Hummingbird: Ultra-Lightweight Cryptography for Resource-Constrained Devices

Daniel Engels[2], Xinxin Fan[1], Guang Gong[1],
Honggang Hu[1], and Eric M. Smith[2]

[1] Department of Electrical and Computer Engineering
University of Waterloo
Waterloo, Ontario, N2L 3G1, Canada
{x5fan,h7hu,ggong}@uwaterloo.ca
[2] Revere Security Corporation
4500 Westgrove Drive, Suite 335, Addison, TX 75001, USA
eric.smith@reveresecurity.com

Abstract. Due to the tight cost and constrained resources of high-volume consumer devices such as RFID tags, smart cards and wireless sensor nodes, it is desirable to employ lightweight and specialized cryptographic primitives for many security applications. Motivated by the design of the well-known Enigma machine, we present a novel ultra-lightweight cryptographic algorithm, referred to as Hummingbird, for resource-constrained devices in this paper. Hummingbird can provide the designed security with small block size and is resistant to the most common attacks such as linear and differential cryptanalysis. Furthermore, we also present efficient software implementation of Hummingbird on the 8-bit microcontroller ATmega128L from Atmel and the 16-bit microcontroller MSP430 from Texas Instruments, respectively. Our experimental results show that after a system initialization phase Hummingbird can achieve up to 147 and 4.7 times faster throughput for a size-optimized and a speed-optimized implementations, respectively, when compared to the state-of-the-art ultra-lightweight block cipher PRESENT [10] on the similar platforms.

Keywords: Constrained devices, lightweight cryptographic primitive, security analysis, efficient implementation.

1 Introduction

With the advent of pervasive computing, various smart devices such as RFID tags, smart cards, and wireless sensor nodes are penetrating into and impacting people's life at a staggering rate and in significant ways. Their applications range from access control and supply-chain management to home automation and healthcare. Since a multitude of applications involve processing of sensitive personal information like health or biomedical data, the increasing demand for integrating cryptographic functions into embedded applications has risen. However, these pervasive smart devices usually have extremely constrained resources

R. Sion et al. (Eds.): FC 2010 Workshops, LNCS 6054, pp. 3–18, 2010.

in terms of computational capabilities, memory, and power supply. Hence, classical cryptographic primitives designed for full-fledged computers might not be suited for resource-constrained smart devices. For instance, the popular 1024-bit RSA algorithm cannot be implemented on RFID tags due to their harsh constrains with respect to gate count and power consumption. Moreover, the tight cost constrains inherent in mass deployments of smart devices also bring forward impending requirements for designing new cryptographic primitives that can perform strong authentication and encryption, and provide other security functionalities for ultralow-power applications in the era of pervasive computing. This emerging research area is usually referred to as *lightweight cryptography*.

The key issue of designing lightweight cryptographic algorithms is to deal with the trade-off among *security*, *cost*, and *performance* [29]. A host of lightweight cryptographic primitives that particularly target resource-constrained smart devices have been published in the past few years and we will focus on lightweight symmetric ciphers in this paper. All the previous proposals can be roughly divided into the following three categories. The first category consists of highly optimized and compact hardware implementations for standardized block ciphers such as AES [17,18,19] and IDEA [23], whereas the proposals in the second category involve slight modifications of a classical block cipher like DES [26] for lightweight applications. Finally, the third category features new low-cost designs, including lightweight block ciphers HIGHT [21], mCrypton [22], SEA [32], PRESENT [10] and KATAN and KTANTAN [11], as well as lightweight stream ciphers Grain [20] and Trivium [12]. Moreover, the design and implementation of lightweight asymmetric ciphers is also an ongoing research direction and a good survey about lightweight cryptography implementations can be found in [14].

Motivated by the design of the well-known Enigma machine, we present a novel ultra-lightweight cryptographic algorithm in this paper, referred to as Hummingbird[1], which is originally designed by Engels, Schweitzer and Smith, for resource-constrained devices like RFID tags and wireless sensor nodes. Hummingbird has a *hybrid structure* of block cipher and stream cipher and was developed with both lightweight software and lightweight hardware implementations for constrained devices in mind. The hybrid model can provide the designed security with small block size and is therefore expected to meet the stringent response time and power consumption requirements for a large variety of embedded applications. Moreover, we also implement Hummingbird on the 8-bit microcontroller ATmega128L from Atmel and the 16-bit microcontroller MSP430 from Texas Instrument (TI), which are the most popular processors used in wireless sensor network platforms because of their low power design, multiple sensor interfaces, and widely available development tools[2]. Our experimental results show that after a system initialization phase Hummingbird can achieve up to 147 and 4.7 times faster throughput than that of the ultra-lightweight block cipher PRESENT [10] for a size-optimized and a speed-optimized implementations,

[1] Hummingbird algorithm is first reported in [15] as a technical report of Center for Applied Cryptographic Research (CACR), University of Waterloo.

[2] The implementation of Hummingbird on a 4-bit microcontroller can be found in [16].

respectively. Due to space limitations, the compact hardware implementation of Hummingbird will be discussed in a separate work.

This paper is organized as follows. Section 2 presents the specification and the design rationale of Hummingbird. In Section 3, we give the security analysis of Hummingbird against common attacks such as differential and linear cryptanalysis. Section 4 treats efficient software implementation of Hummingbird across a range of wireless sensor network processors. Finally, Section 5 concludes this contribution.

