# Exporter des secrets non exportables d'un jeton cryptographique

Yann Tourdot
yann@tourdot.fr

14 Juillet 2012

**Résumé**

Un jeton cryptographique (carte à puce, HSM...) permet de réaliser des opérations cryptographiques (signature électronique, chiffrement...) tout en protégeant les secrets (clés privées, clés secrètes...) nécessaires à la réalisation de ces opérations, de sorte qu'en aucun cas ces secrets ne puissent être exportés du jeton cryptographique.

Dans cet article, nous présentons une attaque qui consiste à détourner une fonction d'authentification de messages offerte par un jeton cryptographique en fonction de chiffrement symétrique dans le but d'exporter instantanément du jeton cryptographique les deux clés secrètes $k_1$ et $k_2$ mises en oeuvre dans l'algorithme d'authentification de messages CMAC [1] et permettre ainsi à un attaquant de calculer des forges sélectives. L'attaque présentée dans cet article s'applique également aux jetons cryptographiques implémentant les API PKCS#11 [2, 3].

CMAC (Cipher-based MAC) est un algorithme d'authentification de messages permettant d'assurer l'authenticité (et donc l'intégrité) de messages de taille arbitraire. CMAC a été conçu pour pallier la restriction de sécurité d'un autre algorithme d'authentification de messages, CBC-MAC [4], qui, pour des raisons de sécurité, ne doit être utilisé que pour calculer les codes d'authentification de messages de taille fixe.

Dans cet article, nous supposons que le lecteur connait les algorithmes d'authentification de messages CMAC et CBC-MAC.

**Mots-clé :** CMAC, sous-clés, authentification de messages, usage de clé, jeton cryptographique

## 1 Symboles

| | |
|---|---|
| $0^b$ | La suite binaire de $b$ bits à "0". |
| $b$ | La taille en bits des blocs de la primitive de chiffrement $CIPH$. $b \in N^*$. |

| | |
|---|---|
| CBC | Le mode opératoire Cipher Block Chaining défini dans [5]. |
| CBC-MAC | L'algorithme d'authentification de messages Cipher Block Chaining Message Authentication Code défini dans [4]. |
| $CIPH$ | Une primitive de chiffrement symétrique par bloc de taille $b$ bits quelquonque (AES, DES...). |
| $CIPH_{k,mode,iv}(m)$ | Le chiffrement symétrique du message $m$ avec la primitive de chiffrement $CIPH$, la clé secrète $k$, le mode opératoire $mode$ (ECB ou CBC) et le vecteur d'initialiation $iv$[1]. |
| CMAC | L'algorithme d'authentification de messages Cipher-based Message Authentication Code défini dans [1]. |
| CMAC_PADDING | L'algorithme de bourrage pour CMAC défini au chapitre 6.2 intitulé "MAC Generation" dans [1]. |
| ECB | Le mode opératoire Electronic CodeBook défini dans [5]. |
| $f_{mac}$ | Une fonction d'authentification de messages offerte par un jeton cryptographique et définie au chapitre 2 de cet article. |
| $ID\_CBC\_MAC$ | L'identifiant de l'algorithme CBC-MAC dans la liste $LIST\_MAC\_ALGO$. |
| $ID\_CMAC$ | L'identifiant de l'algorithme CMAC dans la liste $LIST\_MAC\_ALGO$. |
| $ID\_CIPH$ | L'identifiant de la primitive de chiffrement par bloc $CIPH$ dans la liste $LIST\_CIPH\_ALGO$. |
| $ID\_CMAC\_PADDING$ | L'identifiant de l'algorithme de bourrage CMAC_PADDING dans la liste $LIST\_PADDING\_ALGO$. |
| $ID\_K$ | L'identifiant de la clé secrète $k$. |
| $ID\_ZERO\_PADDING$ | L'identifiant de l'algorithme de bourrage ZERO-PADDING dans la liste $LIST\_PADDING\_ALGO$. |

[1] Le vecteur d'initialisation $iv$ n'est précisé que lorsque le mode opératoire CBC est utilisé.

