Cryptographie

Aléa, pseudo-aléa et chiffrement à flots

Carlos Aguilar

carlos.aguilar@enseeiht.fr

IRIT-IRT

Références

Livres électroniques

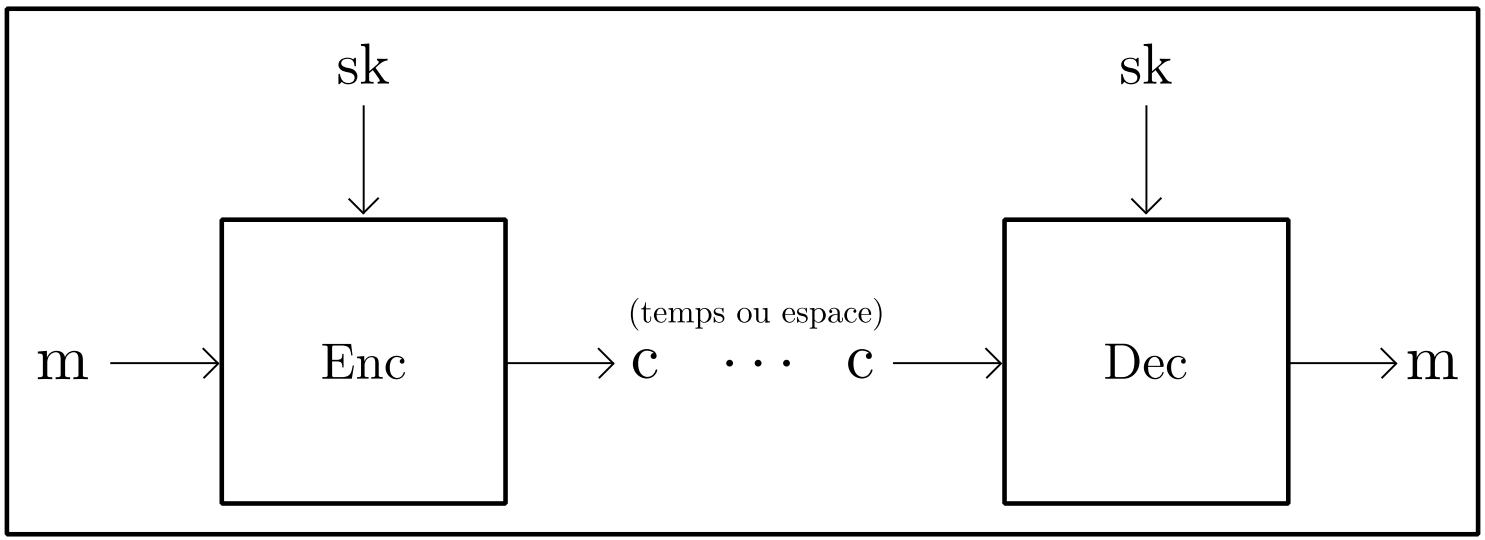
Cornell, https://www.cs.cornell.edu/courses/cs4830/2010fa/lecnotes.pdf
Bristol, http://www.cs.umd.edu/~waa/414-F11/IntroToCrypto.pdf
Stanford, http://crypto.stanford.edu/~dabo/cryptobook/

Cours fortement inspiré de :

[1] Stanford, https://www.coursera.org/course/crypto
[2] Univ. of Virginia, http://www.udacity.com/view#Course/cs387/

