N°=d'ordre : 08/PGS-2004/MT

République Algérienne démocratique et populaire

Université des sciences et de la technologie Houari Boumediene

Faculté des mathématique

Mémoire de poste-Graduation spécialisée en mathématiques

Spécialité : Cryptologie

Présenté par :

Monsieur : Mameri Ahmed

Sujet :

GENERATEURS DE NOMBRES ALEATOIRES

Soutenu le 25/03/2004

Devant le jury composé de:

M^{r} : HACHAICHI Mohamed. S, Maître de conférences, USTHB Président

M^{r} : AISSANI Amar, Professeur, USTHB Directeur de Mémoire

M^{r} : BENTINA Kamel, professeur, USTHB Examinateur

M^{r} : M. A. ABSI, DGSCT. Examinateur

Table des matières

Notations

* **Z/nZ** est noté $\mathbf{Z_n}$ pour alléger la typographie.
* #E designe le nombre d'éléments de l'ensemble E.
* Le logarithme en base a est noté $\log_a$.
* $\lfloor x \rfloor$ désigne la partie entière par défaut du reel x.
* Par(x) désigne la parité de l'entier x.

Introduction :

L'imitation du hasard trouve de nombreuses applications en science. C'est le cas de la simulation, des algorithmes probabilistes, de l'analyse numérique, des jeux de traitement de signal, de la cryptographie, etc.… Et c'est pourquoi mathématiciens et informaticiens ont depuis longtemps unis leurs forces pour résoudre le problème de créer un « générateur de nombres aléatoires » fiable, c'est-à-dire totalement imprévisible !

C'est moins facile qu'on ne pourrait le croire, car un ordinateur n'improvise pas : Il ne sait que suivre son programme, il lui est donc difficile de produire a partir d'une procédure purement mathématique, des chiffres réellement aléatoires de façon totalement imprévisible. Si on connaît la procédure mathématique, on peut obtenir la même suite de nombres, ce n'est donc pas du hasard. Il faut se contenter de produire des séquences de nombres qui ont toutes les apparences de hasard. Dans la pratique, on se contente habituellement de nombres « pseudo aléatoires » générés a partir d'une variable difficile à reproduire.

Le but de ce travail est de présenter les méthodes de conception d' algorithmes de génération de suites binaires . Pour cela, la première partie est consacrée à quelques notions de théorie des nombres aux quelles on fera appel dans la suite de ce travail. La deuxième partie traite de la cryptologie traditionnelle et moderne, en particulier les différentes méthodes de chiffrement. La troisième partie est consacrée aux déférents types de suites de nombres aléatoires : Pseudo aléatoires, pseudo aléatoires cryptographiquement sûres et vraiment aléatoires particulièrement le générateur BBS. La quatrième et dernière partie discute les critères de validation de tels générateurs, en particulier les standards FIPS et NIST.

2. Notions de théorie des nombres

Cette partie rassemble quelques notions de théorie des nombres auxquelles on fera appel par la suite. Il s'agit de l'arithmétique des congruences : algorithme d'Euclide, indicateur d'Euler, petit théorème de Fermat, élément primitif, résidu quadratique, loi de réciprocité. Certaines familiarités avec ces notions sont nécessaires pour aborder les idées principales utilisées en cryptographie.

2.1 Congruences et algorithme d'Euclide

2.1.1 Congruences

On dit que deux entiers a et b sont congrus modulo un entier n strictement positif si $a - b$ est un multiple de n.

On écrit : $a \equiv b \bmod n$.

La relation $\equiv$ (modulo n) est une relation d'équivalence sur l'ensemble Z des entiers. L'ensemble des classes d'équivalences appelé « ensemble des classes de congruence modulo n » où de manière abrégé « ensemble des nombre modulo n »est noté Z_n. Dans la pratique (bien que cela prête parfois à confusion) on notera de la même manière un nombre et la classe de congruence qu'il représente. L'ensemble

$Z_n = \{0,1, \ldots, n-1\}$ est naturellement muni d'une addition et d'une multiplication (les lois quotient) d'éléments neutres 0 et 1. Ces deux opérations confèrent à Z_n une structure d'anneau. En particulier $(Z_n, +)$ est un groupe. Pour que la multiplication permette de définir, elle aussi, une structure de groupe, il faut ne conserver que les éléments inversibles de Z_n, on note leur ensemble Z^*_n.

Si k est inversible modulo n et si a tel que $ka \equiv 0 \bmod n$, alors : $k^{-1}ka \equiv a \equiv 0 \bmod n$, i.e. a est un multiple de n. On en déduit qu' un nombre k ne peut être à la fois inversible modulo n et avoir un diviseur non trivial commun avec n, car il serait diviseur de 0 modulo n.

Pour établir la réciproque, à savoir qu'un nombre premier avec n est inversible modulo n et pour calculer son inverse, nous avons besoin de l'algorithme d'Euclide.

2.1.2 Algorithme d'Euclide

Soient deux entiers strictement positifs, n et k, $k < n$, dont on souhaite calculer le pgcd. L'algorithme consiste en la suite de divisions euclidienne suivantes :

$$n = kq_0 + r_0$$
$$k = r_0 q_1 + r_1$$
$$r_0 = r_1 q_2 + r_2$$
$$\ldots r_{m-1} = r_m q_{m-1} + r_{m+1}.$$

Où $r_i < r_{i-1}$. L'algorithme se termine lorsque le dernier reste est nul ($r_{m+1} = 0$) et le plus grand commun diviseur de n et k est l'avant dernier reste r_m.

2.1.3 Algorithme d'Euclide étendu

Cet algorithme détermine, en plus du pgcd de n et k, deux entiers u et v tels que :

$$u\,n + v\,k = d$$

Nous appelons égalités Euclidiennes les égalités intervenant dans l'algorithme d'Euclide. la première d'entre elles peut se réécrire sous la forme $n - kq_0 = r_0$. Multiplions par q_1 et retranchons à la deuxième égalité Euclidienne, nous obtenons une égalité de la forme $a_1 n + b_1 k = r_1$. Multiplions par q_2 et retranchons à la troisième égalité Euclidienne, nous obtenons $a_2 n + b_2 k = r_2$. De proche en proche, nous obtenons $a_m n + b_m k = r_m$. C'est l'égalité recherchée.

Nous somme maintenant en mesure d'affirmer la proposition suivante :

k est inversible modulo n si et seulement si k est premier avec n.

2.2 La fonction indicatrice d'Euler

Pour tout entier strictement positif n, on note traditionnellement $\varphi(n)$ le nombre d'entiers strictement positifs inférieurs à n et premiers avec n. Ce nombre mesure donc la cardinalité de l'ensemble des éléments inversibles de Z_n ; soit $\varphi(n) = \# Z^*_n$. Il mesure aussi le nombres de générateurs du groupe $(Z_n, +)$: en effet un élément m de ce groupe en est générateur si et seulement s'il existe k tel que $km \equiv 1 \ mod\ n$, ce qui veut exactement dire que m admet un inverse multiplicatif. La fonction φ vérifie des propriétés qui nous seront fort utiles.

Théorème (Fermat-Euler). Si a est premier avec n, alors :

$$a^{\varphi(n)} \equiv 1 \bmod n.$$

Corollaire (Fermat). Si n est premier et $1 \leq a \leq n-1$ alors :

$$a^{n-1} \equiv 1 \bmod n.$$

2.3 Théorème chinois

Théorème (Théorème chinois). Si p et q sont deux entiers premiers entre eux, alors les structures d'anneaux Z_{pq} et $Z_p \times Z_q$ sont isomorphes.

Corollaire Si p et q sont premiers entre eux alors :

$$\varphi(pq) = \varphi(p)\,\varphi(q).$$

Si p est premier, on établi facilement que $\varphi(p^{\alpha}) = p^{\alpha - 1}(p-1)$.

Corollaire Si $n = p_1^{m_1}.p_2^{m_2}...p_k^{m_k}$ est la décomposition en facteurs premiers de n, alors :

$$\varphi(n) = (p_1^{m_1} - p_1^{m_1-1}).(p_2^{m_2} - p_2^{m_2-1})...(p_k^{m_k} - p_k^{m_k-1})$$

Remarquons que cette formule de calcul de $\varphi(n)$ fait appel à la décomposition en facteurs premiers de n.

2.4 Eléments primitifs

Lorsque p est un nombre premier la structure de $(Z^*_p, \times)$ est particulièrement simple.

Théorème Si p est premier, $(Z^*_p, \times)$ est cyclique.

Un générateur de $(Z^*_p, \times)$ est appelé élément primitif .

2.5 Résidus quadratiques

On dit que $x \in Z_n^*$ est un résidu quadratique modulo n si c'est un carré de Z_n^*, i.e. $x = y^2$ pour un y de Z_n^*.

