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,k-minematsu,s-morioka,e-kobayashi}@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 CPUs, from micro-controllers to high-end CPUs. This characteristic is obtained by the use of generalized Feistel combined with an improved block shuffle, introduced at FSE 2010.

Keywords: lightweight block cipher, generalized Feistel, 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, it is often inappropriate for such small devices due to their size/power/memory constraints, even though there are constant efforts for small-footprint AES, e.g., [13,30,39]. To fill the gap, many hardware-oriented lightweight block ciphers have been recently proposed, e.g., [8,12,17,18,20,22,23,26,40,44], and more.

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-i series). For this purpose, we avoid the hardware-oriented design options, most notably a bit permutation, and build a block cipher using 4-bit components.

Design. Specifically, we employ Type-2 generalized Feistel structure [45], 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 quite many rounds. To overcome the problem, we employ the idea of Suzaki and Minematsu at FSE '10 [42] which substantially improves diffusion 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. The features of TWINE are (1) no bit permutation, (2) generalized Feistel-based, and (3) no Galois-Field matrix. The components are only one 4-bit S-box, XOR, and 4-bit-wise permutation (shuffle). As far as we know, this is the first attempt that unifies these three features. There is a predecessor called LBlock [44] which has some resemblances to ours, however TWINE is an independent work and has several concrete design advantages (See Section 3).

Implementation. We implemented TWINE on hardware and software. Our hardware implementations suggest that the encryption-only TWINE can be implemented with 1,503 Gate Equivalent (GE), and a serialized implementation results in 1,011 GEs using a shared sbox architecture. For both cases, we did not consider the hard-wired key or special key signaling (as employed by [40]). These figures are comparable to, or even better than, the leading hardware-oriented proposals, in particular when a standard key treatment is required.

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 doing 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]), and we find that VPI extremely works fine with TWINE. For example, on Intel Core-i5 U560 we observed 4.75 cycles/byte[1] using VPI called `pshufb`. This figure is quite impressive in the realm of (lightweight) block ciphers. For reference, we observed that AES using VPI [19] 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. TWINE's well-balanced performance under multiple platforms makes it suitable to heterogeneous networks, consisting of (e.g.) a huge number of tiny sensor nodes which independetly encrypt sensor information and one server computer which performs the information aggregation and decryption.

Security. As TWINE is a variant of GFS it is definitely important to evaluate the security against attacks suitable to GFS, such as the impossible differential cryptanalysis (IDC) and the saturation cryptanalysis (SC). We perform a thorough analysis (as a new cipher proposal) on TWINE including IDC and SC, and present IDC against 23-round TWINE-80 and 24-round TWINE-128 as the most powerful attacks we have found so far. The attack is fully exploits the key schedule, and 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

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 $|x| = m$,

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

we may write $x_{(m)}$ to emphasize its bit length. If $|x| = 4c$ for a positive integer c, we write $x \to (x_0\|x_1\|\ldots\|x_{c-1})$, where $|x_i| = 4$, is the partition operation into the 4-bit sub-blocks. The opposite operation, $(x_0\|x_1\|\ldots\|x_{c-1}) \to x$, 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.

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 [41,45] 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 permutes the 16 blocks. Unlike original Type-2 GFS, the diffusion layer is not a cyclic shift and is chosen to provide a better diffusion than the cyclic shift from the result of [42]. 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 encryption process is written as Algorithm 2.1.

The data processing part essentially consists of a 4-bit S-box, denoted by S, and a permutation of block indexes, $\pi : \{0, \ldots, 15\} \to \{0, \ldots, 15\}$, where j-th sub-block is mapped to $\pi[j]$-th sub-block. The figure of the round function is in Fig. 1. The decryption of TWINE uses the same S-box and key schedule as used in the encryption, with the inverse block shuffle. See Algorithm 2.2.

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 (the same as one used at the data processing). The 80-bit key schedule uses 6-bit round constants, $\mathrm{CON}^i_{(6)} = \mathrm{CON}^i_{H(3)}\|\mathrm{CON}^i_{L(3)}$ for $i = 1$ to 35, and $Rot\mathbf{z}(x)$ means z-bit left cyclic shift of x. Its pseudocode is in Algorithm 2.3. For 128-bit key, see Appendix A. We remark that CON^i corresponds to 2^i in GF(2^6) with primitive polynomial $z^6 + z + 1$.

