Groupe de travail Réseau
Request for Comments : 4086
BCP : 106
RFC rendue obsolète : 1750
Catégorie : Bonnes pratiques actuelles

D. Eastlake III, Motorola Laboratories
J. Schiller, MIT
S. Crocker
juin 2005
Traduction Claude Brière de L'Isle

Exigences d'aléa pour la sécurité

Résumé
Les systèmes de sécurité sont construits sur des algorithmes cryptographiques forts qui déjouent les tentatives d'analyse de trame. Cependant, la sécurité de ces systèmes dépend de la génération de quantités secrètes pour les mots de passe, les clés cryptographiques, les quantités similaires. L'utilisation de processus pseudo aléatoires pour générer des quantités secrètes peut déboucher sur une pseudo sécurité. Un attaquant sophistiqué peut trouver plus facile de reproduire l'environnement qui a produit les quantités secrètes et rechercher sur le petit ensemble de possibilités que de localiser les quantités dans la totalité de l'espace potentiel des nombres.

Choisir des quantités aléatoires pour déjouer un adversaire motivé et plein de ressources est extraordinairement difficile. Le présent document met l'accent sur les nombreux pièges qui résultent de l'utilisation de sources à faible entropie ou des techniques traditionnelles de génération de nombres pseudo aléatoires pour générer de telles quantités. Il recommande l'utilisation de techniques de matériel vraiment aléatoires et montre que les matériels existants sur de nombreux systèmes peuvent être utilisés à cette fin. Il fait des suggestions pour améliorer le problème lorsqu'une solution matérielle n'est pas disponible, et donne des exemples des tailles que doivent avoir de telles quantités pour certaines applications.

Table des matières

1 Introduction et généralités

Les logiciels de cryptographie sont de plus en plus utilisés, bien qu'il reste beaucoup de chemin à faire pour qu'ils soient partout. Des systèmes comme SSH, IPSEC, TLS, S/MIME, PGP, DNSSEC, et Kerberos arrivent à maturité et commencent à faire partie du paysage du réseau [SSH] [IPSEC] [TLS] [S/MIME] [MAIL_PGP*] [DNSSEC*]. Par comparaison, lorsque la précédente version de ce document [RFC1750] a été publiée en 1994, la seule spécification de sécurité cryptographique de l'Internet dans l'IETF était le protocole d'amélioration de la confidentialité des messages *(Privacy Enhanced Mail protocol)* [MAIL_PEM*].

Ces systèmes fournissent une protection substantielle contre l'espionnage et l'usurpation, cependant, il y a une faille potentielle. Au coeur de tous les systèmes cryptographique se trouve la génération de nombres secrets, indevinables (c'est-à-dire aléatoires).

L'absence de facilités disponibles de façon ouverte pour générer de tels nombres aléatoires (c'est à dire, l'absence de disponibilité générale de sources vraiment imprévisibles) forme une blessure ouverte dans la conception des logiciels cryptographiques. Pour le développeur de logiciels qui veut construire une procédure de génération de clés ou de mots de passe qui fonctionne sur une large gamme de matériels, c'est un problème très réel.

Noter que l'exigence porte sur des données qu'un adversaire ait une très faible probabilité de deviner ou de déterminer. Ceci peut facilement échouer si des données pseudo aléatoires sont utilisées, satisfaisant seulement aux tests statistiques traditionnels de hasard, ou qui se fondent sur des sources de gamme limitée comme des horloges. Parfois, de telles quantités pseudo aléatoires peuvent être devinées par un adversaire qui cherche dans un espace de possibilités étonnamment restreint.

Ce document sur les meilleures pratiques actuelles décrit les techniques pour produire des quantités aléatoires qui résistent aux attaques. Il recommande que les futurs systèmes incluent la génération de nombres aléatoire dans le matériel ou fournissent l'accès aux matériels existants qui peuvent être utilisés à cette fin. Il suggère des méthodes à utiliser si de tels matériels ne sont pas disponibles, et donne des estimations du nombre de bits aléatoires requis pour des applications échantillons.

2 Exigences générales

Aujourd'hui, une exigence d'aléa communément rencontrée est celle de choisir un mot de passe d'utilisateur, habituellement une simple chaîne de caractères. Évidemment, un mot de passe qu'on peut deviner ne fournit aucune sécurité. Pour les mots de passe réutilisables, il est souhaitable que les utilisateurs soient capables de se souvenir du mot de passe. Cela fait qu'il est conseillé d'utiliser des chaînes de caractères prononçables ou des phrases composées de mots ordinaires. Mais ceci n'affecte que le format des informations de mot de passe, et pas l'exigence que le mot de passe soit très difficile à deviner.

De nombreuses autres exigences viennent du domaine cryptographique. Les techniques cryptographiques peuvent être

utilisées pour fournir des services divers, qui incluent la confidentialité et l'authentification. De tels services se fondent sur des quantités, traditionnellement appelées "clés", qui ne sont pas connues d'un adversaire et sont imprévisibles.

Il y a même des utilisations de calcul aléatoire du protocole TCP/IP pour trouver des numéros de séquence initiaux [RFC1948].

Généralement parlant, les exemples ci-dessus illustrent aussi qu'on peut vouloir deux types différents de quantités aléatoires. Dans le cas des mots de passe utilisables par l'homme, la seule caractéristique importante est qu'ils soient imprévisibles. Il n'est pas important qu'ils puissent être composés de caractères ASCII, et donc que le bit de plus fort poids de chaque octet soit zéro, par exemple. D'un autre côté, pour les clés de longueur fixe et autres, on veut normalement des quantités qui soient vraiment aléatoires, c'est à dire, des quantités dont les bits vont réussir les essais statistiques d'aléa.

Dans certains cas, comme celui de l'utilisation de chiffrement symétrique avec clés à utilisation unique ou un algorithme comme la norme US de chiffrement évolué (AES, *Advanced Encryption Standard*) [AES], les parties qui souhaitent communiquer de façon confidentielle et/ou avec authentification doivent toutes connaître la même clé secrète. Dans d'autre cas, où on utilise des techniques cryptographiques symétriques ou à "clé publique", les clés viennent par paires. Une clé de la paire est privée et doit être gardée secrète par une partie ; l'autre est publique et peut être publiée dans le monde entier. Il n'est pas possible de déterminer par le calcul la clé privée à partir de la clé publique, et la connaissance de la clé publique n'est d'aucune utilité pour un adversaire [ASYMMETRIC]. Voir les références générales [SCHNEIER, FERGUSON, KAUFMAN].

La fréquence et le volume des exigences des quantités aléatoires diffèrent considérablement dans les différents systèmes cryptographiques. Avec du RSA pur, les quantités aléatoires ne sont exigées que pour générer une nouvelle paire de clés ; ensuite, un nombre indéterminé de messages peuvent être signés sans autre besoin de nombre aléatoire. L'algorithme de signature numérique de clé publique conçu par l'Institut national américain des normes et des technologies (NIST) exige de bons nombres aléatoires pour chaque signature [DSS]. Un chiffrement avec une clé à utilisation unique (en principe la plus forte technique de chiffrement possible) exige un caractère aléatoire de volume égal pour tous les messages à traiter. Voir les références générales [SCHNEIER, FERGUSON, KAUFMAN].

Dans la plupart de ces cas, un adversaire peut essayer de déterminer la clé "secrète" par essai et erreur. Ceci est possible tant que la clé est suffisamment plus petite que le message que la clé correcte peut être identifiée de façon univoque. La probabilité qu'un adversaire y réussisse doit être rendue suffisamment faible, en fonction de l'application particulière. La taille de l'espace que l'adversaire doit explorer est en relation avec la quantité "d'informations" de clé présente, au sens de la théorie de l'information [SHANNON]. Cela dépend du nombre de valeurs de secret différentes possibles et de la probabilité de chaque valeur, comme suit :

$$\text{Bits d'information} = \sum - p_i * \log_2 (p_i)$$

où i va de 1 au nombre de valeurs de secret possibles et p indice i est la probabilité de la valeur numéro i. (Comme p indice i est inférieur à un, le log sera négatif, ainsi chaque terme de la somme sera non négatif.)

Si il y a 2^n valeurs différentes d'égale probabilité, alors n bits d'information sont présents et un adversaire aurait à essayer, en moyenne, la moitié des valeurs, soit 2^(n-1), avant de prévoir la quantité secrète. Si la probabilité de valeurs différentes est inégale, il y a alors moins d'information présente, et moins d'essais seront, en moyenne, nécessaires à l'adversaire. En particulier, toutes les valeurs que l'adversaire sait être impossibles ou de faible probabilité peuvent être ignorés ab initio par l'adversaire, qui cherchera d'abord parmi les valeurs les plus probables.

Par exemple, considérons un système cryptographique qui utilise des clés de 128 bits. Si ces clés sont déduites en utilisant un générateur de nombres pseudo aléatoires fixes qui a une graine de huit bits, un adversaire a alors besoin de chercher seulement parmi 256 clés (en faisant tourner le générateur de nombres pseudo aléatoires avec toutes les graines possibles), et non 2^128 clés comme il peut sembler être le cas de prime abord. Il y a seulement 8 bits "d'information" dans ces clés de 128 bits.

Alors que l'analyse précédente est correcte en moyenne, elle peut être trompeuse dans certains cas pour l'analyse cryptographique où ce qui est réellement important est le facteur travail pour l'adversaire. Par exemple, supposons qu'on a un générateur de nombres pseudo aléatoires qui génère des clés de 128 bits, comme à l'alinéa précédent, mais qu'il génère zéro la moitié du temps et une sélection aléatoire pour les 2^128 - 1 valeurs restantes le reste du temps. L'équation de Shannon ci-dessus nous dit qu'il y a 64 bits d'information dans une de ces valeurs de clé, mais un adversaire, simplement en essayant la valeur zéro, peut casser la sécurité de la moitié des utilisations, bien que cette moitié soit aléatoire. Et donc, pour les besoins de la cryptographie, il est aussi utile de regarder les autres mesures, telles que de min-entropie, définie par

$$\text{Min-entropie} = - \log_i (\text{maximum} (p))$$

où i est comme ci-dessus. En utilisant cette équation, nous obtenons 1 bit de min-entropie pour notre nouvelle distribution hypothétique, contre 64 bits d'entropie classique de Shannon.

Un spectre continu d'entropies, parfois appelé entropie de Renyi, a été défini, spécifié par le paramètre r. Ici, r = 1 est l'entropie de Shannon et r = infini est min-entropie. Quand r = zéro, c'est juste log (n), où n est le nombre de probabilités différentes de zéro. L'entropie de Renyi est une fonction non croissante de r, aussi min-entropie est toujours la mesure la plus prudente de l'entropie et habituellement la meilleure à utiliser pour l'évaluation cryptographique [LUBY].

Le sens traditionnel du niveau d'aléa testé statistiquement N'EST PAS la même chose que l'imprévisibilité exigée par les usages de sécurité.

Par exemple, l'utilisation d'une séquence constante largement disponible, comme le tableau de nombres aléatoires tiré des tableaux mathématiques de la norme CRC *(CRC Standard Mathematical Table)* est très faible en face d'un adversaire. Un adversaire qui apprend ou prévoit peut facilement casser toutes les sécurités, futures et passées, fondées sur la séquence [CRC]. Pour un autre exemple, utiliser AES avec une clé constante pour chiffrer des entiers qui se suivent comme 1, 2, 3, ... produira un résultat qui a aussi d'excellentes propriétés aléatoires statistiques mais est prévisible. D'un autre côté, prendre six jets successifs d'un dé à six faces et coder les valeurs résultantes en ASCII produirait un résultat statistiquement pauvre avec un résultat réellement imprévisible. Aussi réussir ou échouer aux tests statistiques ne révèle pas si quelque chose est imprévisible ou prévisible.

3 Sources d'entropie

Les sources d'entropie tendent à être très dépendantes de la mise en œuvre. Une fois qu'on a rassemblé une entropie suffisante, elle peut être utilisée comme graine pour produire la quantité pseudo aléatoire de force cryptographique requise, comme décrit aux Sections 6 et 7, après l'avoir débiaisé ou mixée en tant que de besoin, comme décrit aux Sections 4 et 5.

Y a-t-il un espoir d'avoir à l'avenir un vrai hasard fort et transportable ? Cela se pourrait. Tout ce qui est nécessaire est une source physique de nombres imprévisibles.

Le bruit thermique (parfois appelé bruit de Johnson dans les circuits intégrés) ou une source de dégradation radioactive et un oscillateur rapide autonome pourrait directement faire l'affaire [GIFFORD]. C'est un matériel trivial, et il peut facilement être inclus comme partie standard de l'architecture d'un système informatique. La plupart des appareils d'entrée audio (ou vidéo) sont utilisables [TURBID]. De plus, tout système avec un disque rotatif ou un oscillateur annulaire et une source temporelle stable (cristal) ou un équivalent, a une source adéquate d'aléa ([DAVIS] et paragraphe 3.3). Tout ce qui est nécessaire est que les fabricants d'ordinateurs se mettent à comprendre que ce petit matériel supplémentaire et son logiciel d'accès sont utiles et nécessaires.

ANSI X9 est en train de développer un standard qui comporte une partie dédiée aux sources d'entropie. Voir la partie 2 de [X9.82].

3.1 Volume requis

Quelle quantité d'imprévisibilité est nécessaire ? Est-il possible de quantifier l'exigence en termes de, disons nombre de bits aléatoires par seconde ?

