**Attack Process:**

We will implement the TEA which is block cipher. TEA is a variant of the Fiestel cipher, it is designed for speed and simplicity as a main aim, during the encryption of data this tea algorithm will mainly operate on 64 bit data which will be divided into two halves of 32 bit The key size used in TEA is 128bits, and a magic constant is also utilized which is defined as 2^32/(the golden ratio), which equals 2654435769. This constant is added to itself during each round, and is used to prevent attacks that try to take advantage of symmetry between rounds.

This attack will be implemented on 1st round of tea in which the key length is reduced to 64 bits because  size is effectivelyas opposed to the 128bits that would normally be utilized in 64 rounds of TEA. This is because of the fiestel nature of TEA; only half of the 128bit key is used in the first round. The remaining 64 bits of the key are introduced in the 2nd round, but because we are attacking 1 round of TEA we will focus on only half of the 128bit key.  Using known plaintext and their resulting ciphertext, we will perform a brute force attack on the 64bit key by repeatedly guessing 32bits of the key, which will eventually lead us to deduce the other 32bits.

TEA has a 128 bit key that is split up into four 32 bit subkeys, which can be seen as K[0],k[1],k[2],k[3] in thediagram. Delta is defined as a constant, 2^32/(golden ratio), which is 2654435769 as an integer. Multiples of delta are used in each round (mod 2^32).

In the first feistel round R is used as an input to several operations. All addition operations are (mod 2^32).

1. R goes through a left shift of 4 and then is added to K[0]

2. R is added to Delta

3. R goes through a right shift of 5 and then is added to K[1]

An XOR operation is then applied to the result of those three operations and finally, the result of the XOR operation is added to L. This result then becomes R for the next feistel round, because of the swap.

**Description of the partial key search attack:**

We implemented our attack on 1 Feistal round of TEA. In 1 Feistal round of TEA the key size is effectively reduced to 64 bits, which we call the first 32 bits subkey T and the second 32 bits subkey U. Normally a 128 bit key would be used in 2 or more rounds of TEA. This is because of the Feistal nature of TEA; only half of the 128 bit key is used in the first Feistal round. The remaining 64 bits of the key are used in the 2nd round. Using known plaintext and ciphertext pairs, we will perform a brute force attack on the 64 bit bit key by starting from 0 and incrementing by one until we guess a correct value for subkey T. This means that our brute force search will be of the order 2^32. We can do this because of our ability to calculate subkey U once we guess a value for T.

We are able to calculate U because of our ability to solve the encryption formula for subkey U. In the diagram above you can see the steps required to solve the TEA cipher equation for subkey U. The TEA cipher operations are substituted for F(R0, K1), and after some subtracting and XORing on either side of the equation we are end up with an equation solved for subkey U. Because L0, R0, and R1 are known (from the plaintext/ciphertext file generated using the oracle), and Delta is also known, all we have to do is guess a value for subkey T in order to solve for U. Additionally, if we use a different set of L0, R0, and R1 with the same Delta and the same T, we should get the same U. This “different set” of L0, R0, and R1 is really just a different or 2nd plaintext/ciphertext pair that we need to use. If we test a 2nd plaintext/ciphertext pair and we get a different value for U, that means that our guess for subkey T was incorrect, and we can increment our guess and do the calculation again. Thus, we can continually do this until we find a value for T that gives the same value for U for all of the plaintexts and ciphertexts.

**Description of the partial key search attack**.

Our attack on 1 Feistal round of TEA is implemented. In 1 Feistal round of TEA the key size is effectively reduced to 64 bits, which we call the first 32 bits subkey T and the second 32 bits subkey U. Normally a 128 bit key would be used in 2 or more rounds of TEA. This is because of the Feistal nature of TEA; only half of the 128 bit key is used in the first Feistal round. The remaining 64 bits of the key are used in the 2nd round. Using known plaintext and ciphertext pairs, we will perform a brute force attack on the 64 bit bit key by starting from 0 and incrementing by one until we guess a correct value for subkey T. This means that our brute force search will be of the order 2^32. We can do this because of our ability to calculate subkey U once we guess a value for T.

We are able to calculate U because of our ability to solve the encryption formula for subkey U. In the diagram above you can see the steps required to solve the TEA cipher equation for subkey U. The TEA cipher operations are substituted for F(R0, K1), and after some subtracting and XORing on either side of the equation we are end up with an equation solved for subkey U. Because L0, R0, and R1 are known (from the plaintext/ciphertext file generated using the oracle), and Delta is also known, all we have to do is guess a value for subkey T in order to solve for U. Additionally, if we use a different set of L0, R0, and R1 with the same Delta and the same T, we should get the same U. This “different set” of L0, R0, and R1 is really just a different or 2nd plaintext/ciphertext pair that we need to use. If we test a 2nd plaintext/ciphertext pair and we get a different value for U, that means that our guess for subkey T was incorrect, and we can increment our guess and do the calculation again. Thus, we can continually do this until we find a value for T that gives the same value for U for all of the plaintexts and ciphertexts.

