UNIVERSITÉ LAVAL

Cryptographie symétrique : Chiffrement par flot

MOHAMED MEJRI

Groupe LSFM

Département d'Informatique et de Génie Logiciel

Université LAVAL

Québec, Canada

Plan

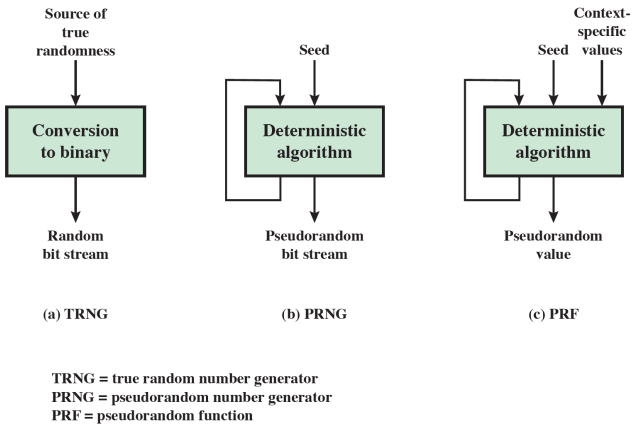
- Variables Aléatoires
- Chiffrement par flots
- La cryptographie par seuil (threshold cryptography)

Variables Aléatoires

- L'utilisation des variables aléatoires est omniprésente : sécurité (clé secrète, clé publique, cookies, etc.), simulation, jeux de hasard, etc.
- Un mauvais générateur peut maitre en cause la sécurité de tout le système.
- Un bon générateur doit satisfaire deux conditions :
  - Uniforme : Les bits ont les mêmes chances d'apparaître
  - Imprévisible : ayant les bits $b_0, \ldots, b_n$, "impossible" de prédire b_{n+1}

Variables Aléatoires

Générateur aléatoire et générateur pseudo-aléatoire



- Vrai générateur aléatoire TRNG .
  - Idéalement, on doit prendre les valeurs d'une vraie source aléatoire (radioactivité, bruit, etc.)
  - Est-ce qu'une vraie valeur aléatoire existe ? c'est une question philosophique
  - Une vraie source aléatoire et couteuse et lente (souvent mécanique)

  Éventuelle conversion de l'analogique vers le numérique. Utilisé pour générer des germes (seed)

Variables Aléatoires

– Générateur de nombres pseudo-aléatoires (PRNG) : génère un flux de bits pseudo-aléatoires à partir d'un germe. Utilisé souvent pour le chiffrement d'un flot. La séquence générée doit être reproduite sur différentes machines (complètement déterministe).
– Fonction pseudo-aléatoire (PRF) : Elle retourne une sortie de longueur fixe. Elle est appelée de temps à autre pour générer une clé, un nonce, etc. Elle peut faire appel à un contexte (mouvement de la souris, horloge, etc.) pour faire en sorte que même si on trouve le germe, il reste d'autres obstacles. Exemple $X_n = (a * X_{n-1} + b)\ mod\ p$
– La sortie d'un PRNG est évaluée via des tests statistiques et sa sécurité via la cryptanalyse
– La preuve de sécurité peut se traduire à montrer que prédire un bit a la même difficulté que résoudre d'autres problèmes connus difficiles (NP), comme la factorisation de grands nombres
– La prédiction doit être difficile vers l'avant (ayant des bits du passé, c'est difficile de prédire un bit du futur avec une probabilité différente de 1/2) et vers l'arrière (ayant des bits, c'est difficile de réduire l'espace de recherche du germe)

Variables Aléatoires

Tests statistiques

- On considère que la sortie d'un PRNG comme aléatoire si elle passe certains tests
- Beaucoup de tests sont disponibles.
- Plus on fait de tests, mieux c'est.
- Un générateur accepté par des tests ne veut pas dire qu'il est bon à coup sûr.
- Un générateur refusé par des tests ne veut pas dire qu'il est mauvais
- Il y a toujours un risque de prendre une mauvaise décision, mais on veut que ce risque soit contrôlé (la probabilité de l'erreur ne dépasse pas une certaine valeur)

Variables Aléatoires

Tests statistiques Le NIST SP 800-22 propose 15 tests dans :

http ://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-22r1a.pdf Il y a trois catégories de tests

- **Uniformité** :
  - Test de fréquence "Monobit Test" : tester si le nombre de 0 et le nombre de 1 sont approximativement les mêmes
  - Recherche d'un motif apériodique : tester si un motif d'une longueur donné se présente avec un très grand nombre d'occurrences
  - Etc.
- **Adaptabilité** (scalability) : si une séquence passe des tests d'uniformité, alors ses sous-séquences doivent aussi passer le test
- **Consistance** : Si on passe les bons tests pour un germe, on doit aussi les réussir pour les autres

Variables Aléatoires

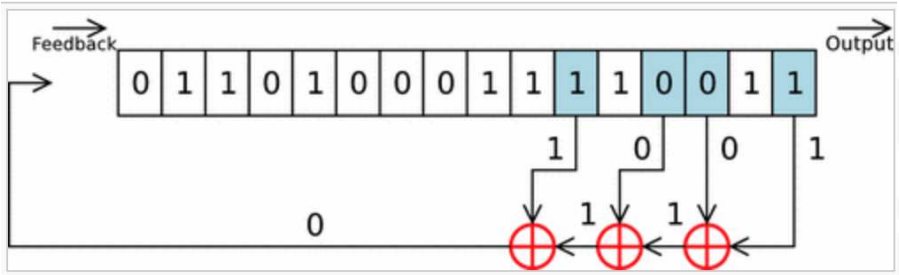
Conception :

