# IFT 3051 – Les canaux subliminaux dans DSA

Thomas Leplus

25 avril 2003

# Table des matières

**1 Introduction** **7**

**2 DSA** **9**
2.1 Présentation . . . . . 9
2.2 Génération de la clé . . . . . 9
2.3 Génération de la signature . . . . . 10
2.4 Vérification de la signature . . . . . 10

**3 Les canaux subliminaux dans DSA** **11**
3.1 Canal par divulgation de la clé privée . . . . . 11
3.2 Canal par convention . . . . . 12
3.3 Canal par partage de secret . . . . . 13
3.4 Prévention et détection . . . . . 13

**4 Notre librairie cryptographique** **15**
4.1 Méthodologie de développement . . . . . 15
4.1.1 Java . . . . . 15
4.1.2 Jar . . . . . 16
4.1.3 Javadoc . . . . . 16
4.1.4 LaTeX . . . . . 16
4.1.5 LaTeX2HTML . . . . . 16
4.1.6 Make . . . . . 16
4.1.7 CVS . . . . . 17
4.2 Algorithmes . . . . . 17
4.2.1 SHA-1 . . . . . 17
4.2.2 DSA . . . . . 17

**5 Notre application** **19**
5.1 Application principale . . . . . 19
5.2 Génération de clés . . . . . 20
5.3 Signature de fichiers . . . . . 20
5.4 Vérification de signatures . . . . . 22

**6 Conclusion** **29**

**7 Références** **31**

# Table des figures

5.1 Fenêtre principale de l'application. . . . . 21
5.2 Génération de clés DSA. . . . . 21
5.3 Liste de tailles de clés. . . . . 23
5.4 Sélection de fichier. . . . . 24
5.5 Résumé d'une génération de clé. . . . . 25
5.6 Signature de fichier. . . . . 26
5.7 Résumé d'une signature de fichier. . . . . 26
5.8 Vérification de signature. . . . . 27
5.9 Résumé de vérification d'une signature (et de son canal). . . . 27

# Chapitre 1

# Introduction

La conception d'algorithmes et de protocoles cryptographiques est une tâche difficile qui demande beaucoup de travail et d'expérience. Et malgré cela, il n'y a souvent aucune garantie que le résultat soit parfaitement sûr. Seules des années de cryptanalyse sans succès peuvent donner confiance en un nouvel algorithme.

Cependant, l'histoire a montré que de nouvelles techniques de cryptanalyse sont régulièrement misent au point (cf. la cryptanalyse différentielle). Et lorsque que ce n'est pas la science qui vient à bout d'un algorithme, c'est souvent la technologie qui s'en charge en fournissant des machines de plus en plus rapides et efficaces (cf. DES).

Parfois aussi, certaines personnes trouvent de nouvelles façons d'utiliser les algorithmes existants. C'est le cas par exemple de GUSTAVUS SIMMONS. Celui-ci avait pour mission vers la fin de la Guerre Froide d'étudier la faisabilité d'un système qui permettrait aux Russes et aux Américains de vérifier en permanence et automatiquement l'étendue de leur arsenal nucléaire respectif. Les conditions pour qu'un tel système puisse être accepté par les deux camps étaient que :

1. Le système ne peut dévoiler qu'une seule information : le nombre total de missiles de l'adversaire (pas leur localisation).
2. Toute tricherie est impossible (en particulier, un missile ne peut pas être compté plusieurs fois par recensement).
3. Le résultat du recensement doit être transmis à chaque camp de façon sécurisée (c'est-à-dire confidentielle, authentifiée et intègre).

Comme SIMMONS l'a raconté lors de sa conférence à l'Université de Cam-

bridge en 1996 [5], il s'est rapidement rendu compte que la condition 1 est nécessairement contradictoire avec la condition 3 car, dans un canal de communication authentifié, on peut toujours sacrifier une part des bits destinés à l'authentification pour transmettre des bits d'information, et cela de façon indétectable pour quelqu'un qui observe le canal. Il a baptisé ce genre de canaux de communication "canaux subliminaux".

SIMMONS a été le premier à réaliser le potentiel de ses canaux subliminaux. Ils peuvent permettre de communiquer de l'information secrète à travers un canal qui est authentifié mais pas confidentiel. Ils peuvent également être utilisés par un programmeur mal intentionné pour qu'un logiciel de signature numérique dévoile la clé privée du signataire à l'intérieur même de sa signature !

Nous allons présenter notre propre implantation des canaux subliminaux dans DSA.

# Chapitre 2

# DSA

## 2.1 Présentation

Le *Digital Signature Algorithm* a été proposé en 1991 par le *National Institute of Standards and Technology* [2]. C'est un système de signature numérique basé sur l'algorithme ElGamal donc, tout comme ElGamal, DSA peut être le support d'un canal subliminal. Notez par ailleurs que l'algorithme russe GOST est quasiment identique à DSA donc toutes les techniques proposées pour DSA sont facilement applicables à GOST.