**Method for determining U after T is found is :**

L1=r0

R1=L0+f(R0,K1)

R1=L0+(((R0<<4+T)XOR((R0>>5)+U)XOR(R0+DELTA1))

R1-L0=(((R0<<4+T)XOR((R0>>5)+U)XOR(R0+DELTA1))

(RI-L0)XOR((R0<<4)+t)= ((R0>>5)+U)XOR(R0+DELTA1))

(RI-L0)XOR((R0<<4)+t)XOR(R0+DELTA1)= (R0>>5)+U

U=(RI-L0)XOR((R0<<4)+t)XOR(R0+DELTA1)-(R0>>5)

**Description of attack program** Our attack program takes one argument which is the name of the file with the plaintext/ciphertext pairs generated by the Oracle program. The plaintext/ciphertext pairs are in the following format:

[64 bit plaintext, 64 bit ciphertext]

The general idea of the attack program is that we only have to bruteforce 2^32 possible values. This is because given a subkey T, we can calculate a value for subkey U by using known plaintext/ciphertext pairs.  The following steps/pseudocode outline the general procedure of the attack program.

1. Read in the file with the plaintext/ciphertext pairs into lists. These values are then converted from strings in the file to 32 bit unsigned integers.

2. Guess a value for subkey T, starting at 0. 3. Calculate the value for U using our guess and the first plaintext/ciphertext pair 4. Calculate the value for U using our guess and the second plaintext/ciphertext pair 5. Do those two values of U match? 6. If not, this is the incorrect guess for T. Increment our guess and go back to Step 3. 6. If yes, we need to verify that this guess for subkey T is correct 7. Repeat steps 3 and 4 ten times with different plaintext/ciphertext pairs 8. If the values of U did not match every time, increment our guess for T and go back to Step 3. 9. If the values of U matched every time, this is the correct guess for T and we’ve found the key!!

An interesting note about our attack methodology was the number of plaintexts/ciphertexts required for the verifying step . We initially had this set at 100 instead of 10, but we decided to reduce it to see if the attack would still succeed.

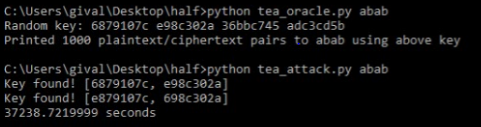
**Analysis:**

Our attack program was successful in finding  random encryption keys that were generated through the use of our oracle program. Our oracle program selects a random 128 bit

key, and then generates 100 random plaintext/ciphertext pairs using 1 Feistal round of TEA encryption. These plaintext/ciphertext pairs are what the attack program uses to calculate the key. This is what the output of the oracle looks like for one of the keys that we generated.

Keep in mind that because we are only attacking 1 Feistal round of TEA, we are only trying to discover the first 64 bits of the key because the other 64 bits of the key are not used in the 1st round of TEA.

**Attack1:**

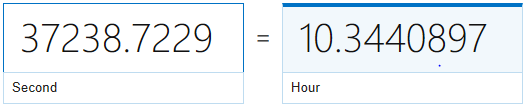


**DESCRIPTION OF THE ATTACK**

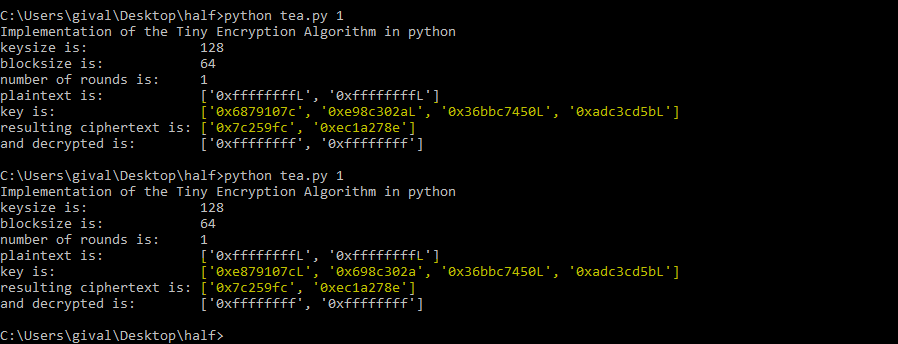
**Random key** :is the key which is used by oracle program in order to develop those pain/cipher text pairs which was further displayed in the file named abab

**Key found!** : this is the key found after our attack was successful the first key found was a part of the random Key whereas second one is the equvivalent key that what we are searching for

**Seconds:** This seconds shows the figure of seconds that were taken in order to check or obtain this equvivalent key (that is 37238.721999 sec in our attack which is nearly equals to nearly 10.35 hrs)



**To prove that these keys are actually equivalent:**



You can understand the behaviour of the equivalent key by observing the text that is being highlighted in the above screenshot (yellow coloured) here you can observe that the change in key even produced the same cipher text and which is the main function and equivalent key and this property of having the equivalent keys will affect the performance of the algorithm that is the security of the algorithm…

This attack was performed by running script on Laptop with following specifications:

OS: Windows 10,64 bit operating system,X64-based processor

Processor: Intel® Core™ i3-6th Gen @ 2.00Ghz clock speed

RAM: 8.00GB