TWINE: A Lightweight, Versatile Block Cipher

Tomoyasu Suzaki, Kazuhiko Minematsu, Sumio Morioka, and Eita Kobayashi

NEC Corporation, 1753 Shimonumabe, Nakahara-Ku, Kawasaki, Japan
t-suzaki@cb.jp.nec.com, k-minematsu@ah.jp.nec.com,
s-morioka@ak.jp.nec.com, e-kobayashi@fg.jp.nec.com

Abstract. This paper presents a 64-bit lightweight block cipher TWINE supporting 80 and 128-bit keys. It enables quite small hardware implementation similar to the previous proposals, yet enables efficient implementations on embedded software. Moreover, it allows a compact implementation of unified encryption and decryption. This characteristic mainly originates from the use of generalized Feistel with many subblocks combined with a recent improvement on the diffusion layer.
Keywords: lightweight block cipher, generalized Feistel, block shuffle

1 Introduction

Recent advances in tiny computing devices, such as RFID and sensor network nodes, give rise to the need of symmetric encryption with highly-limited resources, called lightweight encryption. While we have AES it is often inappropriate for such devices due to their size/power/memory constraints, even though there are constant efforts for small-footprint AES, e.g., [14, 31, 37]. To fill the gap, a number of hardware-oriented lightweight block ciphers have been recently proposed; for instance, DESL [26], HIGHT [21], PRESENT [9], and KATAN/KTANTAN [13], PRINTcipher [25], and many more.

In this paper, we propose a new lightweight 64-bit block cipher TWINE. It supports 80 and 128-bit keys. Our purpose is to achieve hardware efficiency while minimizing the hardware-oriented design choices, such as a bit permutation. The avoidance of such options may be beneficial to software implementation and yield a balanced performance on both software and hardware. Lightweight blockciphers from a similar motivation are also seen in, e.g., KLEIN [19], LBlock [43], and most recently, LED [18] and Piccolo [40].

For this purpose, we employ Type-2 generalized Feistel structure (GFS) proposed[1] by Zheng et al. [44] with 16 nibble-blocks. The drawback of such design is poor (slow) diffusion property, leading to a slow cipher with quite many rounds. To overcome the problem, we employ the idea of Suzaki and Minematsu at FSE '10 [41] which substantially improves diffusion by using a different block shuffle from the original (cyclic shift). As a result, TWINE is also efficient on (embedded) software and enables compact unification of encryption and decryption.

[1] Zheng et al. called it Type-2 Feistel-Type Transformation. The generalized Feistel structure is an alias taken by, e.g., [39, 41].

The features of our proposal are (1) no bit permutation, (2) generalized Feistel-based, and (3) no Galois-Field matrix. The components are only one 4-bit S-box and 4-bit XOR. As far as we know, this is the first attempt that combines these three features. There is a predecessor called LBlock [43] which has some resemblances to ours, however TWINE is an independent work and has several concrete design advantages (See Section 3).

We implemented TWINE on hardware (ASIC and FPGA) and software (8-bit microcontroller). We did not take the fixed key setting, hence keys can be updated. Our hardware implementations suggest that the encryption-only TWINE can be implemented with about $1,500$ Gate Equivalent (GE), and when encryption and decryption are unified, it can be implemented within $1,800$ GEs. We are also trying a serialized implementation and a preliminary result suggests $1,116$ GEs. For software, TWINE is implemented within 0.8 to 1.5 Kbytes ROM. The speed is relatively fast compared to other lightweight ciphers. Though the hardware size is not the smallest, we think the performance of TWINE is well-balanced for both hardware and software.

For security, TWINE employs a technique to enhance the diffusion of GFS, however, it is definitely important to evaluate the security against attacks that exploit the diffusion property of generalized Feistel, such as the impossible differential attack and the saturation attack. We perform a thorough analysis (for a new cipher proposal) on TWINE and present the impossible differential attack against 23-round TWINE-80 and 24-round TWINE-128 as the most powerful attacks we have found so far. The attack is ad-hoc and fully exploits the key schedule, which can be of independent interest as an example of highly-optimized impossible differential attack against GFS-based ciphers.

The organization of the paper is as follows. In Section 2 we describe the specification of TWINE. Section 3 explains the design rationale for TWINE. In Section 4 we present the result of security evaluation, and in section 5 we present the implementation results of both hardware and software. Section 6 concludes the paper.

2 Specification of TWINE

2.1 Notations

The basic mathematical operations we use are as follows. $\oplus$ denotes bitwise exclusive-OR. For binary strings, x and y, $x\|y$ denotes the concatenation of x and y. Let $|x|$ denote the bit length of a binary string x. If $|x| = m$, x is also written as $x_{(m)}$ to emphasize its bit length. If $|x| = 4c$ for some positive integer c, we write $x \rightarrow (x_0\|x_1\|\ldots\|x_{c-1})$, where $|x_i| = 4$, is the partition operation into the 4-bit subsequences. The opposite operation, $(x_0\|x_1\|\ldots\|x_{c-1}) \rightarrow x$, is similarly defined. The partition operation may be implicit, i.e., we may simply write x_i to denote the i-th 4-bit subsequence for any $4c$-bit string x.

2.2 Data Processing Part (Encryption Process)

TWINE is a 64-bit block cipher with two supported key lengths, 80 and 128 bits. If the key length is needed to be specified, we write TWINE-80 or TWINE-128 to denote the corresponding version. The global structure of TWINE is a variant of Type-2 GFS [44] [38] with 16 4-bit sub-blocks. Given a 64-bit plaintext, $P_{(64)}$, and a round key, $RK_{(32\times 36)}$, the cipher produces the ciphertext $C_{(64)}$. Round key $RK_{(32\times 36)}$ is derived from the secret key, $K_{(n)}$ with $n \in \{80, 128\}$, using the key schedule. A round function of TWINE consists of a nonlinear layer using 4-bit S-boxes and a diffusion layer, which permutes the 16 blocks. Unlike Type-2 GFS, the diffusion layer is not a circular shift and is designed to provide a better diffusion that the circular shift, according the result of [41]. This round function is iterated for 36 times for both key lengths, where the diffusion layer of the last round is omitted. The encryption process can be written as Algorithm 2.1.

The S-box, S, is a 4-bit permutation defined as Table 1. The permutation of block indexes, $\pi : \{0, \ldots, 15\} \rightarrow \{0, \ldots, 15\}$, where j-th sub-block (for $j = 0, \ldots, 15$) is mapped to $\pi[j]$-th sub-block, is depicted at Table 2.

The figure of the round function is in Fig. 1.

Algorithm 2.1: TWINE.Enc$(P_{(64)}, RK_{(32\times 36)}, C_{(64)})$

$X^1_{(64)} \leftarrow P$
$RK^1_{(32)} \| \ldots \| RK^{35}_{(32)} \leftarrow RK_{(32\times 36)}$
for $i \leftarrow 1$ **to** 35
do
- $X^i_{0(4)} \| X^i_{1(4)} \| \ldots \| X^i_{14(4)} \| X^i_{15(4)} \leftarrow X^i_{(64)}$
- $RK^i_{0(4)} \| RK^i_{1(4)} \| \ldots \| RK^i_{6(4)} \| RK^i_{7(4)} \leftarrow RK^i_{(32)}$
- **for** $j \leftarrow 0$ **to** 7
  - **do** $X^i_{2j+1} \leftarrow S(X^i_{2j} \oplus RK^i_j) \oplus X^i_{2j+1}$
- **for** $h \leftarrow 0$ **to** 15
  - **do** $X^{i+1}_{\pi[h]} \leftarrow X^i_h$
- $X^{i+1} \leftarrow X^{i+1}_0 \| X^{i+1}_1 \| \ldots \| X^{i+1}_{14} \| X^{i+1}_{15}$

for $j \leftarrow 0$ **to** 7
do $X^{36}_{2j+1} \leftarrow S(X^{36}_{2j} \oplus RK^{36}_j) \oplus X^{36}_{2j+1}$
$C \leftarrow X^{36}$

Table 1. S-box Mapping in the Hexadecimal Notation.

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

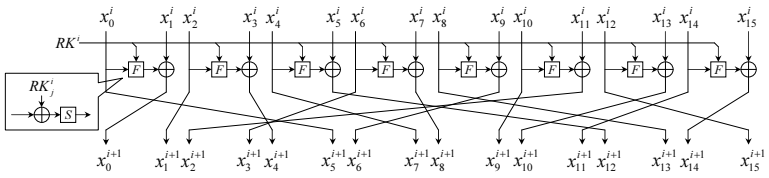

Fig. 1. Round function of TWINE. Data path is 4-bit and each F function is of the form $y = S(x \oplus k)$ with a 4-bit S-box.

Table 2. Block Shuffle π and its Inverse π^{-1}.

j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\pi[j]$	5	0	1	4	7	12	3	8	13	6	9	2	15	10	11	14
$\pi^{-1}[j]$	1	2	11	6	3	0	9	4	7	10	13	14	5	8	15	12

2.3 Key Schedule Part

The key schedule produces $RK_{(32\times 36)}$ from the secret key, $K_{(n)}$, where $n \in \{80, 128\}$. As well as the data processing part, it is a variant of GFS but with much sparser nonlinear functions. The pseudocode of key schedule for 80-bit key is in Algorithm 2.2 and the figure is in Appendix C. For 128-bit key, see Appendix A. Round constant, $\mathrm{CON}^i_{(6)} = \mathrm{CON}^i_{H(3)} \| \mathrm{CON}^i_{L(3)}$, is defined as 2^i in $\mathrm{GF}(2^6)$ with primitive polynomial $z^6 + z + 1$. The exact values are listed at Table 3.

Table 3. Round Constants. CON^i is the rightmost 6-bit (of 8 bits expressed in hexadecimal notation).