Plan

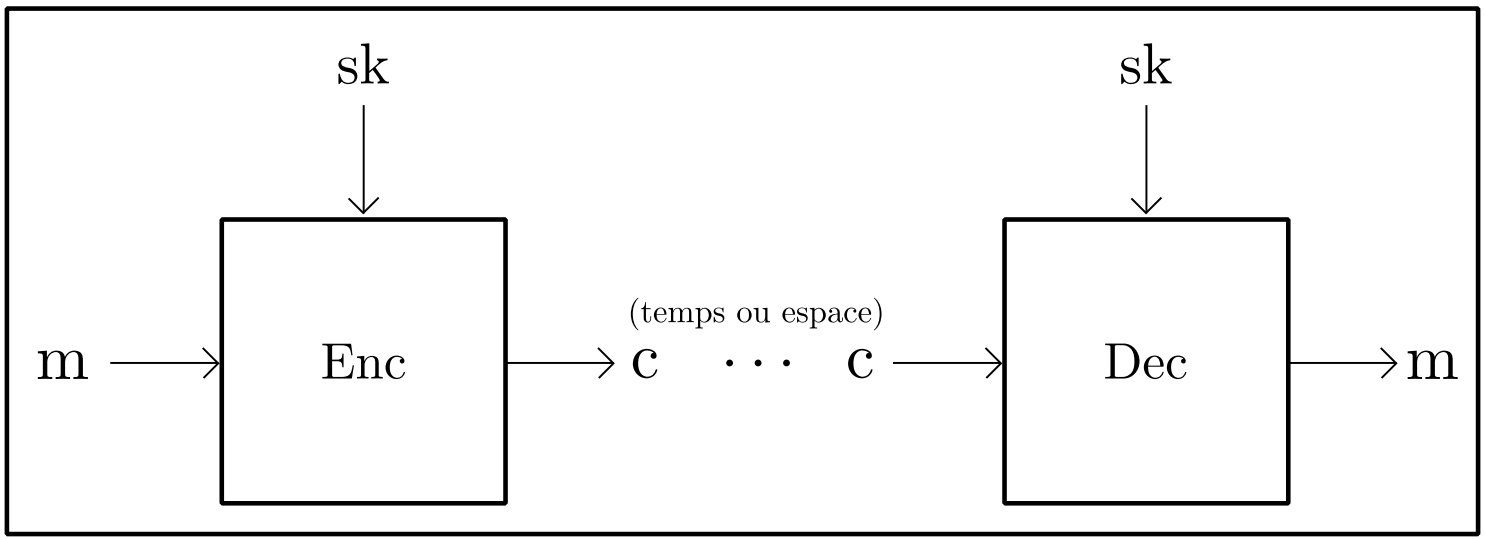
Les algorithmes de chiffrement symétrique



Hypothèses

Les clés sont aléatoires
Les algorithmes sont sûrs

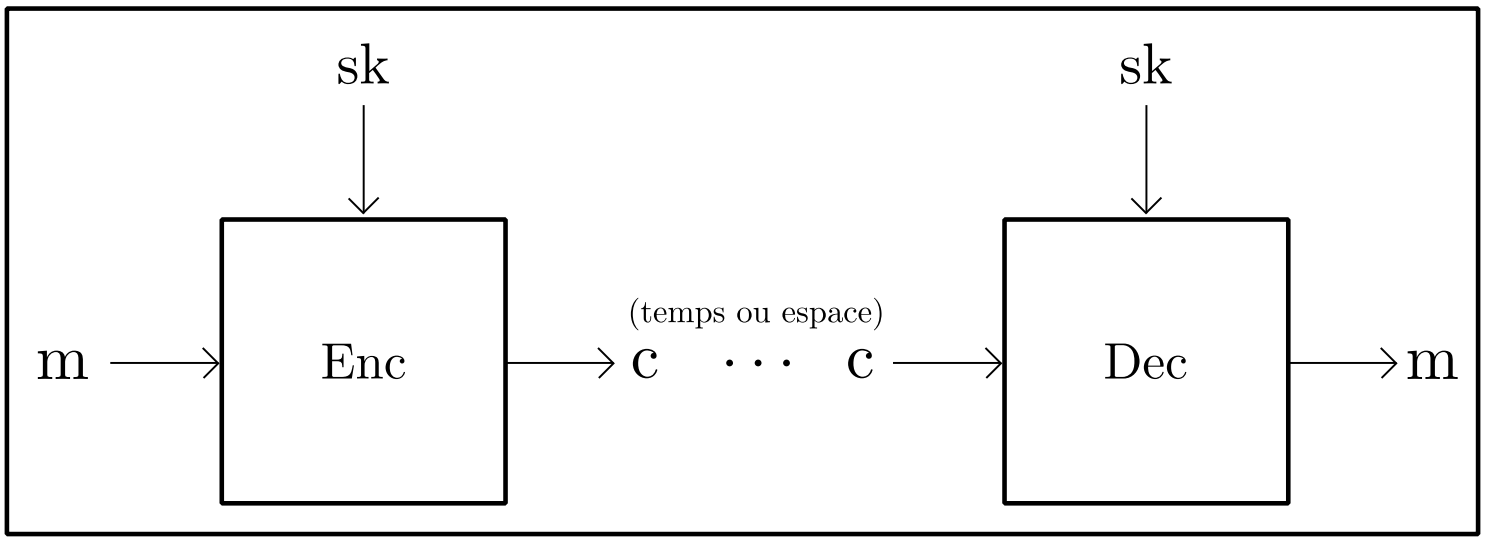
Les algorithmes de chiffrement symétrique



