TWINE: A Lightweight Block Cipher for Multiple Platforms*

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

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

Abstract. This paper presents a 64-bit lightweight block cipher TWINE supporting 80 and 128-bit keys. TWINE realizes quite small hardware implementation similar to the previous lightweight block cipher proposals, yet enables efficient software implementations on various platforms, from micro-controller to high-end CPU. This characteristic is obtained by the use of generalized Feistel structure combined with an improved block shuffle, introduced at FSE 2010.
Keywords: lightweight block cipher, generalized Feistel structure, block shuffle

1 Introduction

Motivation. 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 AES has been widely deployed and we can see constant efforts for small-footprint AES, such as [13, 30, 39], AES is often inappropriate for such small devices due to their size/power/memory constraints. To fill the gap, many lightweight block ciphers have been recently proposed, e.g., [8,12,17,18,20,22,23,25,40,45] and more, mainly focusing on hardware.

In this paper, we propose TWINE, a new lightweight 64-bit block cipher. Our primary goal is to achieve hardware efficiency equivalent to previous proposals, and at the same time good software performance on various CPUs, from low-end micro-controllers to high-end ones (such as Intel Core series). For this purpose, we avoid hardware-oriented design options, most notably bit permutation, and build a block cipher using 4-bit components.

Design. Specifically, we employ Type-2 generalized Feistel structure [46], GFS for short, with 16 nibble blocks. The drawback of such design is a poor diffusion property, resulting in a small-but-slow cipher due to a large number of rounds. To overcome the problem, we employ the idea of Suzaki and Minematsu at FSE 2010 [42]. They showed that we can substantially improves the diffusion property of Type-2 GFS by using a different block shuffle from the original cyclic shift. As a result, TWINE is also efficient on software and enables compact unification of encryption and decryption. TWINE uses neither a bit permutation nor a Galois-Field matrix. It consists of 4-bit S-box, XOR, and 4-bit-wise permutation (shuffle). Thus the design is quite minimalistic. A predecessor called LBlock [45] has some resemblances to ours, however TWINE is an independent work and TWINE has several concrete design advantages over LBlock (See Section 3).

Implementation. We implemented TWINE on hardware and software. Our hardware implementation results show that the encryption-only TWINE can be implemented with $1,503$ Gate Equivalent (GE), and a serialized implementation using a shared S-box architecture requires only $1,011$ GEs. For both cases, we did not consider the hard-wired key or special key signaling (as employed by [40]) hence the keys can be arbitrarily set at the point of use. The results of our hardware implementations are comparable to, or even better than, the leading hardware-oriented proposals, in particular when a standard key treatment is required.

* A preliminary version of this paper appears in the proceedings of SAC 2012 [43]. This is the full version.

We also implemented TWINE on software. On 8-bit micro-controllers, TWINE is implemented within 0.8 to 1.5 Kbytes ROM. The speed is relatively fast compared to other lightweight ciphers. We also tried implementations on 32 and 64-bit CPUs. Due to the nature of GFS and the use of identical 4-bit S-box, TWINE is quite easy to implement using a SIMD instruction performing a vector permutation, which we call vector-permutation instruction (VPI). Starting from Hamburg's works on AES [19], VPI has been recognized as a powerful tool for fast cryptography (e.g. [1, 10, 11]). Our results show that VPI is surprisingly useful for TWINE. For example, on Intel Core-i5 U560, we observed 4.75 cycles/byte[1] using `pshufb` instruction (which is a VPI for Intel CPUs). This figure is quite impressive in the realm of lightweight block ciphers. For reference, we observed that AES using the technique of [19] (with the same `pshufb` instruction) runs at 6.66 cycles/byte on the same processor. As our VPI-based implementation has a quite simple structure, it is easy to understand and port to other CPUs. These results imply TWINE's well-balanced performance under multiple platforms, which makes it suitable to heterogeneous network. For example, consider a network consisting of a huge number of tiny sensor nodes that independently encrypt sensor information, and a server computer which aggregates information from sensor nodes and performs massive decryption.

Security. As TWINE is a variant of GFS, we consider the cryptanalytic methods exploiting the diffusion property of GFS as most critical. Concretely, we focus on the impossible differential cryptanalysis (IDC) and the saturation cryptanalysis (SC). We perform a thorough analysis on TWINE including IDC and SC. As a result we present IDC against 23-round TWINE-80 and 24-round TWINE-128 as the most powerful attacks we have found so far. Our attacks fully exploit the key schedule. They can be seen as an interesting example of highly-optimized IDC 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. Section 4 presents the results of security evaluation, and Section 5 presents the implementation results of both hardware and software. Section 6 concludes the paper.

2 Specification of TWINE

2.1 Notations

A bitwise exclusive-OR is denoted by $\oplus$. For binary strings, x and y, $x\|y$ denotes their concatenation. Let $|x|$ denote the bit length of x. If $|x| = m$, we may write $x_{(m)}$ to emphasize the length of x. If $|x| = 4c$ for a positive integer c, we write $x \to (x_0\|x_1\|\dots\|x_{c-1})$, where $|x_i| = 4$, is the partition operation into 4-bit sub-blocks. The opposite operation, $(x_0\|x_1\|\dots\|x_{c-1}) \to x$, 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

TWINE is a 64-bit block cipher with 80 or 128-bit key. We write TWINE-80 or TWINE-128 to denote the key length. The global structure of TWINE is a variant of Type-2 GFS [41, 46] with 16 4-bit sub-blocks. A round function of TWINE consists of a nonlinear layer using 4-bit S-boxes and a diffusion layer, which is a permutation on 16 blocks. The diffusion layer of the original Type-2 GFS is simply a (left or right) cyclic shift. Following the result of [42], the diffusion layer of TWINE is not a cyclic shift and is chosen to provide a better diffusion than

[1] In a double-block encryption. See Section 5.2.

the cyclic shift. This round function is iterated for 36 times for both key lengths, where the diffusion layer of the last round is omitted. For $i = 1, \ldots, 36$, i-th round uses a 32-bit round key, RK^i, which is derived from the secret key, $K_{(n)}$ with $n \in \{80, 128\}$, using the key schedule.

The data processing part essentially consists of a 4-bit S-box, denoted by S, and a permutation π over the indexes of 4-bit blocks. That is, we have $\pi : \{0, \ldots, 15\} \to \{0, \ldots, 15\}$, where j-th sub-block is mapped to $\pi[j]$-th sub-block. The figure of the round function is in Fig. 3.

Algorithm 2.3 shows the encryption procedure, and Algorithm **??** shows the decryption procedure. They use the same S-box and the key schedule, however the decryption uses the inverse permutation π^{-1}.

2.3 Key Schedule Part

The key schedule produces $RK_{(32\times 36)}$ from the secret key, $K_{(n)}$, for $n \in \{80, 128\}$. It is a variant of GFS with few S-boxes, which is the same as one used at the data processing. The 80-bit key schedule uses 6-bit round constants, $CON^i_{(6)} = CON^i_{H(3)} \| CON^i_{L(3)}$ for $i = 1$ to 35.

The 80-bit and 128-bit key schedules are shown in Algorithm 2.3 and Algorithm 2.3. Here, $Rot\mathbf{z}(x)$ means z-bit left cyclic shift of x.

We also show a picture of 80-bit key schedule in Fig. 4. We note that CON^i corresponds to 2^i in GF(2^6) with primitive polynomial $z^6 + z + 1$.

