4 : $3P + P = 2P + 2P = 4P$
- ◇ Multiplier par 100 : $P \Longrightarrow 2P$, $2P \Longrightarrow 3P$, $3P \Longrightarrow 6P$, $6P \Longrightarrow 12P$, $12P \Longrightarrow 24P$, $24P \Longrightarrow 25P$, $25P \Longrightarrow 50P$ et $50P \Longrightarrow 100P$

▷ les points obtenus sont sur la courbe et apparaissent «*distribuer de manière aléatoire*» ;

▷ Si on donne $Q = nP$ alors il est très difficile de trouver $n \Longrightarrow$ il faut essayer les différentes possibilités : $2P$, $3P$, ...

$\Longrightarrow$ on a système qui est **rapide à calculer dans un sens** :
- ◇ connaissant n et P on cherche Q,
- ◇ **mais qui est difficile** à «*inverser*» : connaissant deux points, P et Q, trouver n tel que $Q = nP$ (problème du logarithme discret).

Avantages par rapport à RSA ?

▷ le problème posé par les courbes elliptiques est plus dur que celui posé par RSA :
$\Longrightarrow$ la taille des clés est **plus petite** pour assurer un niveau de **sécurité équivalent**.
$\Longrightarrow$ une clé de **246 bits ECC** est **équivalente** à une clé **3072 bits RSA** !

Et finalement
quels sont les usages ?

3. Les bases de la cryptographie

e. Chiffrement symétrique vs Chiffrement asymétrique

Chiffrement symétrique	Chiffrement asymétrique
Avantages	
• Rapidité des opérations (adapté à du trafic en temps réel) ; • Clés courtes (256 bits suffisent actuellement) ;	• Facilité d'échange des clés : les seules clés qui ont besoin d'être échangées sont des clés publiques (dont il faut assurer la protection en intégrité) ;
Inconvénients	
• Difficulté d'échange sécurisé des clés secrètes : comment le faire en protégeant ce secret ?	• Lenteur des opérations (peu adapté à du trafic en temps réel) ; • Grande taille des clés (2048 bits minimum actuellement) ;
Exemples d'algorithmes sûrs (janvier 2015)	
• AES.	• RSA.

Un dernier problème pour la route…

Le système de chiffrement à clé publique est universel si **chacun publie sa clé publique** dans un annuaire.
Pour envoyer un message chiffré à Bob, il suffit de trouver **sa clé publique** dans l'annuaire et de s'en servir pour **chiffrer le message** avant de le lui envoyer (seul Bob pourra déchiffrer le message).
Il faut bien sûr que **l'annuaire** soit **sûr**.

Oscar peut avoir substitué sa propre clé publique à celle de Bob afin de pouvoir lire les messages destinés à Bob.

Il peut même les renvoyer à Bob une fois lu !

Ce qui empêche Bob de s'en rendre compte et de changer sa bi-clé, et d'en informer Alice.

Propriété unique de RSA

L'algorithme a la propriété spéciale suivante :

$$\text{chiffrement}(\text{déchiffrement}(M)) = \text{déchiffrement}(\text{chiffrement}(M))$$

C'est-à-dire que l'utilisation de sa **clé privée** pour chiffrer un message M permet de construire un message M' qui peut être déchiffré par sa **clé publique**...ainsi il est **possible de prouver** que l'on dispose de la **clé privée associée à la clé publique** !

Application à l'authentification

En effet, la clé privée est **connue uniquement de son propriétaire**. Avec cette clé je peux chiffrer n'importe quel message que l'on me propose...

...et le message chiffré peut être **déchiffré par tout le monde** grâce à la clé publique ! Si je possède la clé privée **associée** à la clé publique $\Longrightarrow$ je suis la personne **associée** à cette clé publique !

Notion de «*Challenge*/*Response*»

Si Alice veut authentifier Bob (s'assurer que Bob est bien Bob) :

a. elle demande à Bob de lui chiffrer un message (n'importe lequel, si possible un **nouveau à chaque fois**) ;

b. Bob utilise sa **clé privée** pour le faire et renvoie le message chiffré à Alice ;

c. Alice vérifie qu'elle retrouve bien son message en déchiffrant le message chiffré reçu avec la **clé publique de Bob** : si c'est le même message, c'est Bob !

C'est de «**l'authentification vivante**» !

Chiffrement symétrique ou asymétrique ?

Comparaisons entre RSA et DES

RSA

- ⋆ clé de 1024 bits
- ⋆ chiffrement matériel : 300 Kbits/sec
- ⋆ chiffrement logiciel : 21,6 Kbits/sec
- ⋆ *Inconvénient majeur* : un pirate substitue sa propre clé publique à celle du destinataire, il peut alors intercepter et décrypter le message pour le recoder ensuite avec la vraie clé publique et le renvoyer sur le réseau. «L'attaque» ne sera pas décelée.
- ⋆ *Usage* : chiffrer des données courtes (de quelques octets) telles que les clés secrètes et les signatures électroniques.

facteur 1000 !

DES

- ◦ clé de 56 bits
- ◦ chiffrement matériel : 300 Mbits/sec
- ◦ chiffrement logiciel : 2,1 Mbits/sec
- ◦ *Inconvénient majeur* : attaque «*brute force*» rendue possible par la puissance des machines.
- ◦ *Usage* : chiffrement rapide, adapté aux échanges de données de tous les protocoles de communication sécurisés.

Vitesse de chiffrement

- □ il existe un **décalage de puissance de calcul** nécessaire pour le chiffrement/déchiffrement à clé secrète par rapport à celui à clé publique ;
- □ le chiffrement à clé secrète est utilisable pour un débit de données supérieur («réaliste» pour sécuriser une transaction entre deux utilisateurs sur Internet).

Résolution du problème de l'échange des clés secrètes

- ▷ utilisation d'une méthode **hybride** combinant à la fois chiffrement symétrique et asymétrique.

Évaluation de la vitesse de chiffrement en AES-CBC en bibliothèque **purement logicielle** :

xterm

```
pef@cube:~$ openssl speed  aes-128-cbc
Doing aes-128 cbc for 3s on 16 size blocks: 16488826 aes-128 cbc's in 3.00s
Doing aes-128 cbc for 3s on 64 size blocks: 4564896 aes-128 cbc's in 3.00s
Doing aes-128 cbc for 3s on 256 size blocks: 1167784 aes-128 cbc's in 3.00s
Doing aes-128 cbc for 3s on 1024 size blocks: 294818 aes-128 cbc's in 3.00s
Doing aes-128 cbc for 3s on 8192 size blocks: 36948 aes-128 cbc's in 3.00s
OpenSSL 1.0.2g  1 Mar 2016
The 'numbers' are in 1000s of bytes per second processed.
type             16 bytes     64 bytes    256 bytes   1024 bytes   8192 bytes
aes-128 cbc      87940.41k    97384.45k    99650.90k   100631.21k   100892.67k
```

Évaluation de la vitesse de chiffrement en AES-CBC en utilisant les **instructions AES-NI**, c-à-d des **instructions dédiées** du processeur Intel pour réaliser le chiffrement :

xterm

```
pef@cube:~$ openssl speed -evp aes-128-cbc
Doing aes-128-cbc for 3s on 16 size blocks: 137314812 aes-128-cbc's in 3.00s
Doing aes-128-cbc for 3s on 64 size blocks: 37526222 aes-128-cbc's in 3.00s
Doing aes-128-cbc for 3s on 256 size blocks: 9621612 aes-128-cbc's in 3.00s
Doing aes-128-cbc for 3s on 1024 size blocks: 2410345 aes-128-cbc's in 3.00s
Doing aes-128-cbc for 3s on 8192 size blocks: 303188 aes-128-cbc's in 3.00s
OpenSSL 1.0.2g  1 Mar 2016
The 'numbers' are in 1000s of bytes per second processed.
type             16 bytes     64 bytes    256 bytes   1024 bytes   8192 bytes
aes-128-cbc     732345.66k   800559.40k   821044.22k   822731.09k   827905.37k
```

On note une accélération $*8$ des performances !

Il n'y a pas de chiffrement par bloc avec RSA.

xterm

```
pef@cube:~$ openssl speed rsa
Doing 512 bit private rsa's for 10s: 150169 512 bit private RSA's in 9.99s
Doing 512 bit public rsa's for 10s: 2774703 512 bit public RSA's in 10.00s
Doing 1024 bit private rsa's for 10s: 69193 1024 bit private RSA's in 10.00s
Doing 1024 bit public rsa's for 10s: 1160767 1024 bit public RSA's in 10.00s
Doing 2048 bit private rsa's for 10s: 10641 2048 bit private RSA's in 10.00s
Doing 2048 bit public rsa's for 10s: 369236 2048 bit public RSA's in 10.00s
Doing 4096 bit private rsa's for 10s: 1584 4096 bit private RSA's in 10.00s
Doing 4096 bit public rsa's for 10s: 103756 4096 bit public RSA's in 10.00s
OpenSSL 1.0.2g  1 Mar 2016
                  sign    verify    sign/s verify/s
rsa  512 bits 0.000067s 0.000004s  15031.9 277470.3
rsa 1024 bits 0.000145s 0.000009s   6919.3 116076.7
rsa 2048 bits 0.000940s 0.000027s   1064.1  36923.6
rsa 4096 bits 0.006313s 0.000096s    158.4  10375.6
```

La vérification est bien plus rapide que la signature !

Utilisation du mode «OFB» en utilisation du jeu d'instruction dédié du processeur :

xterm

```
pef@cube:~$ openssl speed -evp aes-128-ofb
Doing aes-128-ofb for 3s on 16 size blocks: 126910978 aes-128-ofb's in 3.00s
Doing aes-128-ofb for 3s on 64 size blocks: 33089676 aes-128-ofb's in 3.00s
Doing aes-128-ofb for 3s on 256 size blocks: 8241247 aes-128-ofb's in 3.00s
Doing aes-128-ofb for 3s on 1024 size blocks: 2062682 aes-128-ofb's in 3.00s
Doing aes-128-ofb for 3s on 8192 size blocks: 257968 aes-128-ofb's in 3.00s
OpenSSL 1.0.2g  1 Mar 2016
The 'numbers' are in 1000s of bytes per second processed.
type             16 bytes     64 bytes    256 bytes   1024 bytes   8192 bytes
aes-128-ofb     676858.55k   705913.09k   703253.08k   704062.12k   704424.62k
```

Équivalence du niveau de sécurité des courbes elliptiques

Symmetric Key Length	Standard asymmetric Key Length	Elliptic Curve Key Length
80	1024	160
112	2048	224
128	3072	256
192	7680	384
256	15360	512

Comparaison de vitesse

```
xterm
pef@cube:~$ openssl speed ecdsa
Doing 160 bit sign ecdsa's for 10s: 134037 160 bit ECDSA signs in 9.95s
Doing 160 bit verify ecdsa's for 10s: 36683 160 bit ECDSA verify in 10.00s
Doing 192 bit sign ecdsa's for 10s: 114991 192 bit ECDSA signs in 9.96s
Doing 192 bit verify ecdsa's for 10s: 30191 192 bit ECDSA verify in 10.00s
Doing 224 bit sign ecdsa's for 10s: 105376 224 bit ECDSA signs in 9.97s
Doing 224 bit verify ecdsa's for 10s: 48166 224 bit ECDSA verify in 10.00s
Doing 256 bit sign ecdsa's for 10s: 185621 256 bit ECDSA signs in 9.93s
Doing 256 bit verify ecdsa's for 10s: 87042 256 bit ECDSA verify in 9.99s
OpenSSL 1.0.2g  1 Mar 2016
                              sign    verify    sign/s verify/s
 160 bit ecdsa (secp160r1)   0.0001s   0.0003s  13471.1   3668.3
 192 bit ecdsa (nistp192)    0.0001s   0.0003s  11545.3   3019.1
 224 bit ecdsa (nistp224)    0.0001s   0.0002s  10569.3   4816.6
 256 bit ecdsa (nistp256)    0.0001s   0.0001s  18693.0   8712.9
```

La signature est bien plus rapide que la vérification !

xterm

```
pef@cube:~$ openssl speed rsa
Doing 512 bit private rsa's for 10s: 140959 512 bit private RSA's in 9.99s
Doing 512 bit public rsa's for 10s: 2631882 512 bit public RSA's in 10.00s
Doing 1024 bit private rsa's for 10s: 66152 1024 bit private RSA's in 9.99s
Doing 1024 bit public rsa's for 10s: 1097544 1024 bit public RSA's in 10.00s
Doing 2048 bit private rsa's for 10s: 9965 2048 bit private RSA's in 10.00s
Doing 2048 bit public rsa's for 10s: 365198 2048 bit public RSA's in 10.00s
Doing 3072 bit private rsa's for 10s: 3444 3072 bit private RSA's in 9.99s
Doing 3072 bit public rsa's for 10s: 169980 3072 bit public RSA's in 9.99s
Doing 4096 bit private rsa's for 10s: 1567 4096 bit private RSA's in 10.01s
Doing 4096 bit public rsa's for 10s: 99611 4096 bit public RSA's in 9.99s
Doing 7680 bit private rsa's for 10s: 169 7680 bit private RSA's in 10.03s
Doing 7680 bit public rsa's for 10s: 30041 7680 bit public RSA's in 10.00s
Doing 15360 bit private rsa's for 10s: 33 15360 bit private RSA's in 10.32s
Doing 15360 bit public rsa's for 10s: 7746 15360 bit public RSA's in 9.99s
OpenSSL 1.1.0g  2 Nov 2017
                  sign    verify    sign/s verify/s
rsa  512 bits 0.000071s 0.000004s  14110.0 263188.2
rsa 1024 bits 0.000151s 0.000009s   6621.8 109754.4
rsa 2048 bits 0.001004s 0.000027s    996.5  36519.8
rsa 3072 bits 0.002901s 0.000059s    344.7  17015.0
rsa 4096 bits 0.006388s 0.000100s    156.5   9971.1
rsa 7680 bits 0.059349s 0.000333s     16.8   3004.1
rsa 15360 bits 0.312727s 0.001290s      3.2    775.4
```

La vérification est bien plus rapide que la signature !

Évaluation du chiffrement symétrique : débit

Soit le script suivant :

```
 1 #!/bin/bash
 2
 3 CIPHER=$1
 4
 5 if [ -n "$1" ]; then
 6    echo "Vitesse de création de 700Mo de données sans sauvegarde"
 7    dd if=/dev/zero bs=1m count=700 | pipebench > /dev/null
 8    echo "Enregistrement de 700Mo dans le fichier clair"
 9    dd if=/dev/zero bs=1m count=700 | pipebench > clair
10    echo "Chiffrement $CIPHER sans sauvegarde"
11    cat clair | openssl enc -e -$CIPHER -pass pass:toto | pipebench > /dev/null
12    echo "Chiffrement $CIPHER dans le fichier chiffre-$CIPHER"
13    cat clair | openssl enc -e -aes-128-cbc -pass pass:toto | pipebench > chiffre-$CIPHER
14    echo "Déchiffrement sans sauvegarde"
15    cat chiffre-$CIPHER | openssl enc -d -$CIPHER -pass pass:toto | pipebench > /dev/null
16    echo "Déchiffrement dans le fichier dechiffre-$CIPHER"
17    cat chiffre-$CIPHER | openssl enc -d -aes-128-cbc -pass pass:toto | pipebench > dechiffre-$CIPHER
18 fi
```

- ligne 7 : on crée 700Mo de données sans les sauvegarder pour connaitre la vitesse maximale sans être limité par la vitesse du disque dur ;
- ligne 11 : on chiffre sans sauvegarder $\Longrightarrow$ pas d'influence du débit du disque dur ;
- ligne 13 : on chiffre avec sauvegarde pour permettre l'évaluation du déchiffrement ;
- ligne 15 : on déchiffre sans sauvegarde ;
- ligne 17 : on déchiffre avec sauvegarde ;

Évaluation du chiffrement symétrique : débit

AES-128 en mode CBC

xterm

```
pef@darkstar-8:/Users/pef/tmp/CIPHER~: cipher_test aes-128-cbc
Vitesse de création de 700Mo de données sans sauvegarde
700+0 records in00.00 MB  518.75 MB/second (Mon Feb  5 14:19:59 2018)
700+0 records out
734003200 bytes transferred in 0.235985 secs (3110380209 bytes/sec)
Summary:
Piped  700.00 MB in 00h00m00.23s:    2.92 GB/second

Enregistrement de 700Mo dans le fichier clair
700+0 records in99.21 MB  671.09 MB/second (Mon Feb  5 14:20:00 2018)
700+0 records out
734003200 bytes transferred in 0.974389 secs (753295801 bytes/sec)
Summary:
Piped  700.00 MB in 00h00m00.93s:  749.26 MB/second

Chiffrement aes-128-cbc sans sauvegarde
Summary:
Piped  700.00 MB in 00h00m01.06s:  658.30 MB/second

Chiffrement aes-128-cbc dans le fichier chiffre-aes-128-cbc
Summary:
Piped  700.00 MB in 00h00m01.62s:  429.77 MB/second

Déchiffrement sans sauvegarde
Summary:
Piped  700.00 MB in 00h00m00.37s:    1.81 GB/second

Déchiffrement dans le fichier dechiffre-aes-128-cbc
Summary:
Piped  700.00 MB in 00h00m01.33s:  526.09 MB/second
```

Le déchiffrement est bien plus rapide que le chiffrement !

Évaluation du chiffrement symétrique : débit

AES-128 en mode OFB

xterm

```
pef@darkstar-8:/Users/pef/tmp/CIPHER~: cipher_test aes-128-ofb
Vitesse de création de 700Mo de données sans sauvegarde
700+0 records in00.00 MB  332.03 MB/second (Mon Feb  5 14:20:07 2018)
700+0 records out
734003200 bytes transferred in 0.239618 secs (3063221490 bytes/sec)
Summary:
Piped  700.00 MB in 00h00m00.23s:    2.88 GB/second

Enregistrement de 700Mo dans le fichier clair
700+0 records in99.21 MB  513.28 MB/second (Mon Feb  5 14:20:08 2018)
700+0 records out
734003200 bytes transferred in 1.136870 secs (645635174 bytes/sec)
Summary:
Piped  700.00 MB in 00h00m01.08s:  642.22 MB/second

Chiffrement aes-128-ofb sans sauvegarde
Summary:
Piped  700.00 MB in 00h00m01.16s:  601.74 MB/second

Chiffrement aes-128-ofb dans le fichier chiffre-aes-128-ofb
Summary:
Piped  700.00 MB in 00h00m01.64s:  424.34 MB/second

Déchiffrement sans sauvegarde
Summary:
Piped  700.00 MB in 00h00m01.16s:  603.37 MB/second

Déchiffrement dans le fichier dechiffre-aes-128-ofb
Summary:
Piped  700.00 MB in 00h00m01.20s:  582.84 MB/second
```

Évaluation du chiffrement symétrique : débit

AES-128 en mode CFB

xterm

```
pef@darkstar-8:/Users/pef/tmp/CIPHER~: cipher_test aes-128-cfb
Vitesse de création de 700Mo de données sans sauvegarde
700+0 records in00.00 MB    0.00 B/second (Mon Feb  5 14:20:17 2018)
700+0 records out
734003200 bytes transferred in 0.230358 secs (3186356912 bytes/sec)
Summary:
Piped  700.00 MB in 00h00m00.22s:    2.99 GB/second

Enregistrement de 700Mo dans le fichier clair
700+0 records in99.21 MB  153.12 MB/second (Mon Feb  5 14:20:18 2018)
700+0 records out
734003200 bytes transferred in 0.986273 secs (744219058 bytes/sec)
Summary:
Piped  700.00 MB in 00h00m00.94s:  740.28 MB/second

Chiffrement aes-128-cfb sans sauvegarde
Summary:
Piped  700.00 MB in 00h00m01.39s:  503.24 MB/second

Chiffrement aes-128-cfb dans le fichier chiffre-aes-128-cfb
Summary:
Piped  700.00 MB in 00h00m01.76s:  395.74 MB/second

Déchiffrement sans sauvegarde
Summary:
Piped  700.00 MB in 00h00m01.41s:  493.67 MB/second

Déchiffrement dans le fichier dechiffre-aes-128-cfb
Summary:
Piped  700.00 MB in 00h00m01.21s:  576.78 MB/second
```

Le chiffrement symétrique

est

beaucoup plus rapide

que

le chiffrement asymétrique...

Combinaison symétrique/asymétrique

L'utilisation d'algorithme de chiffrement à clé **asymétrique** : il est coûteux en puissance de calcul nécessaire à le mettre en œuvre.

La **puissance de calcul** des ordinateurs augmente ?
⟹ il est nécessaire d'améliorer la sécurité des algorithmes symétriques et asymétriques pour résister à la cryptanalyse ;
– Comment ? **augmenter la taille** des clé par exemple ;

Conséquence :
▷ le **décalage** entre besoin de calcul entre symétrique et asymétrique **reste** !

Une solution : la combinaison

Il faut trouver un moyen de **partager secrètement** une même clé secrète :
l'échange de la clé secrète d'un algorithme de chiffrement symétrique est «protégé» par un algorithme de chiffrement asymétrique.

Cette clé partagée sera appelée **clé de session**.

C'est un **compromis** entre le chiffrement symétrique et asymétrique permettant de combiner les deux techniques.

Il existe deux méthodes pour construire et partager une clé de session :

▷ **Première** possibilité :
- ◇ construire une **clé de session** à l'aide de la méthode d'échange de clé de **Diffie-Hellman**.
- ◇ les interlocuteurs n'ont **pas besoin de partager une clé** avant de commencer leur communication chiffrée ! *Cette méthode est extrémement employée pour initier un canal de transmission sécurisée avant tout échange.*

▷ **Seconde** possibilité :

a. générer **aléatoirement** une clé de **taille raisonnable** utilisée pour un algorithme de chiffrement symétrique ;

b. **chiffrer** cette clé à l'aide d'un algorithme de **chiffrement à clé publique**, à l'aide de la clé publique du destinataire ;

c. envoyer cette clé chiffrée au destinataire ;

d. le destinataire déchiffre la clé symétrique à l'aide de sa clé privée.

Les deux interlocuteurs disposent ensuite :

□ d'une **clé symétrique commune** qu'ils sont seuls à connaître ;

□ et donc, de la possibilité de **communiquer en chiffrant** leur données à l'aide d'un algorithme de chiffrement symétrique rapide.

*Cela impose que l'un des interlocuteurs **possède la clé publique** de l'autre (pas toujours facile de s'assurer que la clé publique appartient bien à la bonne personne).*

La méthode d'échange des clés de Diffie-Hellman

1. Alice et Bob se mettent en accord sur deux grands nombres premiers n et g avec $(n-1)/2$ premier et quelques conditions sur g.
   Ces nombres sont **publics**.
2. Alice choisit un nombre de, au moins, 2048 bits secret x ;
3. Bob choisit un nombre de, au moins, 2048 bits secret y ;
4. Alice envoie à Bob un message contenant le nombre n, le nombre g et le résultat de $g^x \bmod n$;
5. Bob envoie à Alice le résultat de $g^y \bmod n$;
6. Alice calcule $(g^y \bmod n)^x$;
7. Bob calcule $(g^x \bmod n)^y$.

A et B partagent maintenant la même clé secrète $g^{xy} \bmod n$!

Si Oscar, l'intrus capture g et n, il ne peut pas calculer x et y, car il n'existe pas de méthode utilisable de manière raisonnable pour calculer x à partir de $g^x \bmod n$ (problème du logarithme discret) !

Risque : l'attaque «*Man-in-the-Middle*»

Problème : Oscar peut s'insérer entre Alice et Bob et proposé sa valeur z en lieu et place de x pour Bob et de y pour Alice :

Alice	$\Longrightarrow n, g, g^x \bmod n \Longrightarrow$	Oscar	$\Longrightarrow n, g, g^z \bmod n \Longrightarrow$	Bob
$g^{xz} \bmod n \Longleftarrow$	$g^z \bmod n \Longleftarrow$	$g^{xz} \bmod n$	$\Longleftarrow g^y \bmod n \Longleftarrow$	$g^{yz} \bmod n$
$g^{xz} \bmod n$		$g^{xz} \bmod n, g^{yz} \bmod n$		$g^{yz} \bmod n$

Conclusion : il faut une phase préliminaire d'authentification !

Avantages

- la clé secrète est chiffrée et échangée : pas d'interception possible ;
- elle est changée à chaque communication : sécurité plus robuste dans le temps ;
- après l'échange on bascule le chiffrement en utilisant un **algorithme symétrique** plus rapide ;
- on démarre l'échange avec l'utilisation d'un algorithme asymétrique qui possède l'avantage d'offrir un moyen **d'authentifier** les interlocuteurs.

Remarque

Cela **impose** que l'un des interlocuteurs possède la **clé publique de l'autre** (pas toujours facile de s'assurer que la clé publique appartient bien à la bonne personne).

Attention

La PFS, «*Perfect Forward Secrecy*», correspond à la protection des échanges futurs en cas de **compromission** de la **clé privée du serveur** :

- ▷ si la clé de session est échangée **chiffrée par la clé publique du serveur**
  ⟹les transactions antérieures sont déchiffrables ;
- ▷ si la clé de session est déterminée par DH : la compromission de la clé publique **ne permet pas de déchiffrer** les transactions !

Seule la première possibilité avec Difie-Hellman garantie la PFS...

Et Comment Améliorer la Sécurité
si la puissance des ordinateurs
augmente constamment ?

La sécurité offerte par le chiffrement à clé

La sécurité d'un code à clé est **proportionnelle** à la taille de la clé employée, c-à-d **plus la clé est longue plus il faut de calcul et donc de temps pour arriver à le casser**.

Attaque «*brute force*» :

▷ **essayer toutes les clés possibles** pour déchiffrer le message chiffré ;

⟹ plus la clé est **longue** (nombre de bits), **plus il y a de clés à essayer** (2 fois plus de clé à essayer pour chaque bit ajouté !).

Remarque : *pour un **niveau de sécurité équivalent**, la taille des clés est souvent **plus petite** en chiffrement symétrique qu'en chiffrement asymétrique.*
128bits en AES contre 2048bits par exemple en RSA.

Mais, le **niveau de sécurité** est à mettre en **rapport** avec le type de données à sécuriser :

▷ une **transaction bancaire** doit être sécurisée pendant quelques minutes ;

▷ un **document secret d'état** doit pouvoir être protégé plus de 50 ans par exemple.

Les recommendations pour la taille des clés

BlueKrypt | Cryptographic Key Length Recommendation

Dans la plupart des fonctions cryptographiques, la longueur des clefs est un paramètre sécuritaire important. Un certain nombre de publications académiques et gouvernementales fournissent des recommandations et des techniques mathématiques pour estimer la taille minimale sécuritaire des clefs cryptographiques. Malgré la disponibilité de ces publications, choisir une taille de clef appropriée reste complexe, la lecture et la compréhension de tous ces documents étant nécessaires.

Ce site internet vous permet d'évaluer une longueur de clef appropriée garantissant un niveau de sécurité minimal en implémentant les formules mathématiques et en résumant les informations disponibles dans toutes ces publications. Vous pouvez également comparer ces méthodes facilement. Les longueurs fournies ici sont conçues pour résister aux attaques mathématiques, elles ne prennent pas en considération les attaques algorithmiques, le matériel déficient, etc.

Choix de la méthode

- Équations de Lenstra et Verheul (2000)
- Équations plus récentes de Lenstra (2004)
- Recommandations ECRYPT-CSA (2018)
- Recommandations du NIST (2016)
- Recommandations de l'ANSSI (2014)
- CNSA Suite de l'IAD-NSA (2016)
- Network Working Group RFC3766 (2004)
- Recommandations du BSI (2018)

Comparer les méthodes

Auteur: Damien Giry
Approuvé par Pr. Jean-Jacques Quisquater
Contact: keylength@bluekrypt.com

Je remercie Pr. Arjen K. Lenstra pour son aimable autorisation et ses commentaires.
Pour des informations sur la règlementation en cryptologie: Crypto Law Survey / Digital Signature Law Survey.

Confidentialité (P3P) | Limitation de responsabilité / Droit de reproduction | Détails des mises à jour

https://www.keylength.com/

Le chiffrement d'une communication

ne

permet pas l'authentification...

L'authentification est suivie par l'autorisation

L'autorisation définit les ressources, services et informations que la personne identifiée peut utiliser, consulter ou mettre à jour, exemple : son courrier électronique, des fichiers sur un serveur FTP…

L'approche traditionnelle

Combinaison d'une identification et d'un mot de passe (code secret personnel).
Le mot de passe doit posséder certaines caractéristiques : non trivial, difficile à deviner, régulièrement modifié, secret…
Des outils logiciel ou hardware de génération de mots de passe existent, mais les mots de passe générés sont difficiles à retenir !

L'approche évoluée, la notion de challenge/réponse

Authentification avec du chiffrement asymétrique :

- ▷ Alice envoie à Bob un **message aléatoire** «*challenge*» :
- ▷ Bob renvoie à Alice le **message chiffré à l'aide de sa clé privée** «*réponse*» ;
  exploitation de la propriété chiffrement(*déchiffrement*(M)) = *déchiffrement*(*chiffrement*(M)) ;
- ▷ Alice peut **déchiffrer** ce message chiffré à l'aide de la **clé publique de Bob**… $\Longrightarrow$ c'est Bob !

Si on utilise du chiffrement asymétrique
peut-on faire à la fois de
l'authentification et de l'échange sécurisé ?

Authentification à l'aide du chiffrement à clé publique et échange de clé de session

On suppose que chaque interlocuteur possède **la clé publique** de l'autre. *Ce qui n'est pas évident…*
On désire échanger une clé de session tout en s'assurant de l'identité de chacun.

Scénario : Alice veut échanger avec Bob

1. Alice chiffre avec la **clé publique de Bob** son **identité** et un **nombre aléatoire** N ;
   L'identité permet de sélectionner la clé publique d'«Alice»
2. Alice envoie ce message à Bob qui peut le déchiffrer et retrouver N
   Bob qui reçoit ce message ne sait pas s'il vient d'Alice ou bien d'Oscar (l'intrus)
3. Bob répond par un message chiffré avec la **clé publique d'Alice**, contenant : N, un nombre aléatoire P et S une clé de session ;
4. Alice reçoit le message et le déchiffre à l'aide de sa clé privée
   Si Alice trouve N alors c'est bien Bob qui lui a envoyé le message puisqu'il était le seul à pouvoir déchiffrer N, pas d'intrus qui peut s'insérer dans la communication.
   Ce n'est pas possible non plus que cette réponse soit un message déja échangé puisque N vient juste d'être choisi par Alice (protection contre le rejeu).
5. Alice valide la session en renvoyant à Bob le nombre P chiffré maintenant avec la clé de session S

L'échange est maintenant basculé en chiffrement à clé secrète avec la clé S…

Problème

Comment être sûr de disposer de la bonne clé publique ?
Il faut disposer d'un intermédiaire de confiance qui détient et distribue les clés publiques.

Protocole d'envoi de message et accusé de réception avec authentification

Remarque : Asymétrique=Chiffrement=Déchiffrement

- Alice envoie :

  Message chiffré$_{\text{Alice}\to\text{Bob}}$ = Asymétrique(Clé$_{\text{pub}}$ de Bob, *authentification d'Alice* : Asymétrique(Clé$_{\text{priv}}$ de Alice, **Message clair$_{\text{Alice}}$**))

- Bob déchiffre et vérifie :

  Asymétrique (Clé$_{\text{pub}}$ de Alice, Asymétrique(Clé$_{\text{priv}}$ de Bob, Message chiffré$_{\text{Alice}\to\text{Bob}}$)) $\Longrightarrow$ **Message clair$_{\text{Alice}}$**

- Bob accuse réception, signe, et renvoi :

  Message chiffré$_{\text{Bob}\to\text{Alice}}$ = Asymétrique(Clé$_{\text{pub}}$ de Alice, Asymétrique(Clé$_{\text{priv}}$ de Bob, **Message clair$_{\text{Alice}}$**))

- Alice déchiffre et vérifie :

  Asymétrique (Clé$_{\text{pub}}$ de Bob, Asymétrique(Clé$_{\text{priv}}$ de Alice, Message chiffré$_{\text{Bob}\to\text{Alice}}$)) $\Longrightarrow$ **Message clair$_{\text{Alice}}$**

Attaque sur la clé privée de Bob en soumettant des messages bien choisi

Si Oscar envoie le **Message chiffré$_{\text{Alice}\to\text{Bob}}$** (chiffré pour Bob) en tant que message d'authentification :

- Bob déchiffre et vérifie :

  Asymétrique (Clé$_{\text{pub}}$ de Oscar, *Youpi !* : Asymétrique(Clé$_{\text{priv}}$ de Bob, **Message chiffré$_{\text{Alice}\to\text{Bob}}$**)) $\Longrightarrow$ Message faux

  Message faux = Asymétrique(clé$_{\text{pub}}$ de Oscar, *authentification d'Alice* : Asymétrique(Clé$_{\text{priv}}$ de Alice, **Message clair$_{\text{Alice}}$**)) ;

- Bob accuse réception,signe, et renvoi :

  Message chiffré$_{\text{Bob}\to\text{Oscar}}$ = Asymétrique(Clé$_{\text{pub}}$ de Oscar, Asymétrique(Clé$_{\text{priv}}$ de Bob, Message faux))

- Oscar déchiffre et «*vérifie*» :

  Asymétrique (Clé$_{\text{pub}}$ de Bob, Asymétrique(Clé$_{\text{priv}}$ d'Oscar, Message chiffré$_{\text{Bob}\to\text{Oscar}}$))$\Longrightarrow$ Message faux

- Oscar peut alors récupérer le **Message clair d'Alice** :

  Asymétrique(Clé$_{\text{priv}}$ d'Oscar, Message faux) = *authentification d'Alice* : Asymétrique(Clé$_{\text{priv}}$ de Alice, **Message clair$_{\text{Alice}}$**)

  Asymétrique (Clé$_{\text{pub}}$ de Alice, Asymétrique(Clé$_{\text{priv}}$ de Alice, **Message clair$_{\text{Alice}}$**)) $\Longrightarrow$ **Message clair$_{\text{Alice}}$!**

Solution : ajouter un contrôle d'intégrité pour garantir la structure d'un message correct

- calculer une empreinte, un «*résumé*» du message aléatoire initial, un «*digest*», à l'aide d'une **fonction de hachage** ;
- utiliser cette **empreinte** en combinaison avec le message aléatoire lors du chiffrement ;

$\Longrightarrow$ éviter à Bob de traiter un mauvais message !

Ok pour les échanges sécurisés.

Mais pour un document, comment faire de l'authentification ?

L'authentification d'un document correspond à **identifier son propriétaire/créateur**.

Quelle sorte de chiffrement utiliser pour chiffrer le document ?

- **Chiffrement symétrique ?** Impossible. On peut vérifier la provenance d'un document que par la seule vérification que le document a été chiffré par une clé secrète que l'on connait.
  S'il faut connaître la clé secrète pour vérifier, il est impossible de démontrer la provenance d'un document sans donner cette clé secrète.
  Tous ceux qui veulent vérifier la doit posséder la clé secrète et tout le monde peut modifier le document !
  Le document est toujours chiffré, il ne peut être consulté librement !

- **Chiffrement asymétrique ?** Possible. On chiffre le document avec la clé privée de son propriétaire.
  Tout le monde peut vérifier avec la clé publique du propriétaire du document que ce document a été chiffré avec sa clé privée.
  Sa clé privée est...privée : c'est bien son document !
  *Si la clé privée a été utilisée, cela n'a pu être fait qu'avec l'accord du propriétaire : **non répudiation** !*
  Le document est chiffré mais peut être déchiffré par tous (en se procurant librement la bonne clé publique du propriétaire).
  *Cela revient à de la **signature** !*

Oui, mais le chiffrement asymétrique est lent !

Solution : il faut «compresser» le document avant de le signer (résumé ou «*digest*» en anglais)...
...et s'assurer que la «compression» du document est **bien associée** au bon document ;
...et qu'il n'est **pas facile** de trouver **un autre document** qui a la même «compression» qu'un autre (pour faire accepter un document et le remplacer par un autre).

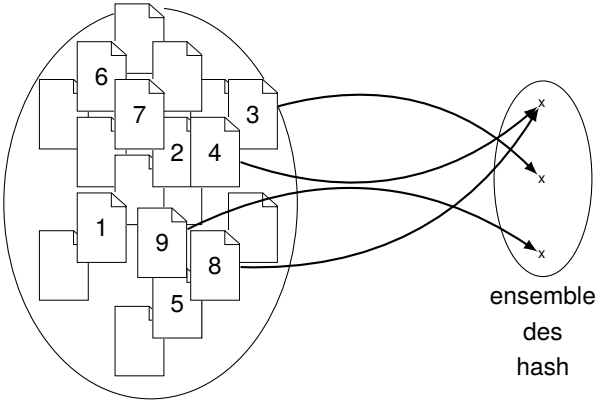
Qu'est-ce qu'une fonction de hachage ?

