Rectangle Cipher

Hex Brains

Department of Computer Science And Engineering
Indian Institute of Technology Bhilai

December 4, 2022

Outline

Points

- Block Cipher
- 16 4X4 S-Boxes in parallel in S-Layer
- 3 rotations composed in P-layer
- Both Hardware and Software Friendly

Outline

Cipher Specifications

- Lightweight Block Cipher
- Bit-Slice Style
- Competetive Software Performance
- Hardware Friendly
- Very Strong Security

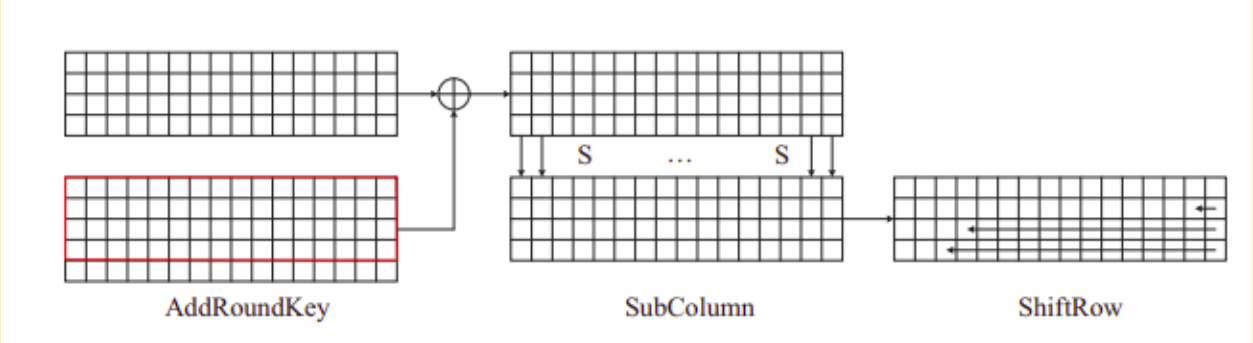
THE ROUND TRANSFORMATION

- AddRoundkey(ARK)
- SubColumn(SC)
- ShiftROw(SR)
- KeySchedule(KS)

Pseudo-Code-

```
GenerateRoundKeys(state):
    for i = 0 to 24 do:
        ARK(state, K_i)
        SC(state)
        SR(state)
    ARK(state, K_25)
```

Shorter Visualization



Differential Distribution Table (DDT)

```
sage: from sage.crypto.sbox import SBox
sage: S = SBox(6,5,12,10,1,14,7,9,11,0,3,13,8,15,4,2)
sage: S
(6, 5, 12, 10, 1, 14, 7, 9, 11, 0, 3, 13, 8, 15, 4, 2)
sage: S.difference_distribution_table()
....:
[16  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
[ 0  0  0  2  0  0  4  2  0  0  0  2  0  0  4  2]
[ 0  0  0  0  0  0  2  2  2  0  2  0  2  4  0  2]
[ 0  0  0  2  0  0  2  0  2  4  2  2  2  0  0  0]
[ 0  0  0  4  0  0  0  4  0  0  0  4  0  0  0  4]
[ 0  2  0  0  4  2  0  0  4  2  0  0  0  2  0  0]
[ 0  2  4  0  2  0  0  0  0  0  0  2  2  2  0  2]
[ 0  0  4  0  2  2  0  0  0  2  0  2  2  0  0  2]
[ 0  2  0  2  0  2  0  2  0  2  0  2  0  2  0  2]
[ 0  2  0  0  0  2  4  0  0  2  0  0  0  2  4  0]
[ 0  0  0  0  0  4  2  2  2  0  2  0  2  0  0  2]
[ 0  4  0  2  0  0  2  0  2  0  2  2  2  0  0  0]
[ 0  0  0  0  4  0  0  0  4  0  4  0  0  0  4  0]
[ 0  2  0  0  0  2  0  0  0  2  4  0  0  2  4  0]
[ 0  0  4  2  2  2  0  2  0  2  0  0  2  0  0  0]
[ 0  2  4  2  2  0  0  2  0  0  0  0  2  2  0  0]
sage:
```

