Simulation and practical implementation of the weak keys, semi weak keys and possible weak keys in DES

Ivan Nikolov

April 2024

***Abstract* - DES, also know as the Data Encryption Standard is a standard that was in commercial use up untill the 1990s, after which it was cracked because of its relative small key size. The small key size was'nt the only problem with DES, there was also a problem of certain keys that did not have the desired properties that were needed to be a good key for the encryption and in this short paper we will be reviewing and going over them. They did not pose significant risk to the algorithm because of how little few they are but it could lead to some kind of information being gained from the system that will help with the cracking.**

1 Introduction

The Data Encryption Standard (DES) is a symmetric-key encryption method developed by IBM in the 1970s, originally based on work by Horst Feistel. It was created in response to a call by the National Bureau of Standards (NBS) for proposals to secure electronic government data. After consultation with the National Security Agency (NSA), a modified version of IBM's algorithm was chosen and published as a Federal Information Processing Standard (FIPS) in 1977. Despite its influential role in the history of cryptography, DES's 56 bit key length is now considered too short for secure use in modern applications.

2 Implementation

2.1 Key Scheduling

The DES algorithm is structured so that it consists of 16 rounds of encryption and decryption. The main difference between encryption and decryption is the keys that are used and generated in each round. The keys are generated in the following manner.

First we drop every 8-th bit from the 64 bit key that we are using and get a 56 bit key (or in hex from 16 to 14) using the PC1 permutation. After that we split the 56 bit key in to two equal in length halves, which I will label as l and r. In every round we will shift these 2 halves by a certain amount, depending on the round number. In the rounds 1,2,9,16 we shift the bits by 1 bit to the left and in all of the other rounds we shift it 2 bits to the left (all of the shifts are circular). If we are decrypting we will be doing the same thing just to the right, but we will ignore the shift to the right in the round 1 and the keys that will be used are in the opposite order. After each bitshift we concatenate the l and r in to a 56 bit length sequence and we use the PC2 permutation on them to change the bits, this output, which i will call the encoded key for the round, or just encodedKey for short is sent to the main encoding function and is used in the encryption/ decryption.

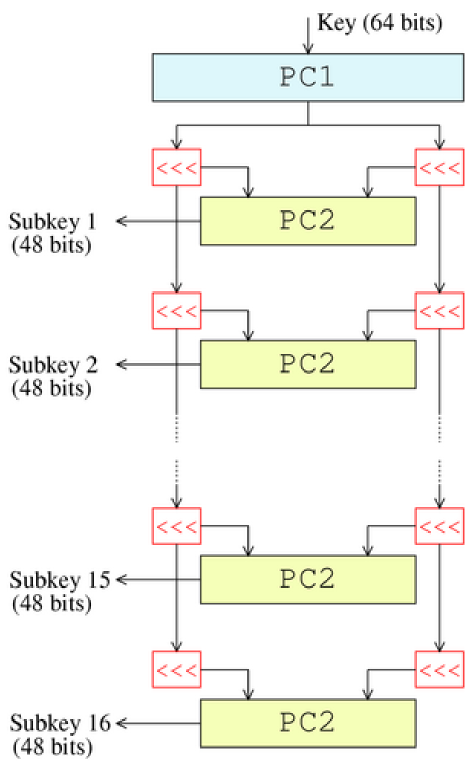

Figure 1:
A visualisation of the key scheduling algorithm in DES

The bulk of this paper will focus on the consequences of this key scheduling algorithm and how it affects the entire DES, but first we need to explain how

DES actually functions on a single round basis. After that explanation we will come back to the key scheduling and how it creates weak, semi-weak and possible weak keys.

