NSUCRYPTO 2020: CPA Game

Ioan Dragomir[1], Gabriel Tulba-Lecu[2], and Mircea-Costin Preoteasa[3]

[1] ioandr@gomir.pw – Technical University of Cluj-Napoca
[2] gabi_tulba_lecu@yahoo.com – Polytechnic Univeristy of Bucharest
[3] mircea_costin84@yahoo.com – Polytechnic Univeristy of Bucharest

Table of Contents

1 Problem summary

Alice and Victor play a game with the following setup and rules:

1. Alice has an block cipher in CBC mode, with an encryption function completely unknown to Victor (thus the existence of a key becomes irrelevant), as well as a secret value $b \in \{0, 1\}$.

2. Victor must determine the secret value b.

3. Victor can submit an indefinite number of queries to Alice, each taking the form of two messages $m_{i,0}$, $m_{i,1}$ of the same length.

3. After each query, Alice responds with $e_i = encrypt(m_{i,b})$, or, intuitively, either encrypts the first message or the second message Victor specified, based on the secret value b.

4. From one query to another, Alice's CBC block cipher state is preserved. For instance, the IV for the third query is the result of the second query.

The goal is to find an winning strategy for Victor to determine b correctly.

2 Solution

As a convention, we will only send queries of the same length as the block size, so that we have a neat correspondence between queries and blocks.

The first goal is to get a known input-output pair for the encryption function used in Alice's block cipher. For each block ≥ 2, we know its IV before sending its plaintext, so we can force any input value x into the encryption function by sending the expected IV xor x as the plaintext. The query output is exactly the encryption function output.

Therefore we can obtain one $(x, encrypt(x))$ pair using a single query, but not using the first query, whose IV we don't know.

Using a known input-output pair, we can submit another query where we reproduce this pair, but only in one of the submitted messages. If we get the same known output, we know the message used by Alice corresponds to the one we set to the known input. Otherwise it's the other message. For example, if we force the known input through $m_{i,0}$ and send a different message to $m_{i,1}$, and then we see the known output, we know $b = 0$; if it's a different output, then $b = 1$.

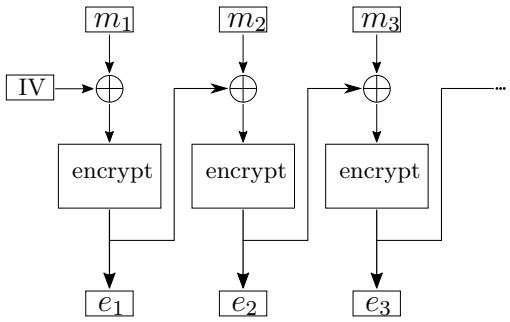
Considering the encryption function is injective, the distinction between $encrypt(0)$ and $encrypt(e_2 \oplus m_{3,1})$ should give this method an optimal 100% success rate, as there would never be any confusion as to which plaintext was chosen for the third block.

2.1 Algorithm

```
m10 = any_valid_plaintext()
m11 = any_valid_plaintext()
e1 = query(m10, m11) # Skip the first query using any valid inputs
e2 = query(e1, e1)   # Obtain encrypt(0)
m31 = any_valid_plaintext()
assert m31 != e2
e3 = query(e2, m31)  # if e2 will be chosen, we will get encrypt(0) again
b = 0 if e3 == e2 else 1
```

3 Intuition

The three queries and their quirks can be better understood by referring to the CBC block cipher mode diagram:



As per the game's rules, we know e_1 before sending m_2, e_2 before m_3, etc.

By setting $m_2 \leftarrow e_1$, they cancel out inside the second xor and the input to the second encrypt is zero. In this way we find the value of $e_2 = encrypt(0)$.

We use the same xor cancelling trick to set only one of $m_{3,0}$, $m_{3,1}$ to e_2 so that, based on whether Alice picks $m_{3,0}$ or $m_{3,1}$ for m_3, e_3 might also be $encrypt(0)$. Whether or not the 3rd ciphertext also has that known $encrypt(0)$ value helps us determine which of the two plaintexts Alice used, and so the secret value of b.