Automatisation d'algorithmes de chiffrement

Pigassou Mathis*
UFR Informatique, Université Toulouse Capitole, France
Email: mathis.pigassou@ut-capitole.fr

***Abstract*—Comment cacher efficacement une information à n'importe quel gouvernement ?**
***Index Terms*—Chiffrement, OpenSSL, KeePassXC**

INTRODUCTION

La sécurité des processus de chiffrement présentés ici repose principalement sur deux facteurs. Le premier est l'inviolabilité des algorithmes qui les composent, par des ordinateurs non quantiques. Le second est la répétition du chiffrement avec, pour chaque itération, une clé de chiffrement différente.

I. MÉTHODOLOGIE

Par processus de chiffrement, on entend un script Shell associé à un (par exemple AES) ou plusieurs algorithmes de chiffrement (par exemple Lucie). L'automatisation porte sur ces processus.

A. Fonctionnement

Après avoir reçu le fichier à chiffrer, le script crée un répertoire dans une base de données KeePassXC (.kdbx) qui contiendra les différentes clés générées lors du chiffrement. Le déchiffrement accèdera à ces clés pour le processus inverse, puis supprimera le dossier. Les clés sont générées à l'aide du générateur de KeePassXC avec environ 13 000 bits d'entropie.

B. Stockage

L'accès à cette base de données doit être restreint et protégé de manière multidimensionnelle : d'abord avec un mot de passe maître, un fichier clé stocké sur une clé USB, et une YubiKey.

II. INSTALLATION

A. Dépendances

```
brew install KeePassXC; brew install openssl@3
```

III. CONFIGURATION

Un répertoire contient deux scripts : l'un pour le chiffrement (AES.sh) et l'autre pour le déchiffrement (AESR.sh).

A. Symétrique

Chaque script doit pouvoir accéder aux variables :

- 'KEYFILEPATH' : chemin d'accès vers la base de données,
- 'YUBIKEY' : numéro du slot configuré et son identifiant (par exemple 2:13528435)
- 'DBPATH' : chemin d'accès à la base de données.

Ces variables sont modifiables dans la partie **Set Up** du script.

B. Asymétrique

Le chiffrement nécessite uniquement la clé publique de la personne à qui transmettre l'information. La variable 'PUBLICKEY' doit contenir le chemin d'accès vers cette clé. Inversement, 'PRIVATEKEY' contiendra le chemin d'accès vers la clé privée du récepteur.

Algorithm 1 Génération de clés RSA

```
openssl genpkey -algorithm RSA -out
↪  ~/desktop/ClePrive.pem -pkeyopt
↪  rsa_keygen_bits:4096
openssl rsa -pubout -in
↪  ~/desktop/ClePrive.pem -out
↪  ~/desktop/ClePublique.pem
```

Algorithm 2 Génération de clés Diffie-Hellman

```
openssl ecparam -name prime256v1 -genkey
↪  -noout -out ~/desktop/ecdh_private.pem
openssl ec -in ~/desktop/ecdh_private.pem
↪  -pubout -out ~/desktop/ecdh_public.pem
```

IV. USAGE

Chaque script doit être rendu exécutable avec la commande suivante :

```
chmod u+x <script_path.sh>
```

Certains algorithmes comme **FRANCK** ne peuvent pas être utilisés en temps raisonnable sur de gros fichiers (+10 Mo). Cependant, ils peuvent être utiles pour chiffrer une clé privée.

CONCLUSION

La sécurité parfaite n'existe pas. Même si ces scripts offrent un niveau de complexité suffisant, les algorithmes implémentés par OpenSSL comme RSA ne résistent pas à l'algorithme de Shor.