Une **fonction de hachage** est une fonction qui fait correspondre **à toute information** une valeur appelée «***hash***».
Exemple : pour accéder rapidement à un livre dans une base de données, on utilise une fonction de hachage pour accéder directement au livre recherché :

f : {*ensemble des titres de livres*} $\mapsto$ {*ensemble des hashs*}
f(*20 000 Lieues sous les mers*) $\rightarrow 598$
f(*Les androïdes rêvent-ils de moutons électriques ?*) $\rightarrow 698593$

*Ainsi, on ne parcours pas tous les titres, mais on va **directement** à la valeur indiquée par la fonction de hachage.*
*Si deux livres possèdent le même hash, on parle de «**collision**». Une bonne fonction de hachage limite les collisions.*

Fonction de hachage cryptographique

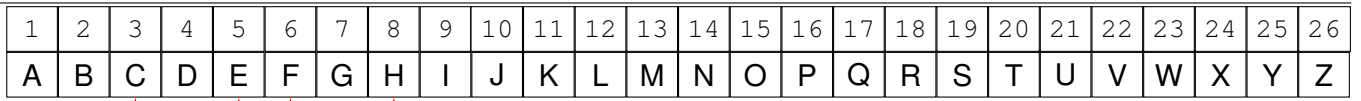


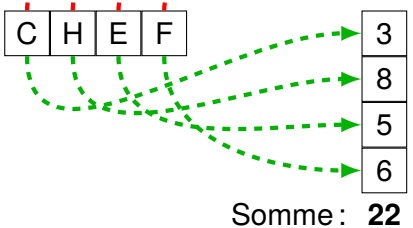
Une **fonction de hachage cryptographique** possède les propriétés suivantes :

- **pour tout document électronique** elle calcule un hash ;
- depuis le hash il est **impossible** de retrouver le document ;
- deux documents **proches** possédent des hashs **très différents** ;
- pour un **hash connu**, il est difficile, voire **impossible** de trouver le document permettant de l'obtenir ;
- il est très difficile, voire **impossible**, de trouver deux documents différents ayant le **même hash** (collision).
- le hash est de taille **fixe** et **limitée** (le document peut être de taille quelconque).

*On parle aussi **d'empreinte** de documents.*

Fonction de hachage cryptographique



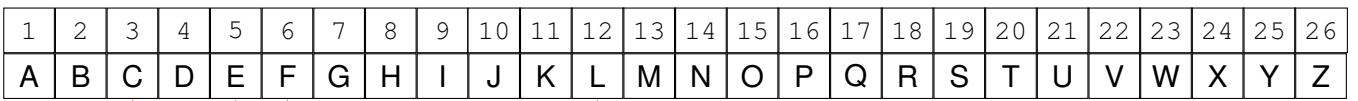


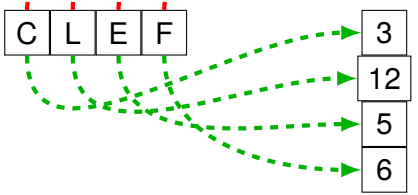
Somme : **22**

CHEF

HASH

77379280478615953124555518798002206243





Somme : **26**

CLEF

HASH

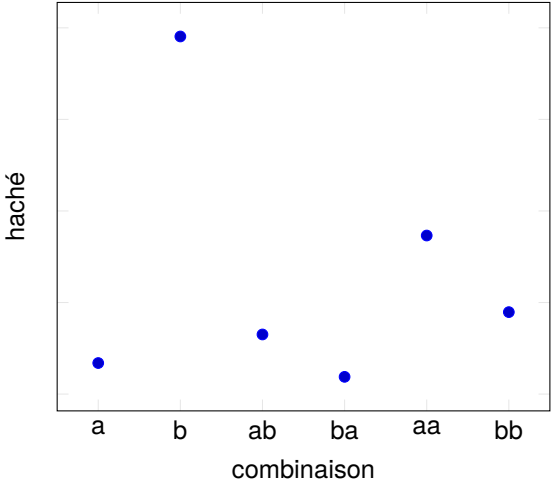
304901597169549689421998892641104281924

*Une fonction de hachage **classique** donne des valeurs **très proches**...*

*Une fonction de hachage **cryptographique** des valeurs **très éloignées** !*

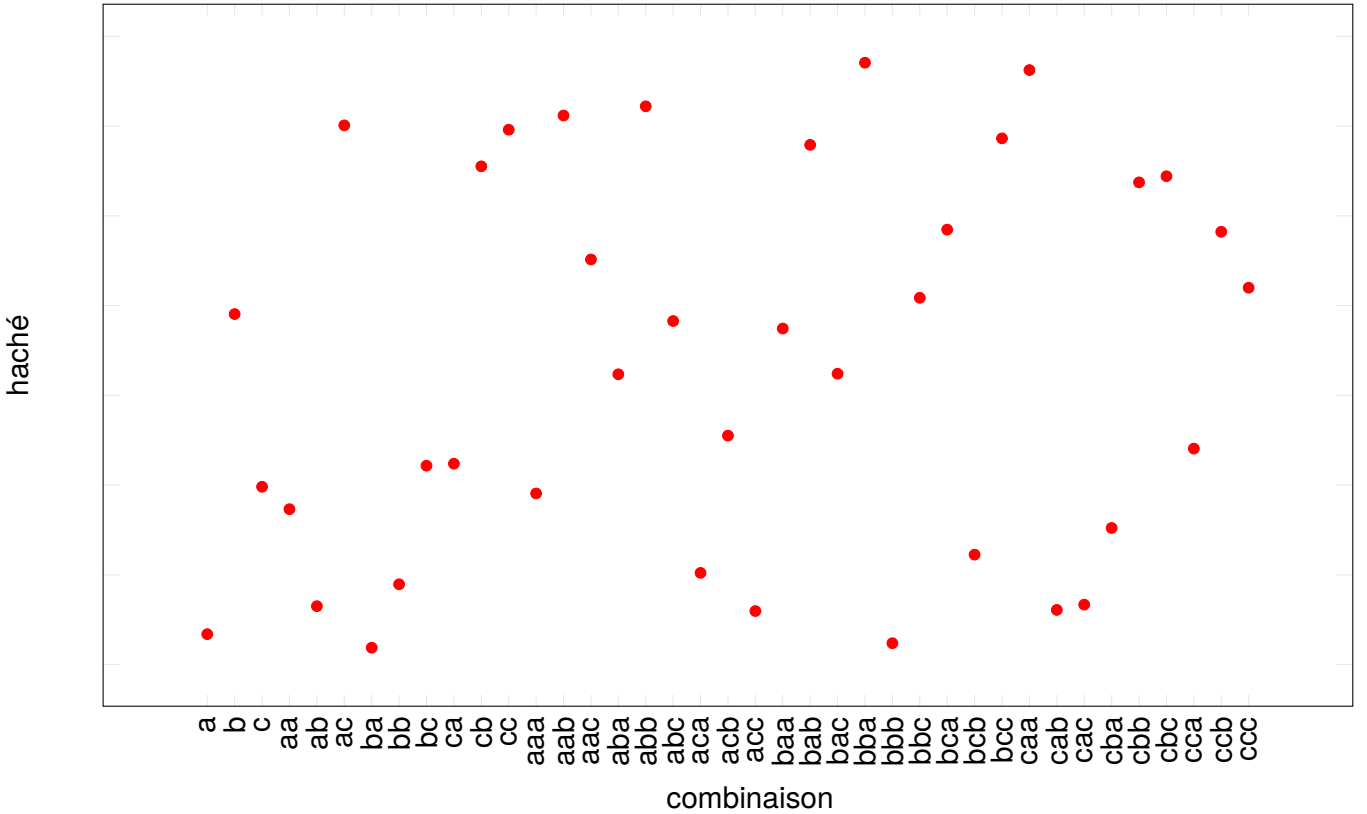
Imaginons que l'on veuille voir quelles sont les valeurs associées par la **fonction de hachage** aux différentes combinaisons possibles obtenues à partir des lettres«*a*» et «*b*», d'au plus 2 lettres :

combinaison	**haché**
a	169552370019632401730582715598587264 97
b	195289424170611159128911017612795795343
ab	325606555493056888658533171298094888 00
ba	941680395931154527312999502951431136 4
aa	865905563437731851846768541823880583 86
bb	447630314541533955979929907481056088 28



Fonction de hachage cryptographique

Tous les mots d'au plus 3 lettres à partir de l'alphabet { a,b,c }.



$\Longrightarrow$ ressemble à une **distribution aléatoire** !

Une **fonction de hachage** est une fonction permettant d'obtenir un **résumé** d'un texte, c-à-d une suite de caractères assez courte représentant le texte qu'il résume.

La fonction de hachage doit être :

- telle qu'elle associe **un et un seul résumé** à un texte en clair (cela signifie que la moindre modification du document entraine la modification de son résumé), c-à-d «sans collision».
- une fonction **à sens unique**, «*one-way function*», afin qu'il soit **impossible** de retrouver le message original à partir du résumé.

$y = H(x)$, mais il est impossible de retrouver x à partir de y !

Propriétés

une fonction de hachage H transforme une entrée de données d'une dimension variable m etdonne comme résultat une sortie de données inférieure et fixe h ($h = H(m)$).

- l'entrée peut être de dimension variable ;
- la sortie doit être de dimension fixe ;
- $H(m)$ doit être relativement facile à calculer ;
- $H(m)$ doit être une fonction à sens unique ;
- $H(m)$ doit être «sans collision».

Utilisation - Authentification et intégrité

Les algorithmes de hachage sont utilisés :

- pour la vérification si un document a été modifié (le changement d'une partie du document change son empreinte), on parle **d'intégrité** ;
- dans la génération des **signatures numériques**, dans ce cas, le résultat h est appelé «empreinte» en le combinant avec les propriétés des chiffrements asymétriques.

Résistance faible au collision

Alice vient de terminer un document M, et vient de transmettre son empreinte $H(M)$ obtenue par une fonction de hachage.

Bob grâce à l'empreinte transmise pourra s'assurer, lorsqu'il récupérera M, que M n'a pas été modifié.

Exemple : la récupération de logiciel sous Linux : l'archive se télécharge sur différents sites et une empreinte est indiquée sur le site site Web officiel pour la vérification de son intégrité.

Si Oscar veut changer le document M par un document M', il doit chercher un document M' tel que $H(M') = H(M)$, c-à-d que les deux documents possèdent la **même empreinte**.

Ce qui doit être très difficile : nécessiter une recherche exhaustive sur plusieurs années par exemple.

Résistance forte au collision

Oscar aimerait pouvoir tromper la vigilance d'Alice dans la signature et le dépôt d'un document M :

- ▷ Il cherche deux documents M et M' ayant la **même empreinte** mais des sens différents : M : un contrat équitable avec Alice, et M' : un contrat à son unique avantage ;
- ▷ Il montre à Alice le document M, d'empreinte $H(M)$, pour lequel elle donne son accord par signature et elle conserve $H(M)$ pour s'assurer à l'avenir d'avoir donner son accord pour le bon document ;
- ▷ puis lorsque plus tard, Alice voudra présenter le document M pour faire valoir son contrat avec Oscar, Oscar lui substituera le document M'
- ▷ Alice voit M' et refuse cet autre contrat.
  Mais Alice est **piégée** car $H(M) = H(M')$ = empreinte qu'elle a conservé (ou bien qu'elle a signé) !

Pour se protéger ?

Alice devrait **légèrement modifier** le document M **sans modifier son sens** avant de l'accepter et d'en mémoriser l'empreinte (ou de la signer) !

Principaux algorithmes

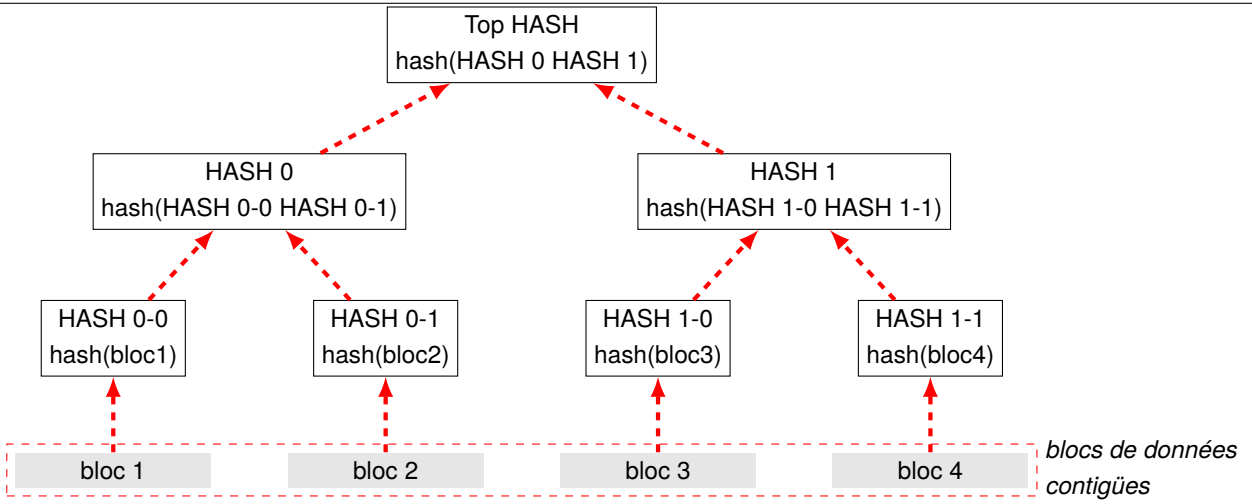
Il existe différents algorithmes réalisant de traitement :

- MD2, MD4 et **MD5** (MD signifiant «*Message Digest*»), développé par Ron Rivest (société RSA Security), créant une empreinte digitale de 128 bits pour MD5.
  Il est courant de voir des documents en téléchargement sur Internet accompagnés d'un fichier MD5, il s'agit du résumé du document permettant de vérifier l'intégrité de ce dernier.
  Son usage est maintenant déconseillé car il possible de choisir la valeur de l'empreinte obtenue.
- **SHA**,«*Secure Hash Algorithm*», pouvant être traduit par Algorithme de hachage sécurisé
  développé par le NIST en 1995. il crée des empreintes d'une longueur de 160 bits.
  C'est un standard SHA0 et SHA1 (devenu le standard SHS).
  Son usage est maintenant déconseillé car il possible de choisir la valeur de l'empreinte obtenue.
  Il a été étendu en SHA256 ou SHA512.
- **RIPEMD** «*Race Integrity Primitives Evaluation Message Digest*», développé par Hans Dobbertin, Antoon Bosselaers et Bart Preneel,
  RIPEMD-128 et RIPEMD-160, créé entre 88 et 92 ;
- **Tiger**, développé par Ross Anderson et Eli Biham, plus rapide que MD5 (132Mb/s contre 37Mb/s sur une même machine, optimisé pour processeur 64bit).

Une forme particulière de fonction de hachage : MAC ou HMAC

Combiner Intégrité + chiffrement symétrique : MAC, «*Message Authentication code*», code d'authentification de message.
Combiner une fonction de **hachage** et une **clé secrète** : HMAC, «*keyed-hash message authentication*».
On transmet l'empreinte du message chiffrée avec une clé secrète qui protège contre toute modification du message : si l'intrus modifie le message et ne connait pas la clé, il ne peut créer un MAC correspondant au nouveau document (le document peut, lui-même, être transmis chiffré ou non).

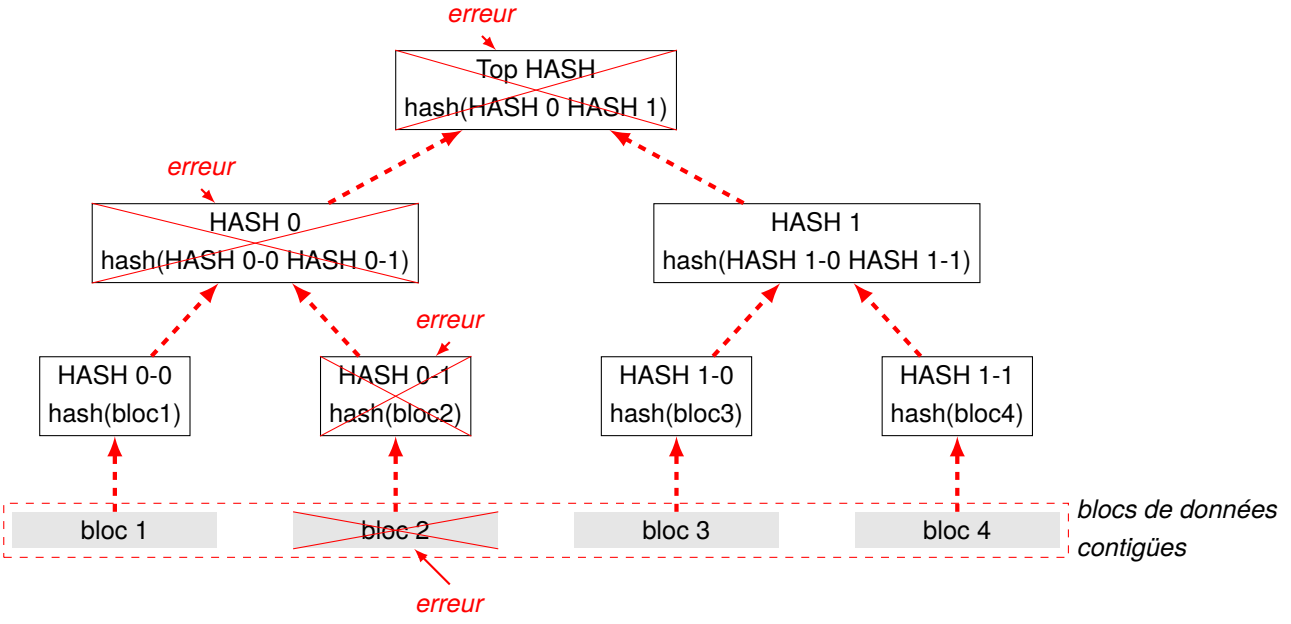
Cela peut servir à **l'intégrité**
de documents **distribués**



L'utilisation de cette arbre permet **d'isoler une erreur** dans un fichier transféré d'Alice vers Bob :

1. Bob veut vérifier que le fichier reçu depuis Alice est sans erreur : il calcule l'abre de merkle sur sa copie de fichier ;
2. Alice transmets le haché Top HASH à Bob, c-à-d celui de la racine de l'arbre ;
3. Bob compare le haché qu'il a calculé avec celui qu'il a reçu d'Alice :
   - ◇ identique ? Le fichier a été transferré **sans erreur**.
   - ◇ différent ? Bob demande à Alice les hachés de la racine de chaque sous-arbre : HASH 0 et HASH 1 ;
4. pour chacun de ses hachés, HASH 0 et HASH 1, Bob peut vérifier par rapport à la valeur qu'il a lui-même calculé dans son abre de Merkle :
   - ◇ dès qu'il trouve une différence de haché, il peut identifier le sous-arbre concerné et descendre jusqu'à la vérification d'une **feuille** de l'arbre, c-à-d le haché de chaque bloc de données en comparant avec la version d'Alice ;
   - ◇ dans le cas où le haché d'un bloc de données n'est pas bon, il peut demander à Alice de lui retransmettre les données de ce bloc uniquement.

Détection d'une erreur et identification du ou des blocs concernés



Si une erreur se produit sur le bloc 2 :

- ▷ Top HASH différent de celui d'Alice ⟹ Bob demande le haché de **chaque sous-arbre** ;
- ▷ HASH 0-1 différent ⟹ Bob demande le haché de **chaque sous-arbre** ;
- ▷ HASH 0 différent ⟹ la feuille correspondant au haché du bloc bloc 2 ⟹ le bloc 2 doit être **retransmis** par Alice ou par **quiconque en possédant une copie** ;

⟹ **application au réseau «*peer-to-peer*».**

Et les fonctions de hachage...

C’est «*cassable*» ?

Authentification basée sur un secret partagé : accès protégé par mot de passe

Pour accèder à un système **protégé par mot de passe** :

- ▷ l'utilisateur choisit un «bon» mot de passe (pour rendre une attaque «*brute force*» difficile) ;
- ▷ le mot de passe est **conservé sur le système** sous forme d'une **valeur «hachée»** par une fonction de hachage H, haché $= H(\text{MotDePasse})$ (pour **éviter de pouvoir récupérer le mot de passe** depuis le fichier qui le contient).
- ▷ lors de la tentative d'authentification :
  - ◇ l'utilisateur saisi son mot de passe ;
  - ◇ le mot de passe est haché par H ;
  - ◇ la valeur hachée obtenue est **comparée** à celle mémorisée : si elle est identique, l'utilisateur est **autorisée**.

Attaque par construction d'une table de chaîne de hachés

L'attaque repose sur le fait que la valeur hachée du mot de passe **a pu être récupérée** (lecture du fichier les contenant).

Trouver une valeur utilisable comme mot de passe qui produise par H la valeur hachée voulue :

- □ construire une table contenant **tous les mots de passe possibles** et toutes les valeurs hachées associées ;
- □ faire une recherche inversée dans cette table.

Si on veut traiter **tous les mots de passe** :

$\Longrightarrow$la table serait **trop grande** à mémoriser ou **trop longue** à regénérer.

$\Longrightarrow$stocker seulement une sélection de hachés qui permettent d'obtenir une chaîne de mots de passe :

- ▷ utiliser une fonction R de réduction qui permet de mapper un haché vers un mot de passe ;
  Attention : ce n'est pas une fonction d'inversion de la fonction de hachage
- ▷ fabriquer des chaînes alternant mots de passe et hachés :
  - ◇ choisir un certain nombre de mots de passe initiaux de manière aléatoire ;
  - ◇ pour un de ces mots de passe alterner n fois l'application de H, puis de R, puis de H, *etc.* et enfin de R
  - ◇ mémoriser uniquement le mot de passe initial et le dernier mot de passe de cette chaîne.
- ▷ pour «inverser» un haché donné, c-à-d trouver un mot de passe permettant de l'obtenir :
  - ◇ on applique R, puis H, *etc.* jusqu'à obtenir un mot de passe de fin de chaîne : il y a une **forte probabilité** que le haché soit contenu dans la chaîne correspondante et que la valeur immédiatement précédente **soit le mot de passe que l'on cherche** !

Construction d'une chaîne et recherche d'un mot de passe correspondant au haché

- On choisit aléatoirement le mot de passe initial `aaaaaa` pour un ensemble de mot de passe constitué de 6 lettres.
- On construit la chaîne :

$$\texttt{aaaaaa} \overset{H}{\rightarrow} \texttt{281DAF40} \overset{R}{\rightarrow} \texttt{sgfnyd} \overset{H}{\rightarrow} \texttt{920ECF10} \overset{R}{\rightarrow} \texttt{kiebgt}$$

- On mémorise la chaîne :

début	fin
aaaaaa	kiebgt

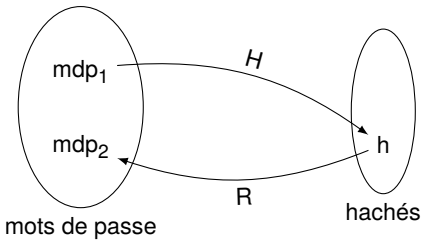
- On recherche maintenant à «casser» le haché `920ECF10` :
  - On applique R : $\texttt{920ECF10} \overset{R}{\rightarrow} \texttt{kiebgt}$ et on cherche le mot de passe `kiebgt` dans la colonne «fin» de la table.
  - On trouve qu'il est présent et que le mot de passe de début est `aaaaaa` : on reconstruit la chaîne jusqu'à obtenir ce haché :

$$\texttt{aaaaaa} \overset{H}{\rightarrow} \texttt{281DAF40} \overset{R}{\rightarrow} \texttt{sgfnyd} \overset{H}{\rightarrow} \texttt{920ECF10}$$

  - Alors le mot de passe est `sgfnyd` !

Formalisation

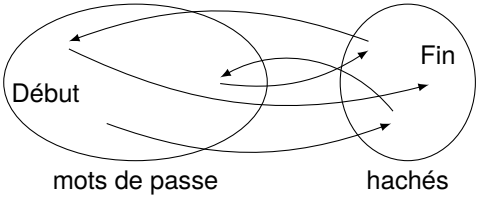
- Soit la fonction H utilisée par le système ;
- Soit un ensemble P de mots de passe possibles, c-à-d accepté par le système ;
- Soit une fonction R de réduction, c-à-d permettant de transformer un haché produit par H en un élément de P.



Exemple :

- $\text{MD}_5(493823) \longrightarrow \texttt{83312542f562bd072f559d9701a45f04}$
- $R(\texttt{83312542f562bd072f559d9701a45f04}) \longrightarrow 222004$

Construction et stockage des séquences $H \Longrightarrow R \Longrightarrow H$

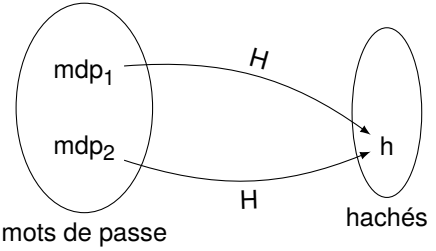


Obtention et mémorisation d'une chaîne :

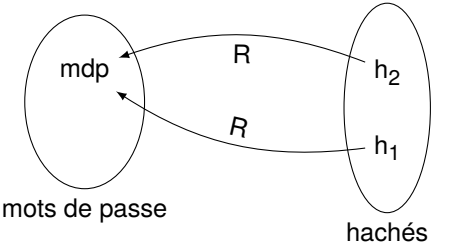
- on choisit un mot de passe qui servira de **début** à la chaîne ;
- on applique une séquence de n opérations de $H \Longrightarrow R$;
- on obtient un haché h de **fin** ;
- on mémorise le mot de passe de début et le mot de passe obtenu sur $R(h)$.

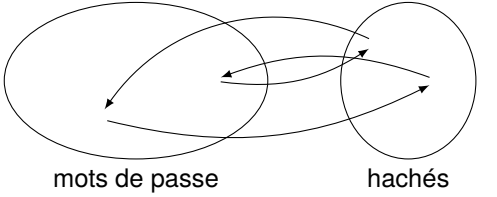
Problème de collisions

Sur la fonction de hachage :



Sur la fonction de réduction :

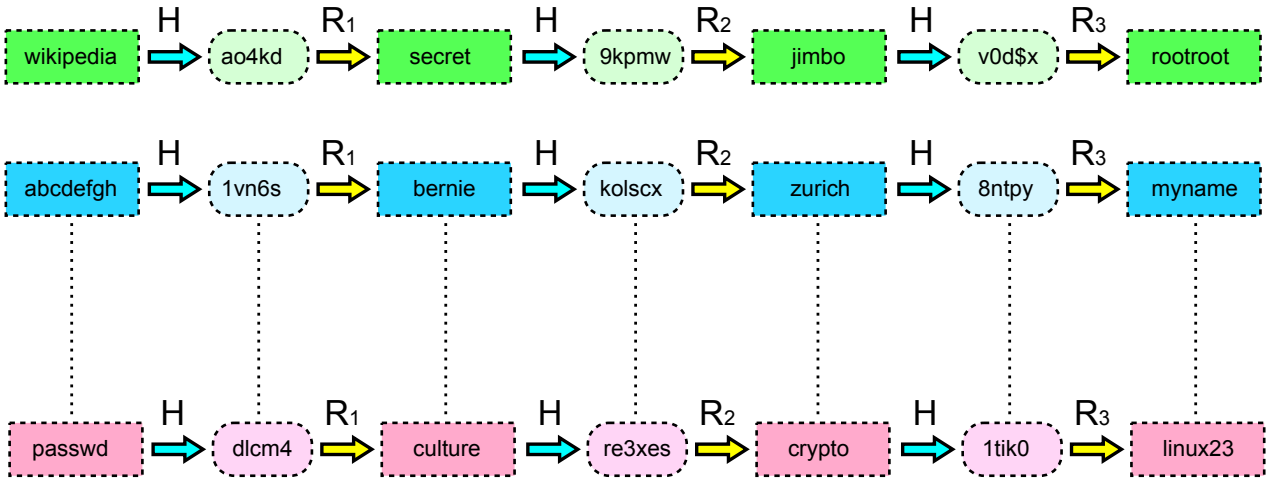




Il peut être difficile de :

- ▷ couvrir tous les mots de passe possibles ;
- ▷ d'éliminer les chaînes conduisant à la même terminaison ;
- ▷ de découvrir lors de la recherche que l'on est dans une boucle : dans ce cas là la chaîne est détruite et les hachés apparaissant dans la boucle ne seront pas traités dans la table.

Solution : la Rainbow table



- ▷ Toujours mémoriser uniquement le mot de passe de début et celui de la fin de la chaîne ;
- ▷ utiliser différentes fonctions de réductions R_1, R_2, R_3, *etc.* pour chaque colonne de la table : d'où l'appellation de «*Rainbow table*» ;

L'utilisation de **différentes fonctions de réduction** permet de :

- ▷ **éviter les collisions** dans la mesure où elle devrait se produire sur la même colonne ;
- ▷ **couvrir plus** de mots de passe.

Protection ?

On utilise la **méthode du «SALT»** qui, même s'il est **connu de tous et lisible** dans le fichier contenant les mots de passe, **impose d'utiliser un motif** dans le mot de passe ce qui empêche l'utilisation de «rainbow tables» génériques.

Les fonctions de hachage ?

Ça peut garantir l'intégrité, mais aussi l'identité !

Un MAC est une étiquette, «*tag*», utilisée lors de la réception d'un message pour :

- ▷ **authentifier** la provenance du message ;
- ▷ garantir que le message n'a **pas été modifié** ;

Il utilise :

- □ un algorithme de sélection de **clé aléatoire** ;
- □ un algorithme de «***signature***» qui retourne une étiquette à partir de la clé et du message ;
- □ un algorithme de **vérification** efficace permettant d'accepter le message (authenticité et intégrité) ou le le rejetter si celui-ci a été modifié ou s'il ne peut être authentifié.

Exemple : le MIC, «*Message Integrity Code*», utilisé dans WPA.

HMAC,«*hash-based message authentication code*»

$$HMAC(K,m) = H((K' \oplus opad) \| H((K' \oplus ipad) \| m))$$

où $K' = zpad \oplus \begin{cases} H(K) & \text{si } K \text{ est plus large que la taille du bloc} \\ K & \text{sinon} \end{cases}$

Où :

- ▷ H est la fonction de hachage ;
- ▷ m est le message ;
- ▷ K est la clé secrète ;
- ▷ $\|$ est la concaténation ;
- ▷ K' est une clé dérivée de la clé secrète afin d'atteindre la taille du block utilisé en utilisant
  - ◇ du «*padding*», bourrage, vers la droite avec des zéros, $zpad$, si elle est de taille plus petite que le bloc ;
  - ◇ ou en la hachant d'abord, puis si nécessaire en utilisant du padding vers la droite avec des zéros ($zpad$) ;
- ▷ $opad$, «*outer padding*», ou bourrage externe, répétition de `0x5c` à la taille du bloc | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 |
- ▷ $ipad$, «*inner padding*», ou bourrage externe, répétition de `0x36` à la taille du bloc | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 |

En utilisant SHA1, la taille du bloc est de 20 octets.

Les fonctions de hachage ?

Ça crée des clés robustes

PBKDF2, «*Password Based Key Derivation Function*», RFC 8018

Utilisation de :

- une PRF, «*Pseudo Random Function*» ;
- un SALT ;
- une répétition de la même opération pour *allonger* le temps de traitement

$\Longrightarrow$ protection contre les attaques «*brute-force*»

$DK = PBKDF2(PRF, Motdepasse, Salt, c, dkLongueur)$ où :

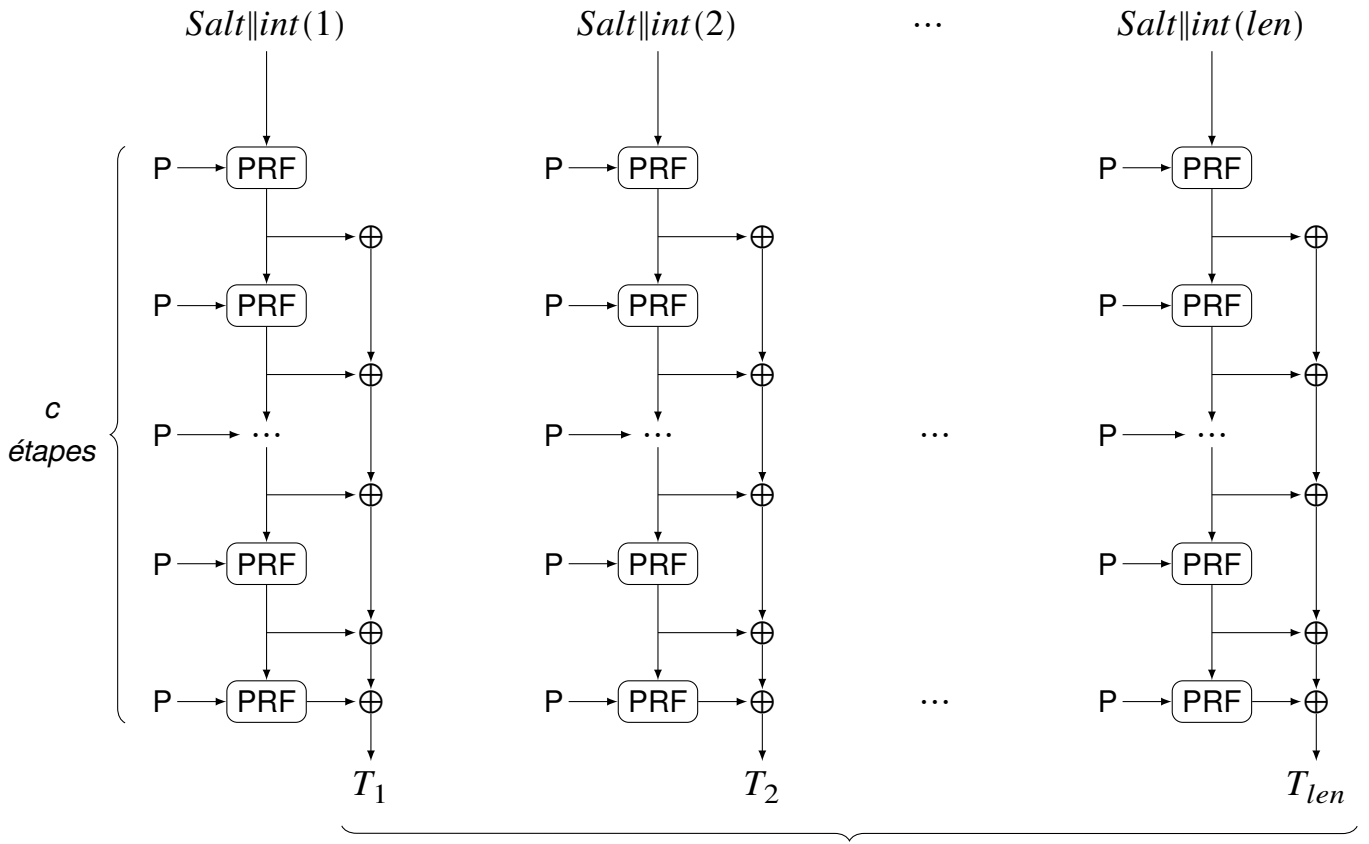
- ▷ DK clé dérivée ;
- ▷ c nombre d'itération ;
- ▷ $dKLongeur$ taille de la clé en bits ;

$DK = T_1 \| T_2 \| ... \| T_{\frac{dkLongueur}{hLongueur}}$ ($hlongueur$ est la taille en bits de la sortie de PRF).

Où chaque T_i est calculé avec une fonction F définie comme :
$T_i = F(Motdepasse, Salt, c, i) = U_1 \oplus U_2 \oplus ... \oplus U_c$ avec les U_j calculés comme :
$U_1 = PRF(Motdepasse, Salt \| INT\_32\_BE(i)) \Longrightarrow$ *entier sur 32 bits en notation big endian...*
$U_2 = PRF(Motdepasse, U_1)$
...
$U_c = PRF(Motdepasse, U_{c-1})$

Exemple pour WPA2 : $DK = PBKDF2(\text{HMAC-SHA1}, motdepasse, ssid, 4096, 256)$ où $ssid$ est l'identifiant du réseau WiFi et $motdepasse$ est la «clé humaine» WPA2.