| | |
|---|---|
| $iv$ | Le vecteur d'initialisation. Suite binaire de $b$ bits. $iv = \{0,1\}^b$. |
| $k$ | La clé secrète d'authentification de messages de taille cohérente avec la primitive de chiffrement par bloc $CIPH$. |
| $k_1$ | La sous-clé secrète CMAC n°1 de taille $b$ bits et dérivée de la clé secrète $k$ selon l'algorithme défini au chapitre 6.1 de [1]. |
| $k_2$ | La sous-clé secrète CMAC n°2 de taille $b$ bits et dérivée de la clé secrète $k$ selon l'algorithme défini au chapitre 6.1 de [1]. |
| $LIST\_CIPH\_ALGO$ | La liste contenant les identifiants des primitives de chiffrement par bloc supportées par la fonction $f_{mac}$. |
| $LIST\_MAC\_ALGO$ | La liste contenant les identifiants des algorithmes d'authentification de messages supportés par la fonction $f_{mac}$. |
| $LIST\_PADDING\_ALGO$ | La liste contenant les identifiants des algorithmes de bourrage supportés par la fonction $f_{mac}$. |
| $LSB_s(m)$ | Le message constitué des $s$ bits de poids faibles du message $m$. $1 \leq s \leq m_{len}$. |
| $m$ | Le message (suite binaire) à protéger en authenticité. $m = \{0,1\}^{m_{len}}$. $m = m_1 \vert m_2 \vert ... \vert m_n$. |
| $m_i$ | Le $i$ ème bloc du message $m$. $1 \leq i \leq n$. Tous les $m_i$ sont des blocs complets[2], sauf $m_n$ qui peut être un bloc complet ou partiel[3]. |
| $m_{len}$ | La longueur en bits du message $m$. $m_{len} \in \mathbb{N}$. |
| $m_n$ | Le dernier bloc, potentiellement partiel, du message $m$. |
| $m \vert m'$ | La concaténation des deux messages $m$ et $m'$. |
| $m \oplus m'$ | Le OU exclusif bit à bit des deux messages $m$ et $m'$. |
| $n$ | Le nombre de blocs dans le message $m$. $n = \lceil \frac{m_{len}}{b} \rceil$. |

[2]Un bloc est dit complet si sa taille est d'exactement $b$ bits.
[3]Un bloc est dit partiel si sa taille est strictement inférieure à $b$ bits.

| | |
|---|---|
| $PADDING(m)$ | Le bourrage du message $m$ avec l'algorithme de bourrage $PADDING$. $PADDING$ est un algorithme de bourrage dont l'identificant est dans la liste $LIST\_PADDING\_ALGO$. |
| $t$ | Le code d'authentification d'un message $m$ ($t$ pour "tag"). |
| ZERO-PADDING | L'algorithme de bourrage ZERO-PADDING également défini comme "méthode de bourrage n°1" dans [6]. |
| $\lceil x \rceil$ | Le plus petit entier plus grand ou égal au nombre réel $x$. |
| $\oslash$ | Indique un paramètre d'entrée omis pour la fonction $f_{mac}$. |

## 2 $f_{mac}$, une simple fonction d'authentification de messages

Pour illustrer l'attaque présentée dans cet article, nous prenons l'exemple d'une fonction d'authentification de messages, notée $f_{mac}$, offerte par un jeton cryptographique, et dont le prototype est le suivant :

$$output_{tag} = f_{mac}(input_{ciph\_algo}, input_{mac\_algo}, input_{padding\_algo}, input_{key\_id}, input_{message})$$

avec :

| **Nom du paramètre** | **Description du paramètre** | **Entrée / sortie** | **Obligatoire / optionnel** |
|---|---|---|---|
| $output_{tag}$ | Code d'authentification du message $input_{message}$. | Sortie. | Obligatoire. |
| $input_{ciph\_algo}$ | Identifiant de primitive de chiffrement par bloc choisi dans la liste $LIST\_CIPH\_ALGO$. | Entrée. | Obligatoire. |
| $input_{mac\_algo}$ | Identifiant d'algorithme d'authentification de messages choisi dans la liste $LIST\_MAC\_ALGO$. | Entrée. | Obligatoire. |

| | | | |
|---|---|---|---|
| $input_{padding\_algo}$ | Identifiant d'algorithme de bourrage choisi dans la liste $LIST\_PADDING\_ALGO$. | Entrée. | Optionnel.[4] |
| $input_{key\_id}$ | Identifiant[5] de la clé secrète utilisée pour l'authentification de messages. | Entrée. | Obligatoire. |
| $input_{message}$ | Message de taille arbitraire dont on veut calculer le code d'authentification. | Entrée. | Obligatoire. |

# 3 Profils des attaquants

On considère les deux profils d'attaquants suivants :

Profil d'attaquant 1 :

- l'attaquant a accès à la fonction $f_{mac}$[6], c'est-à-dire qu'il peut choisir les paramètres d'entrée (*input*) de la fonction $f_{mac}$ et a accès à la sortie (*output*) de la fonction $f_{mac}$.

- l'attaquant dispose d'une liste $CODEBOOK\_1$[7] contenant un ou plusieurs couples $(input_{message}, output_{tag})$.

- l'attaquant ne connait pas la clé secrète $k$.

Profil d'attaquant 2 :

- l'attaquant n'a pas accès à la fonction $f_{mac}$.

- l'attaquant dispose d'une liste $CODEBOOK\_2$[8] contenant un ou plusieurs couples $(input_{message}, output_{tag})$.

- l'attaquant ne connait pas la clé secrète $k$.

[4] Dans certains cas il n'est pas nécessaire de préciser l'algorithme de bourrage car le bourrage est pris en charge nativement par l'algorithme d'authentification de messages. C'est par exemple le cas de l'algorithme d'authentification de messages CMAC qui prend en charge nativement l'algorithme de bourrage CMAC-PADDING (voir Rappel 4 au chapitre 4.4 de cet article).

