# Cryptographie et Linux

Version du 12 juin 2003

**Frédéric Schütz**

schutz@mathgen.ch

**Informations supplémentaires**

La dernière version de ce texte est disponible sur le web à l'adresse

http://www.mathgen.ch/cours/

En addition à la licence ci-dessus, l'auteur original vous serait reconnaissant de lui envoyer les modifications que vous apporterez à ce texte ou toute autre suggestion, afin qu'il puisse les prendre en compte dans une prochaine version et améliorer ce document.

Si ce document est utilisé dans un but lucratif, il n'est pas interdit d'en faire profiter l'auteur original ;-) (ou une organisation à but non-lucratif choisie avec lui)

Toute question, modification, commentaire ou autre peut-être envoyé à l'adresse électronique schutz@mathgen.ch, ou par courrier :

Frédéric Schütz
Ch. de Mont-Rose 33
1294 Genthod
Suisse

# Table des matières

**1 Introduction à la cryptographie** **3**
1.1 Vocabulaire et historique rapide . . . 3
1.2 Les algorithmes de base de la cryptographie . . . 4
1.2.1 Cryptographie symétrique . . . 4
1.2.2 Cryptographie assymétrique (ou “à clé publique”) . . . 4
1.2.3 Fonction de hachage . . . 5
1.2.4 Signature digitale . . . 6
1.2.5 Générateurs de nombres aléatoires . . . 6
1.2.6 Attaques diverses et tailles de clés . . . 7
1.3 Certificats : les passeports digitaux . . . 8
1.4 Les protocoles SSL et TLS . . . 10

**2 Cryptographie et Linux : quelques applications** **11**
2.1 L’intérêt du modèle Open Source en sécurité et cryptographie . . . 11
2.2 PGP et GPG . . . 12
2.3 SSH (Secure SHell) . . . 12
2.4 Réseaux Privés Virtuels . . . 13
2.5 Public Key Infrastructure (PKI) . . . 13
2.6 Chiffrement de disques . . . 13
2.6.1 Loopback . . . 14
2.6.2 Quelques autres systèmes . . . 15

**3 References** **15**

# 1 Introduction à la cryptographie

## 1.1 Vocabulaire et historique rapide

La **cryptographie** est la *discipline qui englobe tous les principes, moyens et méthodes destinés à la transformation de données afin de cacher leur contenu, d’empêcher leur modification ou leur utilisation frauduleuse.* Quand on veut *transformer un message en clair en un message inintelligible*, on dit qu’on le **chiffre**. L’inverse, *transformer un message inintelligible en un message en clair, en connaissant la méthode*, s’appelle **déchiffrer** (c’est le point de vue du receveur légitime du message). Si par contre on essaie de *déchiffrer un message sans connaître la méthode*, on dit qu’on **décrypte** (c’est ce que fait un adversaire qui intercepte le message et essaie de le lire). L’*ensemble des procédés permettant de décrypter* s’appelle la **cryptanalyse**.

Il ne faut pas confondre la cryptographie avec la **stéganographie**, qui est *l’ensemble des techniques qui consistent à cacher l’existence même d’un message* (par exemple en cachant un message dans les bits de poids faible d’une image).

La cryptographie existe à peu près depuis l’invention de l’écriture : on en trouve des exemples dans l’Egypte ancienne, et l’un des codes secrets les plus connus est le *chiffre de César*, que Jules utilisait pour communiquer avec ses généraux (il s’agissait simplement de décaler l’alphabet de 3 lettres : ainsi, le message `attaquez demain` devenait `dwwdtxhc ghpdlq`). Malgré cette histoire ancienne, la cryptographie est restée pratiquement inconnue en dehors des milieux militaires et

diplomatiques jusqu'au début des années 1970. A cette période, le développement conjugué des ordinateurs et des réseaux contribua à son apparition dans le domaine civil : le développement des réseaux créa une demande pour des méthodes de transmission sûres, tandis qu'en parallèle, l'augmentation de la puissance de calcul des ordinateurs mettait la cryptographie à la portée des ordinateurs du marché.

Dans un article célèbre paru en 1976[1], W. Diffie et M. Hellman introduisent pour la première fois publiquement le concept de cryptographie à clé publique et donnent un véritable coup de fouet à la discipline, qui commence à faire son apparition dans les milieux académiques. C'est en 1978 que le concept devient réalité, avec l'article de R. Rivest, A. Shamir et L. Adleman[2] qui décrivent le premier algorithme à clé publique, RSA, qui deviendra par la suite le standard de-facto en la matière (protégé par un brevet aux Etats-Unis, l'algorithme est enfin tombé dans le domaine public début septembre 2000).

Le gouvernement américain choisit en 1977 un standard de cryptographie symétrique, DES (Data Encryption Standard). Le choix se fait dans l'ombre, et l'algorithme porte l'empreinte de la NSA (National Security Agency, l'agence américaine chargée de la sécurité et de l'écoute des communications) qui a changé les spécifications par rapport à la première version proposée par IBM.

En 1991, Phil Zimmerman diffuse la première version du programme PGP (Pretty Good Privacy), la "cryptographie pour les masses" selon lui. Il n'a pas tort : PGP fera office de véritable vulgarisateur scientifique pour le domaine, en permettant à tout le monde d'accéder à une sécurité de qualité, sans forcément avoir à comprendre les détails mathématiques des algorithmes sous-jacents.

L'EFF[3] (Electronic Frontier Foundation) donne le coup de grâce à DES en 1998, en construisant une machine permettant d'essayer toutes les clés possibles, soit $2^{56}$ (environ $10^{17}$), et donc de décoder n'importe quel message. Tout le monde savait que c'était possible (des propositions théoriques de telles machines avaient été faites avant 1980) et que divers services secrets le faisaient de façon routinière, mais l'EFF est le premier organisme à en faire publiquement la démonstration.

Quelques temps auparavant, le NIST (National Institute of Standards and Technology) américain avait lancé un concours international pour décider d'un nouveau standard de cryptographie symétrique, l'AES[4] (Advanced Encryption Standard). Ce concours, modèle de transparence, s'est achevé le 2 octobre 2000, avec l'annonce du choix de l'algorithme Rijndael (prononcer [ReignDal]), d'origine belge, comme proposition pour un nouveau standard, qui devrait être décidé officellement avant l'été 2001.

## 1.2 Les algorithmes de base de la cryptographie

### 1.2.1 Cryptographie symétrique

La cryptographie à clé symétrique est l'idée intuitive que l'on a de la cryptographie, celle que les enfants utilisent quand ils créent des codes pour s'échanger des messages secrets. On suppose que deux personnes, Alice et Bob pour reprendre les noms classiques, désirent s'échanger une information qui doit rester confidentielle, en utilisant un système de cryptographie public (qui n'a pas besoin d'être secret). Ils doivent d'abord se mettre d'accord sur une clé, qui est un paramètre secret du système, connu d'eux seuls, et leur permettra de chiffrer et déchiffrer leurs messages, tout en empêchant quelqu'un d'autre d'autre de décoder les messages qu'ils se sont envoyés. Pour cela, ils peuvent par exemple se recontrer en personne pour définir la clé qu'ils vont utiliser.

