Prévision & génération d'aléa

Matthieu Finiasz

- En cryptographie, on a besoin d'aléa pour beaucoup de choses :
  - génération de clef,
  - masquage aléatoire,
  - génération de nonce/challenge...

    → pas de sécurité en restant déterministe.

- Il est important qu'un adversaire ne puisse pas deviner cet aléa, la sécurité repose sur cela,

  → il faut de l'aléa "imprévisible".

- En plus, il faut pouvoir en générer beaucoup, et vite...

Qu'est-ce qu'un bonne prévision ?

Prévision météo dans le desert

- Dans une région désertique, où il ne pleut qu'un jour par an, deux météorologues font des prévisions :
  - le premier annonce systématiquement qu'il fera beau,
  - le deuxième se trompe avec 1 chance sur 4.

→ Qui des deux fait les meilleures prévisions ?

- Si l'on regarde la probabilité d'annoncer le temps qu'il va faire avec succès :
  - le premier météorologue est correct chaque jour où il fait beau :
    → probabilité de succès de $\frac{364}{365}$,
  - le deuxième météorologue est correct 3 fois sur 4, qu'il fasse beau ou non :
    → probabilité de succès de $\frac{3}{4}$.

- La prévision du premier est donc bien meilleure.

- En revanche, si l'on regarde l'information que nous donne chaque prévision :
  - la première est indépendante du temps qu'il va faire
    → ne nous donne en fait aucune information,
  - pour le deuxième, la probabilité qu'il pleuve est :

$$P(\text{“pluie”}\,|\,\text{“pluie prévue”}) = \frac{\frac{3}{4}\times\frac{1}{365}}{\frac{3}{4\times365}+\frac{364}{4\times365}} = \frac{3}{367}$$

$$P(\text{“pluie”}\,|\,\text{“pluie pas prévue”}) = \frac{\frac{1}{4}\times\frac{1}{365}}{\frac{1}{4\times365}+\frac{3\times364}{4\times365}} = \frac{1}{1093}$$

- La probabilité qu'il pleuve est 9 fois plus élevée quand la pluie est annoncée.

La notion d'information mutuelle

- L'information mutuelle permet de mesurer (en bits) la quantité d'information qu'une variable aléatoire nous donne sur une autre variable aléatoire :

$$I(X;Y) = \sum_{x,y} P(X = x \cap Y = y) \log_2 \left(\frac{P(X=x \cap Y=y)}{P(X=x) \times P(Y=y)} \right)$$

- Si X et Y sont indépendants $I(X;Y) = 0$
  → cas du premier météorologue.
- Pour le deuxième météorologue, l'information mutuelle entre la prévision et le temps est de 0.002 bits
  → la deuxième prévision est donc meilleure.

Quelle est la meilleure notion ?

- Une bonne prévision, est-ce :
  - une prévision juste ?
  - une prévision qui nous donne de l'information ?

Quelle est la meilleure notion ?

- Une bonne prévision, est-ce :
  - une prévision juste ?
  - une prévision qui nous donne de l'information ?
- Si la variable que l'on veut prédire est non-biaisée, les deux notions sont similaires :
  - une prévision juste plus souvent qu'un tirage aléatoire donne de l'information,
  - une prévision qui donne de l'information permet de construire une prévision juste.
- On utilise la notion la plus simple :

  → la probabilité d'être juste.

La génération d'aléa en cryptographie

- Notre but est de générer de l'aléa "imprévisible" :
  - un adversaire ne doit pas pouvoir prédire les bits,
  - son avantage est sa probabilité d'être juste moins $\frac{1}{2}$
    - → il faut minimiser son avantage.
- Soit on a un RNG (random number generator) :
  - générateur physique par exemple,
    - → indépendant de tout ce à quoi l'adversaire a accès,
  - il suffit qu'il soit non-biaisé pour être parfait.
- Soit on a un PRNG (pseudo-random number generator) :
  - l'adversaire observe une (longue) séquence aléatoire,
  - il ne doit pas pouvoir prévoir la suite.