[5] L'identifiant de la clé secrète et non la clé secrète elle-même est donnée en paramètre d'entrée car cela supposerait que les utilisateurs de la fonction $f_{mac}$ (profil d'attaquant 1 par exemple) ont accès à cette clé secrète.

[6] Ce profil d'attaquant doit être envisagé car bien qu'ayant accès à la fonction $f_{mac}$, il ne doit pas être capable d'extraire des secrets du jeton cryptographique.

[7] Le profil d'attaquant 1 peut choisir les couples $(output_{tag}, input_{message})$ de la liste $CODEBOOK\_1$ car il a accès à la fonction $f_{mac}$.

[8] Le profil d'attaquant 2 ne peut pas choisir les couples $(output_{tag}, input_{message})$ de la liste $CODEBOOK\_2$ car il n'a pas accès à la fonction $f_{mac}$.

# 4 Rappels

Avant de présenter l'attaque, il est nécessaire de faire quelques rappels.

## 4.1 Rappel 1 : Pourquoi CBC-MAC ne doit être utilisé que pour authentifier des messages de taille fixe ?

*Objectif: Monter que si une même clé secrète est utilisée pour calculer les codes d'authentification de messages de taille variables, alors le profil d'attaquant 2*[9] *est capable de cacluler des forges sélectives.*

On suppose que la même clé secrète $k$ est utilisée pour calculer les codes d'authentification des messages $m$ et $m'$. On suppose également que l'algorithme d'authentification de messages est CBC-MAC et que la primitive de chiffrement est $CIPH$. On a ainsi :

$$\begin{cases} t = f_{mac}(ID\_CIPH, ID\_CBC\_MAC, ID\_PADDING\_1, ID\_K, m) \\ t' = f_{mac}(ID\_CIPH, ID\_CBC\_MAC, ID\_PADDING\_2, ID\_K, m') \end{cases}$$ [10]

On suppose que les couples $(m, t)$ et $(m', t')$ sont dans la liste $CODEBOOK\_2$ à disposition du profil d'attaquant 2.

Le profil d'attaquant 2 caclule alors le message suivant :

$$m'' = m_1|m_2|...|PADDING\_1(m_n)|m'_1 \oplus t|m'_2|...|PADDING\_2(m'_{n'})$$

Or, comme le montre la Figure 1, on a :

$$\begin{cases} t' = f_{mac}(ID\_CIPH, ID\_CBC\_MAC, ID\_PADDING\_1, ID\_K, m') \\ t' = f_{mac}(ID\_CIPH, ID\_CBC\_MAC, ID\_PADDING\_2, ID\_K, m'') \end{cases}$$

Le profil d'attaquant 2 est donc capable de calculer des forges sélectives, c'est-à-dire calculer, sans avoir accès à la fonction $f_{mac}$, le code d'authentification des messages du type de $m''$.

Dans cet exemple, le profil d'attaquant 2 calcule le code d'authentification $t'$ d'un message $m''$ construit à partir de deux messages $m$ et $m'$, mais le message $m''$ pourrait être construit à partir d'une infinité d'arrangements des messages $m$ et $m'$.

[9] *Le profil d'attaquant 1 est omis. Sa capacité à calculer des forges sélectives ne présente aucun intérêt car il a accès à la fonction d'authentification de messages $f_{mac}$.*

[10] $ID\_PADDING\_1$ et $ID\_PADDING\_2$ sont les identifiants de deux algorithmes de bourrage quelquonques. Ils peuvent être identiques ou différents, cela n'a aucune importance.

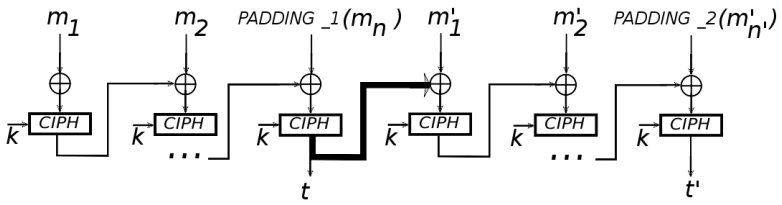


Figure 1 : Calcul de forges sélectives CBC-MAC

## 4.2 Rappel 2 : Pourquoi $k_1$, $k_2$ doivent rester secrètes dans CMAC ?

*Objectif: Monter que si les sous-clés secrètes $k_1$ et $k_2$ sont connues du profil d'attaquant 2[8], alors ce dernier est capable de calculer des forges sélectives.*

L'algorithme d'authentification de messages CMAC est très proche de l'algorithme d'authentification de messages CBC-MAC. La seule différence entre CBC-MAC et CMAC est que CMAC met en oeuvre deux sous-clés secrètes, notées $k_1$ et $k_2$, qui n'interviennent que dans le dernier tour.

