Combinaison de CBC et CBC-MAC

En cours, nous avons vu que la combinaison "encrypt-then-MAC" d'un chiffrement IND-CPA avec un MAC SUF-CMA nous donnait un chiffrement authentifié IND-CCA, et par ailleurs impossible à contrefaire (il est donc impossible pour un adversaire de créer une nouvelle paire "message, tag" valide). Cependant, de mauvaises combinaisons peuvent aboutir à des attaques. Nous donnons ici l'exemple de CBC combiné avec ECBC-MAC en mode "encrypt-and-MAC", dans lequel :

- On chiffre le message avec CBC ;
- On appelle le MAC *sur le message clair* ;
- On renvoie les deux résultats.

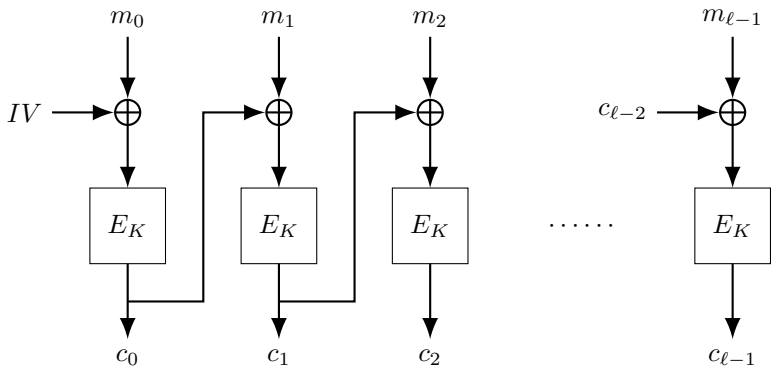

FIGURE 1 – Le mode CBC.

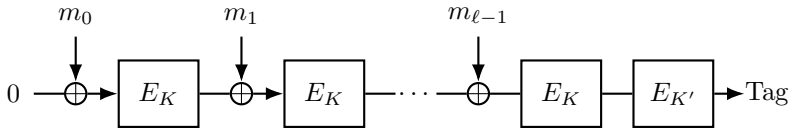

FIGURE 2 – ECBC-MAC.

Suppsons qu'Alice chiffre un message à deux blocs (P_1, P_2) avec la clé K et une IV aléatoire et l'envoie à Bob.

Question 1. *Donner l'expression du chiffré (C_1, C_2) et du tag T. Montrer que si $IV = 0$, $T = E_{K'}(C_2)$.*

Solution.

$$\begin{cases} C_1 = E_K(IV + P_1) \\ C_2 = E_K(P_2 + C_1) \\ T = E_{K'}\left(E_K(P_2 + E_K(P_1))\right) \end{cases} \tag{1}$$

(trivial)

Question 2. *En déduire qu'un attaquant Eve peut créer un nouveau triplet (C_1', C_2', T') valide.*

Solution. *$T' = T$ et on a juste à changer C_1'.*

Question 3. *On suppose maintenant qu'on utilise encrypt-then-MAC. Donner la nouvelle expression du tag. Montrer qu'il y a toujours un problème avec cette construction ; en particulier qu'on peut casser la sécurité IND-CPA.*

Solution.

$$\begin{aligned} T &= E_{K'}(E_K(C_2 + E_K(C_1))) \\ &= E_{K'} \circ E_K(E_K(P_2 + C_1) + E_K(C_1)) \end{aligned}$$

Si $P_2 = 0$ le tag devient constant : $T = E_{K'}E_K(0)$. C'est vrai quelle que soit l'IV.

Dans le jeu IND-CPA l'attaquant peut donc d'abord demander le tag d'un message avec $P_2 = 0$, puis distinguer entre deux messages pour lequel $P_2 = 0$ sur l'un des deux et $\neq 0$ sur l'autre.

Attaque par Oracle de remplissage sur CBC

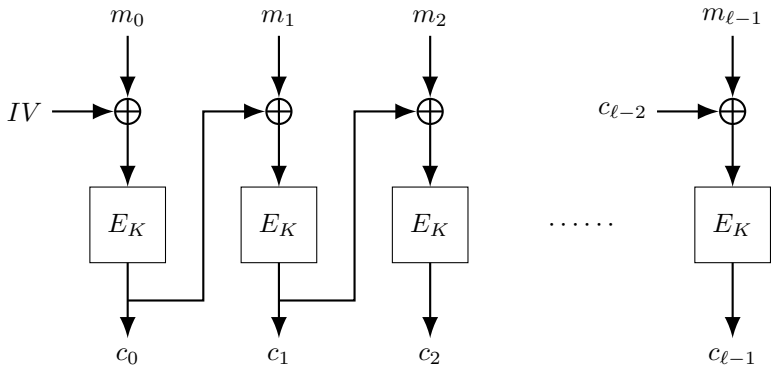

FIGURE 3 – Le mode CBC.

