CRY 2024

Laboratoire #2

18-04-2024

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`, `Crypto.Random` 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 **01.05 à minuit**.
- Il se peut que j'annonce des erreurs sur cyberlearn/teams.

1 Mode opératoire

Vous trouverez dans le fichier `mode_of_op.py` une implémentation d'un mode opératoire inventé par un collègue de 1ère année. [2]

1. Dessinez un schéma représentant ce mode opératoire.
2. Décrivez à l'aide d'un schéma ou mathématiquement le déchiffrement correspondant.
3. Implémentez le déchiffrement correspondant.
4. Effectuez une attaque à texte clair connu. Vous trouverez dans votre fichier de paramètres un texte clair m_1 et son texte chiffré (en base64) c_1. Décryptez le texte chiffré c_2. Expliquez dans votre rapport comment votre attaque fonctionne.

2 Chiffrement authentifié

Le même collègue décide de créer son propre système de chiffrement authentifié. Vous trouverez dans le fichier `encAndMac.py` ce système. L'idée est de chiffrer un message et de calculer son MAC en parallèle. Il s'agit de la construction générique *encrypt-and-MAC*.

Votre collègue a décidé de modifier CTR afin "d'exploiter la puissance des nombres premiers". Il modifie aussi légèrement l'algorithme de MAC Poly1305 (pas encore vu en classe). Le MAC fonctionne de la manière suivante (n'hésitez pas à regarder le code en parallèle afin de comprendre plus rapidement les parties un peu mathématiques) :

1. https://pycryptodome.readthedocs.io
2. Sûrement un logiciel.

1. Tout d'abord la constante $v = \mathsf{AES}_k(\texttt{0xff...ff})$ est calculée.
2. Le texte clair est ensuite divisé en blocs de 128 bits et visualisé comme un entier de 128 bits (big endian). Nous obtenons donc les nombres $m_0, m_1, \dots m_n$.
3. Un masque σ dérivé du nonce est calculé comme $\sigma = \mathsf{AES}_k(\text{nonce} \| \texttt{0x00000000})$
4. Le MAC vaut $\sum_{i=0}^{n} m_i v + \sigma \bmod p$

Nous vous laissons analyser la partir "CTR" tout seul. Vous allez devoir casser cette construction.

1. Décrivez le fonctionnement de la partie CTR. En particulier, intéressez-vous à l'algorithme quand le message a une taille plus grande qu'un bloc.
2. Quels sont les gros problèmes de la partie "CTR" ?
3. Cassez cette construction. Expliquez dans votre rapport comme votre attaque fonctionne. Vous trouverez aussi dans votre fichier de paramètres un texte clair m_1 et son texte chiffré (en base64) (nonce1, c_1, tag1). Décryptez le texte chiffré (nonce2, c_2, tag2) fourni.
   Indice : Essayez de récupérer tout d'abord la valeur de v.
   Indice2 : Vous aurez besoin de faire des calculs dans $\mathbb{Z}_p$. Vous pouvez le faire, soit dans Sage, soit en Python. Dans ce cas, considérez la fonction `pow` pour calculer les inverses modulaires. [3]

3 HMAC

Votre collègue récidive. Il décide maintenant de créer son propre algorithme de MAC. En cherchant sur internet, il tombe sur la construction HMAC qu'il trouve trop compliquée. Il décide donc de la modifier et de simplement hasher le message avec la clef comme IV de la fonction de hashage dans la construction de Merkle-Damgård. Comme fonction de compression, il décide d'utiliser un simple AES. Comme padding, il décide d'utiliser le padding NIST 800-38a (un standard).

Notre apprenti à tellement confiance en son système qu'il décide de l'utiliser pour chiffrer des transactions bancaires. Vous trouverez le code dans le fichier `hmac.py`.

1. Implémentez la fonction de vérification de MAC.
2. On vous donne dans votre fichier de paramètres une transaction bancaire m qui vous est destinée ainsi que son MAC mac (en base64). Produisez un message m' différent de m avec son MAC valide. Expliquez comment vous avez procédé et pourquoi votre MAC est valide. Le destinataire utilisera la fonction `pretty_print()` afin d'afficher le résultat. Faites en sorte que lorsque l'on print m', l'on obtienne une transaction bancaire qui vous verse plus d'argent qu'initialement.

3. https://docs.python.org/3/library/functions.html#pow