Rectangle Cypher

Ruchit Prakash Saxena[1], Anubh Sanoj Gupta[2] and Lavish[3]

[1] IIT Bhilai, Raipur, India, ruchitp@iitbhilai.ac.in
[2] IIT Bhilai, Raipur, India, anubhs@iitbhilai.ac.in
[3] IIT Bhilai, Raipur, India, lavishg@iitbhilai.ac.in

Abstract.
Rectangle Cipher is a bit-slice-styled block cipher, which is lightweight and based on SP-Network, where the bit-slice implementation of the cipher causes its lightweight and fast executions. It attains a very fast and competitive software performance as well as a very low-cost hardware performance. Sixteen 4x4 S-Boxes are present in the substitution layer in parallel and 3 rotations in the permutation layer. Rectangle provides great hardware as well as software performance. So that it offers applications enough flexibility.
Rectangle cipher allows very efficient hardware performance. For example- In the 80-bit key version, a 1-cycle/round parallel execution requires 1600 gates only for throughput about 250-kbits/sec at 3.0 pJ/bit energy efficiency and 100 kHz clock.
Because of the cipher's bit-slice implementation, it attains a fast software performance among all the existing lightweight ciphers. For example in 128-bit SSE instructions, a bit-slice execution of the cipher achieves almost 3.9 cycles/B encryption speed in an average of about 3000B messages.
Rectangle attains a better security performance because of the cipher's Substitution box careful selection and the permutation layer's asymmetric design. The deep analysis of the security reveals that out of 25, a maximum of 18-rounds can be attacked.

Keywords: bit-slice implementation, lightweight block cipher, software efficiency, security performance

1 Introduction

Nowadays, many tiny embedded devices like smart cards, RFIDs, sensor nodes, etc are used in many applications on a large scale, which are generally distinguished by strong cost constraints like energy, area, power in the aspect of hardware and small and optimized code, low memory space in the aspect of the software. Cryptographic Protection is also necessary meanwhile. So that many new lightweight ciphers have been introduced for supplying better security at a lower cost, such as Present, SIMON, Hummingbird, KLEIN, DESL/DESX/DESXL, LBlock, Katan, Piccolo, TWINE, LED, SPECK, and so on.

At CHES'2007, the Present cipher was introduced and became popular due to its better hardware performance, simplicity, and security. It utilizes a bit of permutation in the diffusion layer so that it has a very good hardware performance. Some lightweight ciphers attain a low hardware area while remaining unsuccessful in achieving good software performance, which includes Present, Katan, and Hummingbird. Then at CHES'2011, the LED cipher is introduced where their creators claim that their cipher is not only optimal in hardware performance but also decent in software performance.

However, 2nd-generation lightweight ciphers can be improved by analyzing the idea, success, and exclusion of the 1st-generation proposals. So that the Serpent block cipher was introduced with the new bit-slice technique causing an increase in the software speed

of DES, which is based on the SP-network. It is a 128-bit block cipher, which contains thirty-two 4x4 S-Boxes in the substitution layer. SHA-3, Trivium, JH are 5 other ciphers, which were also profited by the bit-slice mechanism for their software performance. It is worth seeing that these not only perform well in hardware but also in software.

Also, a bit-slice execution is protected from implementation attacks like- cache and timing attacks in contrast with a table-based execution. Though the blueprint goal of all the above ciphers is not lightweight, and for a committed bit-slice lightweight cipher, there is enough field for improvement.

Therefore Rectangle cipher is introduced, based on SP-network which utilizes the bit-slice mechanism. Hence it attains a very fast and competitive software performance as well as a very low-cost hardware performance. Sixteen 4x4 S-Boxes are present in the substitution layer in parallel and 3 rotations in the permutation layer. The top 3 pros of the cipher include very hardware friendly, very fast software speed, and very good security.

Rectangle cipher allows very efficient hardware performance. For example- In the 80-bit key version, a 1-cycle/ round parallel execution requires 1600 gates only for throughput about 250-kbits/sec at 3.0 pJ/bit energy efficiency and 100 kHz clock.

Because of the cipher's bit-slice implementation, it attains a very fast software speed among all the lightweight block ciphers. For example in 128-bit SSE instructions, a bit-slice execution of the cipher achieves almost 3.9 cycles/B encryption speed in an average of about 3000B messages.

