Générateur Pseudo Aléatoire

Porte dérobé du protocole NIST

9 juin 2016

Yankhoba CISSE et Clément HUYART

Table des matières

Chapitre 1

Introduction

1.1 L'aléatoire

1.1.1 Qu'est ce que l'aléatoire

En mathématique, une suite aléatoire est une suite de symbole d'un alphabet ne possédant aucune structure ou règle de prédiction identifiable. L'aléatoire existe dans des phénomènes naturels comme la radioactivité, les bruits thermiques ou électromagnétiques ou bien la mécanique quantique.

Cependant il est impossible pour un être humain de donner une suite aléatoire de chiffre. De même qu'un ordinateur ne peut pas déterminer une vraie suite aléatoire de par sa nature déterministe. Etant donné que l'aléatoire est nécessaire à beaucoup d'application informatique aujourd'hui, il nous faut trouver un moyen de simuler des suites aléatoires, ou plutot pseudo-aléatoires dans le cas d'un ordinateur.

L'aléatoire est nécessaire en cryptographie. Par exemple, il est utilisé dans l'un des protocoles d'échanges de clé le plus utilisé : l'échange de clé de Diffie Hellman.
Pour cela, nous allons étudier les générateurs pseudo-aléatoires.

1.1.2 Générateur pseudo-aléatoire

Un générateur pseudo-aléatoire (Pseudorandom number generator en anglais) est un algorithme qui génère une séquence de nombres présentant certaines propriétés du hasard.

A la différence d'un générateur aléatoire, de tels générateurs ne sont pas entièrement aléatoire. Les deux principales raisons de l'utilisation de ces générateurs sont :

— Il est difficile d'obtenir de vrai nombre aléatoire.
— Les générateurs pseudo-aléatoires sont facilement implémentable sur informatique.

Les plus répandus de nos jours sont ceux qui utilisent une congruence linéaire. Certains utilisent des propriétés de la suite de Fibonacci ou bien des registres à décalage à rétroaction linéaire.

1.1.3 Tester l'aléa

On peut vérifier les propriétés aléatoires d'un générateur à l'aide de test. Les tests les plus utilisés sont des tests statistiques.

Pour les générateurs pseudo-aléatoires, on utilise des tests statistiques dit d'adéquation. On suppose que le générateur suit une loi de distribution P. On va ensuite comparer la distribution réelle du générateur avec P. Par exemple, dans le cas d'un générateur aléatoire simulant une pièce de monnaie, on doit vérifier que la probabilité d'obtenir pile ou face doit être équiprobable.
Les tests d'adéquation les plus utilisés sont les tests du χ^2 et de Kolmogorov-Smirnov.

Avant de nous intéresser au générateur présenté par le NIST, nous allons, dans un premier temps, vous présenter les générateurs utilisant une congruence linéaire.

Chapitre 2

Générateur à congruence linéaire

2.1 Présentation

Un générateur congruentiel linéaire est défini par 4 nombres dit "nombres magiques" et une équation.
On définit :

- m le module tel que $m > 0$
- a le multiplicateur tel que $0 \leq a < m$
- c l'incrément tel que $0 \leq c < m$
- X_0 la valeur initiale tel que $0 \leq X_0 < m$

Le générateur est ensuite définie par l'équation :
$X_{n+1} = a * X_n + c \mod m$
Les propriétés aléatoires du générateur dépendent du choix des 4 nombres magiques. Nous allons voir comment les choisir afin que le générateur ait les propriétés voulues.

2.2 Le choix du module

Etant donné que l'on veut une suite pseudo-aléatoire, on cherche à avoir une période qui soit la plus grande possible. Pour cela on va devoir choisir le module m très grand car la période ne peut pas avoir plus de m éléments. De plus nous voulons choisir une valeur de m de telle sorte que le calcul de X_{n+1} se fasse le plus rapidement possible.

