Survey and Benchmark of Block Ciphers for Wireless Sensor Networks

Yee Wei Law, Jeroen Doumen and Pieter Hartel

Faculty of Electrical Engineering, Mathematics and Computer Science
University of Twente, The Netherlands
{ywlaw, doumen, pieter}@cs.utwente.nl

Abstract. Choosing the most storage- and energy-efficient block cipher specifically for wireless sensor networks (WSNs) is not as straightforward as it seems. To our knowledge so far, there is no systematic evaluation framework for the purpose. In this paper, we have identified the candidates of block ciphers suitable for WSNs based on existing literature. For evaluating and assessing these candidates, we have devised a systematic framework that not only considers the security properties but also the storage- and energy-efficency of the candidates. Finally, based on the evaluation results, we have selected the suitable ciphers for WSNs, namely Rijndael for high security and energy efficiency requirements; and MISTY1 for good storage and energy efficiency.

1 Introduction

A wireless sensor network (WSN) is a network comprised of a large number of sensors that (1) are physically small, (2) communicate wirelessly among each other, and (3) are deployed without prior knowledge of the network topology. Due to the limitation of their physical size, the sensors tend to have storage space, energy supply and communication bandwidth so limited that every possible means of reducing the usage of these resources is aggressively sought. For example, a sensor typically has 8∼120KB of code memory and 512∼4096 bytes of data memory. The energy supply of a sensor is such that it will be depleted in less than 3 days if operated constantly in active mode [1]. The transmission bandwidth ranges from 10kbps to 115kbps. Table 1 compares the sensor node used in the EYES project (`eyes.eu.org`) with Smart Dust [2] and the Intel Research mote [3]. Karlof and Wagner made an interesting observation that WSNs will more likely ride Moore's Law *downward* [4], that is, instead of relying on the computing power to double every 18 months, we are bound to seek ever cheaper solutions. However looking at the current development of WSNs (Table 1), computing power is indeed increasing, though not necessarily at the rate predicted by Moore's Law. Either way, we are conservative and assume that the hardware constraints of WSNs will remain rather constant for some time to come.

Cryptographic algorithms are an essential part of the security architecture of WSNs, using the most efficient and sufficiently secure algorithm is thus an effective means of conserving resources. By 'efficient' in this paper we mean requiring

Table 1. Comparison of the EYES node with Smart Dust and the Intel Research mote.

	Smart Dust	EYES node	Intel mote
CPU	8-bit, 4 MHz	16-bit, 8 MHz	16-bit, 12 MHz
Flash memory	8 KB	60 KB	512 KB
RAM	512 B	2 KB	64 KB
Frequency	916 MHz	868.35 MHz	900 MHz
Bandwidth	10 kbps	115.2 kbps	100 kbps

little storage and consuming little energy. Although transmission consumes more energy than computation, our focus in this paper is on computation and we can only take transmission energy into account when considering the security scheme as a whole. The essential cryptographic primitives for WSNs are block ciphers, message authentication codes (MACs) and hash functions. Among these primitives, we are only concerned with block ciphers in this paper, because MACs can be constructed from block ciphers [5] whereas hash functions are relatively cheap [6]. Meanwhile, public-key algorithms are well-known to be prohibitively expensive [7].

Our selection of block ciphers is RC5 [8], RC6 [9], Rijndael [10], MISTY1 [11], KASUMI [12] and Camellia [13]. Although Rijndael has been selected by the American National Institute of Standards and Technology (NIST) as the Advanced Encryption Standard (AES), after a five-year long standardisation process that includes extensive benchmarking on a variety of platforms ranging from smart cards [14] to high end parallel machines [15], the selection of Rijndael for our platform is *not* obvious. This is because the fact that AES is *on average* the best performer on a range of standard platforms, does not mean that it also performs best on our platform, which is Texas Instruments' 16-bit RISC-based MSP430F149 [16], chosen for its ultra-low power consumption (`www.ti.com`). During the evaluation process of AES, the focus has been on 8-bit smart cards, 32-bit mainstream architectures and 64-bit high-end platforms, but not 16-bit architectures, further highlighting the importance of our study.

The contribution of this paper is three-fold: (1) to identify the candidates of block ciphers suitable for WSNs based on existing literature; (2) to provide a systematic evaluation framework for assessing the suitability of the ciphers, in terms of both security and efficiency; (3) and to select suitable ciphers for WSNs based on the evaluation results.

Our evaluation framework consists of (1) literature survey, and (2) benchmarking. The rest of this paper is organised as follows. Section 2 contains a brief literature survey of the block ciphers, shedding light on their security properties and accomplishing the survey part of our evaluation. Section 3 details aspects of benchmarking. Section 4 provides our evaluation results. Section 5 concludes.

2 Literature Survey

A typical cipher consists of three components: (1) the encryption algorithm, (2) the decryption algorithm and (3) the key expansion algorithm (also known as key scheduling or key setup). The key expansion algorithm expands the *user key* or *cipher key* to a larger intermediate key, to allow (ideally) all bits of the cipher key to influence every round of the encryption algorithm. For most ciphers, key expansion needs only be done once to cater for both encryption and decryption; for other ciphers however, key expansion has to be done separately for encryption and decryption (e.g. for Rijndael). The most important parameters of a block cipher are (1) the *key length* (which determines the block length) it supports, (2) the *word size* and (3) the *nominal number of rounds*. In the ensuing discussion, for each cipher we would give the reasons why we have chosen to evaluate the cipher, as well as provide status quo information on the security strength of the cipher.

2.1 RC5

We have chosen to evaluate RC5 [8] because RC5 is a well-known algorithm that has been around since 1995 without crippling weaknesses. Although `distributed.net` has managed to crack a 64-bit RC5 key in RSA Laboratories Secret-Key Challenge after 1757 days of computing involving 58,747,597,657 distributed work units, the standard key length of RC5 is 128 bits and RC5 has managed to withstand years of cryptanalysis.

In terms of design, RC5 is an innovative cipher that uses *data-dependent* rotations and is fully parameterised in word size, number of rounds and key length. Such flexibility is rare among mainstream ciphers. RC5 is also designed to be suitable for hardware as well as software implementations. Lastly, Perrig et al. [17] have made the pioneering effort of demonstrating the advantage of RC5 on wireless sensors.

Security RC5 is conventionally represented as RC5-$w/r/b$, where w the word size is the number of bits in a word of the target computing platform, r is the number of rounds, and b the key length is the number of bytes of a key. The attacks found up to year 1998 have been summarised in Kaliski and Yin's technical report [18]. The best attack cited is Biryukov and Kushilevitz's [19], which breaks RC5-32/12/16 with just 2^{44} chosen plaintexts (compared with 2^{43} known plaintexts for DES). At 16 rounds, Kaliski and Yin suggests that breaking RC5-32 requires 2^{66} chosen plaintexts. It is for this reason that the nominal number of rounds has been proposed to be increased to 18.

After Biryukov and Kushilevitz, Borst et al. [20] manage to reduce drastically the storage requirements for attacking RC5-32 and RC6-64 using the linear hull effect [21]. This is known as the first *experimentally executed* known plaintext attacks on reduced versions of RC5 (with up to 5 rounds).

Shimoyama et al. [22] introduce another route of attack based on statistical correlations derived from χ^2 tests. They have managed to derive the last round

key of up to 17 rounds by using chosen plaintext attack. While Shimoyama et al. use a chosen plaintext attack, Miyaji et al. [23] use a known plaintext attack, which breaks RC5-32/10 with $2^{63.67}$ plaintexts at a probability of 90%.

The attacks described so far are theoretical. Ironically, intended as a security feature, data-dependent rotation invites implementation-based attacks. One such attack (on RC5-32/12/16) has been proposed by Hanschuh and Heys [24] which only needs about 2^{20} encryption timings and a time complexity between 2^{28} and 2^{40}. Kelsey et al. [25] describe another implementation-based attack by observing the processor flags.