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

```
1. $X^1_{0(4)} \| X^1_{1(4)} \| \ldots \| X^1_{15(4)} \leftarrow P$
2. $RK^1_{(32)} \| \ldots \| RK^{36}_{(32)} \leftarrow RK_{(32\times 36)}$
3. for $i = 1$ to 35 do
4.     $RK^i_{0(4)} \| RK^i_{1(4)} \| \ldots \| RK^i_{7(4)} \leftarrow RK^i_{(32)}$
5.     for $j = 0$ to 7 do $X^i_{2j+1} \leftarrow S(X^i_{2j} \oplus RK^i_j) \oplus X^i_{2j+1}$
6.     for $h = 0$ to 15 do $X^{i+1}_{\pi[h]} \leftarrow X^i_h$
7. for $j = 0$ to 7 do $X^{36}_{2j+1} \leftarrow S(X^{36}_{2j} \oplus RK^{36}_j) \oplus X^{36}_{2j+1}$
8. $C \leftarrow X^{36}_0 \| X^{36}_1 \| \ldots \| X^{36}_{15}$
```

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

```
1. $X^{36}_{0(4)} \| X^{36}_{1(4)} \| \ldots \| X^{36}_{15(4)} \leftarrow C$
2. $RK^1_{(32)} \| \ldots \| RK^{36}_{(32)} \leftarrow RK_{(32\times 36)}$
3. for $i = 36$ to 2 do
4.     $RK^i_{0(4)} \| RK^i_{1(4)} \| \ldots \| RK^i_{7(4)} \leftarrow RK^i_{(32)}$
5.     for $j = 0$ to 7 do $X^i_{2j+1} \leftarrow S(X^i_{2j} \oplus RK^i_j) \oplus X^i_{2j+1}$
6.     for $h = 0$ to 15 do $X^{i-1}_{\pi^{-1}[h]} \leftarrow X^i_h$
7. for $j = 0$ to 7 do $X^1_{2j+1} \leftarrow S(X^1_{2j} \oplus RK^1_j) \oplus X^1_{2j+1}$
8. $P \leftarrow X^1_0 \| X^1_1 \| \ldots \| X^1_{15}$
```

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

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

Fig. 1. Encryption and decryption of TWINE, with S-box S and nibble permutation π

2.4 Test Vectors

The test vectors of TWINE are in Table 1.

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

```
1.  $WK_{0(4)} \| WK_{1(4)} \| \ldots \| WK_{19(4)} \leftarrow K$
2.  for $r = 1$ to 35 do
3.      $RK^r_{(32)} \leftarrow WK_1 \| WK_3 \| WK_4 \| WK_6 \| WK_{13} \| WK_{14} \| WK_{15} \| WK_{16}$
4.      $WK_1 \leftarrow WK_1 \oplus S(WK_0)$
5.      $WK_4 \leftarrow WK_4 \oplus S(WK_{16})$
6.      $WK_7 \leftarrow WK_7 \oplus 0 \| CON^r_H$
7.      $WK_{19} \leftarrow WK_{19} \oplus 0 \| CON^r_L$
8.      $WK_0 \| \ldots \| WK_3 \leftarrow \mathrm{Rot4}(WK_0 \| \ldots \| WK_3)$
9.      $WK_0 \| \ldots \| WK_{19} \leftarrow \mathrm{Rot16}(WK_0 \| \ldots \| WK_{19})$
10. $RK^{36}_{(32)} \leftarrow WK_1 \| WK_3 \| WK_4 \| WK_6 \| WK_{13} \| WK_{14} \| WK_{15} \| WK_{16}$
11. $RK \leftarrow RK^1 \| RK^2 \| \ldots \| RK^{36}$
```

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

```
1.  $WK_{0(4)} \| WK_{1(4)} \| \ldots \| WK_{31(4)} \leftarrow K$
2.  For $r = 1$ to 35 do
3.      $RK^r_{(32)} \leftarrow WK_2 \| WK_3 \| WK_{12} \| WK_{15} \| WK_{17} \| WK_{18} \| WK_{28} \| WK_{31}$
4.      $WK_1 \leftarrow WK_1 \oplus S(WK_0)$
5.      $WK_4 \leftarrow WK_4 \oplus S(WK_{16})$
6.      $WK_{23} \leftarrow WK_{23} \oplus S(WK_{30})$
7.      $WK_7 \leftarrow WK_7 \oplus 0 \| CON^r_H$
8.      $WK_{19} \leftarrow WK_{19} \oplus 0 \| CON^r_L$
9.      $WK_0 \| \ldots \| WK_3 \leftarrow \mathrm{Rot4}(WK_0 \| \ldots \| WK_3)$
10.     $WK_0 \| \ldots \| WK_{31} \leftarrow \mathrm{Rot16}(WK_0 \| \ldots \| WK_{31})$
11. $RK^{36}_{(32)} \leftarrow WK_2 \| WK_3 \| WK_{12} \| WK_{15} \| WK_{17} \| WK_{18} \| WK_{28} \| WK_{31}$
12. $RK_{(32\times 36)} \leftarrow RK^1 \| RK^2 \| \ldots \| RK^{36}$
```

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	
CON^i	0F	1E	3C	3B	35	29	11	22	07	0E	1C	38	33	25	09	12	24	

Fig. 2. Key schedules of TWINE, for 80-bit and 128-bit keys. S-box S is the same as Fig. 1, and key schedule constants, CON^i, are described in the bottom.

3 Design Rationale

3.1 Basic Objective

Our goal is to build a lightweight block cipher enabling compact hardware comparable to previous proposals, while keeping the efficiency on various CPUs, from low-end micro-controller to general-purpose 32-bit or 64-bit CPU.

On LBlock. We remark that LBlock [45], proposed independently of ours, is quite similar to our proposal. It is a 64-bit block cipher using a variant of balanced Feistel whose round function consists of 8 4-bit S-boxes and a nibble-wise permutation and a 8-bit cyclic shift. Such a structure can be transformed into a structure proposed at [42], though we do not know whether

Table 1. Test vectors in 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

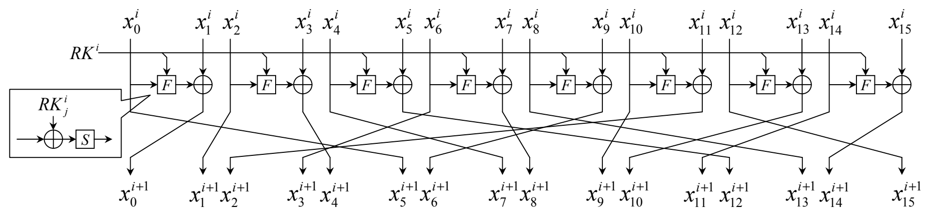
Fig. 3. Round function of TWINE.

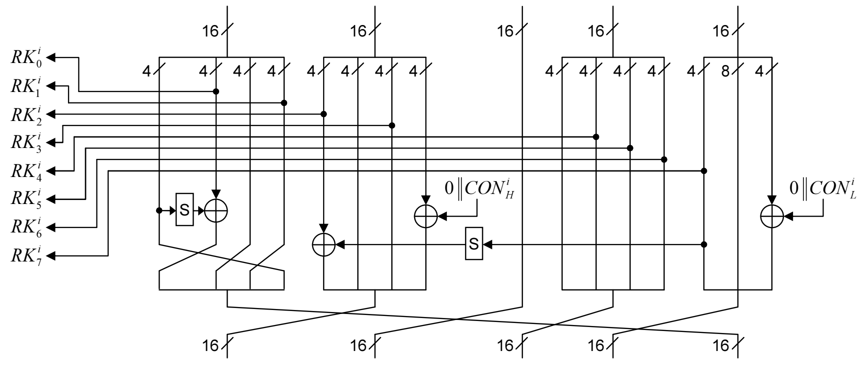
Fig. 4. 80-bit key schedule.

the authors of [45] are aware of it. We investigated LBlock in this representation and found that the LBlock's diffusion layer is equivalent to that of the decryption of TWINE. Note that this choice is reasonable from Table 6 of [42], saying this diffusion layer satisfies both of the fastest diffusion and the highest immunities against linear and differential attacks among other block shuffles. Nevertheless, there are two important differences between TWINE and LBlock, as follows;

1. LBlock uses ten distinct S-boxes, while TWINE uses one S-box, including the key schedule. The design of TWINE contributes to a compact serialized hardware and fast software. Indeed, our fast SIMD implementation was impossible if multiple S-boxes were used.
2. LBlock uses a bit permutation in its key scheduling, which decreases software efficiency. TWINE does not use a bit permutation for both data processing and key scheduling.

3.2 Parameters and Components

Rounds. As far as we investigated, the most powerful attack against TWINE is an optimized 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. Employing the same 36-round for TWINE-80 may look slight odd, however, it enables various multiple-round circuits with a small overhead, since 36 has many factors.