Exemples de RNG

Générateurs “physiques”

Les générateurs “parfaits”

- La meilleure façon de générer de l'aléa est de mesurer un phénomène réellement aléatoire :
  - pile ou face,
  - lancer de dés,
  - boules du loto...
- Il sont parfaits au sens où les lancer sont réellement mutuellement indépendants.
- En revanche :
  - ils sont lents,
  - ils sont difficile à automatiser,
  - ils peuvent être biaisés (dés pipés...).

Générateurs aléatoires classiques

- Pour aller plus vite, le phénomène aléatoire mesuré a lieu dans un module hardware dédié :
  - bruit thermique d'une résistance,
  - désynchronisation d'horloges internes,
  - photons sur un miroir semi-transparent...

- Il est aussi possible de faire cela sans hardware dédié :
  - le nombre de cycles pour exécuter une instruction varie
    → possible de mesurer cela (cf. Havege).

⚠ Ces générateurs peuvent être biaisés : rien ne garantit que le 0 et le 1 ont la même probabilité.

Redressement d'un générateur aléatoire

Méthode de von Neumann

- Un générateur physique est souvent biaisé :
  - 0 et 1 n'ont pas exactement la même probabilité,
    - → il faut le redresser.
- C'est possible en regroupant deux tirages : [van Neumann]
  - 01 donne un 0,
  - 10 donne un 1,
  - 00 et 11 ne donnent rien,
    - → on perd un peu en débit.
- Cela garantit que des tirages indépendants donnent un 0 ou un 1 avec la même probabilité,
  - il faut vérifier que les tirages sont indépendants
    - → avec des tests statistiques.

Tests statistiques

Principe d'un test statistique

- Les tests statistiques sont l'outil principal pour tester la qualité d'un générateur aléatoire.
- Leur fonctionnement est le suivant :
  - on part d'une (longue) séquence générée,
  - on calcule une valeur statistique sur cette séquence,
  - si cette valeur dépasse un seuil, l'aléa est probablement mauvais.
- Une séquence d'aléa parfait peut dépasser le seuil :
  - on calcule donc une p-valeur : la probabilité que la valeur statistique calculée sur de l'aléa parfait, dépasse la valeur statistique calculée sur la séquence observée.
    → elle doit être distribuée uniformément dans $[0; 1]$.

Fonctionnement d'un test statistique

- En pratique, le fonctionnement est un peu plus complexe :
  - on part de m séquences générées,
  - pour chacune on calcule une valeur statistique et la p-valeur associée,
  - on vérifie que les m p-valeurs calculées sont réparties uniformément dans $[0; 1]$,
    - → pour cela on recalcule une p-valeur.
- Ne reste plus qu'à choisir une valeur statistique,
  - il faut aussi pouvoir facilement calculer la p-valeur,
    - → connaître la distribution sur de l'aléa parfait.
  - le NIST propose une série de tests.

Test de répartition uniforme

Le test le plus simple

- Ce test a pour but de vérifier que 0 et 1 ont la même probabilité :
  - on compte les 1 dans une séquence de n bits,
  - la p-valeur est la probabilité d'avoir plus de 1 dans une séquence parfaitement aléatoire de même longueur.

- Si 1 a une probabilité supérieure à $\frac{1}{2}$ dans notre RNG :
  - le nombre de 1 sera plus grand que $\frac{n}{2}$,
  - les p-valeurs seront plus proches de 0,
    - → pas de répartition uniforme.
  - l'aléa sera considéré comme mauvais.

- Mais le test avec les p-valeurs va plus loin !
  - on considère la séquence

    $$0101010101010101010101...$$

  - le nombre de 1 est toujours exactement $\frac{n}{2}$,
  - c'est la moyenne, donc la séquence devrait passer le test.

- Mais le test avec les p-valeurs va plus loin !
  - on considère la séquence
    $$0101010101010101010101...$$
  - le nombre de 1 est toujours exactement $\frac{n}{2}$,
  - c'est la moyenne, donc la séquence devrait passer le test.
- Ce n'est pas le cas, la séquence est très mauvaise :
  - les p-valeur sont toutes égales à 0.5
    $\longrightarrow$ pas de répartition uniforme,
  - le test ne vérifie pas seulement qu'il y a autant de 0 que de 1, il vérifie que la distribution se comporte comme une loi binomiale.