To conclude, RC5-32/18 (18 rounds) should provide sufficient security.

2.2 RC6

We have chosen to evaluate RC6 [9] because like RC5, RC6 is parameterised and has a small code size. RC6 is one of the five finalists that competed in the AES challenge and according to some AES evaluation reports [26][15], it has reasonable performance. Lastly, RC6 has been chosen as the algorithm of choice by Slijepcevic et al. [27] for WSNs. We are interested in seeing if their choice is justified.

Security The technical report of the Cryptography Research and Evaluation Committee (CRYPTREC), Information-technology Promotion Agency (IPA) of Japan [28] has a summary of the attacks known up to year 2001. The first notable attack is by Knudsen and Meier [29]. Their attack is based on statistical correlations derived from χ^2 tests. By observing the the nonuniformness of the five least significant bits from each of the two Feistel halves in RC6, they estimate that $2^{13.8+16.2r}$ plaintexts are required to distinguish $3 + 2r$-round RC6 from a pseudorandom permutation. For example, when $r = 8$, $2^{143.4}$ plaintexts are required.

At the same time, Gilbert et al. [30] present a theoretical attack of 14 rounds RC6 based on an iterative statistical weakness found in RC6's round function. The attack requires 2^{118} known plaintexts, a memory size of 2^{112} and 2^{122} operations.

Takenaka et al. [31][32] propose the Transition Matrix Computation technique for evaluating security against χ^2 attacks, by which they are able to deduce the "weakest key" of RC6 against χ^2 attacks.

Shimoyama et al. [33] show that the 64-bit target extended key of 18-round RC6 with weak key (which exists one in every 2^{90} keys), can be recovered with probability 95% by *multiple linear cryptanalysis* with $2^{127.423}$ known plaintexts, a memory of size 2^{64}, and $2^{193.423}$ computations of the round-function.

To conclude, RC6-32/20 (20 rounds) should provide sufficient security.

2.3 Rijndael

We have chosen to evaluate Rijndael [10] because Rijndael is the *de facto* Advanced Encryption Standard, mandated by the NIST of the United States,

chosen after extensive scrutiny and performance evaluation (`csrc.nist.gov/encryption/aes`). It is also one of the ciphers recommended by the New European Schemes for Signature, Integrity and Encryption (NESSIE) Consortium (`www.cryptonessie.org`), and Japan's CRYPTREC [34]. Rijndael, as AES, is well studied and there are efficient implementations on a wide range of platforms (`www.rijndael.com`). We would however like to obtain first-hand experience of evaluating Rijndael on our particular platform, which has never been studied before during the evaluation process of AES.

Security Like most modern block ciphers, Rijndael is designed with resistance against differential and linear cryptanalysis in mind, using the latest results in cryptographic research [10]. For example, Cheon et al.'s impossible differential cryptanalysis [35] require $2^{91.5}$ chosen ciphertexts to attack 6-round Rijndael-128 (128-bit keys). Gilbert and Minier [36] describe (1) an attack of 7-round Rijndael-196 and Rijndael-256 with 2^{32} chosen plaintexts and a complexity of about 2^{140}, and (2) an attack of 7-round Rijndael-128 with 2^{32} chosen plaintexts and a computational complexity slightly less than exhaustive search. Ferguson et al. [37] report a related-key attack of 9-round Rijndael-256 with time complexity 2^{224}, which is of course far from practical. No attack is known for Rijndael of more than 7 rounds.

On the other hand, although Rijndael has been chosen as the AES, the security of Rijndael has gone through twists and turns of controversy. The algebraic nature of Rijndael [38], has interestingly opened up possible avenues of other non-traditional attacks as summarised in the Crypto-Gram Newsletter of September 2002 [39]. It started with Courtois and Pieprzyk [40][41] presenting evidence that the security of Rijndael might not grow exponentially as intended with the number of rounds. Their technique is based on expressing the S-boxes of Rijndael in an overdefined system of multivariate quadratic equations which can be solved by an algorithm called XSL. XSL is in turn based on XL [42]. The security of Rijndael therefore lies on the computational complexity of XL, which to date remains an open problem [43]. In spite of Moh's dispute [44], whether the technique would *not* work remains to be proven [45]. In the meantime, Murphy and Robshaw [46] derive an alternative representation of Rijndael that is easier to cryptanalyse, by embedding Rijndael in a cipher called BES that uses only simple algebraic operations in $GF(2^8)$. They show that Rijndael encryption can then be described by an extremely sparse overdetermined multivariate quadratic system over $GF(2^8)$, whose solution would recover the key. In another paper, Murphy and Robshaw [47] argue that while XSL does not have estimates accurate enough to substantiate claims of the existence of a key recovery attack, XSL does help solve their $GF(2^8)$ system of equations more efficiently than Courtois et al.'s $GF(2)$ system of equations. Combining Coppersmith [48]'s correction of Courtois et al.'s estimates, Murphy et al. further deduce that the security of Rijndael-128 would be reduced from the theoretical complexity of exhaustive key search, 2^{128} to 2^{100}, *if* XSL is a valid technique.

On the other front, Fuller and Millan [49] unravel serious linear redundancy in the only nonlinear component, i.e. the S-box, of Rijndael: the 8×8 S-box behaves

actually like a 8×1 S-box. They, by studying the invariance properties of the local connection structure of affine equivalence classes, discover that the outputs of the S-box are all equivalent under affine transformation. The essence of their discovery can be summarised in the following simple mathematical expression: Let $b_i(x)$ and $b_j(x)$ be two distinct outputs of the S-box, then there exists a non-singular 8×8 matrix D_{ij} and a binary constant c_{ij} such that $b_j(x) = b_i(x)D_{ij} \oplus c_{ij}$.

Independent of the above development, Filiol shocked the scientific community in January 2003 by announcing a break of Rijndael with his plaintext-dependent repetition codes cryptanalysis technique [50]. By detecting bias in the boolean functions of Rijndael, Filio claimed that he was able to obtain 2 bits of a Rijndael key with only 2^{31} ciphertexts and a computational complexity of mere $\mathcal{O}(2^{31})$. Fortunately, several independent cryptographers were quick to dismiss the claim [51].

To conclude, the research on Rijndael has entered an interesting era. More and more previously unknown properties are now being discovered and analysed [52][53][54]. We are despite the above debate adopting the recommendation of NESSIE [55] and CRYPTREC [34], that Rijndael is secure.

2.4 MISTY1

We have chosen to evaluate MISTY1 [11] because MISTY1 is one of the CRYPTREC-recommended 64-bit ciphers [34] and is the predecessor of KASUMI, the 3GPP-endorsed encryption algorithm [12]. MISTY1 is specifically designed to resist differential and linear cryptanalysis. MISTY1 is designed for high-speed implementations on hardware as well as software platforms by using only logical operations and table lookups. We have also found MISTY1 to be particularly suitable for 16-bit platforms. MISTY1 is a royalty-free open standard documented in RFC2994 [56].

Security Babbage and Frisch [57] demonstrate the possibility of a 7th order differential cryptanalytic attack on 5-round MISTY1. According to them, none of the S-boxes with optimal linear and differential properties has an optimal behaviour with respect to higher order differential cryptanalysis, however as improvement, the number of rounds of the FI function could be increased. As shall be seen later, KASUMI, derived from MISTY1, incorporated such improvement, in that it has four instead of three S-boxes in its FI function.

Kühn found an impossible differential attack on 4-round MISTY1 using 2^{38} chosen plaintexts and 2^{62} encryptions [58]. In the same paper [58], a collision-search attack has also been described – the attack requires 2^{28} chosen plaintexts and 2^{76} encryptions. In a later paper, Kühn [59] shows that the FL function introduces a subtle weakness in 4-round MISTY1. This weakness allows him to launch a slicing attack with as few as $2^{22.5}$ chosen plaintexts on 4-round MISTY1.

