KLEIN: A New Family of Lightweight Block Ciphers

Zheng Gong[1], Svetla Nikova[1,2] and Yee-Wei Law[3]

[1]Faculty of EWI, University of Twente, The Netherlands
{z.gong, s.nikova}@utwente.nl

[2] Dept. ESAT/SCD-COSIC, Katholieke Universiteit Leuven, Belgium
[3] Department of EEE, The University of Melbourne, Australia
yee.wei.law@gmail.com

Abstract

Resource-efficient cryptographic primitives become fundamental for realizing both security and efficiency in embedded systems like RFID tags and sensor nodes. Among those primitives, lightweight block cipher plays a major role as a building block for security protocols. In this paper, we describe a new family of lightweight block ciphers named KLEIN, which is designed for resource-constrained devices such as wireless sensors and RFID tags. Compared to the related proposals, KLEIN has advantage in the software performance on legacy sensor platforms, while in the same time its hardware implementation can also be compact.

Key words. Block cipher, Wireless sensor network, Low-resource implementation.

1 Introduction

With the development of wireless communication and embedded systems, we become increasingly dependent on the so called *pervasive computing*; examples are smart cards, RFID tags, and sensor nodes that are used for public transport, pay TV systems, smart electricity meters, anti-counterfeiting, etc. In particular, wireless sensor networks (WSNs) have attracted more and more attention since their promising applications like environment monitoring, military scouting and healthcare. On those resource-limited devices the choice of security algorithms should be very careful by consideration of the implementation costs. Symmetric-key algorithms, especially block ciphers, still play an important role for the security of the embedded systems. Moreover recent results have shown that a lightweight block cipher can be used not only for encryption, but also for hash [3] and authentication [18] on devices with highly constrained resources. For security and performance concerns, typically sensors are equipped with hardware implementation of AES-128 [11], e.g. the Chipcon CC2420 transceiver chip [4]. But for resource-constrained devices, AES could be too expensive, despite the various approaches that have been proposed to reduce the costs of AES hardware and software implementations [17, 21, 26].

In the literature, quite a few lightweight block ciphers with various design strategies have been proposed [2, 10, 14, 20, 28, 30, 35]. Skipjack is a lightweight block cipher designed by the U.S. National Security Agency (NSA) for embedded applications [35]. The algorithm of Skipjack has an 80-bit key with a 64-bit block length based on an unbalanced Feistel network.

NOEKEON is a hardware-efficient block cipher, which is proposed by Daemen *et al.* [10] and submitted to the NESSIE project at 2000. HIGHT was designed by Hong *et al.* [20] as a generalized Feistel-like cipher, which is suitable for low-resource devices. mCrypton [30] is designed by following the overall architecture of Crypton [29] but with redesign and simplifications of each component function to enable much compact implementation in both hardware and software. At FSE'07, Leander *et al.* [28] proposed a family of new lightweight variants of DES, which are called DESL\DESX\DESXL. The main idea of the new variants of DES is to use just one S-box recursively, instead of eight different S-boxes. Bogdanov *et al.* proposed [2] an ultra-lightweight block cipher which is called PRESENT. The design of PRESENT is extremely hardware efficient, since it uses a fully wired diffusion layer without any algebraic unit. KATAN and KTANTAN are designed as a family of ultra-lightweight block ciphers by De Cannière *et al.* [14]. Both KATAN and KTANTAN use an 80-bit key length with 32, 48, or 64-bit block size, while KTANTAN is more compact in hardware since its key will be unchangeably burnt on devices. In [16], Engels *et al.* proposed a novel ultra- lightweight cryptographic algorithm with 256-bit key length and 16-bit block size, referred to as Hummingbird, for resource-constrained devices.

The security of any block cipher should be extensively analyzed before its wide implementation. Biham *et al.* have discovered an impossible differential attack on 31 of the 32 rounds [1] of Skipjack. A truncated differential attack was also published against 28 rounds of Skipjack by Knudsen *et al.* [24]. In a NESSIE report, Knudsen and Raddum [23] showed that "indirect mode" NOEKEON was still vulnerable to certain peculiar kinds of related-key cryptanalysis, and discovered weaknesses in NOEKEON-variant ciphers which cast doubt on the design strategy behind NOEKEON and thus on its security. As a result NOEKEON was not selected by NESSIE. Although PRESENT has a hardware-efficient diffusion layer, different attacks have been applied to PRESENT due to its diffusion property, e.g. the weak key attack [36, 37], the linear attack [5] and the saturation attack [6]. It remains a challenging work to design a lightweight block cipher which has a good security margin and resists all known attacks.

The performance of a block cipher is also an important factor for resource-constrained devices. Most of the lightweight proposals claim their efficiency in hardware. The area in *Gate equivalents* (GE) is often used as a measure for the compactness of the hardware implementation. Generally speaks, one GE is equal to the area which is required by two-input NAND gate with the lowest driving strength of the appropriate technology [38]. PRESENT, for example, has a compact implementation with 1570 GE in a 64-bit width datapath [3], as well as an very lightweight implementation with 1000 GE [39]. mCrypton and DESXL are also competitive since they are close to the 2000 GE barrier. HIGHT is less attractive since its area in GE is over 3000 GE, which is close to the best AES implementation [19] (with 3100 GE) and [17] (with 3400 GE). KATAN and KTANTAN are the most hardware-efficient block ciphers which require less than 1000 GE [14].

Usually, sensors have better power and hardware capabilities than RFID tags. Since software implementations have the advantages of flexibility and economy in manufacture and maintenance, it is believed that software-efficient block ciphers are more practical for sensors. In this paper, a new family of block ciphers called *KLEIN* is designed for resource-constrained devices. Compared to the related proposals, KLEIN has the advantage of the software performance on legacy sensor platforms and at the same time its hardware implementation can also be compact. Our security analysis shows that KLEIN has a conservative security margin against various cryptanalyses.

The remainder of this paper is organized as follows. Section 2 and 3 describe the design rationale and the specification of the KLEIN family. In Section 4, the security of KLEIN is analyzed by considering known attacks. Compared to related lightweight proposals, a detailed performance of KLEIN is discussed in Section 5. Section 6 concludes the paper.

2 Design Rationale

In this section we discuss the design rationales and the choices we have made when designing KLEIN families.

Key Length. KLEIN is a family of block ciphers, with a fixed 64-bit block size and variable key length - 64, 80 or 96-bits. According to the different key length, we will denote the ciphers by KLEIN-64/80/96, respectively. It is well-known that the key length and the block size are two important factors for a block cipher in the trade-offs between security and performance. Considering the performances in low-resource implementations, key registers and intermediate results have a significant effect on its footprint. Moreover, in ubiquitous computing, data flows are unlikely to be a high-speed throughput, which means a large block size or key length might be unnecessarily for data encryption and authentication. For security concerns, 64-bit key length might be vulnerable if one considers attack models based on pre-computation and large amounts of available storage. We recommend KLEIN-64 to be used for constructing single (double) block length hash functions or message authentication codes and KLEIN-80 and KLEIN-96 to be used for data encryption in any of the operation modes.

