Block ciphers for the IoT – SIMON, SPECK, KATAN, LED, TEA, PRESENT, and SEA compared

Michael Appel[1], Andreas Bossert[1], Steven Cooper[1], Tobias Kußmaul[1], Johannes Löffler[1], Christof Pauer[1], and Alexander Wiesmaier[1,2,3]

[1] TU Darmstadt
[2] AGT International
[3] Hochschule Darmstadt

Abstract. In this paper we present 7 block cipher algorithms `Simon`, `Speck`, `KATAN`, `LED`, `TEA`, `Present` and `Sea`. Each of them gets a short introduction of their functions and it will be examined with regards to their security. We also compare these 7 block ciphers with each other and with the state of the art algorithm the Advanced Encryption Standard (AES) to see how efficient and fast they are to be able to conclude what algorithm is the best for which specific application.

Keywords: Internet of things (IoT); lightweight block ciphers; SIMON; SPECK; KATAN; LED; TEA; PRESENT; SEA

1 Introduction

In modern IT `The Internet of Things(IoT)` is one of the most recent topics. Through the technical progress the internet has increasingly moving into our daily lives. More and more devices get functions to go online interconnect with each other and send and receive data. The increasingly smaller and cheaper expectant electronic control and communication components were installed in particular in recent years, increasingly in things of daily life. Typical fields of application are for example home automation, Security technology in the private or business environment as well as the supporting usage in the industry [1]. Because of the very high price sensitivity in this environment the focus is set on the efficiency of the used programs and algorithms. Requires an optimized algorithm for example just the half on computing time and memory, it is accordingly possible to use cheaper hardware. Extrapolated to the produced number of units a considerable amount can be saved or the IoT technology can be built in accordingly cheaper things. Due to the growing integration of technology in the daily life and inevitably into the highly personal sphere, the claim of confidentiality and security on the collected data and the networked devices is increasing. It is very important to be sure that these connections are secure but also efficient. The state of the art block cipher AES cannot be used for these low-end devices

such as RFID tags or sensor networks because they are often very small, have less computing power or have to be very power saving. So we have very constrained environments. Against this background particularly efficient algorithms have been developed, which are partly specially adapted to the used hardware. Our 7 block ciphers had been developed to fulfil these constraints. KATAN, LED, SIMON and PRESENT have been optimized for performance on hardware devices and SPECK, SEA and TEA for performance in software. At the comparison section we will see how good they fulfil these goals.
We organize this paper as follows: In Section 2 we have a short overview of further related works which also concerned about our 7 algorithms. In section 3 some attack procedures on block ciphers are explained. Section 4, 5, 6, 7, 8, 9 presents SIMON, SPECK, KATAN, LED, TEA, PRESENT and SEA. How they work, the different variants and possible attacks against them. In Section 10 the block ciphers are compared with each other under performance and efficiency points as well as with AES aspects. Section 11 gives a short conclusion.

2 Related Work

There is a growing number of low-cost cryptography and a number of papers dealing with their comparison. The Paper from [2] *Compact Implementation and Performance Evaluation of Block Ciphers* in ATtiny Devices [3] tries to build a uniform comparison platform by using the ATMEL ATtiny45. [4] [3] implement 12 block ciphers including AES, DESL, HEIGHT, IDEA, KASUMI, KATAN, KLEIN, mCrypton, NOEKEON, PRESENT, SEA and TEA on that platform and published the source code as open source. This should serve as a better comparison platform for the future. There is also work done that compares not only block ciphers among each other. [3] covers block ciphers and stream ciphers. For the comparison, they focus on key bits, block bits, cycles per block, throughput at 100 kHz and the area in gate equivalents the algorithm needs for implementation. They split the algorithms they compared into two groups, hardware and software oriented ciphers. [4] focused mainly on stream ciphers in their work *Hardware results for selected stream cipher candidates*[4].
In the area of SIMON and SPECK there exist two similar papers *The SIMON and SPECK Families of Lightweight Block Ciphers* [5] and *SIMON and SPECK: Block Ciphers for the Internet of Things* [6]. The papers explains how the algorithm works, it contains many performance comparisons on constrained platforms and it describes many security aspects. A few of the comparisons were used in this paper too.
One paper which compares `KATAN` with other lightweight block ciphers is the paper *Katan and ktantan - a family of small and efficient hardware-oriented block ciphers* [7] from the designers of `KATAN` . It explains what are the difference between KATAN and several existing block ciphers and compares them. The paper *Modellbildung in der algebraischen kryptoanalyse* [8] handles among other about different attacks on KATAN and compares them with each other. Furthermore

[4] http://www.atmel.com/devices/attiny45.aspx

does it explain how KATAN is designed and explains a new algebraic attack which is better than any other known algebraic attacks.
In the area of TEA and LED there exist many papers which also deal with these algorithms from different points of view. First to call are the papers which generally deal with these encryption methods. They each illustrate one of the algorithms and concentrate on certain properties in different detail degrees. For the Tiny Encryption Algorithm (TEA) the following papers are cited as examples: *TEA, a Tiny Encryption Algorithm*[9], *Tiny Encryption Algorithm (TEA)*[10] and *The Tiny Encryption Algorithm (TEA)*[11]
Another relevant group are the papers which have a special focus on the lightness and therefore the suitability of TEA in very small, computationally weak and cheaper Hardware. Here are specially the papers: *Design and Implementation of Low Power Hardware Encryption for Low Cost Secure RFID Using TEA*[12] and *Hardware Implementation of a TEA-Based Lightweight Encryption for RFID Security* [13] to call.
At last we have the papers which consider XTEA under the security aspects and describe vulnerabilities and possible attacks. These include inter alia the following: *Related-key rectangle attack on 36 rounds of the XTEA block cipher*[14] and *Meet-in-the-Middle Attacks on Reduced-Round XTEA*[15].

3 Attacks

In this section different types of attacks on block ciphers will be shortly described. These attacks play a more or less important role for the in this paper handled ciphers and will be taken up again in the security section. The attacks are: brute-force, linear cryptanalysis, algebraic cryptanalysis, differential cryptanalysis, related-key attacks, meet-in-the-middle attacks, side-channel attacks and combinations of these methods.

Brute-force. A brute-force attack tests systematically all possible keys on the cypher text. It is assumed that the attacker don't have any prior knowledge about the keys that are more probably than other keys. This type of attack is often from minor importance because it is uneconomical to decrypt the cypher text with all possible keys. The complexity of a brute-force attack act as reference for other attacks.
Linear cryptanalysis This attack requires a known-plaintext attacker ahead, i.e. the attacker knows the relative cipher text to a certain plaintext. The idea of the attack is to find linear equations for parts or. single operations of the cypher. The equations try to determine plain text bits, cipher text bits and key bits pairs with a better probability than $\frac{1}{2}$. For these attacks it is an important performance factor how many plaintext ciphertext pairs are needed.
Algebraic cryptanalysis This attack has the same objective as the linear cryptanalysis, but instead of only linear equations also polynomial equations of any degree can be used. An often problem in this context is there is no exact complexity and because of that it is necessary to use other metrics like runtime on a specific test environment.

Differential cryptanalysis. The differential cryptanalysis is a chosen-plaintext attack, i.e. the attacker can encrypt a chosen plaintext. To accomplish the attack pairs $(\Delta X, \Delta Y)$ are compared, whereby ΔX and ΔY are in each case the difference (e.g. XOR) of two values, e.g. the difference of two inputs and outputs of the algorithm. Goal of this analysis is to classify certain keys more likely.

Related-key attack. At a related-key attack it is assumed that the attacker knows not only the cipher text of the originally keys K but also the decryption with key K' which are derived from K i.e. $K' = f(K)$.

Meet-in-the-middle attacks (MITM). MITM attacks assume at least one known pair of plain- and cipher text (known- or. Chosen-plaintext). At the first step the attacker tries to filter keys i.e. the key space is limited. At the second step the right key is searched using brute-force or another attack. More details about this attack technique can be found for example in Takanori Isobe and Kyoki Shibutanii[16].

Side-channel attacks. A side-channel attack doesn't attack the algorithm itself but the physical implementation. The attacker tries for example of the duration or the power consumption of certain operations to infer information. Also the attacker can try to specifically tilt bits for example due to manipulate the applied voltage.

4 Simon and Speck

Simon and Speck is a family of lightweight block ciphers publicly released by the National Security Agency (NSA) in June 2013. [5] Simon and Speck comes with ten distinct block ciphers with differing block and key sizes. The most existing block ciphers were designed to perform well on a single platform and were not meant to provide high performance across a range of devices. The aim of Simon and Speck was to fill the need for secure, flexible, and analysable lightweight block ciphers. Each offers excellent performance on hardware and software platforms, is flexible enough to admit a variety of implementations on a given platform, and is amenable to analysis using existing techniques. Both perform very well across the full spectrum of lightweight applications, but Simon has been optimized for performance in hardware implementations, while its sister algorithm, Speck, has been optimized for software implementations. The reason why the algorithms work so well on each platform is that both are very simple constructed. So it is very easy to find efficient implementations. For algorithms such as AES it required longer time of research to find near-optimal implementations.

The Simon block cipher with an n-bit word (and hence a 2n-bit block) is denoted Simon2n, where n is required to be 16, 24, 32, 48, or 64. Simon2n with an m-word (mn-bit) key will be referred to as Simon2n/mn. For example, Simon64/128 refers to the version of Simon acting on 64-bit plaintext blocks and using a 128-bit key. The notation for the different variants of Speck is entirely analogous to that used for Simon.

4.1 Simon round function

The Simon2n encryption maps make use of the following operations on n-bit words:

- bitwise XOR, $\oplus$,
- bitwise AND, &, and
- left circular shift, S^j , by j bits.

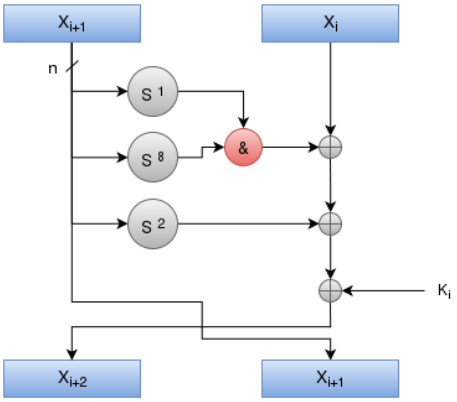


Fig. 1: The Simon Round Function. Derived from [5]

The round functions for Simon 2n take as input an n-bit round key k, together with two n-bit intermediate ciphertext words. The round function is the 2-stage Feistel map

$$R_k(x, y) = (y \oplus f(x) \oplus k, x),$$

where $f(x) = (Sx \& S^8 x) \oplus S^2 x$ and k is the round key. The inverse of the round function, used for decryption, is

$$R_k^{-1}(x, y) = (y, x \oplus f(y) \oplus k).$$

Figure 1 shows the effect of the round function R_{k_i} on the two words of sub cipher (x_{i+1}, x_i) at the i^{th} step of this process.

