Tutorial 2

Shen Jiamin

Question 1

You have intercepted two ciphertexts C_1, C_2 generated by a stream cipher using the same secret key. The first 4 bits of the ciphertext form the IV.

$$C_1 = 0111\ 11011011$$
$$C_2 = 0111\ 00101011$$

You know that the plaintext must be among the following 4 sequences:

$$P_1 = 00000000, P_2 = 11111111, P_3 = 00001111, P_4 = 11000011$$

What are the possible plaintexts of C_1 and C_2?

Answer The important observation here is that the same IV is used for both ciphertexts, which allows us to exploit the properties of the XOR operation.

Stream ciphers encrypt plaintext by XORing it with a keystream generated from the secret key and IV, i.e.

$$C = P \oplus \mathrm{PRG}(K, IV).$$

Since the same IV is used, the keystream for both ciphertexts is identical. Thus

$$C_1 \oplus C_2 = (P_1 \oplus \mathrm{PRG}(K, IV)) \oplus (P_2 \oplus \mathrm{PRG}(K, IV)) = P_1 \oplus P_2 = 11110000.$$

$C_1 \oplus C_1 = 11110000.$

$\oplus$	P_1	P_2	P_3	P_4
P_1	00000000	11111111	00001111	11000011
P_2	11111111	00000000	11110000	00111100
P_3	00001111	11110000	00000000	11001100
P_4	11000011	00111100	11001100	00000000

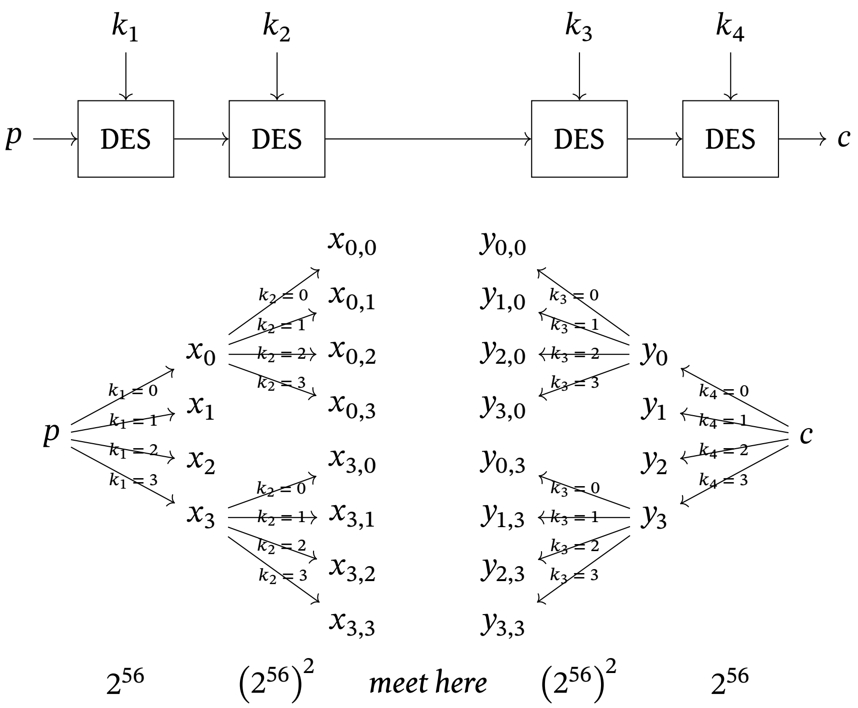
So C_1 and C_2 are P_2 and P_3 in some order.

Notice that the attack is applicable because: (a) a stream cipher is employed; (b) the same secret key and IV are used for generating the two ciphertexts.

Question 2: Meet-in-the-middle

Instead of applying DES three times, Bob wants to apply it four times with 4 different 56-bit keys k_1, k_2, k_3 and k_4. By using meet-in-the-middle attack, what is the of number of cryptographic operations (including encryption and decryption) required for known-plaintext attack? Give your answer in the form of 2^k and approximation (within a multiplicative factor of 2) is suffice. *(Remark: Lecture note mentioned that there is a more efficient meet-in-the-middle attack. Here, the simple meet-in-the-middle in the lecture note is suffice).*

Answer



One can exhaustively enumerate k_1, and for each k_1, exhaustively search all k_2. So, the number of encryptions will be (number of encryptions using k_1) + (number of encryptions using k_2) $= 2^{56} + 2^{112} \approx 2^{112}$. We need the same number of operations for k_3 and k_4. So total is approximately $2^{112} \times 2 = 2^{113}$.

