CS641A Assignment-5

Gunda Abhishek(14807257) Harsha Nalluru(14408) Bhargav Reddy(14468)

18th Feb, 2018

Traversed the chamber using the list of commands already known. Used 'go' in the first command line and the next chapter lead to death for every single command until we tried the 'wave' command and the following chambers can be traversed just by reading the passages.

Final List of commands to get to the cipher.

- go
- wave
- dive
- go
- read

We arrived at the following passage and believed it to be the cipher protocol just like in the previous chamber.

"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. You can see the coded password by simply whispering 'password' near the screen..."

When provided *'password'* as input, we observed *'ktirlqhtlqijmmhqmgkplijngrluiqlq'* as output.

When provided any other string suppose *'qnpjftqrrqggkhjl'*, we get *'jniggkksfsjtmskh'* as output which is of size 16.

The following observations are made using the input-output pairs.

- *h* → *isigmfgmjsmsjijg*
- *hf* → *isigmfgmjsmsjijg*
- *hff* → *isigmfgmjsmsjijg*
- *hfff* → *isigmfgmjsmsjijg*
- *hffff* → *isigmfgmjsmsjijg*
- *hfffff* → *isigmfgmjsmsjijg*
- *hffffff* → *isigmfgmjsmsjijg*

- *hfffffff* → *isigmfgmjsmsjijg*

  .

  .

  .

- *hf*15* → *isigmfgmjsmsjijg*

i.e., each input of size less than 16, is concatenated with f's until it makes a string of size 16.

If provided a string, every character in the output lie in the range of characters $[f \dots u]$.

If provided a string with characters more than 16 then the size of output will be 32 and the character **'f'** is padded to the input until it becomes a size of 32

Similar to the Chapter-4, after providing many inputs, we observed that, if swapped any consequent letters(characters outside [f-u]) with positions 2*i and 2*i+1 in the input, the output remains the same.

- **Cracking the Cipher: (Observations)**

  As we already know that the input size for the cipher is blocks of 16 characters and two characters are counted as a byte which leaves with blocks 64bit input for the cipher, with minimum one block.

  It is also given in the passage that a block of 64 bits is treated as 8x1 vector i.e each element in the input vector has 8bits and corresponds to two characters in the plain-text.

  The above mentioned vector(8x1) is given to be over a Finite Field F_128 with degree 7 irreducible polynomial $x^7 + x + 1$ over F_2

  Let E be the exponential vector(8x1) and A be the invertible Key matrix(8x8)

$$E = \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_7 \end{bmatrix} \alpha_i \in [1, 126], A = \begin{bmatrix} a_0 & b_0 & c_0 & \dots & h_0 \\ a_1 & b_1 & c_1 & \dots & h_1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_7 & b_7 & c_7 & \dots & h_7 \end{bmatrix} \in F\_\{128\}$$

  It is given that the transformations are followed in the following order EAEAE

  In the chamber we observed for every text starting with characters **ff** which is translated to **00** in hex output has the same starting characters i.e **ff** i.e We understood that **0** is going through the transformations unscathed in the starting characters.

  - *ffa* → *ffmmmkgpghlhkhfu*
  - *ffaa* → *ffkqhigolmkjiqhs*
  - *ffffa* → *fffflgkkkrkiggt*
  - *ffffaa* → *ffffjslgmpjskomo*

  Consider the following input vector $I = [0, 1, 1, \dots, 1]$, we know that the final output vectors first element is going to be 0.

After the first three transformations(EAE) input vector changes to

$$I = \begin{bmatrix} (b_0 + c_0 + \cdots + h_0)^{\alpha_0} \\ (b_1 + c_1 + \cdots + h_1)^{\alpha_1} \\ \vdots \\ (b_7 + c_7 + \cdots + h_7)^{\alpha_7} \end{bmatrix}$$

After the next two transformations (AE) the first element changes to
$(a_0(b_0 + c_0 + \cdots + h_0)^{\alpha_0} + b_0(b_1 + c_1 + \cdots + h_1)^{\alpha_1} + \cdots + h_0(b_7 + c_7 + \cdots + h_7)^{\alpha_7})^{\alpha_0} mod(x^7 + x + 1)$

The polynomial is a prime number over 2 which implies that the above expression has to be multiple of 131 which is not possible as it is in the finite Field F_{128} and the only possibility for which the above expression is zero is when $b_0, c_0, d_0, e_0, f_0, g_0, h_0$ are all zero's

Similarly for the input [0,0,1,1,1,1,1,1] we can show that $c_1, d_1, e_1, f_1, g_1, h_1$ are all zero's going similarly we get to the final reduced matrix of A to be

$$A = \begin{bmatrix} a_0 & 0 & 0 & \dots & 0 \\ a_1 & b_1 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_7 & b_7 & c_7 & \dots & h_7 \end{bmatrix}$$

$$E = \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_7 \end{bmatrix} \alpha_i \in [1, 126], A = \begin{bmatrix} a_0 & 0 & 0 & \dots & 0 \\ a_1 & b_1 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_7 & b_7 & c_7 & \dots & h_7 \end{bmatrix}$$