The round functions are composed some number of times which depends on the block and key size. Parameters for all versions of Simon are specified in Table 1.

Table 1: SIMON parameters. Derived from [5]

block size 2n	key size mn	word size n	key words m	const seq	rounds T
32	64	16	4	z_0	32
48	72	24	3	z_0	36
	96		4	z_1	36
64	96	32	3	z_2	42
	128		4	z_3	44
96	96	48	2	z_2	52
	144		3	z_3	54
128	128	64	2	z_2	68
	192		3	z_3	69
	256		4	z_4	72

4.2 Simon key schedules

The key schedule is needed to turn a key into a sequence of round keys. The Simon key schedules employ a sequence of 1-bit round constants specifically for the purpose of eliminating slide properties and circular shift symmetries. The designers provide some cryptographic separation between different versions of Simon having the same block size by defining five such sequences: $z_0, ..., z_4$. Each of these sequences is defined in terms of one of the following period 31 sequences:

$$\begin{aligned} u = u_0u_1u_2... &= 11111010001001010110000111001 10..., \\ v = v_0v_1v_2... &= 10001110111110010011100001011010..., \\ w = w_0w_1w_2... &= 10000100101100111110000110111010.... \end{aligned}$$

The first two sequences are simply $z_0 = u$ and $z_1 = v$. The other three, z_2, z_3, and z_4, have period 62 and are formed by computing the bitwise XOR of the period 2 sequence $t = t_0t_1t_2... = 01010101...$ with u, v, and w, respectively:

$$\begin{aligned} z_2 = (z_2)_0(z_2)_1(z_2)_2... &= 10101111011100000011010010011 00 \\ &\quad 01010000100011111100101101100 11..., \\ z_3 = (z_3)_0(z_3)_1(z_3)_2... &= 11011011101011000110010111100 00 \\ &\quad 00100100010100111001101000011 11..., \\ z_4 = (z_4)_0(z_4)_1(z_4)_2... &= 11010001111001101011011000100 00 \\ &\quad 00101110000110010100100111011 11..., \end{aligned} \tag{1}$$

where $(z_i)_j$ is the j^{th} bit of z_i.
The sequences u, v, and w can be generated as follows: Define 5×5 matrices U, V, and W over GF(2) by

$$U = \begin{bmatrix} 0\,1\,0\,0\,0 \\ 0\,0\,1\,0\,0 \\ 1\,0\,0\,1\,0 \\ 0\,0\,0\,0\,1 \\ 1\,0\,0\,0\,1 \end{bmatrix}, V = \begin{bmatrix} 0\,1\,1\,0\,0 \\ 0\,0\,1\,0\,0 \\ 1\,0\,0\,1\,0 \\ 0\,0\,0\,0\,1 \\ 1\,0\,0\,0\,0 \end{bmatrix}, W = \begin{bmatrix} 0\,1\,0\,0\,0 \\ 0\,0\,1\,0\,0 \\ 1\,0\,0\,1\,0 \\ 0\,0\,0\,0\,1 \\ 1\,0\,0\,0\,0 \end{bmatrix}$$

The i_{th} element of each sequence is then obtained by initializing a 5-bit linear feedback shift register to 00001, stepping i times using the corresponding matrix, and extracting the right-hand bit. Thus $(u)_i = (0,0,0,0,1)U^i(0,0,0,0,1)^t$.

Let $c = 2^n - 4 = 0xff\cdots fc$. For Simon2n with m key words $(k_{m-1}, ..., k_1, k_0)$ and constant sequence z_j, round keys are generated by

$$k_{i+m} = \begin{cases} c \oplus (z_j)_i \oplus k_i \oplus (I \oplus S^{-1})S^{-3}k_{i+1}, & \text{if } m = 2 \\ c \oplus (z_j)_i \oplus k_i \oplus (I \oplus S^{-1})S^{-3}k_{i+2}, & \text{if } m = 3 \\ c \oplus (z_j)_i \oplus k_i \oplus (I \oplus S^{-1})(S^{-3}k_{i+3} \oplus k_{i+1}, & \text{if } m = 4 \end{cases}$$

for $0 \leq i < T - m$. In Figure 2 is the key schedules represented and which version-dependent choice of constant sequence z_j have to used is shown in Table 1. Note that yourself choose the first m key words which will used as the first m round keys. They are loaded into the shift registers with k_0 on the right and k_{m-1} on the left. Only the next ones will be generated with key schedule.

4.3 Speck round function

The Speck2n encryption maps make use of the following operations on n-bit words:

- bitwise XOR, $\oplus$,
- addition modulo 2^n, $+$,
- left and right circular shifts, S^j and S^-j , respectively, by j bits.

For $k \in GF(2)^n$, the key-dependent Speck2n round function is the map R_k: GF(2)$^n\times$ GF(2)$^n \rightarrow$ GF(2)$^n\times$ GF(2)^n defined by

$$R_k(x, y) = ((S^{-\alpha}x + y) \oplus k, S^{\beta}y \oplus (S^{-\alpha}x + y) \oplus k),$$

with rotation amounts $\alpha = 7$ and $\beta = 2$ if $n = 16$ (block size = 32) and $\alpha = 8$ and $\beta = 3$ otherwise.

The inverse of the round function, necessary for decryption, uses modular subtraction instead of modular addition, and is given by

$$R_k^{-1}(x, y) = (S^{\alpha}((x \oplus k) - S^{-\beta}(x \oplus y)), S^{-\beta}(x \oplus y)),$$

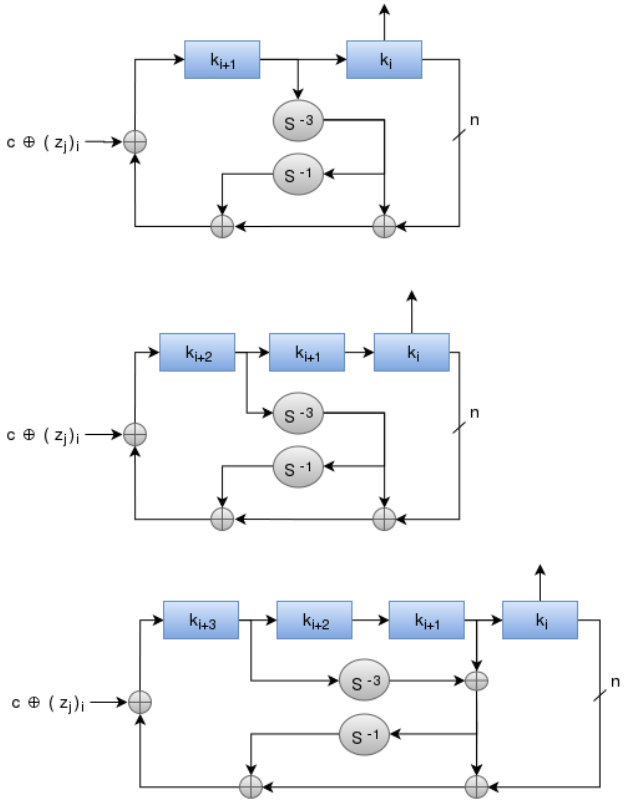


Fig. 2: The Simon two, three, and four-word key expansion. Derived from [5]

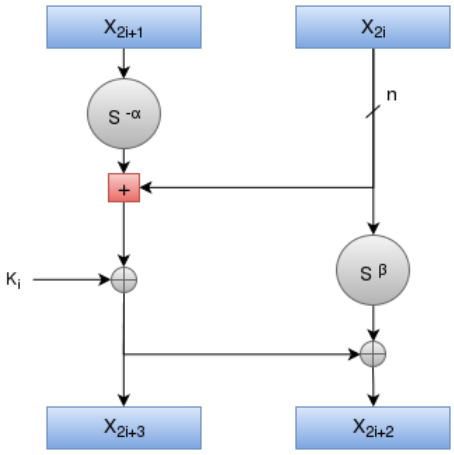


Fig. 3: Speck round function; (x_{2i+1} , x_{2i}) denotes the sub-cipher after i steps of encryption. Derived from [5]

Table 2: Speck parameters. Derived from [5]

block size 2n	key size mn	word size n	key words m	rot α	rot β	rounds T
32	64	16	4	7	2	22
48	72	24	3	8	3	22
	96		4			23
64	96	32	3	8	3	26
	128		4			27
96	96	48	2	8	3	28
	144		3			29
128	128	64	2	8	3	32
	192		3			33
	256		4			34

Parameters for all versions of Speck are specified in Table 2.

The Speck key schedules take a key and from it generate a sequence of T key words $k_0, ..., k_{T-1}$, where T is the number of rounds. The effect of the single round function R_{k_i} is shown in Figure 3. Encryption is then the composition $R_{k_{T-1}} \circ ... \circ R_{k_1} \circ R_{k_0}$, read from right to left.
Note that Speck can be realized as the composition of two Feistel-like maps with respect to two different types of addition, namely,

$$(x, y) \mapsto (y, (S^{-\alpha}x + y) \oplus k) and (x, y) \mapsto (y, S^{\beta}x \oplus y).$$

This decomposition is pictured in Figure 4.

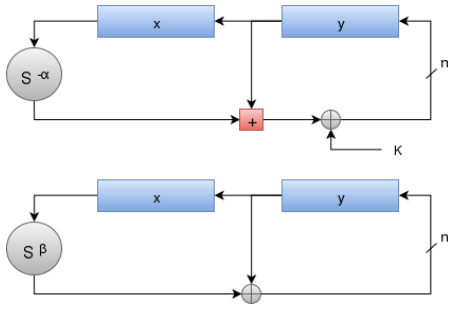


Fig. 4: Speck round function decomposed into Feistel-like steps. Derived from [5]

4.4 Speck key schedules

The Speck key schedules use the own round function to generate round keys k_i. This is useful cause we don't need to implement a new method. Let K be

a key for a Speck2n block cipher. We can write $K = (l_{m-2}, ..., l_0, k_0)$, where $l_i, k_0 \in GF(2)^n$, for a value of m in 2, 3, 4. Sequences k_i and l_i are defined by

$$l_{i+m-1} = (k_i + S^{-\alpha} l_i) \oplus i$$

and

$$k_{i+1} = S^{\beta} k_i \oplus l_{i+m-1}.$$

The value k_i is the i^{th} round key, for $0 \leq i < T$. See Figure 5.