Dérivation de clé symétrique depuis un mot de passe



$DK = T_1\|T_2\|...\|T_{len}$ où $len = \frac{dkLongueur}{hLongueur}$ et P est le $Motdepasse$

Le fichier «/etc/passwd» contient uniquement les infos correspondant au compte de l'utilisateur :

```
xterm
pef@cube:/etc$ cat passwd | grep pef
pef:x:1000:1000:pef,,,:/home/pef:/bin/bash
```

Le contenu de ce fichier doit être accessible par un grand nombre d'outils qui ne doivent pas avoir les droits de l'utilisateur «root»⟹ il est lisible par tous, ***mais*** *mais ne contient pas les «mots de passe»*

Le mot de passe de l'utilisateur est dans le fichier «/etc/shadow», lisible uniquement avec les droits «*root*» :

```
xterm
pef@cube:/etc$ sudo cat shadow | grep pef
pef:$6$x5dLHSND$k.b4lZyenwwY9Z12CwbK884PRpP4y9wk//K8PluEUmgmjwLsX2XWhPNkzlZjfn
QoxOyua2xT7BXo4QZ49Dtdg0:17192:0:99999:7:::
```

Il est possible de le recréer à partir du SALT choisi par le système, ici «`x5dLHSND`», et la méthode choisie pour haché le mot de passe indiquée ici par «`$6$`» :

```
xterm
pef@cube:/etc$ python -c "import crypt; print crypt.crypt('toto', '\$6\$x5dLHSND\$')"
$6$x5dLHSND$k.b4lZyenwwY9Z12CwbK884PRpP4y9wk//K8PluEUmgmjwLsX2XWhPNkzlZjfn
QoxOyua2xT7BXo4QZ49Dtdg0
```

La méhode «`$6$`» correspond à utiliser un haché de haché avec sha-512 et une répétition de 5000 hachés où l'on combine la valeur du haché précédent avec le SALT.

Si on utilise openSSL :

```
xterm
pef@cube:/etc$ openssl passwd -1 -salt x5dLHSND toto
$1$x5dLHSND$gfRJNaMxe8sJzJ9RoYP0m.
```

La méthode utilisée est «`$1$`» correspondant à MD5.

On peut aussi utiliser la commande suivante :

```
xterm
pef@cube:/etc$ mkpasswd  -m sha-512 -S x5dLHSND -s toto
$6$x5dLHSND$k.b4lZyenwwY9Z12CwbK884PRpP4y9wk//K8PluEUmgmjwLsX2XWhPNkzlZjfn
QoxOyua2xT7BXo4QZ49Dtdg0
```

Peut-on combiner Authentification et Intégrité pour un document ?

Le scellement ou sceau ou signature électronique

Signer : **joindre** à un document sa **signature**, c-à-d le **chiffré asymétrique par clé privée**, de l'empreinte du document, obtenue à l'aide d'une fonction de **hachage**.

La **signature** assure :

- ▷ **Authentification** : le document peut être **identifié** comme provenant de la personne ;
- ▷ **Non-répudiation** : l'utilisation de la **clé privée** ne peut être faite sans le **consentement** de son propriétaire ;
- ▷ **Intégrité** : la correspondance de l'empreinte déchiffrée, avec celle courante du document implique qu'il n'y a pas eu de modification par rapport à la version du document utilisée lors du chiffrement.

La **confidentialité** peut être assurée par un **chiffrement symétrique** du document.

Utilisation pour la sécurité du courrier électronique

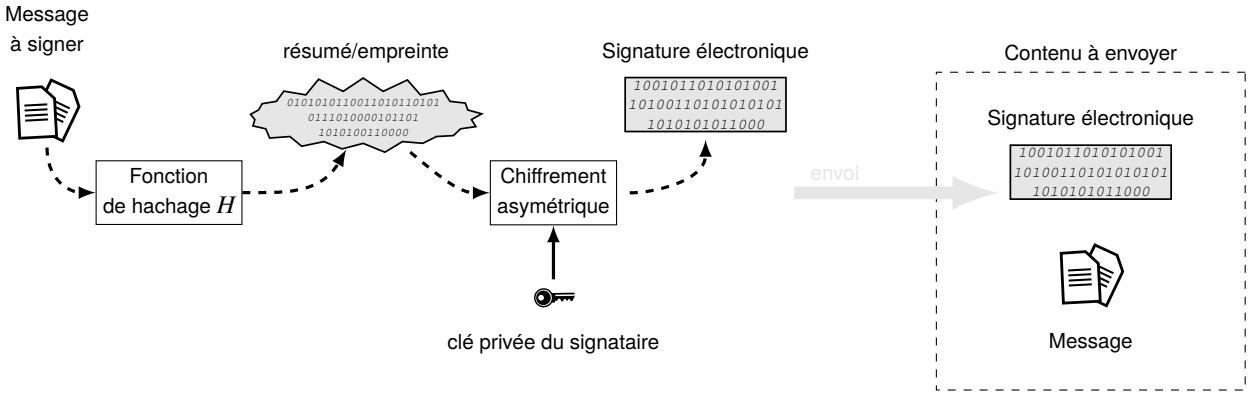
1. L'expéditeur calcule **l'empreinte** de son **texte en clair** à l'aide d'une fonction de hachage ;
2. L'expéditeur chiffre **l'empreinte** avec sa **clé privée** $\Longrightarrow$ signature ;
3. *Le chiffrement du document est **optionnel** si la confidentialité n'est pas nécessaire.*
4. L'expéditeur chiffre le **texte en clair** et **la signature** à l'aide d'un **chiffrement symétrique** dont la **clé secrète** est chiffrée à l'aide de la **clé publique du destinataire** et jointe au message (enveloppe sécurisée).
5. L'expéditeur envoie le **document chiffré** au destinataire ;
6. Le destinataire **déchiffre la clé secrète symétrique** avec **sa clé privée**, puis déchiffre le document ;
7. Le destinataire déchiffre **l'empreinte** avec la clé publique de l'expéditeur (*authentification*) ;
8. Le destinataire calcule **l'empreinte du texte clair** à l'aide de la même fonction de hachage que l'expéditeur ;
9. Le destinataire **compare les deux empreintes**.

optionnel

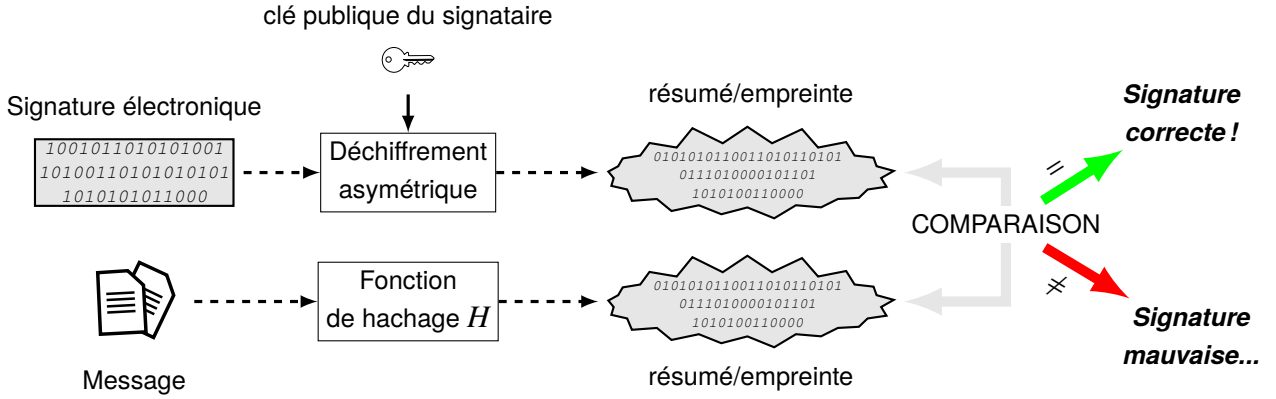
Deux empreintes identiques impliquent que le texte en clair n'a pas été modifié (intégrité).

Le standard américain est le **DSS**, «*Digital Signature Standard*», qui spécifie trois algorithmes : le DSA, «*Digital Signature Algorithm*», **RSA** et **ECDSA** «*Elliptic Curves Digital Signature Algorithm*».

Créer une signature électronique



Vérifier une signature électronique



Cryptographie basée sur les courbes elliptiques

- la **clé privée** est une **valeur entière aléatoire** d, choisie entre 1 et n, où n est **l'ordre du groupe** ;
- la **clé publique** est le point $H = dG$ où G est le **générateur** définissant le groupe ;

Garantie :

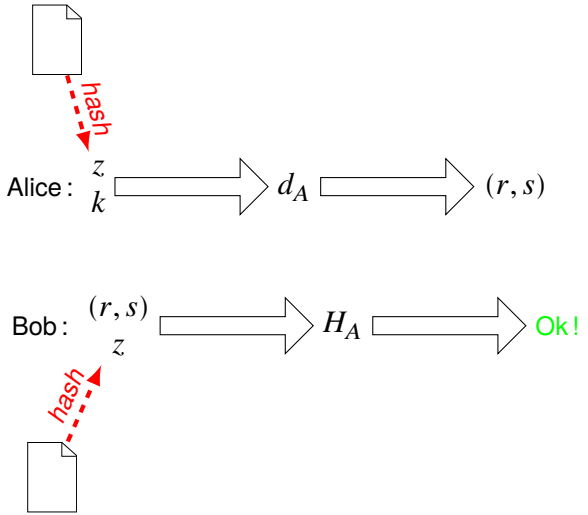
- Si l'on connait d et G, trouver H est facile ;
- Si l'on connait H et G, trouver la clé privée d est difficile $\Longrightarrow$ résoudre le *problème du logarithme discret*...

Signature ECDSA

- Alice veut **signer** un message avec sa clé privée d_A ;
- Bob veut **valider** la signature avec la clé publique d'Alice H_A ;

Fonctionnement :

- on calcule le **hash** du document à signer ;
- on tronque le **hash** de telle manière que le **nombre de bits** soient de même taille que n **l'ordre** du groupe et on appelle z ce nombre ;
- on prend une **valeur entière aléatoire** k entre 1 et $n-1$ où n est l'ordre du groupe ;
- on calcule $P = kG$ où G est le **générateur** du groupe ;
- on calcule $r = x_p \bmod n$ où x_p est la coordonnée suivant l'axe x, ou l'abscisse, de P ;
- *si* $r = 0$ *on choisit un autre* k *et on ré-essaye* ;
- on calcule $s = k^{-1}(z + r * d_A) \bmod n$ où d_A est la clé privée d'Alice et k^{-1} est l'inverse de k pour la multiplication modulo n ;
- *si* $s = 0$ *on choisit avec un nouveau* k *et on ré-essaye* ;
- (r, s) est la **signature** !



L'algorithme de signature :

- ▷ génère un **secret** k ;
- ▷ ce secret est «*caché*» dans r grâce à la **multiplication de point** (facile dans un sens, dure dans l'autre) ;
- ▷ r est «*attaché*» au haché du document par l'équation $s = k^{-1}(z + r * d_A) \bmod n$;
- ▷ pour pouvoir calculer l'inverse de k modulo n, il faut que n soit un nombre premier (sinon pas de groupe !)
- ▷ n est connu d'Alice et Bob, il fait partie du système cryptographique avec a et b de la courbe elliptique
  $\Longrightarrow$ il existe **différents systèmes standardisés** et **prouvés sûrs**.

Vérification de la signature

- ▷ on calcule $u_1 = s^{-1}z \bmod n$ et $u_2 = s^{-1}r \bmod n$;
- ▷ on calcule le point $P = u_1G + u_2H_A$ et la signature est valide seulement si $r = x_p \bmod n$.

Pourquoi ? $P = u_1G + u_2H_A = u_1G + u_2d_AG = (u_1 + u_2d_A)G = (s^{-1}z + s^{-1}r * d_A)G = s^{-1}(z + r * d_A)G$
Et comme $s = k^{-1}(z + r * d_A) \bmod n$ ou $k = s^{-1}(z + r * d_A) \bmod n$, on obtient alors $P = kG$.

C'est quoi un inverse modulaire ?

L'inverse modulaire de a est un entier i tel que

- ◦ $a * i \equiv 1 \bmod n$;
- ◦ $i \equiv a^{-1} \bmod n$.

Exemple : $i \equiv 3^{-1} mod 11$ ou $3 * i \equiv 1 \bmod 11$, ce qui donne $i = 4$ car $3 * 4 = 12 \equiv 1 \bmod 11$.

ECDSA : à quoi cela ressemble ?

xterm

```
pef@darkstar-8:…/ecc/scripts$ python3 ecdsa.py
Curve: secp256k1
Private key: 0x28ecb4b7673815c2b0e49e3ad83e8bbb20ec5fa585f15fd76df01ed4064aa796
Public key:
(0xee19ee92a0e0d1b6271ade3b5e62d65fcd367e0405fa846a3123560aa152709e,
0xc68662c15eb123634652142af365f6798e7f960ca3b21df532c558a182427cc6)

Message: 'Bonjour !'
Signature: (0x737f1035e1de3e0474f3f4f46b10c661ecd6efa4329754aef5d6cdd7a61ec86a,
0xa80ba620a96da5d52a3a6a17089ceae699dd4f17eba7dd5a04500e0a5a43cd50)
Verification: signature matches

Message: 'Coucou !'
Verification: invalid signature

Message: 'Bonjour !'
Public key:
(0xcae60f6d2c9b80fa8b4b0d4afefaca78950933f0258ca6f66cf8be197a7d4184,
0xc9d273e354da7e989f94d2046f4327db6fba06c499afdeb1015709812781f2d0)
Verification: invalid signature
```

même message et la clé publique est la bonne

le message est différent

la clé publique est différente

- la courbe utilisée dans Bitcoin est «*secp256k1*» ;
- La taille de la clé privée est de 32 octets ;
- la clé publique est un point dont chaque coordonnée est sur 32bits ;
- le message est le contenu du document qui va être haché pour donner la valeur z en entrée de l'algorithme.

La société RSA a établit des **standards** pour l'organisation des **échanges cryptographiques** et permettre l'interopérabilité «*PKCS*» ou «*Public Key Cryptographic Standards*» :

- **PKCS#1** : décrit comment chiffrer des données en utilisant l'algorithme à clé publique RSA, afin de pouvoir signer un fichier ou le chiffrer :
  - pour les signatures, le contenu doit d'abord être «hashé» par un algorithme de hachage afin de produire une empreinte.
    C'est cette empreinte qui sera signée avec l'algorithme RSA en utilisant la clé privée du signataire. *La manière de génèrer et signer une empreinte est décrite dans PKCS#7.*
  - pour le chiffrement d'un fichier, le fichier doit d'abord être chiffré par un algorithme à clé secrète comme AES et la **clé secrète** est ensuite elle-même **chiffrée** avec l'algorithme **RSA** en utilisant la **clé publique du destinataire** (il sera donc le seul à pouvoir déchiffrer la clé secrète avec sa clé privée).
    Le chiffrement du fichier et le chiffrement de la clé sont décrits dans PKCS#7.
  - PKCS#1 définit également différents algorithmes de hachage, MD5, SHA1, SHA256.
- **PKCS#3** : standard de **négociation de clé Diffie-Hellman** :
  - PKCS#3 décrit comment implémenter l'algorithme Diffie-Hellman qui sert à «négocier» un secret partagé entre deux parties, sans qu'aucune information privée n'ait à être échangée.
  - Ce secret partagé peut servir à générer une clé de session, qui pourra être utilisée dans un algorithme à clé secrète comme AES.
- **PKCS#7** : Standard de la syntaxe pour un message chiffré, on parle «d'enveloppe sécurisée»
  PKCS#7 décrit une syntaxe générale pour les données devant être chiffrées, comme les signatures numériques par exemple.
  Cette syntaxe supporte la récursivité, ce qui permet de signer un fichier déjà signé par quelqu'un d'autre par exemple.

Et l'utilisation dans
les protocoles de communication ?
Comment modéliser et garantir
qu'un protocole de communication est sécurisé ?

- **l'écoute** des communications, «*eavesdropping*» :
  - écouter de manière passive les messages ;
  - la protection consiste à utilise le chiffrement pour assurer la confidentialité ;
- la **modification** :
  - l'attaquant modifie ou replace des messages ;
  - la protection utilise le chiffrement ;
- le **rejeu**, «*replay*» :
  - l'attaquant envoie un message qui a été échangé précédemment ;
  - la protection est de définir le message clairement afin d'empêcher son utilisation hors contexte ;
- l'attaque de **l'intermédiaire**, «*Man-in-the-Middle*» :
  - l'attaquant s'intègre dans les échanges entre deux interlocuteur ;
  - il peut écouter et modifier les messages échangés ;
- la **réflexion**, «*Reflection*» ;
- le **déni de service**, «*DoS*» :
  - toutes les communications utilisent une certaine quantité de CPU et de mémoire ;
  - un attaque consiste à réaliser des **millions de requêtes** au serveur afin d'épuiser ses ressources ;
  - certains protocoles sont meilleurs que d'autres pour l'éviter ;
- le mélange des **types** de messages, «*Typing Attack*».

Transmission	
$A \rightarrow B : m$	A envoie le message m à B
1. 2. ...	rang du message dans l'échange
Valeurs échangées	
N_A	valeur quelconque choisie par A
Chiffrement symétrique	
K_{AB}	la clé secrète est partagée entre A et B
$\{m\}_K$	le message m est chiffré en symétrique à l'aide de la clé secrète K
Chiffrement asymétrique	
$Sign_A(m)$	A signe le message m avec sa clé privée
$E_{K_S}(m)$	chiffrement avec la clé publique K_S du message m
Attaque	
$I(B)$	l'attaquant I «*impersonate*», prend la place de, B
❶ ❷ ...	rang du message dans l'échange d'attaque

Attaque par réflexion

- similaire à une attaque par rejeu qui utilise le protocole contre lui-même ;
- l'attaquant utilise une «preuve» d'authentification en «*challengeant*» le «*challenger*».

Exemple d'attaque par réflexion

- A et B partagent la clé K ;
- A et B veulent vérifier qu'ils sont bien mis en relation ;

Protocole :

1. $A \rightarrow B \ : \{N_A\}_K$
2. $B \rightarrow A \ : N_A, \{N_B\}_K$
3. $A \rightarrow B \ : N_B$

Attaque sur le protocole :

1. $A \rightarrow B \ : \{N_{A1}\}_K$
   - ❶ $I(B) \rightarrow A \ : \{N_{A1}\}_K$
   - ❷ $A \rightarrow I(B) \ : N_{A1}, \{N_{A2}\}_K$
2. $I(B) \rightarrow A \ : N_{A1}, \{N_{A2}\}_K$
3. $A \rightarrow I(B) \ : N_{A2}$
   - ❸ $I(B) \rightarrow A \ : N_{A2}$

1.$\Longrightarrow$ le protocole est initié de A vers B (Instance 1 du protocole) ;

$\Rightarrow$ I prend la place de B pour A, initie le protocole vers A et renvoie le message de 1) vers A (Instance 2 du protcole) ;

$\Rightarrow$ A déchiffre le message reçue de I et révèle la valeur N_{A1} à I ;

2. $\Longrightarrow$$I$ réponds à A avec la valeur N_{A1} et renvoie la valeur $\{N_{A2}\}_K$ à A et finit de s'authentifier en B auprès de A.

Attaque par déni de service

A utilise sa clé publique K_A pour établir une clé de session K_{AS} :

1. $A \rightarrow S$: A, N_A
2. $S \rightarrow A$: $E_{K_A}(N_A, N_S, K_{AS})$
3. $A \rightarrow S$: $\{N_S\}_{K_{AS}}$

S est vulnérable à une attaque par déni de service, parce que pour chaque connexion :

- ▷ il génère un **nonce** et une **clé symétrique** ;
- ▷ il réalise un **chiffrement** par clé publique ;
- ▷ il alloue de la **mémoire** pour le nonce et la clé.

Une version résistante à cette attaque

A utilise la clé publique K_S de S pour établir la clé de session K_{AS} :

1. $A \rightarrow S$: $E_{K_S}(A, S, Sign_A(N_A, K_{AS}))$
2. $S \rightarrow A$: $\{N_A\}_{K_{AS}}$

Résultat : maintenant, c'est A qui doit réaliser le chiffrement le plus coûteux.

$\Longrightarrow$ *il faudra plus d'attaquants pour réussir le DoS*.

Attaque sur le type des messages échangés

L'attaque consiste à utiliser un **type de message** à la place d'un autre type de message.

Exemple d'exploitation

Protocole :	**Attaque :**
1. $A \rightarrow B : \{N_A\}_{K_{AB}}$	1. $A \rightarrow B : \{N_A\}_{K_{AB}}$
2. $B \rightarrow A : \{N_A+1, N_B\}_{K_{AB}}$	2. $B \rightarrow A : \{N_A+1, N_B\}_{K_{AB}}$
3. $A \rightarrow B : \{N_B+1\}_{K_{AB}}$	3. $A \rightarrow B : \{N_B+1\}_{K_{AB}}$
4. $B \rightarrow A : \{K_S, N'\}_{K_{AB}}$	❹ $I(B) \rightarrow A : \{N_A+1, N_B\}_{K_{AB}}$

▷ L'attaquant I rejoue le message de type 2 ;

$\Longrightarrow A$ va utiliser une mauvaise clé... que l'attaquant peut déterminer si N_A est **prédictible**...

Peut-on apprendre de ces attaques
et en tirer des règles de conception ?

Le meilleur moyen d'éviter les **faiblesses** dans un **protocole** est de le **définir correctement** dès le départ !

1 Le protocole doit être **efficace** :

- ◇ pas de chiffrement inutile ;
- ◇ ne pas inclure de messages inutiles ;

Exemple sur Kerberos :

1. $A \rightarrow S : A, B, N_A$
2. $S \rightarrow A : \{K_{AB}, B, L, N_A, \{K_{AB}, A, L\}_{K_{BS}}\}_{K_{AS}}$
3. $A \rightarrow B : \{A, T_A\}_{K_{AB}}, \{K_{AB}, A, L\}_{K_{BS}}$
4. $B \rightarrow A : \{T_A + 1\}_{K_{AB}}$

1. $A \rightarrow S : A, B, N_A$
2. $S \rightarrow A : \{K_{AB}, B, L, N_A\}_{K_{AS}}, \{K_{AB}, A, L\}_{K_{BS}}$
3. $A \rightarrow B : \{A, T_A\}_{K_{AB}}, \{K_{AB}, A, L\}_{K_{BS}}$
4. $B \rightarrow A : \{T_A + 1\}_{K_{AB}}$

Optimisation : *supprimer le double chiffrement*

2 Chaque message doit être **compréhensible** uniquement à partir de son contenu.

Il doit être possible de décrire en une simple et unique phrase la signification d'un message.

1. $A \rightarrow B : E_B(N_A, A)$
2. $B \rightarrow A : E_A(N_A, N_B)$
3. $A \rightarrow B : E_B(N_B)$

1) $E_X(Y, Z)$ signifie «Z *veux communiquer avec* X *en utilisant* Y»

2) $E_\alpha(\beta, \gamma)$ signifie «*??* α *veut communiquer avec* β *et* γ»

$\Longrightarrow E_A(N_A, N_B)$ ne signifie pas que B veut communiquer avec A en utilisant N_A et N_B parce qu'il n'y a pas de référence à B

Correction :

1. $A \rightarrow B : E_B(N_A, A)$
2. $B \rightarrow A : E_A(N_A, N_B, B)$
3. $A \rightarrow B : E_B(N_B)$

- ◇ Le message 2) $E_X(Y, Z, W)$ signifie «W *veut communiquer avec* X *en utilisant* Y *et* Z» ;
- ◇ le message 3) $E_X(Y)$ signifie «*??* X *accepte la communication avec* X *en utilisant* Y.

  Ici, on a pas besoin de mentionner A*, car seulement* A *connaît* N_B».

3 Les conditions pour qu'un message soit **pris en compte** ou **ignorer** doivent être exprimer clairement pour une compréhension facile d'un auditeur.

4 Si **l'identité** d'un interlocuteur est **nécessaire** pour la bonne compréhension d'un message, il est prudent **d'inclure son identité** dans le message.

Authentification de X pour Y au travers d'un serveur de confiance S

Protocole

1. $A \rightarrow B : A$
2. $B \rightarrow A : N_B$
3. $A \rightarrow B : \{N_B\}_{K_{AS}}$
4. $B \rightarrow S : \{A, \{N_B\}_{K_{AS}}\}_{K_{BS}}$
5. $S \rightarrow B : \{N_B\}_{K_{BS}}$

$\Longrightarrow A$ est authentifié pour B.

Attaque permettant à I d'impersoner A

1. $I(A) \rightarrow B : A$
   - ❶ $I \rightarrow B : I$

   N_B *relatif à* A
2. $B \rightarrow I(A) : N_B^A$
   - ❷ $B \rightarrow I : N_B^I$

   N_B *relatif à* I
3. $I(A) \rightarrow B : \{N_B^A\}_{K_{IS}}$
   - ❸ $I \rightarrow B : \{N_B^A\}_{K_{IS}}$
4. $B \rightarrow S : \{A, \{N_B^A\}_{K_{IS}}\}_{K_{BS}}$
   - ❹ $B \rightarrow S : \{I, \{N_B^A\}_{K_{IS}}\}_{K_{BS}}$
5. $S \rightarrow B : Fail \Longrightarrow$ le message doit être intercepté par I...
   - ❺ $S \rightarrow B : \{N_B^A\}_{K_{BS}}$

5'. $I(S) \rightarrow B : \{N_B^A\}_{K_{BS}} \Longrightarrow I$ est authentifié comme A auprès de B !

2 Le **chiffrement** :
- ◇ est **coûteux** en terme de performance ;
- ◇ doit être justifié, sinon il peut être **redondant** ou **inutile** ;
- ◇ n'est **pas synonyme de sécurité** et son mauvais usage peut conduire à des erreurs.

- **6** Lorsqu'un interlocuteur **signe un message déjà chiffré**, il ne faut pas assumer que l'interlocuteur **connaisse le contenu** de ce message ;
- □ Inversement, si l'interlocuteur **signe un message** et **ensuite le chiffre**, il doit **connaître** le contenu de ce message :
  $A \rightarrow B : A, Sign_A(T_A, N_A, B, X_A, E_B(Y_A))$
  $\Longrightarrow A$ connait la donnée X_A, mais pas forcément la donnée Y_A qui est secrète.
- **7** Les propriétés attendues des «***nonces***» doivent être exprimées clairement :
  - ◇ ce qui en assure la **succession dans le temps**, peut **ne pas** en **garantir l'association** à un interlocuteur donné ;
  - ◇ cette **association** peut être nécessaire et doit être assurée par un autre moyen ;
- **8** L'utilisation d'une **valeur prédictible** comme celle d'un compteur peut en garantir la **nouveauté** lors d'un «*challenge/response*»
- □ Mais si une valeur prédictible doit être efficace, alors elle doit être **protégée** de manière à ce qu'un intrus ne puisse **simuler** un «*challenge*» et **rejouer** une «*response*».

- **9** Si un «*timestamp*» est utilisé pour garantir la «*fraîcheur*» par rapport à une référence de temps absolue, alors la différence entre les horloges des différents interlocuteurs doit être **inférieure à la durée de validité** d'un message.
- □ $\Longrightarrow$ le mécanisme de maintenance de l'horloge sur tous les interlocuteurs devient une **dépendance** pour la confiance dans le protocole.

10 Une **clé** peut avoir été utilisé **récemment** pour chiffrer un nonce alors qu'elle est **ancienne** et probablement **compromise** ;

□ *L'utilisation récente d'une clé ne la rend pas meilleure qu'elle ne devrait*.
Sur le protocole Needham-Schroeder d'établissement de clé :

1. $A \rightarrow S$: A, B, N_A
2. $S \rightarrow A$: $\{N_A, B, K_{AB}, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$
3. $A \rightarrow B$: $\{K_{AB}, A\}_{K_{BS}}$
4. $B \rightarrow A$: $\{N_B\}_{K_{AB}}$
5. $A \rightarrow B$: $\{N_B + 1\}_{K_{AB}}$

L'attaquant a réussi à casser une clé K_{AB} et essaye d'en forcer l'utilisation :

3. $I \rightarrow B$: $\{K_{AB}, A\}_{K_{BS}}$
4. $B \rightarrow I$: $\{N_B\}_{K_{AB}}$
5. $I \rightarrow B$: $\{N_B + 1\}_{K_{AB}}$

Les messages 1) et 2) sont interceptés.

11 Si un **encodage** est utilisé pour donner du sens à un message, alors il est nécessaire que cet encodage soit **identifié**.

□ Dans le cas où cet encodage est dépendant du protocole, il devrait être possible de déduire que le **message appartient au protocole**, mais ausi à un **échange particulier** utilisant ce protocole.

12 Lors de la définition d'un protocole, il est nécessaire de savoir quelles sont les **relations de confiance** dont il dépend et quelles sont les **dépendances** nécessaires.

□ Les raisons de ces relations de confiance doivent être **clairement indiquées** :

- ◇ le protocole Kerberos échoue si le «*timestamp*» sur le serveur de clé n'est pas à la date et heure courantes ;
- ◇ le navigateur Web contient un grand nombre de clés publiques pour vérifier l'identité des sites visités. Si ces clés sont compromises alors l'utilisateur peut être abusé par des faux sites.

Exemple
d'un protocole de «*login/mdp*»
pour le Web

Utilisation d'un «login/mdp» pour l'accès à certaines ressources :

- version «*Basic*» : le «login/mdp» passent en clair dans la connexion TCP et nécessite TLS ;
- version «*Digest*» : méthode «*Challenge/Response*» : le «mdp» n'est jamais envoyé en clair⟹ TLS pas obligatoire.

Démonstration version basic

On utilise la commande «`curl`» qui permet de faire des requêtes HTTP en ligne de commande et de *tracer* les échanges entre le client et le serveur Web :

xterm

```
pef@darkstar-8:/Users/pef$ curl -k -u secret:test -v http://localhost:1337
* Connected to localhost (127.0.0.1) port 1337 (#0)
* Server auth using Basic with user 'secret'
> GET / HTTP/1.1
> Host: localhost:1337
> Authorization: Basic c2VjcmV0OnRlc3Q=
> User-Agent: curl/7.57.0
> Accept: */*
>
* HTTP 1.0, assume close after body
< HTTP/1.0 401 Authorization Required
< Date: Tue, 06 Feb 2018 10:59:47 GMT
< Server: WSGIServer/0.1 Python/2.7.14
< Content-Type: text/plain
< Content-Length: 13
<
* Closing connection 0
Access denied
```

login/mdp

Le «login/mdp» est juste encodé en base64, et peut être récupéré :

xterm

```
pef@darkstar-8:/Users/pef$ echo "c2VjcmV0OnRlc3Q=" | base64 -D
secret:test
```

Démonstration version digest *avec un mauvais login/mdp*

xterm

```
pef@darkstar-8:/Users/pef$ curl -k --digest -u toto:supersecret -v http://localhost:1337
* Connected to localhost (127.0.0.1) port 1337 (#0)
* Server auth using Digest with user 'toto'
> GET / HTTP/1.1
> Host: localhost:1337
> User-Agent: curl/7.57.0
> Accept: */*
>
* HTTP 1.0, assume close after body
< HTTP/1.0 401 Authorization Required
< Date: Tue, 06 Feb 2018 10:27:54 GMT
< Server: WSGIServer/0.1 Python/2.7.14
< Content-Type: text/plain
< WWW-Authenticate: Digest realm="secret", nonce="MTUxNzkxMjg3NA==", algorithm=MD5, qop="auth"
< Content-Length: 13
<
* Closing connection 0
* Issue another request to this URL: 'http://localhost:1337/'
* Server auth using Digest with user 'toto'
> GET / HTTP/1.0
> Host: localhost:1337
> Authorization: Digest username="toto", realm="secret", nonce="MTUxNzkxMjg3NA==", uri="/",
cnonce="YTQxZjBhYjg0OWJhMzY2ZGI5OWU2MzQ3NGI4M2M0MTQ=", nc=00000001, qop=auth, response="bab95d2b7fbc95e431e8fc660f9fac46",
 algorithm="MD5"
> User-Agent: curl/7.57.0
> Accept: */*
>
* HTTP 1.0, assume close after body
< HTTP/1.0 401 Authorization Required
< Date: Tue, 06 Feb 2018 10:27:54 GMT
< Server: WSGIServer/0.1 Python/2.7.14
< Content-Type: text/plain
* Authentication problem. Ignoring this.
< WWW-Authenticate: Digest realm="secret", nonce="MTUxNzkxMjg3NA==", algorithm=MD5, qop="auth"
< Content-Length: 13
<
* Closing connection 1
Access denied
```

login/mdp

indique à l'utilisateur quelle login/mdp utiliser

la fonction de hash à utiliser

nonce choisi par le serveur à chaque erreur 401

Quality of Protection : «auth» *ou* «auth-int» *(avec intégrité)*

nonce choisi par le client

La réponse calculée par :

A1 = MD5(username:password:realm):nonce:cnonce

A2 = request-method:uri-directive-value

response = MD5(MD5(A1):nonce:nc:cnonce:qop:MD5(A2))

accès refusé !

Deux connexions successives sont nécessaires : la première pour récupérer le challenge, la seconde pour donner sa réponse.

Démonstration version digest *avec un mauvais login/mdp*

Côté serveur :

xterm

```
pef@darkstar-8:/Users/pef/Python_snippets$ python rfc2617.py
Serving on port 1337...
127.0.0.1 - - [06/Feb/2018 11:27:54] "GET / HTTP/1.1" 401 13
{ 'algorithm': 'MD5',
  'cnonce': 'YTQxZjBhYjg0OWJhMzY2ZGI5OWU2MzQ3NGI4M2M0MTQ=',
  'nc': '00000001',
  'nonce': 'MTUxNzkxMjg3NA==',
  'qop': 'auth',
  'realm': 'secret',
  'response': 'bab95d2b7fbc95e431e8fc660f9fac46',
  'uri': '/',
  'username': 'toto'}
'c7f121a10d924244c5df2e707e3a28f8'
Auth failed!
127.0.0.1 - - [06/Feb/2018 11:27:54] "GET / HTTP/1.0" 401 13
```

Accés HTTP avec erreur 401 ⟹ Accès refusé

Accés HTTP avec erreur 401 ⟹ Accès refusé

Le serveur vérifie la réponse fournie par le client en recalculant cette réponse avec les mêmes informations que le client devraient partager avec lui, si le nom indiqué correspond à un «login/mdp» qu'il connait.

Ici, l'utilisateur `toto` *n'est pas connu du serveur.*

Démonstration version digest *avec un bon login/mdp*

xterm

```
pef@darkstar-8:/Users/pef/tmp/CIPHER curl -k --digest -u secret:test -v http://localhost:1337
* Connected to localhost (127.0.0.1) port 1337 (#0)
* Server auth using Digest with user 'secret'
> GET / HTTP/1.1
> Host: localhost:1337
> User-Agent: curl/7.57.0
> Accept: */*
>
* HTTP 1.0, assume close after body
< HTTP/1.0 401 Authorization Required
< Date: Tue, 06 Feb 2018 10:28:01 GMT
< Server: WSGIServer/0.1 Python/2.7.14
< Content-Type: text/plain
< WWW-Authenticate: Digest realm="secret", nonce="MTUxNzkxMjg4MQ==", algorithm=MD5, qop="auth"
< Content-Length: 13
<
* Closing connection 0
* Issue another request to this URL: 'http://localhost:1337/'
* Connected to localhost (127.0.0.1) port 1337 (#1)
* Server auth using Digest with user 'secret'
> GET / HTTP/1.0
> Host: localhost:1337
> Authorization: Digest username="secret", realm="secret", nonce="MTUxNzkxMjg4MQ==", uri="/",
cnonce="YmE1ODBkZjVhMzMyYjk1OWYyZjg3NWIwMjMwMWE0NjY=", nc=00000001, qop=auth,
response="3ece24afa6375d1eeda47387b83da8c6",
algorithm="MD5"
> User-Agent: curl/7.57.0
> Accept: */*
>
* HTTP 1.0, assume close after body
< HTTP/1.0 200 OK
< Date: Tue, 06 Feb 2018 10:28:01 GMT
< Server: WSGIServer/0.1 Python/2.7.14
< Content-Type: text/plain
< Content-Length: 3
<
* Closing connection 1
OK!
```

Connexion réussie !

Démonstration version digest *avec un bon login/mdp*

Côté serveur :

xterm

```
127.0.0.1 - - [06/Feb/2018 11:28:01] "GET / HTTP/1.1" 401 13
{ 'algorithm': 'MD5',
  'cnonce': 'YmE1ODBkZjVhMzMyYjk1OWYyZjg3NWIwMjMwMWE0NjY=',
  'nc': '00000001',
  'nonce': 'MTUxNzkxMjg4MQ==',
  'qop': 'auth',
  'realm': 'secret',
  'response': '3ece24afa6375d1eeda47387b83da8c6',
  'uri': '/',
  'username': 'secret'}
'3ece24afa6375d1eeda47387b83da8c6'
{ 'cli': 1517912881, 'srv': 1517912881}
127.0.0.1 - - [06/Feb/2018 11:28:01] "GET / HTTP/1.0" 200 3
```

Accés HTTP avec erreur 401 ⟹ Accès refusé

Accés HTTP avec succés 200 ⟹ Accès autorisé

Le serveur vérifie la réponse fournie par le client en recalculant cette réponse avec les mêmes informations que le client devraient partager avec lui, si le nom indiqué correspond à un «login/mdp» qu'il connait.