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
CON^i	01	02	04	08	10	20	03	06	0C	18	30	23	05	0A	14	28	13	26
i	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36
CON^i	0F	1E	3C	3B	35	29	11	22	07	0E	1C	38	33	25	09	12	24	0B

Algorithm 2.2: TWINE.KeySchedule-80($K_{(80)}, RK_{(32\times 36)}$)

$\mathrm{WK}_{(80)} \leftarrow K$
$\mathrm{WK}_{0(4)} \| \mathrm{WK}_{1(4)} \| \ldots \| \mathrm{WK}_{18(4)} \| \mathrm{WK}_{19(4)} \leftarrow WK$
$\mathrm{RK}^1_{0(4)} \leftarrow \mathrm{WK}_1,\ \mathrm{RK}^1_{1(4)} \leftarrow \mathrm{WK}_3,\ \mathrm{RK}^1_{2(4)} \leftarrow \mathrm{WK}_4,\ \mathrm{RK}^1_{3(4)} \leftarrow \mathrm{WK}_6$
$\mathrm{RK}^1_{4(4)} \leftarrow \mathrm{WK}_{13},\ \mathrm{RK}^1_{5(4)} \leftarrow \mathrm{WK}_{14},\ \mathrm{RK}^1_{6(4)} \leftarrow \mathrm{WK}_{15},\ \mathrm{RK}^1_{7(4)} \leftarrow \mathrm{WK}_{16}$
$\mathrm{RK}^1_{(32)} \leftarrow \mathrm{RK}^1_0 \| \mathrm{RK}^1_1 \| \ldots \| \mathrm{RK}^1_6 \| \mathrm{RK}^1_7$
for $i \leftarrow 2$ **to** 36
do
- $\mathrm{WK}_1 \leftarrow \mathrm{WK}_1 \oplus S(\mathrm{WK}_0)$
- $\mathrm{WK}_4 \leftarrow \mathrm{WK}_4 \oplus S(\mathrm{WK}_{16})$
- $\mathrm{WK}_7 \leftarrow \mathrm{WK}_7 \oplus 0 \| CON^{i-1}_H$
- $\mathrm{WK}_{19} \leftarrow \mathrm{WK}_{19} \oplus 0 \| CON^{i-1}_L$
- $\mathrm{tmp}_0 \leftarrow \mathrm{WK}_0,\ \mathrm{tmp}_1 \leftarrow \mathrm{WK}_1,\ \mathrm{tmp}_2 \leftarrow \mathrm{WK}_2,\ \mathrm{tmp}_3 \leftarrow \mathrm{WK}_3$
- **for** $j \leftarrow 0$ **to** 3
  - **do** $\mathrm{WK}_{j*4} \leftarrow \mathrm{WK}_{j*4+4},\ \mathrm{WK}_{j*4+1} \leftarrow \mathrm{WK}_{j*4+5}$
  - **do** $\mathrm{WK}_{j*4+2} \leftarrow \mathrm{WK}_{j*4+6},\ \mathrm{WK}_{j*4+3} \leftarrow \mathrm{WK}_{j*4+7}$
- $\mathrm{WK}_{16} \leftarrow \mathrm{tmp}_1, \mathrm{WK}_{17} \leftarrow \mathrm{tmp}_2, \mathrm{WK}_{18} \leftarrow \mathrm{tmp}_3, \mathrm{WK}_{19} \leftarrow \mathrm{tmp}_0$
- $\mathrm{RK}^i_0 \leftarrow \mathrm{WK}_1,\ \mathrm{RK}^i_1 \leftarrow \mathrm{WK}_3,\ \mathrm{RK}^i_2 \leftarrow \mathrm{WK}_4,\ \mathrm{RK}^i_3 \leftarrow \mathrm{WK}_6$
- $\mathrm{RK}^i_4 \leftarrow \mathrm{WK}_{13},\ \mathrm{RK}^i_5 \leftarrow \mathrm{WK}_{14},\ \mathrm{RK}^i_6 \leftarrow \mathrm{WK}_{15},\ \mathrm{RK}^i_7 \leftarrow \mathrm{WK}_{16}$
- $\mathrm{RK}^i_{(32)} \leftarrow \mathrm{RK}^i_{0(4)} \| \mathrm{RK}^i_{1(4)} \| \ldots \| \mathrm{RK}^i_{6(4)} \| \mathrm{RK}^i_{7(4)}$

$RK_{(32\times 36)} \leftarrow \mathrm{RK}^1_{(32)} \| \mathrm{RK}^2_{(32)} \| \ldots \| \mathrm{RK}^{35}_{(32)} \| \mathrm{RK}^{36}_{(32)}$

2.4 Decryption Process

The decryption of TWINE is quite similar to the encryption; we use the same S-box and key schedule as used in the encryption, with the inverse block shuffle. See Algorithm 2.3.

Algorithm 2.3: TWINE.Dec($C_{(64)}, RK_{(32\times 36)}, P_{(64)}$)

$X^{36}_{(64)} \leftarrow C$
$\mathrm{RK}^0_{(32)} \| \ldots \| \mathrm{RK}^{35}_{(32)} \leftarrow RK_{(32\times 36)}$
for $i \leftarrow 36$ **to** 2
do
- $X^i_{0(4)} \| X^i_{1(4)} \| \ldots \| X^i_{14(4)} \| X^i_{15(4)} \leftarrow X^i_{(64)}$
- $\mathrm{RK}^i_{0(4)} \| \mathrm{RK}^i_{1(4)} \| \ldots \| \mathrm{RK}^i_{6(4)} \| \mathrm{RK}^i_{7(4)} \leftarrow \mathrm{RK}^i_{(32)}$
- **for** $j \leftarrow 0$ **to** 7
  - **do** $X^i_{2j+1} \leftarrow S(X^i_{2j} \oplus \mathrm{RK}^i_j) \oplus X^i_{2j+1}$
- **for** $h \leftarrow 0$ **to** 15
  - **do** $X^{i-1}_{\pi^{-1}[h]} \leftarrow X^i_h$
- $X^{i-1} \leftarrow X^{i-1}_0 \| X^{i-1}_1 \| \ldots \| X^{i-1}_{14} \| X^{i-1}_{15}$

for $j \leftarrow 0$ **to** 7
do $X^1_{2j+1} \leftarrow S(X^1_{2j} \oplus \mathrm{RK}^1_j) \oplus X^1_{2j+1}$
$P \leftarrow X^1$

3 Design Rationale

3.1 Basic Objective

We focus on the mixed environments of resource-constrained hardware and software, and aim at building a block cipher with a balanced performance under such environments. Specifically, our goals are

1. small footprint in hardware implementation (e.g. under 2,000 GE [23, 33]),
2. small ROM/RAM consumption in software implementation,
3. these goals achieved for the unified encryption/decryption functionality.

The importance of the last item is also pointed out by [1].

On LBlock. We remark that LBlock [43], proposed independently of ours, is quite similar to our proposal. It is a 64-bit block cipher based on the balanced Feistel whose round function consists of 8 4-bit S-boxes followed by a 4-bit block-wise permutation (hence no matrix operation as used by SPN). LBlock also performs 8-bit rotation to the round function's output. Such a structure can be transformed into a further generalized Type-2 GFS proposed by [41], though we do not know whether the authors of LBlock are aware of it. We investigated LBlock in this respect and found that the LBlock's diffusion layer is equivalent to that of the decryption of our proposal. Note that this choice is quite reasonable from Table 6 of [41], as it satisfies both of the fastest diffusion and the highest immunities against linear and differential attacks among other block shuffles.

Nevertheless, there are some important differences between TWINE and LBlock, as follows;

1. LBlock uses ten distinct S-boxes while ours uses one S-box. The use of single S-box rather than multiple ones can contribute to smaller (serialized) hardware and software implementations.
2. LBlock uses a bit permutation in its key scheduling, while ours is completely bit permutation-free, including the key schedule. Hence the design of LBlock does not meet our criteria mentioned at Introduction.

We also would like to point out that the security evaluation of LBlock is insufficient. We already found a saturation attack against 22-round version without considering the key schedule, thus the security margin is smaller than the claimed by the authors (20-round). Using the techniques presented at Section 4, we expect further improvements on the attack.

3.2 Parameters and Components

Considering the basic design goals as above, we choose the 64-bit block size with 80 and 128-bit keys, which is compatible to many previous lightweight blockciphers. The number of rounds is determined from our security analysis. As far as we investigated, the most powerful attack against TWINE is a dedicated

impossible differential attack, which breaks 23-round TWINE-80 and 24-round TWINE-128. From this, we consider 36-round TWINE-128 has a sufficient security margin. When keeping the same size of margin for TWINE-80 and TWINE-128 is our sole goal, the 80-bit key version could reduce the number of rounds from 36. However, considering the security margin and the implementation merit (i.e. 36 has many factors, which enables various multiple-round hardware implementations with a small overhead), we employ the 36-round structure for both key lengths.

Block Shuffle. The block shuffle π comes from a result of FSE '10 [41]. In [41], it was reported that by changing the block shuffle different from the ordinal cyclic shift one can greatly improve the diffusion of Type-2 GFS. Here, goodness-of-diffusion is measured by the minimum number of rounds that diffuses any input sub-block difference to all output sub-blocks, called DRmax. Smaller DRmax means a faster diffusion. DRmax of cyclic shift with k sub-blocks is k, while there exist shuffles with DRmax $= 2\log_2 k$, called "optimum block shuffle" [41]. Our π is such one[2] with $k = 16$, hence DRmax $= 8$ while DRmax $= 16$ for the cyclic shift. DRmax is connected to the resistance against various attacks. For example, Type-2 GFS with 16 sub-blocks has 33-round impossible differential characteristics and 32-round saturation characteristics. However, by using π of Table 2 they can be reduced to 14 and 15 rounds.

There exist multiple optimum block shuffles [41]. Hence π was chosen considering other aspects which is not (directly) related to DRmax. In particular, we focus on the resistance against differential and linear cryptanalysis, i.e., the number of active S-boxes.