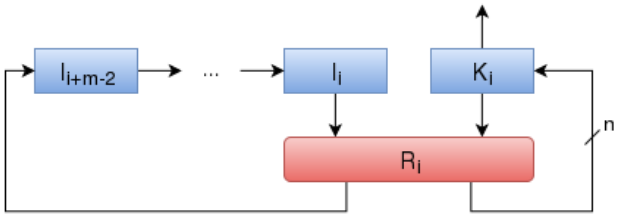


Fig. 5: Speck key expansion, where R_i is the Speck round function with i acting as round key. Derived from [5]

4.5 Security Analysis

Simon and Speck attacks was studied in many articles since its publication in 2013. The most published attacks on Simon and Speck are of the reduced-round variety. The goal of this sort of analysis is to determine the maximal number of rounds that would be susceptible to a theoretical attack (i.e., anything better than an exhaustive key search). A measure of security is the number of rounds that have been attacked, as a percentage of the total. So far no published attack makes it more than about 74% of the way through any version of Simon or Speck. The best attacked rounds for all versions of Simon was with the improved linear hull and differential attacks with dynamic key-guessing technique [17], [18] . The best attacked rounds in Speck is with differential cryptanalysis and improved differential cryptanalysis [19], [20]. The results are shown in table 3.

The content of the table 3 is simple: there are no attacks on any member of the Simon or Speck families, and each block cipher maintains a healthy security margin.

5 KATAN

`KATAN/KTANTAN` is a family of hardware oriented block ciphers designed in 2009 by Chrstophe de Canniere, Orr Dunkelman, and Miroslav Knezevic [7]. In summary the family consists of six block ciphers. They are divided into two sets of

Table 3: Security of Simon and Speck derived from [6] Table 1

size	alg	rounds		ref
		total	attacked	
32/64	Simon	32	23 (72%)	[17]
	Speck	22	14 (64%)	[19]
48/72	Simon	36	24 (67%)	[17]
	Speck	22	14 (64%)	[19]
48/96	Simon	36	25 (69%)	[17]
	Speck	23	15 (65%)	[19]
64/96	Simon	42	30 (71%)	[17]
	Speck	26	18 (69%)	[19]
64/128	Simon	44	31 (70%)	[17]
	Speck	27	19 (70%)	[19]
96/96	Simon	52	37 (71%)	[18],[17]
	Speck	28	16 (57%)	[19]
96/144	Simon	54	38 (70%)	[17]
	Speck	29	17 (59%)	[19]
128/128	Simon	68	49 (72%)	[18],[17]
	Speck	32	17 (53%)	[19]
128/192	Simon	69	51 (74%)	[17]
	Speck	33	18 (55%)	[19],[20]
128/256	Simon	72	53 (74%)	[17]
	Speck	34	19 (56%)	

three `KATAN` block ciphers with 32, 48 or 64-bit block size and three `KTANTAN` block ciphers with the same block size. They share the same 80-bit key size and security level. The difference between `KATAN` and `KTANTAN` is that at `KTANTAN` the key is burnt into the device and cannot be changed. Therefor `KTANTAN` are very small block ciphers and more compact than `KATAN` and can only be used in cases where the device is initialized with one key.

- `KATAN32` has 802 GE and an encryption speed of 12.5 KBit/sec.
- `KATAN48` has 927 GE and an encryption speed of 18.8 KBit/sec.
- `KATAN64` has 1054 GE and an encryption speed of 25.1 KBit/sec.
- `KTANTAN32` has 462 GE and an encryption speed of 12.5 KBit/sec.
- `KTANTAN48`, which is the recommend for RFID tags has 588 GE and an encryption speed of 18.8 KBit/sec.
- `KTANTAN64` has 688 GE and an encryption speed of 25.1 KBit/sec.

A comparison with some other ciphers is shown in Table 4.
The specific design goals from the developers were as follows:[7]

- For an n-bit block size, no differential characteristic with probability greater than 2^{-n} exists for 128 rounds (about half the number of rounds of the cipher).
- For an n-bit block size, no linear approximation with bias greater than $2^{-n/2}$ exists for 128 rounds.

Table 4: Comparison of Ciphers Designed for Low-End Environments (optimized for size). Derived from [7].

Cipher	Block (bits)	Key (bits)	Size (GE)	Gates per Memory Bit	Throughput[1] (Kb/s)	Logic Process
AES-128	128	128	3400	7.97	12.4	0.35μ
AES-128	128	128	3100	5.8	0.08	0.13μ
HIGHT	64	128	3048	N/A	188.25	0.25μ
mCrypton	64	64	2420	5	492.3	0.13μ
DES	64	56	2309[2]	12.19	44.4	0.18μ
DESL	64	56	1848[2]	12.19	44.4	0.18μ
PRESENT-80	64	80	1570	6	200	0.18μ
PRESENT-80	64	80	1000	N/A	11.4	0.35μ
Grain	1	80	1294	7.25	100	0.13μ
Trvivium	1	80	749	2[3]	100[4]	0.35μ
KATAN32	32	80	802	6.25	12.5	0.13μ
KATAN48	48	80	927	6.25	18.8	0.13μ
KATAN64	64	80	1054	6.25	25.1	0.13μ
KTANTAN32	32	80	462	6.25	12.5	0.13μ
KTANTAN48	48	80	588	6.25	18.8	0.13μ
KTANTAN64	64	80	688	6.25	25.1	0.13μ

[1] —A throughput is estimated for frequency of 100 KHz.
[2] —Fully serialized implementation (the rest are only synthesized).
[3] —This is a full-custom design using C2MOS dynamic logic.
[4] —This throughput is projected, as the chip requires higher frequencies.

– No related-key key-recovery or slide attack with time complexity smaller than 280 exists on the entire cipher.
– High enough algebraic degree for the equation describing half the cipher to thwart any algebraic attack.

Also they rank the possible design targets as follows:[7]

– Minimize the size of the implementation.
– Keeping the critical path as short as possible.
– Increase the throughput of the implementation (as long as the increase in the footprint is small).
– Decrease the power consumption of the implementation.

We concentrate on KATAN in this paper so KTANTAN is not examined in more detail.

5.1 Round function and key schedule

We have three variants of the KATAN ciphers. KATAN32, KATAN48 and KATAN64. The main difference between them is the block size and that KATAN48 executes the nonlinear function twice and KATAN64 three times with the same round key per round. For example, we use KATAN32 to describe the key schedule. The plaintext (bit 0-31) is used to generate the ciphertext. For that it is loaded into two registers L_1(bit 19-31) and L_2(bit 0-18) and L_1 and L_2 are shifted to the left (bit i is shifted to position i+i) each round. After that both registers get updated each round with the following nonlinear functions f_a and f_b for 254 rounds.

$$f_a(L_1) = L_1[x_1] \oplus L_1[x_2] \oplus (L_1[x_3] * L_1[x_4]) \oplus (L_1[x_5] * IR) \oplus k_a$$

$$f_b(L_2) = L_2[y_1] \oplus L_2[y_2] \oplus (L_2[y_3] * L_2[y_4]) \oplus (L_2[y_5] * L_2[y_6]) \oplus k_b$$

IR is an irregular update rule (which is only used if IR = 1) shown in Table 6, k_a and k_b are two subkey bits. The bits for x_1 and y_1 for each variant are shown in Table 5[7]. Afther the round the LSB of L_1 is the output of f_b and the LSB of L_2 is the output of f_a.
The key schedule for all variants of the KATAN family accepts a 80-bit key K with the secret key K_0 - K_{79} and the following mapping:

$$k_i = \begin{cases} K_i & \text{for } i = 0...79 \\ k_{i-80} \oplus k_{i-61} = \oplus k_{i-50} \oplus k_{i-13} & \text{Otherwise} \end{cases} \quad (2)$$

The values of k_a and k_b for a round i are k_{2i} and k_{2+i}. And for that $k_a \mid\mid k_b = k_{2i} \mid\mid k_{2+i}$. Figure 6[7] shows a round of the KATAN family.

Table 5: Parameters of the KATAN family. Derived from [7].

Cipher	$\|L_1\|$	$\|L_2\|$	x_1	x_2	x_3	x_4	x_5
KATAN32/KTANTAN32	13	19	12	7	8	5	3
KATAN48/KTANTAN48	19	29	18	12	15	7	6
KATAN64/KTANTAN64	25	39	24	15	20	11	9

Cipher	y_1	y_2	y_3	y_4	y_5	y_6
KATAN32/KTANTAN32	18	7	12	10	8	3
KATAN48/KTANTAN48	28	19	21	13	15	6
KATAN64/KTANTAN64	38	23	33	21	14	9

Table 6: Sequence of the irregular updates. 1 IR is used, 0 IR is not used. Derived from [7].

Rounds	0-9	10-19	20-29	30-39	40-49	50-59
Irregular	1111111000	1101010101	1110110011	0010100100	0100011000	1111000010
Rounds	60-69	70-79	80-89	90-99	100-109	110-119
Irregular	0001010000	0111110011	1111010100	0101010011	0000110011	1011111011
Rounds	120-129	130-139	140-149	150-159	160-169	170-179
Irregular	1010010101	1010011100	1101100010	1110110111	1001011011	0101110010
Rounds	180-189	190-199	200-209	210-219	220-229	230-239
Irregular	0100110100	0111000100	1111010000	1110101100	0001011001	0000001101
Rounds	240-249	250-253				
Irregular	1100000001	0010				

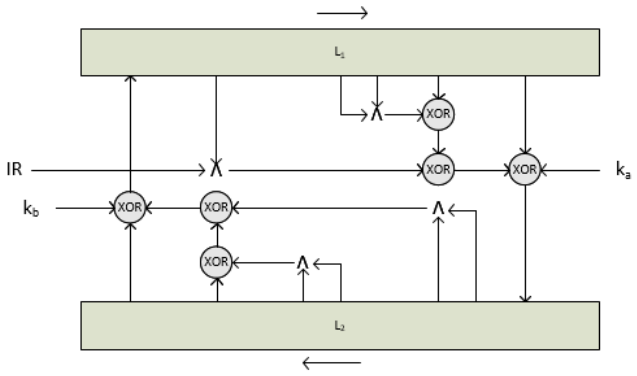


Fig. 6: Outline of a round of the KATAN family. Derived from [7].

5.2 Security Analysis

The first two mentioned design goals ensure that no differential-linear attack or a boomerang attack exist for the entire cipher. Only one successful attack is known at the moment. The attack on KTANTAN32 was presented by Andrey Bogdanov and Christian Reichberger at Selected Areas in Cryptography 2010 [21]. The meet-in-the-middle attack can find the key with a time complexity of 2^{79}. The other variants of the KATAN family are not affected by this attack and still secure. Mainly differential, meet-in-the-middle, algebraic and side channel attacks have been executed on KATAN. First we look at 2 differential attacks.

The authors from [22] used a known chosen plaintext scenario with multiple instances with the same key to attack KATAN. They get 16 differentials for 95 rounds which makes it possible to break 115 rounds of KATAN with a time and data complexity of 2^{32}.

In [23] multiple KATAN instances with a difference in plaintext and key. The attack breaks 120 rounds of KATAN with a time and data complexity of 2^{31}.