Quand Alice veut envoyer un message à Bob, elle utilise la clé et le système pour créer un

[1] Whitfield Diffie et Martin E. Hellman, *New directions in cryptography.* IEEE transactions on Information theory, vol IT-22, 6 (Nov. 1976), 644–654.

[2] Ronald L. Rivest, Adi Shamir, Leonard Adleman, *A method for Obtaining Digital Signatures and Public-Key Cryptosystems.* Communications of the ACM 21,2 (Fév. 1978), 120–126.

[3] `http://www.eff.org`

[4] `http://www.nist.gov/aes/`

texte chiffré, qu'elle envoie ensuite, par poste, email ou n'importe quel moyen. Bob, connaissant la clé, peut utiliser le même système pour déchiffrer le message. Un attaquant qui intercepterait le message ne pourrait pas le lire, même s'il connaît le système, car il ne possède pas la clé.

Ces systèmes, qui utilisent la même clé pour chiffrer et pour déchiffrer un message, sont appelés systèmes de cryptographie symétrique. Il en existe énormément et on peut citer parmi les plus connus DES (l'ancien standard du gouvernement américain), Rijndael (le nouveau standard), IDEA, Twofish, etc.

Le principal avantage de ce type d'algorithme est qu'ils sont à la fois sûrs et très rapides. Par contre, ils ont deux désavantages évidents. Tout d'abord, il faut disposer d'un canal sûr pour échanger la clé (par exemple, rencontrer l'autre en personne) : si vous envoyez la clé par la poste et que votre courrier est ouvert sans que vous le sachiez, vous pourrez utiliser autant de mesures de sécurité que vous le voulez, votre adversaire aura la clé et pourra impunément déchiffrer vos messages.

D'un autre côté, chaque clé ne peut être partagée que par 2 personnes, et si vous avez plusieurs correspondants, vous êtes obligé d'avoir une clé par correspondant, et toutes ces clés doivent absolument être tenues secrètes !

### 1.2.2 Cryptographie assymétrique (ou "à clé publique")

L'idée de la cryptographie à clé publique est la suivante : est-il vraiment obligatoire que la clé qui permet de chiffrer le message soit la même que celle qui permet de le déchiffrer ? On a découvert dans les années 70 que la réponse est non et qu'il existe des algorithmes où les deux clés sont différentes, mais liées, et que si vous connaissez l'une d'elles, il n'est pas possible de connaître l'autre en un temps de calcul "acceptable". En conclusion, vous pouvez sans autre distribuer publiquement la clé de chiffrement : même en possession de cette information, un attaquant ne pourra pas déchiffrer le message qu'une autre personne aura envoyée, car la clé ne sert qu'à créer des secrets, pas à les révéler.

Cette idée est moins intuitive que la cryptographie symétrique. On peut la comparer à des cadenas ouverts que vous distriburiez à vos correspondants : n'importe qui est capable de fermer le cadenas sans la clé, mais même celui qui l'a fermé n'est plus capable de l'ouvrir — seul le possesseur de la clé peut le faire.

Les avantages de cette méthode sont évidents : il n'y a plus besoin de se rencontrer pour échanger secrétement une clé de chiffrement, puisque celle-ci n'a pas besoin de rester secrète. De plus, il suffit d'une seule clé pour toutes les personnes qui veulent m'écrire : il faut simplement la mettre à leur disposition (sur un serveur web par exemple) pour que n'importe qui puisse ensuite chiffrer des messages, mais seul le possesseur de la clé secrète correspondante pourra les déchiffrer.

Ces systèmes souffrent malheureusement de plusieurs désavantages. Tout d'abord, les mathématiques sur lesquelles sont basés ces algorithmes sont beaucoup plus coûteuses en temps de calcul (même si elles ne sont pas compliquées à comprendre), et rendent ces algorithmes très lents. On peut souvent résoudre ce problème en utilisant deux algorithmes, l'un symétrique et l'autre assymétrique : on utilise une clé aléatoire pour chiffrer le message à l'aide de l'algorithme symétrique et on utilise ensuite le système à clé publique pour envoyer la clé au destinataire du message. Une clé étant relativement courte, l'algorithme assymétrique s'exécutera quand même rapidement. Le destinataire utilisera sa clé privée pour retrouver la clé qui permet de déchiffrer le message.

Mais le gros problème de la cryptographie à clé publique est lié à l'identification entre une personne et sa clé : m'importe qui peut créer un couple de clés publique et privée, lui associer une identité arbitraire et publier la clé publique. Si je trouve une telle clé sur un serveur, comment être certain qu'elle appartient bien à la personne à qui je destine mon message ? Nous verrons plus loin des solutions pour résoudre ce problème.

Le système à clé publique le plus connu est sans conteste RSA, mais il existe également d'autres systèmes tels que El Gamal, le chiffrement sur les courbes elliptiques, etc.

### 1.2.3 Fonction de hachage

Une fonction de hachage prend des données quelconques d'une longueur arbitraire, et produit une "empreinte digitale", de taille fixe (souvent 128 ou 160 bits). Par contre, il est impossible de retrouver le texte de départ en ne connaissant que l'empreinte.

Une bonne fonction de hachage possède plusieurs propriétés importantes. En particulier, toute modification du texte doit provoquer une modification de l'empreinte. Pour qu'elle soit sûre, il doit également être impossible de trouver deux textes qui ont la même empreinte. Bien sûr, puisque la taille de l'empreinte est fixée alors que les données sont arbitraires, plusieurs ensembles de données donnent la même empreinte, mais le nombre de textes à essayer avant de trouver une "collision" doit être "trop grand" pour que ce soit possible.

Les fonctions de hachage les plus connues et les plus utilisées sont `md5` (*Message Digest 5*, qui crée une empreinte de 128 bits) et SHA (*Secure Hash Algorithm*, 160 bits).

Ces fonctions ont plusieurs utilités. Une des principales est la vérification de l'intégrité d'un fichier, comme un "checksum" : l'auteur d'un programme met à disposition sur son site l'empreinte d'un fichier qu'il distribue, et vous pouvez la comparer à celle calculée sur le fichier en votre possession pour voir si celui-ci a été modifiée entre les deux calculs. Sous Linux, l'utilitaire `md5sum` est fourni en standard pour calculer des empreintes avec MD5 :

```
[schutz@zorglub:~]$ md5sum gnupg-1.0.0.tar.gz
bba45febd501acf8e19db402506dae94  gnupg-1.0.0.tar.gz
```

On utilise également les fonctions de hachage pour la gestion des mots de passe sous Unix : en ne stockant que l'empreinte d'un mot de passe au lieu du mot lui-même, on évite que quiconque puisse accéder à la liste des mots de passe. Pour vérifier un mot de passe, on calcule son empreinte et on le compare ensuite à celui préalablement stocké.

Une dernière application importante est celle des signatures numériques, que nous allons voir dans le prochain paragraphe.