Listing 1: How the keys are generated

```
def generateKeys(key):
    checkLen(key, 'Key not adequate length (64)', 64)
    keys = []
    l = setBits(key, tables('PCL'))
    r=setBits(key, tables('PCR'))
    for i in range(16):
        l = roundShift(l, i + 1, 1)
        r = roundShift(r, i + 1, 1)
        keys.append(setBits(l + r, tables('PC2')))
    return keys
```

2.2 Deeper look in to the DES implementation

At the start of the encoding or decoding (encoding will be used because it is essentially the same just the allocation of the keys is in a different order), the plaintext that is of length 64 is changed using the initial permutation table (all of the tables will be provided in the end in the appendix).

Listing 2: Initial permutation

```
def initial_permutation(bits):
    return setBits(bits, tables('IP'))

pt = initial_permutation(pt)
```

The setbits function is a function that assigns the bits from a bit sequence that is provided to it and the lookuptable that tells where they should be assigned. Note, not all of the functions will be explained in detail and not all of the functions are the same as in the implementation, for a more detail view I encourage the reader to view the github repo of the proejct.

Listing 3: How setBits() works

```
def setBits(bits, table):
    newbits = ""
    for i in range(len(table)):
        newbits += bits[table[i] - 1]
    return newbits
```

After the IP the plaintext is divided in to two halves of 32 bit length labled l and r respectivley. The right hlaf is inserted in to the Feistel function, which I will label as f(), along side the key for that round. After the the output of the f() is calculated iti s then XOR-ed with the left half of the plaintext, this output I will call b. At the end of the round the plaintext is reconstructed again but the right hand side, the text r, is at the start of of the text and the output b is concateneted to the right hand side of the text r. This is the new plaintext and it is sent again in to the next round and this is done 16 times. After all of the 16th rounds we have to revert the plaintext using the inverse of the inital permutation and that is the output of the DES encrytpion. The entire encoding process goes like this.

Listing 4: How encode() works

```
def encode(pt, key):
    pt = initial_permutation(pt)
    keys = generateKeys(key)
    for i in range(16):
        l, r = pt[:32], pt[32:]
        b = xor(l, feistel(r, keys[i]), 32)
        pt = r + b
    return inverse_initial_permutation(pt[32:] + pt[:32])
```

2.3 Deeper look in to the Feistel function and its implementation

The Feistel function is essential in the DES algorithm and it is where the non linearity and all of the substitution and diffusion is going on. It consists of 4 main components:

- E - Expansion layer,where we take the plaintext that is of length 32 and we expand it to a size of 48 so that it can be used with the 48 length of the key.
- XOR - XOR layer, here we XOR the output of the E, the output is 48 bits again
- S - S boxes layer, where we substitute the bits from the input using special S boxes, which represent 8 different predefined look up tables. They will be provided in the apendix.
- P - Permutation layer, we rearrange the bits using a look up table (this is different from the IP or the IIP).

The Expansion layer consists of a 32 length plaintext that has its bits locations changed using a lookuptable and some of the bits are also duplicated in this layer to get to the size of 48 so that we match the same size as the key for the next layer, the XOR layer.

The XOR layer XORS the key for the round with the output of the expansion.

The S boxes take the output of the XOR layer and they split it in to 8 blocks of 6 bit length and each of these is sent to a different SBox. The first block is sent to S1, the second block is sent to S2 etc.

The block is then again divided in to 2 parts, row and col.

1	1	0	1	0	1

From here we group the first and the last bits, in this case they are 1 and 1 and that is the row. We group the bits from the second to the fifth in another group, that is the col. These bits give us the binary version of the row and col for the row and col for the correct SBox. In this case we are in the first Sbox for the first block and we need to find the element that is located in the 11 row and the 1010 column, or the 3rd row and the 10th column.

In this case in the lookuptable S1 it is the value 3. We convert the value 3 from decimal to binary once again and that is the output of the first SBox.

We do this for all of the other remaining block and SBox pairs and we concatonate the output in the order that they are to get a result of 32 bits length. It is important to emphasize that the SBoxes are non linear, the proof is trivial (just compute S1(x1) XOR S1(x2) and S1(x1 XOR x2)).