2 The Hummingbird Cryptographic Algorithm

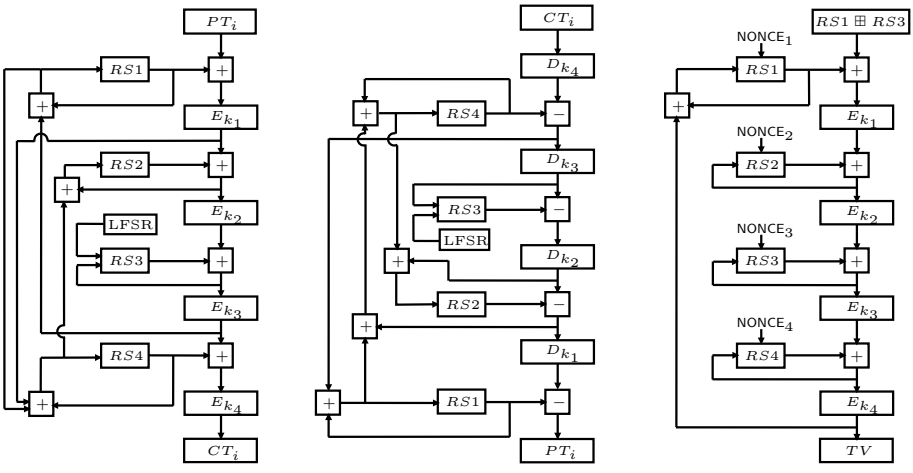
Different from existing (ultra-)lightweight cryptographic primitives which are either block ciphers or stream ciphers, Hummingbird is an elegant combination of the above two cipher structures with 16-bit block size, 256-bit key size, and 80-bit internal state. The size of the key and the internal state of Hummingbird provide a security level which is adequate for many embedded applications. For clarity, we use the notation listed in Table 1 in the algorithm description. A top-level structure of the Hummingbird cryptographic algorithm is shown in Figure 1.

Table 1. Notation

PT_i	the i-th plaintext block, $i = 1, 2, \ldots, n$
CT_i	the i-th ciphertext block, $i = 1, 2, \ldots, n$
K	the 256-bit secret key
$\mathbf{E}_K(\cdot)$	the encryption function of Hummingbird with 256-bit secret key K
$\mathbf{D}_K(\cdot)$	the decryption function of Hummingbird with 256-bit secret key K
k_i	the 64-bit subkey used in the i-th block cipher, $i = 1, 2, 3, 4$, such that $K = k_1 \| k_2 \| k_3 \| k_4$
$E_{k_i}(\cdot)$	a block cipher encryption algorithm with 16-bit input, 64-bit key k_i, and 16-bit output, i.e., $E_{k_i} : \{0,1\}^{16} \times \{0,1\}^{64} \rightarrow \{0,1\}^{16}, i = 1, 2, 3, 4$
$D_{k_i}(\cdot)$	a block cipher decryption algorithm with 16-bit input, 64-bit key k_i, and 16-bit output, i.e., $D_{k_i} : \{0,1\}^{16} \times \{0,1\}^{64} \rightarrow \{0,1\}^{16}, i = 1, 2, 3, 4$
RSi	the i-th 16-bit internal state register, $i = 1, 2, 3, 4$
LFSR	a 16-stage Linear Feedback Shift Register with the characteristic polynomial $f(x) = x^{16} + x^{15} + x^{12} + x^{10} + x^7 + x^3 + 1$
$\boxplus$	modulo 2^{16} addition operator
$\boxminus$	modulo 2^{16} subtraction operator
$\oplus$	exclusive-or (XOR) operator
$m \lll l$	left circular shift operator, which rotates all bits of m to the left by l bits, as if the left and the right ends of m were joined.
$K_j^{(i)}$	the j-th 16-bit key used in the i-th block cipher, $j = 1, 2, 3, 4$, such that $k_i = K_1^{(i)} \| K_2^{(i)} \| K_3^{(i)} \| K_4^{(i)}$
S_i	the i-th 4-bit to 4-bit S-box used in the block cipher, $S_i : \mathbb{F}_2^4 \rightarrow \mathbb{F}_2^4, i = 1, 2, 3, 4$
NONCE_i	the i-th nonce which is a 16-bit random number, $i = 1, 2, 3, 4$
IV	the 64-bit initial vector, such that $IV = \mathsf{NONCE}_1 \| \mathsf{NONCE}_2 \| \mathsf{NONCE}_3 \| \mathsf{NONCE}_4$

2.1 Encryption/Decryption and Initialization

The overall structure of the Hummingbird encryption algorithm (see Figure 1(a)) consists of four 16-bit block ciphers $E_{k_1}, E_{k_2}, E_{k_3}$ and E_{k_4}, four 16-bit internal



(a) Encryption Process (b) Decryption Process (c) Initialization Process

Fig. 1. A Top-Level Description of the Hummingbird Cryptographic Algorithm

state registers $RS1, RS2, RS3$ and $RS4$, and a 16-stage LFSR. The 256-bit secret key K is divided into four 64-bit subkeys k_1, k_2, k_3 and k_4 which are used in the four block ciphers, respectively. A 16-bit plaintext block PT_i is encrypted by first executing a modulo 2^{16} addition of PT_i and the content of the first internal state register $RS1$. The result of the addition is then encrypted by the first block cipher E_{k_1}. This procedure is repeated in a similar manner for another three times and the output of E_{k_4} is the corresponding ciphertext CT_i. Furthermore, the states of the four internal state registers will also be updated in an unpredictable way based on their current states, the outputs of the first three block ciphers, and the state of the LFSR. The decryption process (see Figure 1(b)) follows the similar pattern as the encryption. When using Hummingbird in practice, four 16-bit random nonce NONCE_i are first chosen to initialize the four internal state registers RSi $(i = 1, 2, 3, 4)$, respectively, followed by four consecutive encryptions on the message $RS1 \boxplus RS3$ by Hummingbird running in the initialization mode (see Figure 1(c)). The final 16-bit ciphertext TV is used to initialize the LFSR. Moreover, the 13^{th} bit of the LFSR is always set to prevent a zero register. The LFSR is also stepped once before it is used to update the internal state register $RS3$. The exact encryption/decryption and initialization procedure as well as the internal state updating of Hummingbird are illustrated in Table 2.

