The LED Block Cipher

Jian Guo[1], Thomas Peyrin[2,⋆], Axel Poschmann[2,⋆], and Matt Robshaw[3,⋆⋆]

[1] Institute for Infocomm Research, Singapore
[2] Nanyang Technological University, Singapore
[3] Applied Cryptography Group, Orange Labs, France
{ntu.guo,thomas.peyrin}@gmail.com,
aposchmann@ntu.edu.sg,
matt.robshaw@orange-ftgroup.com

Abstract. We present a new block cipher LED. While dedicated to compact hardware implementation, and offering the smallest silicon footprint among comparable block ciphers, the cipher has been designed to simultaneously tackle three additional goals. First, we explore the role of an ultra-light (in fact non-existent) key schedule. Second, we consider the resistance of ciphers, and LED in particular, to related-key attacks: we are able to derive simple yet interesting AES-like security proofs for LED regarding related- or single-key attacks. And third, while we provide a block cipher that is very compact in hardware, we aim to maintain a reasonable performance profile for software implementation.

Keywords: Lightweight, block cipher, RFID tag, AES.

1 Introduction

Over past years many new cryptographic primitives have been proposed for use in RFID tag deployments, sensor networks, and other applications characterised by highly-constrained devices. The pervasive deployment of tiny computational devices brings with it many interesting, and potentially difficult, security issues.

Chief among recent developments has been the evolution of lightweight block ciphers where an accumulation of advances in algorithm design, together with an increased awareness of the likely application, has helped provide important developments. To some commentators the need for yet another lightweight block cipher proposal will be open to question. However, in addition to the fact that many proposals present some weaknesses [2,10,45], we feel there is still more to be said on the subject and we observe that it is in the "second generation" of work that designers might learn from the progress, and omissions, of "first generation" proposals. And while new proposals might only slightly improve on

⋆ The authors were supported in part by the Singapore National Research Foundation under Research Grant NRF-CRP2-2007-03.

⋆⋆ The author gratefully acknowledges the support of NTU during his visit to Singapore. This work is also supported in part by the European Commission through the ICT program under contract ICT-2007-216676 ECRYPT II.

B. Preneel and T. Takagi (Eds.): CHES 2011, LNCS 6917, pp. 326–341, 2011.

successful initial proposals in terms of a single metric, *e.g.* area, they might, at the same time, overcome other important security and performance limitations. In this paper, therefore, we return to the design of lightweight block ciphers and we describe L*ight* E*ncryption* D*evice*, LED.

During our design, several key observations were uppermost in our mind. Practically all modern block cipher proposals have reasonable security arguments; but few offer much beyond (potentially thorough) *ad hoc* analysis. Here we hope to provide a more complete security treatment than is usual. In particular, related-key attacks are often dismissed from consideration for the application areas that typically use such constrained devices, *e.g.* RFID tags. In practice this is often perfectly reasonable. However, researchers will continue to derive cryptanalytic results in the related-key model [18,2] and there has been some research on how to modify or strengthen key schedules [35,15,39]. So having provable levels of resistance to such attacks would be a bonus and might help confusion developing in the cryptographic literature.

In addition, our attention is naturally focused on the performance of the algorithm on the tag. However, there can be constraints when an algorithm is also going to be implemented in software. This is something that has already been discussed with the design of KLEIN [22] and in the design of LED we have aimed at very compact hardware implementation while maintaining some software-friendly features.

Our new block cipher is based on AES-like design principles and this allows us to derive very simple bounds on the number of active Sboxes during a block cipher encryption. Since the key schedule is very simple, this analysis can be done in a related-key model as well; *i.e.* our bounds apply even when an attacker tries to mount a related-key attack. And while AES-based approaches are well-suited to software, they don't always provide the lightest implementation in hardware. But using techniques presented in [23] we aim to resolve this conflict.

While block ciphers are an important primitive, and arguably the most useful in a constrained environment, there has also been much progress in the design of stream ciphers [14,25] and even, very recently, in lightweight hash functions [23,4]. In fact it is this latter area of work that has provided inspiration for the block cipher we will present here.

2 Design Approach and Specifications

Like so much in today's symmetric cryptography, an AES-like design appears to be the ideal starting point for a clean and secure design. The design of LED will inevitably have many parallels with this established approach, and features such as Sboxes, ShiftRows, and (a variant of) MixColumns will all feature and take their familiar roles.

For the key schedule we chose to do-away with the "schedule", *i.e.* the user-provided key is used repeatedly *as is*. As well as giving obvious advantages in hardware implementation, it allows for simple proofs to be made for the security of the scheme even in the most challenging attack model of related keys. At

first sight the re-use of the encryption key without variation appears dangerous, certainly to those familiar with slide attacks and some of their advanced variants [7,8]. But we note that such a simple key schedule is not without precedent [42] though the treatment here is more complete than previously.

The LED cipher is described in Section 2.1. It is a 64-bit block cipher with two primary instances taking 64- and 128-bit keys. The cipher state is conceptually arranged in a (4×4) grid where each nibble represents an element from $\mathrm{GF}(2^4)$ with the underlying polynomial for field multiplication given by $X^4 + X + 1$.

Sboxes. LED cipher re-uses the PRESENT Sbox which has been adopted in many lightweight cryptographic algorithms. The action of this box in hexadecimal notation is given by the following table.