Block Shuffle. The block shuffle π comes from a result of Suzaki and Minematsu [42]. In [42], 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 for achieving full diffusion for any input difference (that is, any input-block difference diffuses to all output blocks), called DRmax. Smaller DRmax means a faster diffusion. DRmax of cyclic shift with k sub-blocks is k, while there exist shuffles (permutations) with DRmax $= 2\log_2 k$, where [42] called "optimum block shuffle" . Our π is such one[2] with $k = 16$, hence DRmax $= 8$. This shows a significant improvement over DRmax $= 16$ obtained by 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, with permutation π we chose these characteristics are reduced to 14 and 15 rounds.

Optimum block shuffle is not unique in general [42]. Hence our π was chosen considering other aspects which is not directly related to DRmax. In particular, we chose π considering the the number of differentially and linearly active S-boxes (See Table 8 in Section 4).

S-box. The 4-bit S-box is chosen to satisfy that (1) the maximum differential and linear probabilities are 2^{-2}, which is theoretically the minimum for invertible S-box, and (2) the Boolean degree is 3, and (3) the interpolation polynomial contains many terms and has degree 14. Following the AES S-box design, we use a Galois field inversion. Specifically 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), where the zero element is mapped to itself. The irreducible polynomial is $z^4 + z + 1$, and $b = 1$ is a constant, and $f(\cdot)$ is an affine function such that $y = f(x)$ with $y = (y_0 \| y_1 \| y_2 \| y_3)$ and $x = (x_0 \| x_1 \| x_2 \| x_3)$ is determined as $y_0 = x_2 \oplus x_3$, $y_1 = x_0 \oplus x_3$, $y_2 = x_0$, and $y_3 = x_1$.

Key Schedule. The key schedule of TWINE enables on-the-fly operations and produces each round key via a sequential update of key state. In particular, there is no intermediate key and no bit permutation. As hardware efficiency is not our ultimate goal, the design is rather conservative compared to the recent hardware-oriented ones [12,34,40], 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. We here focus on the impossible differential and saturation attacks and explain the basic flows of these attacks, since they are the most critical attacks according to our security evaluation. The results on other attacks, such as differential and linear attacks, will be briefly described.

In this section, we use the following notations. A sequence of c symbols, S, is denoted by $\bar{S}^c$, 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 $F^i_0, \ldots, F^i_{15}$, where F^i_0 is the leftmost one. We let $RK^i_{[j_1,\ldots,j_h]}$ to denote the vector $(RK^i_{j_1}, \ldots, RK^i_{j_h})$. Since RK^i_j is the j-th 4-bit subsequence of RK^i (for $j = 0, 1, \ldots, 15$), this means $F^i_j(x) = S(RK^i_j \oplus x)$. $X^i_{[j_1,\ldots,j_h]}$ is similarly defined.

4.2 Impossible Differential Attack

Generally, impossible differential attack [3] is one of the most powerful attacks against Feistel and GFS-based ciphers, as demonstrated by (e.g.) [14,31,44]. We searched impossible differential characteristics (IDCs) using Kim et al.'s method [21], and found 64 14-round IDCs,

$$(0, \alpha_0, 0, \alpha_1, 0, \alpha_2, \ldots, 0, \alpha_7) \overset{14r}{\not\rightarrow} (\beta_0, 0, \beta_1, 0, \beta_2, 0, \ldots, \beta_7, 0), \tag{1}$$

[2] More precisely, an isomorphic shuffle to one presented at Appendix B ($k = 16$, No. 10) of [42].

where all variables are 4-bit, $\alpha_i \neq 0$, $\beta_j \neq 0$ for some $i, j \in \{0, \ldots, 7\}$ and others are 0. Based on this we can attack against 23-round TWINE-80, where IDC of 5-th to 18-th rounds with $\alpha_0 \neq 0$ and $\beta_4 \neq 0$ is used, and tries to recover the subkeys of the first 4 rounds and the last 5 rounds, 144 bits in total. These subkey bits are uniquely determined via its 80-bit subsequence. A similar attack is possible against 24-round TWINE-128, using the IDC with $\alpha_3 \neq 0$ and $\beta_2 \neq 0$.

The outline of our attack against 23-round TWINE-80 is as follows.

Data Collection. We call a set of 2^{32} plaintexts a *structure* if its i-th (4-bit) 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 difference

$$(p_1, p_2, 0, p_3, \bar{0}^6, p_4, p_5, p_6, p_7, 0, p_0), \tag{2}$$

where p_i is a non-zero 4-bit plaintext difference value. With plaintext pairs in the structure, we want to find the event that the difference of the internal states after the first 4 rounds to match $(0, \alpha_0, \bar{0}^{14})$, i.e. one of the IDCs defined in the left hand side of Eq. (1). To invoke this event, 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 output differential of F_0^1 which has input differential α_1, as shown by Fig. 5. Here, we note that for any non-zero input difference p_x, the output difference of F_j^i is always one of 7 possible values, depending on RK_j^i and p_x. More formally we have;

Proposition 1. *Let $y = F_j^i(x) = S(RK_j^i \oplus x)$ 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 with difference being Eq (2) from a structure.
Key Elimination. After the plaintext pairs have been generated, we encrypt them and seek the ciphertext pairs with a differential

$$(0, \beta_1, 0, \beta_2, \beta_3, \beta_4, \beta_0, \beta_5, \beta_6, \beta_7, \beta_8, \beta_9, \beta_{10}, \beta_{11}, 0, 0), \ {}^\forall\beta_i \neq 0. \tag{3}$$

For each ciphertext pair with differential Eq. (3), 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

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

using the plaintext pair of differential Eq. (2) and the ciphertext pair of differential Eq. (3). 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. In other words, the wrong subkey is eliminated with probability $2^{-44.31}$. The detail of the above procedure is depicted at Table 2.

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 (3) with probability $(2^{-4})^4 \cdot (2^{-1})^8 = 2^{-24}$. Then it seems that we basically need $2^{74.11}$ ciphertext pairs. However, 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. (4). 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 a table called Difference Table. An entry of Difference Table is indexed by $(x, x', y) \in (\{0,1\}^4)^3$ and the entry specifies 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 (see Table 3), or we can directly guess the key if input and output pairs of the corresponding F are fixed, not only their differentials but the values.

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. 5, and five more Fs, $F^{23}_{[2,3,4,5,6]}$, thus total 22 F functions. This amounts to $22/(23 \cdot 8)$ encryptions of 23-round TWINE.

Consequently, we can attack 23-round TWINE-80 with the data complexity $2^{29.55} \cdot 2^{32} = 2^{61.55}$ blocks, the time complexity $2^{84.56} \cdot 22/(23 \cdot 8) = 2^{77.04}$ encryptions, and the memory complexity $2^{80}/64 = 2^{74}$ blocks. In a similar manner, we can attack 24-round TWINE-128 with the data, time and memory complexity being $2^{52.21}$ blocks, $2^{115.10}$ encryptions and 2^{118} blocks respectively. The result is summarized by Table 4.

Table 2. Procedure of round key determination. Here, iF_j^i and oF_j^i denote input and output values of F_j^i, and ioF_j^i denotes a set of iF_j^i and oF_j^i.