## 2.2 Génération de la clé

Pour générer une clé DSA, Alice doit suivre la procédure suivante :

1. Choisir un nombre premier aléatoire $p$ tel que $2^{511} < p < 2^{1024}$ ;
2. Choisir un nombre premier aléatoire $q$ tel que $2^{159} < p < 2^{160}$ et $q|p-1$ ;
3. Trouver un nombre $g \in \mathbb{Z}_p^*$ qui génère le seul groupe cycle d'ordre $q$, c'est-à-dire tel que $\exists a \in \mathbb{Z}_p^*, a^{\frac{p-1}{q}} \neq 1 \bmod p$ ;
4. Choisir un nombre aléatoire $x \in \mathbb{Z}_q^*$ ;
5. Calculer $y = g^x \bmod p$.

La clé privée d'Alice est alors $x$ et sa clé publique est $(p, q, g, y)$.

## 2.3 Génération de la signature

Pour signer un document $m \in \mathbb{N}$, Alice doit tout d'abord calculer le hachage de $m$. Le standard DSA prévoit explicitement l'usage de la fonction de hachage cryptographique SHA-1. Donc Alice calcule $h$ le hachage de $m$ par SHA-1. On note pour la suite que SHA-1 produit des résultats de 160 bits donc $h \in \mathbb{Z}_q^*$.

Ensuite, pour chaque signature, Alice choisit aléatoirement une clé de session $k \in \mathbb{Z}_q^*$.

Enfin, Alice calcule :

$$r = (g^k \bmod p) \bmod q$$
$$s = k^{-1}(h + xr) \bmod q$$

La signature de $m$ est $(r, s)$.

## 2.4 Vérification de la signature

Pour vérifier la signature $(r, s)$ du document $m$, Bob doit tout d'abord recalculer $h$, le hachage SHA-1 de $m$, et se procurer la clé publique $(p, q, g, y)$ d'Alice.

Ensuite, Bob calcule :

$$u_1 = hs^{-1} \bmod q$$
$$u_2 = rs^{-1} \bmod q$$
$$v = (g^{u_1} y^{u_2} \bmod p) \bmod q$$

La signature est authentique si et seulement $v = r$.

# Chapitre 3

# Les canaux subliminaux dans DSA

Dans un article [4] de 1993, SIMMONS étudie les différents canaux subliminaux dans DSA. Nous allons tout d'abord présenter comment implanter le même canal subliminal dans DSA que dans ELGAMAL. Ensuite, nous montrerons comment améliorer notre canal.

## 3.1 Canal par divulgation de la clé privée

Dans cette première version du canal subliminal de DSA, Alice veut signer un document $m \in \mathbb{N}$ de sorte que la signature soit valide et que Bob puisse extraire de la signature un message subliminal $m' \in \mathbb{Z}_q^*$ (160 bits).

Pour ce faire, Alice doit tout d'abord partager avec Bob sa clé privée ($a$). Plus tard, lorsque Alice veut transmettre $m'$ à Bob, Alice calcule la signature $(r, s)$ de $m$ normalement si ce n'est qu'au lieu de prendre une clé de session $k \in \mathbb{Z}_q^*$ aléatoire, Alice prend $k = m'$. Ainsi, lorsque Bob récupère $m$ et $(r, s)$, il peut calculer :

$$s^{-1}(h + ar) = k = m'$$

Bob a bien récupéré le message subliminal $m'$.

Dans cette implantation, le principe est donc d'utiliser la clé de session $k$ pour convoyer le message subliminal. Or cette clé de session est supposée être aléatoire. Les propriétés statistiques du message subliminal risquent donc fortement de compromettre la sécurité de la clé privée. En effet, SIMMONS a montré que l'incertitude sur $x$ est liée à l'incertitude sur $k$. Un attaquant

récupérant un grand nombre de signatures contenant un message subliminal pourrait donc obtenir de l'information sur la clé privée (le nombre $x$) en faisant d'habiles suppositions sur le contenu et la forme des messages subliminaux $m'$ (le nombre $k$).

Une solution consiste à chiffrer le message subliminal pour qu'il retrouve une apparence aléatoire. En particulier, si Bob aussi dispose d'une pair de clés DSA pour pouvoir répondre "subliminalement" à Alice, Alice peut utiliser la clé publique de Bob pour chiffrer les messages subliminaux qui lui sont destinés, de même que Bob peut utiliser la clé publique d'Alice pour lui répondre. Notez qu'il n'est pas évident que l'on puisse chiffrer des messages avec une clé publique DSA (rappelons qu'il ne s'agit à l'origine que d'un système de signature). Pourtant, il a été montré [3] que l'on peut utiliser la fonction de signature DSA pour réaliser RSA et ElGamal !