La réponse est qu'il n'en est pas besoin de beaucoup. Pour AES, la clé peut être de128 bits, et, comme on le montre dans l'exemple de la Section 8, même le plus fort système de sécurité n'exige vraisemblablement pas un matériel de clés de plus de 200 bits. Si une série de clés est nécessaire, elles peuvent être générés à partir d'une graine fortement aléatoire (valeur de départ) en utilisant une séquence cryptographiquement forte, comme expliqué au paragraphe 6.2. Quelques centaines de bits aléatoires générés au démarrage ou une fois par jour sont suffisants si de telles techniques sont utilisées. Même si les bits aléatoires sont générés aussi lentement que un par seconde et qu'il n'est pas possible d'accélérer le processus de génération, il serait tolérable dans la plupart des applications à haute sécurité d'attendre occasionnellement 200 secondes.

Réaliser ces nombres est trivial. Cela peut être fait par une personne qui joue à pile ou face de façon répétée, et presque tous les processus fondés sur un matériel savent le faire plus rapidement.

3.2 Les matériels existants peuvent être utilisés pour l'aléatoire

Comme décrit ci-dessous, de nombreux ordinateurs sont livrés avec un matériel qui peut, avec quelques précautions, être utilisés pour générer des quantités vraiment aléatoires.

3.2.1 Utiliser une entrée son/vidéo existante

De nombreux ordinateurs sont construits avec des entrées qui numérisent une source analogique de la réalité, comme le son d'un microphone ou de l'entrée vidéo d'une caméra. L'"entrée" d'un numériseur de son sans source branchée ou d'une caméra avec l'obturateur de la lentille est essentiellement un bruit thermique. Si le système a assez de gain pour détecter quelque chose, une telle entrée peut fournir des bits aléatoires de qualité raisonnablement bonne. Cette méthode est extrêmement dépendante de la mise en oeuvre matérielle.

Par exemple, sur certains systèmes fondés sur UNIX, on peut lire à partir du dispositif /dev/audio alors que le cordon du microphone n'est pas branché ou alors que le microphone ne reçoit qu'un faible niveau de bruit de fond. De telles données sont essentiellement du bruit aléatoire, bien qu'on ne puisse s'y fier sans vérification, en cas de défaillance du matériel, et il faudra le débiaiser.

En combinant cette approche avec la compression pour débiaiser (voir la Section 4), on peut générer une grande quantité de données aléatoire de qualité moyenne avec la ligne de commande de style UNIX :

```
cat /dev/audio | compress →random-bits-file
```

Un examen détaillé de ce type de source d'aléa apparaît dans [TURBID].

3.2.2. Utiliser des pilotes de disque existants

Les pilotes de disques ont de faibles fluctuations aléatoires dans leur vitesse de rotation, dues à des turbulences chaotiques de l'air [DAVIS, Jakobsson]. L'addition des collections de temps de recherche de disque de bas niveau produit une série de mesures qui contiennent ce caractère aléatoire. De telles données sont habituellement très corrélées, aussi un traitement significatif est-il nécessaire, comme décrit au paragraphe 5.2 ci-dessous. Néanmoins, des expérimentations conduites il y a une dizaine d'années ont montré qu'avec un tel traitement, même des pilotes de disques lents sur les ordinateurs les plus lents de cette époque pouvaient facilement produire 100 bits à la minute, ou plus, d'excellentes données aléatoires.

Toute augmentation de la vitesse du processeur, qui augmente la résolution avec laquelle le mouvement du disque peut être appréhendé ou qui augmente le rythme de recherche du disque, augmente le débit de génération de bits aléatoires possible avec cette technique. Au moment de la rédaction du présent document et avec les matériels modernes, le débit typique de production de bits aléatoires dépasserait 10 000 par seconde. Cette technique est utilisée dans des générateurs de nombres aléatoires inclus dans de nombreuses bibliothèques de systèmes d'exploitation.

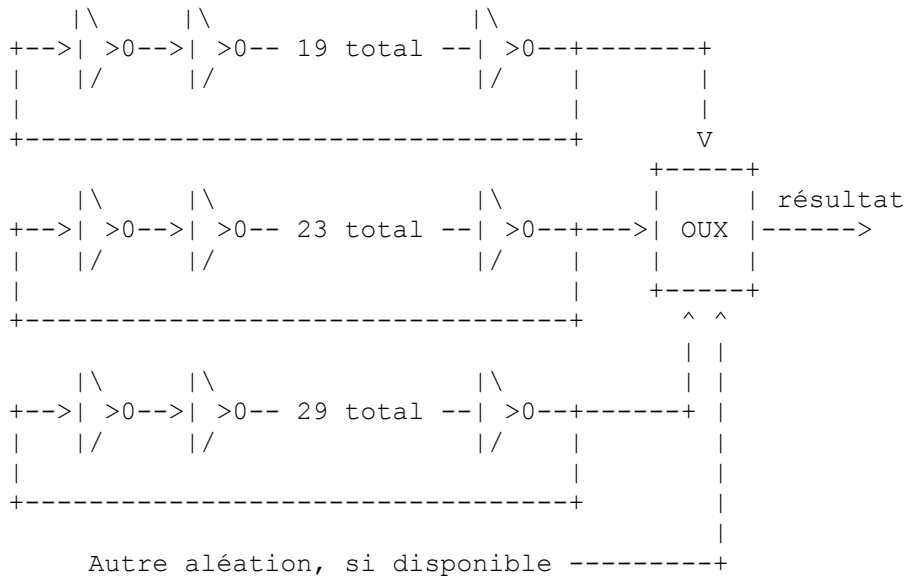
Note : L'inclusion d'antémémoires dans les contrôleurs de disque a peu d'effet sur cette technique si les temps de recherche très courts, qui représentent les entrées/sorties d'antémémoire, sont simplement ignorés.

3.3 Sources d'oscillateur en anneau

Si un circuit intégré est conçu ou programmé sur champ, un nombre impair de portes peuvent être connectées en série pour produire un oscillateur en anneau autonome. En échantillonnant un point de l'anneau à une fréquence fixée (par exemple, déterminée par un oscillateur cristal stable), une certaine quantité d'entropie peut être extraite du fait des variations de la fréquence de l'oscillateur autonome. Il est possible d'augmenter le débit d'entropie en combinant par opérateur XOR les valeurs échantillonnées à partir de quelques oscillateurs en anneau avec des longueurs premières entre elles. Il est parfois recommandé d'utiliser un nombre impair d'anneaux de façon que, même si les anneaux sont plus ou moins synchronisés entre eux, il y aura toujours des transitions binaires échantillonnées. Une autre source possible à échantillonner est la sortie d'une diode bruyante.

Les bits échantillonnés à partir de telles sources devront être fortement débiaisés, comme doivent l'être les temps de rotation d'un disque (voir la Section 4). Une étude d'ingénierie sera nécessaire pour déterminer la quantité d'entropie produite en fonction du concept particulier. Dans tous les cas, cela peut donner de bonnes sources dont le coût en matériel est trivial selon les normes modernes.

À titre d'exemple, IEEE 802.11i suggère le circuit ci-dessous, avec une attention particulière pour la conception de l'isolation des anneaux les uns des autres et des circuits d'horloge pour éviter une synchronisation inopportune, etc., et avec un post traitement important [IEEE_802.11i].



3.4 Problèmes avec les horloges et les séries de nombres

Les horloges d'ordinateur et autres valeurs similaires de système d'exploitation ou de matériel, fournissement significativement moins de bits réels d'imprévisibilité qu'il ne pourrait paraître d'après leurs spécifications.

Des essais ont été faits sur les horloges de nombreux systèmes, et il a été découvert que leur comportement peut varier grandement et de façon inattendue. Une version d'un système d'exploitation fonctionnant sur un ensemble de matériels peut actuellement fournir, disons une résolution d'une microseconde dans une horloge, alors qu'une configuration différente du "même" système peut toujours fournir les mêmes bits de moindre poids et ne compter dans les bits de poids forts qu'à une résolution très inférieure. Cela signifie que des lectures successives de l'horloge peuvent produire des valeurs identiques même si il s'est passé assez de temps pour que la valeur "aurait dû" changer sur la base de la résolution nominale de l'horloge. Il y a aussi des cas où des lectures fréquentes d'une horloge peuvent produire des valeurs séquentielles artificielles, à cause d'un code supplémentaire qui vérifie que l'horloge est restée inchangée entre deux lectures et l'augmente de un ! Concevoir un code d'application portable pour générer des nombres imprévisibles sur la base de telles horloges système est un vrai défi parce que le concepteur du système ne connaît pas toujours les propriétés de l'horloge système.

L'utilisation du numéro de série d'un matériel (comme une adresse MAC Ethernet) peu aussi fournir moins de bits d'unicité qu'on ne pourrait s'attendre. De telles quantités sont habituellement lourdement structurées, et les sous-champs peuvent n'avoir qu'une gamme limitée de valeurs possibles, ou les valeurs peuvent être facilement prévisibles sur la base d'une approximation de la date de fabrication ou d'autres données.

Par exemple, il est vraisemblable qu'une compagnie qui fabrique à la fois des ordinateurs et des adaptateurs Ethernet va, au moins en interne, utiliser son propre adaptateur, ce qui limite de façon significative la gamme des adresses incorporées.

Des problèmes tels que ceux décrits ci-dessus rendent la production de code pour générer des quantités imprévisibles difficile si le code doit être porté à travers diverses plateformes et systèmes informatiques.

3.5 Heure et valeur d'événements externes

Il est possible de mesurer la durée et le contenu des mouvements de la souris, de la frappe des touches, et événements d'usager similaires. C'est une source raisonnable de données imprévisibles, avec quelques qualifications. Sur certaines machines, des entrées comme la frappe des touches sont temporisées. Bien que l'intervalle de la frappe des touches par l'usager puisse avoir une variation et une imprévisibilité suffisantes, il pourrait n'être pas facile d'accéder à cette variation. Un autre problème est l'absence de méthode standard pour échantillonner les détails du déroulement dans le temps. Cela rend difficile l'utilisation de cette technique pour construire un logiciel standard destiné à la distribution sur une large gamme de machines.

La quantité de mouvement de la souris et la frappe réelle des touches sont habituellement d'un accès plus facile que les durées, mais elles donnent moins d'imprévisibilité parce que l'usager peut fournir des entrées extrêmement répétitives.

D'autres événements externes, comme le délai d'arrivée et la longueur des paquets du réseau, peuvent aussi être utilisés, mais seulement avec de grandes précautions. En particulier, la possibilité de manipulations de telles mesures du trafic réseau par un adversaire et le manque d'historique depuis le démarrage du système doivent être considérés avec attention. Si cette entrée est sujette à manipulation, elle ne peut être une source d'entropie fiable.

En principe, presque tout capteur externe, comme un radio récepteur brut ou un thermomètre sur les ordinateurs qui en sont équipés, peut être utilisé. Mais dans chaque cas, il faut prêter une extrême attention à la façon dont ces données pourraient être manipulées par un adversaire et à la quantité d'entropie qu'elles peuvent réellement fournir.

Les techniques ci-dessus sont assez puissantes contre les attaquants qui n'ont pas accès aux quantités mesurées. Par exemple, ces techniques seraient puissantes contre un attaquant hors ligne sans accès à notre environnement et qui essayerait de casser notre graine aléatoire après coup. Dans tous les cas, plus précisément on peut mesurer le rythme temporel ou la valeur d'un capteur externe, plus rapidement on peut générer les bits.

3.6 Sources d'aléa non matérielles

La meilleure source d'entropie d'entrée serait une source aléatoire fondée sur un matériel du genre oscillateur à anneau, temps d'accès d'un pilote de disque, bruit thermique, ou décomposition radioactive. Cependant, si aucune de celles là n'est disponible, il y a d'autres possibilités. Parmi elles on a les horloges système, les mémoires tampon système ou d'entrée/sortie, les numéros de série ou adresses et durées d'usager/système/matériel/réseau, et les entrée d'utilisateur. Malheureusement, chacune de ces sources peut produire des valeurs très limitées ou prévisibles dans certaines circonstances.

Certaines des sources citées ci-dessus devraient être assez fortes sur des systèmes à plusieurs utilisateurs, où chaque utilisateur du système est par nature une source d'aléa. Cependant, sur les petits systèmes à un seul utilisateur ou les systèmes incorporés, spécialement au démarrage, il devrait être possible à un adversaire d'assembler une configuration similaire. Cela peut donner à l'adversaire des entrées pour le processus de mixage qui seraient assez bien corrélées à celles utilisées à l'origine afin de pratiquer une recherche exhaustive.

L'utilisation de plusieurs entrées aléatoires avec une forte fonction de mixage est recommandée et peut surmonter la faiblesse de toute entrée particulière. Le rythme et le contenu de la frappe aléatoire de l'usager peut donner des centaines de bits aléatoires, mais des hypothèses prudentes doivent être faites. Par exemple, une hypothèse raisonnablement prudente serait qu'un intervalle entre la frappe des touches donne au plus quelques bits d'aléa, mais seulement quand l'intervalle est unique dans la séquence des intervalles jusqu'à ce point. Une hypothèse similaire serait qu'un code de clé donne quelques bits d'aléa, mais seulement lorsque le code est unique dans la séquence. Et donc, un intervalle ou code de clé qui duplique une valeur précédente devrait être supposé ne pas fournir d'aléa supplémentaire. Le résultat du mixage de ces durées avec la frappe des caractères pourrait être encore combiné à des valeurs d'horloge et d'autres entrées.