Step	Target subkey	Method	Other fixed data	Step	Target subkey	Method	Other fixed data
1	RK_0^1, RK_5^1, RK_6^1	diff-table		14	$RK_5^{21}, RK_3^{23}, RK_6^{23}$	key rel.	iF_2^{22},ioF_0^{22},iF_5^{21}, oF_4^{22}
2	RK_4^3, RK_1^4	key rel.	iF_3^3,oF_4^2,iF_1^4	15	RK_2^{22}	diff-table	
3	RK_3^1	guess		16	RK_6^{20}	key rel.	
4	RK_4^2	IO value		17	RK_1^{23}	key rel.	ioF_3^{22}
5	RK_7^1	guess	iF_7^2	18	RK_3^{22}	diff-table	
6	RK_7^2	diff-table		19	RK_0^{22}	IO value	
7	RK_1^1	guess	iF_2^2	20	RK_4^{19}	key rel.	
8	RK_5^3	key rel.	oF_6^2	21	RK_4^{22}	IO value	
9	RK_2^2	diff-table		22	RK_0^{23}	guess	ioF_1^{22}
10	RK_2^1	guess		23	RK_1^{22}	diff-table	
11	RK_6^2	IO value		24	RK_4^{21}, RK_7^{21}	key rel.	
12	$RK_2^{23}, RK_4^{23}, RK_5^{23}$	diff-table	ioF_0^{22}	25	$RK_5^{22}, RK_6^{22}, RK_7^{23}$	key rel.	
13	RK_3^{20}, RK_1^{21}	key rel.	iF_3^{20}				

Table 3. Subkey relationships used for the impossible differential attack.

$$
\begin{array}{|l|}
\hline
RK_3^3 = RK_5^1, \quad RK_5^3 = RK_1^1, \quad RK_1^4 = RK_6^1, \quad RK_0^{21} = S[S[S[RK_4^{19}] \oplus RK_3^1] \oplus RK_4^{21}] \oplus RK_7^2 \\
RK_5^{21} = S^{-1}[S^{-1}[RK_1^{21} \oplus S[RK_3^{20}] \oplus RK_1^1] \oplus RK_6^1], \quad RK_0^{22} = RK_4^{19}, \quad RK_2^{22} = S[RK_6^{20}] \oplus S[RK_7^2] \oplus RK_2^2 \\
RK_4^{22} = S[S[S[RK_3^{20}] \oplus RK_1^1] \oplus RK_3^{22}] \oplus S[S[S[S[RK_6^{20}] \oplus S[RK_7^2] \oplus RK_2^2] \oplus S^{-1}[RK_1^{22} \\
\qquad \oplus S^{-1}[S^{-1}[S^{-1}[RK_1^{21} \oplus S[RK_3^{20}] \oplus RK_1^1] \oplus RK_6^1] \oplus S[RK_4^{21}] \oplus RK_5^1]]] \oplus RK_7^{21}] \oplus RK_4^2 \\
RK_5^{22} = S^{-1}[S^{-1}[RK_1^{22} \oplus S^{-1}[S^{-1}[S^{-1}[RK_1^{21} \oplus S[RK_3^{20}] \oplus RK_1^1] \oplus RK_6^1] \oplus S[RK_4^{21}] \oplus RK_5^1]] \oplus RK_6^2] \\
RK_6^{22} = S[S[RK_6^{20}] \oplus S[RK_7^2] \oplus RK_2^2] \oplus S^{-1}[RK_1^{22} \oplus S^{-1}[S^{-1}[S^{-1}[RK_1^{21} \oplus S[RK_3^{20}] \oplus RK_1^1] \\
\qquad \oplus RK_6^1] \oplus S[RK_4^{21}] \oplus RK_5^1]] \\
RK_0^{23} = S[S[S[RK_4^{21}] \oplus RK_5^1] \oplus RK_3^{20}] \oplus S^{-1}[S[S[S[RK_6^{20}] \oplus S[RK_7^2] \oplus RK_2^2] \oplus S^{-1}[RK_1^{22} \\
\qquad \oplus S^{-1}[S^{-1}[S^{-1}[RK_1^{21} \oplus S[RK_3^{20}] \oplus RK_1^1] \oplus RK_6^1] \oplus S[RK_4^{21}] \oplus RK_5^1]]] \oplus RK_7^{21} \oplus RK_7^1] \\
RK_1^{23} = RK_6^{20}, \quad RK_3^{23} = S^{-1}[S^{-1}[RK_1^{21} \oplus S[RK_3^{20}] \oplus RK_1^1] \oplus RK_6^1] \\
RK_4^{23} = RK_3^{20}, \quad RK_5^{23} = RK_1^{21}, \quad RK_6^{23} = S[RK_2^{23}] \oplus S[RK_1^{21}] \oplus S^{-1}[RK_7^2 \oplus RK_0^1] \\
RK_7^{23} = S[S[RK_7^{21}] \oplus S[RK_1^{22}] \oplus S[RK_7^1] \oplus RK_2^1] \oplus RK_4^{19} \\
\hline
\end{array}
$$

Table 4. Complexity of impossible differential attack.

Key	Round	Data (blocks)	Time (encryption)	Memory (blocks)
80	23	$2^{61.55}$	$2^{77.04}$	2^{74}
128	24	$2^{52.21}$	$2^{115.10}$	2^{118}

4.3 Saturation Attack

Saturation attack [16] is also a powerful attack against GFS-based ciphers. For the analysis of TWINE, we consider saturation attack on 4-bit data path. Then the state consists of 2^4 variables, denoted by $\mathtt{S} = (\mathtt{S}_0, \ldots, \mathtt{S}_{15})$, where $\mathtt{S}_i$ has the following four status (here X is the plaintext):

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

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

Let $\alpha = (\alpha_0, \ldots, \alpha_{15})$ and $\beta = (\beta_0, \ldots, \beta_{15})$, $\alpha_i, \beta_i \in \{\mathtt{C}, \mathtt{A}, \mathtt{B}, \mathtt{U}\}$, be the initial state and the t-round state holding with probability 1. If we have $\alpha_i = \mathtt{A}$ and $\beta_j \neq \mathtt{U}$ for some i and j, we call $\alpha \xrightarrow{tr} \beta$ a t-round saturation characteristic (SC). We found that TWINE has 15-round SCs with input consisting of one $\mathtt{C}$ and fifteen $\mathtt{A}$s and output consisting of 4 $\mathtt{B}$s, and the remainings are $\mathtt{U}$. For example;

$$(\mathtt{C}, \bar{\mathtt{A}}^{15}) \xrightarrow{15r} (\mathtt{U}, \mathtt{B}, \mathtt{U}, \mathtt{B}, \bar{\mathtt{U}}^9, \mathtt{B}, \mathtt{U}, \mathtt{B}), \tag{5}$$

$$(\bar{\mathtt{A}}^6, \mathtt{C}, \bar{\mathtt{A}}^9) \xrightarrow{15r} (\bar{\mathtt{U}}^5, \mathtt{B}, \mathtt{U}, \mathtt{B}, \mathtt{U}, \mathtt{B}, \mathtt{U}, \mathtt{B}, \bar{\mathtt{U}}^4). \tag{6}$$

There are other input state patterns that cause one of the two output state patterns listed above, and there is no other output state patterns for 15 rounds.

Suppose we use SC of Eq. (6) to break 22-round TWINE-80. Let *S-structure* denote a set of 2^{60} plaintexts induced from Eq. (6), i.e., the input block X_0 is fixed to a constant and the other blocks take all combinations.

We try to recover a 72-bit subkey vector,

$$\mathcal{K}_{\text{target}} = (RK^{22}, RK^{21}_{[0,2,3,4,5,6,7]}, RK^{20}_{[6,7]}, RK^{16}_0),$$

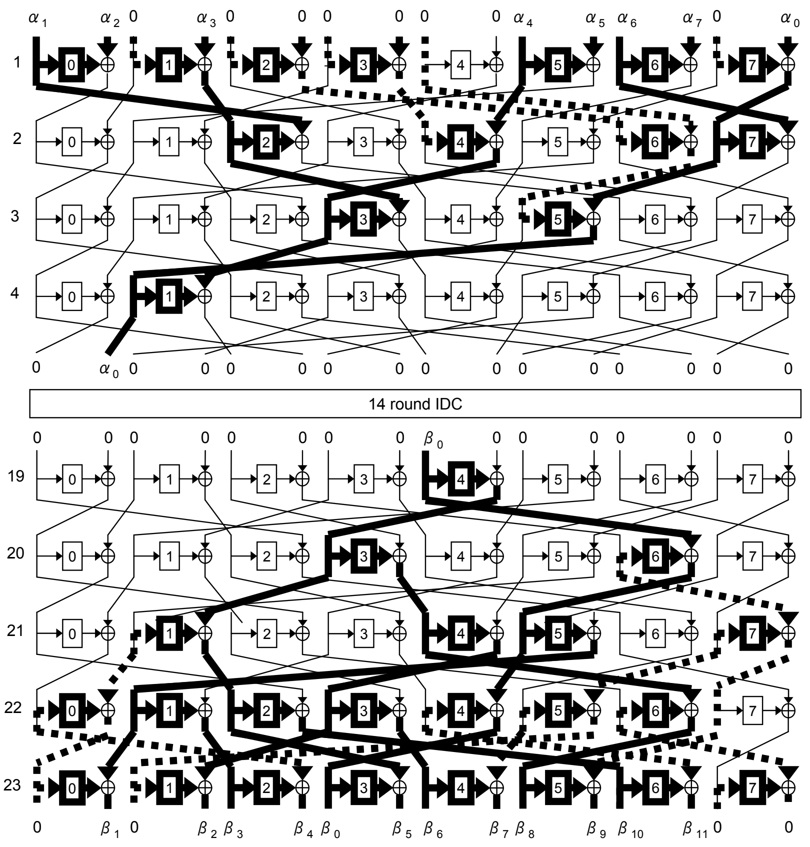

Fig. 5. First 4 rounds and the last 5 rounds in the impossible differential attack.