,

- **More Insight with Inputs**

  For the input [x, 0, 0, 0, 0, 0, 0, 0], after going through the 'EAEAE' sequence of transformations, we obtain the following two expressions as the first and second elements of the output respectively $(a_0(a_0x^{\alpha_0})^{\alpha_0})^{\alpha_0}, (a_1(a_0x^{\alpha_0})^{\alpha_0} + b_1(a_1x^{\alpha_0})^{\alpha_1})^{\alpha_1}$

  For the input [x, y, 0, 0, 0, 0, 0, 0], after going through the 'EAEAE' sequence of transformations, we obtain the following three expressions as the first, second and third elements of the output respectively $(a_0(a_0x^{\alpha_0})^{\alpha_0})^{\alpha_0}, (a_1(a_0x^{\alpha_0})^{\alpha_0} + b_1(a_1x^{\alpha_0} + b_1y^{\alpha_1})^{\alpha_1})^{\alpha_1}, (a_2(a_0x^{\alpha_0})^{\alpha_0} + b_2(a_1x^{\alpha_0} + b_1y^{\alpha_1})^{\alpha_1} + c_2(a_2x^{\alpha_0} + b_2y^{\alpha_1})^{\alpha_2})^{\alpha_2}$

  For the input [x, y, z, 0, 0, 0, 0, 0], after going through the 'EAEAE' sequence of transformations, we obtain the following three expressions as the first, second and third elements of the output respectively $(a_0(a_0x^{\alpha_0})^{\alpha_0})^{\alpha_0}, (a_1(a_0x^{\alpha_0})^{\alpha_0} + b_1(a_1x^{\alpha_0} + b_1y^{\alpha_1})^{\alpha_1})^{\alpha_1}, (a_2(a_0x^{\alpha_0})^{\alpha_0} + b_2(a_1x^{\alpha_0} + b_1y^{\alpha_1})^{\alpha_1} + c_2(a_2x^{\alpha_0} + b_2y^{\alpha_1} + c_2z^{\alpha_2})^{\alpha_2})^{\alpha_2}$

  For the first two inputs(8x1 vectors), the first element in the output remains same

  For the second and third inputs(8x1 vectors), the first element and second element in the output remain same

  The same observation can be extended as follows due to the presence of zeros in the matrix.

We came to a pattern that the if two inputs have common prefixes of size i then both the outputs will have common prefixes of size i irrespective of their differences following from the $(i+1)^{th}$ positional values — $(i \in [1, 8])$

Also, in the output 8x1 vector, the values of the elements upto i index depend only on the values of the elements upto index i of the corresponding 8x1 input vector

Common prefixes of size i in the 8x1 vector means that, common prefixes of size $2 * i$ in the inputs(alphabetical) provided in the *caves.html*

That implies, common prefixes of size $2*i$ in the inputs(provided in the *caves.html*) result in common prefixes of size $2 * i$ in the outputs.

That implies, prefix of size $2 * i$ in the output depends only on the prefix of size $2 * i$ in the input, and independent of the characters in the input present after the position $2 * i$. — $(i \geq 1)$

- **Observation and Verification**

Prefix of size $2 * i$ in the output depends only on the prefix of size $2 * i$ in the input, and independent of the characters in the input present after the position $2 * i$. — $(i \geq 1)$

[*proof.py*]

To verify the above theoretical observation, we generated 1000 inputs each with common first two, four, six characters(in the domain [f-u]), used pyautogui library and retrieved the corresponding 1000 input-output pairs.

From the above input-output pairs, we realized that the inputs with common first two, four, six characters had the common first two, four, six characters in the corresponding outputs.

[*presenting_the_proofs.txt*] For example,

- ***okrk****hlirlkjugpkl* → ***ltfg****kngjgsihkkms*
- ***okrk****jjlttholtouh* → ***ltfg****jgjummgulhii*
- ***okrk****ihhmthrtglkf* → ***ltfg****ffkkkrlphhms*
- ***okrk****iqtpolikmllp* → ***ltfg****hnhpkqjmhtjj*
- ***okrk****mptglhrpknkn* → ***ltfg****hogphnlsliig*
- ***okrk****ruqilskunimh* → ***ltfg****lsjmmnhjjuml*
- ***okrk****iokljqhrmjpg* → ***ltfg****kmmuhqhmjpho*
- ***okrk****gqpnsonljrnu* → ***ltfg****joflfhjrfghg*
- ***okrk****fhnkjimnjlni* → ***ltfg****jqlsljltfflu*
- ***okrk****fpuhsqunmnlo* → ***ltfg****fjfsghlhmjlm*
- ***okrk****jrgfqjlnmtrm* → ***ltfg****hjktfkikjoiq*
- ***okrk****oqfmsjuirhtm* → ***ltfg****hhkmkqgrkpfi*

.

