Q1 Team Name
0 Points

Turing

Q2 Commands
5 Points

List the commands used in the game to reach the ciphertext.

go, wave, dive, go, read, password

Q3 Analysis
50 Points

Give a detailed description of the cryptanalysis used to figure out the password. (Use Latex wherever required. If your solution is not readable, you will lose marks. If necessary the file upload option in this question must be used TO SHARE IMAGES ONLY.)

After entering the commands "go, wave, dive, go, read", we finally got the screen that contains a description of the encryption method used which turned out to be an EAEAE problem. We also got these pieces of information from the final screen :
-> Encryption used is a block cypher with a block size of 8 bytes as 8 x 1 vector over field F_{128}.
-> The vector is constructed using the degree 7 irreducible polynomial for the linear transformation which is $x^7 + x + 1$ over F_2.
-> 8x8 Matrix A, which is used for linear transformation, contains elements from the field F_{128}
-> 8x1 Vector E, which is used for exponentiation, contains elements between 1 and 126.

Cryptanalysis:

We tried many inputs, and we observed that cyphertext contains alphabets from the range ‘f’ to ‘u’. So, the total number of characters used is 16. Thus, the letters can be represented in 4 bits.

To represent a byte, we will need two characters as each character is 4 bits long. Also, the field is F_{128}, in which each element is 7 bit long. So, the MSB is fixed with 0. We also noticed that the encryption of inputs of form ‘aaf’ or ‘bbf’ and ‘aa’ or ‘bb’ respectively are the same. So, we found that ‘f’ is used for padding. Hence, we can conclude that ‘f’ must be mapped to ‘0000’. So, we mapped each letter with a binary number from 0000 to 1111 as follows

{'0000': 'f', '0001': 'g', '0010': 'h', '0011': 'i', '0100': 'j', '0101': 'k', '0110': 'l', '0111': 'm', '1000': 'n', '1001': 'o', '1010': 'p', '1011': 'q', '1100': 'r', '1101': 's', '1110': 't', '1111': 'u'}

So, each byte is from ‘ff’(0000 0000) to ‘mu’(0111 1111).

The next thing we observed is that on keeping the same prefix of different plain texts, we get the same prefix in the corresponding ciphertexts. For e.g., for inputs “mmmmuuuuhnhnhnhn” and “mmmmuuuulklklklk”, we get cipher texts as “gmmhfklohgigjqjj” and “gmmhfkloffhofrkj”. Here, both inputs have the same prefix as “mmmmuuuu”, and ciphertext also contains the same prefix “gmmhfklo”. Hence, the transformation matrix can be a Lower Triangular Matrix.

Now, lets assume input plaintexts as $I_0, I_1, ... , I_7$ and output ciphertexts as $C_0, C_1, ... , C_7$. On setting “ff” as prefix up to k bytes, where k is between 0 to 7, we get “ff” as prefix up to k bytes in ciphertext as well. Also, if we change inputs from “$I_0, ..., I_k, I_{k+1}, ..., I_7$” to “$I_0, ..., I_k, I_{k+1}^{'}, ..., I_7$”, the ciphertext changes only after k bytes. Hence, we can conclude that 0's are only present at the end of each row in the transformation matrix.

Therefore, the transformation matrix is Lower Triangular

Matrix.

Plaintexts generation process:

Then, we generated inputs using “getPlainText.py” in which at most one block is non-zero for any input. For that block, we have 128 different combinations from ‘ff’ to ‘mu’. There can be 8 different positions where we can change a byte. So, in total, we have 8 * 128 different inputs. Inputs are saved in the file “plaintext_file.txt”.

Ciphertexts generation process:

After generating the required number of plaintexts we need to find the ciphertext corresponding to them. For this, we used a python script to establish the connection with the server and one by one generates all ciphertexts in the file "ciphertext_file.txt".

The next step in our analysis is to find the transformation matrix A and exponentiation vector E.

Now since only 1 block has non-zero element in a input of plaintexts, so we are iterating over all possible values of diagonal elements of transformation matrix and also of exponentiation vector since the matrix is Lower triangular matrix. So, say if i^{th} block of a particular input is non-zero, then the corresponding output for that block will be, O = $(a_{i,i} * (a_{i,i} * x^{e_i})^{e_i})^{e_i}$, where $a_{i,j}$ is the element of i^{th} row and j^{th} column of the transformation matrix, e_i is the i^{th} element of exponentiation vector and x is the non-xero block of the input text.