Plages de 0 ou de 1

- Ce test vérifie que le générateur produit des plages de 0 (ou de 1) avec la bonne longueur/fréquence :
  - exactement comme le test précédent,
  - on compte le nombre de plages de 0 et de 1
    → il y en a $\frac{n}{2}$ en moyenne.
- Permet par exemple de rejeter des séquences comme

  00 11 11 00 11 00 00 11...

- On peut aussi mesurer la longueur des plus grandes plages de 1 de chaque séquence.
  → distribution plus compliquée, mais test important.

Autres tests statistiques

- On peut créer autant de tests statistiques que l'on veut
  - il suffit de savoir calculer les p-valeur correspondantes.

- Par exemple :
  - test spectral (transformée de Fourier)
    → détecte des répétitions/motifs
  - rang de matrice $m \times m$
    → détecte des relation linéaires entre bits,
  - occurrences d'un motif donné
    → permet de vérifier que tous les motifs apparaissent,
  - compressibilité des données
    → de l'aléa ne doit pas être compressible...

Exemples de PRNG

Générateurs de nombres pseudo-aléatoires

Principes de base

- Le principe d'un PRNG est assez différent d'un RNG
  - un PRNG part d'une graine
    → souvent générée par un RNG,
  - il possède un état interne qui évolue,
  - les bits de sortie sont dérivés de cet état interne.
- Contrairement à un RNG, il n'y a pas de limite physique au débit d'un PRNG
  → limité par le coût de la mise à jour de l'état interne.
- En revanche, un PRNG ne génère pas d'entropie :
  - seule la graine est réellement aléatoire,
  - il faut donc la renouveler régulièrement.

Limites d'un PRNG

- Un PRNG est forcément cyclique :
  - il y a un nombre fini d'états internes possibles,
  - on revient forcément à l'état initial,
    - → on cherche à maximiser sa période.

- Si un adversaire devine l'état interne :
  - il peut lui aussi le faire évoluer,
  - il peut connaître exactement la suite qui sera générée.

- Il y a donc un compromis entre :
  - générer beaucoup de bits pour avoir un PRNG rapide,
  - générer peu de bits pour mieux masquer l'état interne.

Générateur congruentiel

- Le PRNG le plus simple est le générateur congruentiel :

$$X_{n+1} = a \times X_n + b \mod m$$

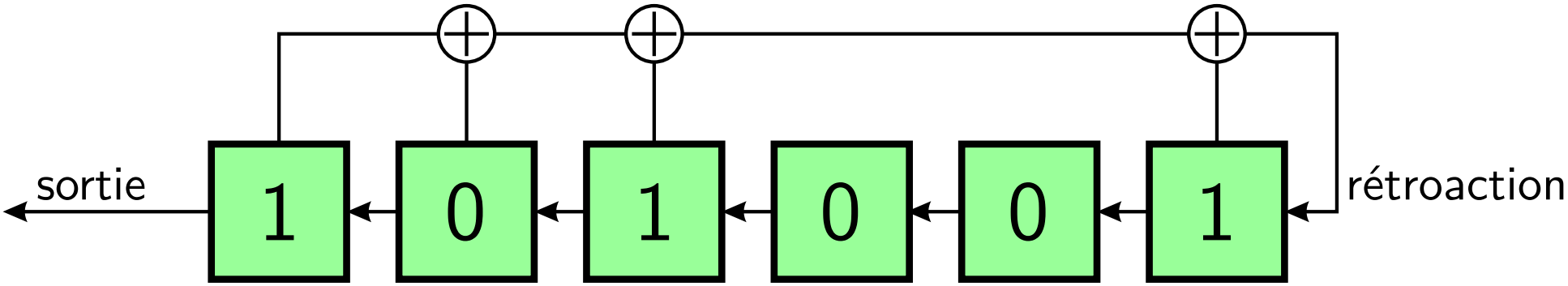
- Sa période est maximale (égale à m) si :
  - b et m sont premiers entre eux,
  - $a - 1$ est multiple de tous les facteurs premiers de m,
  - $a - 1$ est multiple de 4 si m est multiple de 4.
- Par exemple : $m = 2^{32}$, $a = 1103515245$ et $b = 12345$.