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

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

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

Algorithm 2.2: TWINE.Dec($C_{(64)}, RK_{(32\times 36)}, P_{(64)}$)

```
$X^{36}_{0(4)} \| X^{36}_{1(4)} \| \ldots \| X^{36}_{14(4)} \| X^{36}_{15(4)} \leftarrow C$,   $\mathrm{RK}^{1}_{(32)} \| \ldots \| \mathrm{RK}^{36}_{(32)} \leftarrow RK_{(32\times 36)}$
for $i \leftarrow 36$ to 2
do {
    $\mathrm{RK}^{i}_{0(4)} \| \mathrm{RK}^{i}_{1(4)} \| \ldots \| \mathrm{RK}^{i}_{6(4)} \| \mathrm{RK}^{i}_{7(4)} \leftarrow \mathrm{RK}^{i}_{(32)}$
    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   $X^{i-1}_{\pi^{-1}[h]} \leftarrow X^{i}_{h}$
}
for $j \leftarrow 0$ to 7 do   $X^{1}_{2j+1} \leftarrow S(X^{1}_{2j} \oplus \mathrm{RK}^{1}_{j}) \oplus X^{1}_{2j+1}$
$P \leftarrow X^{1}_{0} \| X^{1}_{1} \| \ldots \| X^{1}_{14} \| X^{1}_{15}$
```

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

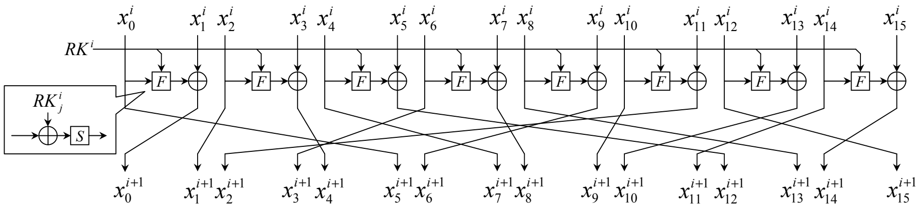

Fig. 1. Round function of TWINE

Algorithm 2.3: TWINE.KeySchedule-80($K_{(80)}, RK_{(32\times 36)}$)

```
$\mathrm{WK}_{0(4)} \| \mathrm{WK}_{1(4)} \| \ldots \| \mathrm{WK}_{18(4)} \| \mathrm{WK}_{19(4)} \leftarrow K$
for $r \leftarrow 1$ to 35
do {
    $\mathrm{RK}^{r}_{(32)} \leftarrow \mathrm{WK}_1 \| \mathrm{WK}_3 \| \mathrm{WK}_4 \| \mathrm{WK}_6 \| \mathrm{WK}_{13} \| \mathrm{WK}_{14} \| \mathrm{WK}_{15} \| \mathrm{WK}_{16}$
    $\mathrm{WK}_1 \leftarrow \mathrm{WK}_1 \oplus S(\mathrm{WK}_0)$,   $\mathrm{WK}_4 \leftarrow \mathrm{WK}_4 \oplus S(\mathrm{WK}_{16})$
    $\mathrm{WK}_7 \leftarrow \mathrm{WK}_7 \oplus 0 \| CON^{r}_{H}$,   $\mathrm{WK}_{19} \leftarrow \mathrm{WK}_{19} \oplus 0 \| CON^{r}_{L}$
    $\mathrm{WK}_0 \| \cdots \| \mathrm{WK}_3 \leftarrow Rot4(\mathrm{WK}_0 \| \cdots \| \mathrm{WK}_3)$
    $\mathrm{WK}_0 \| \cdots \| \mathrm{WK}_{19} \leftarrow Rot16(\mathrm{WK}_0 \| \cdots \| \mathrm{WK}_{19})$
}
$\mathrm{RK}^{36}_{(32)} \leftarrow \mathrm{WK}_1 \| \mathrm{WK}_3 \| \mathrm{WK}_4 \| \mathrm{WK}_6 \| \mathrm{WK}_{13} \| \mathrm{WK}_{14} \| \mathrm{WK}_{15} \| \mathrm{WK}_{16}$
$RK \leftarrow \mathrm{RK}^1 \| \mathrm{RK}^2 \| \ldots \| \mathrm{RK}^{35} \| \mathrm{RK}^{36}$
```

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	

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 multiple CPUs, from low-end microcontroller to general-purpose 32/64-bit CPU.