2.5.1 symbole de Legendre

Pour un nombre p premier, et un entier x premier avec p,que l'on assimilera à un élément de Z_p^*, on définit le symbole de Legendre :

$$\left(\frac{x}{p}\right) = \begin{cases} 1 & \text{si } x \text{ est un résidu quadratique modulo } n \\ -1 & \text{si } x \text{ n'est pas un résidu quadratique modulo } n \end{cases}$$

Théorème (Euler) : Si *p* est un nombre premier impair, alors pour tout nombre *x* premier avec *p :*

$$x^{\frac{p-1}{2}} \equiv \left(\frac{x}{p}\right) \bmod p$$

Conséquence : L'application $x \longrightarrow \left(\frac{x}{p}\right)$ est un morphisme du groupe Z_p^* sur le groupe {-1, 1}.

2.5.2 Symbole de Jacobi

Pour un entier *n* supérieur à 2 quelconque,on définit le symbole de Legendre étendu, encore appelé symbole de Jacobi, par :

$$\left(\frac{x}{n}\right) = \left(\frac{x}{p_1}\right)^{m_1}\left(\frac{x}{p_2}\right)^{m_2}\ldots\left(\frac{x}{p_k}\right)^{m_k}$$

Où x est un élément de Z_n^*, et où $n = p_1^{m_1} p_2^{m_2} \ldots p_k^{m_k}$ est la décomposition en facteurs premiers de *n.*

Notons que l'application $x \rightarrow \left(\frac{x}{n}\right)$ est encore un morphisme de groupe, de Z_n^* sur {-1, 1}.

Remarque : lorsque *n* n'est pas premier, il n'est plus vrai que *x* est résidu quadratique modulo *n* si et seulement si $\left(\frac{x}{n}\right) = 1$. On peut s'en convaincre aisément à l'aide du théorème Chinois.

Nous verrons que l'un des intérêts pour la cryptographie du symbole de Jacobi $\left(\frac{x}{n}\right)$ est qu'il peut être calculé facilement sans qu'il soit nécessaire de connaître la factorisation de *n.* Ceci est une conséquence de la fameuse loi de réciprocité quadratique.

2.5.3 Calcul du symbole de Jacobi (loi de réciprocité)

Montrons tout d'abord comment se calculent $\left(\frac{-1}{n}\right)$ et $\left(\frac{2}{n}\right)$, sans utiliser la factorisation de *n*.

Lemme Pour tout entier impair *n,* on a :

$$\left(\frac{-1}{n}\right) = (-1)^{\frac{n-1}{2}}$$

$$\left(\frac{2}{n}\right) = (-1)^{\frac{n^2-1}{8}}$$

Théorème (loi de réciprocité quadratique). Si *m* et *n* sont premiers entre eux, impairs et strictement supérieurs à 2,alors :

$$\left(\frac{m}{n}\right)\left(\frac{n}{m}\right) = (-1)^{\frac{(n-1)(m-1)}{4}}$$

Remarque : le lemme précédent et la loi de réciprocité permettent de calculer les symboles de Jacobi par réduction successives, de manière analogue à l'algorithme d'Euclide.

Soit a calculer par exemple $(\frac{31}{91})$:

On peut le faire de la manière suivante : $(\frac{31}{91}) = -(\frac{91}{31}) = -(\frac{29}{31}) = -(\frac{31}{29}) = -(\frac{2}{29}) = 1$.

3 Introduction générale à la Cryptologie

La cryptologie se compose de deux activités opposées, complémentaires et étroitement liées :(a) La cryptographie qui cherche à construire des système cryptographiques, ou cryptosystémes, garantissant une certaine sécurité à un ensemble d'informations, en les transformant par une opération de chiffrement ou de signature. (b) La cryptanalyse qui cherche à déjouer de tels systèmes. Tout cryptographe cherche à imaginer toutes les attaques possibles susceptibles de casser son nouveau système.

L'art de la cryptographie date de plusieurs siècles, et son utilisation à surtout été militaire ou diplomatique, beaucoup plus rarement commerciale ou privée. Aujourd'hui cependant, en raison de la multiplication des calculateurs électroniques, des réseaux qui les relient et des données en tout genre qui y transitent, en raison de la facilité et de la modicité du coût des communications même les plus lointaines, les besoins de sécurité se font de plus en plus sentir à tous les niveaux, et c'est pourquoi la cryptologie a connu un essor considérable.

3.1 Cryptographie à clé secrète

La cryptographie traite de la transmission confidentielle de données . C'est l'étude de méthodes permettant de transmettre des messages sous forme déguisée, de telle sorte que seuls les destinataires autorisés soient capables de les lire. Le message à envoyer est appelé message ou texte en clair et sous sa forme déguisée, message chiffré (même s'il n'est pas représenté sous forme de chiffres), ou cryptogramme. Une fonction cryptographique, ou de chiffrement, est donc la donnée d'une transformation, en générale bijective.

$$f : M \rightarrow C$$

où M représente l'ensemble des messages en clair, et C l'ensemble des messages chiffrés. La transformation f^{-1} est la transformation de déchiffrement.

L'histoire a montré que chaque fois qu'une fonction cryptografique f est destinée à être utilisée un nombre important de fois, il devient de plus en plus difficile de la maintenir complètement secrète. Il est donc souhaitable de pouvoir changer régulièrement de fonction f . A' cette fin ; on définit un système cryptografique, ou de chiffrement, ou encore un chiffre comme étant une famille finie

$$F = (f_k)_{k \in K}$$

de fonctions cryptographiques chacune étant paramétrée par un paramètre k, appelé clé.

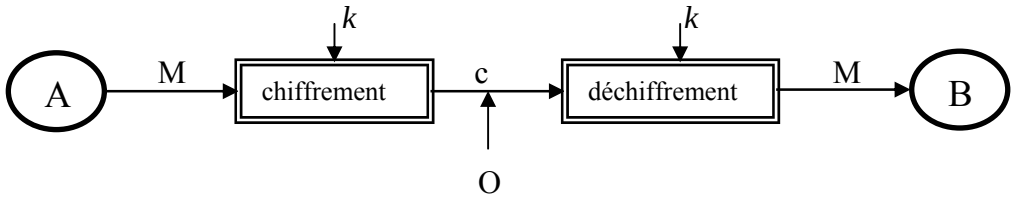


Figure. 1 .

La figure. 1 .illustre le contexte type. Deux entités, expéditeur et destinataire que l'on appelle Alice et Bob, c'est la coutume, communiquant en présence d'un observateur ou cryptanalyste Oscar. Le but du cryptanalyste est de décrypter le cryptogramme transmis C, c'est-à-dire d'en déduire le message émis M. Idéalement, il souhaitera trouver la clé k et la transformation f_k.

Si le système cryptographique F est utilisé sur une grande échelle, il n'est pas raisonnable de le considérer comme complètement secret. Pour cette raison on supposera qu'Oscar connaît entièrement le système de chiffrement F, il ne sait pas, par contre, laquelle des transformations f_k est utilisée. En d'autres termes il lui manque juste la clé secrète k :c'est le principe de Kerckhoffs (1835-1903). Si le système est suffisamment bien conçu, le secret de la seule clé k suffit à assurer la confidentialité d'un message.

3.1.1 Exemples historiques

1- Chiffrement par décalage, ou de Jules César. Le message chiffré se déduit du message en clair par un décalage circulaire des lettres de l'alphabet. On peut considérer que l'ensemble des messages chiffrés C est l'alphabet latin. L'ensemble des clés est :

$$K = \{0,1,\ldots 25\}$$

Si l'on utilise un décalage de quatre lettres de l'alphabet, le texte en clair.

« Tests statistiques » Se transforme en le texte chiffré. « Xiwxw wxexmwxmuyiw »

Suétone rapporte que Jules césar utilisait systématiquement la clé $k = 3$ dans sa correspondance avec ses proches. Notons que le principe de Kerckhoffs n'est pas respecté dans ce cas.

2- Chiffrement par substitution : L'ensemble des messages en clair et l'ensemble des messages chiffrés sont les mêmes que précédemment ; mais on augmente l'ensemble des clés. Plutôt que de se restreindre aux décalages circulaires on autorise toute permutation de l'alphabet. Le nombre de clés possibles égale donc $\# K = 26!$.
Si la clé est k

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

Alors le texte en clair :

« tests statistiques »

se transforme en le texte chiffré

« cixcx xcjcyxcymqix »

3- Chiffrement par permutation : (la terminologie cryptographique traditionnelle est « transposition »). Dans ce cas on ne modifié pas les symboles du texte en clair, mais on les permute, le texte en clair est d'abord découpé en blocs de n symboles appartenant à Σ. Par exemple, Σ peut être l'alphabet latin augmenté du symbole blanc. On aura $M = C = \Sigma^n$ et chaque clé k est une permutation de $\{1,2,\ldots n\}$. On chiffre par la transformation :

$$M \longrightarrow C$$

$$(m_1, m_2, \ldots m_n) \longrightarrow (m_{k(1)}, m_{k(2)}, \ldots m_{k(n)})$$

soit $n = 5$ et la clé $k = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 5 & 2 & 1 & 4 \end{pmatrix}$. le texte en clair :

« tests statistiques +bbb »

se transforme en texte chiffré : « ssett aitstiutsqbbseb »

Remarque : Notons dés maintenant que dans ces systèmes la connaissance de la clé k est censée rendre facile le calcul, tant de la fonction de chiffrement f,que la transformation de déchiffrement f^{-1}. C'est une caractéristique des systèmes cryptographiques traditionnels où à clé secrète :en parle encore de déchiffrement symétrique.

