Rectangle Cipher

KS KeySentinels

Department of CSE & DSAI
Indian Institute of Technology Bhilai

December 7, 2024

Outline

Introduction

- Lightweight Block Cipher
- Based on SP-Network
- 16 4x4 S-boxes in parallel in S-layer
- 3 rotations composed in the P-layer
- Low-cost implementation in hardware
- Competitive speed in software

Outline

Cipher Specifications

- Bit-Slice Style
- Cipher operates over 25 rounds
- Each round consisting of three core operations: AddRoundKey (ARK), SubColumn (SC), and ShiftRow (SR)

AddRoundkey (AR)

It is simple XOR operation between the round subkey and the state.

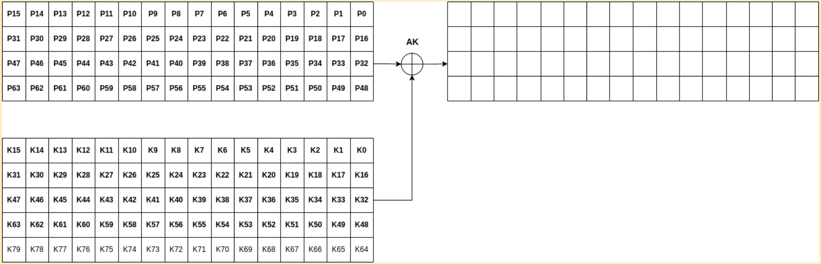

Figure: Above diagram shows add round-key operation

SubColumn (SC)

SubColumn parallels SubBytes, applying S-boxes to the 4 bits in each column of the state matrix.

- Input to the S-box: $\text{Col(j)} = a_{3,j}||a_{2,j}||a_{1,j}||a_{0,j}$ for $0 \leq j \leq 15$
- Output to the S-box: $\text{S(Col(j))} = b_{3,j}||b_{2,j}||b_{1,j}||b_{0,j}$

$$
\begin{array}{ccccc}
\begin{pmatrix} a_{0,15} \\ a_{1,15} \\ a_{2,15} \\ a_{3,15} \end{pmatrix} & \cdots & \begin{pmatrix} a_{0,2} \\ a_{1,2} \\ a_{2,2} \\ a_{3,2} \end{pmatrix} & \begin{pmatrix} a_{0,1} \\ a_{1,1} \\ a_{2,1} \\ a_{3,1} \end{pmatrix} & \begin{pmatrix} a_{0,0} \\ a_{1,0} \\ a_{2,0} \\ a_{3,0} \end{pmatrix} \\
\downarrow S & \cdots & \downarrow S & \downarrow S & \downarrow S \\
\begin{pmatrix} b_{0,15} \\ b_{1,15} \\ b_{2,15} \\ b_{3,15} \end{pmatrix} & \cdots & \begin{pmatrix} b_{0,2} \\ b_{1,2} \\ b_{2,2} \\ b_{3,2} \end{pmatrix} & \begin{pmatrix} b_{0,1} \\ b_{1,1} \\ b_{2,1} \\ b_{3,1} \end{pmatrix} & \begin{pmatrix} b_{0,0} \\ b_{1,0} \\ b_{2,0} \\ b_{3,0} \end{pmatrix}
\end{array}
$$

Figure: Above diagram shows SubColumn operation

S-box of Rectangle Cipher-

ShiftRow (SR)

It is left rotations on the rows of a state matrix, with varying offsets for each row.

