Introduction à la cryptographie
TD5 – Chiffrement par flot, par bloc, et mode de chiffrement

Cécile Pierrot

7 novembre 2019

1 Malléabilité d'un chiffrement par flot

Dans cet exercice, nous considérons un chiffrement par flot, noté E, paramétré par une clé secrète K et un vecteur d'initialisation IV.

Question 1. Rappelez le principe général de fonctionnement d'un chiffrement par flot. Étant donnés un message en clair M, une clé K et un vecteur d'initialisation IV, comment le chiffré C est-il obtenu ?

Supposons qu'Alice ait envie de faire un virement bancaire de 100 € à Mallory. Pour cela, elle utilise un système de chiffrement par flot E dont seules elle et sa banque connaissent la clé privée K. Alice chiffre donc l'ordre de virement M qu'elle envoie alors à sa banque.
Mallory est capable d'intercepter et de modifier ce message chiffré C avant que la banque d'Alice ne le reçoive. Elle ne connaît pas M, mais elle sait que les ordres de virement sont des chaînes de caractères de la forme suivante :

$$M = \langle \textit{date} \rangle : \langle \textit{nonce} \rangle : \langle \textit{émetteur} \rangle : \langle \textit{destinataire} \rangle : \langle \textit{montant} \rangle : \langle \textit{commentaire} \rangle$$

où *nonce* est une chaîne aléatoire de 8 chiffres décimaux, que la banque aura transmise à Alice juste avant que celle-ci ne prépare son ordre de virement.

Question 2. À quoi sert ce *nonce* ?

Dans le cas d'Alice et Mallory, le message est donc de la forme suivante :

$$M = \texttt{2019-01-28:}\langle \textit{nonce} \rangle\texttt{:Alice:Mallory:100:}\langle \textit{commentaire} \rangle$$

Question 3. Comment Mallory peut-elle faire pour obtenir 999 € de la part d'Alice ?

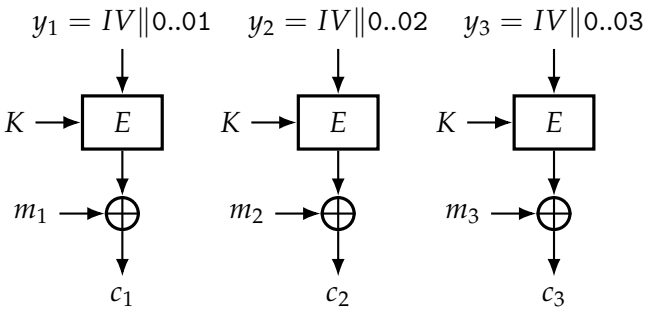
Question 4. Quelle contre-mesure est-il possible de mettre en œuvre pour empêcher ce genre d'attaque ?

2 Modes et vecteur d'initialisation

Nous considérons dans cet exercice un chiffrement par bloc E paramétré par une clé secrète K. Notons n la taille (en bits) des blocs en question.

Question 1. Rappelez les valeurs typiques de n.

Nous nous intéressons tout d'abord au cas du mode CTR (*Counter*), dont nous rappelons ici la définition. Étant donné un vecteur d'initialisation IV de $n - 64$ bits, nous notons $y_i = IV \| i$, pour tout $0 < i < 2^{64}$, la concaténation de cet IV avec l'entier i, représenté sur 64 bits. Le chiffrement du $i^{\text{ème}}$ bloc en clair m_i est alors donné par la formule $c_i = m_i \oplus E_K(y_i)$, pour tout $0 < i < 2^{64}$, comme représenté sur le schéma suivant :



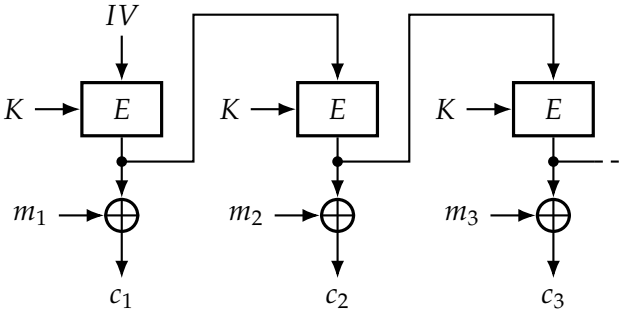
Question 2. Donnez le schéma de déchiffrement, ainsi que la formule correspondante pour calculer chaque bloc m_i en fonction de la clé K, du vecteur d'initialisation IV et du bloc chiffré c_i.

Supposons alors qu'un utilisateur décide d'utiliser toujours le même vecteur d'initialisation IV pour chiffrer plusieurs messages. Supposons de surcroît que vous, l'attaquant, disposiez d'un couple clair / chiffré (M, C).