The final layer of the Feistel function is the permutation layer where we simply change the order of the bits using a lookuptable.

Listing 5: The final version of the function should be as follows

```
def feistel(r, key):
    r = expansion(r)
    r = xor(r, key)
    r = s_boxes(r)
    return perumtation(r)
```

3 The problems with the key scheduling algorithm

In total DES has a key size of 2^{64}, but in reality because of the dropping of the parity bits (every 8th bit), the actually key size is 2^{56}. The key size at the time of DES inception (1970s) was strong and reliable and was only susceptible to differential and linear cryptanalysis , but to strengthen the algorithm changes were made. The key size was strong enough until the 1990s where Moore's law had caught up with the algorithm and beat it. Because of the way the keys are distributed there are certain keys that are undesirable and they are categorized in the separate categories:

- Weak keys - Weak keys are those keys who after removing the parity bits are only made up of 0s,1s or half 0s and half 1s.
- Semi weak keys - Semi weak keys are keys that create only 2 round keys and they are repeated 8 times in the 16 rounds.
- Possible weak keys- Possible weak keys are keys that create only 4 round keys and they are repeated 4 times in the 16 rounds.

Out of these 2 categories weak keys and semi weak keys are the most important.

3.1 Weak keys

As we have already defined, weak keys are keys that generate only a single round key for all of the rounds. There are in total 4 of them, and they are:

- 0101010101010101'
- 'FEFEFEFEFEFEFEFE'
- 'E0E0E0E0F1F1F1F1'
- '1F1F1F1F0E0E0E0E

If we wish to remove the parity bits, we get the next combinations:

HEX format	Parity bits dropped
0101010101010101	0000000 0000000
1F1F1F1F0E0E0E0E	0000000 FFFFFFF
E0E0E0E0F1F1F1F1	FFFFFFF 0000000
0101010101010101	FFFFFFF FFFFFFF

As we can see we have 2 possible values for l and r (left and right halves), and they are all 0 or all F. If we shift a sequence of all 0s it doesn't matter, the sequence will always be 0. In what ever way we shift them we will always get the same output. To note is that the left and right do not have any kind of transfer of bits from one to another, they are totally independent from each other. If that were not the case then this problem wouldn't have the same structure.

As we know the DES algorithm gets it's safety from the key that is used, if the key is known that there is no use and all of the information is known. We have only one point of failure and that is the key. If we encrypt our plaintext with a weak key we do get it encrypted but if we encrypt it again we get our plaintext back. This is because the only difference between encryption and decryption is the order that the keys are used and we only have one round key.

```
For every weak key, the following is true
EK(EK(P))=P because EK(P)=DK(P)
```

Listing 6: Visualisation of an encoding using a weak key.

```
Weak key being used is 0101010101010101
Initial plain text 9FDDD1943D9305CD
This is the key generated in round 1 :000000000000
The plain text is : AF1093216C70F5D8
This is the key generated in round 2 :000000000000
The plain text is : 6C70F5D80AD9373B
This is the key generated in round 3 :000000000000
The plain text is : 0AD9373BC1D01E7F
...
...
This is the key generated in round 15 :000000000000
The plain text is : 0DA5777571505EAD
This is the key generated in round 16 :000000000000
The plain text is : 71505EAD7B7874A3
...
The plaintext at the end is C3490E5AFCD7FC03
Number of unique keys is 1, and they are {'000000000000'}
```

As we can see we do have the encoded message and the roundkey is the same in all 16 rounds.

Listing 7: Visualisation of double encoding using the same weak key.

```
Plain text is 9FDDD1943D9305CD
Encoded is C3490E5AFCD7FC03
Encoded again is 9FDDD1943D9305CD
```

The best way to check if a message is encoded using a weak key we can simplly encode it two times and check if the result is the same as the plaintext.