## 3.2 Canal par convention

Une version un peu spéciale du canal par divulgation de la clé privée permet à Alice de transmettre sa clé privée à Bob sans partager de secret au préalable. Cette technique peut donc être utilisée pour initialiser le canal subliminal : la première signature contient la clé privée et les signature suivantes contiennent des messages.

L'idée est que Alice choisit $k = x$. Ainsi, la signature du message $m$ par Alice devient :

$$r = (g^k \bmod p) \bmod q = (g^x \bmod p) \bmod q = y \bmod q$$

$$s = k^{-1}(h + xr) \bmod q = x^{-1}(h + xy) \bmod q = x^{-1}h + r \bmod q$$

d'où :

$$x = h(s - r)^{-1} \bmod q$$

Bob peut donc retrouver $x$ uniquement à partir de la clé publique de Alice et de la signature.

Le problème est que si tous les paramètres nécessaires pour trouver $x$ sont publiques, tous le monde peut le faire. Il ne faut donc pas qu'un adversaire suspecte que ce canal a été utilisé.

## 3.3 Canal par partage de secret

Le canal subliminal proposé précédemment souffre d'un inconvénient majeur : le signatoire doit révéler sa clé privée au destinataire du message subliminal et cela permet automatiquement à ce destinataire de faire de faussses signatures.

Cependant, DSA permet de transmet quelques bits de messages subliminal sans divulger la clé privée du signataire. Pour cela, les interlocuteurs doivent partager un secret. Plus précisemment, Alice et Bob doivent partager un nombre premier aléatoire $e > q$. Ensuite, lors de la signature d'un message $m$, Alice choisit une clé de session $k$ de sorte que $r$ soit ou non un résidu quadratique modulo $e$ selon qu'Alice veuillent transmettre 0 ou 1. Comme les résidus quadratiques et non quadratiques sont équiprobables et que l'on dispose du symbole de LEGENDRE pour tester la résiduosité quadratique de $r$ dans $\mathbb{Z}_e^*$, la recherche n'a rien de difficile. Par la suite, Bob n'aura plus qu'à tester la résiduosité quadratique de $r$ pour récupérer le bit envoyé par Alice.

Le débit de ce canal subliminal est pour le moins limité. Toutefois, on peut augmenter le nombre de bits du canal en choisissant la résiduosité quadratique de $k$ par rapport à plusieurs nombres premiers secrets simultanément. Cette amélioration est expliquée plus en détails dans l'article de SIMMONS [4] mais le problème est que plus l'on ajoute de contraintes de résiduosité quadratique à $r$. plus il est difficile de trouver un $k$ qui permet à $r$ de respecter ces contraintes.

## 3.4 Prévention et détection

Pour être certain qu'Alice n'implante pas de canal subliminal dans ses signatures, il faut pouvoir s'assurer qu'Alice utilise bien des clé des sessions aléatoires. Pour cela, SIMMONS a proposé un protocole avec tiers de confiance qui permet de générer une signature DSA de sorte qu'Alice ne puisse pas choisir $k$. Le tiers de confiance va donc certifier la signature d'Alice. Le paradoxe de cette solution est que le tiers de confiance peut alors introduire son propre canal subliminal véhiculé par les signatures d'Alice !

La prévention des canaux subliminaux de DSA est donc particulièrement fastidieuse. De même, le canal (avec divulgation mais chiffré ou sans divulgation mais avec secret partagé) est mathématiquement indétectable. Pire encore, si Ève sait qu'Alice et Bob utilise un canal subliminal, il ne peut

pas le prouver sans le secret partagé par Alice et Bob (la clé privée d'Alice dans le canal avec divulgation ou le nombre premier $e$ dans le canal sans divulgation).

# Chapitre 4

# Notre librairie cryptographique

Pour illustrer notre étude des canaux subliminaux, nous avons choisit de réaliser une application qui permette de générer des pairs de clés et des les utiliser pour signer des fichiers et pour vérifier ces signatures.

Nous avons tout d'abord réalisé une librairie cryptographique générique implantant SHA-1 et DSA.

## 4.1 Méthodologie de développement

Tout d'abord, nous allons présenter rapidement les outils que nous avons utilisés pour nous assister dans notre travail de développement.

### 4.1.1 Java

Java est le langage de programmation inventé par Sun Microsystem avec lequel nous avons réalisé aussi bien notre librairie cryptographique que l'application qui l'utilise. Java est un langage orienté objet qui a la particularité d'être très portable.

Par contre, on peut reprocher aux programmes Java une certaine lenteur d'exécution. Cela peut sembler un gros défaut pour une librairie cryptographique supposé réaliser des calculs lourds mais il s'avère dans la pratique que la puissance des machines actuelles compense largement ce défaut.

De plus, Java a l'avantage d'être fourni avec une vaste librairie d'objets très utiles, en particulier les objets BigInteger qui permettent de gérer les

grands entiers dont nous avons besoin en cryptographie. Cette librairie a de surcroît l'avantage d'être particulièrement bien documentée[1].