On LBlock. We remark that LBlock [44], 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 the authors of [44] 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 TWINE. Note that this choice is reasonable from Table 6 of [42], as it satisfies both of the fastest diffusion and the highest immunities against linear and differential attacks among other block shuffles.

Nevertheless, there are important differences between TWINE and LBlock[2]. First, LBlock uses ten distinct S-boxes while TWINE uses single S-box. TWINE's design contributes to a compact serialized hardware and fast software (indeed, our fast SIMD implementation was impossible if multiple S-boxes were used). Second, LBlock uses a bit permutation in its key scheduling, which decreases software efficiency.

3.2 Parameters and Components

Rounds. 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. Employing the same 36-round for TWINE-80 may look slight odd, however, it enables various multiple-round hardware implementations with a small overhead as 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 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" [42]. Our π is such one[3] 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, using π of Algorithm 2.1 they can be reduced to 14 and 15 rounds.

There exist multiple optimum block shuffles [42]. Hence π was chosen considering other aspects which is not (directly) related to DRmax. In particular, we

[2] We also would like to point out that the security evaluation of LBlock is insufficient. We already found a saturation attack against 22-round LBlock without considering the key schedule, thus the security margin is smaller than the claimed by the authors (20-round), though a recent work [25] shows a 21-round impossible differential attack.

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

chose π considering the the number of differentially and linearly active S-boxes (See Table 1 in Section 4).

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, 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) (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 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 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 [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. Due to the page limit, 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 in our evaluation. The results on other attacks, such as differential and linear attacks, will also be briefly described.

In this section, we use the notations X^i_j and RK^i_j following Algorithm 2.1, and define $F^i_j(x) \stackrel{\text{def}}{=} S(\mathrm{RK}^i_j \oplus x)$ for $i = 1, \ldots, 36$, $j = 0, \ldots, 15$, and denote $F^i_j(x) \oplus F^i_j(x \oplus \delta)$ by $F^i_j(\delta)$. For any symbol S let $\bar{\mathsf{S}}^k$ denote the sequence of k symbols, e.g. $\bar{0}^3$ means $(0, 0, 0)$ and $\bar{A}^3$ means (A, A, A).

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, 43]. We searched impossible differential characteristics (IDCs) using Kim et al.'s method [21], and found 64 14-round IDCs

$$(0,\alpha_0,0,\alpha_1,0,\alpha_2,0,\alpha_3,0,\alpha_4,0,\alpha_5,0,\alpha_6,0,\alpha_7) \overset{14r}{\not\to} (\beta_0,0,\beta_1,0,\beta_2,0,\beta_3,0,\beta_4,0,\beta_5,0,\beta_6,0,\beta_7,0), \tag{1}$$

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 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 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

$$(p_1, p_2, 0, p_3, \bar{0}^6, p_4, p_5, p_6, p_7, 0, p_0), \text{ where } p_i \in \{0,1\}^4 \text{ is non-zero.} \tag{2}$$

We want the 4-round output pairs to be compliant with the left hand side of Eq. (1) with $\alpha_0 \neq 0$ and other α_is being zero. Hence plaintext pairs having no chance to do that are discarded. Here, the property of S-box shows that for any non-zero p_x, $F^i_j(p_x)$ is one of 7 possible values, depending on RK^i_j and p_x. Using this property we identify $2^{54.56}$ plaintext pairs of Eq. (2) that have a chance. Then we encrypt such plaintext pairs and search the ciphertext pairs having the difference

$$(0, c_1, 0, c_2, c_3, c_4, c_0, c_5, c_6, c_7, c_8, c_9, c_{10}, c_{11}, 0, 0), \tag{3}$$

where all c_is are non-zero 4-bit values. We prepare $2^{29.55}$ structures and obtain $2^{68.11}$ ciphertext pairs of the difference Eq. (3) out of all $2^{84.11}$ ciphertext pairs.