Les systèmes ci-dessus sont bien peu résistants, le chiffrement par décalage utilise un ensemble de clés trop petit. Le cryptanalyste n'a qu'à essayer successivement toutes les clés possibles pour retrouver le message en clair à partir du message chiffré. Le chiffrement par substitution d'un texte écrit dans une langue naturelle ne résiste pas à une analyse de fréquences. Par exemple en « Français », la lettre la plus fréquente est « E ».

Retenons que pour rendre la tâche du cryptanalyste plus difficile, il est important de concevoir des systèmes tels que le texte chiffré ait un aspect aléatoire, même si le message en clair correspondant ne l'est pas.

3.1.2 Le système de Vernam, ou « one-time pad »

Le système de Vernam (1926), ou « one-timepad », fonctionne de la manière suivante. Les messages, tant en clair que chiffrés, s'écrivent dans un même alphabet Σ,que l'on représente par Z_m (ou $m = \#\Sigma$). Un message de n symboles $m = \{x_1, x_2, \ldots x_n\}$, se chiffre par la transformation f_k.

$$f_k : M = Z^n_m \longrightarrow C = Z^n_m$$

$$(x_1, x_2, \ldots x_n) \longrightarrow (y_1, y_2, \ldots y_n)$$

Avec :

$$y_i = (x_i + k_i) \bmod n$$

Le n-uple $k = (k_1, k_2, \ldots k_n)$ constitue la clé de la transformation et il est choisi aléatoirement dans Z^n_m. En particulier, cela veut dire que si un nouveau message est envoyé, un nouveau n-uple k est utilisé comme clé et que pour chaque message, émetteur et récepteur partagent autant de symboles de clé que de symboles qu'il souhaite se transmettre : (un n-upelé k n'est jamais sciemment réutilisé, d'où l'appellation « one-time ». Ce système peut être considéré comme l'aboutissement de la cryptographie traditionnelle.

3.1.3 Le chiffrement à clé secrète : point de vue moderne

Le « one-time pad » assure une sécurité inconditionnelle, car il ne préjuge pas de la puissance de calcul du cryptanalyste qui peut être illimitée. A partir de là une notion de sécurité calculatoire apparaît. En effet, une sécurité inconditionnelle impose de manier des clés très longues, ce qui n'est pas praticable pour la plupart des applications. On utilisera donc des clés courtes et le cryptanalyste aura en théorie tous les éléments pour clé ou message en clair. Cependant, s'il n'a pas la puissance de calcul pour le faire, il n'aboutira pas.

Les algorithmes à clé secrète peuvent être classés en deux catégories. Certains opèrent sur le message en clair un bit à la fois, ceux-ci sont appelés algorithmes de chiffrement en continu ou par flot qui sont directement issus du système de Vernam. L'idée est tout simplement de remplacer la suite aléatoire constituant la clé par une suite pseudo-aléatoire, engendrée de manière déterministe par une clé courte , et de chiffrement par blocs qui agit globalement sur des ensembles de *n* symboles, en général des bits. Autrement dit, la fonction de chiffrement est de la forme :

$$f_k : \{0,1\}^n \longrightarrow \{0,1\}^n$$

Le paramètre *k* représente la clé secrète. On peut citer comme exemples d'algorithmes de chiffrement par blocs le DES [1].

3.2 Cryptographie à clé publique

Les algorithmes de chiffrement à clé publique sont conçus de telle manière que : La clé de chiffrement soit différente de la clé de déchiffrement. De plus, la clé de déchiffrement ne peut pas être calculée (du moins en un temps raisonnable) à partir de la clé de chiffrement. De tels algorithmes sont appelés « à clé publique » parce que la clé de chiffrement peut être rendue publique : n'importe qui peut utiliser la clé de chiffrement pour chiffrer un message mais seul celui qui possède la clé de déchiffrement peut déchiffré le message chiffré résultant. Dans de tels systèmes, la clé de chiffrement est appelée clé publique et la clé de déchiffrement est appelée clé privée (ou secrète). Par fois, les messages seront chiffrés avec la clé privée et déchiffrés avec la clé publique, une telle technique est utilisée pour signatures numériques. On peut citer comme exemple l'algorithme de chiffrement et signature RSA [16]

4 Générateurs algorithmiques et générateurs physiques

Considérons un système de chiffrement en continu. L'ensemble M des messages en clair est infini et consiste en un ensemble de suite finies sur un alphabet $\sum$ que l'on prendra ici égal à $\{0,1\}$, on note $M=\{0,1\}^*$. L'ensemble des clés K est fini et chaque clé permet d'engendrer une suite aléatoire (ou suite pseudo aléatoire) $a=(a_i)$. la fonction de chiffrement associée transforme un message $M=(m_1,m_2,\ldots,m_n)$ en un cryptogramme $C=(c_1,c_2,\ldots,c_n)$ où $c_i=(m_i+a_i)\bmod 2$.

On ne réutilise pas les bits de (a_i), c'est-à-dire que lorsqu'on envoie un nouveau message $(m_{n+1},m_{n+2},\ldots m_{n+p})$, on utilise les bits $a_{n+1},a_{n+2},\ldots,a_{n+p}$ de la suite a.

Le problème de la sécurité des systèmes de chiffrement en continu est un peu plus facile à formuler que celui des systèmes de chiffrement par bloc , car tout dépend de la qualité cryptographique du générateur pseudo aléatoire(ou aléatoire), c'est-à-dire de la fonction

$$K \rightarrow \{0,1\}^N$$
$$k \rightarrow a=(a_i)$$

supposons en effet que le cryptanalyste réalise une attaque à couple clair-chiffre connu , c'est-à-dire qu'il parvient à obtenir certains bits du message en clair, disons $m_1,m_2,\ldots,m_n$ ainsi que les bits correspondants $c_i=(m_i+a_i)\bmod 2$ du cryptogramme.

Il découvre donc les bits $a_1,a_2,\ldots,a_n$ de la suite a. Son problème est d'en déduire les bits suivants $a_{n+1},a_{n+2},\ldots$ et si possible de déterminer la suite a toute entière.

Un système de chiffrement en continu est donc bon s'il est associé à des suites pseudo aléatoires (ou aléatoires) difficiles à reconstituer à partir d'un morceau. Penchons nous maintenant sur le problème de la génération de « bonnes » suites pseudo aléatoires (ou aléatoires).

4.1 Suites aléatoires

Maintenant nous entrons dans le domaine des philosophes. Existe-il quelque chose d'aléatoire ?, qu'est ce que une suite aléatoire ?, comment savoir si une suite est aléatoire ? est ce que « 1101110100 » est plus aléatoire que « 1010101010 » ?. La mécanique quantique nous apprend des phénomènes physiques que l'on peut qualifier d'aléatoire dans le monde réel. Mais est ce que ce caractère aléatoire est préservé quand on le transpose dans le monde macroscopique des puces informatiques et des machines à état fini ? toute philosophie mise à part, de notre point de vue un générateur de suite est

vraiment aléatoire s'il jouit de la propriété suivante : Il ne peut pas être reproduit de manière fiable .

Si on exécute le générateur de suites deux fois avec exactement les mêmes entrées (du moins des entrées aussi identiques que possible) vous obtenez deux suites aléatoires différentes. De plus , des vraies suites aléatoires ne peuvent pas être comprimées.

4.2 Suite de nombres pseudo aléatoires

Fabriqué d'une manière déterministe, une suite aléatoire de bits ou plus généralement, de symboles appartenant à un même alphabet Σ, représente une contradiction dans les termes. Comment définir donc une suite pseudo aléatoire ? Informellement, une suite *a* est pseudo aléatoire si elle est produite par un algorithme et s'il est algorithmiquement difficile de prévoir avec une probabilité supérieure à $\frac{1}{2}$ le bit a_{n+1} à partir des *n* premiers bits précédents $a_1,a_2,\ldots,a_n$ (c'est-à-dire imprévisible).

Cette condition assez vague à été formalisée, en particulier par Yao (1982),dans le cadre de la théorie de la complexité.