We then implemented linear transformation and exponentiation operations over the finite field of 128 with generator $x^7 + x + 1$ using the code in “decryption.py”.

What we did is for each (plaintext, ciphertext) pair we iterated over all possible values of e_i and $a_{i,i}$ and see if inputs map to the output, and if it does then we add them to the corresponding list of possible values of E and A.

The possible pairs obtained per block are :

Block	$a_{i,i}$	e_i
B0	[27,84,84]	[1,19,107]
B1	[46,63,70]	[45,93,116]
B2	[105,43,107]	[17,41,69]
B3	[124,12,88]	[51,80,123]
B4	[47,112,96]	[65,92,97]
B5	[5,11,121]	[36,42,49]
B6	[27,66,70]	[22,37,68]
B7	[93,38,5]	[18,21,88]

Now, we need to eliminate incorrect pairs of $a_{i,i}$ and e_i. Also we need to find other non-diagonal elements. To do so, we used some more plaintext-ciphertext pairs, and iterate over above ($a_{i,i}, e_i$) pairs to find elements in range 0-127 such that equation for O (defined above) holds. we used $a_{i,i}$ and $a_{j,i}$ to find the required element $a_{i,j}$, after doing this step we will get every element next to each diagonal element.

Also for diagonal elements, we reduced possible pairs to 1 element each, after doing this the final values we get are

Block	$a_{i,i}$	e_i
B0	84	19
B1	70	116
B2	43	41
B3	12	80
B4	112	92

B5	11	42
B6	27	22
B7	38	21

Now for the remaining elements of matrix A, we iterated over all values from 0-127 and eliminated the values that don't satisfy O using the final values of elements of transformation matrix A and exponentiation vector E.

The complete Linear Transformation matrix after the above step is,

84	0	0	0	0	0	0	0
115	70	0	0	0	0	0	0
12	29	43	0	0	0	0	0
125	16	2	12	0	0	0	0
97	33	1	119	112	0	0	0
25	51	31	47	110	11	0	0
21	120	22	100	2	88	27	0
66	13	85	27	21	66	31	38

And the Exponentiation vector is,

19	116	41	80	92	42	22	21

Decrypting the Final Password:

Once we get the final transformation matrix A and Exponentiation vector E, we used them to decrypt the encrypted password "gskohggkhqglmhlumrmnjhlohlimjtgm", Since the encrypted password is 32 characters(16 bytes) long, so we processed it in two blocks of 16 characters(8 bytes). We applied the transformations in reverse order as $E^{-1}(A^{-1}(E^{-1}(A^{-1}(E^{-1}(password)))))$ on each block.

After applying these transformations we get the decrypted values for both the blocks as,

values for block 0 : [115,117,114,118,100,111,111,114]

values for block 1 : [102,100,48,48,48,48,48,48]

We tried mapping these values according to this mapping{ff:0, ..., mu:127} and got the decrypted text as "mimkmhmlljlulumhllljffffffffffff". Discarding the last f's as they are used for padding our password might be "mimkmhmlljlulumhlllj" but this was not the correct password.

Since the mapping is between 0 to 127 we thought this might map to ASCII values. So we tried mapping these values with ASCII characters and found the password to be "survdoorfd000000". Discarding the last 0's we got the password to be "survdoorfd", which turned out to be the correct password.

Hence the Final password is "survdoorfd".

No files uploaded

Q4 Password
5 Points

What was the final commands used to clear this level?

survdoorfd

Q5 Codes
0 Points

It is mandatory that you upload the codes used in the cryptanalysis. If you fails to do so, you will be given 0 for the entire assignment.

▾ Group-Turing.zip Download

```
Binary file hidden. You can download it using the button above.
```

Assignment 5

● GRADED

2 DAYS, 20 HOURS LATE

GROUP

Dinkar Tewari
Deepak Raj
Rohit kushwah
View or edit group

TOTAL POINTS

55 / 60 pts

QUESTION 1

Team Name **0** / 0 pts

QUESTION 2

Commands **5** / 5 pts

QUESTION 3

Analysis **45** / 50 pts

QUESTION 4

Password **5** / 5 pts

QUESTION 5

Codes **0** / 0 pts