RECTANGLE: A Bit-slice Lightweight Block Cipher Suitable for Multiple Platforms

Wentao Zhang[1], Zhenzhen Bao[1], Dongdai Lin[1], Vincent Rijmen[2], Bohan Yang[2], Ingrid Verbauwhede[2]

1.State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China
2.KU Leuven, Dept. of Electrical Engineering ESAT/COSIC and iMinds, Security Dept.
{*zhangwentao, baozhenzhen, ddlin*}*@iie.ac.cn*
{*vincent.rijmen, bohan.yang, ingrid.verbauwhede*}*@esat.kuleuven.be*

Abstract. In this paper, we propose a new lightweight block cipher named RECTANGLE. The main idea of the design of RECTANGLE is to allow lightweight and fast implementations using bit-slice techniques. RECTANGLE uses an SP-network. The substitution layer consists of 16 4×4 S-boxes in parallel. The permutation layer is composed of 3 rotations. As shown in this paper, RECTANGLE offers great performance in both hardware and software environment, which provides enough flexibility for different application scenario. The following are 3 main advantages of RECTANGLE. First, RECTANGLE is extremely hardware-friendly. For the 80-bit key version, a one-cycle-per-round parallel implementation only needs 1600 gates for a throughput of 246 Kbits/sec at 100 KHz clock and an energy efficiency of 3.0 pJ/bit. Second, RECTANGLE achieves a very competitive software speed among the existing lightweight block ciphers due to its bit-slice style. Using 128-bit SSE instructions, a bit-slice implementation of RECTANGLE reaches an average encryption speed of about 3.9 cycles/byte for messages around 3000 bytes. Last, but not least, we propose new design criteria for the RECTANGLE S-box. Due to our careful selection of the S-box and the asymmetric design of the permutation layer, RECTANGLE achieves a very good security-performance tradeoff. Our extensive and deep security analysis shows that the highest number of rounds that we can attack, is 18 (out of 25).

Key words: lightweight cryptography, block cipher, design, bit-slice, hardware efficiency, software efficiency

1 Introduction

Small embedded devices (including RFIDs, sensor nodes, smart cards) are now widely used in many applications. They are usually characterized by strong cost constraints, such as area, power, energy consumption for hardware aspect, and low memory, small code size for software aspect. Meanwhile, they also require cryptographic protection. As a result, many new lightweight ciphers have been proposed to provide strong security at a lower cost than standard solutions. Since symmetric-key ciphers, especially block ciphers, play an important role in the security of small embedded devices, the design of lightweight block ciphers has been a very active research topic over the last few years.

In the literature, quite a few lightweight block ciphers with various design strategies have been proposed, such as DESL/DESX/DESXL [35], Hummingbird [25], KATAN/ KTANTAN [22], KLEIN [28], LBlock [52], LED[30], Piccolo [48], PRESENT [14], SIMON and SPECK [3], TWINE [49] and so on. PRESENT was proposed at CHES'2007, and has attracted a lot of attention from cryptographic researchers due to its simplicity, impressive hardware performance and strong security. The design of PRESENT is extremely hardware-efficient, since it uses a bit permutation as its diffusion layer, which is a simple wiring in hardware implementation. In 2012, PRESENT was adopted as ISO/IEC lightweight cryptography standard. Many

This paper is the full version. Please cite the following journal version: Wentao Zhang, Zhenzhen Bao, Dongdai Lin, Vincent Rijmen, Bohan Yang, Ingrid Verbauwhede. RECTANGLE: A Bit-slice Lightweight Block Cipher Suitable for Multiple Platforms, SCIENCE CHINA Information Sciences, to appear.

lightweight ciphers, including PRESENT, KATAN/KTANTAN and Hummingbird, succeed in achieving a low area in hardware but the software performance is not good. For example, the permutation layer of PRESENT is extremely low-cost in hardware, but it is the true performance bottleneck for many software implementations. However, high software performance is also needed from the same algorithm for many classical lightweight applications, as pointed out in [3, 4, 28, 30, 36]. LED is proposed at CHES'2011, the designers claim that LED is not only very compact in hardware but also maintains a reasonable performance profile for software implementation.

Among the new proposals, some present weaknesses, including ARMODILLO-2, Hummingbird-1 and KTANTAN [15, 41, 46]. Furthermore, as pointed out in [30], designers of "second generation" lightweight ciphers can learn from the progress and the omissions of the"first generation" proposals. The S-box of PRESENT is mainly selected according to its hardware area instead of security of the underlying cipher. Hence, the S-box of PRESENT is "weak" with respect to cipher security. As pointed out in [33], the PRESENT S-box is among the 8 percent worst S-boxes with respect to clustering of one bit linear trails. Along with the strong symmetry of the PRESENT permutation layer, there are very serious clustering problems both for linear trails and differential trails [12, 16, 33, 42, 50]. We give more details in Section 3. As a result, for PRESENT, the best distinguisher so far can reach 24 rounds [16], which can be used to mount a shortcut attack on 26-round PRESENT (out of 31).

The bit-slice technique was introduced for speeding up the software speed of DES [6], and was used in the design of the Serpent block cipher [2]. In a bit-slice implementation, one software logical instruction corresponds to simultaneous execution of n hardware logical gates, where n is the length of a subblock. Take Serpent for example. Serpent is a 128-bit SP-network block cipher. The substitution layer is composed of 32 4×4 S-boxes, thus the subblock length is $n = 128/4 = 32$ for a bit-slice implementation. JH [51], Keccak(SHA-3) [5], Noekeon [19] and Trivium [23] are 4 other primitives that can benefit from the bit-slice technique for their software performance. It is worth noticing that JH, Keccak, Noekeon, Serpent and Trivium not only perform well in hardware but also in software. Furthermore, a bit-slice implementation is safe against implementation attacks such as cache and timing attacks compared with a table-based implementation [39]. However, the main design goal of all the mentioned bit-sliced ciphers is not "lightweight", and there is plenty of room for improvement when it comes to a dedicated lightweight block cipher with bit-slice style.

1.1 Contributions

In this paper, we present a new lightweight block cipher RECTANGLE. The design of RECTANGLE makes use of the bit-slice technique in a lightweight manner, hence to achieve not only a very low cost in hardware but also a very competitive performance in software. As a result, RECTANGLE adopts the SP-network structure. The substitution layer (S-layer) consists of 16 4×4 S-boxes in parallel. The permutation layer (P-layer) is composed of 3 rotations. The following are 3 main advantages of RECTANGLE:

1. RECTANGLE is extremely hardware-friendly. The bit-sliced design principle of RECTANGLE allows for very efficient and flexible hardware implementations. For the 80-bit key version, using UMC $0.13\mu m$ standard cell library at 100 KHz , our round-based implementation could obtain a throughput of 246 Kbits/sec and an energy efficiency of 3.0 pJ/bit with only 1600 gates, and our serialized implementation could obtain a throughput of 14.0 Kbits/sec and an energy efficiency of 32.05 pJ/bit with only 1111 gates. The round-based implementation can also be easily extended to parallel implementation. More details are given in Section 5.1.
2. Due to its bit-slice style, RECTANGLE achieves a very competitive software speed among the existing lightweight block ciphers. The S-box of RECTANGLE can be implemented using a sequence of 12 basic logical instructions. The P-layer of RECTANGLE is composed of 3 rotations, which makes it very friendly for both hardware and software implementations. On a 2.5GHz Intel(R) Core i5-2520M CPU, for one block data, our bit-sllice implementation gives a speed of about 30.5 cycles/byte for encryption and 32.2 cycles/byte for decryption; with a parallel mode of operation, a bit-slice implementation of RECTANGLE reaches an average encryption speed of about 3.9 cycles/byte for messages around 3000 bytes, using Intel 128-bit SSE instructions. In addition, our implementations of RECTANGLE on Atmel studio show that RECTANGLE also has a very impressive performance on 8-bit microcontrollers. More

details are given in Section 5.2. We expect that RECTANGLE also has very good performance on 16- and 32-bit microcontrollers.

3. Last but not least. We propose new design criteria for the RECTANGLE S-box. Due to our careful selection of the RECTANGLE S-box, together with the asymmetric design of the P-layer, RECTANGLE achieves a very good security-performance tradeoff. After our extensive and deep security analysis, we can mount a shortcut attack on 18-round RECTANGLE (out of 25), which is the highest number of rounds that we can attack.

This paper is organized as follows. Section 2 presents a specification of RECTANGLE; Section 3 discusses the security of RECTANGLE against known attacks; Section 4 motivates the design choices of RECTANGLE; Section 5 presents the hardware and software implementation results of the cipher; Section 6 presents the relation of RECTANGLE to several early designs. Section 7 concludes the paper.

2 The RECTANGLE Block Cipher

RECTANGLE is an iterated block cipher. The block length is 64 bits, and the key length is 80 or 128 bits.

2.1 The Cipher State and the Subkey State

A 64-bit plaintext, or a 64-bit intermediate result, or a 64-bit ciphertext is collectively called as a cipher state. A cipher state can be pictured as a 4×16 rectangular array of bits, which is the origin of the cipher name **RECTANGLE**. Let $W = w_{63}||\cdots||w_1||w_0$ denote a cipher state, the first 16 bits $w_{15}||\cdots||w_1||w_0$ are arranged in row 0, the next 16 bits $w_{31}||\cdots||w_{17}||w_{16}$ are arranged in row 1, and so on, as illustrated in Fig. 1. A 64-bit subkey is similarly pictured as a 4×16 rectangular array. In the following, for convenience of description, a cipher state is described in a two-dimensional way, as illustrated in Fig. 2.

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

Fig. 1. A Cipher State

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

Fig. 2. Two-dimensional Way

2.2 The Round Transformation

RECTANGLE is a 25-round SP-network cipher. Each of the 25 rounds consists of the following 3 steps: AddRoundkey, SubColumn, ShiftRow. After the last round, there is a final AddRoundKey.

AddRoundkey: A simple bitwise XOR of the round subkey to the intermediate state.

SubColumn: Parallel application of S-boxes to the 4 bits in the same column. The operation of SubColumn is illustrated in Fig. 3. The input of an S-box is $Col(j) = a_{3,j}||a_{2,j}||a_{1,j}||a_{0,j}$ for $0 \leq j \leq 15$, and the output is $S(Col(j)) = b_{3,j}||b_{2,j}||b_{1,j}||b_{0,j}$.