The third attack is a meet-in-the-middle-attack from [24]. They break 153 rounds from KATAN with a time and data complexity of $2^{78.5}$ and a memory complexity of 2^{76}. So this is a more theoretical attack and not practicable with the current technologies.

Another meet-in-the-middle-attack is from [25]. 174 rounds are broken with a time complexity of $2^{78.5}$, a memory complexity of $2^{26.6}$ and a data complexity of $2^{27.6}$.

An algebraic attack is from [26]. The attack breaks 79 rounds with a time complexity of $2^{76.5}$ and a data complexity of 2^5.

A little better is the algebraic attack from [8]. The attack breaks 80 rounds with a time complexity of 2^{72} and a data complexity of 2^7.

In Table 7 a comparison of the attacks is shown.

Table 7: Comparison of the attacks

attack	method	rounds	time	memory	data
[22]	differential	115	2^{32}	—	2^{32}
[23]	differential	120	2^{31}	—	2^{31}
[24]	meet-in-the-middle	153	$2^{78.5}$	2^{76}	$2^{78.5}$
[25]	meet-in-the-middle	174	$2^{78.5}$	$2^{26.6}$	$2^{27.6}$
[26]	algebraic	79	$2^{76.5}$	—	2^5
[8]	algebraic	80	2^{72}	—	2^7

6 LED

Light Encryption Device is a symmetric block cipher which was published from Guo et al.[27] in 2011. The cipher is lightweight and can efficiently be implemented in hardware. Because of these properties the authors suggest that LED is basically suitable for encryption and decryption in the IoT area. A concrete use case is the secure storage and transmission of RFID tags.

6.1 Mathematical notations

This paper uses the following mathematical operators:

Integer addition: The addition of two integer numbers modulo 2^n is written down as $x \boxplus y$. Where $x, y \in \mathbb{Z}_{2^n}$.The value of n results from the context.

Exclusive or (xor): $x \oplus y$

Bitwise Shift: The logical shift from x to y bits is written down as $x << y$ (to left). The logical shift from x to y to right is written down as $x >> y$.

6.2 Encryption

LED uses a block size of 64 bits. The key length is 64 bit (LED-64) or 128 bit (LED-128). Even key length between 64 bit and 128 bit are basically possible for example 80 bit. In this case the remaining bits will be padded with the prefix of the key (padding). We call the actual bytes of the blocks to be encrypted as a `State`. The state and the keys will be written down as a (4×4) matrix. The matrix entries of the state and key are respectively 4 bit blocks (Nibble) and represents elements of the body $\mathbb{F}_{2^4}$. With a key length of 128 bit, there are accordingly 2 matrices, each with 64 bit. A round consists of 4 sub-steps: `AddConstants`, `SubCells`, `ShiftRows` and `MixColumnsSerial`. The number of rounds are 32 (LED-64) or 48 (LED-128). Below each sub-step of a round will be described.

AddConstants. At the beginning of each round the operation AddConstants is performed. For this purpose, the State with the following matrix is added bitwise using XOR:

$$\begin{pmatrix} 0 & (rc_5||rc_4||rc_3) & 0 & 0 \\ 1 & (rc_2||rc_1||rc_0) & 0 & 0 \\ 2 & (rc_5||rc_4||rc_3) & 0 & 0 \\ 3 & (rc_2||rc_1||rc_0) & 0 & 0 \end{pmatrix}$$

The bit vector $(rc_5, rc_4, rc_3, rc_2, rc_1, rc_0)$ is initialized with 0 before the first round. Before each addition the matrix on the last round is taken, the bit vector is shifted by one positon to the left and rc_0 is set to $rc_5 \oplus rc_4 \oplus 1$.
SubCells. In this step every Nibble of the state will be replaced by another (Sbox). The Sbox was borrowed by the block cypher `PRESENT` [28].

Table 8: PRESENT Sbox.

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

ShiftRows. For $i = 1, 2, 3, 4$ each i-th is shifted cyclical $i - 1$ positions to the left.
MixColumnsSerial. Each column vector v of the state is replaced with $M \cdot v$. To multiply the elements, the polynomial $X^4 + X + 1$ is used.

$$M = \begin{pmatrix} 4 & 2 & 1 & 1 \\ 8 & 6 & 5 & 6 \\ B & E & A & 9 \\ 2 & 2 & F & B \end{pmatrix}$$

After every step (4 rounds) and at the start of the encryption is also still the operation `addRoundKey` performed. In addition to that the key is added to the state by use of XOR. In the case of LED-128 the two keys switch after each step. The name round key is misleading at this point because the key never changes. An outline of the encryption process is shown in Fig. 7.

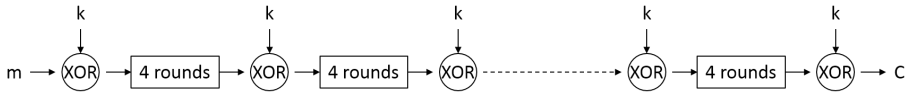


Fig. 7: LED encryption

6.3 Security

LED is very similar to AES with regard to the rounding operation. The round number is comparatively set high with 32 (LED-64) or 48 (LED-128). AES-128 uses 10 rounds. However, should be noted that AES adds the round key each round. LED uses the operation AddConstants instead. Therefore, one step (4 rounds) in LED is equivalent to 4 rounds single-key AES. This comparison is drawn because the security for 4 rounds single-key AES was further explored [29][30].

In the following, an overview of the attacks is given that exists on LED[5] are given. Vincent Grosso et al.[31] research algebraic attacks on LED. Evaluated is the time needed for the key recovery with different number of key bits. At full number of rounds, the key can be recovered in about 5 minutes[6] when 16

[5] The given values refer to `LED-64`, if not otherwise specified.
[6] A statement about the used test environment is not taken.

of 64 key bits are unknown. Result for the case that more than 16 key bits are unknown are not provided. It is also noted that the attack is only an advantage over a brute-force attack brings if at least 9 key bits are unknown.

In the papers by Xinjie Zhao et al.[32] and Philipp Jovanovic et al.[33] algebraic attacks are combined with side-channel attacks. Both attacks require that the same plain text can be encrypted with the same key twice. At the second encryption in the 30 round after the operation SubCells it is attempted to produce an error in the first entry of the state so that at this entry is a random value. Based on the difference of the outputs and the inverse rounds a system of equations is formed with that the key space of $2^{19} \sim 2^{25}$ or $2^{6} \sim 2^{17}$ can be reduced [33][32].

Takanori Isobe und Kyoji Shibutani[16] research MITM attack on lightweight ciphers. It is about a chosen-plaintext attack with the objective to recover the key. In the case `LED-64` rounds are attacked where 2^8 plain-cipher-text pairs and 2^{56} encryptions are needed. At `LED-128` are 16 rounds at a time complexity of 2^{112} and a data complexity of 2^{16} achieved.

Mendel et al.[34] describe differential attacks in the single- and related-key context. Objective of the attack is the key-recovery. In the related-key attack the authors succeed to attack in 16 rounds at $2^{62.7}$ encryptions. However, the attack also has with $2^{62.7}$ a high data- and memory complexity. In the paper is also an attack shown on `LED-128` provided that K_0 will guess (data complexity 2^{64}) the complete key $K_0||K_1$ at 2^{96} encryptions can be recovered.

That in 2015 published paper by Ivica Nikoli et al.[29] attacks with a combination of MITM and differential crypto analysis the so far most rounds (without side-channel) with 20 (`LED-64`) or 40 (`LED-128`).In contrast to the previously noted attacks its objective is not the key-recovery. Instead at a successful attack the attacker can distinguish a randomly permutation of the 2^k possible permutations of the used key (distinguish attack). This attack is less threatening than a key recovery but on this basis with less complexity further attacks can be accomplished. The results are as follows. In the case `LED-64`20 rounds at a time complexity of $2^{60.2}$ and a data complexity of $2^{61.5}$ are attacked. For `LED-128` are 10 rounds with $2^{60.3}$ encryptions and a memory complexity of 2^{60} affected.

A summary of the attacks is shown in table 9[29]. In addition to the complexities, the attack conditions (single-key (SK), related-key (RK), chosen-key (CK)), the number of attacked rounds and the effect of the attack (key-recovery (KR), distinguisher (D)) listed.

7 Tea family

7.1 Introduction TEA

The Tiny Encryption Algorithm (TEA) was developed with the objective to design a high performance and mathematical not to complicated encryption algorithm which in particular also for use on low-performing small computers in the IoT environment is. The algorithms of the TEA-family are variants of

Table 9: Attacks on LED[29]

Cipher	Attack	Type	Rounds	Time	Data	Memory	Ref
LED-64 (32 rounds)	MITM (SK)	KR	8	2^{56}	2^{8}	2^{11}	[16]
	Linear/Differential (CK/RK)	KR	16	$2^{62.7}$	$2^{62.7}$	$2^{62.7}$	[34]
	Linear/Differential (CK)	D	15	2^{16}	–	2^{16}	[27]
	MITM/Differential (CK)	D	16	$2^{33.5}$	–	2^{32}	[29]
	MITM/Differential (CK)	D	20	$2^{60.2}$	–	$2^{61.5}$	[29]
LED-128 (48 rounds)	MITM (SK)	KR	16	2^{112}	2^{16}	2^{19}	[16]
	Linear/Differential (CK/RK)	KR	24	2^{96}	2^{64}	2^{32}	[34]
	Linear/Differential (CK/RK)	D	27	2^{16}	2^{16}	2^{32}	[27]
	MITM/Differential (CK)	D	32	$2^{33.5}$	–	2^{32}	[29]
	MITM/Differential (CK)	D	40	$2^{60.3}$	–	2^{60}	[29]

a Feistel Cipher and thus block ciphers. TEA encrypts 64 bit blocks which are directly split into 32 bit blocks. The classical TEA algorithm uses a 128-bit length key. TEA is a round based encryption method. The number of the used rounds are variable but 32 Tea cycles are recommended. Due to the symmetrical construction of the encryption algorithm (see point 7.3) is one cycle in TEA equivalent to two Feistel rounds. [10]

The algorithm exceeds the performance of DES (see point 7.7) and can be implemented in all programming languages. For many common programming languages exist reference implementations which can be used with little effort. With a strength of 32 cycles is the test implementation 60% faster than the reference implementation with 56-Bit DES. The encryption strength of TEA can be further increased by increasing the encryption cycle. [9]

7.2 "The golden number"

To counter attacks which try to exploit the symmetric of the encryption rounds it is a frequent practice by some encryption methods to include golden numbers at each round. This has the effect that there are no bits which do not change in sequential rounds. The classic golden number is defined as: [35]

$$\frac{1+\sqrt{5}}{2} \tag{3}$$

TEA uses a derived constant from the golden number: [9]

$$(\sqrt{5}-1)2^{31} \tag{4}$$

This constant initialized the variable delta and equates to a rounded integer: [9]:

$$delta = 2654435769_{10} = 9E3779B9_{16} \tag{5}$$

The mathematical definition of the constant should counter the suspicion that it is not a random number but a conscious weakening of the algorithm installing

a backdoor. In cryptography it is for this reason a frequent practice not to use hardcoded random chosen numbers but to generate them by a simple comprehensible mathematical operation.[35]

7.3 Encryption algorithm

As already described under 7.1TEA is a block-cipher encryption method which can only encrypt 64 bit blocks. To encrypt the 64-bit block gets split in two 32 bit blocks. One block named L (left) the other R (right). The blocks get interchanged after each encryption round. The 128-bit key gets split in 4 sub keys and named with K[0-3]. The first 32 bit are in key K[0], the second 32 bit are in key K[1] etc. the encryption steps are shown in 8 and get formally introduced under section 7.5. For further details, you can also look into the mentioned sources. [10]

7.4 TEA encryption routine characteristics

The TEA encryption algorithm has the following characteristics:

- As usual for a Feistel cipher every round I has 2 inputs Left(i) and Right(i) from the second round it is in each case the output of the opposite side from the round before. In each case the variables will be initialised with one half of the block to be encrypted.
- The in every round used key K[i] is a part of the 128 Bit long key K.
- The constant delta will be initialised with a golden number derived constant and ensures that the partial keys generate different ciphers and have no relevant cryptographic significance. (see section 7.2).
- The encryption algorithm don't use random numbers i.e. identical text by the same key leads to the identical cipher.