### 1.2.4 Signature digitale

La signature digitale est un concept dérivé de la cryptographie à clé publique, mais fonctionnant "à l'envers". La personne qui veut signer un document électronique utilise sa clé privée pour calculer un petit morceau de données, la signature, qu'elle joint avec son message (qui n'est pas forcément chiffré). Cette signature est calculée de telle façon que toute personne en possession de la clé publique est capable de vérifier que la signature correspond bien au document qu'elle accompagne. Par contre, si le document change ne serait-ce que d'une lettre, la vérification ne marche plus, et la clé publique ne suffit pas pour recalculer une signature valable.

Si vous recevez un tel document signé et que la vérification est correcte, vous êtes certain à la fois de l'intégrité du message et de l'identité de son expéditeur (car lui seul possède la clé privée qui permettait de générer la signature correspondant à ce message).

Le même problème que pour la cryptographie à clé publique apparaît : la vérification me prouve qu'une certaine clé secrète a calculé la signature. Comment être certain que le propriétaire de cette clé est bien celui qu'il prétend être ? Nous verrons plus loin quelles sont les solutions utilisées.

### 1.2.5 Générateurs de nombres aléatoires

Les systèmes de cryptographie que nous avons vus ont besoin de clés pour fonctionner. Pour qu'ils soient sûrs, ces clés doivent être choisies de façon parfaitement aléatoires parmi toutes les clés possibles : si pour une clé de 128 bits, seule une petite fraction des $2^{128}$ clés possibles est utilisée, le système perd entièrement sa sécurité. Le problème du générateur aléatoire est souvent sous-estimé et il est déjà arrivé qu'un mauvais générateur cause la perte de tout un système de

sécurité, par exemple dans Netscape Navigator 2.0, où une valeur d'initialisation mal choisie et pas assez aléatoire suffisait pour réduire à néant toute la sécurité[5].

Fabriquer des nombres aléatoires est quelque chose d'extrêmement difficile pour un être humain : même sans nous en rendre compte, les nombres "au hasard" que nous donnons auront une certaine structure (par exemple, la suite "1111" peut être tirée par un vrai générateur aléatoire, mais un humain l'évitera en raison de son apparente régularité). Etonnamment, le problème est tout aussi compliqué pour un ordinateur, qui est justement conçu pour fonctionner de la manière la plus déterministe possible, et dont on attend qu'il redonne le même résultat chaque fois qu'on relance un programme ! La solution est alors d'utiliser des générateurs *pseudo*-aléatoire, une formule mathématique qui donne des nombres qui ne sont pas réellement aléatoires, puisque obtenus par un programme déterministe, mais qui *semblent* l'être, et il est suffisamment difficile de faire la différence pour que le système soit sûr.

Il reste tout de même un problème : chaque fois que le générateur est démarré, il faut qu'il donne une suite de nombres différente ! Pour cela, l'ordinateur utilise des événements physiques auquel il a accès pour créer une petite valeur d'initialisation, "vraiment" aléatoire, appelée la graine (*seed* en anglais) à partir de laquelle il créé d'autres valeurs pseudo-aléatoires. De tels événement physiques peuvent être par exemple l'heure actuelle du système, l'état de la mémoire (processus, etc) ou d'un périphérique à un instant donné, position de la souris, intervalle de temps entre deux frappes de touches du clavier, etc. En plus de cela, on peut prendre soin de sauvegarder l'état actuel du générateur avant d'éteindre la machine, et de le restorer au prochain démarrage, pour être sûr de ne pas repartir au même point.

Linux possède un très bon générateur de bits aléatoires, et la partie des sources du noyau qui le code est extrêmement bien documentée[6]. Deux devices permettent d'obtenir des nombres aléatoires :

- `/dev/random` fournit toujours des nombres vraiment aléatoires, tirés d'événements physiques. En contrepartie, il peut prendre du temps pour les générer, voire se bloquer s'il manque d'informations pour les construire. Il suffit alors de taper sur le clavier ou de bouger la souris pour faire avancer les choses, mais c'est difficilement faisable sur un serveur, raison pour laquelle on utilise existe un autre device sur ces machines.
- `/dev/urandom` fournit des nombres vraiment aléatoires à l'image de `/dev/random` tant qu'il possible de les créer, mais contrairement à ce dernier, il ne se bloque pas dès le moment où ce n'est plus possible, et utilise à la place un générateur pseudo-aléatoire pour créer des nouveaux nombres à partir des anciens. En l'utilisant, vous ne savez donc pas quelle est la qualité des nombres aléatoires que vous recevrez en retour, mais vous serez assuré d'en obtenir autant que demandé.

D'un point de vue concret, les deux devices fournissent des nombres aléatoires de qualité suffisante pour toutes les applications courantes, y compris du point de vue de la sécurité.

Pour voir la différence entre les deux devices, on peut utiliser la commande `cat` *`device`* et voir quand l'un s'arrête ou recommence à fournir des nombres. La combinaison de touches `CTRL-C` est nécessaire pour arrêter le défilement, et la commande `reset` peut également être utile si l'affichage du terminal a été déréglé.

### 1.2.6 Attaques diverses et tailles de clés

Une erreur commune consiste à croire que la sécurité d'un système de cryptographie réside uniquement dans la force de son algorithme. Bien sûr, cette force est indispensable, mais il existe énormément de moyens d'attaquer un système cryptographique, dont certains semblent très tordues ! On a déjà parlé de Netscape et son mauvais générateur aléatoire, mais on peut aussi citer par exemple :

[5] `http://www.ddj.com/articles/1996/9601/9601h/9601h.htm`

[6] on peut regarder en particulier le fichier `/usr/src/linux/drivers/char/random.c`

- L'attaque par force brute : c'est celle qui a été menée par l'EFF contre DES en 1998 et qui consiste à essayer toutes les clés l'une après l'autre jusqu'à trouver la bonne.
- Les attaques théoriques contre l'algorithme : un algorithme mal conçu peut par exemple garder dans le texte chiffré quelques propriétés statistiques du texte clair, qui peuvent suffire pour le cryptanalyser. D'autres attaques théoriques utilisent par exemple des paires de textes clairs et examinent les différences entre les paires de textes chiffrés correspondants.
- L'analyse de puissance différentielle (DPA, *"differential power analysis"*[7]) : cette attaque permet dans certains cas de trouver certains des paramètres secrets d'une carte à puce (comme par exemple une clé de déchiffrement ou de signature) en mesurant sa consommation électrique.
- L'effet tempest : le rayonnement électromagnétique d'un équipement informatique peut permettre à un attaquant situé à plusieurs dizaines de mètres d'un système informatique de reconstituer l'affichage de son écran ou les frappes de touche du clavier[8]. La seule solution vraiment efficace est la cage de Faraday (mais si vous pensez vraiment en avoir besoin, commencez par vérifier que vous ne souffrez pas d'un complexe de paranoïa !
- Un article de Wired du 2 décembre 2000[9] explique comment le FBI a inséré clandestinement un dispositif d'écoute dans l'ordinateur d'un suspect, et a pu ainsi récupérer les mots de passe utilisés pour protéger des clés cryptographiques au moment où ils étaient tapés au clavier.
- Le facteur humain est sûrement le plus difficile à contrôler, et le meilleur système de sécurité peut s'écrouler si un utilisateur choisit `toto` comme mot de passe, laisse traîner ses clés sans protection ou tombe dans un piège et les donne à quelqu'un qu'il croît être de confiance.