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

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

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

Fig. 3. SubColumn Operates on the Columns of the State

The S-box used in RECTANGLE is a 4-bit to 4-bit S-box $S : F_2^4 \rightarrow F_2^4$. The action of this S-box in hexadecimal notation is given by the following 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

ShiftRow: A left rotation to each row over different offsets. Row 0 is not rotated, row 1 is left rotated over 1 bit, row 2 is left rotated over 12 bits, and row 3 is left rotated over 13 bits. Let $\lll x$ denote left rotation over x bits, the operation ShiftRow is illustrated in Fig. 4.

$$(a_{0,15} \cdots a_{0,2}\ a_{0,1}\ a_{0,0}) \xrightarrow{\lll 0} (a_{0,15} \cdots a_{0,2}\ a_{0,1}\ a_{0,0})$$

$$(a_{1,15} \cdots a_{1,2}\ a_{1,1}\ a_{1,0}) \xrightarrow{\lll 1} (a_{1,14} \cdots a_{1,1}\ a_{1,0}\ a_{1,15})$$

$$(a_{2,15} \cdots a_{2,2}\ a_{2,1}\ a_{2,0}) \xrightarrow{\lll 12} (a_{2,3} \cdots a_{2,6}\ a_{2,5}\ a_{2,4})$$

$$(a_{3,15} \cdots a_{3,2}\ a_{3,1}\ a_{3,0}) \xrightarrow{\lll 13} (a_{3,2} \cdots a_{3,5}\ a_{3,4}\ a_{3,3})$$

Fig. 4. ShiftRow Operates on the Rows of the State

2.3 Key Schedule

RECTANGLE can accept keys of either 80 or 128 bits.
80-bit key For a 80-bit seed key (user-supplied key) $V = v_{79}||\cdots||v_1||v_0$, the key is firstly stored in a 80-bit key register and arranged as a 5×16 array of bits, see Fig. 5.

$$\begin{bmatrix} v_{15} \cdots v_2 & v_1 & v_0 \\ v_{31} \cdots v_{18} & v_{17} & v_{16} \\ v_{47} \cdots v_{34} & v_{33} & v_{32} \\ v_{63} \cdots v_{50} & v_{49} & v_{48} \\ v_{79} \cdots v_{66} & v_{65} & v_{64} \end{bmatrix} \qquad \begin{bmatrix} \kappa_{0,15} \cdots \kappa_{0,2} & \kappa_{0,1} & \kappa_{0,0} \\ \kappa_{1,15} \cdots \kappa_{1,2} & \kappa_{1,1} & \kappa_{1,0} \\ \kappa_{2,15} \cdots \kappa_{2,2} & \kappa_{2,1} & \kappa_{2,0} \\ \kappa_{3,15} \cdots \kappa_{3,2} & \kappa_{3,1} & \kappa_{3,0} \\ \kappa_{4,15} \cdots \kappa_{4,2} & \kappa_{4,1} & \kappa_{4,0} \end{bmatrix}$$

Fig. 5. A 80-bit Key State and its Two-dimensional Representation

Let $Row_i = \kappa_{i,15}||\cdots||\kappa_{i,1}||\kappa_{i,0}$ denote the i-th row of the key register, $0 \leq i \leq 4$. Row_i can be regarded as a 16-bit word. At round i ($i = 0, 1, \cdots, 24$), the 64-bit round subkey K_i consists of the first 4 rows of the current contents of the key register, i.e., $K_i = Row_3||Row_2||Row_1||Row_0$. After extracting K_i, the key register is updated as follows:

1. Applying the S-box S to the bits intersected at the 4 uppermost rows and the 4 rightmost columns, i.e.,
$$\kappa'_{3,j}||\kappa'_{2,j}||\kappa'_{1,j}||\kappa'_{0,j} := S(\kappa_{3,j}||\kappa_{2,j}||\kappa_{1,j}||\kappa_{0,j}), \quad j = 0, 1, 2, 3$$
2. Applying a 1-round generalized Feistel transformation, i.e.,
$Row'_0 := (Row_0 \lll 8) \oplus Row_1$,
$Row'_1 := Row_2$,
$Row'_2 := Row_3$,
$Row'_3 := (Row_3 \lll 12) \oplus Row_4$,
$Row'_4 := Row_0$
3. A 5-bit round constant RC[i] is XORed with the 5-bit key state $(\kappa_{0,4}||\kappa_{0,3}||\kappa_{0,2}||\kappa_{0,1}||\kappa_{0,0})$, i.e.,
$$\kappa'_{0,4}||\kappa'_{0,3}||\kappa'_{0,2}||\kappa'_{0,1}||\kappa'_{0,0} := (\kappa_{0,4}||\kappa_{0,3}||\kappa_{0,2}||\kappa_{0,1}||\kappa_{0,0}) \oplus \mathrm{RC}[i]$$

Finally, K_{25} is extracted from the updated key state. The round constants RC[i] ($i = 0, 1, \cdots, 24$) are generated by a 5-bit LFSR. At each round, the 5 bits $(rc_4, rc_3, rc_2, rc_1, rc_0)$ are left shifted over 1 bit, with the new value to rc_0 being computed as $rc_4 \oplus rc_2$. The initial value is defined as RC[0] := $0x1$. We list all the round constants in Appendix A.

128-bit key For a 128-bit seed key, the key is firstly stored in a 128-bit key register and arranged as

a 4×32 array of bits. The corresponding two-dimensional representation of the 128-bit key state is as follows:

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

Let $Row_i = \kappa_{i,31}||\cdots||\kappa_{i,1}||\kappa_{i,0}$ denote the i-th row of the key register, $0 \leq i \leq 3$. Row_i can be regarded as a 32-bit word. At round $i (i = 0, 1, \cdots, 24)$, the 64-bit round subkey K_i consists of the 16 rightmost columns of the current contents of the key. After extracting the round subkey K_i, the key register is updated as follows:

1. Applying the S-box S to the 8 rightmost columns, i.e.,
$$\kappa'_{3,j}||\kappa'_{2,j}||\kappa'_{1,j}||\kappa'_{0,j} := S(\kappa_{3,j}||\kappa_{2,j}||\kappa_{1,j}||\kappa_{0,j}), \quad 0 \leqslant j \leqslant 7$$
2. Applying a 1-round generalized Feistel transformation, i.e.,
$Row'_0 := (Row_0 \lll 8) \oplus Row_1$,
$Row'_1 := Row_2$,
$Row'_2 := (Row_2 \lll 16) \oplus Row_3$,
$Row'_3 := Row_0$
3. A 5-bit round constant RC$[i]$ is XORed with the 5-bit key state $(\kappa_{0,4}||\kappa_{0,3}||\kappa_{0,2}||\kappa_{0,1}||\kappa_{0,0})$, where RC$[i]$ $(i = 0, 1, \cdots, 24)$ are the same as those used in the 80-bit key schedule.

Finally, K_{25} is extracted from the updated key state.

2.4 The Cipher

The encryption algorithm of RECTANGLE consists of 25 rounds and a final subkey XOR. The following is a pseudo C code:

$GenerateRoundKeys( )$
for i = 0 to 24 do
{ $AddRoundKey(STATE, K_i)$
$SubColumn(STATE)$
$ShiftRow(STATE)$
}
$AddRoundKey(STATE, K_{25})$

3 Security Analysis

In this section, we present the results of our security analysis of RECTANGLE.

3.1 Differential Cryptanalysis

Differential [9] and linear [37] cryptanalysis are among the most powerful techniques available for block ciphers. To attack an n-bit block cipher using differential cryptanalysis (DC), there must be a predictable difference propagation over all but a few rounds with a probability significantly larger than 2^{1-n}. A difference propagation is composed of a set of differential trails, where its probability is the sum of the probabilities of all differential trails that have the specified input difference and output difference [20]. For RECTANGLE, to be resistant against DC, it is a necessary condition that there is no difference propagation with a probability higher than 2^{-63}.

M.Matsui has presented a search algorithm for the best differential/linear trail of DES in [38], which uses branch-and-bound methods. Based on this algorithm, we have written a program to search for the best differential trails of RECTANGLE from 1 round to 15 rounds , and the results are presented in Table 1. The probability of the best 15-round differential trail is 2^{-66}.

Table 1. Probabilities of the Best Differential Trails of RECTANGLE

♯ R	Prob.	♯ R	Prob.	♯ 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}

Because of the simplicity of the ShiftRow transformation, we also need to consider the security of RECTANGLE against multiple differential cryptanalysis [12] and the structure attack [50]. From a differential point of view, since all the operations in RECTANGLE have rotational symmetry, every trail has up to 16 rotation equivalent variants. For 15-round RECTANGLE, based on the branch-and-bound algorithm, we have searched for all the differential trails with probability between 2^{-66} and 2^{-76} (up to a rotation equivalence) and examined all the difference propagations made up of the investigated trails. The following are the experimental results:

1. There are 32 best difference propagations with probability $1300 \times 2^{-76} \approx 2^{-65.66}$ each. Each is composed of 7 differential trails. Among the 7 trails, one with probability 2^{-66}, two with probability 2^{-69} each, one with probability 2^{-72}, one with probability 2^{-75}, and two with probability 2^{-76} each.
2. Among all the difference propagations, the maximum number of trails of a difference propagation is 131, i.e., a difference propagation is composed of at most 131 different differential trails. For such a difference propagation, the probability is $421 \times 2^{-76} \approx 2^{-67.28}$.

From result 1, the probability of the best difference propagation is lower than 2^{-63}. From the two results, it can be seen that the clustering of differential trails of RECTANGLE is very limited, which can not be used to construct an effective difference propagation with more than 14 rounds.

For comparison, we give some statistical data concerning the serious clustering of differential trails of 16-round PRESENT from [50]. For 16-round PRESENT, the probability of the best differential trail is 2^{-70}. There exists a 16-round difference propagation satisfying the following properties:

1. It includes 31996 differential trails when the probability is restricted between 2^{-70} and 2^{-80}. The probability is $2^{-62.175}$ when only considering these 31996 trails;
2. It includes 83720 differential trails when the probability is restricted between 2^{-70} and 2^{-92}. The probability is $2^{-62.133}$ when considering all the 83720 trails.