7.5 Formal definition of the encryption

To clarify the functionality are hereinafter the encryption functions of the TEA block cipher listed. Figure 8 shows the encryptions steps graphically. As introduced in section 7.3 the respectively part of the used key is named K[0-3] and the respectively part of the current block to be encrypted is named with Right(i) or. Left(i). Delta[i] is a modification of the under section 7.2 introduced golden number. As described delta is defined as follows:

$$delta = (\sqrt{5} - 1)2^{31} = 2654435769_{10} = 9E3779B9_{16} \tag{6}$$

One TEA round consists of two Feistel rounds and for that reason the control variable i increases by 2 each round. The variable i indicates the Feistel rounds. In one TEA round the following operations were executed:

$$delta[i] = (i + 1)/2 * delta \tag{7}$$

$$Left[i+1] = Right[i] \tag{8}$$

$$Right[i+1] = Left[i] \boxplus F(Right[i], K[0,1], delta[i]) \tag{9}$$

$$Left[i+2] = Right[i+1] \tag{10}$$

$$Right[i+2] = Left[i+1] \boxplus F(Right[i+1], K[2,3], delta[i]) \tag{11}$$

$$F(M, K[j,k], delta[i]) = ((M << 4) \boxplus K[j]) \oplus (M \boxplus delta[i]) \oplus ((M >> 5) \boxplus K[k]) \tag{12}$$

Equation 8 and 9 form the first step of a TEA round (the first Feistel round). The equations 10 and 11 close the TEA round (the second Feistel round). F() referred the so called round function (equation 12) which contains the significant steps of the cryptographically operations. The function is always the same but get called with different parameters each Feistel round.

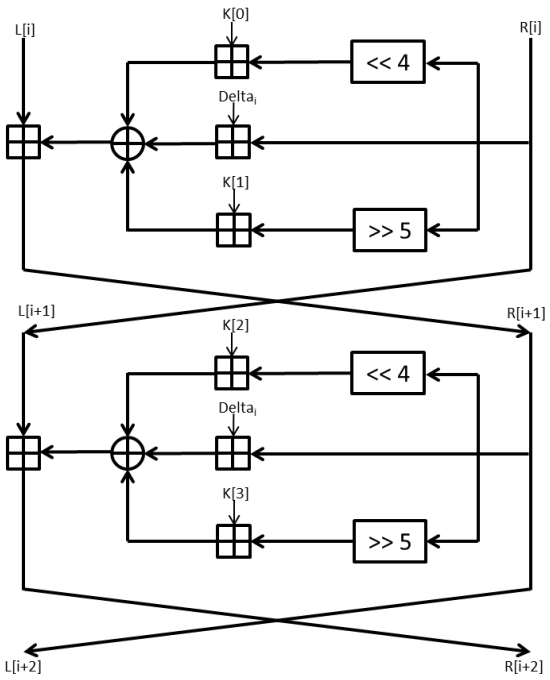


Fig. 8: Encryption steps TEA, Source: Based on [36]

7.6 Vulnerabilities of TEA

TEA has been handled in some crypto analysis and examined for vulnerabilities. The current known vulnerabilities of the original TEA implementation are:

- Hash collisions used as hash functions: TEA wasn't developed for being used as a hash function and does not meet the central condition of preimage resistance for cryptographically hash functions. That means it is possible with comparatively little effort to find to a given hash value Y an input value X its hash value after using the hash function also maps to Y. Therefore, specifically collisions can be calculated to a given hash value. [11] [37]
- Key cracking: Vulnerability for simple attempts of keys (brute force). In combination with a known plaintext-ciphertext pair the necessary iterations strongly decrease and the efficiency of the attack strongly increase.[11]
- Equivalent keys: Because of a constructional vulnerability at the TEA encryption algorithm every key to decrypt a cipher has 3 equivalent keys which can although be used to decrypt the cipher. This means that the effective key space of a 128 Bit long key is reduced to 126 Bit.[11]

The following describes detailed and exemplarily the equivalent key vulnerability of the TEA algorithm depended on [38] and derived the mathematical backgrounds. If two different keys (K and K') in an encryption system with identical plaintext generate the identical cipher both keys K and K' were called as equivalent. Following equation express this:

$$E_K(T) = E_{K'}(T) \tag{13}$$

Conversely that means that a with K encrypted cipher can be decrypted with K':

$$D_{K'}(E_K(T)) = T \tag{14}$$

At a good encryption system the claim should be that there are no equivalent keys. The number of the equivalence classes of the ciphers are in this case 2^k whereby k is the key length in Bit. For TEA with a 128 Bit key length it should be $2^{128} \approx 3,4 * 10^{38}$different equivalence classes. Analysis show that in TEA are only 2^{126} different equivalence classes with a key length of 128 Bit. In this case there are only 2^{126}differentiated from each other cyphers by a given plaintext T. For every cipher there are 4 each other equivalent keys $K_0 ... K_3$ by which the plaintext can be decrypted. The following example illustrates this:

$$\forall a, b \in \mathbb{Z}_{2^{32}} \tag{15}$$

$$2^{31} \boxplus 2^{31} = 0 \tag{16}$$

$$a \boxplus 2^{31} = a \boxplus 80000000_h \tag{17}$$

This means:

$$a \boxplus b = (a \oplus 80000000_h) \boxplus (b \oplus 80000000_h) \tag{18}$$

In this way the round function of TEA can be manipulated:

$$F(M, K[j,k], delta[i]) = F(M, (K[j] \oplus 80000000_h, K[k] \oplus 80000000_h, delta[i])) \tag{19}$$

Every 128 Bit key K0K3 has three equivalent keys in the form of:

$$(K[0], K[1], K[2] \oplus 80000000_h, K[3] \oplus 80000000_h \tag{20}$$

$$(K[0] \oplus 80000000_h, K[1] \oplus 80000000_h, K[2], K[3]) \tag{21}$$

$$(K[0] \oplus 80000000_h, K[1] \oplus 80000000_h, K[2] \oplus 80000000_h, K[3] \oplus 80000000_h) \tag{22}$$

So that a 128 Bit key with the TEA encryption has only a key space of 126 Bit ant thereby the security from a 126 Bit key. [38]

7.7 Performance

As described under section 7.1 the primary goal of developing TEA was to achieve a high performance. In an exemplarily test under a Java environment TEA was 18 times faster than the Java provided DES implementation. Further tests have shown that TEA (128 Bit, 32 iterations) are 60% faster than 56 Bit DES and 4 times faster than 168 Bit 3DES. Because for the following described block versions Block TEA and XXREA it is explicitly recommended by the authors to use larger data blocks a further performance increase is expected.[11]

7.8 Further development XTEA (eXtended TEA)

The XTEA (eXtended TEA) algorithm is a further development of TEA and corrects et al. the under section 7.6 described vulnerability of the equivalent keys. Like TEA works XTEA with 64 Bit blocks and a 128 Bit key length. Recommended are also 64 encryption rounds. [14] The improvements compared to TEA were achieved due a more complex key management and a change of the Shift, XOR and addition operations. [39] Due to the current state of research even XTEA isn't an encryption method without vulnerabilities. It exists descriptions of successful attacks against XTEA due exploitation of a related key vulnerability[14]. The XTEA algorithm was aware weakened due a partly significant decrease of the encryption cycles. It can however be assumed that due an appropriate greater effort an attack is also applicable at the recommended 64 encryption rounds. These attacks are described inter alia in [40], [14] and [15].

7.9 Modification Block TEA

Block TEA was published simultaneously with XTEA and differs only slightly technically from XTEA. In contrast to XTEA Block TEA don't need a fixed block size but it can also work with blocks of any size. This means that Block TEA don't need an operation mode to ensure confidentiality and authenticity. Block TEA is applied directly to the entre message. Internally the round function (see 7.3) is iteratively and cyclically applied to the entire message The used round function is identical to XTEA. So that Block TEA has the same vulnerabilities as XTEA. [15] [39]

7.10 Further development Corrected Block TEA (also referred as XXTEA)

Corrected Block TEA or XXTEA is an in 1998 published further development of Block TEA. As Block Tea it does not have a fixed block size and can be applied to the entire message. The goal to develop XXTEA was to correct the known vulnerabilities of Block TEA. For this some changes have been made in the round function. The reference implementation of XXTEA Correction to xtea is available at [41]. Also for XXTEA it already exists documented successful attacks. The paper Cryptanalysis of XXTEA [42] describes a successful chosen plaintext attack with 2^{59} plain- ciphertext pairs.

8 PRESENT

PRESENT is a lightweight block cipher introduced in 2007 by Orange Labs, Ruhr University Bochum and the Technical University of Denmark. [43] Present is designed to met the constraints of IoT specified above. [43] focused on security and hardware efficiency when designing the algorithm. The block size is 64-bit and the key size is either 80-bit or 128-bit. The most compact hardware implementation of PRESENT needs 1570 (GE) (Assumed 32-bit XOR = 80 GE, 32-bit arithmetic ADD = 148 GE, 192-bit FF = 1344 GE and SHIFT = 0 GE) [43] and is therefore competitive with today's leading compact stream ciphers, which need 1300-2600 GE according to [4].

