mCrypton – A Lightweight Block Cipher for Security of Low-Cost RFID Tags and Sensors

Chae Hoon Lim and Tymur Korkishko

Dept. of Internet Engineering, Sejong University, Korea
Samsung Advanced Institute of Technology (SAIT), Korea
chlim@sejong.ac.kr, k.tymur@samsung.com

Abstract. This paper presents a new 64-bit block cipher **mCrypton** with three key size options (64 bits, 96 bits and 128 bits), specifically designed for use in resource-constrained tiny devices, such as low-cost RFID tags and sensors. It's designed by following the overall architecture of Crypton but with redesign and simplification of each component function to enable much compact implementation in both hardware and software. A simple hardware implementation of mCrypton is also presented to demonstrate its suitability to our target applications. Our prototype implementation based on the straightforward 1 cycle/round architecture just requires about 3500 to 4100 gates for both encryption and decryption, and about 2400 to 3000 gates for encryption only (under 0.13μm CMOS technology). The result shows that the hardware complexity of mCrypton is quite well within an economic range of low-cost RFID tags and sensors. A more compact implementation under development promises that further size reduction around 30% could be achievable using the 5 cycles/round architecture.

1 Background

The ubiquitous computing paradigm pursues true elimination of time and space barriers by embedding wirelessly networked processors in everyday objects and thereby making a variety of services available to users all the time everywhere. The ubiquitous computing vision however could bring a great deal of security risks due to the ubiquity of tiny interconnected devices embedded into everyday environments [2, 11]. In particular, much research attention has been recently paid to the security and privacy issues of RFID and sensor networks [4, 9].

Traditionally, block ciphers have been used as a basic security building block for most resource-constrained applications, such as smart cards and security tokens. The same and even more compelling reasoning can apply to tiny ubiquitous devices such as low-cost RFID tags and sensors. In such resource-constrained devices it is undesirable or even impossible to implement multiple security primitives for cost reason. So a compact, hardware- and software-efficient block cipher could be the most promising candidate for security in such applications.

Design Constraints. Typical ubiquitous computing devices however impose new constraints in block cipher design due to their size and shape [11]. First of all, the chip area required for hardware implementation of a block cipher should

J. Song, T. Kwon, and M. Yung (Eds.): WISA 2005, LNCS 3786, pp. 243–258, 2006.

be small enough not to much increase the cost of ubiquitous devices due to the added security feature. For example, in the case of low-cost RFID tags, one of the most resource-scarce ubiquitous computing devices, it is estimated that security resources available to a 5 cent design may be limited to hundreds of bits of storage, roughly 500-5,000 gates [12]. Note that low-cost RFID tags only have a simple logic for data processing even without CPU, so the only way to implement a crypto algorithm would be its hardware integration into tag chips.

Another and more critical issue in tiny ubiquitous devices is the limited amount of power available. Only a small, finite amount of energy may be available through a miniature battery to the tiny processors embedded in ubiquitous computing devices such as mote-class sensors. Even more cheap devices, such as passive RFID tags, cannot be self-powered and thus should obtain energy from larger communicating devices through electromagnetic coupling. This limited power availability places a bound on the total amount of computation such devices can perform, rather than on the speed. Therefore, the most relevant performance figure here might be bits per joule rather than the traditional bits per second. In this respect, most widely-used block ciphers such as AES may not be much attractive for use in such limited computing environments.

Design Objectives and Choices. The block cipher mCrypton is designed with above new constraints in low-cost ubiquitous computing devices in mind. The goal is to design a block cipher with extreme efficiency in resource usage and power consumption, so that they can be hardware integrated or software implemented in tiny processors embedded in inexpensive everyday commodities. The design of mCrypton is based on the overall architecture of Crypton [7] (mCrypton actually stands for a miniature of Crypton and can be thought of as a 64-bit variant of Crypton with variable key sizes). The basic building blocks were redesigned to fit the block/key sizes and the overall architecture was a little bit simplified for better implementation efficiency. The key scheduling algorithm was also completely redesigned.

The main objective of designing mCrypton is to come up with a block cipher optimized for resource-constrained applications, so we decided to use the parameters of 64-bit block length and variable key lengths of 64 bits, 96 bits and 128 bits. Note that a large volume of bulk data encryption is unnecessary or even impossible in most tiny ubiquitous computing devices. Therefore, there will be no security concern with small block size and it will be a natural choice for new design of a block cipher with specific application to extremely resource-constrained devices. We also decided to provide three key size options (for minimal, moderate and standard security, respectively) for better flexibility of cost-security tradeoffs. Note that production cost may be one of most critical factors in practice for large scale deployment of tiny devices such as low-cost RFID tags.

Minimizing power consumption certainly should be one of most important considerations in software/hardware design for tiny ubiquitous devices. In general, a block cipher will be more power-efficient in hardware/software implementations if it can be implemented using less amount of computing resources. So one obvious goal in designing a block cipher should be to achieve low complex-

ity in hardware and software while providing sufficient security. Furthermore, power consumption in CMOS hardware largely depends on signal transition frequency during the processing. For example, branched signal paths may cause dynamic hazards (multiple signal transitions before being stable) due to different arrival times at a logic gate, which consumes extra power. So, from the standpoint of algorithm design, it would be preferable to make signal paths as uniform as possible and to reduce signal transition probability as possible as one can. These considerations would provide good reasons of basing our design on the overall structure of Crypton: It has a regular and quite uniform structure with its component functions efficiently implementable in both hardware and software.

Related Work. There is no published literature recognized by the authors for new design of block ciphers targeted to tiny ubiquitous devices. As a related work to RFID security, Bono et al. reported successful reverse engineering and key cracking for the secret algorithm (with only 40-bit key size) embedded in the currently circulating TI RFID tags [1]. Their work once again signifies the importance of using well-scrutinized open crypto algorithm for wide deployment of security products. As a related work to efficient implementation on RFID tags, Feldhofer et al. presented an 8-bit architecture, encryption-only mode implementation of AES for RFID authentication, which consumes about 3,600 gates and requires about 1,000 clock cycles at 100KHz for one block encryption [3].