Therefore, we believe that it is impossible to construct an effective 15-round (multiple) differential distinguisher for RECTANGLE. Full dependency is reached already after 4 rounds, hence we believe 25-round RECTANGLE is enough to resist against (multiple) differential cryptanalysis.

Using one 14-round difference propagation, we can mount an attack on 18-round RECTANGLE, which is the highest number of rounds that we can attack. For reference, Appendix E presents the distinguisher and the attack complexity.

3.2 Linear Cryptanalysis

Assume a linear trail hold with probability p , define the bias ϵ as $(p - \frac{1}{2})$, the correlation contribution C as 2ϵ. To attack an n-bit block cipher using linear cryptanalysis (LC), there must be a predictable linear propagation over all but a few rounds with an amplitude significantly larger than $2^{-\frac{n}{2}}$. A linear propagation is composed of a set of linear trails, where its amplitude is the sum of the correlation contributions of all linear trails that have the specified input and output selection patterns [20]. The correlation contributions of the linear trails are signed and their sign depends on the value of the round keys. For RECTANGLE, to be resistant against LC, it is a necessary condition that there is no linear propagation with an amplitude higher than 2^{-32}. Since the strong round key dependence of interference makes locating the input and output selection patterns for which high correlations occur practically infeasible [20], we have to use the following theorem for an estimation.

Table 2. Correlation potentials of the best linear trails of RECTANGLE

♯ R	Cor. Pot.	♯ R	Cor. Pot.	♯ R	Cor. Pot.
1	2^{-2}	6	2^{-20}	11	2^{-50}
2	2^{-4}	7	2^{-26}	12	2^{-56}
3	2^{-8}	8	2^{-32}	13	2^{-62}
4	2^{-12}	9	2^{-38}	14	2^{-68}
5	2^{-16}	10	2^{-44}	15	2^{-74}

Theorem 1 ([20]). *The square of a correlation (or correlation contribution) is called correlation potential. The average correlation potential between an input and an output selection pattern is the sum of the correlation potentials of all linear trails between the input and output selection patterns:*

$$E(C_t^2) = \sum_i (C_i)^2$$

where C_t is the overall correlation, and C_i the correlation coefficient of a linear trail.

We have modified the search program used in the differential case to search for the best linear trails of RECTANGLE from 1 round to 15 rounds, and the results are presented in Table 2. Similarly, we also need to consider the security of RECTANGLE against multiple linear cryptanalysis [11] and multidimensional linear cryptanalysis [27]. For 15-round RECTANGLE, the correlation potential of the best linear trail is 2^{-74}. Also based on the branch-and-bound algorithm, we have searched for all the linear trails with a correlation potential between 2^{-74} and 2^{-80} (up to a rotation equivalence) for 15-round RECTANGLE and examined all the linear propagations made up of the investigated trails. The following are the experimental results:

1. There are 128 best linear propagations with an average correlation potential $1860 \times 2^{-80} \approx 2^{-69.14}$ each, which is lower than 2^{-64}. Each is composed of 891 linear trails. Among the 891 trails, 2 with correlation potential 2^{-74} each, 26 with correlation potential 2^{-76} each, 151 with correlation potential 2^{-78} each, and 712 with correlation potential 2^{-80} each.
2. Among all the linear propagations, the maximum number of trails of a linear propagation is 891. Actually, the best linear propagations have the maximum number of trails.

For comparison with PRESENT, there are the two facts:

1. There exists a 16-round linear propagation of PRESENT, which is composed of $435,600$ linear trails with a correlation potential 2^{-64} each [42]. Thus, the average correlation potential is $435600 \times 2^{-64} \approx 2^{-45.26}$.
2. There exists a 23-round linear propagation of PRESENT, which is composed of $367,261,713$ linear trails with a correlation potential 2^{-92} each [42]. Thus, the average correlation potential is about $2^{-63.54}$.

From the above results and a comparison with PRESENT, it can be seen that the clustering of linear trails of RECTANGLE is limited, which can not be used to construct an effective linear propagation with more than 14 rounds. Therefore, we believe that it is impossible to construct an effective 15-round (multiple, multidimensional) linear distinguisher for RECTANGLE. Full dependency is reached already after 4 rounds, hence we believe 25-round RECTANGLE is enough to resist against linear cryptanalysis and its extension attacks.

3.3 Statistical Saturation Attack

The statistical saturation (SS) attack [17] is specially designed for PRESENT. Using the weak diffusion of the PRESENT permutation. More specifically, for 4 selected S-box positions, 8 out of 16 input bits are directed to the same 4 S-box positions after the permutation. Using this property, there exists a theoretical attack against 24-round PRESENT.

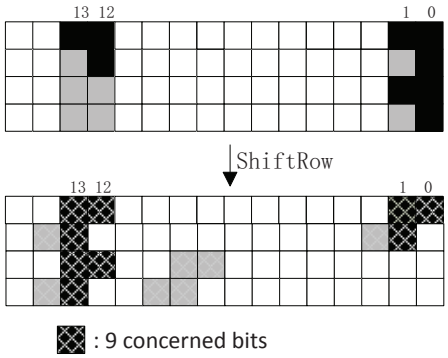

Fig. 6. A Weak Property of ShiftRow

Algorithm 1

Set the subkey in each round to a random value.
for $r = 1$ to 10 do

{ 1. Choose a set of 2^{40} plaintexts which have a constant value in the 4 columns with an index set $\{0, 1, 12, 13\}$, while having random values in the other $64 - 16 = 48$ bits.
2. Calculate the distribution of the outputs in the concerned 9 bit positions after r-round encryption, and compute the squared Euclidian distance between this distribution and uniform distribution. Let O denote the output after r-round encryption, j denote the value of the 9-bit string $O_{3,13}||O_{2,13}||O_{2,12}||O_{1,13}||O_{1,1}||O_{0,13}||O_{0,12}||O_{0,1}||O_{0,0}$, the distance is defined as:

$$Dis = \sum_{j=0}^{2^9-1} (\frac{counter[j]}{2^m} - \frac{1}{2^9})^2$$

where $counter_j$ denotes the times of occurence of j among all the 2^m values.
}

Due to the weak diffusion of the permutation layer of RECTANGLE, we must consider the security of RECTANGLE against statistical saturation attack. Consider the 4 columns with an index set $\{0, 1, 12, 13\}$, then 9 out of 16 bits are still directed to the same 4 column positions after ShiftRow. Fig. 6 illustrates this property, and Algorithm 1 presents the procedure of our experiment. By choosing 8 different 16-bit constant values (with the same seed key) and by choosing 8 different keys (with the same set of 2^{40} plaintexts), we have executed Algorithm 1 for 16 times. Based on the results in Table 11 (see Appendix C), we expect that the distance can be estimated by multiplying about 2^{-4} when adding a round to the distinguisher. According to the estimate of data complexity in [13], we estimate that the longest SS distinguisher of RECTANGLE can reach 15 rounds at most, and the distinguisher can be used to attack 18-round RECTANGLE at most. Considering the full rounds is 25, we believe there is enough security margin for RECTANGLE against the SS attack.

3.4 Impossible Differential Cryptanalysis

Impossible differential cryptanalysis [8] exploits differential trails with probability 0. Impossible differential distinguishers are usually constructed by meet-in-the-middle approach.

We found some 8-round impossible differential distinguishers for RECTANGLE. Here is one. Firstly, a 4-round differential trail with probability 1 along the encryption direction, then a 4-round differential trail with probability 1 along the decryption direction. More exactly, let (i, j) denote the bit position in the i-th row and j-th column of a cipher state, as Fig. 2 shows. Given a pair of round inputs in round 0 which are equal in all bits except two bit positions $(2, 0)$ and $(3, 0)$, then the round outputs in round 3 must be equal in the position $(2, 12)$. In the backward direction, given a pair of round outputs in round 7 which are equal in all bits except one bit position $(2, 0)$, then the round inputs in round 4 can not be equal in the bit position $(2, 12)$. It is obvious that the output in round 3 equals to the input in round 4, thus a contradiction.

Notice that the following properties of the RECTANGLE S-box are used to construct the above distinguisher. Let $x = x_3||x_2||x_1||x_0$, where x_i is the i-th bit of x, $i = 0, 1, 2, 3$. Let $(\Delta x \rightarrow \Delta y)$ denote a differential with input difference Δx and output difference Δy. For the RECTANGLE S-box, $(1100 \rightarrow **0)$ holds with probability 1. For the inverse S-box, $(0100 \rightarrow *1**)$ holds with probability 1, where "$*$" denotes

an unknown bit. Since 4-round RECTANGLE reaches the full dependency, it is expected that full-round RECTANGLE has enough security against impossible differential cryptanalysis.

3.5 Integral Cryptanalysis

Integral cryptanalysis (or square attack) [18, 32] considers the propagation of sums of many values. An integral distinguisher holds with probability 1.

We found some 7-round higher-order integral distinguishers. Here is one. First is a 4-round integral distinguisher. Choose a set of two plaintexts P and P^*, set $P_{2,0} = P_{3,0} = 0$ and $P^*_{2,0} = P^*_{3,0} = 1$, while they have a fixed random value in the other 62 bit positions. Note that P and P^* only have a non-zero difference in column 0, with $P_{Col(0)} \oplus P^*_{Col(0)} = 1100$. The difference $(1100 \rightarrow ***0)$ holds with probability 1 for the S-box. Then, after 4-round encryption, the round outputs in round 3 have a zero difference in the following 4 bit positions: $(0,0), (1,1), (2,12), (3,13)$. In other words, the XOR sum in any of the 4 bit positions equals to 0. Next, consider the decryption direction. Choose a set of 2^{48} plaintexts which have certain fixed values in column 0, 13, 14 and 15, while having all the 2^{48} possible values in the other 12 columns. After 3-round encryption, the 2^{48} intermediate values can be divided into 2^{47} subsets, the two values in each subset satisfy the data condition of the above 4-round distinguisher. Hence, after 7-round encryption, the XOR sum of all the 2^{48} outputs in any of the 4 bit positions equals to 0, which is a 7-round integral distinguisher. Similarly, it is expected that full-round RECTANGLE has enough security against integral cryptanalysis.

