Cryptographie Avancée - TP 8
ZZ3 F5 - Réseaux et Sécurité Informatique
Fonction de hachages, Merkle-Damgard et MACs

Rendu : au plus tard le 27 novembre, par mail à charles.olivier-anclin@uca.fr.

Récupérez les deux fichiers sources `tea_ref.c` et `tea.c`.

Dans ce TD, nous allons travailler sur le chiffrement *Tiny Encryption Algorithm* (TEA), un chiffrement par bloc utilisant la structure du réseau de Feistel proposé par David Wheeler et Roger Needham en 1994. La fonction de chiffrement de TEA est simple et n'utilise que des opérations élémentaires, ce qui permet de l'implémenter sur des systèmes très limités en ressources. Le fichier `tea_ref.c` contient l'implémentation standard de ce chiffrement.

De façon surprenante, peu de failles ont été découvertes sur TEA malgré la simplicité de son algorithme. Deux autres versions, nommées XTEA et XXTEA, ont été proposées par les mêmes auteurs pour corriger certains défaut. Malgré tout, à ce jour, aucune attaque critique n'a été décrite au sujet de TEA.

Vous pouvez consulter l'article décrivant ce chiffrement à l'url `https://link.springer.com/content/pdf/10.1007/3-540-60590-8_29.pdf`.

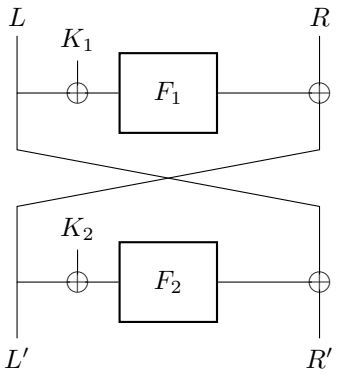
Un diagramme, représentant deux tours du réseau de Feistel de TEA est disponible à l'url `https://upload.wikimedia.org/wikipedia/commons/a/a1/TEA_InfoBox_Diagram.png`

1 Réseau de Feistel de TEA

Exercice 1. Le réseau de Feistel de TEA diffère du réseau de Feistel standard en trois points :

- La fonction F dépend de l'itération (on notera F_i la fonction utilisée à l'itération i).
- On utilise la même clé de chiffrement à chaque itération (pas de *key schedule* ni de sous-clés).
- L'opération utilisée est $+$ (modulo 2^{32}) et non $\oplus$.

TEA utilise 64 itérations du réseau de Feistel pour chiffrer un bloc v de 32 bits (séparé en deux sous-blocs l_0 et r_0 de 16 bits chacun) avec une clé k de 128 bits. On utilisera donc 32 itérations du diagramme suivant pour chiffrer un bloc.



$l_{64}\|r_{64}$ est le chiffré de $v = l_0\|r_0$.

a. Exprimez $l_1\|r_1$ et $l_2\|r_2$ en fonction de l_0 et r_0.

b. A-t-on besoin d'inverser la fonction F pour déchiffrer un bloc ?

c. Exprimez $l_0\|r_0$ en fonction de l_1 et r_1, puis en fonction de l_2 et r_2.

d. Donnez le diagramme d'une itération permettant de déchiffrer un bloc.

Exercice 2. Le fichier `tea.c` contient le code de la fonction F. Dans cet exercice, vous devez compléter ce fichier.

a. Implémentez la fonction `feistel_enc` permettant de chiffrer un bloc avec le chiffrement TEA.

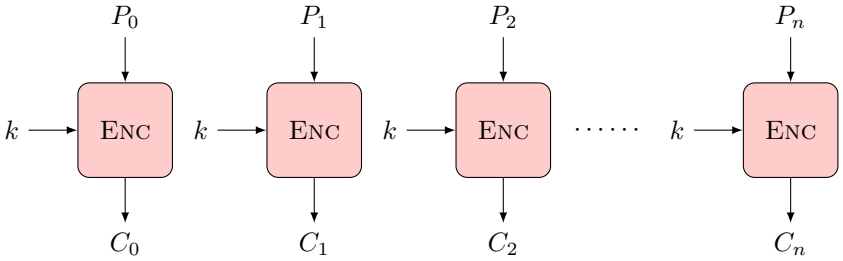
b. Implémentez la fonction `feistel_dec` permettant de déchiffrer un bloc avec le chiffrement TEA.

c. Vérifiez que votre implémentation renvoie bien les mêmes valeurs que l'implémentation standard donnée dans le fichier `tea_ref.c`.

2 Modes de chiffrement

Nous allons à présent implémenter plusieurs modes de chiffrement pour utiliser TEA. Dans chacun des exercices de cette partie, vous devez compléter votre code dans le fichier `tea.c`.

Exercice 3. Le mode *Electronic CodeBook* (ECB) consiste à chiffrer successivement chacun des blocs du message. Ce mode de chiffrement est représenté par le diagramme suivant.