On the other hand, the TinySec implementation experience provides valuable information on feasibility of software implementation of a block cipher in low-cost sensor nodes [5]. Implementation experiments in Mica2 mote (8 MHz 8-bit Atmel ATMEGA128L MCU with 4KB of RAM, 128 KB of flash (program space) and 4KB of EEPROM, Chipcon radio module of up to 19.2 Kbps bandwidth) showed that there was almost no performance degradation even with software implementation of RC5 in Mica2 sensor nodes. Of course, the situation may be different for sensor nodes with faster radio, such as Telos (8 MHz 16-bit TI MSP430 MCU with 4KB of RAM, 60KB of flash and 16KB of EEPROM) whose IEEE 802.15.4 radio can transmit at a much faster data rate of 250 Kbps [10].

General rule of thumb on the performance of security primitives required for sensor nodes is that one block processing should be completed in under a few byte times to avoid performance degradation due to the added cryptographic operation, where byte time refers to the time required to transmit a single byte over the radio [5]. Interestingly, Law et al. reported bench-marking data for various block ciphers on TI MSP430 MCU adopted by the Telos mote [6]. Their performance result (on speed-optimized counter mode) shows that RC5 requires 85μsec (at 8MHz) per block encryption using 5.2Kbytes of code memory, while AES requires 27μsec using 13.3Kbytes of memory. Since the byte time of Telos mote is 32μsec, we can see that software implementation of cryptographic operations may be acceptable even for low-end sensor nodes. This also shows that software efficiency (in particular on low-end 8-bit and 16-bit microprocessors) should be an important consideration in designing a block cipher for ubiquitous computing security.

Notation. The following notation will be used throughout this paper:

- A 4-bit string is denoted by nibble and one byte of data is represented by two 4-bit nibbles numbered from left to right (i.e., $b = b_0 \| b_1$). Similarly, one word of data consists of two bytes numbered from left to right.
- An 8-byte data consisting of 16 nibbles $\{a_0, a_1, \cdots, a_{15}\}$ is internally represented as a 4×4 nibble array as follows:

$$A = \begin{pmatrix} a_0 & a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 & a_7 \\ a_8 & a_9 & a_{10} & a_{11} \\ a_{12} & a_{13} & a_{14} & a_{15} \end{pmatrix} = \begin{pmatrix} A_r[0] \\ A_r[1] \\ A_r[2] \\ A_r[3] \end{pmatrix} = (A_c[0]A_c[1]A_c[2]A_c[3]),$$

  where $A_r[i]$ and $A_c[i]$ denote the i-th row and column of A, respectively.

- For an array A, A^t denotes transposition of A.
- $X^{\lll k}$: left rotation of a 16-bit word X by k-bit positions.
- $f \circ g$: composition of functions f and g, i.e., $(f \circ g)(x) = f(g(x))$.
- $\bullet, \oplus$: bit-wise logical operations for AND and XOR, respectively.

2 Algorithm Specifications

mCrypton processes an 8-byte data block by representing it into a 4×4 nibble array as in Crypton [7]. The round transformation consists of four steps: nibble-wise substitution, column-wise bit permutation, column-to-row transposition, and then key addition. The encryption process involves 12 repetitions of the same round transformation. The decryption process can be made almost identical to the encryption process with a different key schedule.

2.1 Basic Building Blocks

Nonlinear Substitution γ. The nonlinear transformation γ consists of nibble-wise substitutions on a 4×4 nibble array using four 4-bit S-boxes, S_i ($0 \le i \le 3$), such that $S_2 = S_0^{-1}$ and $S_3 = S_1^{-1}$ (see Section 3.2 for details). Each component substitution function γ_i operates on the 4-nibble vector of the i-th row (or column). That is, for a 4-nibble word $a = (a_0, a_1, a_2, a_3)$

$$\gamma_i(a) = (S_i(a_0), S_{i+1}(a_1), S_{i+2}(a_2), S_{i+3}(a_3)),$$

where indices are taken modulo 4 (see Fig.1).

a_0	a_1	a_2	a_3	$\longrightarrow$	$S_0(a_0)$	$S_1(a_1)$	$S_2(a_2)$	$S_3(a_3)$
a_4	a_5	a_6	a_7		$S_1(a_4)$	$S_2(a_5)$	$S_3(a_6)$	$S_0(a_7)$
a_8	a_9	a_{10}	a_{11}		$S_2(a_8)$	$S_3(a_9)$	$S_0(a_{10})$	$S_1(a_{11})$
a_{12}	a_{13}	a_{14}	a_{15}		$S_3(a_{12})$	$S_0(a_{13})$	$S_1(a_{14})$	$S_2(a_{15})$

Fig. 1. The nibble-wise substitution γ

The transformation γ and γ^{-1} can thus be defined for 4×4 data array A by

$$\begin{aligned}\gamma(A) &= (\gamma_0(A_c[0])\ \gamma_1(A_c[1])\ \gamma_2(A_c[2])\ \gamma_3(A_c[3])) \\ &= (\gamma_0(A_r[0])\ \gamma_1(A_r[1])\ \gamma_2(A_r[2])\ \gamma_3(A_r[3]))^t \\ \gamma^{-1}(A) &= (\gamma_2(A_c[0])\ \gamma_3(A_c[1])\ \gamma_0(A_c[2])\ \gamma_1(A_c[3])) \\ &= (\gamma_2(A_r[0])\ \gamma_3(A_r[1])\ \gamma_0(A_r[2])\ \gamma_1(A_r[3]))^t.\end{aligned}$$

Note that the symmetry in S-box arrangement ensures that γ/γ^{-1} and τ commute, i.e., $\tau \circ \gamma = \gamma \circ \tau$ and $\tau \circ \gamma^{-1} = \gamma^{-1} \circ \tau$ (see below the definition for τ). Obviously, we have $\gamma_i(a) = \gamma_0(a^{\ll 16-4i})^{\ll 4i}$.