### 4.1.2 Jar

Jar est une application de compression de fichier compatible avec le format Zip mais qui offre en plus l'avantage que Java est capable d'exécuter un programme directement dans une archive Jar. Ainsi, Jar peut servir en quelque sorte de format de déploiement pour des petites applications comme la notre.

Jar est fourni en standard avec Java.

### 4.1.3 Javadoc

L'outil Javadoc permet de convertir les commentaires en HTML que nous placés dans notre code en un site Internet contenant une documentation complète et facilement navigable de notre librairie.

Cela permet aux développeurs qui souhaitent profiter de notre travail de rapidement comprendre l'API de notre librairie.

Javadoc est fourni en standard avec Java.

### 4.1.4 LaTeX

LaTeX est le système de création de documents que nous avons utilisé pour créer ce rapport.

### 4.1.5 LaTeX2HTML

LaTeX2HTML est un outil de conversion de documents LaTeX en site Internet.

LaTeX2HTML est fourni en standard avec la majorité des distributions LaTeX.

### 4.1.6 Make

Make est un outil classique dans le mode Unix qui permet de gérer facilement tous les processus de compilation de notre projet (que ce soit la

[1] La documentation officielle de Java est disponible sur le site java.sun.com

compilation de programmes comme de documents).

### 4.1.7 CVS

CVS est un outil de gestion de sources qui nous a permis de répartir efficacement notre travail de programmation et de rédaction de documents.

Les fonctionnalités proposées par CVS sont principalement les suivantes :

- gestion des numéros de version
- gestion de l'archivage des différentes versions
- gestion du journal des changements
- annulation des changements
- fusion des changements (quand nous devions travailler sur le même fichier)

## 4.2 Algorithmes

### 4.2.1 SHA-1

L'algorithme SHA-1 est une fonction de hachage cryptographique standardisé par le NIST [1]. Elle prend en entrée une séquence d'octets de longueur arbitraire et retourne une empreinte de 160 bits.

La fonction SHA-1 est conçu sur un modèle assez proche de celui des fonctions MD mais son espace image est environ quatre milliard de fois plus large ($2^{160}$ empreintes possible contre $2^{128}$) ce qui ne peut pas nuire à la sécurité de l'algorithme.

### 4.2.2 DSA

DSA de son côté a visiblement été conçu avec d'avantage de considérations pratiques à l'esprit. La réalisation de DSA ne pose donc pas de problèmes particuliers. Il suffit de suivre la description de l'algorithme [2] en profitant des méthodes de calcul modulaire et de test de primalité de la classe BigInteger de Java.

# Chapitre 5

# Notre application

Avec notre librairie, nous avons fait un logiciel de signatures numériques DSA. Il existe déjà de nombreuses applications qui permettent de faire des signatures numériques DSA [1]. La particularité de notre application est qu'elle utilise un canal subliminal pour cacher dans les signatures qu'elle génère la clé privée du signataire (cf. 3.2). Ainsi, lors de la vérification de la signature, la clé privée est révélée au vérificateur.

Ce comportement n'est probablement pas souhaité par le signataire mais il permet d'illustrer comment un programmeur malicieux pourrait utiliser les canaux subliminaux pour créer une trappe secrète dans son logiciel sans que cela ne puisse être détecté. En effet, les signatures générées restent parfaitement valides selon le standard DSA.

## 5.1 Application principale

Au lancement de notre application, trois choix sont proposés à l'utilisateur :

- la génération de clés DSA
- la signature de fichiers
- la vérification de signatures

Ces trois choix sont proposés au travers de la fenêtre représentées par la figure 5.1. L'utilisateur choisit l'action de son choix en cliquant sur le bouton approprié.

[1] On citera en particulier l'application GnuPG.

## 5.2 Génération de clés

La génération de clés DSA suit le procédé normale du standard DSS. Les modifications liées à l'introduction d'un canal subliminal n'interviennent qu'au moment de la signature.

La fenêtre pour la génération de clés est représentée sur la figure 5.2. L'interface permet à l'utilisateur de choisir la taille des clés qu'il souhaite généré par le biais du menu déroulant montré dans la figure 5.2. Ensuite, l'utilisateur doit spécifier les fichiers dans lesquels il souhaite enregistrer la clé publique et la clé privée. Pour cela, il peut directement saisir le chemin du fichier ou cliquer sur le bouton sélectionner correspondant pour naviguer sur son disque dur avec une interface du type de celle représentée par la figure 5.2.

Enfin, une fois tous les champs renseignés, l'utilisateur peut cliquer sur OK pour lancer la génération de la pair de clés DSA. Une fois l'opération terminée, un résumé est affiché sous la forme de la fenêtre représentée à la figure 5.2.