Key Elimination. For each ciphertext pair satisfying Eq. (3), we try to eliminate the wrong guesses for the 80-bit (sub)key vector $(\mathcal{K}_1 \| \mathcal{K}_2 \| \mathcal{K}_3)$, where $|\mathcal{K}_1| = 20$, $|\mathcal{K}_2| = 52$, $|\mathcal{K}_3| = 8$ and $\mathcal{K}_1 = (\mathrm{RK}^1_{[1,2,3,7]}, \mathrm{RK}^{23}_0)$, $\mathcal{K}_2 = (\mathrm{RK}^1_{[0,5,6]}, \mathrm{RK}^2_{[2,4,6,7]}, \mathrm{RK}^{23}_{[2,4,5]}, \mathrm{RK}^{22}_{[1,3,4]})$ and $\mathcal{K}_3 = (\mathrm{RK}^{22}_{[0,2]})$ (here $\mathrm{RK}^i_{[a,b,c]}$ denotes $\mathrm{RK}^i_a \| \mathrm{RK}^i_b \| \mathrm{RK}^i_c$). First, we guess $\mathcal{K}_1$ (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 the property of S-box mentioned above. Once $\mathcal{K}_1$ and $\mathcal{K}_2$ have been fixed, each RK^i_j in $\mathcal{K}_3$ will have $(2 \cdot 6 + 4)/15 \approx 1.07$ candidates, as we have no restrictions on the input difference for Fs relating to these subkeys. From this observation, we expect to eliminate $2^{20} \cdot 2.28^{13} \cdot 1.07^2 \approx 2^{35.69}$ candidates from a set of 2^{80} values for each plaintext-ciphertext pair. In other words, the wrong subkey is eliminated with probability $2^{-44.31}$.

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.

4.3 Saturation Attack

Saturation attack [16] is also a powerful attack against GFS-based ciphers. The attack traces the set of variables $(\mathsf{S}_0, \ldots, \mathsf{S}_{15})$, where S_k denotes the saturation status of k-th nibble which is one of the followings:

$$\begin{array}{llll} \textbf{Constant } (C): \forall i,j, & X_i = X_j & \textbf{All } (A) & : \forall i \neq j,\ X_i \neq X_j \\ \textbf{Balance } (B) & : \bigoplus_i X_i = 0 & \textbf{Unknown } (U) & : \text{Others} \end{array}$$

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 and the t-round states. If we have $\alpha_i = A$ and $\beta_j \neq U$ for some i and j with probability 1 (i.e. for all keys), $\alpha \overset{tr}{\to} \beta$ is said to be an t-round saturation characteristic (SC). TWINE has 15-round SC with α consisting of one C and fifteen As and β contains 4 Bs (the remainings are U), for example;

$$(\bar{A}^{12}, C, \bar{A}^3) \overset{15r}{\to} (\bar{U}^3, B, \bar{U}^5, B, \bar{U}^3, B, U, B), \text{ and} \tag{4}$$

$$(\bar{A}^6, C, \bar{A}^9) \overset{15r}{\to} (U, B, \bar{U}^3, B, U, B, \bar{U}^3, B, \bar{U}^4). \tag{5}$$

Suppose we use SC of Eq. (5) to break 22-round TWINE-80. We recover 108-bit subkey. From the key schedule, the actual subkey bits needed to be guessed are 72 bits. First we encrypt a set of 2^{60} plaintexts (called S-structure) induced from the left hand side of Eq. (5), and obtain a set of 2^{60} ciphertexts. Now X_j^i has 2^{60} variations for each i, j, and we let $\oplus X_j^i$ to denote the sum of these 2^{60} variations. We also define $F_j^i out$ as $F_j^i(X_j^i)$ and define $\oplus F_j^i out$ analogously. Next we calculate $\oplus X_0^{17}$ and $\oplus F_0^{16} out$ for each 108-bit subkey candidate. Here X_0^{17} is uniquely determined by a certain 40 subkey bits (out of 108 bits). Similarly $F_0^{16} out$ is determined by a certain 60 subkey bits, and the intersection is 28 bits (thus we need 72-bit search). The computation of $\oplus F_0^{16} out$ requires $2^{73.80}$ F evaluations (amount to $2^{66.34}$ encryptions of 22-round TWINE). For any subkey guess if $\oplus X_0^{17}$ equals to $\oplus F_0^{16} out$ the saturation status of $\oplus X_1^{16}$ is B. If not, then the guess is wrong and thus eliminated. As this elimination is expected to occur with probability $1 - 1/2^4$, we can reduce the number of subkey candidates from 2^{72} to 2^{68} for one S-structure. With additional 8-bit key guess, the master key is recovered. 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 X_0^{17}, which is negligible) encryptions. We can further reduce the time complexity by using multiple S-structures. Using 4 structures, we can attack 22-round TWINE-80 with the data, time and memory complexity being 2^{62} blocks, $2^{68.43}$ encryptions and 2^{67} blocks respectively.