2.2 16-Bit Block Cipher

Four identical 16-bit block ciphers are employed in a consecutive manner in the Hummingbird encryption scheme. The 16-bit block cipher is a typical substitution-permutation (SP) network with 16-bit block size and 64-bit key as shown in

Table 2. Encryption/Decryption and Initialization of Hummingbird

Encryption Process	Decryption Process	Initialization Process (Four Rounds Encryption)
$V12_t = E_{k_1}(PT_i \boxplus RS1_t)$	$V34_t = D_{k_4}(CT_i) \boxminus RS4_t$	$V12_t = E_{k_1}((RS1_t \boxplus RS3_t) \boxplus RS1_t)$
$V23_t = E_{k_2}(V12_t \boxplus RS2_t)$	$V23_t = D_{k_3}(V34_t) \boxminus RS3_t$	$V23_t = E_{k_2}(V12_t \boxplus RS2_t)$
$V34_t = E_{k_3}(V23_t \boxplus RS3_t)$	$V12_t = D_{k_2}(V23_t) \boxminus RS2_t$	$V34_t = E_{k_3}(V23_t \boxplus RS3_t)$
$CT_i = E_{k_4}(V34_t \boxplus RS4_t)$	$PT_i = D_{k_1}(V12_t) \boxminus RS1_t$	$TV = E_{k_4}(V34_t \boxplus RS4_t)$
Internal State Updating		
$\mathrm{LFSR}_{t+1} \leftarrow \mathrm{LFSR}_t$		
$RS1_{t+1} = RS1_t \boxplus V34_t$		$RS1_{t+1} = RS1_t \boxplus TV_t$
$RS3_{t+1} = RS3_t \boxplus V23_t \boxplus \mathrm{LFSR}_{t+1}$		$RS2_{t+1} = RS2_t \boxplus V12_t$
$RS4_{t+1} = RS4_t \boxplus V12_t \boxplus RS1_{t+1}$		$RS3_{t+1} = RS3_t \boxplus V23_t$
$RS2_{t+1} = RS2_t \boxplus V12_t \boxplus RS4_{t+1}$		$RS4_{t+1} = RS4_t \boxplus V34_t$

Figure 2. It consists of four regular rounds and a final round that only includes the key mixing and the S-box substitution steps. The 64-bit subkey k_i is split into four 16-bit round keys $K_1^{(i)}, K_2^{(i)}, K_3^{(i)}$ and $K_4^{(i)}$ which are used in the four regular rounds, respectively. Moreover, the final round utilizes two keys $K_5^{(i)}$ and $K_6^{(i)}$ directly derived from the four round keys (see Figure 2). Like any other SP network, one regular round comprises of three stages: a key mixing step, a substitution layer, and a permutation layer. For the key mixing, a simple exclusive-OR operation is used in this 16-bit block cipher for efficient implementation in both software and hardware. The substitution layer is composed of 4 Serpent-type S-boxes [1] with 4-bit inputs and 4-bit outputs, having additional properties whose selecting criteria is described in the appendix of [15]. According to the nine criteria presented in [15], we select four S-boxes, the action of which in hexadecimal notation is described in Figure 2. The permutation layer in this 16-bit block cipher is given by the linear transform $L : \{0,1\}^{16} \rightarrow \{0,1\}^{16}$ defined as follows:

$$L(m) = m \oplus (m \lll 6) \oplus (m \lll 10),$$

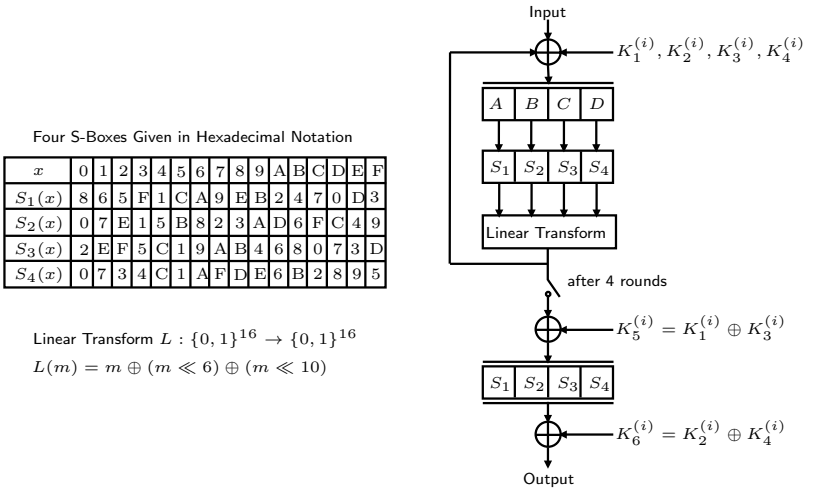
where $m = (m_0, m_1, \cdots, m_{15})$ is a 16-bit data block.

Remark 1. To further reduce the consumption of the memory, area and power of Hummingbird in both software and hardware implementations, four S-boxes used in Hummingbird can be replaced by a single S-box, which is repeated four times in the 16-bit block cipher. The compact version of Hummingbird can achieve the same security level as the original Hummingbird and will be implemented on wireless sensor nodes in this paper.

2.3 Design Rationale of Hummingbird

The design of the Hummingbird cryptographic algorithm is motivated by the well-known Enigma machine[3] and takes into account both security and effi-

[3] In Enigma machine each rotor has 26 contacts, whereas in Hummingbird each virtual rotor (i.e., a 16-bit block cipher) has $2^{16} = 65536$ contacts.



x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S_1(x)$	8	6	5	F	1	C	A	9	E	B	2	4	7	0	D	3
$S_2(x)$	0	7	E	1	5	B	8	2	3	A	D	6	F	C	4	9
$S_3(x)$	2	E	F	5	C	1	9	A	B	4	6	8	0	7	3	D
$S_4(x)$	0	7	3	4	C	1	A	F	D	E	6	B	2	8	9	5

Fig. 2. The Structure of the Block Cipher in the Hummingbird Cryptographic Algorithm

ciency simultaneously. The encryption/decryption process of Hummingbird can be viewed as the continuous running of a rotor machine, where four small block ciphers act as four virtual rotors which perform permutations on 16-bit words. The salient characteristics of Hummingbird lies in implementing extraordinarily large virtual rotors with custom block ciphers and using successively changing internal states to step each virtual rotor in various and unpredictable ways. Besides a novel cipher structure, Hummingbird is also designed to protect against the most common attacks such as linear and differential cryptanalysis, which will be discussed in detail in Section 3. Moreover, extremely simple arithmetic and logic operations are extensively employed in Hummingbird, which make it well-suited for resource-constrained environments.