4.3 Générateurs physiques

L'exemple le plus simple, connu de tous ceux qui sont familiers du calcul des probabilités, est le jeu de « pile » ou « face » permettant de générer une variable aléatoire de Bernoulli qui prend les valeurs 0 ou 1 avec des probabilités identiques, égales à $\frac{1}{2}$, le jet d'un dé « parfait » permet de générer une variable aléatoire à valeurs dans $\{1,2,3,4,5,6\}$ avec des probabilités identiques, égales à $\frac{1}{6}$.La suite de nombres au hasard peut être obtenue également à l'aide d'une roulette telle qu'on peut en voire dans les fêtes foraines, on peut utilis également des dispositifs mécaniques plus sophistiqués tels que la machine servant au tirage de loto ou de kora^{+}. Ainsi pour générer une variable binaire, on a longtemps utilisé des sources de particules radioactives, un compteur dénombre les particules détectées durant un temps Δt. On considère alors la variable binaire y = 1 si le nombre de particules détectées durant le temps Δt est pair, y = 0 si ce nombre est impair. Un autre procédé utilise le niveau de bruit d'une valve électronique. La valeur du voltage *u(t)* est un processus aléatoire qu'on observe à des temps discrets $t_1,t_2,\ldots,t_i\ldots$On choisit un niveau de section *c* de manière adéquate, par exemple tel que $p(u_i=1)$ soit le plus proche possible de ½ et on pose :

$$u_i = \begin{cases} 0 & \text{si } u(t_i) < c \\ 1 & \text{si } u(t_i) \geq c \end{cases}$$

La logique de création de u_i peut être plus complexe afin de réduire les biais ou garantir d'autres propriétés statistiques de la loi uniforme U(0,1).

4.4 Générateurs algorithmiques

Les suites produites par des procèdes physiques ont été longtemps (elles sont toujours) acceptées comme aléatoires. Ce pendant ces procèdes sont coûteux et les suites obtenues ainsi présentent des biais et dépendances . Bien que produisant également des nombres pseudo aléatoires , les générateurs algorithmiques ont l'avantage d'être simples à réaliser sur ordinateur. C'est en ce sens que la simulation a été l'une des toutes premières applications des premières générations d'ordinateurs dés 1940. Il existe une multitude de générateurs algorithmiques de nombres au hasard ; certains sont mieux testés que d'autres. La complexité d'un générateur ne conduit pas forcement à une suite plus « aléatoire » que celle d'un générateur plus simple . on conseille d'ailleurs en général d'utiliser un algorithme simple et bien assimilé.

Lorsqu'on génère une suite $\{U_i\}$ par un procède algorithmique ; on doit avant tout s'assurer que l'algorithme choisi possède une bonne forme mathématique simple et facilement programmable sur ordinateur. L'algorithme doit donc être récursif :

$$u_{i+1}=f(u_i,u_{i-1},\ldots,u_{i-k}) \quad (k<i)\ i=1,2,\ldots \quad \textbf{(1)}$$

La forme la plus simple est :

$$u_{i+1}=f(u_i),\ i=1,2,\ldots \quad \textbf{(2)}$$

où *f* et une fonction donnée à valeurs dans [0, 1], qui peut avoir une forme plus ou moins « chaotique » . La suite générée à l'aide de l'algorithme (1) ou (2) sera dans tous les cas pseudo aléatoire car produite par un moyen déterministe. Les algorithmes récurrents possèdent un autre inconvénient qui réside dans le fait que la suite $\{u_i\}$sera toujours périodique. C'est pourquoi, l'utilisation aléatoire d'une telle suite ne pourra se faire que sur une succession d'épreuves inférieures à la période. La question qui se pose alors est de savoir déterminer une période suffisamment grande pour garantir un nombre raisonnable de termes de la suite. Malgré les inconvénients cités, les nombres pseudo aléatoire obtenus a partir de procédés récurrents sont très commodes en simulation :

-L'algorithme est facilement programmable et nécessite un faible espace mémoire.

-Toute suite de nombres peut être répétée autant de fois qu'on le désire, et a partir de n'importe quel rang, il suffit pour cela de donner un nombre initial u_0.

-Une telle suite de nombres peut être testée sur l'uniformité une seule fois afin de pouvoir l'utiliser avec un niveau acceptable de confiance.

-Rapidité pour garantir une utilisation multiple des nombres générés.

La plus part des langages informatiques et logiciels sont dotés d'un sous-programme de génération de suites aléatoires uniforme sur [0,1].

4.4.1 Exemples de générateurs algorithmiques

4.4.1.1 Générateur de Fibonaci : Il est de la forme

$$y_{i+1} = y_i + y_{i-1} \bmod n .$$

Ce générateur est juste cité ici pour l'histoire, car il a été montré qu'il ne produisait pas de nombres pseudo aléatoires satisfaisant, notamment en cryptographie.

-Il faut un espace mémoire plus large pour stocker les *j* nombres.

-Pour *j* petit, les nombre consécutifs obtenus sont fortement corrélés.

4.4.1.2 Générateurs congruentiels(GCL) : C'est en fait toute une famille de générateurs définis par

$$y_{i+1} = (\lambda \, y_i + c) \bmod m$$

où λ, c et m sont des entiers. Ce procedé signifie que y_{i+1} et le reste de la division de $\lambda \; y_i + c$ par *m*, où *m* est un nombre entier de grande taille(généralement une puissance de 2 pour les machines binaires), λ est nombre entier compris entre 0 et $m-1$, y_0 est appelée la racine (où le germe) du générateur. En guise de nombres pseudo aléatoires, on utilise les nombres $u_i = \frac{y_i}{m}$ qui sont équidistribués sur [0,1] lorsque $i \to \infty$ pour c = 0, on obtient le générateur multiplicatif (de Lehmer)qui imite bien l'effet d'une roulette, et pour $c \neq 0$ on obtient le générateur mixte. A titre d'exemple les bibliothèques FORTRAN NAG ET FORTRAN IMSL utilisent l'algorithme précèdent avec les valeurs respectives :

$$\lambda = 13^{13}, m = 2^{59} \approx 5{,}8.10^{17}, c = 0 \text{ et} \lambda = 397204094, \mathrm{m} = 2^{31} - 1, \mathrm{c} = 0$$

Si on désire produire toujours la même séquence (ce qui est pratique à des fins des tests),on rentre toujours la même valeur de y_0. Si en préfère que la séquence soit toujours différente, on initialise y_0 avec une grandeur toujours différente,l'heure système par exemple.

Dans tous les cas,les nombres de la suite sont compris entre 0 et *m*-1.

Si on choisi *c* no nul, il est toujours possible d'obtenir une période de longueur *m* (donc pas de risque de blocage puisqu'on va retrouver tous les nombres entre 0 et *m*-1 dans la suite), D.Knuth [3, 12] fait la démonstration des critères que doivent remplir λ , *c* et *m* pour cela :

1- *c* et *m* doivent être premiers entre eux.

2- λ -1 doit être un multiple de *p*, pour tout *p* nombre premier diviseur de *m*

3- λ -1 doit être un multiple de 4 si *m* et un multiple de 4.

4- si *m* est une puissance de 2, le bit de poids faible des nombres produits alternativement 0 et 1.

Si *c* = 0 la situation est plus compliquée parce qu'il reste toujours la possibilité de blocage (si y_0 =0), donc la période du générateur ne peut être au mieux que *m*-1, l'analyse en général est assez difficile, pour $m = 2^n$ avec $n > 3$ on peut dire que :

- λ mod 8 doit être égale à 3 ou 4.
- La période de générateur est 2^{n-2} et tous les bits de poids faible des nombres produits sont identique (il faut donc décaler le nombre produit de 1 vers la droite)

Exemple : $x_{n+1}=(31415821\ x_n + 1) \bmod 100000000$

On a bien respecté les critères de Knuth.

- 1 et 100000000 sont bien premiers entre eux. Par conséquent, on est certain qu'il n'y aura pas de problème de blocage, ou de série de nombres tous pairs ou impairs. Tous les nombres entre 0 et 99999999 vont sortir et chacun une fois tous les 100000000 tirages.
- 31415821 - 1 = 31415820 est bien un multiple de 2 et 5 (les seuls nombres premiers qui divisent 100000000).
- C'est aussi un multiple de 4 (car 100000000 est lui-même un multiple de 4).

Regardons ce que ça donne, par exemple pour $x_0 = 0$.

Les nombres produits sont :

1, 3145822, 40519863, 62952524, 25482205, 90965306, 70506227, 6817368, 12779129, 29199910, 45776111, 9252132, 22780373, 20481234, 81203115, …

Avoir la certitude que tous les nombres vont sortir ne suffit pas à garantir qu'ils seront suffisamment aléatoires !