Demo A running example of the attack using a tiny cipher:
`https://gist.github.com/shnjmn/28b42c130eeb081ac1be78177ade6996#file-qn2-py`

Remarks

- If applied 3 times, meet-in-the-middle needs approximately 2^{112} operations. So, increase from 3 to 4 times only increase the "difficulty" from 2^{112} to 2^{113} (or "bit-strength" from 112 to 113).
- What about applying $2t-1$ times vs applying it $2t$ times, when $t = 3, 4, \ldots$?

$$\begin{array}{rcl} 2t-1 \text{ times} & \cdots & 2^{56t} \text{ operations} \\ 2t \text{ times} & \cdots & 2^{56t+1} \text{ operations} \end{array}$$

Question 3

Alice sends instructions to Bob daily using mobile phone in the following way. Each instruction is represented as ASCII string, and follows the format:

`action:date`

The date is the 8-byte "`dd/mm/yy`" format.
Actions are "`buy`", "`sell`", "`sell_everything`", and "`hold_and_see`".

Example

```
buy:02/01/22
sell_everything:03/01/22
```

The message will be using AES under some mode-of-operation. The ciphertext, which is a binary string, is then converted to a text message using some tools, e.g. `uuencode`. The text message is then sent to Bob using SMS.

The mobile phone, after each re-start, will set the IV to be the string of all zeros, and then increases it by one (i.e. treat it as binary number and increment by 1, similar to the CTR mode) for every new encryption. An attacker is able to sniff the SMS channel between Alice and Bob.

Let us consider two settings:

(a) Suppose the mode-of-operation is CBC mode. What information regarding the plaintext can be inferred by the attacker? Note that under CBC mode, message padding is required if the message length is not multiples of 16-byte. In this question, the padding is simply done by appending bytes of zeros at the end of the message string.

(b) Suppose the mode-of-operation is CTR mode. What information regarding the plaintext can be inferred by the attacker?

(Remark: The question doesn't state precisely what type of info can be sniffed from the communication channel. From the context, it should be clear that the sniffed data are IV and the ciphertext.)

Demo Messages and ciphertexts used for demonstration are generated using the following script: https://gist.github.com/shnjmn/28b42c130eeb081ac1be78177ade6996#file-qn3-py

Answer The block size for AES is 16 bytes. The message (in plaintext) length can be inferred from the action and date format.

- `buy:dd/mm/yy` (12 bytes)
- `sell:dd/mm/yy` (13 bytes)
- `sell_everything:dd/mm/yy` (24 bytes)
- `hold_and_see:dd/mm/yy` (21 bytes)

(a) **(Ciphertext length)** "`buy`" and "`sell`" instructions after padding will take 1 blocks; while "`sell_everything`" and "`hold_and_see`" instructions will take 2 blocks. Simply from the length of the ciphertext, the attacker can tell whether the action is in {`sell`, `buy`} or in {sell_everything, hold_and_see}

IV	Ciphertext	Action Inferred
0	63bc7932...2935655a	{sell, buy}
0	4016231f...c8a3f888	{sell, buy}
1	2f8f9261...922bc452 : e2618598...ebcb1f86	{sell_everything, hold_and_see}
0	717554c9...7b163ad8 : 5f7c9a17...fefc8815	{sell_everything, hold_and_see}

(IV reuse) Due to the re-start and the way IV is generated, attacker can get a few ciphertexts with the same IV. (Note that IV is sent in clear.) The attacker may collect multiple pairs of IV and ciphertext. Among those 2-block ciphertexts, i.e. (v, c_1, c_2), having the same IV v, the attacker can count how many of them have the same c_1. If there are multiple ciphertexts with the same c_1, the attacker can infer that their plaintext also share the same 16-byte prefix, which must be "sell_everything:". And those having different c_1 must correspond to "hold_and_see:dd/".