3 Security Analysis of the Hummingbird Cryptographic Algorithm

In this section, we analyze the security of the Hummingbird cryptographic algorithm by showing that it is resistant to the most common attacks to block ciphers and stream ciphers including birthday attack, differential and linear cryptanalysis, etc. Note that Hummingbird has a hybrid mode of block cipher and stream cipher (This is the reason that the analysis in [10] can not be employed directly here.), which can be considered as a finite state machine with the internal state $(RS1, RS2, RS3, RS4, \text{LFSR})$. However, the value of LFSR does not depend on those of $RS1$, $RS2$, $RS3$, and $RS4$. The purpose of using the LFSR is to guarantee the period of the internal state is at least 2^{16}.

A. Birthday Attack on the Initialization. For a fixed key, one may want to find two identical internal states $(RS1, RS2, RS3, RS4, \text{LFSR})$ initialized by two different IVs using the birthday attack. However, if we fix the key in the initialization procedure of the Hummingbird encryption scheme, the mapping $(RS1_t, RS2_t, RS3_t, RS4_t) \rightarrow (RS1_{t+1}, RS2_{t+1}, RS3_{t+1}, RS4_{t+1})$ is one-to-one. Hence the birthday attack does not work in this case.

B. Differential Cryptanalysis. Let $\mathbf{E}_K(x)$ denote the encryption function of Hummingbird with 256-bit key K. Recall that $E_k(x)$, defined in Section 2, denotes the 16-bit block cipher encryption used in Hummingbird with 64-bit key k. Then $\mathbf{E}_K(x)$ is the composition of four $E_k(x)$. For a function $F(x)$ from $\mathbb{F}_2^m$ to $\mathbb{F}_2^m$, the differential between $F(x)$ and $F(x+a)$, where $+$ is the bit-wise addition, is denoted by $D_F(a,b)$ and defined as follows.

$$D_F(a,b) = |\{x \,|\, F(x) + F(x+a) = b,\, x \in \mathbb{F}_2^m\}|.$$

For many keys, we have computed the differentials of both $\mathbf{E}_K(x)$ and $E_k(x)$. Note that from Section 2, we know that there are five rounds in $E_k(x)$. We list the differential $D_{E_k}(a,b)$ for each round in the following Table 3 (a). For substantially large amount of initial vectors IV and keys K, the differentials for both $E_k(x)$ and $\mathbf{E}_K(x)$ satisfy the following inequalities.

$$\max_{a,b \in \mathbb{F}_2^{16}, a \neq 0} \{D_{E_k}(a,b)\} \leq 20, \text{ and } \max_{a,b \in \mathbb{F}_2^{16}, a \neq 0} \{D_{\mathbf{E}_K}(a,b)\} \leq 20.$$

In other words, the differential of $\mathbf{E}_K(x)$ has the same upper bound as $E_k(x)$, the block cipher component in $\mathbf{E}_K$. We also tested the reduced version of Hummingbird for more instances of different pairs of (IV, K). From those experimental results, in general, the standard differential cryptanalysis method is not applicable to Hummingbird with practical time complexity.

C. Linear Cryptanalysis. For the linear cryptanalysis of $\mathbf{E}_K(x)$, we need to consider $|\hat{\mathbf{E}}_K(a,b)|$, the absolute value of the Walsh transform of $\mathbf{E}_K(x)$, where

$$\hat{\mathbf{E}}_K(a,b) = \sum_{x \in \mathbb{F}_2^{16}} (-1)^{\langle a, \mathbf{E}_K(x) \rangle + \langle b, x \rangle},\ a, b \in \mathbb{F}_2^{16}, a \neq 0$$

Table 3. Differential and Linear Properties of the 16-bit Block Cipher

(a) Differential Properties

# of Rounds	$\max_{a \neq 0, b} D_{E_k}(a,b)$
0	16384
1	1024
2	98
3	20
4	20

(b) Linear Properties

$wt(a)$	$wt(b)$	Constant c
1	1	4.703125
1	2	4.359375
1	3	4.500000
2	1	4.390625
2	2	4.281250
2	3	4.828125
3	1	4.968750
3	2	4.718750
3	3	4.781250

where $\langle x, y\rangle$ is the inner product of two binary vectors x and y (see Appendix for the detail). Unlike the case for the differential of $\mathbf{E}_K(x)$ or $E_k(x)$, we cannot perform an exhaustive computation for $|\hat{\mathbf{E}}_K(a,b)|$ for all $a, b \in \mathbb{F}_2^{16}, a \neq 0$, since there are around 2^{48} instances for (a, b, x) that need to be verified for a pair of fixed IV and key K. For some fixed pairs of (IV, K), we have computed random subsets of (a, b) with size around 2^{20}. Those experimental results show that $|\hat{\mathbf{E}}_K(a,b)| \leq c \cdot \sqrt{2^{16}}$, where $c \leq 4.96875$. We list some data in Table 3 (b). We also conducted the experiments for an 8-bit version of the Hummingbird encryption scheme which means that all the rotors $RS_i, i = 1, 2, 3, 4$ and LFSR contain only 8 bits. The Walsh transform of this reduced version of Hummingbird is bounded by $5 \cdot \sqrt{2^8}$ for many pairs of IV and key. This is the supporting evidence (albeit weak) that the absolute value of the Walsh transform of Hummingbird encryption function could be bounded by the square root of 2^{16} multiplying by a constant. Hence, Hummingbird seems to be resistant to linear cryptanalysis attack with practical time complexity.