Quelques générateurs

Park et Muller ont propose un générateur standard convenablement testé et dont le code est portable sur toutes les machines, ils l'ont appelé le standard minimal, il est défini par :

$$x_{n+1}= 16807\ x_n \bmod(2^{31}-1)$$

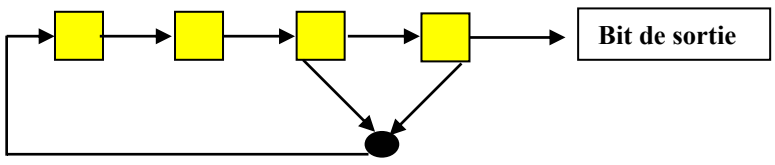
D.Knuth recense [3] un certain nombres de générateurs congruentiels linéaires de qualité. S'il n'est pas possible d'utiliser le standard minimal, on pourra essayer par exemple :

- $x_{n+1} = (1664525\ x_n + 1013904223) \bmod 2^{32}$ générateur de Knuth et Lewis.
- $x_{n+1} = 69069 \bmod 2^{32}$ générateur de Marsaglia au multiplicande remarquable, utilisé sur le Vax.
- $x_{n+1}= (31167285\ x_n + 1) \bmod 2^{48}$ générateur de Lavaux et Jenssens.
- $x_{n+1}= (6364136223846793005\ x_n + 1) \bmod 2^{64}$ générateur Haynes.

L'avantage des générateurs linéaires congruentiels est qu'ils sont rapides, ne nécessitant que peu d'opérations par bit. Malheuresement, les générateurs linéaires congruentiels ne sont pas intéressant pour la cryptographie, car ils sont plus facilement prévisibles. Les générateurs linéaires congruentiels ont été cassés pour la première fois par Joan Boyar[13,14,15].

4.4.1.3 Les registres à décalage à rétroaction linéaire (RDRL)

Cette méthode pour générer des nombres pseudo aléatoires est basée sur les registres à décalage à rétroaction linéaire. L'idée est de commencer avec un registre rempli arbitrairement avec des 0 et des 1, puis de décaler cette suite d'un cran vers la droite. On remplira la position la plus à gauche par la somme modulo 2 du contenu des deux registres les plus à droite (avant le décalage) comme l'indique le schéma ci-dessous :



Un RDRL a 4 bits

Remarque : On peut imaginer d'autre façons de remplir la position toute à gauche, par exemple, en additionnant modulo 2 les trois derniers bits et l'antépénultième, etc.

Exemple : Prenons un registre à 4 bits que l'on remplira pour commencer avec le motif 1111. En utilisant le schéma ci- dessus on obtiendra successivement la suite d'états suivante avant de se répéter : 1111, 0111, 0011, 0001, 1000, 0100, 0010, 1001, 1100, 0110, 1011, 0101, 1010, 1101, 1110.

En reconvertissant les contenus successifs du registre en base dix, on obtient une suite pseudo aléatoire de nombres. Evidemment, le cycle est déterminé quand on retrouve le motif initial. Remarquons que l'on a obtenu tous les motifs possibles, sauf 0000 et cela aurait été le cas pour tous les états initiaux possibles, sauf 0000. On a trouvé dans notre exemple un cycle de longueur maximale, la suite de sortie est la chaîne de bits les moins significatifs « 111100010011010 ».

En théorie un RDRL à n bits peut engendrer des suites pseudo aléatoires de longueur 2^n-1 bits avant de se répéter. Pour obtenir cela, le registre à décalage doit passer par tous ses 2^n-1 états internes. Seul les RDRL avec certaines séquences de dérivation passeront par tous les 2^n-1 états internes, ils sont appelés RDRL de période maximale.

Pour qu'un RDRL soit de période maximale, le polynôme formé par les éléments de la séquence de dérivation + 1 doit être un polynôme primitif modulo 2.

Un polynôme primitif[17,11] de degré n est un polynôme irréductible qui divise $x^{2^n-1}-1$ mais pas x^d+1 pour tout d qui divise 2^n-1. Par exemple, le polynôme

$$x^{32}+x^7+x^5+x^3+x^2+x+1$$

est un polynôme primitif modulo 2.

Les RDRL sont des générateurs de suites pseudo aléatoire par eux-mêmes mais ils ont des propriétés non aléatoire gênantes. Les suites de bits successifs sont linéaires, ce qui les rend tout sauf utiles pour la cryptographie. Pour un RDRL de taille n, l'état interne est donné par les n bits précédents de sortie de générateur Même si le schéma de rétroaction est inconnu, il ne faut que $2n$ bits consécutifs de la sortie pour la déterminer.

De grands nombres aléatoires engendrés à partir de bits successifs de telles suites sont fortement corrélées et pour certains types d'applications, pas du tout aléatoires. Malgré cela, les RDRL sont souvent utilisés comme brique de base d'algorithmes de chiffrement.

5 Générateurs de nombres pseudo aléatoires crypto graphiquement sûrs

Les applications cryptographiques exigent plus des générateurs pseudo aléatoires de suites que les autres applications informatiques. « Aléatoire du point de vue cryptographie » ne signifie pas statistiquement aléatoire bien que cela en fasse partie.
Pour qu'une suite pseudo aléatoire soit crypto graphiquement sûre, le générateur doit en plus satisfaire la propriété suivante : Il est imprévisible. Il doit être impossible par calcul de prédire quel sera le bit aléatoire suivant, connaissant complètement l'algorithme ou le matériel qui engendre la suite et connaissant tous les bits déjà engendrés. Alors suites pseudo aléatoires crypto graphiquement sûres ne peuvent être compromis que si on connait le secret : le germe utilisé pour le générateur.

5.1 Approche par la théorie de la complexité

Dans le cadre de cette approche des chiffrements en continu, le cryptographe essaie d'utiliser la théorie de la complexité pour prouver que ces générateurs sont sûrs. De ce fait, ces générateurs ont tendance à être plus compliqués, basés sur le même type de problèmes que nous avons déjà vus pour la cryptographie à clef publique. Et, comme la cryptographie à clef publique ils ont tendance a être lents.

5.2 quelques exemples

5.2.1Générateur de Blum-Micali

La sécurité de ce générateur dépend de la difficulté de calculer des logarithmes discrets[18]. Soit a un nombre premier et p un nombre premier impair. Un germe aléatoire, x_0, démarre le processus :

$$x_{i+1} = (a)^{x}{}_{i} \bmod p.$$

La sortie de générateur vaut 1 si $x_i < (p\text{-}1)/2$, et 0 si non.

Si p est assez grand pour que le calcul du logarithme discret modulo p soit infaisable, alors le générateur est sûr.

5.2.2 Générateur RSA

Le générateur RSA[20,21]. est basé sur la difficulté de la factorisation. Les paramètres initiaux sont : n, le produit de deux nombres premiers, et un germe aléatoire x_0 inférieur à n.

$$x_{i+1} = (x_i)^e \bmod n, \text{ e un entier inversible modulo } \varphi(n).$$

La sortie du ce générateur est le bit le moins significatif de x_i. Si n est assez grand, alors le générateur est sûr.

5.2.3 Générateur pseudo aléatoire fondé sur la difficulté de déterminer le caractère quadratique.

5.2.3.1 Fonction carré quadratique :

La fonction carré modulaire

$$f : Z_n \longrightarrow Z_n$$
$$x \longmapsto x^2$$

n'est pas à proprement parler une fonction type RSA, puisque l'exposant 2 n'est jamais inversible modulo $\varphi(n)$. En particulier elle n'est pas bijective pour $n > 2$. Cet inconvénient est compensé par une propriété très agréable d'un point de vue théorique, Contrairement aux fonctions de type RSA. On sait prouver formellement qu'inverser la fonction carré modulo n est aussi difficile que factoriser n.

Quelques constatations :

Sur Z_P avec p premier, dans ce cas la fonction $x \longrightarrow x^2$ est facile à inverser : Si $p = 3 \bmod 4$ et $c = x^2$, on trouve une racine de c en calculant $d = c^{(p+1)/4}$

Puisque $d^2 = c^{(p+1)/2} = c^{(p-1)/2}. c = x^{p-1}. c = c$.

Si p n'est pas congru à 3mod 4, cela n'est pas aussi simple, mais l'on peut calculer une racine carré presque aussi efficacement.

Sur Z_n où $n = pq$, p et q premiers : Si on connaît p et q, il est facile de trouver une racine carré de c. Il suffit de trouver des racines modulo p et modulo q et d'appliquer le théorème chinois. Concrètement cela signifie que l'on trouve u et v tels que $u^2 = c \bmod p$ et $v^2 = c \bmod q$ puis on forme $x = au + bv$ où

$$\begin{cases} a = 1 \bmod p \\ a = 0 \bmod q \end{cases} \text{ et } \begin{cases} b = 1 \bmod q \\ b = 0 \bmod p \end{cases}$$

on a $x^2 = c \bmod n$. Remarquons encore que modulo un tel n tout carré inversible admet 4 racines.

5.2.3.2 Le caractère quadratique

Considérons le problème suivant : Etant donné un entier x modulo un grand nombre n, on veut savoir si x est ou non résidu quadratique sans que l'on

s'intéresse au calcul explicite de la racine si elle existe. On dira que l'on souhaite

déterminer le caractère quadratique de *x*. Bien entendu, modulo un nombre premier *p*, il est encore plus facile de calculer le caractère quadratique que d'expliciter une racine carrée. Celui-ci est donné par le symbole de Legendre, facilement calculable en un temps polynomial, par exemple par applications successives de la loi de réciprocité. Comment, par contre, déterminer le caractère quadratique d'un inversible *x* si *n* est composé, plus particulièrement si $n = pq$, ou p,q sont premiers ?