In a similar manner (using SC of Eq, (5)), 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.

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 1). 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 ($\varDelta$) and linear ($\varGamma$) 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 $\varDelta$ ($\varGamma$). They involve 30 active S-boxes, and thus the characteristic probability is 2^{-60}.

$$\begin{aligned} \varDelta &= (\bar{0}^9, 1, 0, 1, 0, 1, 0, 0) \overset{14r}{\to} (\bar{0}^3, 1, \bar{0}^4, 1, 0, 0, 1, 0, 0, 1, 1), \\ \varGamma &= (\bar{0}^6, 1, 1, \bar{0}^3, 1, 0, 0, 1, 1) \overset{14r}{\to} (\bar{0}^9, 1, \bar{0}^3, 1, 0, 1). \end{aligned} \tag{6}$$

Compared to the impossible differential attack, we naturally 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 imply the more attackable rounds in the key guessing.

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

Table 1. 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 2 shows the search result, where $\varDelta$KS, $\varDelta$RK, $\varDelta X$ 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

Table 2. 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					

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.

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 3 with a comparison. Note that for some algorithms other than TWINE, the synthesis was not done at 100KHz, hence we estimated the throughput by scaling. Table 4 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.

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. For encryption-only TWINE-80, it achieved 1,011 GEs. We are still working on it, and the details will be given in the near future.

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

Table 3. 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	563	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 [26]	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.

Table 4. Component sizes of TWINE-80 encryption

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

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. All versions precompute the round keys, i.e. they do not use an on-the-fly key schedule.

In the speed-first version, two rounds are processed in one loop. This removes the block shuffle between the first and second rounds. RAM load (LD) is faster than ROM load (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.

Table 5. 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).

Table 5 shows comparison of TWINE and other lightweight block ciphers. We list the (scaled) throughput/code ratio for a performance measure (See Table 5 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). Examples of VPI are, `vperm` in Motorola AltiVec, `pshufb` in Intel SSE (SSSE3), and `vtbl` in ARM NEON. The power of VPI was first presented by Hamburg [19] for AES, and then 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. In our VPI-based code, we transform TWINE into an equivalent form shown by the left of Table 2. This form cyclically invokes 4 different shuffles (called half shuffle) on 8 nibbles. Here, "index of RK" denotes the index of round key, RK, given to the round function (from left to right).

For 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 (see the right of Table 2). Here, the left (right) half of input data is in `xmm0`, (`xmm1`), and `eax` contains the address of round key. 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 6 shows the result, where x/y denotes x encryption speed and y decryption speed in cycles per byte. We also implemented VPI-based AES [19] and (popular) T-table AES and measured their performance figures. We observe single-block TWINE is comparable to VPI-based AES, and double-block TWINE is even faster. The key schedule for 80-bit (128-bit) key spends about 200 (290) cycles on Core i7 2600S.

round	index of RK	half shuffle
$4i+1$	$0,1,2,3,4,5,6,7$	$[1,0,4,5,2,3,7,6]$
$4i+2$	$0,2,6,4,3,1,5,7$	$[5,3,7,1,6,0,4,2]$
$4i+3$	$0,6,5,3,4,2,1,7$	$[6,7,3,2,5,4,0,1]$
$4i+4$	$0,5,1,4,3,6,2,7$	$[2,4,0,6,1,7,3,5]$

```
movdqa xmm2,[eax]   : load RK
pxor   xmm2,xmm0    : ⊕ RK
movdqa xmm3,[sbox]  : load S-box
pshufb xmm3,xmm2    : apply S-box
pxor   xmm1,xmm3    : ⊕ S-box out
pshufb xmm0,[sh]    : half shuffle
```

Fig. 2. (Left) 4-round structure for SIMD-based implementation. (Right) A code of round function.