based on the fact that (the state of) X_1^{15} is B with these 15-round SCs. Here, that event that the state of X_1^{15} is B implies the coincidence of states between X_0^{16} and the output of F_0^{16}, the leftmost F function in the round 16, computed from the S-structure. From this ovservation, we calculate the sums of F_0^{16} output 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.

Step 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}.

Step 2. Write all ciphertexts except their rightmost (16-th) block, X_{15}^{22}, to the 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

$$\mathcal{K}_1 = (RK^{22}_{[0,1,2,3,4,5,6]}, RK^{21}_{[2,3,4,5,7]}, RK^{20}_6, RK^{17}_2, RK^{16}_0)$$

and compute the sum of outputs of F_0^{16} using all entries in $\mathcal{L}_1$ with each guess for $\mathcal{K}_1$. More precisely, $RK^{20}_{[0,5]}$,$RK^{19}_{[1,7]}$,RK^{18}_3 are required to compute F_0^{16} outputs. More precisely, $RK^{20}_{[0,5]}$,$RK^{19}_{[1,7]}$,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 $\mathcal{K}_2$ (of Step 3). See Table 5.

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 that sets F_0^{16} output sum as $s \in \{0,1\}^4$. See also Fig. 6.

Table 5. Subkey relationships used for our saturation attack.

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

Step 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

$$\mathcal{K}_2 = (RK^{22}_{[2,3,4,5,7]}, RK^{21}_{[0,5,6]}, RK^{20}_7, RK^{18}_2)$$

and compute the sum of X^{16}_0 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^{16}_0. The key group that sets the sum of X^{16}_0 as $s' \in \{0,1\}^4$ is denoted by $\mathcal{G}_2(s')$. See also Fig. 6.

Step 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 $\mathcal{K}_{\text{target}}$. 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 $RK^i_j(v)$ and $RK^i_j(w)$. Both v and w contain guesses of $RK^{22}_{[2,3,4,5]}$ and RK^{21}_5. If the following four equations,

$$RK^{22}_{[2,3,4,5]}(v) = RK^{22}_{[2,3,4,5]}(w), \quad RK^{21}_5(v) = RK^{21}_5(w), \tag{7}$$

$$RK^{21}_6(w) = S(RK^{21}_2(v)) \oplus RK^{18}_2(w), \text{ and} \tag{8}$$

$$RK^{22}_7(w) = S(S(RK^{20}_7(w)) \oplus S^{-1}(RK^{20}_6(v) \oplus RK^{17}_2(v))) \oplus RK^{21}_0(w) \tag{9}$$

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

Taking Step 2 for example, we provide details. We first guess RK^{22}_0 and compute X^{21}_1, which is equivalent to $F^{22}_0(X^{22}_0) = S(RK^{22}_0 \oplus X^{22}_0)$, 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^{16}_0 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.

As checks are done with respect to 4-bit internal values, the above procedure with one S-structure rejects a wrong key candidate for $\mathcal{K}_{\text{target}}$ with probability $1 - 2^{-4}$ (i.e. the size of candidates is reduced to 1/16), hence we basically need at least 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. More specifically, 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 for the same SC can help. The result is shown by Table 6. According to our investigation, the attack with 5 or more S-structures has higher time complexity than that with 4 S-structures, hence the best one is with 4 S-structures.

Consequently, using 4 S-structures, we can attack 22-round TWINE-80 with the data complexity 2^{62} blocks, the time complexity $2^{68.43}$ encryptions, and the memory complexity 2^{67} blocks. In a similar manner, we can attack 23-round TWINE-128 with the data, time and memory complexity being $2^{62.81}$ blocks, $2^{106.14}$ encryptions and 2^{103} blocks respectively. The result is summarized by Table 7.

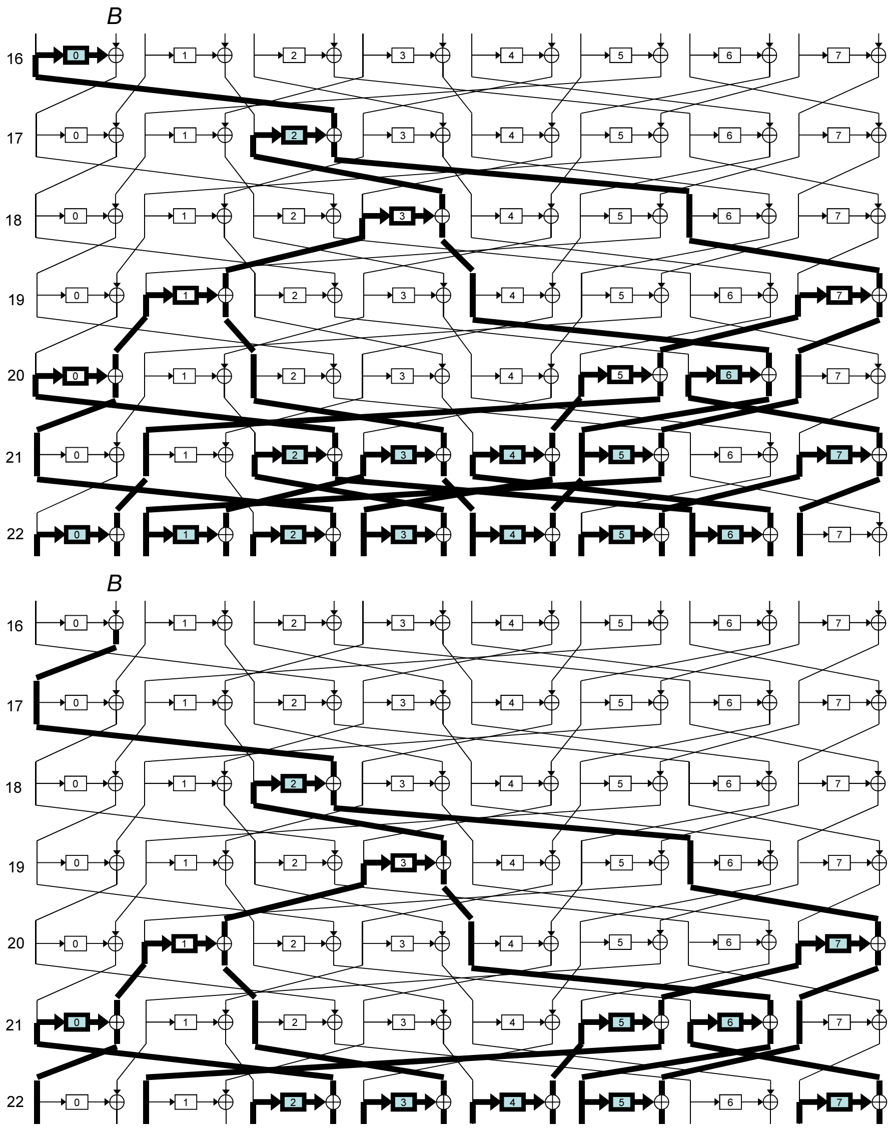

Fig. 6. Step 2 (Upper) and Step 3 (Lower) of saturation attack.

Table 6. Complexity of saturation attack using multiple S-structures.

Number of S-structures	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}

Table 7. Complexity of saturation attack.