On considère le mode CBC avec un chiffrement à bloc de n octets. Un texte en clair n'a pas forcément une longueur multiple de la taille des blocs. Pour contourner ce problème, on utilise un remplissage (*padding*) : la norme PKCS7 stipule que l'on remplit les octets restants par le **nombre d'octets nécessaires**. Par exemple, "Hello World" deviendra "Hello World`0x0505050505`" pour compléter en blocs de 8 octets, où `0x05` est la notation hexadécimale pour le nombre 5. Dans la suite le i-ème octet d'un bloc m pourra être noté $m[i]$.

Soit Bob un serveur et Alice un client qui communiquent à l'aide de CBC, combiné avec un MAC sûr. L'adversaire Eve intercepte un message secret envoyé par Alice et communique ensuite avec le serveur Bob. Lorsque Eve (se faisant passer pour Alice) envoie un message à Bob, celui-ci effectue les opérations suivantes :

1. Déchiffrement du message avec CBC ;
2. Vérification que le padding est correct. Si ce n'est pas le cas, renvoyer "Erreur" à Alice ;
3. Vérification du tag. Si le tag est incorrect, renvoyer "Erreur" à Alice.

Question 4. *Montrer qu'en étudiant le temps de réponse de Bob, Eve peut utiliser le serveur comme un* oracle de padding $O^{padding}$ *qui prend en entrée un texte chiffré c et renvoie :*

- $\top$ *si le padding du texte clair correspondant est correct (c'est-à-dire s'il se termine par le bon nombre de symboles "*`0x0i`*")*
- $\bot$ *sinon*

Solution. *C'est le principe fondamental d'une "timing attack". Bien que le message d'erreur soit le même dans les deux cas, le tag n'est vérifié que lorsque le padding est correct, ce qui signifie que lorsque le padding est correct, le temps est plus élevé. Eve peut distinguer entre les deux situations, ce qui donne cet oracle.*

Soit $C = (c_0, \ldots, c_{t-1})$ un texte chiffré intercepté. Nous nous concentrons d'abord sur le déchiffrement de c_{t-1}, le dernier bloc de C. Soit $C' = [r_0\| \ldots \|r_{n-1}], c_{t-1}$ un (faux) chiffré à deux blocs produit par Eve. Ici $[r_0\| \ldots \|r_{n-1}]$ est une concaténation de n octets r_i choisi par Eve, et c_{t-1} est un bloc complet récupéré sur le chiffré intercepté. Soit $(m'_0, m'_1) = \mathrm{CBC}^{-1}(C')$ (le déchiffrement légitime par Bob).

Question 5. *Donner une relation entre $m'_1, c_{t-2}, [r_0\| \ldots \|r_{n-1}]$ et m_{t-1} le dernier bloc du texte en clair correspondant à c.*

Solution. *On a : $m'_1 = (r_0, \ldots, r_{n-1}) \oplus E_K^{-1}(c_{t-1})$ et $m_{t-1} = c_{t-2} \oplus E_K^{-1}(c_{t-1})$.*
On a : $m'_1 = (r_0, \ldots, r_{n-1}) \oplus E_K^{-1}(c_{t-1})$.
Par conséquent : $m_{t-1} = c_{t-2} \oplus m'_1 \oplus [r_0\| \ldots \|r_{n-1}]$ ou encore $m'_1 = [r_0\| \ldots \|r_{n-1}] \oplus c_{t-2} \oplus m_{t-1}$.
N'oublions pas que c_{t-2} est connu ici puisque le chiffré a été intercepté. Cela signifie que si on connaît un octet de m'_1, on connaît l'octet correspondant de m_{t-1}.