Bit Permutation π. The bit permutation π bit-wise mixes each column of 4×4 array A using column permutation π_i for each column i $(0 \le i \le 3)$ (Fig.2):

$$\pi(A) = (\pi_0(A_c[0])\ \pi_1(A_c[1])\ \pi_2(A_c[2])\ \pi_3(A_c[3]))$$

Each component column permutation π_i is defined for nibble columns $a = (a_0, a_1, a_2, a_3)^t$ and $b = (b_0, b_1, b_2, b_3)^t$ by

$$b = \pi_i(a) \ \Leftrightarrow\ b_j = \oplus_{k=0}^{3}(m_{i+j+k \ mod\ 4} \bullet a_k),$$

where four masking nibbles m_i's are given by

$$m_0 = \mathtt{1110}_2,\ m_1 = \mathtt{1101}_2,\ m_2 = \mathtt{1011}_2,\ m_3 = \mathtt{0111}_2.$$

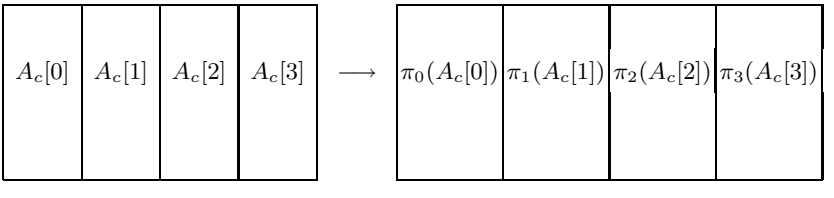

Fig. 2. The column-wise bit permutation π

Note that the π transformation is involution (i.e. $\pi = \pi^{-1}$) and satisfies the shift property: $\pi_i(a) = \pi_0(a)^{\ll 4i}$. and $\pi_i(a^{\ll 4k}) = \pi_i(a)^{\ll 16-4k}$,where cyclic shift on a column vector should be interpreted aover its row-transformed equivalent.

Column-to-Row Transposition τ. It simply moves the nibble at the (i, j)-th position to the (j, i)-th position, i.e., $B = \tau(A) \Leftrightarrow b_{ij} = a_{ji}$. Obviously, $\tau^{-1} = \tau$.

Key Addition σ. For a round key $K = (K[0], K[1], K[2], K[3])$, $B = \sigma_K(A)$ is defined by $B_r[i] = A_r[i] \oplus K[i]$ $(0 \le i \le 3)$.

2.2 Encryption and Decryption

The encryption round transformation ρ of mCrypton consists of applying γ, π, τ and σ in sequence to the 4×4 data array. More specifically, the round functions for encryption and decryption are defined (for round key K) by

$$\rho_K = \sigma_K \circ \tau \circ \pi \circ \gamma,$$
$$\rho_K^{-1} = \gamma^{-1} \circ \pi \circ \tau \circ \sigma_K$$

For 4×4 data array A and round key K, we can express $B = \rho_K(A)$ using the component functions γ_i's and π_i's as

$$B_c[i] = \pi_i(\gamma_i(A_c[i]))^t \oplus K[i] = \pi_0(\gamma_0(A_c[i]^{\ll 16-4i}))^t \oplus K[i] \quad (0 \le i \le 3).$$

Let us define ρ'_K as $\rho'_K = \sigma_K \circ \tau \circ \pi \circ \gamma^{-1}$, i.e., the round transformation obtained by replacing γ by γ^{-1} in ρ_K, which will be used as a decryption round transformation below. Then $\rho'_K(A)$ can be similarly expressed as:

$$B_c[i] = \pi_i(\gamma_{i+2}(A_c[i]))^t \oplus K[i] = \pi_i((\gamma_i(A_c[i]^{\ll 8})^{\ll 8}))^t \oplus K[i] \quad (0 \le i \le 3)$$

Let K_e^i be the i-th encryption round key consisting of 4 words, derived from a user-supplied key K using the encryption key schedule. The encryption transformation E_K of mCrypton under key K consists of an initial key addition and 12 times repetitions of ρ and then a final output transformation. More specifically, E_K can be described as

$$E_K = \phi \circ \rho_{K_e^{12}} \circ \rho_{K_e^{11}} \circ \cdots \circ \rho_{K_e^2} \circ \rho_{K_e^1} \circ \sigma_{K_e^0},$$

where ϕ is defined by $\phi = \tau \circ \pi \circ \tau$.

Since γ^{-1} uses the same S-boxes as γ only with a different arrangement, we can imagine that decryption process can be made to have almost the same architecture as encryption process by using ϕ-transformed round keys. The decryption transformation D_K can be shown to have almost the same form as E_K:

$$D_K = \phi \circ \rho'_{K_d^{12}} \circ \rho'_{K_d^{11}} \circ \cdots \circ \rho'_{K_d^2} \circ \rho'_{K_d^1} \circ \sigma_{K_d^0},$$

where the decryption round keys are defined by

$$K_d^{r-i} = \phi(K_e^i) \quad \text{for} \quad 0 \le i \le 12.$$

Note that the output transformation ϕ can be incorporated into the final round as $\phi \circ \rho_{K_e^{12}} = \tau \circ \pi \circ \tau \circ \ (\sigma_{K_e^{12}} \circ \tau \circ \pi \circ \gamma) = \sigma_{K_d^0} \circ \tau \circ \gamma$.

2.3 Key Scheduling

mCrypton supports three key sizes: 64 bits, 96 bits and 128 bits. The 64-bit key size may certainly be not enough for adequate security in general computing environments, but it may provide still good security in resource-constrained, cost-driven applications such as low-cost RFID tags. On the other hand, with

96-bit keys we will be able to achieve moderate security in most applications. In general, however, it would be more desirable to use the current standard key size of 128 bits in any application if there is no severe restriction on available resources.