Ici, l'utilisateur `secret` *possèdant le mdp* `test` *est reconnu.*

Peut-on *automatiser*
la vérification des protocoles ?

- un ensemble de **règles logiques** et **d'inférences** pour définir et analyser les protocoles d'échange d'information ;
- permet d'évaluer si l'information échangée est :
  - de **confiance** ;
  - **sécurisée** contre les écoutes ;
- dans un contexte où toute information échangée peut être :
  - **écoutée** par tous ;
  - **modifiée** «*tampering*» ;
- Une modélisation composée :
  - de **définitions** :
    - P et Q sont des **agents** qui communiquent ;
    - X est un **message** ;
    - K est une **clé de chiffrement** ;
  - d'un ensemble de **règles** :

P croit X	P agit comme si X est vrai/valide, et pourra utiliser X dans un nouveau message
P a le contrôle de X	l'avis de P sur X est fiable et on peut lui faire confiance
P a dit X	à un certain moment P a transmis X dans lequel il avait confiance, mais P peut ne plus croire X
P a vu X	P a reçu le message X et peut le lire et le renvoyer
X_K	X est chiffré avec la clé K
$fresh(X)$	X n'a pas été précédemment envoyé dans un message
$key(K, P \leftrightarrow Q)$	P et Q peuvent communiquer avec la clé partagée K

Exemple d'assertions et d'inférence :
- ▷ Si P croit $key(K, P \leftrightarrow Q)$, et P a vu X_K, alors P croit (Q a dit X) ;
- ▷ Si P croit (Q a dit X) et P croit $fresh(X)$, alors P croit (Q croit X) ;

P doit croire que X est «frais» ici.

Si X n'est pas connu pour être «frais», alors cela peut être un ancien message, rejoué par un attaquant.

Exemple d'assertions et d'inférence :

▷ Si P croit (Q a le contrôle de X) et P croit (Q croit X), alors P croit X ;

▷ Si P croit que Q a dit $X\|Y$, la concaténation de X et Y, alors P croit que Q a dit X et P croit aussi que Q a dit Y.

À l'aide de cette notation, on peut formaliser un protocole d'authentification et prouver que certains agents peuvent communiquer en utilisant certaines clés.

Si la preuve échoue alors il est possible qu'un attaquant puisse compromettre le protocole.

Exemple «*Challenge/Response*»

a. $A \to B : N$

b. $B \to A : \{B, N\}K_{AB}$

A envoi un challenge à B, que B renvoi avec son identité, le tout chiffré avec la clé partagée K_{AB}.

$\Longrightarrow$**Problème** : si le générateur aléatoire utilisé pour fournir N n'est pas bon, c-à-d $fresh(N)$ n'est pas garanti, il est possible de revoir N et de **rejouer** $\{B, N\}K_{AB}$ à la place de A auprès de B (en particulier dans le cas d'un système automatique que l'on peut déclencher de très nombreuses fois jusqu'à ré-obtenir un N pour lequel on a enregistrer le $\{B, N\}K_{AB}$).

Amélioration *challenge/response with two-factor authentication*

a. $S \to U : N$

b. $U \to P : N$, PIN

c. $P \to U : \{N, \text{PIN}\}K_{SP}$

d. $U \to S : \{N, \text{PIN}\}K_{SP}$

▷ S le serveur, U l'utilisateur, P le générateur de mot de passe : un boitier ressemblant à une calculatrice de poche que possède l'utilisateur et qui est «*tamper-resistant*» ;

▷ N le nonce, «*number used once*» donné par le serveur lors de l'accès de l'utilisateur ;

▷ PIN le PIN, «*Personal Identification Number*» de l'utilisateur qui initie le générateur de mot de passe.

Exemple le protocole d'authentification «*Wide Mouth Frog Protocol*»

- les agents A et B veulent établir une **connexion sécurisée** ;
- un serveur de confiance S permet l'**authentification mutuelle** de ces agents ;
- le protocole va permettre à l'agent A d'envoyer une clé symétrique K_{AB} à l'agent B, via le serveur S.

On suppose que S est toujours honnête.

a. $A \to S$: $A, \{N_A, B, K_{AB}\}K_{AS}$
b. $S \to B$: $\{N_S, A, K_{AB}\}K_{BS}$

L'agent A dispose de la clé K_{AS}, l'agent B de la clé K_{BS}.

▷ A et S croient $key(K_{AS}, A \leftrightarrow S)$
▷ B et S croient $key(K_{BS}, B \leftrightarrow S)$
▷ A croit $key(K_{AB}, A \leftrightarrow B)$, A initie une communication avec B, il crée la clé K_{AB}
▷ B veut accepter la clé K_{AB} si elle provient bien de A :

dérivations ↓

B croit (A a le contrôle de $key(K_{AB}, A \leftrightarrow B)$)
B veut faire confiance à S pour relayer honnêtement la clé de A :
B croit (S a le contrôle de (A croit $key(K_{AB}, A \leftrightarrow B)$))
Si B croit que S croit que A veut utiliser une clé particulière pour communiquer avec B, alors B va faire confiance à S et va le croire également.
Le but est d'obtenir : B croit $key(K_{AB}, A \leftrightarrow B)$.

Analyse :

- Propriétés de **secret** et **d'authentification** (message, agent) :
  - dans a) S peut authentifier A s'il est capable de retrouver le format du message chiffré avec K_{AS} ;
  - dans b) B peut authentifier S de la même façon ;
- Propriétés de **fraîcheur** :
  - le message a) utilise un N_A qui n'est pas forcèment «frais» ce qui peut amener à une **attaque par rejeu**
    $\Longrightarrow$ l'attaquant ayant découvert une clé K_{AB} peut forcer l'utilisation de cette clé en renvoyant le paquet vers S.

Exemple le protocole d'authentification «*Wide Mouth Frog Protocol*»

On voudrait que S enregistre tous les N_A et tous les N_S pour garantir leur «fraicheur» pour éviter les attaques par rejeu.
On peut aussi utiliser une horloge partagée par A, B et S :

a. $A \to S : A, \{N_A, B, K_{AB}\}K_{AS}$
b. $S \to B : \{N_S, A, K_{AB}\}K_{BS}$

$\Longrightarrow$

a. $A \to S : A, \{t, B, K_{AB}\}K_{AS}$
b. $S \to B : \{t + \Delta_t, A, K_{AB}\}K_{BS}$
A envoi dans a) l'heure t et S envoi dans b) l'heure $t + \Delta_t$

- ▷ S croit $key(K_{AS}, A \leftrightarrow S)$ et S a vu $\{t, K_{AB}\}K_{AS} \Longrightarrow S$ croit (A a dit $\{t, K_{AB}\}$)
  S croit $fresh(t) \Longrightarrow$ le message n'est pas un rejeu $\Longrightarrow S$ croit (A croit $key(K_{AB}, A \leftrightarrow B)$)
- ▷ S envoi à B $\{t + \Delta_t, A, A\ croit\ key(K_{AB}, A \leftrightarrow B)\}K_{BS}$
- ▷ B reçoit b) chiffré avec K_{BS} et B croit $key(K_{BS}, B \leftrightarrow S)$
  alors B croit (S a dit $\{t + \Delta_t, A, A\ croit\ key(K_{AB}, A \leftrightarrow B)\}$)
- ▷ les horloges sont **synchronisées** alors B croit $fresh(t + \Delta t)$ et $fresh(A\ croit\ key(K_{AB}, A \leftrightarrow B))$
- ▷ B croit que le message de S est «*frais*», B croit (S croit ($A\ croit\ key(K_{AB}, A \leftrightarrow B)$))
- ▷ B croit (S a le contrôle de (A croit $key(K_{AB}, A \leftrightarrow B)$)) $\Longrightarrow B$ croit (A croit $key(K_{AB}, A \leftrightarrow B)$)
- ▷ B croit que (A a le contrôle de $key(K_{AB}, A \leftrightarrow B)) \Longrightarrow B$ croit $key(K_{AB}, A \leftrightarrow B)$
- ▷ B **peut contacter** A directement avec K_{AB} !

Le protocole Kerberos

a. $A \to S : A, B$
b. $S \to A : \{T_S, L, K_{AB}, B, \underline{\{T_S, L, K_{AB}, A\}K_{BS}}\}K_{AS}$ ❶
c. $A \to B : \{T_S, L, K_{AB}, A\}K_{BS}, \{A, T_A\}K_{AB}$
d. $B \to A : \{T_A + 1\}K_{AB}$

L est le «*lifetime*», c-à-d la durée d'autorisation de l'accès de A à la ressource B.
❶ est le ticket d'autorisation donné à A pour accéder à B
La clé K_{AB} est fournie dans le ticket pour B et également fournie à A chiffrée avec K_{AS}

$\Longrightarrow A$ peut ensuite vérifier le ticket auprès de B et B confirme son acceptation à A avec $T_A + 1$
$\Longrightarrow$Les **horloges** de chaque appareil doivent être **synchronisées** entre elles.

Attention

La logique BAN utilise des définitions et un **système de dérivation/d'inférence** pour, à partir d'hypothèses posées par le protocole de communication sécurisé, déterminer les **propriétés de sécurités** fournies par ce protocole.
Elle a pour but de :

▷ démontrer que les propriétés de sécurité voulues sont **bien garanties** par le protocole ;

▷ identifier les **vulnérabilités** ou fournir une piste pour les trouver quand ces propriétes ne sont pas garanties.

Critiques de la logique BAN :

□ de nombreuses **simplifications** pour automatiser l'analyse de protocole de communication sécurisé ;

□ des **hypothèses simplificatrices** comme le modèle d'attaquant de Dolev-Yao :

◇ côté **réseau** : l'attaquant est **tout puissant** : il peut écouter, intercepter et forger des messages : «*l'attaquant sert de support à la transmission du message*» ;

◇ côté **crypto** : l'attaquant est **limité** par la cryptographie mise en place : il ne peut manipuler les bits d'un message, ni deviner la clé...
$\Longrightarrow$ *difficulté à intégrer des connaissances partielles ou des attaques sur les algorithmes crytptographiques, contrairement au monde réel* ;
Utilisation d'opérateurs abstraits : par exemple l'utilisateur x utilise du chiffrement symétrique D_x et E_x avec $D_x E_x = E_x D_x = 1$, et $E_x(M)$ ne révèle rien sur M.

□ la **sémantique** des opérations utilisées pour modéliser le protocole est **limitée** :

◇ ne permet pas d'intégrer les **connaissances** que peut acquérir l'adversaire (interactions mutiples avec la cible, exploitation de vulnérabilités n'ayant pas de lien directe avec le protocole modélisé) ;

◇ l'univers des **conclusions** possibles obtenues après inférence/dérivation n'est pas garantie comme étant suffisament proche du réel...
$\Longrightarrow$ **risque de preuve que le protocole est sûr alors qu'il ne l'est pas** ;
$\Longrightarrow$ **Abandon de ce modèle**...*mais il reste une introduction à la modélisation de protocole*.

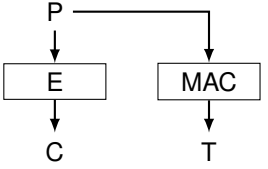
Et si on combinait, chiffrement symétrique
Authentification et Intégrité ?

On veut à la fois :

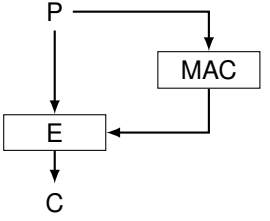
- ▷ **chiffrer** des données⟹ *confidentialité* ;
- ▷ s'assurer que les donnés **viennent bien** d'un interlocuteur choisi⟹ *authentification* ;
- ▷ s'assurer que les données n'ont **pas été modifiées** ou corrompues⟹ *intégrité*.

Trois façons de procéder

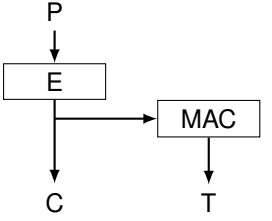
P est le «*plaintext*», C le chiffré, T le «*tag*» d'authentification et E le chiffrement.



□ chiffrer et MAC ;



□ MAC **puis** chiffrer ;



□ chiffrer **puis** MAC ;

Le choix du chiffrement ou d'un MAC spécifique n'est pas important du moment que chacun soit sécurisé et qu'ils ne ***partagent pas la même clé****.*

Chiffrer et MAC

- ▷ $C = E(K_1, P)$
- ▷ $T = MAC(K_2, P)$
- ▷ $P = D(K_1, C)$
- ▷ $T = MAC(K_2, P)$ valide ou non P

$\Longrightarrow$ le MAC, si pas un PRF, «*pseudo random function*», peut donner des indications sur le «*plaintext*» P !

Dans SSH, chaque paquet de données chiffré C est suivi du «tag» $T = MAC(K, N\|P)$ *où N est le numéro de séquence du paquet et le MAC utilisé est HMAC-SHA-256 qui ne donne aucune information sur P.*

MAC puis chiffrer

- ▷ $T = MAC(K_2, P)$
- ▷ $C = E(K_1, P\|T)$
- ▷ $P\|T = D(K_1, C)$, d'abord déchiffrer
- ▷ $T = MAC(K_2, P)$ valide ou non P

$\Longrightarrow$Le destinataire doit déchiffrer le message avant de pouvoir en calculer le MAC, il peut réaliser ce travail pour rien avec des messages corrompus ou malveillants.

$\Longrightarrow$le «*tag*» est caché, l'attaquant ne peut rien apprendre sur P.

*Utilisé dans TLS jusqu'à la version 1.3 où il est remplacé par l'utilisation d'un **chiffrement authentifiant**.*

Chiffrer puis MAC

- ▷ $C = E(K_1, P)$
- ▷ $T = MAC(K_2, C)$
- ▷ $T = MAC(K_2, C)$
- ▷ $P = D(K_1, C)$, seulement si T est correct.

$\Longrightarrow$ le destinataire du message n'a qu'un MAC à calculer pour s'assurer que le message n'est pas corrompu

$\Longrightarrow$ si le message est corrompu, il n'a pas besoin de le déchiffrer.

$\Longrightarrow$ un attaquant ne peut envoyer de combinaison de C et de T pour forcer le destinataire à déchiffrer.

C'est l'approche choisie par IPSec.

TLS assure :

▷ la **confidentialité** avec du chiffrement ;

▷ l'**intégrité** avec un MAC, «*Message Authentication Code*».

Mais...

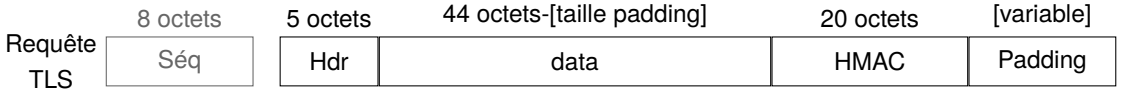
□ doit-on **authentifier** les données en clair, puis les **chiffrer** ? «*MAC-then-Encrypt*»

□ ou doit-on **chiffrer** les données, puis **authentifier** ces données chiffrées ? «*Encrypt-then-MAC*»

Les **échanges du protocole** protégé par TLS sont chiffrés suivant des **blocs de taille choisi** :

▷ un protocole échange souvent des **données de petites tailles**⟹ chiffrées dans des blocs ;
Exemple : le protocole SMTP échange des salutations, l'adresse de l'expéditeur, son destinataire ainsi que le contenu du courrier. Chaque message du client est acquitté par un message de la part du serveur. Seul le contenu du courrier excéde les 50 octets en moyenne des autres.

▷ suivant le chiffrement utilisé, comme AES en mode CBC, le **dernier bloc** échangé doit être **complété**, «*padded*», avec des octets supplémentaires pour atteindre la taille exigé.



Le «*padding*» arrive **après le MAC** car on est en mode «*MAC-then-Encrypt*».

Chaque bloc dispose après son «*Header*» :

◇ d'un **numéro de séquence** qui est stocké et incrémenté pour chaque bloc ;

◇ d'une partie **données** qui va être chiffrée :

⋆ de taille multiple de 16 (ici, on choisira une taille max de 64) ;

⋆ composée de :

▷ des données à échanger, **data**, sur au plus 44 octets (ici, 64 - 20 pour le HMAC) ;

▷ d'un HMAC pour l'intégrité des données en clair ;

▷ d'un «*padding*» : un nombre n précédé par n copies de lui-même :

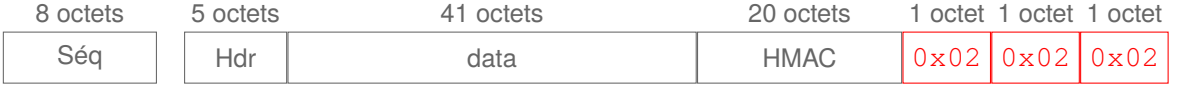
Différents blocs donnent différents padding

Ici, on a fixé la taille des données à 64 qui est un multiple de 16, soient $64 - 20 = 44$ octets maximum de données.
D'où un «*padding*» de :

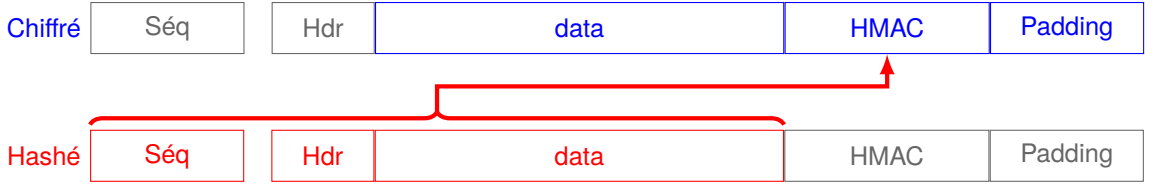
- 1 pour 43 octets de données :

- 2 pour 42 octets :

- 3 pour 41 octets :

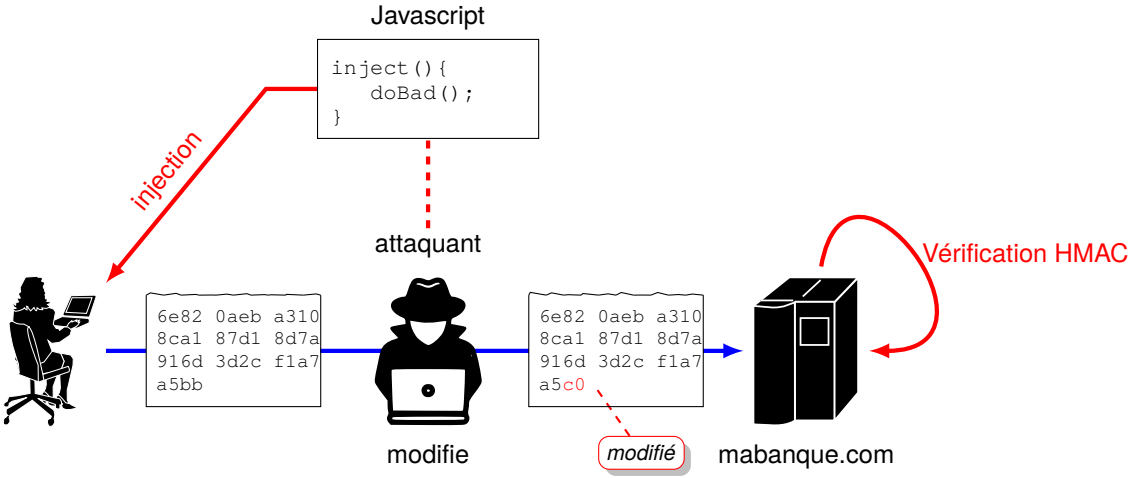


Pour le déchiffrement d'un bloc



- ▷ déchiffrer l'intégralité des données ;
- ▷ regarder le dernier octet, le supprimer, ainsi que tous les octets de «*padding*» associés ;
- ▷ localiser le HMAC (les derniers 20 octets après suppression du «*pading*») ;
- ▷ calculer le HMAC :
  - ◇ prendre le numéro de séquence, les 5 octets d'en-tête, «*Hdr*» et le message ;
  - ◇ utiliser la clé partagée utilisée pour le HMAC.
- ▷ si le HMAC est correct : **accepter** le bloc TLS.

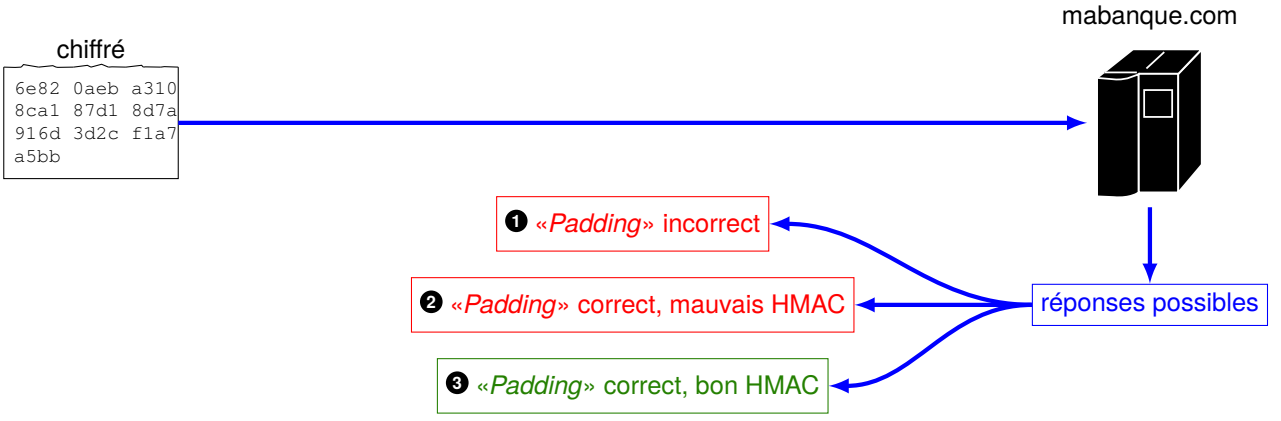
Un «*padding oracle*» est un moyen utilisé par un attaquant capable de **modifier les données chiffrées** transmises à un serveur de **récupérer le texte en clair**.



L'attaque par «*padding oracle*» :

- l'attaquant possède la capacité de modifier le chiffré :
  - ◇ dans le cas où il est dans le réseau local de sa victime, il peut utiliser de l'«*ARP spoofing*» pour rediriger le trafic de la victime vers l'attaquant au lieu du routeur ;
  - ◇ sinon, il peut injecter du code Javascript pour forcer le navigateur de la victime à envoyer des requêtes HTTPS ;
- Si le serveur TLS se **comporte différemment** lors du déchiffrement :
  - ▷ avec un «*padding*» incorrect ❶ ;
  - ▷ avec un «*padding*» correct ⟹ HMAC **incorrect** ❷ ou HMAC **correct** ❸ ;

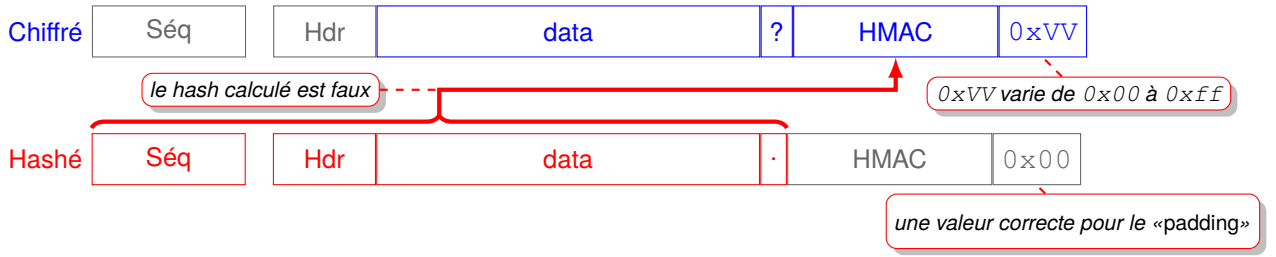
⟹l'attaquant peut **construire des chiffrés** qui vont lui fournir suffisamment d'information pour **retrouver le texte en clair**.



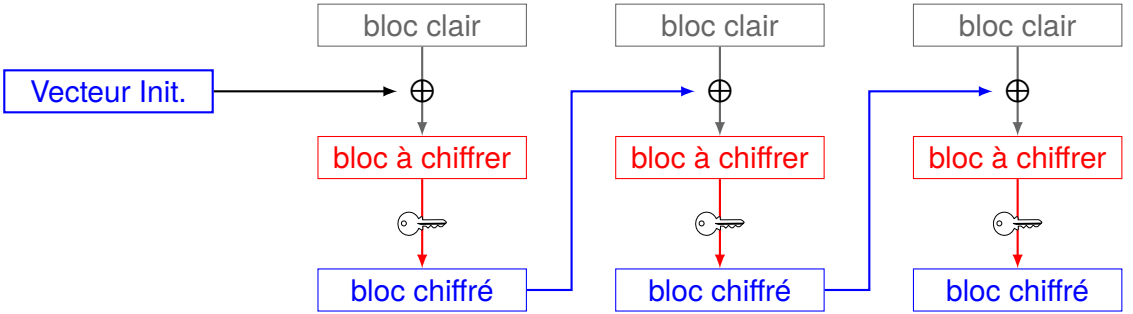
Attaque

L'attaquant :

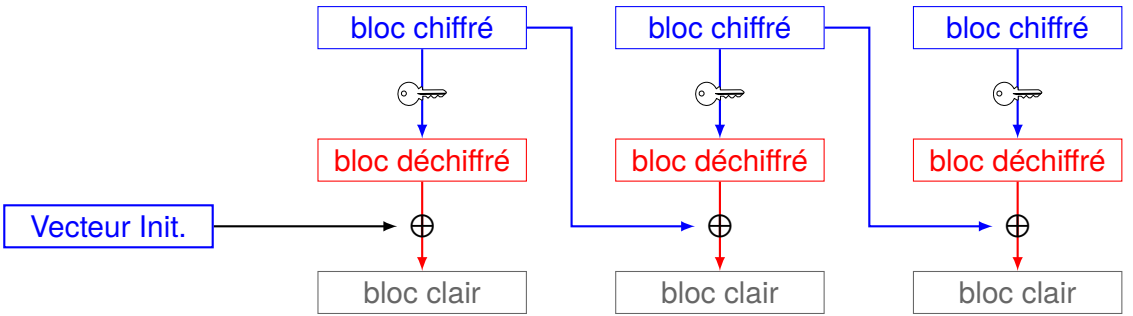
- ▷ exploite la structure de CBC ;
- ▷ construit 256 chiffrés dont le dernier octet se déchiffre entre `0x00` et `0xff` :
  - ◇ avec l'**erreur retournée**, il **déduit** lequel de ces messages se déchiffre vers `0x00`, qui est un «*padding*» correct.



Chiffrement CBC

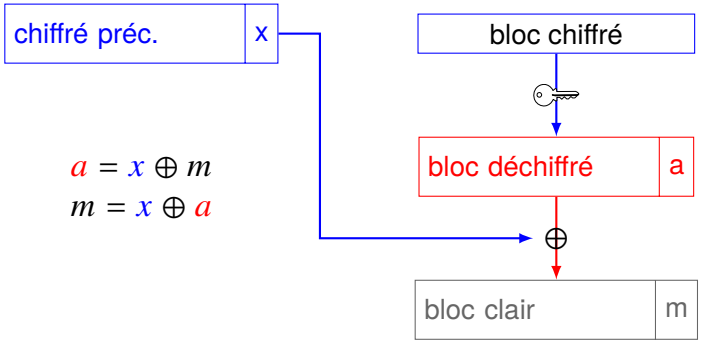


Déchiffrement CBC



«*Padding Oracle*» : déroulement de l'attaque sur un «*padding*» de 1 octet

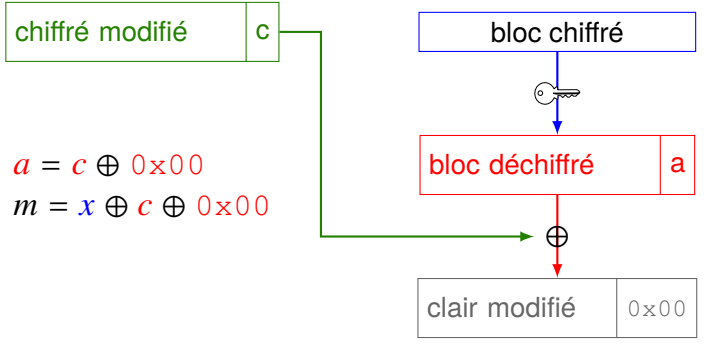
Déchiffrement CBC en opération normale



- ▷ m est le dernier octet du message en clair ;
- ▷ x est le dernier octet du bloc chiffré précédent qui peut être manipulé par l'attaquant ;
- ▷ a est le dernier octet du bloc obtenu par déchiffrement du bloc chiffré courant relayé par l'attaquant⟹ la **cible** de l'attaque.

«*Padding Oracle*» tentative de récupération du dernier octet

L'attaquant essaye les 256 possibilités pour le dernier octet du bloc chiffré précédent en espérant que le chiffré courant soit déchiffré vers la valeur zéro, ce qui lui donnera l'erreur ❷ : «*Padding correct, mauvais HMAC*».

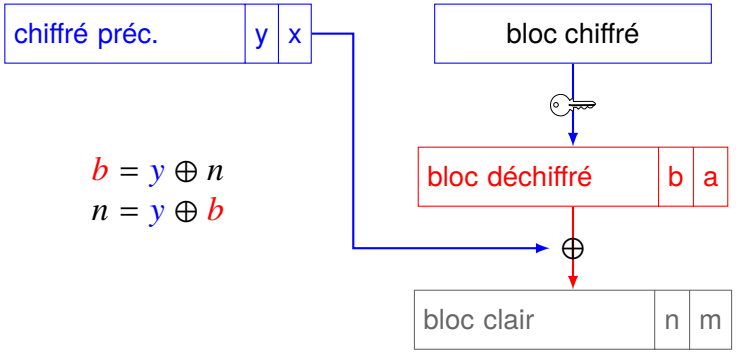


- ▷ si le «*padding*» est correct⟹l'attaquant peut déterminer la valeur de a ;
- ▷ l'attaquant disposant de x, il lui est facile de retrouver m.

L'attaquant dispose d'une méthode lui permettant d'exploiter le serveur TLS pour lui faire révéler la valeur de a⟹ il devient un «Oracle».

«*Padding Oracle*» : généralisation de l'attaque

Déchiffrement CBC en opération normale

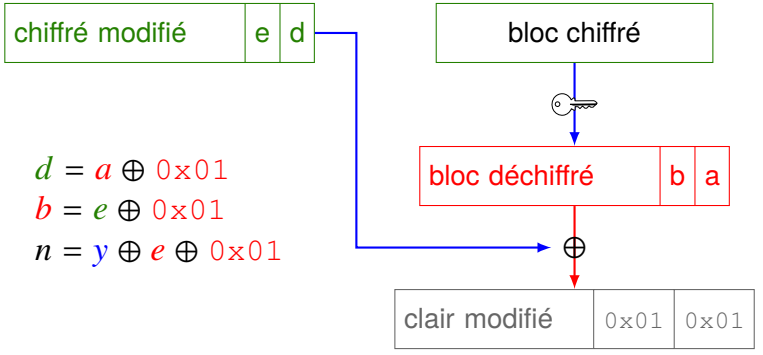


- ▷ l'octet b est la cible ;
- ▷ l'octet y est sous contrôle de l'attaquant ;
- ▷ b peut être déterminé par y et n ;

$\Longrightarrow$ n doit être manipulé pour faire partie du «*padding*» !

- ▷ une fois b connu, n est trouvable.

«*Padding Oracle*» tentative de récupération de l'avant dernier octet



- ▷ l'octet a est déjà connu de l'attaquant par l'attaque précédente$\Longrightarrow$il sera fixé à `0x01` par une valeur bien choisie de d ;
- ▷ l'octet n va être manipulé pour atteindre une valeur correcte pour un «*padding*» sur deux octets, c-à-d `0x01` ;
- ▷ l'attaquant envoi 256 messages avec les différentes valeurs possibles pour e

$\Longrightarrow$l'oracle nous révèle quelle est la bonne valeur de e pour obtenir un bon «*padding*», c-à-d avec n à `0x01`.

Pourquoi l'attaque réussit ?

▷ Le «*padding*» n'est pas authentifié ;

▷ l'attaquant peut deviner la valeur en faisant la différence entre une bonne ou une mauvaise suggestion.

L'attaque ne révèle qu'un **peu d'information** sur le texte en clair **à chaque tentative**

⟹c'est une attaque par «*side-channel*», «*canaux cachés*», c-à-d une fuite d'information observable par un tiers.

La correction ?

▷ retourner la même valeur d'erreur pour ❶ et ❷⟹l'attaquant n'a plus d'oracle ;

Mais...

Le **travail du serveur** n'est pas le même pour chaque cas :

⇒«*padding*» incorrect : lecture des octets de padding

⇒«*padding*» correct, mauvais HMAC : lecture des octets de padding, calcul du HMAC ;

⇒«*padding*» correct, bon HMAC : lecture des octets de padding, calcul du HMAC ;

En observant le **temps de réponse** du serveur en cas d'erreur, on peut **découvrir** s'il correspond à ❶ ou ❷ !

C'est une «*timing attack*» !

Pour palier aux inconsistences de temps de réponses dues au travail interne du serveur, on peut relancer plusieurs fois l'attaque et déterminer statistiquement dans quel cas on se trouve.

La correction ?

▷ dans les deux cas, le serveur calcule un HMAC même si le «*padding*» est invalide.

Mais... Il peut y avoir une «*timing attack*» sur le **temps de calcul du HMAC** qui peut être variable suivant la taille des données à hasher...

La correction ?

Ne plus utiliser CBC qui est victime des attaques BEAST et POODLE et le modèle «*MAC-then-Encrypt*»...

⟹Utiliser **AEAD**, «*Authenticated Encryption with Associated Data*», qui **mixe authentification** et **chiffrement**.

- alternative au chiffrement + MAC.
- fonctionne comme un chiffrement normal mais retourne un **chiffré** et un «*tag*» d'authentification.

Fonctionnement

▷ $AE(K, P) = (C, T)$
▷ AE correspond à «*authenticated encryption*»;
$\Longrightarrow$ réalise le même travail qu'une combinaison de chiffrement et MAC, mais de manière plus simple, plus rapide et souvent plus sécurisée.

▷ $AD(K, C, T) = P$
▷ AD correspond à «*authenticated decryption*»;
Si C ou T sont invalides alors AD va retourner une erreur et éviter au destinataire de traiter un message P qui peut avoir été «*forgé*».
$\Longrightarrow$ Si AD retourne un «*plaintext*», on peut être sûr que l'expéditeur connait la clé secrète.

Avantages

Un **chiffrement authentifié** est **plus fort** qu'un simple chiffrement car le déchiffrement n'aura lieu que si le «*tag*» d'authentification est **valide**.
$\Longrightarrow$ cela évite les attaques par **choix du chiffré** ,«*chosen ciphertext*», où l'attaquant crée des chiffrés et en demande le déchiffrement (comme dans l'attaque du «*padding oracle*» sur TLS).

Contraintes

▷ **l'authentification** doit être **aussi forte** que celle fournie par un MAC afin d'éviter qu'un attaquant forge un couple (C, T) que la fonction AD va déchiffrer et accepter.

Associated Data

Ce sont des données qui sont **authentifiées** mais **non chiffrées**.

- Si on utilise un **chiffrement authentifié** alors toutes les données fournies en entrée sont **chiffrées** et **authentifiées**.
- Mais, si l'on veut authentifier tout un message contenant une **partie non chiffrée** et une autre **partie chiffrée** ?

Exemple d'un paquet réseau composé d'un en-tête et d'un contenu :

- ▷ *le contenu est chiffré pour cacher les données transmises ;*
- ▷ *l'en-tête est non chiffré, car il contient des informations nécessaires à son acheminement ;*

$\Longrightarrow$ *mais on veut authentifier l'en-tête pour savoir que le paquet provient bien du bon expéditeur.*

Chiffrement authentifié

Un algorithme AEAD permet d'associet des données en clair à des des données chiffrées de telle manière que si les données en clair sont corrompues alors le «*tag*» d'authentification est invalide et le chiffré est rejeté.

- ▷ $AEAD(K, P, A) = (C, A, T)$ où K est une clé, P le «*plaintext*», les données associées A et un «*tag*» T d'authentification ;
- ▷ $AEAD$ laisse les données associées non chiffrées telles quelles, et fournit le chiffré C de P ;
- ▷ le «*tag*» T dépend à la fois de P et de A et ne sera valide que si C ou A n'auront pas été modifiés.

Le **déchiffrement** est réalisé par $ADAD(K, C, A, T) = (P, A)$ et échoue si C ou A sont modifiés ou corrompus.