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

MixColumnsSerial. We re-use the tactic adopted in [23] to define an MDS matrix for linear diffusion that is suitable for compact serial implementation. The MixColumnsSerial layer can be viewed as four applications of a hardware-friendly matrix A with the net result being equivalent to using the MDS matrix M where

$$(A)^4 = \begin{pmatrix} 0\,1\,0\,0 \\ 0\,0\,1\,0 \\ 0\,0\,0\,1 \\ 4\,1\,2\,2 \end{pmatrix}^4 = \begin{pmatrix} 4\,2\,1\,1 \\ 8\,6\,5\,6 \\ \mathrm{B}\,\mathrm{E}\,\mathrm{A}\,9 \\ 2\,2\,\mathrm{F}\,\mathrm{B} \end{pmatrix} = M.$$

The basic component of LED will be a sequence of four identical rounds used without the addition of any key material. This basic unit, that we later call "step", makes it easy to establish security bounds for the construction.

2.1 Specification of LED

For a 64-bit plaintext m the 16 four-bit nibbles $m_0 \| m_1 \| \cdots \| m_{14} \| m_{15}$ are arranged (conceptually) in a square array:

$$\begin{bmatrix} m_0 & m_1 & m_2 & m_3 \\ m_4 & m_5 & m_6 & m_7 \\ m_8 & m_9 & m_{10} & m_{11} \\ m_{12} & m_{13} & m_{14} & m_{15} \end{bmatrix}$$

This is the initial value of the cipher STATE and note that the state (and the key) are loaded row-wise rather than in the column-wise fashion we have come to expect from the AES; this is a more hardware-friendly choice, as pointed out in [38].

The key is viewed nibble-wise and loaded nibble-by-nibble into one or two arrays, K_1 and K_2, depending on the key length. Our primary definition is for

64- or 128-bit keys, but other key lengths, *e.g.* the popular choice of 80 bits, can be padded to give a 128-bit key thereby giving a 128-bit key array. By virtue of the order of loading the tables, any key that is padded (with zeros) to give a 64- or 128-bit key array will effectively set unused nibbles of the key array to 0.

$$\begin{bmatrix} k_0 & k_1 & k_2 & k_3 \\ k_4 & k_5 & k_6 & k_7 \\ k_8 & k_9 & k_{10} & k_{11} \\ k_{12} & k_{13} & k_{14} & k_{15} \end{bmatrix} \quad \text{for 64-bit keys giving } K_1$$

$$\begin{bmatrix} k_0 & k_1 & k_2 & k_3 \\ k_4 & k_5 & k_6 & k_7 \\ k_8 & k_9 & k_{10} & k_{11} \\ k_{12} & k_{13} & k_{14} & k_{15} \end{bmatrix} \begin{bmatrix} k_{16} & k_{17} & k_{18} & k_{19} \\ k_{20} & k_{21} & k_{22} & k_{23} \\ k_{24} & k_{25} & k_{26} & k_{27} \\ k_{28} & k_{29} & k_{30} & k_{31} \end{bmatrix} \quad \text{for 128-bit keys giving } K_1 \| K_2$$

The operation `addRoundKey(`STATE`,`K_i`)` combines nibbles of subkey K_i with the state, respecting array positioning, using bitwise exclusive-or. There is no key schedule, or rather this is the sum total of the key schedule, and the arrays K_1 and, where appropriate, K_2 are repeatedly used without modification. Encryption is described using the previously mentioned `addRoundKey(`STATE`,`K_i`)` and a second operation, `step(`STATE`)`. This is illustrated in Figure 1.

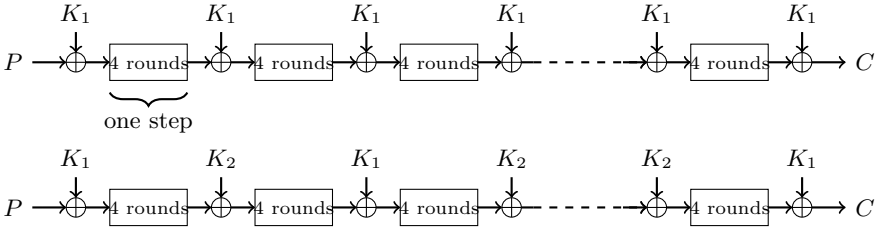

Fig. 1. The use of key arrays K_1 and K_2 in `LED` showing both a 64-bit key array (top) and a 128-bit key array (bottom)

The number of steps during encryption depends on whether there are one or two key arrays.

```
for i = 1 to 8 do {
    addRoundKey(STATE,K1)
    step(STATE)
}
addRoundKey(STATE,K1)
```

for 64-bit key arrays

```
for i = 1 to 6 do {
    addRoundKey(STATE,K1)
    step(STATE)
    addRoundKey(STATE,K2)
    step(STATE)
}
addRoundKey(STATE,K1)
```

for 128-bit key arrays

The operation `step(`STATE`)` consists of four rounds of encryption of the cipher state. Each of these four rounds uses, in sequence, the operations `AddConstants`, `SubCells`, `ShiftRows`, and `MixColumnsSerial` as illustrated in Figure 2.

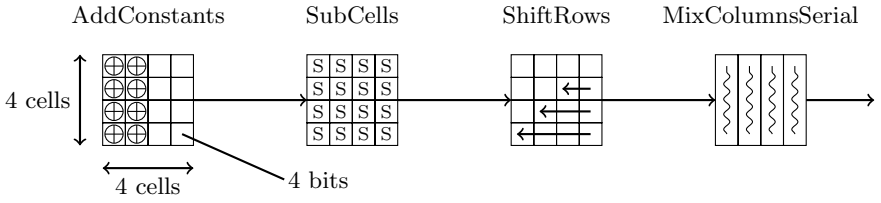

Fig. 2. An overview of a single round of `LED`

`AddConstants`. A round constant is defined as follows. At each round, the six bits (rc_5, rc_4, rc_3, rc_2, rc_1, rc_0) are shifted one position to the left with the new value to rc_0 being computed as $\mathrm{rc}_5 \oplus \mathrm{rc}_4 \oplus 1$. The six bits are initialised to zero, and updated *before* use in a given round. The constant, when used in a given round, is arranged into an array as follows:

$$\begin{bmatrix} 0 & (\mathrm{rc}_5 \| \mathrm{rc}_4 \| \mathrm{rc}_3) & 0 & 0 \\ 1 & (\mathrm{rc}_2 \| \mathrm{rc}_1 \| \mathrm{rc}_0) & 0 & 0 \\ 2 & (\mathrm{rc}_5 \| \mathrm{rc}_4 \| \mathrm{rc}_3) & 0 & 0 \\ 3 & (\mathrm{rc}_2 \| \mathrm{rc}_1 \| \mathrm{rc}_0) & 0 & 0 \end{bmatrix}$$

The round constants are combined with the state, respecting array positioning, using bitwise exclusive-or.