Question 6. *On prend $r_0 = \ldots = r_{n-2} = 0$ et on fait varier r_{n-1}. Supposons qu'il existe une seule valeur de cet octet pour laquelle $O^{padding}(C') = \top$. Montrer qu'on peut en déduire le dernier octet de m_{t-1}.*

Solution. *On appelle l'oracle de remplissage pour déterminer si m'_1 est correct. Si oui, cela signifie que $m'_1 = [r_0, \ldots, r_{n-1}] \oplus (c_{t-2} \oplus m_{t-1})$ se termine par 1 octet `0x01`, par deux octets `0x02`, ou par trois octets `0x03`, etc.*
On prend $r_0 = \ldots = r_{n-2} = 0$ et on modifie la valeur de r_{n-1}. Si les derniers octets de $(c_{t-2} \oplus m_{t-1})$ sont `0x02`$*$ ou `0x0303`$*$ ou (etc.), il y aura deux valeurs possibles rendant le padding correct : $r_{n-1} =$ `0x01` $\oplus (c_{t-2} \oplus m_{t-1})[n-1]$, ou $r_{n-1} =$ `0x0i` $\oplus (c_{t-2} \oplus m_{t-1})[n-1]$. Sinon, seule la première valeur est possible. On en déduit $m_{t-1})[n-1]$.

Question 7. *Montrer qu'on peut obtenir le dernier octet de m_{t-1} dans tous les cas en (au plus) 257 requêtes à $O^{padding}$.*

Solution. *Le cas qu'il reste à traiter est si on a trouvé deux valeurs possibles du padding. Dans ce cas il suffit de modifier r_{n-2} : au lieu de prendre 0 on prend une valeur arbitraire. Alors $(r_0, \ldots, r_{n-2}, r_{n-1}) \oplus (c_{t-2} \oplus m_{t-1})$ ne peut plus se terminer par deux octets `0x02` / trois octets `0x03` / etc, et il ne reste donc plus qu'un padding possible. La question précédente demande 256 requêtes (une pour chaque valeur de l'octet), celle-ci ne demande qu'une seule requête puisqu'il reste à re-tester une des deux valeurs de r_{n-1} obtenues à la question précédente.*

Question 8. *Montrer comment configurer $m'_1[n-1]$ à `0x02` et l'utiliser pour récupérer $m_{t-1}[n-2]$. En déduire comment récupérer m_{t-1} entièrement.*

Solution. *On modifie la valeur de r_{n-1}. On est alors ramené au problème précédent, sauf qu'il n'y a qu'un seul padding valide (`0x0202`).*

Question 9. *Pouvons-nous récupérer tous les blocs du message ?*

Solution. *Non, il manquera le bloc m_0. En effet à chaque fois on utilise une relation avec c_{t-2} (qui est connu) pour déchiffrer c_{t-1}, mais cela ne fonctionne plus pour le tout premier bloc de message.*

On utilise maintenant une nouvelle fonction de padding. Étant donné un message M, si b est le dernier bit de M (0 ou 1), on complète M avec le complémentaire de b. Si M est vide, on renvoie un bloc de zéros. Si M se termine par un bloc complet, on ajoute un bloc de zéros. Ce padding est évidemment correct (et réversible, c'est-à-dire qu'on peut toujours revenir d'un message paddé au message initial).

Question 10. *L'attaque par oracle de padding s'applique-t-elle dans ce cas ? Pourquoi ?*

Solution. *Non, car tous les messages sont valides. Donc l'oracle de padding renverrait toujours* $\top$.

Remark 1. L'attaque de remplissage sur CBC s'est avérée très puissante en pratique, et a été la source de nombreuses vulnérabilités, notamment en combinaison avec une attaque de *downgrade* sur un protocole.

Réseaux de Feistel

Soient $G_i : \{0,1\}^\ell \times \{0,1\}^\ell \to \{0,1\}^\ell$ des familles de fonctions, et soit $d \geq 1$ un entier. Le réseau de Feistel de profondeur d associé à G_i est la famille de fonctions $F^{(d)} : \{0,1\}^{2\ell} \times \{0,1\}^{\ell d} \to \{0,1\}^{2\ell}$ définie comme suit :

$$F^{(d)}((K_i)_{i\in[1,d]}, x) : \begin{cases} 1: & L_0||R_0 \leftarrow x \\ 2: & \text{Pour } i \in [1,d] \text{ faire} \\ 3: & \quad L_i \leftarrow R_{i-1}, \quad R_i \leftarrow G_i(K_i, R_{i-1}) \oplus L_{i-1} \\ 4: & \text{Retourner } L_d||R_d \end{cases}$$

Question 11. *Dessinez une représentation d'un réseau de Feistel de profondeur 3.*

Question 12. *Montrez qu'un réseau de Feistel est inversible, même si les fonctions G_i ne le sont pas.*

Les réseaux de Feistel sont un moyen de construire une permutation inverse de manière efficace à partir d'un ensemble de fonctions pseudo-aléatoires. Dans le reste de cet exercice, nous supposons que $G_i(K, \cdot)$ est une famille de fonctions pseudo-aléatoires.