Table 6. 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 for many valuable comments. 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 1999. LNCS, vol. 1592, pp. 12–23. Springer, Heidelberg (1999)

4. Biham, E., Shamir, A.: Differential cryptanalysis of the data encryption standard. Springer, London (1993)
5. Biryukov, A. (ed.): FSE 2007. LNCS, vol. 4593. Springer, Heidelberg (2007)
6. Biryukov, A., Nikolić, 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 2010. LNCS, vol. 6110, pp. 322–344. Springer, Heidelberg (2010)
7. Biryukov, A., Wagner, D.: Slide Attacks. In: Knudsen, L.R. (ed.) FSE 1999. LNCS, vol. 1636, pp. 245–259. Springer, Heidelberg (1999)
8. Bogdanov, A.A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M., Seurin, Y., Vikkelsoe, C.: PRESENT: An Ultra-Lightweight Block Cipher. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 450–466. Springer, Heidelberg (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, Gaj (eds.) [15], pp. 272–288
13. Canright, D.: A Very Compact S-Box for AES. In: Rao, J.R., Sunar, B. (eds.) CHES 2005. LNCS, vol. 3659, pp. 441–455. Springer, Heidelberg (2005)
14. Chen, J., Jia, K., Yu, H., Wang, X.: New Impossible Differential Attacks of Reduced-Round Camellia-192 and Camellia-256. In: Parampalli, Hawkes (eds.) [32], pp. 16–33
15. Clavier, C., Gaj, K. (eds.): CHES 2009. LNCS, vol. 5747. Springer, Heidelberg (2009)
16. Daemen, J., Knudsen, L.R., Rijmen, V.: The Block Cipher SQUARE. In: Biham, E. (ed.) FSE 1997. LNCS, vol. 1267, pp. 149–165. Springer, Heidelberg (1997)
17. Gong, Z., Nikova, S., Law, Y.W.: KLEIN: A New Family of Lightweight Block Ciphers. In: Juels, A., Paar, C. (eds.) RFIDSec 2011. LNCS, vol. 7055, pp. 1–18. Springer, Heidelberg (2012)
18. Guo, J., Peyrin, T., Poschmann, A., Robshaw, M.J.B.: The LED Block Cipher. In: Preneel, Takagi (eds.) [35], pp. 326–341
19. Hamburg, M.: Accelerating AES with Vector Permute Instructions. In: Clavier, Gaj (eds.) [15], pp. 18–32
20. Hong, D., Sung, J., Hong, S.H., Lim, J.-I., Lee, S.-J., Koo, B.-S., Lee, C.-H., Chang, D., Lee, J., Jeong, K., Kim, H., Kim, J.-S., Chee, S.: HIGHT: A New Block Cipher Suitable for Low-Resource Device. In: Goubin, L., Matsui, M. (eds.) CHES 2006. LNCS, vol. 4249, pp. 46–59. Springer, Heidelberg (2006)
21. Kim, J.-S., Hong, S.H., Sung, J., Lee, S.-J., Lim, J.-I., Sung, S.H.: Impossible Differential Cryptanalysis for Block Cipher Structures. In: Johansson, T., Maitra, S. (eds.) INDOCRYPT 2003. LNCS, vol. 2904, pp. 82–96. Springer, Heidelberg (2003)
22. Knudsen, L.R., Leander, G., Poschmann, A., Robshaw, M.J.B.: PRINTcipher: A Block Cipher for IC-Printing. In: Mangard, Standaert (eds.) [27], pp. 16–32

23. Leander, G., Paar, C., Poschmann, A., Schramm, K.: New Lightweight DES Variants. In: Biryukov (ed.) [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.-S., Kwon, T., Yung, M. (eds.) WISA 2005. LNCS, vol. 3786, pp. 243–258. Springer, Heidelberg (2006)
25. Liu, Y., Gu, D., Liu, Z., Li, W.: Impossible Differential Attacks on Reduced-Round LBlock. In: Ryan, M.D., Smyth, B., Wang, G. (eds.) ISPEC 2012. LNCS, vol. 7232, pp. 97–108. Springer, Heidelberg (2012)
26. 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.html`
27. Mangard, S., Standaert, F.-X. (eds.): CHES 2010. LNCS, vol. 6225. Springer, Heidelberg (2010)
28. Matsui, M.: Linear Cryptanalysis Method for DES Cipher. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 386–397. Springer, Heidelberg (1994)
29. Minematsu, K., Suzaki, T., Shigeri, M.: On Maximum Differential Probability of Generalized Feistel. In: Parampalli, Hawkes (eds.) [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 2011. LNCS, vol. 6632, pp. 69–88. Springer, Heidelberg (2011)
31. Özen, O., Varıcı, K., Tezcan, C., Kocair, Ç.: Lightweight Block Ciphers Revisited: Cryptanalysis of Reduced Round PRESENT and HIGHT. In: Boyd, C., González Nieto, J. (eds.) ACISP 2009. LNCS, vol. 5594, pp. 90–107. Springer, Heidelberg (2009)
32. Parampalli, U., Hawkes, P. (eds.): ACISP 2011. LNCS, vol. 6812. Springer, Heidelberg (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, Standaert (eds.) [27], pp. 219–233
35. Preneel, B., Takagi, T. (eds.): CHES 2011. LNCS, vol. 6917. Springer, Heidelberg (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 2008. LNCS, vol. 5189, pp. 89–103. Springer, Heidelberg (2008)
39. Satoh, A., Morioka, S., Takano, K., Munetoh, S.: A Compact Rijndael Hardware Architecture with S-Box Optimization. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 239–254. Springer, Heidelberg (2001)
40. Shibutani, K., Isobe, T., Hiwatari, H., Mitsuda, A., Akishita, T., Shirai, T.: Piccolo: An Ultra-Lightweight Blockcipher. In: Preneel, Takagi (eds.) [35], pp. 342–357

