TP AES Report

Martinez Anthony, Vallet Nicolas

May 1, 2023

1 Introduction

The aim of this TP is to performed the square attack on $3^{1/2}$ round of AES.

To compile the program *make*, and to remove all executable *make clean*.
Utility of each executables :
testRoundKey, check if our function to reverse the key works
testXTimeFunction, test the value of Xtime function
keyedFunction, use the distinguisher to retrieve if the oracle is using the AES or a random permutation
squareAttackAES, retrieve the correct key after 3 and half round.

2 Exercice n°1 : Warming up

2.1 Q.1

The objective of the function *xtime* is to multiply the polynome p given as an input by X modulus $X^8 + X^4 + X^3 + X + 1$.
It works as follow :

```
/*
 * Constant-time ''broadcast-based'' multiplication by $a$ in $F_2[X]/X^8 +
  * X^4 + X^3 + X + 1$
 * Function used to multiply p by x over the group F_2[X]/X^8 + X^4 + X^3 +
 * X + 1
 */
uint8_t xtime(uint8_t p)
{
    //Get m equal to the coefficient of degree 7 of p.Therefore, m = 1
    //if the coefficient of degree 7 of p is equal to 1 or m = 0
    //otherwhise
    uint8_t m = p >> 7;
    //Switch m from 0 to 1 or 1 to 0 according to its previous value
    m ^= 1;
    //Set m equal to 255 if the coefficient of degree 7 of p is 1
    //or 0 otherwhise
    m -= 1;
    //Set m to be equal to 0x1B -> 0001 1011 which corresponds to
    //the polynome X^4 + X^3 + X + 1 if the coefficient of degree 7
    // of p is 1 or 0 otherwhise
    m &= 0x1B;
    //Proceed to the multiplication by shifting every bits of p to
    //the left and apply the modulus by subtracting m, wich is equivalent
    //to xor with m as the coefficient are only 0 or 1
    return ((p << 1) ^ m);
}
```

In order to realise the variant of *xtime*, we change the equation **m &= 0x1B** to **m &= 0x7B**.
The new function *xtimeVariant* is available in the file *aes-128_enc.c*.

2.2 Q.2

The function *prev_aes128_round_key* is available in the program *aes-128_enc.c*.
The correcteness of the function is tested in the file *testRoundKey.c* with the key
k = 2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c availabe in the appendix A of the AES standard.
We try our program by running first *next_aes128_round_key* then *prev_aes128_round_key* and we obtain k again. Therefore we can conclude that our function is working properly.

2.3 Q.3

The keyed function $\mathcal{F}$ has been implemented in the file *oracle.c* as *keyedFunction*.
If $k_1 = k_2$, $\mathcal{F}(k_1||k_2, x) = \mathcal{E}(k_1, x) \oplus \mathcal{E}(k_2, x) = AES_3(k_1, x) \oplus AES_3(k_2, x) = 0 \ \forall x$ because AES is deterministic.
$\bigoplus_{i=0}^{255} \mathcal{F}(k_1||k_2, p_i) = \bigoplus_{i=0}^{255}(\mathcal{E}(k_1, p_i) \oplus \mathcal{E}(k_2, p_i)) = \bigoplus_{i=0}^{255}(AES_3(k_1, p_i) \oplus AES_3(k_2, p_i)) = \bigoplus_{i=0}^{255} AES_3(k_1, p_i) \oplus \bigoplus_{i=0}^{255} AES_3(k_2, p_i) = 0$.
As we can see above, the 3-round square distinguisher works for $\mathcal{F}$.
To test this property, we create a program that will randomly choose between the keyed function and a random permutation to cipher all values from 0 to 255 and use the square attack to determine if the values have been ciphered with a random permutation or the keyed function.
The test program is *keyedFunction.c*.
We run the program several times and it was able to determine each time if it was the keyed function or the random permutation that was used. Therefore it confirms the property.

3 Exercice n°2 : Key-recovery attack for $3^{1/2}$-round AES

3.1 Q.1

The attack is implemented in the file *squareAttackAES.c*. To retrieve the key and store each guessed byte, we are using a linked list implemented in the file *struct.h*. All the functions for the list are in *struct_function.c*.
The algorithm is approximately as follow :

```
byte_of_key_guessed[16] //each case of our tab lead to a linked list with
                        //the byte that can be the key

while(there is still some byte in our
        byte_of_key_guessed that's are not unique)
{
    new A-set //compute the new A-set
    sum_xor = 0 //size of 1 byte
    for( index from 0 to 15 ) //[0, ..., 15] index of the byte guessed
    {
        set_all_byte_not_found_for_this_A-set(byte_of_key_guessed, index)
        for (key from 0 to 256 ) //all possible value byte for the key
        {
            for each set in our A-set
            {
                res = reverse_demi_turn(set, key)
                sum_xor ^= res
            }
            if sum_xor == 0
            {
                if it's the first A-set
                {
                    add_byte_in_list(byte_of_key_guessed, index, key)
                } else
                {
                    set_byte_works_for_A-set(byte_of_key_guessed, index, key)
                }
            }
        }
        remove_all_byte_not_found_for_this_A-set(byte_of_key_guessed, index)
    }

}
```

We test it with several random keys :

Intial key	Key found by the program
101 9 229 41 173 1 185 30 221 44 38 38 20 254 48 131	101 9 229 41 173 1 185 30 221 44 38 38 20 254 48 131
95 242 43 3 106 106 66 119 100 45 66 19 155 148 224 41	95 242 43 3 106 106 66 119 100 45 66 19 155 148 224 41
176 208 206 10 64 1 171 166 7 132 154 86 156 5 9 235	176 208 206 10 64 1 171 166 7 132 154 86 156 5 9 235
206 5 75 89 78 168 68 37 110 53 155 20 6 123 123 64	206 5 75 89 78 168 68 37 110 53 155 20 6 123 123 64
28 241 107 179 210 158 67 107 184 28 167 171 251 65 3 100	28 241 107 179 210 158 67 107 184 28 167 171 251 65 3 100

As we can see in the previous table, for each initial key, the program is able to find the good key. Therefore, we can conclude that our program is working properly.

3.2 Q.2

We have modified the MixColumns step in the function *aes128_enc.c* by repalcing the *xtime* function by *xtimeVariant* realised in the first exercice, and we've seen that's the attack still works. The modified file is located in the directory **testChangingFunction**.
We've also changed the MDS matrix used in the mixColumn and the SBox and we've seen that the attack still works.
The modified files are located respectively in the directories **testChangingMDSMatrix** and **testChangingSBox**