S-box. The 4-bit S-box is chosen to satisfy

1. The maximum differential and linear probabilities are 2^{-2}, which is theoretically the minimum for invertible S-box,
2. The boolean degree is 3,
3. The interpolation polynomial contains many terms and has degree 14.

Following the AES S-box design, we searched S-boxes satisfying the above while being representable as a composition of Galois field inversion and an affine transformation. More precisely, our S-box is defined as $y = S(x) = f((x \oplus b)^{-1})$, where a^{-1} denotes the inverse of a in GF(2^4) (the zero element is mapped to itself.) with irreducible polynomial $z^4 + z + 1$, and $b = 1$ is a constant, and $f(\cdot)$ is an affine function defined as

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 0\ 0\ 1\ 1 \\ 1\ 0\ 0\ 1 \\ 1\ 0\ 0\ 0 \\ 0\ 1\ 0\ 0 \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \tag{1}$$

for $y = f(x)$ with $y = (y_0 \| y_1 \| y_2 \| y_3)$ and $x = (x_0 \| x_1 \| x_2 \| x_3)$.

[2] More precisely, an isomorphic shuffle in Appendix B ($k = 16$, No. 10) of [41].

Key Schedule. The key schedule has many design options. We choose one which enables (1) on-the-fly operations and (2) produces each round key via sequential update of a key state, that is, there is no intermediate key. As mentioned, it uses no bit permutation. As hardware efficiency is not our ultimate goal, the design is rather conservative compared to the recent hardware-oriented ones [13,34,40], yet quite simple. For security, we want our key schedule to have sufficient resistance against slide, meet-in-the-middle, and related-key attacks.

4 Security Evaluation

4.1 Overview

We examined the security of TWINE against various attacks for both 80 and 128-bit keys. Since it is hard to describe all the evaluation results due to the page limit, we focus on the most critical attacks in our evaluation; the impossible differential and saturation attacks. For simplicity we only describe the details of the attacks against TWINE-80; the results on TWINE-128 will be briefly described in the summary. The short summary on other attacks, such as differential and linear attacks, will also be given.

In this section, we use the following notations. $\bar{S}^c$ denotes the sequence of c symbols S, e.g. $\bar{0}^3$ means $(0,0,0)$ and $\bar{U}^3$ means (U,U,U). The F function in the i-th round is labeled as $F_0^i, \ldots, F_{15}^i$, where F_0^i is the leftmost one. We let $\mathrm{RK}^i_{[j_1,\ldots,j_h]}$ to denote the vector $(\mathrm{RK}^i_{j_1}, \ldots, \mathrm{RK}^i_{j_h})$. Since RK^i_j is the j-th 4-bit subsequence of RK^i (for $j = 0, 1, \ldots, 15$), this means $F^i_j(x) = S(\mathrm{RK}^i_j \oplus x)$. $X^i_{[j_1,\ldots,j_h]}$ is similarly defined.

4.2 Impossible Differential Attack

Generally, impossible differential attack [6] is one of the most powerful attack against Feistel and GFS-based ciphers, as demonstrated by (e.g.) [15, 32, 42]. We searched impossible differential characteristics (IDCs) using Kim et al.'s method [24], and found 64 14-round IDCs. We here present a highly-optimized attack against 23-round TWINE-80. It exploits the key schedule and is based on the following 14-round IDC (for 4-bit blocks);

$$(0, \alpha, \bar{0}^{14}) \overset{14r}{\nrightarrow} (\bar{0}^8, \beta, \bar{0}^7), \text{ where } \alpha \neq 0 \text{ and } \beta \neq 0. \tag{2}$$

Our attack uses the above IDC to 5-th to 18-th rounds of TWINE, and tries to recover the subkeys of the first 4 rounds and last 5 rounds, 144 bits in total. These subkey bits are uniquely determined via its 80-bit subsequence; see Appendix D.

The details of our attack are as follows.

Data Collection. We call a set of 2^{32} plaintexts a *structure* if its i-th sub-blocks are fixed to a constant for all $i = 2, 4, 5, 6, 7, 8, 9, 14 \in \{0, \ldots, 15\}$ and the remaining 8 sub-blocks take all 2^{32} values. Suppose we have one structure. From it we extract plaintext pairs having the differential

$$(\alpha_1, \alpha_2, 0, \alpha_3, \bar{0}^6, \alpha_4, \alpha_5, \alpha_6, \alpha_7, 0, \alpha_0), \tag{3}$$

where α_i is a non-zero 4-bit value. For such a plaintext pair, we want to make sure that the differential of the internal state after the first 4 rounds to match the left hand side of Eq. (2). For this, the output differentials with respect to some F functions (in the first 4 rounds) have to be canceled out. For example, α_2 must be the differential of F with input differential α_1, as shown by Fig. 2. Here, we use the following observation;

Proposition 1. *Let $y = F_j^i(x) = S(\mathrm{RK}_j^i \oplus x)$ and $y' = F_j^i(x')$. For fixed $\Delta_x = x \oplus x' \neq 0$, $\Delta_y = y \oplus y'$ has always 7 possible values, for any i and j. Moreover, for a fixed $\Delta_x \neq 0$ let $\tau(\Delta_y)$ be the function of Δ_y which represents the number of possible RK_j^i values. Then $\tau(\Delta_y)$ equals 2 for some 6 values of Δ_y and 4 for the remaining one.*

Hence we have a set of 7 possible values for α_2, which is determined by α_1. Considering this restriction, we can extract $2^{54.56}$ plaintext pairs from a structure with its differential being Eq (3).

Key Elimination. After the plaintext pairs have been generated, we encrypt them and seek the ciphertext pairs with a differential

$$(0, \beta_1, 0, \beta_2, \beta_3, \beta_4, \beta_0, \beta_5, \beta_6, \beta_7, \beta_8, \beta_9, \beta_{10}, \beta_{11}, 0, 0), \ {}^{\forall}\beta_i \neq 0. \tag{4}$$

For each ciphertext pair with differential Eq. (4), we try to eliminate the wrong (impossible) candidates for the 80-bit (sub)key vector $(\mathcal{K}_1 \| \mathcal{K}_2 \| \mathcal{K}_3)$, where

$$\begin{aligned} &\mathcal{K}_1 = (\mathrm{RK}^1_{[1,2,3,7]}, \mathrm{RK}^{23}_0), \mathcal{K}_2 = (\mathrm{RK}^1_{[0,5,6]}, \mathrm{RK}^2_{[2,4,6,7]}, \mathrm{RK}^{23}_{[2,4,5]} \mathrm{RK}^{22}_{[1,3,4]}), \\ &\mathcal{K}_3 = (\mathrm{RK}^{22}_{0,2}), \end{aligned} \tag{5}$$

using the plaintext pair of differential Eq. (3) and the ciphertext pair of differential Eq. (4). This can be done as follows. First, we guess the 20-bit $\mathcal{K}_1$ (which can take all possible values). After $\mathcal{K}_1$ is guessed, the number of each 4-bit subkey candidates in $\mathcal{K}_2$ is $(2 \cdot 6 + 4)/7 \approx 2.28$ on average from Proposition 1. Moreover, once $\mathcal{K}_1$ and $\mathcal{K}_2$ are fixed, each 4-bit subkey of $\mathcal{K}_3$ will have $(2 \cdot 6 + 4)/15 \approx 1.07$ candidates, as we have no restrictions on the input differential for Fs relating to these subkeys. From this observation, we can expect to eliminate $2^{20} \cdot (16/7)^{13} \cdot (16/15)^2 \approx 2^{35.69}$ candidates from a set of 2^{80} values for each plaintext-ciphertext pair, i.e. a guess of 80-bit $(\mathcal{K}_1 \| \mathcal{K}_2 \| \mathcal{K}_3)$ is eliminated with probability $2^{-44.31}$. The detail of the above procedure is depicted at Table 12 in Appendix D.

From this, to determine $(\mathcal{K}_1 \| \mathcal{K}_2 \| \mathcal{K}_3)$ with probability almost one, we need N ciphertext pairs, where N satisfies $2^{80}(1 - 2^{-44.31})^N \approx 1$. This implies $N \approx 2^{50.11}$. Assuming the ciphertext's randomness, we can expect a ciphertext pair of differential Eq (4) with probability $(2^{-4})^4 \cdot (2^{-1})^8 = 2^{-24}$. This implies that we basically need $2^{74.11}$ ciphertext pairs. But in fact we need some more. In the key elimination we need to compute some other subkeys (64 bits in total), which is uniquely determined by the key of Eq. (5). These keys contain RK_4^{19},RK_4^{21}, and RK_6^{23} and they can cause a contradiction with other keys. If this event occurs,

the corresponding plaintext/ciphertext pair turns out to be useless. Considering the probability of this event we need 2^{10} times more pairs, thus we eventually need $2^{84.11}$ ciphertext pairs.

Since one structure enables to produce $2^{54.56}$ plaintext pairs of the desired difference, we need to generate $2^{29.55}$ structures (by using $2^{29.55}$ distinct constants) and run the above key elimination procedure for all structures.

Details of Key Elimination. In the key elimination we combine several techniques to reduce the complexity. In particular we use the Difference Table. Its entry is indexed by $(x, x', y) \in (\{0,1\}^4)^3$ and the entry is a set $\mathcal{K} = \{k : y = S(k \oplus x) \oplus S(k \oplus x')\}$. We also use the relationships between subkeys induced from the key schedule, or we can directly guess the key if input and output pairs of the corresponding F are fixed (not only their differentials).

Complexity Estimation. For each plaintext-ciphertext pair, the procedure regarding $\mathcal{K}_2$ and $\mathcal{K}_3$ requires the 17 evaluations of F function, shown on the dotted lines in Fig.s 2 and 3, and $F^{23}_{[2,3,4,5,6]}$, total 22 functions. This amounts to $22/(23 \cdot 8)$ encryptions of 22-round TWINE. Consequently, we can attack 23-round TWINE-80 with the time complexity $2^{50.11+10} \cdot 2^{20} \cdot 22/(23 \cdot 8) = 2^{77.04}$ encryptions, and the memory complexity $2^{80}/64 = 2^{74}$ blocks.