The best known attack on 5-round MISTY1 so far is Knudsen and Wagner's integral cryptanalysis [60], at a cost of 2^{34} chosen plaintexts and 2^{48} time complexity.

To conclude, the security margin provided by MISTY1 is relatively small compared with modern 128-bit ciphers like Rijndael, but with full nominal rounds, MISTY1 is still reasonably secure.

2.5 KASUMI

We have chosen to evaluate KASUMI [12] because KASUMI, as the 3GPP-endorsed encryption algorithm [12], is presumably well-suited for embedded applications, and has gone through considerable expert scrutiny.

Security KASUMI more or less inherits the security and performance benefits of MISTY1.

At the same time reporting attacks on MISTY1 and MISTY2, Kühn is also able to extend the attacks to KASUMI [58]. His impossible differential attack on 6-round KASUMI requires 2^{55} chosen plaintexts and 2^{100} encryptions.

Kang et al. [61][62] prove that 3-round KASUMI is not a pseudorandom permutation ensemble but 4-round KASUMI is a pseudorandom permutation ensemble. However Tanaka et al. [63] show that 4-round KASUMI without FL functions can be attacked using effective 2nd order differentials. The attack requires only 1,416 chosen plaintexts and $2^{22.11}$ computational cost, $2^{29.89}$ times faster than brute-force search.

To conclude, like MISTY1, the security margin provided by KASUMI is relatively small compared with modern 128-bit ciphers like Rijndael.

2.6 Camellia

We have chosen to evaluate Camellia [13] because Camellia is one of the NESSIE- and CRYPTREC-recommended 128-bit ciphers [34]. Camellia is designed for high-speed implementations on hardware as well as software platforms by using only logical operations and table lookups. Aoki et al. [64] claim their hardware implementation of Camellia occupies only 7.875K gates using a 0.11μm CMOS ASIC library and is in the smallest class among all existing 128-bit block ciphers. Camellia is designed not only to be resistant to differential cryptanalysis, linear cryptanalysis, higher order differential attacks, interpolation attacks, related-key attacks, truncated differential attacks, boomerang attacks and slide attacks, but also with a large safety margin in view of anticipated progress in cryptanalysis techniques [64][65]. Futhermore, Camellia is royalty-free.

Security He and Qing [66] discover that the Square attack [67] is not only applicable to the block cipher Square but also to ciphers with a Fesitel structure. The complexity of their attack on 6-round Camellia is 3328 chosen plaintexts and 2^{112} encryptions. Sugita et al. [68] found a non-trivial 7-round impossible differential. Lee et al. [69]'s truncated differential cryptanalysis of 7-round Camellia without the FL function, requires only 192 encryptions but $2^{82.6}$ chosen plaintexts. Yeom et al. [70] propose a Square attack on 9-round Camellia at a cost of $2^{50.5}$ chosen plaintexts and $2^{202.2}$ encryptions. Hatano et al.'s attack of 11-round Camellia

with 256-bit keys, requires 2^{93} chosen plaintexts and $2^{255.6}$ encryptions, which is just a little less than brute force search [71].

To conclude, 18-round Camellia offers sufficient security margin.

3 Methodology and Consideration

For benchmarking, we consider: (1) the cipher parameters, (2) the cipher operation modes, (3) the compiler toolchain, and (4) the implementation sources.

3.1 Cipher Parameters

To evaluate the ciphers, we feed each of the ciphers with 30 bytes of plaintext, since according to Perrig et al. [17], a message of 30 bytes is rather realistic for WSNs. Table 2 lists the parameters we have adopted for each cipher (some of them actually have fixed, unadjustable parameters but we list them anyway for the sake of clarity). Although RC5 and RC6 allow variable word size, without the backing of relevant cryptanalytic research, we are not sure how many rounds to use if we pick a non-standard word size of 16 bits, which is the word size of our platform. Therefore, for RC5 and RC6, we are using the standard word size of 32 bits.

Table 2. Cipher parameters.

Cipher	Key length	Rounds	Block length	Cipher	Key length	Rounds	Block length
RC5-32	128	18	64	MISTY1	128	8	64
RC6-32	128	20	128	KASUMI	128	8	64
Rijndael	128	10	128	Camellia	128	18	128

3.2 Operation Modes

The naïve approach of encrypting a message longer than one block, by dividing the message into multiple blocks and encrypting the blocks separately, is called the *electronic codebook mode* (ECB) mode. ECB is insecure since an adversary can construct valid ciphertexts from the original ciphertext by arbitrarily rearranging, repeating, omitting blocks from the original ciphertext. More secure operation modes are used in practice. These operation modes do not only affect the security, but also the energy-efficiency of the encryption scheme. Their fault tolerance against ciphertext error (where bits are changed), and synchronisation error (where bits might are added or lost) must also be taken into account (Table 3). In our implementation of CBC, ciphertext stealing [72] is supported, so padding is not required. Meanwhile, some researchers [73] claim that CBC has an information leakage vulnerability due to the birthday paradox.

Table 3. Comparison of operation modes.

Operation mode	Against ciphertext error...	Against synchronisation error...
Cipher-Block Chaining (CBC)	A transmission bit error in a block affects only the decryption of two subsequent blocks.	Irrecoverable.
Cipher Feedback Mode (CFB)	Recoverable after a certain number of blocks.	Recoverable after a certain number of blocks.
Output Feedback Mode (OFB)	Error in one ciphertext block only results in error in the corresponding plaintext block.	Irrecoverable.
Counter (CTR)	Similar to OFB.	Irrecoverable.

3.3 Compilers

For compilation, we are currently only using IAR Systems' MSP430 C Compiler V2.20A/W32 (`www.iar.com`). For debugging and profiling, we use IAR Systems' Embedded Workbench 3.2 with the integrated C-Spy Debugger and profiling plug-in. Another viable compiler is the GNU C compiler in the MSPGCC toolchain (`mspgcc.sf.net`), however we are not using it due to the lack of profiling support by the toolchain.

When performing our benchmarking, we compare maximum size optimisation with maximum speed optimisation. When using maximum speed optimisation, we also configure the compiler to perform common subexpression elimination (`COMMON_SUBEXP_ELIM`), loop unrolling (`LOOP_UNROLL`), function inlining (`FUNC_INLINE`) and code motion (`CODE_MOTION`). When we measure the code size, we use the Release version, but when measuring the speed we use the Debug version because only then can we profile the code, and the debug information does not induce additional cycles. All code is compiled to use the hardware multiplier. The object format for the Release version is Texas Instruments' msp430-txt format (for the Debug version, IAR's own proprietary format UBROF version 9 is used). Although this compiler does support inline assembler code, it is far less advanced than that offered by, say, the GNU C compiler, hence we are currently not taking advantage of the feature.

3.4 Implementation Sources

To avoid reinventing the wheel, we try to use and improve as much existing source code as possible. We briefly compare a few source code libraries that we are aware of in Table 4:

To conclude this section, we have adapted most of our code from OpenSSL, and for the ciphers that do not have public implementation, we have adapted the reference source code from the original papers the ciphers are proposed in. Our source code can be found at `http://www.cs.utwente.nl/~ywlaw/src/crypto_test.zip`.

Table 4. Comparison of several cryptographic libraries.