3.6 Key Schedule Attacks

Among key schedule attacks, the most effective ones are slide attack [10] and related-key cryptanalysis [7]. For RECTANGLE, the adding of different round constants in the key schedule prevents slide attacks. For 80-bit seed keys, the union of subkey bits of any consecutive 2 rounds depends on each of the 80 bits of the seed key. For 128-bit seed keys, the union of subkey bits of any consecutive 4 rounds depends on each of the 128 bits of the seed key. The generalized Feistel transformations are designed to provide appropriate diffusion. We believe that the above properties are sufficient for RECTANGLE to resist against key schedule attacks.

4 Motivation for Design Choices of RECTANGLE

In this section, we justify the choices we took during the design of RECTANGLE.

4.1 Bit-Slice Technique and Lightweight Block Cipher

Consider a 64-bit SP-network block cipher, the S-layer consists of 16 4×4 S-boxes in parallel, thus the subblock length is 16 for a bit-slice implementation. Let a 64-bit state be arranged as a 4×16 array. First, apply the same S-box to each column independently. Then, the P-layer should make each column dependent on some other columns, aiming to provide good diffusion. In such a situation, 16-bit rotations are probably the best choice: they are simple wirings in hardware implementation; they can achieve the goal of mixing up different columns; they can be easily implemented in software using bit-slice technique. So far, we got the framework of RECTANGLE.

4.2 The ShiftRow Transformation

Let c_i $(i = 0, 1, 2, 3)$ denote the left rotation offset of the i-th row. The choice criteria of c_i are as follows:

1. The four offsets are different;
2. $c_0 < c_1 < c_2 < c_3$, and $c_0 = 0$;
3. Full dependency after a minimal number of rounds.

Our experimental result shows that there are 16 candidates satisfying the above criteria. For each of the 16 candidates, after 4 rounds each of the 64 input bits influences each of the 64 output bits. From them, we choose $(c_1, c_2, c_3) = (1, 12, 13)$ as the rotation offsets of the ShiftRow transformation.

4.3 Design Criteria of the S-box

A 4×4 S-box is typically much more compact in hardware than an 8×8 S-box. Serpent uses 8 different S-boxes, however. The use of different S-boxes for different rounds does not result in a plausible improvement of the resistance against known attacks (a variant view from [20]). Moreover, the hardware area can be reduced using only one S-box. Hence, we decide to use only one 4×4 S-box for RECTANGLE.

Let S denote a 4×4 S-box. Let $\triangle I, \triangle O \in F_2^4$, define $ND_S(\triangle I, \triangle O)$ as:

$$ND_S(\triangle I, \triangle O) = \sharp\{x \in F_2^4 | S(x) \oplus S(x \oplus \triangle I) = \triangle O\}.$$

Let $\Gamma I, \Gamma O \in F_2^4$, define the imbalance $Imb_S(\Gamma I, \Gamma O)$ as:

$$Imb_S(\Gamma I, \Gamma O) = |\sharp\{x \in F_2^4 | \Gamma I \bullet x = \Gamma O \bullet S(x)\} - 8|.$$

where $\bullet$ denotes the inner product on F_2^4.

The design criteria of the S-box of RECTANGLE are as follows:

1. Bijective, i.e., $S(x) \neq S(x')$ for any $x \neq x'$.
2. For any non-zero input difference $\triangle I \in F_2^4$ and any non-zero output difference $\triangle O \in F_2^4$, we require:
$$ND_S(\triangle I, \triangle O) \leq 4.$$
3. Let $\triangle I \in F_2^4$ be a non-zero input difference and $\triangle O \in F_2^4$ a non-zero output difference. Let $wt(x)$ denote the Hamming weight of x. Define $SetD1_S$ as:
$$SetD1_S = \{(\triangle I, \triangle O) \in F_2^4 \times F_2^4 | wt(\triangle I) = wt(\triangle O) = 1 \ and \ ND_S(\triangle I, \triangle O) \neq 0\}.$$
Let $CarD1_S$ denote the cardinality of $SetD1_S$, we require $CarD1_S = 2$.
4. For any non-zero input selection pattern $\Gamma I \in F_2^4$ and any non-zero output selection pattern $\Gamma O \in F_2^4$, we require:
$$Imb_S(\Gamma I, \Gamma O) \leq 4.$$
5. Let $\Gamma I \in F_2^4$ be a non-zero input selection pattern and $\Gamma O \in F_2^4$ a non-zero output selection pattern, define $SetL1_S$ as:
$$SetL1_S = \{(\Gamma I, \Gamma O) \in F_2^4 \times F_2^4 | wt(\Gamma I) = wt(\Gamma O) = 1 \ and \ Imb_S(\Gamma I, \Gamma O) \neq 0\}.$$
Let $CarL1_S$ denote the cardinality of $SetL1_S$, we require $CarL1_S = 2$.
6. No fixed point, i.e., $S(x) \neq x$ for any $x \in F_2^4$.

4.4 Selection of the S-box of RECTANGLE

In the following, an S-box means a 4×4 S-box.

Definition 1 ([34]). *Two S-boxes S and S' are called **affine equivalent** if there exist bijective linear mappings A, B and constants $a, b \in F_2^4$ such that $S'(x) = B(S(A(x) + a)) + b$. The equivalence is called **affine equivalence**.*

If an S-box satisfies criteria 1, 2 and 4 (see Section 4.3), then any of its affine equivalent S-boxes also satisfies criteria 1, 2 and 4.

Definition 2 ([34]). *Two S-boxes S and S' are called **permutation-then-XOR equivalent** if there exist 4×4 permutation matrices P_0, P_1 and constants $a, b \in F_2^4$ such that $S'(x) = P_1(S(P_0(x) + a)) + b$. The equivalence is called **PE equivalence** for short.*

If an S-box satisfies criteria 1-5 (see Section 4.3), then any of its PE equivalent S-boxes also satisfies criteria 1-5.

It is a surprising fact that all 4×4 S-boxes satisfying criteria 1, 2 and 4 can be classified into only 16 affine equivalence classes [34]. By using the 16 representatives of the 16 affine equivalence classes presented in [34] (Using another set of the 16 representatives in [21], the same result is derived), we have designed an efficient algorithm to classify all 4×4 S-boxes fulfilling criteria 1-5 into PE classes, it is also a surprising fact that there are only 4 different PE classes. We list a representative for each PE class in Table 3. In each row of Table 3, the first integer represents the image of 0, the second the image of 1, and so on.

Table 3. Representatives for all the 4 PE classes fulfilling criteria 1-5

PE_0	6,0,8,15,12,3,7,13,11,14,1,4,5,9,10,2
PE_1	3,2,8,13,15,5,6,10,9,14,4,7,0,12,11,1
PE_2	6,8,15,4,12,7,9,3,11,1,0,14,5,10,2,13
PE_3	8,1,6,12,5,15,10,3,7,11,13,2,0,14,9,4

Table 4. 5 groups of the 32 S-boxes

Group Number	1	2	3	4	5
$Prob\_D15$	66	66	66	66	67
$(Cor\_L15)^2$	74	72	70	66	66
Number of S-boxes	4	8	4	8	8

Up to adding constants before and after an S-box, which does not change any of the criteria 1-5 and furthermore does not change the probability of the best differential/linear trail for a specific number of rounds, there are $4 \times 4! \times 4! = 2304$ S-boxes that can be generated from the 4 representatives in Table 3. The 2304 S-boxes are denoted as $\{S_i\}, i = 0, 1, \cdots 2303$.

Algorithm 2 is designed to discard a part of the S-box candidates which can result in a differential (or linear) trail with a single active S-box in each round. Consider 2-round RECTANGLE, it can be easily seen that the i-th ($i = 0, 1, 2, 3$) bit of an S-box output in the first round will be the i-th bit of an S-box input in the second round. Hence, if the condition in Step 2 holds, it can be easily verified that there exists a differential trail with a single active S-box in each round, for any number of rounds. Similarly, if the condition in Step 4 holds, then there exists a linear trail with a single active S-box in each round, for any number of rounds.

One may say that the permutation layer lets the positions inside the S-box invariant may introduce some weaknesses. However, we believe that it will not be a problem for the security of the cipher, since the S-box has the property that each of the 4 output bits depends on all of the 4 input bits, and for 4-round RECTANGLE each of the 64 output bits depends on all of the 64 input bits.

We implemented Algorithm 2. The result shows that 1776 S-boxes are discarded and only 528 S-boxes are remained. Next, we create a further filtering by considering the security of the underlying cipher against differential and linear cryptanalysis.

Let $Prob\_D15$ denote the probability of the best 15-round differential trail, and $(Cor\_L15)^2$ the correlation potential of the best 15-round linear trail. Fix the ShiftRow transformation, for each choice of the remained 528 S-boxes, check whether the two inequalities hold for the underlying cipher : $Prob\_D15 < 2^{-64}$ and $(Cor\_L15)^2 < 2^{-64}$. Our experimental result shows that only 32 S-boxes satisfy the two inequalities. According to the values of $Prob\_D15$ and $(Cor\_L15)^2$, the 32 S-boxes can be divided into 5 groups, the result is illustrated in Table 4.

Algorithm 2

INPUT: 2304 S-boxes $\{S_i\}, i = 0, 1, \cdots 2303$.
OUTPUT: Discard a part of the S-boxes which can result in a differential (or linear) trail with a single active S-box in each round.

for i = 0 to 2303 do

{ 1. For the i-th S-box S_i, calculate the two $(\triangle I, \triangle O)$ pairs which belong to the set $SetD1_S$. Let $(\triangle I_j, \triangle O_j)$ denote the two pairs, $j = 1, 2$.
2. If $(\triangle I_1 = \triangle O_1)$ or $(\triangle I_2 = \triangle O_2)$ or $(\triangle I_2 = \triangle O_1 \ and \ \triangle I_1 = \triangle O_2)$, then discard the S-box and $i \leftarrow i + 1$; else go to the following Step 3.
3. For the i-th S-box S_i, calculate the two $(\Gamma I, \Gamma O)$ pairs which belong to the set $SetL1_S$. Let $(\Gamma I_j, \Gamma O_j)$ denote the two pairs, $j = 1, 2$.
4. If $(\Gamma I_1 = \Gamma O_1)$ or $(\Gamma I_2 = \Gamma O_2)$ or $(\Gamma I_2 = \Gamma O_1 \ and \ \Gamma I_1 = \Gamma O_2)$, then discard the S-box and $i \leftarrow i + 1$.
}

For the 5 groups, by checking the probability of the best differential trail and the correlation coefficient of the best linear trail up to 15 rounds, we finally choose group 1. There are 4 S-boxes in group 1, which belong to 2 different PE classes. Thus, we only need to consider 2 out of the 4 S-boxes. By adding constants before and after the S-box, we can get $2 \times 16 \times 16 = 512$ different S-boxes. Among the 512 S-boxes, we choose one with no fixed point and low area requirement as the S-box for RECTANGLE. One can refer to [53] for more details on selection of the RECTANGLE S-box.