La morale de l'histoire est toujours la même : la sécurité est une chaîne, qui n'est pas plus forte que son maillon le plus faible et il est donc indispensable de se prémunir contre toutes ces attaques à la fois.

Une autre erreur très répandue consiste à comparer la force de différents systèmes uniquement en regardant la longueur des clés qu'ils utilisent. Or, une même longueur de clé peut être suffisante pour un algorithme, alors qu'elle ne le sera pas pour un autre. Ceci est dû au fait que pour certains algorithmes, il existe des attaques autres que la force brute qui permettent de retrouver la clé sans devoir essayer toutes les possibilités. Par exemple, on considère qu'une clé symétrique de 128 bits n'est pas cassable à long terme, alors que pour un système de cryptographie à clé publique basé sur la *cryptographie sur les courbes elliptiques*[10], une même longueur de clé est considérée comme sûre, mais seulement pour un proche avenir. A l'autre extrême, une clé de 128 bits pour le système à clé publique RSA n'offre aucune sécurité, car elle peut être cassée en quelques minutes sur n'importe quel PC standard.

Toujours concernant la taille des clés, il ne faut pas oublier que le nombre de clé possibles augmente de façon exponentielle en fonction du nombre de bits à disposition. Ainsi, une clé symétrique de 128 bits n'offre **pas** deux fois plus de possibilités qu'une clé de 64 bits, mais $2^{64}$ (soit environ $10^{19}$) fois plus ! Par contre, une clé à 65 bits offre effectivement deux fois plus de possibilités qu'une clé de 64 bits.

## 1.3 Certificats : les passeports digitaux

Si les algorithmes de cryptographie à clé publique ou permettant de réaliser des signatures numériques sont mathématiquement très efficaces, on a déjà vu que leur leur implémentation et leur utilisation réelles posent plus de problèmes. En effet, une signature numérique ou une clé secrète ne sont que des suites de bits, aléatoire dans le cas de la clé, ou découlant d'un calcul mathématique pour la signature. En elle-même, une signature ne donne donc aucune information ou garantie sur l'identité du possesseur de la clé : il reste donc à garantir la correspondance entre une clé et une identité.

[7] P. Kocher, `http://www.cryptography.com`
[8] On peut trouver une série de documents sur le sujet sur `http://cryptome.org`
[9] `http://www.wired.com/news/politics/0,1283,40541,00.html`
[10] `http://world.std.com/~dpj/elliptic.html`

A petite échelle, il est possible de garantir directement la correspondance entre une clé et une identité : la personne peut me donner directement sa clé de la main à la main (sur disquette, ou en vérifiant avec moi que la clé que j'ai est bien la sienne). Ceci n'est que rarement possible, par exemple en raison de la distance ou du nombre de personnes qu'il faudrait rencontrer. D'un autre côté, vous ne pouvez pas simplement récupérer une clé sur un serveur web ou par email : rien ne garantit que le nom associé à la clé est bien celui de son possesseur (le site web a pu être piraté, ou l'email peut être truqué). En effet, rien ne m'empêche de créer une clé et de lui attribuer l'identité que je veux. En utilisant cette clé publique par la suite, vous risquez d'envoyer des données confidentielles à une mauvaise personne. Pour un exemple frappant, il suffit d'aller consulter un serveur de clés publiques (par exemple `http:www.keyserver.net`) et de rechercher les clés de personnes connues, tels Bill Clinton, Bill Gates ou Georges Bush — pensez-vous vraiment que ce sont leurs clés ?

Une solution consiste à utiliser des certificats numériques, sorte de passeports du monde digital. Il s'agit d'un fichier contenant la clé publique d'une entité (personne, entreprise, ...) ainsi que les informations sur son identité, le tout signé par une autorité, qui atteste ainsi de la correspondance entre la clé et son propriétaire. Ainsi, si vous faites confiance à cette autorité, il vous suffit de vous assurer que vous possédez bien sa clé publique, et celle-ci vous suffira ensuite pour vous assurer de l'exactitude de toutes les clés signées par cette autorité.

La notion de passeport correspond parfaitement au sens usuel du mot : un passport est un document par lequel l'Etat atteste la correspondance entre vous (en photo) et votre identité. L'apparence d'un passeport, que tout le monde peut contrôler, s'apparente à une clé publique, tandis que les techniques permettant de créer ce passeport sont la clé privée de l'Etat. Si vous connaissez l'apparence des passeports d'un pays et faites confiance à ce pays, vous pourrez facilement vous assurer de l'identité de n'importe lequel des ressortissants de ce pays.

La grand différence du monde numérique avec le monde réel est qu'il n'y existe pas d'autorité étatique reconnue implicitement par tous. C'est donc à chaque utilisateur de décider à quelles autorités il fait confiance ou non. Dans le monde réel, ça revient à se demander si on se satisfait comme preuve d'identité d'une carte bancaire, d'étudiant ou de golfeur — c'est-à-dire à se demander si on fait confiance à une banque, une école ou une association sportive pour authentifier une personne (ce sera sûrement le cas pour une banque, peut-être pour une école, et sûrement pas pour l'association, à moins que vous en soyez membre aussi). De la même façon, certains organismes d'authentification numérique établissent des certificats par email, sans même vérifier l'identité des demandeurs. D'autres, tel le défunt Swisskey[11], demandent que l'on se présente personnellement dans un service d'enregistrement (les chambres de commerce, l'UBS et la Poste en faisaient partie) avec des papiers d'identité officiels avant d'émettre le certificat. L'autorité de loin la plus connue est Verisign[12], qui offre différentes méthodes d'authentification suivant le prix que vous êtes prêt à payer. Il est également possible de créer soi-même son autorité de certification dans le cadre d'une infrastructure de clés publiques (Public Key Infrastructure ou PKI), voir 2.5 pour plus de détails.

D'un point de vue technique, un certificat se présente sous la forme d'un fichier binaire, dont on peut examiner le contenu en format texte à l'aide de la commande `openssl`[13].

On voit bien sur la figure que le certificat contient d'abord les informations sur l'identité de son possesseur, ainsi que sa clé publique (la clé privée correspondante étant uniquement dans les mains du possesseur). La partie signature de la fin garantit l'authenticité des informations précédentes.