.

.

The above inputs have '**okrk**' as the common prefix, '**ltfg**' is the common prefix for the corresponding outputs.

With similar input-output pairs, we realized that the inputs with common prefixes(of lengths two, four, six, and so on) have the common prefixes(of lengths two, four, six, and so on) in the corresponding outputs.

- **Cracking the Password**

  When provided *'password'* as input, we observed *'ktirlqhtlqijmmhqmgkplijngrluiqlq'* as output.

  – *ktirlqhtlqijmmhqmgkplijngrluiqlq* has to be decrypted.

  As we know that, the prefix of size $2 * i$ in the output depends only on the prefix of size $2 * i$ in the input, and independent of the characters in the input present after the position $2 * i$. — $(i \geq 1)$

  We use the above principle repeatedly and decrypt the above string

  Consider the first two characters in the output *kt*

  If we put $i = 1$ in the above principle, the prefix of size 2 in the output depends only on the prefix of size 2 in the input and independent of the characters in the input present after the position 2, that implies the following

  It is the output of an element the following $\{xy : x, y \in [f, u]\}$

  $\{xy : x, y \in [f, u]\}$ has 256 elements, it is easy to obtain the input corresponding to *'kt'*

  [*crack_the_password.py*]

  We used *pyautogui*, and automated the process of finding the input corresponding to *'kt'*

  Suppose the result is $\alpha\beta$

  Once we obtain the first two characters, we then look for the next two characters of the input

  Consider the first four characters in the output *'ktir'*

  It is the output of an element the following $concatenate(\alpha\beta, \{xy : x, y \in [f, u]\})$

  $\{xy : x, y \in [f, u]\}$ has 256 elements, it is easy to obtain the input corresponding to *ktir*

  [*crack_the_password.py*]

  We used *pyautogui*, and automated the process of finding the input corresponding to *ktir*

  By now, we obtained the first four characters $\alpha\beta\gamma\delta$ of the input corresponding to the encrypted password.

  In the similar way, we iterate through the *'ktirlqhtlqijmmhqmgkplijngrluiqlq'* and find out all the characters in the corresponding input.

  [*crack_the_password.py*]

  We used the libraries *pyautogui, pyperclip* mainly and decrypted the whole password.

  It took about 1755 iterations(1755 input-output pairs) to decrypt *ktirlqhtlqijmmhqmgkplijngrluiqlq*.

- **Tweaking the 'password'**:

  By following the above presented method, when decrypted *ktirlqhtlqijmmhqmgkplijngrluiqlq*, we obtained ***lhlgmjmkmglqlompmoltmglilqlmlgmh*** as the corresponding input.

Entered the string directly in the website, but failed.

We remembered the same situation encountered in Chapter-4.

In Chapter-4, we converted each 8-bit set into ASCII characters and then submitted, which worked in **The Great Caves**

In order to do the same, we converted the string ***lhlgmjmkmglqlompmoltmglilqlmlgmh*** into bits(each character represents 4 bits – *utils.py*), and obtained the following 128-bit string

- *submit_password.py*

  01100010011000010111010001110101011100010110101101101001011110100111100101101110 0

  11100010110001101101011011001110110000101110010

Divided this 128-bit string into blocks of size - 8 bits, and converted each block of 8 - bits into ASCII characters and then obtained this string ***batuqkizynqckgar***

This when submitted in the **The Great Caves**, passed the chapter and entered into Chapter-6 successfully.

- **Codes and their usages:**

  utils.py:

  - contains all the member functions that are necessary for cracking the program, includes generation of random 64 bits, generating bits to string and vice-versa

  login_in_caves.py:

  - Used *pyautogui*, automated the script to login

  proof.py:

  - Script to practically prove the theoretical result
  - Used *pyautogui, pyperclip, random* libraries, and generated 1000s of input-output pairs to prove the result
  - *presenting_the_proofs.txt* has the corresponding input-output pairs

  crack_the_password.py:

  - Used *pyautogui*, automated the process of finding the input corresponding to *ktirlqhtlqijmmhqmgkplijngrluiqlq*
  - *crack_trials_finals.txt* has corresponding input-output pairs at each iteration
  - *password_final.txt* contains the useful input-output pairs, the input-output pairs actually required to crack the password
  - Finally obtained *lhlgmjmkmglqlompmoltmglilqlmlgmh*

  submit_password.py:

  - Tweaks the string *lhlgmjmkmglqlompmoltmglilqlmlgmh*, converted into 128-bit string and

– then Divided the 128-bit string into blocks of size - 8 bits,
– and converted each block of 8- bits into ASCII characters
– and then obtained this string ***batuqkizynqckgar***