Hypothèses → Questions légitimes

Les clés sont aléatoires
Les algorithmes sont sûrs

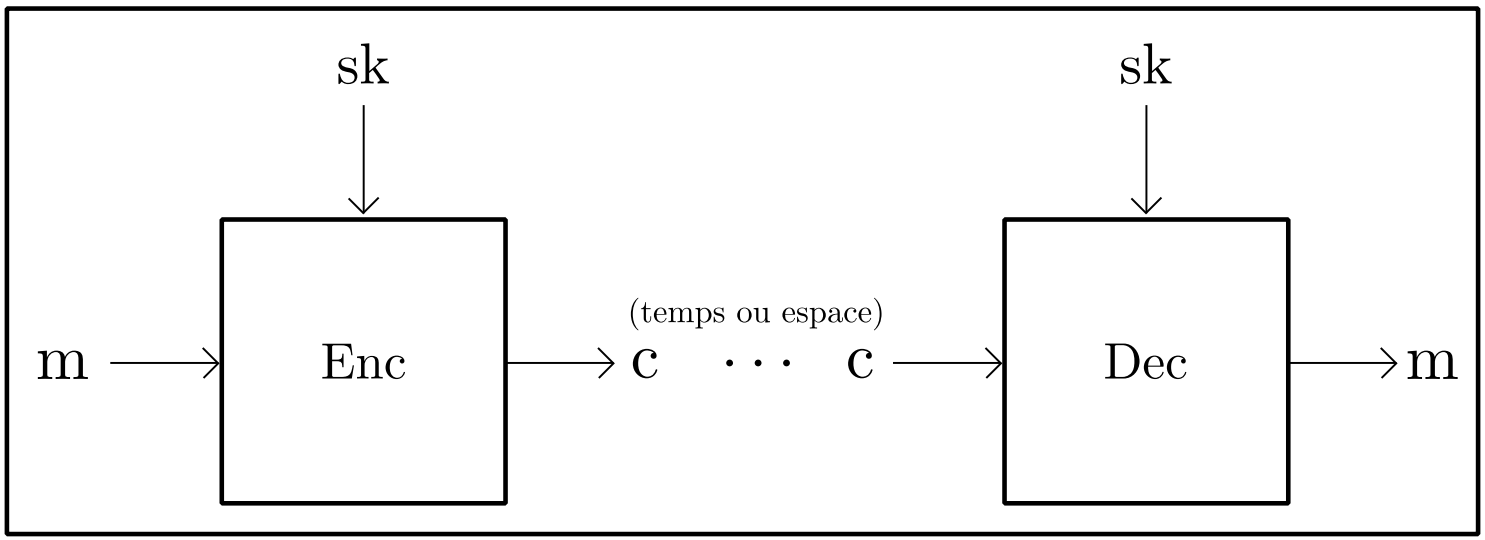
Les algorithmes de chiffrement symétrique



Hypothèses → Questions légitimes

Les clés sont aléatoires → Comment générer ces clés ?
Les algorithmes sont sûrs → Comment faire des algorithmes sûrs ?

Les algorithmes de chiffrement symétrique



Hypothèses → Questions légitimes

Les clés sont aléatoires → Comment générer ces clés ?
Les algorithmes sont sûrs → Comment faire des algorithmes sûrs ?
Les clés sont partagées → Comment les partager sans les dévoiler ?

Aléa ou pseudo-aléa

Générateur aléatoire (pur)

- la distribution de sortie suit parfaitement la distribution cible (hypothèse)

$\Rightarrow$ on ne peut reproduire son état ($\Rightarrow$ on ne peut pas la cloner)

$\Rightarrow$ aucun test (e.g. statistique) ne permet de les distinguer

Exemple pour la distribution uniforme binaire : chaque bit a exactement une probabilité 1/2 d'être à 1 et est indépendant des autres (modèle théorique)

Générateur pseudo-aléatoire (PRNG)

- la distribution de sortie est proche de la distribution cible
- prend en entrée une graine permettant de reproduire son état

$\Rightarrow$ des tests permettent de le distinguer d'une source d'aléa

Cryptographiquement sûr :
Calculatoirement indistingable d'un générateur d'aléa pur

Plan

Comment générer de l'aléa pur ?

Moyens physiques

Phénomènes à composante quantique

- émission photoélectrique
- radiations ...