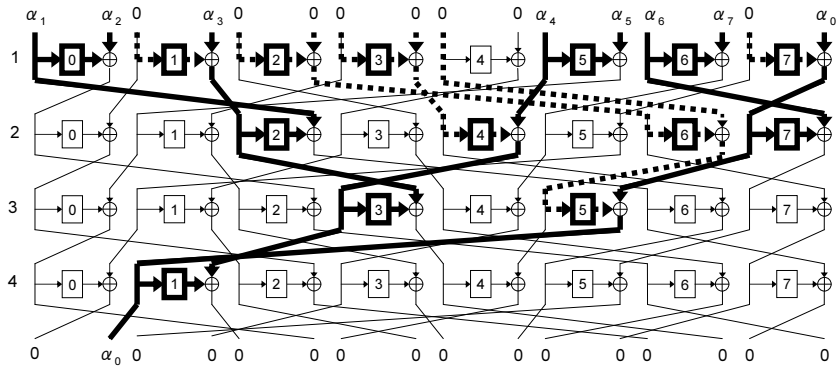

Fig. 2. The First 4 Rounds in the Impossible Differential Attack.

4.3 Saturation Attack

Saturation attack [16] is also a powerful attack against GFS-based ciphers. We consider 4-bit-wise saturations. The state consists of 2^4 variables, denoted by $S = (S_0, \ldots, S_{15})$, where S_i has the following four status (here X is the plaintext):

Constant $(C) : \forall i, j \;\; X_i = X_j$ **All** (A) $: \forall i, j \;\; i \neq j \Leftrightarrow X_i \neq X_j$

Balance (B) $: \bigoplus_i^{2^4-1} X_i = 0$ **Unknown** (U) : Others

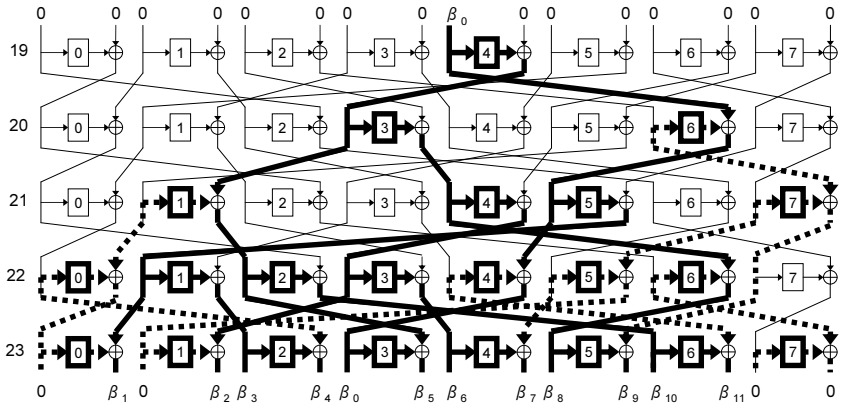

Fig. 3. The Last 5 Rounds in the Impossible Differential Attack.

Let $\alpha = (\alpha_0, \ldots, \alpha_{15})$ and $\beta = (\beta_0, \ldots, \beta_{15})$, $\alpha_i, \beta_i \in \{C, A, B, U\}$, be the initial state and the t-round state holding with probability 1. If we have $\alpha_i = A$ and $\beta_j \neq U$ for some i and j, we call $\alpha \xrightarrow{tr} \beta$ an t-round saturation characteristic (SC).

TWINE has 15-round SC with input consisting of one C and fifteen As and output consisting of 4 Bs (the remaining consists of U), for example;

$$(\bar{A}^{12}, C, \bar{A}^{3}) \xrightarrow{15r} (\bar{U}^{3}, B, \bar{U}^{5}, B, \bar{U}^{3}, B, U, B), \tag{6}$$

$$(\bar{A}^{6}, C, \bar{A}^{9}) \xrightarrow{15r} (U, B, \bar{U}^{3}, B, U, B, \bar{U}^{3}, B, \bar{U}^{4}). \tag{7}$$

Suppose we use SC of Eq. (7) to break 22-round TWINE-80. Let *S-structure* denote a set of 2^{60} plaintexts induced from Eq. (7), i.e., the input block X_6 is fixed to a constant and the other blocks take all combinations.

Our attack recovers a 72-bit subkey vector

$$\mathcal{K}_{\text{target}} = (\mathrm{RK}^{22}, \mathrm{RK}^{21}_{[0,2,3,4,5,6,7]}, \mathrm{RK}^{20}_{[6,7]}, \mathrm{RK}^{16}_{0})$$

based on the fact that (the state of) X_1^{15} is B with these 15-round SCs. Here, the state of X_1^{15} being B implies the coincidence of states between X_0^{16} and F_0^{16}, the output of the leftmost F function in the round 16, computed from the S-structure. From this, we calculate the sums of F_0^{16} and X_0^{16} independently, and choose a key that makes these values the same as a candidate for the correct key. The basic procedure for one S-structure is as follows.

1. Encrypt an S-structure and obtain 2^{60} ciphertexts. As our target is the 22-round version, the ciphertext is written as X^{22}.
2. List all ciphertexts except their rightmost (16-th) block, X_{15}^{22}. The result is denoted by List $\mathcal{L}_1$. It is merely a set of 60-bit values and the values with even appearances need not be stored. We then guess

$$\mathcal{K}_1 = (\mathrm{RK}^{22}_{[0,1,2,3,4,5,6]}, \mathrm{RK}^{21}_{[2,3,4,5,7]}, \mathrm{RK}^{20}_{6}, \mathrm{RK}^{17}_{2}, \mathrm{RK}^{16}_{0})$$

and compute the sum of outputs of F_0^{16} (the leftmost F function in the round 16) using all entries in $\mathcal{L}_1$ with each guess for $\mathcal{K}_1$[3]. We remark that a pair of an entry of $\mathcal{L}_1$ and a guess for $\mathcal{K}_1$ uniquely determines the output of F_0^{16}. The key guesses are grouped according to the sum of F_0^{16}'s outputs. Let $\mathcal{G}_1(s)$ be the key group with F_0^{16} output sum $s \in \{0,1\}^4$.

3. Count the appearance of 48-bit ciphertext subsequences, $X^{22}_{[0,2,4,5,6,7,8,9,10,11,14,15]}$, and list those have odd counts to form the list $\mathcal{L}_2$. We then guess

$$\mathcal{K}_2 = (\mathrm{RK}^{22}_{[2,3,4,5,7]}, \mathrm{RK}^{21}_{[0,5,6]}, \mathrm{RK}^{20}_{7}, \mathrm{RK}^{18}_{2})$$

and compute the sum of X_0^{16} using all entries in $\mathcal{L}_2$ with each guess for $\mathcal{K}_2$. The key guesses are grouped according to the sum of X_0^{16}. The key group with X_0^{16} sum being $s' \in \{0,1\}^4$ is denoted by $\mathcal{G}_2(s')$.

4. Extract the all "consistent" combinations from $\mathcal{G}_1(s)$ and $\mathcal{G}_2(s)$ for all $s \in \{0,1\}^4$, and output them as the set of valid key candidates for $\mathcal{K}_{\mathrm{target}}$. This can be done as follows. Let $v \in \mathcal{G}_1(0000)$ and $w \in \mathcal{G}_2(0000)$. We denote the guess for RK^i_j in v and w by $\mathrm{RK}^i_j(v)$ and $\mathrm{RK}^i_j(w)$. Both v and w contain guesses of $\mathrm{RK}^{22}_{[2,3,4,5]}$ and RK^{21}_5. If the following four equations,

$$\mathrm{RK}^{22}_{[2,3,4,5]}(v) = \mathrm{RK}^{22}_{[2,3,4,5]}(w), \quad \mathrm{RK}^{21}_5(v) = \mathrm{RK}^{21}_5(w), \tag{8}$$

$$\mathrm{RK}^{21}_6(w) = S(\mathrm{RK}^{21}_2(v)) \oplus \mathrm{RK}^{18}_2(w), \tag{9}$$

$$\mathrm{RK}^{22}_7(w) = S(S(\mathrm{RK}^{20}_7(w)) \oplus S^{-1}(\mathrm{RK}^{20}_6(v) \oplus \mathrm{RK}^{17}_2(v))) \oplus \mathrm{RK}^{21}_0(w) \tag{10}$$

hold true, a valid key candidate for $\mathcal{K}_{\mathrm{target}}$ is obtained by combining v and w. The check is done for all pairs from $\mathcal{G}_1(s) \times \mathcal{G}_2(s)$, and for all $s \in \{0,1\}^4$.

Taking Step 2 for example, we explain the detailed procedure. We first guess RK^{22}_0 and compute $X^{21}_1 (= F^{22}_0(X^{22}_0) = S(\mathrm{RK}^{22}_0 \oplus X^{22}_0))$ using $\mathcal{L}_1$ with 2^{64} F evaluations. Then we substitute $X^{22}_{[0,1]}$ (8 bits) written in $\mathcal{L}_1$ with X^{21}_1 (4 bits) and obtain a list of 56-bit sequences, and collect the values with odd appearance to form a new list, called $\mathcal{L}_{1,1}$. Next, we guess RK^{22}_2 and compute X^{21}_5 based on the guess with 2^{64} F evaluations. We then substitute $X^{22}_{[4,5]}$ with X^{21}_5 in $\mathcal{L}_{1,1}$ and obtain the list of 52-bit sequences and collect the values with odd appearance to form a new list, called List $\mathcal{L}_{1,2}$. The above procedure is repeated to gradually reduce the list size. Eventually the computation of the sum of F_0^{16} outputs requires $2^{73.80}$ F evaluations (equivalently $2^{66.34}$ encryptions of 22-round TWINE). A similar complexity reduction can also be applied to Step 3, however, the computation of Step 3 is much smaller than that of Step 2 (due to the small space for the key guess) in any case.

