Differential cryptanalysis

Hao CHENG

April 6, 2018

1 On the AES

1.1 Inner Workings

1.1.1

32-bits column given to me is `a = {0xde 0x28 0x4c 0x49}`
After `SubBytes`, the result is `SubBytes(a) = {0x1d 0x34 0x29 0x3b}`
Then after the `MixColumns`, the result is:

$$b = \texttt{MixColunms(SubBytes(a))} = \{\texttt{0x74 0x35 0x36 0x4c}\}$$

1.1.2

Because $b \oplus b^* = \Delta$, so $b^* = b \oplus \Delta$. We could simply get the value of b^*:

$$b^* = \{\texttt{0xc4 0x84 0x4f 0x5b}\}$$

Then I use `inverse MixColunms` and `inverse SubBytes`, getting the final result:

$$a^* = \texttt{SubBytes}^{-1}(\texttt{MixColumns}^{-1}(b^*)) = \{\texttt{0x1c 0x37 0xf4 0x53}\}$$

1.2 Importance of ShiftRow

1.2.1

Without `ShiftRow`, each column will be independent from others. And assuming there isn't the key derivation. We could separate the 128-bit-key to four 32-bit-key for four columns. So if we choose the brute force to attack the cipher, the search space is $4 * 2^{32} = 2^{34}$ instead of 2^{128}. So actually using one pair could attack this cipher.

1.2.2

In the last question, the key search space is 2^{34}. So the expectation is 2^{33}. For the time complexity, it will be $O(2^{33} * T)$, which T is the time of single AES.
For the space complexity, it is constant O(n). Because we can try key one by one, so the space only contains one key, one pair of plaintext and ciphertext, some local states of AES.

1.2.3

Brute force attack. On average the time of it is 2^{33}.

1.3 Importance of MixColumns

For different number of S-box in the single column, it will lead to different amount of active S-box. Only 1 active S-box in the column, it must lead to 4 active S-boxes. For 2 active S-boxes, it will lead to 3 or 4 active S-boxes. And for 3 acive S-boxes, it will lead to 2, 3 or 4 active S-boxes. For 4 active s-boxes, it could lead to any amount.

1.3.1

$4 \rightarrow 2 \rightarrow 5$ is not possible.
There are some conditions of 4 active S-boxes:(4), (3,1), (2,2), (2,1,1),(1,1,1,1). It couldn't be (2,1,1),(1,1,1,1), these conditions must lead to more than 2 active S-boxes, because they have more than 2 columns. If it is (3,1), it will lead to more than 4 active S-boxes, because there is only 1 active box in one column. If it is (2,2), as mentioned before, it will lead to(3,3) (3,4) (4,4), all of them are more than 2 active S-boxes.
Only (4) is possible, after first `MixColumn`, there will be 2 active S-boxes in a single column. And after the `ShiftRow`, these 2 active S-boxes will separate to different columns. So, finally the result is 8 active S-boxes.All them are impossible.

1.3.2

$4 \rightarrow 1 \rightarrow 4$ is possible.
Because if 4 active S-boxes are located in the single column, it's possible that leading to 1 active box. And if there is only 1 active box, after the `MixColunm`, it must be 4 active S-boxes.

1.3.3

$5 \rightarrow 5 \rightarrow 5$ is not possible.
There are some conditions of 5 active S-boxes: (4,1), (3,2),(3,1,1), (2,2,1), (2,1,1,1). Let's analyze each condition.
(2,1,1,1) and (3,1,1) are impossible to lead to 5 active S-boxes. For 1 S-box in single column, it must lead to 4 active S-boxes.(2,2,1) also will lead to more than 5 S-boxes.
(3,2) is possible to lead to (2,3). But after the `ShiftRow`, 3 S-boxes in single column will separate to 3 different columns. There must be one column has only 1 active S-box and finally will lead to 4 active S-boxes. In addition, adding the active S-boxes leaded by other 2 columns, they must be more than 5. So it is impossible.
(4,1) is possible to lead to (1,4). But after the `shiftRow`, 4 S-boxes in single column will separate to 4 different columns. Similarly with last condition, finally the number of active S-boxes must more than 5.
So all of these conditions are impossible.

1.4 Double AES

The source code is *Attack_Double_AES.cpp*
I use meet-in-the-middle method to attack the double AES. Like method introduced in the slide to attack the double DES. I guess K1 to build a sorted table $AES_{K1}(P) = A(K1)$ and guess K2 to build a sorted table $AES^{-1}_{K2}(C) = B(K2)$. Just find $A(K1) = B(K2)$. The 256-bit key could be found using two 128-bit known plaintext/ciphertext pairs in just 2^{128} time and memory. For this question, because the key is consist of 104-bit **0** and 24-bit **key**. So the time and memory is just 2^{24}.
In my program, I use $map < ciphertext, K1 >$ to store the $A(K1)$, and then every time try different K2 get the $B(K2)$ to find if there is the same ciphertext in map. If there is the same ciphertext, then the value of this ciphertext is the K1 that I need. In addition, the K2 is also what I need. K = (K1,K2) is the whole key.
But when running it, there are always some problems of memory assignments making it fail to run.

2 Differential Cryptanalysis of the DES

2.1

Firstly constructing the difference distribution table of S_1 with using the x, δ_{in} and δ_{out}. With differential cryptanalysis, we could get the possible pairs: (0x2f, 0x36) and (0x36,0x2f). After XOR with x = 0x3a, we could get the possible k = 0x15, 0x0c.
For the entropy, according to the formula: $H(K) = \Sigma p_i * \log_2(1/p_i) = 2*0.5*\log_2(2) = 1[bit]$

2.2

Key = b28454448ca4c22
The input of F_7 is $C_{iR} \oplus C^*_{iR}$. And the output difference of F_7 is $\Delta_R \oplus R_{iL} \oplus R^*_{iL}$. And then put the output of F_7 through the P^{-1} to get the output of S-box. Now we could get the possible pairs of plaintext and ciphertext.
Now getting the possible keys to do that we need actual inputs of F_7. The input of F_7 are both c_{iR} and c^*_{iR}, we run those inputs through Expansion permutation and XOR them with previously discovered inputs of S-boxes.For each S-box, we add corresponding partial subkeys. Finally, we repeat this procedure for another pair creating new list of sets, where each corresponds to S-box. In the end, we perform an intersection for each S-box. Which gives us a list of sets, which are now smaller.
Eventually, we have a list of possible subkeys for round 7. For each possible subkey, we reverse the key scheduling and we end up with possible partial master key. We have 48 bits out of 56. We can now bruteforce the rest of bits and retrieve the full key.