3.2 Semi weak keys

As we have already defined, semi weak keys are keys that generate only two round keys and they are repeated 8 times in the 16 rounds. There are in total 12 of them, 6 pairs of two, and they are:

HEX format	Key 1	Key 2
011F011F010E010E	0000004319BD	000000BCE642
FEE0FEE0FEF1FEF1	FFFFFF4319BD	FFFFFFBCE642
FE1FFE1FFE0EFE0E	6EAC1AFFFFFF	9153E5FFFFFF
E01FE01FF10EF10E	9153E5BCE642	6EAC1A4319BD
...	...	...
...	...	...

Listing 8: Visualisation of an encoding using a semi weak key.

```
Semi weak key being used is 1F011F010E010E01
Initial plain text 9FDDD1943D9305CD
This is the key generated in round 1 :000000BCE642
The plain text is : AF1093216452D5F1
This is the key generated in round 2 :0000004319BD
The plain text is : 6452D5F1E822B1BD
This is the key generated in round 3 :0000004319BD
The plain text is : E822B1BDD00A3298
This is the key generated in round 4 :0000004319BD
The plain text is : D00A3298C585987F
...
...
This is the key generated in round 13 :000000BCE642
The plain text is : 53450DF6351543CA
This is the key generated in round 14 :000000BCE642
The plain text is : 351543CAD8222D88
This is the key generated in round 15 :000000BCE642
The plain text is : D8222D880041BAB1
This is the key generated in round 16 :0000004319BD
The plain text is : 0041BAB1A80DE95D
...
The plaintext at the end is 3708115D0B4E254E
Number of unique keys is 2, and they are
    {'000000BCE642', '0000004319BD'}
```

For the semi weak keys the the property that EK=DK for a given plantext P does not uphold because we have two different round keys, but there is a different property. As we stated earlier, there are 6 pairs and we can use these pairs because each pair creates the same round keys but the order is different. If the K1 from the pair creates E1, then K2 from the pair creates E2 and in the secound round they swap, the k1 creates E2 and K2 creates E1. Because of this property the following is true. EK1=DK2 or EK1(EK2(P))=P

Listing 9: Visualisation of an encoding using a semi weak key.

```
Semi weak key being used is 011F011F010E010E
Initial plain text 9FDDD1943D9305CD
This is the key generated in round 1 :0000004319BD
The plain text is : AF1093215D52DDF8
This is the key generated in round 2 :000000BCE642
The plain text is : 5D52DDF8E807F004
This is the key generated in round 3 :000000BCE642
The plain text is : E807F004FD3CB2FE
This is the key generated in round 4 :000000BCE642
The plain text is : FD3CB2FEA1958CCD
...
...
This is the key generated in round 13 :0000004319BD
The plain text is : 16D40B151D0A677E
This is the key generated in round 14 :0000004319BD
The plain text is : 1D0A677E4340E1DF
This is the key generated in round 15 :0000004319BD
The plain text is : 4340E1DFF480FE8D
This is the key generated in round 16 :000000BCE642
The plain text is : F480FE8DB474B516
...
The plaintext at the end is 0609DF0ADDDC98EE
Number of unique keys is 2, and they are
    {'0000004319BD', '000000BCE642'}
```

From both the visualisations we can see that the round keys are the same just in reverse order,which is how we defined our encryption and decryption. It‘s logical that EK1(EK2(P))=P is true.