D. Structural Attack. The Hummingbird encryption scheme may be viewed as a certain operation mode of a block cipher. For example, the ciphertext can be viewed as the internal state of a block cipher in CBC mode. In [4,5], Biham investigated some operation modes of block ciphers. He found that many triple modes are not as secure as one expected. In [6], Biham and Knudsen broke the ANSI X9.52 CBCM Mode. However, the internal state transition in Hummingbird encryption scheme is much more complicated than those studied by [4,5,6]. Hence, those attacks cannot be simply applied to the Hummingbird encryption scheme. In [33], by choosing IV, Wagner presented some new attacks on some modes proposed by Biham. Because IV initialization is used in the Hummingbird encryption scheme, Wagner's attacks are not applicable.

E. Algebraic Attack. For the Hummingbird encryption scheme, the degree of each S-box in the block cipher $E_k(x)$ is maximized. Moreover, each block cipher $E_k(x)$ consists of five rounds. Thus, there are totally 20 rounds for the Hummingbird encryption scheme, i.e., $\mathbf{E}_K(x)$. Furthermore, the internal state transition involves modulo 2^{16} operation. Hence it is hard to apply efficient linearization techniques for algebraic attacks to Hummingbird.

F. Cube Attack. The success probability of cube attack is high if the degree of the internal state transition function in a stream cipher is low. For example, the degree of internal state transition function of Trivium grows slowly [13]. However, for the Hummingbird encryption scheme, the degree of the internal state transition function is very high. In addition, Hummingbird encryption scheme has a hybrid mode of block cipher and stream cipher. We have tested both the 16-bit block cipher $E_k(x)$ used in the Hummingbird encryption scheme and the Hummingbird encryption function $\mathbf{E}_K(x)$. We note that no linear equations of key bits can be used in the way as suggested in [13].

G. Slide and Related-Key Attack. Both slide attacks [8,9] and related-key attacks [3] need to exploit the weakness of key scheduling. However, there is no key scheduling in Hummingbird. In particular, the subkeys used in four small block ciphers are independent. In addition, the four rotors affect the output of

each small block cipher in a nonlinear way. Hence, both slide attacks [8] and related-key attacks cannot be applied to the Hummingbird.

H. Interpolation and Higher Order Differential Attack. Interpolation and higher order differential attacks [24,25] can be applied to block ciphers with the low algebraic degree. As we discussed before for algebraic attack, the algebraic degree of the Hummingbird encryption is high. Hence it is difficult to apply interpolation and higher order differential attacks to the Hummingbird.

I. Complementation Properties. The DES has the following well-known complementation property, namely that if C is the ciphertext of the plaintext P under key K, then $\overline{C}$ is the ciphertext of $\overline{P}$ under key $\overline{K}$, where $\overline{x}$ is the bitwise complement of x. However, Hummingbird does not have this weakness due to the presence of the carry propagation resulting from four rotors.

4 Efficient Implementation of Hummingbird on Low-Power Microcontrollers

In this section, we present software implementation results of the compact version of Hummingbird (i.e., a single 4×4 S-box S_1 is used four times in the 16-bit block cipher) on two microcontrollers ATmega128L and MSP430, which are the processors used for the wireless sensor nodes MICAz and TELOSB/TMote Sky, respectively. We also provide two implementation variants for each platform, one of which is optimized for **code size** and the other for **speed**. Moreover, two variants can perform both encryption and decryption.

4.1 8-Bit Microcontroller ATmega128L and Development Tools

The ATmega128L from Atmel is a low-power 8-bit microcontroller based on the AVR enhanced RISC architecture. The processor is equipped with 133 powerful and highly-optimized instructions and most of them can be executed with one clock cycle. Moreover, ATmega128L comes with 128 KBytes of In-System Self-Programmable Flash, 4 KBytes EEPROM and 8 KBytes Internal SRAM. Optionally it can handle up to 64 KBytes of external memory space. Its clock frequency can run from 0 to 8 MHz and the power supplies can go from 2.7 to 5.5 V. In addition, at a frequency of 4 MHz with a power supply of 5 V the ATmega128L microcontroller draws 5.5 mA current when active, 2.5 mA in Idle Mode and less than 15 μA in Power-down Mode.

In order to implement and test the performance of Hummingbird on the target platform, we use a combination of the integrated development environment AVR Studio 4.17 [2] from Atmel and the open-source WinAVR-20090313 tool kit [34] for our purpose. While AVR Studio is used as an editor and a simulator, the WinAVR provides a GNU GCC compiler with the according libraries and a linker.

4.2 16-Bit Microcontroller MSP430 and Development Tools

Our second target microcontroller is a 16-bit MSP430F1611 from Texas Instrument, which is different in many ways from the Atmel chip. The MSP430F1611

microcontroller has a traditional *von-Neumann* architecture and all special function registers (SFRs), peripherals, RAM, and Flash/ROM share the same address space. Moreover, it comes with 48 KBytes Flash memory and 10 KBytes RAM. The clock frequency of the MSP430F1611 ranges from 0 to 8 MHz and the power supplies can go from 1.8 to 3.6 V. In particular, the MSP430F1611 microcontroller features the ultralow power consumption. At a frequency of 1 MHz and a voltage supply of 2.2 V the chip draws 200 μA current in Active Mode, 0.7 μA in Real-time Clock Mode, and 0.1 μA in Off Mode (RAM Retention). Although the instruction set of the MSP430F1611 only contains 27 instructions, 7 different addressing modes provide great flexibility in data manipulation.