- Deux catégories
  - Algorithmes conçus spécialement pour générer des nombres pseudo-aléatoires
  - Algorithmes conçus à partir de systèmes cryptographiques (symétriques, asymétrique, fonction de hachage)

Variables Aléatoires

Conception : Algorithmes conçus spécialement pour générer des nombres pseudo-aléatoires

– LSFR (Lenear Feedback Shift Register)



source : wikipedia.org

– Blum, Blum, Shub (BBS) : Très célèbre
  – calculer $x_0 = seed$, calculer $x_{i+1} = x_i^2 \ mod \ p$
  – à chaque itération on retient le bit le moins significatif de x_i
  – on prouve que prédire un bit avec une probabilité supérieure à 1/2 a la même la difficulté que la factorisation d'un grand nombre.

i	X_i	B_i
0	20749	
1	143135	1
2	177671	1
3	97048	0
4	89992	0
5	174051	1
6	80649	1
7	45663	1
8	69442	0
9	186894	0
10	177046	0

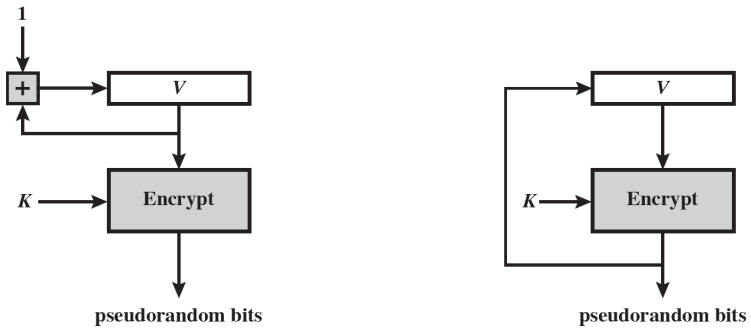
Blum, Blum, Shub (BBS)

Variables Aléatoires

Conception :

Algorithmes conçus à partir de systèmes cryptographiques

- **CTR mode** : recommandé dans NIST SP 800-90, ANSI standard X.82, et RFC 4086
- **OFB mode** : recommandé dans X9.82 et RFC 4086



(a) CTR Mode (b) OFB Mode

- le germe est formé de k et de V. Dans le cas de AES-128, K et V sont sur 128bits chaque
- dans le cas de CTR, V est incrémenté de 1 après chaque opération
- dans le cas de CTR, V est remplacé par la valeur générée

Variables Aléatoires

Conception : Algorithmes conçus à partir de systèmes cryptographiques

OFB

Output Block	Fraction of One Bits	Fraction of Bits that Match with Preceding Block
1786f4c7ff6e291dbdfdd90ec3453176	0.57	—
5e17b22b14677a4d66890f87565eae64	0.51	0.52
fd18284ac82251dfb3aa62c326cd46cc	0.47	0.54
c8e545198a758ef5dd86b41946389bd5	0.50	0.44
fe7bae0e23019542962e2c52d215a2e3	0.47	0.48
14fdf5ec99469598ae0379472803accd	0.49	0.52
6aeca972e5a3ef17bd1a1b775fc8b929	0.57	0.48
f7e97badf359d128f00d9b4ae323db64	0.55	0.45

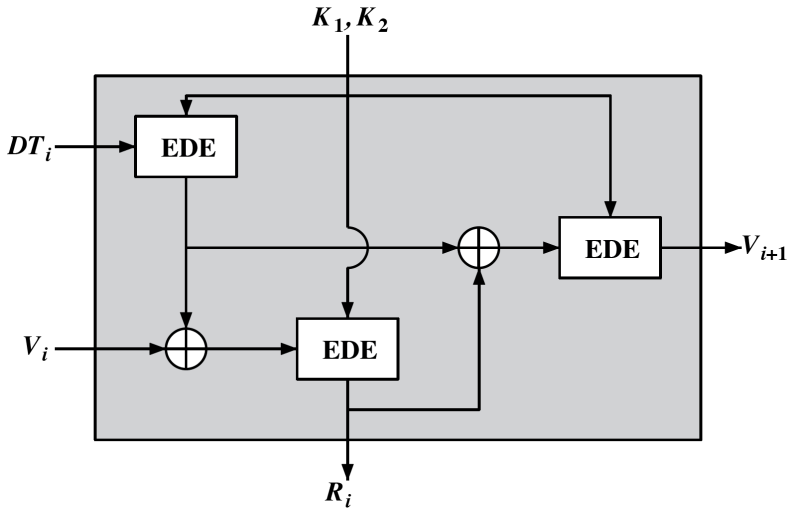
CTR

Output Block	Fraction of One Bits	Fraction of Bits that Match with Preceding Block
1786f4c7ff6e291dbdfdd90ec3453176	0.57	—
60809669a3e092a01b463472fdcae420	0.41	0.41
d4e6e170b46b0573eedf88ee39bff33d	0.59	0.45
5f8fcfc5deca18ea246785d7fadc76f8	0.59	0.52
90e63ed27bb07868c753545bdd57ee28	0.53	0.52
0125856fdf4a17f747c7833695c52235	0.50	0.47
f4be2d179b0f2548fd748c8fc7c81990	0.51	0.48
1151fc48f90eebac658a3911515c3c66	0.47	0.45

Variables Aléatoires

Conception : Algorithmes conçus à partir de systèmes cryptographiques

- ANSI X9.17 : utilisé par des applications financières, dans PGP, etc.
  - parmi les meilleures générateurs pseudo-aléatoires
  - utilise 3DES (EDE : $E_{K_1} \circ D_{K_2} \circ E_{K_1}$) avec deux clés K_1 et K_2
  - prend aussi DT_i (Date and Time) et un vecteur V_i qui changent à chaque itération
  - retourne R_i