La génération des clés est la partie la plus lente de l'application. En effet, la génération d'une pair de clés DSA avec notre application prend entre 3 secondes et 2 minutes (selon la chance que l'on a lors de la recherche de grands nombres premiers) avec un processeur Intel Pentium 4 à 2 GHz et 512 Mo de mémoire vive.

## 5.3 Signature de fichiers

Notre application se contente de produire des signatures parfaitement correctes selon le standard DSS si ce n'est que le nombre aléatoire $k$ utilisé est toujours choisit égal au $x$ de la clé privée du signataire. La clé privée est de toute façon nécessaire pour la signature donc l'utilisateur est obligé de fournir $x$ à notre application au moment de la signature.

La fenêtre pour la signature de fichiers est représentée sur la figure 5.3. L'interface permet à l'utilisateur de spécifier le fichier contenant la clé privée à utiliser pour la signature, le fichier à signer et le fichier où enregistrer la signature.

Enfin, une fois tous les champs renseignés, l'utilisateur peut cliquer sur OK pour lancer la signature du fichier. Une fois l'opération terminée, un résumé est affiché sous la forme de la fenêtre représentée à la figure 5.3.

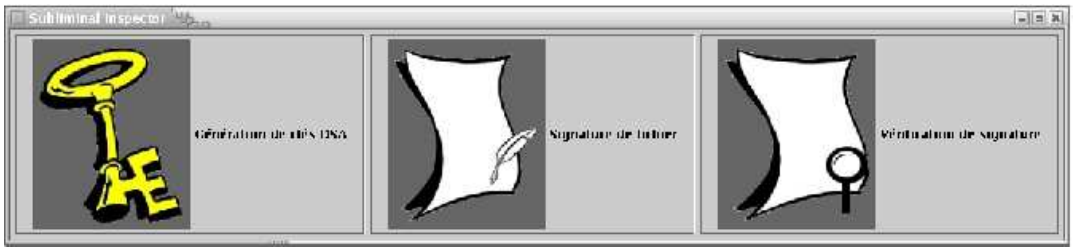

FIG. 5.1 – Fenêtre principale de l'application.


FIG. 5.2 – Génération de clés DSA.

## 5.4 Vérification de signatures

La vérification de la signature se passe tout à fait normalement.

Par contre, on procède ensuite à l'extraction du message subliminal, le $x$ de la clé privée du signataire.

Pour cela, on se rappelle que lors de la signature, notre application utilise comme paramètre aléatoire $k = x$. Les valeurs $r$ et $s$ de la signature sont donc :

$$r = (g^k \bmod p) \bmod q = (g^x \bmod p) \bmod q = y \bmod q$$

$$s = k^{-1}(h + xr) \bmod q = x^{-1}(h + xy) \bmod q = x^{-1}h + y \bmod q$$

On tire donc de la seconde équation que :

$$x = h(s - y)^{-1} \bmod q$$

Comme $h$, $s$ et $y$ sont publiques, on peut donc sans difficulté calculer $x$. On note une particularité intéressante de notre méthode : aucun partage de secret n'est nécessaire entre le signataire et le vérificateur pour que le vérificateur apprenne la clé secrète du signataire. La seule convention est que $x$ à été utilisé comme valeur de $k$. Ainsi, notre méthode permettrait par exemple à un programmeur malicieux de dissimuler une brèche dans un logiciel de signature numérique qui lui permettrait d'obtenir la clé privée de tous ces utilisateurs sans leur consentement et sans que les vérificateurs honnêtes des signatures ne remarque aucune anomalie dans les signatures qui leurs sont présentées.

La fenêtre pour la vérification de signatures est représentée sur la figure 5.4. L'interface permet à l'utilisateur de spécifier le fichier contenant la clé publique à utiliser pour la vérification, le fichier signé et le fichier où est enregistrée la signature.

Enfin, une fois tous les champs renseignés, l'utilisateur peut cliquer sur OK pour lancer la signature du fichier. Une fois l'opération terminée, un résumé est affiché sous la forme de la fenêtre représentée à la figure 5.4.


FIG. 5.3 – Liste de tailles de clés.

