ICR - Labo #2 : *Conception et implémentation d'un container sécurisé pour des données médicales*

G.Burri

26 novembre 2014

1 Introduction

2 Choix des algorithmes et des paramètres

- *RSA-2048* pour la signature ainsi que pour le chiffrage des clefs *AES* et *HMAC*. Le padding *PKCS#1 v1.5* est utilisé ;
- *HMAC-SHA256* pour la vérification de l'intégrité ;
- *AES-CBC256* pour le chiffrement symétrique du contenu du fichier et des méta-données. Le padding *PKCS7* est utilisé.

3 format du container

Le format est définit comme suit en *EBNF*. Les valeurs entre crochets correspondent soit à une taille en bits soit à un type.

```
container = header, ciphertext ;
header = mac[256], signature[2048], keys[2048] ;
ciphertext = AES(plaintext) ;
plaintext = meta-data, file-content ;
meta-data = meta-data-size[int32], { key-value-pair } ;
key-value-pair = key[string], value[string] ;
string = size[8], content-utf8 ;
```

`meta-data-size` permet de connaître la taille des méta-données afin de les déchiffrer au préalable du contenu du fichier.

`keys` correspond aux clefs k_c et k_a ainsi qu'a l'IV le tout chiffré avec *RSA-2048*. La taille des données chiffrées est égale à $k_c + k_a + iv = 256 + 256 + 128 = 640\,bits$.

Les méta-données (`meta-data`) peuvent contenir, par exemple, le nom du fichier, sa date de création, ses droits, ou tout autres données associées.

4 processus

4.1 chiffrement

Entrées :

— f : contenu du fichier
— $metas$: métas données associées au fichier
— k_{pub} : clef publique RSA
— $k_{signpriv}$: clef privé de signature DSA
Processus :

1. Génération d'une clef 256 bits pour $AES \rightarrow k_c$.
2. Génération d'une clef 256 bits pour $MAC \rightarrow k_a$.
3. Génération d'un IV 128 bits pour le mode $CBC \rightarrow iv$.
4. Construction du *plaintext*, voir format ci dessus.
5. Chiffrement du *plaintext* avec *AES-CBC256*, k_c et $iv \rightarrow ciphertext$.
6. Calcul de MAC de $ciphertext \rightarrow mac$.
7. Signature de mac avec $k_{signpriv} \rightarrow sig$.
8. Chiffrement de $k_c + k_a + iv$ avec $k_pub \rightarrow keys$.
9. Renvoie $mac + sig + keys + ciphertext$.

Où + dénote la concaténation.

4.2 déchiffrement

5 Implémentation

5.1 Utilisation

5.2 Organisation du code

6 Niveaux de sécurité

6.1 Quel est le niveau de sécurité que l'on souhaite atteindre ?

— Confidentialité : les données chiffrées ne doivent pas pouvoir être décryptées par un attaquant.
— Authentification : un attaquant ne doit pas pouvoir forger un container, une signature est réalisée à l'aide d'une paire de clef publique-privée.
— Intégrité : il ne faut pas que les données chiffrées aient pu être altérées par un attaquant.

6.2 Comment s'assure-t-on que les données sont stockées de manière confidentielle ? En particulier ce qui concerne les méta-données

Les méta-données ainsi que les données sont chiffrées ensemble. Voir le format du container décrit précédemment.

6.3 Comment s'assure-t-on que les données stockées sont authentiques ? Quels sont les risques à prendre en compte ?

L'empreinte des données est signée à l'aide d'une clef privée donnée en paramètre de l'*API*, ceci représente la signature qui est placée dans le container. Lors du déchiffrement, la clef publique correspondante est donnée puis utilisée pour déchiffrer l'empreinte qui est comparée à l'empreinte des données.

6.4 Comment s'assure-t-on que les données stockées sont intègres ?

Cela est réalisé avec un *MAC*, dans notre nous utilisons *HMAC-SHA256* sur les données chiffrées (*Encrypt-then-MAC*).

6.5 Quels sont les clefs cryptographiques requises qu'il est nécessaire de gérer ?

6.5.1 Clefs externes

Concerne les clefs externes à l'*API*.
— Une paire de clefs *RSA-2048* pour la signature.
— Une paire de clefs *RSA-2048* pour le chiffrement des clefs *AES*.

6.5.2 Clefs internes

Concerne les clefs gérer à l'intérieur du container.
— Une clef de 256 bits pour *AES*.
— Une clef de 256 bits pour *HMAC*.

7 Conclusion