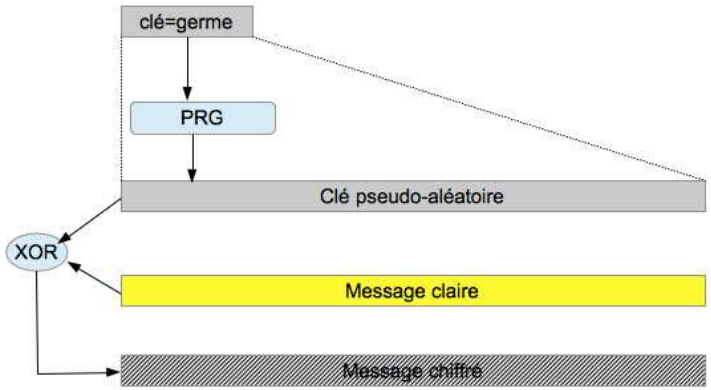


Plan

- Variables Aléatoires
- **Chiffrement par flots**
- La cryptographie par seuil (threshold cryptography)

OTP-PRNG

- Le OTP est un système parfait contre les attaques à textes chiffrés
- En plus, il est très rapide : une seule opération XOR
- Cependant, pour atteindre la perfection, il faut que la clé soit de même taille que le message et elle ne peut être utilisée qu'une seule fois
- Est-il possible de le rendre pratique tout en le gardant sécuritaire ?
- Pour le rendre pratique, on va trouver un moyen de générer des clés de grandes tailles à partir d'une petite clé : En utilisant les PRNG (Pseudo Randon Number Generator)



OTP-PRNG

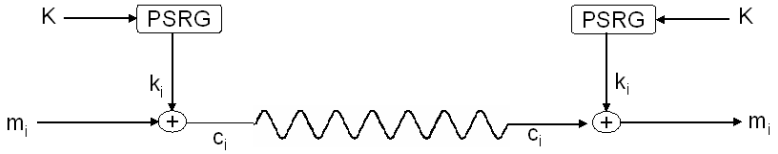
- La version OTP-PRNG n'est pas parfaite car la taille de la clé est inférieure à la taille du message et la clé n'est pas utilisée une seule fois
- Le PRNG doit apparaitre comme aléatoire : il n'y a pas d'algorithme qui permet de distinguer les sorties du PRNG d'une vraie séquence aléatoire uniforme dans un temps raisonnable
- Un PRNG est imprévisible si on à partir de i bits générés par le PRG, il impossible de trouver d'autres bits avec une probabilité $1/2 + \epsilon$ avec ϵ est non négligeable ($\epsilon > 1/2^{80}$)
- Remarque : un PRNG linéaire $r[i+1] = (a * r[i] + b) mod\ n$ est prévisible malgré qu'il a des belles propriétés statistiques. Si on a quelques bits, on peut trouver a et b et deviner le reste du flux.
- Remarque : la fonction $random()$ de $glibc$ est un PRNG prévisible car

$$r[i+1] = (r[i] + r[i-31]) mod\ 2^{32}$$

et elle ne devrait jamais être utilisé pour construire un chiffrement par flot . Kerberos version 4 a utilisé la fonction $random$ et il a été attaqué à ce niveau

Chiffrement par flots

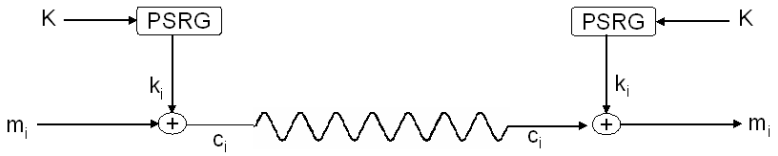
- L'opération de chiffrement s'opère sur chaque élément du texte clair (caractère, bit). Les systèmes de chiffrement en chaîne sont généralement simples et très rapides.



- Très utilisé pour protéger les données multimédia :
  - RC4 : utilisé dans SSL et WiFi 802.11
  - E0/1 : utilisé dans le Bluetooth
  - A5/1, A5/2, A5/3 : utilisés dans le GSM
- Pas besoin d'avoir le message au complet ni d'avoir sa longueur pour commencer à chiffrer
- On peut les diviser en deux classes : synchrone et asynchrone

Chiffrement par flots

➪ Synchrone :

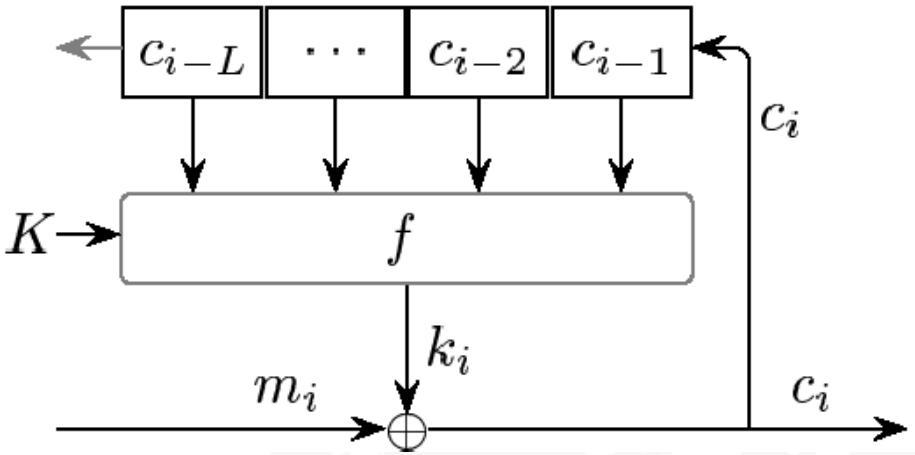


- PSRG (Pseudo Random Generato) : La sécurité du système repose sur la qualité du générateur pseudo-aléatoire. Si $k_i = 0 \Rightarrow c_i = m_i$. Si la séquence k_i est infinie et complètement aléatoire, on aura le One-Time-Pad. Généralement, on se situe entre ces deux cas.
- Pas de propagation d'erreurs (bien). Pas de diffusion (mauvais).
- Si lors de la communication, des bits seront perdus ou créés, la synchronisation sera difficile.