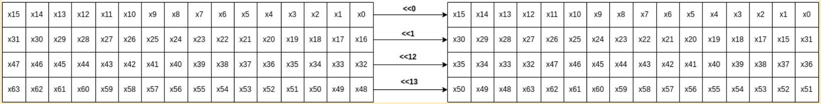

Figure: Above diagram shows shift row operation

Pseudo Code

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

Differential Distribution Table (DDT)

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

Linear Approximation Table (LAT)

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

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
   $Row'_0 := (Row_0 \ll 8) \oplus Row_1$
   $Row'_1 := Row_2$
   $Row'_2 := Row_3$
   $Row'_3 := (Row_3 \ll 12) \oplus Row_4$
   $Row'_4 := Row_0$
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
   $Row'_0 := (Row_0 \ll 8) \oplus Row_1$
   $Row'_1 := Row_2$
   $Row'_2 := (Row_2 \ll 16) \oplus Row_3$ $Row'_3 := Row_0$
3. A 5-bit round constant is XORed with the 5-bit key state

Outline

Differential Attack

- Differential Cryptanalysis is one of the 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 the 18-round Rectangle cipher.
- A 25-round Rectangle cipher is sufficient to withstand this differential cryptanalysis attack.

Integral Attack

- Implemented Square Attack, which uses a 4-round integral distinguisher.
- After 4 rounds, the XOR sum in any 4-bit positions equals 0, i.e., the balanced property:

$$\bigoplus S_{(0,0)} = \bigoplus S_{(1,1)} = \bigoplus S_{(2,13)} = \bigoplus S_{(3,13)} = 0$$

- Visual Representation:

(0,0)	(0,1)	(0,2)	(0,3)	(0,4)	(0,5)	(0,6)	(0,7)	(0,8)	(0,9)	(0,10)	(0,11)	(0,12)	(0,13)	(0,14)	(0,15)
(1,0)	(1,1)	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)	(1,7)	(1,8)	(1,9)	(1,10)	(1,11)	(1,12)	(1,13)	(1,14)	(1,15)
(2,0)	(2,1)	(2,2)	(2,3)	(2,4)	(2,5)	(2,6)	(2,7)	(2,8)	(2,9)	(2,10)	(2,11)	(2,12)	(2,13)	(2,14)	(2,15)
(3,0)	(3,1)	(3,2)	(3,3)	(3,4)	(3,5)	(3,6)	(3,7)	(3,8)	(3,9)	(3,10)	(3,11)	(3,12)	(3,13)	(3,14)	(3,15)

- Decryption: We choose 248 plaintexts such that:
  - Columns 0, 13, 14, 15 maintain the CONSTANT property.
  - Other 12 columns maintain the **ALL** property.
- 2^{48} Intermediate values $\implies$ 2^{47} subsets $\implies$ 2 values.

Outline

Software Implementation

encryptor.py

- Uses a 25-round encryption process with operations: AddRoundKey, SubColumn, and ShiftRows.
- Accepts a 16-character hexadecimal input, padding it if needed.
- Generates a 20-character random key for encryption.

decryptor.py

- Reverses encryption steps using precomputed round keys.
- Computes all 25 round keys beforehand for accurate decryption.
- Outputs the decrypted plaintext as a hexadecimal string.

Outline

Application

Encrypt message in QR codes using RECTANGLE

- encryptor.py - contains the encryption of RECTANGLE cipher
- decryptor.py - contains the decryption of RECTANGLE cipher
- generate_qr.py - encrypts the message and embeds it in QR code
- decrypt_qr.py - scans and retrieves the plaintext from the QR-code

Generated QR Code

Figure: Above QR Codes contain encrypted messages

Outline

Brownie Point Nominations

- Rectangle is based on SP-Network.
- It is slightly similar to AES.
- Out of 25, the maximum of 18-rounds can be attacked.
- The remaining 7-rounds are for security purposes.
- It attains a very fast software as well as hardware performance.

Outline

Conclusion

- It is lightweight bit-slice block cipher.
- It provides applications enough flexibility.
- It has the ability to trigger various new cryptographic problems.
- Its security is encouraged.

Thanks

Team Members

- Siddhi Agarwal (12141570)
- Kriti Gupta (12140940)

Implementation Info

- Github Link: https://github.com/agaSiddhi/RECTANGLE-cipher