Cette stratégie peut donner un code portable pratique pour produire de bons nombres aléatoires pour la sécurité, même si certaines des entrées sont très faibles sur certains des systèmes cibles. Cependant, cela peut toujours échouer contre une attaque de haut niveau contre des petits systèmes à un seul utilisateur, ou incorporés, tout spécialement si l'adversaire a été à même d'observer dans le passé le processus de génération. Une source aléatoire fondée sur le matériel est encore préférable.

4. Débiaisage

Y a-t-il des exigences spécifiques sur la forme de la distribution des quantités rassemblées pour l'entropie pour produire les nombres aléatoires ? La bonne nouvelle est qu'il n'est pas nécessaire que la distribution soit uniforme. Tout ce qui est nécessaire pour encadrer les performances est une estimation de sa non uniformité. Des techniques simples pour débiaiser un flux binaire sont données ci-dessous, et des techniques cryptographiques plus fortes sont décrites au paragraphe 5.2.

4.1 Utilisation de la parité de flux pour débiaiser

À titre d'exemple simple mais pas particulièrement pratique, considérons une chaîne de bits suffisamment longue et transposons la chaîne en "zéros" ou "uns". La transposition ne donnera pas une distribution parfaitement uniforme, mais elle peut l'être d'aussi près qu'on le souhaite. Une transposition qui sert à notre objet est de prendre la parité de la chaîne. Cela a l'avantage d'être robuste à tous les degrés de biais jusqu'au biais estimé maximum et sa mise en œuvre matérielle est triviale.

L'analyse suivante donne le nombre de bits qui doivent être échantillonnés :

Supposons que le ratio de uns et de zéros est (0,5 + E) à (0,5 - E), où E est entre 0 et 0,5 et est la mesure de "l'excentricité" de la distribution. Considérons la distribution de la fonction de parité de N échantillons binaires. Les probabilités respectives que la parité soit un ou zéro seront la somme des termes impairs ou pairs dans le développement binomial de (p + q)^N, où p = 0,5 + E, la probabilité d'un un, et q = 0,5 - E, la probabilité d'un zéro.

Ces sommes peuvent être calculées facilement comme $1/2 * ((p + q)^N + (p - q)^N)$ et $1/2 * ((p + q)^N - (p - q)^N)$.

(Formule qui correspond à la probabilité que la parité soit 1 dépend de N impair ou pair.)

Comme p + q = 1 et p - q = 2E, ces expressions se réduisent à $1/2 * [1 + (2E)^N]$ et $1/2 * [1 - (2E)^N]$.

Aucune d'elle ne sera jamais exactement 0,5 à moins que E ne soit égal à zéro, mais on peut les amener arbitrairement proches de 0,5. Si nous voulons que les probabilités soient dans un écart delta d de 0,5, par exemple, alors $(0,5 + (0,5 * (2E)^N)) < 0,5 + d$.

Résoudre pour N donne N > log(2d)/log(2E). (Noter que 2E est inférieur à 1, donc son log est négatif. La division par un nombre négatif renverse le sens d'une inégalité.)

Le tableau suivant donne la longueur N de la chaîne qui doit être échantillonnée pour divers degrés de biais afin d'arriver à 0,001 d'une distribution 50/50.

Prob(1)	E	N
0,5	0,00	1
0,6	0,10	4
0,7	0,20	7
0,8	0,.30	13
0,9	0,40	28
0,95	0,45	59
0,99	0,49	308

La dernière entrée montre que même si la distribution est biaisée à 99 % en faveur des uns, la parité d'une chaîne de 308 échantillons sera dans les 0,001 d'une distribution 50/50. Mais, comme nous allons le voir au paragraphe 5.2, il y a des techniques bien plus fortes qui extraient plus de l'entropie disponible.

4.2 Utilisation de transpositions de transition pour débiaiser

Une autre technique, due à l'origine à von Neumann [VON_NEUMANN], est d'examiner un flux binaire comme une séquence de paires sans chevauchement. On peut alors éliminer toute paire 00 ou 11 trouvée, interpréter 01 comme 0 et 10 comme 1. Supposant que la probabilité d'un 1 est 0,5 + E et que la probabilité d'un 0 est 0,5 - E, où E est l'excentricité de la source comme décrit au paragraphe précédent. La probabilité de chaque paire est alors donnée dans le tableau suivant :

paire	**probabilité**
00	(0,5 - E)^2 = 0,25 - E + E^2
01	(0,5 - E)*(0,5 + E) = 0,25 - E^2
10	(0,5 + E)*(0,5 - E) = 0,25 - E^2
11	(0,5 + E)^2 = 0,25 + E + E^2

Cette technique va complètement éliminer tout biais mais exige un nombre indéterminé de bits d'entrée pour tout nombre particulier de bits de sortie désiré. La probabilité que toute paire particulière soit éliminée est de 0,5 + 2E^2, de sorte que le nombre espéré de bits d'entrée pour produire X bits de résultat est X/(0,25 - E^2).

Cette technique suppose que les bits viennent d'un flux où chaque bit a la même probabilité d'être un 0 ou un 1 comme tout

autre bit dans le flux et que les bits ne sont pas corrélés, c'est-à-dire, que les bits viennent de distributions identiquement indépendantes. Si des bits de remplacement viennent de deux sources corrélées, l'analyse ci-dessus ne tient plus.

La technique ci-dessus fournit aussi une autre illustration de la façon dont une simple analyse statistique peut être trompeuse si on ne surveille pas les schémas qui pourraient être exploités par un adversaire. Si l'algorithme est légèrement mal lu et qu'on utilise des paires de bits successives en chevauchement au lieu de paires sans chevauchement, l'analyse statistique donnée serait la même. Cependant, au lieu de fournir une série non biaisée de 1 et de 0 non corrélés, elle produirait une séquence totalement prévisible de 1 et de 0 alternés avec exactitude.

4.3. Utilisation de FFT pour débiaiser

Lorsque des données réelles consistent en bits fortement corrélés, elles peuvent encore contenir des quantités utiles d'entropie. Cette entropie peut être extraite par diverses transformations, dont les plus puissantes sont décrites ci-dessous au paragraphe 5.2.

Il est intéressant d'utiliser la transformée de Fourier des données, ou sa variante optimisée, la FFT, pour des raisons principalement théoriques. On peut montrer que cette technique va éliminer les fortes corrélations. Si des données adéquates sont traitées et si les corrélations restantes se dissolvent, des lignes spectrales qui approchent de l'indépendance statistique et un aléa à distribution normale, peuvent être produites [BRILLINGER].

4.4 Utilisation de la compression pour débiaiser

Des techniques de compression réversibles fournissent aussi une méthode grossière de débiaisage d'un flux binaire biaisé. Cela découle directement de la définition de la compression réversible et de la formule de la Section 2 sur la quantité d'information dans une séquence. Comme la compression est réversible, la même quantité d'information doit être présente dans la plus petite sortie comme elle était présente dans la plus longue entrée. D'après l'équation de Shannon sur l'information, ceci n'est possible que si, en moyenne, les probabilités des différentes séquences les pus petites sont plus uniformément distribuées que ne l'étaient les probabilités des plus longues séquences. Donc, les plus petites séquences doivent être débiaisées par rapport à l'entrée.

Cependant, de nombreuses techniques de compression ajoutent une préface quelque peu prévisible à leur flux de sortie et peuvent insérer périodiquement une séquence similaire dans leur sortie ou introduire autrement des schémas subtils de leur cru. Elles devraient être considérées seulement comme des techniques grossières comparées à celles décrites au paragraphe 5.2. Au minimum, le commencement de la séquence compressée devrait être sauté et seuls les bits ultérieurs devraient être utilisés pour les applications qui requièrent des bits grossièrement aléatoires.

5 Mixage

Quelle est la meilleure stratégie globale pou obtenir de nombres aléatoires imprévisibles en l'absence d'une forte source d'entropie matérielle fiable ? C'est d'obtenir des entrées d'un certain nombre de sources non corrélées et de les mélanger à l'aide d'une forte fonction de mixage. Une telle fonction préservera l'entropie présente dans toutes les sources, même si d'autres quantités combinées arrivent à être fixées ou aisément prévisibles (faible entropie). Cette approche peut être conseillée même avec une bonne source matérielle, car le matériel peut lui aussi avoir des défaillances. Cependant, il faut peser ceci avec soin par rapport à l'accroissement possible des chances d'une défaillance générale due à la complexité logicielle supplémentaire.

Une fois qu'on a de bonnes sources, comme certaines de celles énumérées à la Section 3, et qu'on les a mixées comme décrit dans cette section, on a une graine forte. Elle peut maintenant être utilisée pour produire de grandes quantités de matériel cryptographique fort, comme décrit dans les Sections 6 et 7.

Une fonction de mixage forte est celle qui combine les entrées et produit un résultat dans lequel chaque bit de sortie est une fonction non linéaire complexe différente de tous les bits d'entrée. En moyenne, changer tout bit d'entrée va changer environ la moitié des bits de sortie. Mais comme la relation est complexe et non linéaire, il n'est pas garanti qu'un bit de sortie particulier change quand un bit d'entrée particulier est changé.

Considérons le problème de la conversion d'un flux binaire qui est biaisé vers 0 ou 1 ou qui a un schéma quelque peu prévisible en un flux plus court qui serait plus aléatoire, comme exposé à la Section 4. C'est simplement un autre cas où une forte fonction de mixage est désirée, pour mixer les bits d'entrée et produire un plus petit nombre de bits de sortie. La

technique donnée au paragraphe 4.1, qui utilise la parité d'un nombre de bits, est simplement le résultat de leurs combinaisons XOR successives. Ceci est examiné comme une fonction de mixage triviale immédiatement ci-après. L'utilisation de plus fortes fonctions de mixage pour extraire plus d'aléa dans un flux de bits biaisés est examinée au paragraphe 5.2. Voir aussi [NASLUND].

5.1 Fonction de mixage triviale

Pour les besoins de l'exposé nous décrivons un exemple trivial d'entrées d'un seul bit utilisant la fonction OU Exclusif (XOR). Cette fonction est équivalente à l'addition sans retenue, comme montré dans le tableau ci-dessous. C'est un cas simplifié dans lequel le bit de sortie change toujours pour un changement d'un bit d'entrée quelconque. Mais, en dépit de sa simplicité, il donne une illustration utile.

entrée 1	**entrée 2**	**sortie**
0	0	0
0	1	1
1	0	1
1	1	0

Si les entrées 1 et 2 ne sont pas corrélées et sont combinées de cette façon, la sortie sera alors un bien meilleur bit aléatoire (moins biaisé) que ne le sont les entrées. Si on suppose une "excentricité" E comme défini au paragraphe 4.1 ci-dessus, l'excentricité de sortie se rapporte comme suit à l'excentricité d'entrée :

$$E_{sortie} = 2 * E_{entrée\ 1} * E_{entrée\ 2}$$

Comme E n'est jamais supérieur à 1/2, l'excentricité est toujours améliorée, excepté dans le cas où au moins une entrée est une constante totalement biaisée. Ceci est illustré dans le tableau suivant, où les valeurs du haut et du côté gauche sont les deux excentricités d'entrée et les entrées sont l'excentricité de sortie :

E	0,00	0,10	0,20	0,30	0,40	0,50
0,00	0,00	0,00	0,00	0,00	0,00	0,00
0,10	0,00	0,02	0,04	0,06	0,08	0,10
0,20	0,00	0,04	0,08	0,12	0,16	0,20
0,30	0,00	0,06	0,12	0,18	0,24	0,30
0,40	0,00	0,08	0,16	0,24	0,32	0,40
0,50	0,00	0,10	0,20	0,30	0,40	0,50

Cependant, noter que les calculs ci-dessus supposent que les entrées ne sont pas corrélées. Si les entrées l'étaient, par exemple, la parité du nombre de minutes de minuit à deux horloges précises à quelques secondes près, chacune peut paraître aléatoire si elles sont échantillonnées à des intervalles aléatoires beaucoup plus longs que la minute. Et si elles sont toutes deux échantillonnées et combinées par la fonction XOR, le résultat sera zéro la plupart du temps.

5.2 Plus fortes fonctions de mixage

La norme de chiffrement évolué [AES] du gouvernement américain est un exemple de fonction de mixage forte pour plusieurs quantités binaires. Elle prend jusqu'à 384 bits d'entrée (128 bits de "données" et 256 bits de "clé") et produit 128 bits de sortie, dont chacun dépend d'une fonction complexe non linéaire des tous les bits d'entrée. D'autres fonctions de chiffrement qui ont cette caractéristique, comme [DES], peuvent aussi être utilisées pour mixer toutes leurs clés et leurs bits d'entrée de données.

Une autre bonne famille de fonctions de mixage est le "résumé de message" ou les fonctions de hachage telles que les normes de hachage sécurisé [SHA*] *(Secure Hash Standards)* du gouvernement américain et la série des [MD4, MD5]. Ces fonctions prennent toutes une quantité pratiquement illimitée d'entrées et produisent un résultat relativement court de longueur fixe qui mixe tous les bits d'entrée. La série MD* produit des sorties de 128 bits, SHA-1 produit 160 bits, et les autres fonctions SHA produisent jusqu'à 512 bits.