`SubCells`. Each nibble in the array STATE is replaced by the nibble generated after using the PRESENT Sbox.

`ShiftRow`. Row i of the array STATE is rotated i cell positions to the left, for $i = 0, 1, 2, 3$.

`MixColumnsSerial`. Each column of the array STATE is viewed as a column vector and replaced by the column vector that results after post-multiplying the vector by the matrix M (see earlier description in this section).

The final value of the STATE provides the ciphertext with nibbles of the "array" being unpacked in the obvious way. Test vectors for `LED` are provided at `https://sites.google.com/site/ledblockcipher/`.

3 Security Analysis

The `LED` block cipher is simple to analyze and this allows us to precisely evaluate the necessary number of rounds to ensure proper security.

Our scheme is meant to be resistant to classical attacks, but also to the type of related-key attacks that have been effective against AES-256 [9] and other ciphers [2]. We will even study the security of LED in a hash function setting, *i.e.* when it is used in a Davies-Meyer or similar construction with a compression function based on a block cipher. In other words, we will consider attackers that have full access to the key(s) and try to distinguish the fixed permutations from randomly chosen ones. While this analysis provides additional confidence in the security of LED, it is not our intent to propose a hash function construction.

We chose a conservative number of rounds for LED. For example, when using a 64-bit key array we use 32 AES-like rounds that are grouped as eight "big" add-key/apply-permutation steps that are each composed of four AES-like rounds. Further, our security margins are even more conservative if one definitively disregards related-key attacks; as will be seen with the following proofs.

3.1 The Key Schedule

The LED key schedule has been chosen for its simplicity and security. Because it is very simple to analyze, it allows us to directly derive a bound on the minimal number of active Sboxes, even in the scenario of related-key attacks. The idea is to first compute a bound on the number of active big steps (each composed of 4 AES-like rounds). Then, using the well known 4-round proofs for the AES, one can show that one active big step will contain at least 25 active Sboxes. Note that this bound is tight as we know 4-round differential paths containing exactly this number of active Sboxes.

When not considering related-key attacks, we directly obtain that any differential path for LED will contain at least $\lfloor r/4 \rfloor \cdot 25$ active Sboxes. For related-key attacks, we have to distinguish between the different key-size versions.

64-Bit Key Version. If we assume that differences are inserted in the key input, then every subkey K_1 in the 64-bit key variant of LED will be active. Therefore, one can easily see that it is impossible to force two consecutive non-active big steps and we are ensured that for every two big steps at least one is active. Overall, this shows that any related-key differential path contains at least $\lfloor r/8 \rfloor \cdot 25$ active Sboxes.

128-Bit Key Version. If we assume that differences are inserted in the key input, then we have to separate two cases. If the two independent parts K_1 and K_2 composing the key both contain a difference, then we end up with exactly the same reasoning as for the 64-bit key variant: at least $\lfloor r/8 \rfloor \cdot 25$ active Sboxes will be active. If only one of the two independent parts composing the key contains a difference, then subkeys with and without differences are alternatively incorporated after each big step. The non-active subkeys impact on the differential paths is completely void and thus in this case one can view LED as being composed of even bigger steps of 8 AES-like rounds instead. The very same reasoning then applies again: it is impossible to force two consecutive of these new bigger steps to be inactive and therefore we have at least $\lfloor r/16 \rfloor \cdot 50$ active Sboxes ensured

Table 1. Minimal number of active Sboxes and upper bounds on the best differential path and linear approximation probability for the 64-bit key array and 128-bit key array versions of LED (in both the single-key (SK) and related-key (RK) settings)

	LED-64 SK	LED-64 RK	LED-128 SK	LED-128 RK
minimal no. of active Sboxes	200	100	300	150
differential path probability	2^{-400}	2^{-200}	2^{-600}	2^{-300}
linear approx. probability	2^{-400}	2^{-200}	2^{-600}	2^{-300}

for any differential path (since the best differential path for 8 rounds trivially contains 50 active Sboxes).

We summarize in Table 1 the results obtained for the two main versions of LED, both for single-key attacks and related-key attacks. Note that the bounds on the number of active Sboxes are tight as we know differential paths meeting them (for example the truncated differential path for each active big step can simply be any of the 4-round path for AES-128 with 25 active Sboxes).

For LED-128, since we are using two independent key parts one can peel off the first and last key addition (which is always the first key part K_1). Thus, an attacker can remove one big step on each side of the cipher, for a total of 8 rounds, with a complexity of 2^{64} tries on K_1. This partially explains why the versions of LED using two independent key parts have 16 more rounds than for LED-64.

3.2 Differential/Linear Cryptanalysis

Since LED is an AES-like cipher, one can directly reuse extensive work that has been done on the AES. We will compute a bound on the best differential path probability (where all differences on the input and output of all rounds are specified) or even the best differential probability (where only the input and output differences are specified), in both single- and related-key settings.

