

TP N°3

Analyse de la vulnérabilité Padding Oracle (CBC)

1 Objectifs pédagogiques

- Comprendre le fonctionnement détaillé du mode d'opération CBC (Cipher Block Chaining).
- Analyser la faiblesse de la gestion des erreurs de padding PKCS#7.
- Implémenter une attaque par canal auxiliaire (*side-channel*) pour déchiffrer un message sans connaissance de la clé secrète.

2 Rappels théoriques

Le mode CBC déchiffre un bloc C_i en utilisant la formule suivante :

$$P_i = D_K(C_i) \oplus C_{i-1} \quad (1)$$

Où C_{i-1} représente le bloc de texte chiffré précédent (ou l'IV pour le premier bloc).

La norme PKCS#7 stipule que pour un bloc de N octets, si k octets de bourrage (*padding*) sont nécessaires pour compléter le bloc, chaque octet de bourrage doit avoir la valeur k . Exemple (bloc de 16 octets) : S'il reste 3 octets libres, le padding sera 03 03 03.

3 Environnement technique

Utiliser la bibliothèque pycryptodome pour manipuler les primitives cryptographiques.

Listing 1 - Installation de la bibliothèque

```
pip install pycryptodome
```

4 Énoncé du problème

Un serveur distant (simulé par une fonction locale) accepte des messages chiffrés en AES-128-CBC. Le serveur ne renvoie jamais le texte en clair, mais il indique si le déchiffrement a réussi ou si une erreur de formatage (padding) est survenue.

On suppose que l'attaquant peut interroger l'oracle un nombre illimité de fois.

4.1 Étape 1 : L'Oracle de déchiffrement

Utilisez la fonction `oracle(ciphertext)` fournie ci-dessous. Elle simule un serveur vulnérable :

Code de l'Oracle (Serveur)

```
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad, unpad

# Configuration du serveur (Ne pas
modifier) KEY = b" TP_CRYPT 0 _ 2026 _ #"
IV = b"INITIAL_VECTOR_1"

def oracle (
    ciphertext): try:
        iv_received = ciphertext [:16]
        data_received = ciphertext [16:]
        cipher = AES.new (KEY , AES.MODE_CBC , iv= iv_received
        )
        # unpad leve une ValueError si le padding est
        invalide unpad ( cipher.decrypt( data_received ) ,
        16)
        return True
    except ( ValueError ,
            KeyError): return False
```

4.2 Étape 2 : Interception du message

Le flux suivant (IV + Bloc chiffré) a été intercepté :

```
intercepted_ct = b"INITIAL_VECTOR_1" + b"\x0c\x82\x05\xed\x03\x13@
\x92\x07\x01\xe3\x02\x90\xec\x92\x04"
```

4.3 Étape 3 : Travail à réaliser (L'attaque)

Écrire un script Python capable de retrouver le texte clair octet par octet. Méthodologie suggérée :

1. Isoler les deux blocs (C_{i-1} et C_i).
2. Modifier l'octet n du bloc C_{i-1} pour forcer une valeur de padding spécifique.
3. Itérer les 256 valeurs possibles de cet octet jusqu'à ce que `oracle()` retourne `True`.
4. Dédurre la valeur du texte clair via la propriété du XOR :

$$P[n] = C_{i-1}^{original}[n] \oplus C_{i-1}^{test}[n] \oplus Valeur_Padding_Cible \quad (2)$$

5 Questions de réflexion

1. Pourquoi l'attaquant n'a-t-il pas besoin de connaître la KEY pour réussir ?
2. Si le message comporte 3 blocs (IV + 2 blocs de données), comment adapter l'attaque ?
3. Quelle modification simple côté serveur permettrait de rendre cette attaque impossible tout en conservant le mode CBC ?
4. Pourquoi le mode AES-GCM n'est-il pas sensible à cette attaque ?