Bien que les fonctions de résumé de message soient conçues pour des quantités d'entrées variables, AES et les autres fonctions de chiffrement peuvent aussi être utilisées pour combiner un nombre quelconque d'entrées. Si 128 bits de sortie est adéquat, les entrées peuvent être empaquetées en quantités de 128 bits de données et de "clés" AES successives, en bourrant de zéros si nécessaire ; la quantité est ensuite chiffrée par les "clés" en utilisant AES en mode codet électronique *(Electronic Codebook)*. Autrement, l'entrée peut être empaquetée dans une clé de 128 bits et plusieurs blocs de données et un CBC-MAC peut être calculé [MODES].

Un mixage plus complexe devrait être utilisé si plus de 128 bits de sortie sont nécessaires et qu'on veut employer AES (mais noter qu'il est absolument impossible d'obtenir plus de bits "d'aléa" en sortie qu'on en a mis en entrée). Par exemple,

supposons que les entrées sont enveloppées dans trois quantités, A, B, et C. On peut utiliser AES pour chiffrer A avec B puis avec C comme clés pour produire la première partie du résultat, puis chiffrer B avec C et ensuite A pou plus de résultat et, si nécessaire, chiffrer C avec A et puis B pour encore plus de résultat. On peut encore produire plus de résultat en inversant l'ordre des clés donné ci-dessus. On peut faire la même chose avec des fonctions de hachage, en hachant divers sous-ensembles des données d'entrée ou des copies différentes des données d'entrée avec des préfixes différents et/ou des suffixes pour produire plusieurs sorties.

À titre d'exemple d'utilisation d'une fonction de mixage forte, reconsidérons le cas d'une chaîne de 308 bits, dont chacun est biaisé à 99 % en faveur du zéro. La technique de parité donnée au paragraphe 4.1 réduit cela à un bit, avec seulement une déviance de 1/1000 d'une probabilité égale de zéro ou de un. Mais, en appliquant l'équation de l'information donnée à la Section 2, cette séquence biaisée de 308 bits contient plus de 5 bits d'information. Et donc, en la hachant avec SHA-1 et en prenant les 5 bits du bas, le résultat donnerait 5 bits aléatoires non biaisés et pas le seul bit donné par le calcul de parité de la chaîne. Autrement, pour certaines applications, on peut utiliser le résultat du hachage tout entier pour retenir presque tous les 5+ bits d'entropie dans une quantité de 160 bits.

5.3 Utilisation de boîtes de substitution pour le mixage

De nombreuses fonctions de chiffrement de bloc modernes, y compris DES et AES, incorporent des modules appelés S-Boxes (boîtes de substitution). Elles produisent un plus petit nombre de sorties à partir d'un plus grand nombre d'entrées, à travers une fonction de mixage complexe non linéaire qui a pour effet de concentrer l'entropie limitée des entrées dans la sortie.

Les S-Boxes incorporent parfois des fonctions booléennes coudées (fonctions d'un nombre pair de bits produisant un bit de sortie avec une non linéarité maximum). En cherchant dans les sorties toutes les paires d'entrées différant par une position binaire particulière, exactement la moitie des sorties sont différentes. Une S-Box dans laquelle chaque bit de sortie est produit par une fonction coudée telle que toute combinaison linéaire de ces fonctions soit aussi une fonction coudée est appelée une "S-Box parfaite".

Les S-boxes et diverses applications répétées ou cascades de telles boîtes peuvent être utilisées pour le mixage [SBOX1, SBOX2].

5.4 Diffie-Hellman comme fonction de mixage

L'échange de clés exponentiel Diffie-Hellman est une technique qui donne un secret partagé entre deux parties. Il peut être impossible pour un tiers de déterminer ce secret par le calcul même si il peut observer tous les messages entre les deux parties à la communication. Ce secret partagé est un mélange des quantités initiales générées par chacune des parties [D-H].

Si ces quantités initiales sont aléatoires et non corrélées, le secret partagé combine alors leur entropie mais, bien sûr, il ne peut pas produire plus d'aléa que la taille du secret partagé généré.

Bien qu'il soit vrai que le calcul Diffie-Hellman soit effectué en privé, un adversaire qui peut observer une des clés publiques et connaît le module utilisé a seulement besoin de chercher dans l'espace de l'autre clé secrète afin d'être capable de calculer le secret partagé [D-H]. Aussi, à titre conservatoire, serait il préférable de considérer le Diffie-Hellman public comme produisant une quantité dont la prévisibilité correspond à la pire des deux entrées. À cause de cela et du fait que Diffie-Hellman est un gros calcul, son utilisation comme fonction de mixage n'est pas recommandée.

5.5 Utilisation d'une fonction de mixage pour étirer les bits aléatoires

Bien qu'il ne soit pas nécessaire pour une fonction de mixage de produire autant ou moins de bits de résultat que son entrée, les bits de mixage ne peuvent pas "étirer" le montant d'imprévisibilité aléatoire présente dans les entrées. Et donc, quatre entrées de 32 bits chacune, dans lesquelles sont 12 bits imprévisibles (comme 4 096 valeurs également probables) dans chaque entrée, ne peuvent pas produire plus de 48 bits imprévisibles en sortie. La sortie peut être étendue sur des centaines ou des milliers de bits, par exemple, en les mixant avec des entiers successifs, mais l'espace de recherche de l'adversaire habile est toujours de 2^{48} possibilités. De plus, le mixage à moins de bits qu'il n'y a d'entrées tendra à renforcer le caractère aléatoire du résultat.

Le dernier tableau du paragraphe 5.1 montre que le mixage d'un bit aléatoire avec un bit constant par OU Exclusif va produire un bit aléatoire. Bien que ce soit vrai, cela ne donne pas le moyen d'"étirer" un bit aléatoire en plus d'un. Si, par exemple, un bit aléatoire est mixé avec un 0 puis avec un 1, cela produit une séquence de deux bits mais ce sera toujours 01 ou 10. Comme il n'y a que deux valeurs possibles, il n'y a toujours que le seul bit aléatoire de l'origine.

5.6 Autres facteurs du choix d'une fonction de mixage

Pour une utilisation locale, AES présente l'avantage d'avoir été soumis à de nombreux essais de recherche de fautes, d'avoir un logiciel raisonnablement efficace, et d'être largement documenté et mis en oeuvre avec des matériels et logiciels disponibles dans le monde entier y compris avec un code source libre. La famille SHA* a été un peu moins étudiée et tend à exiger plus de cycles de CPU qu'AES mais ce n'est pas une raison pour croire qu'il y a des fautes. SHA* et MD5 sont tous deux dérivés de l'algorithme MD4 antérieur. Leur code source est disponible [SHA*, MD4, MD5]. Quelques signes de faiblesse ont été trouvés dans MD4 et MD5. En particulier, MD4 n'a que trois tours et il y a plusieurs coupures indépendantes des deux premiers ou des deux derniers tours. Et des collisions ont été trouvées dans le résultat de MD5.

AES a été choisi par un processus international robuste et public. Lui et SHA* ont été garantis par l'agence nationale de la sécurité des USA (NSA) sur la base de critères qui restent pour la plupart secrets, comme c'était le cas pour DES. Bien que cela ait été la cause de beaucoup de spéculations et de doutes, les recherches sur DES au fil des ans ont indiqué que l'implication de la NSA dans les modifications de son concept, dont l'origine est chez IBM, était principalement de la renforcer. Il n'y a eu aucune annonce des faiblesses cachées ou particulières trouvées dans DES. Il est vraisemblable que les modifications apportées par la NSA à MD4 pour produire les algorithmes SHA ont de même renforcé ces algorithmes, éventuellement contre des menaces qui ne sont pas encore connues de la communauté cryptographique publique.

Lorsque les longueurs d'entrée sont imprévisibles, les algorithmes de hachage sont d'utilisation plus pratique que les algorithmes de chiffrement de bloc car ils sont généralement conçus pour accepter des entrées de longueur variable. Les algorithmes de chiffrement de bloc exigent généralement un algorithme de bourrage supplémentaire pour s'accommoder des entrées qui ne sont pas un multiple pair de la taille de bloc.

Au moment de la rédaction du présent document, les auteurs n'ont eu connaissance d'aucune revendication de brevets sur les algorithmes de base de AES, DES, SHA*, MD4, et MD5 autres que les brevets pour lesquels une licence irrévocable sans redevance a été accordée au monde entier. Il peut, bien sûr, y avoir des brevets essentiels dont les auteurs ne sont pas avertis ou des brevets sur des mises en œuvre ou utilisation ou d'autres brevets pertinents produits ou à produire.

6. Générateurs de nombres pseudo aléatoires

Lorsqu'une graine a une entropie suffisante, à partir de l'entrée comme décrit à la Section 3 et éventuellement débiaisée et mixée comme décrit aux Sections 4 et 5, on peut étendre cette graine par un algorithme pour produire un grand nombre de quantités aléatoires cryptographiquement fortes. De tels algorithmes sont indépendants de la plateforme et peuvent fonctionner de la même façon sur tout ordinateur. Pour que les algorithmes soient sûrs, leur entrée et le fonctionnement interne doivent être protégés de l'observation des adversaires.

La conception de tels algorithmes de génération de nombres pseudo aléatoires, comme le concept des algorithmes de chiffrement symétriques, n'est pas une tâche d'amateurs. Le paragraphe 6.1 ci-dessous établit une liste des mauvaises idées utilisées qui ont été utilisées par des algorithmes qui ont échoué. Pour savoir ce qui marche sautez le paragraphe 6.1 et lisez le reste de cette section et la Section 7, qui décrit certains algorithmes de génération de nombre pseudo aléatoire et donne leurs références. Voir la Section 7 et la Partie 3 de [X9.82].

6.1 Quelques mauvaises idées

Les paragraphes ci-dessous décrivent un certain nombre d'idées qui peuvent paraître raisonnables mais conduisent à une génération non sûre de nombre pseudo aléatoire.

6.1.1 L'illusion des manipulations complexes

Une approche qui peut donner une apparence trompeuse d'imprévisibilité est de prendre un algorithme très complexe (ou un excellent générateur traditionnel de nombres pseudo aléatoires avec de bonnes propriétés statistiques) et de calculer une clé cryptographique en commençant par des données limitées telles que la valeur de l'horloge système de l'ordinateur comme graine. Les adversaires qui savent en gros l'heure de lancement du générateur auront un nombre relativement faible de valeurs de graine à tester, car ils vont connaître les valeurs vraisemblables de l'horloge système. Un grand nombre de bits pseudo aléatoires peuvent bien être générés, mais l'espace de recherche qu'un adversaire devra explorer pourrait être assez petit.

Et donc, les manipulations très fortes ou très complexes de données ne serviront à rien si l'adversaire peut apprendre quelle est la manipulation et si il n'y a pas assez d'entropie dans la valeur de départ de la graine. Il peut habituellement utiliser le nombre limité de résultats découlant d'un nombre limité de valeurs de graine pour battre en brèche la sécurité.

Une autre erreur stratégique sérieuse est de supposer qu'un algorithme de génération de nombres pseudo aléatoires très complexe va produire de forts nombres aléatoires, alors qu'il n'y a pas eu de théorisation ou d'analyse de l'algorithme. Il y

a un excellent exemple de cette erreur vers le début du chapitre 3 de [KNUTH], où l'auteur décrit un algorithme complexe. Il était prévu que le programme de langage machine correspondant à l'algorithme serait si compliqué qu'une personne essayant de lire le code sans commentaires ne saurait pas ce que fait le programme. Malheureusement, l'utilisation réelle de cet algorithme a montré qu'il convergeait presque immédiatement vers une seule valeur répétée dans un cas et un petit cycle de valeurs dans un autre cas.

Non seulement les manipulations complexes ne sont d'aucune aide pour une gamme limitée de graines, mais choisir aveuglément une manipulation complexe peut détruire l'entropie dans une bomme graine !

6.1.2 L'illusion du choix dans une grosse base de données

Une autre approche qui peut donner une apparence trompeuse d'imprévisibilité est de choisir au hasard une quantité dans une base de données et de supposer que sa force est en rapport avec le nombre total de bits de la base de données. Par exemple, les serveurs USENET normaux traitent de nombreux mégaoctets d'informations par jour [USENET_1, USENET_2]. Supposons qu'on choisisse une quantité aléatoire en allant chercher 32 octets de données à partir d'un point de départ choisi au hasard dans ces données. Cela ne donne pas 32*8 = 256 bits d'imprévisibilité. Même si beaucoup des données sont en langage humain qui ne contient pas plus de 2 ou 3 bits d'informations par octet, cela ne donne pas 32*2 = 64 bits d'imprévisibilité. Pour un adversaire qui a accès à la même base de données Usenet, l'imprévisibilité ne réside que dans le point de départ de la sélection. Ce qui est peut-être un petit peu plus qu'un couple de douzaines de bits d'imprévisibilité.

Le même argument s'applique à la sélection de séquences dans les données d'un enregistrement sur CD/DVD disponible au public ou autre base de données accessible à un large public. Si l'adversaire a accès à la même base de données, cette étape de "sélection à partir d'un large volume de données" ne vaut rien. Cependant, si on peut faire une sélection dans les données, à laquelle l'adversaire n'a pas accès, comme une mémoire tampon système sur un système actif multi-utilisateurs, cela peut aider.

6.1.3. Séquences pseudo aléatoires traditionnelles

Ce paragraphe parle des sources traditionnelles de nombres déterministes ou "pseudo aléatoires". Ils commencent normalement par une quantité "graine" et utilisent des opérations numériques ou logiques simples pour produire une séquence de valeurs. Noter qu'aucune des techniques exposées dans ce paragraphe ne convient à une utilisation cryptographique. Elles sont présentées dans un but d'information générale.