⚠ Tous les bits de l'état ne sont pas très "aléatoires" :

- le bit de poids faible alterne entre 0 et 1,
- pour un a petit, le bit de poids fort ferait des longues plages.

Utilisation de registres à décalage
LFSR (Linear Feedback Shift Register)

- Le générateur congruentiel est adapté à des processeurs qui font facilement des calculs entiers
  - il faut au moins programmer une multiplication
    - → nécessite pas mal de portes logiques.
- Pour des puces dédiées, il est plus simple d'utiliser des registres à décalage à rétroaction linéaire :



polynôme : $X^6 + X^5 + X^4 + X + 1$.

Utilisation de registres à décalage
LFSR (Linear Feedback Shift Register)

- La période du LFSR dépend de son polynôme de rétroaction :
  - s'il est primitif de degré d, la période est $2^d - 1$,
    - → l'état interne passe par toutes les valeurs possibles.
- La séquence générée possède de bonne propriétés statistiques :
  - bonne proportion de 0 et 1 (juste un 1 de plus),
  - bon nombre de plages de 0 et de 1,
  - bonnes longueurs de plages
- Mais la séquence est même un peu trop régulière :
  - pas de plage de 0 de longueur d,
  - ne passe pas le test de rang de matrices $(d+1)\times(d+1)$.

LFSR pour la cryptographie

- Dans un contexte cryptographique, retrouver l'état interne doit être difficile :
  - si le polynôme est connu : d bits suffisent,
  - si le polynôme est inconnu : $2d$ bits suffisent.

- Pour protéger l'état interne, on filtre les bits de sortie :
  - combinaison de plusieurs registres,
  - filtre non linéaire, poinçonnage...
- C'est le principe des chiffrements à flot :
  - l'initialisation est la clef,
  - on masque le message avec la suite pseudo-aléatoire,
    → GSM : A5/1 ou A5/2, Bluetooth : E0.

Autres méthodes usuelles

- Mersenne Twister :
  - l'état interne est composé de 624 mots de 32 bits,
    → récurrence d'ordre 624
  - un peu comme un LFSR avec des mots de 32 bits,
  - possède une période de $2^{19937} - 1$,
    → passe à peu près tous les tests statistiques.

- Fonction de hachage cryptographique :
  - on utilise une fonction comme MD5, SHA-1...
  - l'état interne est de la taille de la sortie de la fonction,
  - un biais donnerait une attaque sur la fonction
    → très bons en pratique, mais sans preuves.

Un PRNG prouvé sûr

Qu'est-ce qu'un PRNG prouvé sûr ?

- Un PRNG est prouvé sûr, si on peut démontrer que distinguer une suite qu'il génère d'une suite réellement aléatoire nécessite de résoudre un problème difficile.

- Aucun test statistique ne doit permettre de le distinguer :
  - il suffit que le générateur soit imprédictible
    → soit à gauche, soit à droite.
  - il faut prouver que prédire nécessite de résoudre un problème difficile,
    → typiquement, un problème mathématique.

Le générateur de Blum-Blum-Shub

- Le fonctionnement est très simple :
  - choisir p et q premiers congrus à $3 \mod 4$.
  - calculer $N = pq$,
  - initialiser $x_0 = x^2 \mod N$,
    (avec $\mathrm{pgcd}(x, p) = \mathrm{pgcd}(x, q) = 1$),
  - à chaque étape calculer $x_i = x_{i-1}^2 \mod N$.
    → extraire le bit de poids faible (ou fort) de x_i.
- Par exemple, $p = 19$, $q = 11$ et $N = 209$:
  - on choisit $x = 2$ donc $x_0 = 4$,
  - cela produit la suite
    4, 16, 47, 119, 158, 93, 80, 130, 180, 5, 25, 207, 4...
    → 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0...