We use CrossWorks for MSP430 Version 2 from Rowley Associates [31] to implement and simulate Hummingbird on the target platform. The CrossWorks for MSP430 bundles an ANSI C compiler, macro assembler, linker/locator, libraries, core simulator, flash downloader, JTAG debugger, and an integrated development environment CrossStudio. Different optimization levels can be set to generate codes with either smallest size or fastest speed.

4.3 Size Optimized Implementation

Note that the final round of the 16-bit block cipher in Hummingbird requires two derived round keys $K_5^{(i)} = K_1^{(i)} \oplus K_3^{(i)}$ and $K_6^{(i)} = K_2^{(i)} \oplus K_4^{(i)}$ (see Figure 2). For a size optimized implementation it is wise to calculate the above two keys $K_5^{(i)}$ and $K_6^{(i)}$ on-the-fly, which can save the storage requirements by 16 bytes. Moreover, the single S-box is implemented as a byte array with 16 elements, in which the lower half of a byte is used to store the value of the Hummingbird S-box and the higher half of a byte is padded with zeros. The S-box look-up of 16-bit block is conducted sequentially and 4 bits are processed each time. To generate the code with minimal size, we set the optimization level to be "OPT = s" for GCC compiler in WinAVR-20090313 and choose "Minimize Size" as the optimization strategy in CrossStudio, respectively.

Performance Results. Table 4 summarizes the memory consumption and cycle count of two lightweight ciphers Hummingbird and PRESENT on 8-bit and 16-bit microcontrollers for the size optimized implementation.

From Table 4 we note that the code size of Hummingbird is about 13% and 69% smaller than that of PRESENT on the 8-bit and 16-bit microcontrollers,

Table 4. Memory Consumption and Cycle Count Comparison (Size Optimized Implementation)

Cipher	Key Size [bit]	Block Size [bit]	8-bit/16-bit Microcontroller	Flash Size [bytes]	Hex Code Size [Kbytes]	SRAM Size [bytes]	Init. [cycles]	Enc. [cycles/ block]	Dec. [cycles/ block]
Hummingbird	256	16	ATmega128L	1,308	3.68	0	14,735	3,664	3,868
			MSP430F1611	1,064	2.95	0	9,667	2,414	2,650
PRESENT [29]	80	64	ATmega163	1,474	–	32	–	646,166	634,614
			C167CR	–	9.67	–	–	1,442,556	1,332,062

respectively. In addition, Hummingbird needs a relatively long initialization process when compared to the block cipher PRESENT because of the hybrid structure of block cipher and stream cipher adopted in Hummingbird. However, after an initialization procedure, Hummingbird encryption algorithm can achieve the throughput of 17.5 Kbps and 26.5 Kbps at a frequency of 4 MHz on the 8-bit and 16-bit microcontrollers, respectively. Under the same settings, the throughput of Hummingbird decryption algorithm can amount to 16.5 Kbps and 24.2 Kbps, respectively. Therefore, for the size optimized implementation, the throughput of Hummingbird is about 40 and 148 times faster than that of PRESENT on the target 8-bit and 16-bit platforms, respectively. Considering the cost of the initialization phase in Hummingbird, we compare the overall performance of Hummingbird and PRESENT for encrypting and decrypting messages with different length in the following Table 5.

Table 5. Overall Performance Comparison at 4 MHz (Size Optimized Implementation)

(a) Encryption Performance Comparison

Message Length	Microcontroller Word Length [bit]	PRESENT [29] Encryption [ms]	Hummingbird Encryption [ms]	Performance Improvement
64-bit	8	161.54	7.35	95.5%
	16	360.64	4.83	98.7%
128-bit	8	323.08	11.01	96.6%
	16	721.28	7.24	98.9%
192-bit	8	484.62	14.68	96.9%
	16	1,081.92	9.66	99.1%

(b) Decryption Performance Comparison

Message Length	Microcontroller Word Length [bit]	PRESENT [29] Decryption [ms]	Hummingbird Decryption [ms]	Performance Improvement
64-bit	8	158.65	7.55	95.2%
	16	333.02	5.07	98.5%
128-bit	8	317.31	11.42	96.4%
	16	666.03	7.72	98.8%
192-bit	8	475.96	15.29	96.8%
	16	999.05	10.37	98.9%

For the size optimized implementation, Table 5 shows that one can achieve around $95\% \sim 99\%$ performance improvements when using Hummingbird instead of PRESENT to encrypt or decrypt message blocks with length 64-bit, 128-bit, and 192-bit.

4.4 Speed Optimized Implementation

For a speed optimized implementation, we precompute and store all required round keys $K_5^{(i)}$ and $K_6^{(i)}$ (see Figure 2) in an array and this precomputation procedure requires additional 16 bytes of data memory and has to done once

when a new key is used. Furthermore, in order to accelerate the implementation of S-box layer in Hummingbird, we use a more efficient technique that combines two identical 4×4 S-boxes $S(x)$'s to form a larger 8×8 S-box $S_{8\times8}(x)$ such that $S_{8\times8}(x_1 \| x_2) = S(x_1) \| S(x_2)$, where x_1 and x_2 are 4-bit inputs to the two 4×4 S-boxes $S(x)$'s, respectively. Using the S-box $S_{8\times8}(x)$ significantly reduces the time for the S-box loop-up at the cost of 512 bytes of data memory (Note that both $S_{8\times8}(x)$ and $S^{-1}_{8\times8}(x)$ have 256 entries of each 1 byte). To generate the code with maximal speed, we set the optimization level to be "OPT = 3" for GCC compiler in WinAVR-20090313 and choose "Maximize Speed" as the optimization strategy in CrossStudio, respectively.

Performance Results. Table 6 summarizes the memory consumption and cycle count of two lightweight ciphers Hummingbird and PRESENT on 8-bit and 16-bit microcontrollers for the speed optimized implementation.