Rectangle attains a better security performance because of the cipher's Substitution box careful selection and the permutation layer's asymmetric design. The deep analysis of the security reveals that out of 25, the maximum of 18-rounds can be attacked. The remaining 7-rounds are for security purposes.

2 Rectangle Cipher Implementation

Rectangle cipher is a block cipher, where the block length is 64 bits, which can be seen as a 4x16 rectangle array. And the key length is either 80 or 128 bits. It is a twenty-five round substitution-permutation network cipher, where each of the twenty-five rounds consists of the three steps: AddRoundkey(ARK), SubColumn(SC), ShiftRow(SR). After the last round, there is a final AddRoundKey to get the final cipher block. Its encryption algorithm consists of twenty-five rounds and a final ARK.

The pseudocode for the Rectangle cipher is given below:-

```
GenerateRoundKeys(state):
.     for i = 0 to 24 do:
.         ARK(state, Ki)
.         SC(state)
.         SR(state)
.     ARK(state, K25)
.
```

State (Cipher and Subkey State)- Cipher state is a cluster of a 64-bit plaintext or intermediate result or ciphertext as shown in the figure below. A cipher state is nothing but a 4x16 rectangular array of bits, due to which the cipher name was given "Rectangle". Let W indicates the cipher state, such that W = w63|| ... ||w1||w0, where wi denotes 1-bit word. The 1st 16 bits are arranged in row 0 (i.e. w15|| ... ||w1|| w0), the next 16 bits are arranged in row-1 (i.e. w31|| ... || w17 || w16), and so on. Similarly, a 64-bit subkey can also be seen as a 4×16 rectangular array bits.

$$\begin{bmatrix} w_{15} & \cdots & w_2 & w_1 & w_0 \\ w_{31} & \cdots & w_{18} & w_{17} & w_{16} \\ w_{47} & \cdots & w_{34} & w_{33} & w_{32} \\ w_{63} & \cdots & w_{50} & w_{49} & w_{48} \end{bmatrix}$$

Cipher State

$$\begin{bmatrix} a_{0,15} & \cdots & a_{0,2} & a_{0,1} & a_{0,0} \\ a_{1,15} & \cdots & a_{1,2} & a_{1,1} & a_{1,0} \\ a_{2,15} & \cdots & a_{2,2} & a_{2,1} & a_{2,0} \\ a_{3,15} & \cdots & a_{3,2} & a_{3,1} & a_{3,0} \end{bmatrix}$$

Subkey State

AddRoundkey (ARK)- This operation is nothing but an easy bitwise xor between the round subkey and the state at the intermediate of the algorithm.

SubColumn (S)- SubColumn is similar to the Subbytes operation, where the parallel application of S-boxes gets applied to the 4 bits in the same column. This operation is shown below in the figure. If the input of an S-box is Col(j) = a3,j || a2,j || a1,j || a0,j for $0 \leq j \leq 15$, then the result would be S(Col(j)) = b3,j || b2,j || b1,j || b0,j.

$$\begin{pmatrix} a_{0,15} \\ a_{1,15} \\ a_{2,15} \\ a_{3,15} \end{pmatrix} \cdots \begin{pmatrix} a_{0,2} \\ a_{1,2} \\ a_{2,2} \\ a_{3,2} \end{pmatrix} \begin{pmatrix} a_{0,1} \\ a_{1,1} \\ a_{2,1} \\ a_{3,1} \end{pmatrix} \begin{pmatrix} a_{0,0} \\ a_{1,0} \\ a_{2,0} \\ a_{3,0} \end{pmatrix}$$

$$\downarrow S \quad \cdots \quad \downarrow S \quad \downarrow S \quad \downarrow S$$

$$\begin{pmatrix} b_{0,15} \\ b_{1,15} \\ b_{2,15} \\ b_{3,15} \end{pmatrix} \cdots \begin{pmatrix} b_{0,2} \\ b_{1,2} \\ b_{2,2} \\ b_{3,2} \end{pmatrix} \begin{pmatrix} b_{0,1} \\ b_{1,1} \\ b_{2,1} \\ b_{3,1} \end{pmatrix} \begin{pmatrix} b_{0,0} \\ b_{1,0} \\ b_{2,0} \\ b_{3,0} \end{pmatrix}$$