4.5 The Key Schedule

The design criteria of 80-bit (resp. 128-bit) key schedule are as follows:

1. The union of subkey bits of any 2 (resp. 4) consecutive rounds depends on each of the 80 bits of the seed key;
2. The 1-round 5-subblock (resp. 4-subblock) generalized Feistel transformation is used to provide appropriate diffusion;
3. Use round constants to eliminate symmetries.

4.6 The Number of Rounds

Our analysis showed that the highest number of attacked rounds is 18. We decide to add 7 rounds as a security margin, and take 25 as the round number of RECTANGLE.

5 Performance in Various Environments

5.1 Hardware Implementation

We implemented RECTANGLE in Verilog HDL and used Mentor Graphics Modelsim SE PLUS 6.6d for functional simulation. All proposed hardware designs in this paper were synthesized with Synopsys Design Compiler D-2010.03-SP4 to the UMC's $0.13\mu m$.1P8M Low Leakage Standard cell Library with the following typical values: voltage of $1.2V$ and temperature of $25°C$. We used a round-based architecture which is a direct mapping of the algorithm, frequently used for implementation evaluation. This architecture requires no overhead on multiplexers and flip-flops and typically has the lowest energy/bit, providing a good tradeoff among area, time and throughput. Moreover, it is easily reduced to a serial architecture or unfolded into a parallel architecture to meet specific demands for different application scenarios. We only present the detail of a round-based architecture. The source code of our hardware implementations can be found in [44].

Round-based Architecture Round-based RECTANGLE-80 uses 64/80-bit datapaths for state and key respectively. It performs one round in one clock cycle. The state datapath consists of the 64-bit register (for storing), the S-layer, the P-layer and the 64-bit XOR of key addition. Besides the 80-bit register for key storing, the S-boxes, P-layer and XORs are utilized to update the subkey. A Finite State Machine is used to generate control logic. The plaintext and the key are loaded into each register via multiplexers. Then on each of the following 25 clock cycles, data is read out from the registers, passed through the state and key datapaths and stored back to register respectively. Finally, we can obtain the ciphertext at the output of the 64-bit XOR. Fig. 7 illustrates the design diagram of RECTANGLE-80. For the 128-bit version, the state datapath is the same as the 80-bit version, and the key datapath has four more S-boxes and a different generalized Feistel transformation.

As indicated in Table 5, the area consumption of a round-based RECTANGLE-80 is 1600 GE (Gate Equivalent: The size of one NAND gate under specified technology). The most area consuming parts are the flip-flops for the state and key storing, the 20 S-boxes and two 64-bit XOR arrays. Based on this specified CELL Library, our S-box consumes around 18.8 GE. The P-layer of round function is only wiring. The round-based RECTANGLE-80 has a simulated power consumption of $74.31\mu w$ at $10MHz$. For the round-based RECTANGLE-128, the area consumption is 2064 GE and the simulated power consumption is $72.15\mu w$ at $10MHz$.

Table 5. Implementation results of round-based RECTANGLE-80&128

	Key:80-bit		Key:128-bit	
module	Area(GE)	%	Area(GE)	%
data state	400	25.01	400	19.38
s-layer	307	19.19	300	14.54
p-layer	0	0	0	0
key XOR	176	11.00	176	8.53
FSM	3	0.19	3	0.15
KS:key state	500	31.26	800	38.77
KS:S-box	75	4.69	150	7.27
KS:p-layer	96	6.00	192	9.30
KS:counter-XOR	42.5	2.66	42.5	2.06
sum	1600	100	2064	100

Table 6. Comparison of lightweight cipher implementations (Area vs. Throughput)

	Key size	Block size	Cycles per Block	Tech. μm	Area (GE)	Tput.At 100KHz(Kbps)
Block Ciphers						
AES-128[40]	128	128	226	0.13	2400	56.6
LED-64[30]	64	64	1248	0.18	966	5.1
PICCOLO-80[48]	80	64	27	0.13	1496	237
PRESENT-80[45]	80	64	32	0.18	1570	200
RECTANGLE-80	80	64	26	0.13	1599.5	246
RECTANGLE-128	128	64	26	0.13	2063.5	246
Stream Ciphers						
Grain[29]	80	1	1	0.13	1294	100
Trivium[29]	80	1	1	0.13	2599	100

Results and Comparisons A comparison of round-based implementations of RECTANGLE and other ciphers is presented in Table 6. The throughput is calculated in bits per second. The result in Table 6 illustrates that RECTANGLE has a rather high throughput with a compact area consumption.

Table 7 gives a comparison of the 3 architectures of RECTANGLE-80 and other ciphers. The power consumption is estimated on the gate level by PowerCompiler, based on the switching activates generated by a real testbench. The power strongly depends on the clock frequency and technology. To draw a fair comparison, energy per bit is used to represent the energy efficiency. The results show that RECTANGLE meets the needs under different scenarios and has a rather low energy consumption. The round-based architecture has a good tradeoff between the area and the throughput. The parallel implementation achieves a high throughput rate but consumes the most area and power. The serial design has more ideal compact structure. However, the cost of this area saving is the increasing processing time of 457 cycles.

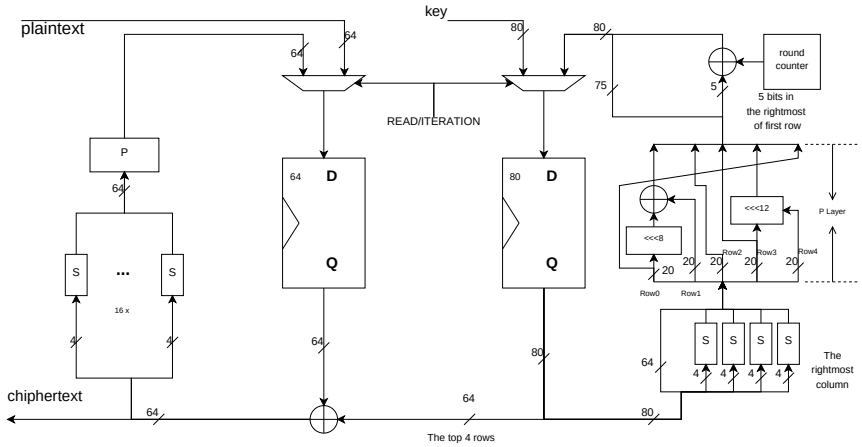

Fig. 7. The datapath of the round-based RECTANGLE-80

Table 7. Comparison of 3 different architectures of implementations

	Tech. (μm)	Datapath (Bit)	Freq. (MHz)	Area (GE)	Tput	Energy/Bit (pJ/bit)
Round-based						
HWang AES[31]	0.18	128	50	79K	582$Mbps$	93
PICCOLO80[48]	0.13	64	0.1	1496	237$Kbps$	–
PRESENT80[45]	0.18	64	10	1570	20.6$Mbps$	3.74
RECTANGLE-80	0.13	64	10	1600	24.6$Mbps$	3.0
Parallel						
PRESENT80[45]	0.18	64	200	27027	10.22$Gbps$	0.67
RECTANGLE-80	0.13	64	200	24512	12.8$Gbps$	0.32
Serial						
AES-128[40]	0.13	8	0.1	2400	56.6$Kbps$	–
LED-64[30]	0.18	4	0.1	966	5.1$Kbps$	–
PICCOLO80[3]	0.13	64	0.1	1043	14.8$Kbps$	–
PRESENT80[45]	0.18	4	0.1	1075	11.4$Kbps$	221.1
RECTANGLE-80	0.13	4	0.1	1111	14.0$Kbps$	32.05

"-" means the value is unavailable at the time of writing.

Table 8. Comparison of Software Performance of LED, Piccolo, PRESENT and RECTANGLE

	LED	Piccolo	PRESENT	RECTANGLE
block length	64	64	64	64
key length	64	80	80	80
one block enc.	65	67.1 [4]	62	30.5
SSE enc.		4.57 [36]	4.73 [36]	3.9
(cycles/byte)	-	16 para. blocks	32 para. blocks	8 para. blocks

"one block enc." is for a single block encryption.
"SSE enc." is for multiple parallel encryptions using 128-bit SSE instructions.
"para." means parallel. "-" means the value is unavailable at the time of writing.

5.2 Software Implementation

We firstly present our implementation results of RECTANGLE on 64-bit processors. Here we consider two different cases: one block encryption and 8-block parallel encryption. Secondly, we present our implementation results of RECTANGLE on 8-bit micro-controllers.

On 64-bit Processors We implemented RECTANGLE on a 2.5GHz Intel(R) Core i5-2520M CPU running a 64-bit operating system with an Intel C++ compiler.

For one block data, our bit-slice implementation gives a speed of about 30.5 cycles/byte for encryption and 32.2 cycles/byte for decryption. The S-box S can be implemented using a sequence of 12 logical instructions (see Appendix D), the P-layer only needs 3 rotations, and the subkey addition needs 4 XORs. The above 3 functions can be compiled under a low register pressure by the Intel C++ compiler. The inverse S-box can be also implemented using 12 logical instructions. In addition, the memory footprint of the encryption/decryption routines is also very low. Our implementation is quite straightforward, it only uses 6 basic instructions: AND, OR, NOT, XOR, ROL and MOV. It is worth noting that each of the 4 rows of a cipher state only uses 16 out of the 64 bits of the general registers, which means that the encryption/decryption speed can be improved greatly by using all the 64 bits of the registers (e.g., 4 parallel encryptions). The key schedule cost is about 293 cycles for an 80-bit seed key and 259 cycles for a 128-bit seed key.