Bit erroné	Bit perdu
m_i 00001111$\cdots$ k_i 11010110$\cdots$ c_i 11011001$\cdots$ $b_0 b_1 \ldots$ b_1 erroné c'_i 10011001$\cdots$ k_i 11010110$\cdots$ m'_i 01001111$\cdots$	m_i 00001111$\cdots$ k_i 11010110$\cdots$ c_i 11011001$\cdots$ $b_0 b_1 \ldots$ b_1 perdu c'_i 1011001$\cdots$ k_i 11010110$\cdots$ m'_i 0110010$\cdots$

Chiffrement par flots

➪ Asynchrone :



- L'encryption d'un bit m_i dépend de la clé et des bits $c_{i-L}, \ldots, c_{i-1}$: $k_i = f_K(c_{i-L}, \ldots, c_{i-1})$
- Une erreur au niveau d'un bit c_i affectera les bits $c_{i+1} \ldots c_{i+L}$.
- Si lors de la communication, un bit est perdu ou créé, la synchronisation sera automatique après L étapes.

Chiffrement par flots

➪ Asynchrone (suite) : $L = 4$:

Bit erroné		Bit perdu	
encryption	decryption	encryption	decryption
$m_0 \oplus f_K(c_{-4}c_{-3}c_{-2}c_{-1}) = c_0$	$c_0 \oplus f_K(c_{-4}c_{-3}c_{-2}c_{-1}) = m_0$	$m_0 \oplus f_K(c_{-4}c_{-3}c_{-2}c_{-1}) = c_0$	$c_0 \oplus f_K(c_{-4}c_{-3}c_{-2}c_{-1}) = m_0$
$m_1 \oplus f_K(c_{-3}c_{-2}c_{-1}\ c_0) = c_1$	$c_1 \oplus f_K(c_{-3}c_{-2}c_{-1}\ c_0) = m_1$	$m_1 \oplus f_K(c_{-3}c_{-2}c_{-1}\ c_0) = c_1$	$c_1 \oplus f_K(c_{-3}c_{-2}c_{-1}\ c_0) = m_1$
$m_2 \oplus f_K(c_{-2}c_{-1}\ c_0\ c_1) = c_2$	$c'_2 \oplus f_K(c_{-2}c_{-1}\ c_0\ c_1) = m'_2$	$m_2 \oplus f_K(c_{-2}c_{-1}\ c_0\ c_1) = c_2$	
$m_3 \oplus f_K(c_{-1}\ c_0\ c_1\ c_2) = c_3$	$c_3 \oplus f_K(c_{-1}\ c_0\ c_1\ c'_2) = m'_3$	$m_3 \oplus f_K(c_{-1}\ c_0\ c_1\ c_2) = c_3$	$c_3 \oplus f_K(c_{-2}c_{-1}\ c_0\ c_1) = m'_3$
$m_4 \oplus f_K(\ c_0\ c_1\ c_2\ c_3) = c_4$	$c_4 \oplus f_K(\ c_0\ c_1\ c'_2\ c_3) = m'_4$	$m_4 \oplus f_K(\ c_0\ c_1\ c_2\ c_3) = c_4$	$c_4 \oplus f_K(c_{-1}\ c_0\ c_1\ c_3) = m'_4$
$m_5 \oplus f_K(\ c_1\ c_2\ c_3\ c_4) = c_5$	$c_5 \oplus f_K(\ c_1\ c'_2\ c_3\ c_4) = m'_5$	$m_5 \oplus f_K(\ c_1\ c_2\ c_3\ c_4) = c_5$	$c_5 \oplus f_K(\ c_0\ c_1\ c_3\ c_4) = m'_5$
$m_6 \oplus f_K(\ c_2\ c_3\ c_4\ c_5) = c_6$	$c_6 \oplus f_K(\ c'_2\ c_3\ c_4\ c_5) = m'_6$	$m_6 \oplus f_K(\ c_2\ c_3\ c_4\ c_5) = c_6$	$c_6 \oplus f_K(\ c_1\ c_3\ c_4\ c_5) = m'_6$
$m_7 \oplus f_K(\ c_3\ c_4\ c_5\ c_6) = c_7$	$c_7 \oplus f_K(\ c_3\ c_4\ c_5\ c_6) = m_7$	$m_7 \oplus f_K(\ c_3\ c_4\ c_5\ c_6) = c_7$	$c_7 \oplus f_K(\ c_3\ c_4\ c_5\ c_6) = m_7$
$m_8 \oplus f_K(\ c_4\ c_5\ c_6\ c_7) = c_8$	$c_8 \oplus f_K(\ c_4\ c_5\ c_6\ c_7) = m_8$	$m_8 \oplus f_K(\ c_4\ c_5\ c_6\ c_7) = c_8$	$c_8 \oplus f_K(\ c_4\ c_5\ c_6\ c_7) = m_8$