IV	Ciphertext	Action Inferred
1	458b5e24ce...1a6707db6b : 728e212e43...b8886b6962	hold_and_see
1	2f8f92614b...a9922bc452 : d4ce148bff...7b3f792677	sell_everything
1	2f8f92614b...a9922bc452 : 4f7e533a15...279d87bded	sell_everything
1	2f8f92614b...a9922bc452 : 4f3c4d30b2...e41925924a	sell_everything
2	b4d1ee5b1e...dc67169b13 : c3c4409f4a...c779cd9a52	hold_and_see
2	4a389fd40f...a402686a4d : ea6c970fc2...6b97a28f77	hold_and_see
2	12b9ddb33a...35d40817ef : 2490a4cd3a...680f69e8d9	hold_and_see
2	6ee87ea934...70173ff15d : f8dd64ccdd...9689c4476d	hold_and_see
2	0bf7555715...956da692a7 : ce9675758d...a876684d0b	hold_and_see
2	bd0ee3cbd6...e34c357ba7 : 2dc84a3e30...294846fc04	sell_everything
2	bd0ee3cbd6...e34c357ba7 : cc16fc3a03...9b975d6c12	sell_everything
2	bd0ee3cbd6...e34c357ba7 : db3188cbcd...3b0b3a0a68	sell_everything
3	e2197deccd...10576b3800 : 18273a6710...1385ebc6cf	hold_and_see
3	ee12b819f8...9584a0183a : 076c93674b...3b67f90fac	sell_everything

The above works because, due to same IV, the ciphertext is the same iff the plaintext is the same. "sell_everything:" is 16 bytes and fit into a block, and it appears as first block in the plaintext. By CBC, this 1st block of plaintext will be encrypted to the same ciphertext. On the contrary, "hold_and_see:" is less than 16 bytes and will be followed by the "day" that likely to be different. So the ciphertext likely to be different.

There could be some cases that Alice sent multiple "hold_and_see" messages with the same "day". Due to the same "day" in the date, the first block of "hold_and_see:dd/" are the same for two different captures. In such cases, attacker would observe two groups of ciphertexts having the same first block. Likely that the larger one is "sell_everything:".

Remark:

- Note that the 3rd block of ciphertext would be different.
- Unfortunately, the above can't apply to "buy" and "sell".
- Whether attacker know the date or not would not affect the outcome.

(b) **(Ciphertext length)** CTR mode keeps the length of plaintext and ciphertext the same. So, the attacker can immediately know the exact action being performed based on the ciphertext length.

IV	Ciphertext	Action Inferred
0	`2b8790a2c04e014e0647326591`	`sell`
0	`3a9785f4ca4d1f510e502f62`	`buy`
1	`70edb6a23fd3d5f78151b6d118dc9942 c7a4859b71646b16`	`sell_everything`
0	`308d90aaa51f5e05690c78329ed77175 33b0f5fc55`	`hold_and_see`

(IV reuse) If two messages are encrypted with the same IV, the XOR of ciphertexts will reveal the XOR of the corresponding plaintexts.

For example, given two ciphertexts:

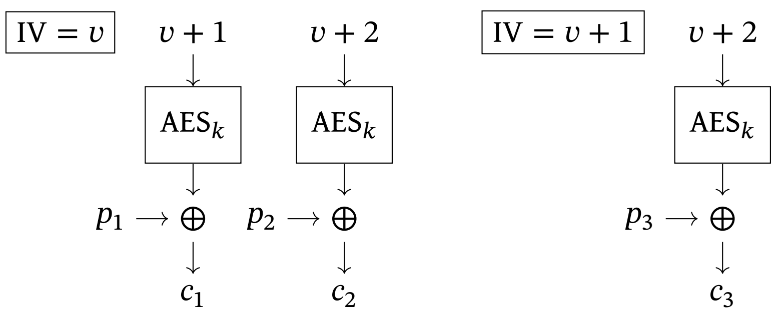
	IV	Ciphertext
m_1	3	`4277681b7e40bbea18a68499`
m_2	3	`53677d4d1116e2bf52f0c2c4b9fdccda f8798d2929a11b48`

Observing the lengths of the ciphertexts, we see that m_1 has 12 bytes of ciphertext, and m_2 has 24 bytes of ciphertext. Thus, m_1 is a "buy" instruction, and m_2 is a "`sell_everything`" instruction. Therefore, we know that the first 12 bytes of m_2 are "`sell_everyth`".