Bruits physiques à composante chaotique

- bruit thermique
- bruit atmosphérique

Sources disponibles

- Processeurs (Intel, bruit thermique)
- Générateurs hardware d'aléa (carte PCI)
- Services web : `http://www.fourmilab.ch/hotbits/` (émissions radioactives), `https://www.random.org/` (bruit athmosphérique), etc.

Exemple : les systèmes de type Unix

Linux

`/dev/random`

- mélange plusieurs sources et cumule les bits
- évalue l'entropie (`/proc/sys/kernel/random/entropy_avail`)
- en lecture, passe les bits cumulés (pool) par une fonction de hachage
- quand le pool est vide il bloque

$\Rightarrow$ génération de clés très long terme

Alternative : les CSPRNG non déterministes

Principe de base

CSPRNG utilisant le pool d'aléa pour régénérer dès que possible la graine

Par système

- /dev/urandom dans les systèmes Linux : CSPRNG basé sur une fonction de hachage
- /dev/random dans la famille BSD, OS X : Yarrow puis Fortuna ; dans ces systèmes /dev/random == /dev/urandom
- CryptGenRandom (Windows) : même principe que les autres mais implémentation non dévoilée (attaqué en 2007 puis réparée)

Pourquoi ils sont considérés sûrs

Beaucoup de gens ont essayé de les casser sans succès
Ont été analysés et standardisés contrairement aux générateurs physiques

Plan

Chiffrement à flots (1/2)

Rappel

PRNG, fonction (déterministe) $\{0,1\}^s \to \{0,1\}^n$ avec $n \gg s$
CSPRNG : pour un k aléatoire indistingable d'une source d'aléa pur

Idée : reprendre l'algo de l'OTP

message $\oplus$ masque = chiffré chiffré $\oplus$ masque = message

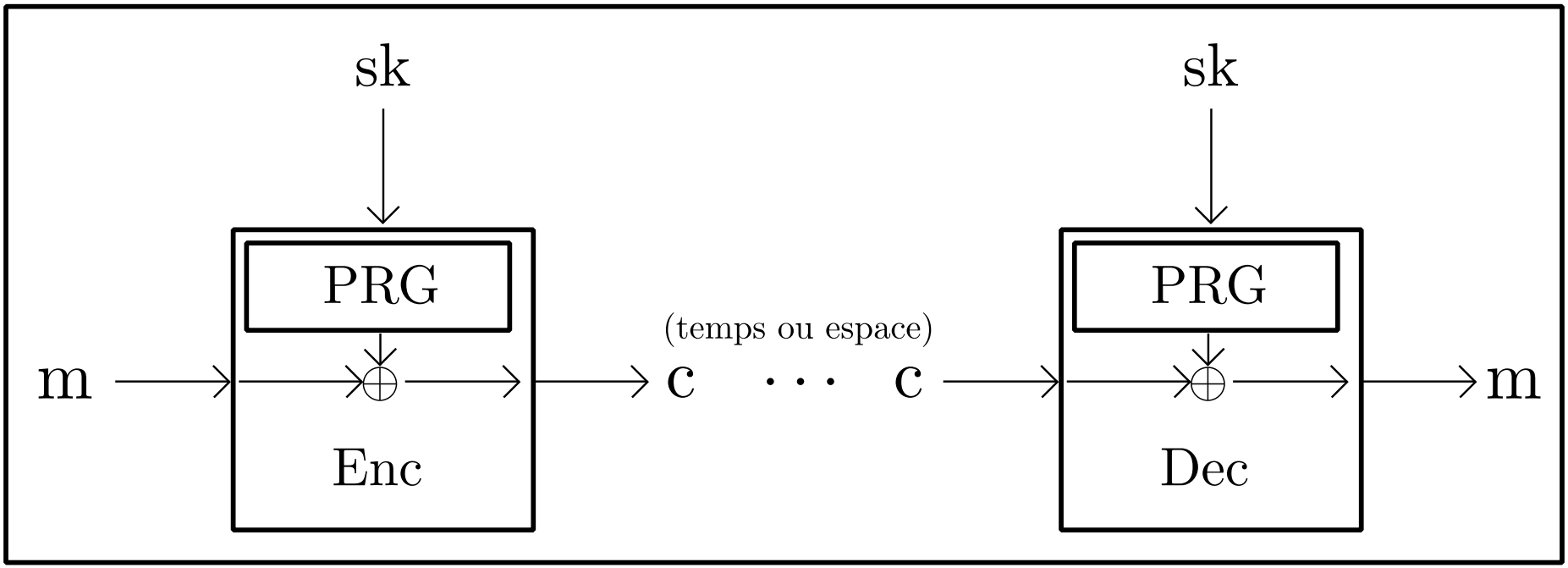
... en remplaçant le masque aléatoire par un masque pseudoaléatoire
Le masque pseudoaléatoire est bien sûr généré par un CSPRNG

Utilisation

Très utilisé en télécommunications :

- le flot généré bit par bit permet de chiffrer/déchiffrer une communication au fur et à mesure simplement
- le pseudo-aléa peut être pre-généré pour éviter d'introduire des délais

Chiffrement à flots (2/2)



Il faut faire attention à la synchro !

