CS641A Assignment-5

Sherlocked

Bholeshwar Khurana (170214) **Yaghyesh Chouhan (170813)** **Sarthak Dubey (180674)**

1 How we reached the cipher text:

1. We were in a passage which was curving downwards. We used the command **"go"** to move through the passage.
2. We were on a free fall with nothing to grab. We used the command **"go"** but we died.
3. We started again, but instead of using "go", we used the command **"wave"** and reached the next screen.
4. We hit some water body. We then used the command **"dive"**.
5. We found some well-lit passage in the wall which was leading to somewhere deep inside. We used the command **"go"** to move through the passage.
6. We found a door with a glass panel right next to it. We used the command **"read"** to read the glass panel. The glass panel read:

   "This is another magical screen. And this one I remember perfectly... Consider a block of size 8 bytes as 8 x 1 vector over F_{128} – constructed using the degree 7 irreducible polynomial $x^7 + x + 1$ over F_2. Define two transformations: first a linear transformation given by invertible 8 x 8 key matrix A with elements from F_{128} and second an exponentiation given by 8 x 1 vector E whose elements are numbers between 1 and 126. E is applied on a block by taking the ith element of the block and raising it to the power given by ith element in E. Apply these transformations in the sequence EAEAE on the input block to obtain the output block. Both E and A are part of the key."

2 How we cracked the cipher text:

1. The aforementioned problem is a weak modification of AES, quite similar to SASAS.
2. We found that two letters constitute a byte in the input and output. Also the 16 letters (4 bits, so 2^4 letters) are from *'f' to 'u'*, which are mapped with numbers 0 to 15 respectively. Also the MSB of each byte in the output was observed to be 0 to bring each byte in GF(128).
3. We observed that the first i bytes of the ciphertext came out to be zero whenever a chosen plaintext of a 8 byte vector was passed with the first i bytes equal to zero. Also, changing i^{th} byte of input caused a change only in the bytes after the i^{th} byte in the output. We therefore concluded that the key matrix or Affine matrix used is a *lower triangular matrix*.
4. Using **generate_inputs.py**, we generated inputs of the form (00..0xx0..0) and stored it in inputs.txt. The corresponding outputs are fetched by **script.sh** and stored in outputs.txt.
5. The inputs we used had only one non-zero byte at a time. Consider an input where i^{th} byte is x_i (say) ($x_i \neq 0$). Now, as the matrix is lower-triangular, the corresponding output should be $(a_{i,i} * (a_{i,i} * x^{ei})^{ei})^{ei} ... eq(1)$
   where $a_{i,i}$ is i^{th} diagonal element of the Affine matrix, and ei is the i^{th} element of exponent matrix. Now, for each value of $a_{i,i}$ and ei, we iterate over all plaintext-ciphertext pairs, compare them and store the possible values of $a_{i,i}$ and ei. This is done using **possible_diag_exp.py**.

6. The possible diagonal elements were: [[[96, 104, 126], [4, 97], [78, 121, 122], [83, 73, 85], [95, 53, 124], [50, 43, 28]]]
   The possible Exponential elements were: [[18, 21, 88], [11, 34, 82], [20, 108], [29, 43, 55], [26, 113, 115], [10, 54, 63], [33, 102, 119], [23, 48, 56]]
7. Next, we iterate on above pairs using more plaintext-ciphertext pairs such that equation (1) holds. This way, we end with only one element for each position in diagonal positions and exponential vector. Also, we use $a_{i,i}$ and $a_{j,j}$ to find $a_{i,j}$,i.e, the elements next to diagonal elements. This is done using **filter_diag_exp.py**
   The diagonal elements are: [[[126], [85], [4], [20], [78], [85], [53], [50]]]
   The exponential elements are: [[88], [34], [20], [43], [26], [63], [102], [23]]
8. Now, for each of the remaining elements, we try all possible values and filter out the ones which do not satisfy equation (1) given the elements we already found. We continued this process using different plaintext-ciphertext pairs till we broke the cipher.
9. The key matrix was found to be:

126	0	0	0	0	0	0	0
32	85	0	0	0	0	0	0
68	100	4	0	0	0	0	0
126	91	29	20	0	0	0	0
74	91	122	17	78	0	0	0
76	83	69	0	94	85	0	0
58	111	40	93	114	77	53	0
39	122	108	68	78	2	62	50

10. The exponential matrix was found to be

[88 34 20 43 26 63 102 23]

11. Finally, we used **decrypt.py** to decrypt the 'password' (hjhmmjhmmoloigjqhgkhgqltjsihjqfi"). We did this by checking whether the output of EAEAE function matches the corresponding encrypted byte of password for all possible values of that byte. The password was hence found to be:

cgbhjmbixebsjcvr