On peut toujours calculer le symbole de Jacobi $(\frac{x}{n})$. Si celui-ci vaut -1, on sait que *x* n'est pas résidu quadratique modulo *n*. Mais si $(\frac{x}{n}) = 1$, il y'a exactement autant de résidus quadratiques modulo *n* que de non résidus de symbole de symbole de Jacobi égale à 1. Soit $\varphi(n)/4$. A l'heure actuelle, on ne dispose d'aucun algorithme pour déterminer le caractère quadratique d'un tel *x* sans passer par la factorisation de *n*. La communauté cryptographique fait couramment l'hypothèse qu'il est difficile de déterminer le caractère quadratique. Notons que c'est une hypothèse a priori plus forte que la factorisation. Si l'on sait factoriser on sait déterminer le caractère quadratique . Réciproquement il n'est pas clair que savoir déterminer le caractère quadratique permette de factoriser plus facilement.

5.2.3.3 Générateur Blum Blum Shub :

Actuellement le générateur le plus simple et le plus efficace est appelé le générateur Blum Blum Shub d'après le nom de ses inventeurs. Nous utilisons l'abréviation BBS bien que de temps a autre il soit appelé générateur à résidu quadratique[19].

La théorie sous-jacente au générateur BBS est liée aux résidus quadratiques modulo *n*. Voici comment il fonctionne :

Trouver deux grands nombres premiers *p* et *q* qui soient congrus à 3 modulo 4. Le produit de ces nombres est un entier de Blum. Choisir un autre entier aléatoire qui soit premier par rapport à *n*. Calculer $x_0 = x^2 \bmod n$ .C'est le germe du générateur.

Maintenant, on peut commencer à calculer des bits le i^e bit pseudo aléatoire est le bit le moins significatif de x_i où $x_i = (x_{i-1})^2 \bmod n$.

La propriété étrange de ce générateur est que nous n'avons pas à calculer tous les i-1 bits précèdent pour obtenir le i^e bit. Si on connaît p et q, on peut calculer le i^e bit directement :

« b_i est le bit le moins significatif de x_i ou $x_i = x_0^{(2^i) \bmod (p-1)(q-1)} \bmod n$ ».

Cette propriété signifie qu'on peut utiliser ce générateur pseudo aléatoire de bits comme un crypto système de chiffrement en continu pour des fichiers à accès aléatoire.

La sécurité de ce schéma dépend de la difficulté de factoriser n. On peut rendre n public, ainsi tout le monde peut engendrer des bits en utilisant le générateur. Toutefois, à moins que les cryptanalystes puissent factoriser n, ils ne peuvent jamais prédire la sortie de générateur même pas sous la forme d'affirmations telles que « le bit suivant à 50 % de chances d'être un 1 ».

Plus fort, le générateur BBS est imprévisible à gauche et imprévisible à droite. Cela signifie que, étant donné une suite engendrée par le générateur, les cryptanalystes ne peuvent pas prédire le bit suivant ou le bit précèdent de la suite. Ce qui n'est pas de la sécurité basée sur un générateur de bits compliqué que personne ne comprend, mais bien sur les mathématiques sous- jacente à la factorisation de n. Pour ceux qui veulent des suites pseudo aléatoires de bits crypto graphiquement sûres, les suites que génèrent le BBS ont un très bon comportement pseudo aléatoire.

Cet algorithme est lent, mais il existe des améliorations. Il Semble que l'on puisse utiliser plus que le bit le moins significatif de x_i comme bit pseudo aléatoire. Si L est la longueur de x_i, les $\log_2 L$ bits moins significatifs de x_i peuvent être utilisés[19,20].

Le générateur BBS est relativement lent et n'est pas utilisé pour le chiffrement en contenu. Toutefois, pour des applications de haute sécurité, telle que la génération de clefs, ce générateur est le meilleur.

Application :

Le programme suivant, écrit en langage C, permet de générer une suite binaire pseudo aléatoire et d'écrire le résultat dans un fichier texte. Ce programme est basé sur l'algorithme BBS.

```
#include <stdio.h>
#include <math.h>
int main(){
unsigned long int siute[8];
unsigned long int n,p,q,s,sauv;
FILE*fichier;
Intnombre;
Int coef=0;
Int taille ;
// Lire les nombres p,q,s
printf("donner p,q,s ") ;
scanf( "%d %d %d \n", &p,&q,&s) ;
//Lire la taille de fichier
printf("donner en octer la taille de fichier a générer:") ;
scanf("%d\n", &taille) ;
sauv=s ;
n=p*q ;
//ouvrir le fichier "suite. txt" en mode d'écriture
fichier=fopen("suite.txt","w") ;
int i=0 ;
while (i<taille) {
                    nombre=0;
                    for(int cpt=0;cpt<8;cpt++){
                                if(cpt==0) suite [cpt]=(sauv*sauv)% n;
                                else       (suite[cpt-1]*suite[cpt-1])%n ;

                                if(suite[cpt]%2==0) coef=0 ;else coef=1
                                // récupérer le bit de degré (7-cpt)
                                nombre= nombre+(coef*(int)pow(2,7-cpt) ;
                          }
                          // écrire "nombre" dans le fichier texte.
                          Fputc((unsignedchr)nombre,fichier) ;
                          Sauv=suite[cpt-1] ;
                          I++ ;
}
fclose(fichier) ;
return0 ;
}
```

6. Tests statistiques

Dans cette partie, nous présentons les techniques statistiques usuelles utilisées pour la validation de générateurs aléatoires et pseudo-aléatoires. Nous présentons également quelques standards tels que les normes FIPS ou NIST.

6.1 Tests d'hypothèses :

6.1.1 Lois d'échantillonnages

Soient n variables aléatoires indépendantes X_1, X_2, …, X_n suivant une loi normale N(0, 1). Alors : $X=\sum_{i=1}^{n} X_i^2$ suit une loi de khi-deux à n degrés de liberté. Pour Y qui suit une loi normale N(0, 1) et Z qui suit une loi de khi-deux à n degrés de liberté, la variable aléatoire $T = \frac{y}{\sqrt{z/n}}$ suit une une loi de student à n degrés de liberté.

6.1.2 Principe d'un test d'hypothèse :

Soit X une variable aléatoire de fonction de répartition $F(x)$. Soit n observations $x_1, x_2, \ldots, x_n$ le but est d'estimer $F(x)$, la loi de X. On cherche à tester l'hypothèse nulle

$H_0 : F(x) = F_0(x)$ (ou $F_0(x)$ est une loi hypothétique donnée)

Cotre une hypothèse alternative

$H_1 : F(x) \neq F_0(x)$

On cherche une règle de décision qui permette de décider laquelle des hypothèses est la bonne (la plus probable).

Soit le tableau suivant :

H_0 (pas de relation de différence)	H_0 En réalité vraie (Hypothèse d'uniformité)	H_0 En réalité fausse (il existe une différence)
Décision H_0 fausse (i.e. accepter H_1)	Erreur de type 1 :risque α De dire qu'il y a quelque chose alors qu'il n'y a rien (risque d'illusion)	$1-\alpha$
Décision H_0 vraie (Acceptation de l'hypothèse nulle)	$1-\beta$: puissance de test	Erreur de type 2 :(risque β) De dire qu'il n'y a rien alors qu'il y a quelque chose.(risque de négligence)

Où : α = P(rejeter H_0 / H_0 vraie) et β = P(accepter H_0 / H_0 fausse).

En général la règle de décision porte sur une certaine statistique T= T(x_1, x_2, …, x_n) qu'on appelle critère ou test.

6.1.2.1. Règle de décision

Si T $\in$ A (Domaine d'acceptation) accepter H_0.

Si T $\in$ R (Domaine de rejet) rejeter H_0.

La statistique de test T est choisie, telle que sa loi sachant H_0 $f_T(x/ H_0)$ est connue. En général A =(T_1, T_2). La règle de décision est donc :

Accepter H_0 si $T_1 \leq T \leq T_2$

Rejeter H_0 si $T_1 > T$ ou $T > T_2$

La région de rejet R est choisie de telle manière que les erreurs α et β soient minimales.

On a recours au procédé suivant :

On fixe α (appelé niveau de signification du test). Parmi tous les tests de niveau α i, e P(rejet H_0 / H_0vraie) = P(T$\in$R / H_0 vraie) = $\int_R f_T(x / H_0)\, dx < \alpha$. On cherche le test tel que 1-β soit maximal.

6.1.2.2 Démarche du test

1- Fixer α = (petit soit 5%, 1%,…) en cryptographie on prend généralement 1%

2- choisir un échantillon de n observations x_1, x_2, …, x_n

3- calculer la valeur observée du critère T = T(x_1, x_2, …x_n).

Si $T > u_{(1-\alpha)/2}$ ou $T < u_{\alpha/2}$ rejeter H_0.