Sources	Advantages	Disadvantages
OpenSSL (`www.openssl.org`)	(1) Its cipher implementations are widely believed to be highly optimised with the help of the cipher designers themselves and after years of tweaking. (2) It uses low-level C and assembly language, avoiding the overhead that would have been induced by higher-level languages like C++. (3) It supports most if not all cipher operation modes.	(1) It only implements standard ciphers such as the DES, RC4, RC5, AES and no experimental modern ciphers (e.g. RC6, MISTY1, KASUMI, Camellia). (2) The cipher implementations do not present a uniform interface, like other C++ libraries do.
Crypto++ (`www.eskimo.com/~weidai/cryptlib.html`)	(1) This free C++ class library of cryptographic schemes by Wei Dai, supports most if not all cipher operation modes. (2) The binaries of Crypto++ version 5.0.4 have received FIPS 140-2 level 1 validation. (3) It is well-suited for benchmarking.	(1) By using C++, it incurs performance overhead and porting issues for embedded platforms. (2) Although it does implement some non-standard modern ciphers (e.g. RC6), it does not implement many others (e.g. MISTY1, KASUMI and Camellia).
Botan (`botan.randombit.net`)	(1) Supports most if not all cipher operation modes. (2) It is well-suited for benchmarking.	Similar to Crypto++.
Catacomb (`www.excessus.demon.co.uk/misc-hacks`)	(1) Supports most if not all cipher operation modes. (2) It is well-suited for benchmarking. (3) Compared with Crypto++ and Botan, it uses faster and more portable C.	Although it does implement some non-standard modern ciphers (e.g. RC6), it does not implement many others (e.g. MISTY1, KASUMI and Camellia).

4 Results

We have performed our measurements in standalone mode, i.e. without interaction with an OS. We have taken care in making the interface of the cipher implementations as uniform as possible, so that no difference in performance is a result of the difference in the interfaces. Our benchmark parameters are memory and CPU cycles. Given in this section are the results in terms of these two parameters.

4.1 Memory

We refer to two types of memory: (1) code memory, in the form of Flash memory and (2) data memory, in the form of RAM. The memory organisation of MSP430F149 is such that data memory ranges from address 0200h to 09FFh, whereas code memory ranges from 01100h to 0FFFFh. There are three types of

segments for this platform: CODE, CONST and DATA. A typical policy is such that CODE and CONST segments are put in the code memory, while DATA segments (CSTACK, DATA16_AN, DATA16_I, DATA16_N, DATA16_Z and HEAP) in the data memory.

For the memory usage of CODE and CONST segments, we can read it off the memory map listing generated by the compiler. For the memory usage of DATA segments except CSTACK, we can also read it off the memory map listing. For CSTACK however, we have to rely on manual inspection of the code, and our estimation is conservative. The storage for plaintext, ciphertext, cipher key as well as the expanded key is not included in the code memory nor data memory in Table 5. The module named 'skey' in Table 5 and other tables henceforth refers to the key expansion algorithm. The code memory for an operation mode module, say a CBC module, takes into account the code memory for the barebone encryption code, the CBC code as well as the S-boxes (or lookup tables in general). The details of which compiler switches are used for which modules, together with the measurement figures can be found in the file `results.xls` of the source distribution (cf Section 3.4).

4.2 CPU Cycles

The computational complexity of an algorithm translates directly to its energy consumption. Assuming the energy per CPU cycle is fixed (which is justified in the Appendix), then by measuring the number of CPU cycles executed per byte, we get the amount of energy consumed by byte. For example, the processor we are using, MSP430F149 [16] draws a nominal current of 420μA at 3V and at a clock frequency of 1MHz in active mode – this means that the energy per instruction cycle (*for the processor alone*) is theoretically 1.26 nJ.

The results can be found in Table 6. Observe that among all modes, only CBC is asymmetrical in the sense that encryption and decryption consume a slightly different number of CPU cycles. From Table 5 and 6, we can see that OFB is not only the most space-saving but also the fastest mode.

4.3 Observation

First we analyse Table 5 and Table 6 cipher by cipher:

RC5-32: We note that size optimisation and speed optimisation result in vastly different code sizes, with the effect being most prominent in the OFB mode, for which the ratio between the size optimised code and speed optimised code is 1:5.1. For the key setup module (skey), we use the reference implementation version to optimise its size, but the OpenSSL version to optimise its speed. The size ratio between these two key setup implementations is 1:1.2 whereas the speed ratio is 1:2 – a heavy penalty in speed for a little saving in code memory.

RC6-32: Nechvatal et al. [26] note that on the Z80 processor, the key setup of RC6 is very time-consuming, and it takes about four times as many cycles as encryption (of one block) does. Our measurement confirms this observation. There

Table 5. Memory requirements in bytes.

Cipher	Module	Size optimised		Speed optimised	
		Code mem.	Data mem.	Code mem.	Data mem.
RC5-32	skey	533	92	650	300
	CBC	1653	52	6045	52
	CFB	1067	39	5175	39
	OFB	997	36	5097	36
	CTR	1129	54	5229	54
RC6-32	skey	559	60	559	60
	CBC	2121	86	2329	86
	CFB	1810	57	1912	57
	OFB	1740	54	1834	54
	CTR	1946	88	2042	88
Rijndael	skey	1298	14	1844	14
	CBC	13448	92	13998	92
	CFB	12888	63	13190	63
	OFB	12816	60	13110	60
	CTR	12966	94	13262	94
MISTY1	skey	577	12	1053	12
	CBC	6973	58	7969	58
	CFB	6381	45	7093	45
	OFB	6311	42	7015	42
	CTR	6443	60	7147	60
KASUMI	skey	500	58	584	42
	CBC	9665	64	9541	64
	CFB	9073	51	10745	51
	OFB	9003	48	9791	48
	CTR	9135	66	9923	66
Camellia	skey	4068	160	5208	160
	CBC	16096	117	24530	117
	CFB	15479	88	23617	88
	OFB	15409	85	23539	85
	CTR	15617	119	23749	119

For a digest of this table, see Section 4.1.

Table 6. CPU cycles for key expansion (per key) and operation modes (per byte).

Cipher	Module	Size optimised		Speed optimised	
		Encryption	Decryption	Encryption	Decryption
RC5-32	skey	81704	81704	40565	40565
	CBC	1247	1270	712	699
	CFB	1250	1250	691	691
	OFB	1225	1225	668	668
	CTR	1236	1236	679	679
RC6-32	skey	95114	95114	93808	93808
	CBC	1222	1247	1132	1132
	CFB	1229	1229	1129	1129
	OFB	1205	1205	1120	1120
	CTR	1211	1211	1125	1125
Rijndael	skey	1745	6769	1313	5034
	CBC	321	331	218	223
	CFB	324	324	212	212
	OFB	299	299	204	204
	CTR	309	309	213	213
MISTY1	skey	882	882	601	601
	CBC	478	485	525	531
	CFB	479	479	511	511
	OFB	455	455	503	503
	CTR	467	467	515	515
KASUMI	skey	2571	2571	1381	1381
	CBC	568	576	708	715
	CFB	570	570	695	695
	OFB	545	545	686	686
	CTR	557	557	699	699
Camellia	skey	23290	23290	15345	15345
	CBC	971	1001	441	441
	CFB	974	974	432	432
	OFB	950	950	393	393
	CTR	960	960	433	433

is no significant difference between the size-optimised and the speed-optimised version in terms of memory and CPU cycles.

Rijndael: Rijndael has a big S-box, and as a result its code memory requirement exceeds 10KB, which is about the current size of the EYES operating system [74]. Among our selection, Rijndael is the only cipher that has key setup for decryption, and we note that this key setup is about four times as slow as that for encryption. Although the key setup code size of Rijndael is 2-3 times that of RC6, the performance is 50-70 times that of RC6.

MISTY1: The code size requirement of MISTY1 is between RC5, RC6 on the low end, and Rijndael on the high end. In terms of performance, it is also between RC5, RC6 and Rijndael. It is perhaps of interest to note that the speed-optimised version actually performs poorer than the size-optimised version (except for key setup). The size-optimised version also provides more than 10% of savings in storage.

KASUMI: Although the key setup of KASUMI is linear [75], it takes more than twice as long as that of MISTY1. Although the code size of KASUMI is 40-50% larger than that of MISTY1, KASUMI has comparable performance to MISTY1 in terms of encryption. Like MISTY1, the size-optimised version of KASUMI interestingly provides better performance than the speed-optimised version.