As the best differential transition probability of the PRESENT Sbox is 2^{-2}, using the previously proven minimal number of active Sboxes we deduce that the best differential path probability on 4 active rounds of LED is upper bounded by $2^{-2 \cdot 25} = 2^{-50}$. By adapting the work from [40], the maximum differential probability for 4 active rounds of LED is upper bounded by

$$\max \left\{ \max_{1 \le u \le 15} \sum_{j=1}^{15} \{DP^S(u,j)\}^5, \max_{1 \le u \le 15} \sum_{j=1}^{15} \{DP^S(j,u)\}^5 \right\}^4 = 2^{-32}$$

where $DP^S(i,j)$ stands for the differential probability of the Sbox to map the difference i to j. The duality between linear and differential attacks allows us to similarly apply the same approaches to compute a bound on the best linear approximation. Over four rounds the best linear approximation probability is upper bounded by 2^{-50} and the best linear hull probability is upper bounded by 2^{-32}.

Since we previously proved that all rounds will be active in the single-key scenario and half of them will be active in the related-key scenario, we can easily compute the upper bounds on the best differential path probability and the best linear approximation probability for each version of LED (see Table 1). Note that this requires that random subkeys be used at each round to make the Sbox inputs independant. In the case of LED the subkeys are simulated by the addition of round constants and the derived bounds give a very good indication of the quality of the LED internal permutation with regards to linear and differential cryptanalysis.

3.3 Cube Testers and Algebraic Attacks

We applied the most recent developed cube testers [3] and its zero-sum distinguishers to the LED fixed-key permutation, the best we could find within practical time complexity is at most three rounds (with the potential to be doubled under a meet-in-the-middle scenario). Note, in case of AES, "zero-sum" property is also referred as "balanced", found by the AES designers [16], in which 3-round balanced property is shown. To the best of our knowledge, there is no balanced property found for more than 3 AES rounds.

The PRESENT Sbox used in LED has algebraic degree 3 and one can check that $3 \cdot \lfloor r/4 \rfloor \cdot 25 \ggg 64$ for all LED variants. Moreover, the PRESENT Sbox is described by $e = 21$ quadratic equations in the $v = 8$ input/output-bit variables over $GF(2)$. The entire system for a fixed-key LED permutation therefore consists of $(16 \cdot r \cdot e)$ quadratic equations in $(16 \cdot r \cdot v)$ variables. For example, in the case of the 64-bit key version, we end up with 10752 equations in 4096 variables. In comparison, the entire system for a fixed-key AES permutation consists of 6400 equations in 2560 variables. While the applicability of algebraic attacks on AES remains unclear, those numbers tends to indicate that LED offers a higher level of protection.

3.4 Other Cryptanalysis

The slide attack is a block cipher cryptanalysis technique [7] that exploits the degree of self-similarity of a permutation. In the case of LED, all rounds are made different thanks to the round-dependent constants addition, which makes the slide attack impossible to perform.

Integral cryptanalysis is a technique first applied on SQUARE [17] that is particularly efficient against block ciphers based on substitution-permutation networks, like AES or LED. The idea is to study the propagation of sums of values; something which is quite powerful on ciphers that only use bijective components. As for AES, the best integral property can be found on three rounds, or four rounds with the last mixing layer removed. Thus, two big LED steps avoid any such observation. Considering the large number of rounds of LED, we believe integrals attacks are very unlikely to be a threat.

Rotational cryptanalysis [28] studies the evolution of a rotated variant of some input words through the round process. It was proven to be quite successful against some Addition-Rotation-XOR (ARX) block ciphers and hash functions.

LED is an Sbox-oriented block cipher and any rotation property in a cell will be directly removed by the application of the Sbox layer. Even if one looks for a rotation property of cell positions, this is unlikely to lead to an attack since the constants used in a LED round are all distinct and any position rotation property between columns or lines is removed after the application of two rounds.

Methods to find better bounds on the algebraic degree were recently published in [12]. With the first two rounds combined as Super-Sboxes, the best algebraic degree we can find for fixed-key LED permutation and its inverse are $3, 11, 33, 53, 60, 62$, for r rounds with $r = 1, \ldots, 6$. Using this technique, one can distinguish up to 12 rounds with complexity bounded by 2^{63}, in the known key model.

3.5 LED in a Hash Function Setting

Studying a block cipher in a hash function setting is a good security test since it is very advantageous for the attacker. In this scenario he will have full control on all inputs. In the so-called known-key [29] or chosen-key models, the attacker can have access or even choose the key(s) used, and its goal is then to find some input/output pairs having a certain property with a complexity lower than what is expected for randomly chosen permutation(s). Typically, the property is that the input and output differences or values are fixed to a certain subset of the whole domain.

While we conduct an analysis of the security of LED in a hash function setting, we would like to emphasize that our goal is not to build a secure hash function. However, we believe that this section adds further confidence in the quality of our block cipher proposal.

Rebound and Super-Sbox Attacks. The recent rebound attack [37] and its improved variants (start-from-the-middle attack [36] and Super-Sbox cryptanalysis [21,31]) have much improved the best known attacks on many hash functions, especially for AES-based schemes. The attacker will first prepare a differential path and then use the available freedom degrees to the most costly part of the trail (often in the middle) so as to reduce the overall complexity. The costly part is called the controlled rounds, while the rest of the trail are the uncontrolled rounds and they are verified probabilistically. The rebound attack and its variants allows the attacker to nicely use the freedom degrees so that the controlled part is as big as possible. At the present time, the most powerful technique in the known-key setting allows the attacker to control three rounds and no method is known to control more rounds, even if the key is chosen by the attacker.