The key scheduling algorithm for mCrypton consist of two stages: round key generation through nonlinear S-box transformation and then key variables update through simple rotations (word-wise rotation and bitwise rotation within word). The simple linear key variables update makes it easy to carry out backward processing for decryption key schedule and the nonlinear round key generation together with linear update of key variables in each round provides the basis for the security against various attacks on key scheduling algorithms.

Let $K = \{K[i]\}_{i=0}^{t-1} = (K[0], K[1] \cdots K[t-1])$ be the user key ($t = 4, 6, 8$ for key sizes of 64, 96, 128 bits, respectively), where $K[i]$ represents the i-th 16-bit key word in K. Let $C[i]$ be the round constant for round i (we will regard the initial key addition as round 0 for notational purpose). Each round constant $C[i]$ consists of four identical nibbles, i.e., $C[i] = \texttt{0x}c_ic_ic_ic_i$, where c_i is generated by x^i in $\mathrm{GF}(2^4)$ defined by the irreducible polynomial $f(x) = x^4 + x + 1$ (That is, $c_0 = 1, c_1 = 2, \cdots, c_4 = 3, c_5 = 6, \cdots$, etc.).

Specific key schedules for each key size are now presented in the following. Here $U = \{U[i]\}_{i=0}^{t-1}$ and $V = \{V[i]\}_{i=0}^{t-1}$ will be used as key registers for state update in encryption and decryption key schedules, respectively. Note that $\phi_i = \tau \circ \pi_i \circ \tau$ $(0 \le i \le 3)$. The S-box operation on a word in the key schedule is performed in nibble-wise with the same S-box S_0, i.e., for $a = (a_0, a_1, a_2, a_3)$, $S(a) = (S_0(a_0), S_0(a_1), S_0(a_2), S_0(a_3))$. We also use four masking words M_i to take the i-th nibble from word, i.e., $M_0 = \texttt{0xf000}$, $M_1 = \texttt{0x0f00}$, $M_2 = \texttt{0x00f0}$, $M_3 = \texttt{0x000f}$.

Key Schedule for 64-Bit Keys

– Encryption round keys: The key register U is first initialized with K and then encryption round keys are computed for round $r = 0, 1, \cdots, 12$ as:

$$\begin{aligned} T &\leftarrow S(U[0]) \oplus C[r], \quad T_i \leftarrow T \bullet M_i \quad (0 \le i \le 3), \\ K_e^r &\leftarrow (U[1] \oplus T_0, U[2] \oplus T_1, U[3] \oplus T_2, U[0] \oplus T_3) \\ U &\leftarrow (U[1], U[2], U[3], U[0]^{\lll 3}) \end{aligned}$$

– Decryption round keys: The key register V is first initialized as

$$V \leftarrow (K[0]^{\lll 9}, K[1]^{\lll 9}, K[2]^{\lll 9}, K[3]^{\lll 9}).$$

Then decryption round keys are successively computed as follows: for round $r = 0, 1, \cdots, 12$,

$$\begin{aligned} T &\leftarrow S(V[0]) \oplus C[12 - r], \quad T_i \leftarrow T \bullet M_i \quad (0 \le i \le 3), \\ K_d^r &\leftarrow (\phi_0(V[1] \oplus T_0), \phi_1(V[2] \oplus T_1), \phi_2(V[3] \oplus T_2), \phi_3(V[0] \oplus T_3)) \\ V &\leftarrow (V[3]^{\lll 13}, V[0], V[1], V[2]) \end{aligned}$$

Key Schedule for 96-Bit Keys

– Encryption round keys: The key register U is first initialized with the user key K and encryption round keys are successively computed as follows: for round $r = 0, 1, \cdots, 12$,

$$\begin{aligned} T &\leftarrow S(U[0]) \oplus C[r], \quad T_i \leftarrow T \bullet M_i \quad (0 \le i \le 3), \\ K_e^r &\leftarrow (U[1] \oplus T_0, U[2] \oplus T_1, U[3] \oplus T_2, U[4] \oplus T_3) \\ U &\leftarrow (U[5], U[0]^{\lll 3}, U[1], U[2], U[3]^{\lll 8}, U[4]) \end{aligned}$$

– Decryption round keys: The key register V is first initialized as

$$V \leftarrow (K[0]^{\lll 6}, K[1]^{\lll 6}, K[2]^{\lll 6}, K[3]^{\lll 6}, K[4]^{\lll 6}, K[5]^{\lll 6}),$$

and decryption round keys are successively computed as follows: for round $r = 0, 1, \cdots, 12$,

$$\begin{aligned} T &\leftarrow S(V[0]) \oplus C[12 - r], \quad T_i \leftarrow T \bullet M_i \quad (0 \le i \le 3), \\ K_d^r &\leftarrow (\phi_0(V[1] \oplus T_0), \phi_1(V[2] \oplus T_1), \phi_2(V[3] \oplus T_2), \phi_3(V[4] \oplus T_3)) \\ V &\leftarrow (V[1]^{\lll 13}, V[2], V[3], V[4]^{\lll 8}, V[5], V[0]) \end{aligned}$$

Key Schedule for 128-Bit Keys

– Encryption round keys: The key register U is first initialized with the user key K and encryption round keys are successively computed as follows: for round $r = 0, 1, \cdots, 12$,

$$\begin{aligned} T &\leftarrow S(U[0]) \oplus C[r], \quad T_i \leftarrow T \bullet M_i \quad (0 \le i \le 3), \\ K_e^r &\leftarrow (U[1] \oplus T_0, U[2] \oplus T_1, U[3] \oplus T_2, U[4] \oplus T_3) \\ U &\leftarrow (U[5], U[6], U[7], U[0]^{\lll 3}, U[1], U[2], U[3], U[4]^{\lll 8}) \end{aligned}$$