Question 13. *Rappeler (brièvement) la définition d'une PRF.*

Solution. *En tirant K_i uniformément au hasard, $G_i(K_i, \cdot)$ est indistinguable d'une fonction aléatoire.*

Question 14. *Montrer que ni $F^{(1)}$ ni $F^{(2)}$ ne sont des PRF.*

Solution. *Pour $F^{(1)}$ c'est trivial car on n'a agi que sur la moitié de l'entrée. Pour $F^{(2)}$ il suffit de prendre une entrée avec R_0 constant et L_0, L'_0 prenant deux valeurs distinctes, alors $L_2 \oplus L'_2$ vaut $L_0 \oplus L'_0$. (On n'en a pas parlé en cours, mais techniquement il s'agit d'une attaque différentielle).*

Question 15. *On considère maintenant le jeu PRP joué par un adversaire PPT. On commence par remplacer les fonctions $G_i(K_i, \cdot)$ par de vraies fonctions aléatoires G'_i. Justifier que la probabilité que, pour deux requêtes différentes de l'adversaire, on ait une collision en R_1 (c'est-à-dire $R_1^i = R_1^j$), est négligeable.*

Solution. *$R_1 = L_0 \oplus G'_1(R_0)$ donc si on a une collision, on a trouvé un quadruplet $L_0^i, R_0^i, L_0^j, R_0^j$ tel que $L_0^j \oplus G'_1(R_0^j) = L_0^i \oplus G'_1(R_0^i)$. On a $R_0^j \neq R_0^i$ nécessairement (sinon cela signifie que l'adversaire est en train de faire deux fois la même requête, on peut éliminer les requêtes en double).*

Par hypothèse G'_1 est une fonction aléatoire, donc chaque fois que l'adversaire effectue une nouvelle requête à $F^{(3)}$ sur l'entrée L_0^i, R_0^i, la sortie $G'_1(R_0^i)$ est tirée uniformément au hasard (et indépendamment de L_0^i). On peut faire la liste de toutes les valeurs $G'_1(R_0^j) \oplus G'_1(R_0^i)$, et utiliser une union bound : la probabilité qu'un élément de cette liste soit égal à un des $L_0^i \oplus L_0^j$ est négligeable, donc c'est le cas pour la somme. (C'est une manière très inélégante, mais simple, de montrer que la proba est négligeable ; pour faire une preuve précise on a besoin de plus de travail).

Question 16. *Justifier que $F^{(3)}$ est une PRF (la preuve complète étant très technique, on demande seulement une intuition ici).*

Solution. *Tant que l'évènement d'une collision en R_1 n'arrive pas, l'adversaire ne peut pas distinguer entre l'accès à $F^{(3)}$ et l'accès à une fonction aléatoire. Pour le voir, il est utile de penser à une fonction aléatoire de manière « paresseuse », c'est-à-dire qu'on sélectionne une sortie au hasard à chaque nouvelle requête. Dans le cas $F^{(3)}$, les sorties sont $R_2 = L_1 \oplus G_1(K_1, R_1) = R_0 \oplus G(K, R_1)$ et $L_2 = R_1 = L_0 \oplus G_2(K_2, R_0)$. À chaque nouvelle requête, la valeur R_1 est tirée uniformément au hasard et la valeur R_2 aussi (indépendamment puisqu'on a deux fonctions aléatoires indépendantes). Donc c'est aussi le cas de la sortie complète de $F^{(3)}$.*

Question 17. *Trouvez une attaque qui distingue $F^{(3)}$ d'une permutation aléatoire, si l'on autorise accès à la permutation inverse (on montre ainsi que $F^{(3)}$ n'est pas une permutation pseudo-aléatoire « forte » , strong PRP – en revanche, c'est le cas de $F^{(4)}$).*