Optimal Platform. In practice, most of lightweight block ciphers are hardware-oriented, which is necessary in low-end devices such as smart cards and RFID tags. Compared to hardware-oriented designs, software implementations have more flexibility and lower costs on manufacturing and maintaining. The manufacturing costs are cut down to the minimum if the underlying platform can support the computation and the memory of a software implementation. For software-oriented implementations, once a lightweight block cipher is implemented, then it can be used for constructing hash functions and message authentication codes. Later in case necessary the implementation can be easily refined by updating the software. Hardware implementations are usually preferable in applications where high-speed throughputs or constrained resources are imposed. Otherwise, e.g. for legacy sensors such as IRIS [7] and TelosB [8], software implementations are recommended. This arguments motivate us to work on a software-oriented design, which will be more suitable for sensors. However, KLEIN can also be efficiently implemented in hardware. KLEIN remains lightweight in both hardware and software implementations as we will demonstrate later in the paper.

Critical Threats. First of all, a block cipher should be secure against conventional cryptanalyses, such as differential and linear attacks. Besides the conventional cryptanalyses, a good design for resource-constrained devices should also counteract related-key and side-channel attacks. Usually low-resource applications cannot afford a truly random, or a complicated pseudorandom number generator for initializing keys. Furthermore, hardware implementations of block ciphers are still vulnerable to side-channel attacks. Even if a block cipher is secure against mathematical attacks, attackers may recover keys from side-channel information, e.g. differential power analysis from a compromised sensor node. The extra costs paid on the protection of hardware attacks, e.g. the masking and re-keying techniques, might be acceptable in normal environment. But for low-resource implementations, the protection costs should be as small as possible. The side-channel leakage of masked CMOS gates [31] exploits that it is not a trivial work to provide a secure hardware implementation. KLEIN has a well balanced key schedule with respect to resistance against related-key attacks and the agility of the keys. Also the secret sharing method for the resistance of side-channel attacks [34] will be discussed in our design.

3 Specification of KLEIN

In this section we specify the cipher structure of KLEIN. Also the design principles will be discussed, which are followed during the design process of KLEIN. For each of the components of KLEIN, our choices will be motivated to achieve the well balanced trade-off between performance and security. The test vectors of KLEIN can be found in Appendix A.

3.1 Structure of KLEIN

The structure of KLEIN is a typical Substitution-Permutation Network (SPN), which is also used in many advanced block ciphers, e.g. AES and PRESENT. In our first estimation for obtaining a reasonable security margin and asymmetric iteration, we choose the number of rounds N_R as 12/16/20 for KLEIN-64/80/96 respectively. A high-level description of the KLEIN encryption routine is described in Figure 1.

```
$sk^1 \leftarrow$ KEY;
STATE $\leftarrow$ PLAINTEXT;
for $i = 1$ to $N_R$ do
  $AddRoundKey$(STATE, $sk^i$);
  $SubNibbles$(STATE);
  $RotateNibbles$(STATE);
  $MixNibbles$(STATE);
  $sk^{i+1} = KeySchedule(sk^i, i)$;
end for
CIPHERTEXT $\leftarrow AddRoundKey$(STATE, $sk^{N_R+1}$);
```

Figure 1: The encryption routine of KLEIN.

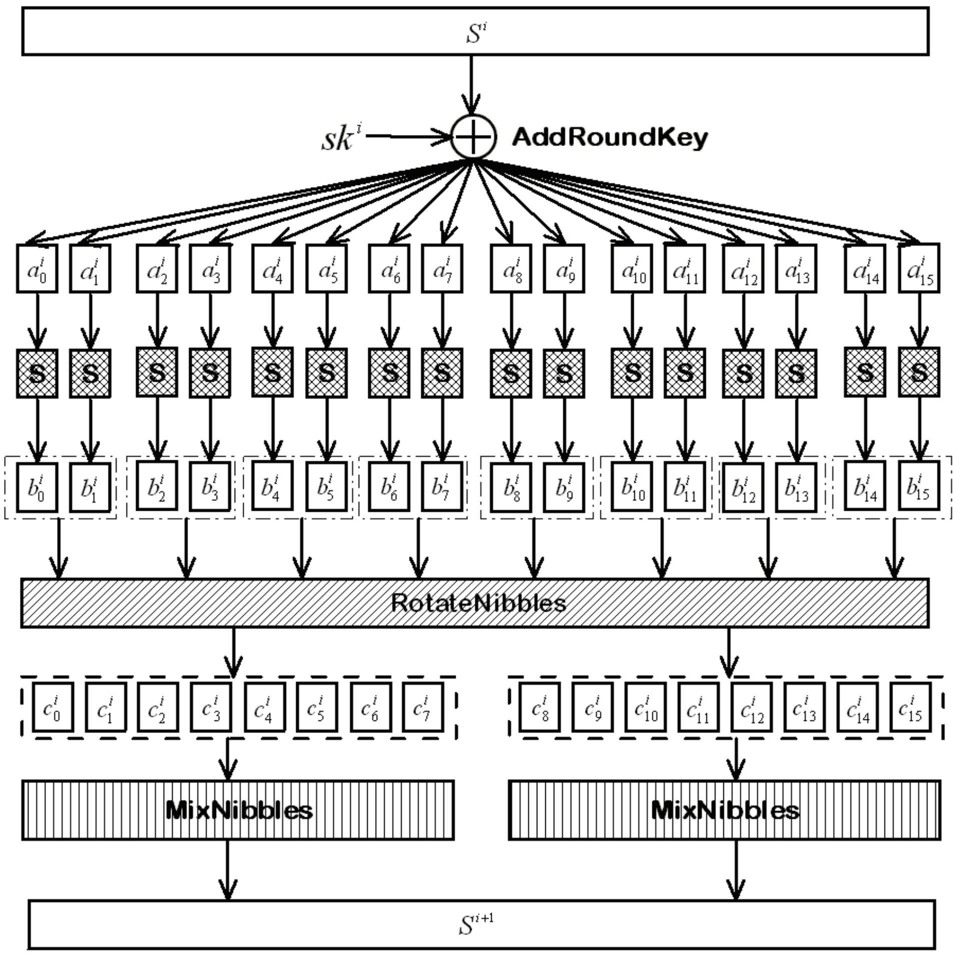
Note that many lightweight block ciphers are proposed to use only the filter counter mode and hence, the implementation costs of decryptions can be avoided. In the design of KLEIN, its lightweight property should also take the decryption algorithm into consideration without fixing on any cipher mode.

3.2 The Round Transformation

The input and output of KLEIN are considered to be one-dimensional arrays of bytes. During the round transformation, all the operations can be optimized with byte-oriented algorithms. Figure 2 graphically outlines the round transformation of KLEIN.

3.2.1 The SubNibbles Step

Before the SubNibbles step, the input text will be xored with the i-th round key sk^i, where $i \in [1, N_R]$. In the SubNibbles step, the xored results will be divided into 16 of 4-bit nibbles and input to the same 16 S-boxes. The KLEIN S-box S is a 4×4 involutive permutation. The non-linear permutation executed by S is described in Table 1. The implementation costs of such



S^i
sk^i
AddRoundKey
a_0^i a_1^i a_2^i a_3^i a_4^i a_5^i a_6^i a_7^i a_8^i a_9^i a_{10}^i a_{11}^i a_{12}^i a_{13}^i a_{14}^i a_{15}^i
S
b_0^i b_1^i b_2^i b_3^i b_4^i b_5^i b_6^i b_7^i b_8^i b_9^i b_{10}^i b_{11}^i b_{12}^i b_{13}^i b_{14}^i b_{15}^i
RotateNibbles
c_0^i c_1^i c_2^i c_3^i c_4^i c_5^i c_6^i c_7^i
c_8^i c_9^i c_{10}^i c_{11}^i c_{12}^i c_{13}^i c_{14}^i c_{15}^i
MixNibbles
MixNibbles
S^{i+1}

Figure 2: The round transformation of KLEIN.

a 4-bit S-box is much lower than that of an 8-bit S-box either by hardware or by software. By choosing an involutive S-box, we can also save the implementation costs for its inverse. Since the same S-boxes are used in the SubNibbles step, it allows a serialization of the design for an extremely small footprint. Moreover, we just need to provide one single side-channel protection for the S-box. Thus the overhead of an extra protection on its inverse is unnecessary.