Le format binaire se prêtant mal à certaines applications telles que le courrier électronique, on lui préfère souvent le format PEM (*Privacy Enhanced Mail*) qui utilise le codage BASE64 pour donner un fichier composé uniquement des 64 caractères A–Z, a–z, 0–9, + et /, qui sont les moins susceptibles d'être modifiés pendant un envoi. Le certificat ci-dessus transformé en PEM donne un fichier ressemblant à

[11] `http://www.swisskey.ch`
[12] `http://www.verisign.com`
[13] `http://www.openssl.org`

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            01:02:00:00:0f:d1
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=CH, O=Swisskey AG, OU=00851000000500000394, OU=Public CA Services,
                L=Zuerich, CN=Swisskey Personal ID CA 1024
        Validity
            Not Before: Oct 27 13:40:21 2000 GMT
            Not After : Oct 27 23:59:00 2002 GMT
        Subject: O=Private Individual, OU=00851000112470000116, OU=16.03.1975, C=CH,
                 CN=Frederic Schuetz/Email=schutz@mathgen.ch
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:b1:88:5c:03:71:ad:4c:86:7f:be:bf:5e:19:11:
                    7b:fa:d5:30:45:d9:03:07:78:6a:f7:44:72:77:0a:
                    80:2f:be:ab:67:a9:c6:56:4f:b0:c7:50:f6:b8:62:
                    74:c9:1c:d3:6d:92:6b:ee:5c:95:30:ad:da:c7:04:
                    60:01:1c:bf:9b:09:07:d7:4c:38:11:03:a0:9e:f2:
                    cf:31:f7:e0:6e:51:15:e7:7d:5b:9b:3a:20:d9:58:
                    d7:4b:63:c8:33:70:6c:15:93:17:e8:55:da:38:a1:
                    35:4c:5d:ce:86:e4:b8:ab:78:92:f7:ce:74:55:f3:
                    22:7c:7b:b1:35:55:72:aa:61
                Exponent: 65537 (0x10001)

    Signature Algorithm: sha1WithRSAEncryption
        27:99:c4:49:5f:a4:60:1d:cc:43:16:87:b0:78:39:2f:be:ab:
        fb:57:2c:ee:a6:a5:58:c6:25:a6:9a:8e:9c:ee:27:b8:5a:07:
        be:5b:80:8b:8b:bb:75:e6:54:13:f9:8f:06:8e:94:12:04:7c:
        df:79:ad:c2:9e:4a:80:91:68:fe:69:81:1d:60:3d:21:9c:5f:
        06:08:11:59:64:a3:8c:dc:fe:fb:44:0e:d4:d0:09:6a:05:85:
        74:8a:1b:82:45:35:b3:3b:f7:86:4b:59:b7:ee:b1:59:47:cb:
        6e:8c:1a:eb:6f:14:a3:2c:52:dd:c5:a8:59:53:5c:ce:72:fd:
        2c:c1:3a:ea:f5:e6:a2:1d:9d:2c:8e:76:4c:56:a3:59:e8:99:
        a0:d8:4e:e2:d4:2d:54:0d:00:11:e7:f3:8a:9f:51:92:09:8c:
        3e:23:c4:a9:34:01:da:13:2a:9d:8c:d3:e1:93:20:ad:59:ec:
        49:ab:4c:d1:f8:e6:4e:0b:03:d2:da:32:21:97:44:cb:8a:6b:
        d8:26:70:12:c8:18:f3:45:ad:ee:25:75:41:e9:b9:06:38:5f:
        16:3d:61:1e:ee:8f:42:5c:f4:81:6f:fe:79:6b:84:e5:15:d1:
        d9:94:28:d9:81:bc:76:b6:41:e2:f5:3e:5c:8d:c7:fa:29:cd:
        ab:54:c1:8c
```

FIG. 1 – Exemple d'un certificat délivré par SwissKey