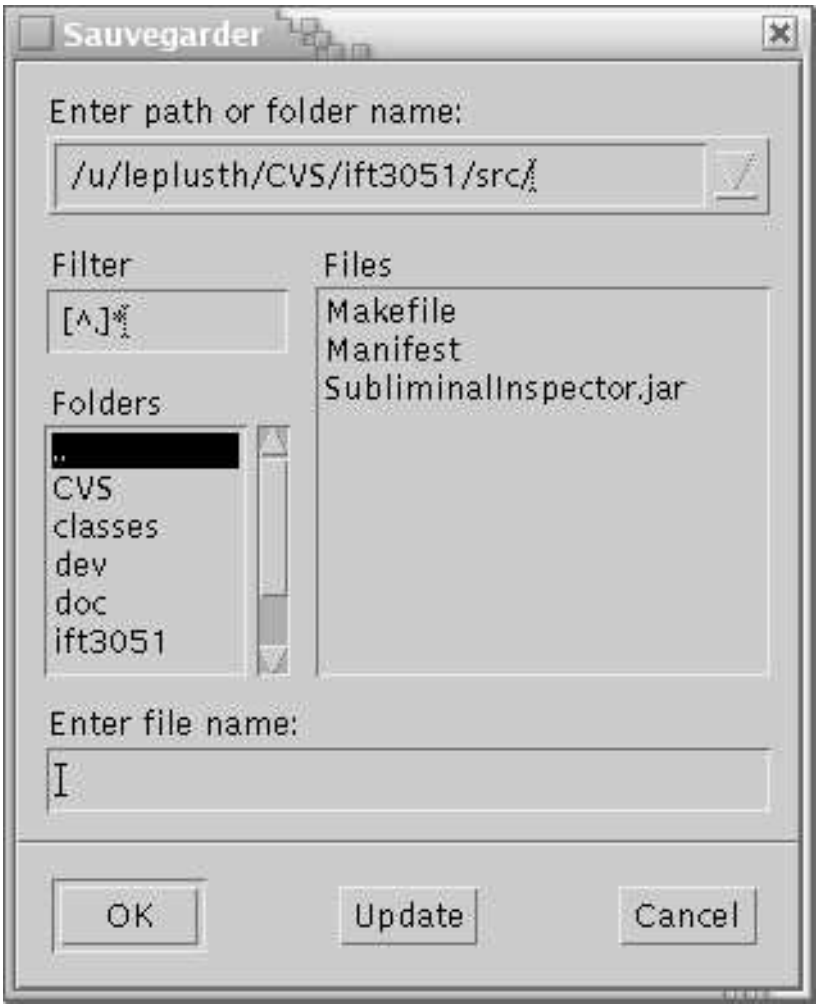


FIG. 5.4 – Sélection de fichier.