Table 1: The 4-Bit S-box used in KLEIN.

Input	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Output	7	4	A	9	1	F	B	0	C	3	2	6	8	E	D	5

Since the SubNibbles step is the only non-linear layer in KLEIN, a natural requirement is an optimal resistance against linear and differential cryptanalyses. Therefore the choice of the S-box S fulfills the following conditions.

1. The S-box satisfies $\mathsf{S}(\mathsf{S}(x)) = x, x \in \mathbb{F}_2^4$, thus it can be used both in the encryption and in the decryption.
2. The S-box has no fixed points, i.e. $\mathsf{S}(x) \neq x, x \in \mathbb{F}_2^4$.
3. For any non-zero input difference $\Delta_I \in \mathbb{F}_2^4$ and output difference $\Delta_O \in \mathbb{F}_2^4$, it holds that
$$\sharp\{x \in \mathbb{F}_2^4 | \mathsf{S}(x) + \mathsf{S}(x + \Delta_I) = \Delta_O\} \leq 4. \tag{1}$$
Furthermore, if $wt(\Delta_I) = wt(\Delta_O) = 1$, we have
$$\sharp\{x \in \mathbb{F}_2^4 | \mathsf{S}(x) + \mathsf{S}(x + \Delta_I) = \Delta_O\} \leq 2. \tag{2}$$
4. For any non-zero $a, b \in \mathbb{F}_2^4$, it holds that
$$|S_b^{\mathcal{W}}(a)| = |\sum_{x \in \mathbb{F}_2^4} (-1)^{b \cdot \mathsf{S}(x) + a \cdot x}| \leq 8. \tag{3}$$
Furthermore, if $wt(a) = wt(b) = 1$, we have
$$|S_b^{\mathcal{W}}(a)| = |\sum_{x \in \mathbb{F}_2^4} (-1)^{b \cdot \mathsf{S}(x) + a \cdot x}| \leq 4. \tag{4}$$

The 4-bit S-box used in PRESENT satisfies $\sharp\{x \in \mathbb{F}_2^4 | \mathsf{S}(x) + \mathsf{S}(x + \Delta_I) = \Delta_O\} = 0$ if $wt(\Delta_I) = wt(\Delta_O) = 1$, which assures a better avalanche effect [2]. However, the PRESENT S-box is not an involution. According to our exhaustive search result, there is no such an involutive 4-bit S-box that can satisfy this additional property. Let $x = x_3||x_2||x_1||x_0$ denote the 4-bit input to the S-box S and let $\mathsf{S}(x) = y_3||y_2||y_1||y_0$. By using the algebraic normal form (ANF), the KLEIN S-box can be represented by the following Boolean functions.

$$\begin{aligned} y_0 &= 1 + x_0 + x_1 + x_3 + x_0x_2 + x_1x_2 + x_1x_3 + x_0x_1x_2 + x_0x_1x_3 \\ y_1 &= 1 + x_0 + x_2 + x_3 + x_1x_2 + x_1x_3 + x_2x_3 + x_0x_1x_3 \\ y_2 &= 1 + x_1 + x_2 + x_0x_2 + x_1x_2 + x_0x_3 + x_0x_1x_2 + x_0x_2x_3 + x_1x_2x_3 \\ y_3 &= x_1 + x_3 + x_0x_2 + x_0x_3 + x_0x_1x_3 + x_1x_2x_3 \end{aligned} \tag{5}$$

The differential distribution and the input-output correlation of the KLEIN S-box are given in Table 2 and Table 3, respectively. For each input differential Δ_I, the maximum probability of any output differential Δ_O is up to $4/16 = 2^{-2}$. Let p be the probability of a linear characteristic. The correlation of the linear characteristic over S is given by $q = (2p - 1)^2$ [32]. From the input-output correlation of S, it is straightforward that any linear characteristic over S has a correlation of at most $(2 \times \frac{4}{16} - 1)^2 = 2^{-2}$.

Table 2: Differential distribution table of S.

Δ_O / Δ_I	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	4	2	0	2	0	2	0	0	2	0	0	2	2
2	0	0	0	0	0	4	0	0	0	0	2	2	0	4	2	2
3	0	4	0	2	2	0	0	0	0	0	2	0	0	2	4	0
4	0	2	0	2	2	0	2	0	0	2	0	2	0	2	0	2
5	0	0	4	0	0	2	0	2	2	0	2	4	0	0	0	0
6	0	2	0	0	2	0	4	0	2	0	2	2	2	0	0	0
7	0	0	0	0	0	2	0	2	2	2	0	0	2	0	4	2
8	0	2	0	0	0	2	2	2	2	2	0	2	0	0	0	2
9	0	0	0	0	2	0	0	2	2	0	0	2	2	2	2	2
A	0	0	2	2	0	2	2	0	0	0	4	0	2	0	2	0
B	0	2	2	0	2	4	2	0	2	2	0	0	0	0	0	0
C	0	0	0	0	0	0	2	2	0	2	2	0	4	2	0	2
D	0	0	4	2	2	0	0	0	0	2	0	0	2	2	0	2
E	0	2	2	4	0	0	0	4	0	2	2	0	0	0	0	0
F	0	2	2	0	2	0	0	2	2	2	0	0	2	2	0	0

Table 3: Input-output correlation table of S.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	4	4	-4	-4	-4	-4	-4	-4	0	0	8	-8
2	0	0	0	8	-4	-4	4	-4	0	-8	0	0	4	-4	-4	-4
3	0	0	8	0	0	0	8	0	4	-4	-4	-4	-4	4	4	4
4	0	4	-4	0	0	-4	-4	-8	4	0	-8	4	-4	0	0	4
5	0	4	-4	0	-4	8	0	-4	0	-4	4	0	4	8	0	4
6	0	-4	4	8	-4	0	-8	4	4	0	0	4	0	4	4	0
7	0	-4	-4	0	-8	-4	4	0	-8	4	-4	0	0	4	4	0
8	0	-4	0	4	4	0	4	-8	0	4	8	4	-4	0	4	0
9	0	-4	-8	-4	0	-4	0	4	4	-8	4	0	-4	0	4	0
A	0	-4	0	-4	-8	4	0	-4	8	4	0	-4	0	-4	0	-4
B	0	-4	0	-4	4	0	4	0	4	0	-4	8	8	4	0	-4
C	0	0	4	-4	-4	4	0	0	-4	-4	0	8	-8	0	-4	-4
D	0	0	-4	4	0	8	4	4	0	0	-4	4	0	-8	4	4
E	0	8	-4	4	0	0	4	4	4	4	0	0	-4	4	0	-8
F	0	-8	-4	4	4	4	0	0	0	0	-4	-4	-4	4	-8	0

3.2.2 The RotateNibbles Steps

During the i-th round where $i \in [1, N_R]$, 16 nibbles $b_0^i, b_1^i, \cdots, b_{15}^i$ will be rotated left two bytes per round, which is illustrated in Figure 3. The inverse operation will be simply rotate right two bytes per round. Nevertheless, the RotateNibbles step can also be combined with the MixNibbles step to avoid the hardware or software costs.