[KNUTH] fait un exposé classique sur les nombres pseudo aléatoires. Les applications qu'il mentionne sont des simulations des phénomènes naturels, d'échantillonnage, d'analyse numérique, d'essais de programmes d'ordinateur, de prise de décision, et de jeux. Aucun d'eux n'a les mêmes caractéristiques que les utilisations de sécurité dont nous parlons. Seuls dans les deux derniers pourrait se trouver un adversaire qui essaye de trouver la quantité aléatoire. Cependant, dans ces cas, l'adversaire a normalement une seule chance d'utiliser une valeur prévisible. Pour prévoir les mots de passe ou essayer de casser le schéma de chiffrement, l'adversaire a normalement beaucoup de chances, peut-être un nombre illimité, de prévoir la valeur correcte. Parfois, l'adversaire peut mémoriser le message à casser et l'attaquer de façon répétée. Les adversaires sont aussi censés s'aider d'un ordinateur.

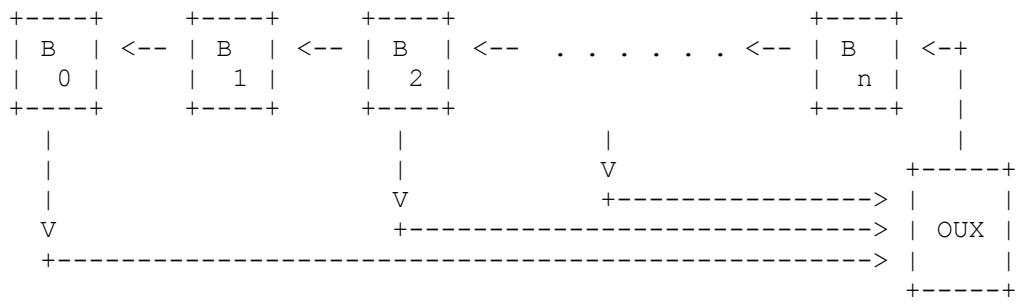
Pour tester le caractère aléatoire des nombres, Knuth suggère diverses mesures, y compris statistiques et spectrales. Ces essais vérifient des choses comme l'auto corrélation entre différentes parties d'une séquence "aléatoire" ou la distribution de ses valeurs. Mais ces essais peuvent être satisfaits par une séquence aléatoire mémorisée constante, telle que la séquence "aléatoire" publiées dans les Tableaux mathématiques standard CRC [CRC]. En dépit du fait de satisfaire à tous les essais suggérés par Knuth, cette séquence ne convient pas pour une utilisation cryptographique, car les adversaires doivent être supposés avoir des copies de toutes les séquences "aléatoires" couramment publiées et être capables de cibler la source et prédire ses valeurs futures.

Une technique typique de génération de nombre pseudo aléatoire est le générateur de nombre pseudo aléatoire à congruence linéaire. Cette technique utilise une arithmétique modulaire, où la valeur numérotée N+1 est calculée à partir de la valeur numérotée N par

$$V_{N+1} = (V_N * a + b)(\text{Mod } c)$$

La technique ci-dessus a une forte relation avec les générateurs de nombre pseudo aléatoire à registre à décalage linéaire, qui sont bien compris cryptographiquement [SHIFT*]. Dans de tels générateurs, les bits sont introduits à une extrémité d'un registre à décalage comme le OU Exclusif (OUX, somme binaire sans report) de bits d'entrées choisies dans le registre.

Par exemple, considérons ce qui suit :



$$V_{N+1} = ((V_N * 2) + B_0 \text{ XOR } B_2 \ldots) (\text{Mod } 2^n)$$

Les qualités des algorithmes traditionnels de générateur de nombre pseudo aléatoire sont mesurées par des essais statistiques sur de telles séquences. Des valeurs choisies avec soin a, b, c, et V initial ou un placement choisi avec soin de l'entrée de registre à décalage dans le processus simple ci-dessus peuvent produire d'excellentes statistiques.

Ces séquences peuvent être adéquates dans des simulations (expériences Monte Carlo) tant que la séquence est orthogonale à la structure de l'espace à explorer. Même là, des schémas subtils peuvent causer des problèmes. Cependant, de telles séquences sont clairement mauvaises dans des utilisations d'applications de sécurité. Elles sont complètement prévisibles si l'état initial est connu. Selon la forme du générateur de nombres pseudo aléatoires, la séquence peut être déterminée à partir de l'observation d'une faible portion de la séquence [SCHNEIER, STERN]. Par exemple, avec les générateurs ci-dessus, on peut déterminer V(n+1) ce qui donne la connaissance de V(n). En fait, il a été montré qu'avec ces techniques, même si seulement un bit des valeurs pseudo aléatoires est livré, la graine peut être déterminée à partie de courtes séquences.

Non seulement les générateurs à congruence linéaire ont été cassés, mais les techniques pour casser tous les générateurs à congruence polynomiale sont maintenant connues [KRAWCZYK].

6.2 Séquences cryptographiquement fortes

Dans les cas où une série de quantités aléatoires doit être générée, un adversaire peut apprendre quelques valeurs dans la séquence. En général, les adversaires ne devraient pas être capables de prédire d'autres valeurs à partir de celles qu'ils connaissent.

La technique correcte est de commencer par une forte graine aléatoire, pour prendre des positions cryptographiquement forte à partir de cette graine [FERGUSON, SCHNEIER], et de ne pas révéler l'état complet du générateur dans les éléments de la séquence. Si chaque valeur de la séquence peut être calculée d'une façon fixée par la valeur précédente, alors quand n'importe quelle valeur est compromise, toutes les valeurs futures peuvent être déterminées. Cela serait le cas, par exemple, si chaque valeur est une fonction constante des valeurs utilisées précédemment, même si la fonction est une fonction très forte de résumé de message non réversible.

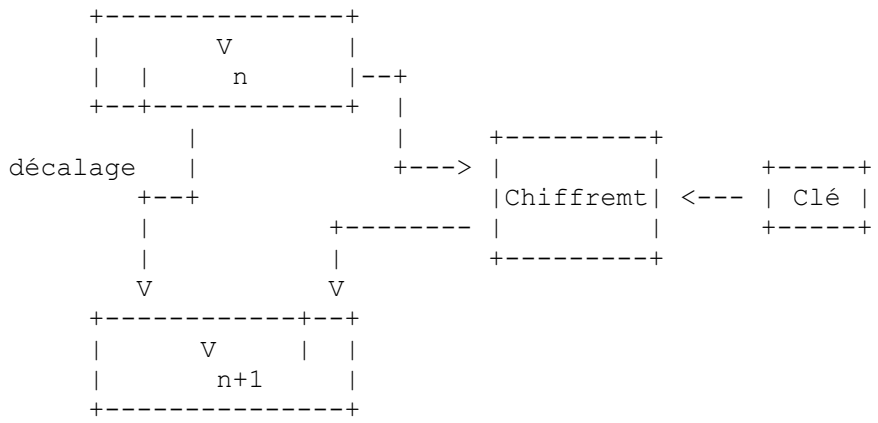
(Noter que si une technique de génération de séquence de valeurs de clés est assez rapide, elle peut être utilisée de façon triviale comme base d'un système de confidentialité. Si deux parties utilisent la même technique de génération de séquence et débutent avec le même matériel de graine, elles vont générer des séquences identiques. Cela pourrait, par exemple, être combiné par opérateur OUX à une extrémité avec des données envoyées pour les chiffrer, et combiné par opérateur OUX avec ces données telles que reçues pour les déchiffrer, du fait des propriétés réversibles de l'opération OUX. On appelle généralement ceci un chiffrement de flux simple.)

6.2.1 Séquences OFB et CTR

Une façon de produire une séquence forte est de prendre une valeur de graine et de hacher les quantités produites par concaténation de la graine avec des entiers successifs, ou quelque chose de ce genre, et de masquer les valeurs obtenues de façon à limiter la quantité d'état de générateur disponible pour l'adversaire.

Il peut aussi être possible d'utiliser un algorithme de "chiffrement" avec une clé aléatoire et une valeur de graine pour chiffrer des entiers successifs, comme dans le chiffrement en mode compteur (CTR). Autrement, on peut faire rétroagir toutes les valeurs de résultat provenant du chiffrement sur la valeur à chiffrer pour la prochaine itération. C'est un exemple particulier de mode de rétroaction sur le résultat (OFB, *output feedback*) [MODES].

Un exemple est donné ci-dessous dans lequel on utilise le décalage et le masquage pour combiner une partie de la rétroaction de sortie avec une partie de la vieille entrée. Ce type de rétroaction partielle devrait être évité pour les raisons décrites ci-dessous.



Noter que si un décalage de un est utilisé, c'est la même chose que la technique du registre à décalage décrite au paragraphe 6.1.3, mais avec la très importante différence que la rétroaction est déterminée par une fonction complexe non linéaire de tous les bits plutôt que par une simple combinaison linéaire ou polynomiale de sorties provenant de quelques sources de positions binaires.

Donald W. Davies a montré que cette sorte de rétroaction partielle de sortie décalée affaiblit significativement un algorithme, par rapport à l'alimentation en retour de tous les bits de sortie comme entrée. En particulier, pour DES, le chiffrement répété d'une quantité complète de 64 bits va donner une répétition attendue environ toutes les 2^63 itérations. Rétroagir avec qui que ce soit de moins de 64 bits (et de plus de 0) va donner une répétition attendue entre 2^31 et 2^32 itérations !

Pour prédire les valeurs d'une séquence à partir d'autres quand la séquence a été générée par ces techniques est équivalent à casser le crypto système ou à inverser le hachage "non réversible" avec seulement des informations partielles disponibles. Moins il y a d'information révélée à chaque itération, pus dur ce sera pour un adversaire de prédire la séquence. Et donc il est meilleur de n'utiliser qu'un bit provenant de chaque valeur. Il a été démontré que dans certains cas, cela rend impossible de casser un système même lorsque le système cryptographique est réversible et pourrait être cassé si chaque valeur générée était révélée.

6.2.2 Le générateur de séquence Blum Blum Shub

Le générateur qui a actuellement la plus forte preuve publique de force est appelé générateur Blum Blum Shub, d'après le nom de ses inventeurs [BBS]. Il est aussi très simple et se fonde sur des résidus quadratiques. Son seul désavantage est qu'il est gourmand en calcul comparé aux techniques traditionnelles données au paragraphe 6.1.3. Ce n'est pas un inconvénient majeur si il est utilisé pour des besoins d'une fréquence relativement modérée, comme ceux de la génération de clés de session.

Il suffit de choisir deux grands nombres premiers (disons, p et q) qui donnent chacun un reste de 3 lorsque on les divise par 4. Soit n = p * q. Choisissons ensuite un nombre aléatoire, x, qui est premier par rapport à n. La graine initiale pour le générateur et la méthode de calcul des valeurs suivantes sont alors

$$s_0 = (x^2)(\text{Mod } n)$$
$$s_{i+1} = (s_i^2)(\text{Mod } n)$$

Veillez à n'utiliser que quelques bits du bas de chaque s. Il est toujours plus sûr de n'utiliser que les bits de moindre poids. Si on n'utilise pas plus que les $\log_2(\log_2(s_i))$ bits de moindre poids, la prédiction de tout bit additionnel d'une séquence générée de cette manière est de manière prouvée aussi dure que de factoriser n. Tant que le x initial est secret, n peut être rendu public si on le souhaite.

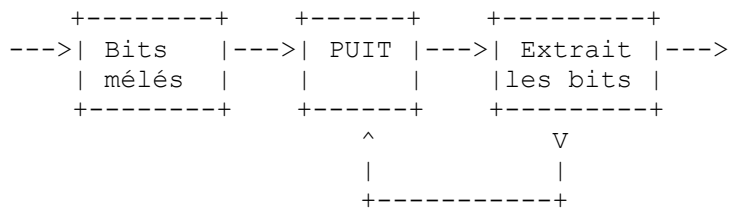
Une caractéristique intéressante de ce générateur est que toutes les valeurs s peuvent être calculées directement. En particulier,

$$s_i = (s_0^{((2\hat{}i) \text{ (Mod } ((p-1)*(q-1)))))} (\text{Mod } n)$$

Cela signifie que dans les applications où de nombreuses clés sont générées de cette façon, il n'est pas nécessaire de les sauvegarder toutes. Chaque clé peut être effectivement indexée et récupérée à partir de ce petit indice et des s et n initiaux.

6.3 Techniques du puits d'entropie

De nombreuses sources de nombres pseudo aléatoires modernes, telles que celles décrites aux paragraphes 7.1.2 et 7.1.3 utilisent la technique de l'entretien d'un "puits" de bits et de la fourniture d'opérations pour des entrées fortement mixées avec de l'aléa dans le puits et l'extraction de bits pseudo aléatoire du puits. Ceci est illustré à la figure suivante.



Les bits à introduire dans le puits peuvent venir de toutes les sources diverses de matériel, d'environnement ou d'utilisateur discutées plus haut. Il est aussi de pratique courante de sauvegarder l'état du puits à la fermeture du système et de le restaurer au redémarrage, quand une mémorisation stable est disponible.

Il faut aussi veiller à ce que suffisamment d'entropie ait été ajoutée au puits pour prendre en charge les utilisations particulières de sortie désirées. Voir dans [RSA_BULL1] des suggestions similaires.