Les deux sous-clés secrètes $k_1$ et $k_2$ sont dérivées de la clé secrète $k$ selon l'algorithme décrit dans le chapitre 6.1 de [1] intitulé "Subkey Generation", ont chacune une taille de $b$ bits, et doivent rester secrètes sinon il est possible pour le profil d'attaquant 2 de calculer des forges sélectives.

Connaissant uniquement la clé secrète $k$, il est très facile de calculer $k_1$ et $k_2$, alors que l'inverse n'est pas vrai. Le choix de l'utilisation de la sous-clé secrète $k_1$ ou de la sous-clé secrète $k_2$ est uniquement fonction du message dont on veut calculer le code d'authentification, plus précisément cela dépend si sa taille en bits est multiple de $b$ (dans ce cas, utilisation de $k_1$) ou non (dans ce cas, utilisation de $k_2$).

On suppose que le profil d'attaquant 2 connaît les deux sous-clés secrètes $k_1$, $k_2$.

On suppose que la même clé secrète $k$ est utilisée pour calculer les codes d'authentification des messages $m$ et $m'$. On suppose également que l'algorithme d'authentification de messages est CMAC et que la primitive de chiffrement est $CIPH$. On a ainsi :

$$\begin{cases} t = f_{mac}(ID\_CIPH, ID\_CMAC, \oslash, ID\_K, m) \\ t' = f_{mac}(ID\_CIPH, ID\_CMAC, \oslash, ID\_K, m') \end{cases}$$ [11]

On suppose que les couples $(m, t)$ et $(m', t')$ sont dans la liste $CODEBOOK\_2$ à disposition du profil d'attaquant 2.

Le profil d'attaquant 2 caclule alors le message suivant :

[11] Le paramètre $input_{padding\_algo}$ identifiant l'algorithme de bourrage vaut $\oslash$, autrement dit, il est omis, car comme l'indique la note de bas de page 3, l'algorithme CMAC prend en charge le bourrage du message dont on veut calculer le code d'authentification selon l'algorithme CMAC_PADDING.

$$m'' = m_1|m_2|...|CMAC\_PADDING(m_n) \oplus k_1|m'_1 \oplus t|m'_2|...|CMAC\_PADDING(m'_{n'})$$

Or, comme le montre la Figure 2, on a :

$$\begin{cases} t' = f_{mac}(ID\_CIPH, ID\_CMAC, ID\_CMAC\_PADDING, ID\_K, m') \\ t' = f_{mac}(ID\_CIPH, ID\_CMAC, ID\_CMAC\_PADDING, ID\_K, m'') \end{cases}$$

Le profil d'attaquant 2 est donc capable de calculer des forges sélectives, c'est-à-dire calculer, sans avoir accès à la fonction $f_{mac}$, les codes d'authentification des messages du type de $m''$.

Dans cet exemple, le profil d'attaquant 2 calcule le code d'authentification $t'$ d'un message $m''$ construit à partir de deux messages $m$ et $m'$, mais le message $m''$ pourrait être construit à partir d'une infinité d'arrangements des messages $m$ et $m'$.

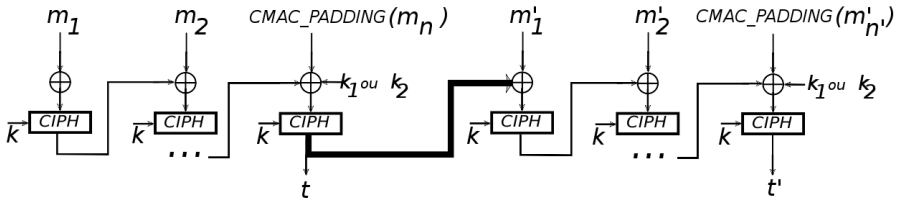


Figure 2 : Calcul de forges sélectives CMAC

## 4.3 Rappel 3 : Pourquoi $CIPH_{k,ECB,\emptyset}(0^b)$ doit rester secret dans CMAC ?

*Objectif: Monter que si $CIPH_{k,ECB,\emptyset}(0^b)$ est connu des profils d'attaquants 1 et 2, alors ces derniers sont capables de calculer les deux sous-clés $k_1$ et $k_2$.*

[1] mentionne dans son chapitre 5.3 intitulé "Subkeys" que "Any intermediate value in the computation of the subkey, in particular, $CIPH_k(0^b)$[12], shall also be secret." En effet, comme le montre le chapitre 6.1 de [1] intitulé "Subkey Generation", les deux sous-clés $k_1$et $k_2$ se calculent uniquement à partir de $CIPH_{k,ECB,\emptyset}(0^b)$.

La connaissance de $CIPH_{k,ECB,\emptyset}(0^b)$ implique donc la connaissance des deux sous-clés secrètes $k_1$et $k_2$. Or comme les deux sous-clés secrètes $k_1$et $k_2$ doivent rester secrètes pour empêcher qu'un attaquant soit capable de calculer des forges sélectives[13], cela implique que $CIPH_{k,ECB,\emptyset}(0^b)$ doit donc lui aussi rester secret.