Si non accepter H_0.

6.1.3 Test d'ajustement

6.1.3.1 test du khi-deux : Soit un échantillon aléatoire de taille n extrait d'une population et divisé en k classes d'effectifs respectifs n_1, n_2, …, n_k et de probabilité respectives P_1, P_2,…P_k théoriques.

Il s'agit de tester :

$$\begin{cases} H_0 : F(x) = F_0(x) \\ H_1 : F(x) \neq F_0(x) \end{cases}$$

La quantité $\sum (n_i - n\,P_i)^2/nP_i$ pour i =1, … ,k suit une loi χ^2_{k-1} à k-1 degrés de liberté si nP_i est supérieur à 5.

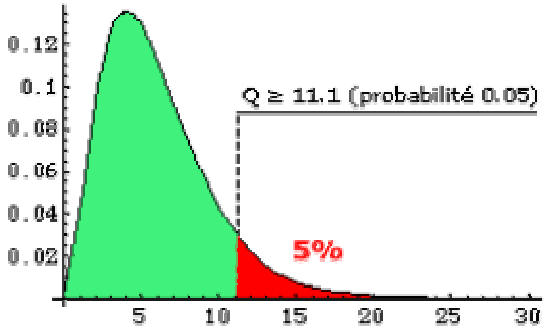
Si la variable aléatoire est continue, on associe une probabilité P_j à la j^{eme} classe en utilisant la densité de probabilité.

Le seuil de signification α étant fixé, on lit sur la table du khi-deux la valeur $\chi^2_{k-1}(\alpha)$ en fonction de nombre de degrés de liberté, k-1 est le nombre de variables indépendantes. En effet comme $\sum_{i=1}^{k} n_i = n$, il suffit de connaître en fait k-1 variables.

On a : $P=[\chi^2_{k-1} < \chi^2_{k-1}(\alpha)]=1-\alpha$

Si la valeur calculé $\chi^2_{k-1}= \sum(n_i-nP_i)^2/nP_i$ pour i=1,...,k est telle que : $\chi^2_{k-1} < \chi^2_{k-1}(\alpha)$ on accepte l'hypothèse d'ajustement c'est-à-dire H_0 ; si non on la rejette, la loi théorique proposée ne peut représenter valablement la distribution expérimentale. La quantité nP_i est l'effectif théorique.

Exemple : On a lancé un dé 90 fois et on a obtenu les issues 1 à 6 (k=6) avec les effectifs suivants 14, 16, 13, 18, 17, 12. Si le dé n'est pas pipé (notre hypothèse), on attend comme effectifs moyens théorique 15 pour toute les issues. La quantité $(14-15)^2/15+(16-15)^2/15+(13-15)^2/15+(18-15)^2/15+(17-15)^2/15+(12-15)^2/15$ implique que $\chi^2_5= 1,87$. Pour k-1=5 degrés de liberté et un seuil de tolérance α=5%, la valeur $Q = \chi^2_5(\alpha)= 11,1$ (voir figure ci- dessous). Comme 1, 87<11,1, on accepte l'hypothèse selon laquelle le dé est régulier.



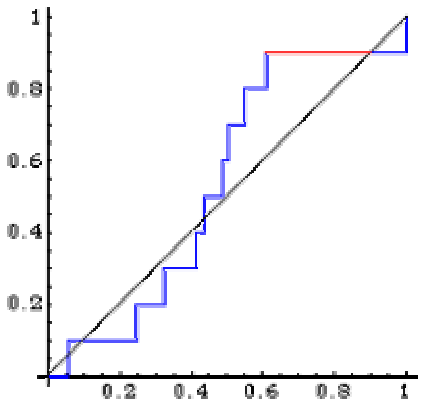
Fonction de répartition de la loi de χ^2 pour 5 degrés de liberté et un seuil de tolérance α=5%.

6.1.3.2 Test de colmogorov smirnov : Soit F(x) la fonction de répartition pour une population et soit $F_n(x)$ la fonction de répartition empirique (fréquence cumulée relative)pour un échantillon de taille n dont l'ordre est croissant, on détermine D_n= Max $|F_n(x)-F(x)|$ et on compare cet écart à des valeurs critiques particulières données par les tables $d_n(n)$.

On rejette H_0 si $D_n \geq d_n(n)$ à un seuil de signification α.

Exemple :

Soit la suite ordonnée de n=10 nombres pseudo aléatoires :0.0582795, 0.248849, 0.326321, 0.411576, 04335057, 0.485013, 0.5035, 0.547168, 0.611682, 0.998397. La fonction de répartition $F_n(x)$ de cet échantillon est comparée avec la fonction de répartition uniforme F(x)= x. La figure ci-dessous nous facilite de désigner la distance entre les fonctions théoriques et empiriques.



10 nombres, D_n=0.2883<0.410
on accepte l'hypothèse pour
α=5%.

6.2 Validation des générateurs

Enumérons quelques critères permettant de vérifier que la suite générée est bien aléatoire et issue d'une variable aléatoire de la loi uniforme U[0, 1].

Si s ={s_0, s_1, … } est une suite infinie, on notera s^n = {s_0, s_1,… s_{n-1}}la sous suite finie.

La suite s est N-périodique si s_{i+N}= s_i pour tout i≥0. Elle est dite périodique si elle est N-périodique pour un certain entier positif N. si s est une séquence périodique de période N, alors le cycle de s est la sous suite s^N.

Une série (run) de s est une sous suite de 0 consécutifs (ou de 1 consécutifs) qui n'est ni précédée, ni suivie par le même symbole. Une série de 0 est appelée trou (gap) et une série de 1, un bloc.

Rappelons pour des raisons historiques les postulats de golomb, au nombre de 3 qui formulent les conditions nécessaires pour qu'une suite pseudo aléatoire ressemble à une suite vraiment aléatoire.

Postulat 1 : Dans le cycle s^N de s le nombre de 1 diffère du nombre de0 au moins d'une unité.

Postulat 2 : Dans le cycle s^N , au moins la moitié des séries (runs) sont de longueur 1, au moins un quart de longueur 2, au moins un huitième de longueur 3,…Tant que le nombre de séries excède 1. de plus, pour chacune de ses longueurs, il y a presque au tant de trous que de blocs.

Postulat 3 : La fonction d'auto-corrélation est de la forme :

$$N.C(t)=\sum_{i=0}^{N-1}(2s_i-1).(2s_{i+t}-1)=\begin{cases} N, \text{ si } t=0 \\ K, \text{ si } 1\le t\le N-1 \end{cases}$$

Une suite binaire qui satisfait les postulats de Golomb est dite pseudo bruitée ou pn-séquence.

Exemple : soit la suite périodique de période N=15 de cycle

S^{15}=0,1,1,0,0,0,1,1,1,1,0,1

Vérifions que cette suite satisfait les postulats.

P1 : le nombre de 0 est 7 et le nombre de 1 est 8.

P2 : La suite à 8 séries. 4 séries de longueur 1, 2 séries de longueur 2, 1 série de longueur 4.

P3 : La fonction d'auto-corrélation C(t) prend deux valeurs : C(0)= 1 et C(t)= -1/15, Pour 1≤ t ≤ 14.

C'est donc bien une pn-séquence.

Soit s=s_0, s_1,…s_{n-1}une séquence binaire de longueur n.

6.2.1. Test de fréquence (test monobit). Il teste si la suite possède presque autant de 0 que de 1, comme on pourrait s'y attendre dans une séquence aléatoire. Soit n_0 le nombre de 0, et n_1 le nombre de 1. le critère du test est basé sur la statistique

$$\chi_1=(n_0-n_1)^2/n$$

qui suit approximativement une loi de khi-deux à 1 degré de liberté pour $n\geq 10$ (en pratique on recommande, comme d'ailleurs pour les tests suivants, de prendre une taille d'échantillon assez grande, par exemple supérieure à 10 000.

6.2.2. Test de séries (test à deux bits). Ce test détermine si les nombres d'occurrences de 00, 01, 10, 11 en tant que sous suites de s sont approximativement égaux, comme on pourrait l'espérer d'une suite vraiment aléatoire. Soit n_0 le nombre de 0, et n le nombre de 1 et n_{00}, n_{01}, n_{10}, n_{11} les nombres d'occurrences de 00, 01, 10, 11 respectivement. Notons que $n_{00}+n_{01}+n_{10}+n_{11}=n-1$. Le critère de test est basé sur la statistique

$$\chi_2= \frac{4}{n-1}(n_{00}^2+n_{01}^2+n_{10}^2+n_{11}^2)-\frac{2}{n}(n_0^2+n_1^2)+1$$

qui suit approximativement une loi de khi-deux à 2 degrés de liberté pour $n\geq 21$.