SubColumn Operation on the State Columns

The S-Box used in Rectangle is denoted as S : $F_2^4 \rightarrow F_2^4$, i.e. a 4-bit to 4-bit S-Box in hexadecimal which is given below in the table-

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S(x)$	6	5	C	A	1	E	7	9	B	0	3	D	8	F	4	2

. Rectangle Cipher S-Box

Differentil Distribution Table (DDT) and Linear Approximation Table (LAT) for the above S-Box are shown in the figures below-

[16	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]

DDT for Rectangle cipher S-Box

[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]

LAT for Rectangle cipher S-Box

ShiftRow (SR)- It is just a left rotation to each row. These rotations are executed over different offsets, like- 0bit for row-0, 1bit for row-1, 12bits for row-2, and 13 bits for row-3. Assume that the left rotation over x bits be indicated by "<<x", then the operation ShiftRow is shown in the figure below-

<< **0** — (a0,15 ... a0,2 a0,1 a0,0) → (a0,15 ... a0,2 a0,1 a0,0)
<< **1** — (a1,15 ... a1,2 a1,1 a1,0) → (a1,14 ... a1,1 a1,0 a1,15)
<< **12** — (a2,15 ... a2,2 a2,1 a2,0) → (a2,3 ... a2,6 a2,5 a2,4)
<< **13** — (a3,15 ... a3,2 a3,1 a3,0) → (a3,2 ... a3,5 a3,4 a3,3)

Key Schedule- Rectangle can accept keys of either 80 or 128 bits.

CASE-1: 80-bit key

Let the user-provided key, in this case, be V, such that V = v79|| ... || v1 || v0 (Since, 80-bit seed key), the key is initially stored in an 80-bit key register and ordered as a 5×16 rectangle array (shown in the figure below).

$$\begin{bmatrix} v_{15} & \cdots & v_2 & v_1 & v_0 \\ v_{31} & \cdots & v_{18} & v_{17} & v_{16} \\ v_{47} & \cdots & v_{34} & v_{33} & v_{32} \\ v_{63} & \cdots & v_{50} & v_{49} & v_{48} \\ v_{79} & \cdots & v_{66} & v_{65} & v_{64} \end{bmatrix}$$

80-bit Key State

$$\begin{bmatrix} \kappa_{0,15} & \cdots & \kappa_{0,2} & \kappa_{0,1} & \kappa_{0,0} \\ \kappa_{1,15} & \cdots & \kappa_{1,2} & \kappa_{1,1} & \kappa_{1,0} \\ \kappa_{2,15} & \cdots & \kappa_{2,2} & \kappa_{2,1} & \kappa_{2,0} \\ \kappa_{3,15} & \cdots & \kappa_{3,2} & \kappa_{3,1} & \kappa_{3,0} \\ \kappa_{4,15} & \cdots & \kappa_{4,2} & \kappa_{4,1} & \kappa_{4,0} \end{bmatrix}$$