$\Longrightarrow$ A ou P peuvent être vides :

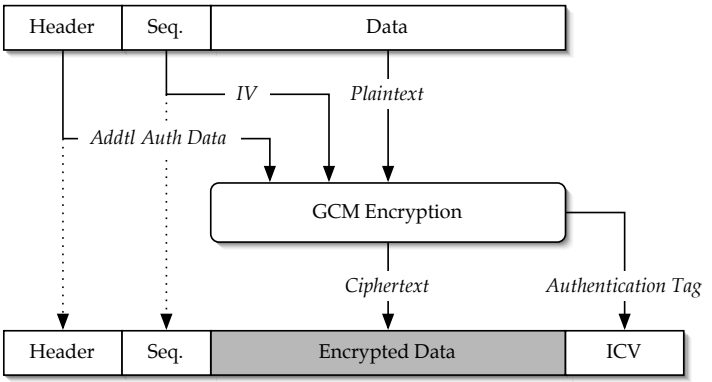
- ◦ si A est vide, $AEAD$ se comporte comme un chiffrement authentifié ;
- ◦ si P est vide, $AEAD$ est juste un MAC.

AES-GCM : le standard de chiffement authentifié

- ▷ basé sur AES et le «*Galois Counter Mode*» ;
- ▷ le GCM est une modification du mode CTR qui calcule un «*tag*» d'authentification

C'est le seul chiffrement authentifié standard de NIST SP800-38D, et il est utilisé dans IPSec, SSH et TLS 1.2.

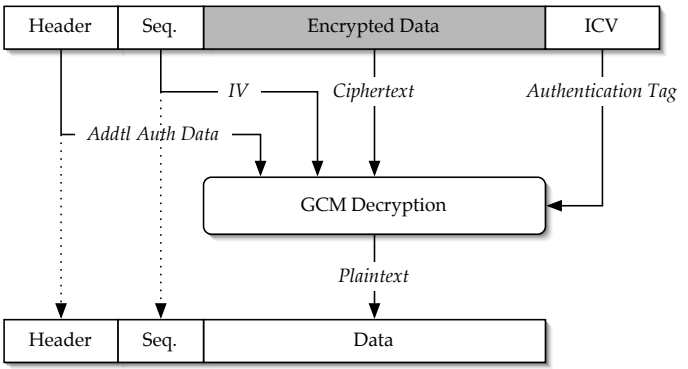
Paquet avec chiffrement et authentification :



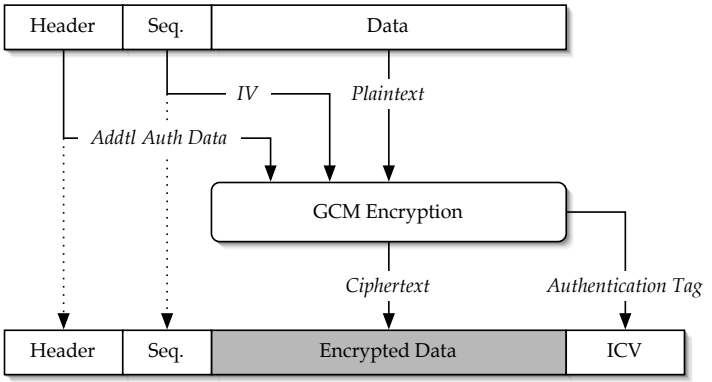
- les données, «*Data*», correspond aux données chiffrées par AES-128, soient des blocs de données chiffrées de taille multiple de 16 octets ;
- le numéro de séquence, «Seq.» du mode compteur, pour gérer la fragmentation/défragmentation des messages de taille supérieure à celle d'un paquet.
- l'entête, «*Header*», correspond à l'adresse du capteur ;
- le ICV, «*Integrity Check Value*», correspond à l'**étiquette d'autentification** calculée par AES-GCM, qui est au plus de 16 octets.

  *Assure **l'authentification** et l'**intégrité** du message grâce au GMAC,* «Galois Message Authentication Code».

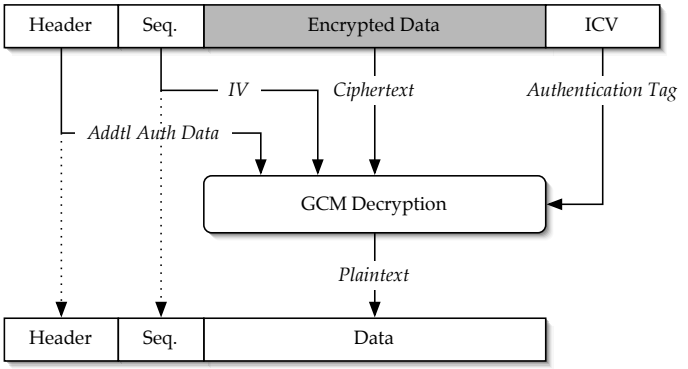
Pour le déchiffrement et l'authentification du paquet :



Pour le chiffrement authentifié d'un paquet



Pour le déchiffrement authentifié d'un paquet



Wikipedia

*The authentication strength depends on the **length of the authentication tag**, as with all symmetric message authentication codes. However, the use of shorter authentication tags with GCM is discouraged.*

The bit-length of the tag, denoted t, is a security parameter. In general, t may be any one of the following five values: 128, 120, 112, 104, or 96. For certain applications, t may be 64 or 32, but the use of these two tag lengths constrains the length of the input data and the lifetime of the key.

Appendix C in NIST SP 800-38D provides guidance for these constraints, for example :

- ▷ *if t = 32 and the maximal packet size is 210 bytes, then the authentication decryption function should be invoked no more than 211 times;*
- ▷ *if t = 64 and the maximal packet size is 215 bytes, then the authentication decryption function should be invoked no more than 232 times.*

Avec openSSL en ligne de commande ?

xterm
```
$ echo 'toto' | openssl enc -aes-128-gcm -k "secret" -out tester_gcm.bin
$ openssl enc -aes-128-gcm -d -k "secret" -in tester_gcm.bin
toto
bad decrypt
```
le «tag» ne peut être vérifié

*openSSL en ligne de commande **ne conserve pas** le «tag» d'authentification...*

Sous Python ?

On installe la bibliothèque «*cryptography*» :

xterm
```
$ python3 -m pip install cryptography
```

On utilise les «*hazmat*», «*hazardous materials*» :

xterm
```
Python 3.7.6 (default, Dec 30 2019, 19:38:26)
[Clang 11.0.0 (clang-1100.0.33.16)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import os
>>> from cryptography.hazmat.primitives.ciphers.aead import AESGCM
>>> data = b"message secret"
>>> aad = b"donnees authentifiees mais pas chiffree"
>>> key = AESGCM.generate_key(bit_length=128)
>>> aesgcm = AESGCM(key)
>>> nonce = os.urandom(12)
>>> ct = aesgcm.encrypt(nonce, data, aad)
>>> aesgcm.decrypt(nonce, ct, aad)
b'message secret'
```
le tag est concaténé aux données chiffrées

Autres ?

Il est possible/recommander d'utiliser la combinaison **ChaCha20Poly1305** définie dans la RFC 7539 de mai 2015, utilisé dans TLS 1.3.

Donc au final, la signature électronique reprend les principes du courrier...papier ?

Notion de copie et d'original d'un document papier

Une photocopie est différente de l'original (*ou presque...*). *Essayez de présenter la photocopie d'un billet pour acheter dans une boutique !*

Une personne est identifiée par sa signature (analyse graphologique) Cette signature engage la personne qui l'a écrite :

- ▷ c'est une preuve **d'acceptation** pour un contrat et **d'engagement** à le remplir ;
- ▷ c'est une **autorisation** de transfert d'argent dans le cas d'un chèque ;
- ▷ c'est une **identification** dans le cas d'une lettre que l'on envoie.

Cette signature est **reconnue** par la législation francaise.

Notion de copie «certifiée conforme» réalisable auprès de la mairie ou bien d'un commissariat. Cette notion a d'ailleurs **disparue**, face à l'avancée des moyens de reproduction et de l'utilisation systématique de l'impression machine pour les documents administratifs (plus ou presque de partie manuscrite présente sur le document ou bien reproduite électroniquement).

Signature

Une signature manuscrite idéale est réputée posséder les propriétés suivantes :

- ☐ elle ne peut être imitée ;
- ☐ elle **authentifie** le signataire ;
- ☐ la signature appartient **à un seul document** (elle n'est pas réutilisable) ;
- ☐ le document ne peut être partiellement ou totalement modifié ;
- ☐ la signature ne peut être **reniée** ;
- ☐ la signature peut être **contrôlée**.

Le cas du document électronique

Il est **reproductible** à l'infini sans modification.
C'est ce qui le rend virtuellement éternel.

Le **droit de copie**, dite de sauvegarde, est apparu avec l'apparition de «programme informatique» sur support duplicable (bande magnétique, disquette, CD...).

Il peut être modifié pour faire disparaitre ou apparaitre des éléments supplémentaires. *Suppression du nom de l'auteur d'un document de traitement de texte, ajout d'un texte de propriété sur une image...*

Il peut être **attribuer** à n'importe quel propriétaire. *Un fichier MP3 peut appartenir à une personne disposant du CD qui a servi de source à son encodage ou bien à une autre...*

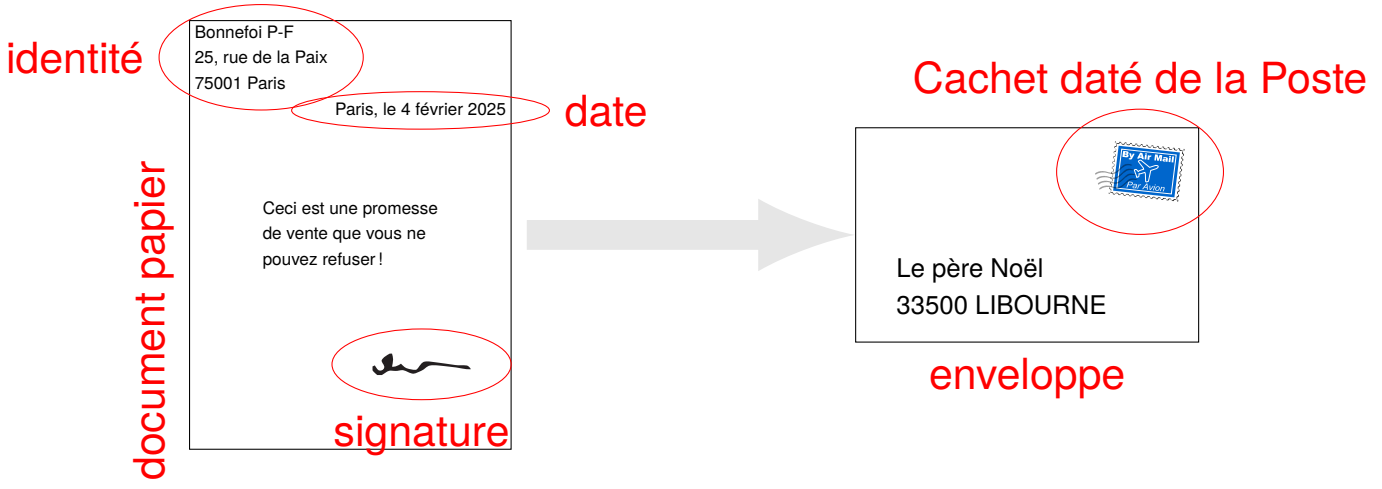
Limitation à la consultation

Une **nouvelle forme de propriété** est apparue avec lui : celle liée à la **consultation** du contenu sans possibilité d'exploitation ou de reproduction en vu de conservation.
C'est le cas du DVD ou du Bluray dont le contenu ne peut (ne pouvait) être accéder que pour le visionner mais pas pour l'enregistrer ou le modifier.

La sécurité «écrite» : la signature et l'envoi par la Poste

Pour être recevable comme document engageant la responsabilité de celui qui l'envoi le document doit posséder :

- une indication claire de **l'identité** ;
- une **signature** ;
- ces deux indications doivent être apposés **sur un même papier** (pas de collage, ...) ;
- mis dans une **enveloppe** avec le **cachet de la Poste**.



Les aspects juridiques nécessaires pour la version électronique

Vers la signature électronique : des considérations juridiques

Les régimes juridiques doivent admettre les écrits numériques comme :

- **recevables** (le juge a le droit de les considérer) ;
- potentiellement **probants** (ils apportent la preuve s'ils sont difficilement falsifiable).

Les travaux de normalisation se concentrent sur deux aspects :

- **l'interopérabilité** pour une signature électronique universellement interprétée et reconnue

  définition de standards d'interprétation **non ambigüe** des signatures ;
  des **algorithmes** de calcul et des **modes** de fonctionnement ;
  des initiatives privées (RSA Security Inc) ont déjà établi des formats de messages ;

- la **sécurité** :
  la norme internationale des «critères communs» de **spécification** et **d'évaluation sécuritaire** ouvre la perspective de la reconnaissance des signatures **entre pays** par le fait que leurs niveaux de sécurité soient équivalents.

La vérification des **caractéristiques de sécurité** des systèmes est effectuées par des **sociétés spécialisées**, les *évaluateurs* ; dont les compétences sont surveillées entre autres, par une autorité émanant de l'état **l'ANSSI**, «Agence Nationale de la Sécurité des Systèmes d'Information» (anciennement appelée la DCSSI).
Vous trouverez à `http://www.ssi.gouv.fr/administration/reglementation/`*, les documents relatifs au RGS, Référentiel Général de Sécurité définissant les exigences francaises en matière de sécurité relative à la cryptographie.*
Le **risque zéro** n'existe pas et **l'arsenal juridique** et technique doit **prendre en compte ce fait**, en prévoyant les **conséquences d'accidents majeurs** (fraudes ou dysfonctionnement) dans des **plans de secours**.

Le 13 décembre 1999, de la directive 1999/93/CE relative à «un cadre communautaire pour les signatures électroniques»

La loi du 13 mars 2000

Au contraire de la directive, la loi francaise ne rentre dans aucune considération technique. Elle définit de facon générale la signature, au regard des fonctions assurées par celle-ci : «*La signature nécessaire à la perfection d'un acte juridique identifie celui qui l'appose. Elle manifeste le consentement des parties aux obligations qui découlent de cet acte*» (art. 1316-4 du Code Civil).

Le code civil définit également les conditions de l'équivalence du support électronique et du support papier à titre de preuve, sous réserve que quatre conditions soient respectées :

Les quatre conditions posées par le code civil pour que le support numérique soit admissible comme preuve au même titre que le support papier

1. **pouvoir identifier** la personne dont émane l'écrit électronique au moyen d'un procédé fiable ;
2. l'écrit électronique a été **créé** dans des conditions de nature à en **garantir l'intégrité** ;
3. l'écrit électronique est **conservé** dans des conditions de nature à en **garantir l'intégrité** ;
4. utiliser un procédé fiable **garantissant le lien** de la **signature électronique** avec **l'acte** auquel elle s'attache.

Le décret du 30 mars 2001

Le décret est un texte technique, qui constitue la transposition de la **directive européenne** sur la signature électronique.

Il distingue la «signature électronique» de la «signature électronique sécurisée» :

- la **signature électronique** est celle qui respecte les conditions posées par le code civil ;
- la **signature électronique sécurisée** est celle qui répond de plus aux exigences du décret, et présente de ce fait une présomption de fiabilité.

Le décret précise les conditions de mise en œuvre de la «signature électronique sécurisée», qui bénéficie d'une présomption de fiabilité :

- elle est établie grâce à un dispositif sécurisé de création de signature électronique ;
- sa vérification repose sur l'utilisation d'un **certificat électronique qualifié**.

23 juillet 2014 : adoption règlement n° 910/2014/UE, eIDAS, «electronic identification and trust services»

- **abroge** la 1999/93/C à partir du 1er juillet 2016 et s'appliquera en effet directement à partir de cette date **dans tous les états membres de l'UE**
  viendra remplacer les lois nationales, ce qui engendra des modifications profondes dans la législation de certains pays en matière de signature et d'identification électronique.

- **création d'un nouvel objet juridique** : **la signature électronique de personne morale** :
  - nouveau concept juridique – qui n'existait pas en droit Francais – à savoir la **signature de personne morale** ou **cachet électronique**.
    Jusqu'à présent, seules les personnes physiques pouvaient créer une signature électronique. Les entreprises, les administrations et les associations vont désormais pouvoir signer en leur nom des documents qui seront recevables comme preuve en justice.
- **pas d'obligation de carte à puce** : le recours à un **support physique qualifié** ne fait pas partie des exigences applicables aux dispositifs de création de signature électronique qualifiés, permettant d'obtenir un niveau de sécurité juridique maximal sur les documents signés.

Le règlement consacre le marché européen de la signature électronique dans le secteur public et les relations avec les administrés, ainsi que le principe de reconnaissance mutuelle des moyens d'identification électronique délivrés par les Etats membres, tout en exigeant un haut niveau de sécurité pour l'ensemble des méthodes utilisées.

https://www.ssi.gouv.fr/entreprise/reglementation/confiance-numerique/le-reglement-eidas/

http://www.droit-technologie.org/actuality-1663/l-europe-remet-a-plat-les-regles-en-matiere-de-signature-electronique.html

Tous les acteurs intéressés – les citoyens/consommateurs ainsi que les entreprises et les autorités publiques – doivent être en mesure de recourir aux ***technologies de l'information et de la communication*** *(et notamment l'internet) en* ***toute confiance*** *et avec un* ***niveau de sécurité élevé****. Cela suppose notamment qu'ils puissent* ***identifier avec certitude*** *un cocontractant potentiel et que les obstacles à l'utilisation de certains outils (tels que des services de signature, d'horodatage ou de recommandé électroniques) soient levés.*

Par ailleurs, ces objectifs doivent être atteints dans un ***contexte transfrontalier paneuropéen*** *de sorte que, par exemple, un citoyen belge puisse s'identifier auprès d'une administration publique italienne, en utilisant des technologies fournies par une entreprise allemande (ce qui requiert, entre autres, que les moyens techniques utilisés soient* ***interopérables****).*

D'une part*, elles consacrent la reconnaissance mutuelle des* ***moyens d'identification électronique*** *délivrés par un Etat membre et qui seraient utilisés dans un autre Etat membre. On songe à l'hypothèse d'un étudiant belge qui souhaiterait s'inscrire en ligne dans une université espagnole, en établissant son identité au moyen de sa carte d'identité électronique. Pour les entreprises, cette reconnaissance mutuelle est importante dans le cadre des marchés publics et des réponses aux appels d'offres. Une procédure est mise en place, pour organiser la coopération entre les Etats membres et les autorités européennes (spécialement la Commission), et garantir l'interopérabilité des moyens d'identification utilisés.*

D'autre part*, le* ***règlement eIDAS*** *établit un* ***cadre juridique*** *pour* ***plusieurs services de confiance****. Outre la signature électronique, sont également visés le* ***cachet électronique*** *(qui doit permettre de garantir l'origine et l'intégrité d'un document électronique délivré par une personne morale),* ***l'horodatage électronique*** *(pour prouver l'existence des données à un moment particulier), les services d'envois recommandés et* ***l'authentification de site internet*** *(pour s'assurer qu'un site web est géré par celui qui s'en prétend titulaire).*

Il y a violation de secret de la correspondance lorsqu'une tierce personne en prend connaissance sans le consentement préalable de l'émetteur d'un courrier à caractère privé ou en dehors du cadre de la Loi.

Une correspondance reste la **propriété intellectuelle** de son auteur bien que le support physique soit la propriété du destinataire.

La convention européenne de sauvegarde des droits de l'Homme et des libertés fondamentales du 4 novembre 1950, rappelle en son article 8, «le droit au respect de la correspondance».

Union européenne

Au sein de l'Union européenne, le secret de la correspondance est garanti par la directive européenne 97/66 du 15 décembre 1997 qui fait obligation aux états membres de garantir par leur législation :

- la **confidentialité** des communications passées par la voie des télécommunications et d'interdire «à toute autre personne que les utilisateurs, sans le consentement des utilisateurs concernés, d'écouter, d'intercepter, de stocker les communications ou de les soumettre à quelque autre moyen d'interception ou de surveillance, sauf lorsque ces activités sont légalement autorisées».

France

En France, la **violation de secret** de la correspondance est actuellement réprimée par les articles 226-15 et 432-9 du code pénal et par l'article L 33-1 du code des postes et télécommunications.

Le ministre délégué au Budget et à la Réforme de l'Etat, Jean-Francois Copé, a présenté un projet de loi ratifiant l'ordonnance du 8 décembre 2005 relative aux échanges électroniques entre les usagers et les autorités administratives, et entre les autorités administratives elles-mêmes.

Cette ordonnance, prise sur le fondement de la loi du 9 décembre 2004, de simplification du droit, vient renforcer l'attirail juridique nécessaire au bon développement de «l'administration électronique» dans le pays.

L'e-administration

Elle concerne :

- l'ensemble des **échanges électroniques** ;
- télé-services ou courriels échangés avec les administrations, qu'il s'agisse des administrations de l'Etat, des collectivités territoriales, de leurs établissements publics administratifs, des organismes de sécurité sociale ou des autres organismes de droit privé gérant des services publics administratifs.

L'ordonnance a établi une **équivalence juridique** entre le **courrier électronique** et le **courrier sur support papier** en prévoyant notamment que la *saisine de l'administration par voie électronique est régulière* et doit faire l'objet d'un *accusé de réception* ou d'un *accusé d'enregistrement* informant l'usager que sa demande a été prise en compte.

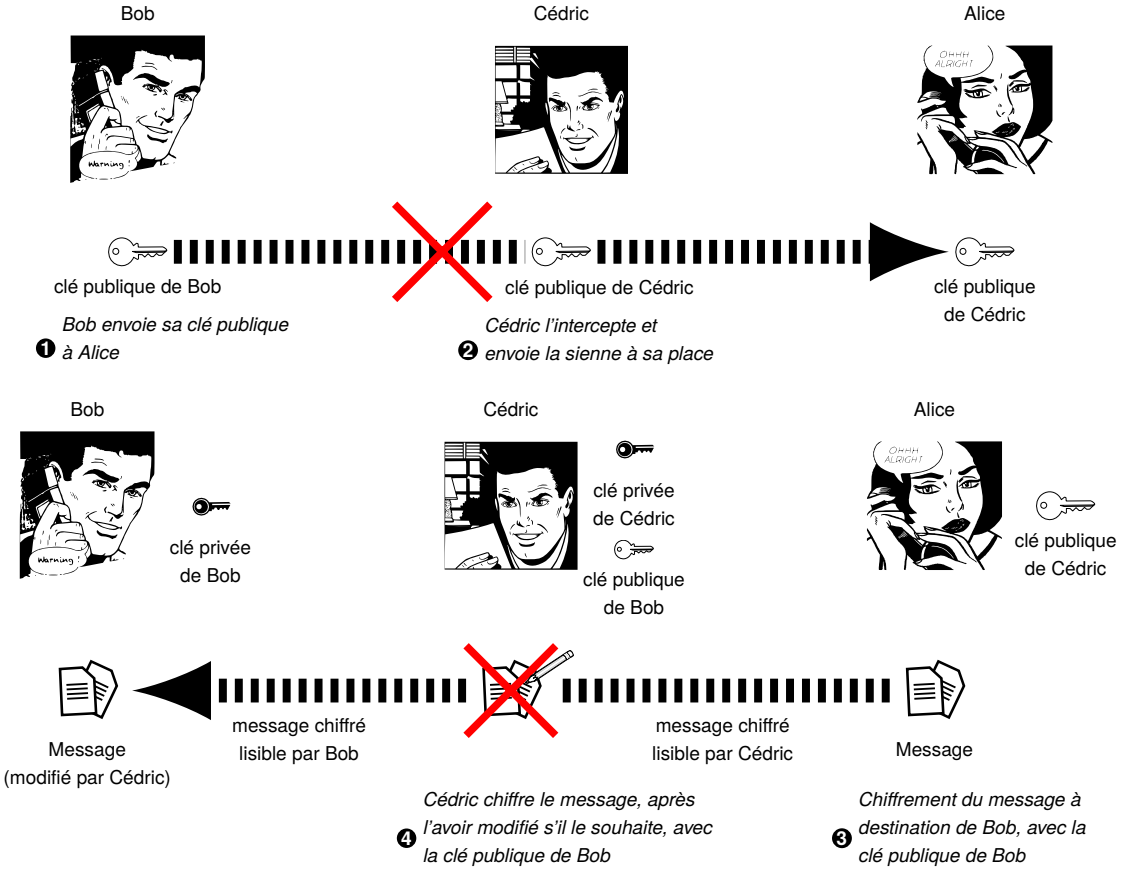
Elle offre ainsi la possibilité aux usagers de disposer d'un **espace de stockage en ligne**, personnalisé et personnalisable, qui a pour vocation d'accueillir les documents administratifs les concernant, ainsi qu'un bloc-notes contenant des formulaires en ligne.

Ce service sera expérimenté début 2006 avant sa mise en place en 2007. Le texte permet également la mise place des conditions permettant la **signature électronique** de leurs actes par les autorités administratives.

Alors ?

Plus de problème ?

Le problème de l'échange de clé publique



Le problème de la diffusion des clés publiques

Pour diffuser des clés publiques, on peut utiliser un **annuaire** (LDAP par exemple).

Le problème est de s'assurer que la clé que l'on récupère provient bien de la personne concernée : **rien ne garantit** que la clé est bien celle de l'utilisateur à qui elle est associée.

Attaque

Un pirate peut **remplacer** la clé publique de la victime présente dans l'annuaire par **sa clé publique**.

Ainsi, il peut **déchiffrer** tous les messages ayant été chiffrés avec cette clé.

Il peut même, ensuite, renvoyer à son véritable destinataire le message (modifié ou non) en le chiffrant avec la clé publique originale pour **ne pas être démasqué** !

But de la PKI

- Gérer les problèmes posés par le **maintien de lien entre des clés publiques et des identités** au travers de différentes applications.
  Sans PKI, il faudrait définir de nombreuses solutions de sécurité et espérer une certaine interopérabilité ainsi qu'un même niveau de protection entre elles.
- Gérer le **partage de la confiance** entre usagers, en utilisant un tiers pour **confirmer** la propriété d'un «credential», c-à-d un document conférant une identité ou une qualité, appelé «certificat».
- Être **reconnu** «comme de confiance» par les **différents usagers** :
  - un usager n'a plus à connaître directement un autre usager avec lequel il veut établir une relation de confiance :
    - il connaît un tiers de confiance partagé avec cet autre usager ;
    - il établit un lien de confiance avec cet autre usager au travers du tiers.

  C'est le modèle du «tiers de confiance».

Composants d'une PKI

- des **certificats électroniques** ;
- des **autorités d'enregistrement**, «*Registration Authority*», et de **certification**, «*Certification Authority*» ;
- un **procédé standardisé** de vérification.

La carte d'identité de la clé publique

Le certificat contient :

- une identité ;
- la clé publique associée à cette identité ;
- une **preuve de cette association** fournie par le tiers de confiance.

Comment fournir la **preuve de la confiance** du tiers ?

Utiliser la **signature électronique** de ce tiers !

La construction du certificat

L'ensemble des informations (le nom de l'autorité de certification, du propriétaire du certificat...) est **signé** par l'**autorité de certification**, représentant le tiers, à l'aide de la signature électronique :

- une fonction de hachage crée une **empreinte** de ces informations ;
- ce résumé est **chiffré** à l'aide de la **clé privée** de l'autorité de certification, la clé publique ayant été préalablement largement diffusée ou elle même signée par une autorité de niveau supérieur.

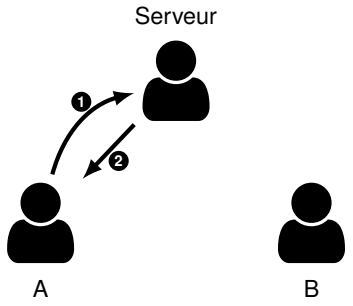
Grâce à cette signature électronique, il est possible de s'assurer de la fiabilité du certificat.

Cette méthode repose sur la confiance dans une structure dont on dispose de la clé publique, une autorité supérieure : VeriSign, GTE, Certinomis, CommerceNet, Thawte, ...

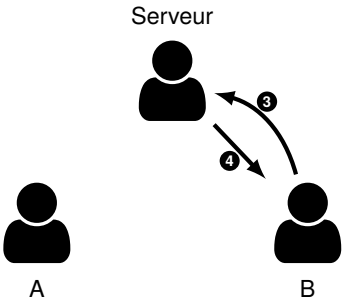
Fonctionnement

- **Si je fais confiance** à cette autorité de certification, **alors je fais confiance** aux certificats signés par elle ;
- Un certificat signé par elle associe une clé publique à une identité : **je fais confiance** à cette association (après vérification de la signature grâce à la clé publique de cette autorité de certification) ;
- **J'utilise la clé publique** contenue dans le certificat pour **l'identité associée**.

Enregistrement auprès du tiers de confiance

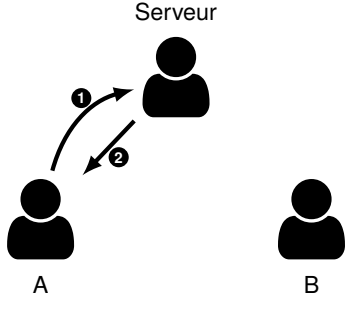


⇒A s'enregistre auprès de S
⇒S donne la clé partagée k_{AS}

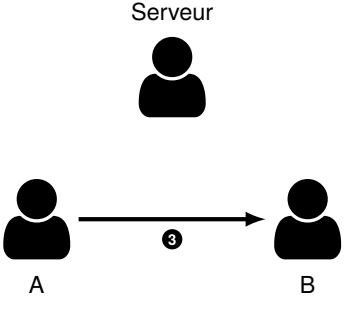


⇒B s'enregistre auprès de S
⇒S donne la clé partagée k_{BS}

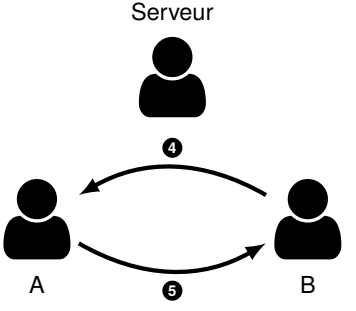
Comment établir un canal sécurisé entre A et B à un instant t ? On passe par S à cet instant t



⇒A veut communiquer avec B
⇒S donne la clé partagée k_{AB}
et le «*token*» $\{k_{AB}\}_{K_{SB}}$



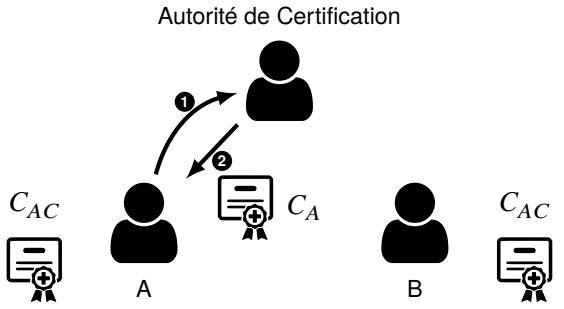
⇒«*token*» $\{k_{AB}\}_{K_{SB}}$



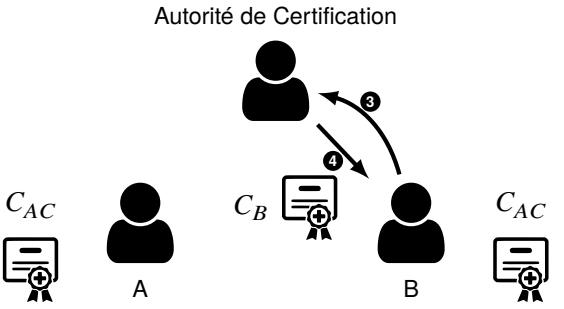
⇒échange utilisant k_{AB}
⇒échange utilisant k_{AB}

⟹ **Le serveur doit être accessible tout le temps par A et B.**

Enregistrement auprès du tiers de confiance

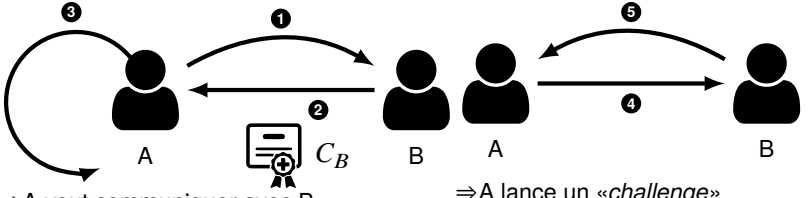


⇒A s'enregistre auprès de l'AC
⇒l'AC donne un certificat C_A à A
Signé avec la clé privée associée à C_{AC}



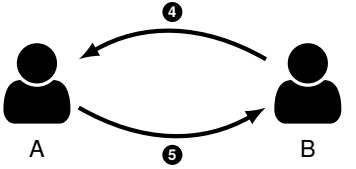
⇒B s'enregistre auprès de l'AC
⇒l'AC donne un certificat C_B à B
Signé avec la clé privée associée à C_{AC}

Comment établir un canal sécurisé entre A et B à un instant t ?



⇒A veut communiquer avec B
⇒B envoie son certificat C_B à A
⇒A **vérifie** que C_B est bien signé par C_{AC}

⇒A lance un «*challenge*»
⇒B s'authentifie auprès de A avec une «*réponse*»
une clé de session k_S est partagée



⇒échange utilisant k_S
⇒échange utilisant k_S

⟹ **L'AC n'a pas besoin d'être tout le temps accessible par A et B.**
⟹ **Les certificats embarquent la confiance.** *Ici, seul B s'est authentifié auprès de A (modèle du Web).*

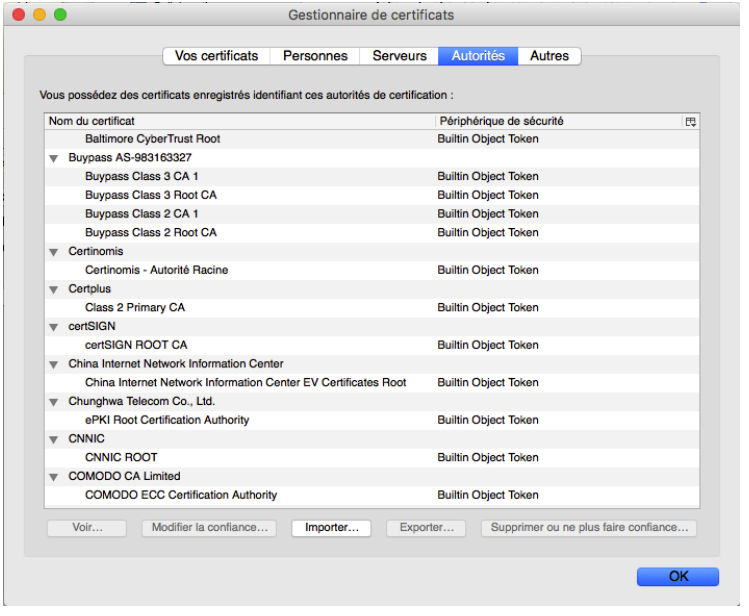
Connaissance et acceptation d'un tiers de confiance

Cela consiste à **adhérer** auprès de l'**autorité de certification** qui représente le tiers de confiance.

Cette CA, «*Certification Authority*», **émets** des certificats.

Cet organisme intègre sa **clé publique** au niveau :

- du **navigateur** de la machine dans le cas de la sécurisation d'une transaction web ;
- du **système d'exploitation** pour la vérification des mises à jour ou l'installation de logiciel.



3. Les bases de la cryptographie

g. Certificats électroniques

Comment connaitre les autorités de certification ?

- Elles sont directement intégrées par les éditeurs dans les systèmes d'exploitation et/ou les navigateurs ;
- L'utilisateur est également libre de rajouter l'autorité de certification de son choix si il choisit de faire confiance à des certificats signés par une autorité non-intégrée dans son navigateur.