Linear Approximation Table (LAT)

```
sage: from sage.crypto.sbox import SBox
sage: S = SBox(6,5,12,10,1,14,7,9,11,0,3,13,8,15,4,2)
sage: S
(6, 5, 12, 10, 1, 14, 7, 9, 11, 0, 3, 13, 8, 15, 4, 2)
sage: S.linear_approximation_table()
[ 8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
[ 0  0  0  4  0 -4  0  0  2 -2 -2 -2 -2 -2  2 -2]
[ 0  0  0  0  0  0  4  4  0  0  4 -4  0  0  0  0]
[ 0  0  0 -4  4  0  0  0 -2  2 -2 -2 -2 -2  2 -2]
[ 0  0  0  0  0  0 -4  4  0  0  0  0  0  0  4  4]
[ 0  0 -4  0  0 -4  0  0 -2  2 -2 -2  2  2 -2  2]
[ 0  0  0  0  0  0  0  0  4  4  0  0 -4  4  0  0]
[ 0  0 -4  0 -4  0  0  0 -2  2  2  2 -2 -2  2 -2]
[ 0  0  0 -4 -2 -2  2 -2  0 -4  0  0 -2  2  2  2]
[ 0  0  0  0 -2  2  2 -2  2  2 -2 -2  4  0  4  0]
[ 0  0  0 -4 -2 -2 -2  2  4  0  0  0  2 -2 -2 -2]
[ 0  0  0  0  2 -2  2 -2  2  2  2  2  0 -4  0  4]
[ 0  4  0  0 -2  2 -2 -2  0  0  0 -4 -2 -2 -2  2]
[ 0  4  4  0 -2 -2  2  2 -2  2 -2  2  0  0  0  0]
[ 0 -4  0  0 -2  2  2  2  0  0 -4  0 -2 -2 -2  2]
[ 0  4 -4  0  2  2  2  2  2 -2 -2  2  0  0  0  0]
sage:
```

Key Schedule

For 80-bit key

1. SC to the bits at the 4 uppermost rows and the 4 rightmost columns
2. Using a 1-round generalized Feistel transformation
   Row'0 := (Row0 « 8) $\oplus$ Row1
   Row'1 := Row2
   Row'2 :=Row3
   Row'3 := (Row3 « 12) $\oplus$ Row4
   Row'4 := Row0
3. A 5-bit round constant RC[i] is XORed with the 5-bit key state for i $\in$ (1,2,..,24).

Key Schedule

For 128-bit key

1. SC to the bits at the 8 rightmost columns.
2. Using a 1-round generalized Feistel transformation
   Row'0 := (Row0 « 8) $\oplus$ Row1
   Row'1 := Row2
   Row'2 := (Row2 « 16) $\oplus$ Row3 Row'3 := Row0
3. A 5-bit round constant is XORed with the 5-bit key state

Security Analysis

Integral Cryptanalysis

- We implemented the Square attack which used a 4-round integral distinguisher
- Encryption : After 4-rounds, the XOR sum in any 4 bit positions equals to 0, i.e. (Balanced property) $\oplus$ S4[0] $= \oplus$ S4[17] $= \oplus$ S4[43] $= \oplus$ S4[60] $= 0$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63

- Decryption : We choose 2^{48} plaintexts s.t. cols - 0, 13, 14, 15 maintain CONSTANT property and other 12 cols maintain the ALL property.
- 2^{48} Intermediate values 2^{47} subsets 2 values.
- $4 \rightarrow 7 \rightarrow 25$ rounds with same integral distinguisher.

SECURITY ANALYSIS

Differential Cryptanalysis

- Differential Cryptanalys is strongest techniques for the cryptanalysis of block ciphers
- Using the algorithm based on the branch and bound method, the best differential trails from round-1 to round-15 were found.

$\sharp R$	Prob.	$\sharp$R	Prob.	$\sharp$R	Prob.
1	2^{-2}	6	2^{-18}	11	2^{-46}
2	2^{-4}	7	2^{-25}	12	2^{-51}
3	2^{-7}	8	2^{-31}	13	2^{-56}
4	2^{-10}	9	2^{-36}	14	2^{-61}
5	2^{-14}	10	2^{-41}	15	2^{-66}

- Using the 14-round differential propagation, we can mount an attack on 18-round Rectangle cipher
- 25-round Rectangle is enough to behold out against this differential cryptanalysis attack.