Question 3. Quelle est le nom de ce type d'attaque ?

Question 4. Pouvez-vous utiliser la connaissance de M et C afin de décrypter d'autres messages chiffrés avec la même clé K et le même vecteur d'initialisation IV ? Si oui, comment faites-vous ?

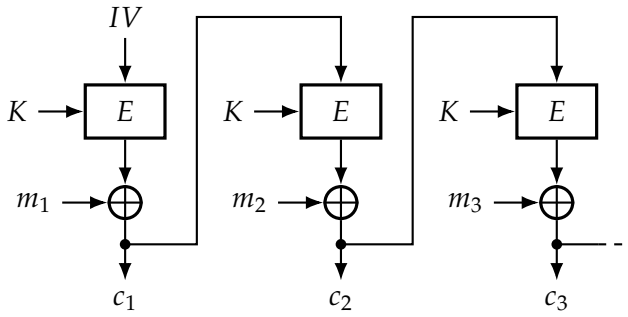
Considérons alors le mode OFB (*Output Feedback*) suivant :



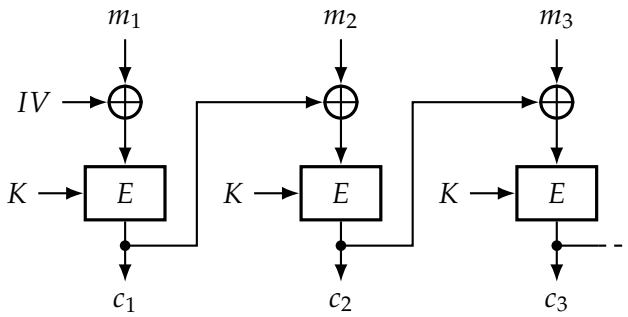
Question 5. Donnez les formules de chiffrement et de déchiffrement de ce mode.

Question 6. Est-ce qu'une attaque à clair connu est possible sur le mode OFB si un même vecteur d'initialisation IV est utilisé pour tous les messages ?

Question 7. Mêmes questions pour le mode CFB (*Cipher Feedback*) suivant :



Question 8. Mêmes questions pour le mode CBC (*Cipher Block Chaining*) suivant :



3 Pour aller plus loin : Oracle de *padding* dans SSL 3.0 – l'attaque POODLE

Le but de cet exercice est de révéler une faille du *padding* tel que défini pour SSL 3.0 dans le cas du mode de chiffrement CBC [1]. SSL 3.0 est la dernière version de SSL, proposée en novembre 1996, il a depuis passé la main à son successeur, TLS. La date de sortie de SSL 3.0 explique que nous allons considérer des primitives que nous savons cassées depuis, comme c'est le cas pour SHA-1. Le protocole SSL est notamment banni en 2014 à la suite de l'attaque POODLE que nous abordons ici. Cette méthode vous sera utile si vous souhaitez vous attaquer au problème *AES-CBC* proposé sur root.me .
Pour cela, nous nous plaçons donc dans le cas d'une connexion SSL 3.0 établie entre un client et un serveur, et utilisant un chiffrement par bloc E (AES-128, par exemple) avec le mode CBC. Il est à noter que, dans le cas de ce mode de chiffrement, SSL 3.0 demande à ce que tous les messages soient authentifiés par un algorithme de MAC construit à partir d'une fonction de hachage cryptographique H (MD5 ou SHA-1).
Nous noterons n_E la taille (en octets) des blocs pour l'algorithme de chiffrement E, et n_H la taille (en octets) des hachés produits par la fonction de hachage H.

Question 1. Donnez n_E et n_H dans le cas où $E =$ AES-128 et $H =$ SHA-1.

Lors du *handshake*, le client et le serveur génèrent ainsi une clé secrète de chiffrement K_{Enc}, ainsi qu'une clé secrète pour le MAC, notée K_{Mac}. De plus, client et serveur maintiennent chacun des compteurs internes Seq_{W} et Seq_{R}, initialisés à zéro puis incrémentés à chaque message envoyé ou reçu, respectivement.
Lorsque l'une des parties (le client, mettons) souhaite envoyer un message M de ℓ octets et de type t à l'autre partie (le serveur, donc), elle procède ainsi :

1. elle calcule le MAC correspondant $Mac = \text{MAC}^{H}_{K_{\text{Mac}}}(Seq_{\text{W}} \| t \| \ell \| M)$, de longueur n_H ;