As checks are done w.r.t. 4-bit internal values, the above procedure with one S-structure rejects a wrong key candidate for $\mathcal{K}_{\mathrm{target}}$ with probability $1 - 2^{-4}$ (i.e. the size of candidates is reduced to 1/16), hence we basically need at least

[3] More precisely, $\mathrm{RK}^{20}_{[0,5]}$,$\mathrm{RK}^{19}_{[1,7]}$,RK^{18}_3 are required to compute F_0^{16} outputs. Also RK^{20}_1 and RK^{19}_3 are required to compute X_0^{16} in Step 3. These RKs can be computed from $\mathcal{K}_1$ or $\mathcal{K}_2$. See Appendix E.

18 S-structures to identify the right key. However, this is impossible as each sub-block is 4-bit. To elude the problem, we exploit the key schedule; the structure of the key schedule allows us to derive the 80-bit key with 2^{68} candidates for 72-bit subkey, and an exhaustive search for the remaining 8-bit subkey, and the final key check, which is trivial.

Summarizing, the attack with an S-structure requires 2^{60} plaintexts to be encrypted, and 2^{77} (which follows from $2^{66.34} + 2^{76} + \rho$, where ρ denotes the computation of Step 3, which is negligible) encryptions. The memory complexity is 2^{67} (64-bit) blocks. If we want to further reduce the complexity, using multiple S-structures (using distinct constants with the same SC) can help. The result is shown by Table 4. According to our investigation, the attack with 5 or more structures has higher time complexity than that with 4 structures, hence the best one is with 4 structures.

Table 4. Complexity of Saturation Attack.

# of Struct.s	Data	Time (Enc)	Memory (Block)
1	2^{60}	$2^{77}(\approx 2^{60} + 2^{66.34} + \rho + 2^{76} + 2^{12})$	2^{67}
2	2^{61}	$2^{73}(\approx 2^{61} + (2^{66.34} + \rho) \cdot 2 + 2^{72} + 2^{8})$	2^{67}
3	$2^{61.59}$	$2^{68.97}(\approx 2^{61.59} + (2^{66.34} + \rho) \cdot 3 + 2^{68} + 2^{4})$	2^{67}
4	2^{62}	$2^{68.43}(\approx 2^{62} + (2^{66.34} + \rho) \cdot 4 + 2^{64})$	2^{67}

4.4 Differential / Linear Cryptanalysis

To evaluate the resistance against differential cryptanalysis (DC) [5] and linear cryptanalysis (LC) [29], we need to know the number of differentially and linearly active S-boxes, denoted by AS_D and AS_L, respectively. We performed a computer-based search for differential and linear paths, and evaluated AS_D and AS_L for each round. As a result, the numbers of AS_D and AS_L are the same, as shown by Table 5. Since our S-box has 2^{-2} maximum differential and linear probabilities, the maximum differential and linear characteristic probabilities are both 2^{-64} for 15 rounds. Examples of 14-round differential (Δ) and linear (Γ) characteristics having the minimum I/O weights are as follows. They involve 30 active S-boxes, and thus the characteristic probability is 2^{-60}.

$$\begin{aligned} \Delta &= (0^9, 1, 0, 1, 0, 1, 0, 0) \overset{14r}{\rightarrow} (0^3, 1, 0^4, 1, 0, 0, 1, 0, 0, 1, 1), \\ \Gamma &= (0^6, 1, 1, 0^3, 1, 0, 0, 1, 1) \overset{14r}{\rightarrow} (0^9, 1, 0^3, 1, 0, 1). \end{aligned} \tag{11}$$

Here, 1 denotes an arbitrary non-zero difference (mask) and 0 denotes the zero difference (mask) for Δ (Γ). Compared to the impossible differential attack, we naturally expect the key recovery attacks exploiting the key schedule with these

differential/linear characteristics are less powerful, since 14-round impossible differential characteristic has much fewer (only 2) weights, and fewer weights imply the more attackable rounds in the key guessing. We remark that a computer-based search for the maximum differential probability (rather than the characteristic probability) of GFS was proposed by [30]. However, applying their algorithm to our 16-block case seems infeasible due to the computational complexity.

Table 5. Number of Differentially and Linearly Active S-boxes.

Round	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
AS_D, AS_L	0	1	2	3	4	6	8	11	14	18	22	24	27	30	32

4.5 Key Schedule-based Attacks

Related-Key Differential Attacks. The related-key attack, proposed by Biham [4], is an attack applicable to the environment where the adversary can somehow modify the key input. We focus on the typical setting, that is, the adversary is allowed to insert a key differential. In order to evaluate the resistance of TWINE against the related-key attack, we implemented the search method proposed by Biryukov et al. [8], which counts the number of active S-boxes for combined data processing and key schedule parts. See [8] for the detail of the algorithm. We (naturally) searched 4-bit truncated differential paths. As S-box has maximum differential probability being 2^{-2}, we needed 40 (64) active S-boxes for TWINE-80 (TWINE-128).

Due to the computational constraint the full-search is only feasible for TWINE-80. As a result, we confirmed that the number of active S-boxes reaches 40 for the 22-round. Appendix F provides the number of active S-boxes and the corresponding truncated differential paths.

Other Attacks. For the slide attack [7], the key schedule of TWINE inserts distinct constants for each round. This is a typical way to thwart slide attacks and hence we consider TWINE is immune to the slide attack.

For Meet-In-The-Middle (MITM) attack, we confirmed that the round keys for the first 3 (5) rounds contain all key bits for the 80-bit (128-bit) key case. Thus, we consider it is difficult to mount the basic MITM attack against the full-round TWINE. Note that the recently-proposed MITM variant, called biclique attack [11], may work even when all key bits are used in the relatively small number of rounds. The evaluation of such attack against TWINE is a future topic.

The result of our security evaluation for TWINE is summarized at Table 6.

Table 6. Summary of Attacks on TWINE.

Key (bits)	Attack	Rounds	Data (blocks)	Time (encryption)	Memory (blocks)
80	Impossible Diff.	23	$2^{61.39}$	$2^{76.88}$	2^{74}
	Saturation	22	2^{62}	$2^{68.43}$	2^{67}
128	Impossible Diff.	24	$2^{52.21}$	$2^{115.10}$	2^{118}
	Saturation	23	$2^{62.81}$	$2^{106.14}$	2^{103}

5 Implementation

5.1 Hardware

We implemented TWINE on ASIC using a 90*nm* standard cell library with logic synthesis done by *Synopsys Design Compiler Version D-2010.03-SP1-1*. Following the recent trend in the lightweight implementations [9, 13], the figures are shown for the case when Scan Flip-Flops (FFs) are used. In our library, a D-FF and 2-to-1 MUX cost 5.5 GE and 2.25 GE, and a Scan FF costs 6.75 GE: hence this technique saves 1.0 GE per 1-bit storage.

The data path of TWINE-80 encryption circuit is in Fig. 4, and the implementation result is shown by Table 7. We also show the detail of TWINE-80 encryption implementation in Table 8. The figures must be taken with cares, because they are related to the type of memory unit (FF), technology, library, etc, as pointed out by [13]. Even single XOR gate has several grades, from fast-but-large and slow-but-small. As suggested by [13], we list Gates/Memory Bit in the table, which denotes the size (in Gate Equivalent (GE)) of 1-bit memory device used for the key and states.

We did not perform a thorough logic minimization of the S-box circuit, which currently costs 30 GEs. The S-box logic minimization can further reduce the size.

We also tried a serialized implementation. Though it is not yet finished, the preliminary result indicates that encryption-only TWINE-80 can be implemented 1,116 GEs. The details of our serialized implementation will be given in a forthcoming paper.

5.2 Software

To evaluate the performance on embedded software, we implement TWINE on Atmel AVR 8-bit Micro-controller. The target device is ATmega163, which has 16K bytes Flash, 512 bytes EEPROM and 1024 bytes SRAM. We implemented the four versions: speed-first, ROM-first (minimizing the consumption), and RAM-first, and the double-block implementation, where two message blocks are processed in parallel. Such an implementation works for parellelizable modes of operations, e.g., the counter mode and PMAC. All implementations include the precomputation of round keys, i.e. they do not use an on-the-fly key schedule.

Table 7. ASIC Implementation Results. For some implementations, the figures of Throughput and Cycles/Block is an estimated value.

Algorithm	Function	Block size (bit)	Key size (bit)	Cycles/ block	Throughput (Kbps@100KHz)	Area (GE†)	Gates / Memory bit	Type
TWINE	Enc	64	80	36	178	1503	6.75	round
TWINE	Enc+Dec	64	80	36	178	1799	6.75	round
TWINE	Enc	64	128	36	178	1866	6.75	round
TWINE	Enc+Dec	64	128	36	178	2285	6.75	round
TWINE	Enc	64	80	540	11.8	1116	6.75	serial
PRESENT [36]	Enc	64	80	547	11.4	1000	n/a	serial
PRESENT [9]	Enc	64	80	32	200	1570	6	round
AES [31]	Enc	128	128	226	57	2400	6	serial
mCRYPTON [27]	Enc	64	64	13	492.3	2420	5	round
SEA [28]	Enc+Dec	96	96	93	103	3758	n/a	round
HIGHT [21]	Enc+Dec	64	128	34	188.25	3048	n/a	round
KLEIN [19]	Enc	64	80	17	376.4	2629	n/a	round
KLEIN [19]	Enc	64	80	271	23.6	1478	n/a	serial
DES [26]	Enc	64	56	144	44.4	2309	12.19	serial
DESL [26]	Enc	64	56	144	44.4	1848	12.19	serial
KATAN [13]	Enc	64	80	254	25.1	1054	6.25	serial
Piccolo [40]	Enc	64	80	27	237	1496¶	6.25	round
Piccolo [40]	Enc+Dec	64	80	27	237	1634¶	6.25	round
Piccolo [40]	Enc	64	80	432	14.8	1043¶	6.25	serial
Piccolo [40]	Enc+Dec	64	80	432	14.8	1103¶	6.25	serial
LED [18]	Enc	64	80	1872	3.4	1040	6/4.67◇	serial
PRINTcipher [25]	Enc	48	80	48	12.5	503⋆	n/a	round

† Gate Equivalent : cell area/2-input NAND gate size (2.82).
¶ Includes a key register that costs 360 GEs; Piccolo can be implemented without a key register if key signal holds while encryption.
◇ Mixed usage of two memory units.
⋆ Hardwired key.