Key	Rounds	Data (blocks)	Time (encryption)	Memory (blocks)
80	22	2^{62}	$2^{68.43}$	2^{67}
128	23	$2^{62.81}$	$2^{106.14}$	2^{103}

4.4 Differential / Linear Cryptanalysis

The security against differential cryptanalysis (DC) [4] and linear cryptanalysis (LC) [28] are typically evaluated by 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 (Table 8). 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 ($\mathit{\Delta}$) and linear ($\mathit{\Gamma}$) characteristics having the minimum I/O weights are as follows. Here, 1 denotes an arbitrary non-zero difference (mask) and 0 denotes the zero difference (mask) for $\mathit{\Delta}$ ($\mathit{\Gamma}$). They involve 30 active S-boxes, and thus the characteristic probability is 2^{-60}.

$$\begin{aligned}\mathit{\Delta} &= (\bar{0}^9, 1, 0, 1, 0, 1, 0, 0) \xrightarrow{14r} (\bar{0}^3, 1, \bar{0}^4, 1, 0, 0, 1, 0, 0, 1, 1),\\ \mathit{\Gamma} &= (\bar{0}^6, 1, 1, \bar{0}^3, 1, 0, 0, 1, 1) \xrightarrow{14r} (\bar{0}^9, 1, \bar{0}^3, 1, 0, 1).\end{aligned} \tag{10}$$

Compared to the impossible differential attack, we expect the key recovery attacks exploiting the key schedule with these differential or linear characteristic are less powerful, since they have larger weight (number of non-zero variables) than that of 14-round IDC (having weight 2) and fewer weights generally imply the more attackable rounds in the key guessing.

We also remark that a computer-based search for the maximum differential probability of GFS, rather than the characteristic probability, was reported by Minematsu et al. [29]. However, applying their algorithm to our 16-block case seems computationally infeasible.

Table 8. List of differentially and linearly active S-boxes.

Round	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
AS_D, AS_L	0	1	2	3	4	6	8	11	14	18	22	24	27	30	32	35	36	39	41	44

4.5 Key Schedule-based Attacks

Related-Key Differential Attacks. The related-key attack proposed by Biham [2] works when the adversary can somehow modify the key input, typically insert a key differential. For evaluation of such attack, we implemented the search method by Biryukov et al. [6], which counts the number of active S-boxes for combined data processing and key schedule parts. See [6] for the algorithmic details. We 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).

The full-search was only computationally feasible for TWINE-80. As a result, the number of active S-boxes reaches 40 for the 22-round. Table 9 shows the search result, where ΔKS, ΔRK, ΔX and AS denote key difference, subkey difference, data difference, and the number of active S-boxes.

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 MITM attack (at least in its basic form) against the full-round TWINE.

Table 9. Truncated differential and its active S-box numbers.

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

5 Implementation

5.1 Hardware

We implemented TWINE on ASIC using a 90nm standard cell library with logic synthesis done by *Synopsys DC Version D-2010.03-SP1-1*. Following [8,12], we used Scan Flip-Flops (FFs). 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 result is shown by Table 10. Note that for some algorithms other than TWINE, the synthesis was not done at 100KHz, hence we estimated the throughput by scaling. Table 11 shows the detail of TWINE-80 round-based implementation, where single round function is computed in a clock. 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. The figures must be taken with cares, because they depend on the type of FF, technology, library, etc [12]. As suggested by [12], we list Gates/Memory Bit in the table, which denotes the size (in GE) of 1-bit memory device used for the key and states.

Serialized Architecture. For serialized implementation, we employ a shared sbox architecture design where single S-box is repeatedly used in the data processing and the key scheduling. Our implementation employs a rotater which consists of shifter and permutator (which performs a permutation on the shifter outputs). For encryption-only TWINE-80, it achieved $1,011$ GEs. The details will be given in a separate paper.

Table 10. ASIC implementation results.

Algorithm	Function	Block (bit)	Key (bit)	Cycles/ block	Throughput (Kbps@100KHz)	Area (GE†)	Gates / Memory bit	Type
TWINE	Enc	64	80	36	178	1,503	6.75	round
TWINE	Enc+Dec	64	80	36	178	1,799	6.75	round
TWINE	Enc	64	128	36	178	1,866	6.75	round
TWINE	Enc+Dec	64	128	36	178	2,285	6.75	round
TWINE	Enc	64	80	393	16.2	1,011	6.75	serial
PRESENT [38]	Enc	64	80	547	11.4	1,000	n/a	serial
PRESENT [8]	Enc	64	80	32	200	1,570	6	round
AES [30]	Enc	128	128	226	57	2,400	6	serial
mCRYPTON [24]	Enc	64	64	13	492.3	2,420	5	round
SEA [25]	Enc+Dec	96	96	93	103	3,758	n/a	round
HIGHT [20]	Enc+Dec	64	128	34	188.25	3,048	n/a	round
KLEIN [17]	Enc	64	80	17	376.4	2,629	n/a	round
KLEIN [17]	Enc	64	80	271	23.6	1,478	n/a	serial
DES [23]	Enc	64	56	144	44.4	2,309	12.19	serial
DESL [23]	Enc	64	56	144	44.4	1,848	12.19	serial
KATAN [12]	Enc	64	80	254	25.1	1,054	6.25	serial
Piccolo [40]	Enc	64	80	27	237	1,496¶	6.25	round
Piccolo [40]	Enc+Dec	64	80	27	237	1,634¶	6.25	round
Piccolo [40]	Enc	64	80	432	14.8	1,043¶	6.25	serial
Piccolo [40]	Enc+Dec	64	80	432	14.8	1,103¶	6.25	serial
LED [18]	Enc	64	80	1872	3.4	1,040	6/4.67◇	serial
PRINTcipher [22]	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.

5.2 Software

We implemented TWINE on Atmel AVR 8-bit micro-controller. The target device is ATmega163, which has 16K bytes Flash, 512 bytes EEPROM and 1,024 bytes SRAM. We built the four versions: speed-first, ROM-first (minimizing the consumption), and RAM-first, and the double-block, where two message blocks are processed in parallel. Such an implementation works for parellelizable mode of operations, e.g., counter mode. All versions precompute the round keys, i.e. they do not use an on-the-fly 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.

Table 11. Hardware implementation of TWINE-80 encryption.

Data Processing (GE)		Key Scheduling (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

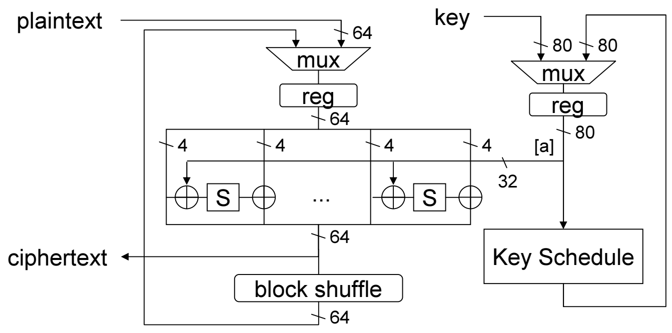

Fig. 7. Data path of TWINE-80 encryption.

Table 12 shows comparison of TWINE and other lightweight block ciphers. We list the (scaled) throughput/code ratio for a performance measure (See Table 12 for the formula), following [37]. AES's performance is still quite impressive, however, one can also observe a good performance of TWINE.

Vector Permutation Instruction. We also implemented TWINE on CPU equipped with a SIMD instruction performing a vector permutation, which we call Vector Permutation Instruction (VPI). Many modern CPUs have VPIs, for example, `vperm` in Motorola AltiVec, `pshufb` in Intel SSE (SSSE3), and `vtbl` in ARM NEON. The use of VPI for the implementation of crypto function was first presented by Hamburg [19] for AES. After [19] the same technique has been applied to various cryptographic functions, e.g. [1,10,11]. However, to the best of our knowledge VPI-based *lightweight* block cipher implementation is not known to date.