41. Shirai, T., Shibutani, K., Akishita, T., Moriai, S., Iwata, T.: The 128-Bit Blockcipher CLEFIA (Extended Abstract). In: Biryukov (ed.) [5], pp. 181–195
42. Suzaki, T., Minematsu, K.: Improving the Generalized Feistel. In: Hong, S., Iwata, T. (eds.) FSE 2010. LNCS, vol. 6147, pp. 19–39. Springer, Heidelberg (2010)
43. Tsunoo, Y., Tsujihara, E., Shigeri, M., Saito, T., Suzaki, T., Kubo, H.: Impossible Differential Cryptanalysis of CLEFIA. In: Nyberg, K. (ed.) FSE 2008. LNCS, vol. 5086, pp. 398–411. Springer, Heidelberg (2008)
44. Wu, W., Zhang, L.: LBlock: A Lightweight Block Cipher. In: Lopez, J., Tsudik, G. (eds.) ACNS 2011. LNCS, vol. 6715, pp. 327–344. Springer, Heidelberg (2011)
45. 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 1989. LNCS, vol. 435, pp. 461–480. Springer, Heidelberg (1990)

A Key Schedule for 128-Bit Key

```
Algorithm A.1: TWINE.KeySchedule-128($K_{(128)}, RK_{(32\times36)}$)

$\mathrm{WK}_{0(4)}\|\mathrm{WK}_{1(4)}\|\ldots\|\mathrm{WK}_{30(4)}\|\mathrm{WK}_{31(4)} \leftarrow K$
for $r \leftarrow 1$ to $35$
    do
        $\mathrm{RK}^r_{(32)} \leftarrow \mathrm{WK}_2\|\mathrm{WK}_3\|\mathrm{WK}_{12}\|\mathrm{WK}_{15}\|\mathrm{WK}_{17}\|\mathrm{WK}_{18}\|\mathrm{WK}_{28}\|\mathrm{WK}_{31}$
        $\mathrm{WK}_1 \leftarrow \mathrm{WK}_1 \oplus S(\mathrm{WK}_0), \mathrm{WK}_4 \leftarrow \mathrm{WK}_4 \oplus S(\mathrm{WK}_{16}),$
        $\mathrm{WK}_{23} \leftarrow \mathrm{WK}_{23} \oplus S(\mathrm{WK}_{30})$
        $\mathrm{WK}_7 \leftarrow \mathrm{WK}_7 \oplus 0\|CON^r_H, \mathrm{WK}_{19} \leftarrow \mathrm{WK}_{19} \oplus 0\|CON^r_L$
        $\mathrm{WK}_0\|\cdots\|\mathrm{WK}_3 \leftarrow Rot4(\mathrm{WK}_0\|\cdots\|\mathrm{WK}_3)$
        $\mathrm{WK}_0\|\cdots\|\mathrm{WK}_{31} \leftarrow Rot16(\mathrm{WK}_0\|\cdots\|\mathrm{WK}_{31})$
$\mathrm{RK}^{36}_{(32)} \leftarrow \mathrm{WK}_2\|\mathrm{WK}_3\|\mathrm{WK}_{12}\|\mathrm{WK}_{15}\|\mathrm{WK}_{17}\|\mathrm{WK}_{18}\|\mathrm{WK}_{28}\|\mathrm{WK}_{31}$
$\mathrm{RK}_{(32\times36)} \leftarrow \mathrm{RK}^1\|\mathrm{RK}^2\|\ldots\|\mathrm{RK}^{35}\|\mathrm{RK}^{36}$
```

B 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