```
-----BEGIN CERTIFICATE-----
MIIFATCCA+mgAwIBAgIGAQIAAA/RMA0GCSqGSIb3DQEBBQUAMIGZMQswCQYDVQQG
EwJDSDEUMBIGA1UEChMLU3dpc3NrZXkgQUcxHjAcBgNVBAsTFTAwODUxMDAwMDAw
[...]
nYzT4ZMgrVnsSatM0fjmTgsD0toyIZdEy4pr2CZwEsgY80Wt7iV1Qem5BjhfFj1h
Hu6PQlz0gW/+eWuE5RXR2ZQo2YG8drZB4vU+XI3H+inNq1TBjA==
-----END CERTIFICATE-----
```

## 1.4 Les protocoles SSL et TLS

Il est bien connu que la plupart des protocoles qui régissent Internet ne sont pas sûrs : par exemple, n'importe quelle machine qui est sur le chemin d'un paquet peut le lire ou le modifier, et si plusieurs machines sont connectées à un même segment de réseau connecté par un hub, chacune d'elles a accès aux paquets envoyés et reçus par les autres. Pire encore, il est possible d'usurper l'adresse IP d'une machine (*IP spoofing*) et de recevoir ainsi les paquets qui lui étaient destinés. Tout ceci avait assez peu d'importance avant l'avénement de l'Internet "commercial", mais la transmission de numéros de cartes de crédit et autres informations financières rend indispensables des méthodes de protection. Pour garantir la sécurité d'une communication entre deux applications (un client et un serveur) sur Internet, Netscape a developpé un protocole nommé SSL (*Secure Socket Layer*[14]) qui offre les services suivants :

- confidentialité
- intégrité
- authentification du serveur, et optionnellement du client.

Si SSL est principalement utilisé pour sécuriser la transmission de pages web par le protocole http, il est également adapté à la sécurisation d'autres protocoles (le courrier avec IMAP, POP ou SMTP, le transfert de fichiers avec FTP, etc). Actuellement dans sa version 3.0, il a servi de base à un standard Internet appelé TLS (*Transport Layer Security*[15]).

Le client qui veut entamer une connexion sécurisée envoie au serveur des informations sur sa configuration ainsi qu'un nombre aléatoire. Le serveur répond de même, en y ajoutant son certificat. Le client doit ensuite vérifier le certificat reçu, puis, à l'aide des nombres aléatoires, créer une clé de chiffrement symétrique pour la session, qu'il enverra au serveur de façon chiffrée (en utilisant la clé publique contenue dans le certificat). La session chiffrée commence alors par l'envoi d'un message de test, et à partir de là, tous les échanges seront chiffrés et authentifiés.

Du point de vue de l'implémentation, la bibliothèque `OpenSSL` fournit l'ensemble des fonctions cryptographiques nécessaires, ainsi que des utilitaires en ligne de commande permettant de tester une configuration ou de générer ses propres certificats. Vous pouvez ensuite appeler ces fonctions depuis vos programmes, client ou serveur.

Pour le serveur web Apache, le module `mod_ssl` fournit l'interface avec la bibliothèque `OpenSSL` et lui permet d'offrir des liaisons sécurisées par SSL. Du point de vue des clients, cette fonction est la plupart du temps fournie en standard dans la plupart des navigateurs (`netscape` par exemple), ou disponible dans une version spéciale (`lynx-ssl` pour `lynx`). Le protocole utilisé sera alors https (http sécurisé), et utilisera le port 443 au lieu du port 80 pour http.

Les algorithmes cryptographiques utilisés peuvent êtes arbitraires, cependant les restrictions légales d'exportation des algorithmes trop puissants des Etats-Unis ont longtemps restreint la qualité de chiffrement. Ce sont surtout les navigateurs qui étaient bridés, car des solutions telles que `mod_ssl` pour Apache sont de toute façon développées en Europe et ne souffrent pas de telles restrictions. Pour être sûr du niveau de sécurité offert par votre navigateur, vous pouvez l'utiliser pour aller voir la page `https://www.fortify.net/sslcheck.html`, qui vous détaillera ses capacités, et vous permettra de le "fortifier" s'il est bridé.

[14] `http://developer.netscape.com/docs/manuals/security/ssslin/index.htm`
[15] décrit dans le RFC 2246, `http://sunsite.cnlab-switch.ch/ftp/doc/standard/rfc/22xx/2246`

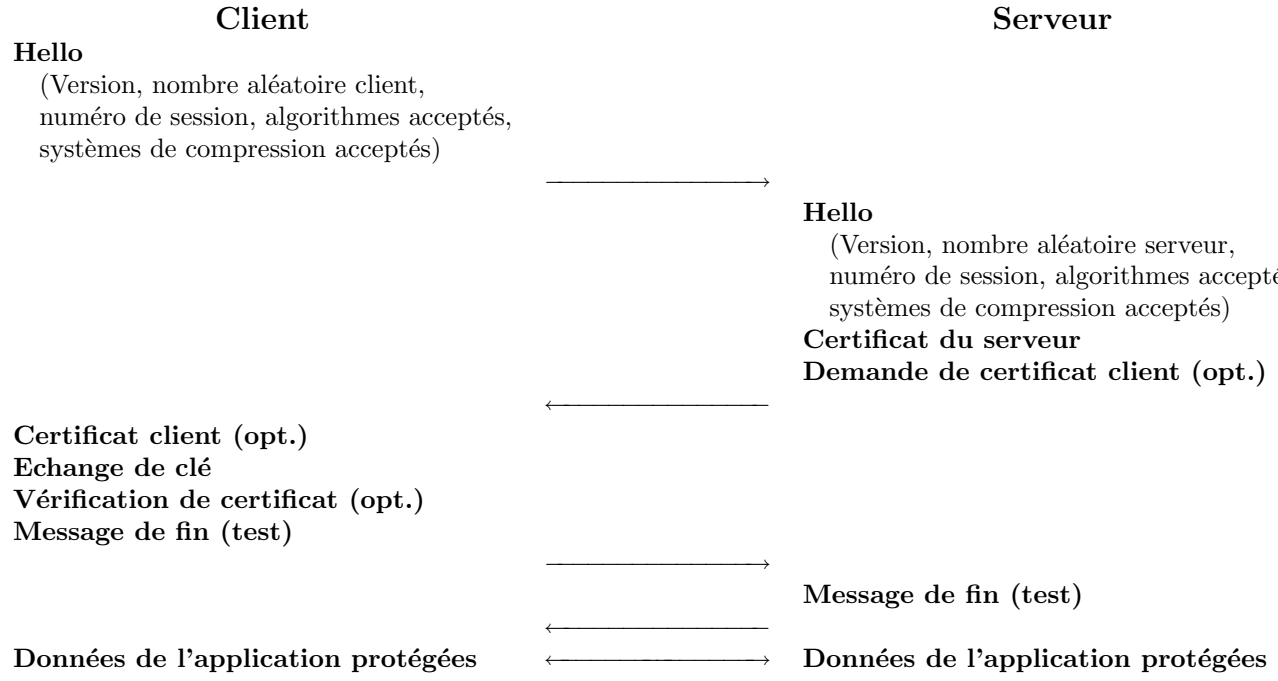


FIG. 2 – Schéma du "handshake" de SSL

# 2 Cryptographie et Linux : quelques applications

## 2.1 L'intérêt du modèle Open Source en sécurité et cryptographie

La sécurité informatique est un domaine complexe, ce qui pose un problème aux utilisateurs : comment savoir quel est le bon produit à utiliser, et quels sont ceux qui sont peu sûrs ? Pour la plupart des applications informatiques, n'importe quel utilisateur est capable de dire si le programme est bon ou pas, suivant le résultat qu'il produit, sa facilité d'utilisation, etc. En sécurité, c'est exactement l'inverse : un bon et un mauvais produit peuvent tourner en parallèle pendant des années sans que l'on ne remarque une seule différence ! C'est uniquement au moment d'une attaque que l'on s'apercevra qu'un des logiciels fait bien son travail tandis que l'autre ne sert à rien. Et ce n'est pas la publicité d'un éditeur qui vous aidera à y voir plus clair ! La seule manière de pouvoir évaluer un logiciel de sécurité consiste à regarder son code source, ou au moins, à s'assurer que ledit code est disponible et que d'autres personnes l'ont examiné. C'est pour cette raison que le modèle open-source est particulièrement adapté aux outils de sécurité.

En cryptographie pure, la situation est encore plus extrême, car on ne dispose pas de méthode permettant de prouver à 100% la sécurité d'un algorithme et seule l'épreuve du temps permet de réellement savoir ce qu'il en est. Si n'importe qui est capable de créer un algorithme "maison" qu'il n'est pas capable de casser lui-même, il est plus difficile de créer un algorithme que les autres n'arrivent pas à casser, et encore plus de faire en sorte qu'il résiste pendant plusieurs années !

La seule solution est de faire confiance à des algorithmes publics et renommés, qui ont bénéficié de l'attention de cryptologues réputés pendant de nombreuses années, plutôt qu'à des systèmes secrets, dont la seule garantie est la parole du vendeur ! *Tous* les algorithmes propriétaires ou qui n'ont pas été soumis à la critique publique se sont révélés mauvais dès que d'autres personnes les ont étudiés (même certaines fois ceux créés par des cryptologues réputés !) Dans ce domaine, l'open-source n'est pas un modèle parmi d'autres, c'est le *seul possible* !

Pour ces raisons, un logiciel (même s'il n'est pas "open source") devrait toujours indiquer les algorithmes qu'il utilise, pour justifier sa sécurité. *Tous* les éditeurs qui refusent de donner cette

information sous prétexte de sécurité essayent en fait de protéger un mauvais algorithme en le gardant secret, et ceci ne tient que jusqu'à ce que quelqu'un décide de décompiler le programme. A ce moment-là, ce sont vos données confidentielles qui seront mises en jeu — tandis qu'avec un algorithme connu, une décompilation n'apprendra rien de plus au pirate.

## 2.2 PGP et GPG

Les deux programmes PGP et GPG permettent de faire du chiffrement et des signatures numériques au niveau de fichiers.