Outline

Code

Encryption

encryptor.py

```python
        # print("beforeee",state)
        state = subcolumn(state)
        print("after sbox",state)
        # print("afterrr",state)
        shiftrows(state)
        print("after shift",state)
        # print(state)
        keyupdate(key , j)
        # print("----------------------------------------\n")
        print(state)
        # print("key")
        # print(key)
        print("----------------------------------------\n")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
['11011100101000 10', '0001011100110001', '0100010011110000', '0011111010100110']
----------------------------------------

Round : 21
after adding key ['0111000101110011', '1011110110000001', '0011010010100101', '0100101011110010']
added key ['1010110111010001', '1010101010110000', '0111000001010101', '0111010001010100', '1110111011000001']
after sbox ['0011111000100101', '0000111100101000', '1000010010101000', '1011001100100001']
after shift ['0011111000100101', '0001111001010000', '1000100001001010', '0011011001100100']
['0011111000100101', '0001111001010000', '1000100001001010', '0011011001100100']
----------------------------------------

Round : 22
after adding key ['0100111000011111', '0110101000000001', '0111000111001110', '1101010000101001']
added key ['0111000000111010', '0111010001010001', '1111100110000100', '1110001001001101', '0111111100000000']
after sbox ['1010000111111001', '1110101000101110', '0010000000011111', '1001101000101111']
after shift ['1010000111111001', '1110101000101101', '1111001000000001', '1111001101000101']
['1010000111111001', '1101010001011101', '1111001000000001', '1111001101000101']
----------------------------------------

Round : 23
after adding key ['0110101000101000', '0011011000010010', '0111001100100101', '1100101101100100']
added key ['1100101111010001', '1110001001001111', '1000000100100100', '0011100000100001', '0100011100110010']
after sbox ['1111000001101001', '1101001011100000', '0101010010111010', '1110010101110110']
after shift ['1111000001101001', '1010010111000001', '1010010101001011', '1101110010101110']
['1111000001101001', '1010010111000001', '1010010101001011', '1101110010101110']
----------------------------------------

Round : 24
after adding key ['1111011001110001', '1001110111100100', '1011000111111011', '0001011000110000']
added key ['0000011000011000', '0011100000100101', '0001010010110000', '1100101010011110', '0011011110000111']
after sbox ['1100010111011010', '0011000110110001', '0110110110010101', '1010110001010101']
after shift ['1100010111011010', '0110001101100010', '0101011011011001', '1011010110001010']
['1100010111011010', '0110001101100010', '0101011011011001', '1011010110001010']
----------------------------------------

FINAL STATE
['1110010001010010', '1010100111110110', '0010110111110111', '0101010000100011']
hexform of cipher is: 4a276f95efb4c42a
shreyas@shreyas-HP-Pavilion-Laptop-15-cc1xx:~/Downloads/rectangle_cipher$
```

Code

Decryption

```
def decrypt(cipher,ke):
    print("let's start",cipher)
    k = ke
    temp=""
    key=[]
    for i in range(len(k)):
        res=int(k[i],16)
        bn=bin(res).replace("0b","")
        bn='0'*(4-len(bn))+bn
        temp+=bn
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
rev shift ['0101011101011111', '1110000011101101', '0000001100111001', '0010101101011111']
rev sbox ['1010001110011011', '1110001110010010', '1001110010001011', '1100101111011101']
['0000011101101110', '1000010101111000', '1101110101011100', '0101110011111010']
currentkey ['1010010011110101', '0110011011101010', '0100000111010111', '1001011100100111', '0010111111011110']
------------------------------------
before doing ['0000001110110110', '1000010101111000', '1101110101011100', '0101110011111010']
rev shift ['0000011101101110', '0100010101111100', '1110101011001101', '1110011111010010']
rev sbox ['0100111011010010', '1001010101111111', '0111000010001111', '0011111101100001']
['1010111011100001', '1000000001010010', '0000101001000011', '0101100110000010']
currentkey ['1110001100110011', '0001010100101101', '0111101011001100', '0110011011100011', '1110011110111001']
------------------------------------
before doing ['1010111011100001', '1000000001010010', '0000101001000011', '0101100110000010']
rev shift ['1010111011100001', '0100000000101001', '1010010000110000', '1100110000010010']
rev sbox ['1111110100001100', '1100110000111001', '0000101011101010', '1001010100001111']
['0101110001000001', '0110010000101101', '0101000000110001', '1000000000100100']
currentkey ['1010000101001101', '1010100000010100', '0101101011011011', '0001010100101011', '1010101110011110']
------------------------------------
before doing ['0101110001000001', '0110010000101101', '0101000000110001', '1000000000100100']
rev shift ['0101110001000001', '1011001000010110', '0000001100010101', '0000000100100100']
rev sbox ['1010000010001111', '1011000100000010', '1110110001100110', '1011000010101010']
['0000000000001111', '1001110000110001', '1101001110001001', '0001100010110100']
currentkey ['1010000010000000', '0010101001101011', '0011111111101111', '1010100000011110', '0001000001011010']
------------------------------------
before doing ['0000000000001111', '1001110000110001', '1101001110001001', '0001100010110100']
rev shift ['0000000000001111', '1100111000011000', '0011100010011101', '1100010110100000']
rev sbox ['1100011001000101', '1111101010001000', '0011001100101010', '0011001101100000']
['1001110110101100', '0000011010000010', '1100100110110101', '0001111001011010']
currentkey ['0101101111101001', '1111000000001010', '1111101010011111', '0010110100111010', '0000110100111100']
------------------------------------
before doing ['1001110110101100', '0000011010000010', '1100100110110101', '0001111001011010']
rev shift ['1001110110101100', '0000001101000001', '1001101101011100', '1111001011010000']
rev sbox ['1001101011001111', '1001000110010001', '0110011111100001', '1111000000000011']
['0100100001010101', '0000111101111100', '0000000000000000', '0000000000000000']
currentkey ['1100100101011010', '1001111011101101', '0110011111100001', '1111000000000011', '0101010110011111']
------------------------------------
decrypted text:
['0100100001010101', '0000111101111100', '0000000000000000', '0000000000000000']
hexform of plaintext is: aa123ef000000000
shreyas@shreyas-HP-Pavilion-Laptop-15-cc1xx:~/Downloads/rectangle_cipher$
```

