Chiffrement symétrique

Louiza Khati

Pour ce TP, vous aurez besoin des fichiers `script_eleve.py` et `secrets.txt`. Dans le premier fichier vous trouverez des fonctions intéressantes pour réaliser ce TP.

1 Chiffrement par bloc

1.1 Padding

Il est possible d'utiliser différents paddings et plusieurs sont d'ailleurs standardisés. Nous allons découvrir trois d'entre eux.

1. Rappeler pourquoi il est nécessaire de "padder" un message.
2. Donner une condition nécessaire sur le padding pour qu'un message soit toujours déchiffré correctement.

Utiliser le module `Crypto.Util.Padding` du package `Crypto.util` pour padder un message.

3. Définir la variable `plaintext` de type byte tel que `plaintext = b'toto'`
4. Utiliser les trois paddings suivants sur ce message pour avoir des valeurs dont la taille est un multiple de 16 octets : `'x923'`, `'iso7816'` et `'pkcs7'`.
5. Afficher le résultat et décrivez comment sont paddés les données dans les trois cas.
6. Définir la variable `full` de type byte tel que `full = b'tototototototototo'` et afficher sa valeur paddée. Analyser.

Remarque : Les attaques par oracle de padding (hors programme) sont dues à la mauvaise implémentation de la vérification du padding et non au padding lui-même.

Utiliser un chiffrement par bloc pour garantir la confidentialité des données.

1.2 AES en mode ECB

7. Écrire une fonction `AES_encrypt_ECB()` qui prend comme entrées une clé `key` et un message `plaintext` et renvoie un chiffré `ciphertext` du message avec l'AES en mode ECB. correspondant. Il faudra prendre en compte le padding (prenez celui de votre choix).
8. Écrire la fonction de déchiffrement correspondante `AES_decrypt_ECB()`.
9. Tester votre fonction en chiffrant et déchiffrant des messages de votre choix avec des clés générées aléatoirement.
   La méthode `get_random_bytes()` du package `Crypto.Random` pourra être utilisée.

10. Essayer de chiffrer des messages avec des clés de tailles différentes (15, 16, 24, 32, 64 octets). Que constatez-vous ? Est-ce dû à la primitive de chiffrement par bloc ou au mode opératoire ECB ?
11. Chiffrer le mot `m_1 = b'aaaaaaaaaaaaaaaabbbbbbbbbbbbbbbbaaaaaaaaaaaaaaaa'` et afficher le résultat sous forme hexadécimale. Que constatez-vous ?

1.3 AES en mode CBC

12. De même, écrire les fonctions `AES_encrypt_CBC()` et `AES_decrypt_CBC()`. Cette fois-ci il faudra veiller à générer un IV aléatoire et renvoyer cet IV en tant que premier bloc du chiffré.
13. Chiffrer deux fois le même message `plaintext = b'Hello'`. Que constatez-vous ? A quel paramètre est dû ce comportement ?
14. Déchiffer les messages retrouvés sur un serveur non protégé `secrets.txt`.