– Decryption round keys: The key register V is first initialized as

$$\begin{aligned} (V[0], V[1], \cdots, V[7]) \leftarrow (&K[4]^{\lll 3}, K[5]^{\lll 14}, K[6]^{\lll 3}, K[7]^{\lll 14}, \\ &K[0]^{\lll 14}, K[1]^{\lll 3}, K[2]^{\lll 14}, K[3]^{\lll 3}), \end{aligned}$$

and decryption round keys are successively computed as follows: for round $r = 0, 1, \cdots, 12$,

$$\begin{aligned} T &\leftarrow S(V[0]) \oplus C[12 - r], \quad T_i \leftarrow T \bullet M_i \quad (0 \le i \le 3), \\ K_d^r &\leftarrow (\phi_0(V[1] \oplus T_0), \phi_1(V[2] \oplus T_1), \phi_2(V[3] \oplus T_2), \phi_3(V[4] \oplus T_3)) \\ V &\leftarrow (V[3]^{\lll 13}, V[4], V[5], V[6], V[7]^{\lll 8}, V[0], V[1], V[2]). \end{aligned}$$

3 Security Analysis

3.1 Diffusion Property of Linear Transformation

First note that it suffices to consider any one component transformation π_i of π to examine the diffusion property of π, since π acts on each column independently. It

is also easy to see that any column vector with n ($n < 4$) nonzero nibbles is transformed by π_i into a column vector with at least $4 - n$ nonzero nibbles (this number 4 is called as the diffusion order of π_i). This is due to the operation of exclusive-or sum in π. More important is that such input vectors giving minimum diffusion take only a very small fraction of all possible inputs due to the masked bit permutation.

Let us examine in more detail the set of 16-bit numbers giving minimal diffusion. For this, we define two sets of 4-bit values, Ω_x and Ω_y, as

$$\Omega_x = \{0x1, 0x2, 0x4, 0x8\}, \quad \Omega_y = \{0x5, 0xa\} \cup \Omega_x.$$

Let I_j be a set of input vectors with j nonzero nibbles which are transformed by π_i into output vectors with $4 - j$ nonzero nibbles. Then all possible 16-bit values with minimum diffusion can be obtained as:

$$\begin{aligned}
I_1 &= \{(x,0,0,0)^t, (0,x,0,0)^t, (0,0,x,0)^t, (0,0,0,x)^t \mid x \in \Omega_x\},\\
I_2 &= \{(x,x,0,0)^t, (0,x,x,0)^t, (0,0,x,x)^t, (x,0,0,x)^t \mid x \in \Omega_x\},\\
I_2^* &= \{(y,0,y,0)^t, (0,y,0,y)^t \mid y \in \Omega_y\},\\
I_3 &= \{(0,x,x,x)^t, (x,0,x,x)^t, (x,x,0,x)^t, (x,x,x,0)^t \mid x \in \Omega_x\}.
\end{aligned}$$

Then, it is easy to see that an element in I_j is transformed by π_i into some element in I_{4-j} depending on the nonzero value x. The set I_2^*, containing two separated nonzero nibbles, is somewhat special: it has 12 elements and is closed under π_i. In summary,

$$\begin{aligned}
&a \in I_j \Rightarrow \pi_i(a) \in I_{4-j} \text{ for } j = 1, 2, 3,\\
&a \in I_2^* \Rightarrow \pi_i(a) \in I_2^*.
\end{aligned}$$

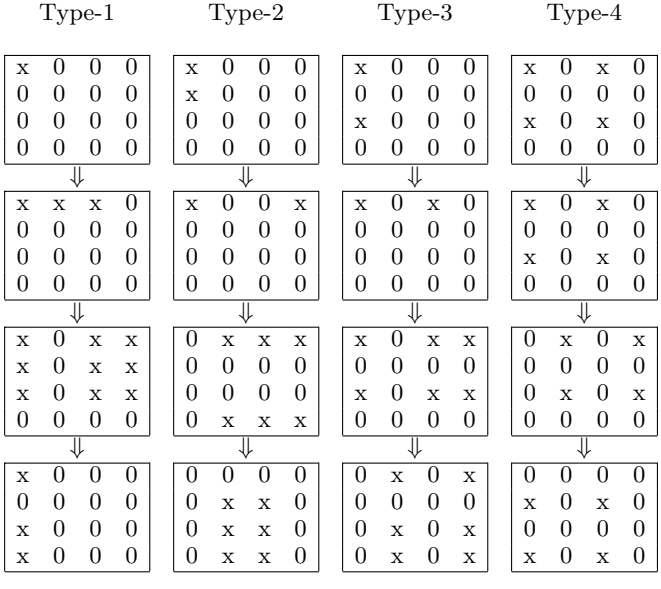

Fig. 3. Examples of active nibble propagation in each diffusion type (x : active nibble)

Since $|I_1| = |I_2| = |I_3| = 16$ and $|I_2^*| = 12$, we can see that there are only 60 vectors with minimum diffusion. Observe that the nonzero nibbles in each input vector should have the same value to achieve minimum diffusion. Also note that the two values in $\Omega_y - \Omega_x$ can only occur in the set I_2^*.

Now let us examine the diffusion effect of $\tau \circ \pi$ through consecutive rounds by assuming that in each round the S-box output can take any desired value, irrespective of the input value. This assumption is to maximally take into account the probabilistic nature of S-box transformation without details of the S-box characteristics. Since it suffices to consider worst-case propagations, we only examine inputs with 1, 2, or 3 nonzero nibbles in any one column vector of a 4×4 nibble array, say the first column. The result is depicted in Fig.3. The sum of the number of nonzero nibbles throughout the evolution is of great importance to ensure resistance against differential and linear cryptanalysis(DC/LC). It is easy to see that the number of nonzero nibbles per round is repeated with period 4 and their sum up to round 8 is at least 32.