Software Application

client-server application

- server.py-opens socket and waits for the client to connect and share message.
- client.py-establishes connection i.e binds with socket of server.py
- encryptor.py-contains the encryption of the RECTANGLE cipher
- decryptor.py- contains the decryption of the RECTANGLE cipher

Software

client.py | decryptor.py | encryptor.py

client.py

```python
server.send(bytes(string,"utf-8"))
ack = server.recv(1024)
ack = ack.decode("utf-8")
print(ack)

if(string=="E"):
    # print("Enter a 16 length multiple string to encrypt")

    toenc = input("Enter a message in hex form: ")
    print("OK!!!Sending your data to encrypt")
    time.sleep(1)
    server.send(bytes(toenc,"utf-8"))
```

server.py | encryptor.py

server.py

```python
import socket
import encryptor
import decryptor
import os

s = "server"
ip = "127.0.0.4"
port = 1240

server = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
server.bind((ip,port))
server.listen(5)

client,address = server.accept()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
shreyas@shreyas-HP-Pavilion-Laptop-15-cc1xx:~/Downloads/part_4$ python3 client.py
What do you want to perform Encryption or Decryption
 Write E for Encryption
 Write D for Decryption
Else write EXIT for quitting the connectionE
OK
Enter a message in hex form: 1234abcd
OK!!!Sending your data to encrypt
this is a message from server.......
the encrypted text is e779c8197f39d909 and the key is 52b1ef542b4571876195!!
 Keep the key secret

What do you want to perform Encryption or Decryption
 Write E for Encryption
 Write D for Decryption
Else write EXIT for quitting the connectionD
OK
Enter the encrypted messagee779c8197f39d909
Enter the key52b1ef542b4571876195
OK!!!Sending your data to decrypt
this is a message from server.......
the decrypted text is : 1234abcd00000000

What do you want to perform Encryption or Decryption
 Write E for Encryption
 Write D for Decryption
Else write EXIT for quitting the connection
```

```
shreyas@shreyas-HP-Pavilion-Laptop-15-cc1xx:~/Downloads/part_4$ python3 server.py
connection est - 127.0.0.1 : 55802
GOT RESp
connection est - 127.0.0.1 : 55802
GOT RESp
['e779c8197f39d909', '52b1ef542b4571876195']
connection est - 127.0.0.1 : 55802
```

Outline

Slide One

- Infer that it is similar to AES block cipher.
- It is being found that out of 25 total rounds of encryption, 18 rounds are prone to attack .
- The left over 7-rounds are for security purpose.

Outline

Conclusion

RECTANGLE

1. Bit-slice block cipher.
2. The cipher is optimized a lot to be less prone to many attacks .
3. Provides the application enough flexibility.

Thanks

Team Members

- Shreyas Pande
- Niket Srivastav
- Prathamesh Gujar

Implementation Info

- Github Link: https://github.com/shreyaspande2003/Rectangle-Cipher