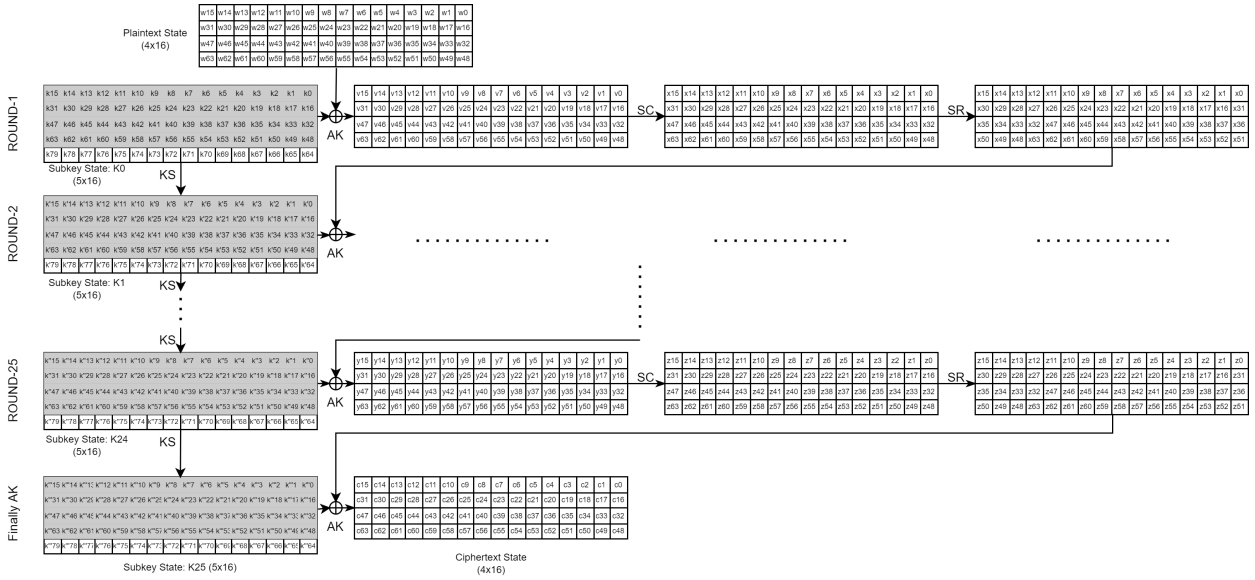
Key's 2D Representation

Suppose Rowi = ki,15|| ... || ki,1|| ki,0 indicate the i-th key register's row, such that $0 \leq i \leq 4$, where Rowi is considered to a sixteen bit word. At round i where i belongs to 0 to24, the sixty-four bit round subkey Ki contains the 1st 4 rows of the key register's contents (as shown in the figure given below), i.e., Ki = Row3 || Row2 || Row1 || Row0. Once the key register Ki found out, the key register gets updated using the following iterations-

1. Using the S-box S, we perform Subcoumn to the partial bits that are intersected at the 4 uppermost rows and the 4 rightmost columns (shaded as grey in the figure given below), i.e., k'3,j || k'2,j || k'1,j || k'0,j := S(k3,j || k2,j || k1,j || k0,j), such that j belongs to (0, 1, 2, 3).

2. Using 1-round generalized Feistel transformation, i.e.,
Row'0 := (Row0 << 8) $\oplus$ Row1,
Row'1 := Row2,
Row'2 := Row3,
Row'3 := (Row3 << 12) $\oplus$ Row4,
Row'4 := Row0

3. Now on the 5-bit key state (i.e. k0,4|| k0,3 || k0,2 || k0,1|| k0,0), we perform a XOR operation with RC[i] (a 5-bit round constant), i.e., k'0,4 || k'0,3 || k'0,2 || k'0,1 || k'0,0 := (k0,4|| k0,3|| k0,2|| k0,1 || k0,0) $\oplus$ RC[i]
Finally, we get K25 using the above iterations on the updated key state. 5-bit LFSR is used to generate the round constants RC[i] for all i belonging to 0 to 24. At each round, the 5 bits (rc4, rc3, rc2, rc1, rc0) are left shifted over 1 bit, with the new value to rc0 which is evaluated as rc4 $\oplus$ rc2. We take RC[0] := 0x1 at the start. Now, using that we can derive all the round constants(for i=0 to 24)- (0X01, 0X02, 0X04, 0X09, 0X12, 0X05, 0X0B, 0X16, 0X0C, 0X19, 0X13, 0X07, 0X0F, 0X1F, 0X1E, 0X1C, 0X18, 0X11, 0X03, 0X06, 0X0D, 0X1B, 0X17, 0X0E, 0X1D).



. 80-bit key block diagram from 80.png

CASE-2: 128-bit key

Let the user-provided key, in this case, be V, such that V = v127|| ... || v1 || v0 (Since, one hundred and twenty eight bit seed key), the key is initially taken in a key register of 128-bit, which can be ordered as a 4x32 rectangle array. The corresponding 2D representation of the key state is shown below-

$$\begin{bmatrix} \kappa_{0,31} & \cdots & \kappa_{0,2} & \kappa_{0,1} & \kappa_{0,0} \\ \kappa_{1,31} & \cdots & \kappa_{1,2} & \kappa_{1,1} & \kappa_{1,0} \\ \kappa_{2,31} & \cdots & \kappa_{2,2} & \kappa_{2,1} & \kappa_{2,0} \\ \kappa_{3,31} & \cdots & \kappa_{3,2} & \kappa_{3,1} & \kappa_{3,0} \end{bmatrix}$$

