VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

CRYPTOGRAPHY AND NETWORK SECURITY
AAT REPORT
on

Double DES and MITM

Submitted by

Ajith Kumar G (1BM19CS009)
Derek Stanley Kannathe (1BM19CS045)

Under the Guidance of
Prof. Nandini Vineeth
Assistant Professor, BMSCE

in partial fulfillment for the award of the degree of
BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019
Mar-2022 to Jun-2022

B. M. S. College of Engineering,

Bull Temple Road, Bangalore 560019
(Affiliated To Visvesvaraya Technological University, Belgaum)

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the AAT work entitled “**Double DES and MITM**" is carried out by **Ajith Kumar G (1BM19CS009), and Derek Stanley Kannathe (1BM19CS045)** who are bonafide students of **B. M. S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2022. The AAT report has been approved as it satisfies the academic requirements in respect of **Cryptography and Network Security (20CS6PCCNS)**work prescribed for the said degree.

Signature of the Guide
Prof. Nandini Vineeth
Assistant Professor
BMSCE, Bengaluru

Signature of the HOD
Dr. Jyothi S Nayak
Associate Prof.& Head, Dept. of CSE
BMSCE, Bengaluru

B.M.S. COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DECLARATION

We, Ajith Kumar G (1BM19CS009) and Derek Stanley Kannathe (1BM19CS045), students of 6th Semester, B.E, Department of Computer Science and Engineering, B. M. S. College of Engineering, Bangalore, hereby declare that this AAT entitled "Double DES and MITM" has been carried out by us under the guidance of Prof. Nandini Vineeth, Assistant Professor, Department of CSE, B. M. S. College of Engineering, Bangalore during the academic semester Mar-2022-Jun-2022

We also declare that to the best of our knowledge and belief, the development reported here is not from part of any other report by any other students.

Signature

Ajith Kumar G (1BM19CS009)

Derek Stanley Kannathe (1BM19CS045)

Chapter 1: Introduction

Double DES:

Double DES is an encryption technique which uses two instances of DES on the exact plain text. In both instances, it uses different keys to encrypt the plain text. Both keys are required at the time of decryption. The 64-bit plain text goes into the first DES instance which is then converted into a 64-bit middle text using the first key and then it goes to the second DES instance which gives 64-bit cipher text by using the second key.

However, double DES uses a 112-bit key but gives a security level of 2^{56} not 2^{112} and this is because of the meet-in-the-middle attack which can be used to break through double DES

Meet In The Middle Attack:

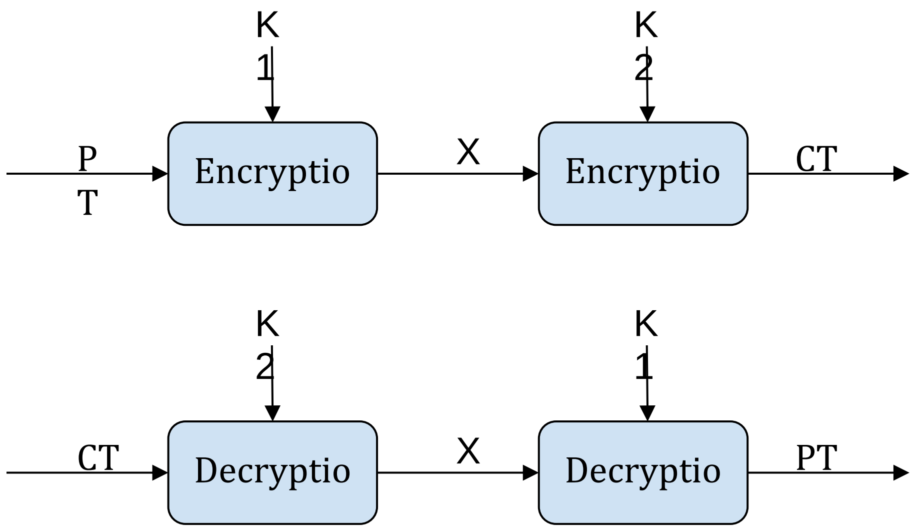
A Meet-in-the-Middle (MitM) Attack is a type of cryptanalytic attack where the attacker needs some type of space or time tradeoff to support the attack. MITM attempts can decrease the amount of difficulty needed to perform the assault in its original state.