In order to ease the analysis, we assume pessimistically that the attacker can control four rounds, that is one full active big step, with a negligible computation/memory cost (even if one finds a method to control four AES-like rounds in the chosen-key model, it will not apply here since no key is inserted during four consecutive rounds). In the case of 64-bit key LED, the attacker can control two independent active big steps and later merge them by freely fixing the key

value. However, even in this advantageous scenario for the attacker we are ensured that at least two big steps will be active and uncontrolled, and this seems sufficient to resist distinguishing attacks. Indeed, for two active big steps of LED, the upper bound for the best differential path probability and the best linear approximation probability (respectively the best differential probability and the best linear hull probability) is 2^{-100} (respectively 2^{-64}).

For the 128-bit key version, we can again imagine that the attacker to control and merge two active big steps with a negligible computation/memory cost. Even if so, with the same reasoning we are ensured that at least four big steps will be active and uncontrolled, and again this seems sufficient since for four active big steps of LED, the upper bound for the best differential path probability and the best linear approximation probability (respectively the best differential probability and the best linear hull probability) is 2^{-200} (respectively 2^{-128}).

Integral Attacks. One can directly adapt the known-key variant of integral attacks from [29] to the LED internal permutation. However, this attack can only reach seven rounds with complexity 2^{28}, which is worse than what can be obtained with previous rebound-style attacks.

4 Performance and Comparison

4.1 Hardware Implementation

We used *Mentor Graphics ModelSimXE 6.4b* and *Synopsys DesignCompiler A-2007.12-SP1* for functional simulation and synthesis of the designs to the *Virtual Silicon* (VST) standard cell library *UMCL18G212T3*, which is based on the *UMC L180 0.18µm 1P6M* logic process with a typical voltage of 1.8 V. For synthesis and for power estimation (using *Synopsys Power Compiler* version *A-2007.12-SP1*) we advised the compiler to keep the hierarchy and use a clock frequency of 100 KHz, which is a widely cited operating frequency for RFID applications. Note that the wire-load model used, though it is the smallest available for this library, still simulates the typical wire-load of a circuit with a size of around 10,000 GE.

To substantiate our claims on the hardware efficiency of our LED family, we have implemented LED-64 and LED-128 in VHDL and simulated their post-synthesis performance. As can be seen in Figure 3, our serialized design consists of seven modules: MCS, State, AK, AC, SC, Controller, and Key State.

State comprises a $4 \cdot 4$ array of flip-flop cells storing 4 bits each. Every row constitutes a shift-register using the output of the last stage, *i.e.* column 0, as the input to the first stage (column 3) of the same row and the next row. Using this feedback functionality *ShiftRows* can be performed in 3 clock cycles with no additional hardware costs. Further, since *MixColumnsSerial* is performed on column 0, also a vertical shifting direction is required for this column. Consequently, columns 0 and 3 consist of flip-flop cells with two inputs (6 GE), while columns 1 and 2 can be realized with flip-flop cells with only one input (4.67 GE).

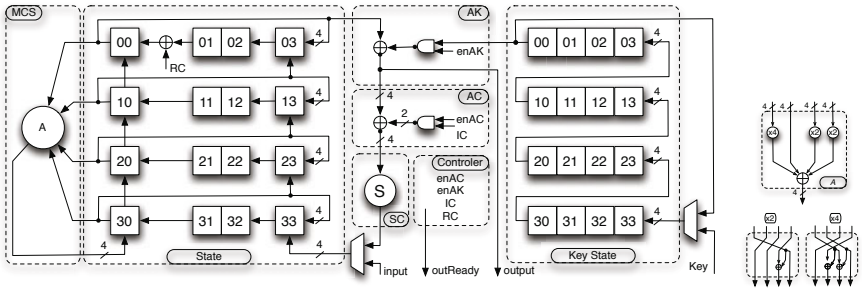

Fig. 3. Serial hardware architecture of `LED` (left) and A with its sub-components (right)

The key is stored in `Key State`, which comprises of a 4-bit wide simple shift register of the appropriate length, i.e. 64 or 128. Please note that the absence of a key-schedule of `LED` has two advantages: it allows 1) to use the most basic, and thus cheapest, flip-flops (4.67 GE per bit); and 2) to hardwire the key in case no key update is required. In the latter case additional combinational logic is required to select the appropriate key chunk, which reduces the savings to 278 GE and 577 GE for `LED`-64 and `LED`-128, respectively. For arbitrary key lengths the area requirements grow by 4.67 GE per bit. An `LED`-80 with the same parameters as `PRESENT`-80 would thus require approximately 1,040 GE with a flexible key and around 690 GE with fixed key.

`MCS` calculates the last row of A in one clock cycle. The result is stored in the `State` module, that is in the last row of column 0, which has been shifted upwards at the same time. Consequently, after 4 clock cycles the *MixColumnsSerial* operation is applied to an entire column. Then the whole state array is rotated by one position to the left and the next column is processed. As an example of the hardware efficiency of *MCS* we depict A in the upper and its sub-components in the lower right part of Figure 3. In total only 40 GE and 20 clock cycles are required to perform *MCS*, which is 4 clock cycles slower but 85% smaller than a serialized implementation of the AES *MixColumns* [24]. If we take into account that AES operates on 8 bits and not like `LED` on 4 bits, the area savings are still more than 40%.

`AK` performs the *AddRoundKey* operation by XORing the roundkey every fourth round. For this reason the input to the XNOR gate is gated with a NAND gate.

`AC` performs one part of the *AddConstant* operation by XORing the first column of the round constant matrix (a simple arithmetic 2-bit counter) to the first column of the state matrix. For this reason, the input to the XNOR gate is gated with a NAND gate. In order to use a single control signal for the addition of the round constants, which span over the first two columns, the addition of the second column of the round constant matrix to the second column of the state array is performed in the `State` module.