The details of our VPI-based implementation is as follows. We first transform TWINE into an equivalent form presented below. This form uses 4 different shuffles working on 8 blocks (nibbles) rather than 16 blocks in a cyclic manner. "nibble index of RK" denotes the index of round key RK given to $F_0, F_1, \ldots, F_7$. For example, in $(4i+2)$-th round F_0 takes RK_0^{4i+1} and F_1 takes RK_2^{4i+1}. The difference in RK indices in each round can be absorbed to the key schedule and causes no bad effect in the actual encryption/decryption. "half shuffle" denotes a shuffle of 8 nibbles.

For the case of Intel CPU with SSSE3, we use `pshufb` for block shuffle and S-box, and an encryption round of TWINE is computed using only 6 instructions;

Table 12. Software implementations on ATmega163.

Algorithm	Key (bit)	Block (bit)	Lang	ROM (byte)	RAM (byte)	Enc (cyc/byte)	Dec (cyc/byte)	ETput /code†	DTput /code‡
TWINE(speed-first)	80	64	asm	1,304	414	271	271	2.14	2.14
TWINE(ROM-first)	80	64	asm	728	335	2,350	2,337	0.40	0.40
TWINE(RAM-first)	80	64	asm	792	191	2,350	2,337	0.43	0.43
TWINE(double block)	80	64	asm	2,294	386	163	163	2.29	2.29
PRESENT [33]	80	64	asm	2,398	528	1,199	1,228	0.28	0.28
DES [36]	56	64	asm	4,314	n/a	1,079	1,019	0.21	0.22
DESXL [36]	184	64	asm	3,192	n/a	1,066	995	0.29	0.31
HIGHT [36]	128	64	asm	8,836	n/a	307	307	0.36	0.36
IDEA [36]	128	64	asm	596	n/a	338	1,924	4.97	0.87
TEA [36]	128	64	asm	1,140	n/a	784	784	1.11	1.11
SEA [36]	96	96	asm	2,132	n/a	805	805	0.58	0.58
AES [9]	128	128	asm	1,912	432	125	181	3.42	2.35

† Encryption Throughput per code: (1/Enc)/(ROM + RAM) (scaled by 10^6)
‡ Decryption Throughput per code: (1/Dec)/(ROM + RAM) (scaled by 10^6)

```
movdqa     xmm2, [eax]       : Load RK
pxor       xmm2, xmm0        : RK xor #0...#7
movdqa     xmm3, [sbox]      : S-box Load
pshufb     xmm3, xmm2        : S-box Look up
pxor       xmm1, xmm3        : #8...#15 xor output of S-box
pshufb     xmm0, shuffle     : half shuffle
```

Here, input data `#0` to `#7` is in `xmm0`, `#8` to `#15` is in `xmm1`, and `eax` contains the address of round key. Note that this implementation is not possible for LBlock due to the use of multiple S-boxes. We remark that this code can treat two blocks at once (which we call double-block code), since each nibble data is stored in a byte structure and XMM registers are 128-bit.

Table 14 shows a performance of TWINE. For comparison, we also implement VPI-based implementation of AES [19] and (standard) T-table-based AES and measured their performance figures. We see single-block TWINE is comparable to VPI AES, and double-block TWINE is faster than VPI AES. Key Schedule for 80-bit (128-bit) key requires about 200 (290) cycles, on Core i7 2600S. Recently, Matsuda and Moriai [27] presented interesting results on SIMD-based, bitslice implementations of lightweight blockciphers. The speed of their implementations are comparable to our double-block implementations, though the number of parallel blocks at the process is larger than ours.

Table 13. 4-round structure for SIMD-based implementation.

round ($0 \leq i \leq 8$)	nibble index of RK	half shuffle
$4i + 1$	$0, 1, 2, 3, 4, 5, 6, 7$	$(1, 0, 4, 5, 2, 3, 7, 6)$
$4i + 2$	$0, 2, 6, 4, 3, 1, 5, 7$	$(5, 3, 7, 1, 6, 0, 4, 2)$
$4i + 3$	$0, 6, 5, 3, 4, 2, 1, 7$	$(6, 7, 3, 2, 5, 4, 0, 1)$
$4i + 4$	$0, 5, 1, 4, 3, 6, 2, 7$	$(2, 4, 0, 6, 1, 7, 3, 5)$

Table 14. Enc/Dec speed (in cycles/byte) of TWINE and AES on Intel CPUs.

Processor (codename)	TWINE(single)	TWINE(double)	AES(VPI)	AES(T-table)
Core i5 U560 (Arrandale)	9.47 / 9.49	4.77 / 4.77	6.66 / 9.12	14.26 / 19.27
Core i7 2600S (Sandy Bridge)	11.10 / 11.11	5.55 / 5.55	7.42 / 9.44	14.04 / 21.17
Core i3 2120 (Sandy Bridge)	15.06 / 15.06	7.55 / 7.53	10.28 / 12.37	19.03 / 28.68
Xeon E5620 (Westmere-EP)	13.62 / 13.65	6.87 / 6.87	14.72 / 17.82	31.60 / 42.69
Core2Quad Q9550 (Yorkfield)	15.16 / 15.60	7.93 / 7.95	12.16 / 14.39	22.74 / 30.94
Core2Duo E6850 (Conroe)	26.85 / 26.86	14.85 / 14.86	22.04 / 25.82	22.43 / 30.76

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 software performance from micro-controller to high-end CPU. 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 of SAC 2012 for many valuable comments. We thank Yosuke Todo for pointing out an oversight in the pre-proceeding with respect to the saturation characteristic, and Huiling Zhang for pointing out an error also regarding the saturation characteristic in the proceeding. We thank Maki Shigeri, Etsuko Tsujihara, Teruo Saito, Takeshi Kawabata, Hiroyasu Kubo and Daisuke Ikemura for discussions and helping implementations.

References

1. Bernstein, D.J., Schwabe, P.: NEON crypto (2012), `http://cr.yp.to/papers.html`
2. Biham, E.: New Types of Cryptanalytic Attacks Using Related Keys. J. Cryptology 7(4), 229–246 (1994)
3. Biham, E., Biryukov, A., Shamir, A.: Cryptanalysis of Skipjack Reduced to 31 Rounds Using Impossible Differentials. In: Stern, J. (ed.) EUROCRYPT. Lecture Notes in Computer Science, vol. 1592, pp. 12–23. Springer (1999)
4. Biham, E., Shamir, A.: Differential cryptanalysis of the data encryption standard. Springer-Verlag, London, UK, UK (1993)
5. Biryukov, A. (ed.): Fast Software Encryption, 14th International Workshop, FSE 2007, Luxembourg, Luxembourg, March 26-28, 2007, Revised Selected Papers, Lecture Notes in Computer Science, vol. 4593. Springer (2007)
6. Biryukov, A., Nikolic, I.: Automatic Search for Related-Key Differential Characteristics in Byte-Oriented Block Ciphers: Application to AES, Camellia, Khazad and Others. In: Gilbert, H. (ed.) EUROCRYPT. Lecture Notes in Computer Science, vol. 6110, pp. 322–344. Springer (2010)
7. Biryukov, A., Wagner, D.: Slide Attacks. In: Knudsen, L.R. (ed.) FSE. Lecture Notes in Computer Science, vol. 1636, pp. 245–259. Springer (1999)
8. 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. Lecture Notes in Computer Science, vol. 4727, pp. 450–466. Springer (2007)
9. Bos, J.W., Osvik, D.A., Stefan, D.: Fast Implementations of AES on Various Platforms. SPEED-CC – Software Performance Enhancement for Encryption and Decryption and Cryptographic Compilers (2009), `http://www.hyperelliptic.org/SPEED/`
10. Brumley, B.B.: Secure and Fast Implementations of Two Involution Ciphers. Cryptology ePrint Archive, Report 2010/152 (2010), `http://eprint.iacr.org/`
11. Calik, C.: An Efficient Software Implementation of Fugue. Second SHA-3 Candidate Conference (2010), `http://csrc.nist.gov/groups/ST/hash/sha-3/Round2/Aug2010/index.html`