Merkle and Hellman introduced the terms of meet-in-the-middle attack. This attack contains encryption from one end and decryption from another and connects the result in the middle, therefore is the name meet-in-the-middle.

MITM can create the form of dividing the target connection into two so that each element can be addressed independently. It can mean changing an attack requiring X amount of time into one requiring Y time and Z space. The goal is to significantly decrease the effort required to implement a brute-force attack.

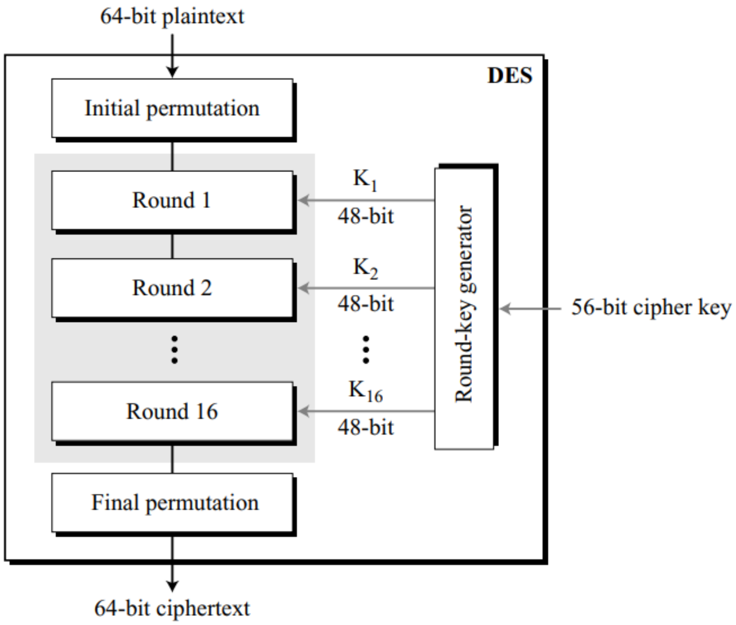
Chapter 2: Methodology

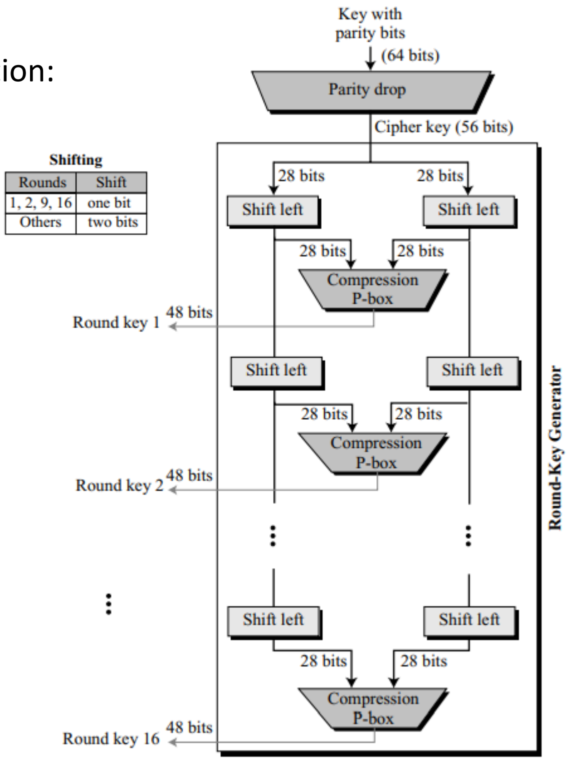
Double DES Structure:



Single Des Structure:

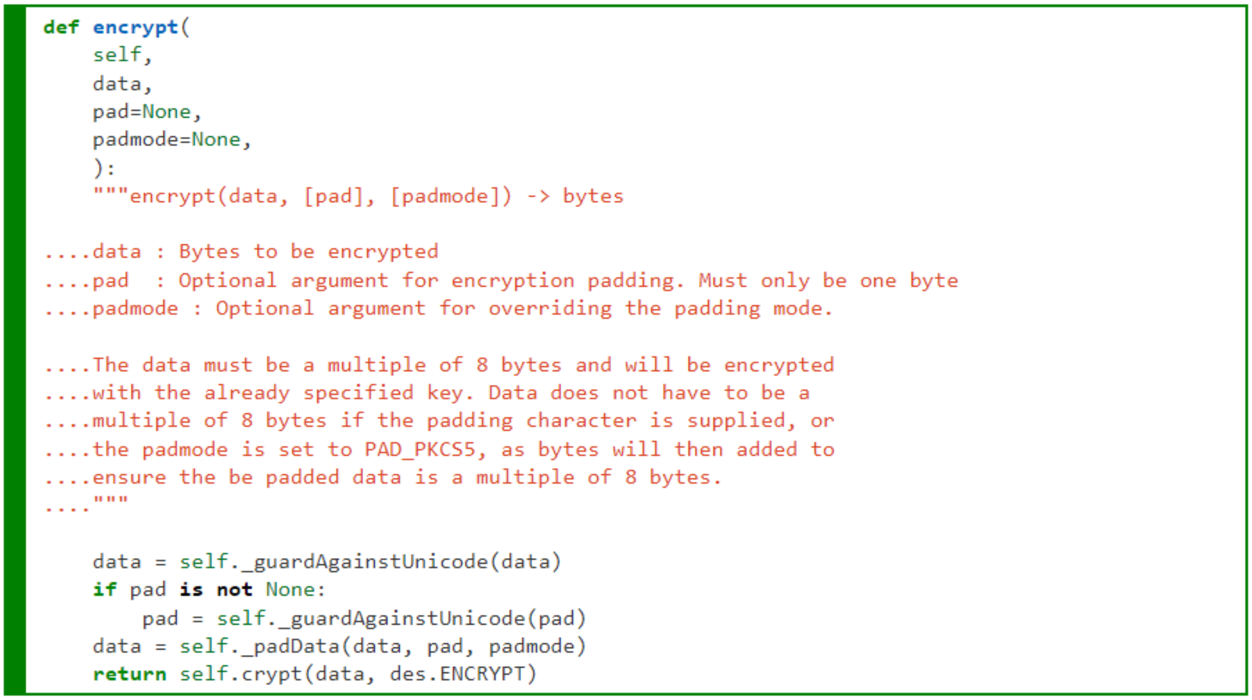
Key Generation:





Rounds	Shift
1, 2, 9, 16	one bit
Others	two bits

Encryption:



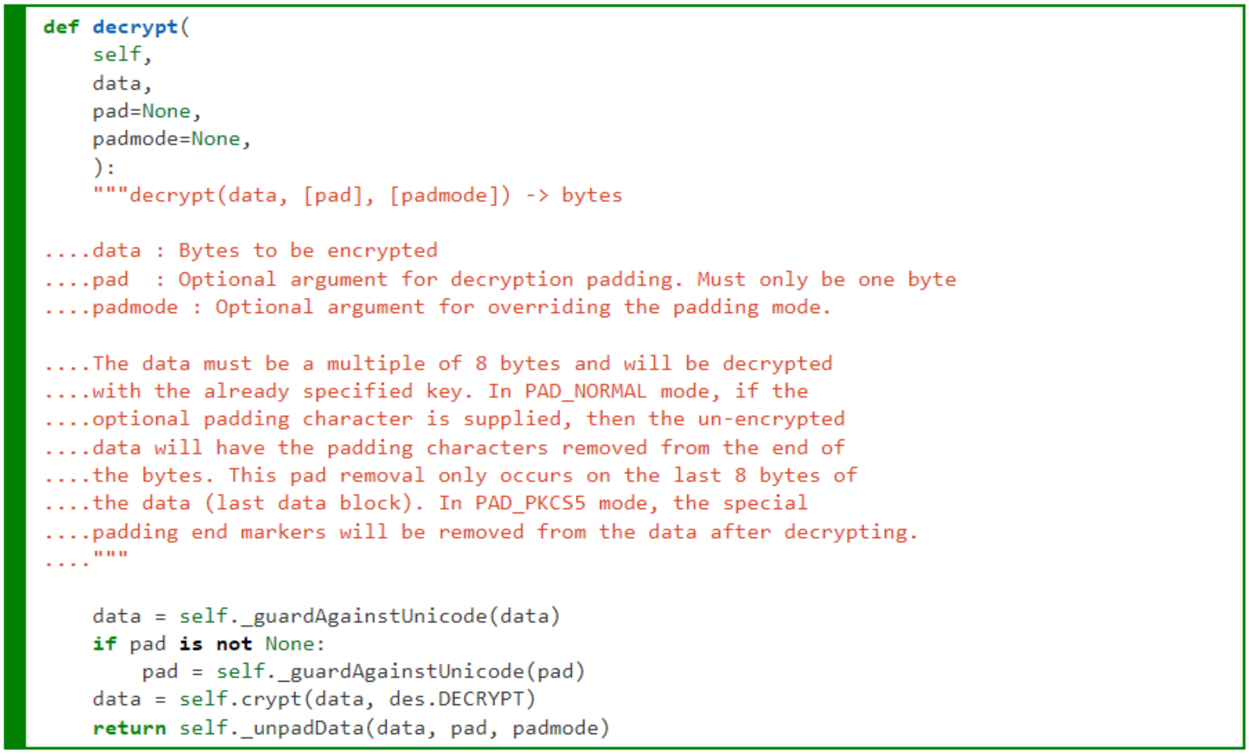
```
def encrypt(
    self,
    data,
    pad=None,
    padmode=None,
    ):
    """encrypt(data, [pad], [padmode]) -> bytes

....data : Bytes to be encrypted
....pad  : Optional argument for encryption padding. Must only be one byte
....padmode : Optional argument for overriding the padding mode.

....The data must be a multiple of 8 bytes and will be encrypted
....with the already specified key. Data does not have to be a
....multiple of 8 bytes if the padding character is supplied, or
....the padmode is set to PAD_PKCS5, as bytes will then added to
....ensure the be padded data is a multiple of 8 bytes.
...."""

    data = self._guardAgainstUnicode(data)
    if pad is not None:
        pad = self._guardAgainstUnicode(pad)
    data = self._padData(data, pad, padmode)
    return self.crypt(data, des.ENCRYPT)
```