Camellia: Camellia has the largest code size among our selection of ciphers, but its performance is only worse than Rijndael in our selection. Speed optimisation increases the code size by about 50% compared with the size-optimised version, but the performance at the same time more than doubles.

To conclude our observation, we now discuess the relative rankings in Table 7.

In terms of key setup, MISTY1 is an attractive option with the least data memory requirement and CPU cycles, and modest code memory requirement. Rijndael also achieves very good ranking in terms of data memory and speed. Although RC5-32 and RC6-32 have small code size, they fare badly in the data memory and speed genre. Camellia achieves very low ranking in memory requirements and average ranking in performance. KASUMI is an average performer in all categories.

In terms of encryption/decryption, Rijndael offers the highest speed at the expense of relatively large code size and data memory requirement, but in a way not worse than Camellia. MISTY1 is a solid performer in all categories, never falling below the 3rd place, whereas KASUMI is average in all categories. RC5-32 is the opposite of Rijndael, failing in speed, but gaining in lean code size and RAM footprint. The only advantage of RC6-32 is its compact code size. Camellia with speed optimisation is only less efficient than Rijndael but its code size is about 10 times RC5-32's. RC6-32 appears to be the most energy-consuming cipher on average.

Table 7. Ranking of ciphers by key setup and encryption (CBC/CFB/OFB/CTR).

By key setup:

Rank	Size optimised			Speed optimised		
	Code mem.	Data mem.	Speed	Code mem.	Data mem.	Speed
1	RC5-32	MISTY1	MISTY1	RC6-32	MISTY1	MISTY1
2	KASUMI	Rijndael	Rijndael	KASUMI	Rijndael	Rijndael
3	RC6-32	KASUMI	KASUMI	RC5-32	KASUMI	KASUMI
4	MISTY1	RC6-32	Camellia	MISTY1	RC6-32	Camellia
5	Rijndael	RC5-32	RC5-32	Rijndael	Camellia	RC5-32
6	Camellia	Camellia	RC6-32	Camellia	RC5-32	RC6-32

By encryption (CBC/CFB/OFB/CTR):

Rank	Size optimised			Speed optimised		
	Code mem.	Data mem.	Speed	Code mem.	Data mem.	Speed
1	RC5-32	RC5-32	Rijndael	RC6-32	RC5-32	Rijndael
2	RC6-32	MISTY1	MISTY1	RC5-32	MISTY1	Camellia
3	MISTY1	KASUMI	KASUMI	MISTY1	KASUMI	MISTY1
4	KASUMI	RC6-32	Camellia	KASUMI	RC6-32	RC5-32
5	Rijndael	Rijndael	RC6-32	Rijndael	Rijndael	KASUMI
6	Camellia	Camellia	RC5-32	Camellia	Camellia	RC6-32

5 Conclusion

First about the operation mode to use. Apart from having the highest memory and energy efficiency, OFB has desirable fault-tolerance characteristics, i.e. a ciphertext error affects only the corresponding bit(s) of the plaintext. Therefore in an error-prone environment such as the wireless network, OFB is particularly useful. However in the event of loss of synchronisation, OFB has to rely on an external mechanism to regain synchronisation. This applies also to CTR, but regaining synchronisation is much more efficient for CTR than for OFB, because CTR is parallelisable. In terms of energy-efficiency, CTR is only next to OFB. Although CTR has the highest RAM usage among all the modes, the potentially great savings in resynchronisation justifies the cost in RAM usage. Our conclusion is that the use of CTR is recommended.

On the suitable cipher to use, we will reach our verdict by first ruling out the unlikely candidates, but first we would like to emphasise that the processor we are currently using, MSP430F149 is one of the most high-end in the Texas Instrument's MSP430 series. In other words, a total code memory of 59.7KB and data memory of 2KB is a hard limit in the MSP430 family of processors. For this reason, we first rule out Camellia which occupies a one-third of the total code memory even after size optimisation, even though it has good energy-efficiency for encryption.

The next to rule out is RC6-32, which has poor key agility (the ability to change keys quickly and with a minimum amount of resources) and energy efficiency. Although RC5-32 is faster than RC6-32, its key agility is worse than RC6-32. This rules out RC5-32. KASUMI is only better than MISTY1 in terms of its code size for key setup. So we would consider MISTY1 instead of KASUMI, ruling out KASUMI for further consideration.

Finally the verdict: for maximum energy-efficiency, we recommend Rijndael; whereas for truly constrained environments, we recommend MISTY1. Although MISTY1 is slower than Rijndael, in terms of key setup, MISTY1 uses less memory and CPU cycles than Rijndael does. Furthermore, MISTY1 uses less RAM for encryption, not to mention that the overall code size of MISTY1 is half of Rijndael's. The only drawback of MISTY1 is that it is (believed to be) less secure than Rijndael and it only caters for one key length, but we forsee that there are applications that do not require a security level higher than MISTY1 can provide.

Reviewing some of the proposals in the literature so far, we see that we are disagreeing with Perrig et al. [17] at using RC5, and with Slijepcevic et al. [27] at using RC6. Another aspect to consider is that most embedded processors do not support the variable-bit rotation instruction like ROL of the Intel architecture [76] which greatly improves the performance of RC5. Slijepcevic et al. suggest varying the number of rounds to achieve different levels of security with RC6, but increasing the number of rounds beyond the nominal value to increase the level of security is a speculative approach compared with increasing the key length. That is, using longer keys for higher security is a more appropriate approach. One non-technical reason against the use of RC6 is that the cipher is patented and not royalty-free.

In conclusion, we have presented a detailed benchmark for one of the most important cryptographic primitives in a WSN, i.e. block ciphers. Taking into account the security properties, storage- and energy-efficiency of a set of candidates, we have arrived at a systematically justifiable selection of block ciphers and operation modes. Instead of taking for granted prevalent suggestions, we now have results that we can use not only for convincing ourselves but also other researchers in the area of WSN.

Acknowledgement

This work is partially supported by the EU under the IST-2001-34734 EYES project. The authors would like to thank Paul Havinga and Alexander Kholosha for their constructive remarks.

References

1. Law, Y., Dulman, S., Etalle, S., Havinga, P.: Assessing Security-Critical Energy-Efficient Sensor Networks. Technical Report TR-CTIT-02-18, Centre for Telematics and Information Technology, University of Twente, The Netherlands (2002)