In the case of a parallel mode of operation such as CTR, using Intel 128-bit SSE instructions can give RECTANGLE a very impressive performance. First, consider 8 64-bit blocks, put the first 16 bits of each of the 8 blocks to the first 128-bit vector register, the second 16 bits of each of the 8 blocks to the second 128-bit vector register, and so on. Next, consider messages with x blocks ($1 \leq x \leq 512$). If x is not a multiple of 8, then the corresponding part of the 128-bit registers is set to all zero. Since RECTANGLE is designed as a bit-sliced cipher, the cost of data load and data format conversion is very low, which takes less than 0.2 cycles/byte when $x \geq 7$. Our bit-slice implementation of RECTANGLE reaches an average speed of about 3.9 cycles/byte for messages with a length of around 3000 bytes.

Table 8 gives software performance comparisons of RECTANGLE with LED, Piccolo and PRESENT. For one block encryption, we implemented LED, Piccolo and PRESENT on our platform using the codes

Table 9. Performance of RECTANGLE on Atmel ATtiny45 Processor

Method	Key size	Code size [bytes]		RAM [bytes]	Cycles		
	[bits]	enc.+k.s.	dec.+k.s.	enc./dec.	enc.	dec.	k.s.
Static	80	636	638	226	1920	1945	1878
	128	614	616	232	1920	1945	1462
Method	Key size	Code size [bytes]		RAM [bytes]	Cycles		
	[bits]	enc.+e.k.	dec.+e.k.	enc./dec.	enc.	dec.	k.s.
Fixed	80/128	574	576	8	2129	2154	-
Method	Key size	Code size [bytes]		RAM [bytes]	Cycles		Unrolled
	[bits]	enc.+k.s.	dec.+i.k.s.	enc./dec.	enc.+ k.s.	dec.+i.k.s.	
On-the-fly	80	500	504	18	2801	2851	1-round
	128	488	492	24	2438	2488	
	80	1284	1304	18	2617	2667	5-round
	128	1092	1112	24	2112	2162	4-round

"enc.", "dec.","k.s. ", "i.k.s. " and "e.k. "means encryption, decryption, key schedule, inverse key schedule and expanded key respectively.

found in [4]. For LED and PRESENT, our test timings are consistent with those in [4]. For Piccolo, we obtained a slower timing result compared to [4], hence, we cite the result in [4] for Piccolo. From Table 8, we can see that the software performances of RECTANGLE on 64-bit processors are quite impressive.

On 8-bit Micro-controllers In the following, we present performance of RECTANGLE on 8-bit micro-controllers, particularly on Atmel ATtiny45, which uses an 8-bit RISC processor with 32 single-byte general purpose registers, 256 bytes of SRAM and 4K bytes of programmable flash memory. To achieve optimal performance, all the implementations are assembly coded, and the codes are compiled using Atmel studio 6.2.

Table 9 shows the performance metrics of RECTANGLE. If not specified, the codes for encryption, decryption and key schedule are one-round unrolled. The encryption cost is the number of cycles for transforming a plaintext into a ciphertext, including any data & key loading and data write-back. In our implementations, we consider three cases. In the first case, the seed key is expanded and the round keys are loaded into SRAM. We denote this case by "static". The second case, the key schedule is not implemented, and the round keys are stored in flash. We include the time to load the key from flash into registers. We denote this case by "fixed". The third case, the round keys are generated "on-the-fly". The computation of the key schedule(resp. the inverse key schedule) is included in the encryption (resp. decryption) costs. We denote this case by "on-the-fly". In this third case, the functions can be implemented in two ways, one-round unrolled and 5-round unrolled for RECTANGLE-80 (resp. 4-round unrolled for RECTANGLE-128). During the execution, all the running states are held in registers, thus there is no need for extra SRAM and additional data loading. The flash requirement includes the memory used to store the code, the lookup tables, one input block and the master key (in the case of "fixed", also includes the round keys).

In our implementations, the S-box and the P-layer are respectively implemented using a sequence of logical instructions. With four additional registers, the RECTANGLE S-box needs 26 instructions, and the inverse S-box needs 27 instructions. Both the P-layer and the inverse P-layer need 20 instructions. Note that each of the above mentioned instructions needs one single cycle. For the key schedule, the round constant additions are implemented as lookup tables.

In [24], the authors provide implementations of 12 block ciphers on an ATMEL ATtiny45 8-bit micro-controller. Compare our implementations for RECTANGLE in Table 9 with the results reported in [24], it can be concluded that RECTANGLE has an outstanding performance on 8-bit microcontrollers.

6 Relation to Early Designs

The main idea of the design of RECTANGLE is to allow lightweight and fast implementations using bit-slice techniques. Serpent and Noekeon are two early bit-sliced block ciphers. However, the design goal of the two ciphers is general-purpose instead of lightweight, almost all aspects need to be reconsidered when it comes to

a dedicated lightweight block cipher, including the block length, the key length, the selection of the S-box, the design of the P-layer and the design of the key schedule.

Many block ciphers use parallel 4×4 S-boxes to provide confusion such as Serpent, Noekeon, PRESENT, LED, KLEIN, LBlock and TWINE. In this paper, we proposed new design criteria for the RECTANGLE S-box, i.e. $CarD1_S = CarL1_S = 2$. The new criteria are mainly motivated by the existing security analysis of PRESENT, specifically (multiple) differential/linear cryptanalysis on reduced-round PRESENT [16, 42, 50]. Moreover, one can get more confidence in the security of RECTANGLE by comparing the security of PRESENT and RECTANGLE against (multiple) differential/linear cryptanalysis, which were shown in subsections 3.1 and 3.2.

The design of the P-layer of RECTANGLE depends largely on the bit-slice technique, which is determined by 3 rotation offsets. Compared with the P-layers of Serpent and Noekeon, the P-layer of RECTANGLE is much more friendly in hardware. Compared with the P-layer of PRESENT, the P-layer of RECTANGLE is much more friendly in software.

7 Conclusion

We have proposed RECTANGLE, a new lightweight block cipher based on the bit-slice technique. RECTANGLE is a simple design. The bit-sliced design principle allows for both low-cost hardware and efficient software implementations. Largely due to our careful selection of the S-box, RECTANGLE achieves a very good security-performance tradeoff. We want to point out that the selection of the P-layer is also important. The RECTANGLE P-layer is composed of 3 rotations, which is not only extremely low-cost in hardware but also very efficient in software. In addition, the combination of the S-box and the P-layer brings the cipher a very limited clustering of differential/linear trails. We believe that RECTANGLE is an interesting design and we feel that it can trigger several new problems in cryptographic design and analysis. In the end, we encourage further security analysis of RECTANGLE.

Acknowledgements

We are very grateful to Begül Bilgin, Joan Daemen, Junfeng Fan, Benedikt Gierlichs, Zheng Gong and Nicky Mouha for their helpful comments. The research presented in this paper is supported by the National Natural Science Foundation of China (No.61379138), the Research Fund KU Leuven (OT/13/071), the "Strategic Priority Research Program" of the Chinese Academy of Sciences (No.XDA06010701), and the National High Technology Research and Development 863 Program of China (No.2013AA014002).

References

1. Atmel AVR 8-bit Instruction Set, http://www.atmel.com/images/doc0856.pdf
2. Anderson, R., Biham, E., Knudsen, L.R.: Serpent: A Proposal for the Advanced Encryption Standard. NIST AES proposal (1998)
3. Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., and Wingers, L. The SIMON and SPECK Families of Lightweight Block Ciphers. Cryptology ePrint Archive, Report 2013/404, 2013. http://eprint.iacr.org/
4. Benadjila1, R., Guo, J., Lomn, V., Peyrin, T.: Implementing Lightweight Block Ciphers on x86 Architectures, Cryptology ePrint Archive: Report 2013/445, http://eprint.iacr.org/2013/445
5. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Keccak Specifications. NIST SHA-3 Submission (2008), http://keccak.noekeon.org/
6. Biham, E.: A Fast New DES Implementation in Software. In: Biham, E. (ed.) FSE 1997. LNCS, vol. 1267, pp. 260–272. Springer (1997)
7. Biham, E.: New Types of Cryptanalytic Attacks Using Related Keys. J. Cryptology 7(4), 229–246 (1994)
8. Biham, E., Biryukov, A., Shamir, A.: Cryptanalysis of Skipjack Reduced to 31 Rounds Using Impossible Differentials. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 12–23. Springer (1999)
9. Biham, E., Shamir, A.: Differential Cryptanalysis of DES-like Cryptosystems. Journal of Cryptology 4(1), 3–72 (1991)
10. Biryukov, A., Wagner, D.: Slide Attacks. In: Knudsen, L.R. (ed.) FSE 1999. LNCS, vol. 1636, pp. 245–259. Springer (1999)