Bien évidemment ...

Il ne faut pas utiliser deux fois la même clé dans le OTP

Même problème pour les chiffrements à flots
Première version de PPTP (Windows NT) :

- Messages client-serveur : c_1, c_2, c_3 masqués par $PRG(k)$
- Messages serveur-client : s_1, s_2, s_3 masqués par $PRG(k)$

$\Rightarrow$ on peut obtenir $(c_1 \| c_2 \| c_3) \oplus (s_1 \| s_2 \| s_3)$

Principe des précaution

Pour des usages différents utiliser différentes clés
Même pour la même fonction dans deux sens différents ($C \rightarrow S$ et $S \rightarrow C$)

Réduction de sécurité

Si PRNG cryptographiquement sûr alors chiffrement à flots sûr

Preuve : si attaque sur chiffrement alors on distingue le PRNG d'une source d'aléa pure puisqu'on ne peut pas attaquer l'OTP

Plan

Generateurs de nombres pseudo-aléatoires déterministes

Définition

Algorithme déterministe permettant à partir d'une graine de générer une longue chaîne de nombres avec des bonnes propriétés statistiques

Appelés par le NIST DRBGs (Deterministic Random Bit Generators)

Usages

Algorithmique (e.g. tris randomisés), jeux, etc.
Avantage face à l'aléa pur : la vitesse, le déterminisme
Désavantage face à l'aléa pur : une potentielle prédictibilité

Cryptographie

Seulement les PRNG dits cryptographiquement sûrs (CSPRNG) peuvent être utilisés en cryptographie

La plupart des PRNG ne sont pas cryptographiquement sûrs

PRNGs non-sûrs classiques

Générateur à congruentiel linéaire

$X_{n+1} = (aX_n + c) \mod m$ avec typiquement $m = 2^{32}$, a, c, définis dans le code, et X_0 la graine initialisant le générateur

C'est la fonction Rand standard en C/C++/Java/Delphi/Pascal

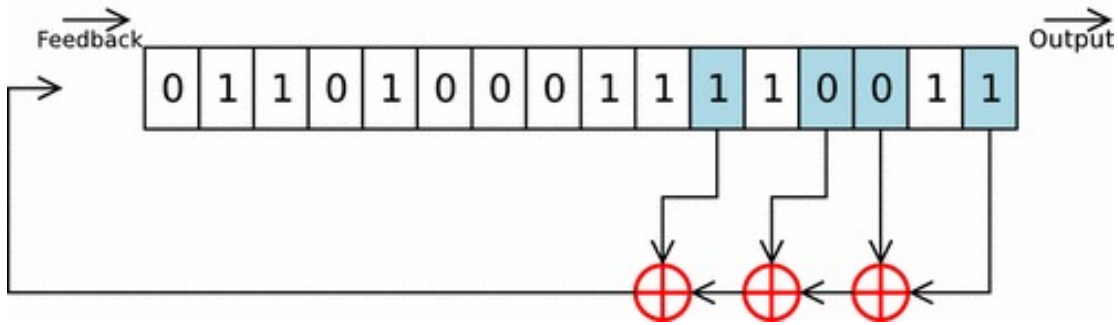
Mersenne twister

Beaucoup plus compliqué à expliquer, meilleures propriétés statistiques

Par défaut dans R, Python, Ruby, PHP2, Sage, Matlab, Maple, Scilab, GMP, Visual C++, présent dans C++11, boost, etc.

Les registres à décalage (LFSR)

Principe



Source : `https://fr.wikipedia.org/wiki/Registre_a_decalage_a_retroaction_lineaire`

Rapide mais dangereux

Extrêmement rapide, très peu coûteux en hardware
Doit être modifié (plusieurs LFSRs + fonction non-linéaire) pour être sûr
Utilisé dans Content Scrambling System (CSS pour les DVDs) A5/1 (GSM) et E0 (Bluetooth) ... tous sont cassés

Comment vérifier l'aléa ou le pseudo-aléa ?