Le problème est que le coût d'une division est élevé. Pour minimiser ce coût, une bonne valeur pour m est la taille d'un mot machine.

Un mot machine est une unité de base manipulée par un microprocesseur en informatique. La pluplart des machines actuelles utilisent des mots machine de 32 ou 64 bits.
Soit $\omega = 2^e$ la taille d'un mot machine.
Dans un ordinateur, les additions et les multiplications se font naturellement modulo ω. En prenant $m = \omega$, les itérations de la suite sont donc plus rapide à calculer. Mais en pratique, les bits de poids faible sont beaucoup moins aléatoire que les bits de poids fort. Une solution à ce problème est de faire un décalage afin de conserver que les bits de poids fort.

2.3 Le choix du multiplicateur

Nous allons montrer comment choisir le multiplicateur a afin que la longueur de la suite soit maximale. Une longue période est essentielle afin de conserver les propriétés de la suite. Après avoir calculé la période maximale, il faut vérifier que la taille de la période soit plus longue que les nombres utilisés par l'application. Le choix de la période dépend donc aussi de l'application utilisée.
De telle suite existe, par exemple :
En prenant $a = c = 1$, $X_{n+1} = X_n + 1 \mod m$ est de période maximale. Cependant cette suite n'est pas aléatoire...

Le cas oú $a = 0$ est rejeté immédiatement, car la suite (X_n) serait constante de valeur c.
Le cas oú $a = 1$ est aussi rejeté, car la suite (X_n) serait une suite arithmétique de raison c modulo m.

Supposons que $2 \leq a$. Posons $b = a - 1$. On a $1 \leq b$.

Théorème 2.3.1. *Un générateur congruentiel linéaire définie par m, a, c et X_0 a une période de longueur m si et seulement si :*

1. *c est premier à m*
2. *pour chaque nombre premier p divisant m, b est un multiple de p*
3. *Si m est un multiple de 4, b est un multiple de 4*

Avant de commencer la démonstration, nous allons énoncer 3 lemmes qui nous servirons.

Lemme 2.3.1. *Soit $G=(X_0, a, c, m)$ un générateur à congruence linéaire et $m = p_1^{e_1} \dots p_k^{e_k}$ sa décomposition en facteur premier. La longueur λ de la période du générateur G est le ppcm des λ_j pour $1 \leq j \leq k$ avec λ_j la longueur de la période du générateur définie par $(X_0 \pmod{p_j^{e_j}}, a \pmod{p_j^{e_j}}, c \pmod{p_j^{e_j}}, p_j^{e_j})$.*

Lemme 2.3.2. *Soit a un entier tel que $1 < a < p^e$ avec p premier.*
Si λ est le plus petit entier positif tel que $\frac{a^\lambda - 1}{a-1} \equiv 0 \pmod{p^e}$ alors :

$$\lambda = p^e \text{ si et seulement si } \begin{cases} a \equiv 1 \pmod{p} \text{ si } p > 2 \\ a \equiv 1 \pmod{4} \text{ si } p = 2 \end{cases}$$

Démonstration théorème 2.3.1. Soient (X_0, a, c, m) un générateur à congruence linéaire, λ sa période.

Grâce au lemme 2.3.1, il suffit de montrer le théorème 2.3.1 dans le cas oú m est une puissance d'un nombre premier.
En effet, $m = p_1^{e_1} \dots p_k^{e_k} = \lambda = ppcm(\lambda_1, \dots, \lambda_k) \leq \lambda_1 \dots \lambda_k \leq p_1^{e_1} \dots p_k^{e_k}$ est vrai si et seulement si $\lambda_j = p_j^{e_j}$ $1 \leq j \leq k$

Si $a = 1$, le théorème est vérifié. En effet, $X_{n+1} = X_n + c \pmod{m}$ est de période m si et seulement si c est premier avec m. On peut donc supposer par la suite que $a > 1$.