11. Biryukov, A., De Cannière, C., Quisquater, M.: On Multiple Linear Approximations. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 1–22. Springer (2004)
12. Blondeau, C., Gerard, B.: Multiple Differential Cryptanalysis: Theory and Practice. In: Joux, A. (ed.) FSE 2011. LNCS, vol. 6733, pp. 35–54. Springer (2011)
13. Blondeau, C., Nyberg, K.: Links Between Truncated Differential and Multidimensional Linear Properties of Block Ciphers and Underlying Attack Complexities. In Nguyen, P.Q., Oswald., E. (eds.): EUROCRYPT 2014. LNCS, vol. 8441, pp.165–182. Springer (2014)
14. Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y., Vikkelsoe, C.: PRESENT: An Ultra-Lightweight Block Cipher. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 450–466. Springer (2007)
15. Bogdanov, A., Rechberger, C.: A 3-Subset Meet-in-the-Middle Attack: Cryptanalysis of the Lightweight Block Cipher KTANTAN. In: Biryukov, A., Gong, G., Stinson, D.R. (eds.) SAC 2010. LNCS, vol. 6544, pp. 229–240. Springer (2011)
16. Cho, J.: Linear Cryptanalysis of Reduced-Round PRESENT. In: Pieprzyk, J. (ed.) CT-RSA 2010. LNCS, vol. 5985, pp. 302–317. Springer (2010)
17. Collard, B., Standaert, F.X.: A Statistical Saturation Attack against the Block Cipher PRESENT. In: Fischlin, M. (ed.) CT-RSA 2009. LNCS, vol. 5473, pp. 195–210. Springer (2009)
18. Daemen, J., Knudsen, L. R., Rijmen, V.: The block cipher Square. In Biham, E., editor, Fast Software Encryption, FSE 1997, LNCS, vol. 1267, pp. 149–165. Springer (1997)
19. Daemen, J., Peeters, M., Van Assche, G., Rijmen, V.: Nessie Proposal: the Block Cipher Noekeon, Nessie submission (2000), http://gro.noekeon.org/
20. Daemen, J., Rijmen, V.: The Design of Rijndael: AES - The Advanced Encryption Standard. Springer (2002)
21. De Cannière, C.: Analysis and Design of Symmetric Encryption Algorithms, Doctoral Dissertaion, KULeuven (2007)
22. De Cannière, C., Dunkelman, O., Knežević, M.: KATAN and KTANTAN - A Family of Small and Efficient Hardware-Oriented Block Ciphers. In: Clavier, C., Gaj, K. (eds.) CHES'09. LNCS, vol. 5747, pp. 272–288. Springer (2009)
23. De Cannière, C., Preneel, B. Trivium. In: Robshaw, M., Billet , O.(eds.), New Stream Cipher Designs - The eSTREAM Finalists, LNCS, vol. 4986. pp. 244–266, Springer (2008)
24. Eisenbarth, T., Gong, Z., Güneysu, T., Heyse, S., Indesteege, S., Kerckhof, S., Koeune, F., Nad, T., Plos, T., Regazzoni, F., Standaert, F.-X., Oldenzeel, L.O.: Compact Implementation and Performance Evaluation of Block Ciphers in ATtiny Devices, AfricaCrypt'2012, LNCS, vol. 7374, pp. 172–187, Springer (2012)
25. Engels, D., Saarinen, M.-J.O., Schweitzer, P., Smith, E.M.: The Hummingbird-2 Lightweight Authenticated Encryption Algorithm. In: Juels, A., Paar, C. (eds.) RFIDSec 2011. LNCS, vol. 7055, pp. 19–31. Springer (2012)
26. Feldhofer, M., Wolkerstorfer, J., Rijmen, V.: AES Implementation on A Grain of Sand. IEE Proceedings on Information Security 152(1), 13–20 (2005)
27. Hermelin, M., Cho, J.Y., Nyberg, K.: Multidimensional Extension of Matsui's Algorithm 2. In: Dunkelman, O. (ed.) FSE 2009. LNCS, vol. 5665, pp. 209–227. Springer (2009)
28. Gong, Z., Nikova, S., Law, Y.W.: KLEIN: A New Family of Lightweight Block Ciphers. In: Juels, A., Paar, C. (eds.) RFIDSec 2011. LNCS, vol. 7055, pp. 1–18. Springer (2012)
29. Good, T., Benaissa, M.: Hardware Results for Selected Stream Cipher Candidates. In: Preproceedings of SASC 2007, pp. 191–204 (2007)
30. Guo, J., Peyrin, T., Poschmann, A., Robshaw, M.: The LED Block Cipher. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 326–341. Springer (2011)
31. Hwang, D., Tiri, K., Hodjat, A., Lai, B.-C., Yang, S., Schaumont, P., Verbauwhede, I.: AES-Based Security Coprocessor IC in 0.18-μm CMOS with Resistance to Differential Power Analysis Side-Channel Attacks. In: IEEE Transactions on Solid-State Circuits, pp. 781–792 (2006)
32. Knudsen, L.R., Wagner, D.: Integral Cryptanalysis. In: Daemen, J., Rijmen, V. (eds.) FSE 2002. LNCS, vol. 2365, pp. 112–127. Springer (2002)
33. Leander, G.: On Linear Hulls, Statistical Saturation Attacks, PRESENT and a Cryptanalysis of PUFFIN. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 303–322. Springer (2011)
34. Leander, G., Poschmann, A.: On the Classification of 4 bit S-boxes. In: Carlet, C., Sunar, B. (eds.) WAIFI 2007. LNCS, vol. 4547, pp. 159–176. Springer (2007)
35. Leander, G., Paar, C., Poschmann, A., Schramm, K.: New Lightweight DES Variants. In: Biryukov, A. (ed.) FSE 2007. LNCS, vol. 4593, pp. 196–210. Springer (2007)
36. Matsuda, S., Moriai S.: Lightweight Cryptography for the Cloud: Exploit the Power of Bitslice Implementation. In: Prouff E., Schaumont P. (eds.), CHES 2012, LNCS, vol. 7428, 408–425. Springer (2012)
37. Matsui, M.: Linear Cryptanalysis Method for DES Cipher. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 386–397. Springer (1994)

38. Matsui, M.: On Correlation between the Order of S-Boxes and the Strength of DES. In: De Santis, A. (ed.) EUROCRYPT 1994. LNCS, vol. 950, pp. 366–375. Springer (1995)
39. Matsui, M., Nakajima, J.: On the Power of Bitslice Implementation on Intel Core2 Processor. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 121–134. Springer (2007)
40. Moradi, A., Poschmann, A., Ling, S., Paar, C., and Wang. H., Pushing the Limits: A Very Compact and a Threshold Implementation of AES. In K. Paterson, editor, Advances in Cryptology - EUROCRYPT 2011, LNCS, vol. 6632, pp. 69–88. Springer (2011)
41. Naya-Plasencia, M., Peyrin, T.: Practical Cryptanalysis of ARMADILLO2. In: Canteaut, A. (ed.) FSE 2012. LNCS, vol. 7549, pp. 146–162. Springer (2012)
42. Ohkuma, K.: Weak Keys of Reduced-Round PRESENT for Linear Cryptanalysis. In: Jacobson Jr., M.J., Rijmen, V., Safavi-Naini, R. (eds.) SAC 2009. LNCS, vol. 5867, pp. 249–265. Springer (2009)
43. Plos, T., Dobraunig, C., Hofinger, M., Oprisnik, A., Wiesmeier, C., Wiesmeier, J.: Compact Hardware Implementations of the Block Ciphers mCrypton, NOEKEON, and SEA. In Galbraith, S., Nandi, M. (eds.) Progress in Cryptology-INDOCRYPT 2012. LNCS, vol. 7668, pp. 358–377. Springer (2012)
44. RECTANGLE hardware impelementation codes: http://homes.esat.kuleuven.be/ byang/rectangle/
45. Rolfes, C., Poschmann, A., Leander, G., Paar, C.: Ultra-Lightweight Implementations for Smart Devices - Security for 1000 Gate Equivalents. In: Grimaud, G., Standaert, F.-X. (eds.) CARDIS 2008. LNCS, vol. 5189, pp. 89–103. Springer (2008)
46. Saarinen, M.-J.O.: Cryptanalysis of Hummingbird-1. In: Joux, A. (ed.) FSE 2011. LNCS, vol. 6733, pp. 328–341. Springer (2011)
47. Shan J Y, Hu L, Song L, et al. Related-Key Differential Attack on Round Reduced RECTANGLE-80, Cryptology ePrint Archive: Report 2014/986. http://eprint.iacr.org/2014/986
48. Shibutani, K., Isobe, T., Hiwatari, H., Mitsuda, A., Akishita, T. and Shirai, T. Piccolo: An Ultra-Lightweight Blockcipher. In: Preneel, B., Takagi, T. (eds.) CHES 2011, LNCS, vol. 6917, pp. 342–57. Springer (2011)
49. Suzaki, T., Minematsu, K., Morioka, S., Kobayashi, E.: Twine: A Lightweight, Versatile Blockcipher. In: ECRYPT Workshop on Lightweight Cryptography (2011), http://www.uclouvain.be/crypto/
50. Wang, M., Sun, Y., Tischhauser, E., Preneel, B.: A Model for Structure Attacks, with Applications to PRESENT and Serpent. In: Canteaut, A. (ed.) FSE 2012. LNCS, vol. 7549, pp. 49–68. Springer (2012)
51. Wu, H.: The Hash Function JH. Submission to NIST (2008), http://icsd.i2r.a-star.edu.sg/staff/hongjun/jh/jh.pdf
52. Wu, W., Zhang, L.: LBlock: A Lightweight Block Cipher. In: Lopez, J., Tsudik, G. (eds.) ACNS 2011. LNCS, vol. 6715, pp. 327–344. Springer (2011)
53. Zhang, W., Bao, Z., Rijmen, V., Liu, M.: A New Classification of 4-bit Optimal S-boxes and its Application to PRESENT, RECTANGLE and SPONGENT, In: Leander, G. (ed.) FSE 2015. LNCS, vol.9054, pp. 494–515. Springer (2015)

A The round constants

$RC[0] = 0X01$, $RC[1] = 0X02$, $RC[2] = 0X04$, $RC[3] = 0X09$, $RC[4] = 0X12$, $RC[5] = 0X05$, $RC[6] = 0X0B$, $RC[7] = 0X16$, $RC[8] = 0X0C$, $RC[9] = 0X19$, $RC[10] = 0X13$, $RC[11] = 0X07$, $RC[12] = 0X0F$, $RC[13] = 0X1F$, $RC[14] = 0X1E$, $RC[15] = 0X1C$, $RC[16] = 0X18$, $RC[17] = 0X11$, $RC[18] = 0X03$, $RC[19] = 0X06$, $RC[20] = 0X0D$, $RC[21] = 0X1B$, $RC[22] = 0X17$, $RC[23] = 0X0E$, $RC[24] = 0X1D$.

B Test Vectors

See Table 10.

C A Part of Experimental Results against Statistical Saturation Attack

See table 11.