Since the two ciphertexts are encrypted with the same IV, we have $m_1 \oplus m_2 = c_1 \oplus c_2$.[1]

$$\begin{aligned}
m_1 \oplus \texttt{"sell_everyth"} &= \texttt{4277681b7e40bbea18a68499} \oplus \texttt{53677d4d1116e2bf52f0c2c4} \\
&= \texttt{111015566f5659554a56465d} \\
m_1 &= \texttt{111015566f5659554a56465d} \oplus \texttt{"sell_everyth"} \\
&= \texttt{111015566f5659554a56465d} \oplus \texttt{73656c6c5f65766572797468} \\
&= \texttt{6275793a30332f30382f3235} = \texttt{"buy:03/08/25"}
\end{aligned}$$

(Counter overlap) Suppose two instructions are consecutively encrypted, and the first ciphertext consists of 3 blocks (including the IV). Let the first ciphertext be (v, c_1, c_2) and the corresponding plaintext is $p_1 \| p_2$. Let the third ciphertext be $(v+1, c_3)$ or $(v+1, c_3, c_4)$ depending its size, and the corresponding plaintext of c_3 is p_3.



[1] In this case, when XORing two bit-strings of different lengths, we truncate the longer one to the length of the shorter one.

Note that by CTR mode, $c_2 = p_2 \oplus \text{AES}(v + 2)$, and $c_3 = p_3 \oplus \text{AES}(v + 2)$. That is, the counter is the same. So, similar to the "zebra" example in lecture note, the attacker can obtain $p_2 \oplus p_3 = c_2 \oplus c_3$.

For example, consider the following tiny capture:

	IV	Ciphertext
m_3	0	`2b8790a2a51b46044406693fcd892260 33baf5fe589991a7`
m_4	1	`6be7b6aa3fd7cdf6ac5ba7dc4b82c657 c7ad85997c`

Observing the lengths of the ciphertexts, we see that m_3 has 24 bytes of ciphertext, and m_4 has 21 bytes of ciphertext. Thus, m_3 is a "`sell_everything`" instruction, and m_4 is a "`hold_and_see`" instruction. That is

- m_3 is `"sell_everything:aa/bb/cc"`;
- m_4 is `"hold_and_see:xx/yy/zz"`.

From how CTR mode works, we know for m_3:

$$\begin{aligned} \text{AES}_k(1) \oplus \texttt{"sell\_everything:"} &= \texttt{2b8790a2a51b46044406693fcd892260} \\ \text{AES}_k(2) \oplus \texttt{"aa/bb/cc"} &= \texttt{33baf5fe589991a7} \qquad (1) \end{aligned}$$

And for m_4:

$$\begin{aligned} \text{AES}_k(2) \oplus \texttt{"hold\_and\_see:xx/"} &= \texttt{6be7b6aa3fd7cdf6ac5ba7dc4b82c657} \qquad (2) \\ \text{AES}_k(3) \oplus \texttt{"yy/zz"} &= \texttt{c7ad85997c} \end{aligned}$$

By XORing (1) and (2), we obtain:

$$\begin{aligned} \texttt{"hold\_and"} \oplus \texttt{"aa/bb/cc"} &= \texttt{6be7b6aa3fd7cdf6} \oplus \texttt{33baf5fe589991a7} \\ &= \texttt{585d4354674e5c51} \\ \texttt{"aa/bb/cc"} &= \texttt{585d4354674e5c51} \oplus \texttt{"hold\_and"} \\ &= \texttt{585d4354674e5c51} \oplus \texttt{686f6c645f616e64} \\ &= \texttt{30322f30382f3235} = \texttt{"02/08/25"} \end{aligned}$$

In that way, we can recover the plaintext of m_3, which is `"sell_everything:02/08/25"`.

Question 4: (Padding Oracle)

Consider the padding oracle attack described in the lecture note. Suppose the attacker knows that the 16-byte plaintext is the sequence

$$\langle b_1, b_2, b_3, b_4, b_5, b_6, b_7, b_8, b_9, \mathtt{00}, \mathtt{00}, \mathtt{FF}, \mathtt{04}, \mathtt{04}, \mathtt{04}, \mathtt{04} \rangle\,,$$

where the numbers are in hexadecimal representation, and the attacker does not know the value of the b_i's. Describe how the attacker determine the value of b_9. In particular, describe how to decide the value of v in the lecture note.