Table 8. TWINE-80 Encryption Hardware Implementation.

Data Processing Part (GE)		Key Scheduling Part (GE)	
Data Register	432	Key Register	540
S-box	240	round const comp.	2
Round Key XOR	64	round const XOR	12
S-box out XOR	64	S-box	60
		S-box out XOR	16
		RC register	33
		State register	6
		Others/Control	34
		Total	1503

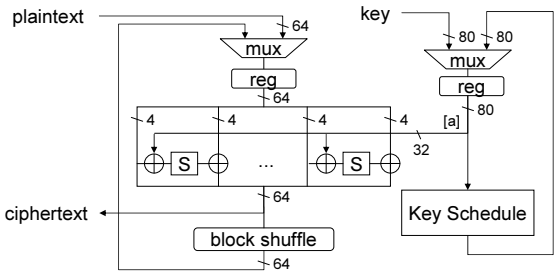

Fig. 4. Data path of TWINE-80 encryption, when Scan FF is not used (i.e. the case with MUX and D-FF). The bit boundary $[a]$ indicates certain 32 bits of 80-bit key state specified from the key schedule.

In the speed-first implementation, two rounds are processed in one loop. This removes the block shuffle between the first and second rounds. A further speeding up is possible if more rounds are contained in one loop at the cost of increased memory. Our program keeps all 4-bit blocks in the distinct registers. RAM load instruction (LD) is faster than ROM load instruction (LPM), hence the S-box and the constants are stored at RAM. The data arrangement is carefully considered to avoid carry in the address computation. The double-block implementation stores the S-boxes in ROM.

Our result is in Table. 9, and a comparison is in Table 10. In Table 10 we list the (scaled) throughput/code ratio for a performance measure (See Table 10 for the formula), following [35]. AES's performance is still quite impressive, however, one can also observe a good performance of TWINE.

One might be interested in the performance of TWINE under 32/64-bit CPU. We are currently working on this, in particular using the vector permutation instructions, which was shown to be very powerful for AES [20].

Table 9. Software Implementation of TWINE on ATmega163.

Target	Key schedule (cycles)	Encryption (cycles/block)	Decryption (cycles/block)	ROM (bytes)	RAM (bytes)
Speed-first	2,170	2,165	2,166	1,304	414
ROM-first	12,022	18,794	18,689	728	335
RAM-first	12,058	18,794	18,688	792	191
Double-block	1,887	1,301	1,302	2,294	386

Table 10. Comparison of Software Implementation on AVR.

Algorithm	Key (bits)	Block (bits)	Language	ROM (bytes)	RAM (bytes)	Enc (cyc/byte)	Dec (cyc/byte)	ETput /Code†	DTput /Code‡
TWINE	80	64	asm	1,304	414	271	271	2.14	2.14
PRESENT [33]	80	64	asm	2,398	528	1,199	1,228	0.28	0.28
PRESENT [17]	80	64	N/A	936	0	1,340	1,405	0.80	0.76
DES [17]	56	64	N/A	4,314	0	1,079	1,019	0.21	0.22
DESXL [17]	184	64	N/A	3,192	0	1,066	995	0.29	0.31
HIGHT [17]	128	64	N/A	5,672	0	371	371	0.48	0.48
IDEA [17]	128	64	N/A	596	0	338	1,924	4.97	0.87
TEA [17]	128	64	N/A	1,140	0	784	784	1.11	1.11
SEA [17]	96	96	N/A	2,132	0	805	805	0.58	0.58
AES [12]	128	128	asm	1,912	432	125	181	3.42	2.35

† Encryption Throughput per Code: $(1/\text{Enc})/(\text{ROM} + \text{RAM})$ (scaled by 10^6)
‡ Decryption Throughput per Code: $(1/\text{Dec})/(\text{ROM} + \text{RAM})$ (scaled by 10^6)

6 Conclusions

We have presented a lightweight block cipher TWINE, which has 64-bit block and 80 or 128-bit key. It is primary designed to fit extremely-small hardware, yet provides a notable performance under embedded software. This characteristic mainly originates from the Type-2 generalized Feistel with a highly-diffusive block shuffle. We performed a thorough security analysis, in particular for the impossible differential and saturation attacks. Although the result implies the sufficient security of full-round TWINE, its security naturally needs to be studied further.

Acknowledgments. The authors would like to thank the anonymous reviewers for many useful comments. We thank Maki Shigeri, Etsuko Tsujihara, and Teruo Saito for discussions, and Daisuke Ikemura for investigation on hardware-related issues.

References

1. Final report of European project IST-1999-12324, New European Schemes for Signatures, Integrity, and Encryption. 2004.
2. `http://www.lightweightcrypto.org/implementations.php`
3. `http://cis.sjtu.edu.cn/index.php /Software_Implementation_of_Block_Cipher_PRESENT_for_8-Bit_Platforms`
4. E. Biham. "New Types of Cryptanalytic Attacks Using Related Keys." Journal of Cryptology, vol. 7, no. 4, pp. 229-246, 1994.
5. E. Biham and A. Shamir. "Differential Cryptanalysis of the Data Encryption Standard." Springer-Verlag, 1993.

6. E. Biham, A. Biryukov and A. Shamir, "Cryptanalysis of skipjack reduced to 31 rounds using impossbile differentials." *Advances in Cryptology* - EUROCRYPT '99, LNCS 1592, pp.12-23, 1999.
7. A. Biryukov and D. Wagner. "Slide Attacks." FSE'99, LNCS 1636, pp.245-259.
8. A. Biryukov and I. Nikolič. "Automatic Search for Related-Key Differential Characteristics in Byte-Oriented Block Ciphers: Application to AES, Camellia, Khazad, and Others." EUROCRYPT 2010.
9. A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M.J.B. Robshaw, Y. Seurin and C. Vikkelsoe, "PRESENT: An Ultra-Lightweight Block Cipher", CHES 2007, LNCS 4727, pp. 450-466, 2007.
10. A. Bogdanov, L.R. Knudsen, G. Leander, C. Paar, A. Poschmann, M.J.B. Robshaw, Y. Seurin, and C. Vikkelsoe. "Small-Footprint Block Cipher Design -How far can you go?" 3rd Conference on RFID Security, 2007.
11. A. Bogdanov, D. Khovratovich and C. Rechberger. "Biclique Cryptanalysis of the Full AES." cryptology eprint archive 2011/449.
12. J. W. Bos, D. A. Osvik, D. Stefan. "Fast Implementations of AES on Various Platforms." SPEED-CC – Software Performance Enhancement for Encryption and Decryption and Cryptographic Compilers, 2009.
13. C. D. Canniere, O. Dunkelman and M. Knezevic. "KATAN and KTANTAN - A Family of Small and Efficient Hardware-Oriented Block Ciphers." CHES '09, pp. 272-288, 2009.
14. D. Canright. "A Very Compact S-Box for AES." *Cryptographic Hardware and Embedded Systems*- CHES'05, LNCS 3659, pp. 441-455, 2005.
15. J. Chen, K. Jia, H. Yu, and X. Wang. "New Impossible Differential Attacks of Reduced-Round Camellia-192 and Camellia-256." ACISP 2011, LNCS 6812, pp.16-33, 2011
16. J. Daemen, L. R. Knudsen, and V. Rijmen, "The Block Cipher SQUARE." *Fast Software Encryption*-FSE'97, LNCS 1267, pp.149-165, 1997.
17. T. Eisenbarth, S. Kumar, C. Paar, A. Poschmann, and L. Uhsadel. "A Survey of Lightweight Cryptography Implementations." IEEE Design & Test of Computers – Special Issue on Secure ICs for Secure Embedded Computing, 24(6):522-533, November/December 2007.
18. J. Guo, T. Peyrin, A. Poschmann, M. J. B. Robshaw. "The LED Block Cipher.", CHES'10, LNCS 6225, pp. 326-341.
19. Z. Gong, S. Nikova and Y.-W. Law. "KLEIN: A New Family of Lightweight Block Ciphers." RFIDsec 2011.
20. M. Hamburg. "Accelerating AES with Vector Permute Instructions." CHES 2009, LNCS 5747, pp. 18-32.
21. D. Hong, J. Sung, S. Hong, J. Lim, S. Lee, B. Koo, C. Lee, D. Chang, J. Lee, K. Jeong, H. Kim, J. Kim and S. Chee, "HIGHT: A New Block Cipher Suitable for Low-Resource Device." CHES 2006, LNCS 4249, pp. 46-59, 2006.
22. M. Izadi, B. Sadeghiyan, S. S. Sadeghian, and H. A. Khanooki. "MIBS: A New Lightweight Block Cipher." CANS'09, LNCS 5888, pp. 334-348, 2009.
23. A. Juels and S. A. Weis. "Authenticating Pervasive Devices with Human Protocols." CRYPTO'05, LNCS 3126, pp. 293-198, 2005.
24. J. Kim, S. Hong, J. Sung, C. Lee, S. Lee, "Impossible Differential Cryptanalysis for Block Cipher Structures." INDOCRYPT 2003, LNCS 2904, pp.82-96, 2003.
25. L. R. Knudsen, G. Leander, A. Poschmann and M. J. B. Robshaw. "PRINTcipher: A Block Cipher for IC-Printing." CHES'10, LNCS 6225, pp. 16-32.
26. G. Leander, C. Paar, A. Poschmann, and K. Schramm. "New Lightweight DES Variants." *Fast Software Encryption*-FSE'07, LNCS 4593, pp. 196-210, 2007.

