Cryptographie et Linux

Frédéric Schütz

Frederic.Schutz@cui.unige.ch

CUI, Université de Genève

Jeudi 21 octobre 1999

Plan

1) Introduction à la cryptographie

- Vocabulaire
- Historique rapide
- Algorithmes symétriques
- Algorithmes asymétriques
- Fonctions de hashage
- Signature digitale
- Nombres aléatoires
- Attaques diverses et tailles de clés.
- Questions légales

2) Cryptographie et Linux

- L'intérêt du modèle Open Source pour les outils de cryptographie et de sécurité en général
- Quelques applications sous Linux : PGP/GPG, SSH, chiffrement de disques.

Vocabulaire

- **Cryptographie** : *Discipline qui englobe tous principes, moyens et méthodes destinés à la transformation de données afin de cacher leur contenu, d'empêcher leur modification ou leur utilisation frauduleuse.*
- **Chiffrer** : *Transformer un message en clair en un message inintelligible.*
- **Déchiffrer** : *Transformer un message chiffré en un message en clair en connaissant la méthode.*
- **Décrypter** : *idem, mais sans connaître la méthode.*
- **Cryptanalyse** : *ensemble des procédés permettant de décrypter.*
- **Stéganographie** : *Ensemble des techniques qui consistent à cacher l'existence même d'un message (par ex. cacher un message dans les bits de poids faible d'une image).*

Historique rapide

- existe à peu près depuis l'invention de l'écriture (Egypte, Jules César, ...)
- jusqu'à 1970 : pratiquement inconnue en dehors des milieux militaires et diplomatiques.
- développement de l'informatique + réseaux
  $\Longrightarrow$ apparition dans le domaine civil.
- 1976 : invention de la cryptographie à clé publique (W. Diffie et M. Hellmann)
- 1977 : choix du standard DES (Data Encryption Standard)
- 1978 : algorithme RSA (Rivest, Shamir, Adleman)
- 1991 : sortie de PGP
- 1998 : 1ère démo concrète que DES n'est plus sûr.
- 2001 (?) : choix de l'algorithme AES (Advanced Encryption Standard), successeur de DES.

Algorithmes symétriques

Clé : *le paramètre secret d'un chiffrement, contrairement à la marche à suivre, l'algorithme, qui peut être publique.*

La même clé est utilisée pour chiffrer et déchiffrer les messages.

Marche à suivre :

1. je génère une clé aléatoirement ;
2. je la transmets à mon correspondant de manière sûre ;
3. je l'utilise pour chiffrer mon message ;
4. il utilise la même clé pour le déchiffrer.

Avantages : algorithmes rapides

Désavantages :

– il faut disposer d'un canal sûr pour échanger la clé.

– besoin d'échanger d'une clé pour chaque correspondant (50 correspondants = 50 clés à garder secrètes).

Exemples : DES, 3DES, IDEA, Blowfish, ...

Algorithmes asymétriques

Idée :

- clé de chiffrement $\neq$ clé de déchiffrement
- la connaissance de l'une des clés ne permet pas de connaître l'autre en un temps acceptable.

$\Longrightarrow$ clé de chiffrement peut être diffusée publiquement

Marche à suivre :

1. Je génère une clé publique/privée
2. Je garde la clé privée et publie l'autre (page web, serveur de clés, annuaire)
3. Celui qui veut m'écrire récupère ma clé et l'utilise pour chiffrer un message qu'il m'envoie
4. Je déchiffre ce message avec ma clé privée.

Avantages :

- transmission de clé facilitée
- une seule clé suffit pour tous les gens qui veulent m'écrire

Désavantages :

- lent
- authentification des clés ?

Exemples : RSA, El Gamal, EC, ...

Fonctions de hashage

Principe : produit une “empreinte digitale” à partir de données quelconques.

Propriétés :

- taille de l’empreinte fixe (souvent 128 ou 160 bits) ;
- toute modification du texte provoque une modification de l’empreinte ;
- donné l’empreinte, quasiment impossible de retrouver le texte ;
- donnés deux textes, quasiment impossible qu’ils aient la même empreinte.

Applications :

- vérification de l’intégrité d’un fichier ;
- vérification des mots de passe ;
- signatures.

Exemples : MD5, SHA.

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

Le principe des mots de passe sous Unix

- mots de passe ne sont pas stockés en clair
- empreinte stockée dans `/etc/passwd`
- fonction hashage : DES modifié (texte chiffré impossible à déchiffrer)
- salt : paramètre de chiffrement aléatoire, 4096 possibilités ; même mot de passe chiffré deux fois donne deux résultats différents

$$\underbrace{\texttt{schutz}}_{\text{userid}} : \underbrace{\texttt{MN}}_{\text{salt}} \underbrace{\texttt{ax0hcJfa0HA}}_{\text{hash password}} : \underbrace{\texttt{1211}}_{\text{uid}} : \underbrace{\texttt{350}}_{\text{gid}} : \underbrace{\texttt{SCHUETZ Frederic}}_{\text{description}} :$$

$$\underbrace{\texttt{/user/l1/schutz}}_{\text{home dir}} : \underbrace{\texttt{/bin/csh}}_{\text{shell}}$$

Attaques :

- Impossible de retrouver le mot de passe à partir de l'empreinte
- Essayer des mots de passe (dictionnaire) jusqu'à trouver le bon

Solutions :

- Shadow passwords : mots de passe stockés dans un fichier interdit en lecture
- Faire soi-même l'attaque pour trouver les mots de passe faciles
- Remplacer DES par un autre système plus lent (PAM)

Signature digitale

Buts :

- authentifier la personne qui a écrit un message
- vérifier que ce message n'a pas été changé pendant son transfert.