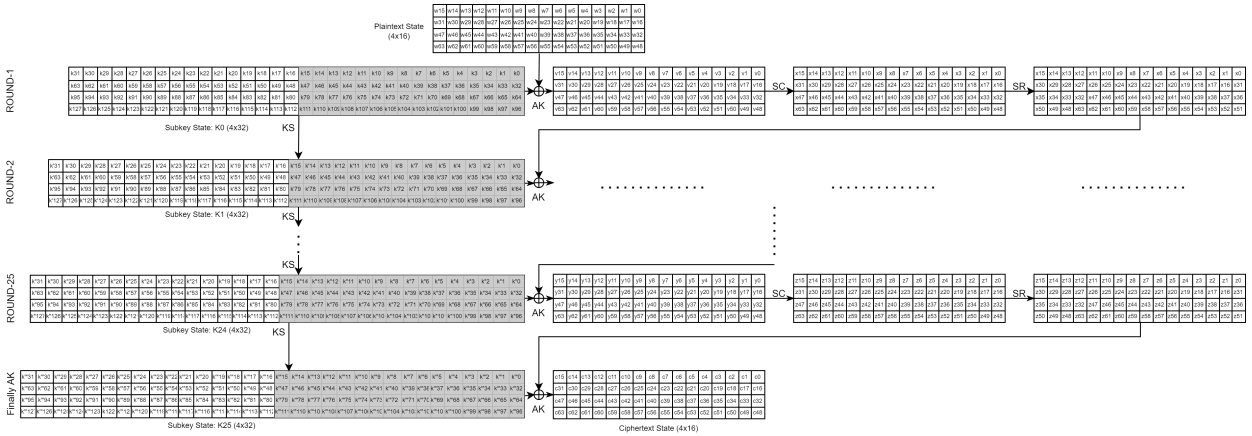
128-bit key's 2-D Representation

Suppose Rowi = ki,31|| ... || ki,1|| ki,0 indicate the i-th key register's row, such that 0 ≤ i ≤ 3, where Rowi is considered to a thirty-two bit word. At round i where i belongs to 0 to 24, the 64-bit round subkey Ki contains the rightmost 16 columns of the current contents of the key (shown in the figure below). Once the round subkey Ki is found out , the key register gets updated using the following iterations-

1. Using the S-box S, we perform Subcoumn to the partial 8 rightmost columns (shaded in the figure given below), i.e., k'3,j || k'2,j|| k'1,j || k'0,j := S(k3,j || k2,j || k1,j || k0,j), such that 0 ≤ j ≤ 7

2. Using 1-round generalized Feistel transformation, i.e.,
Row'0 := (Row0 << 8) ⊕ Row1,
Row'1 := Row2,
Row'2 := (Row2 << 16) ⊕ Row3,
Row'3 := Row0

3. Now on the 5-bit key state (k0,4|| k0,3|| k0,2|| k0,1|| k0,0), we perform a XOR operation with RC[i] (a 5-bit round constant), where round constants RC[i] for i belongs to 0 to 24 are the same used above in the 80-bit key schedule. Then finally, we get K25 using the above iterations on the updated key state.



. 128-bit key block diagram from 128.png

3 Security Analysis

3.1 Integral Cryptanalysis

4 Software Implementation

5 Conclusion

Rectangle, a lightweight block cipher has been proposed, with a bit-slice technique based on SP-Network, which causes its lightweight and fast executions. It attains a very fast and competitive software performance as well as a very low-cost hardware performance. 16 4×4 S-Boxes are present in the substitution layer in parallel and 3 rotations in the permutation layer. Rectangle provides great hardware as well as software performance. So that it offers applications enough flexibility. Rectangle attains a better security performance because of the cipher's Substitution box careful selection and the permutation layer's asymmetric design. It is believed that the Rectangle cipher is a simple and interesting design and it has the ability to trigger various new cryptographic problems. And at the end, Rectangle cipher security is encouraged further.

6 References

1. Research Paper sources- https://eprint.iacr.org/2014/084.pdf
2. Image (80-bit key and 128-bit key)- https://app.diagrams.net/
3. Image sources- http://eprint.iacr.org/