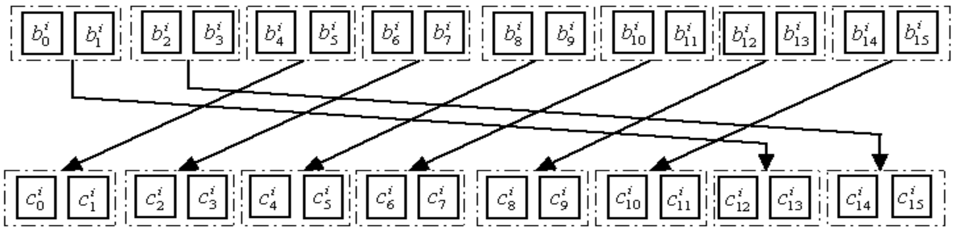


Figure 3: The RotateNibbles step.

3.2.3 The MixNibbles Step

The MixNibbles step is a bricklayer permutation of the state. The i-th round input nibbles $\{c_0^i, c_1^i, \cdots, c_{15}^i\}$ will be divided into 2 tuples, which will be proceeded the same as the MixColumns step in Rijndael. The tuples of the state are considered as polynomials over $\mathbb{F}_2^8$ and multiplied modulo $x^4 + 1$ with a fixed polynomial $c(x) = 03 \cdot x^3 + 01 \cdot x^2 + 01 \cdot x + 02$. The inverse is also a fixed multiplication polynomial $d(x) = 0\text{B} \cdot x^3 + 0\text{D} \cdot x^2 + 09 \cdot x + 0\text{E}$. The output of the MixNibbles step will be the intermediate state s^{i+1} for the next round transformation, which can be represented by the following matrix equations.

$$\begin{bmatrix} s_0^{i+1}||s_1^{i+1} \\ s_2^{i+1}||s_3^{i+1} \\ s_4^{i+1}||s_5^{i+1} \\ s_6^{i+1}||s_7^{i+1} \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \times \begin{bmatrix} c_0^i||c_1^i \\ c_2^i||c_3^i \\ c_4^i||c_5^i \\ c_6^i||c_7^i \end{bmatrix}, \begin{bmatrix} s_8^{i+1}||s_9^{i+1} \\ s_{10}^{i+1}||s_{11}^{i+1} \\ s_{12}^{i+1}||s_{13}^{i+1} \\ s_{14}^{i+1}||s_{15}^{i+1} \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \times \begin{bmatrix} c_8^i||c_9^i \\ c_{10}^i||c_{11}^i \\ c_{12}^i||c_{13}^i \\ c_{14}^i||c_{15}^i \end{bmatrix}.$$

Note that the balance between the diffusion property and the software performance let us make this choice. Although bit-shifting operations are often used in the diffusion layer of many lightweight block ciphers (e.g. PRESENT and NOEKEON), it actually loses the efficiency in software implementations. From the number of active Sboxes, it seems a better choice that the MixNibbles step chooses a matrix multiplication in $GF(2^4)$. However, a byte-oriented matrix multiplication has advantage in the software implementations for 8-bit processors (e.g. Skipjack). By using the similar implementation of the MixColumns step for 8-bit processors [11], we can use just one 256-byte look-up table to optimize the MixNibbles step. Also the same look-up table can be used to optimize its inverse. After the 12/16/20 rounds of KLEIN-64/80/96, the MixNibbles step can still provide a high number of active Sboxes for the security of KLEIN. The property of the MixColumns step of Rijndael has been well-analyzed, the details can be found in the literature [9, 11, 13].

3.2.4 Key schedule

For round transformations, all practical block ciphers use varied key schedules to expand a relative small master key to a series of dependent round keys. Since KLEIN will be used to construct block-cipher-based hash functions and message authentication codes, the key schedule should be agile even if keys are frequently changed. On the other hand, the key schedule should also consider a proper complexity for the security. To avoid the potential related-key weakness whilst balancing the performance, the key schedule of KLEIN is designed as follows.

1. Input: a 64/80/96-bit master key mk for KLEIN-64/80/96.

2. Key scheduling: Let i be the round counter of KLEIN-64/80/96. In the first round so that $i = 1$, the initial subkey $sk^1 = mk = sk_0^1||sk_1^1||\cdots||sk_t^1$ where $t = 7/9/11$ for KLEIN-64/80/96. For KLEIN-64, the $(i+1)$-th subkey sk^{i+1} can be derived from the i-th subkey sk^i as follows.

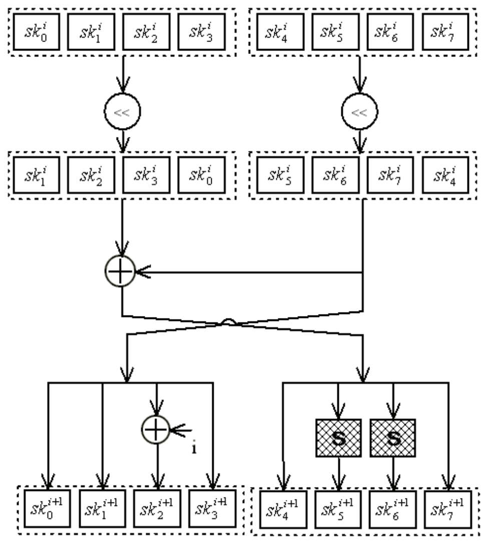
   (a) Divide the i-th subkey sk^i into two tuples, such that $a = (sk_0^i, sk_1^i, \cdots, sk^i_{\lfloor \frac{t}{2} \rfloor})$ and $b = (sk^i_{\lceil \frac{t}{2} \rceil}, sk^i_{\lceil \frac{t}{2} \rceil+1}, \cdots, sk_t^i)$ for the next step. For KlEIN-64, we have $a = (sk_0^i, sk_1^i, sk_2^i, sk_3^i)$ and $b = (sk_4^i, sk_5^i, sk_6^i, sk_7^i)$.

   (b) Cycling left shift one byte position in (a, b), obtain $a' = (sk_1^i, \cdots, sk^i_{\lfloor \frac{t}{2} \rfloor}, sk_0^i)$ and $b' = (sk^i_{\lceil \frac{t}{2} \rceil+1}, \cdots, sk_t^i, sk^i_{\lceil \frac{t}{2} \rceil})$ for the next step. For KLEIN-64, we have $a' = (sk_1^i, sk_2^i, sk_3^i, sk_0^i)$ and $b' = (sk_5^i, sk_6^i, sk_7^i, sk_4^i)$.

   (c) Swap the tuple (a', b') with a Feistel-like structure, such that $a'' = b'$ becomes the left tuple, whilst $b'' = a' \oplus b'$ becomes the right tuple.

   (d) Xor round counter i with the third byte in the left tuple a'', and substitute the second and the third bytes of the right tuple b'' by using the KLEIN S-box S.

3. Output: iteratively execute the above step for different key lengths, truncate the leftmost 64 bits of subkey sk^i for the i-th round transformation.



sk_0^i sk_1^i sk_2^i sk_3^i sk_4^i sk_5^i sk_6^i sk_7^i
<<
<<
sk_1^i sk_2^i sk_3^i sk_0^i sk_5^i sk_6^i sk_7^i sk_4^i
i
S
S
sk_0^{i+1} sk_1^{i+1} sk_2^{i+1} sk_3^{i+1} sk_4^{i+1} sk_5^{i+1} sk_6^{i+1} sk_7^{i+1}

Figure 4: The KeySchedule algorithm of 64-bit key length.