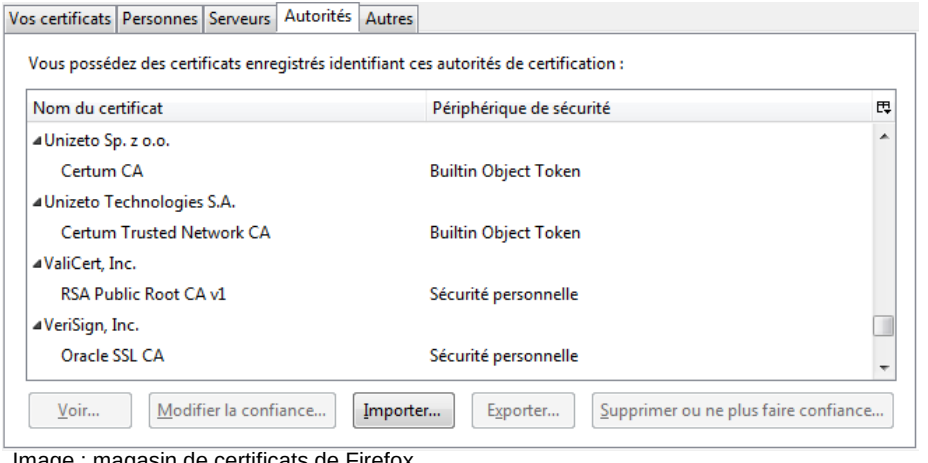


Image : magasin de certificats de Firefox

3. Les bases de la cryptographie

g. Certificats électroniques

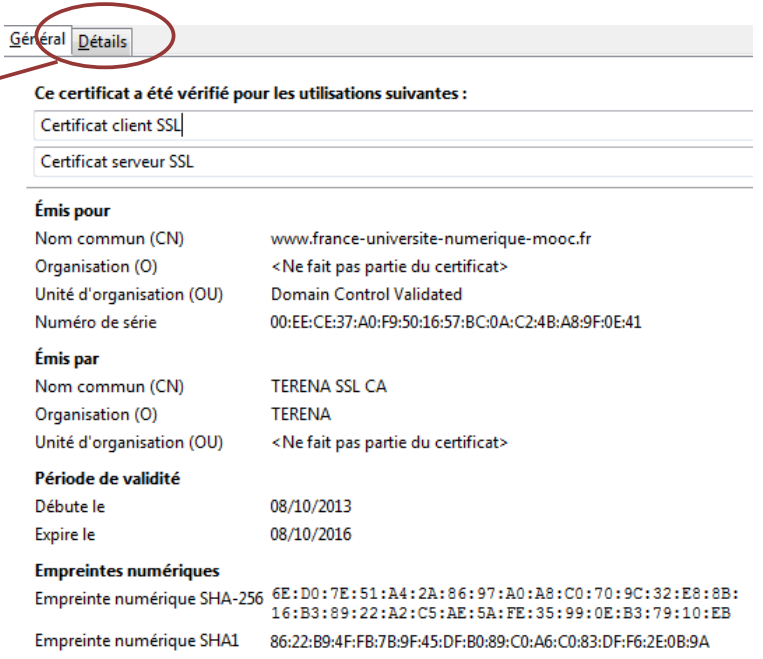
Exemple d'un certificat pour le site web www.france-universite-numerique-mooc.fr

Les détails techniques du certificat, la clé et la signature se trouvent dans **Détails**

Détenteur de la clé publique

Autorité de certification

Dates de validité du certificat



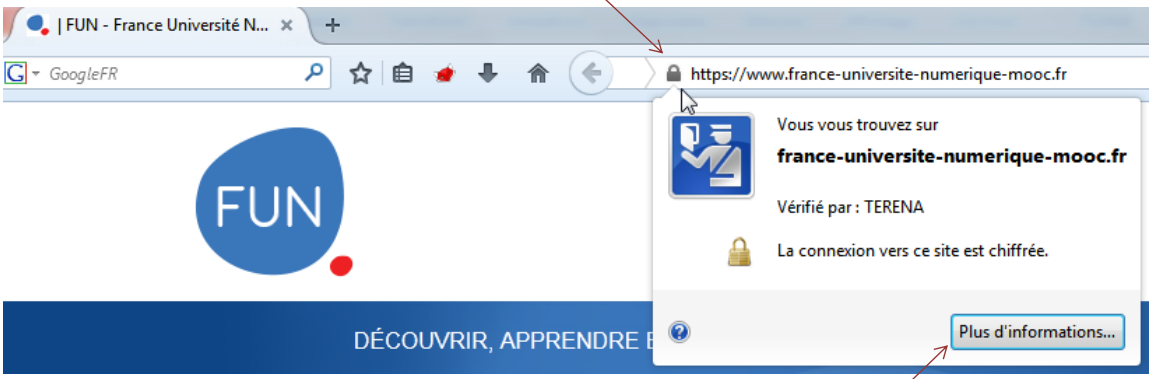
3. Les bases de la cryptographie

g. Certificats électroniques

Où trouver les certificats dans un navigateur ?

Exemple avec Firefox pour ouvrir le certificat d'un site WEB

Cliquer sur le cadenas à côté de l'URL

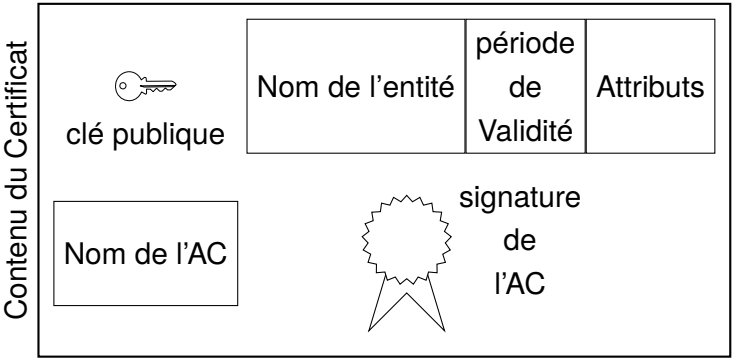


Cliquer ici pour afficher le certificat

Notion de certificat

Un **certificat** permet d'associer une **clé publique** à une **entité** (une personne, une machine, ...) afin d'en assurer la **validité**.

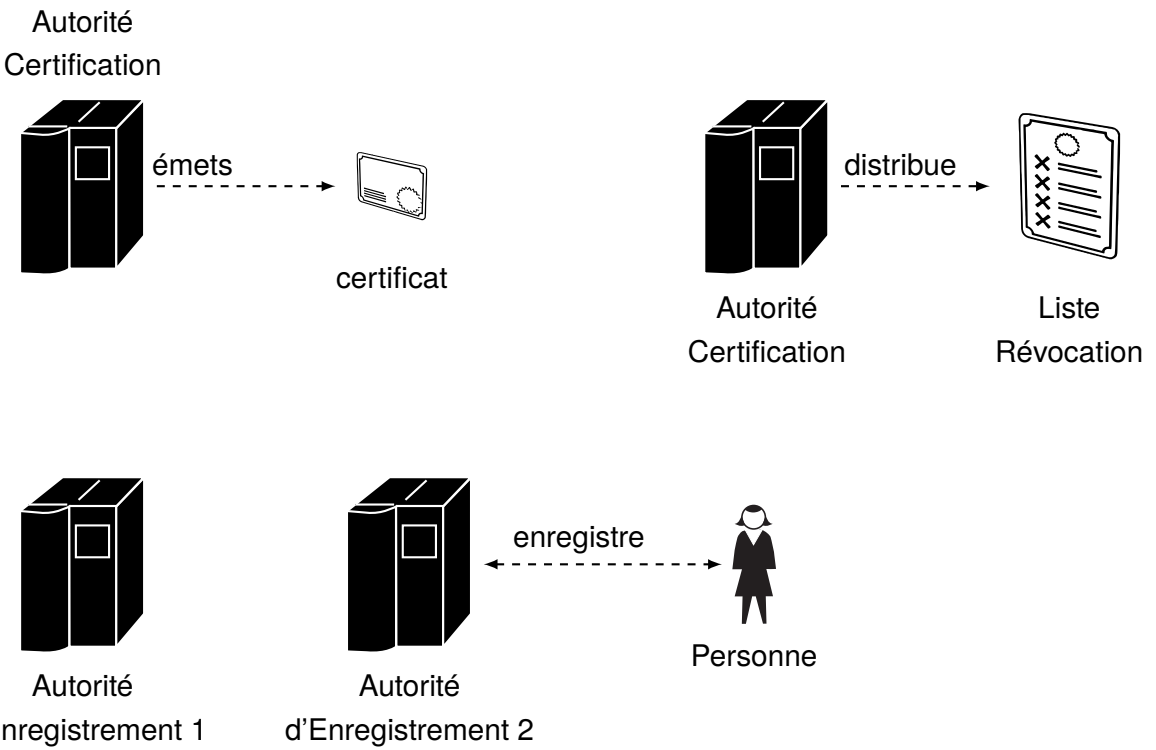
Ce certificat doit ne pas pouvoir être modifié, ni créer sans contrôle : il faut à la fois intégrité et authentification du créateur.



L'**Autorité de Certification** doit assurer des fonctions de **gestion** de certificats (émission, révocation, stockage, récupération et fiabilité des certificats), elle doit :

- **délivrer** les certificats ;
- assigner une **date de validité** aux certificats (équivalent à la date limite de péremption des produits alimentaires) ;
- **révoquer** éventuellement des certificats avant cette date en cas de compromission de la clé privée associée (ou du propriétaire).

Composition d'une PKI



peut exister en plusieurs exemplaires :
distribution géographiques,
catégories de clients...

Utilisation de PKI, reposant sur des AC déjà connues comme Verisign, Thawte *etc.*

Il existe plusieurs «classes» de certificat en fonction du niveau de confiance demandé et du prix du traitement.
Des informations d'identité plus ou moins précises sont demandées en fonction du niveau de confiance choisi.

- **Classe 1** = une **clé publique** associée à une **adresse email**.
  - le demandeur recoit automatiquement un mail de confirmation ;
  - il n'y a pas de vérification de l'identité du titulaire.
    Il permet de faire de la signature de courrier électronique et du chiffrement de contenu. suffisant pour un particulier, service parfois gratuit.

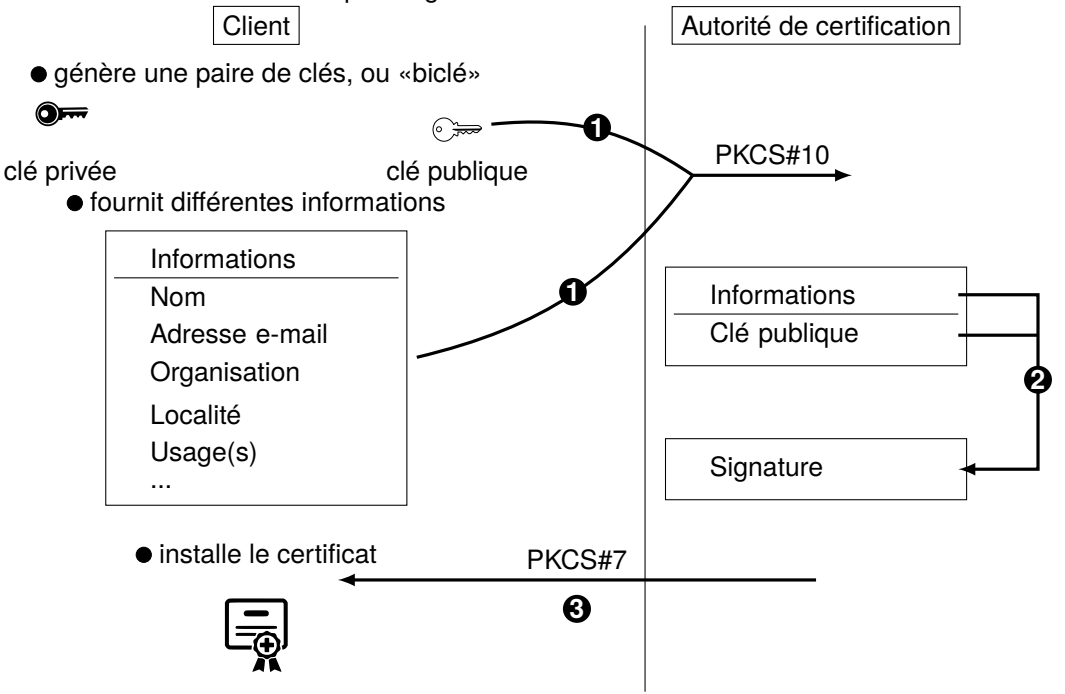
- **Classe 2** = la vérification des informations d'identité fournies est effectuée,
  - la présentation physique peut être nécessaire ;
  - il y aura compensation financière en cas de litige ;
  - il permet de faire :
    - la même chose qu'un classe 1, mais avec de plus grandes garanties
    - de la signature d'application : lors du déploiement de ce logiciel, on pourra vérifier qu'il provient bien de la société de développement et qu'il n'a subit aucune altération.

    Pour un usage professionnel le certificat de classe 2 ou supérieur est nécessaire.

- **Classe 3** = la vérification des informations d'identité fournies est effectuée,
  - la présentation physique est nécessaire ;
  - il y aura compensation financière en cas de litige ;

  Un certificat de classe 3 permet de créer sa propre CA et de délivrer des certificats en interne.

L'utilisateur peut lui-même créer son couple de clés et renseigner les informations de son certificat. Il construit une requête de certificat et la transmets à l'AC pour signature.

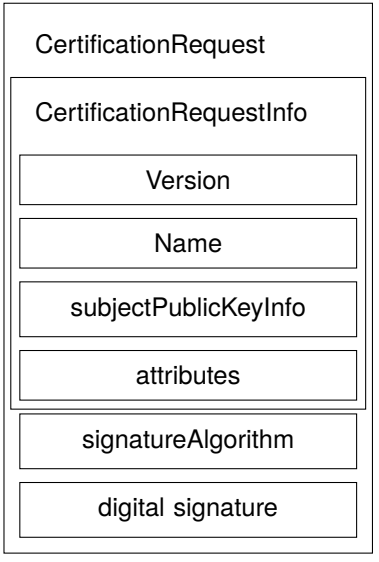


⇒L'utilisateur envoi **sa clé publique**, ainsi que ces informations d'identification dans une requête au format PKCS#10 ;

⇒l'autorité de certification effectue la **signature** de ces informations après vérifications ;

⇒le **certificat** est retourné à l'utilisateur au format PKCS#7.

PKCS#10



In cryptography, PKCS #11 is one of the Public-Key Cryptography Standards,[1] and also refers to the programming interface to create and manipulate cryptographic tokens.
The PKCS #11 standard defines a platform-independent API to cryptographic tokens, such as hardware security modules (HSM) and smart cards, and names the API itself "Cryptoki" from "cryptographic token interface" and pronounced as "crypto-key".

PKCS#7

RFC 2315 : «*PKCS #7: Cryptographic Message Syntax Version 1.5*»
compatible with Privacy-Enhanced Mail (PEM) in that signed-data and signed-and-enveloped-data content, constructed in a PEM-compatible mode, can be converted into PEM messages without any cryptographic operations.
RFC 1422 : «*Privacy Enhancement for Internet Electronic Mail: Part II: Certificate-Based Key Management*»

La création d'un certificat peut s'accompagner de son stockage local

Ce stockage peut être fait dans un «*Key Store*», une sorte de BD locale accessible par différentes applications, au travers d'une API :

- PKCS#11 sous Unix,
- CAPI, «*Crypto API*» sous Windows,
- JCA, «*Java Crypto API*» sous Java.

La création d'un certificat peut s'accompagner de son échange sur le réseau

Les certificats peuvent être stockés dans des annuaires comme LDAP, «*Lightweight Directory Access Protocol*».

Lors de l'établissement d'une communication sécurisée, un des interlocuteurs (le plus souvent le serveur) transmets son certificat à l'autre interlocuteur.

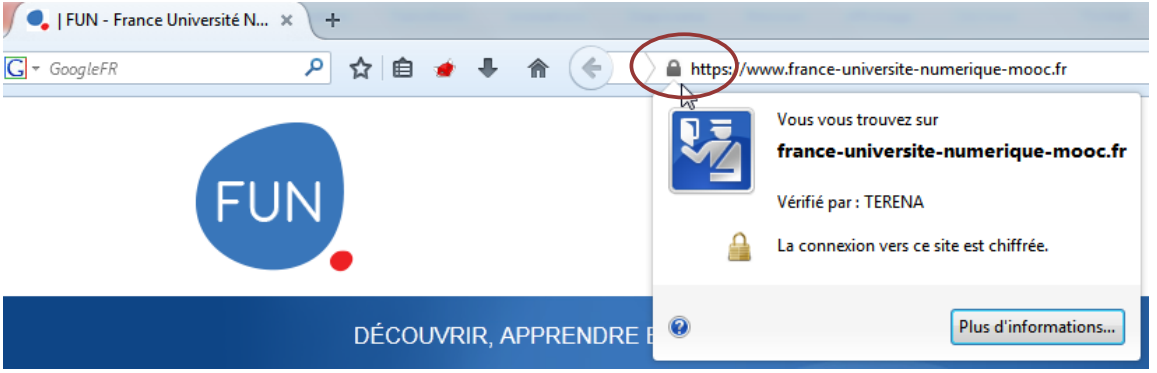
Ainsi, l'interlocuteur peut **vérifier la signature** de l'autre interlocuteur.

Dans le cas de SSL, «*Secure Socket Layer*», ou TLS, «*Transport Layer Security*», c'est le **serveur Web** qui remets son certificat au client (ce certificat peut être accompagné de la «chaîne de certification» qui le lie au certficat de l'autorité de certification qui, lui, **doit être connu** du client).

Dans le cas, où l'on veut communiquer tout de suite en mode chiffré, il faut récupérer le certificat dans un «*certificate repository*» ou dépôt de certificats.

3. Les bases de la cryptographie

g. Certificats électroniques



Puisque le certificat du site WEB est disponible et valide, cela amène donc deux avantages à l'utilisateur, caractéristiques du HTTP**S**

- Nous sommes confiants que **le site WEB est légitime** (i.e. le certificat a été vérifié et signé par une autorité de certification de confiance) ;
- Puisque le certificat contient la clé publique du site WEB, nous pouvons donc **chiffrer nos connexions vers ce site** (méthode : chiffrement avec la clé publique du destinataire comme nous l'avons vu au préalable dans ce cours).

Certificat : contenu détaillé

Certificat au format X509 v3

Subject	p-fb.net
Issuer	Mon AC
Not Valid after	1/1/2025
Not Valid before	1/1/2019
CDP, *CRL Distribution Points*	URL=http://fdn/CRL/ca.crl
AIA, *Authority Information Access*	URL=http://fdn/ca.crt
Public key Algorithm	RSA Encryption (1.2.840.113549.1.1.1)
Extended Key Usage (2.5.29.37)	Client Authentication (1.3.6.1.5.5.7.3.2)
	Email Protection (1.3.6.1.5.5.7.3.4)
Serial number	1054459
Public Key	256 bytes : A7 A1 7F A7 14 9F C9 65 6A F1 C2 3F 94 5A 98 4D C0 3F 35 15 85 4E 86 00 D4 08 A4 CF 6E B8 31 26 8D C3 0D 16 58 DD BD 58 33 CB A5...
...	
Signature de l'AC	128 bytes : A1 12 73 46 DA 9B 70 1E F5...
Fingerprints	SHA-256 EE 00 06 3B 8C 94 B4 79 C5 C5 3C 1A 55 87 A8 4B C5 06 6A 26 29 27 C3 6B 5B 6D F5 C9 CE 5E BA 6B
	SHA-1 27 9B 59 5E 4C 1F E0 EB 99 62 EB B7 25 92 38 49 35 9B A8 4E

identité du sujet : FQDN ou identité

émetteur du certificat

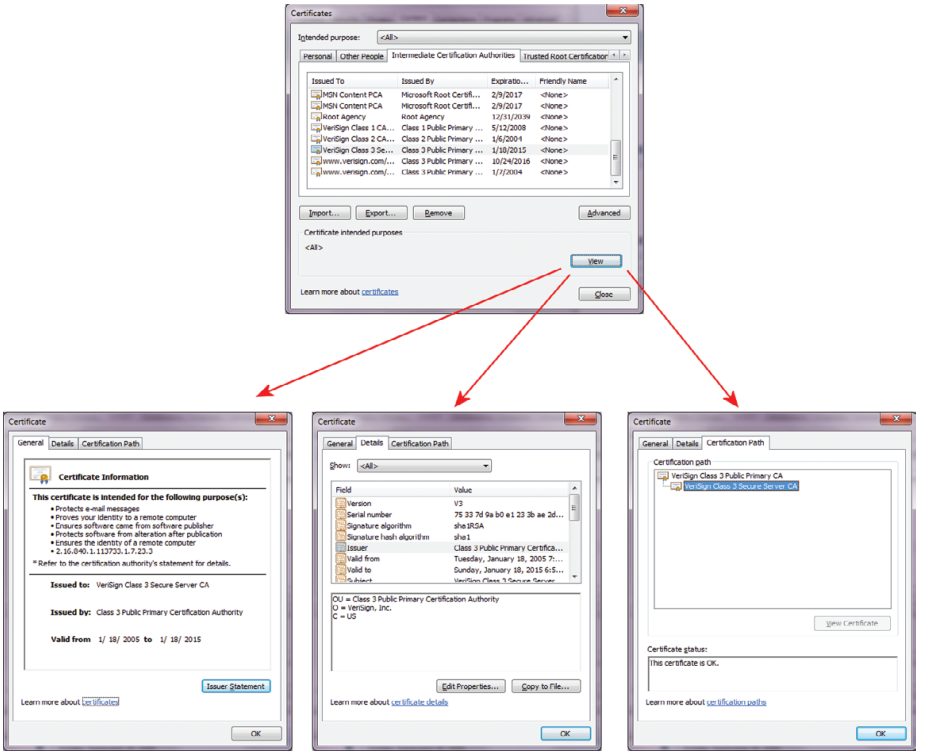
durée de validité

les usages possibles de la clé publique

permet d'identifier de manière unique le certificat pour la CRL par exemple

permet d'identifier le certificat et son contenu : il est calculé localement

Le contenu d'un certificat



Deux formats de stockage pour le certificat

- DER, «*Distinguished Encoding Rules*»
  - représentation des différents champs au format ASN.1 :

```
ASN.!: Certificate ::= SEQUENCE{
           tbsCertificate      TBSCertificate,
           signatureAlgorithm   AlgorithmIdentifier,
           signature         BIT STRING
           }
```

- PEM, «*Privacy Enhanced Mail*»
  - utilisation de l'encodage base64 pour coder le contenu du certificat au format DER ;
  - un délimiteur de début et de fin :

```
-----BEGIN CERTIFICATE-----
MIIE9DCCA9ygAwIBAgISAW8vmnSJXCcjcbEoeBrtUpckMA0GCSqGSIb3DQEBCwUA
MEoxCzAJBgNVBAYTAlVTMRYwFAYDVQQKEw1MZXQncyBFbmNyeXB0MSMwIQYDVQQD
ExpMZXQncyBFbmNyeXB0IEF1dGhvcml0eSBYMTAeFw0xNjAxMTgxOTE5MDBaFw0x
...
NjA0MTcxOTE5MDBaMBMxETAPBgNVBAMTCHAtZmIubmV0MIIBIjANBgkqhkiG9w0B
AQEFAAOCAQ8AMIIBCgKCAQEAr3DjZ6mag9uY8JzAavAD9gRhtZQdMtvf9dcFDAgy
e5j4/TCy/Q+ttiYAE9Ap5EO5P32/55XYU5n8c3OyIDQQTP/P7I+ncutNIj53f2IC
-----END CERTIFICATE-----
```

Les différentes extensions des fichiers

- .cer, .crt, .der : en général des certificats au format DER ;
- .pem : au format PEM ;
- .p7b, .p7c : PKCS#7 «Signed Data structure without data, just sertificate(s) or CRL(s)»
- .p12 : PKCS#12, contient le certificat (clé publique comprise) et la clé privée chiffrée avec un chiffrement symétrique pour empêcher sa lecture ;
- .pfx : prédécesseur de PKCS#12.

Le contenu du certificat

Il est construit suivant une **norme** reconnue internationalement pour faciliter **l'interopérabilité**.

Un certificat est un petit fichier séparé en **deux parties** :

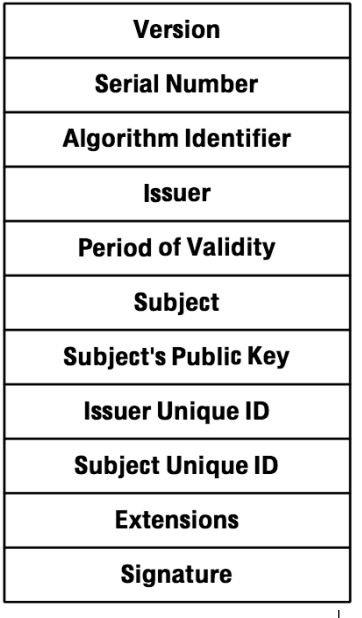
- une contenant les informations suivantes (norme ISO X509 utilisée par les annuaires) :
  - le **nom** de **l'autorité** de **certification** *appelé Issuer*
  - le **nom** du **propriétaire** du certificat *appelé Subject*
  - la **date** de **validité** du certificat *très important*
  - l'algorithme de chiffrement utilisé
  - la **clé publique** du propriétaire
  - un **numéro** de **version** *définie la version de X509 utilisée*
  - un **numéro** de **série** *identifier le certificat sur la CA qui l'a délivré*
  - **l'algorithme** de **signature** utilisé
  - les **usages** du certificat *signature, chiffrement, authentification web…*
  - des **extensions**
- une autre contenant la **signature** de **l'autorité** de **certification**

La notion de DN, « Distinguished Name »

*The distinguished name is a label that follows the X.500 standard. This standard defines a naming convention that can be employed so that each subject within an organization has a **unique name**.*

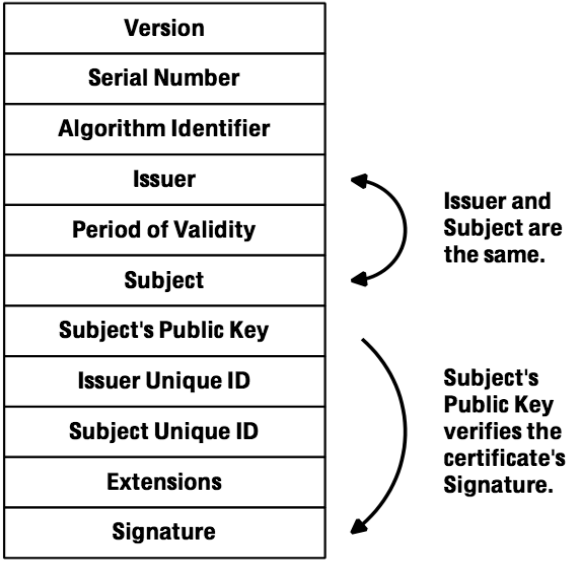
An example is { Country = US,
Organization = Real Secure,
Organizational Unit = R&D,
Location = Washington }.

CAs use distinguished names to identify the owners of specific certificates.

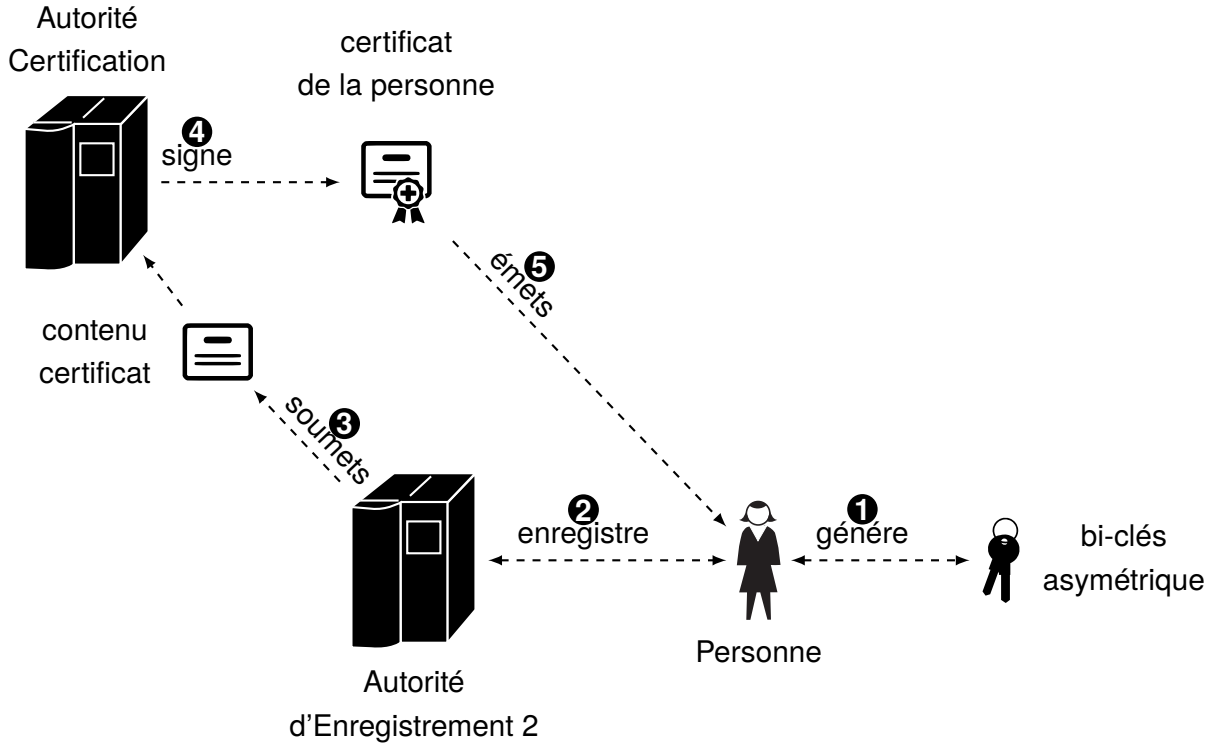


Le certificat de la « CA »

Un certificat « auto-signé »
Il est
- installé directement dans l'OS ou le navigateur Web
- signé par la CA elle-même : intégrité, preuve de possession de la clé privée.



Composition d'une PKI : enregistrement



L'autorité de certification doit être de confiance

- ▷ Elle doit être plus «*qu'un logiciel*» : elle englobe des procédures, des «policies», du matériel, des bâtiments, des gens pour vérifier les informations d'identité etc.
- ▷ Chaque CA doit avoir un CPS, «*Certification Practices Statement*» qui précise :
  - ◇ Comment les identités sont vérifiées ;
  - ◇ Quelles sont les étapes suivies par la CA pour générer un certificat, le maintenir et le distribuer ;
  - ◇ Les éléments qui permettent de justifier la confiance que l'on peut lui apporter et comment elle va s'acquitter de ses responsabilités ;
  - ◇ Les recours juridiques possibles dans le cas où l'AC est compromise ;
  - ◇ Que faire en cas de compromission de sa clé privée ou de sa perte ;
  - ◇ Les données qui seront inscrits dans le certificat ;
  - ◇ La gestion de la révocation ;
- ▷ Ce sont des documents qui doivent être rédigés par : les ressources humaines, le responsable des ressources informatiques et le service juridique.
- ▷ Cette CPS doit **être consultée** afin de **vérifier** que l'entreprise ou l'individu peut l'utiliser **conformément** à ses exigences.

Confiance dans la PKI

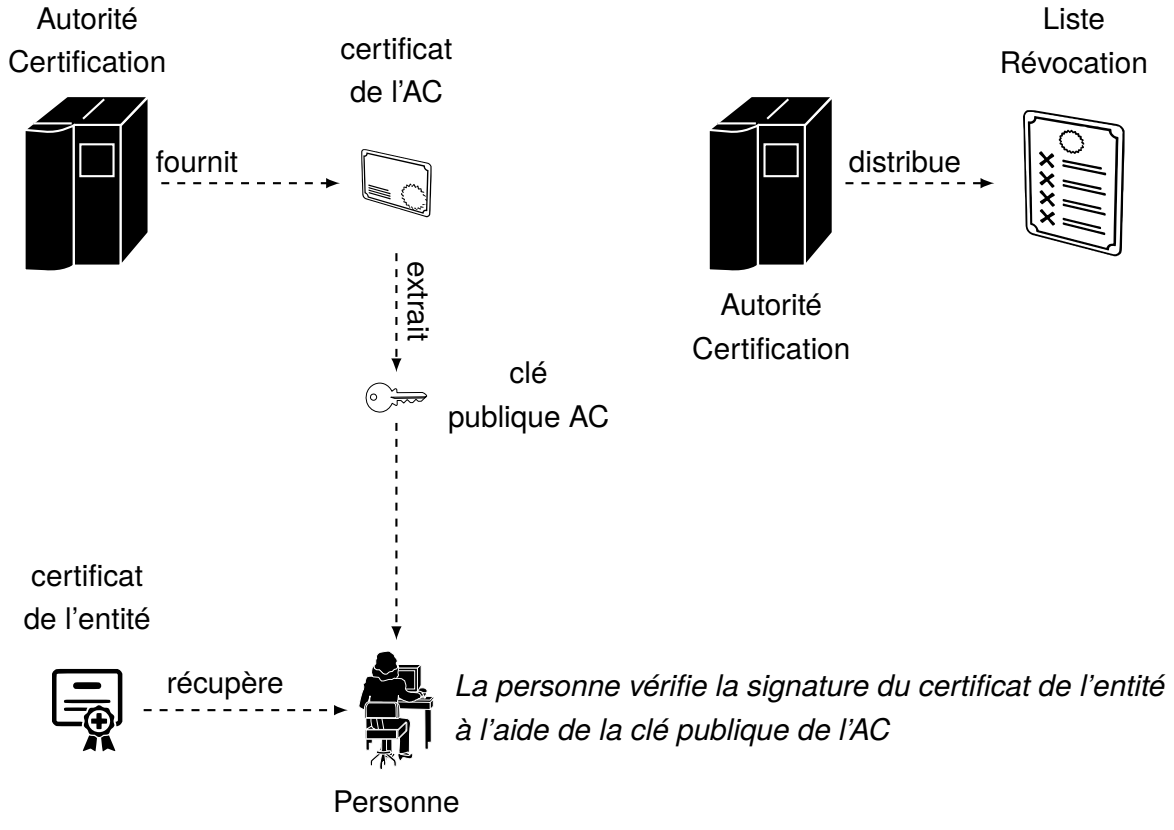
L'ensemble des personnes et des services doivent faire confiance à la PKI pour :

- □ signature des courriers;
- □ chiffrement ;
- □ authentification sur des applications maison…

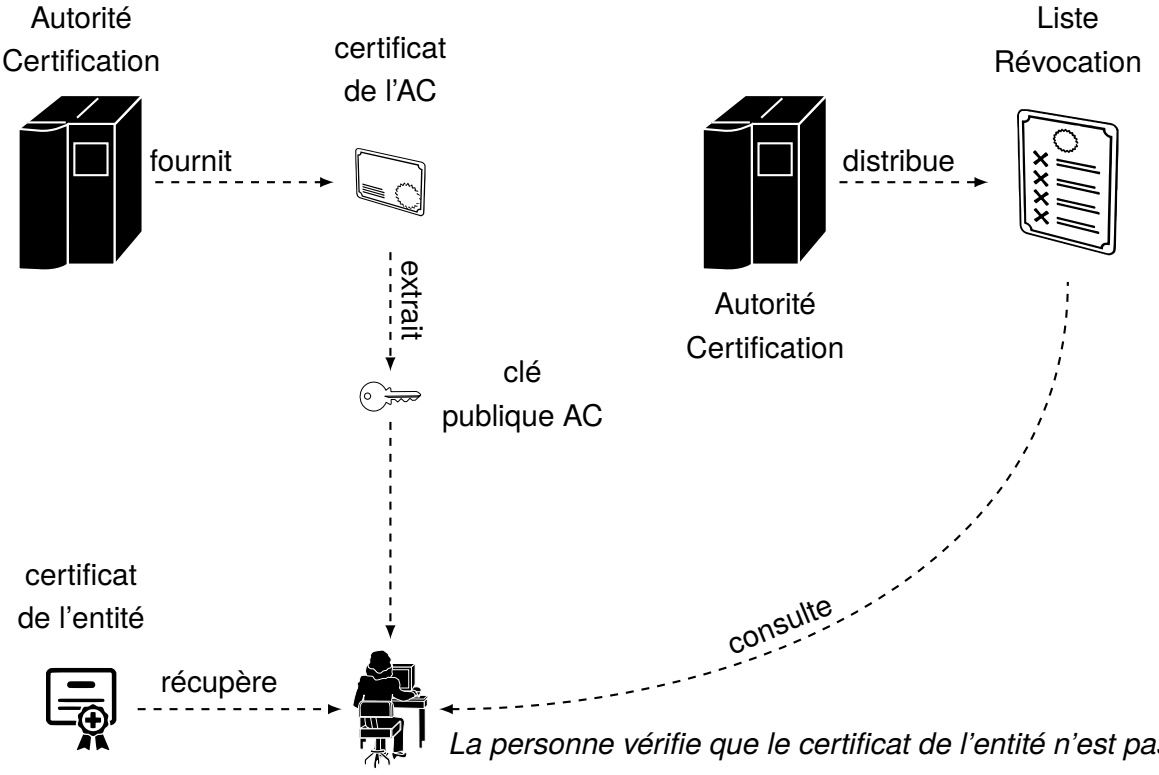
Ils doivent être capable de :

- ▷ **vérifier** un certificat ;
- ▷ contacter l´Autorité de Certification afin de vérifier la **validité du certificat** auprès de la **liste de révocation**.

