HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

Laboratoire 2 :

Cryptographie

Département : Technologies de l'Information et de la Communication (TIC)
Unité d'enseignement : Cryptographie (CRY)

Auteurs : Harun Ouweis
Professeur : A. Duc
Date : 1er mai 2024

Hes·so
Haute Ecole Spécialisée
de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences and Arts
Western Switzerland

1 Introduction

Ce rapport présente les solutions aux différents challenges proposés dans le laboratoire de cryptographie. Nous explorerons divers modes opératoires, techniques de chiffrement authentifié, et implémentations de HMAC modifiés.

2 Mode opératoire

Cette section détaille le fonctionnement du mode opératoire inventé par notre collègue.

2.1 Schéma du mode opératoire

Le schéma ci-dessous illustre le processus complet du mode opératoire :

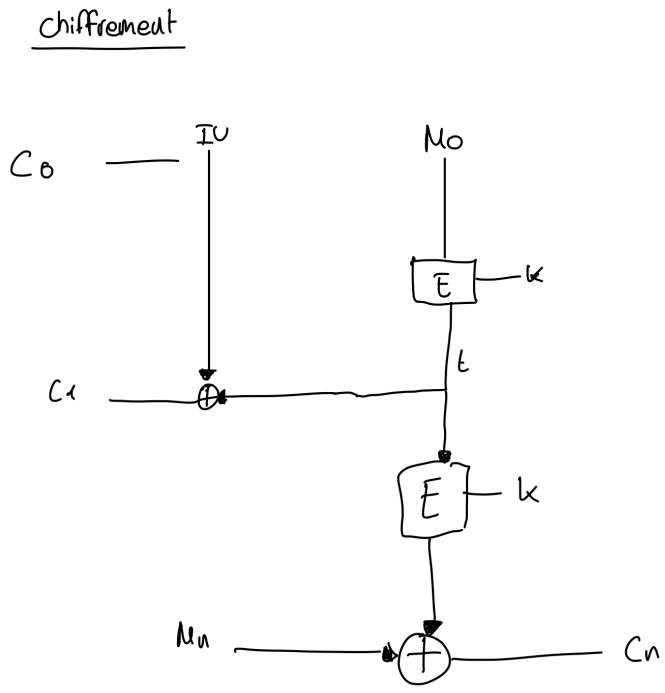

Figure 1 – Schéma du mode opératoire

2.2 Processus de chiffrement

Le chiffrement fonctionne de la manière suivante :

— Le premier bloc du message est chiffré avec la clé, puis XORé avec l'IV pour créer le premier bloc chiffré.
— Pour les blocs suivants, un flux de clé est créé en chiffrant le bloc précédent, puis XORé avec chaque bloc clair.
— Les blocs chiffrés sont ensuite concaténés pour créer le texte chiffré final.

Ce processus a un inconvénient majeur : chaque bloc utilise la même clé, ce qui entraîne un flux de clé constant. Cela rend le chiffrement vulnérable à une attaque par texte clair connu.

2.3 Déchiffrement correspondant

Le déchiffrement fonctionne en inversant les étapes du chiffrement de la manière suivante :

Le premier bloc est décrypté en inversant le XOR :
$m_1 = \text{AES.decrypt}(c_1 \oplus IV)$
Pour chaque bloc suivant :
$t = \text{AES.encrypt}(t)$
$m_i = c_i \oplus t$

La clé reste la même pour les étapes de chiffrement et de déchiffrement, ce qui permet de déchiffrer le message correctement.

2.4 Cassure par texte clair connu

La méthode de chiffrement présente un problème fondamental dans sa conception, ce qui la rend vulnérable à une attaque par texte clair connu.

Problème fondamental : Le mode opératoire utilise la même clé pour chiffrer chaque bloc, créant ainsi un flux de clé constant à chaque itération. De plus, le premier bloc chiffré est XORé avec un vecteur d'initialisation (IV) aléatoire, mais les blocs suivants sont directement liés au bloc précédent chiffré, rendant le chiffrement prévisible si la même clé est utilisée.