Answer To launch padding oracle attack, the attacker knows a pair of valid IV v and ciphertext c. Since the question also assumes that the attacker knows part of the plaintext, we know

$$v \oplus \mathrm{AES}_k^{-1}(c) = \langle b_1, b_2, b_3, b_4, b_5, b_6, b_7, b_8, b_9, \mathtt{00}, \mathtt{00}, \mathtt{FF}, \mathtt{04}, \mathtt{04}, \mathtt{04}, \mathtt{04} \rangle\,.$$

There are 8 bytes from b_9 to the end of the block. To find b_9, the attacker need to manipulate the IV so that the decryption of the block has a valid padding of 8 bytes. That is, we need to find a v' such that

$$v' \oplus \mathrm{AES}_k^{-1}(c) = \langle b_1, b_2, b_3, b_4, b_5, b_6, b_7, b_8, \mathtt{08}, \mathtt{08}, \mathtt{08}, \mathtt{08}, \mathtt{08}, \mathtt{08}, \mathtt{08}, \mathtt{08} \rangle\,.$$

XOR the two equations, we have

$$v' \oplus v = \langle \mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}, t, \mathtt{08}, \mathtt{08}, \mathtt{F7}, \mathtt{0C}, \mathtt{0C}, \mathtt{0C}, \mathtt{0C} \rangle\,,$$

where $t = b_9 \oplus 08$ is the only unknown.

Thus we can find t by brute-forcing all possible values.

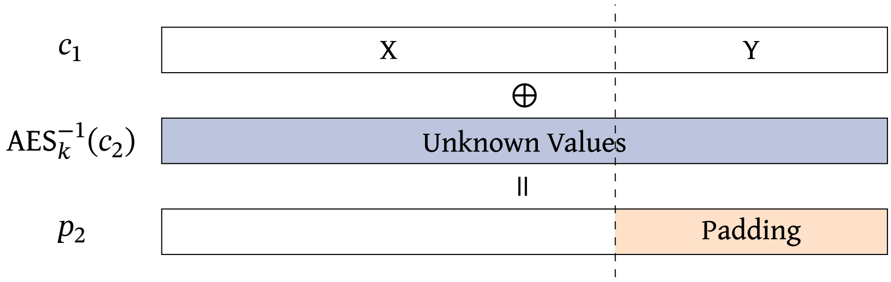
for $t = 0$ to FF **do**
  let $v = IV \oplus \langle 0, 0, 0, 0, 0, 0, 0, 0, t, \mathtt{08}, \mathtt{08}, \mathtt{F7}, \mathtt{0C}, \mathtt{0C}, \mathtt{0C}, \mathtt{0C} \rangle$.
  send $v \| c$ to oracle.
  if yes **then**
    return $t \oplus 08$.
  end if
end for

Question 5

The lecture notes assume the attacker knows the number of padding bytes. Now, consider a scenario where this information is unknown. Suppose the attacker has access to the one-block IV and a two-block ciphertext, but does not know how many padding bytes are present. Describe a method the attacker can use to determine the number of padding bytes, using as few oracle queries as possible.

Answer Unlike the lecture note, here, we assume the ciphertext (including the IV) is three blocks.

Let the IV and ciphertext be denoted as v and $c_1 \| c_2$, respectively. To find about c_2, the attacker should modify c_1 and query the oracle (instead of modifying the IV).



Imagine flipping a byte in c_1. If the flipped byte lies in region X, the modification will not affect the padding, and the oracle will return "valid." If the flipped byte lies in region Y, the modification will alter a padding byte, and the oracle will return "invalid." The core idea is therefore to identify the boundary between valid and invalid responses.

Method 1: Assume that the length is i, and test it. Test one by one starting from $1, \ldots, 16$.

for $i = 1$ to 16 **do**
  $c_1[i] \leftarrow c_1[i] \oplus \mathrm{FF}$.
  send $c_1 \| c_2$ to oracle.
  if no **then**
    return $17 - i$
  end if
end for

Method 2: Improved version. Use binary search to find the i that goes from valid to invalid.

(Optional remark: can't do better than binary search. There are 16 possible outcomes, and every query would eliminate at most halve.)

Question 6: Padding oracle attack is practical

Search the CVE database for a known vulnerability that is based on AES-CBC padding oracle attack. (Note: there are also padding oracle attack on other encryption scheme).

Answer There are many. E.g. CVE-2023-2197.

> HashiCorp Vault Enterprise 1.13.0 up to 1.13.1 is vulnerable to a padding oracle attack when using an HSM in conjunction with the CKM_AES_CBC_PAD or CKM_AES_CBC encryption mechanisms. An attacker with privileges to modify storage and restart Vault may be able to intercept or modify cipher text in order to derive Vault's root key. Fixed in 1.13.2