8.1 Algorithm Specifications

PRESENT is a classical substitution permutation network (SPN) consisting of 31 rounds. At first 32 round keys are generated. The first 31 rounds consits of an XOR operation to introduce a round key K_i for $1 \leq i \leq 32$, where K_{32} is used for post-whitening. Post-whitening obfuscates the structure of the linear bitwise permutation and the non-linear substitution layer of round 31. Each of the 31 rounds exists of three operations. First the current round key is applied to the block being encrypted. Then an S-Box is performed that holds Shannon's property of confusion.[43] Confusion means that each character of the ciphertext

should depend on several parts of the key. The last step of each round is a permutation. We will now explain the operations in more detail. Figs. 9 and 10 give an illustrative presentation.

```
generateRoundKeys()
for i=1 to 31 do
    addRoundKey(STATE, K_i)
    sBoxLayer(STATE)
    pLayer(STATE)
end for
addRoundKey(STATE, 32)
```

Fig. 9: A top-level algorithmic pseudocode of PRESENT (derived from [43])

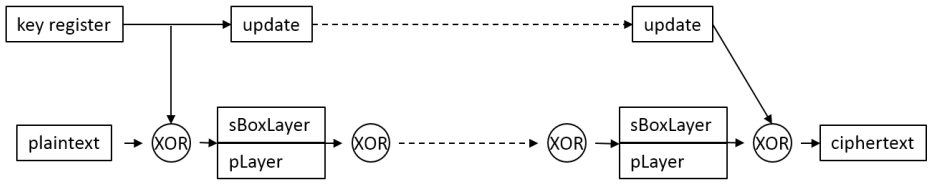


Fig. 10: A top-level algorithmic flowdiagramm of PRESENT (derived from [43])

– *addRoundKey*

The round key $K_i = K^i_{63} \dots K^i_0$ for $1 \leq i \leq 32$ is applied to the 64-bit block $b_{63} \dots b_0$ for $0 \leq j \leq 63$ by a bitwise exclusive OR.

$$b_j \rightarrow b_j \oplus K^i_j \quad i \in \{1, \dots, 32\},\ j \in \{0, \dots, 63\} \tag{23}$$

– *sBoxLayer*

PRESENT uses a 4-bit to 4-bit non-linear S-Box. Therefore the current block is considered as a sixteen 4-bit word $w_{15} \dots w_0$ where $w_i = b_{4 \cdot i+3} || b_{4 \cdot i+2} || b_{4 \cdot i+1} || b_{4 \dots i}$ for $0 \leq i \leq 15$. The S-Box itself looks as follows

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
S[x]	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

– *pLayer*

- In the bit permutation layer the bit i is moved to bit position P(i).

$$P(i) = i \cdot 16 \mod 63 \quad i \in \{0, ..., 63\} \tag{24}$$

– *Key schedule*

PRESENT can be used with either 80- or 128-bit keys. We will focus on the 80-bit key generation. For more information on the 128-bit version, see [43]. The 80-bit key K, is represented as $k_{79}...k_0$. At round i the 64 bit round key $K_i = K_{63}...K_0$ consists of the 64 leftmost bits of the current key K.

$$K_i = K_{63}...K_0 = k_{79}...k_{16} \tag{25}$$

The update routine consists of three parts. First the key register is rotated by 61 bit positions to the left. Then the left most four bits are passed through the S-Box. The last step is an XOR between the least significant bits of the round counter and the bits $k_{19}...k_{15}$ of the key.

$$\begin{aligned} 1.[k_{79}...k_0] &= [k_{18}...k_0k_{63}k_{19}] \\ 2.[k_{79}k_{78}k_{77}k_{76}] &= S[k_{79}k_{78}k_{77}k_{76}] \\ 3.[k_{19}k_{18}k_{17}k_{16}k_{15}] &= [k_{19}k_{18}k_{17}k_{16}k_{15}] \oplus round_{counter} \end{aligned} \tag{26}$$

The update algorithm for the 120-bit version of PRESENT works almost the same. It takes the leftmost 64-bit into consideration and it has two active S-Boxes in the update routine. For more details see the appendix of the original paper[43].

8.2 Security Analysis

Although it is possible to implement PRESENT both in software and hardware the latter is advised since the major goal for PRESENT when designing it was hardware performance. This aspect is elaborated later. Applications using PRESENT are unlikely to require the encryption of large amounts of data, since the devices it is designed for have low resources.

Two analysis techniques for cryptographic algorithms are differential and linear cryptanalysis. To prove the resistance of PRESENT to those attacks [43] provided a lower bound to the number of so-called active S-boxes in a differential characteristic. This can be captured by the following theorem, proven in [43].

Theorem 1 Any five-round differential characteristic of PRESENT has a minimum of 10 active S-boxes

[43] made four observations to prove the resistance to differential attacks of PRESENT.

– The input bits to an S-Box come from 4 distinct S-Boxes of the same group.
– The input bits to a group of four S-boxes come from 16 different S-Boxes.
– The four output bits from a particular S-Box enter four distinct S-boxes, each of which belongs to a distinct group of S-boxes in the subsequent round.
– The output bits of S-boxes in distinct groups go to distinct S-boxes.

Taking theorem 1 into account, we note that any 25 rounds must have at least 5 x 10 = 50 active S-Boxes. Advanced cryptanalysis techniques allow to remove the outer rounds from a cipher to exploit the characteristic, but the authors think that this is not enough to break up to 25 rounds.

For a *linear attack* it would need about 2^{84} known plaintext/ciphertext pairs to break 31 rounds of PRESENT. Such an amount of data exceeds the available text and is therefore not sufficient at these days. *Structural attacks* are well suited to analyze AES-like ciphers. Such ciphers have word like structures where one word is typically one byte. The bitwise design of PRESENT shall protect against those attacks because the bitwise operations used in the cipher disrupt the word-wise structure.

8.3 Attacks

The first attack on PRESENT is a statistical saturation attack and can be seen as an example of partitioning cryptanalysis. It extracts information about the key by observing non-uniform distributions in the ciphertext [44] and therefore exploits the diffusion properties in block ciphers. It is possible to break up to 15 rounds of PRESENT using $2^{35.6}$ plaintext-ciphertext pairs. The principal attack uses a weakness in the diffusion layer of PRESENT.

Nakahara et al. [45] present a linear algebraic cryptanalysis of reduced round variants for PRESENT. They introduce a pure algebraic cryptanalysis of 5-rounds within that experiment, they were able to recover half of the bits of the key in less than three minutes using an ordinary desktop PC. The attack complexity with respect to time, data (known plaintext), memory, key size for a linear reduced-round attack of PRESENT can be found in Table 4 of [45].

Hernandez-Castro et al. [46] tested the strength of PRESENT's key schedule algorithm of both variants with 80 and 128 bit keys. They used a probabilistic metaheuristic search for semi-equivalent keys, annihilators and entropy minima. Surprisingly, the results show that the 128-bit key seems to be weaker than the 80-bit key. The entropy per byte was 4.006811 (80-bit) compared to 3.744336 (128-bit). The authors affiliated this effect with the theory that there is a reduced number of global optima for the 80-bit version and multiple ones for the 128-bit version [46].

8.4 Performance

As mentioned before PRESENT requires about 1570 GE when implemented in hardware. In Table 10 a breakdown view of the single components for the

Table 10: List of GE needed for PRESENT implementation (derived from [43])

module	GE	%	module	GE	%
data state	384.39	24.48	KS: key state	480.49	30.61
s-layer	448.45	28.57	KS: S-box	28.03	1.79
p-layer	0	0	KS: Rotation	0	0
counter: state	28.36	1.81	KS: counter-XOR	13.35	0.85
counter: combinatorial	12.35	0.79	key-XOR	170.84	10.88
other	3.67	0.23			
			Overall	**1569.93**	**100**

hardware implementation can be seen.[43] The most GE are needed to implement the flip-flops for storing the key and the data state, followed by the S-layer and the key XOR. The bit permutation can be implemented using simple wiring and therefore needs no GE. There is a more detailed comparison later in the paper.

9 SEA

Most current block ciphers like AES are designed to only find a good tradeoff between cost, security and performance. SEA on the other hand was designed as a low-cost encryption algorithm running on very limited processing resources. [47] defined the following design goals: low memory, small code size, and limited instruction set. Additionally, they proposed the flexibility as an additional design goal because many block ciphers are designed to run on one specific platform, or processor size and perform very badly is run on a different platform or processor size due to the inflexible design. $SEA_{n,b}$ is parametric in text, key and processor size.

9.1 Algorithm Specifications

One of the stated design goals is that $SEA_{n,b}$ should run on many different platforms, but should behave similar. To achieve this goal, $SEA_{n,b}$ is parametric in the following parameters:

- n: plaintext size, key size
- b: processor size
- $n_b = \frac{n}{2b}$: number of words per Feistel round
- n_r: number of block cipher rounds

The only constraint is that: n has to be a multiple of $6b$. For example for an 8-bit processor there can be the following block ciphers: $SEA_{48,8}$, $SEA_{96,8}$, $SEA_{144,8}$... [47] suggested that the number of rounds is

$$n_r = \frac{3n}{4} + 2 \cdot (n_b + \lfloor \frac{b}{2} \rfloor). \tag{27}$$

This is further explained in section 9.2.

Basic operations $SEA_{n,b}$ only uses a limited number of elementary operations: XOR, S-Box, word rotation, bit rotation and addition $mod 2^b$. They are defined as follows

1. **XOR $\oplus$:**

$$\oplus : \mathbb{Z}_2^{\frac{n}{2}} \times \mathbb{Z}_2^{\frac{n}{2}} \to \mathbb{Z}_2^{\frac{n}{2}} : x, y \to z = x \otimes y \Leftrightarrow z(i) = x(i) \oplus y(i) \quad 0 \leq i \leq \frac{n}{2} - 1$$

2. **S-box S:**

$$S : \mathbb{Z}_{2^b}^{n_b} \to \mathbb{Z}_{2^b}^{n_b} : x \to x = S(x) \Leftrightarrow \tag{28}$$

$$\begin{aligned} x_{3i} &= (x_{3i+2} \wedge x_{3i+1}) \otimes x_{3i}, \\ x_{3i+1} &= (x_{3i+2} \wedge x_{3i}) \otimes x_{3i+1}, \\ x_{3i+2} &= (x_{3i} \vee x_{3i+1}) \otimes x_{3i+2}, \quad 0 \leq i \leq \frac{n_b}{3} - 1 \end{aligned} \tag{29}$$

where $\wedge$ denotes bitwise AND and $\vee$ denotes bitwise OR.