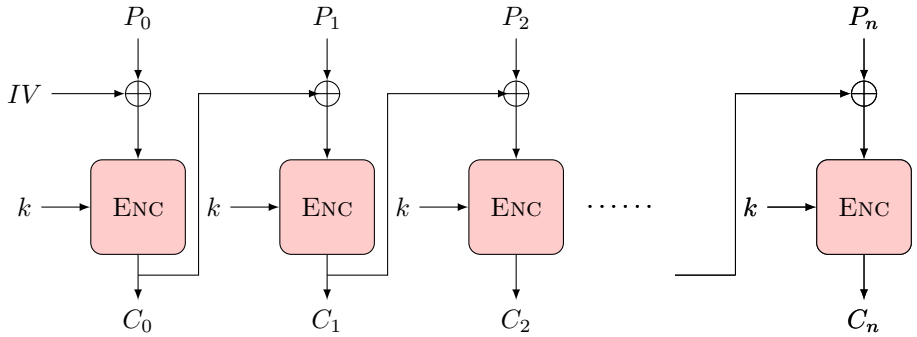


a. Comment déchiffre-t-on un message chiffré avec ce mode ?

b. Implémentez la fonction `ecb_encrypt` qui prend en paramètre un message v de `nb_blocks` blocs et une clé de chiffrement k, et qui chiffre v avec TEA en mode ECB.

c. Implémentez la fonction `ecb_decrypt` qui déchiffre un message chiffré avec TEA en mode ECB.

Exercice 4. Le mode `Cipher Block Chaining` CBC utilise un vecteur d'initialisation aléatoire iv pour chiffrer les messages. Ce vecteur est nécessaire au déchiffrement et est transmis avec le message chiffré (il n'est donc pas secret). De plus, chaque bloc de message chiffré est utilisé lors du chiffrement du bloc de message clair suivant, c'est ce qu'on appelle le *chaînage*. Ce mode de chiffrement est représenté par le diagramme suivant.



a. Comment déchiffre-t-on les messages avec ce mode ? Représentez le déchiffrement sous forme de diagramme.

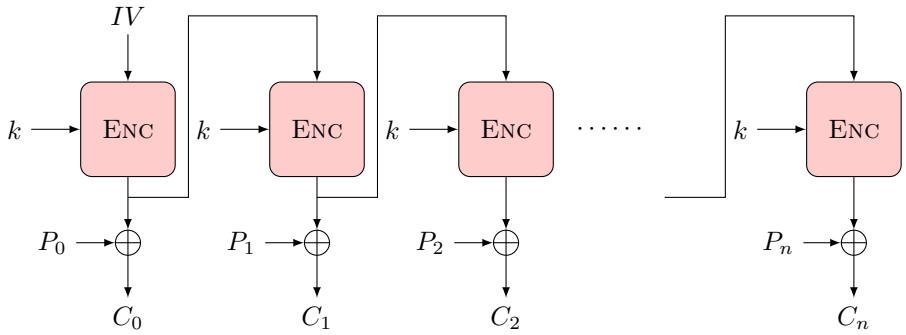
b. À quoi sert le vecteur d'initialisation iv ? Pourquoi est-il nécessaire de chaîner les chiffrements des blocs ?

c. Quelles sont les opérations que l'on peut paralléliser avec ce mode ? Comparez avec le mode ECB.

d. Implémentez la fonction `cbc_encrypt` qui prend en paramètre un message v de `nb_blocks` blocs, une clé de chiffrement k et un vecteur d'initialisation iv, et qui chiffre v avec TEA en mode CBC.

e. Implémentez la fonction `cbc_decrypt` qui déchiffre un message chiffré avec TEA en mode CBC.

Exercice 5. Le mode `Output FeedBack` OFB est représenté par le diagramme suivant.



a. Comment déchiffre-t-on les messages chiffrés avec ce mode ?

b. Ce mode permet d'utiliser un chiffrement par blocs comme un chiffrement par flot. Expliquez pourquoi.

c. Quelles sont les opérations que l'on peut paralléliser avec ce mode ? Comparez avec le mode ECB et CBC.

d. Montrez que dans ce mode, il est possible de pré-calculer les opérations les plus coûteuses du chiffrement sans connaître le message. Est-ce possible avec les modes ECB et CBC ?

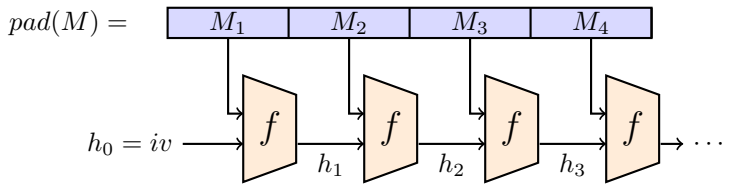
e. Que ce passe-t-il si l'on chiffre plusieurs messages avec le même vecteur iv ? Donnez un exemple d'attaque dans ce cas.

f. Implémentez la fonction `ofb_stream` qui prend en paramètre un nombre de blocs `nb_blocks`, une clé de chiffrement k et un vecteur d'initialisation iv, et qui calcule un flot de chiffrement pour `nb_blocks` blocs avec TEA en mode OFB.

g. Implémentez la fonction `ofb_encrypt` qui prend en paramètre un nombre de blocs `nb_blocks`, un message en clair v et un flot de chiffrement `stream` calculé au préalable, et qui chiffre v avec TEA en mode OFB. Est-ce necessaire d'implémenter une fonction `ofb_decrypt` ? Pourquoi ?