[12]La notation $CIPH_k(0^b)$ dans [1] correspond à $= CIPH_{k,ECB,\emptyset}(0^b)$ dans cet article.
[13]Voir Rappel 2 au chapitre 4.2 de cet article.

## 4.4 Rappel 4: Le CMAC-PADDING

*Objectif: Présenter l'algorithme de bourrage CMAC-PADDING pris en charge nativement par l'algorithme d'authentification de messages CMAC.*

L'algorithme d'authentification de messages CMAC prend en charge nativement un algorithme de bourrage, le CMAC-PADDING. Le CMAC-PADDING est défini à l'étape 4 de l'algorithme d'authentification de messages CMAC au chapitre 6.2 de [1] intitulé "MAC Generation". Il consiste à n'ajouter aucun bit à la fin du message $m$ à bourrer si ce dernier a une taille en bits $m_{len}$ multiple de $b$ (cas 1) et à ajouter un bit à "1" suivi d'autant de bits à "0" que nécessaires (possiblement aucun) pour obtenir un message bourré ayant une taille en bits multiple de $b$ sinon (cas 2). L'algorithme de bourrage utilisé par CMAC dans le cas 2 correspond à l'algorithme de bourrage identifié comme "méthode de bourrage n°2" dans [6]. On a ainsi:

$$\forall m, CMAC - PADDING(m) = m \text{ si } m_{len} = k \times b, k \in N^* \text{ (cas 1)}$$

$$\forall m, CMAC - PADDING(m) = m|10^j \text{ avec } j = n \times b - m_{len} - 1 \text{ si } \nexists k \in N^* / m_{len} = k \times b \text{ (cas 2)}$$

## 4.5 Rappel 5 : Le ZERO-PADDING

*Objectif: Monter que l'algorithme de bourrage ZERO-PADDING permet d'obtenir dans certains cas un message bourré valant $0^b$.*

Le ZERO-PADDING est l'un des plus simples algorithmes de bourrage. Il correspond à la méthode de bourrage n°1 définie dans la norme [6] et consiste à ajouter, si nécessaire, autant de bits à "0" à la fin d'un message $m$ de taille $m_{len}$ à bourrer pour obtenir un message bourré, noté $ZERO-PADDING(m)$, ayant une taille en bits multiple de $b$.

Ainsi, dans le cas d'un message $m$ ayant une taille en bits multiple de $b$, aucun bit à "0" n'est ajouté à la fin du message $m$. On a ainsi :

$$\forall m, ZERO - PADDING(m) = m \text{ si } m_{len} = k \times b, k \in N^*$$

Dans le cas particulier de $m = 0^b$, on a donc $ZERO-PADDING(0^b) = 0^b$, car $m_{len} = k \times b$, avec $k = 1$.

## 4.6 Rappel 6 : Similitudes du mode opératoire CBC et de l'algorithme d'authentification de messages CBC-MAC

*Objectif: Montrer que le mode opératoire CBC est très proche de l'algorithme d'authentification de message CBC-MAC.*

Cette propriété découle directement de la définition du mode opératoire CBC et de l'algorithme d'authentification de messages CBC-MAC. En effet, le code d'authentification du message $m$ bourré avec l'algorithme de bourrage $PADDING\_1$, $PADDING\_1(m)$, correspond au dernier bloc du chiffré de $PADDING\_1(m)$, lorsque le vecteur d'initialisation $iv$ vaut $0^b$. On a ainsi :

$$\begin{aligned} t &= f_{mac}(ID\_CIPH, ID\_CBC\_MAC, ID\_PADDING\_1, ID\_K, m) \\ &= LSB_b(CIPH_{k,CBC,0^b}(PADDING\_1(m))) \end{aligned}$$

## 4.7 Rappel 7 : Les chiffrements en mode ECB et CBC produisent des chiffrés identifques dans certains cas

*Objectif: Montrer que lorsqu'un message $m$ bourré avec le padding $PADDING\_1$, $PADDING\_1(m)$, a une taille de $b$ bits, son chiffré en mode ECB est identique à son chiffré en mode CBC* lorsque le vecteur d'initialisation $iv$ vaut $0^b$.

Cette propriété découle directement de la définition des modes opératoires ECB et CBC. En effet, lorsque le message $m$ bourré avec l'algorithme de bourrage $PADDING\_1$, $PADDING\_1(m)$, à chiffrer a une taille de $b$ bits, il ne peut y avoir de chainage de blocs lors du chiffrement en mode CBC, car il n'y a qu'un seul bloc. On a ainsi:

$$\forall m, CIPH_{k,ECB\oslash}(PADDING\_1(m)) = CIPH_{k,CBC,0^b}(PADDING\_1(m)) \\ \text{si } (PADDING\_1(m))_{len} = b$$

## 4.8 Rappel 8 : Mécanismes PKCS#11 de protection des clés cryptographiques contre l'exportation

*Objectif: Présenter deux mécanismes PKCS#11 de protection des clés cryptographiques contre l'exportation.*

PKCS#11 est un des standards de la famille PKCS (Public-Key Cryptography Standards) publiés par RSA Laboratories.