Chiffrement par flots

⇨ RC4

- Développé par Ron Rivest en 1987
- Breveté et tenu secret pat la société RSA Inc.
- Détails dévulguées sur Usenet en 1994
- Largement utilisé : WEP, WPA, SSL/TSL, Oracle, SQL, etc.
- Grande simplicité et une excellente performance

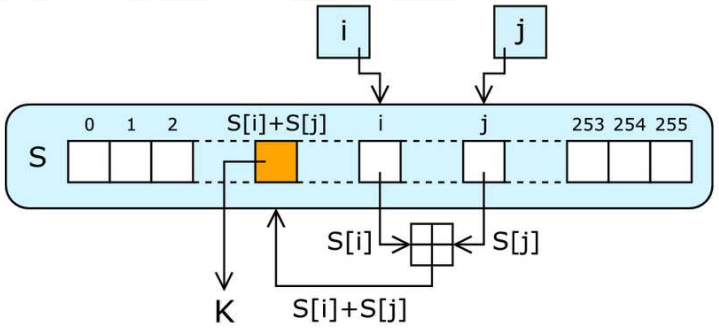
Chiffrement par flots

➪ RC4 Initialisation : pour une clé K de longueur l

```
for i = 0 to 255 do
      S[i] := i          // permutation identité
j := 0
for i = 0 to 255 do
      j := (j + S[i] + K[i mod l]) mod 256
      swap(S[i]; S[j])
```

➪ Génération de flux pseudo-aléatoire

```
i := 0
j := 0
while génération do
      i := i + 1 mod 256
      j := j + S[i ] mod 256
      swap(S[i ]; S[j ])
      output S[S[i ] + S[j ] mod 256]
```



source : wikipedia

Chiffrement par flots

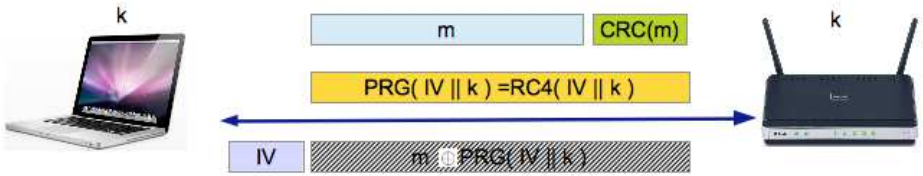
⇨ Problème de chiffrement de plusieurs messages avec la même clé

$$E_k(M_1) \oplus E_k(M_2) = M_1 \oplus M_2$$

- 1ère solution : ne pas réinitialiser le générateur de clés entre deux chiffrements
  - on sera obligé de sauvegarder l'état du générateur
  - pas très intéressante comme solution
- 2ème solution : utiliser une entrée auxiliaire IV (vecteur d'initialisation)
  - Générer IV aléatoirement
  - Caclculer k_{IV} en combinant (concaténation, etc.) k avec IV
  - Encrypter le message avec le flux généré avec k_{IV}
  - Envoyer IV avec le message chiffré

Chiffrement par flots

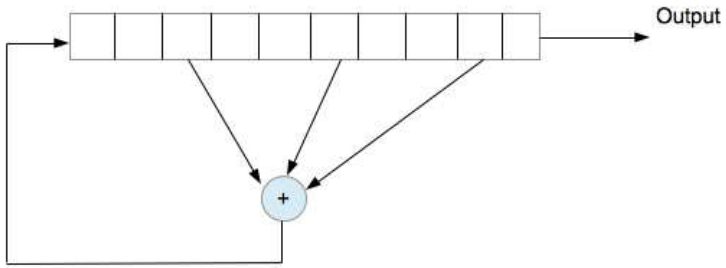
➪ 802.11b (WEP)



- Après 2^{24} le même IV sera réutilisé (voir problème 2PAD)
- Les clés ont le même suffixe : elles sont dépendantes
- Il y a des IV faibles qui aident à révéler des bits de la clé
- En 2001 (Fluhrer, Mantin et Shamir) ont montré qu'à partir de 10^6 trames, on peut retrouver la clé
- l'attaque a été raffinée pour pouvoir retrouver la clé à partir de 40 000 trames

Chiffrement par flots

➪ LSFR (Linear Feedback Shift Register)



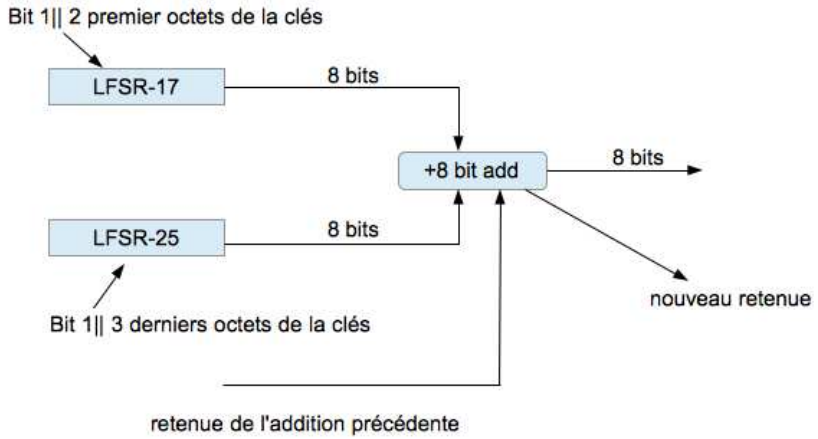
- Un LSFR est cyclique : La séquence maximale générée par un LSFR de taille L est $2^L - 1$ bits.
- On peut déterminer si la séquence du LSFR est maximale ou non.