De plus, la période est de longueur m si et seulement si tout x tels que $0 \leq x < m$ apparaissent une et une seule fois dans la période si et seulement si la longueur de la période du générateur avec $X_0 = 0$ vaut m. On peut donc supposer que $X_0 = 0$.

Démontrons par récurrence que $X_{n+k} = a^k X_n + \frac{(a^k-1)c}{b} \pmod{m}$.
Pour $k = 0$, la formule est vraie ($X_n = X_n$).
Supposons que la formule est vraie pour un certains k. Montrons la pour $k + 1$.
$X_{n+k+1} = aX_{n+k} + c$
$= a \times a^k X_n + \frac{a(a^k-1)c}{b} + c$
$= a^{k+1} X_n + \frac{a^{k+1}c - ac + ac - c}{b}$ (car $b = a - 1$)
$= a^{k+1} X_n + \frac{(a^{k+1}-1)c}{b}$

On a donc montré que $X_{n+k} \equiv a^k X_n + \frac{(a^k-1)c}{b} \pmod{m}$

On a donc l'équation $X_{0+n} = a^n X_0 + \frac{(a^n-1)c}{a-1} \pmod{m}$
Donc $X_n = \frac{(a^n-1)c}{a-1} \pmod{m}$ (1)

Si c n'est pas premier à m, X_n ne prend jamais la valeur 1, donc la condition (i) du théorème est nécessaire.

La période est de longueur m si et seulement si la plus petite valeur de n telle que $X_n = X_0 = 0$ est $n = m$.

Les points (ii) et (iii) sont démontrés directement à l'aide du lemme 2.3.2, ce qui conclus la démonstration du théorème.

□

On a donc montré que pour que le générateur ait une période maximale, il faut que les nombres magiques aient les propriétés données par le théorème 2.3.1.

2.3.1 Efficacité et test

Le choix des nombres magiques influe grandement sur l'efficacité du générateur. En effet, nous avons vu que dans le cas trivial oú $a = 0$, le générateur était de période maximale mais était constant. Le choix de a et c est longuement discuté dans The Art Of Computer Programming de Donald E. Knuth.
Nous avons aussi montré que il existait des générateur à congruence linéaire qui pouvaient avoir une période maximale. Cependant prendre une période maximale n'est pas toujours nécessaire selon l'utilisation du générateur et le choix du module.

Pour vérifier qu'un générateur est vraiment aléatoire, on lui fait subir un test spectral. Le test spectral est un test statistique qui ne peut s'appliquer qu'aux générateurs à congruence linéaire. Le test spectral teste l'uniformité des nombres produits par un générateur sur une période entière. Knuth a énoncé que dans la pratique un "mauvais" générateur (ie un générateur qui ne produisait pas du vrai aléa) ne passait jamais le test spectral.

Chapitre 3

Le Générateur présenté par le NIST

3.1 Introduction

3.1.1 Courbe elliptique

Soit K un corps.
Une courbe elliptique est un cas particulier de courbe algébrique qui peut être représentée dans un plan par une équation cubique de la forme :
$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_5$ avec $a_1, a_2, a_3, a_4, a_5 \in K$.
Dans le cadre de ce projet, nous nous intéresserons uniquement aux courbes elliptiques données par des équations dites de Weierstrass, c'est à dire des équations de la forme :
$y^2 = x^3 + ax + b$ avec $a, b \in K$.

Théorème 3.1.1. *Soit E une courbe elliptique sur un corps K.*
Alors $E(K)$ est un groupe abélien pour la loi d'addition géométrique de deux points sur une courbe elliptique.

Définition 3.1.1. *Soit E une courbe elliptique sur un corps K. Soit $G \in E(K)$.*
Connaissant $P \in E(K)$, trouver $n \in \mathbb{N}$, s'il existe, tel que $P = nG$, revient à résoudre le problème du logarithme discret.