Listing 10: Visualisation of double encoding using a pair of semi weak keys.

```
        encoded=encode(PT,SWK[0])
        encoded2=encode(encoded,SWK[1])

If we compare the the outputs with the original
message we get.

Plain text is 9FDDD1943D9305CD
Encoded is 0609DF0ADDDC98EE
Encoded again is 9FDDD1943D9305CD
```

3.3 Possible weak keys

As we have already defined, possible weak keys are keys that generate only two four keys and they are repeated 4 times in the 16 rounds. There are in total 48 of them, 12 pairs of 4, and they are:

HEX format	Key 1	Key 2	Key 3	Key 4
01011F1F01010E0E	000000D17D47	0000009264FA	0000006D9B05	0000002E82B8
1F1F01010E0E0101	000000D17D47	0000009264FA	0000006D9B05	0000002E82B8
E0E01F1FF1F10E0E	5D23662E82B8	338F7C9264FA	CC70836D9B05	A2DC99D17D47
E0E0FEFEF1F1FEFE	FFFFFF6D9B05	FFFFFF9264FA	FFFFFF2E82B8	FFFFFFD17D47
...	...	...	...	...
...	...	...	...	...

Listing 11: Visualisation of an encoding using a possible weak key.

```
Possible weak key being used is 01011F1F01010E0E
Initial plain text 9FDDD1943D9305CD
This is the key generated in round 1 :000000D17D47
The plain text is : AF1093217F46FD70
This is the key generated in round 2 :0000006D9B05
The plain text is : 7F46FD70E53EF612
This is the key generated in round 3 :0000009264FA
The plain text is : E53EF6121F3CB9F1
This is the key generated in round 4 :0000006D9B05
The plain text is : 1F3CB9F194A42EE3
This is the key generated in round 5 :0000009264FA
The plain text is : 94A42EE37EBF7C50
...
...
This is the key generated in round 13 :0000002E82B8
The plain text is : DB3B4ACD28C47291
This is the key generated in round 14 :000000D17D47
The plain text is : 28C47291F7B14B78
This is the key generated in round 15 :0000002E82B8
The plain text is : F7B14B7862B7E11A
This is the key generated in round 16 :0000009264FA
The plain text is : 62B7E11AC2146527
...
The plaintext at the end is 2DE3350232ADCC68
Number of unique keys is 4, and they are
{'0000002E82B8', '0000009264FA', '0000006D9B05', '000000D17D47'}
```

Listing 12: Visualisation of an encoding using a semi weak key.

```
For the values  000000D17D47, 0000006D9B05, 0000009264FA,
0000002E82B8 there is a group of 4 keys
that generate them, and they are:
    01011F1F01010E0E,
    011F1F01010E0E01,
    1F01011F0E01010E,
    1F1F01010E0E0101
```

These keys don't have such obvious encryption/decryption symmetry properties as the weak and semi-weak keys, but they nonetheless produce a much simpler key schedule than usual, which might conceivably be exploited somehow.

HexFormat	01011F1F01010E0E	011F1F01010E0E01	1F01011F0E01010E	1F1F01010E0E0101
Round	Key 1	Key 2	Key 3	Key 4
1	000000D17D47	0000009264FA	0000006D9B05	0000002E82B8
2	0000006D9B05	000000D17D47	0000002E82B8	0000009264FA
3	0000009264FA	0000002E82B8	000000D17D47	0000006D9B05
4	0000006D9B05	000000D17D47	0000002E82B8	0000009264FA
5	0000009264FA	0000002E82B8	000000D17D47	0000006D9B05
...	...	...	...	...
...	...	...	...	...
13	0000002E82B8	0000006D9B05	0000009264FA	000000D17D47
14	000000D17D47	0000009264FA	0000006D9B05	0000002E82B8
15	0000002E82B8	0000006D9B05	0000009264FA	000000D17D47
16	0000009264FA	0000002E82B8	000000D17D47	0000006D9B05

4 Analysis and conclusion

In total there are 4+12+48 = 64, which means the probabily of picking a key that isn't a strong key is $64/2^{56} = 1/2^{50}$ which is extremely unlikely and they should not be taken in to consideration in most casese. Some scholars even say that it gives more information to the attack if the system actually avoids these types of keys, but non the less, no analytical attack has proven effective against DES and still to this day the easiest way to attack and decrypt DES is just by using brute force for the key space.

5 Appendix

Here are all of the lookuptables that have been used in the implementation of DES.They will be in the form of lists so that they can be easily copied.