7 Exemples et normes de génération aléatoire

Plusieurs normes publiques et exemples largement répandus existent maintenant pour la génération de clés ou autres quantités cryptographiques aléatoires. Certaines, au paragraphe 7.1, comportent une source d'entropie. D'autres, décrites au paragraphe 7.2, fournissent le générateur de forte séquence de nombres pseudo aléatoires mais supposent l'entrée d'une graine aléatoire ou une entrée provenant d'une source d'entropie.

7.1 Générateurs complètement aléatoires

Trois normes sont décrites ci-après. Les deux normes les plus anciennes utilisent DES, avec son bloc de 64 bits et sa limite de taille de clé, mais toute fonction de mixage de force égale ou plus forte pourrait s'y substituer [DES]. Le troisième est une norme plus moderne et plus forte fondée sur SHA-1 [SHA*]. Enfin sont décrits les générateurs modernes de nombres aléatoires largement déployés de UNIX et de Windows.

7.1.1 Recommandations du Ministère de la Défense US pour la génération de mots de passe

Le Ministère de la Défense des États Unis d'Amérique [DoD] a des recommandations spécifiques pour la génération des mots de passe. Il suggère d'utilise la norme de chiffrement des données [DES, *Data Encryption Standard*] américaine en mode de rétroaction de sortie [MODES] comme suit :

Utiliser un vecteur d'initialisation déterminé à partir de l'horloge système, de l'identifiant du système, de l'identifiant de l'usager, et de la date et l'heure ;

Utiliser une clé déterminée à partir des registres d'interruption du système, des registres d'état du système et des compteurs du système ; et,

Comme texte en clair, utiliser une quantité de 64 bits externe générée de façon aléatoire comme les octets ASCII pour les 8 caractères tapés par un administrateur de système.

Le mot de passe peut alors être calculé d'après le "texte chiffré" de 64 bits généré par DES en mode de rétroaction de sortie de 64 bits. Autant de bits qu'il est nécessaire peuvent être tirés de ces 64 bits et étendus en un mot prononçable, une phrase, ou un autre format si un être humain a besoin de se souvenir du mot de passe.

7.1.2 L'appareil /dev/random

Plusieurs versions du système d'exploitation UNIX fournissent un générateur de nombres aléatoires résident dans le noyau. Certains de ces générateurs utilisent des événements capturés par le noyau durant le fonctionnement normal du système.

Par exemple, dans certaines versions de Linux, le générateur consiste en un puits aléatoire de 512 octets représenté comme 128 mots de 4 octets chacun. Lorsqu'un événement survient, comme une interruption de pilote de disque, l'heure de l'événement est combiné par opérateur OUX dans le puits, et le puits est manipulé via un polynôme primitif de degré 128.

Le puits lui-même est traité comme une mémoire tampon en anneau, les nouvelles données étant combinées par opérateur OUX (après manipulation avec le polynôme) à travers le puits tout entier.

Chaque appel qui ajoute de l'entropie au puits estime la quantité de vraie entropie vraisemblable contenue par l'entrée. Le puits lui-même contient un accumulateur qui estime l'entropie totale globale du puits.

Les événements d'entrée viennent de plusieurs sources, comme indiqué ci-dessous. Malheureusement, pour les serveurs machines sans opérateurs humain, les premières et troisièmes ne sont pas disponibles, et de l'entropie peut s'ajouter lentement dans ce cas.

1. Interruptions du clavier. La durée de l'interruption et le code de balayage sont ajoutés au puits. Cela a pour effet d'ajouter de l'entropie provenant de l'opérateur humain en mesurant les temps d'arrivée entre les frappes de touches.

2. Accès au disque et autres interruptions. Un système utilisé par une personne aura vraisemblablement un schéma d'accès au disque difficile à prévoir. (Mais tous les pilotes de disque ne prennent pas en charge la capture de ces informations de temps d'accès avec une précision suffisante pour être utilisable.)

3. Mouvements de la souris. Les temps et les positions de la souris s'ajoutent.

Quand des octets aléatoires sont nécessaires, le puits est haché avec SHA-1 [SHA*] pour donner les octets d'aléa retournés. Si il faut plus d'octets que les vingt du résultat de SHA-1, le résultat haché est remis dans le puits et un nouveau hachage est effectué pour obtenir les vingt octets suivants. Comme des octets sont retirés du puits, l'estimation de l'entropie est diminuée en conséquence.

Pour assurer un puits raisonnablement aléatoire au démarrage du système, les schémas standard de démarrage et de clôture sauvegardent le puits sur un fichier du disque à la fermeture et lisent de fichier au démarrage du système.

Il y a deux interfaces en sortie d'utilisateur. /dev/random retourne des octets du puits mais les bloque quand l'entropie estimée tombe à zéro. Comme l'entropie est ajoutée au puits à partir d'événements, plus de données deviennent disponibles via /dev/random. Les données aléatoires obtenues d'un tel appareil /dev/random conviennent pour la génération de clés à long terme, si suffisamment de bits aléatoires sont dans le puits ou sont ajoutés dans un délai raisonnable.

/dev/urandom fonctionne comme /dev/random ; cependant, il fournit des données même quand l'estimation de l'entropie du puits d'aléa tombe à zéro. Cela peut être adéquat pour des clés de session ou pour d'autres tâches de génération de clés pour lesquelles le bocage en attendant plus de bits aléatoire n'est pas acceptable. Le risque de continuer à prendre des données même quand l'estimation de l'entropie du puits est faible dans les résultats passés peut être calculé à partir des résultats en cours, si un attaquant peut inverser SHA-1. Étant donné que SHA-1 est conçu pour être non réversible, c'est un risque raisonnable.

Pour obtenir des nombres aléatoires sous Linux, Solaris, ou autres systèmes UNIX équipés de code tel que décrit ci-dessus, tout ce qu'une application a à faire est d'ouvrir /dev/random ou /dev/urandom et de lire le nombre d'octets désiré.

(Le dispositif aléatoire Linux a été rédigé par Theodore Ts'o. Il se fondait d'assez loin sur le générateur de nombres aléatoires de PGP 2.X et PGP 3.0 (aka PGP 5.0).)

7.1.3 Fenêtres CryptGenRandom

Les recommandations de Microsoft aux utilisateurs du système d'exploitation largement répandu Windows est généralement d'utiliser l'appel au générateur de nombres pseudo aléatoires CryptGenRandom avec le fournisseur de service cryptographique CryptAPI. Cela comporte un raccourci vers une bibliothèque de fournisseurs de services cryptographiques, un pointeur sur une mémoire tampon par laquelle l'appelant peut fournir l'entropie et dans laquelle est retourné le pseudo aléa, et une indication du nombre d'octets d'aléa qui sont désirés.

Le fournisseur de service cryptographique Windows CryptAPI mémorise une variable d'état de graine avec chaque utilisateur. Lorsque CryptGenRandom est invoqué, cela est combiné avec tout aléa fourni dans l'invocation et avec diverses données systèmes et d'usager telles que l'identifiant de processus, l'identifiant de trame, l'horloge système, l'heure du système, le compteur du système, l'état de la mémoire, les grappes libres du disque, et un hachage du bloc d'environnement d'usager. Ces données sont toutes fournies à SHA-1, et le résultat est utilisé pour donner naissance à un flux de clés RC4. Ce flux de clés est utilisé pour produire les données pseudo aléatoires requises et pour mettre à jour la variable d'état de graine de l'usager.

Les utilisateurs de Windows ".NET" trouveront probablement plus facile d'utiliser l'interface de méthode RNGCryptoServiceProvider.GetBytes. Pour des informations complémentaires, voir [WSC].

7.2 Générateurs supposant une source d'entropie

Les générateurs de nombres pseudo aléatoires décrits dans les trois paragraphes suivants supposent tous qu'une valeur de graine d'une entropie suffisante leur est fournie. Ils peuvent alors générer une séquence forte (voir au paragraphe 6.2) à partir de cette graine.

7.2.1 Génération de nombre pseudo aléatoire X9.82

Le comité X9F1 de l'ANSI est à la phase finale de la création d'une norme de génération de nombres pseudo aléatoires couvrant à la fois les générateurs de vrais aléas et les générateurs de nombres pseudo aléatoires. Elle inclut un certain nombre de générateurs de nombres pseudo aléatoires fondés sur des fonctions de hachage, dont un sera probablement fondé sur des constructions de hachage SHA HMAC [RFC2104]. La version du projet de ce générateur est décrite ci-dessous, en omettant un certain nombre de dispositifs facultatifs [X9.82].

Dans les paragraphes qui suivent, la construction du hachage HMAC est simplement appelée HMAC mais, bien sûr, une fonction SHA d'une norme particulière doit être choisie pour une utilisation particulière. Généralement parlant, si la force des valeurs pseudo aléatoires à générer doit être de N bits, la fonction SHA choisie doit générer N bits de sortie ou plus, et une source d'au moins N bits d'entropie d'entrée sera requise. On doit utiliser la même fonction de hachage tout au long d'une instanciation de ce générateur.

7.2.1.1 Notation

Dans les paragraphes suivants, on utilise la notation donnée ci-après :

hash_length est la taille de sortie de la fonction de hachage sous-jacente utilisée.

input_entropy est la chaîne binaire d'entrée qui fournit l'entropie au générateur.

K est une chaîne binaire de taille hash_length qui fait partie de l'état du générateur et est mise à jour au moins à chaque fois que des bits aléatoires sont générés.

V est une chaîne binaire de taille hash_length qui fait partie de l'état du générateur. Elle est mise à jour chaque fois que sont générés hash_length bits de résultat.

"|" représente la concaténation.

7.2.1.2 Initialisation du générateur

Les octets de V sont tous mis à zéro, sauf le bit de plus faible poids de chaque octet qui est mis à un.
Les octets de K sont tous mis à zéro, puis on règle :
K = HMAC (K, V | 0x00 | input_entropy)
V = HMAC (K, V)
K = HMAC (K, V | 0x01 | input_entropy)
V = HMAC (K, V)

Note : Tous les algorithmes SHA produisent un nombre entier d'octets, de sorte que les longueurs de K et de V seront des nombres entiers de bits.

7.2.1.3 Génération de bits aléatoires

Quand on invoque le résultat, on met simplement : V = HMAC (K, V)
et on utilise les bits d'entrée de V. Si plus de bits sont nécessaires que la longueur de V, mettre "temp" à une chaîne binaire de zéros et effectuer de façon répétée :
V = HMAC (K, V)
temp = temp | V
et s'arrêter dès que temp est égal ou plus long que le nombre de bits d'aléa requis. Utiliser le nombre requis de bits d'entrée de temp. La définition de l'algorithme interdit de demander plus de 2^35 bits.

Après l'extraction et la sauvegarde des bits pseudo aléatoires du résultat comme décrit ci-dessus, on doit aussi effectuer comme suit deux HMAC de plus avant de retourner :
K = HMAC (K, V | 0x00)
V = HMAC (K, V)

7.2.2 Génération de clé X9.17

L'Institut national américain de normalisation a spécifié la méthode suivante pour générer une séquence de clés [X9.17] :
s_0 est la graine initiale de 64 bits.
g_n est la séquence des quantités de clé de 64 bits générées
k est une clé aléatoire réservée pour générer cette séquence de clé.

t est l'heure à laquelle une clé est générée, à la plus fine résolution disponible (jusqu'à 64 bits).
DES (K, Q) est le chiffrement en DES de la quantité Q avec la clé K.

Alors :

g_n = DES (k, DES (k, t) XOR s_n)
s_{n+1} = DES (k, DES (k, t) XOR g_n)

Si g indice n doit être utilisé comme clé DES, tous les huitièmes bits devraient être ajustés à la parité pour cet usage, mais les 64 bits entiers non modifiés g devraient être utilisés pour calculer le prochain s.

7.2.3 Génération de nombre pseudo-aléatoire DSS

L'appendice 3 de la norme de signature numérique (DSS, *Digital Signature Standard*) du NIST [DSS] donne une méthode pour produire une séquence de quantités de 160 bits aléatoires à utiliser comme clés privées ou autres. Cela a été modifié par l'avis d'amendement 1 [DSS_CN1] pour produire l'algorithme suivant afin de générer des nombres pseudo aléatoires d'usage général :

t = 0x 67452301 EFCDAB89 98BADCFE 10325476 C3D2E1F0
$XKEY_0$ = graine initiale

Pour j = 0 à ...

XVAL = ($XKEY_j$ + entrée facultative d'usager) (Mod 2^512)
X_j = G(t, XVAL)
$XKEY_{j+1}$ = (1 + $XKEY_j$ + X_j) (Mod 2^512)

Les quantités X ainsi produites sont les séquences pseudo aléatoires de valeurs de 160 bits. Deux fonctions peuvent être utilisées pour "G" ci-dessus. Chacune produit une valeur de 160 bits et prend deux arguments, une valeur de 160 bits et une valeur de 512 bits.

La première se fonde sur SHA-1 et fonctionne en réglant les cinq variables de liaison, notées H_x dans la spécification SHA-1, au premier argument divisé en cinquièmes. On suit alors les étapes (a) à (e) de la section 7 de la spécification NIST SHA-1 sur le second argument comme si c'était un bloc de données de 512 bits. Les valeurs de la variable de liaison après ces étapes sont alors concaténées pour produire le résultat de G [SHA*].