Decryption:



```
def decrypt(
    self,
    data,
    pad=None,
    padmode=None,
    ):
    """decrypt(data, [pad], [padmode]) -> bytes

....data : Bytes to be encrypted
....pad  : Optional argument for decryption padding. Must only be one byte
....padmode : Optional argument for overriding the padding mode.

....The data must be a multiple of 8 bytes and will be decrypted
....with the already specified key. In PAD_NORMAL mode, if the
....optional padding character is supplied, then the un-encrypted
....data will have the padding characters removed from the end of
....the bytes. This pad removal only occurs on the last 8 bytes of
....the data (last data block). In PAD_PKCS5 mode, the special
....padding end markers will be removed from the data after decrypting.
...."""

    data = self._guardAgainstUnicode(data)
    if pad is not None:
        pad = self._guardAgainstUnicode(pad)
    data = self.crypt(data, des.DECRYPT)
    return self._unpadData(data, pad, padmode)
```

Meet in the middle Attack on Double DES:

The double DES encryption is illustrated by the following equation.

$$c = Encrypt_{k2}\left(Encrypt_{k1}(m)\right)$$

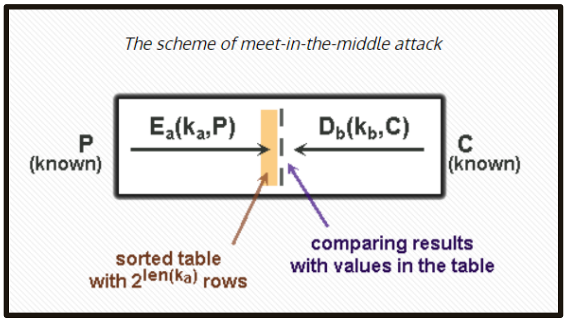
It appears that this is much more secure than DES since the intruder has to find now two keys $\{k_1, k_2\}$ in order to decrypt the cipher message. The intruder now has to search through 2^{112} combinations of keys (k_1, K_2) instead of 2^{56} for normal DES.