3 Fonction de hachage et code d'authentification depuis un chiffrement par blocs

Exercice 6. Nous allons à présent construire une fonction de hachage en utilisant la structure de Merkle-Damgard. Cette structure permet de produire une fonction de hachage à partir d'une fonction f dite *de compression* et d'un vecteur d'utilisation. La structure de Merkle-Damgard est décrite par le diagramme suivant.



Nous allons utiliser la fonction de compression de Davies-Meyer sur TEA (nous verrons par la suite que c'est une très mauvaise idée). En désignant le chiffrement avec TEA en mode ECB d'un message y de deux blocs avec la clé x par $\texttt{encrypt}_y(x)$, on aura la fonction de compression suivante :

$$f(x, y) = \texttt{encrypt}_y(x) \oplus y.$$

Pour le premier bloc de message, on calculera donc $f(iv, v_0) = \texttt{encrypt}_{v_0}(iv) \oplus v_0$.

a. Quelle est la taille en bits des empreintes produites par cette fonction de hachage ?

b. Quelle est la taille en bits du vecteur d'initialisation ?

c. Implémentez la fonction `hash` qui prend en paramètre un message en clair v et un vecteur d'initialisation iv, et qui calcule l'empreinte de v par la fonction de hachage décrite précédemment.

Exercice 7. La construction HMAC permet d'obtenir un MAC (pour *Message Authentication Code*) à partir d'une fonction de hachage H. Il suffit de hacher la concaténation de la clé secrète k avec le message v à authentifier pour obtenir le code c :

$$c = \mathsf{HMAC}_k(v) = H(k\|v).$$

Pour vérifier que le code c authentifie bien v avec la clé k, il suffit de recalculer $H(k\|v)$ et de comparer le résultat avec c.

a. Implémentez la fonction `hash_mac` qui prend en paramètre un message en clair v, un vecteur d'initialisation iv et une clé k et qui calcule le HMAC de v avec la fonction de hachage implémenté à l'exercice précédent.

b. Implémentez la fonction `hash_mac_verification` qui prend en paramètre un message en clair v, un code c, un vecteur d'initialisation iv et une clé k et qui vérifie si le code c est correct. La fonction renvoie 1 si le code est correcte, 0 sinon.

Exercice 8. Le chiffrement TEA ne garantit pas l'intégrité des messages : l'utilisateur qui déchiffre un message ne sait pas si ce dernier a été altéré ou non pendant sa transmission, puisque n'importe quel chaîne de bits est déchiffrable. Pour garantir l'intégrité des messages, on peut utiliser le paradigme *encrypt-then-MAC* en utilisant la clé secrète pour produire un code d'authentification du chiffré. Dans ce cas, un utilisateur malveillant que ne connaît pas la clé secrète ne peut pas altérer le message chiffré, puisque il n'est pas capable de produire un nouveau code d'authentification pour le chiffré altéré.

a. Implémentez la fonction `cbc_encrypt_mac` qui prend en paramètre un message v de `nb_blocks` blocs, une clé de chiffrement k et un vecteur d'initialisation pour le chiffrement iv et un vecteur d'initialisation pour la fonction de hachage iv_h et qui chiffre v avec TEA en mode CBC puis l'authentifie avec un HMAC basé sur TEA.

b. Implémentez la fonction `cbc_decrypt_mac` qui déchiffre un message chiffré avec TEA en mode CBC si est seulement si le code d'authentification correspondant est valide. Si le code est invalide, le on dira que le déchiffrement échoue.

4 Exploitation d'une faille de TEA

Exercice 9. Comme nous l'avons vu au début du TD, le chiffrement TEA n'utilise pas de *key schedule*. Si cela simplifie son implémentation et participe à le rendre remarquablement efficace, cela implique aussi certaines propriétés exploitables. Par exemple, le fait de remplacer le dernier bit des deux premiers blocs de 32 bits de la clé par leur complémentaires ne modifie pas le message chiffré. Cette propriété n'a pas d'impact critique sur la sécurité de TEA en tant que chiffrement, en revanche elle pose de gros problèmes lorsque TEA est utilisé comme une fonction de hachage. Microsoft en a d'ailleurs fait les frais, en utilisant par erreur TEA pour le hachage sur sa console de jeu Xbox.

a. Vérifiez la propriété énoncée ci-dessus en produisant deux chiffrés identiques qui chiffrent le même message avec deux clés différentes.

b. Montrez que la fonction de hachage qui utilise TEA comme fonction de compression n'est pas résistante aux collisions.

c. Proposez une attaque contre l'intégrité du chiffrement authentifié que vous avez implémenté dans l'exercice 7. Vous devez montrer qu'il est possible d'altérer le déchiffrement du message sans faire échouer l'algorithme de déchiffrement. Implémentez votre attaque.