1. Décrit dans la RFC 6101 (`https://tools.ietf.org/html/rfc6101`), Section 5.2.3.2.

2. elle incrémente son compteur Seq_W ;
3. elle forme la chaîne $M' = M \| Mac$, de longueur $\ell' = \ell + n_H$ octets ;
4. elle calcule le nombre d'octets de *padding* à rajouter $k = n_E - 1 - (\ell' \bmod n_E)$;
5. elle prend une chaîne quelconque *Pad* de k octets ;
6. elle forme la chaîne $M'' = M' \| Pad \| k$, de longueur $\ell'' = \ell' + k + 1$ (k est stocké sur 1 octet) ;
7. elle chiffre M'' avec $E_{K_{\text{Enc}}}$ en mode CBC et obtient le chiffré C (l'IV utilisé pour initialiser CBC est connu : il correspond au dernier bloc chiffré du précédent message envoyé) ;
8. elle envoie enfin le message $t \| \texttt{0x0300} \| \ell'' \| C$ au destinataire (la chaîne `0x0300` représente la version du protocole utilisé : SSL 3.0).

Question 2. Montrez que ℓ'' est bien un multiple de n_E, la taille des blocs de l'algorithme de chiffrement E. Quelle est la longueur du chiffré C ?

Question 3. Décrivez le processus de déchiffrement et de vérification mis en œuvre par le destinataire d'un tel message. À quels endroits la vérification peut-elle échouer ? Pour quelles raisons ?

Supposons maintenant que le message que le client souhaite envoyer au serveur soit de longueur ℓ telle que $\ell' = \ell + n_H$ soit un multiple de n_E. En d'autres termes, la chaîne $M' = M \| Mac$ tient sur un nombre entier de blocs, que l'on note $M' = M'_1 \| \dots \| M'_r$, avec $r = (\ell + n_H)/n_E$.

Question 4. Dans ce cas, quelle est la longueur k du *padding* ? Quelle est la longueur finale du message M'' ainsi *paddé* ? Que vaut son dernier bloc ?

Supposons alors que Mallory puisse intercepter et modifier le message chiffré $t \| \texttt{0x0300} \| \ell'' \| C$ correspondant avant de le faire suivre au serveur. Supposons que Mallory puisse aussi écouter la réponse éventuelle du serveur, et ainsi détecter s'il s'agit d'une réponse normale (de type $t' = \texttt{0x17}$) ou bien d'une erreur au niveau du protocole SSL 3.0 (de type $t' = \texttt{0x15}$).
Mallory intercepte donc $t \| \texttt{0x0300} \| \ell'' \| C$, où C est sur $r + 1 = \ell''/n_E$ blocs. Elle dispose ainsi de $C = C_1 \| \dots \| C_r \| C_{r+1}$.
Pour un indice de bloc i choisi, avec $0 < i \leq r$, Mallory décide de remplacer le bloc chiffré C_{r+1} par une copie du bloc chiffré C_i. Elle construit ainsi le chiffré $\widetilde{C} = C_1 \| \dots \| C_r \| C_i$, et envoie alors au serveur le message modifié $t \| \texttt{0x0300} \| \ell'' \| \widetilde{C}$.

Question 5. Comment le message ainsi reçu va-t-il être déchiffré par le serveur ? Quel va être la valeur du $(r+1)^{\text{ème}}$ bloc ainsi déchiffré par $D_{K_{\text{Enc}}}$ en mode CBC ? Donnez la valeur de ce bloc en fonction de M'_i et de données connues de Mallory.

Question 6. Détaillez dans quel(s) cas la vérification du message ainsi reçu par le serveur va échouer. Dans quel(s) cas va-t-elle réussir ? Quelle information Mallory peut-elle alors obtenir sur C à partir du type t' de la réponse qu'elle interceptera de la part du serveur ? Avec quelle probabilité cela arrivera-t-il ?

Puisque la réponse du serveur permet de savoir si le *padding* est correct ou non, on parle alors d'*oracle de padding*. Dans le cas de SSL 3.0, cette attaque s'appelle POODLE [2] (pour *Padding Oracle On Downgraded Legacy*).
Afin de la rendre utilisable en pratique, on peut la coupler avec l'attaque BEAST [3] (pour *Browser Exploit Against SSL/TLS*). Ceci sera peut être étudié et mis en œuvre si nous en avons le temps.

2. Bodo Möller, Thai Duong et Krzysztof Kotowicz, *This POODLE Bites : Exploiting The SSL 3.0 Fallback*, 2014. Disponible à l'adresse `https://www.openssl.org/~bodo/ssl-poodle.pdf`.

3. Thai Duong et Juliano Rizzo, *Here Come The ⊕ Ninjas*, 2011. Disponible à l'adresse `http://netifera.com/research/beast/beast_DRAFT_0621.pdf`.