Utilise la cryptographie asymétrique "à l'envers" :

3'. Quand je veux signer un message, je le chiffre avec ma clé privée.

4'. Celui qui veut vérifier la signature déchiffre le message avec la clé publique.

Si le résultat est bon, je suis sûr :

- de l'identité de l'expéditeur (lui seul possédait la clé permettant de chiffrer).
- que personne n'a modifié le message en route (sinon déchiffrement incorrect).

Pratiquement : on signe l'empreinte du message (+ rapide)

Problème à résoudre : Signature n'est sûre que si on est certain du propriétaire de la clé, ce qui n'est pas le cas pour une signature récupérée sur un serveur.

Solutions : PKI (Public Key Infrastructures), signatures de clés, certificats, ...

Nombres aléatoires

But d'un ordinateur : faire des calculs les plus exacts, de manière totalement prévisible

$\Longrightarrow$ difficile d'obtenir d'un ordinateur des nombres vraiment aléatoires.

En général : nombres pseudo-aléatoires (donnés par une formule mathématique et qui semblent aléatoires)

Solution (tirée de PGP) :

- récupérer information aléatoire de l'environnement (intervalle entre deux touches tapées, etc)
- construire un réservoir de bits aléatoires.
- utiliser une fonction de hashage pour obtenir le résultat.

Linux : deux devices

1. `/dev/random`
   - fournit des nombres de bonne qualité,
   - peut prendre du temps s'il manque de matière pour les construire
2. `/dev/urandom` :
   - fournit autant de nombres qu'on veut
   - sans garantie de qualité.

(réf. `/usr/src/linux/drivers/char/random.c`)

Attaques diverses et tailles de clés

Énormément d'attaques différentes (contre un algorithme ou son implémentation) :

- force brute [DES en 1998, `http://www.eff.org`]
- attaques théoriques contre l'algorithme
- *"differential power analysis"* : attaque les smart cards (utilisant p.ex. RSA) en mesurant la consommation électrique [P. Kocher, `http://www.cryptography.com`]
- Netscape : bon algorithme, mais mauvais générateur aléatoire.

sécurité globale = maillon le plus faible de la chaîne !

Erreurs fréquentes sur la taille des clés :

clé symétrique de 128 bits (pas cassable à long terme)

$\neq$ clé asymétrique EC de 128 bits (pas cassable à court terme)

$\neq$ clé asymétrique RSA de 128 bits (cassable avec un PC)

clé symétrique de 128 bits

$\neq$ 2 fois plus de possibilités que clé symétrique à 64 bits

Cryptographie : un art difficile

- **Facile** : créer un algorithme qu'on n'est pas capable de casser soi-même.
- **Plus difficile** : créer un algorithme que les autres n'arrivent pas à casser
- **Encore plus difficile** : ... même après plusieurs années d'études
- **Impossible** : garantir qu'un algorithme donné est sûr à 100 %

Conclusions

- Mieux vaut faire confiance à des algorithmes publics, qui ont bénéficié de l'attention des cryptanalystes connus pendant de nombreuses années...
- ... plutôt qu'à des systèmes secrets, dont la seule garantie est la parole d'un technicien ou d'un vendeur.

... ça ne vous rappelle rien ?

L'intérêt du modèle Open Source pour les outils de sécurité et de cryptographie

- La plupart du temps : même sans le source, possible de dire si un programme est bon ou mauvais (fonctionalité)
- Sécurité $\neq$ fonctionalité : mauvais outil de sécurité ressemble comme deux gouttes d'eau à un bon outil (efficace, ne crashe pas, chiffre et déchiffre, ...)
- Seule manière de faire la différence : faire examiner le programme (source)
- ...et encore, ça ne suffit pas toujours !

Règle d'or en cryptographie : ne <u>jamais</u> utiliser d'algorithme qui n'a pas été soumis à la critique publique.

Conséquences :

- Aucun algorithme "propriétaire" ne peut être considéré comme sûr (exception : NSA)
- Sécurité $\longleftrightarrow$ obscurité : la sécurité d'un système ne doit pas être basée uniquement sur le fait que l'algorithme est secret (décompilation).
- Un bon programme qui utilise la cryptographie doit absolument indiquer quels algorithmes il utilise.

Questions légales

1. Deux objectifs contradictoires :

Développement de la crypto pour protection de la vie privée et de l'industrie	$\longleftrightarrow$	restriction de la crypto pour permettre à l'État d'accéder aux données privées (± légalement)

CH
- quasiment aucune restriction (secret bancaire et bonne industrie cryptographique oblige)
- excepté accords de Wassenaar (successeur du Cocom ; ne concerne pas les logiciels libres).