- Dans $\mathbb{Z}_p$ ou $\mathbb{Z}_q$:
  - un élément inversible sur 2 est un résidu quadratique,
  - chaque élément a 2 racine carrées,
    - → l'une est paire, l'autre impaire,
  - une seule est un résidu quadratique.

$$\sqrt{16} \mod 19 = \pm 4 \mod 19 = \{4, 15\}$$
$$\sqrt{16} \mod 11 = \pm 4 \mod 11 = \{4, 7\}$$

Le générateur de Blum-Blum-Shub

Éléments de preuve

- Dans $\mathbb{Z}_p$ ou $\mathbb{Z}_q$:
  - un élément inversible sur 2 est un résidu quadratique,
  - chaque élément a 2 racine carrées,
    - $\longrightarrow$ l'une est paire, l'autre impaire,
  - une seule est un résidu quadratique.
- Dans l'anneau $\mathbb{Z}_N$, isomorphe à $\mathbb{Z}_p \times \mathbb{Z}_q$ (CRT) :
  - un élément inversible sur 4 est un résidu quadratique,
  - chaque élément a 4 racines carrées,
    - $\longrightarrow$ 2 paires, 2 impaires,
  - une seule est un résidu quadratique.

$$\sqrt{16} \mod 209 = \{4, 205, 15, 194\}$$

Le générateur de Blum-Blum-Shub

Éléments de preuve

- Pour décider de la parité de x_{i-1} en connaissant x_i, il faut décider laquelle des racines est un résidu quadratique,
  - l'hypothèse de résiduosité quadratique est de considérer ce problème comme difficile
    → on ne sait pas faire sans factoriser N.

- Le générateur BBS est donc imprédictible à gauche :
  - aucun test statistique (qui ne factorise pas N) ne peut donc aider à distinguer une séquence de BBS,
  - si la factorisation de N est inconnue, BBS est sûr,
    → il faut donc jeter p et q après calcul de N.

Période

- La période de BBS est "l'ordre" de l'opération d'élévation au carré,
  → l'ordre (multiplicatif) de 2 dans les puissances.
- Dans $\mathbb{Z}_N$ le plus grand sous-groupe multiplicatif cyclique est d'ordre $\lambda(N) = \mathrm{ppcm}(p-1, q-1)$,
  - soit g un générateur de ce sous-groupe (g^i)
    → les puissances i sont dans l'anneau $\mathbb{Z}_{\lambda(N)}$,
  - dans cet anneau, l'ordre de 2 divise $\lambda(\lambda(N))$.
- Pour maximiser la période on prend :
  - $p = 2p_1 + 1$ et $p_1 = 2p_2 + 1$ avec p_1 et p_2 premiers,
  - $q = 2q_1 + 1$ et $q_1 = 2q_2 + 1$ avec q_1 et q_2 premiers,
    → $\lambda(\lambda(N)) = 2p_2q_2$, période de 1, 2, ou beaucoup.

Conclusion sur BBS

- BBS est donc un PRNG :
  - prouvé sûr, si factoriser est difficile (N de 2048 bits)
    $\longrightarrow$ sous l'hypothèse de résiduosité quadratique,
  - imprédictible à gauche, mais pas à droite,
    (il suffit de connaître x_i pour générer la suite),
  - qui passe avec succès tous les tests statistiques,
  - avec une période longue (ou égale à 2 si on n'a pas de chance),
  - très lent (quelques Mbits/s au max).

- Au lieu du bit de parité, on peut extraire le bit de poids fort, les 2, ou jusqu'à $O(\log\log(N))$ bits de poids faible.

Conclusion

Conclusion

- La génération d'aléa est un problème compliqué :
  - plusieurs notions d'imprédictibilité existent,
    - → en crypto, la proba de deviner le bit suivant,
  - beaucoup de méthodes existent pour générer de l'aléa
    - → il n'y en a pas une meilleure qu'une autre.
- En pratique, la meilleure dépend du débit nécessaire :
  - un vrai RNG est le mieux pour des très bas débits,
  - pour aller plus vite, il faut un PRNG
    - → initialisé par un véritable RNG,
  - BBS est très bien mais cher,
  - le Mersenne twister est très rapide, mais pas très sûr pour de la cryptographie...