Composition d'une PKI : utilisation et vérification du certificat

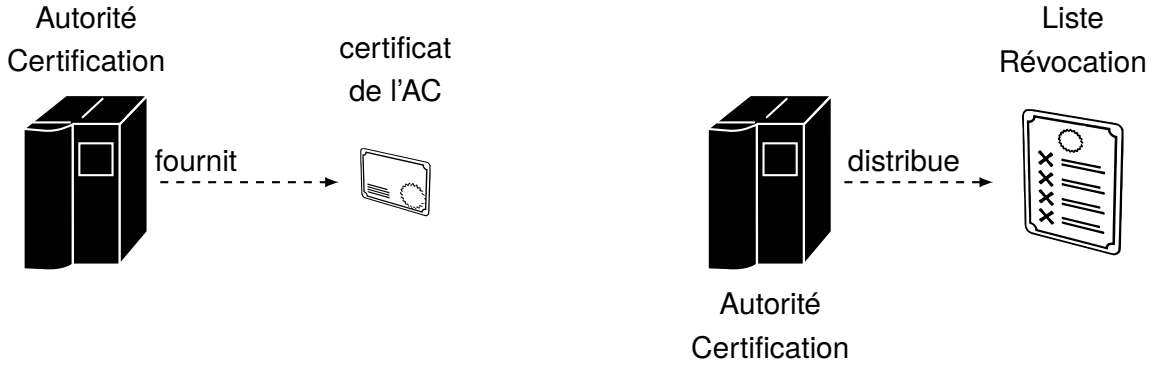


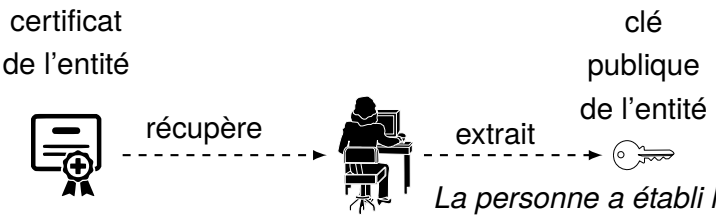
Composition d'une PKI : consultation liste révocation



La personne vérifie que le certificat de l'entité n'est pas révoqué (la liste de révocation est elle-même signée par l'AC)

Composition d'une PKI : établissement de la confiance





La personne a établi la confiance dans l'entité,
elle extrait la clé publique de l'entité
elle peut ***authentifier*** *l'entité ou lui transmettre un* ***message chiffré****.*

Des certificats pour qui ? Pour Quoi ?

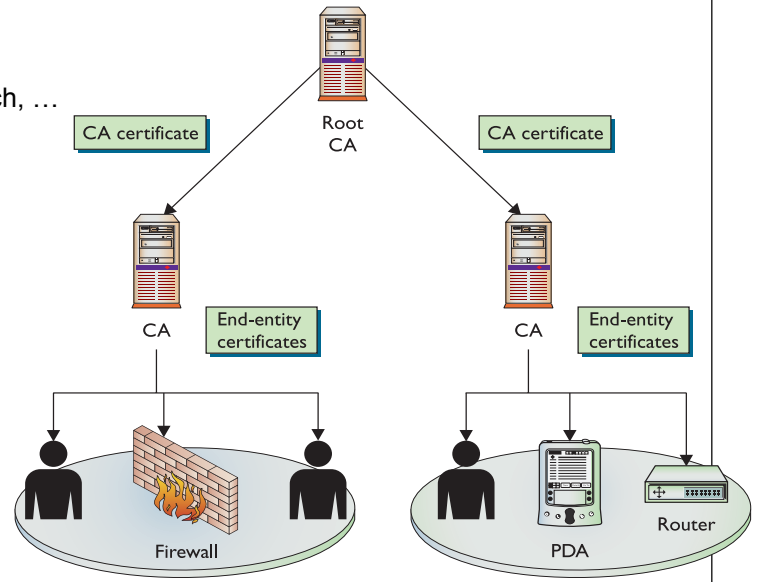
Certificate Attributes

L'attribution de certificat :

- À la CA ;
- À des CAs intermédiaires ;

*Les certificats suivant sont appelés « end-entity certificate » car ils ne **peuvent servir à signe**r de nouveaux certificats !*

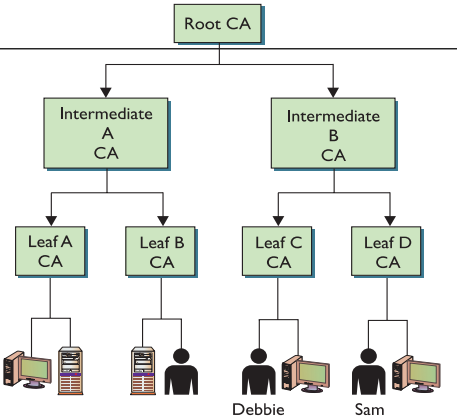
- À des individus ;
- À des matériels réseau actif : firewall, switch, …
- À des postes de travail : laptop, PDA, etc.



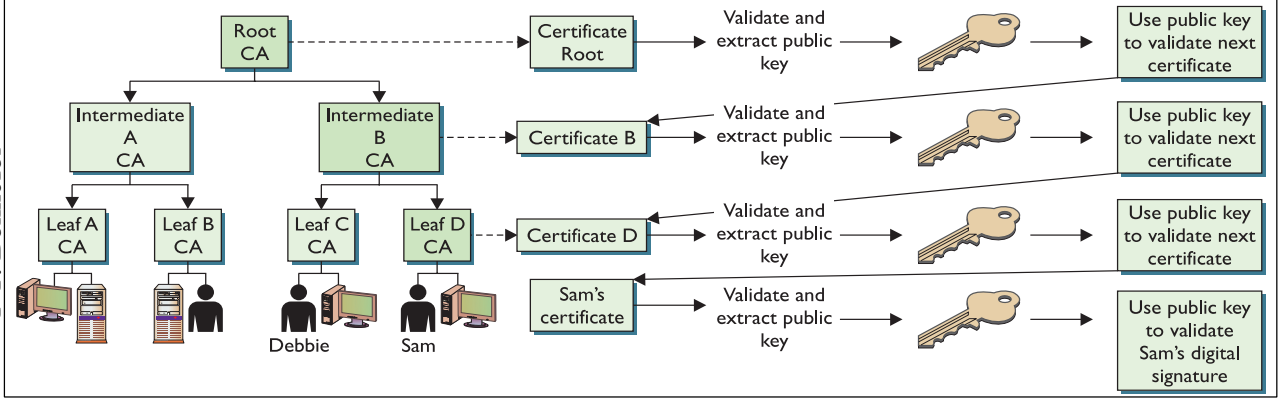
Valider la « chaîne » de certificat

La notion de chaîne de certificat

Il peut exister une « hiérarchie de confiance » :
- simplifier les procédures d'enregistrement ;
- simplifier la gestion des certificats ;
- spécialiser certaines CA intermédiaires dans certain domaines.



Il faut valider l'ensemble des certificats de la chaîne

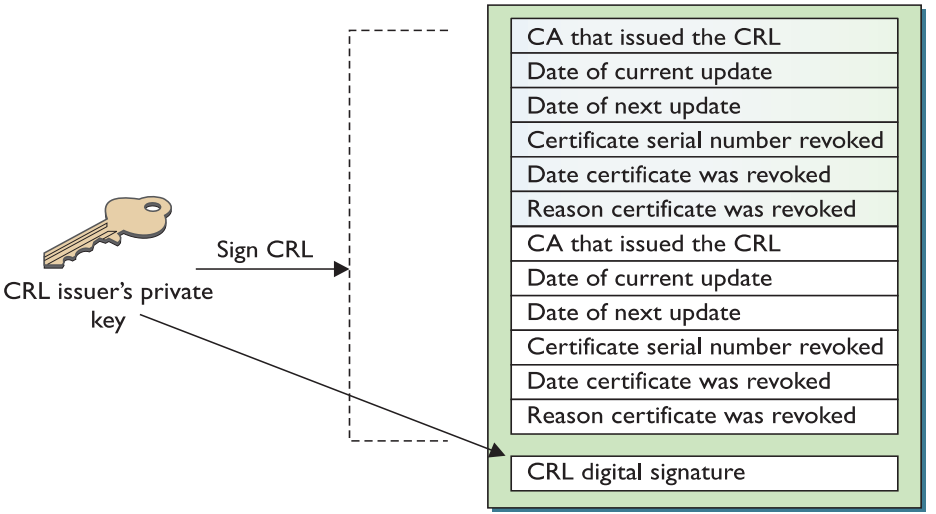


Vérification d'un certificat et la CRL

Vérification d'un certificat

L'utilisation que fait le titulaire du certificat ne concerne plus l'AC, mais les diverses applications qui sont compatibles.

- Vérifier que le **certificat** n'a **pas expiré**, c'est-à-dire que sa **date** de **validité** est **correcte** ;
- **Authentifier l'empreinte** (provenance de l'AC avec sa clé publique) et **l'intégrité** (pas de modification du certificat avec une fonction de hachage) ;
- **consulter** la **liste** de **révocation** de l'AC pour savoir s'il n'a pas été révoqué (l'AC peut retirer sa confiance à un certificat, le propriétaire du certificat peut avoir perdu sa clé privée...).
  Cette liste de révocation ou CRL, « certificate revocation list » est constituée de :



Les raisons d'invalider un certificat

Le fonctionnement de la CRL
C'est une « blacklist ».
Il existe une « delta CRL » qui permet d'indiquer uniquement les modifications par rapport à une version antérieure pour limiter la taille de cette liste.

Reason Code	*Reason*
0	*Unspecified*
1	*All keys compromised; indicates compromise or suspected compromise*
2	*CA compromise; used only to revoke CA keys*
3	*Affiliation changed; indicates a change of affiliation on the certificate*
4	*Superseded; the certificate has been replaced by a more current one*
5	*Cessation; the certificate is no longer needed, but no reason exists to suspect it has been compromised*
6	*Certificate hold; indicates the certificate will not be issued at this point in time*
7	*Remove from CRL; used with delta CRL to indicate a CRL entry should be removed*

La raison n°6 peut indiquer également que le titulaire est en vacances…

Les extensions de certificat

« Key usage extension »
Elles permettent de définir les usages de la clé publique contenue dans le certificat :

- DataEncipherment: pour faire du chiffrement de données
- DigitalSignatures : pour faire de la vérification de signature

Attention : RSA peut faire du chiffrement et de la signature, mais DSA ne peut faire que de la signature !

- KeyEncipherment ou Symmetric Key Encryption : il est parfois nécessaire de transmettre une clé de chiffrement asymétrique pour faire du chiffrement de données lorsque la clé contenue dans le certificat ne peut faire que de la signature (Application : dans SSL)
- NonRepudiation : pour s'assurer de la date à laquelle est faire une opération pour ne pas pouvoir la répudier :
  Alice conteste l'utilisation de sa clé privée pour la signature d'un document, elle insiste sur le fait qu'elle a signalé la compromission de sa clé privée avant que le document ne soit signé.
  La preuve sera fournie par un « Timestamp » signé par une TSA, « TimeStamp Athority » qui prouvera si cette révocation a eu lieu avant la signature dudit document.
- KeyCertSign : la clé sert à vérifier la signature des certificats par rapport à la CA
- CRLSign : la clé peut être utilisée pour signer la CRL
- Etc.

Il existe des extensions « *critical* » ou « *noncritical* » :

- Cela correspond à un « drapeau » placé dans le certificat ;
- Si une extension est marquée « *critical* », il faut que l'extension **doit être comprise** et traitée correctement par le récepteur. Dans le cas où le récepteur ne sait pas gérer une extension critique, alors il ne doit pas utiliser le certificat.
  Le certificat ne peut servir qu'à l'usage spécifié comme critique.
- Si une extension est marquée comme « *noncritical* » alors la clé publique peut servir à d'autres usages que ceux spécifiés dans le certificat.

Identifier les éléments de la sécurité : l'OID

La notion d'OID, « Object Identifier »

Un OID est un identifiant servant à nommer un objet, lorsque l'on désire que cet identifiant soit reconnu de manière internationale et pour une très longue durée.

Cet identifiant est :
- associé à un nœud dans un espace de nom **hiérarchique**
- définie de manière formelle par le standard ASN.1 de l'ITU-T, appelé X690
- une succession de nombre séparés par des points, chacun identifiant un nœud de l'arbre.

Le premier nœud identifie la nature de l'identifiant :
0 : ITU-T
1 : ISO
2 : définition conjointe ITU-T et ISO

L'OID sert, en sécurité, à identifier chaque élément des ceritifcats X.509 : le DN, « distinguished Name », le CPS, « Certificate Practice Statements », etc.
L'OID est utilisé dans de nombreux domaines, comme SNMP, LDAP (RFC 2256) etc.
Il est possible d'obtenir un OID auprès de l'IANA.

Pour CERTINOMIS :

NOS OIDS RGS

OID	USAGE	NIVEAU	GAMME	PRODUIT
1.2.250.1.86.2.2.1.1.1	Authentification	1 étoile	Organisation	CERTINOMIS WEB ACCESS*

Identifier les éléments de la sécurité : l'OID

Exemple sur le RGS français :
Le document accessible à l'URL
http://www.ssi.gouv.fr/IMG/pdf/RGS_fonction_de_securite_Authentification_Serveur_V2-3.pdf
est associé à l'OID 1.2.250.1.137.2.2.1.2.1.10

Si on consulte le site www.oid-info.com, on obtient l'info :

	{iso(1) member-body(2) fr(250) type-org(1)}	(ASN.1 notation)
OID:	1.2.250.1	(dot notation)
	/ISO/Member-Body/250/1	(OID-IRI notation)

Current Registration Authority

Name:	Martine Degardin	If you want to email the current Registration Authority, please replace "&" by "@" in the email address
Address:	AFNOR 11 avenue Francis de Pressensé 93571 Saint-Denis la Plaine Cédex France	

La génération de la « biclé »

Centralisée ou distribuée

- Dans la version centralisée, les clés sont générées **sur un serveur** puis mise à disposition des utilisateurs.
  - Avantages :
    - Utilisation de processeurs cryptographiques spécialisés avec de **bon PRNGs** (la génération peut être coûteuse en ressource de calcul) ;
    - Possibilité de faire du « key recovery » en cas de perte de clé (à ne pas confondre avec le« key escrow ») ;
  - Inconvénients :
    - Obligation de transmettre de manière sécurisée les clés aux utilisateurs ;
    - Obligation d'assurer la disponibilité du serveur ;
    - Pas d'exploitation de la non répudiation pour l'utilisateur
- Dans la version décentralisée, les clés sont générées **par l'utilisateur**.
  - Avantages :
    - Permet la non répudiation ;
    - Risques de pertes ou de mauvaise protection.
  - Inconvénients
    - Utilisation de mauvaises solutions de génération de clés.

Les solutions

- Les clés privées doivent être **stockées de manière chiffrée** à l'aide d'un algorithme symétrique : l'accès à la clé privée est déverrouillé par l'entrée d'une *passphrase* fournie à l'algo de chiffrement.
- La **durée de vie** des clés doit être bien choisie et ne plus être utilisée après, éventuellement elle peut être détruite.
- Les **clé privées** ne doivent **pas** être **partagées**.
- Les logiciels manipulant ces clés doivent être **audités**.

La conservation et le « partage » de la biclé

Tiers de séquestre ou « Key Escrow »

« Entité chargée de conserver les conventions secrètes des utilisateurs, de les leur restituer sur demande ou de les remettre aux autorités judiciaires ou de sécurité sur leur requête. »

Avantages :

- permettre d'avoir accès au contenu d'échanges réalisés par un collaborateur dans le cadre de son travail dans l'entreprise ;
- permettre de faire évoluer l'archivage des documents : déchiffrer un document et le rechiffrer avec des moyens cryptographiques mis à jour.

Moyens :

- Utiliser des clés pour des usages distincts : une **biclé** pour **chiffrer** et une **biclé** pour **signer**. *La biclé utilisée pour chiffrer peut être conservée par l'entreprise pour accéder aux documents du collaborateur.*

Obligation légale

L'utilisation d'un tiers de séquestre peut être nécessaire dans le cas de l'archivage.

Besoins

- **Archiver** un document pendant un certain temps (obligation légale) ;
- **Prouver l'antériorité** d'un dépôt de document (le cachet de la Poste faisant foi...)
- Permettre la **traçabilité** d'une opération.
- Garantir la **non-répudiation**.

But

Fournir la preuve de l'existence d'un message à un instant donné.

Horodateur **neutre** $\Longrightarrow$ opérations techniques :

▷ Aucun contrôle du contenu du message

▷ Pas de contrôle du bien fondé de la requête

Contenu du jeton d'horodatage

- ☐ Politique d'horodatage utilisée
- ☐ Nom du tiers horodateur et son numéro authentification ;
- ☐ Marque de temps ;
- ☐ Empreinte du message à horodater ;
- ☐ Numéro de série unique ;
- ☐ Signature du tiers horodateur ;
- ☐ Diverses autres informations de service.

Archivage électronique, scellement

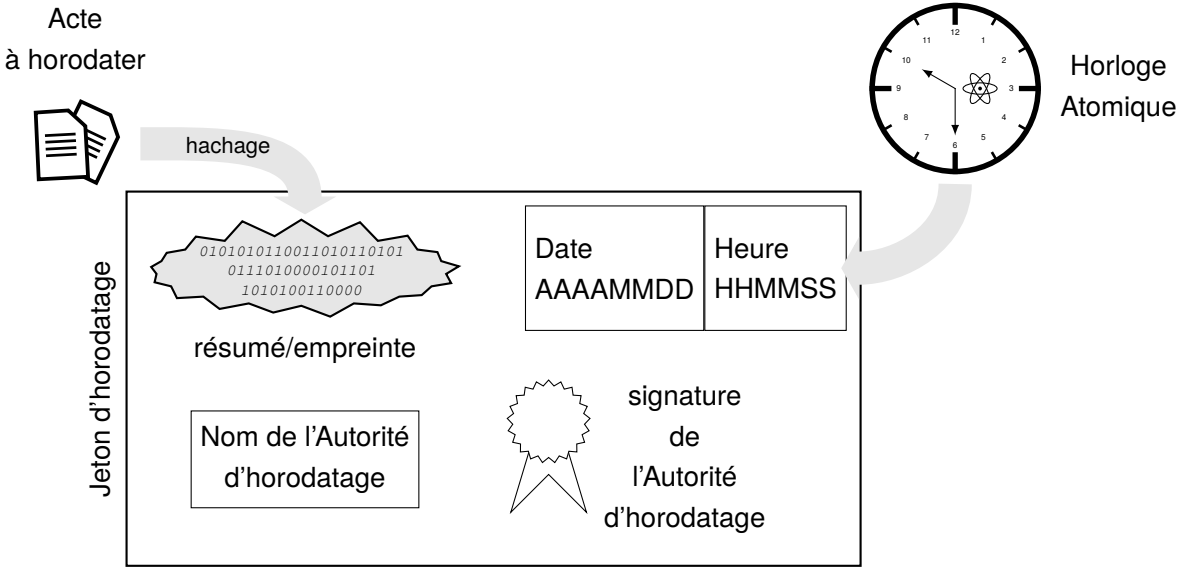
Garantir :

▷ l'intégrité dans le temps ;

▷ la pérennité de l'information.

L'autorité d'horodatage, délivre des «*jetons d'horodatage*» pour **certifier l'existence de données** ou **l'établissement d'une transaction**, à une **date** et une **heure précise** à la seconde près, tel le «*cachet de la poste*» qui fait foi.

⟹Certifier l'intégrité d'un fichier ou d'une opération entre la date d'horodatage et celle de vérification.

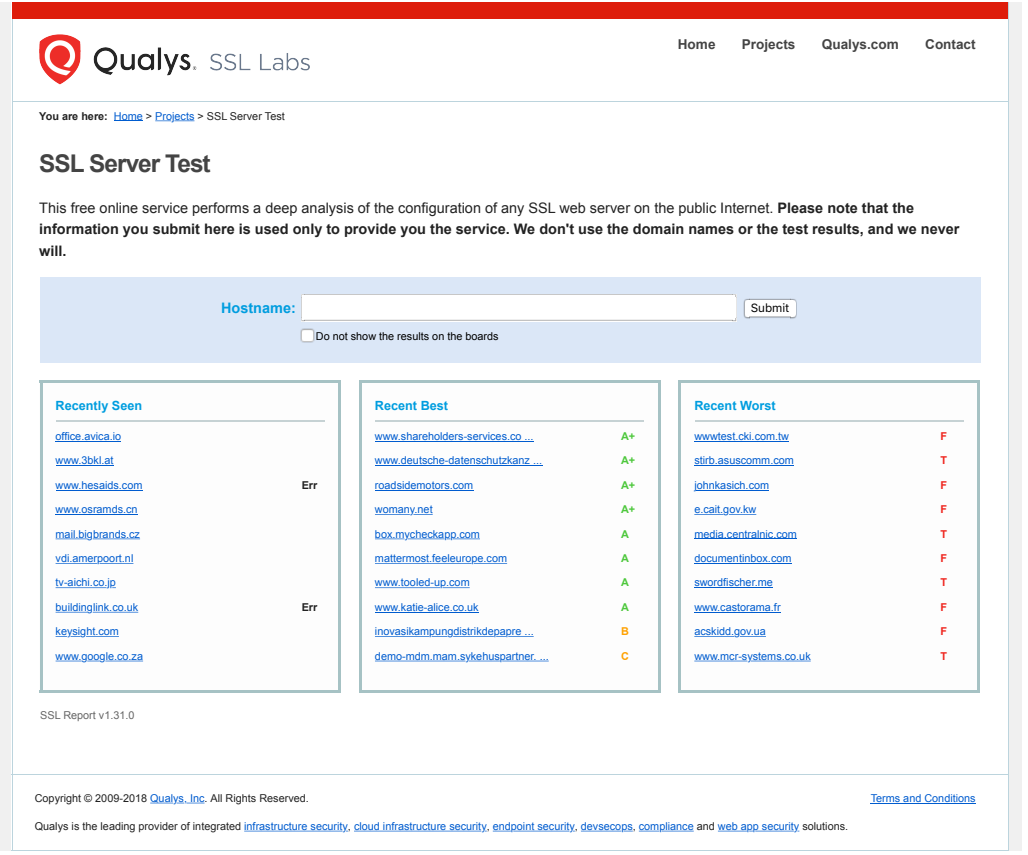


L'heure et la date sont **absolues** données en référence UTC, «*Universal Time Coordinated*», et correspondent à l'heure et la date **courante** lors de la création du jeton.

Le jeton assure **l'intégrité**, **l'antériorité** et **l'opposabilité** (protection contre la contestation liée au temps).

Évaluation de la sécurité de TLS

Audit externe `https://www.ssllabs.com/ssltest/`



Qualys. SSL Labs

Home Projects Qualys.com Contact

You are here: Home > Projects > SSL Server Test

SSL Server Test

This free online service performs a deep analysis of the configuration of any SSL web server on the public Internet. **Please note that the information you submit here is used only to provide you the service. We don't use the domain names or the test results, and we never will.**

Hostname: Submit

Do not show the results on the boards

Recently Seen		Recent Best		Recent Worst	
office.avica.io		www.shareholders-services.co ...	A+	wwwtest.cki.com.tw	F
www.3bkl.at		www.deutsche-datenschutzkanz ...	A+	stirb.asuscomm.com	T
www.hesaids.com	Err	roadsidemotors.com	A+	johnkasich.com	F
www.osramds.cn		womany.net	A+	e.cait.gov.kw	F
mail.bigbrands.cz		box.mycheckapp.com	A	media.centralnic.com	T
vdi.amerpoort.nl		mattermost.feeleurope.com	A	documentinbox.com	F
tv-aichi.co.jp		www.tooled-up.com	A	swordfischer.me	T
buildinglink.co.uk	Err	www.katie-alice.co.uk	A	www.castorama.fr	F
keysight.com		inovasikampungdistrikdepapre ...	B	acskidd.gov.ua	F
www.google.co.za		demo-mdm.mam.sykehuspartner. ...	C	www.mcr-systems.co.uk	T

SSL Report v1.31.0

Copyright © 2009-2018 Qualys, Inc. All Rights Reserved. Terms and Conditions

Qualys is the leading provider of integrated infrastructure security, cloud infrastructure security, endpoint security, devsecops, compliance and web app security solutions.

Vérification qu'un certificat est bien émis depuis la bonne AC

Site de référence `https://www.grc.com/fingerprints.htm`

Gibson Research Corporation · Privacy Recovery

Home ▾ | SpinRite ▾ | Services ▾ | Freeware ▾ | Research ▾ | Other ▾ | Search

Fingerprints

Is your employer, school, or Internet provider **eavesdropping** on your **secure** connections?

1,077 sets of fingerprints checked per day
1,588,775 sets of fingerprints checked for our visitors

Secure browser connections can be intercepted and decrypted by authorities who spoof the authentic site's certificate. But **the authentic site's fingerprint CANNOT be duplicated!**

Domain Name	Certificate Name	EV	Security Certificate's **Authentic** Fingerprint
www.grc.com	grc.com	●	15:9A:76:C5:AE:F4:90:15:79:E6:A4:99:96:C1:D6:A1:D9:3B:07:43
www.facebook.com	*.facebook.com	–	BD:25:8C:1F:62:A4:A6:D9:CF:7D:98:12:D2:2E:2F:F5:7E:84:FB:36
www.paypal.com	www.paypal.com	●	BB:20:B0:3F:FB:93:E1:77:FF:23:A7:43:89:49:60:1A:41:AE:C6:1C
www.wikipedia.org	*.wikipedia.org	–	4B:3E:D6:B6:A2:C7:55:E8:56:84:BE:B1:42:6B:B0:34:A6:FB:AC:24
twitter.com	twitter.com	●	26:5C:85:F6:5B:04:4D:C8:30:64:5C:6F:B9:CF:A7:D2:8F:28:BC:1B
www.blogger.com	*.blogger.com	–	80:72:8D:F5:50:E1:78:55:2E:4E:EA:2C:41:6E:EF:B8:13:48:21:36
www.linkedin.com	www.linkedin.com	–	3A:60:39:E8:CE:E4:FB:58:87:B8:53:97:89:8F:04:98:20:BF:E3:91
www.yahoo.com	*.www.yahoo.com	–	AE:69:9D:5E:BD:DC:E6:ED:57:41:11:26:2F:19:BB:18:EF:BE:73:B0
wordpress.com	wordpress.com	●	79:1A:83:83:21:20:F6:6D:9D:1E:77:5F:ED:89:16:FC:8E:A0:E0:C3
www.wordpress.com	*.wordpress.com	–	54:E0:89:DF:28:53:83:00:10:5D:D4:37:64:FD:E7:D0:F5:ED:5B:C0

Each site's authentic security certificate fingerprint (shown above) was just now obtained by GRC's servers from each target web server. If your web browser sees a different fingerprint for the same certificate (carefully verify the Certificate Name is identical) that forms strong evidence that something is intercepting your web browser's secure connections and is creating fraudulent site certificates.

Custom Site Fingerprinting

In addition to the well-known web sites listed above, GRC's web server can obtain and display the "fingerprint" of any HTTPS-capable public web server's secure connection certificate. Simply enter the domain name of the server you wish to fingerprint, then press Enter or click the "Fingerprint Site" button:

https:// Fingerprint Site

Google and Apple are different: Some visitors are being confused by Google's and Apple's certificate fingerprints which change and may not match. Please see the "What can go wrong with this test?" section at the bottom of this page for an explanation of the complexities.

Extended Validation Certificate

D'après Wikipedia

An **Extended Validation Certificate** (EV) is a certificate used for HTTPS websites and software that **proves the legal entity** controlling the website or software package. Obtaining an EV certificate requires verification of the requesting entity's identity by a certificate authority (CA).

EV certificates use the same encryption as organization-validated certificates and domain-validated certificates: the increase in security is due to the identity validation process, which is indicated within the certificate by the policy identifier.

EV certificates are validated against both the **Baseline Requirements** and the **Extended Validation requirements**, which place additional requirements on how authorities vet companies.

These include manual checks :

- ▷ of all the domain names requested by the applicant ;
- ▷ against official government sources ;
- ▷ against independent information sources ;

and **phone calls** to the company to confirm the position of the applicant.
If the certificate is **accepted**, the **government-registered serial number** of the business as well as the physical address are stored in the EV certificate.

www.apple.com
Issued by: DigiCert SHA2 Extended Validation Server CA
Expires: Monday, 25 March 2019 at 13:00:00 Central European Standard Time
This certificate is valid

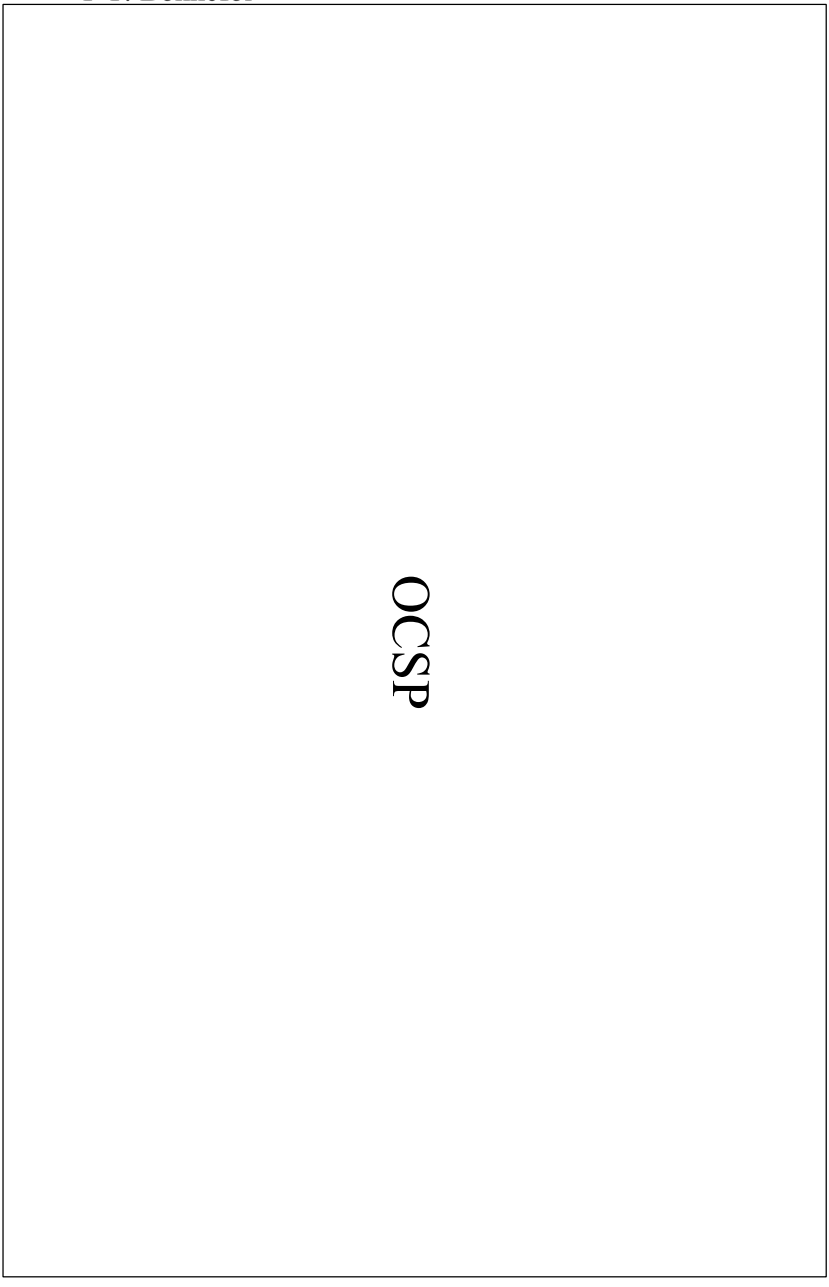
▼ **Trust**

When using this certificate: Use System Defaults ?

Secure Sockets Layer (SSL) no value specified
X.509 Basic Policy no value specified

▼ **Details**

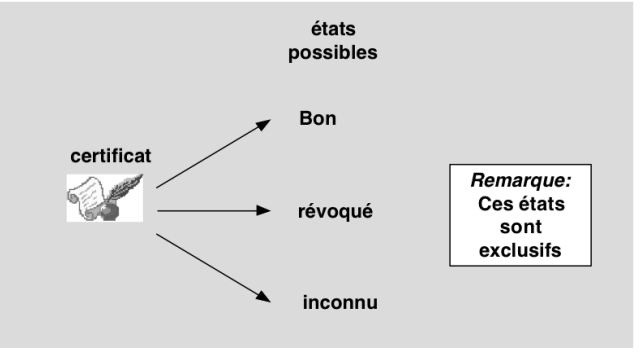
Subject Name	
Business Category	Private Organization
Inc. Country	US
Inc. State/Province	California
Serial Number	C0806592
Country	US
State/Province	California
Locality	Cupertino
Organization	Apple Inc.
Organizational Unit	Internet Services for Akamai
Common Name	www.apple.com


OCSP

OCSP : Online Certificate Status Protocol

Ce protocole :

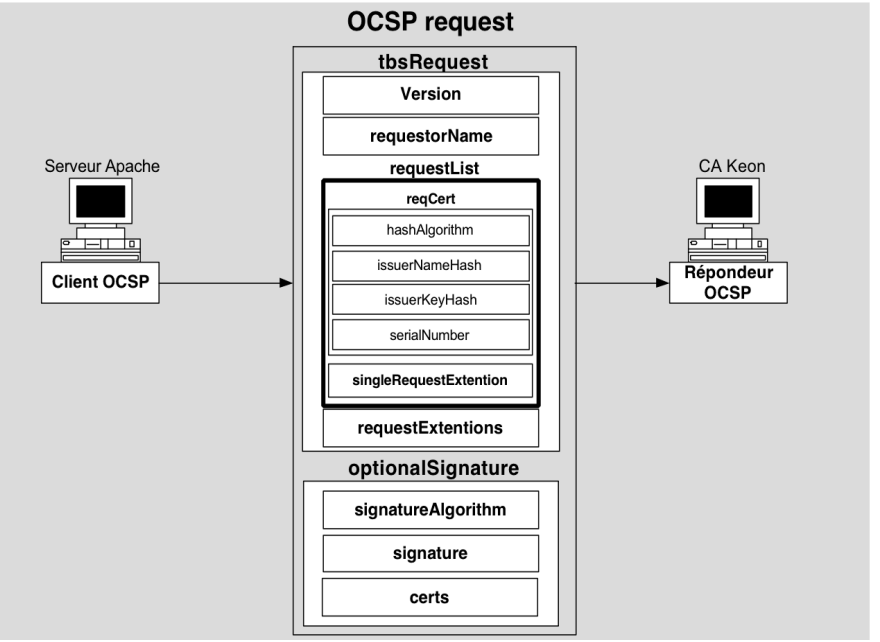
- permet de vérifier que le certificat a ou non été révoqué ;
- fonctionne suivant le mécanisme requête/réponse.



OCSP : la requête

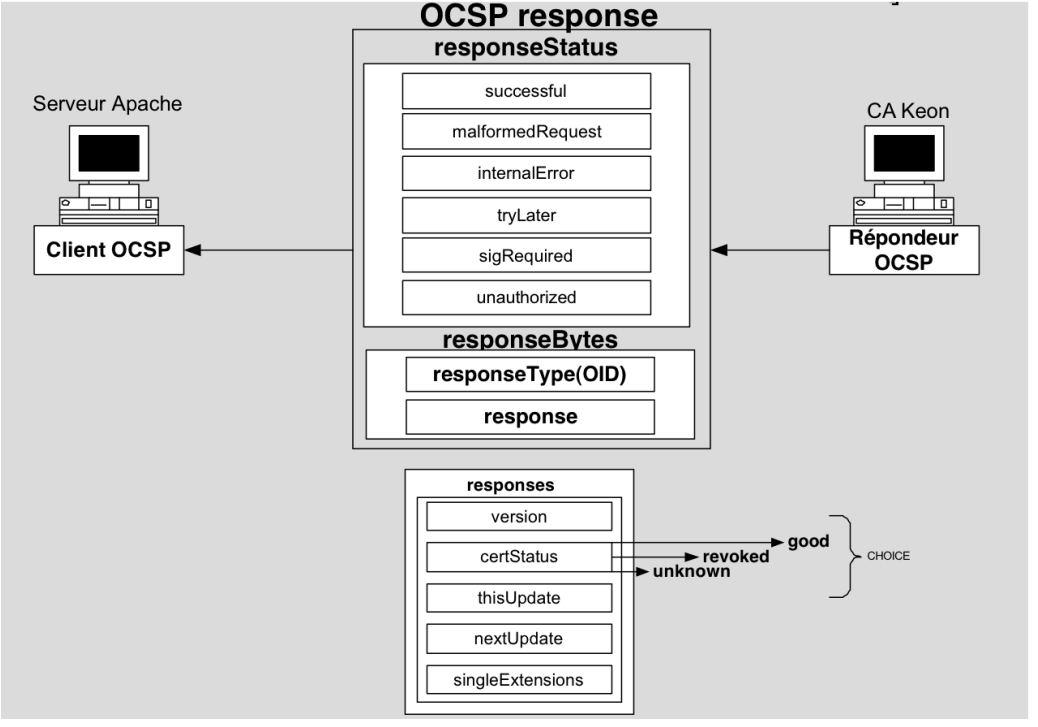
Exemple :

Ici le serveur Apache vérifie le certificat donné par le client, c-à-d le navigateur web auprès de l'AC qui a délivré le certificat.