➪ Systèmes basés sur des LSFR

- CSS (Content Scrambling System) basé sur 2 LSFRs
- GSM utilise (A5/1,2) : basé sur 3 LSFRs
- Bluetooth (E0) : 4 LSFR Tous ces algorithmes ont été attaqués

Chiffrement par flots

⇨ CSS clé de 40 bits (limité par une la loi américaine liée à d'exportation)



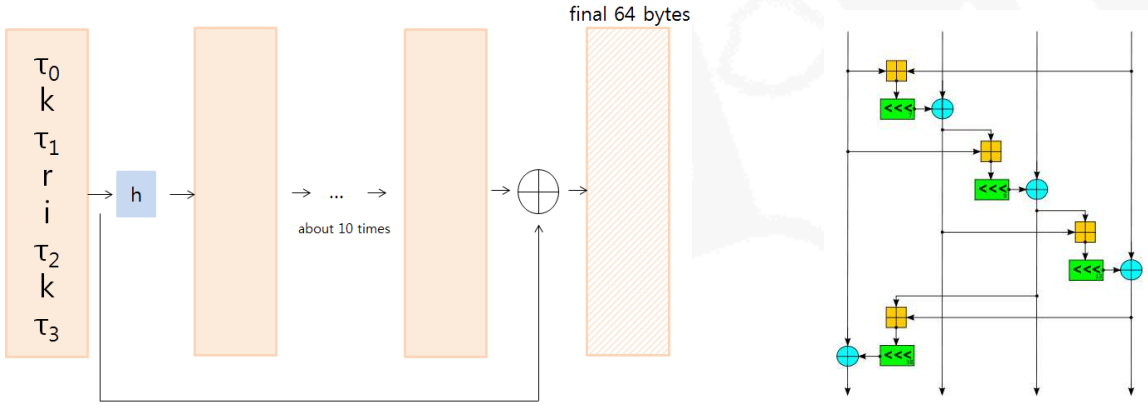
- Attaque par recherche exhaustive est possible. D'autres attaques plus rapides existent
- Attaque dans un temps de l'ordre de 2^{16}
- Supposons que l'on connait quelques octets (octets liés au format) d'un film chiffré
- On parcoure les différentes valeurs possibles de premier LSFR et on détermine le contenu du deuxième
- On valide la clé sur d'autres octets connus (on décrypte et on regarde le résultat)

Chiffrement par flots moderne

- eStream (2008) La clé est utilisé une seule fois. Chaque fois qu'elle est utilisée, elle est combinée avec un nonce. $PRG : \{0,1\}^s \times Nonce \longrightarrow \{0,1\}^n$ avec $n >> s$
- Exemple de eStream : Salsa20/12 et Sosemanuk
- Salsa20 : $Salsa20 : \{0,1\}^{128\ ou\ 256} \times \{0,1\}^{64} \longrightarrow \{0,1\}^n$

$$Salsa20(k,r) = H(k,r,0)||H(k,r,1)||\ldots$$

H h



$\tau_0, \tau_1, \tau_2, \tau_3$ sont des constantes de 8 octets chaque, k (8 octets), r (8 octets) et i (8 octets)

Chiffrement par flots moderne

➪ Performance AMD Opteron, 2.2 GHz (Linux)

PRG	Speed (Mb/sec)
RC4	126
Salsa2012	643
Sosemanuk	727

Chiffrement par flots

⇨ Autres remarques **OTP-2PAD**

➳ Sécurité de 2PAD (comme OPT mais une clé est utilisée pour chiffrer deux ou plusieurs messages).

- $c_1 = m_1 \oplus PRG(k)$ et $c_2 = m_2 \oplus PRG(k)$ ce qui fait que $c_1 \oplus c_2 = m_1 \oplus m_2$
- quand les messages m_1 et m_2 ne sont pas généré aléatoirement, il y a moyen de trouver m_1 et m_2 à partir de $m_1 \oplus m_2$
- si le texte est en anglais par exemple, on se basant sur la redondance, on peut trouver m_1 et m_2.

➳ Le projet (Venona 1941-1946) http ://en.wikipedia.org/wiki/Venona_project

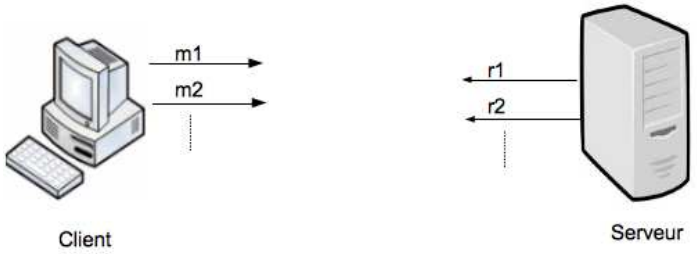
- les Russes ont utilisé la même clé pour chiffrer plusieurs messages
- bien que la clé est générée aléatoirement (en lançant des dés), les Américains ont pu déchiffrer plusieurs messages

➳ chiffrer un fichier ou un disque dur : il faut éviter le chiffrement par flots, car si un pirate copie les différentes versions d'un même fichier chiffré il aura un 2PAD et il pourrait déchiffrer le fichier

Chiffrement par flots

➪ Autres remarques **OTP-2PAD**

➙ MS-PTTP (windows NT) Le client et le serveur se partagent la même clé qu'il utilise pour s'échanger de message en utilisant PRG.



