# Padding Oracle Attack

Tipologia di attacco: Chosen Ciphertext

Diana Pamfile
Matricola:1943337
Data 15.07.2024

# 1 Chosen Ciphertext Attack

Il Chosen Ciphertext Attack (CCA) è una tipologia di attacco in cui l'attaccante ha l'accesso al testo cifrato e riesce a decifrare tale testo. Questo tipo di attacco si basa sulla possibilità di interagire con un "oracolo", un sistema che permette di decfifrare i testi cifrati per conto dell'attaccante. Un tipo specifico di chosen ciphertext attack è proprio il Padding Oracle Attack.

# 2 Padding Oracle Attack

Questo attacco sfrutta la gestione del padding nei sistemi di crittografia a chiave simmetrica, come ad esempio il Cipher Block Chaining per ottenere i testi in chiaro. Il Padding Oracle Attack è particolarmente pericoloso, in quanto può essere effettuato senza conoscere la chiave di cifratura, basandosi sulla manipolazione del padding e sulle risposte dell'oracolo.

## 2.1 Il padding

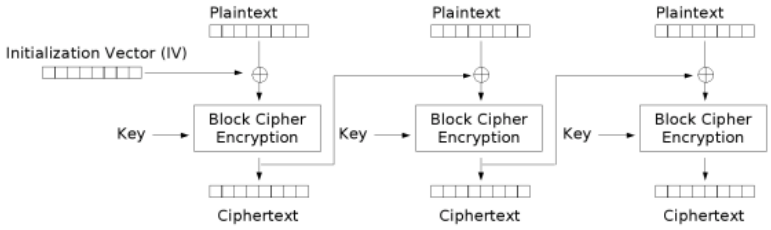
Il padding, che significa "riempimento", è una tecnica utilizzata per garantire che i dati che devono essere cifrati abbiano una lunghezza specifica. Tale tecnica si rivela cruciale nella modalità di cifratura a blocchi, in quanto se usata bene riesce ad evitare problemi di sicurezza. Nella cifratura a blocchi è richiesto che il plaintext sia esattamente della lunghezza dei blocchi di cifratura, il padding consiste nell'aggiungere dei byte supplementari per raggiungere tale dimensione.

## 2.2 L'oracolo

L'oracolo è un sistema che fornisce risposte su una determinata richiesta. In questo contesto è particolarmente importante poiché permette di rivelare se il padding del plaintext decriptato è valido oppure no. L'attaccante sfrutta le risposte dell'oracolo per riuscire a dedurre il plaintext originale senza conoscere la chiave.

# 3 Algoritmo AES

Per cifrare i testi ho implementato un algoritmo AES (Advanced Encryption Standard) in modalità CBC (cipher Block Chaining) che consente di cifrare dati di lunghezza casuale suddividendoli in blocchi.




Cipher Block Chaining (CBC) mode encryption

## 3.1 Librerie utilizzate

```
from Crypto.Random import get_random_bytes
from Crypto.Cipher import AES
import base64
```

`Crypto.Random`: utilizzata per generare byte casuali per la chiave e per l'initialization vector.
`Crypto.Cipher`: necessaria per eseguire l'algoritmo di cifratura AES.
`base64`: utilizzata per codificare il testo cifrato in formato leggibile.

## 3.2 Funzione XOR

```
def xor(b1, b2):
    return bytearray([b1[i] ^ b2[i] for i in
       range(len(b1))])
```

E' una funzione ausiliaria, utilizzata per fare lo xor tra i blocchi prima di cifrarli.