Table 6. Memory Consumption and Cycle Count Comparison (Speed Optimized Implementation)

Cipher	Key Size [bit]	Block Size [bit]	8-bit/16-bit Microcontroller	Flash Size [bytes]	Hex Code Size [Kbytes]	SRAM Size [bytes]	Init. [cycles]	Enc. [cycles/ block]	Dec. [cycles/ block]
Hummingbird	256	16	ATmega128L	10,918	30.5	0	8,182	1,399	1,635
			MSP430F1611	1,360	3.76	0	4,824	1,220	1,461
PRESENT [29]	80	64	ATmega163	2,398	–	528	–	9,595	9,820
			C167CR	–	92.2	–	–	19,464	33,354

From Table 6 we note that the code size of Hummingbird is about 78% larger and 96% smaller than that of PRESENT on the 8-bit and 16-bit microcontrollers, respectively. The main reason is that the -O3 option of the GCC compiler aggressively optimizes for speed by unrolling all loops in the code, which drastically increase the size of the code. Assuming that the microcontrollers operate at the frequency of 4 MHz, Hummingbird encryption algorithm can achieve the throughput of 45.7 Kbps and 52.5 Kbps on the 8-bit and 16-bit microcontrollers, respectively, which is about 0.7 and 2.5 times faster than that of PRESENT on the similar platforms. Base on the same assumption, the throughput of Hummingbird decryption algorithm can amount to 39.1 Kbps and 43.8 Kbps on the 8-bit and 16-bit microcontrollers, respectively, which is around 0.5 and 4.7 times faster than that of PRESENT on the similar platforms. Combining the overhead of the initialization phase in Hummingbird, we compare the overall performance of Hummingbird and PRESENT for encrypting and decrypting messages with different length in the following Table 7 for the speed optimized implementation.

For the speed optimized implementation, Table 7 shows that on 8-bit microcontrollers Hummingbird encryption is about 28.9% slower than PRESENT encryption when the message length is 64 bits. Furthermore, Hummingbird decryption is about 33.2% and 7.5% slower than PRESENT decryption for messages

Table 7. Overall Performance Comparison at 4 MHz (Speed Optimized Implementation)

(a) Encryption Performance Comparison

Message Length	Microcontroller Word Length [bit]	PRESENT [29] Encryption [ms]	Hummingbird Encryption [ms]	Performance Improvement
64-bit	8	2.40	3.38	-28.9%
	16	4.87	2.43	50.1%
128-bit	8	4.80	4.72	1.7%
	16	9.68	3.65	62.3%
192-bit	8	7.20	6.06	15.8%
	16	14.61	4.87	66.7%

(b) Decryption Performance Comparison

Message Length	Microcontroller Word Length [bit]	PRESENT [29] Decryption [ms]	Hummingbird Decryption [ms]	Performance Improvement
64-bit	8	2.46	3.68	-33.2%
	16	8.34	2.67	67.9%
128-bit	8	4.92	5.32	-7.5%
	16	16.68	4.13	75.2%
192-bit	8	7.38	6.95	5.8%
	16	25.02	5.59	77.6%

with length 64-bit and 128-bit, respectively. The main reason is that Hummingbird has a hybrid structure which involves a relatively long initialization process when compared to the block cipher PRESENT. However, on 16-bit microcontrollers Hummingbird is consistently faster (around 50% $\sim$ 78% performance improvements are achieved) than PRESENT for different message blocks in our experiment because the size (i.e., 16 bits) of the block and the internal state registers is perfectly suited to the architecture of 16-bit microcontrollers.

5 Encryption Mode and Concluding Remarks

In this paper we present a novel ultra-lightweight cryptographic algorithm, Hummingbird, which is a combination of block cipher and stream cipher. There are two modes related to Hummingbird as follows: (a) **Enigma Mode**: this is the mode where Hummingbird is used as a word-based cipher (16-bit word) where the plaintext is transitioned through a series of rotors. The ciphertext is dependent on the plaintext; (b) **Stream Mode**: this is the mode of Hummingbird where two values in the internal state ($RS1 \boxplus RS3$) are fed into the input of Hummingbird. The output is a keystream that is XOR'ed with plaintext. The hybrid structure adopted in Hummingbird can provide the designed security with small block size which is expected to meet the stringent response time and power consumption requirements in a large variety of embedded applications. We show that Hummingbird seems to be resistant to the most common attacks to block ciphers and stream ciphers including birthday attacks, differential and linear cryptanalysis,

structure attacks, algebraic attacks, cube attacks, etc. Moreover, efficient software implementations of Hummingbird on 8-bit and 16-bit microcontrollers are also presented. When compared to the ultra-lightweight block cipher PRESENT implemented on similar platforms, our experimental results show that after a system initialization procedure Hummingbird can achieve up to 147 and 4.7 times faster throughput for a size-optimized and a speed-optimized implementations, respectively.

Acknowledgement

This work is supported by NSERC Strategic Grant. The authors would like to thank the two anonymous reviewers for their insightful comments.