Figure 4 illustrates the KeySchedule algorithm of KLEIN-64. The key schedule of KLEIN is feasible for different key sizes. To save the memory for storing intermediate values, the subkeys of KLEIN can be generated during each round transformation. During the performance tuning on sensors, we observed that the on-the-fly key schedule of KLEIN is more resource-efficient than the traditional optimization such that all subkeys are computed in advance. Also the Feistel-like structure provides more complexities to resist weak key attacks, which was found on the PRESENT block cipher recently [5, 36].

4 Security Analysis

In this section we will present a security analysis of KLEIN, showing his resistance against various cryptanalytic attacks.

4.1 Linear and Differential Attacks

The resistance of linear and differential attacks of a block cipher is mainly based on the branch number, i.e. the number of active S-boxes in a certain number of rounds. In Rijndael, the authors use the Maximum Distance Separable (MDS) code to achieve the maximal branch number in a small number of rounds. By combining the RotateNibbles and MixNibbles steps, KLEIN can achieve a balance between the minimum number of active S-boxes and the software performance for resource-constrained devices.

Theorem 1 *Any four-round differential characteristic of KLEIN has a minimum of 15 active S-boxes.*

Proof. The MixColumns step in Rijndael is based on an Maximum Distance Separable code and the distance between any two distinct words called branch number is 5 [11]. Since we use the same matrix multiplication in the MixNibbles step of KLEIN, the branch number of MixNibbles is also 5. Since MixNibbles is computed with multiplications in $GF(2^8)$, an active byte in the diffusion layer of KLEIN implies one or two nibbles (i.e. the leftmost and the rightmost 4 bits) are active. For simplicity, we assume every active byte only has one active nibbles. Let $\Delta_i = L_i || R_i$ be the i-th round input difference characteristic, where L_i and R_i denote the left and the right 4-byte tuples respectively. The differential patterns of four-round KLEIN can be analyzed as follows.

- If there is 1 non-zero byte in L_1, it will be at least one active S-box in the first round. After the Rotate and MixNibbles, the difference will be propagated to 4 bytes either in the left or the right 4-byte tuple. Thus Δ_2 will have minimum 4 active S-boxes in the second round. For simplicity, we assume L_2 contains 4 active bytes while R_2 remains zero. After the RotateNibbles step, both the left and the right tuples will have 2 active bytes. Since the branch number of MixNibbles is 5, the minimum number of active bytes with the differential characteristic Δ_3 will be 6. After RotateNibbles in the third round, if the active bytes in L_3 is 2 or 3, R_3 will have 4 or 3 active bytes respectively. In either of the situations, Δ_4 will have minimum 3+1 or 2+2 active bytes. In this general case, the minimum active S-boxes after four rounds are $1 + 4 + 6 + 4 = 15$.

- If there is 2 non-zero bytes in L_1, the difference will be propagated to at least 3 bytes after MixNibbles. For simplicity, we assume L_2 contains 3 active bytes while R_2 remains zero. After RotateNibbles, the active bytes in L_2 is 1 or 2, while R_2 will have 2 or 1. Since the branch number of MixNibbles is 5, the minimum number of active bytes with the differential characteristic Δ_3 will be 7. After RotateNibbles in the third round, if the active bytes in L_3 is 3 or 4, R_3 will have 4 or 3 active bytes respectively. In either of the situations, Δ_4 will have minimum 2+1 or 1+2 active bytes. In this general case, the minimum active S-boxes after four rounds are $2 + 3 + 7 + 3 = 15$.

- In the case of 3 (or 4) active bytes in L_1, the difference patterns of the first three rounds will be identical to the case of 2 active bytes in the last three rounds. The minimum active bytes after three rounds are $3 + 7 + 3$ (or $4 + 6 + 4$). In the third round, first we choose all

3 (or 4) active bytes are moved to L_3 after RotateNibbles. After MixNibbles, the active bytes will be at least 2 (or 1) since the branch number is 5. In any other choice, the active bytes in the forth round will be no less than 2 (or 1). Thus the minimum active S-boxes after four rounds are $3 + 7 + 3 + 2 = 15$ (or $4 + 6 + 4 + 1 = 15$) in this general case.

Without loss of generality, the same differential patterns will be followed where all active bytes are in R_1. If both L_1 and R_1 have one or more active bytes, it is straightforward that the minimum number of active S-boxes will be no less than 15. Thus any four-round differential characteristic of KLEIN has a minimum of 15 active S-boxes. $\square$

In RotateNibbles and MixNibbles, if we choose operations over $GF(2^4)$ for the MDS code, the active S-boxes in a four-round differential characteristic can be lower bounded by 25. Although a higher number of active S-boxes means KLEIN can use less rounds to be secure, our tuning experiments in sensors show that the software performance will be sacrificed by operations over $GF(2^4)$ (e.g. bit-shifting from leftmost to rightmost). However, it always requires a trade-off between the performance and the security. For ultra-lightweight in hardware, any differential characteristic of PRESENT has only 10 active S-boxes after 5 rounds.

In Rijndael, the coefficients of the MixColumns step are selected for assuring that both the differential branch number and the linear branch number are equal to 5. Based on the combination of RotateNibbles and MixNibbles, KLEIN also has the same property on the branch numbers. Therefore the minimum number of active S-boxes in a four-round linear approximation can be derived from the four-round differential propagation result of KLEIN. For brevity, the proof is omitted here.

Theorem 2 *Any four-round linear approximation of KLEIN has a minimum of 15 active S-boxes.*

The strength of a cipher against differential attacks is reflected by the maximum probability of differential , i.e. a collection of characteristics. However, in cryptanalysis we often assume that one characteristic has a much larger probability than the other characteristics of the differential. Thus a characteristic with the maximum probability is taken as an estimate of the probability of the differential. Similar assumptions can be found in linear attacks as well. Based on the minimum active S-boxes of characteristics in certain rounds, we can also derive the resistance of the differential and linear attacks on KLEIN. Since any differential characteristic over the KLEIN S-box has a maximum 2^{-2} possibility, the security against differential attacks of KLEIN-64 can be estimated as follows.

Lemma 1 *Let ϵ^d_{12R} be the maximum probability of a differential characteristic of 12 rounds of KLEIN-64. Then $\epsilon^d_{12R} \leq (2^{-2})^{12\times15/4} \approx 2^{-90}$.*

Since any linear characteristic over the KLEIN S-box has a correlation 2^{-2}, the security against linear attacks of the full-round KLEIN-64 is described as follows.

Lemma 2 *Let ϵ^l_{12R} be the maximal bias of a linear approximation of 12 rounds of KLEIN-64. Then $\epsilon^l_{12R} \leq (2^{-2})^{12\times15/4} \approx 2^{-90}$.*

By following the similar analysis, we note that the security of KLEIN-80/96 against linear and differential attacks can be gauged with the rounds 16/20.

Lemma 3 *Let ϵ^d_{16R} be the maximum probability of a differential characteristic of 16 rounds of KLEIN-80. Then $\epsilon^d_{16R} \leq (2^{-2})^{16\times15/4} \approx 2^{-120}$.*

Lemma 4 *Let ϵ^l_{16R} be the maximum bias of a linear approximation of 16 rounds of KLEIN-80. Then $\epsilon^l_{16R} \leq (2^{-2})^{16\times16/4} \approx 2^{-120}$.*

Lemma 5 *Let ϵ^d_{20R} be the maximum probability of a differential characteristic of 20 rounds of KLEIN-96. Then $\epsilon^d_{20R} \leq (2^{-2})^{20\times15/4} \approx 2^{-150}$.*

Lemma 6 *Let ϵ^l_{20R} be the maximum bias of a linear approximation of 20 rounds of KLEIN-96. Then $\epsilon^l_{20R} \leq (2^{-2})^{20\times15/4} \approx 2^{-150}$.*

From the above results, we assure that KLEIN families have a good security margin in the full rounds. The extra security margin of a block cipher may benefit the lightweight design of block-cipher-based hash functions or message authentication codes [12, 18].