Comme méthode de remplacement, NIST définit aussi une autre fonction G fondée sur plusieurs applications de la fonction de chiffrement DES [DSS].

8 Exemples d'exigence d'aléa

Nous donnons ci-dessous deux exemples qui montrent des calculs approximatifs d'aléas nécessaires pour la sécurité. Le premier est pour des mots de passe à sécurité modérée, alors que le second suppose un besoin de clé cryptographique de très haute sécurité.

De plus, [ORMAN] et [RSA_BULL13] donnent des informations sur les longueurs de clés publiques qui devraient être utilisées pour échanger des clés symétriques.

8.1 Génération de mot de passe

Supposons que les mots de passe d'utilisateur changent une fois par an et qu'on souhaite que la probabilité qu'un adversaire puisse deviner le mot de passe pour un compte particulier soit inférieure à un millième. Supposons de plus que l'envoi d'un mot de passe au système soit le seul moyen d'essayer un mot de passe. La question cruciale est alors combien de fois l'adversaire peut-il essayer les différentes possibilités. Supposons que des délais aient été introduits dans un système de sorte qu'un adversaire ne puisse faire un essai de mot de passe qu'au plus toutes les six secondes. Cela fait 600 par heure, ou environ 15 000 par jour, ou environ 5 000 000 d'essais par an. En supposant une sorte de surveillance, il est peu vraisemblable que quelqu'un puisse réellement essayer continuellement pendant un an. Même si les fichiers de journalisation ne sont vérifiés qu'une fois par mois, 500 000 essais sont plausibles avant qu'on remarque l'attaque et que des mesures soient prises pour changer les mots de passe et rendre plus difficiles les essais des mots de passe.

Pour avoir une chance sur mille de deviner le mot de passe sur 500 000 essais implique un univers d'au moins 500 000 000 mots de passe, ou environ 2^29. Et donc 29 bits d'aléa sont nécessaires. Ceci peut probablement être réalise en utilisant les

entrées recommandées par le Ministère de la défense US pour la génération des mots de passe, car il y a 8 entrées qui sont en moyenne probablement au dessus de 5 bits d'aléa chacune (voir le paragraphe 7.1). En utilisant une liste de 1 000 mots, le mot de passe pourrait être exprimé par une phrase de trois mots (1 000 000 000 de possibilités). En utilisant des lettres insensibles à la casse et des chiffres, six caractères devraient suffire ((26+10)^6 = 2,176,782,336 de possibilités).

Pour un mot de passe de plus forte sécurité, le nombre de bits requis augmente. Diminuer la probabilité de 1 000 exige d'accroître l'univers des mots de passe du même facteur, ce qui ajoute environ 10 bits. Et donc, pour avoir seulement une chance sur un million qu'un mot de passe soit deviné dans le scénario ci-dessus exigerait 39 bits d'aléa et un mot de passe qui serait une phrase de quatre mots sur une liste de1 000 mots, ou huit lettres/chiffres. Pour passer à une chance sur 10^9, 49 bits d'aléa sont nécessaires, ce qui implique une phrase de cinq mots ou un mot de passe de dix lettres/chiffres.

Dans un système réel, bien sûr, il y a d'autres facteurs. Par exemple, plus les mots de passe sont longs et difficiles à mémoriser, plus les utilisateurs seront enclins à les écrire, d'où il résulte un risque supplémentaire de compromission.

8.2 Une clé de chiffrement à très haute sécurité

Supposons qu'une clé de très haute sécurité soit nécessaire pour le chiffrement/déchiffrement symétrique entre deux parties. Supposons aussi qu'un adversaire peut observer les communications et connaître l'algorithme qui est utilisé. Dans le champ des possibilités aléatoires, l'adversaire peut essayer les valeurs des clés dans l'espoir de trouver la bonne. Supposons de plus qu'une attaque en force de l'essai des clés est le mieux que puisse faire l'adversaire.

8.2.1 Efforts pour l'essai des clés

Quels efforts devra-t-il faire pour essayer chaque clé ? Pour les applications de très haute sécurité, il vaut mieux supposer une faible valeur d'effort. Même si il est clair que cela prendrait des dizaines de milliers de cycles d'ordinateur ou plus pour essayer une seule clé, il peut y avoir des schémas qui permettent d'essayer d'énormes blocs de valeurs de clés avec beaucoup moins d'efforts par clé. Et donc, il est probablement meilleur de supposer au plus quelques centaines de cycles par clé. (Il n'y a pas de limite inférieure claire à cela, car les ordinateurs fonctionnent en parallèle sur un certain nombre de bits et un mauvais algorithme de chiffrement pourrait permettre d'essayer de nombreuses clés ou même de groupes de clés en parallèle. Cependant, nous devons supposer une certaine valeur et pouvons espérer qu'un algorithme raisonnablement fort a été choisi pour notre hypothèse de tâche de haute sécurité.)

Si l'adversaire peut commander des processeurs parallèles ou un grand réseau de stations de travail, 10^11 cycles par seconde est probablement une hypothèse minimale de nos jours. En se projetant quelques années dans le futur, il devrait y avoir au minimum une amélioration d'un ordre de grandeur. Et donc, il est raisonnable de supposer que 10^10 clés pourraient être testées par seconde, ou 3,6*10^12 par heure ou 6*10^14 par semaine, ou 2.4*10^15 par mois. Cela implique le besoin d'un minimum de 63 bits d'aléa dans les clés, pour être sûr qu'elles ne peuvent pas être trouvées en un mois. Même alors, il est possible que, dans quelques années, un adversaire très déterminé et plein de ressources puisse casser la clé en quinze jours ; en moyenne, il n'a besoin d'essayer que la moitié des clés.

Ces questions sont exposées en détail dans "Longueurs minimales de clés pour chiffrements symétriques pour fournir une sécurité commerciale adéquate : Rapport du groupe ad hoc des cryptographes et informaticiens" [KeyStudy] qui était financé par la Business Software Alliance. Il conclut qu'une longueur de clé raisonnable en 1995 pour la très haute sécurité est dans la gamme de 75 à 90 bits et, comme le coût de la cryptographie ne varie pas beaucoup avec la taille de clé, il recommande 90 bits. Pour mettre à jour ces recommandations, il suffit d'ajouter 2/3 de bit par an pour la Loi de Moore [MOORE]. Ceci nous amène à une détermination, pour l'année 2004, d'une longueur de clé raisonnable dans la gamme de 81 à 96 bits. En fait, aujourd'hui, il est de plus en plus courant d'utiliser des clés de plus de 96 bits, comme 128 bits (ou plus longues) avec AES et des clés avec des longueurs effectives de 112 bits avec triple-DES.

8.2.2 Attaques par rencontre au milieu

Si du texte choisi ou du texte clair connu et le texte chiffré résultant sont disponibles, une attaque "par rencontre au milieu" *(meet-in-the-middle)* est possible si la structure de l'algorithme de chiffrement la permet. (Dans une attaque de texte clair connu, l'adversaire connaît tout ou partie (éventuellement un en-tête standard ou des champs de queue) des messages chiffrés. Dans une attaque de texte en clair choisi, l'adversaire peut forcer le chiffrement d'un texte en clair de son choix, qui peut faire suite à un texte qui va exciter l'intérêt, et qui est envoyé par l'adversaire sur un canal crypté parce que le texte est si intéressant.

Ce qui suit est une explication très simplifiée de l'attaque par rencontre au milieu : l'adversaire peut chiffrer à moitié le texte en clair connu ou choisi avec toutes les clés possibles de la première moitié, trier le résultat, et ensuite à moitié déchiffrer le texte codé avec les clés de la seconde moitié. Si il trouve une correspondance, la clé complète peut être

assemblée à partir des moitiés et utilisée pour déchiffrer les autres parties du message ou les autres messages. Au mieux, ce type d'attaque peut diminuer de moitié l'exposant du travail requis par l'adversaire tout en ajoutant un très important, mais à peu près constant, facteur d'effort. Et donc, si cette attaque peut être montée, un doublement de la quantité d'aléa dans la très forte clé d'un minimum de 192 bits (96*2) est nécessaire pour l'année 2004, sur la base de l'analyse [KeyStudy].

Cette quantité d'aléa est bien au-delà de la limite recommandée pour les entrées par le Ministère de la défense US pour la génération de mots de passe et pourrait exiger la mesure des temps de frappe de l'usager, la génération de nombres aléatoires par le matériel ou d'autres sources d'aléa.

L'attaque par rencontre au milieu suppose que l'algorithme cryptographique peut être décomposé de cette façon. Heureusement, aucun algorithme moderne n'a cette faiblesse, mais il peut y avoir des cas où on n'en est pas sûr, ou même, où on ne sait pas avec quel algorithme une clé sera utilisée. Même si un algorithme de base n'est pas soumis à une attaque par rencontre au milieu, une tentative pour produire un algorithme plus fort en appliquant deux fois l'algorithme de base (ou deux algorithmes différents en séquence) avec des clés différentes gagnera moins de sécurité supplémentaire qu'espéré. Un tel algorithme composite serait l'objet d'attaques par rencontre au milieu.
D'énormes ressources peuvent être nécessaires pour monter une attaque par rencontre au milieu, mais elles sont probablement à la portée des services de sécurité des grandes nations. Par nature, toutes les nations espionnent le trafic des autres nations.

8.2.3 Autres considérations

[KeyStudy] considère aussi les possibilités d'un matériel dédié au cassage de code et à l'intérêt d'avoir une marge de sécurité adéquate.

Noter que les calculs de longueur de clé comme ceux qui figurent ci-dessus sont controversés et dépendent de diverses hypothèses sur les algorithmes cryptographiques utilisés. Dans certains cas, un professionnel ayant une bonne connaissance des techniques de cassage d'algorithme et de la force de l'algorithme utilisé pourrait se satisfaire de moins de la moitié des 192 bits de taille de clé calculée ci-dessus.

Pour d'autres exemples de principes de conception prudente, voir [FERGUSON].

9 Conclusion

La génération de quantités secrètes "aléatoires" imprévisibles pour les utilisations de sécurité est une tâche essentielle mais difficile.

Les techniques de matériel pour produire l'entropie nécessaire seraient relativement simples. En particulier, le volume et la qualité n'ont pas besoin d'être élevés, et les matériels d'ordinateur existants, comme une entrée audio ou un pilote de disque, peuvent être utilisés.

Des techniques de calcul largement disponibles peuvent traiter des quantités aléatoires de faible qualité provenant de multiples sources, ou une plus grande quantité de telles entrées de faible qualité provenant d'une source, pour produire une plus petite quantité de matériel de clé de meilleure qualité. En l'absence de sources matérielles d'aléa, diverses sources d'usager et de logiciels peuvent fréquemment être utilisées à la place, avec des précautions. Cependant, la plupart des systèmes modernes ont déjà des matériels, comme des pilotes de disques ou des entrées audio, qui peuvent être utilisés pour produire de l'aléa de grande qualité.

Une fois qu'est disponible un matériel de graine de clé de haute qualité en quantité suffisante (quelques centaines de bits), les techniques de calcul sont disponibles pour produire des séquences cryptographiquement fortes de séquences de quantités imprévisibles par le calcul à partir de ce matériel de graine.

10 Considérations sur la sécurité

La totalité du présent document concerne les techniques et recommandations pour générer des quantités "aléatoires" imprévisibles à utiliser comme mots de passe, clés cryptographiques, vecteurs d'initialisation, numéros de séquence, et applications de sécurité similaires.

11 Remerciements

Des remerciements particuliers à Paul Hoffman et John Kelsey pour leurs commentaires détaillés et à Peter Gutmann, qui a permis l'incorporation d'éléments de son article "Software Generation of Practically Strong Random Numbers"

(génération logicielle de nombre aléatoires pratiquement forts).
Les personnes suivantes (par ordre alphabétique) ont contribué de façon substantielle au présent document :
Steve Bellovin, Daniel Brown, Don Davis, Peter Gutmann, Tony Hansen, Sandy Harris, Paul Hoffman, Scott Hollenback, Russ Housley, Christian Huitema, John Kelsey, Mats Naslund, et Damir Rajnovic.

Les personnes suivantes (par ordre alphabétique) ont contribué à la RFC 1750, le prédécesseur de ce document:
David M. Balenson, Don T. Davis, Carl Ellison, Marc Horowitz, Christian Huitema, Charlie Kaufman, Steve Kent, Hal Murray, Neil Haller, Richard Pitkin, Tim Redmond, et Doug Tygar.

Appendice A Changements depuis la RFC 1750

1. Des remerciements supplémentaires ont été ajoutés.
2. Insertion du paragraphe 5.3 sur le mixage avec des S-boxes.
3. Ajout du paragraphe 3.3 sur les sources d'aléa par oscillateur en anneau.
4. Ajout de AES et des membres de la série SHA produisant plus de 160 bits. L'utilisation de AES a été développée et celle de DES diminuée.
5. Ajout du paragraphe 6.3 sur les techniques de puits d'entropie.
6. Ajout du paragraphe 7.2.3 sur les techniques de génération de nombres pseudo aléatoires données dans FIPS 186-2 (avec l'avis de modification 1), du paragraphe 7.2.1 sur celles données dans X9.82, section 7.1.2 sur les techniques de génération de nombres aléatoires du dispositif /dev/random dans Linux et autres systèmes UNIX, et du paragraphe 7.1.3 sur les techniques de génération de nombres aléatoires dans le système d'exploitation Windows.
7. Ajout des références à l'étude "Minimal Key Lengths for Symmetric Ciphers to Provide Adequate Commercial Security" publiée en janvier 1996 [KeyStudy] et à la [RFC1948].
8. Ajout d'avertissements sur l'utilisation de Diffie-Hellman comme fonction de mixage et, à cause de ces avertissements et des conséquences sur l'intensité des calculs, d'une remmandatio contre son utilisation.
9. Ajout de références à l'effort X9.82 et aus articles [TURBID] et [NASLUND].
10. Ajout de l'exposé sur min-entropy et l'entropie de Renyi et des références au livre de [LUBY].
11. Restructuration majeure, changements de formulation mineurs, et diverses mises à jour de références.