On observant le flux, on trouve que

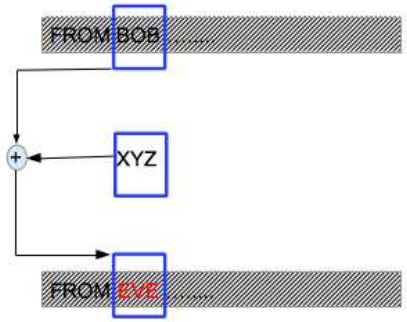
- les messages envoyés par le client sont $(m_1||\ldots||m_n) \oplus PRG(k)$
- les messages envoyés par le serveur sont $(r_1||\ldots||r_n) \oplus PRG(k)$
- pour éviter cette faiblesse (2PAD), il faut utiliser deux clés (une clé par direction)

Chiffrement par flots

➪ Autres remarques

➙ Intégrité : Le chiffrement par flot en général et OTP en particulier n'offre pas l'intégrité

- un intrus qui capte $m \oplus K$ peut le modifie en envoyant $m \oplus K \oplus m' = m \oplus m' \oplus K$
- supposons que m est un courriel qui commence par $FORM : BOB$ et le pirate veut le modifier pour le rendre $FROM : EVE$.



- supposons que le pirate sait aussi que le message provient de BOB.
- tout ce qu'il a à faire est de trouver XYZ

$$BOB \oplus K \oplus XYZ = EVE \oplus K \Longrightarrow BOB \oplus XYZ = EVE$$

$$426F62 \oplus XYZ = 457665 \Longrightarrow XYZ = 426F62 \oplus 457665 = 071907$$

Plan

- Variables Aléatoires
- Chiffrement par flots
- **La cryptographie par seuil (threshold cryptography)**

La cryptographie par seuil (threshold cryptography)

➪ Problème : Pour exécuter une opération telle que le déchiffrement d'un fichier, il faut l'autorisation de k utilisateurs parmi n.

➪ Solution 1 : Diviser le secret sur les utilisateurs

- Exemple secret= 1201 3727 3309 5553 2154
- Utilisateur 1 connait 1201, utilisateur 2 connait 3727, utilisateur 3 connait 5553, utilisateur 4 connait 2154.
- Problème : besoin de tous les utilisateurs pour retrouver le secret

➪ Solution 2 : Construire un polynôme de degré k,.

$$y = f(x) = s + \sum_{i=1}^{k-1} a_i x^i$$

avec $s = f(0)$ est le secret qui permet d'exécuter l'opération.

- Trouver n points (x_i, y_i) passant par le polynôme
- chaque utilisateur aura un point comme son propre secret.
- k points permettent de résoudre les inconnus $(s, a_1, \ldots a_{k-1})$, y compris le secret s

La cryptographie par seuil (threshold cryptography)

➪ Trouver le polynôme : Interpolation Polynomiale de Lagrange

- Étant donnés $k+1$ points $(x_0, y_0), \ldots, (x_k, y_k)$ avec tous les x_i différents
- L'Interpolation Polynomiale de Lagrange passant par ces points est :

$$L(x) = \sum_{i=1}^{k} y_i l_i(x)$$

avec

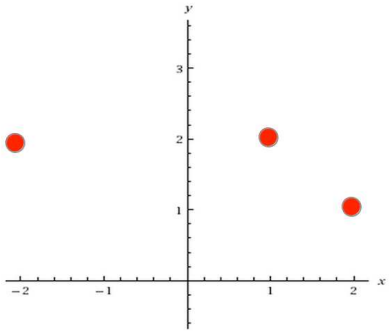
$$l_i(x) = \prod_{0 \leq m \leq k \ et \ m \neq i} \frac{(x - x_m)}{x_i - x_m}$$

- Il est claire que $l_{j \neq i}(x_i) = 0$ et $l_i(x_i) = 1$ donnant ainsi que $L(x_i) = y_i + 0 + \ldots + 0 = y_i$, ce qui montre que pour tout (x_i, y_i), on a :

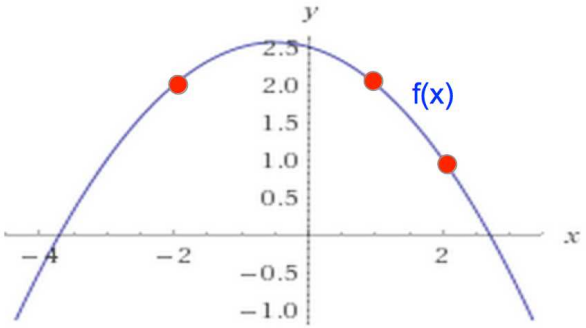
$$L(x_i) = y_i$$

La cryptographie par seuil (threshold cryptography)

➪ Trouver le polynôme : Interpolation Polynomiale de Lagrange (exemple)



(1,2),(−2,2),(2,1)



f(x)=?

La cryptographie par seuil (threshold cryptography)

➪ Trouver le polynôme : Interpolation Polynomiale de Lagrange (exemple)

- Étant donnés 3 points $(x_0, y_0) = (1, 2),\ (x_1, y_1) = (-2, 2),\ et\ (x_2, y_2) = (2, 1)$ avec tous les x_i différents
- L'Interpolation Polynomiale de Lagrange passant par ces points est :

$$L(x) = \sum_{i=0}^{2} y_i l_i(x)$$

avec

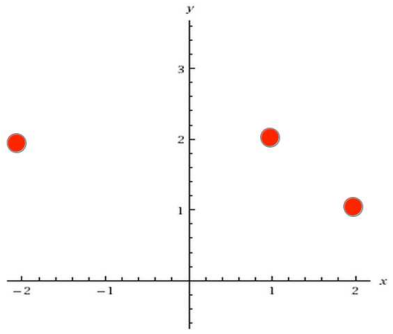
$$\begin{aligned}
l_0(x) &= \frac{(x-x_1)}{x_0-x_1}\frac{(x-x_2)}{x_0-x_2} = \frac{(x+2)}{1+2}\frac{(x-2)}{1-2} = -\frac{1}{3}(x^2-4) \\
l_1(x) &= \frac{(x-x_0)}{x_1-x_0}\frac{(x-x_2)}{x_1-x_2} = \frac{(x-1)}{-2-1}\frac{(x-2)}{-2-2} = \frac{1}{12}(x^2-3x+2) \\
l_2(x) &= \frac{(x-x_0)}{x_2-x_0}\frac{(x-x_1)}{x_2-x_1} = \frac{(x-1)}{2-1}\frac{(x+2)}{2+2} = \frac{1}{4}(x^2+x-2)
\end{aligned}$$