3. **Word rotation R:**

$$\begin{aligned} R : \mathbb{Z}_{2^b}^{n_b} \to \mathbb{Z}_{2^b}^{n_b} : x \to y = R(x) \Leftrightarrow & y_{i+1} = x_i, \quad 0 \leq i \leq n_b - 2, \\ & y_0 = x_{n_b - 1} \end{aligned} \tag{30}$$

4. **Bit rotation r:**

$$\begin{aligned} r : \mathbb{Z}_{2^b}^{n_b} \to \mathbb{Z}_{2^b}^{n_b} : x \to y = r(x) \Leftrightarrow & y_{3i} = x_{3i} \ggg 1, \\ & y_{3i+1} = x_{3i+1}, \\ & y_{3i+2} = x_{3i+1} \lll 1, \quad 0 \leq i \leq \frac{n_b}{3} - 1 \end{aligned} \tag{31}$$

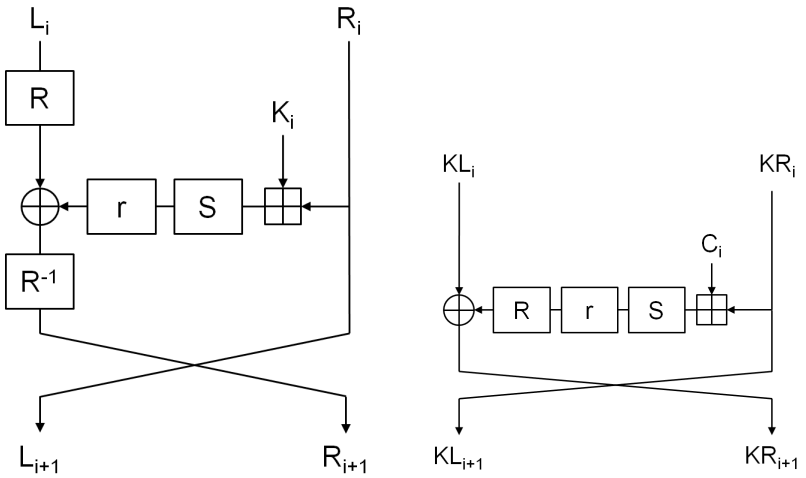
where $\lll$ denotes a left shift and $\ggg$ a right shift.

5. **Addition mod2^b $\boxplus$:**

$$\boxplus : \mathbb{Z}_2^{\frac{n}{2}} \times \mathbb{Z}_2^{\frac{n}{2}} \to \mathbb{Z}_2^{\frac{n}{2}} : x, y \to z = x \boxplus y \Leftrightarrow z_i = x_i \boxplus y_i, \quad 0 \leq i \leq n_b - 1 \tag{32}$$

$SEA_{n,b}$ uses a simple Feistel round for encryption/decryption round as well as for the key round. Figure 11a shows the encryption/decryption round. At the beginning of each round the plaintext is split into two blocks L_i and R_i of n_b words. For encryption the left block L_i is then rotated as describe in equation 30 and xored with the right block R_i as follows using the basic operations introduced earlier:

$$\begin{aligned} R_{i+1} &= R(L_i) \oplus r(S(R_i \boxplus K_i)) \\ L_{i+1} &= R_i \end{aligned}$$



(a) Encryption/Decryption Round (b) Key Round

Fig. 11: SEA Rounds [47]

During decryption the left block L_i is not rotated as during encryption, instead after the xor the block is rotated using the inverse of the word rotation as describe in equation 30 as follows:

$$R_{i+1} = R^{-1}\left(L_i \oplus r(S(R_i \boxplus K_i))\right)$$
$$L_{i+1} = R_i$$

Figure 11b shows the key round. At the beginning the key is split into blocks KL_i and KR_i of n_b words. The left block KL_i is xored with the right block as follows:

$$KR_{i+1} = KL_i \oplus R(r(S(KR_i \boxplus Ci)))$$
$$KL_{i+1} = KR_i$$

The key schedule is designed so that the key round is the same for encryption as well as decryption. To accomplish this and allow different number of rounds, after $\lfloor \frac{n_r}{2} \rfloor$ rounds the blocks KL_i and KR_i are switch, which leads to reverse the earlier key derivation and lead to the following key expansion:

$$K_0, K_1 ..., K_{\lfloor \frac{n_r}{2} \rfloor}, K_{\lfloor \frac{n_r}{2} \rfloor - 1}, ...K_1, K_0 \tag{33}$$

9.2 Security Analysis

To ensure resistance against linear and differential cryptanalysis, [47] propose that the number of rounds should be $n_r \geq \frac{3n}{4}$. They further show that to prevent

both structural attacks and outer rounds improvements of statistical attacks that SEA is secure if the number of rounds is equal or greater than the number of rounds needed for complete diffusion. For SEA, complete diffusion is achieved after $n_b + \lfloor \frac{b}{2} \rfloor$ rounds. To propagate one active bit to all words it takes at most n_b rounds. This part is done by the combination of the word rotation with the S-box. The diffusion inside each block takes at most $\lfloor \frac{b}{2} \rfloor$ rounds. Getting a more conservative approach [47] propose doubling the number of rounds necessary for complete diffusion. The total number of rounds is

$$\frac{3n}{4} + 2 \cdot (n_b + \lfloor \frac{b}{2} \rfloor). \tag{34}$$

We are not aware of any known attack against SEA.

10 Comparison

Table 11 shows a comparison of Simon, Speck, AES, KATAN, LED, TEA, PRESENT and SEA. SEA, SPECK and TEA were designed to be implemented in software while SIMON, KATAN, LED and PRESENT were designed to be implemented in hardware.
The speed of each cipher can depend on the amount of GE used in a hardware implementations. We can see that SIMON is overall the best for hardware implementations. It combines the best results in area, throughput, efficiency(Throughput/GE) and is adjustable to the needed security level. KATAN can be more efficiency and reach a higher throughput but therefore it needs more area. In compare to KATAN, LED is more adjustable to the needed security level with less throughput but in compare to SIMON, LED is worse in every result. PRESENT needs nearly the same area as LED but reaches a higher throughput so that it is more efficiency. AES has a very high throughput and because of that a good efficiency but it needs to much area for the constraints in IoT. Even SPECK which was designed to be used in software implementations has better results than LED and PRESENT. We don't found a representative hardware implementation of TEA.
For software implementations Table 12 shows a comparison of PRESENT, KATAN, SEA, TEA and AES implemented in software. Implementing PRESENT in software is difficult because of the bitwise permutation that PRESENT uses. In assemble, there is no instruction for this operation which leads to a large performance decrease. As we can see AES is clearly the fastest one in software in Table 12, but it also has the highest code size as well as energy consumption. SEA and TEA have nearly the same results but TEA needs just the half of the code-size than SEA. In Table 11 we can see that Speck have the best throughput and has the lowest memory usage of all software implementations. For LED no software implementations are known.

Table 11: Performance comparisons. Hardware refers to an ASIC implementation, and software to an implementation on an 8-bit micro- controller; clock speeds are 100 kHz (hardware) and 16 MHz (software). The best performance for a given size is indicated in red, the second best in blue. Numbers in brackets are estimates. Derived from "Simon and Speck: Block Ciphers for the Internet of Things" [6] table 1.1 and [7]

		hardware			software		
size	name	area (GE)	throughput (kbps)	efficiency (bps/GE)	flash (bytes)	SRAM (bytes)	throughput (kbps)
32/80	KATAN32	802	12.5	16	-	-	-
48/80	KATAN48	927	18.8	20	-	-	-
48/96	SIMON	763	15.0	20	196	0	589
	SPECK	884	12.0	14	134	0	943
64/64	LED	966	5.1	5	-	-	
64/80	LED	1040	3.4	3	-	-	-
	PRESENT	1030	12.4	12	[487]	0	96
	KATAN64	1054	25.1	24	272	18	14
64/96	SIMON	838	17.8	21	274	0	540
	SPECK	984	14.5	15	182	0	888
	LED	1116	3.4	3	-	-	-
64/128	SIMON	1000	16.7	17	282	0	515
	SPECK	1127	13.8	12	186	0	855
	LED	1265	3.4	3	-	-	-
	TEA	-	-	-	-	0	163.2
	PRESENT	1339	12.1	9	[487]	[0]	[96]
96/96	SIMON	984	14.8	15	454	0	454
	SPECK	1134	13.8	12	276	0	866
	SEA	4313	103.2	24	-	0	158,8
128/128	SIMON	1317	22.9	17	732	0	342
	SPECK	1396	12.1	9	396	0	768
	AES	2400	56.6	24	943	33	445

Table 12: Comparison of software implementations of ciphers (at 4MHz).

Cipher	**Block Size** [bits]	**Key Size** [bits]	**Code Size** [bytes]	**RAM** [bytes]	**Cycles** [enc+key]	**Cycles** [dec+key]	**Throughput** [Kbps]	**Energy** [μJ]
PRESENT [2]	64	80	1000	18	11342	13599	-	45.3
PRESENT[3]	64	80	936	0	10723	11239	23.7	-
KATAN[48]	64	80	338	18	72063	88525	-	289.2
SEA [2]	96	96	426	24	41604	40860	-	30.3
SEA [3]	96	96	2132	0	7408	9654	39.7	-
TEA [48]	64	128	648	24	7408	7539	-	30.3
TEA [49]	64	128	1140	0	6271	6299	40.8	-
AES [2]	128	128	1659	33	4557	7015	-	19.2
AES [3]	128	128	2606	0	6637	7429	77.1	-

11 Conclusion

It can be said that the background for what the algorithm is developed is very important. Each of the researched algorithms has specific advantages and disadvantages. AES is the state of the art block cipher and achieved very good results in hardware and software implementations if we are just looking at the performance(efficiency) and security level but because of the constraints in IoT other algorithms fit better.
PRESENT should only be used as hardware implementation, because the performance in software is very bad. The advantages of SEA are its simplicity, its scalability, and the "on-the fly" key derivation. Even though it was originally designed to be implemented in software a number of papers [50] [51] show that it can be implemented in hardware as well with a good performance. The results of TEA are very similar to those of SEA. LED can be adjusted to the needed security level but has a low throughput.
Our conclusion is that SIMON and SPECK are over all our recommended block ciphers. The advantage from Simon and Speck is the simplicity and flexibility. These two properties make it possible to implement the algorithms in different ways. The algorithms can be very small to run on FPGA, microcontroller, and microprocessor implementations, but can also achieve very high throughput on all of these platforms.
If you don't trust SIMON and SPECK because of the NSA background and the aforementioned or other reasons, `KATAN` is a very good alternative for hardware implementations and performs very near to SIMON. `KATAN` can be very efficiency, so if the area space is not very limited and a high throughput is needed `KATAN` is the right alternative choice. Also you can choose between 3 variants of `KATAN` depending on the needed security level. If we are looking for a software implementation alternative for SPECK we recommend TEA because it has little higher throughput and less code size than SEA at a nearly similar energy consumption and security level.