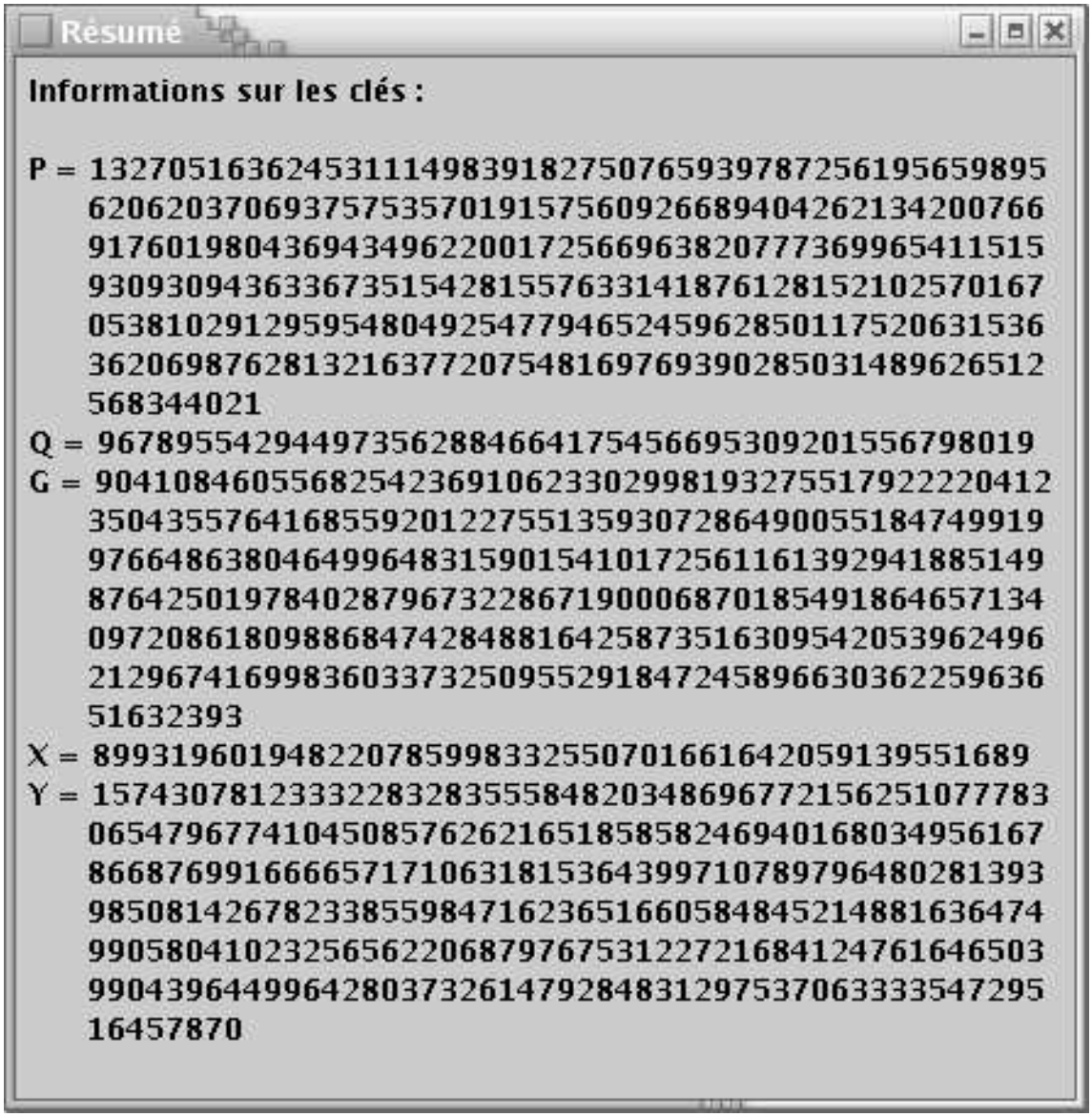


FIG. 5.5 – Résumé d'une génération de clé.

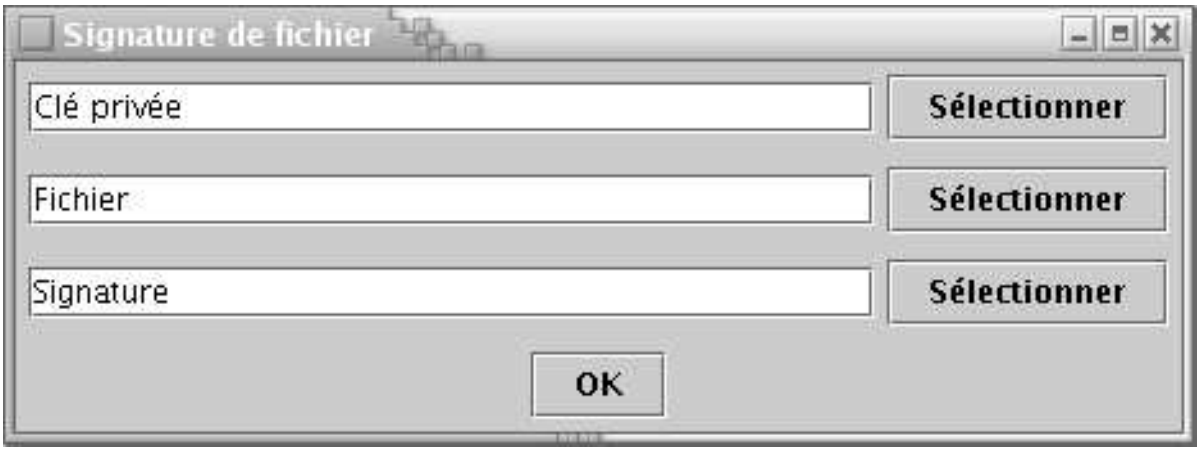


FIG. 5.6 – Signature de fichier.

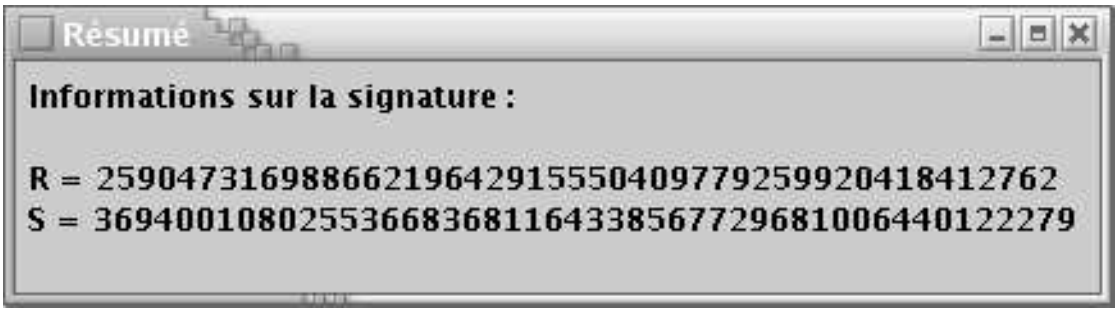


FIG. 5.7 – Résumé d'une signature de fichier.