En effet trouver le point G à partir du point $P = nG$ reviendrait à casser le cryptosystème de El-Gamal qui repose lui même sur le problème du logarithme discret.

La résolution de ce problème peut s'avérer plus facile pour certaines courbes elliptiques E mal choisies telles que les courbes supersingulières.

3.2 Le NIST et son générateur

3.2.1 Le NIST

Le NIST (pour National Institute of Standards and Technology) est une agence américaine ayant pour but de promouvoir l'économie en développant des technologies et la métrologie (science des mesures et des standards de concert avec l'industrie). Ce sont notamment eux qui sont à l'origine des normes pour les mesures du poids, de la longueur ou bien du volume au Etats Unis.
Ils sont aussi en charge des standards en cryptographie. Ils déterminent quels algorithmes peuvent être utilisés et quelles propriétés ils doivent vérifier. Par exemple, dans le cadre du générateur que nous étudions, il était conseillé de prendre p un nombre premier tel que p est supérieur à 10^{77}.
L'algorithme que nous étudions est appelé dual_ec_drbg dans la norme du NIST. Il a été ajouté au standard du NIST au début des années 2000.

Dan Shumow et Niels Ferguson ont publié en 2007 un article soupconnant l'existence d'une backdoor dans l'algorithme permettant à la NSA de deviner les propriétés aléatoires de l'algorithme. Cependant la démonstration de l'existence de la backdoor a été ignorer par l'ensemble des acteurs en cryptographie.
En 2013, le New York Times et le Guardian publient un article impliquant la NSA dans l'existence d'une faille de l'algorithme. Ils reprennent l'article de Shumow et Ferguson. De plus il ajoutent que les standards de l'algorithme ont été dicté par la NSA au NIST. La connaissance d'une faille par la NSA n'a cependant pas été prouvée... Nous allons donc présenter l'algorithme dans la suite de ce rapport et expliquer pourquoi il ne permet pas de générer du "vrai" aléa.

3.2.2 L'algorithme

Le standard du NIST donne une liste de sextuplés (E, p, n, f, P, Q) pour définir le générateur. E est la courbe elliptique sur $\mathbb{F}_p$ avec p premier définie par $E : \ y^2 = f(x)$. La taille de p diffère selon les versions de l'algorithme. Il y a 3 tailles possibles présentées par le NIST : 256, 384 et 512 bits.
On a $n = \#E(\mathbb{F}_p)$. Rien n'est indiqué pour n, cependant n est premier dans tous les exemples donnés par le NIST.
Et enfin P et Q sont deux points de la courbe elliptique, ie $P, Q \in E(\mathbb{F}_p)$.

Le NIST ne donne aucune explication sur les choix de (E, p, n, f, P, Q). On remarque cependant que $p > 10^{77}$. Soit $A \in E(\mathbb{F}_p), A = (x_a, y_a) x_a, y_a \in \mathbb{F}_p$.
On définit la fonction, x_1 :

$E(\mathbb{F}_p) \setminus 0 \longmapsto \mathbb{Z}$
$A \to x'_a$ un représentant de x_a modulo p.

Pour générer une suite de nombre aléatoire suivant le standard du NIST, on commence par choisir une graine $s \in \mathbb{N}$ qui devra rester secrète afin de protéger l'aléa.

Data: P, Q, s
Result: b suite de bit aléatoire
r=x_1(s*P);
s'=x_1(r*P);
t=x_1(r*Q);
b= extract_bit t;

Algorithm 1: Algorithme donné par le NIST

La fonction extract_bit retire les 16 bits de poids fort dans t et garde les autres. Pour continuer à générer des nombres aléatoires, on refait l'algorithme en se servant de s' comme nouveau secret.