27. C. H. Lim and T. Korkishko. "mCrypton - A Lightweight Block Cipher for Security of Low-Cost RFID Tags and Sensors". *Information Security Applications*-WISA'05, LNCS 3786, pp. 243-258, 2005.
28. F. Mace, F.-X. Standaert, and J.-J. Quisquater. "Implementations of the Block Cipher SEA for Constrained Applications." Proceedings of the Third International Conference on RFID Security, RFIDSec 2007, pp.103-114.
29. M. Matsui, "Linear cryptanalysis of the data encryption standard." EUROCRYPT'93, LNCS 765, pp.386-397, Springer-Verlag, 1994.
30. K. Minematsu, T. Suzaki, and M. Shigeri. "On Maximum Differential Probability of Generalized Feistel." ACISP 2011, LNCS 6812, pp. 89-105, 2011.
31. A. Moradi, A. Poschmann, S. Ling, C. Paar, and H. Wang. "Pushing the Limits: A Very Compact and a Threshold Implementation of AES." Eurocrypt 2011, LNCS 6632, pp. 69-88.
32. O. Özen, K. Varici, C. Tezcan, Ç. Kocair. "Lightweight Block Ciphers Revisited: Cryptanalysis of Reduced Round PRESENT and HIGHT." ACISP 2009, LNCS 5594, pp. 90-107, 2009.
33. A. Poschmann. "Lightweight Cryptography - Cryptographic Engineering for a Pervasive World." Cryptology ePrint Archive, Report 2009/516, 2009.
34. A. Poschmann, S. Ling, and H. Wang. "256 Bit Standardized Crypto for 650 GE - GOST Revisited." CHES'10, LNCS 6225, pp. 219-233.
35. S. Rinne, T. Eisenbarth, and C. Paar. "Performance Analysis of Contemporary Lightweight Block Ciphers on 8-bit Microcontrollers." SPEED – Software Performance Enhancement for Encryption and Decryption, 2007.
36. C. Rolfes, A. Poschmann, G. Leander, and C. Paar. "Ultra-Lightweight Implementations for Smart Devices - Security for 1000 Gate Equivalents." Smart Card Research and Advanced Application Conference (CARDIS 2008), LNCS 5189, pp. 89-103, 2008.
37. Akashi Satoh, Sumio Morioka, Kohji Takano and Seiji Munetoh, "A Compact Rijndael Hardware Architecture with S-Box Optimization." ASIACRYPT2001, LNCS Vol.2248, pp.239-254, Dec 2001.
38. T. Shirai, K. Shibutani, T. Akishita, S. Moriai and T. Iwata, "The 128-bit Blockcipher CLEFIA." *Fast Software Encryption*-FSE'07, LNCS 4593, pp. 181-195, 2007.
39. K. Shibutani. "On the Diffusion of Generalized Feistel Structures Regarding Differential and Linear Cryptanalysis." SAC'10, LNCS 6544, pp. 211-228, 2011.
40. K. Shibutani, T. Isobe, H. Hiwatari, A. Mitsuda, T. Akishita, and T. Shirai. "Piccolo: An Ultra-Lightweight Blockcipher." CHES 2011, LNCS 6917, pp. 342-357, 2011.
41. T. Suzaki and K. Minematsu. "Improving the Generalized Feistel." *Fast Software Encryption*-FSE'10, LNCS 6147, pp.19-39, 2010.
42. Y. Tsunoo, E. Tsujihara, M. Shigeri, T. Saito, T. Suzaki, and H. Kubo. "Impossible Differential Cryptanalysis of CLEFIA." FSE 2008. LNCS 5086, pp.398-411, 2008.
43. W. Wu and L. Zhang. "LBlock: A Lightweight Block Cipher. " 9th International Conference on Applied Cryptography and Network Security, ACNS '11, LNCS 6715, pp. 327-344.
44. Y. Zheng, T. Matsumoto, and H. Imai. "On the Construction of Block Ciphers Provably Secure and Not Relying on Any Unproved Hypotheses." *Advances in Cryptology* - CRYPTO '89, LNCS 435, pp. 461-480, 1989.

A Key Schedule for 128-bit Key

Algorithm A.1: TWINE.KeySchedule-128($K_{(128)}, RK_{(32\times 36)}$)

$\mathrm{WK}_{(128)} \leftarrow K$
$\mathrm{WK}_{0(16)} \| \mathrm{WK}_{1(16)} \| \ldots \| \mathrm{WK}_{6(16)} \| \mathrm{WK}_{7(16)} \leftarrow WK$
$\mathrm{RK}^1_{0(4)} \leftarrow \mathrm{WK}_2,\ \mathrm{RK}^1_{1(4)} \leftarrow \mathrm{WK}_3,\ \mathrm{RK}^1_{2(4)} \leftarrow \mathrm{WK}_{12},\ \mathrm{RK}^1_{3(4)} \leftarrow \mathrm{WK}_{15}$
$\mathrm{RK}^1_{4(4)} \leftarrow \mathrm{WK}_{17},\ \mathrm{RK}^1_{5(4)} \leftarrow \mathrm{WK}_{18},\ \mathrm{RK}^1_{6(4)} \leftarrow \mathrm{WK}_{28},\ \mathrm{RK}^1_{7(4)} \leftarrow \mathrm{WK}_{31}$
$\mathrm{RK}^1_{(32)} \leftarrow \mathrm{RK}^1_0 \| \mathrm{RK}^1_1 \| \ldots \| \mathrm{RK}^1_6 \| \mathrm{RK}^1_7$
for $i \leftarrow 2$ **to** 36
do
- $\mathrm{WK}_1 \leftarrow \mathrm{WK}_1 \oplus S(\mathrm{WK}_0)$
- $\mathrm{WK}_4 \leftarrow \mathrm{WK}_4 \oplus S(\mathrm{WK}_{16})$
- $\mathrm{WK}_{23} \leftarrow \mathrm{WK}_{23} \oplus S(\mathrm{WK}_{30})$
- $\mathrm{WK}_7 \leftarrow \mathrm{WK}_7 \oplus 0 \| CON^{i-1}_H$
- $\mathrm{WK}_{19} \leftarrow \mathrm{WK}_{19} \oplus 0 \| CON^{i-1}_L$
- $\mathrm{tmp}_0 \leftarrow \mathrm{WK}_0,\ \mathrm{tmp}_1 \leftarrow \mathrm{WK}_1,\ \mathrm{tmp}_2 \leftarrow \mathrm{WK}_2,\ \mathrm{tmp}_3 \leftarrow \mathrm{WK}_3$
- **for** $j \leftarrow 0$ **to** 6
  do
  - $\mathrm{WK}_{j*4} \leftarrow \mathrm{WK}_{j*4+4},\quad \mathrm{WK}_{j*4+1} \leftarrow \mathrm{WK}_{j*4+5}$
  - $\mathrm{WK}_{j*4+2} \leftarrow \mathrm{WK}_{j*4+6},\ \mathrm{WK}_{j*4+3} \leftarrow \mathrm{WK}_{j*4+7}$
- $\mathrm{WK}_{28} \leftarrow \mathrm{tmp}_1, \mathrm{WK}_{29} \leftarrow \mathrm{tmp}_2, \mathrm{WK}_{30} \leftarrow \mathrm{tmp}_3, \mathrm{WK}_{31} \leftarrow \mathrm{tmp}_0$
- $\mathrm{RK}^i_0 \leftarrow \mathrm{WK}_2,\ \mathrm{RK}^i_1 \leftarrow \mathrm{WK}_3,\ \mathrm{RK}^i_2 \leftarrow \mathrm{WK}_{12},\ \mathrm{RK}^i_3 \leftarrow \mathrm{WK}_{15}$
- $\mathrm{RK}^i_4 \leftarrow \mathrm{WK}_{17},\ \mathrm{RK}^i_5 \leftarrow \mathrm{WK}_{18},\ \mathrm{RK}^i_6 \leftarrow \mathrm{WK}_{28},\ \mathrm{RK}^i_7 \leftarrow \mathrm{WK}_{31}$
- $\mathrm{RK}^i_{(32)} \leftarrow \mathrm{RK}^i_{0(4)} \| \mathrm{RK}^i_{1(4)} \| \ldots \| \mathrm{RK}^i_{6(4)} \| \mathrm{RK}^i_{7(4)}$

$RK_{(32\times 36)} \leftarrow \mathrm{RK}^1_{(32)} \| \mathrm{RK}^2_{(32)} \| \ldots \| \mathrm{RK}^{35}_{(32)} \| \mathrm{RK}^{36}_{(32)}$

B Test Vectors

Table 11. Test Vectors in the Hexadecimal Notation.

key length	80-bit	128-bit
key	`00112233 44556677 8899`	`00112233 44556677 8899AABB CCDDEEFF`
plaintext	`01234567 89ABCDEF`	`01234567 89ABCDEF`
ciphertext	`7C1F0F80 B1DF9C28`	`979FF9B3 79B5A9B8`

C 80-bit Key Schedule

Figure 5 depicts the key schedule for TWINE-80.

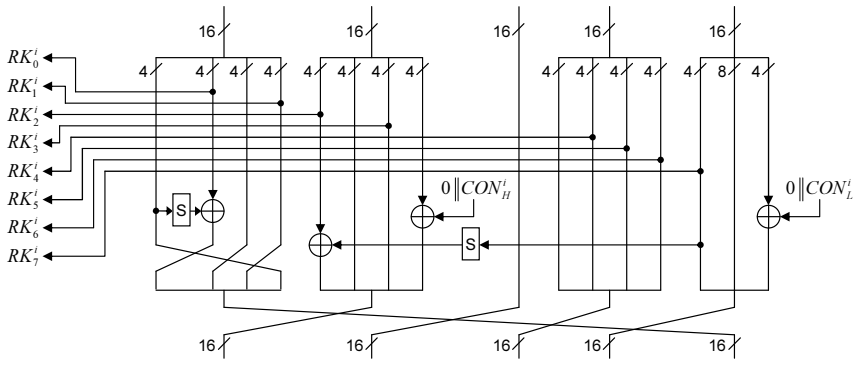

Fig. 5. 80-bit Key Schedule.