IP = [58, 50, 42, 34, 26, 18, 10, 2, 60, 52, 44, 36, 28, 20, 12, 4, 62, 54, 46, 38, 30, 22, 14, 6, 64, 56, 48, 40, 32, 24, 16, 8, 57, 49, 41, 33, 25, 17, 9, 1, 59, 51, 43, 35, 27, 19, 11, 3, 61, 53, 45, 37, 29, 21, 13, 5, 63, 55, 47, 39, 31, 23, 15, 7]

IIP = [40, 8, 48, 16, 56, 24, 64, 32, 39, 7, 47, 15, 55, 23, 63, 31, 38, 6, 46, 14, 54, 22, 62, 30, 37, 5, 45, 13, 53, 21, 61, 29, 36, 4, 44, 12, 52, 20, 60, 28, 35, 3, 43, 11, 51, 19, 59, 27, 34, 2, 42, 10, 50, 18, 58, 26, 33, 1, 41, 9, 49, 17, 57, 25]

P = [16, 7, 20, 21, 29, 12, 28, 17, 1, 15, 23, 26, 5, 18, 31, 10, 2, 8, 24, 14, 32, 27, 3, 9, 19, 13, 30, 6, 22, 11, 4, 25]

PCL = [57, 49, 41, 33, 25, 17, 9, 1, 58, 50, 42, 34, 26, 18, 10, 2, 59, 51, 43, 35, 27, 19, 11, 3, 60, 52, 44, 36]

PCR = [63, 55, 47, 39, 31, 23, 15, 7, 62, 54, 46, 38, 30, 22, 14, 6, 61, 53, 45, 37, 29, 21, 13, 5, 28, 20, 12, 4]

PC2 = [14, 17, 11, 24, 1, 5, 3, 28, 15, 6, 21, 10, 23, 19, 12, 4, 26, 8, 16, 7, 27, 20, 13, 2, 41, 52, 31, 37, 47, 55, 30, 40, 51, 45, 33, 48, 44, 49, 39, 56, 34, 53, 46, 42, 50, 36, 29, 32]

EXPANSION = [32, 1, 2, 3, 4, 5, 4, 5, 6, 7, 8, 9, 8, 9, 10, 11, 12, 13, 12, 13, 14, 15, 16, 17, 16, 17, 18, 19, 20, 21, 20, 21, 22, 23, 24, 25, 24, 25, 26, 27, 28, 29, 28, 29, 30, 31, 32, 1]

S1 = [[14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7], [0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8], [4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0], [15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13]]

S2 = [[15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10], [3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5], [0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15], [13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9]]

S3 = [[10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8], [13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1], [13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7], [1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12]]

S4 = [[7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15], [13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9], [10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4], [3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14]]

S5 = [[2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9], [14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6], [4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14], [11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3]]

S6 = [[12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11], [10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8], [9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6], [4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13]]

S7 = [[4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1], [13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6], [1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2], [6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12]]

S8 = [[13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7], [1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2], [7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8], [2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11]]

[2][1] [3]

References

[1] Lars R Knudsen and John E Mathiassen. On the role of key schedules in attacks on iterated ciphers. In *Computer Security–ESORICS 2004: 9th European Symposium on Research in Computer Security, Sophia Antipolis, France, September 13-15, 2004. Proceedings 9*, pages 322–334. Springer, 2004.

[2] Soe Soe Mon, Khin Aye Thu, and Thida Soe. Analysis of weak keys on des algorithm. *International Journal of Advance Research and Innovative Ideas in Education*, 5:940–949, 2019.

[3] Judy H. Moore and Gustavus J. Simmons. Cycle structure of the des with weak and semi-weak keys. In Andrew M. Odlyzko, editor, *Advances in Cryptology — CRYPTO' 86*, pages 9–32, Berlin, Heidelberg, 1987. Springer Berlin Heidelberg.