12. Cannière, C.D., Dunkelman, O., Knezevic, M.: KATAN and KTANTAN - A Family of Small and Efficient Hardware-Oriented Block Ciphers. In: Clavier and Gaj [15], pp. 272–288
13. Canright, D.: A Very Compact S-Box for AES. In: Rao, J.R., Sunar, B. (eds.) CHES. Lecture Notes in Computer Science, vol. 3659, pp. 441–455. Springer (2005)
14. Chen, J., Jia, K., Yu, H., Wang, X.: New Impossible Differential Attacks of Reduced-Round Camellia-192 and Camellia-256. In: Parampalli and Hawkes [32], pp. 16–33
15. Clavier, C., Gaj, K. (eds.): Cryptographic Hardware and Embedded Systems - CHES 2009, 11th International Workshop, Lausanne, Switzerland, September 6-9, 2009, Proceedings, Lecture Notes in Computer Science, vol. 5747. Springer (2009)
16. Daemen, J., Knudsen, L.R., Rijmen, V.: The Block Cipher Square. In: Biham, E. (ed.) FSE. Lecture Notes in Computer Science, vol. 1267, pp. 149–165. Springer (1997)
17. Gong, Z., Nikova, S., Law, Y.W.: KLEIN: A New Family of Lightweight Block Ciphers. In: Juels, A., Paar, C. (eds.) RFIDSec. Lecture Notes in Computer Science, vol. 7055, pp. 1–18. Springer (2011)
18. Guo, J., Peyrin, T., Poschmann, A., Robshaw, M.J.B.: The LED Block Cipher. In: Preneel and Takagi [35], pp. 326–341
19. Hamburg, M.: Accelerating AES with Vector Permute Instructions. In: Clavier and Gaj [15], pp. 18–32
20. Hong, D., Sung, J., Hong, S., Lim, J., Lee, S., Koo, B., Lee, C., Chang, D., Lee, J., Jeong, K., Kim, H., Kim, J., Chee, S.: HIGHT: A New Block Cipher Suitable for Low-Resource Device. In: Goubin, L., Matsui, M. (eds.) CHES. Lecture Notes in Computer Science, vol. 4249, pp. 46–59. Springer (2006)
21. Kim, J., Hong, S., Sung, J., Lee, C., Lee, S.: Impossible Differential Cryptanalysis for Block Cipher Structures. In: Johansson, T., Maitra, S. (eds.) INDOCRYPT. Lecture Notes in Computer Science, vol. 2904, pp. 82–96. Springer (2003)
22. Knudsen, L.R., Leander, G., Poschmann, A., Robshaw, M.J.B.: PRINTcipher: A Block Cipher for IC-Printing. In: Mangard and Standaert [26], pp. 16–32
23. Leander, G., Paar, C., Poschmann, A., Schramm, K.: New Lightweight DES Variants. In: Biryukov [5], pp. 196–210
24. Lim, C.H., Korkishko, T.: mCrypton - A Lightweight Block Cipher for Security of Low-Cost RFID Tags and Sensors. In: Song, J., Kwon, T., Yung, M. (eds.) WISA. Lecture Notes in Computer Science, vol. 3786, pp. 243–258. Springer (2005)
25. Mace, F., Standaert, F.X., Quisquater, J.J.: ASIC Implementations of the Block Cipher SEA for Constrained Applications. Proceedings of the Third International Conference on RFID Security (2007), `http://www.rfidsec07.etsit.uma.es/confhome.htm`
26. Mangard, S., Standaert, F.X. (eds.): Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings, Lecture Notes in Computer Science, vol. 6225. Springer (2010)
27. Matsuda, S., Moriai, S.: Lightweight Cryptography for the Cloud: Exploit the Power of Bitslice Implementation. In: Prouff, E., Schaumont, P. (eds.) CHES. Lecture Notes in Computer Science, vol. 7428, pp. 408–425. Springer (2012)
28. Matsui, M.: Linear Cryptoanalysis Method for DES Cipher. In: Helleseth, T. (ed.) EUROCRYPT. Lecture Notes in Computer Science, vol. 765, pp. 386–397. Springer (1993)
29. Minematsu, K., Suzaki, T., Shigeri, M.: On Maximum Differential Probability of Generalized Feistel. In: Parampalli and Hawkes [32], pp. 89–105
30. Moradi, A., Poschmann, A., Ling, S., Paar, C., Wang, H.: Pushing the Limits: A Very Compact and a Threshold Implementation of AES. In: Paterson, K.G. (ed.) EUROCRYPT. Lecture Notes in Computer Science, vol. 6632, pp. 69–88. Springer (2011)
31. Özen, O., Varici, K., Tezcan, C., Çelebi Kocair: Lightweight Block Ciphers Revisited: Cryptanalysis of Reduced Round PRESENT and HIGHT. In: Boyd, C., Nieto, J.M.G. (eds.) ACISP. Lecture Notes in Computer Science, vol. 5594, pp. 90–107. Springer (2009)
32. Parampalli, U., Hawkes, P. (eds.): Information Security and Privacy - 16th Australasian Conference, ACISP 2011, Melbourne, Australia, July 11-13, 2011. Proceedings, Lecture Notes in Computer Science, vol. 6812. Springer (2011)
33. Poschmann, A.: Lightweight Cryptography - Cryptographic Engineering for a Pervasive World. Cryptology ePrint Archive, Report 2009/516 (2009), `http://eprint.iacr.org/`
34. Poschmann, A., Ling, S., Wang, H.: 256 Bit Standardized Crypto for 650 GE - GOST Revisited. In: Mangard and Standaert [26], pp. 219–233
35. Preneel, B., Takagi, T. (eds.): Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings, Lecture Notes in Computer Science, vol. 6917. Springer (2011)
36. Rinne, S.: Performance Analysis of Contemporary Light-Weight Cryptographic Algorithms on a Smart Card Microcontroller. SPEED – Software Performance Enhancement for Encryption and Decryption (2007), `http://www.hyperelliptic.org/SPEED/start07.html`

37. Rinne, S., Eisenbarth, T., Paar, C.: Performance Analysis of Contemporary Lightweight Block Ciphers on 8-bit Microcontrollers. SPEED-CC – Software Performance Enhancement for Encryption and Decryption and Cryptographic Compilers (2009), `http://www.hyperelliptic.org/SPEED/`
38. 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. Lecture Notes in Computer Science, vol. 5189, pp. 89–103. Springer (2008)
39. Satoh, A., Morioka, S., Takano, K., Munetoh, S.: A Compact Rijndael Hardware Architecture with S-Box Optimization. In: Boyd, C. (ed.) ASIACRYPT. Lecture Notes in Computer Science, vol. 2248, pp. 239–254. Springer (2001)
40. Shibutani, K., Isobe, T., Hiwatari, H., Mitsuda, A., Akishita, T., Shirai, T.: Piccolo: An Ultra-Lightweight Blockcipher. In: Preneel and Takagi [35], pp. 342–357
41. Shirai, T., Shibutani, K., Akishita, T., Moriai, S., Iwata, T.: The 128-Bit Blockcipher CLEFIA (Extended Abstract). In: Biryukov [5], pp. 181–195
42. Suzaki, T., Minematsu, K.: Improving the Generalized Feistel. In: Hong, S., Iwata, T. (eds.) FSE. Lecture Notes in Computer Science, vol. 6147, pp. 19–39. Springer (2010)
43. Suzaki, T., Minematsu, K., Morioka, S., Kobayashi, E.: TWINE: A Lightweight Block Cipher for Multiple Platforms. Pre-proceeding of SAC 2012 (2012)
44. Tsunoo, Y., Tsujihara, E., Shigeri, M., Saito, T., Suzaki, T., Kubo, H.: Impossible Differential Cryptanalysis of CLEFIA. In: Nyberg, K. (ed.) FSE. Lecture Notes in Computer Science, vol. 5086, pp. 398–411. Springer (2008)
45. Wu, W., Zhang, L.: LBlock: A Lightweight Block Cipher. In: Lopez, J., Tsudik, G. (eds.) ACNS. Lecture Notes in Computer Science, vol. 6715, pp. 327–344 (2011)
46. Zheng, Y., Matsumoto, T., Imai, H.: On the Construction of Block Ciphers Provably Secure and Not Relying on Any Unproved Hypotheses. In: Brassard, G. (ed.) CRYPTO. Lecture Notes in Computer Science, vol. 435, pp. 461–480. Springer (1989)