3.2 S-Boxes Construction and Their Property

The maximum characteristic and linear approximation probabilities for an $n \times n$ S-box S (δ_S and λ_S for short) can be defined as follows. Let X and Y be the set of all possible 2^n inputs/outputs of S, respectively. Then δ_S and λ_S are defined by

$$\delta_S \stackrel{\text{def}}{=} \max_{\Delta x \neq 0, \Delta y} \frac{\#\{x \in X | S(x) \oplus S(x \oplus \Delta x) = \Delta y\}}{2^n},$$

$$\lambda_S \stackrel{\text{def}}{=} \max_{\Gamma x, \Gamma y \neq 0} \left(\frac{|\#\{x \in X | x \bullet \Gamma x = S(x) \bullet \Gamma y\} - 2^{n-1}|}{2^{n-1}} \right)^2.$$

The nonlinear transformation adopted in mCrypton is substitution using four 4×4 S-boxes, S_i $(i = 0, 1, 2, 3)$ such that $S_0^{-1} = S_2$ and $S_1^{-1} = S_3$. These 4-bit S-boxes were searched for over some limited space of good 4-bit permutations produced by field inversion and affine transformation in $\mathrm{GF}(2^4)$ (actually in $\mathrm{GF}((2^2)^2)$, i.e., $x \to ax^{-1} + b$, $a, b \in \mathrm{GF}((2^2)^2)$. The main selection criteria is that the number of high-probability difference pairs (selection patterns, resp.) in the resulting S-boxes should be as small as possible when the input is restricted to the minimal diffusion set Ω_y. This is to ensure that high-probability differences/selection patterns should be more rapidly diffused by linear transformations and that it should be more difficult to form a chain of high-probability

Table 1. The selected 4×4 S-boxes

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S_0	4	15	3	8	13	10	12	0	11	5	7	14	2	6	1	9
S_1	1	12	7	10	6	13	5	3	15	11	2	0	8	4	9	14
S_2	7	14	12	2	0	9	13	10	3	15	5	8	6	4	11	1
S_3	11	0	10	7	13	6	4	2	12	14	3	9	1	5	15	8

S-box characteristics/linear approximations through consecutive rounds. Table 1 shows the four 4-bit S-boxes selected.

The inversion function in GF(2^4) is well-known to be differentially 4-uniform and have the nonlinearity of 2^{-1}, so the characteristic/linear probabilities of the S-boxes are limited to at most $\delta_{S_i} = \frac{4}{16} = 2^{-2}$ and $\lambda_{S_i} = (\frac{4}{8})^2 = 2^{-2}$. More importantly, if the input is restricted to the minimum diffusion set Ω_y, the maximum entry value in DC/LC tables is at most 2. So, for the best-case analysis of DC and LC, we define these probabilities as

$$p_d \stackrel{\text{def}}{=} \delta_{S_i}^{\Omega_y} = \frac{2}{16} = 2^{-3}, \quad p_l \stackrel{\text{def}}{=} \lambda_{S_i}^{\Omega_y} = (\frac{2}{8})^2 = 2^{-4}.$$

3.3 Differential/Linear Cryptanalysis

The complexity of DC and LC is completely determined by the number of active S-boxes involved and their characteristic/linear approximation probabilities. Since the number of active S-boxes involved in any 8-round characteristic/linear approximation is at least 32, we can obtain the most rough upper bound for the best 8-round characteristic/linear approximation probability as $(2^{-2})^{32} = 2^{-64}$ without details of the S-box characteristic/nonlinear properties.

However, the minimum number of active S-boxes can be obtained only for the difference pairs/selection patterns in the minimum diffusion set and the best S-box characteristic/linear approximation probabilities that can be achieved for the values in the minimum diffusion set in our selected S-boxes are at most $p_d = 2^{-3}$ and $p_l = 2^{-4}$. Since it is reasonable to assume that a characteristic/linear approximation involving a smaller number of active S-boxes with smaller S-box characteristic/linear approximation probabilities should give better overall probability than a characteristic/linear approximation involving a larger number of active S-boxes with larger probabilities. Therefore, we can obtain a tighter bound for the 8-round characteristic/linear approximation probabilities as

$$p_{C_8} \leq (p_d)^{32} = 2^{-96}, \quad p_{L_8} \leq (p_l)^{32} = 2^{-128}.$$

Actually we can find such characteristics (no linear approximation, however) by careful examination of DC/LC tables together with minimum diffusion patterns. Note however that the probability of 2^{-64} is the threshold for applicability of DC/LC since the number of all possible difference pairs/selection patterns cannot exceed 2^{64} in 64-bit block ciphers. There also exist a number of variants or generalizations of differential and linear cryptanalysis, but theses attacks are unlikely to much reduce the attack complexity. We thus strongly believe that 12-round mCrypton is far secure against differential/linear cryptanalysis.

We should also consider a variety of other cryptanalysis techniques for the security of mCrypton, such as algebraic attacks, related key attacks and key schedule cryptanalysis, etc. We believe that these attacks are equally unlikely for 12-round mCrypton as in the case of Crypton (see [7] for further discussion on the applicability of these attacks to mCrypton).

4 Implementation Efficiency

4.1 Software Efficiency

The overall structure of mCrypton allows a very high degree of parallelism. This results in high efficiency and flexibility in both software and hardware implementations. The encryption round of mCrypton can be efficiently implemented using lookup tables by precomputing and storing 4 tables, each containing sixteen 16-bit words, such that for $0 \leq j \leq 16$,

$$\begin{aligned}
SS_0[j] &= S_0[j] \wedge m_0 \parallel S_0[j] \wedge m_1 \parallel S_0[j] \wedge m_2 \parallel S_0[j] \wedge m_3, \\
SS_1[j] &= S_1[j] \wedge m_1 \parallel S_1[j] \wedge m_2 \parallel S_1[j] \wedge m_3 \parallel S_1[j] \wedge m_0, \\
SS_2[j] &= S_2[j] \wedge m_2 \parallel S_2[j] \wedge m_3 \parallel S_2[j] \wedge m_0 \parallel S_2[j] \wedge m_1, \\
SS_3[j] &= S_3[j] \wedge m_3 \parallel S_3[j] \wedge m_0 \parallel S_3[j] \wedge m_1 \parallel S_3[j] \wedge m_2,
\end{aligned}$$

where $\parallel$ denotes concatenation of bit strings. These four extended S-boxes altogether take a storage of only 128 bytes, small enough to be used even in very limited computing environments such as mote-class sensor nodes. With these lookup tables, we can implement one round of mCrypton only using 20 table lookups (16 to SS tables and 4 to round key tables).