Références informatives

[AES] "Specification of the Advanced Encryption Standard (AES)", United States of America, US National Institute of Standards and Technology, FIPS 197, novembre 2001.

[ASYMMETRIC] G. Simmons, Ed., "Secure Communications and Asymmetric Cryptosystems", AAAS Selected Symposium 69, ISBN 0-86531-338-5, Westview Press, 1982.

[BBS] L. Blum, M. Blum et M. Shub, "A Simple Unpredictable Pseudo-Random Number Generator", SIAM Journal on Computing, v. 15, n. 2, 1986.

[BRILLINGER] D. Brillinger, "Time Series: Data Analysis and Theory", Holden-Day, 1981.

[CRC] "C.R.C. Standard Mathematical Tables", Chemical Rubber Publishing Company.

[DAVIS] Davis, D., Ihaka, R., and P. Fenstermacher, "Cryptographic Randomness from Air Turbulence in Disk Drives", Advances in Cryptology - Crypto '94, Springer-Verlag Lecture Notes in Computer Science #839, 1984.

[DES "Data Encryption Standard", US National Institute of Standards and Technology, FIPS 46-3, October 1999. Aussi, "Data Encryption Algorithm", American National Standards Institute, ANSI X3.92-1981. Voir aussi FIPS 112, "Password Usage", qui comporte le code FORTRAN pour effectuer DES.

[D-H] E. Rescorla, "Méthode d'accord de clés Diffie-Hellman", RFC 2631, juin 1999.

[DNSSEC1] R. Arends, R. Austein, M. Larson, D. Massey et S. Rose, "Introduction et exigences de la sécurité DNS", RFC 4033, mars 2005.

[DNSSEC2] R. Arends, R. Austein, M. Larson, D. Massey et S. Rose, "Enregistrements de ressources pour les extensions de sécurité DNS", RFC 4034, mars 2005.

[DNSSEC3] R. Arends, R. Austein, M. Larson, D. Massey et S. Rose, "Modifications du protocole pour les extensions de

sécurité DNS", RFC 4035, mars 2005.

[DoD] "Password Management Guideline", United States of America, Department of Defense, Computer Security Center, CSC-STD-002-85, April 1885. (Voir aussi "Password Usage", FIPS 112, qui comporte CSC-STD-002-85 comme un des appendices. FIPS 112 est actuellement disponible à : http://www.idl.nist.gov/fipspubs/fip112.htm.)

[DSS] "Digital Signature Standard (DSS)", US National Institute of Standards and Technology, FIPS 186-2, janvier 2000.

[DSS_CN1] "Digital Signature Standard Change Notice 1", US National Institute of Standards and Technology, FIPS 186-2 Change Notice 1, 5, octobre 2001.

[FERGUSON] N. Ferguson et B. Schneier, "Practical Cryptography", Wiley Publishing Inc., ISBN 047122894X, avril 2003.

[GIFFORD] Gifford, D., "Natural Random Number", MIT/LCS/TM-371, septembre 1988.

[IEEE_802.11i]"Amendment to Standard for Telecommunications and Information Exchange Between Systems - LAN/MAN Specific Requirements - Part 11: Wireless Medium Access Control (MAC) and physical layer (PHY) specifications: Medium Access Control (MAC) Security Enhancements", IEEE, janvier 2004.

[IPSEC] S. Kent, et R. Atkinson, "Architecture de sécurité pour le protocole Internet", RFC 2401, novembre 1998.

[Jakobsson] M. Jakobsson, E. Shriver, B. Hillyer, et A. Juels, "A practical secure random bit generator", Débats de la cinquième conférence ACM sur la sécurité informatique et des communications, 1998.

[KAUFMAN] C. Kaufman, R. Perlman, et M. Speciner, "Network Security: Private Communication in a Public World", Prentis Hall PTR, ISBN 0-13-046019-2, 2ème édition 2002.

[KeyStudy] M. Blaze, W. Diffie, R. Riverst, B. Schneier, T. Shimomura, E. Thompson, et M. Weiner, "Minimal Key Lengths for Symmetric Ciphers to Provide Adequate Commercial Security: A Report by an Ad Hoc Group of Cryptographers and Computer Scientists", janvier 1996. Actuellement disponible à : http://www.crypto.com/papers/keylength.txt and http://www.securitydocs.com/library/441.

[KNUTH] D. Knuth, "The Art of Computer Programming", Volume 2 : Seminumerical Algorithms, Chapter 3 : Random Numbers, Addison-Wesley Publishing Company, 3ème édition, novembre 1997.

[KRAWCZYK] Krawczyk, H., "How to Predict Congruential Generators", Journal of Algorithms, V. 13, N. 4, décembre 1992.

[LUBY M. Luby, "Pseudorandomness and Cryptographic Applications", Princeton University Press, ISBN 0691025460, 8 janvier 1996.

[MAIL_PEM1] J. Linn, "Amélioration de la confidentialité pour la messagerie électronique de l'Internet. Partie I : Chiffrement de message et procédures d'authentification", RFC 1421, février 1993.

[MAIL_PEM2] S. Kent, "Amélioration de la confidentialité pour la messagerie électronique de l'Internet. Partie II : Gestion de clés fondée sur le certificat", RFC 1422, février 1993.

[MAIL_PEM3] D. Balenson, "Amélioration de la confidentialité pour la messagerie électronique de l'Internet. Partie III : Algorithmes, modes, et identifiants", RFC 1423, février 1993.

[MAIL_PEM4] B. Kaliski, "Amélioration de la confidentialité pour la messagerie électronique de l'Internet. Partie IV : Certification de clé et services qui s'y rapportent", RFC 1424, février 1993.

[MAIL_PGP1] J. Callas, L. Donnerhacke, H. Finney et R. Thayer, "Format de message OpenPGP", RFC 2440, novembre 1998.

[MAIL_PGP2] M. Elkins, D. Del Torto, R. Levien, R. et T. Roessler, "Sécurité MIME avec OpenPGP", RFC 3156, août 2001.

[S/MIME] RFC 2632 à 2634: Ramsdell, B., "S/MIME Version 3 Certificate Handling", RFC 2632, juin 1999. B. Ramsdell, "S/MIME Version 3 Message Specification", RFC 2633, juin 1999.

P. Hoffman, "Enhanced Security Services for S/MIME", RFC 2634, juin 1999.

[MD4] R. Rivest, "The MD4 Message-Digest Algorithm", RFC 1320, avril 1992.

[MD5] R. Rivest, "The MD5 Message-Digest Algorithm ", RFC 1321, avril 1992.

[MODES] "DES Modes of Operation", US National Institute of Standards and Technology, FIPS 81, décembre 1980. Aussi : "Data Encryption Algorithm - Modes of Operation", American National Standards Institute, ANSI X3.106-1983.

[MOORE] Loi de Moore : la croissance exponentielle de la densité logique des circuits de silicium. Formulée à l'origine par Gordon Moore en 1964 comme un doublement chaque année à partir de 1962, dans la fin des années 1970 le taux est tombé au doublement tous les 18 mois et en est resté là jusqu'à la date du présent document. Voir "The New Hacker's Dictionary", Third Edition, MIT Press, ISBN 0-262-18178-9, Eric S. Raymond, 1996.

[NASLUND] M. Naslund et A. Russell, "Extraction of Optimally Unbiased Bits from a Biased Source", IEEE Transactions on Information Theory. 46(3), mai 2000.

[ORMAN] H. Orman et P. Hoffman, "Détermination de la force des clés publiques utilisées pour échanger des clés symétriques", BCP 86, RFC 3766, avril 2004.

[RFC1750] D. Eastlake 3, S. Crocker et J. Schiller, "Recommandations d'aléa pour la sécurité", RFC 1750, décembre 1994.

[RFC1948] S. Bellovin, "Se défendre contre les attaques de numéro de séquence", RFC 1948, mai 1996.

[RFC2104] H. Krawczyk, M. Bellare et R. Canetti, "HMAC :Hachage de clé pour l'authentification de message", RFC 2104, février 1997.

[RSA_BULL1] "Suggestions for Random Number Generation in Software", RSA Laboratories Bulletin #1, janvier 1996.

[RSA_BULL13] R. Silverman, "A Cost-Based Security Analysis of Symmetric and Asymmetric Key Lengths", RSA Laboratories Bulletin #13, avril 2000 (révisé en novembre 2001).

[SBOX1] S. Mister et C. Adams, "Practical S-box Design", Selected Areas in Cryptography, 1996.

[SBOX2] K. Nyberg, "Perfect Non-linear S-boxes", Advances in Cryptography, Eurocrypt '91 Proceedings, Springer-Verland, 1991.

[SCHNEIER] B. Schneier, "Applied Cryptography: Protocols, Algorithms, and Source Code in C", 2ème édition, John Wiley & Sons, 1996.

[SHANNON] C. Shannon, "The Mathematical Theory of Communication", University of Illinois Press, 1963. Publication originelle de Bell System Technical Journal, July and October, 1948.

[SHIFT1] S. Golub, "Shift Register Sequences", Aegean Park Press, édition révisée, 1982.

[SHIFT2] W. Barker, "Cryptanalysis of Shift-Register Generated Stream Cypher Systems", Aegean Park Press, 1984.

[SHA] "Secure Hash Standard", US National Institute of Science and Technology, FIPS 180-2, 1 août 2002.

[SHA_RFC] D. Eastlake 3 et P. Jones, "Algorithme 1 de hachage sécurisé (SHA1)", RFC 3174, septembre 2001.

[SSH] Produits du groupe de travail SECSH, travaux en cours, 2005.

[STERN] J. Stern, "Secret Linear Congruential Generators are not Cryptographically Secure", Proc. IEEE STOC, 1987.

[TLS] T. Dierks et C. Allen, "Le protocole TLS, version 1.0", RFC 2246, janvier 1999.

[TURBID] J. Denker, "High Entropy Symbol Generator", <http://www.av8n.com/turbid/paper/turbid.htm>, 2003.

[USENET_1] B. Kantor et P. Lapsley, "Protocole de transfert des nouvelles du réseau", RFC 977, février 1986.

[USENET_2] S. Barber, "Extensions NNTP communes", RFC 2980, octobre 2000.

[VON_NEUMANN] J. Von Nuemann, "Various techniques used in connection with random digits", Von Neumann's Collected Works, Vol. 5, Pergamon Press, 1963.

[WSC] M. Howard et D. LeBlanc, "Writing Secure Code, Second Edition", Microsoft Press, ISBN 0735617228,décembre 2002.

[X9.17] "American National Standard for Financial Institution Key Management (Wholesale)", American Bankers Association, 1985.

[X9.82] "Random Number Generation", American National Standards Institute, ANSI X9F1, Work in Progress. Part 1 - Overview and General Principles. Part 2 - Non-Deterministic Random Bit Generators Part 3 - Deterministic Random Bit Generators

Adresse des auteurs

Donald E. Eastlake 3rd	Jeffrey I. Schiller	Steve Crocker
Motorola Laboratories	MIT, Room E40-311	
155 Beaver Street	77 Massachusetts Avenue	
Milford, MA 01757 USA	Cambridge, MA 02139-4307 USA	
Tél : +1 508-786-7554 (w)	Tél : +1 617-253-0161	
+1 508-634-2066 (h)		
mél : Donald.Eastlake@motorola.com	mél : jis@mit.edu	mél : steve@stevecrocker.com

Déclaration de copyright

Propriété intellectuelle

L'IETF ne prend pas position sur la validité et la portée de tout droit de propriété intellectuelle ou autres droits qui pourrait être revendiqués au titre de la mise en œuvre ou l'utilisation de la technologie décrite dans le présent document ou sur la mesure dans laquelle toute licence sur de tels droits pourrait être ou n'être pas disponible o pas plus qu'elle ne prétend avoir accompli aucun effort pour identifier de tels droits. Les informations sur les procédures de l'ISOC au sujet des droits dans les documents de l'ISOC figurent dans les BCP 78 et BCP 79.

Des copies des dépôts d'IPR faites au secrétariat de l'IETF et toutes assurances de disponibilité de licences, ou le résultat de tentatives faites pour obtenir une licence ou permission générale d'utilisation de tels droits de propriété par ceux qui mettent en œuvre ou utilisent la présente spécification peuvent être obtenues sur répertoire en ligne des IPR de l'IETF à http://www.ietf.org/ipr.

L'IETF invite toute partie intéressée à porter son attention sur tous copyrights, licences ou applications de licence, ou autres droits de propriété qui pourraient couvrir les technologies qui peuvent être nécessaires pour mettre en œuvre la présente norme. Prière d'adresser les informations à l'IETF à ietf- ipr@ietf.org.

Remerciement

Le financement de la fonction d'édition des RFC est actuellement fourni par l'activité de soutien administratif de l'IETF (IASA, *Administrative Support Activity*).