1. PGP (Pretty Good Privacy, Phil Zimmerman, 1991)
   - PGP est le premier programme de cryptographie grand public qui ait été diffusé dans le grand public.
   - Il utilise la cryptographie symétrique pour le chiffrement, et la cryptographie à clé publique pour transmettre la clé et signer.
   - Pendant plusieurs années, il a posé de gros problèmes légaux (algorithmes brevetés, exportations illégales)
   - Network Associate (NAI) l'a racheté entretemps et en a produit des versions commerciales, mais le source est toujours fourni et l'utilisation privée toujours gratuite.
   - Version internationale (exportée légalement) : `http://www.pgpi.com`
2. GPG 1.0 (GNU Privacy Guard, Werner Koch, septembre 1999)
   - GPG est un clône entièrement libre (sous licence GPL) de PGP
   - il n'utilise aucun algorithme breveté (si nécessaire, ceux-ci sont néanmoins disponibles sous forme de modules séparés).
   - entièrement développé en Europe, il n'est pas soumis aux restrictions légales américaines.
   - il est entièrement compatible avec PGP 5.0
   - `http://www.gnupg.org`

Ces deux programmes sont faciles à interfacer avec les lecteurs de mail (par exemple avec `pgp4pine`[16] pour l'interface avec Pine). Entre les deux, le choix de GPG s'impose nettement.

## 2.3 SSH (Secure SHell)

- permet login sur autre machine, exécution de commandes et copie de fichiers
- remplace `rlogin`, `rsh`, `rcp`, qui sont notoirement vulnérables
- fournit une authentification sûre et du chiffrement. En particulier, aucun mot de passe n'est transmis en clair.
- plusieurs méthodes :
  - clé publique (identification de l'hôte ou de l'utilisateur)
  - mot de passe (idem telnet, mais chiffré)
  - ...
- permet d'offrir des connexions X sécurisées ou de sécuriser d'autres protocoles, en faisant du tunneling de connections TCP arbitraires (FTP, POP, ...)
- existe en version libre, voir par exemple `http://www.openssh.com`
- pour plus d'informations : `http://www.employees.org/~satch/ssh/faq/`

## 2.4 Réseaux Privés Virtuels

Un Réseau Privé Virtuel est un réseau sécurisé entre plusieurs hôtes qui sont connectés via un réseau non sûr, par exemple Internet, et de façon transparente pour les utilisateurs finaux.

[16] `http://pgp4pine.flatline.de/`

Plusieurs solutions existent sous Linux, et on peut citer entre autres Free S/WAN (Free Secure Wide Area Network, `http://www.xs4all.nl/~freeswan/`, qui implémente le protocole IPSEC (IP SECurisé) et `CIPE` (Crypto IP Encapsulation, `http://sites.inka.de/~W1011/devel/cipe.html`).

## 2.5 Public Key Infrastructure (PKI)

Si l'on désire créer sa propre autorité de certification dans le cadre d'une PKI, il n'est pas nécessaire de dépenser des sommes énormes pour acheter des systèmes spécialisés : tous les logiciels nécessaires pour construire un tel système facilement sont en effet disponibles dans le monde libre.

La pierre angulaire d'un tel système est le logiciel `openssl` déjà cité. Ce logiciel possède deux fonctions principales : premièrement, il fournit une bibliothèque de fonctions permettant d'établir une communication sécurisée entre deux applications distantes, et deuxièmement, il fournit des utilitaires permettant de créer des certificats numériques, des listes de révocation de certificats, etc, qui sont les fonctions principales dans une PKI. Il est possible de créer une petite autorité de certification en utilisant uniquement ce logiciel, quelques scripts pour simplifier les opérations répétitives et un serveur web tel que Apache pour mettre à disposition des utilisateurs les informations de l'autorité. C'est ainsi que fonctionne par exemple l'autorité de certification de l'Université de Genève, `http://unigeca.unige.ch` (voir le mémoire de licence associé `http://cui.unige.ch/eao/www/memoires/Hugentobler/`).

Pour une PKI de taille plus importante, vous aurez sûrement besoin d'ajouter d'autres logiciels, tels que par exemple OpenLDAP[17] (Lightweight Directory Access Protocol) pour gérer les informations des utilisateurs. Plusieurs projets proposent d'intégrer `openssl`, `apache`, `openldap` et d'autres logiciels, le plus souvent en utilisant Perl, pour créer un vrai logiciel libre de gestion de PKI. On peut citer entre autres :

- OpenCA (`http://www.openca.org/`)
- IDX-PKI (`http://idx-pki.idealx.org/`), certainement la plus aboutie des deux solutions, maintenue par la société française IDEALX (`http://www.idealix.com/`) et déjà installée dans plusieurs grandes institutions.

## 2.6 Chiffrement de disques

Un des désavantages des programmes PGP et GPG est qu'il est nécessaire de les appeler individuellement pour déchiffrer chaque fichier avant de pouvoir le lire ou le modifier, puis de le rechiffrer après modification, avant d'effacer la version en clair. Il existe d'autres méthodes qui travaillent au niveau d'un disque ou d'une partition et qui permettent de travailler de façon transparente pendant un temps voulu, et de rechiffrer automatiquement les données après coup.

Le système de loopback de Linux est la méthode la plus standard et la plus utilisée pour faire cela et on en parlera séparément. Les autres méthodes sont données surtout pour information (on pourra trouver plus d'informations sur la page `http://drt.ailis.de/crypto/linux-disk.html`).

Quelques commentaires s'appliquent néanmoins à tous les systèmes de ce genre en général. Tout d'abord, ils déchiffrent les informations au moment de l'insertion de la partition dans le système de fichier (à l'aide de la commande `mount`) et une fois que c'est fait, les fichiers qui étaient chiffrés ne sont plus soumis qu'aux autorisations d'accès Unix standard, et peuvent donc potentiellement être lues par les autres utilisateurs. En particulier, vous devez avoir confiance en `root` sur la machine si ce n'est pas vous, car vous ne pouvez pas l'empêcher d'accéder aux données une fois déchiffrées. Dans tous les cas, un `root` malicieux peut aussi enregistrer le mot de passe que vous avez tapé au clavier pour accéder aux données plus tard.

Un autre point important concerne les performances : les prévisions les plus optimistes comptent sur un ralentissement de l'ordre de 2 à 4 fois des opérations de lecture et d'écriture sur des partitions chiffrées, en raison du travail supplémentaire pour le chiffrement et le déchiffrement.

[17] `http://www.openldap.org`

### 2.6.1 Loopback

Le loopback est un device Linux qui vous permet de considérer un fichier comme étant un système de fichiers à lui tout seul, et de l'insérer dans votre arborescence. Une des utilisations les plus courantes consiste à prendre l'image ISO d'un CD-Rom et à le monter comme s'il s'agissait d'un réel CD. Ceci se fait en deux étapes : on commence par associer le fichier .iso à un des devices loopback (`/dev/loop0` à `/dev/loop7`) à l'aide de la commande `losetup` (ceci peut nécessiter de recompiler le noyau si le support pour le lookpback n'est pas inclus) :