F
- soumis à autorisation et limitation de puissance
- conditions irréalistes (et inefficaces !) jusqu'en mars 1999.

  $\Longrightarrow$ limite assouplie en attendant changement de la loi.

USA
- utilisation libre
- exportation très réglementée (matériel de guerre).
- !! download = exportation !!
- autorisé sur papier ("free speech")

Conséquences pour Linux :

- manque bons outils de cryptographie dans le noyau (`http://www.kerneli.org`)
- manque bons outils dans certaines distributions

2. Brevet sur certains algorithmes

RSA (expire l'an prochain), IDEA, ...

PGP (Pretty Good Privacy)
GPG (GNU Privacy Guard)

1. PGP (Phil Zimmerman, 1991)
   - 1er programme de cryptographie grand public
   - cryptographie symétrique pour le chiffrement ;
   - cryptographie à clé publique pour transmettre la clé et signer.
   - gros problèmes légaux (algorithmes brevetés, exportations illégales)
   - racheté par NAI $\longrightarrow$ versions commerciales (source toujours fourni et gratuit pour utilisation privée)
   - version internationale (exportée légalement) : `http://www.pgpi.com`
2. GPG 1.0 : (Werner Koch, septembre 1999)
   - clône entièrement libre (GPL) de PGP
   - sans algorithmes brevetés
   - développé en Europe
   - entièrement compatible avec PGP 5.0
   - `http://www.gnupg.org`

Faciles à interfacer avec lecteurs de mail
(p.ex. Pine avec pgp4pine, `http://pgp4pine.flatline.de/`).

SSH (Secure SHell)

- permet login sur autre machine, exécution commande, copie fichiers
- remplace `rlogin`, `rsh`, `rcp`, notoirement vulnérables
- authentification + chiffrement (pas de mots de passe transmis en clair)
- plusieurs méthodes :
  - clé publique (identification de l'hôte ou de l'utilisateur)
  - mot de passe (idem telnet, mais chiffré)
  - ...
- connexions X sécurisées
- forwarding de connections TCP arbitraires (FTP, POP, ...)
- 2 versions du protocole, non-compatibles
  - SSH1 : logiciel à licence très libre, disponible sur beaucoup de plateformes + ports non commerciaux
  - SSH2 : plus sûr, moins libre, peu disponible
- `http://www.employees.org/~satch/ssh/faq/`

Chiffrement de disques

Source : `http://drt.ailis.de/crypto/linux-disk.html`

- Attention sur un système multi-utilisateurs : système déchiffré et monté $\Longrightarrow$ uniquement soumis aux autorisations d'accès UNIX standards.
- ralentissements (lectures et écritures 2 à 4 fois plus lentes)

1. Loopback Encryption :
   - Loopback device : *permet de monter un fichier normal comme système de fichier.*
   - nécessite patch du noyau (`http://www.kerneli.org`)
   - option lors de l'initialisation du loopback (`losetup`)
   - exemple complet : `man losetup`
2. ehd (Encrypted Home Directory)
   - patch `login`
   - utilise Loopback Encryption pour un répertoire personnel.
   - accessible dès login, retiré automatiquement au logout.

Chiffrement de disques (II)

3. CFS (Cryptographic File System)
   - ne nécessite pas de toucher au noyau
   - se greffe sur NFS, permet d'éviter transfert de fichiers en clair.
   - peu transparent pour l'utilisateur (commandes spéciales pour accéder aux fichiers)
4. TCFS (Transparent Cryptographic File System)
   - améliore CFS - complétement transparent
   - patch pour le noyau, 2.0.x seulement pour l'instant
   - blocks chiffrés sur le serveur, envoyés tels quels et déchiffrés par le client.
   - seule personne qui doit être de confiance : root du client.
   - `http://tcfs.dia.unisa.it/`
5. ppdd (Practical Privacy Disc Driver)
   - patch du noyau
   - protection au niveau de la machine : dès que root active le driver, accessible par tout le monde (permissions standards)
   - possible de chiffrer la partition root
   - `http://linux01.gwdg.de/~alatham/ppdd.html`

Chiffrement de disques (III)

6. StegFS (Steganographic File System)

   Cache l'existence même d'un système de fichiers.

   – Crée un système ext2, remplit avec données aléatoires.
   – Fichiers normaux : utilise ext2
   – Fichiers cachés : utilise la clé et le nom du fichier pour dériver un emplacement sur un bloc libre
   – Chiffre et stocke le fichier
   – Effacement de fichier : écrit des données aléatoires.

   Problème :

   disque trop plein $\Longrightarrow$ fichiers cachés sont effacés

   – petite proportion du disque utilisée
   – fichiers inscrits plusieurs fois.

   `http://ban.joh.cam.ac.uk/~adm36/StegFS/`

7. CryptFS

   `http://www.cs.columbia.edu/~ezk/`

8. BestCrypt

   – commercial, mais livré avec sources
   – disponible sour MS Windows et MacOS
   – `http://www.jetico.com/`