Processus de cassure : Pour casser ce chiffrement, nous procédons comme suit :

1. Nous utilisons un texte clair m1 et son texte chiffré c1 pour récupérer le flux de clé t_1.
2. Nous comparons t_1 à t_2, obtenu en ayant XOR les blocs chiffrés de c2.
3. Si t_1 et t_2 sont identiques, alors l'intégralité de c2 peut être décryptée en utilisant ce flux.
4. Le flux de clé permet alors de décrypter chaque bloc de c2 :

$$texteclair = \text{keystream} \oplus \text{c2\_blocks}[i]$$

5. Enfin, les blocs décryptés sont concaténés pour former le texte en clair final.

Résultat de l'attaque : Nous avons réalisé cette attaque avec les données fournies par l'enseignant et obtenu le texte suivant :

```
C2 décrypté: b'This is a long enough message for it to be secure.
The secret flag is melanges\x02\x02'
```

Cette attaque montre comment l'utilisation d'une clé constante et la liaison des blocs chiffrés rendent le chiffrement vulnérable à une attaque par texte clair connu.

3 Chiffrement authentifié

Cette section décrit le système de chiffrement authentifié développé par notre collègue, en mettant en évidence le fonctionnement de la partie CTR, les problèmes rencontrés, et la manière de casser cette construction.

3.1 Fonctionnement de la partie CTR

La partie CTR (Counter Mode) est une méthode de chiffrement en flux où chaque bloc de texte clair est chiffré par un flux de clé généré en chiffrant un compteur. Le chiffrement fonctionne de la manière suivante :

- Le message est divisé en blocs de 128 bits.
- Chaque bloc est chiffré en additionnant son contenu avec la sortie d'un chiffreur AES en mode CTR.
- Le chiffreur CTR utilise un nonce de 12 octets pour initialiser le compteur, et le compteur est incrémenté à chaque bloc.
- Chaque bloc chiffré est concaténé pour former le texte chiffré final.

Lorsqu'un message est plus long qu'un seul bloc, chaque bloc utilise un flux de clé unique généré par le chiffreur CTR, ce qui garantit que chaque bloc est chiffré indépendamment.

3.2 Problèmes de la partie CTR

La partie CTR présente plusieurs problèmes majeurs :

- **rejouabilité des flux de clé :** Le compteur est réinitialisé à chaque itération, ce qui rend le flux de clé pour chaque bloc prévisible et vulnérable à une attaque de relecture.
- **faible protection du message :** La simple addition des blocs de texte clair avec les flux de clé peut être inversée avec une soustraction modulaire, facilitant la récupération du texte clair.
- **faible couplage du MAC :** La structure MAC ne tient pas compte des variations dans la partie CTR, rendant la construction vulnérable aux attaques combinées.

3.3 Cassage de la construction

Pour casser cette construction, nous procédons comme suit :

Étape 1 : calcul de Sigma : Utilisez le premier bloc de texte clair m1 et son texte chiffré c1 pour calculer sigma, la différence modulaire entre ces deux blocs :

$$\sigma = ((c1\_blocs[0]) - (m1\_blocs[0])) \mod p$$

Étape 2 : somme des blocs : Calculez la somme des blocs du texte clair m1 et du texte chiffré c2 pour préparer les étapes suivantes :

$$\sum_{i=0}^{n} m_i = \left(\sum_{i=0}^{n} c_i - n \cdot \sigma\right) \mod p$$

Étape 3 : calcul de V : Utilisez sigma, la somme des blocs m1, et le tag1 pour calculer v, la constante nécessaire pour la partie MAC :

$$v = \left(((tag1) - sigma) \times modInverse(\sum_{i=0}^{n} m_i, p)\right) \mod p$$

Étape 4 : trouver Sigma pour le deuxième message : En utilisant la formule de MAC, nous pouvons isoler sigma pour le deuxième message comme suit :

$$\sigma_2 = ((tag2 - v \cdot \text{sumC2}) \cdot \text{modInverse}(1 - v \cdot n, p)) \mod p$$

Étape 5 : déchiffrement : Utilisez v et $sigma_2$ pour déchiffrer chaque bloc de c2 en inversant le processus de chiffrement :

$$texteclair = \bigoplus_{c2_{bloc} \in c2\_blocs} (((c2_{bloc}[i]) - \sigma_2) \mod p)$$

Résultat de l'attaque : Nous avons appliqué cette méthode avec les données fournies par l'enseignant et obtenu le texte suivant :

```
Cassage du chiffrement du deuxième message
Texte clair 2 = b'Congrats! The secret is alluring'
```

Cette attaque montre comment la structure CTR et la construction MAC peuvent être cassées pour révéler le texte clair du message.

4 HMAC

Dans cette section, nous présentons la construction HMAC proposée par notre collègue. Nous décrivons en détail les vulnérabilités spécifiques à cette implémentation et expliquons comment nous pouvons les exploiter pour falsifier des transactions bancaires.