4.2 Key Schedule Attacks

Since there are no established guidelines for the design of key schedules, both a wide variety of designs and a wide variety of schedule-specific attacks have been proposed. The most effective attacks come under the general heading of related-key attacks and slide attacks, and both rely on the build-up of identifiable relationships between different sets of subkeys. To counter this threat, we use a round-dependent counter so that the subkey sets cannot easily be symmetric. We also use the same KLEIN S-box to provide the non-linearity of the subkeys whilst saving the implementation costs. For related-key attacks, we have the following properties for protection.

- For KLEIN-64/80/96, each bit in the key register depends on at least 4 user-supplied bits after 4/5/6 rounds.
- For KLEIN-64/80/96, all the bits in the key register are a non-linear function of the 64/80/96-bit user-supplied key by 8/10/12 rounds.

4.2.1 Integral Attack

Integral cryptanalysis are usually applied to exploit vulnerabilities in byte-oriented block ciphers, such as AES [11]. An integral attack will investigate the propagation of sums of many values, whilst a differential attack will consider the propagation of differences between pairs. In a byte-oriented cipher, the sum of a group differences might be a predictable value after certain rounds. For a better software performance, the design of KLEIN also adapts a byte-oriented structure like AES. Thus it also faces a similar vulnerability on integral attacks [25].

First we consider a five-round integral attack on KLEIN, which is based on the attack given by Knudsen and Wagner on AES [25]. The attacker chooses a group of 256 plaintexts, which have equal values in all bytes except one. According to the mathematic properties of the RotateNibbles and MixNibbles steps, the sum of 256 bytes will be zero after three rounds of encryption. Then the attacker will guess 4 key bytes in the fourth round and 1 key bytes in the fifth round. If the $4 + 1 = 5$ key bytes are right, the sum of all 256 values should also be zero after five rounds. For KLEIN-80/96, we can extend the above attack to six rounds. Thus we need a collection of 2^{32} plaintexts in the first round and guess $4 + 5 = 9$ bytes in total. It is straightforward that any integral attack on KLEIN over seven rounds will be more complicated than exhaustive key searches. Except byte-oriented integral attacks, One could try nibble-oriented attacks with 16 plaintexts which have equal values except one nibble. However, since the MixNibbles step is fully based on multiplications in $GF(2^8)$, the sum of 16 nibbles is unpredictable after three rounds. Therefore nibble-oriented integral attacks will not be more feasible than byte-oriented ones.

4.3 Algebraic Attack

The algebraic attack as well as the *Cube Attack* [15], requires the algebraic form describing the output bits has a relatively small degree in terms of the input bits being processed. To exploit the algebraic relations between input and output bits of a block cipher, attackers may consider a subset of input bits whilst leave the other fixed. In the S-box of KLEIN, every output bit can be represented by a 3-degree polynomial with 4 input variables in ANF. For total 64 input bits, the complexity of finding the polynomials for the entire cipher soon becomes too large. In the full-round KLEIN-64, the number of S-boxes in the encryption and the key schedule equals $n = 12 \times 16 + 12 \times 4 = 240$. It is well-known that any 4-bit S-box can be represented by at least 21 quadratic equations over $GF(2)$. Thus in KLEIN-64, we have the number of quadratic equations $n \times 21 = 5040$ in $n \times 8 = 1920$ variables. By changing the number of rounds, similar results can easily be extended to KLEIN-80 and KLEIN-96. In our experiment, we were unable to transform three-round KLEIN-64 to the ANF equations in a reasonable time.

4.4 Side-Channel Attack

Since it is easy to add noises and loops in the software implementation of a block cipher to avoid side-channel attacks, here we will discuss on how to secure the hardware implementation of KLEIN. Except for the SubNibbles step, KLEIN is completely linear. The S-box of KLEIN can be implemented to resist side-channel attacks even in the presence of glitches using the secret sharing method proposed by Nikova *et al.* [34]. Also the linear part of KLEIN can be securely processed by using independent shares. A survey of lightweight cryptography and DPA countermeasures [33] estimates that the masking based on secret sharing will increase the hardware overhead with a factor of 3, which is still promising because it has a moderate area overhead and was theoretically proven to be secure against DPA attacks [34].

5 Performance

Here we analyze the performance of KLEIN. Based on legacy low-resource sensors TelosB (with 16-bit TI MSP430 microcontroller) and IRIS with (8-bit ATmega128L microcontroller), a detailed comparison amongst KLEIN and other candidates is given in Table 4. These two platforms are chosen because of their opposing characteristics: TelosB has more RAM than IRIS (10 KB vs 8 KB) but IRIS has a larger Flash memory than TelosB (128 KB vs 48 KB); TelosB's transceiver CC2420 supports hardware AES encryption but IRIS does not. The processing speeds are measured in encryption/decryption, whilst the storage costs are calculated in together. We note that all software implementations are optimized by using look-up tables, while the tables are stored in ROM for lower RAM costs. For low-resource devices, a lower cost of RAM would be benefit for power consumptions and manufactory expenses. Since the CC2420 chip on TelosB also supports AES hardware encryption, we also test its performance by implementing the standalone AES encryption of CC2420 [40]. The result shows AES hardware implementation has a great improvement on RAM and ROM costs, while the processing speed is even lower than the software implementation. We consider this latency is caused by the fact that the hardware AES encryption function should power up the CC2420 chip on TelosB in advance. Maybe the latency can be prevented by setting CC2420 in the standby mode, but the power consumptions will be increased as well.

From a wide range of block ciphers, PRESENT is chosen because it is ultra-lightweight for highly resource-constrained [3], while Skipjack is proven to be software-efficient for 8-bit processors [27]. Both of them have similar block and key sizes with KLEIN. For Hummingbird encryption, Engels *et al.* [16] shows that the speed optimized implementation on 8-bit microcontrollers is about 28.9% slower than PRESENT encryption when the message length is 64 bits. On 16-bit microcontrollers Hummingbird achieves around $50\% \sim 78\%$ performance improvements for different message blocks [16]. The performance comparison in Table 4 shows that KLEIN is competitive for low-resource applications, especially suitable for sensors. Although the block processing speeds of KLEIN are slower than Skipjack, it is a reasonable trade-off for the security margin of KLEIN.

Table 4: The software performance of KLEIN and related block ciphers.