References

1. Gabler Wirtschaftslexikon. Internet der dinge. 2015. visited 28-July-2015.
2. Thomas Eisenbarth, Zheng Gong, Tim Güneysu, Stefan Heyse, Sebastiaan Indesteege, Stéphanie Kerckhof, François Koeune, Tomislav Nad, Thomas Plos, Francesco Regazzoni, et al. Compact implementation and performance evaluation of block ciphers in attiny devices. In *Progress in Cryptology-AFRICACRYPT 2012*, pages 172–187. Springer, 2012.
3. Thomas Eisenbarth, Sandeep Kumar, Christof Paar, Axel Poschmann, and Leif Uhsadel. A survey of lightweight-cryptography implementations. *IEEE Design & Test of Computers*, 24(6):522–533, 2007.
4. T Good and M Benaissa. Hardware results for selected stream cipher candidates. *State of the Art of Stream Ciphers*, pages 191–204, 2007.
5. Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. The simon and speck families of lightweight block ciphers. Cryptology ePrint Archive, Report 2013/404, 2013. http://eprint.iacr.org/.
6. Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. Simon and speck: Block ciphers for the internet of things. Cryptology ePrint Archive, Report 2015/585, 2015. http://eprint.iacr.org/.
7. Christophe Cannière, Orr Dunkelman, and Miroslav Knežević. Katan and ktantan – a family of small and efficient hardware-oriented block ciphers. In *Proceedings of the 11th International Workshop on Cryptographic Hardware and Embedded Systems*, pages 272–288. Springer-Verlag, 2009.
8. Frank-Michael Quedenfeld. Modellbildung in der algebraischen kryptoanalyse. 2015.
9. David J. Wheeler. Tea, atin yencryption algorithm. Cambridge University England.
10. Benjamin Andrews. Tiny encryption algorithm (tea) cryptography 4005.705.01 graduate team acd final report.
11. Derek Williams. The tiny encryption algorithm (tea). Columbus State University.
12. P. Israsena. Design and implementation of low power hardware encryption for low cost secure rfid using tea.
13. P. Israsena and S. Wongnamkum. Hardware implementation of a tea-based lightweight encryption for rfid security.
14. Jiqiang Lu. Related-key rectangle attack on 36 rounds of the xtea block cipher. *Int. J. Inf. Secur.*, 8(1):1–11, January 2009.
15. Gautham Sekar. Meet-in-the-middle attacks on reduced-round xtea. Belgium.
16. Takanori Isobe and Kyoji Shibutani. Security analysis of the lightweight block ciphers xtea, led and piccolo. In *Proceedings of the 17th Australasian Conference on Information Security and Privacy*, ACISP'12, pages 71–86, Berlin, Heidelberg, 2012. Springer-Verlag.
17. Huaifeng Chen and Xiaoyun Wang. Improved linear hull attack on round-reduced simon with dynamic key-guessing techniques. Cryptology ePrint Archive, Report 2015/666, 2015. http://eprint.iacr.org/.
18. Ning Wang, Xiaoyun Wang, Keting Jia, and Jingyuan Zhao. Differential attacks on reduced simon versions with dynamic key-guessing techniques. Cryptology ePrint Archive, Report 2014/448, 2014. http://eprint.iacr.org/.
19. I. Dinu. Improved differential cryptanalysis of round-reduced speck. In *Selected Areas in Cryptography*, pages 147–164. Springer-Verlag, 2014.
20. S.Lucks F.Abed, E.List and J.Wenzel. Differential cryptanalysis of round-reduced simon and speck. Fast Software Encryption, FSE 2014, LNCS. Springer, 2014.

21. Andrey Bogdanov and Christian Rechbergerv. A 3-subset meet-in-themiddle attack: Cryptanalysis of the lightweight block cipher ktantan. In *Selected Areas in Cryptography*, pages 229–240. Springer-Verlag, 2011.
22. Martin R. Albrecht and Gregor Leander. An all-in-one approach to differential cryptanalysis for small block ciphers. In *Selected Areas in Cryptography*, pages 1–15. Springer-Verlag, 2013.
23. Willi Meier Simon Knellwolf and Mar´a Naya-Plasencia. Conditional differential cryptanalysis of trivium and katan. In *Selected Areas in Cryptography*, pages 200–212. Springer-Verlag, 2012.
24. Thomas Fuhr and Brice Minaud. Match box meet in the middle attack against katan. In *Fast Software Encryption*, pages 61–81. Springer-Verlag, 2015.
25. Yu Sasaki Takanori Isobe and Jiageng Chen. Related-key boomerang attacks on katan32/48/64. In *Information Security and Privacy*, pages 268–285. Springer-Verlag, 2013.
26. Gregory V. Bard Nicolas Courtois Jorge Nakahara Jr. Pouyan Sepehrdad and Bingsheng Zhang. Algebraic, aida/cube and side channel analysis of katan family of block ciphers. In *Progress in Cryptology - INDOCRYPT*, pages 176–196. Springer-Verlag, 2010.
27. Jian Guo, Thomas Peyrin, Axel Poschmann, and Matt Robshaw. The led block cipher. In *Proceedings of the 13th International Conference on Cryptographic Hardware and Embedded Systems*, CHES'11, pages 326–341, Berlin, Heidelberg, 2011. Springer-Verlag.
28. A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. Robshaw, Y. Seurin, and C. Vikkelsoe. Present: An ultra-lightweight block cipher. In *Proceedings of the 9th International Workshop on Cryptographic Hardware and Embedded Systems*, CHES '07, pages 450–466, Berlin, Heidelberg, 2007. Springer-Verlag.
29. Ivica Nikolic, Lei Wang, and Shuang Wu. Cryptanalysis of round-reduced LED. *IACR Cryptology ePrint Archive*, 2015:429, 2015.
30. Patrick Derbez, Pierre-Alain Fouque, and Jérémy Jean. Improved key recovery attacks on reduced-round AES in the single-key setting. In *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, pages 371–387, 2013.
31. Vincent Grosso, Christina Boura, Benoît Gérard, and François-Xavier Standaert. A note on the empirical evaluation of security margins against algebraic attacks (with application to low cost-ciphers led and piccolo). In *33rd WIC Symposium on Information Theory in the Benelux*, 2012.
32. Xinjie Zhao, Shize Guo, Fan Zhang, Tao Wang, Zhijie Shi, and Keke Ji. Algebraic differential fault attacks on led using a single fault injection, 2012.
33. P. Jovanovic, M. Kreuzer, I. Polian, and Universitt Passau. An algebraic fault attack on the led block cipher.
34. Florian Mendel, Vincent Rijmen, Deniz Toz, and Kerem Varici. Differential analysis of the led block cipher. *IACR Cryptology ePrint Archive*, 2012:544, 2012. informal publication.
35. Wikipedia. Nothing up my sleeve number. visited 28-July-2015.
36. Wikipedia. Tiny encryption algorithm. 2015. visited 28-Aug-2015.
37. Wikipedia. Kryptologische hashfunktion. 2015. visited 21-November-2015.
38. VIKRAM REDDY ANDEM. A cryptanalysis of the tiny encryption algorithm. University of Alabama.
39. Wikipedia. Xtea. 2015. visited 15-July-2015.

40. Youngdai Ko, Seokhie Hong, Wonil Lee, Sangjin Lee, and Ju-Sung Kang. Related key differential attacks on 27 rounds of xtea and full-round gost. In Bimal Roy and Willi Meier, editors, *Fast Software Encryption*, volume 3017 of *Lecture Notes in Computer Science*, pages 299–316. Springer Berlin Heidelberg, 2004.
41. David J. Wheeler. Correction to xtea. Cambridge University England.
42. Elias Yarrkov. Cryptanalysis of xxtea.
43. Andrey Bogdanov, Lars R Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew JB Robshaw, Yannick Seurin, and Charlotte Vikkelsoe. *PRESENT: An ultra-lightweight block cipher*. Springer, 2007.
44. Baudoin Collard and F-X Standaert. A statistical saturation attack against the block cipher present. In *Topics in Cryptology–CT-RSA 2009*, pages 195–210. Springer, 2009.
45. Jorge Nakahara Jr, Pouyan Sepehrdad, Bingsheng Zhang, and Meiqin Wang. Linear (hull) and algebraic cryptanalysis of the block cipher present. In *Cryptology and Network Security*, pages 58–75. Springer, 2009.
46. Julio Cesar Hernandez-Castro, Pedro Peris-Lopez, and Jean-Philippe Aumasson. On the key schedule strength of present. In *Data Privacy Management and Autonomous Spontaneus Security*, pages 253–263. Springer, 2012.
47. François-Xavier Standaert, Gilles Piret, Neil Gershenfeld, and Jean-Jacques Quisquater. Sea: A scalable encryption algorithm for small embedded applications. In *Smart Card Research and Advanced Applications*, pages 222–236. Springer, 2006.
48. Thomas Eisenbarth, Zheng Gong, Tim Güneysu, Stefan Heyse, Sebastiaan Indesteege, Stéphanie Kerckhof, François Koeune, Tomislav Nad, Thomas Plos, Francesco Regazzoni, François-Xavier Standaert, and Loic van Oldeneel tot Oldenzeel. Compact implementation and performance evaluation of block ciphers in attiny devices. In *Proceedings of the 5th International Conference on Cryptology in Africa*, AFRICACRYPT'12, pages 172–187, Berlin, Heidelberg, 2012. Springer-Verlag.
49. Thomas Eisenbarth, Sandeep Kumar, Christof Paar, Axel Poschmann, and Leif Uhsadel. A survey of lightweight-cryptography implementations. *IEEE Des. Test*, 24(6):522–533, November 2007.
50. Franois Mace, Franois-Xavier Standaert, Jean-Jacques Quisquater, et al. Asic implementations of the block cipher sea for constrained applications. In *Proceedings of the Third International Conference on RFID Security-RFIDSec*, pages 103–114, 2007.
51. François Macé, F-X Standaert, and J-J Quisquater. Fpga implementation (s) of a scalable encryption algorithm. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 16(2):212–216, 2008.
52. Jiazhe Chen, Meiqin Wang, and Bart Preneel. Impossible differential cryptanalysis of the lightweight block ciphers tea, xtea and hight. In *AFRICACRYPT'12*, pages 117–137, 2012.

12 Appendix

Table 13 shows summarized different attacks on TEA and LED. There are on the one hand attacks listed which attack so far the most rounds (apart of side channel attacks) and on the other hand there is a MITM attack which can be applied on LED and XTEA[16]. At XTEA the MITM attacks the so far most rounds.

Table 13

Cipher	Attack	Type	Rounds	Time	Data	Memory	Ref
`LED-64`	MITM (SK)	KR	8	2^{56}	2^{8}	2^{11}	[16]
(32 rounds)	MITM/Differential (CK)	D	20	$2^{60.2}$	–	$2^{61.5}$	[29]
`LED-128`	MITM (SK)	KR	16	2^{112}	2^{16}	2^{19}	[16]
(48 rounds)	MITM/Differential (CK)	D	20	$2^{60.2}$	–	$2^{61.5}$	[29]
`XTEA` (64 rounds)	MITM (SK)	KR	29	2^{124}	2^{45}	2^{4}	[16]
`TEA` (64 rounds)	Linear	KR	23	$2^{119.64}$	2^{64}	–	[52]