Exemple : On considère la séquence (non aléatoire) s de longueur n=160 obtenue en reproduisant 4 fois la séquence suivante :

11100 01100 01000 10100 11101 11100 10010 01001

Ici, $n_{00}=44$, $n_{01}=40$, $n_{10}=40$, $n_{11}=35$, et la valeur calculée $\chi_2=0.6252$ alors que la valeur théorique pour un degré de liberté égal à 2 $\chi_2^{théor}=5.9915$, et la suite s passe ce test.
(**α** = 5% est le niveau de tolérance).

6.2.3. Poker test. Soit m un entier positif tel que $[n/m]\geq 5.2^m$ et soit k=[n/m]. On divise la séquence s en k parties disjointes de longueur m chacune ; soit n_i le nombre d'occurrences de la séquence de type i de longueur m, $1\leq i\leq 2^m$. Si les séquences de longueur m apparaissent un nombre identique de fois dans s, comme on pourrait l'attendre d'une séquence vraiment aléatoire. Le critère du test

$$\chi_3=\frac{2^m}{k}\left(\sum_{i=1}^{2^m} n_i^2\right)-k$$

qui suit approximativement une loi de khi-deux à 2^m degrés de liberté.

6.2.4. Test de série (Run test). Le but ici est de déterminer si le nombre de séries ou runs (de 0 ou de 1) de différentes longueur dans la séquence s est égal au nombre de séries attendu dans une séquence vraiment aléatoire. Le nombre de gaps(ou de blocs) de longueur i dans une séquence aléatoire de longueur n est égal à $e_i=(n-i+3)/2^{i+2}$. Soit k le plus grand entier i pour lequel $e_i \geq 5$. Soit d'autre part B_i et G_i les nombres de gaps et de blocs de longueur i dans s pour chaque i, $1 \leq i \leq k$. Le critère de test est

$$\chi_4 = \sum_{i=1}^{k} \frac{(B_i - e_i)^2}{e_i} + \sum_{i=1}^{k} \frac{(G_i - e_i)^2}{e_i}$$

qui suit approximativement une loi de khé-deux à 2k-1 degrés de liberté.

Exemple : (suite). On a $e_1=20.25$, $e_2=10.0625$, $e_3=5$, $k=3$. Il y a 25, 4, 5 blocs de longueur 1, 2, 3 respectivement et 8, 20, 12 gaps de longueur 1, 2, 3 respectivement. La valeur calculée du critère est $\chi_4=31.7913$ et celle du khi-deux théorique $\chi_4^{théor}=9.4877$. On constate que la suite s ne passe pas ce test. C'est en principe suffisant pour rejeter définitivement la suite.

6.2.5. Test d'auto corrélation. Il permet de tester la corrélation entre la séquence s et ses versions translatées (non cycliques). Soit d un entier fixe, $1 \leq d \leq [n/2]$. Le nombre de bits dans s qui ne sont pas égaux à leurs d- translations est :

$$A(d) = \chi_5 = \sum_{i=1}^{n-d-1} s_i \oplus s_{i+d}$$

où $\oplus$ dénote l'opérateur XOR. Le test est basé sur le critère

$$\chi_5 = 2\left(A(d) - \frac{n-d}{2}\right)/\sqrt{n-d}$$

qui suit approximativement une loi normale standard pour $n-d \geq 10$.

Il est important de savoir qu'il existe d'autres tests que doit subir un générateur dans le but de l'accepter ou de le rejeter, comme dans les deux cas suivants :

Norme FIPS140-1. Elle spécifie 4 tests statistiques de l'uniformité. L'utilisateur ne sélectionne pas un niveau de signification α. Par contre, elle fournit des bornes que la valeur calculée de la statistique doit satisfaire. La sortie du générateur représentée par une séquence s de longueur de 20 000 bits doit passer tous les tests suivant, faute de quoi elle est rejetée.

(i) test monobit. Le nombre n_1 de 1 doit satisfaire $9654 < n_1 < 10346$.

(ii) poker test. La statistique χ_3 est calculée pour m=4. Le test est positif si $1.03<\chi_3<57.4$.

(iii) test de série (run test). Les nombres B_i de blocs et de G_i de gaps de longueur i dans s sont dénombrés pour chaque i, $1\leq i\leq 6$. (pour les besoins de ce test, les séries de

Longueur de la série	Intervalle requis
1	2267-2733
2	1079-1421
3	502-748
4	223-402
5	90223
6	90-223

longueur >6 sont considérées égales à 6). Le test est positif si les 12 valeurs de B_i, G_i, $1\leq i\leq 6$, se trouvent dans les limites spécifiées par les intervalles de la table ci- dessous.

(v) test de longues séries (long run test). La séquence s passe ce test s'il n'y a pas de séries de longueur 34 ou plus.

Pour des applications à haute sécurité, la norme FIPS 140-1 demande à ce que les quatre tests soient réalisés chaque fois que le GBA est amélioré. La norme permet également l'utilisation de test alternatifs d'efficacité supérieure.

Package du NIST

C'est une liste de 16 tests statistiques développés par le NIST (national institutes of standards and technology) pour tester si une suite binaire produite par un GBA est bien aléatoire (ou plutôt proche d'une suite vraiment aléatoire). Certains d'entre eux ont été étudiés précédemment.

1 Test de fréquence monobit.

2 Test de fréquence entre blocs.

3 Test des runs.

4 Test des longues séries.

5 Test de matrice aléatoire. **6** Test spectral.

7 The Non overlapping template matching test**.**

8 The Overlapping Template Matching test.

9 Test statistique univesel de Maurer.

10 Test de compression de Lempel-Ziv.

11 Test de complexité linéaire.

12 Serial test.

13 Test de l'entropie.

14 Test des sommes cumulées.

15 Test des excursions aléatoires.

16 Variante du test des excursions aléatoires.

Conclusion

Dans un cadre statistique, ludique ou à des fins de simulations, l'utilisation d'un générateur algorithmique, qui ne peut être alors que pseudo aléatoire, ne pose aucun problème particulier. Soit on utilise un générateur reconnu comme bon, soit on en fabrique un en s'appuyant sur les théorèmes de Knuth, puis on lui fait passer un maximum de tests afin de s'assurer qu'il satisfait aux conditions qu'on lui impose.

Par contre, utiliser un tel générateur à des fins cryptographiques, est déconseillé si la sécurité du chiffrement repose sur le seul secret des constantes utilisées dans le générateur ; c'est pourquoi il est d'une importance capitale d'utiliser un générateur s'approchant le plus possible d'un générateur vraiment aléatoire.

Références

[1] Bruce schneier, cryptographie appliquée

[2] Khaldi Khaled, méthodes statistiques

[3] Knuth D.E, the art of computer programming semi numerical algorithms chapter 3 Addison Wesley, 1981

[4] J.P Barthé lemy. G. Cohen A. Labstein, complexité algorithmiques et problèmes de communications

[5] Gilles Zemor, cours de cryptographie

[6] L. Adleman and M. Huang, primalitytestingand abelian varietiesovers finite fields, vol 1512 of lecture notes in math, springer- verlay 1992

[7] Pierre Larbin, produiredes nombreau hazard, Novembre 1994

[8] Park. S. K Miller K. W. Random Number Générators: cood ones are hard to find, comm. Of TACM, vol 31, 10 oct 1988 P 1192-1201

[11] N. Zierler et J. Brillhart. Onprimitive trinomials information and control, vol.15. P 67-69, 1969.

[12].P.L'ECUYER.Randon Numbers for Simulation . communications of the ACM,vol 33 ,n 10, p : 85-97, octobre 1990

[13] J.C.LAGARIAS et J.REEDS.nnique extraporation of polynomial recurrences SIAM jounal on computing , vol 17, n 2, p :342-362, 1988

[14] H.KRAWCZYK.How to perdict congruentials .journal of algorithms vol 13, n 4, p:527-545

[15] KRAWCZYK.How to perdict congruentiels Generators advances in cryptology-crypto'89 proceedings, p: 138-153 springer-verlag, Berlin, 1986

[16] M.J.KOCHANSKI-Developing an RSA chip. Advances in cryptology crypto'85 proceedings, p: 350-357 springer-verlag, Berlin, 1986

[17]S.W.GOLOMB.shift register sequences.holden-day,san Francisco, CA, 1967.(reimprimé par Aglon Park press,1982)

[18]H.BLUM et S.MICALI How to enerate cryptographically strong sequeences of pseudo random bits. SIAM Journal on computing, vol.13, n 4, p: 850-864, 1984.

[19] L. BLUM ,M .BLUM,M SHUB. Asimple unpredictable pseudo-random number generator.SIAM journal on computing , vol.15,n 2, p:364-383, 1986.

[20]w.ALEXI,B-Z.CHOR,O.GOLDREICHet C.P SCHNORR.RSA and rabin function: certain parts are as hard as the whole. SIAM journal on computing vol.17, n 2, p: 194-209 avril 1988

[21]W.ALEXI.B-Z .CHOR, O.GOLDREICH et C.P SCHNORR.RSA and rabin function: certain parts are as hard as the whole. Proceedings of the 25th IEEE anuel symposium on the foundations of computer science, p:449-457, 1984.