Note that for decryption we need 8-bit rotated versions of the above extended SS-boxes and we also need the original S-boxes for key scheduling. This will not be any problem in most computing environments since the storage requirement is still at most 320 bytes altogether. Further, there may be no need of storing rotated versions of SS tables in more resource-constrained 8-bit processors, since the same SS tables can be used for decryption as well by referring to the second byte of the table entry first. The four S-boxes may also be stored more compactly only using 32 bytes of storage if desirable. So in this minimal setting we only need 160 bytes of storage for four SS-boxes and four S-boxes.

We also need to consider the key scheduling overhead in software implementations. Real-time computation of round keys for every block of encryption/decryption should be the last choice even in the resource-constrained devices since its computational overhead is never negligible. mCrypton requires two set of 52 round keys of 16 bits for both encryption and decryption, corresponding to a storage of 208 bytes. This amount of temporary storage (RAM) will not be much overhead even in typical sensor nodes such as Mica2 motes. Therefore, we can see that mCrypton can be very efficiently implemented even in the very restricted 8-bit computing environments. Furthermore, mCrypton will be particularly efficient on 16-bit platforms such as Telos motes, since most operations are performed over 16-bit words.

4.2 Hardware Efficiency

Efficiency in low-cost hardware implementation is one of main design objectives of mCrypton. Each component function is carefully designed with hardware implementations in mind. To check the hardware complexity of mCrypton, a simple, straightforward hardware was designed and simulated. The processor is based

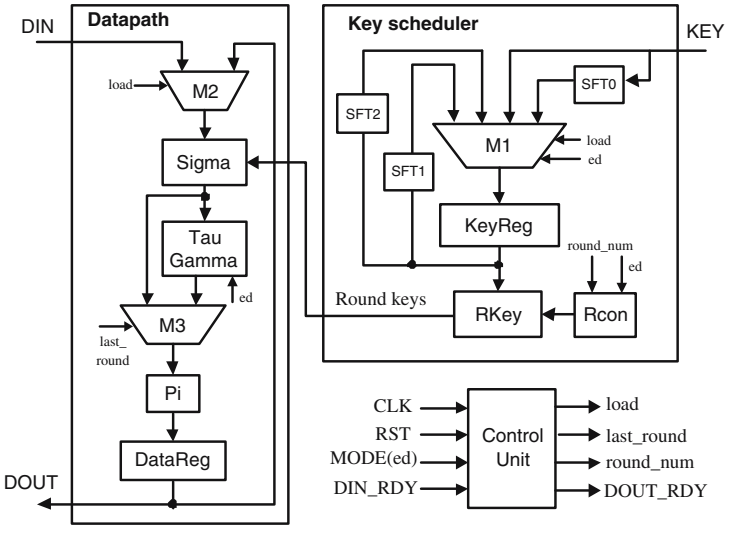

Fig. 4. Hardware architecture of mCrypton processor

on the implementation of a single round per clock cycle and transforms input data in 13 clocks as depicted in Fig.4.

The circuit can start processing as soon as key and data are available on their input pins (loaded in parallel), and execute both encryption and decryption in 13 rounds, where each round consists of $\pi \circ \gamma \circ \tau \circ \sigma$, except for the last round, which corresponds to $\pi \circ \sigma$. Here note that for simplicity we process data and key internally in column basis, instead of in row basis described in the specification. Encryption and decryption rounds share the same datapath logic. Note that encryption and decryption rounds make use of just different arrangement of the same S-boxes. Therefore, γ and γ^{-1} can be implemented using a single set of 16 S-boxes and a pair of appropriate selectors (multiplexers). This is actually achieved in the γ transformation in Fig.4.

Key scheduler logic generates round keys from a given user key (64, 96, or 128 bits) and supplies them to the datapath for both encryption and decryption. The initial secret key for encryption (decryption, resp.) is first loaded into the key register KeyReg from which round keys are generated by the RKey component, where round constants $C[r]$'s are generated by the Rcon component. The key register is then updated for next round key generation through specified rotations SFT1 (SFT2 for decryption, resp.).

The architecture shown in Fig.4 has been implemented for each key size using $0.13\mu m$ CMOS technology. The encryption-only mode is also implemented as well as the full (encryption and decryption) mode, since encryption capability is often sufficient for security in more resource-constrained low-cost RFID tags. The resulting gate counts (1 gate = 2-input NAND gate-equivalent) are summarized in Table 2. As can be seen from the table, elimination of decryption components

Table 2. Hardware complexity (number of gates) of the mCrypton processor

Mode	Encryption & decryption			Encryption-only		
Key size	64 bits	96 bits	128 bits	64 bits	96 bits	128 bits
Key scheduler	1338	1649	1952	736	992	1249
KeyReg	320	480	640	320	480	640
Rcon	52	52	52	21	21	21
ϕ function	288	288	288	0	0	0
S-box	107	107	107	107	107	107
Other logic	571	722	865	288	384	481
Round func.	2020	2020	2020	1588	1588	1588
DataReg	320	320	320	320	320	320
γ function	880	880	880	448	448	488
π function	288	288	288	288	288	288
Key xor(σ)	192	192	192	192	192	192
Other logic	340	340	340	340	340	340
Control unit	61	61	61	61	61	61
Routing	54	59	75	35	40	51
Total	3473	3789	4108	2420	2681	2949

greatly (more than 25%) reduces the overall complexity. We can see that the full mode consumes about 3.5K to 4.1Kgates while the encryption-only mode about 2.4K to 3.0Kgates, depending on key sizes. The gate count for encryption-only modes appears to be well within an economic range of 5-cent RFID tags.