PKCS#11 définit les interfaces (API) des jetons cryptographiques dont six relatives à l'authentification de messages[14].

La dernière version de PKCS#11 est la version 2.20 [2] et date de juin 2004. Une version 2.30 [3] de PKCS#11 est disponible en version "draft" depuis avril 2009.

Dans PKCS#11, les clés secrètes sont représentées par la classe d'objet $CKO\_SECRET\_KEY$ qui possède notamment les deux attributs suivants:

$CKA\_EXTRACTABLE$: booléen indiquant si la clé est exportable ($CK\_TRUE$) ou non ($CK\_FALSE$).

[14] Voir Annexe 1 de cet article.

$CKA\_ALLOWED\_MECHANISMS$: liste des mécanismes cryptographiques pouvant être utilisés avec la clé secrète.

# 5 Présentation de l'attaque

## 5.1 Résumé de l'attaque

L'attaque présentée dans cet article consiste à détourner une fonction d'authentification de messages $f_{mac}$ en une fonction de chiffrement symétrique pour, dans un premier temps, obtenir la valeur $CIPH_{k,ECB,\oslash}(0^b)$, puis, dans un second temps, calculer les deux sous-clés secrètes $k_1$ et $k_2$ et finalement calculer des forges sélectives.

## 5.2 Conditions d'exploitation

Pour que l'attaque présentée dans cet article soit réalisable, les deux conditions suivantes sont nécessaires et suffisantes :

Condition 1 : la liste $LIST\_PADDING\_ALGO$ doit au moins contenir $ID\_ZERO\_PADDING$.

Condition 2 : la liste $LIST\_MAC\_ALGO$ doit au moins contenir $ID\_CBC\_MAC$ et $ID\_CMAC$.

## 5.3 Exploitation

Les étapes de réalisation de l'attaque présentée dans cet article, bien que similaires, dépendent du profil d'attaquant.

On distingue donc les deux cas d'exploitation suivants :

Cas 1 : l'attaquant a accès à la fonction $f_{mac}$ (profil d'attaquant 1)

Cas 2 : l'attaquant n'a pas accès à la fonction $f_{mac}$ (profil d'attaquant 2)

### 5.3.1 Cas 1 : l'attaquant a accès à la fonction $f_{mac}$

Si l'attaquant a accès à la fonction $f_{mac}$ (profil d'attaquant 1), alors il obtient les deux sous-clés secrètes $k_1$ et $k_1$ de la manière suivante :

Etape 1 : l'attaquant calcule

$$\begin{aligned} t &= f_{mac}(ID\_CIPH, ID\_CBC\_MAC, \\ &\qquad ID\_ZERO\_PADDING, ID\_K, 0^b) \end{aligned}$$

Or dans ce cas, $t = CIPH_{k,ECB,\oslash}(0^b)$.

*Preuve :*

Le message $0^b$, ayant une taille de $b$ bits, lorsqu'il est bourré avec l'algorithme de bourrage $ZERO - PADDING$, on a[15] :

$$ZERO - PADDING(0^b) = 0^b$$

On a également[16] :

$$\begin{aligned} t &= f_{mac}(ID\_CIPH, ID\_CBC\_MAC, \\ &\quad ID\_ZERO\_PADDING, ID\_K, 0^b) \\ &= LSB_b(CIPH_{k,CBC,0^b}(ZERO\_PADDING(0^b))) \\ &= LSB_b(CIPH_{k,CBC,0^b}(0^b)) \end{aligned}$$

La taille des blocs de la primitive $CIPH$ étant de $b$ bits, on a donc :

$$t = LSB_b(CIPH_{k,CBC,0^b}(0^b)) = \quad CIPH_{k,CBC,0^b}(0^b)$$

Or comme le message $0^b$ a une taille de $b$ bits, on a[17] :

$$t = CIPH_{k,CBC,0^b}(0^b) = \quad CIPH_{k,ECB,\oslash}(0^b)$$

Etape 2 : connaissant $t = CIPH_k(0^b)$, l'attaquant calcule les deux sous-clés secrètes $k_1$ et $k_2$ selon l'algorithme décrit au chapitre 6.1 de [1] intitulé "Subkey Generation".

Etape 3 : l'attaquant connait désormais les deux sous-clés secrètes $k_1$ et $k_2$. Il est capable de calculer des forges sélectives[18] à partir des couples $(input_{message}, output_{tag})$ de la liste $CODEBOOK\_1$ en sa possession.

### 5.3.2 Cas 2 : l'attaquant n'a pas accès à la fonction $f_{mac}$

Si l'attaquant n'a pas accès à la fonction $f_{mac}$ (profil d'attaquant 2) , alors il obtient les deux sous-clés secrètes $k_1$ et $k_1$ de la manière suivante :

Etape 1 : parmi tous les couples $(input_{message}, output_{tag})$ de la liste $CODEBOOK\_2$ en sa possession, l'attaquant cherche le couple pour lequel $input_{message} = 0^b$. Si ce couple n'est pas dans la liste $CODEBOOK\_2$, alors l'attaque échoue, sinon l'attaquant passe à l'étape 2.