## 3.3 Classe AES in modalità CBC

All'interno di questa classe vengono inizializzati chiave e vettore di inizializzazione con bytes generati randomicamente nel caso in cui non vengano esplicitamente fornite. La funzione cuore della classe è `encrypt`, che permette di cifrare il plaintext.

```
def encrypt(self, plaintext):
        if isinstance(plaintext, str):
            plaintext = plaintext.encode('utf-8')
        plaintext = self._add_padding(plaintext)
        blocks = self.split_blocks(plaintext)
        cipher = AES.new(self.key, AES.MODE_CBC,
           self.iv)
        ciphertext = self.iv
        blocco_precedente = self.iv
```

```
        for block in blocks:
            encrypted_block =
                cipher.encrypt(xor(blocco_precedente,
                block))
            ciphertext += encrypted_block
            blocco_precedente = encrypted_block
        return
            base64.b64encode(ciphertext).decode('utf-8')
```

Questa funzione prende in input un plaintext ed effettua subito un controllo per vedere se è del tipo `str`. In caso affermativo, lo converte in byte. Ora che il plaintext è sicuramente nel formato corretto, lo passa alla funzione `add padding`, che si occupa di aggiungere padding per i dati che non sono multipli di 16 (in questo caso utilizziamo AES a 128 bit).

Ad esempio, se il testo è "Padding Oracle" e ha una lunghezza di 14 caratteri (byte), per renderlo un multiplo di 16, vanno aggiunti 2 byte (0x02, 0x02), rispettando lo standard PKCS#7. Si ottiene quindi "Padding Oracle\0x02\0x02".

I dati manipolati vengono quindi passati alla funzione `split blocks` che divide i dati in blocchi da 16 byte.
Si inizializza il ciphertext con l'iv (inizialization vector) in modo che esso si trovi sempre al blocco iniziale.
Si crea ora una istanza di un cifrario con chiave e iv della classe. Questo cifrario verrà utilizzato all'interno di un ciclo for, che scorre tutti i blocchi presenti nella lista dei blocchi suddivisi. Si prende un blocco e il blocco precedente (nella prima iterazione esso sarà iv), si effettua lo xor tra essi e successivamente si cifra il risultato, che verrà salvato nel ciphertext.
Alla fine di questa funzione, codifichiamo in base64 in quanto questa rappresentazione permette di convertre dati binari (byte) in formato testuale, e decodifichiamo in 'UTF-8' per avere un ciphertext pulito e leggibile.

# 4 Attacco

L'attaco ha diverse funzioni chiave.
La funzione `attack` è la funzione che si occupa di compiere l'attacco. Inizialmente decodifica il ciphertext dalla base64 e suddivide il ciphertext in blocchi di 16 byte. Itera su ogni blocco nella lista di blocchi suddivisi e per ogni blocco chiama la funzione `decifra blocco`, che ritorna un blocco decifrato da aggiunge al plaintext. Il padding del plaintext verrà, infine, controllato ed eliminato, per ottenere il reale plaintext.
La funzione `decifra blocco` è il fulcro dell'attacco.

```
def decifra_blocco(block):

    iv_prova = [0] * 16
    for pad_val in range(1, 16 + 1):
```

```
        padding_iv = [pad_val ^ b for b in iv_prova]

        for candidate in range(256):
            padding_iv[-pad_val] = candidate
            iv_guess = bytes(padding_iv)

            if oracolo(iv_guess, block):
                if pad_val == 1:
                    padding_iv[-2] ^= 1
                    iv_guess = bytes(padding_iv)
                    if not oracolo(iv_guess, block):
                        continue
                break
        else:
            raise Exception("Nessun padding valido
                trovato")

        iv_prova[-pad_val] = candidate ^ pad_val
    return bytes(iv_prova)
```