	Performance on TelosB				
Algorithm	Key length (bit)	Block size (bit)	RAM (byte)	ROM (byte)	Block processing speed (ms)
AES-128 (software implementation)	128	128	222	12568	1.64/1.67
AES-128 (hardware encryption)	128	128	64	2570	2.29
PRESENT-80 (software encryption)	80	64	44	5086	5.02
Skipjack	80	64	56	3212	0.63/0.65
KLEIN-64	64	64	46	4712	0.94/1.50
KLEIN-80	80	64	46	4864	1.26/2.06
KLEIN-96	96	64	46	5052	1.60/2.77
	Performance on IRIS				
Algorithm	Key length (bit)	Block size (bit)	RAM (byte)	ROM (byte)	Block processing speed (ms)
AES-128 (software implementation)	128	128	295	15670	1.39/1.44
PRESENT-80 (software encryption)	80	64	82	5176	2.50
Skipjack	80	64	133	4206	0.40/0.45
KLEIN-64	64	64	97	4088	0.44/0.66
KLEIN-80	80	64	97	4148	0.57/0.92
KLEIN-96	96	64	97	4228	0.71/1.20

In KLEIN, the MixNibbles step will be a non-straightforward part of hardware design. Since it is the same to the MixColumns step of AES, we may simply borrow the idea from AES hardware implementations. Feldhofer *et al.* [17] shows a hardware-efficient implementation of the MixColumns step, which only costs about 340 GE with a 32-bit width. Because the MixNibbles step can be paralleled by two 32-bit tuples, Feldhofer *et al.*'s implementation can also be used in KLEIN families. Both of the RotateNibbles and InterchangeNibbles steps are simple byte-shift operations, which can be implemented with a minimum hardware. Figure 5 shows the architecture of KLEIN-64 encryption with a 64-bit width datapath.

For a hardware implementation, a complete-including the analog part-low-cost RFID tag might have between 1,000 and 10,000 GE, and for security components may occupy up to 2,000

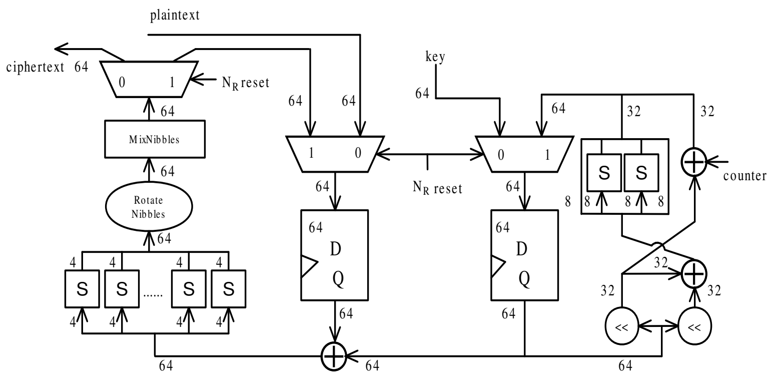


Figure 5: The data path of KLEIN-64 encryption.

GE [22]. The area restriction could be looser on sensors. By using the Virtual Silicon (VST) standard cell library on the UMC 0.18μm L180 Process [3, 17], the area in GE of KLEIN families with a 64-bit datapath is estimated in Table 5. We note that a further area optimization can be derived by a serialization of the design, i.e. with a 4-bit width datapath.

Table 5: The area in GE of KLEIN and related block ciphers.

	Hardware Encryption		
Algorithm	Logic process (μm)	Datapath (bits)	Area in GE
AES-128 [17]	0.35	32	3400
PRESENT-80 [3]	0.18	64	1570
KLEIN-64	0.18	64	1981
KLEIN-80	0.18	64	2097
KLEIN-96	0.18	64	2213

6 Conclusion

In this paper, we have proposed a new lightweight block cipher KLEIN. The goal of our design is to provide a practical and secure cipher for low-resource applications, especially for wireless sensor networks. Analyzing the security and performance of a block cipher is a complex and time-consuming task even though our preliminary results have been encouraging. Although KLEIN mainly focuses on software implementations, it also enjoys hardware efficiency from its simple structure with an involutive S-box. The various key lengths of KLEIN offer a flexibility and a moderate security level for ubiquitous applications. Therefore, our design increases the available options of lightweight block ciphers for low-resource applications.

References

[1] E. Biham, A. Shamir, and A. Biryukov. Cryptanalysis of skipjack reduced to 31 rounds using impossible differentials. In J. Stern, editor, *Advances in Cryptology - EUROCRYPT 1999*, volume LNCS 1592, pages 12–23. Springer, 1999.

[2] A. Bogdanov, L.R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. Robshaw, Y. Seurin, and C. Vikkelsoe. PRESENT: An ultra-lightweight block cipher. In P. Paillier and I. Verbauwhede,

editors, *Cryptographic Hardware and Embedded Systems - CHES 2007*, volume LNCS 4727, pages 450–466. Springer Heidelberg, 2007.

[3] A. Bogdanov, G. Leander, C. Paar, A. Poschmann, M.J.B. Robshaw, and Y. Seurin. Hash functions and RFID tags: Mind the gap. In R. Safavi-Naini, editor, *Cryptographic Hardware and Embedded Systems - CHES 2008*, volume LNCS 5154, pages 283–299. Springer, 2008.

[4] Chipcon. CC2420: 2.4 GHz IEEE 802.15.4 / ZigBee-ready RF transceiver. Available at http://focus.ti.com/lit/ds/symlink/cc2420.pdf.

[5] J.Y. Cho. Linear cryptanalysis of reduced-round PRESENT. In J. Pieprzyk, editor, *CT-RSA 2010*, volume LNCS 5985, pages 302–317, 2010.

[6] B. Collard and F.-X. Standaert. A statistical saturation attack against the block cipher PRESENT. In M. Fischlin, editor, *CT-RSA 2009*, volume LNCS 5473, pages 195–210, 2009.

[7] Crossbow. IRIS wireless measurement system. Available at http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/IRIS_Datasheet.pdf.

[8] Crossbow. TelosB mote platform. Available at http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/TelosB_Datasheet.pdf.

[9] J. Daemen, L.R. Knudsen, and V. Rijmen. Linear frameworks for block ciphers. *Designs, Codes and Cryptography*, Vol. 22, No. 1:65–87, Springer, January 2001.

[10] J. Daemen, M. Peeters, G. Van Assche, and V. Rijmen. The Noekeon block cipher. The NESSIE Proposal, 2000.

[11] J. Daemen and V. Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer-Verlag, 2002.

[12] J. Daemen and V. Rijmen. A new MAC construction ALRED and a specific instance Alpha-MAC. In H. Gilbert and H. Handschuh, editors, *FSE 2005*, volume LNCS 3557, pages 1–17. Springer, 2005.

[13] J. Daemen and V. Rijmen. New criteria for linear maps in AES-like ciphers. *Cryptography and Communications*, Vol. 1, No. 1:47–69, Springer, April 2009.

[14] C. De Cannière, O. Dunkelman, and M. Knežević. Katan and Ktantan - a family of small and efficient hardware-oriented block ciphers. In C. Clavier and K. Gaj, editors, *Cryptographic Hardware and Embedded Systems - CHES 2009*, volume LNCS 5747, pages 272–288. Springer, 2009.

[15] I. Dinur and A. Shamir. Cube attacks on tweakable black box polynomials. Cryptology ePrint Archive, Report 2008/385, 2008.

[16] D. Engels, X. Fan, G. Gong, H. Hu and E.M. Smith. Hummingbird: Ultra-Lightweight Cryptography for Resource-Constrained Devices. In R. Sion et al., editors, *FC 2010 Workshops*, volume LNCS 6054, pages 3–18. Springer, 2010.

[17] M. Feldhofer, J. Wolkerstorfer, and V. Rijmen. AES implementation on a grain of sand. *Information Security, IEE Proceedings*, 152(1):13–20, 2005.

[18] Z. Gong, P. Hartel, S. Nikova, and B. Zhu. Towards secure and practical MACs for body sensor networks. In B.K. Roy and N. Sendrier, editors, *Progress in Cryptology - INDOCRYPT 2009*, volume LNCS 5922, pages 182–198. Springer, 2009.