Similar to the Brute Force Attack, you are provided with a known "plaintext / ciphertext" pair, i.e. you know that the encryption of m^* is c^*. The meet in the middle attack tries to guess i^*, the intermediate (hence **'the MIDDLE'**) ciphertext that is the result of one DES encryption of c^*, and also the result of one DES decryption of c^*.

$$Encrypt_{k1}(m^*) = i^* = Decrypt_{k2}(c^*)$$

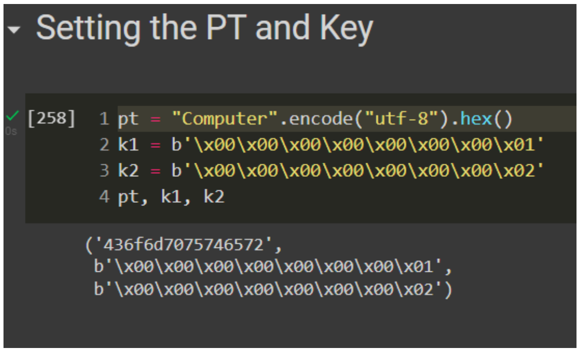
The Attack works as follows:

1. Calculate all DES encryptions (different k_1s) of m^* and store them in an array.
2. Calculate all DES decryptions (different k_2s) of c^* and store them in an array.
3. Find the intersection between the two arrays, and find the corresponding (k_1, k_2).



Chapter 3: Results and Discussion

Setting PT and sample Key:

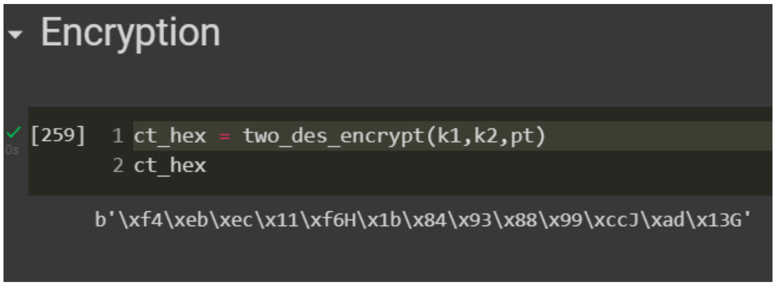


Setting the PT and Key

```
[258]
pt = "Computer".encode("utf-8").hex()
k1 = b'\x00\x00\x00\x00\x00\x00\x00\x01'
k2 = b'\x00\x00\x00\x00\x00\x00\x00\x02'
pt, k1, k2
```

```
('436f6d7075746572',
 b'\x00\x00\x00\x00\x00\x00\x00\x01',
 b'\x00\x00\x00\x00\x00\x00\x00\x02')
```

Encryption:

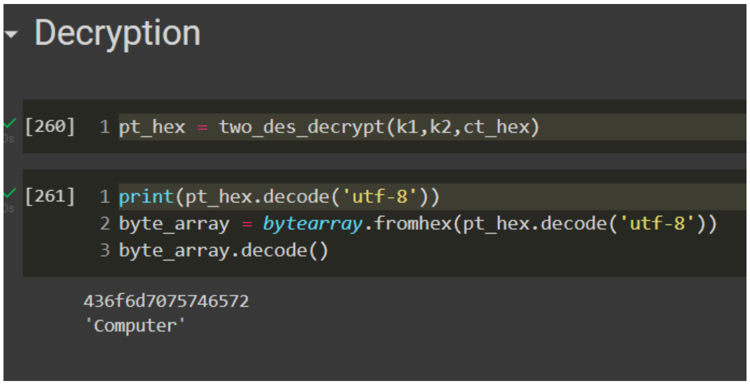


Encryption

```
[259]
ct_hex = two_des_encrypt(k1,k2,pt)
ct_hex
```

```
b'\xf4\xeb\xec\x11\xf6H\x1b\x84\x93\x88\x99\xccJ\xad\x13G'
```

Decryption:



Decryption

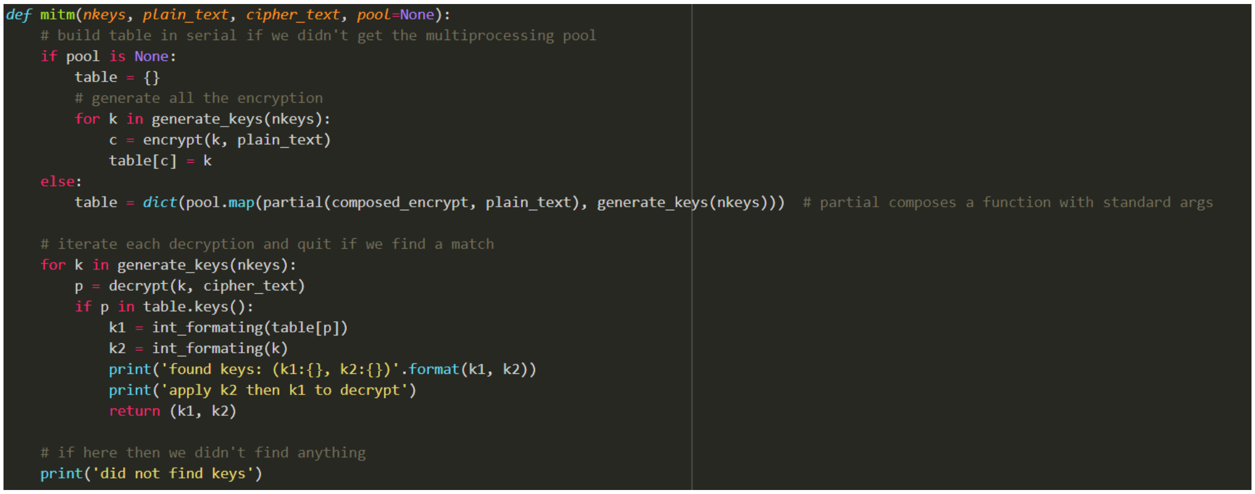
```
[260]
pt_hex = two_des_decrypt(k1,k2,ct_hex)
```

```
[261]
print(pt_hex.decode('utf-8'))
byte_array = bytearray.fromhex(pt_hex.decode('utf-8'))
byte_array.decode()
```

```
436f6d7075746572
'Computer'
```

Meet in the Middle Attack:

Function Definition:

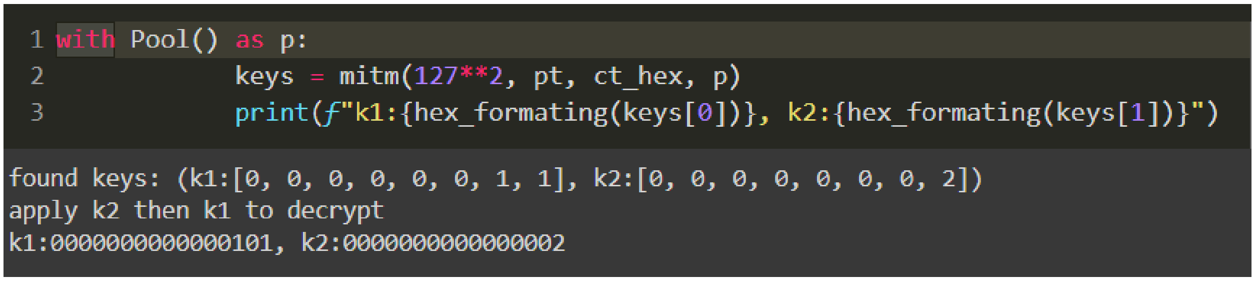


```python
def mitm(nkeys, plain_text, cipher_text, pool=None):
    # build table in serial if we didn't get the multiprocessing pool
    if pool is None:
        table = {}
        # generate all the encryption
        for k in generate_keys(nkeys):
            c = encrypt(k, plain_text)
            table[c] = k
    else:
        table = dict(pool.map(partial(composed_encrypt, plain_text), generate_keys(nkeys)))  # partial composes a function with standard args

    # iterate each decryption and quit if we find a match
    for k in generate_keys(nkeys):
        p = decrypt(k, cipher_text)
        if p in table.keys():
            k1 = int_formating(table[p])
            k2 = int_formating(k)
            print('found keys: (k1:{}, k2:{})'.format(k1, k2))
            print('apply k2 then k1 to decrypt')
            return (k1, k2)

    # if here then we didn't find anything
    print('did not find keys')
```