Table 10. Test Vectors of RECTANGLE

	Key	Plaintext	Ciphertext
REC-80	0000000000000000 0000000000000000 0000000000000000 0000000000000000 0000000000000000	0000000000000000 0000000000000000 0000000000000000 0000000000000000	0010110110010110 1110001101010100 1110100010110001 0000100001110100
REC-80	1111111111111111 1111111111111111 1111111111111111 1111111111111111 1111111111111111	1111111111111111 1111111111111111 1111111111111111 1111111111111111	1001100101000101 1010101000110100 1010111000111101 0000000100010010
REC-128	00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000	0000000000000000 0000000000000000 0000000000000000 0000000000000000	1010111011100110 0011011000010011 0100010010100100 1001100111101110
REC-128	11111111111111111111111111111111 11111111111111111111111111111111 11111111111111111111111111111111 11111111111111111111111111111111	1111111111111111 1111111111111111 1111111111111111 1111111111111111	1110100000111110 1110111111101110 0100101000010101 0111101001000110

Table 11. Experimental Results using the 2nd Property

	Dis for 8 rounds	Dis for 9 rounds	Dis for 10 rounds
1	$2^{-27.49}$	$2^{-33.61}$	$2^{-39.13}$
2	$2^{-28.88}$	$2^{-33.87}$	$2^{-39.35}$
3	$2^{-31.13}$	$2^{-35.04}$	$2^{-39.62}$
4	$2^{-31.92}$	$2^{-35.77}$	$2^{-39.64}$
5	$2^{-30.70}$	$2^{-33.43}$	$2^{-39.01}$
6	$2^{-33.17}$	$2^{-34.70}$	$2^{-39.30}$
7	$2^{-29.81}$	$2^{-33.01}$	$2^{-38.89}$
8	$2^{-32.65}$	$2^{-35.09}$	$2^{-39.54}$
9	$2^{-31.57}$	$2^{-34.32}$	$2^{-39.39}$
10	$2^{-31.49}$	$2^{-34.03}$	$2^{-39.03}$
11	$2^{-31.56}$	$2^{-36.82}$	$2^{-39.79}$
12	$2^{-30.13}$	$2^{-37.29}$	$2^{-39.54}$
13	$2^{-32.07}$	$2^{-36.41}$	$2^{-39.6}$
14	$2^{-31.14}$	$2^{-34.7}$	$2^{-37.37}$
15	$2^{-28.93}$	$2^{-34.49}$	$2^{-39.26}$
16	$2^{-29.52}$	$2^{-34.18}$	$2^{-39.45}$
AVG	$2^{-29.94}$	$2^{-34.39}$	$2^{-39.10}$

each value in the last row is an average over the 16 values in the same column.

D A Bit-slice Description of RECTANGLE

In the following, we present an equivalent description of SubColumn and ShiftRow transformations. Based on them, one can easily write a code for a software implementation of RECTANGLE, i.e., a bit-slice implementation. Our software implementation of RECTANGLE is just based on these results. Although the representation of *SubColumn* given here is also suitable for a hardware implementation, we point out that it

is not the best one. Indeed, we have found a better representation of $SubColumn$ when it comes to area-first hardware implementation.

D.1 SubColumn

As shown in Fig. 2, a 64-bit state is described as a 4×16 array. Let $A_i = a_{i,15}||a_{i,14}||\cdots||a_{i,2}||a_{i,1}||a_{i,0}$ denote the i-th row, $i = 0, 1, 2, 3$. A_i can be regarded as a 16-bit word.

Let A_0, A_1, A_2, A_3 be 4 16-bit inputs of SubColumn, B_0, B_1, B_2, B_3 be the 4 16-bit outputs, where A_i and B_i denote the i-th row of the cipher state. Let T_i denote 16-bit temporary variables, $i = 1, 2, 3, 5, 6, 8, 9, 11, 12$. The SubColumn transformation can be computed in the following 12 steps:

1. $T_1 = \sim A_1$; 2. $T_2 = A_0 \,\&\, T_1$; 3. $T_3 = A_2 \oplus A_3$; 4. $B_0 = T_2 \oplus T_3$;
5. $T_5 = A_3 | T_1$; 6. $T_6 = A_0 \oplus T_5$; 7. $B_1 = A_2 \oplus T_6$; 8. $T_8 = A_1 \oplus A_2$;
9. $T_9 = T_3 \,\&\, T_6$ 10. $B_3 = T_8 \oplus T_9$; 11. $T_{11} = B_0 | T_8$; 12. $B_2 = T_6 \oplus T_{11}$;

where "$\sim$" is NOT, "&" is bitwise AND, "$|$" is bitwise OR.

D.2 ShiftRow

Let B_0, B_1, B_2, B_3 be 4 16-bit inputs of ShiftRow transformation, C_0, C_1, C_2, C_3 the 4 16-bit outputs. Then:

$C_0 = B_0$; $C_1 = B_1 \lll 1$; $C_2 = B_2 \lll 12$; $C_3 = B_3 \lll 13$.

where "$A \lll x$" denotes a left rotation over x bits within a 16-bit word A.

E One 14-round Difference Propagation

For 14-round RECTANGLE, the probability of the best differential trail is 2^{-61}. We have searched for all 14-round differential trails with probability between 2^{-61} and 2^{-71} (up to a rotation equivalence), and examined all the difference propagations made up of these investigated trails, the following are the experimental results:

1. There are 32 best difference propagations with probability $1300 \times 2^{-71} \approx 2^{-60.66}$ each.
2. Among all the difference propagations, the maximum number of trails of a difference propagation is 115.
3. There are 11128 difference propagations with a probability larger than 2^{-64}.

Among the 11128 effective 14-round difference propagations, we choose one from the point of view of the highest number of attacked rounds. The difference propagation we have chosen is composed of 2 differential trails. Among the 2 trails, one with probability 2^{-63}, the other with probability 2^{-66}, thus the total probability of this difference propagation is $2^{-63} + 2^{-66} \approx 2^{-62.83}$. Using the above 14-round difference propagation, we can mount an attack on 18-round RECTANGLE. The attack complexity is as follows. The data complexity is 2^{64} plaintexts. The memory complexity is 2^{72} key counters. For a 80-bit seed key, the time complexity is about $2^{78.67}$ 18-round encryptions; for a 128-bit seed key, the time complexity is about $2^{126.66}$ 18-round encryptions. The success rate of the attack is about 67.5%.

The following table gives the input difference and the output difference of the 14-round difference propagation:

Input Difference of Round 0	Output Difference of Round 13
0000000000000000	0000000000000000
0010000100000000	0000000000000010
0000000100000000	0001000000000000
0000000000000000	0000000000000000

The first differential trail with probability 2^{-63}:

Round Index	Round Prob. ($-log_2$)	Input Difference of the Round	Output Difference of the S-box
0	5	0000000000000000 0010000100000000 0000000100000000 0000000000000000	0000000000000000 0010001000000000 0010000000000000 0000000000000000
1	5	0000000000000000 0100001000000000 0000001000000000 0000000000000000	0000000000000000 0100001000000000 0100000000000000 0000000000000000
2	5	0000000000000000 1000010000000000 0000010000000000 0000000000000000	0000000000000000 1000010000000000 1000000000000000 0000000000000000
3	5	0000000000000000 0000100000000002 0000100000000000 0000000000000000	0000000000000000 0000000000000000 0000000000000000 0000000000000000
4	5	0000000000000000 0001000000000010 0001000000000000 0000000000000000	0000000000000000 0001000000000010 0000000000000010 0000000000000000
5	5	0000000000000000 0010000000000100 0010000000000000 0000000000000000	0000000000000000 0010000000000100 0000000000000100 0000000000000000
6	5	0000000000000000 0100000000001000 0100000000000000 0000000000000000	0000000000000000 0100000000001000 0000000000001000 0000000000000000
7	5	0000000000000000 1000000000010000 1000000000000000 0000000000000000	0000000000000000 0100000000010000 0000000000010000 0000000000000000
8	5	0000000000000000 0000000000100001 0000000000000001 0000000000000000	0000000000000000 0000000000100001 0000000000100000 0000000000000000
9	5	0000000000000000 0000000001000010 0000000000000010 0000000000000000	0000000000000000 0000000000000010 0000000001000000 0000000001000000
10	5	0000000000000000 0000000000000100 0000000000000100 0000000000001000	0000000000001000 0000000000000100 0000000000000000 0000000000000000
11	3	0000000000001000 0000000000001000 0000000000000000 0000000000000000	0000000000000000 0000000000000000 0000000000000000 0000000000001000
12	3	0000000000000000 0000000000000000 0000000000000000 0000000000000001	0000000000000001 0000000000000000 0000000000000000 0000000000000000
13	2	0000000000000001 0000000000000000 0000000000000000 0000000000000000	0000000000000000 0000000000000001 0000000000000001 0000000000000000

The second differential trail with probability 2^{-66}:

Round Index	Round Prob. ($-log_2$)	Input Difference of the round	Output Difference of the S-box
0	5	0000000000000000 0010000100000000 0000000100000000 0000000000000000	0000000000000000 0010001000000000 0010000000000000 0000000000000000
1	5	0000000000000000 0100001000000000 0000001000000000 0000000000000000	0000000000000000 0100001000000000 0100000000000000 0000000000000000
2	5	0000000000000000 1000010000000000 0000010000000000 0000000000000000	0000000000000000 1000010000000000 1000000000000000 0000000000000000
3	5	0000000000000000 0000100000000002 0000100000000000 0000000000000000	0000000000000000 0000000000000000 0000000000000000 0000000000000000
4	5	0000000000000000 0001000000000010 0001000000000000 0000000000000000	0000000000000000 0001000000000010 0000000000000010 0000000000000000
5	5	0000000000000000 0010000000000100 0010000000000000 0000000000000000	0000000000000000 0010000000000100 0000000000000100 0000000000000000
6	5	0000000000000000 0100000000001000 0100000000000000 0000000000000000	0000000000000000 0100000000001000 0000000000001000 0000000000000000
7	5	0000000000000000 1000000000010000 1000000000000000 0000000000000000	0000000000000000 0100000000010000 0000000000010000 0000000000000000
8	5	0000000000000000 0000000000100001 0000000000000001 0000000000000000	0000000000000000 0000000000100001 0000000000100000 0000000000000000
9	5	0000000000000000 0000000001000010 0000000000000010 0000000000000000	0000000000000000 0000000000000010 0000000001000000 0000000001000000
10	5	0000000000000000 0000000000000100 0000000000001100 0000000000001000	0000000000001000 0000000000000100 0000000000000000 0000000000001000
11	6	0000000000001000 0000000000001000 0000000000000000 0000000000000001	0000000000000001 0000000000000000 0000000000000000 0000000000001000
12	3	0000000000000001 0000000000000000 0000000000000000 0000000000000001	0000000000000001 0000000000000000 0000000000000000 0000000000000000
13	2	0000000000000001 0000000000000000 0000000000000000 0000000000000000	0000000000000000 0000000000000001 0000000000000001 0000000000000000