Chapitre 4

La backdoor

Une backdoor est un secret permettant à un certain groupe de personne d'avoir accès à des informations qui ne sont pas censés être accessible.
Ce générateur a longtemps été utilisé pour générer de l'aléatoire mais en 2007 Dan Shumow et Niels Ferguson ont réussi à démontrer que l'on pouvait prédire l'aléa générer par cet algorithme. Ils ont démontré qu'avec seulement 32 bits récupérés en sortie de l'algorithme, il était possible d'identifier la graine qui nous sert à générer les nombres.

4.1 Présentation de la backdoor

Soit (E, p, n, f, P, Q) un générateur défini comme précédemment.
Etant donné que P et Q sont des éléments d'un groupe cyclique, $\exists e \in \mathbb{Z}$ tel que $P = e \times Q$. En déterminant un tel e il est facile de trouver la graine s nous permettant de générer les nombres. Un tel e représente donc la backdoor dans cet algorithme.

Supposons que l'on ait un tel e. Soit b le résultat de l'algorithme. Etant donné que la fonction extract_bit ne garde que les 16 bits de poids faibles, on peut stocker les 2^{16} pré-images t possibles avec b=extract_bits(t). Pour chaque t, il y a au plus deux abscisses possibles. Les calculer revient à résoudre une équation quadratique.

Nous venons donc de calculer au plus $2^{17}A$ tel que t=$x_1(A)$.
L'un de ces A vaut $A = r \times Q$.
Il nous reste plus qu'a calculer $e * A$ pour tous les A précédemment calculer.
L'un d'entre eux vaut :
$e \times (r \times Q) = r \times (e \times Q) = r \times P$.
Finalement, on peut calculer $s' = x_1(r \times P)$. On trouve le bon A en comparant ce que l'on obtient avec la sortie de l'algorithme. Shumow et Ferguson ont remarqué que en pratique, seulement 32 bits nous permettent de retrou-

ver la graine.

En supposant que l'on ait un e tel que $P = e \times Q$, on a l'algorithme suivant pour déterminer le secret s nous permettant de casser le générateur :

Data: e entier tel que $P = e \times Q$, o bloc de bit généré par le générateur que l'on veut attaquer
Result: S ensemble de valeur possible de la graine
S={};
for $0 \leq u \leq 2^{16} - 1$ **do**
  x=$u|o$;
  $z \equiv x^3 + ax + b \pmod{p}$;
  if $y \equiv z^{\frac{1}{2}} \pmod{p}$ *existe* **then**
    $A = (x, y)$ est sur la courbe elliptique.
    $S = S \bigcup x_1(e \times A)$
  end
end

Algorithm 2: Algorithme permettant de casser le générateur donné par le NIST

Pour programmer ces algorithmes, nous utilisons PARI-GP. Afin de savoir si le point (x, y) de l'algorithme appartient à la courbe elliptique, nous utilisons la fonction ellordinate de GP. Cette fonction est rapide car tester si un point appartient à une courbe elliptique revient à calculer un symbole de Legendre. La complexité de cet algorithme est donc en $O(2^{16} \times k)$ avec k le coût de un ellordinate.

Le NIST ne précisait pas comment les points P et Q étaient choisis. Mais, étant donné que le NIST est obligé de consulter la NSA lors de la conception de standard cryptographique, la NSA était au courant de la sélection des points. Le NIST choisissait un point P, choisissait un e entier et choisissait $Q = e \times P$. La NSA possedait exclusivement la backdoor.

4.2 En pratique

En effet, trouver un e tel que $P = e \times Q$ est extremement difficile étant donné que cela revient à résoudre le problème du logarithme discret.
Mais en partant de e, il n'est pas difficile de créer une paire (P, Q) tel que $P = e \times Q$. Pour prouver l'existence de la backdoor, Shumow et Ferguson ont injecté leur propre valeur de Q dans l'algorithme :

- Choisir aléatoirement d.
- Remplacer Q par $Q_2 = d \times P$
- On injecte Q_2 dans l'algorithme présenté par le NIST
- On trouve un ensemble de graine possible à l'aide l'algorithme présenté précédemment.