`SC` performs the *SubCells* operation and consists of a single instantiation of the corresponding Sbox. We used an optimized Boolean representation of the

PRESENT Sbox,[1] which only requires 22.33 GE. It takes 16 clock cycles to perform *AddConstant* and *SubCells* on the whole state.

Controller uses a Finite State Machine (FSM) to generate all control signals required. The FSM consists of one idle state, one init state to load the initial values, one state for the combined execution of AC and SC, 3 states for ShR and two states for MCS (one for processing one column and another one to rotate the whole state to the left). Several LFSR-based counters are required: 6-bit for the generation of the second column of the round constants matrix, 4-bit for the key addition scheduling and 2-bit for the transition conditions of the FSM. Besides, a 2-bit arithmetic counter is required for the generation of the first column of the round constants matrix. Its LSB is also used to select either the 3 MSB $rc_5||rc_4||rc_3$ or the 3 LSB $rc_2||rc_1||rc_0$ of the 6-bit LFSR-based counter. In total the control logic sums up to 199 GE.

It requires 39 clock cycles to perform one round of LED, resulting in a total latency of 1248 clock cycles for LED-64 and 1872 clock cycles for LED-128. The estimated power consumption at a frequency of 100 KHz and a supply voltage ov 1.8V is 1.67μW for LED-64 (1.11μW with a hard-wired key) and 2.2μW for LED-128 (1.11μW). It is a well-known fact that at low frequencies, as typical for low-cost applications, the power consumption is dominated by its static part, which is proportional to the amount of transistors involved. Furthermore, the power consumption strongly depends on the used technology and greatly varies with the simulation method. To address these issues and to reflect the time-area-power trade-off inherent in any hardware implementation a new figure of merit (FOM) was proposed by [5]. In order to have a fair comparison, we omit the power values in Table 2 and only compare cycles per block, throughput at 100 KHz (in kilo bits per second), the area requirements (in GE), and FOM (in nano bits per clock cycle per GE squared).

Table 2 compares our results to previous work, sorted according to key flexibility and increasing security levels. Note that we have not been able to include all recent proposals and we have restricted ourselves to block ciphers for our comparison. Other techniques such as HUMMINGBIRD [19] and ARMADILLO [5] are of some interest in the literature, though attacks on early versions have lead to some redesign [45,1,20]. As can be seen from Table 2, the block cipher LED is the smallest when compared to other block ciphers with similar key and block size.

4.2 Software Implementation

We have made two implementations of LED; one for reference and clarity with the second being optimized for performance (by using table lookups). The measurements were taken on an Intel(R) Core(TM) i7 CPU Q 720 clocked at 1.60GHz.

In the optimised implementation, we represent the LED state as a single 64-bit word and we build eight lookup tables each with 256 64-bit entries. This is similar to many AES implementations, except we treat two consecutive nibbles

[1] Due to Dag Arne Osvik.

Table 2. Hardware implementation results of some block ciphers. [44] also synthesized the same architecture of PRESENT and yielded a lower gate count of 1,000 GE. However, the number quoted below is from the same library used here and hence is a fairer choice for comparison. * denotes estimated values.

	Algorithm	Ref.	key size	block size	cycles/ block	T'put (@100 KHz)	Tech. [μm]	**Area** [GE]	FOM [$\frac{bits \times 10^9}{clk \cdot GE^2}$]
Flexible Keys	DESL	[32]	56	64	144	44.4	0.18	**1,848**	130
	LED-64		64	64	1,248	5.1	0.18	**966**	55
	KLEIN-64	[22]	64	64	207	N/A	0.18	**1,220**	N/A
	LED-80*		80	64	1,872	3.4	0.18	**1,040**	32
	PRESENT-80	[44]	80	64	547	11.7	0.18	**1,075**	101
	PRESENT-80	[11]	80	64	32	200.0	0.18	**1,570**	811
	KATAN64	[13]	80	64	255	25.1	0.13	**1,054**	226
	KLEIN-80	[22]	80	64	271	N/A	0.18	**1,478**	N/A
	LED-96*		96	64	1,872	3.4	0.18	**1,116**	27
	KLEIN-96	[22]	96	64	335	N/A	0.18	**1,528**	N/A
	mCrypton	[33]	96	64	13	492.3	0.13	**2,681**	685
	SEA	[34]	96	96	93	103.0	0.13	**3,758**	73
	LED-128		128	64	1,872	3.4	0.18	**1,265**	21
	PRESENT-128	[41]	128	64	559	11.4	0.18	**1,391**	59
	PRESENT-128	[11]	128	64	32	200.0	0.18	**1,886**	562
	HIGHT	[26]	128	64	34	188.0	0.25	**3,048**	203
	AES	[38]	128	128	226	56.6	0.13	**2,400**	98
	DESXL	[32]	184	64	144	44.4	0.18	**2,168**	95
Hard-wired Keys	LED-64		64	64	1,280	5.13	0.18	**688**	108
	PRINTcipher-48	[30]	80	48	768	6.2	0.18	**402**	387
	KTANTAN64	[13]	80	64	255	25.1	0.13	**688**	530
	LED-80*		80	64	1,872	3.4	0.18	**690**	72
	LED-96*		96	64	1,872	3,42	0.18	**695**	71
	LED-128		128	64	1,872	3.42	0.18	**700**	70
	PRINTcipher-96	[30]	160	96	3072	3.13	0.18	**726**	59

(2×4 bits) as a unit for the lookup table. Hence SubCells, ShiftRows and MixColumnsSerial can all be achieved using eight table lookups and XORs.