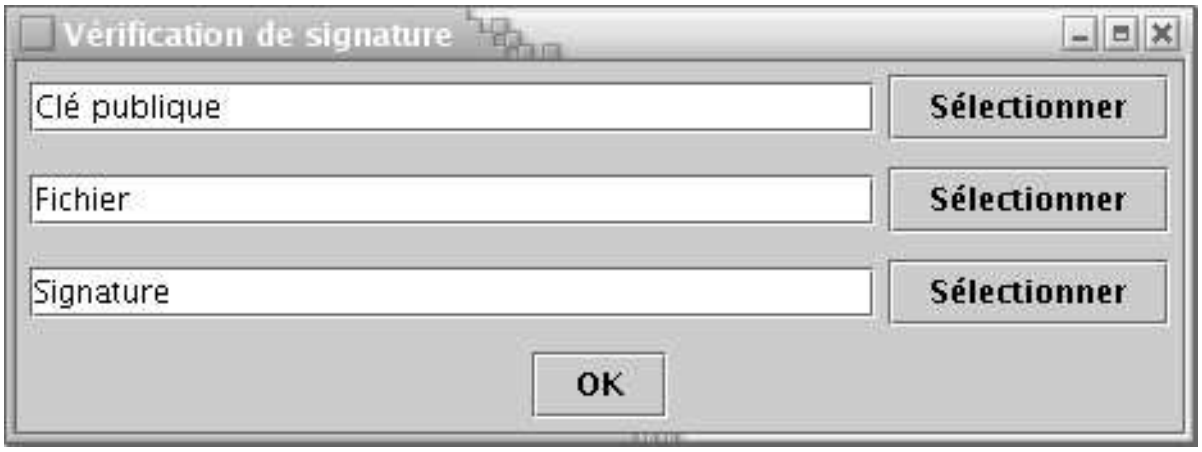


FIG. 5.8 – Vérification de signature.

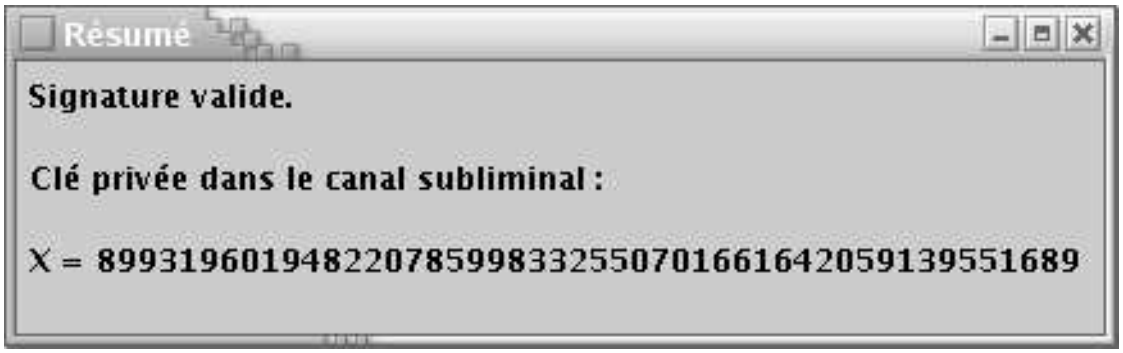


FIG. 5.9 – Résumé de vérification d'une signature (et de son canal).

# Chapitre 6

# Conclusion

Les canaux subliminaux tels que SIMMONS les a imaginés sont de puissants moyens de communications entre deux personnes disposant d'un canal authentifié mais non confidentiel. Toutefois, nous avons vu que les canaux subliminaux peuvent également être utilisé à l'insu des participants et au profit d'un adversaire au courant de l'existence du canal.

Notre étude des canaux subliminaux nous a donc appris que la cryptographie peut parfois être utilisée pour nuire à ses utilisateurs. On en conclut un des principes fondamentaux de la sécurité informatique : il vaut mieux utiliser un logiciel libre dont le code source est constamment audité par de nombreux volontaires que de faire une confiance aveugle dans des solutions propriétaires qui nous mettent à la merci de l'honnêteté de leurs concepteurs.

# Chapitre 7

# Références

Pour notre étude des algorithmes présentés dans ce rapport, nous nous sommes principalement reporté à l'ouvrage de référence *Applied Cryptography* de Bruce Shneier.

Pour résoudre les problèmes d'implantation, nous nous sommes principalement reporté à l'ouvrage de référence *Handbook of Applied Cryptography* de Alfred J. Menezes, Paul C. Van Oorschot, Scott A. Vanstone.

Les autres articles auxquels nous faisons référence dans notre rapport sont listés sur la page de bibliographie suivante.

# Bibliographie


[1] National Institute of Standards and Technology. *FIPS PUB 180-1 : Secure Hash Standard (SHS)*. American National Standards Institute, 1995.

[2] National Institute of Standards and Technology. *FIPS PUB 186-2 : Digital Signature Standard (DSS)*. American National Standards Institute, 2000.

[3] Bruce Schneier. *Applied Cryptography : Protocols, Algorithms, and Source Code in C*. John Wiley and Sons, 2nd edition, 1995.

[4] G. J. Simmons. Subliminal communication is easy using the DSA. In Tor Helleseth, editor, *Advances in Cryptology - EuroCrypt '93*, pages 218–232, Berlin, 1993. Springer-Verlag. Lecture Notes in Computer Science Volume 765.

[5] G. J. Simmons. The history of subliminal channels. In *Procedings of the 1st International Workshop on Information Hiding*, pages 237–256, Cambridge, U.K., 1996. Isaac Newton Institute, University of Cambridge.