— On compare la sortie de l'algorithme du NIST modifié avec la sortie de l'algorithme ou l'on a injecter les différentes graines trouvées. Lorsque les sorties correspondent c'est que l'on a trouvé la bonne graine.

A l'aide de notre d, on arrive à calculer la graine qui est censé être secrète. On a donc la preuve de l'existence d'une backdoor.

4.3 Amélioration possible

Une première amélioration serait de tronquer plus de 16 bits dans le générateur. En garder que 8 rendrait beaucoup plus long le calcul des antécédents de la sortie du générateur.

Dans notre algorithme, nous calculons 2^{16} antécédents, ce qui se fait en temps raisonnable avec un ordinateur. Si l'on ne gardait que les 8 premiers bits générer, il faudrait calculer 2^{248} antécédents dans le cas oú la taille de p est de 256 bits. Le calcul serait alors beaucoup plus long.

Une manière beaucoup plus drastique d'opérer serait de ne garder que le premier ou les deux premiers bits générer. Cela ferait monter à 2^{254} ou 2^{255} le nombre d'antécédent à calculer.
Cette amélioration n'est cependant pas suffisante car l'augmentation de la puissance de calcul des ordinateurs permettrait de casser à nouveau l'algorithme.

Une deuxième amélioration serait de générer un nouveau Q pour chaque tour de boucle du générateur.
On ne pourrait plus avoir une unique relation $P = e \times Q$. Cela empecherait donc de trouver la graine.
Ce qui nous permet de trouver la graine au i^{eme} tours est l'équation :
$x_1(e \times A) = x_1(e \times r_i \times Q) = x_1(r_i \times P) = s_{i+1}$
Cependant si on remplace notre unique Q par un Q_i différent à chaque tour, l'équation n'est plus vrai. Si l'on connait l'existence d'un e tel que $P = e \times Q_i$, l'équation $P = e \times Q_{i+1}$ est fausse car $Q_i \neq Q_{i+1}$.
La connaissance d'une suite $e_1, \ldots, e_k, \ldots$ tel que $P = e_1 \times Q_1, \ldots P = e_k \times Q_k \ldots$ permet toujours de casser l'algorithme. Cependant si le choix, des Q_k est laissé à l'utilisateur, une entité extérieur ne peut pas casser l'algorithme.

Chapitre 5

Conclusion

La génération de nombre aléatoire est essentielle dans nombreuse application cryptographique par exemple dans l'échange de clé de Diffie Hellman. La sécurité des protocoles reposent sur le fait que ces nombres sont vraiment aléatoire et ne puissent pas être prédits à l'avance.

La backdoor du protocole a été dévoilé en 2007 lors d'une conférence. Dan Shumow et Niels Ferguson pensaient secouer le monde de la cryptographie avec ces révélations mais elles n'ont reçu qu'un accueil mitigé. Il a fallu attendre 6 ans que le New York Times et le Guardians révèlent que la NSA avait effectivement accès au secret de l'algorithme. La NSA n'a jamais avoué avoir eu connaissance de la backdoor, mais il est très difficile de croire une telle version étant donné que les meilleurs cryptographe au monde travaillent pour eux.
Les journalistes ayant révélés l'affaire, supposent que la NSA s'en est servi pour récupérer des clés échangées via l'algorithme de Diffie Hellman mais cela n'a jamais pu être prouver...

5.0.1 Remerciement

Nous tenons à remercier Monsieur BRASCA pour ses suggestions et ses remarques ainsi que ses réponses claires à nos questions.

5.0.2 Sources

— `https://jiggerwit.wordpress.com/2013/09/25/the-nsa-back-door-to-nist/`

— `http://rump2007.cr.yp.to/15-shumow.pdf`

— `http://csrc.nist.gov/publications/nistpubs/800-90A/SP800-90A.pdf`

— *Art Of Computer Programming Volume 2* de Donald E. Knuth