Overall, we need to access $8 \times 32 \times 2 = 512$ 32-bit words of memory (or $8 \times 32 = 256$ 64-bit words of memory). In contrast, an AES implementation with four tables of 256 entries would require $(16+4) \times 10 = 200$ accesses. This suggests that LED-64 should be about 2.5 times slower than AES on 32-bit platforms with table-based implementations, and similarly LED-128 will be 3.8 slower than AES, while the optimized table-based implementation runs 57 and 86 cycles per byte for LED-64 and LED-128, respectively.

5 Conclusion

In this paper we have presented the block cipher LED. Clearly, given its novelty, the cipher should not be used in applications until there has been sufficient

independent analysis. Nevertheless, we hope that our design is of some interest and we have focused our attention on what seem to be the neglected areas of key schedule design and protection against related-key attacks. Furthermore, we have done so while working in one of the more challenging design spaces—that of constrained hardware implementation—and we have proposed one of the smallest block ciphers in the literature (for comparable choices of parameters) while striving to maintain a competitive performance in software. Additional information on LED will be made available via https://sites.google.com/site/ledblockcipher/ and we welcome all comments and analysis.

References

1. Abdelraheem, M., Blondeau, C., Naya-Plasencia, M., Videau, M., Zenner, E.: Cryptanalysis of Armadillo-2, http://eprint.iacr.org/2011/160.pdf
2. Ågren, M.: Some Instant- and Practical-Time Related-Key Attacks on KTANTAN32/48/64, http://eprint.iacr.org/2011/140
3. Aumasson, J.-P., Dinur, I., Meier, W., Shamir, A.: Cube Testers and Key Recovery Attacks on Reduced-Round MD6 and Trivium. In: Dunkelman, O. (ed.) FSE 2009. LNCS, vol. 5665, pp. 1–22. Springer, Heidelberg (2009)
4. Aumasson, J.-P., Henzen, L., Meier, W., Naya-Plasencia, M.: Quark: A Lightweight Hash. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 1–15. Springer, Heidelberg (2010)
5. Badel, S., Dagtekin, N., Nakahara, J., Ouafi, K., Reffé, N., Sepehrdad, P., Susil, P., Vaudenay, S.: ARMADILLO: A Multi-purpose Cryptographic Primitive Dedicated to Hardware. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 398–412. Springer, Heidelberg (2010)
6. Barreto, P., Rijmen, V.: The Whirlpool Hashing Function. Submitted to NESSIE (September 2000), http://www.larc.usp.br/~pbarreto/WhirlpoolPage.html (revised May 2003)
7. Biryukov, A., Wagner, D.: Slide Attacks. In: Knudsen, L.R. (ed.) FSE 1999. LNCS, vol. 1636, pp. 245–259. Springer, Heidelberg (1999)
8. Biryukov, A., Wagner, D.: Advanced Slide Attacks. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 589–606. Springer, Heidelberg (2000)
9. Biryukov, A., Khovratovich, D.: Related-Key Cryptanalysis of the Full AES-192 and AES-256. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 1–18. Springer, Heidelberg (2009)
10. Blondeau, C., Naya-Plasencia, M., Videau, M., Zenner, E.: Cryptanalysis of ARMADILLO2, http://eprint.iacr.org/2011/160
11. Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y., Vikkelsoe, C.: PRESENT: An Ultra-Lightweight Block Cipher. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 450–466. Springer, Heidelberg (2007)
12. Boura, C., Canteaut, A., De Cannière, C.: Higher-Order Differential Properties of Keccak and *Luffa*. In: Joux, A. (ed.) FSE 2011. LNCS, vol. 6733, pp. 252–269. Springer, Heidelberg (2011)
13. De Cannière, C., Dunkelman, O., Knežević, M.: KATAN and KTANTAN — A Family of Small and Efficient Hardware-Oriented Block Ciphers. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 272–288. Springer, Heidelberg (2009)