2. Hill, J., Szewczyk, R., Woo, A., Hollar, S., Culler, D.E., Pister, K.S.J.: System architecture directions for networked sensors. In: Architectural Support for Programming Languages and Operating Systems. (2000) 93–104
3. Kling, R.: Intel® Research mote. Network Embedded Systems Technology, Winter 2003 Retreat, January 15-17 (2003) `http://webs.cs.berkeley.edu/retreat-1-03/slides/imote-nest-q103-03-dist.ppt`.
4. Karlof, C., Wagner, D.: Secure routing in wireless sensor networks: Attacks and countermeasures. Ad Hoc Networks (2003)
5. Preneel, B.: Cryptographic primitives for information authentication - state of the art. In Preneel, B., Rijmen, V., eds.: State of the Art in Appplied Cryptography. Volume 1528 of LNCS., Springer-Verlag (1998) 50–105
6. Crosby, S., Wallach, D.: Denial of service via algorithmic complexity attacks. In: 12th USENIX Security Symposium, USENIX Association (2003) To appear.
7. Carman, D., Kruus, P., Matt, B.: Constraints and approaches for distributed sensor network security. Technical Report #00-010, NAI Labs (2000)
8. Rivest, R.: The RC5 Encryption Algorithm. In: Proc. 1994 Leuven Workshop on Fast Software Encryption, Springer-Verlag (1995) 86–96
9. Rivest, R., Robshaw, M., Sidney, R., Yin, Y.: The RC6™Block Cipher. Specification version 1.1 (1998)
10. Daemen, J., Rijmen, V.: AES Proposal: Rijndael (1999)
11. Matsui, M.: New Block Encryption Algorithm MISTY. In Biham, E., ed.: Fast Software Encryption, 4th International Workshop, FSE '97. Volume 1267 of LNCS., Springer-Verlag (1997) 54–68
12. 3rd Generation Partnership Project: Specification of the 3GPP Confidentiality and Integrity Algorithms Document 2: KASUMI Specification. ETSI/SAGE Specification Version: 1.0 (1999)
13. Aoki, K., Ichikawa, T., Kanda, M., Matsui, M., Moriai, S., Nakajima, J., Tokita, T.: Specification of Camellia – A 128-Bit Block Cipher. Specification Version 2.0, Nippon Telegraph and Telephone Corporation and Mitsubishi Electric Corporation (2001)
14. Hachez, G., Koeune, F., Quisquater, J.J.: cAESar results: Implementation of Four AES Candidates on Two Smart Cards. In: 2nd AES Candidate Conference (AES2). (2000)
15. Worley, J., Worley, B., Christian, T., Worley, C.: AES Finalists on PA-RISC and IA-64: Implementations & Performance. In: Proc. 3rd AES Conference (AES3). (2001)
16. Texas Instruments, Inc.: MSP430x13x, MSP430x14x Mixed Signal Microcontroller. Datasheet (2001)
17. Perrig, A., Szewczyk, R., Wen, V., Culler, D., Tygar, J.: SPINS: Security Protocols for Sensor Networks. In: Proceedings of the 7th Ann. Int. Conf. on Mobile Computing and Networking, ACM Press (2001) 189–199
18. Kaliski, B., Yin, Y.: On the Security of the RC5 Encryption Algorithm. Technical Report TR-602, RSA Laboratories (1998)
19. Biryukov, A., Kushilevitz, E.: Improved Cryptanalysis of RC5. In: Advances in Cryptology – EUROCRYPT '98, International Conference on the Theory and Application of Cryptographic Techniques. Volume 1403 of LNCS., Springer-Verlag (1998) 85–99
20. Borst, J., Preneel, B., Vandewalle, J.: Linear Cryptanalysis of RC5 and RC6. In Knudsen, L., ed.: Fast Software Encryption, 6th International Workshop, FSE '99. Volume 1636 of LNCS., Springer-Verlag (1999) 16–30

21. Nyberg, K.: Linear approximations of block ciphers. In: Advances in Cryptology – EUROCRYPT '94, Workshop on the Theory and Application of Cryptographic Techniques. Volume 950 of LNCS., Springer-Verlag (1995) 439–444
22. Shimoyama, T., Takeuchi, K., Hayakawa, J.: Correlation Attack to the Block Cipher RC5 and the Simplified Variants of RC6. In: Proc. 3rd AES Conference (AES3). (2000)
23. Miyaji, A., Nonaka, M., Takii, Y.: Known plaintext correlation attack against RC5. In Preneel, B., ed.: Topics in Cryptology – CT-RSA 2002, The Cryptographers' Track at the RSA Conference 2002. Volume 2271 of LNCS., Springer-Verlag (2002) 131–148
24. Handschuh, H., Heys, H.: A Timing Attack on RC5. In Tavares, S., Meijer, H., eds.: Selected Areas in Cryptography '98, SAC'98. Volume 1556 of LNCS., Springer-Verlag (1998) 306–318
25. Kelsey, J., Schneier, B., Wagner, D., Hall, C.: Side channel cryptanalysis of product ciphers. In: Computer Security (ESORICS'98). Volume 1485 of LNCS., Springer-Verlag (1998) 97–110
26. Nechvatal, J., Barker, E., Bassham, L., Burr, W., Dworkin, M., Foti, J., Roback, E.: Report on the Development of the Advanced Encryption Standard (AES). Technical report, NIST (2000)
27. Slijepcevic, S., Tsiatsis, V., Zimbeck, S., Srivastava, M., Potkonjak, M.: On communication security in wireless ad-hoc sensor networks. In: 11th IEEE Int. Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises. (2002) 139–144
28. Cryptography Research and Evaluation Committee, Information-Technology Promotion Agency, Japan: Analysis of RC6. CRYPTREC 1086 (2001)
29. Knudsen, L., Meier, W.: Correlations in reduced round variants of RC6. In: Fast Software Encryption, 7th International Workshop, FSE 2000. Volume 1978 of LNCS., Springer-Verlag (2000) 94–108
30. Gilbert, H., Handschuh, H., Joux, A., Vaudenay, S.: A Statistical Attack on RC6. In: Fast Software Encryption, 7th International Workshop, FSE 2000. Volume 1978 of LNCS., Springer-Verlag (2000) 64–74
31. Takenaka, M., Shimoyama, T., Koshiba, T.: Theoretical Analysis of "Correlations in RC6". Cryptology ePrint Archive: Report 2002/176 (2002)
32. Takenaka, M., Shimoyama, T., Koshiba, T.: Theoretical Analysis of χ^2 Attack on RC6. In: Proc. 8th Australasian Conference on Information Security and Privacy (ACISP2003). Volume 2727 of LNCS., Springer-Verlag (2003) 142–153
33. T. Shimoyama, M.T., Koshiba, T.: Multiple linear cryptanalysis of a reduced round RC6. In Daemen, J., Rijmen, V., eds.: Fast Software Encryption, 9th International Workshop, FSE 2002. Volume 2365., Springer-Verlag (2002) 76–88
34. Cryptography Research and Evaluation Committee, Information-Technology Promotion Agency, Japan: E-government recommended ciphers list. Web page: `http://www.ipa.go.jp/security/enc/CRYPTREC/fy15/cryptrec20030425_spec01.html` (2003)
35. Cheon, J., Kim, M., Kim, K., J.-Y. Lee, S.W.K.: Improved Impossible Differential Cryptanalysis of Rijndael and Crypton. In Kim, K., ed.: 4th International Conference on Information Security and Cryptology, ICISC 2001. Volume 2288 of LNCS., Springer-Verlag (2002) 39–49
36. Gilbert, H., Minier, M.: A collision attack on seven rounds of Rijndael. In: Proc. 3rd AES Conference (AES3). (2000)