4.1 Fonctionnement de la construction HMAC

La construction HMAC (Hash-based Message Authentication Code) proposée par notre collègue est basée sur une simplification de la construction standard.

Le fonctionnement général est le suivant :

— Le message est d'abord paddé en ajoutant un byte, suivi de zéros pour atteindre une longueur multiple de 16 bytes.
— Le message est ensuite divisé en blocs de 16 bytes.
— Chaque bloc est compressé en utilisant AES en mode ECB, avec le bloc comme clé et l'état de hashage précédent comme donnée d'entrée.
— La sortie de la compression est XORée avec l'état de hashage précédent pour générer le nouvel état de hashage.
— Le tag final est l'état de hashage obtenu après le dernier bloc.

4.2 Vulnérabilités de l'implémentation HMAC

Cette implémentation présente plusieurs vulnérabilités spécifiques :

— **faible couplage de la clé :** Utiliser la clé comme IV rend la construction vulnérable aux attaques de type "collision de la clé", où des messages différents peuvent générer le même tag.
— **faible résistance aux collisions :** La structure Merkle-Damgård elle-même est vulnérable aux attaques de collision, permettant à un attaquant de produire des messages différents ayant le même tag.
— **permet la falsification :** La construction permet de falsifier un MAC valide en ajoutant simplement un bloc supplémentaire au message initial, sans connaître la clé d'origine. Cela permet d'effectuer une attaque par extension, qui compromet sérieusement l'intégrité et la sécurité du code d'authentification.

Détail de la falsification : La falsification est possible car l'état de hashage final du message initial est utilisé comme point de départ pour le hashage du message étendu. En connaissant le MAC du dernier bloc du message initial, nous pouvons ajouter un nouveau bloc et générer un nouveau MAC en utilisant la fonction ‘h‘.

Pour falsifier le message :

1. Nous paddons d'abord le dernier bloc du message initial pour atteindre la taille de bloc requise.
2. Nous ajoutons un nouveau montant supérieur, paddons le tout pour atteindre la bonne taille, et générons le MAC pour ce nouveau message en utilisant la fonction ‘h‘.
3. La fonction ‘h‘ utilise le MAC précédent comme point de départ pour compresser le nouveau bloc en utilisant AES en mode ECB.
4. La sortie de cette compression est XORée avec l'état de hashage précédent pour générer le nouvel état.
5. La nouvelle balise permet de valider le message étendu, en maintenant l'intégrité apparente de la transaction.

4.3 Exploitation de la construction HMAC

Pour exploiter cette construction, nous procédons comme suit :

Étape 1 : création du message initial : Nous générons un message initial m contenant des informations de transaction bancaires, et calculons son MAC en utilisant la fonction HMAC proposée.

Étape 2 : création d'un nouveau message : Nous créons un nouveau message m' basé sur le message initial, mais avec un montant supérieur. Le message est paddé et concaténé avec le nouveau montant :

$$m' = \text{pad}(m) + b\text{"123456"}$$

Étape 3 : génération du nouveau MAC : Nous calculons le MAC du message falsifié en utilisant la balise du message initial en tant qu'état de hashage initial. La fonction 'h' génère le nouvel état de hashage pour le message étendu en utilisant AES en mode ECB.

Étape 4 : vérification : Nous vérifions le nouveau message en utilisant la fonction de vérification avec la clé d'origine et comparons le MAC généré :

$$verify(forged\_message, k, forged\_mac)$$

Résultat de l'attaque : Nous avons appliqué cette méthode à une transaction bancaire fournie et avons pu générer un message falsifié avec un MAC valide. Ce message falsifié a pu être vérifié avec la clé d'origine, montrant que la construction est vulnérable à la falsification.

```
Verification of original message with key = True
Verification of forged message with original key = True
Forged message:
Sender: Alexandre Duc; Destination account 12-1234-12. Amount CHF123123456
```

Cependant, en utilisant les données fournies par l'enseignant, notre fonction 'verify' retournera "False" car nous ne possédons pas la clé d'origine. Néanmoins, l'enseignant ayant la clé pourra vérifier et obtenir "True", démontrant que notre message falsifié est valide.

```
Verification of forged message with my key = False
Forged message for provided message:
Sender: Alexandre Duc; Destination account 12-1234-12. Amount CHF123123456
```

Cette attaque démontre clairement les faiblesses spécifiques de cette construction HMAC et souligne la nécessité d'une structure plus robuste et sécurisée.

5 Signatures

Yverdon-les-Bains le 1er mai 2024

Harun Ouweis