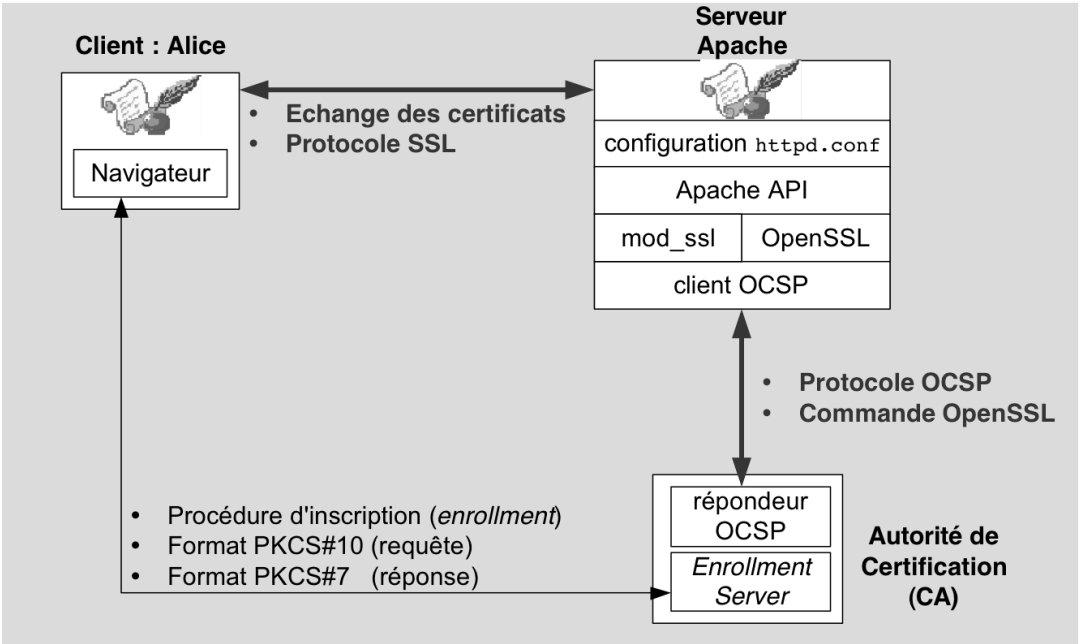
OCSP : la réponse

Ici, le serveur Apache reçoit la réponse qui lui indique le statut du certificat du client.



Intégration Apache et gestion des certificats

Ici, le serveur Apache comme le client sont authentifiés par des certificats issus de la même AC.



Application des certificats
Pour l'établissement de connexion sécurisée

Applications

- Applications de confiance (applet JAVA, pilote Windows XP, …)
- Sécurisation des processus « web services » en particulier les serveurs d'authentification (SSO)
- Horodatage
- Signature
- E-commerce
- Dématérialisation de procédure administrative (Workflow)
- E-Vote

Les Usages

- Messagerie S/MIME : signature (certificat de l'émetteur) et/ou chiffrement (certificat du destinataire)
- SSL ou TLS : en particulier HTTPS pour chiffrer les sessions du client et authentifier le serveur. Plus rarement authentifier le client.
  SSL -> POPS, IMAPS, LDAPS, SMTP/TLS, …
- VPN et IPsec

SSL: Secure Socket Layer

Principe :

- utiliser une couche avec chiffrement et authentification au dessus de TCP/IP ;
- implémenter par dessus le protocole de communication (SMTP, POP, HTTP, etc.) ;

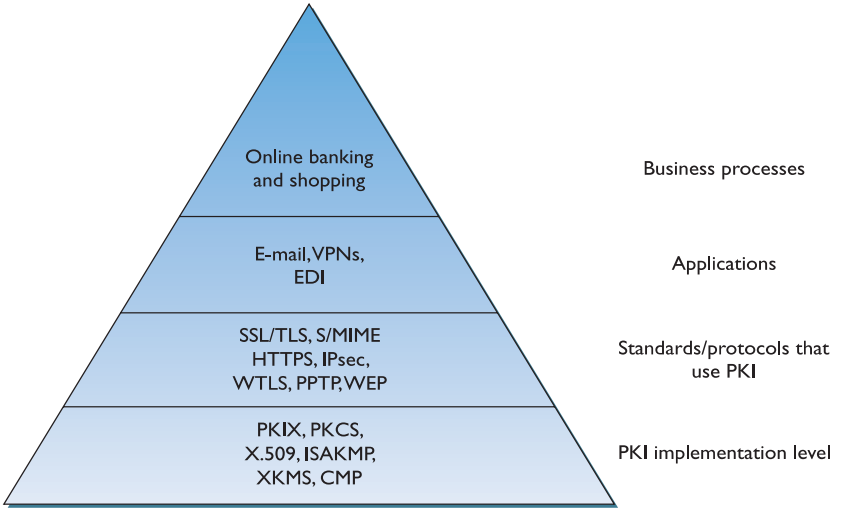
Avantage : Applicable à toutes les applications sur TCP (sans réécriture de celles-ci)

A ce jour, probablement le domaine d'application le plus utilisé (OpenSSL).

Rapports entre la PKI et les protocoles de communication

La PKI repose sur :

- Des services fournis par PKI X.509 (PKIX)
  la certification, le stockage, l'enregistrement, la révocation
- Des formats : PKCS, « Public Key Cryptography Standards »
  pour l'échange des éléments entre les applications : S/MIME, SSL, TLS
- Une organisation des données d'annuaire : X509.
  pour définir les DN, indentifier les serveurs, indiquer les URLs des services, donner les usages



Des protocoles de gestion de clés

- Internet Security Association and Key Management Protocol (ISAKMP)
- XML Key Management Specification (XKMS)

Des protocoles de gestion de certificats

- Certificate Management Pro- tocol (CMP).

Des protocoles de gestion de communication chiffrées :

- Wired Equivalent Privacy (WEP) pour chiffrer des communications sans fil 802.11 ;
- IP Security (IPsec) et Point-to-Point Tunneling Protocol (PPTP) pour réaliser des VPNs.

D'autres encore

- Secure/Multipurpose Internet Mail Extensions (S/MIME) pour l'e-mail;
- Secure Sockets Layer (SSL), Transport Layer Security (TLS), etWireless Transport Layer Security (WTLS) pour des communications sécurisées ;
- Hypertext Transfer Protocol Secure (HTTPS) pour les navigateurs Web.

Le modèle PKIX et les RFCs associées

Subject	RFC Number	Title	Notes
Certificates and CRL profiles	RFC 3281	An Internet Attribute Certificate Profile for Authorization	
	RFC 3739	Internet X.509 Public Key Infrastructure: Qualified Certificates Profile	Obsoletes RFC 3039
	RFC 4055	Additional Algorithms and Identifiers for RSA Cryptography for Use in the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile	
	RFC 4476	Attribute Certificate (AC) Policies Extension	
	RFC 5055	Server-based Certificate Validation Protocol (SCVP)	
	RFC 5280	Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile	Obsoletes RFC 4630, RFC 4325, RFC 3280
Certificate management protocols	RFC 2560	X.509 Internet Public Key Infrastructure Online Certificate Status Protocol – OCSP	
	RFC 3709	Internet X.509 Public Key Infrastructure: Logotypes in X.509 Certificates	
	RFC 3820	Internet X.509 Public Key Infrastructure Proxy Certificate Profile	
	RFC 4043	Internet X.509 Public Key Infrastructure Permanent Identifier	
	RFC 4059	Internet X.509 Public Key Infrastructure Warranty Certificate Extension	
	RFC 4158	Internet X.509 Public Key Infrastructure: Certification Path Building	
	RFC 4210	Internet X.509 Public Key Infrastructure Certificate Management Protocols	Obsoletes RFC 2510
	RFC 4211	Internet X.509 Public Key Infrastructure Certificate Request Message Format (CRMF)	Obsoletes RFC 2511
	RFC 4386	Internet X.509 Public Key Infrastructure Repository Locator Service	
	RFC 4387	Internet X.509 Public Key Infrastructure Operational Protocols: Certificate Store Access via HTTP	
	RFC 4683	Internet X.509 Public Key Infrastructure Subject Identification Method (SIM)	
	RFC 4985	Internet X.509 Public Key Infrastructure Subject Alternative Name for Expression of Service Name	
	RFC 5272	Certificate Management Messages over CMS	Obsoletes RFC 2797
	RFC 5273	Certificate Management over CMS (CMC): Transport Protocols	
	RFC 5274	Certificate Management Messages over CMS (CMC): Compliance Requirements	
Certificate policies and CPSs	RFC 3647	Internet X.509 Public Key Infrastructure Certificate Policy and Certification Practices Framework	Obsoletes RFC 2527

(continued)

Le modèle PKIX et les RFC associées

Subject	RFC Number	Title	Notes
Operational protocols	RFC 2528	Internet X.509 Public Key Infrastructure Representation of Key Exchange Algorithm (KEA) Keys in Internet X.509 Public Key Infrastructure Certificates	
	RFC 2585	Internet X.509 Public Key Infrastructure Operational Protocols: FTP and HTTP	
	RFC 3779	X.509 Extensions for IP Addresses and AS Identifiers	
	RFC 4334	Certificate Extensions and Attributes Supporting Authentication in Point-to-Point Protocol (PPP) and Wireless Local Area Networks (WLAN)	Obsoletes RFC 3770
	RFC 5019	The Lightweight Online Certificate Status Protocol (OCSP) Profile for High-Volume Environments	
Time-stamp and data validation	RFC 2875	Diffie-Hellman Proof-of-Possession Algorithms	
	RFC 3029	Internet X.509 Public Key Infrastructure Data Validation and Certification Server Protocols	
	RFC 3161	Internet X.509 Public Key Infrastructure Time Stamp Protocols (TSP)	
	RFC 3628	Policy Requirements for Time-Stamping Authorities	
Other PKIX topics	RFC 3279	Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and CRI Profile	Updated by RFC 4491
	RFC 3379	Delegated Path Validation and Delegated Path Discovery Protocol Requirements	
	RFC 3874	A 224-bit One-way Hash Function: SHA-224	
	RFC 4491	Using the GOST R 34.10-94, GOST R 34.10-2001 and GOST R 34.11-94 Algorithms with the Internet X.509 Public Key Infrastructure Certificate and CRL Profile	Updates RFC 3279

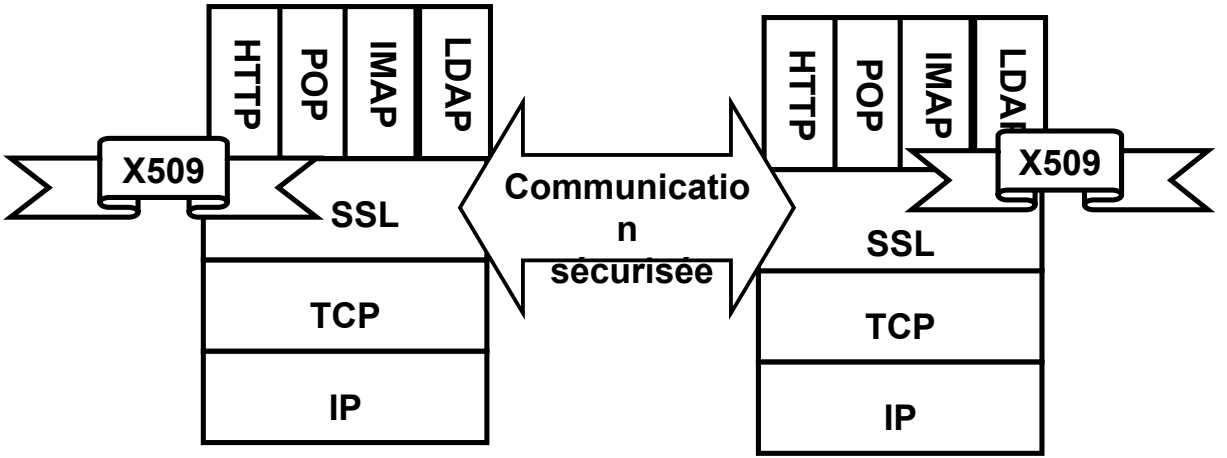
Les formats PKCS

Standard	Title and Description
PKCS #1	RSA Cryptography Standard: Definition of the RSA encryption standard.
PKCS #2	No longer active; it covered RSA encryption of message digests and was incorporated into PKCS #1.
PKCS #3	Diffie-Hellman Key Agreement Standard: Definition of the Diffie-Hellman key-agreement protocol.
PKCS #4	No longer active; it covered RSA key syntax and was incorporated into PKCS #1.
PKCS #5	Password-Based Cryptography Standard: Definition of a password-based encryption (PBE) method for generating a secret key.
PKCS #6	Extended-Certificate Syntax Standard: Definition of an extended certificate syntax that is made obsolete by X.509 v3.
PKCS #7	Cryptographic Message Syntax Standard: Definition of the cryptographic message standard for encoded messages, regardless of encryption algorithm. Commonly replaced with PKIX Cryptographic Message Syntax.
PKCS #8	Private-Key Information Syntax Standard: Definition of a private key information format, used to store private key information.
PKCS #9	Selected Attribute Types: Definition of attribute types used in other PKCS standards.
PKCS #10	Certification Request Syntax Standard: Definition of a syntax for certification requests.
PKCS #11	Cryptographic Token Interface Standard: Definition of a technology-independent programming interface for cryptographic devices (such as smart cards).
PKCS #12	Personal Information Exchange Syntax Standard: Definition of a format for storage and transport of user privates keys, certificates, and other personal information.
PKCS #13	Elliptic Curve Cryptography Standard: Description of methods for encrypting and signing messages using elliptic curve cryptography.
PKCS #14	Pseudo-random Number Generation: A standard for pseudo-random number generation.
PKCS #15	Cryptographic Token Information Format Standard: Definition of a format for storing cryptographic information in cryptographic tokens.

Principes généraux de SSL

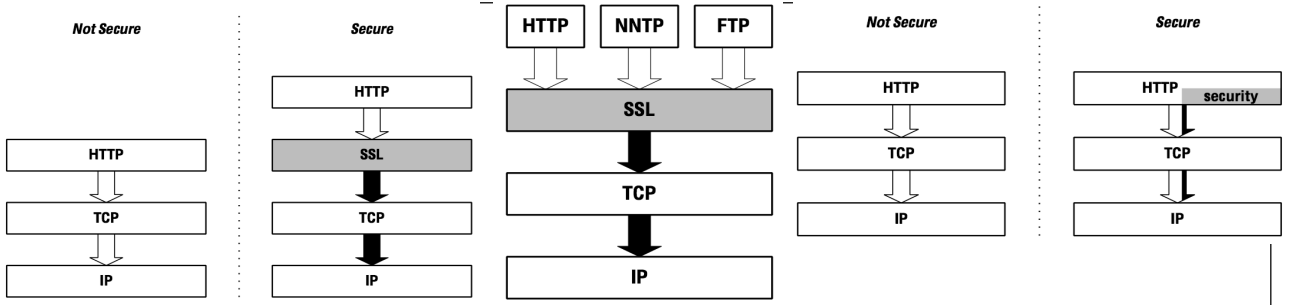
L'utilisation de SSL permet :

- D'établir un canal de communication chiffré ;
- D'authentifier l'un ou l'autre des interlocuteurs, voire les deux, à l'aide de certificat (au format x509).



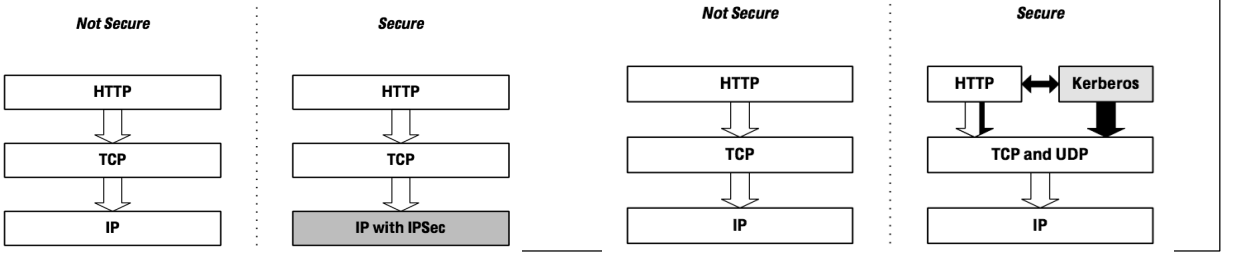
P-F. Bonnefoi

Le positionnement de SSL dans la sécurité réseau



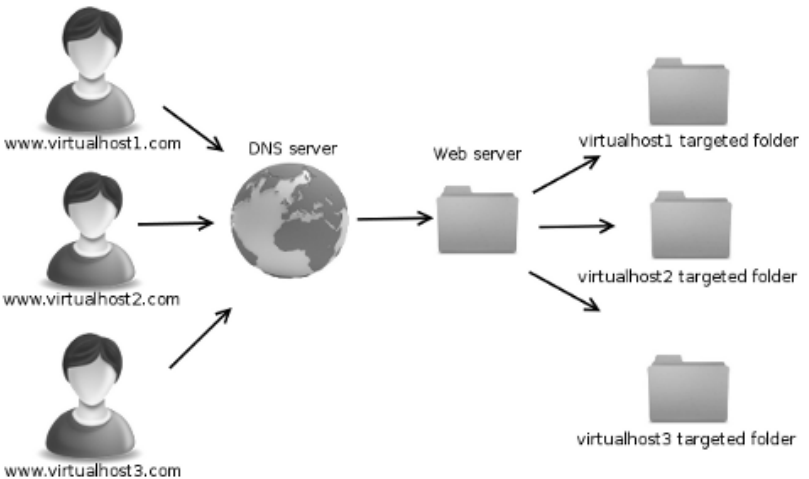
Protocol Architecture	Example	A	B	C	D	E
Separate Protocol Layer	SSL	●	●	○	○	●
Application Layer	S-HTTP	●	○	●	○	●
Integrated with Core	IPSEC	●	●	○	●	○
Parallel Protocol	Kerberos	○	●	○	○	●

Benefits: A – Full Security B – Multiple Applications C – Tailored Services
D – Transparent to Application E – Easy to Deploy



SSL et Virtual Host sous serveur Web Apache : l'extension SNI

Le Virtual Host avec Apache



Le navigateur se connecte au port 80 du serveur Web en HTTP et envoie sa requête :
GET http://www.virtualhost1.com/
host: www.virtualhost1.com

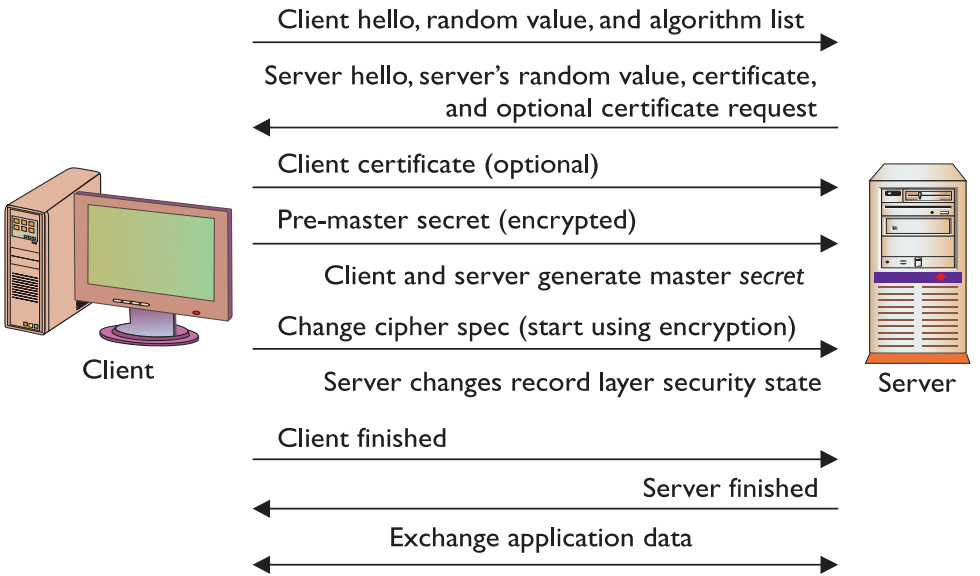
Mais en HTTPS, est-ce que cela marche ?

Le SNI, « Server Name Indication »

Établissement de la connexion SSL

Différentes étapes

- Négociation des algorithmes de crypto à utiliser ;
- Authentification du serveur (celle du client peut être demandée);
- Mise en place des éléments de sécurité (choix algo+clés) ;
- Passage en mode de communication chiffrée.



Déroulement de la communication

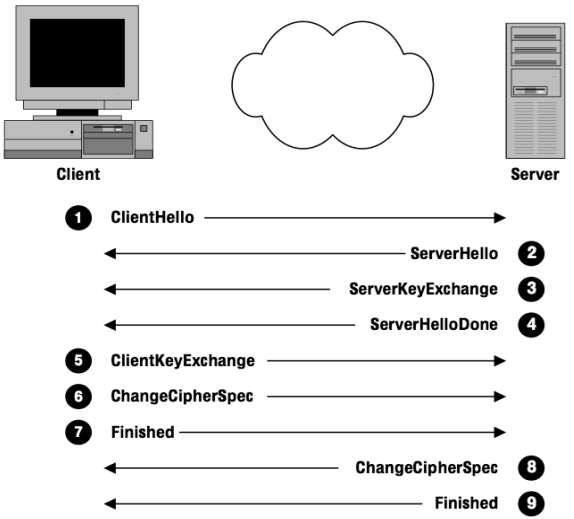
Différents types de messages

- SSL correspond à une connexion TCP dans laquelle circule des messages.
- Il y a négociation des moyens crypto ;
- Il y a authentification des interlocuteurs ;
- Il est possible de :
  - Conserver les paramètres d'une session pour une nouvelle connexion sécurisée (c-à-d. une nouvelle connexion TCP) ;
  - Renégocier les paramètres pour les faire évoluer : « rekeying » afin d'améliorer la sécurité.

Message	Description
Alert	Informs the other party of a possible security breach or communication failure.
ApplicationData	Actual information that the two parties exchange, which is encrypted, authenticated, and/or verified by SSL.
Certificate	A message that carries the sender's public key certificate.
CertificateRequest	A request by the server that the client provide its public key certificate.
CertificateVerify	A message from the client that verifies that it knows the private key corresponding to its public key certificate.
ChangeCipherSpec	An indication to begin using agreed-upon security services (such as encryption).
ClientHello	A message from the client indicating the security services it desires and is capable of supporting.
ClientKeyExchange	A message from the client carrying cryptographic keys for the communications.
Finished	An indication that all initial negotiations are complete and secure communications have been established.
HelloRequest	A request by the server that the client start (or restart) the SSL negotiation process.
ServerHello	A message from the server indicating the security services that will be used for the communications.
ServerHelloDone	An indication from the server that it has completed all its requests of the client for establishing communications.
ServerKeyExchange	A message from the server carrying cryptographic keys for the communications.

Le protocole SSL

Détails de l'établissement de la communication SSL



Step	Action
1	Client sends ClientHello message proposing SSL options.
2	Server responds with ServerHello message selecting the SSL options.
3	Server sends its public key information in ServerKeyExchange message.
4	Server concludes its part of the negotiation with ServerHello-Done message.
5	Client sends session key information (encrypted with server's public key) in ClientKeyExchange message.
6	Client sends ChangeCipherSpec message to activate the negotiated options for all future messages it will send.
7	Client sends Finished message to let the server check the newly activated options.
8	Server sends ChangeCipherSpec message to activate the negotiated options for all future messages it will send.
9	Server sends Finished message to let the client check the newly activated options.

P-F. Bonnefoi

Le protocole SSL

Le message ClientHello

Field	Use
Version	Identifies the highest version of the SSL protocol that the client can support.
RandomNumber	A 32-byte random number used to seed the cryptographic calculations.
SessionID	Identifies a specific SSL session.
CipherSuites	A list of cryptographic parameters that the client can support.
CompressionMethods	Identifies data compression methods that the client can support.

- Le numéro de version : SSLv3/TLSv1
  Il y a rétro-compatibilité :
  la connaissance de la v3 implique celle de la v2.
  Si le client envoi un V3 est le serveur est en v2, alors le client peut décider de continuer **ou non** en v2.
  Attention : il peut y avoir des failles en employant de vieilles versions.

- Une valeur aléatoire servant à initialiser les différents protocoles cryptographiques
  Pour rendre cette valeur différente à chaque connexion on y met la date et l'heure (4 octets sur les 32 octets). *On évite ainsi les « rejeu » ou « replay ».*
  Les 28 octets suivants sont choisis de manière aléatoire robuste (PRNG).

- Un identifiant de session, pour éventuellement reprendre une session déjà mise en place lors d'une ancienne connexion TCP (il peut être vide).

- Les « CipherSuites » donnent la liste des algorithmes supportés par le client.

- Les « CompressionMethods » : permet de faire de la compression **avant le chiffrement** avec un algorithme proposé.
  *Attention : la compression de données chiffrées est impossible car son entropie **doit être** maximale ! Ou alors il y a une faille dans l'implémentation…*

Le protocole SSL

Le message ServerHello

Il est similaire au ClientHello, avec un SessionID qui est toujours différent de vide.

C'est à ce moment qu'est décidé :

- l'usage de la version de SSL
- le choix des algorithmes cryptographiques et des tailles de clé pour la session.

Le message ServerKeyExchange ou message Certificate

Ce message contient la clé **asymétrique** publique du Serveur (attention, le contenu de ce message n'est pas chiffré).

Le format du message dépend de l'algorithme choisi (pour RSA, le message contient l'exposant public et le module de la clé publique du serveur).

Il **peut aussi transmettre un certificat** pour assurer son authentification en mode PKI.

Dans le cas où la clé publique ne peut servir au chiffrement, il faut ajouter un ServerKeyExchange, contenant une clé asymétrique publique signée par le certificat.

Le message ClientKeyExchange

Ce message transmets les informations pour les clés de chiffrement **symétrique** qui sera utilisée pour chiffrer les échanges : « Master-Key ».

Cette « Master-Key » est chiffrée à l'aide de la clé publique du serveur (elle est échangée de manière sécurisée).

Cela permet de finaliser l'authentification du serveur (sans la clé privée associée, il ne pourrait pas déchiffrer cette clé de session).

Le message ChangeCipherSpec

Ce message permet de basculer la communication en mode chiffré.

Il peut y avoir une clé différente pour chaque sens de la communication (il pourrait y avoir également un choix différents pour l'algorithme de chiffrement, mais cela n'a pas été implémenté).

Ainsi, chaque interlocuteur dispose de deux états, « states » : « read state » pour ce qu'il reçoit et « write state » pour ce qu'il envoie.

Le protocole SSL

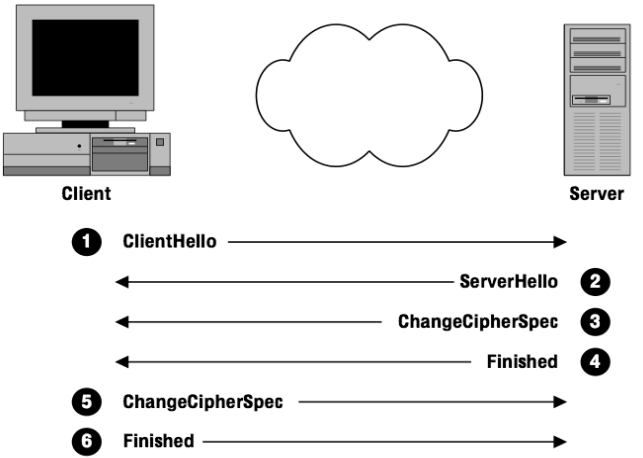
Réutiliser une session précédente

Dans ce cas le protocole est simplifié. Le serveur peut décider de ne pas réutiliser cette session (problème de cache), et de forcer la renégociation (envoi d'un nouveau sessionID au client).

Attention à la sécurité : plus la session est prolongée plus elle peut être attaquée.

Par contre c'est très intéressant dans le cas du Web où différentes requêtes arrivent très rapidement.

Step	Action
1	Client sends ClientHello message specifying a previously established SessionID.
2	Server responds with ServerHello message agreeing to this SessionID.



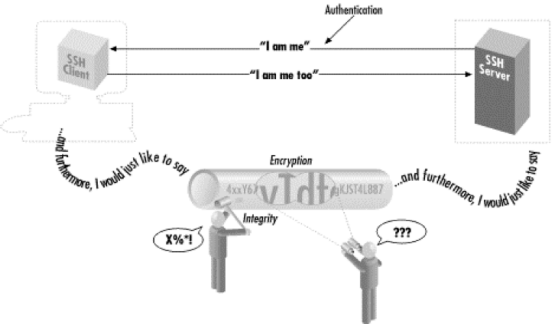
Step	Action
3	Server sends ChangeCipherSpec message to reactivate the session's security options for messages it will send.
4	Server sends Finished message to let the client check the newly reactivated options.
5	Client sends ChangeCipherSpec message to reactivate the negotiated options for all future messages it will send.
6	Client sends Finished message to let the server check the newly reactivated options.

Une utilisation « très utile » de SSL : SSH, « Secure Shell »

SSH

SSH vs SSL

- SSH est implémenté par dessus SSL : communications entièrement chiffrées et sécurisées ;
- Les échanges sont « tout de suite » chiffrés avec Diffie-Helmann (DH), contrairement à SSL ;



- SSH utilise une authentification par « login/mdp » (défaut) ou par clés asymétriques (DSA, RSA) :

  On crée une biclé, clé publique/clé privée et on installe la clé privée, id_rsa, sur la machine maître :

  ```
  ssh-keygen –t rsa –f $HOME/.ssh/id_rsa crée id_rsa
  ```

  On copie la clé publique, id_rsa.pub, dans un fichier particulier sur la machine sur laquelle on veut se connecter automatiquement :

  ```
  $HOME/.ssh/authorized_keys ou $HOME/.ssh/authorized_keys2
  ```

- SSL permet l'utilisation de PKIX, SSH repose sur l'utilisation directe de clés asymétriques
  des patchs existent pour permettre l'utilisation de certificat dans SSH…

SSH

Les capacités de SSH

- Utilisation de SSH pour exécuter des commandes à distance au travers d'une connexion sécurisée, avec un shell :
  `ssh toto@mon_adresse_machine`
  en demandant l'exécution directe d'une commande sur la machine distante :
  `ssh toto@mon_adresse_machine ma_commande_a_executer`

- Échange de fichiers de manière sécurisée « en direct » :
  `scp mon_fichier toto@mon_adresse_machine:mon_fichier`

  `scp –r toto@mon_dresse_machine:mon_repertoire mon_chemin`

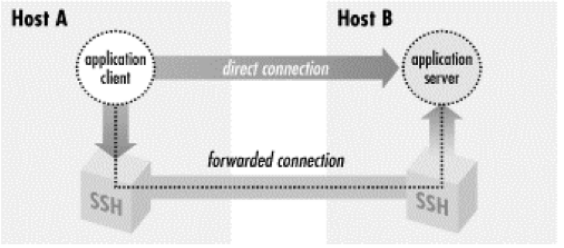
- Échange de fichiers de manière sécurisée « à la ftp » :
  `sftp toto@mon_adresse_machine`
  `sftp>`

- Redirections de connexion TCP *(on désignera « relais », la machine à laquelle on se connecte en SSH).*
  - Locale vers distante, *c-à-d. se connecter à une machine distante depuis la machine relais* ;
  - Distante vers locale, *c-à-d. se connecter à la machine cliente par la machine relais ;*
  - Dynamique (serveur socks)*, c-à-d. créer « à la volée » des connexions depuis la machine relais ;*
  - De session X, *c-à-d. récupérer les fenêtres X11 du serveur sur le client*.

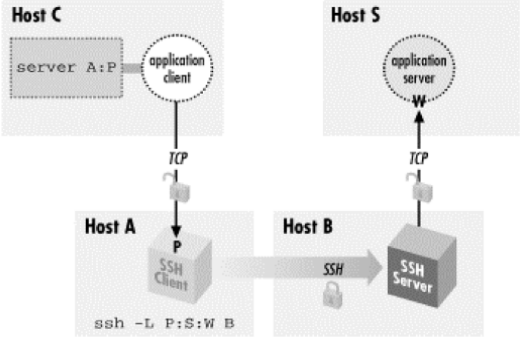
- Création de VPN :
  - De niveau 2 (trame) ;
  - De niveau 3 (ip).

SSH : redirection ou « forwarding », locale

Utilisation de SSH comme d'un tunnel



L'application client sur la machine A ne fait pas une connexion directe vers l'application serveur sur la machine B, mais passe par le « tunnel » chiffré offert par la connexion SSH :

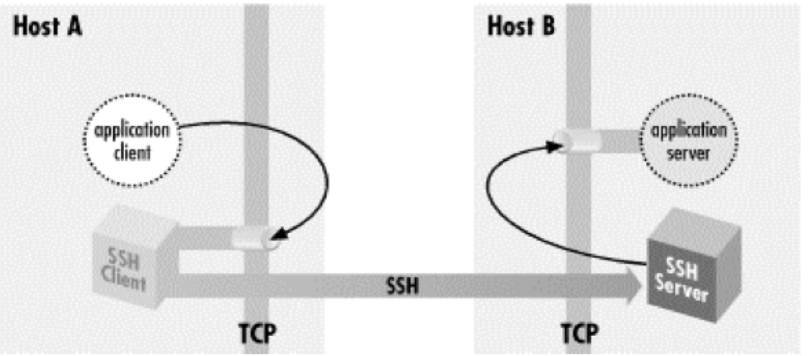


`ssh –L <port_local_P>:<adresse_S>:<port_cible_W> toot@adresse_B`

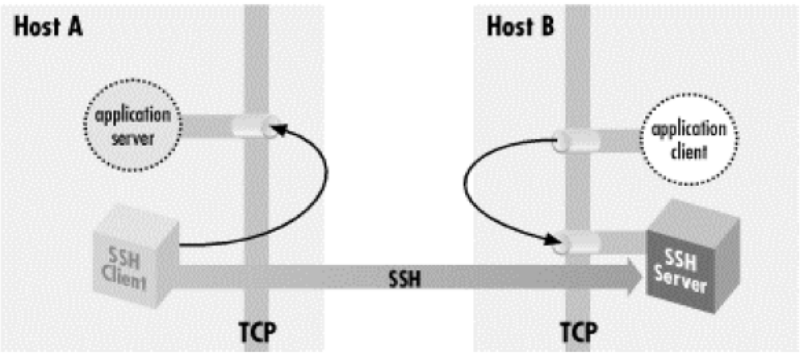
`ssh –g –L` … *l'option –g, gateway, permet à une machine C d'emprunter le tunnel*

SSH : redirection ou « forwarding », distante

Locale :



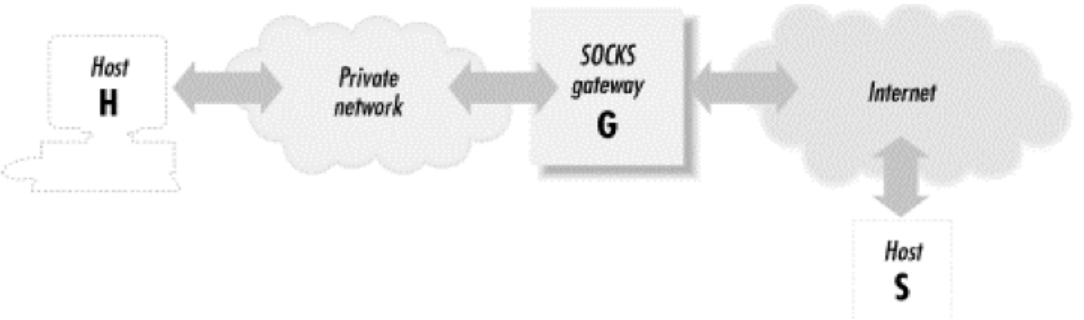
Distante :




```
ssh –R <port_distant>:<adr_cible_locale>:<port_cible_locale> machine_relais
```

SSH : redirection ou « forwarding », dynamique

Création d'un proxy « SOCKS »




```
ssh –ND <port> toto@ma_machine
```

On peut se servir de ce serveur socks dans son navigateur Web :

Ici, on lance :
ssh –ND 1080 toto@machine
Et on configure l'accès au serveur SOCKS sur le port 1080

Ainsi chaque nouvelle connexion du navigateur est recréée et tunnelisée par l'intermédiaire de la connexion SSH.