In questa funzione viene inizializzato un vettore di inizializzazione (iv prova) di 16 byte tutti a zero. Questo vettore verrà utilizzato per produrre un padding corretto.
Itero per ogni possibile valore di padding (da 1 a 16), creando un nuovo padding iv attraverso lo xor tra il padding corrente (ovvero pad val) e ogni byte dell' iv prova, con lo scopo di verificare se viene prodotto un padding valido.
Itero per ogno possibile byte in un range da 0 a 255, e ad ogni iterazione avrò un possibile candidato.
Modifico il byte che corrisponde al valore di padding corrente con il byte candidato.
Avviene la chiamata alla funzione oracolo, che prende un vettore ipotizzato e il blocco attuale, rispondendo True nel caso di un padding valido, False altrimenti.
Un caso importante da gestire sono i falsi positivi, che avviene quando il valore di padding è 1. Percui modifica il byte precedente e verifica di nuovo, nel caso in cui l'oracolo risponde negativamente allora continua a cercare un candidato.
Infine, viene memorizzato nell'iv prova il valore ottenuto con lo xor tra il byte candidato e il valore di padding, ottendendo il valore decifrato.
La funzione ritorna un blocco decifrato.

Un'altra funzione importante è quella dell'oracolo.

```
def oracolo(iv, ciphertext):
    try:
        oracle = AES.new(cipher.key, AES.MODE_CBC, iv)
        plaintext = oracle.decrypt(ciphertext)
        check_and_strip_padding(plaintext)
        return True
```

```
    except ValueError:
        return False
```

E' proprio attraverso le risposte dell'oracolo che possiamo tentare di decifrare un ciphertext. Viene creata una nuova istanza del cifrario AES, chiamata oracle, con la chiave del cifrario originale e IV ipotizzato. Questo oracle tenta di decifrare il blocco e lo salva in una variabile, pliantext. Successivamente controlla che il padding del plaintext sia corretto. Se il padding è valido, lo rimuove e ritorna True, False altrimenti, permettendo di individuare i byte corretti del ciphertext.

# 5 Risultati

Per trovare il possibile byte candidato l'algoritmo varia nei tentativi. Di seguito possiamo osservare i tentativi per trovare ogni byte della stringa "Questa è una stringa super segreta" cifrata.

```
Testo Cifrato:
   pYI/vhXbcTKzaICW+1Pzdx6qyLnmJJms9WOnezSDuvtt7HRC8
+L6uZeHzjC56NUQVIw5roIQm+/qMRtRfE9PRw==

#Lista Candidati:

#Blocco 1:
[117, 113, 35, 101, 107, 115, 39, 160, 202, 42, 106,
   120, 126, 107, 122, 65]

#Blocco 2:
[115, 101, 102, 119, 37, 116, 98, 120, 124, 121, 43,
   109, 106, 96, 102, 98]

#Blocco 3:
[12, 15, 14, 9, 8, 11, 10, 5, 4, 7, 6, 1, 0, 111,
   123, 117]
```

Per quanto riguarda la decifratura dei blocchi, essa inizia dall'ultimo byte scorrendo verso il primo. Avviene con successo e produce per ogni blocco:

```
Blocco 1: b'Questa \xc3\xa8 una st'
Blocco 2: b'ringa super segr'
Blocco 3: b'eta\r\r\r\r\r\r\r\r\r\r\r\r\r'
```

Blocchi che verranno concatenati in una variabile "plaintext". Rimuovendo il padding e controllando che esso sia corretto, l'attacco produce come output:

```
Testo Cifrato:
   pYI/vhXbcTKzaICW+1Pzdx6qyLnmJJms9WOnezSDuvtt7HRC8
+L6uZeHzjC56NUQVIw5roIQm+/qMRtRfE9PRw==
```

```
3 #Messaggio segreto: Questa e' una stringa super
     segreta
```

# 6 Contromisure

Le contromisure per difendersi da questo attacco possono essere:

- Risposta uniforme ai messaggi di errore, evitando di rivelare informazioni importanti attraverso la differenza nei messaggi di errore.
- Randomizzare gli IV, in quanto se non si sa la posizione di esso è più difficile da individuare.
- Aggiunta di autenticazione dei messaggi, MAC. Si può utilizzare Hash-Based Message Autentication Code, ovvero una tecnica che combina una chiave segreta con una funzione di hash per produrre un codice di autenticazio. Questo codice si aggiunge al messaggio originale prima della cifratura e viene validato dopo la decifratura, permettendo di verificare che il messaggio non sia stato alterato.