37. Ferguson, N., Kelsey, J., Lucks, S., Schneier, B., Stay, M., Wagner, D., Whiting, D.: Improved Cryptanalysis of Rijndael. In Schneier, B., ed.: Fast Software Encryption, 7th International Workshop, FSE 2000. Volume 1978 of LNCS., Springer-Verlag (2001)
38. ande R. Schroeppel, N.F., Whiting:, D.: A Simple Algebraic Representation of Rijndael. In: Selected Areas in Cryptography, 8th Annual International Workshop, SAC 2001. Volume 2259 of LNCS., Springer-Verlag (2001) 103–111
39. Schneier, B.: AES News. Crypto-gram newsletter, Counterpane Internet Security, Inc. (2002)
40. Courtois, N., Pieprzyk, J.: Cryptanalysis of Block Ciphers with Overdefined Systems of Equations. Cryptology ePrint Archive: Report 2002/044 (2002)
41. Courtois, N., Pieprzyk, J.: Cryptanalysis of Block Ciphers with Overdefined Systems of Equations. In Zheng, Y., ed.: Advances in Cryptology – ASIACRYPT 2002: 8th International Conference on Theory and Application of Cryptology and Information Security. Volume 2501 of LNCS., Springer-Verlag (2002) 267–287
42. Courtois, N., Goubin, L., Meier, W., Tacier, J.D.: Solving underdefined systems of multivariate quadratic equations. In: PKC 2002. Volume 2274 of LNCS., Springer-Verlag (2002) 211–227
43. Courtois, N., Patarin, J.: About the XL Algorithm over GF(2). In Joye, M., ed.: Topics in Cryptology – CT-RSA 2003, The Cryptographers' Track at the RSA Conference 2003. Volume 2612 of LNCS., Springer-Verlag (2003) 141–157
44. Moh, T.: On the Courtois-Pieprzyk's Attack on Rijndael . Web page (2002) `http://www.usdsi.com/aes.html`.
45. Schneier, B.: More on AES Cryptanalysis. Crypto-gram newsletter, Counterpane Internet Security, Inc. (2002)
46. Murphy, S., Robshaw:, M.: Essential Algebraic Structure within the AES. In Yung, M., ed.: Advances in Cryptology – CRYPTO 2002, 22nd Annual International Cryptology Conference. Volume 2442 of LNCS., Springer-Verlag (2002) 1–16
47. Murphy, S., Robshaw, M.: Comments on the Security of the AES and the XSL Technique (2002) `http://www.isg.rhul.ac.uk/~mrobshaw/rijndael/xslnote.pdf`.
48. Coppersmith, D.: Re: Impact of courtois and pieprzyk results. Forum message (2002) `http://aes.nist.gov/aes/`.
49. Fuller, J., Millan, W.: On Linear Redundancy in the AES S-Box. Cryptology ePrint Archive: Report 2002/111 (2002)
50. FILIOL, E.: Plaintext-dependant repetition codes cryptanalysis of block ciphers - the aes case. Cryptology ePrint Archive: Report 2003/003 (2003)
51. Courtois, N., Johnson, R., Junod, P., Pornin, T., Scott, M.: Did Filiol Break AES? Cryptology ePrint Archive: Report 2003/022 (2003)
52. Barkan, E., Biham, E.: In How Many Ways Can You Write Rijndael. In Zheng, Y., ed.: Advances in Cryptology – ASIACRYPT 2002: 8th International Conference on Theory and Application of Cryptology and Information Security. Volume 2501 of LNCS., Springer-Verlag (2002) 160–175
53. Tri Van Le: Novel Cyclic and Algebraic Properties of AES. Cryptology ePrint Archive: Report 2003/108 (2003)
54. Youssef, A., Tavares, S.: On Some Algebraic Structures in the AES Round Function. Cryptology ePrint Archive: Report 2002/144 (2002)
55. Preneel, B., Biryukov, A., Oswald, E., Rompay, B.V., Granboulan, L., Dottax, E., Murphy, S., Dent, A., White, J., Dichtl, M., Pyka, S., Schafheutle, M., Serf, P., Biham, E., Barkan, E., Dunkelman, O., Quisquater, J.J., Ciet, M., Sica, F.,

Knudsen, L., Parker, M., Raddum, H.: NESSIE Security Report. Deliverable D20, NESSIE Consortium (2003)
56. Ohta, H., Matsui, M.: A Description of the MISTY1 Encryption Algorithm. RFC 2994, Network Working Group, IETF (2000)
57. Babbage, S., Frisch, L.: On MISTY1 Higher Order Differential Cryptanalysis. In: 3rd International Conference on Information Security and Cryptology, ICISC 2000. Volume 2015 of LNCS., Springer-Verlag (2001) 22–36
58. Kühn, U.: Cryptanalysis of Reduced-Round MISTY. In: Advances in Cryptology – EUROCRYPT 2001. Volume 2045 of LNCS., Springer-Verlag (2001) 325–339
59. Kühn, U.: Improved Cryptanalysis of MISTY1. In: Fast Software Encryption, 9th International Workshop, FSE 2002. Volume 2365 of LNCS., Springer-Verlag (2002) 61–75
60. Knudsen, L., Wagner, D.: Integral cryptanalysis. In Daemen, J., Rijmen, V., eds.: Fast Software Encryption, 9th International Workshop, FSE 2002. Volume 2365 of LNCS., Springer-Verlag (2002) 112–127
61. Kang, J.S., Shin, S.U., Hong, D., Yi, O.: Provable Security of KASUMI and 3GPP Encryption Mode $f8$. In Boyd, C., ed.: Advances in Cryptology - ASIACRYPT 2001: 7th International Conference on the Theory and Application of Cryptology and Information Security. Volume 2248 of LNCS., Springer-Verlag (2001) 255–271
62. J.-S. Kang and O. Yi and D. Hong and H. Cho: Pseudorandomness of MISTY-Type Transformations and the Block Cipher KASUMI. In Varadharajan, V., Mu, Y., eds.: Proceedings of the 6th Australasian Conference on Information Security and Privacy, ACISP 2001. Volume 2119 of LNCS., Springer-Verlag (2001) 60–73
63. TANAKA, H., ISHII, C., KANEKO, T.: On the strength of KASUMI without FL functions against Higher Order Differential Attack. In: 3rd International Conference on Information Security and Cryptology, ICISC 2000. Volume 2015 of LNCS., Springer-Verlag (2001) 14–21
64. Aoki, K., Ichikawa, T., Kanda, M., Matsui, M., Moriai, S., Nakajima, J., Tokita, T.: Camellia: A 128-Bit Block Cipher Suitable for Multiple Platforms. In Stinson, D., Tavares, S., eds.: Proc. Selected Areas in Cryptography (SAC'00). Number 2012 in LNCS, Springer-Verlag (2001) 39–56
65. Matsui, M., Tokita, T.: MISTY, KASUMI and Camellia Cipher Algorithm. Mitsubishi Electric ADVANCE (Cryptography Edition) **100** (2000) 2–8
66. Y. He and S. Qing: Square Attack on Reduced Camellia Cipher. In Qing, S., Okamoto, T., Zhou, J., eds.: Information and Communications Security: Third International Conference, ICICS 2001. Volume 2229 of LNCS., Springer-Verlag (2001) 238–245
67. Daemen, J., Knudsen, L., Rijmen, V.: The Block Cipher SQUARE. In Biham, E., ed.: Fast Software Encryption, 4th International Workshop, FSE '97. Volume 1267 of LNCS., Springer-Verlag (1997) 149–165
68. M. Sugita and K. Kobara and H. Imai: Security of Reduced Version of the Block Cipher Camellia against Truncated and Impossible Differential Cryptanalysis. In Boyd, C., ed.: Advances in Cryptology - ASIACRYPT 2001: 7th International Conference on the Theory and Application of Cryptology and Information Security. Volume 2248 of LNCS., Springer-Verlag (2001) 193–207
69. Lee, S., Hong, S., Lee, S., Lim, J., Yoon, S.: Truncated Differential Cryptanalysis of Camellia. In Kim, K., ed.: 4th International Conference on Information Security and Cryptology, ICISC 2001. Volume 2288 of LNCS., Springer-Verlag (2002) 32–38
70. Yeom, Y., Park, S., Kim, I.: On the Security of CAMELLIA against the Square Attack. In Daemen, J., Rijmen, V., eds.: Fast Software Encryption, 9th International Workshop, FSE 2002. Volume 2365 of LNCS., Springer-Verlag (2002) 128–142

71. Hatano, Y., Sekine, H., Kaneko, T.: Higher Order Differential Attack of *Camellia*(II). In Nyberg, K., Heys, H., eds.: Selected Areas in Cryptography. 9th Annual Int. Workshop, SAC 2002. Volume 2595 of LNCS., Springer-Verlag (2002) 129–146
72. Schneier, B.: Applied Cryptography: Protocols, Algorithms and Source Code in C. 2nd edn. John Wiley & Sons, Inc. (1996)
73. LASEC: Information leakage in electronic transactions. `http://lasecwww.epfl.ch/birthday.shtml` (2003)
74. Mulder, J.: PEEROS: PreEmptive Eyes Real-Time Operating System. Master's thesis, University of Twente (2003) `http://wwwhome.cs.utwente.nl/~hoesel/EyesOS/PeerOS%20Report-Job%20Mulder.pdf`.
75. Dunkelman, O.: Comparing MISTY1 and KASUMI. NESSIE Public Report NES/DOC/TEC/WP5/029/a, Computer Science Department, Technion (2002)
76. Intel Corporation: Intel Architecture Software Developers Manual Volume 2: Instruction Set Reference. (1997)
77. Chien, P., Wen, V.: CS199 – StrongARM Energy Measurement Report. Online slides: `http://www.cs.berkeley.edu/~vwen/strongarm/slides/cs199.ppt` (1998)
78. Texas Instruments, Inc.: MSP430x1xx Family User's Guide. (2003)

Appendix: Per-Cycle Energy Consumption

Different instructions may take different number of clock cycles, resulting in different energy consumption per instruction. Even different instructions with the same number of clock cycles may consume different amounts of energy, because of the nature of the instruction itself, for example an instruction that accesses the memory would naturally consume more energy than an instruction that accesses the registers. We will however show that the *energy consumed per cycle* does not vary much from instruction to instruction.