[15] Voir Rappel 5 au chapitre 4.5 de cet article.
[16] Voir Rappel 6 au chapitre 4.6 de cet article.
[17] Voir Rappel 7 au chapitre 4.7 de cet articke.
[18] Voir Rappel 2 au chapitre 4.2 de cet article.

Etape 2 : identique à l'étape 2 du cas 1.

Etape 3 : identique à l'étape 3 du cas 1.

# 6 Cas des jetons cryptographiques implémentant PKCS#11

L'attaque présentée dans cet article s'applique également aux jetons cryptographiques implémentant les API PKCS#11 [2, 3]. Il est possible d'exporter d'un jeton cryptographique les deux sous-clés secrètes $k_1$ et $k_2$, et ce, même si la clé secrète $k$ dont sont dérivées les deux sous-clés n'est pas exportable au sens PKCS11[19].

En effet, la fonction d'authentification de messages $f_{mac}$ n'est définie dans cet article qu'à titre d'illustration. La fonction $f_{mac}$ pourrait très bien être une fonction d'authentification de messages de haut-niveau offerte par un jeton cryptographique utilisant elle-même les API PKCS#11 d'authentification de messages[20], l'attaque présentée dans cet article resterait réalisable selon le même mode opératoire.

De même, si au lieu d'avoir accès à la fonction d'authentification de messages $f_{mac}$, le profil d'attaquant 1 a accès directement aux API PKCS#11 d'authentification de messages du jeton cryptographique, l'attaque resterait réalisable, toujours selon le même mode opératoire.

Les spécifications de l'algorithme d'authentification de messages CMAC [1] date de 2005, alors que la dernière version de PKCS#11 (v2.20) [2] date de 2004. CMAC n'a été intégré que dans la nouvelle version de PKCS#11 (v2.30) [3] qui est toujours en version "draft" depuis avril 2009.

Pour empêcher que l'attaque présentée dans cet article ne s'applique aux jetons cryptographiques implémentant les API PKCS#11, ce n'est pas l'attribut $CKM\_EXTRACTABLE$ définissant les règles d'exportation des clés cryptographiques auquel il faut prêter attention, mais l'attribut $CKA\_ALLOWED\_MECHANISMS$ définissant les mécanismes cryptographiques pouvant être utilisés avec les clés cryptographiques. En effet, ce n'est qu'en empêchant l'utilisation d'une clé secrète CMAC avec les algorithmes CBC-MAC et ZERO-PADDING[21] que l'attaque présentée dans cet article devient impossible à réaliser. Comme le mentionne le chapitre 5.2 de [1] intitulé "Block Cipher": "The key shall be secret and shall be used exclusively for the CMAC mode of the chosen block cipher".

[19] En PKCS#11, une clé cryptographique n'est pas exportable si son attribut $CKA\_EXTRACTABLE = CK\_FALSE$ (voir Rappel 8 chapitre 4.8 de cet article).

[20] Voir Annexe 1 de cet article.

[21] Voir chapitre 5.2 de cet article qui indique que les deux seules conditions nécessaires et suffisantes à la réalisation de l'attaque sont la possibilité d'utiliser une clé secrète CMAC avec l'algorithme de bourrage ZERO-PADDING (condition 1) et l'algorithme d'authentification de messages CBC-MAC (condition 2).

# 7 Usage des clés cryptographiques et Référentiel Général de Sécurité

L'Annexe B1 [5] du Référentiel Général de Sécurité (RGS) [6] mentionne, au travers des exigences RègleGestSym-1 et RègleGestAsym-1, qu'une même clé cryptographique, qu'elle soit symétrique ou asymétrique, ne doit être utilisée que pour un usage :

RègleGestSym-1. L'emploi d'une même clé pour plus d'un usage est exclu.

RègleGestAsym-1. L'emploi d'un même bi-clé à plus d'un usage est exclu.

Le problème est que l'Annexe B1 du RGS ne définit précisément ce qu'elle entend par le terme "usage". Tout au plus, à la lecture des trois extraits de l'Annexe B1 du RGS ci-dessous on en déduit que le "chiffrement", l"'authentification" et la "signature électronique" sont trois usages différents.

Extrait 1 : "Les deux principaux risques généraux dans l'emploi de clés secrètes sont, d'une part, l'usage d'une clé pour plusieurs emplois (en confidentialité et intégrité par exemple), et d'autre part l'emploi de clés partagées par un nombre important d'utilisateurs ou d'équipements."

Extrait 2 : "L'emploi d'une même clé à plus d'un usage, par exemple pour chiffrer avec un mécanisme de confidentialité et assurer l'intégrité avec un mécanisme différent, est source de nombreuses erreurs."