[19] P. Hamalainen, T. Alho, M. Hannikainen, and T. D. Hamalainen. Design and implementation of low-area and low-power AES encryption hardware core. In *DSD '06: Proceedings of the 9th EUROMICRO Conference on Digital System Design*, pages 577–583. IEEE Computer Society, 2006.

[20] D. Hong, J. Sung, S. Hong, J. Lim, S. Lee, B. Koo, C. Lee, D. Chang, J. Lee, K. Jeong, H. Kim, J. Kim, and S. Chee. HIGHT: A new block cipher suitable for low-resource device. In L. Goubin and M. Matsui, editors, *Cryptographic Hardware and Embedded Systems - CHES 2006*, volume LNCS 4249, pages 46–59. Springer, 2006.

[21] M. Healy, T. Newe, and E. Lewis. Analysis of hardware encryption versus software encryption on wireless sensor network motes. In S.C. Mukhopadhyay and G.S. Gupta, editors, *Smart Sensors and Sensing Technology 2008*, volume LNEE 20, pages 3–14. Springer, 2008.

[22] A. Juels and S.A. Weis. Authenticating pervasive devices with human protocols. In V. Shoup, editor, *Advances in Cryptology-CRYPTO 2005*, volume LNCS 3126, pages 198–293. Springer-Verlag, 2005.

[23] L.R. Knudsen and Håvard Raddum. On Noekeon. The NESSIE Report, April 2001.

[24] L.R. Knudsen, M.J.B. Robshaw, and D. Wagner. Truncated differentials and skipjack. In M. Wiener, editor, *Advances in Cryptology - CRYPTO 1999*, volume LNCS 1666, pages 165–180. Springer, 1999.

[25] L.R. Knudsen and D. Wagner. Integral cryptanalysis. In J. Daemen and V. Rijmen, editors, *FSE 2002*, volume LNCS 2365, pages 112–127. Springer, 2002.

[26] R. Könighofer. A fast and cache-timing resistant implementation of the AES. In T. Malkin, editor, *CT-RSA 2008*, volume LNCS 4964, pages 187–202. Springer, 2008.

[27] Y. W. Law, J. Doumen, and P. H. Hartel. Survey and benchmark of block ciphers for wireless sensor networks. *ACM Trans. Sen. Netw.*, 2(1):65–93, 2006.

[28] G. Leander, C. Paar, A. Poschmann, and K. Schramm. New lighweight DES variants. In A. Biryukov, editor, *Fast Software Encryption 2007 - FSE 2007*, volume LNCS 4593, pages 196–210. Springer, Berlin, 2007.

[29] C.H. Lim. A revised version of Crypton: Crypton v1.0. In L.R. Knudsen, editor, *Fast Software Encryption-FSE'99*, volume LNCS 1636, pages 31–45. Spinger-Verlag, 1999.

[30] C.H. Lim and T. Korkishko. mCrypton - a lightweight block cipher for security of low-cost RFID tags and sensors. In J. Song, T. Kwon, and M. Yung, editors, *WISA 2005*, volume LNCS 3786, pages 243–258. Springer, Berlin, 2005.

[31] S. Mangard, T. Popp, and B.M. Gammel. Side-channel leakage of masked CMOS gates. In A.J. Menezes, editor, *Topics in Cryptology - CT-RSA 2005*, volume LNCS 3376, pages 351–365. Springer, 2005.

[32] Mitsuru Matsui. New structure of block ciphers with provable security against differential and linear cryptanalysis. In Dieter Gollmann, editor, *FSE*, volume 1039 of *Lecture Notes in Computer Science*, pages 205–218. Springer, 1996.

[33] A. Moradi and A. Poschmann. Lightweight cryptography and DPA countermeasures: A survey. Workshop on Lightweight Cryptography for Resource-Constrained Devices - WLC'2010, to appear, 2010.

[34] S. Nikova, V. Rijmen, and M. Schläffer. Secure hardware implementation of non-linear functions in the presence of glitches. In P.J. Lee and J.H. Cheon, editors, *ICISC 2008*, volume LNCS 5461, pages 218–234. Springer, 2009.

[35] National Institute of Standards and Technology. Skipjack and kea algorithm specifications (version 2.0). NIST online document. Available at http://csrc.nist.gov/groups/ST/toolkit/documents/skipjack/skipjack.pdf, May 1998.

[36] K. Ohkuma. Weak keys of reduced-round PRESENT for linear cryptanalysis. In M.J. Jacobson Jr., V. Rijmen and R. Safavi-Naini, editors, *Selected Areas in Cryptography - SAC 2009*, volume LNCS 5867, pages 249–265, Springer-Verlag, 2009.

[37] O. Özen, K. Varici, C. Tezcan, and Ç. Kocair. Lightweight block ciphers revisited: Cryptanalysis of reduced round PRESENT and HIGHT. In C. Boyd and J.G. Nieto, editors, *ACISP 2009*, volume LNCS 5594, pages 90–107. Springer, 2009.

[38] C. Paar, A. Poschmann, and M. Robshaw. New designs in lightweight symmetric encryption. In P. Kitsos and Y. Zhang, editors, *RFID Security: Techniques, Protocols and System-on-Chip Design*, pages 349–371. Springer, 2008.

[39] C. Rolfes, A. Poschmann, G. Leander, and C. Paar. Ultra-lightweight implementations for smart devices — security for 1000 gate equivalents. In G. Grimaud and F.-X. Standaert, editors, *CARDIS'08: Proceedings of the 8th IFIP WG 8.8/11.2 International Conference on Smart Card Research and Advanced Applications*, volume LNCS 5189, pages 89–103. Springer-Verlag. 2008.

[40] B. Zhu. The standalone AES encryption of CC2420 (Tinyos 2.10 and MICAz). Available at http://cis.sjtu.edu.cn/index.php/Bo_Zhu, Decemeber 2008.

Appendix A. Test Vectors of KLEIN

Test vectors for KLEIN-64/80/96 are given in the following tables. Readers may use it to check the correctness of the code by themselves.

Table 6: Test vectors for KLEIN-64.

Key	Message	Cipher
0000 0000 0000 0000	FFFF FFFF FFFF FFFF	CDC0 B51F 1472 2BBE
FFFF FFFF FFFF FFFF	0000 0000 0000 0000	6456 764E 8602 E154
1234 5678 90AB CDEF	FFFF FFFF FFFF FFFF	5923 56C4 9971 76C8
0000 0000 0000 0000	1234 5678 90AB CDEF	629F 9D6D FF95 800E

Table 7: Test vectors for KLEIN-80.

Key	Message	Cipher
0000 0000 0000 0000 0000	FFFF FFFF FFFF FFFF	6677 E20D 1A53 A431
FFFF FFFF FFFF FFFF FFFF	0000 0000 0000 0000	8224 7502 273D CC5F
1234 5678 90AB CDEF 1234	FFFF FFFF FFFF FFFF	3F21 0F67 CB23 687A
0000 0000 0000 0000 0000	1234 5678 90AB CDEF	BA52 39E9 3E78 4366

Table 8: Test vectors for KLEIN-96.

Key	Message	Cipher
0000 0000 0000 0000 0000 0000	FFFF FFFF FFFF FFFF	DB9F A7D3 3D8E 8E36
FFFF FFFF FFFF FFFF FFFF FFFF	0000 0000 0000 0000	15A3 A033 86A7 FEC6
1234 5678 90AB CDEF 1234 5678	FFFF FFFF FFFF FFFF	7968 7798 AFDA 0BC3
0000 0000 0000 0000 0000 0000	1234 5678 90AB CDEF	5006 A987 A500 BFDD