CRY 2025

Laboratoire #2

08-04-2025

Préambule

- Ce laboratoire demande de résoudre différents challenges basés sur la cryptographie symétrique. Chaque étudiant doit rendre un rapport **individuel** avec ses propres solutions ainsi que son code. Les façons d'arriver à la solution sont libres mais doivent être expliquées dans le rapport.
- Vous pouvez par contre discuter ensemble pendant que vous résolvez les problèmes. Essayez de ne pas spoiler !
- Voici une liste de fonctions qui pourraient vous être utiles en python :
  - dans le module pycryptodome [1] : `Crypto.Cipher.AES` et `Crypto.Util.strxor`.
  - dans le module base64 : `base64.b64encode()`, `base64.b64decode()`
- Toutes les données **binaires** sont en **base64**. N'oubliez pas de les décoder avant de les utiliser.
- Certains exercices peuvent contenir des mots tirés aléatoirement. Ne vous en offensez pas et googlez-les à vos risques et périls.
- Vous trouverez vos entrées personnalisées dans le zip `nom_prenom.zip` sur cyberlearn.
- Le **rapport** et votre **code** doivent être rendus dans un zip sur cyberlearn avant le **27.04 à minuit**.
- Il se peut que j'annonce des erreurs sur cyberlearn/teams.

1 Spongy AES (1.5 pts)

Vous trouverez dans le fichier `sponge.py` une fonction de hashage custom.

1. Dessinez un schéma de la construction pour un message de 19 caractères ASCII. Soyez le plus précis possible concernant le "rate" et la taille de la sortie.
2. Vous avez que le message a la forme suivante FLAG{...} avec 13 caractères entre les accolades. Vous trouverez dans votre fichier de paramètres le hash de ce message. Récupérez le message. Expliquez dans votre rapport comment vous avez procédé.
   Indice : vous aurez besoin de faire *un peu* de bruteforce. Vous savez aussi que le flag ne contient uniquement des caractères ASCII imprimables.

2 GCM (2.5 pts)

Vous trouverez dans le fichier `gcm.sage` une version custom d'AES-GCM. Pour simplifier, nous acceptons de chiffrer uniquement des messages qui ont une taille multiple de 128 bits (ceci n'est pas une erreur).

1. Trouvez une erreur d'implémentation dans ce code. La fonction f dans le code est correcte et fait partie du standard. Elle permet de fixer une représentation des éléments. L'inverse de la fonction f est f elle-même.
2. Implémentez la déchiffrement GCM tout en conservant l'erreur d'implémentation. Testez votre implémentation.

1. https://pycryptodome.readthedocs.io

3. Vous trouverez dans votre fichier de paramètres deux textes chiffrés (iv, chiffré, tag) ainsi que le texte clair correspondant au premier message. Que remarquez-vous d'étrange dans ces textes chiffrés ?
4. Récupérez le texte clair correspondant au deuxième texte chiffré (iv2, c2, t2). Expliquez votre attaque dans votre rapport.
5. Cassez aussi l'intégrité en fournissant un texte chiffré valide (iv, c, t) **en base64** correspondant au message `mChall` que vous trouverez dans votre fichier de paramètres.

3 Speck (1 pt)

Speck est un système de chiffrement par blocs designé par la NSA. Son but est d'être particulièrement léger et est pensé pour l'IoT. Vous trouverez dans le fichier `speck.py` une implémentation de Speck avec la classe `speck`. Vous n'avez **pas** besoin de comprendre le fonctionnement ni les détails de cet algorithme. Speck n'est pour l'instant pas cassé. La version de Speck choisie a une taille de bloc de 32 bits et une taille de clef de 64 bits.

Une tour radio souhaite transmettre un mot de passe de quatre caractères à ses clients. Pour cela, elle décide de le chiffrer avec Speck. Comme elle ne sait pas quand le client sera connecté, elle décide de diffuser en boucle la séquence suivante : le mot de passe suivi d'une séquence de 4 bytes à 0 indiquant la fin du mot de passe.

Cette longue séquence est chiffrée en un seul message avec Speck et le mode opératoire CBC. L'IV est envoyé en premier.

1. Lorsqu'un client légitime se connecte, il n'a pas forcément pu récupérer l'IV. Expliquez comment il peut tout de même récupérer le mot de passe. Un client **connait la clef** secrète utilisée par Speck. On suppose que le message est aligné sur la taille des blocs.
2. Vous avez intercepté une partie de cette séquence chiffrée [2] mais vous n'avez pas réussi à récupérer l'IV utilisé pour chiffrer le message. Cette séquence (en base64) se trouve dans un fichier séparé nommé `prenom_nom-speck.txt`. Le main du fichier `speck.py` contient aussi tout le code de la tour radio. Le fichier contenant la séquence chiffrée est très grand. **N'essayez pas de l'ouvrir avec votre éditeur de texte**. A la place, chargez-le directement dans votre script Python de la manière suivante

```
f = open("prenom_nom-speck.txt","r")
ct = f.read()
```

   Récupérez le mot de passe et **indiquez-le** dans votre rapport. Expliquez aussi comment fonctionne votre attaque et pourquoi elle fonctionne.
   Indice : est-ce que des blocs de texte chiffrés se répètent ? Faites un schéma pour comprendre ce que cela implique.
3. Est-ce que cette attaque fonctionnerait aussi sur AES-CBC ? Justifiez et soyez précis.

2. Le message intercepté est aligné sur la taille des blocs.