Extrait 3 : "L'emploi d'un même bi-clé à plus d'un usage, par exemple pour chiffrer et signer, est une source d'erreurs graves."

Une fonction cryptographique n'offrant qu'un usage[22] (ici usage "authentification de messages") peut parfois être détournée pour offrir, même de manière extrêmement limitée[23], d'autres usages (ici usage "chiffrement symétrique"), et ce uniquement en utilisant certaines valeurs particulières de paramètres cryptographiques.

Or un tel détournement de fonction cryptographique peut avoir de graves conséquences comme le montre l'attaque présentée dans cet article (ici extraction des deux sous-clés secrètes CMAC $k_1$ et $k_2$).

# 8 Conclusion

Cet article montre que, dans certains cas, il est possible d'exporter très facilement[24] des secrets d'un jeton cryptographique (HSM, carte à puce...), et ce,

[22] Au sens de l'Annexe B1 du Référentiel Général de Sécurité.

[23] L'attaque présentée dans cet article ne nécessite de détourner la fonction d'authentification de message $f_{mac}$ en fonction de chiffrement symétrique uniquement pour la valeur $0^b$.

[24] Voire instantanément pour le profil d'attaquant 1.

même si ces derniers ne sont pas censés pouvoir être exportés. L'attaque présentée dans cet article se prête particulièrement aux jetons cryptographiques de type HSM dont l'objectif est de fournir de nombreux services cryptographiques pouvant être appelés avec de multiples paramètres cryptographiques.

Cet article montre qu'il y a potentiellement danger dès lors qu'une fonction cryptographique peut être détournée en une autre fonction cryptographique, et ce, même de manière extrêmement limitée[25]. L'attaque présentée dans cet article est réalisable même si l'attaquant n'interagit pas avec le jeton cryptographique (profil d'attaquant 2), bien que pour des raisons d'efficacité il soit préférable qu'il puisse le faire (profil d'attaquant 1).

De manière particulière, pour que l'attaque présentée dans cet article ne soit pas réalisable, il suffit qu'une clé secrète utilisée pour l'authentification de messages avec l'algorithme CMAC ne puisse pas également être utilisée avec les algorithmes CBC-MAC et ZERO-PADDING. Toutes les clés secrètes ne respectant pas cette règle devraient être considérées comme étant compromises.

De manière générale, une clé cryptographique, qu'elle soit symétrique ou asymétrique, doit toujours être utilisée avec les mêmes paramètres cryptographiques.

Enfin, il serait souhaitable que l'Annexe B1 du RGS, lors de sa prochaine mise à jour, définisse précisément ce qu'elle entend par le terme "usage". Je propose de définir l'"usage", non pas comme une fonction cryptographique de haut niveau (chiffrement, authentification de messages, signature électronique...) comme ça l'est actuellement dans l'Annexe B1 du RGS, mais comme un ensemble de paramètres cryptographiques fixés. Tout ensemble de paramètres cryptographiques fixés doit constituer un usage. Dès qu'au moins un paramètre cryptographique de cet ensemble est modifié, cela doit correspondre à un autre usage et une autre clé cryptographique doit donc être utilisée. C'est à cette seule condition que la règle "Une clé, un usage" prend tout son sens.

[25] L'attaque présentée dans cet article ne nécessite de détourner la fonction d'authentification de message $f_{mac}$ en fonction de chiffrement symétrique uniquement pour la valeur $0^b$.

# 9 Annexes

## 9.1 Annexe 1: API PKCS#11 v2.20 pour l'authentification de messages

| Category | Function | Description |
|---|---|---|
| Signing and MACing functions | C_SignInit | initializes a signature operation |
| | C_Sign | signs single-part data |
| | C_SignUpdate | continues a multiple-part signature operation |
| | C_SignFinal | finishes a multiple-part signature operation |
| | C_SignRecoverInit | initializes a signature operation, where the data can be recovered from the signature |
| | C_SignRecover | signs single-part data, where the data can be recovered from the signature |

# Références


[1] NIST, Recommendation for Block Cipher Modes of Operation : The CMAC Mode for Authentication, Special Publication 800-38B, May 2005.

[2] PKCS #11 v2.20 : Cryptographic Token Interface Standard, RSA Laboratories, 28 June 2004.

[3] PKCS #11 v2.30 : Cryptographic Token Interface Standard, RSA Laboratories, 15 April 2009.

[4] FIPS Publication 81, DES Modes of Operation. U.S. DoC/NIST, December 1980.

[5] FIPS PUB 113, Computer Data Authentication, 30 May 1985.

[6] ISO/IEC 9797-1 :1999 Information technology – Security techniques – Message Authentication Codes (MACs) – Part 1 : Mechanisms using a block cipher.

[7] Annexe B1, Mécanismes cryptographiques Règles et recommandations concernant le choix et le dimensionnement des mécanismes cryptographiques, version 1.20 du 26 janvier 2010.

[8] ANSSI, DGME, Référentiel Général de Sécurité, version 1.0 du 6 mai 2010.