14. De Cannière, C., Preneel, B.: Trivium. In: Robshaw and Billet [43], pp. 244–266
15. Choy, J., Zhang, A., Khoo, K., Henricksen, M., Poschmann, A.: AES variants secure against related-key differential and boomerang attacks. In: Ardagna, C.A., Zhou, J. (eds.) WISTP 2011. LNCS, vol. 6633, pp. 191–207. Springer, Heidelberg (2011), http://eprint.iacr.org/2011/072
16. Daemen, J., Rijmen, V.: AES Proposal: Rijndael. NIST AES proposal (1998)
17. Daemen, J., Knudsen, L.R., Rijmen, V.: The Block Cipher SQUARE. In: Biham, E. (ed.) FSE 1997. LNCS, vol. 1267, pp. 149–165. Springer, Heidelberg (1997)
18. Dunkelman, O., Keller, N., Shamir, A.: A Practical-Time Related-Key Attack on the KASUMI Cryptosystem Used in GSM and 3G Telephony. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 393–410. Springer, Heidelberg (2010)
19. Engels, D., Fan, X., Gong, G., Hu, H., Smith, E.M.: Ultra-Lightweight Cryptography for Low-Cost RFID Tags: Hummingbird Algorithm and Protocol, http://www.cacr.math.uwaterloo.ca/techreports/2009/cacr2009-29.pdf
20. Engels, D., Saarinen, M.-J.O., Smith, E.M.: The Hummingbird-2 Lightweight Authenticated Encryption Algorithm, http://eprint.iacr.org/2011/126.pdf
21. Gilbert, H., Peyrin, T.: Super-Sbox Cryptanalysis: Improved Attacks for AES-Like Permutations. In: Hong and Iwata [27], pp. 365–383
22. Gong, Z., Nikova, S., Law, Y.-W.: A New Family of Lightweight Block Ciphers. In: Juels, A., Paar, C. (eds.) RFIDSec 2011. Springer, Heidelberg (to appear, 2011), http://www.rfid-cusp.org/rfidsec/files/RFIDSec2011DraftPapers.zip
23. Guo, J., Peyrin, T., Poschmann, A.: The PHOTON Family of Lightweight Hash Functions. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 222–239. Springer, Heidelberg (2011)
24. Hämäläinen, P., Alho, T., Hännikäinen, M., Hämäläinen, T.D.: Design and Implementation of Low-Area and Low-Power AES Encryption Hardware Core. In: DSD, pp. 577–583 (2006)
25. Hell, M., Johansson, T., Maximov, A., Meier, W.: The Grain Family of Stream Ciphers. In: Robshaw and Billet [43], pp. 179–190
26. Hong, D., Sung, J., Hong, S., Lim, J., Lee, S., Koo, B.S., Lee, C., Chang, D., Lee, J., Jeong, K., Kim, H., Kim, J., Chee, S.: HIGHT: A New Block Cipher Suitable for Low-Resource Device. In: Goubin, L., Matsui, M. (eds.) CHES 2006. LNCS, vol. 4249, pp. 46–59. Springer, Heidelberg (2006)
27. Hong, S., Iwata, T. (eds.): FSE 2010. LNCS, vol. 6147. Springer, Heidelberg (2010)
28. Khovratovich, D., Nikolic, I.: Rotational Cryptanalysis of ARX. In: Hong and Iwata [27], pp. 333–346
29. Knudsen, L.R., Rijmen, V.: Known-Key Distinguishers for Some Block Ciphers. In: Kurosawa, K. (ed.) ASIACRYPT 2007. LNCS, vol. 4833, pp. 315–324. Springer, Heidelberg (2007)
30. Knudsen, L.R., Leander, G., Robshaw, M.J.B.: PRINTcipher: A Block Cipher for IC-Printing. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 16–32. Springer, Heidelberg (2010)
31. Lamberger, M., Mendel, F., Rechberger, C., Rijmen, V., Schläffer, M.: Rebound Distinguishers: Results on the Full Whirlpool Compression Function. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 126–143. Springer, Heidelberg (2009)
32. Leander, G., Paar, C., Poschmann, A., Schramm, K.: New Lightweight DES Variants. In: Biryukov, A. (ed.) FSE 2007. LNCS, vol. 4593, pp. 196–210. Springer, Heidelberg (2007)

33. Lim, C., Korkishko, T.: mCrypton – A Lightweight Block Cipher for Security of Low-Cost RFID Tags and Sensors. In: Kwon, T., Song, J., Yung, M. (eds.) WISA 2005. LNCS, vol. 3786, pp. 243–258. Springer, Heidelberg (2006)
34. Mace, F., Standaert, F.-X., Quisquater, J.-J.: ASIC Implementations of the Block Cipher SEA for Constrained Applications. In: RFID Security - RFIDsec 2007, Workshop Record, Malaga, Spain, pp. 103–114 (2007)
35. May, L., Henricksen, M., Millan, W.L., Carter, G., Dawson, E.: Strengthening the Key Schedule of the AES. In: Batten, L., Seberry, J. (eds.) ACISP 2002. LNCS, vol. 2384, pp. 226–240. Springer, Heidelberg (2002)
36. Mendel, F., Peyrin, T., Rechberger, C., Schläffer, M.: Improved Cryptanalysis of the Reduced Grøstl Compression Function, ECHO Permutation and AES Block Cipher. In: Jacobson Jr., M.J., Rijmen, V., Safavi-Naini, R. (eds.) SAC 2009. LNCS, vol. 5867, pp. 16–35. Springer, Heidelberg (2009)
37. Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: The Rebound Attack: Cryptanalysis of Reduced Whirlpool and Grøstl. In: Dunkelman, O. (ed.) FSE 2009. LNCS, vol. 5665, pp. 260–276. Springer, Heidelberg (2009)
38. Moradi, A., Poschmann, A., Ling, S., Paar, C., Wang, H.: Pushing the Limits: A Very Compact and a Threshold Implementation of AES. In: Paterson, K. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 69–88. Springer, Heidelberg (2011)
39. Nikolić, I.: Tweaking AES. In: Biryukov, A., Gong, G., Stinson, D.R. (eds.) SAC 2010. LNCS, vol. 6544, pp. 198–210. Springer, Heidelberg (2011)
40. Park, S., Sung, S.H., Lee, S., Lim, J.: Improving the Upper Bound on the Maximum Differential and the Maximum Linear Hull Probability for SPN Structures and AES. In: Johansson, T. (ed.) FSE 2003. LNCS, vol. 2887, pp. 247–260. Springer, Heidelberg (2003)
41. Poschmann, A.: Lightweight Cryptography - Cryptographic Engineering for a Pervasive World. Number 8 in IT Security. Europäischer Universitätsverlag, Published: Ph.D. Thesis, Ruhr University Bochum (2009)
42. Robshaw, M.J.B.: Searching for Compact Algorithms: CGEN. In: Nguyen, P. (ed.) VIETCRYPT 2006. LNCS, vol. 4341, pp. 37–49. Springer, Heidelberg (2006)
43. Robshaw, M.J.B., Billet, O. (eds.): New Stream Cipher Designs. LNCS, vol. 4986. Springer, Heidelberg (2008)
44. Rolfes, C., Poschmann, A., Leander, G., Paar, C.: Ultra-Lightweight Implementations for Smart Devices – Security for 1000 Gate Equivalents. In: Grimaud, G., Standaert, F.-X. (eds.) CARDIS 2008. LNCS, vol. 5189, pp. 89–103. Springer, Heidelberg (2008)
45. Saarinen, M.-J.O.: Cryptanalysis of Hummingbird-1. In: Joux, A. (ed.) FSE 2011. LNCS, vol. 6733, pp. 328–341. Springer, Heidelberg (2011)