References

1. Anderson, R., Biham, E., Knudsen, L.: Serpent: A Proposal for the Advanced Encryption Standard, http://www.cl.cam.ac.uk/~rja14/Papers/serpent.pdf
2. Atmel. AVR Studio 4.17, http://www.atmel.com/dyn/Products/tools_card.asp?tool_id=2725
3. Biham, E.: New Types of Cryptanalytic Attacks Using Related Keys. J. of Cryptology 7, 229–246 (1994)
4. Biham, E.: Cryptanalysis of Multiple Modes of Operation. J. Cryptology 11(1), 45–58 (1998)
5. Biham, E.: Cryptanalysis of Triple Modes of Operation. J. Cryptology 12(3), 161–184 (1999)
6. Biham, E., Knudsen, L.R.: Cryptanalysis of the ANSI X9.52 CBCM Mode. J. Cryptology 15(1), 47–59 (2002)
7. Biham, E., Shamir, A.: Differential Cryptanalysis of the Data Encryption Standard. Springer, New York (1993)
8. Biryukov, A., Wagner, D.: Slide Attacks. In: Knudsen, L.R. (ed.) FSE 1999. LNCS, vol. 1636, pp. 245–259. Springer, Heidelberg (1999)
9. Biryukov, A., Wagner, D.: Advanced Slide Attacks. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 589–606. Springer, Heidelberg (2000)
10. 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)
11. 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)
12. De Cannière, C., Preneel, B.: Trivium – A Stream Cipher Construction Inspired by Block Cipher Design Principles. ECRYPT Stream Cipher (2005), http://www.ecrypt.eu.org/stream/papersdir/2006/021.pdf
13. Dinur, I., Shamir, A.: Cube Attacks on Tweakable Black Box Polynomials. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 278–299. Springer, Heidelberg (2010)

14. Eisenbarth, T., Kumar, S., Paar, C., Poschmann, A., Uhsadel, L.: A Survey of Lightweight-Cryptography Implementations. IEEE Design & Test of Computers 24(6), 522–533 (2007)
15. Engels, D., Fan, X., Gong, G., Hu, H., Smith, E.M.: Ultra-Lightweight Cryptography for Low-Cost RFID Tags: Hummingbird Algorithm and Protocol, Centre for Applied Cryptographic Research (CACR) Technical Reports, CACR 2009-29, http://www.cacr.math.uwaterloo.ca/techreports/2009/cacr2009-29.pdf
16. Fan, X., Hu, H., Gong, G., Smith, E.M., Engels, D.: Lightweight Implementation of Hummingbird Cryptographic Algorithm on 4-Bit Microcontroller. In: The 1st International Workshop on RFID Security and Cryptography 2009 (RISC 2009), pp. 838–844 (2009)
17. Feldhofer, M., Dominikus, S., Wolkerstorfer, J.: Strong Authentication for RFID Systems Using the AES Algorithm. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 357–370. Springer, Heidelberg (2004)
18. Feldhofer, M., Wolkerstorfer, J., Rijmen, V.: AES Implementation on a Grain of Sand. IEE Proceedings Information Security 15(1), 13–20 (2005)
19. Hämäläinen, P., Alho, T., Hännikäinen, M., Hämäläinen, T.D.: Design and Implementation of Low-Area and Low-Power AES Encryption Hardware Core. In: The 9th EUROMICRO Conference on Digital System Design: Architectures, Methods and Tools - DSD 2006, pp. 577–583. IEEE Computer Society, Los Alamitos (2006)
20. Hell, M., Johansson, T., Meier, W.: Grain: A Stream Cipher for Constrained Environments. International Journal of Wireless and Mobile Computing 2(1), 86–93 (2007)
21. Hong, D., Sung, J., Hong, S., Lim, J., Lee, S., Koo, B.S., Lee, C., Chang, D., Lee, J., Jeong, K., Kim, H., 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)
22. Lim, C., Korkishko, T.: mCrypton - A Lightweight Block Cipher for Security of Low-cost RFID Tags and Sensors. In: Song, J.-S., Kwon, T., Yung, M. (eds.) WISA 2005. LNCS, vol. 3786, pp. 243–258. Springer, Heidelberg (2006)
23. Liu, D., Yang, Y., Wang, J., Min, H.: A Mutual Authentication Protocol for RFID Using IDEA, Auto-ID Labs White Paper, WP-HARDWARE-048 (March 2009), http://www.autoidlabs.org/uploads/media/AUTOIDLABS-WP-HARDWARE-048.pdf
24. Jakobsen, T., Knudsen, L.: The Interpolation Attack on Block Ciphers. In: Biham, E. (ed.) FSE 1997. LNCS, vol. 1267, pp. 28–40. Springer, Heidelberg (1997)
25. Lai, X.: Higher Order Derivatives and Differential Cryptanalysis. In: Proceedings of Symposium on Communication, Coding and Cryptography, in honor of James L. Massey on the occasion of his 60'th birthday (1994)
26. 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)
27. Leander, G., Poschmann, A.: On the Classification of 4 Bit S-Boxes. In: Carlet, C., Sunar, B. (eds.) WAIFI 2007. LNCS, vol. 4547, pp. 159–176. Springer, Heidelberg (2007)
28. Matsui, M.: Linear Cryptanalysis Method for DES Cipher. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 386–397. Springer, Heidelberg (1994)
29. Poschmann, A.: Lightweight Cryptography - Cryptographic Engineering for a Pervasive World, Ph.D. Thesis, Department of Electrical Engineering and Information Sciences, Ruhr-Universit*ä*et Bochum, Bochum, Germany (2009)

30. 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)
31. Rowley Associates. CrossWorks for MSP430, `http://www.rowley.co.uk/msp430/index.htm`.
32. Standaert, F.-X., Piret, G., Gershenfeld, N., Quisquater, J.-J.: SEA: A Scalable Encryption Algorithm for Small Embedded Applications. In: Domingo-Ferrer, J., Posegga, J., Schreckling, D. (eds.) CARDIS 2006. LNCS, vol. 3928, pp. 222–236. Springer, Heidelberg (2006)
33. Wagner, D.: Cryptanalysis of Some Recently-Proposed Multiple Modes of Operation. In: Vaudenay, S. (ed.) FSE 1998. LNCS, vol. 1372, pp. 254–269. Springer, Heidelberg (1998)
34. WinAVR. Suite of Executable, Open Source Software Development Tools for the Atmel AVR Series of RISC Microprocessors Hosted on the Windows Platform, `http://winavr.sourceforge.net/`
35. Youssef, A., Gong, G.: On the Interpolation Attacks on Block Ciphers. In: Schneier, B. (ed.) FSE 2000. LNCS, vol. 1978, pp. 109–120. Springer, Heidelberg (2001)