D Supplementary Information for Impossible Differential Cryptanalysis

The following is the subkey relationships used for the impossible differential cryptanalysis.

$$\mathrm{RK}_3^3 = \mathrm{RK}_5^1$$
$$\mathrm{RK}_5^3 = \mathrm{RK}_1^1$$
$$\mathrm{RK}_1^4 = \mathrm{RK}_6^1$$
$$\mathrm{RK}_0^{21} = S[S[S[\mathrm{RK}_4^{19}] \oplus \mathrm{RK}_3^1] \oplus \mathrm{RK}_4^{21}] \oplus \mathrm{RK}_7^2$$
$$\mathrm{RK}_5^{21} = S^{-1}[S^{-1}[\mathrm{RK}_1^{21} \oplus S[\mathrm{RK}_3^{20}] \oplus \mathrm{RK}_1^1] \oplus \mathrm{RK}_6^1]$$
$$\mathrm{RK}_0^{22} = \mathrm{RK}_4^{19}$$
$$\mathrm{RK}_2^{22} = S[\mathrm{RK}_6^{20}] \oplus S[\mathrm{RK}_7^2] \oplus \mathrm{RK}_2^2$$
$$\mathrm{RK}_4^{22} = S[S[S[\mathrm{RK}_3^{20}] \oplus \mathrm{RK}_1^1] \oplus \mathrm{RK}_3^{22}] \oplus S[S[S[S[\mathrm{RK}_6^{20}] \oplus S[\mathrm{RK}_7^2] \oplus \mathrm{RK}_2^2] \oplus S^{-1}[\mathrm{RK}_1^{22} \oplus S^{-1}[S^{-1}[S^{-1}[\mathrm{RK}_1^{21} \oplus S[\mathrm{RK}_3^{20}] \oplus \mathrm{RK}_1^1] \oplus \mathrm{RK}_6^1] \oplus S[\mathrm{RK}_4^{21}] \oplus \mathrm{RK}_5^1]]] \oplus \mathrm{RK}_7^{21}] \oplus \mathrm{RK}_4^2$$
$$\mathrm{RK}_5^{22} = S^{-1}[S^{-1}[\mathrm{RK}_1^{22} \oplus S^{-1}[S^{-1}[S^{-1}[\mathrm{RK}_1^{21} \oplus S[\mathrm{RK}_3^{20}] \oplus \mathrm{RK}_1^1] \oplus \mathrm{RK}_6^1] \oplus S[\mathrm{RK}_4^{21}] \oplus \mathrm{RK}_5^1]] \oplus \mathrm{RK}_6^2]$$
$$\mathrm{RK}_6^{22} = S[S[\mathrm{RK}_6^{20}] \oplus S[\mathrm{RK}_7^2] \oplus \mathrm{RK}_2^2] \oplus S^{-1}[\mathrm{RK}_1^{22} \oplus S^{-1}[S^{-1}[S^{-1}[\mathrm{RK}_1^{21} \oplus S[\mathrm{RK}_3^{20}] \oplus \mathrm{RK}_1^1] \oplus \mathrm{RK}_6^1] \oplus S[\mathrm{RK}_4^{21}] \oplus \mathrm{RK}_5^1]]$$
$$\mathrm{RK}_0^{23} = S[S[S[\mathrm{RK}_4^{21}] \oplus \mathrm{RK}_5^1] \oplus \mathrm{RK}_3^{20}] \oplus S^{-1}[S[S[S[\mathrm{RK}_6^{20}] \oplus S[\mathrm{RK}_7^2] \oplus \mathrm{RK}_2^2] \oplus S^{-1}[\mathrm{RK}_1^{22} \oplus S^{-1}[S^{-1}[S^{-1}[\mathrm{RK}_1^{21} \oplus S[\mathrm{RK}_3^{20}] \oplus \mathrm{RK}_1^1] \oplus \mathrm{RK}_6^1] \oplus S[\mathrm{RK}_4^{21}] \oplus \mathrm{RK}_5^1]]] \oplus \mathrm{RK}_7^{21} \oplus \mathrm{RK}_7^1]$$
$$\mathrm{RK}_1^{23} = \mathrm{RK}_6^{20}$$
$$\mathrm{RK}_3^{23} = S^{-1}[S^{-1}[\mathrm{RK}_1^{21} \oplus S[\mathrm{RK}_3^{20}] \oplus \mathrm{RK}_1^1] \oplus \mathrm{RK}_6^1]$$
$$\mathrm{RK}_4^{23} = \mathrm{RK}_3^{20}$$
$$\mathrm{RK}_5^{23} = \mathrm{RK}_1^{21}$$
$$\mathrm{RK}_6^{23} = S[\mathrm{RK}_2^{23}] \oplus S[\mathrm{RK}_1^{21}] \oplus S^{-1}[\mathrm{RK}_7^2 \oplus \mathrm{RK}_0^1]$$
$$\mathrm{RK}_7^{23} = S[S[\mathrm{RK}_7^{21}] \oplus S[\mathrm{RK}_1^{22}] \oplus S[\mathrm{RK}_7^1] \oplus \mathrm{RK}_2^1] \oplus \mathrm{RK}_4^{19}$$

Table 12. Procedure of Round Key Determination.

Step	target key	method	
1	$\mathrm{RK}_0^1, \mathrm{RK}_5^1, \mathrm{RK}_6^1$	diff-table	
2	$\mathrm{RK}_4^3, \mathrm{RK}_1^4$	key rel.	Input of F_3^3 and output of F_4^2 are decided from diff-table. Input of F_1^4 is decided from diff-table.
3	RK_3^1	guess	
4	RK_4^2	IO value	
5	RK_7^1	guess	Input of F_7^2 is decided.
6	RK_7^2	diff-table	
7	RK_1^1	guess	Input of F_2^2 is decided.
8	RK_5^3	key rel.	Output of F_6^2 is decided.
9	RK_2^2	diff-table	
10	RK_2^1	guess	
11	RK_6^2	IO value	
12	$\mathrm{RK}_2^{23}, \mathrm{RK}_4^{23}, \mathrm{RK}_5^{23}$	diff-table	IO value of F_0^{22} is decided.
13	$\mathrm{RK}_3^{20}, \mathrm{RK}_1^{21}$	key rel.	Input of F_3^{20} is decided from diff-table.
14	$\mathrm{RK}_5^{21}, \mathrm{RK}_3^{23}, \mathrm{RK}_6^{23}$	key rel.	Input of F_2^{22} is decided. IO value of F_0^{22} is decided. Input of F_5^{21} is decided, then output of F_4^{22} is decided.
15	RK_2^{22}	diff-table	
16	RK_6^{20}	key rel.	
17	RK_1^{23}	key rel.	IO value of F_3^{22} is decided.
18	RK_3^{22}	diff-table	
19	RK_0^{22}	IO value	
20	RK_4^{19}	key rel.	
21	RK_4^{22}	IO value	
22	RK_0^{23}	guess	IO value of F_1^{22} is decided.
23	RK_1^{22}	diff-table	
24	$\mathrm{RK}_4^{21}, \mathrm{RK}_7^{21}$	key rel.	
25	$\mathrm{RK}_5^{22}, \mathrm{RK}_6^{22}, \mathrm{RK}_7^{23}$	key rel.	

E Subkey Relationships used for Saturation Attack

The following is the subkey relationships used for the saturation attack.

$$
\begin{aligned}
\mathrm{RK}_4^{21} &= \mathrm{RK}_3^{18} \\
\mathrm{RK}_5^{21} &= \mathrm{RK}_1^{19} \\
\mathrm{RK}_6^{21} &= S[\mathrm{RK}_2^{21}] \oplus \mathrm{RK}_2^{18} \\
\mathrm{RK}_2^{22} &= \mathrm{RK}_7^{19} \\
\mathrm{RK}_3^{22} &= \mathrm{RK}_5^{20} \\
\mathrm{RK}_4^{22} &= \mathrm{RK}_3^{19} \\
\mathrm{RK}_5^{22} &= \mathrm{RK}_1^{20} \\
\mathrm{RK}_6^{22} &= S^{-1}[\mathrm{RK}_7^{21} \oplus \mathrm{RK}_0^{20}] \\
\mathrm{RK}_7^{22} &= S[S[\mathrm{RK}_7^{20}] \oplus S^{-1}[\mathrm{RK}_6^{20} \oplus \mathrm{RK}_2^{17}]] \oplus \mathrm{RK}_0^{21}
\end{aligned}
$$

F Related-key Truncated Differential and Its Active S-box Numbers

Table 13 shows the number of active S-boxes using related-key differential, where ΔKS, ΔRK, ΔX and AS denote key state difference, subkey difference, data difference, and the number of active S-boxes.

Table 13. Truncated Differential and Its Active S-box Numbers.

Round	ΔKS	ΔRK	ΔX	AS	Round	ΔKS	ΔRK	ΔX	AS
1	4D010	A2	A255	0	12	60402	80	A0E2	22
2	D8108	E1	6931	6	13	0402C	05	A630	27
3	010C3	08	9896	8	14	C02C0	88	8D39	30
4	10C30	46	4462	9	15	02C01	10	5A2E	33
5	0C302	20	2288	10	16	2C010	22	62C3	35
6	C3020	94	9411	11	17	C0104	80	8191	38
7	30201	40	0968	14	18	01041	08	0824	38
8	02016	12	1306	15	19	10410	42	4202	39
9	20160	0C	4545	19	20	04102	00	0081	39
10	01604	00	108C	20	21	41020	84	8100	41
11	16040	58	D840	21	22	10208	41	4124	41