```
losetup /dev/loop0 /tmp/image.iso
```

et on peut ensuite insérer le système de fichiers dans l'arborescence, comme si on avait affaire à un vrai CD-Rom :

```
mount -t iso9660 /dev/loop0 /cdrom
```

A la fin du travail, on enlève le tout avec

```
umount /dev/loop0
losetup -d /dev/loop0
```

On peut de la même manière créer un système de fichiers chiffré, mais ceci nécessite l'application d'un patch au noyau (en raison des restrictions à l'exportation des Etats-Unis), ainsi qu'à l'utilitaire `losetup`[18]. On peut ensuite créer un fichier vide de la taille désirée (ici 100 Ko), qui contiendra ensuite notre système de fichiers :

```
dd if=/dev/zero of=/file bs=1k count=100
```

On configure ensuite le loopback, en précisant qu'on veut chiffrer, et on donnant l'algorithme désiré (ici blowfish). Celui-ci demande ensuite un mot de passe qui sera utilisé comme clé de chiffrement :

```
losetup -e blowfish /dev/loop0 /file
Password:
```

On peut ensuite créer un système de fichiers, de type `ext2` par exemple, sur notre device, et le monter comme n'importe quel autre système de fichiers :

```
mkfs -t ext2 /dev/loop0 100
mount -t ext2 /dev/loop0 /mnt
```

Une fois qu'on aura fini de travailler dessus, il suffit de le démonter et de détacher le fichier du loopback, et le système sera en sécurité, chiffré à l'intérieur du fichier :

```
umount /dev/loop0
losetup -d /dev/loop0
```

### 2.6.2 Quelques autres systèmes

1. **ehd** (Encrypted Home Directory) : ehd permet de chiffrer son répertoire personnel en utilisant le système du loopback, et se présente sous la forme d'un patch pour l'utilitaire `login`. Dès que vous entrez votre mot de passe et vous logguez, votre répertoire personnel est déchiffré est monté. Au moment où vous faites un logout, il est démonté. Vos données ne sont donc jamais accessibles quand vous n'êtes pas connecté.

[18] On peut trouver les patches et toute la documentation nécessaire sur `http://www.kerneli.org`

2. **CFS** (Cryptographic File System), de Matt Blaze : CFS ne nécessite pas de toucher au noyau et se greffe au dessus de NFS pour éviter que les fichiers ne soient transmis en clair. Il est malheureusement assez peu transparent pour l'utilisateur, puisqu'il faut utiliser des commandes spéciales pour accéder aux fichiers.
3. **TCFS** (Transparent Cryptographic File System) améliore nettement CFS en rendant toute la procédure transparente pour l'utilisateur. En contrepartie, il nécessite un patch pour le noyau. TCS stocke les blocs chiffrés sur le serveur et les envoie tels quels au client, qui est responsable de les déchiffrer. De cette manière, seul le `root` du client doit être de confiance, car même celui du serveur ne peut pas accéder aux données. TCFS est assez bien supporté (nouvelles versions récentes), on peut trouver plus d'informations sur le site `http://tcfs.dia.unisa.it/`
4. **ppdd** (Practical Privacy Disc Driver) : ppdd utilise également un patch sur le noyau, mais il offre une protection au niveau de la machine : dès que `root` a activé le driver, la partition est accessible à tout le monde, selon les permissions Unix standards. La manière dont il est construit permet entre autres de chiffrer la partition `/` du système. On peut trouver des informations à `http://linux01.gwdg.de/~alatham/ppdd.html`.
5. **StegFS** (Steganographic File System) : StegFS est plus une curiosité qu'un système vraiment utile, mais il n'en est pas moins intéressant. Il est capable de créer un système de fichiers dont même l'existence est tenue secrète : seule la personne qui sait qu'il est là (et qui connaît un mot de passe) est capable d'y accéder ! Il fonctionne en deux étapes : tout d'abord, il créé un système `ext2` standard, qu'il remplit avec des données aléatoires. Pour les fichiers normaux, il les stocke sur ce système. Pour des fichiers cachés, il utilise le nom du fichier et la clé pour dériver un emplacement sur un bloc libre, il chiffre le fichier et le stocke sur ce bloc. Pour effacer un fichier, il le recouvre de données aléatoire. Le problème évident est que s'il y a trop de fichiers normaux, les fichiers chiffrés sont effacés (car il n'est pas possible de savoir où ils sont sans le mot de passe). Pour éviter cela, les fichiers cachés sont écrits à plusieurs emplacements, et il ne faut remplir qu'une petite proportion du disque avec des fichiers normaux. Il existe plusieurs implémentations de ce concept, on peut en trouver une sur `http://ban.joh.cam.ac.uk/~adm36/StegFS/`

Sans donner de détails, il existe d'autres systèmes, on peut citer entre autres BestCrypt (`http://www.jetico.com/`), qui est un logiciel commercial, mais livré avec ses sources, et qui est aussi disponible sous MS-Windows et MacOS, ou CryptFS (`http://www.cs.columbia.edu/~ezk/`).

# 3 References

**Quelques sites Internet.**

Il est vivement recommandé de s'abonner à la lettre de Bruce Schneier pour être tenu au courant de la vie du monde de la cryptographie et de la sécurité.

- Site généraliste sur la cryptographie
  `http://www.ssh.fi/tech/crypto/`
- "Crypto-Gram", lettre d'information mensuelle par email de Bruce Schneier
  `http://www.counterpane.com/crypto-gram.html`
- "FAQ sur la taille des clés" de Thomas Pornin
  `http://www.di.ens.fr/~pornin/faq-cle.html`
- "Crypto Law Survey" de Bert-Jaap Koops, le point sur les législations nationales
  `http://cwis.kub.nl/~frw/people/koops/lawsurvy.htm`
- "FAQ about today's cryptography" de RSA laboratory
  `http://www.rsasecurity.com/rsalabs/faq/`
- "Linux Security HOWTO" de Kevin Fenzi et Dave Wreski
  `http://scrye.com/~kevin/lsh/`
- "Encryption and Security Tutorial" de Peter Gutmann
  `http://www.cs.auckland.ac.nz/~pgut001`

**Quelques livres.**

Les deux premiers livres cités ont une approche historique du sujet, alors que les suivants sont classés (subjectivement) du moins au plus technique. Si vous n'en lisez qu'un, prenez celui de Jacques Stern. Il est concis, peu technique et fait un bon tour du sujet pour une première approche.

- Simon Singh, "The CodeBook". *Un livre récent sur l'histoire de la cryptographie, traduit en français. Très vulgarisé, un peu trop quelques fois.*
- David Khan, "The codebreakers". *Livre historique sur la cryptographie pré-informatique. Malgré ses 30 ans, il reste la bible sur le sujet. Une des éditions antérieures a été traduite en français.*
- Jacques Stern, "La science du secret", Odile Jacob. *Une très bonne introduction à la cryptographie sous un angle non-technique.* `http://www.dmi.ens.fr/~stern/`
- Bruce Schneier, "Applied cryptography", John Wiley & Sons (1996). *Un excellent livre, largement plus technique mais restant abordable, il mériterait néanmoins une petite mise à jour. Existe en français.* `http://www.counterpane.com/applied.html`
- Alfred J. Menezes, Paul C. Van Oorschot, Scott A. Vanstone, "Handbook of Applied Cryptography", CRC Press Series on Discrete Mathematics and Its Applications. *Un peu sec, c'est une vraie bible, a utiliser comme outil de référence. Disponible intégralement sous forme postscript ou PDF.*
  `http://www.cacr.math.uwaterloo.ca/hac/`
- Douglas Stinson, "Cryptographie : théorie et pratique." *Beaucoup plus théorique, il explique les mathématiques qui sont derrière la cryptographie.* (traduit de l'anglais)
  `http://www.cacr.math.uwaterloo.ca/~dstinson/index.html`