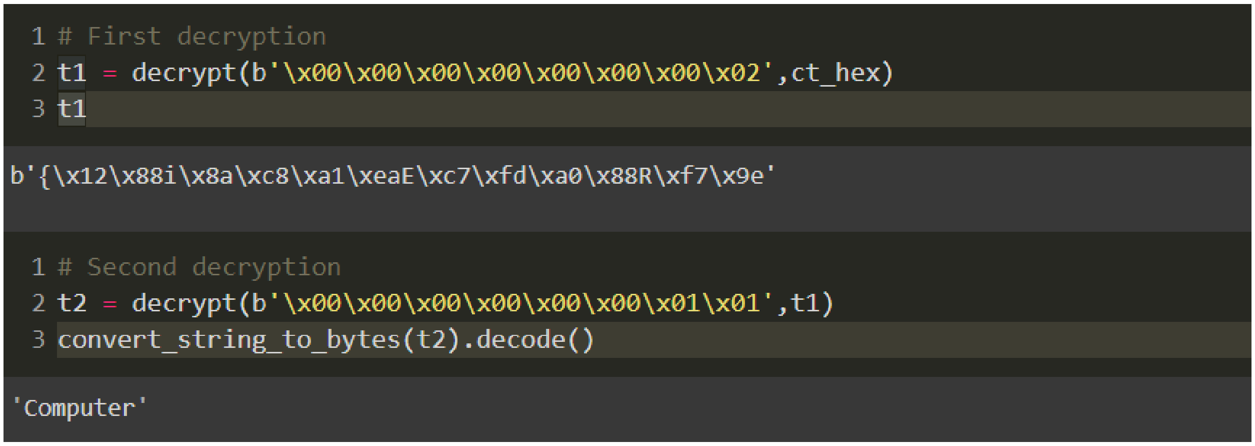
Execution:



```python
with Pool() as p:
            keys = mitm(127**2, pt, ct_hex, p)
            print(f"k1:{hex_formating(keys[0])}, k2:{hex_formating(keys[1])}")
```

```
found keys: (k1:[0, 0, 0, 0, 0, 0, 1, 1], k2:[0, 0, 0, 0, 0, 0, 0, 2])
apply k2 then k1 to decrypt
k1:0000000000000101, k2:0000000000000002
```

Checking:



```python
# First decryption
t1 = decrypt(b'\x00\x00\x00\x00\x00\x00\x00\x02',ct_hex)
t1
```

```
b'{\x12\x88i\x8a\xc8\xa1\xeaE\xc7\xfd\xa0\x88R\xf7\x9e'
```

```python
# Second decryption
t2 = decrypt(b'\x00\x00\x00\x00\x00\x00\x01\x01',t1)
convert_string_to_bytes(t2).decode()
```

```
'Computer'
```

Chapter 4: Conclusion and Future Work

There is a concept of effective/actual key strength. For Double DES the effective key strength is 2^{57} even though double DES uses 2^{112} keys. The below example will make it clear.

Assume that we are a cryptanalyst who has access to plain text and encrypted text. Our aim is to recover the secret key. Assume X (plaintext) -> Y (After 1st encryption) -> Z (after 2nd encryption).

We start with X and try all the 2^{56} combinations for the secret keys by encrypting X. This will give us a big list of possible values for X. Next, we take Z and try all the 2^{56} combinations for the secret key by decrypting Z. This will give us a big list of possible values for X.

The amount of effort you have put in $\mathbf{2^{56} + 2^{56} = 2^{57}}$

Now we do a simple lookup between the two lists to find a matching value. As soon as we see a matching value X in both lists, you have found the secret key. So this means that with the effort of 2^{57} keys we have broken the encryption.

Here MITM helps the brute force attack by dividing the attack into two parts thus decreasing the amount of time required.

Triple DES

Now considering they came up with Triple DES where they used encryption and decryption 3 times. This led to a huge increase in the number of keyspaces.

For meet in the middle attack, it can be applied but the amount of encryption and decryption will come to $2^{56}+2^{112} \approx 2^{112}$ which makes it difficult to compute as for encrypting it would take 2^{56} and for decryption, it would take around 2^{112} because of double des which has to be decrypted together.

References:

[1] Mallik, Avijit & Ahsan, Abid & Shahadat, Mhia & Tsou, Jia-Chi. (2019). Man-in-the-middle-attack: Understanding in simple words. 3. 77-92. 10.5267/j.ijdns.2019.1.001.

[2] Van Oorschot PC, Wiener MJ (1990) A known-plaintext attack, on two-key triple encryption. In: Dawgard I (ed) Advances in cryptology – EUROCRYPT'90. Lecture notes in computer science, vol 473. Springer, Berlin, pp 318–325

[3] http://www.crypto-it.net/eng/attacks/meet-in-the-middle.html

[4] https://github.com/Ajith-Kumar-G/Clg-Labs/tree/main/VI%20sem/CNS_AAT