Tests statistiques

NIST SP800-22 Statistical Test Suite

`http://csrc.nist.gov/groups/ST/toolkit/rng/documents/sts-2.1.2.zip`

Diehard tests `http://stat.fsu.edu/pub/diehard/`

Limitations

Utiles pour montrer que quelque chose n'est pas du tout aléatoire
Faciles à contourner surtout quand on les connaît à l'avance
Un générateur MT passe les deux batteries de tests ... mais n'est pas sûr

Algorithmes à connaître

LFSR

CSS, A5/1-2 (GSM/GPRS), E0 (bluetooth), SNOW2/SNOW3G (UMTS/LTE)

RC4

WEP, TKIP, HTTPS

Standards crypto (Projet eSTREAM)

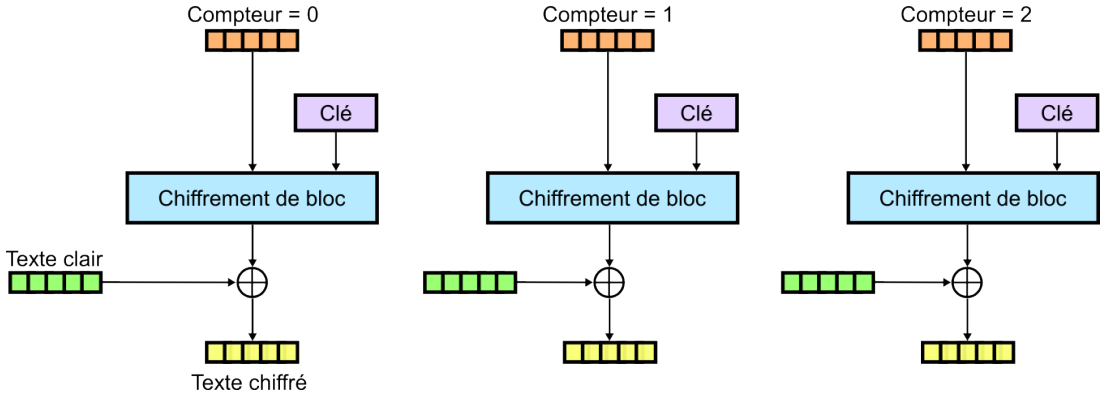
Salsa20, SOSEMANUK (Fr)

Chiffrements par bloc en mode CTR

Transparents suivants
AES-CTR : leader absolu
Remarque : l'ITU a arrêté de jouer avec la crypto et l'utilise pour A5/3 (LTE)

Le mode compteur (CTR)

Principe



Source : `https://fr.wikipedia.org/wiki/Mode_doperation_(cryptographie)`

Propriétés

Si AES est une PRP alors AES-CTR est un CSPRNG $\rightarrow$ chiffrement sûr
Parallélisable au chiffrement/déchiffrement, accès aléatoire

Remarques

En pratique on coupe le compteur en IV et compteur (e.g. 64/64)
Output Feedback Mode (OFB) marche aussi (moins populaire et pratique)

Les générateurs prouvés (1/2)

Blum Blum Shub [BBS 86]

À chaque pas :

- on génère $x_{n+1} = x_n^2 \mod M$
- on donne en sortie la parité (par exemple) de x_{n+1}

pour M un nombre difficile à factoriser (typiquement de plus de 1024 bits)
pour x_0 sans facteurs communs avec M

C'est lent mais peut remplacer avantageusement une source d'aléa

Sécurité

Calculatoirement indistingable de l'aléa si le problème de la résiduosité quadratique est difficile

Les générateurs prouvés (2/2)

Dual_EC_DRBG (Dual Elliptic Curve DRBG)

Partie du standard NIST SP800-90A (Recommendation for Random Number Generation Using Deterministic Random Bit Generators)

Backdoor de la NSA confirmé par de nombreuses sources

Historique

Une tonne d'indices dès sa proposition (c.f. recherche Internet)
Bullrun program : fragilisation de la cryptographie (250M$/an [NY Times])
Point marquant contrat 10M$ RSA $\rightarrow$ leur générateur par défaut

NIST recommande de l'abandonner depuis 2014

Problèmes

Courbe elliptique et point de départ fixés (choisis malicieusement)
Trop de bits générés par tour

Le Standard NIST SP800-90A

Étude d'un standard

Document annexe

Fin

Prochain cours

Distribution de clés