The critical path delay (CPL) of our implemented architecture can be relatively long, since it traverses from round key generation to round function evaluation. However, it turned out that the maximum CPL was still less than 9 ns (allowing frequency over 100MHz) even for the full mode of 128-bit key version. We did not much concentrate on speed issues during our implementation, since operating frequencies in our target applications are extremely low: Most modern UHF RFID chips use on-board oscillators with frequencies over 1MHz and most mote-class sensor nodes operate at frequencies below 10MHz. Nevertheless, if

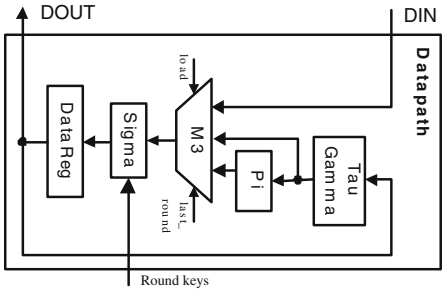

Fig. 5. Alternative datapath architecture for shorter delay

higher performances are preferred, the architecture may be modified as shown in Fig.5. This architecture may reduce the CPL to almost a half of the original but may somewhat increase the gate count in the case of encryption-only mode, mainly due to the added complexity of the last round key conversion.

To further reduce the hardware complexity, we may adopt multiple clock cycles/round architectures. At the minimum we may process input data in column-by-column basis, based on the 5 cycles/round architecture. In this case the core datapath logic can only implement four S-boxes, one π_i transformation and 16 XORs. We are now developing such an architecture for more compact implementation. Our preliminary analysis based on the this architecture promises that the overall hardware complexity can be considerably reduced (by about 30%). For example, the encryption-only mode with 128-bit keys may be implemented with about 2000 gates and the full mode with about 2500 gates.

Finally, we note that we did not consider any specific power consumption minimization during the implementation. Clearly we would have to sacrifice the circuit size more or less to apply power consumption minimization strategies (e.g., see [8]). One way to reduce power consumption in the present architecture would be to reduce the operating frequency (say, far below 100KHz), as far as it satisfies the minimum response time required by standards such as EPC Gen2 and ISO/IEC 18000-6 (e.g., see [3]).

5 Conclusion

We presented a 64-bit block cipher mCrypton specifically designed for security in resource-constrained applications, such as low-cost RFID tags and sensors, and analyzed its security and efficiency. mCrypton is based on the proven architecture of Crypton with some improvements in hardware and software efficiency under restricted environments. It also incorporates a flexible key schedule with key sizes of 64 bits, 96 bits and 128 bits, which may provide greater flexibility in cost-security tradeoffs often encountered in cost-driven applications such as low-cost RFID tags. Our preliminary security analysis shows that mCrypton is far secure against well-known attacks on block ciphers such as differential and linear cryptanalysis. We also demonstrated through hardware simulation that mCrypton is well-suited for our target applications. Our simple hardware design shows that it can be implemented with the complexity of about 2.4K to 4.1K gates, depending on key sizes and capabilities (encryption-only, encryption and decryption). Furthermore, we expect that a more compact 5 cycles/round architecture under development could considerably reduce the complexity (by about 30%). As another possible future work, we could perform validation of software efficiency on the 8/16-bit processors used in typical sensor nodes.

References

1. S.Bono, M.Green, A. Stubblefield, A.Juels, A.Rubin and M.Szydlo, Security analysis of a cryptographically-enabled RFID device, In *14th USENIX Security Symposium*, Baltimore, Maryland, July-August 2005.

2. R.Campbell, J.A.-Muhtadi, P.Naldurg, G.Sampemane1 and M.D.Mickunas, Towards Security and Privacy for Pervasive Computing, In *Software Security – Theories and Systems*, LNCS 2609, Springer-Verlag, 2003, p.1-15.
3. M.Feldhofer, S.Dominikus and J.Wolkerstorfer, Strong authentication for RFID systems using the AES algorithm, In *Cryptographic Hardware and Embedded Systems - CHES 2004*, LNCS 3156, Springer-Verlag, 2004, pp.357-370.
4. S.L.Garfinkel, A.Jeuls and R.Pappu, RFID privacy: An overview of problems and proposed solutions, *IEEE Security & Privacy*, May/June 2005, pp.34-43.
5. C.Karlof, N.Sastary and D.Wagner, TinySec: A link layer security architecture for wireless sensor networks, In *ACM SenSys 2004*, Nov. 3-5, 2004.
6. Y.W.Law and J.M.Doumen and P.H.Hartel, Benchmarking block ciphers for wireless sensor networks (Extended abstract), In *1st IEEE Int. Conf. on Mobile Ad-hoc and Sensor Systems(MASS)*, Fort Lauderdale, Florida, Oct. 2004.
7. C.H.Lim, A revised version of CRYPTON: CRYPTON v1.0, In *Fast Software Encryption-FSE'99*, LNCS 1636, Spinger-Verlag, 1999, pp.31-45
8. S.Morioka and A.Satoh, An Optimized S-Box Circuit Architecture for Low Power AES Design, In *Cryptographic Hardware and Embedded Systems - CHES 2002*, LNCS 2523, Springer-Verlag, 2003, pp.172-186.
9. A.Perrig, J.Stankovic and D.Wagner, Security in wireless sensor networks, *Commun. of ACM*, 47(5), June 2004, pp.53-57.
10. J.Polastre, R.Szewczyk and D.Culler, Telos: enabling ultra-low power wireless research, In *Proceedings of the 4th Int. Conf. on Information Processing in Sensor Networks: Special track on Platform Tools and Design Methods for Network Embedded Sensors (IPSN/SPOTS)*, April 25-27, 2005.
11. F.Stajano and R.Anderson, The Resurrecting Duckling: Security Issues for Ubiquitous Computing, *IEEE Security & Privacy*, April 2002, pp.22-26.
12. S.A.Weis, S.E.Sarma, R.L.Rivest and D.W.Engels, Security and privacy aspects of low-cost radio frequency identification systems, *Int. Conference on Security in Pervasive Computing - SPC 2003*, LNCS 2802, Springer-Verlag, 2003, pp.454-469.