- Donc

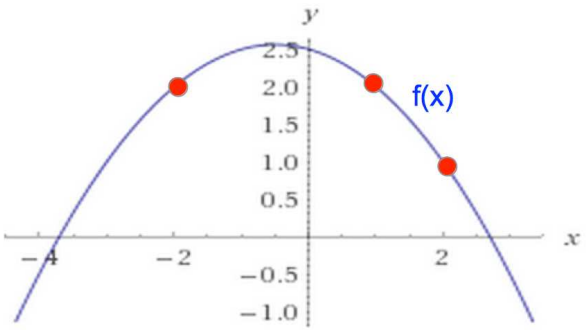
$$L(x) = -\frac{1}{4}x^2 - \frac{1}{4}x + \frac{5}{2}$$

La cryptographie par seuil (threshold cryptography)

➪ Trouver le polynôme : Interpolation Polynomiale de Lagrange (exemple)



(1,2),(−2,2),(2,1)



$$f(x) = -\frac{1}{4}x^2 - \frac{1}{4}x + \frac{5}{2}$$

La cryptographie par seuil (threshold cryptography)

➪ Algorithme de Shamir de partage de secrets : Publié par Adi Shamir 1979 et basé sur l'interpolation polynomiale de Lagrange dans $\mathbf{Z}_q$ (modulo q) avec q est premier

- Étant donnés un seuil (k, n) et un secret s dans $\mathbf{Z}_q$ avec q est premier
- Choisir aléatoirement k coefficients $a_1, \ldots, a_{k-1}$
- Fixer $a_0 = s$
- Construire le polynôme $f(x) = a_0 + a_1 x^1 + \ldots + a_{k-1} x^{k-1}$
- Fixer les points $s_i = (i, f(i)) \; mod \; q$, pou $i = 1, \ldots, n$
- Chaque utilisateur aura au moins un point s_i
- k utilisateurs peuvent se mettre ensemble pour trouver s en utilisant l'interpolation de Lagrange.
- Tous les calculs se font modulo q

➪ Trouver le secret :

$$s = L(0) = \sum_{i=1}^{k} f(i) l_i(0) \text{ avec } l_i(0) = \prod_{0 \leq x_m \leq k \; et \; x_m \neq i} \frac{x_m}{x_m - i}$$

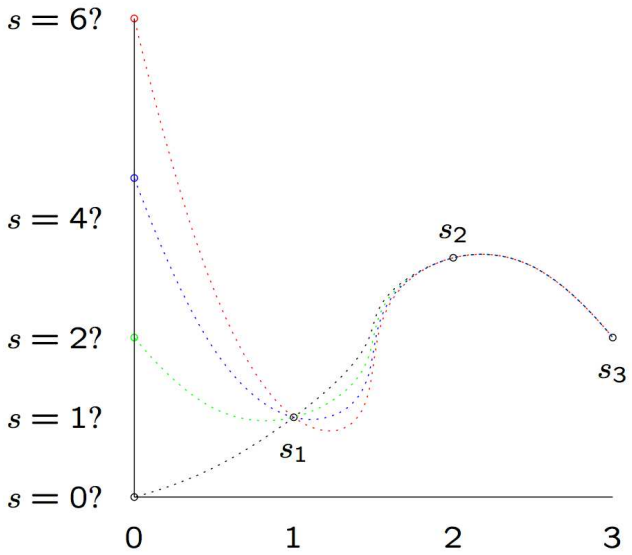
La cryptographie par seuil (threshold cryptography)

➪ Trouver le secret (plus général) : ayant k points, $(x_1, y_1), \ldots, (x_k, y_k)$, d'une interprétation de Lagrange, le secret sera :

$$s = L(0) = \sum_{i=1}^{k} y_i l_i(0) \text{ avec } l_i(0) = \prod_{0 \leq x_m \leq k \text{ et } x_m \neq x_i} \frac{x_m}{x_m - x_i}$$

La cryptographie par seuil (threshold cryptography)

➪ Algorithme de Shamir de partage de secrets : Interpolation Polynomiale de Lagrange (exemple)



T-79.159 Cryptography and Data Security, 24.03.2004 Lecture 9: Secret Sharing, Threshold Cryptography, MPC, Helger Lipmaa

La cryptographie par seuil (threshold cryptography)

➪ Algorithme de Shamir de partage de secrets : Exercice

- Étant donnés un seuil $(3, 5)$ et un secret s dans $\mathbf{Z}_{23}$
- Nos 5 utilisateurs ont eu chacun un point : Alice (2,14), Bob (1,9), Dave (3,21), Felix (4,8), George (5,19)
- Montrer que ces points sont les 5 "s_i" générés par l'algorithme de Shamir quand $a_0 = 6, a_1 = 2, \ et \ a_2 = 1$
- Montrer comment Bob, Felix et Dave peuvent se mettre ensemble pour trouver le secret.