Methodology To achieve this, we should ideally measure the energy consumed by each cycle of different instructions. However measuring such energy is difficult without instrumenting the chip, so we measure the current instead. If we fix the voltage V, by measuring the current I, we get the power P. If a cycle is t_c seconds long, and an instruction consists of c cycles, let $e_1, ..., e_c$ be the energy consumed on each cycle, then

$$VI = P = \frac{e_1 + ... + e_c}{ct_c} = \frac{\bar{e}}{t_c} \implies \bar{e} = VIt_c \tag{1}$$

where $\bar{e} = \frac{e_1+...+e_c}{c}$ is the average energy consumed per cycle. Since V and t_c are fixed, by measuring I, we are in fact measuring $\bar{e}$. There is a possibility that $e_i \ll \bar{e}$ and $e_j \gg \bar{e}$ for some $i \neq j$ and yet $\bar{e}$ does not vary much from instruction to instruction, meaning that even if $\bar{e}$ is constant, we cannot claim that "energy per cycle" is constant. However this is not a problem, because every instruction is always executed as a whole, with the energy-lean cycle(s) compensating the energy-consuming cycle(s). Worth mentioning is that this method is consistent with Chien and Wen's [77].

The current is measured when an instruction `xxx` is executed in an infinite loop:

```
Mainloop    xxx <some randomised operands>
            xxx <some other randomised operands>
            ...100 times
            jmp Mainloop
```

Note that in the above template, one `jmp` instruction after every 100 times of the measured instruction does not affect the measurement much, moreover we can measure the current of `jmp` without the influence of other instructions:

```
Mainloop    jmp Label2
Label2      jmp Mainloop
```

Whenever immediate constant operands, offsets, data are involved, they are randomly generated. In fact, all the test programs are generated by a Perl script.

Since there are 7 addressing modes [78], an instruction like `mov.w` can be used in *at least* 7 modes depending on the type of its operands, e.g. '`mov.w R12, 2(R14)`', '`mov.w @R12+, R14`' etc. Fortunately not all modes of the same instruction are generated by the compiler. We only test those modes of the instruction generated by the compiler. To find these in-use modes, we have written a Perl script to parse the assembler code of our block cipher algorithms (generated by the compiler). For example, the only mode used for the instruction `and.b` is '`and.b #C,Rn`', and we only measure this particular mode of `and.b` (where '`#C`' stands for an immediate constant, and '`Rn`' stands for a register).

Our test programs are generally divided into 3 parts: (1) the program, (2) the source data, and (3) the destination data. For example, while the instruction '`mov.w @R12, 2(R14)`' itself resides in the program area, '`@R12`' points to a word in the source data area, whereas '`2(R14)`' points to a word in the destination data area. Referring to Table 8, $I_{\texttt{RAM}}$ refers to the current when the program, source data and destination data are loaded in the RAM; whereas $I_{\texttt{Flash}}$ refers to the current when the program and the source data are loaded in the Flash *but* the destination data in the RAM. The destination data is always loaded in the RAM because they are meant to be overwritten byte-by-byte, while Flash can only be erased one sector at a time. We do not consider cache because there is none in the processor. Logically $I_{\texttt{RAM}}$ is lower than $I_{\texttt{Flash}}$ since accessing the RAM is cheaper, however we will show that the difference is only about 6% of the mean.

Instead of measuring the current consumed by the processor alone, we have measured the current consumed by the entire EYES sensor node. This is acceptable because the measured instructions do not invoke functions on the peripheral circuits, and assuming the leakage current in the peripheral circuits stays constant independent of the measured instructions.

Results There are no entries in Table 8 for '`ret`', '`pop.b Rn`' and '`pop.w Rn`'. Instead, '`ret`' is measured along with '`call #L`' and '`call Rn`'; '`pop.b Rn`' along with '`push.b Rn`'; and '`pop.w Rn`' along with '`push.w Rn`', '`push.w Rn`' and '`push.w X(Rn)`'. This is fair because in real-world applications, a `pop` is always associated with a `push`, so is a `ret` associated with a `call`.

The average current is 2.93mA, with a standard deviation of 0.05. From the table, we can see that the most energy-consuming instructions are '`mov.b @Rn+,Rn`' and '`xor.b @Rn+,Rn`', consuming a current of 3.03mA (when the program and the source data are loaded in the Flash), whereas '`bis.w Rn,Rn`', '`mov.b Rn,Rn`', '`mov.w Rn,Rn`', the rotation instructions, '`swpb Rn`' and '`sxt Rn`' are the cheapest, consuming a current of 2.85mA (when everything is loaded in the RAM). While it is easy to appreciate why the latter instructions elicit the least current, it is interesting to learn that the instruction mode of '`@Rn+,Rn`' draws a higher current than '`@Rn+,X(Rn)`' (although this does *not* mean that the '`@Rn+,Rn`' over 2 cycles, consumes more energy than '`@Rn+,X(Rn)`' over 5 cycles).

All in all, the difference between the largest and the smallest current is only 6% of the mean, and we conclude that $\bar{e}$ is more or less consistent, or in other words it is safe to assume that "energy per cycle" is more or less consistent for our particular hardware platform.

Table 8. Measured currents $I_{\texttt{Flash}}$ and $I_{\texttt{RAM}}$ for each instruction in mA (V=2.994V, t_c=0.22μs, see text for the definition of $I_{\texttt{Flash}}$ and $I_{\texttt{RAM}}$).

Instruction	c	$I_{\texttt{Flash}}$	$I_{\texttt{RAM}}$
`add.b #C,Rn`	2	2.98	2.89
`add.b Rn,Rn`	1	2.95	2.87
`add.w #C,Rn`	2	2.99	2.91
`add.w #C,X(Rn)`	5	2.99	2.94
`add.w Rn,Rn`	1	2.96	2.87
`addc.w #C,Rn`	1	2.99	2.90
`addc.w #C,X(Rn)`	5	2.97	2.90
`addc.w X(Rn),Rn`	3	2.99	2.90
`addc.w X(Rn),X(Rn)`	6	2.99	2.90
`and.b #C,Rn`	1	2.99	2.89
`and.w #C,Rn`	1	2.99	2.90
`and.w #C,X(Rn)`	5	2.97	2.90
`and.w @Rn,Rn`	2	2.99	2.90
`and.w X(Rn),Rn`	5	3.00	2.90
`bis.w @Rn,Rn`	2	2.98	2.89
`bis.w Rn,Rn`	1	2.93	2.85
`bis.w X(Rn),Rn`	3	2.98	2.89
`bit.b #C,X(Rn)`	5	2.96	2.88
`bit.w #C,Rn`	2	2.98	2.90
`bit.w #C,X(Rn)`	5	2.96	2.89
`br #L`	3	2.96	2.87
`call #L`	5	2.95	2.88
`call Rn`	4	2.96	2.90
`clrc`	1	2.97	2.92
`cmp.w #C,Rn`	2	3.00	2.91
`cmp.w #C,X(Rn)`	5	2.98	2.90
`cmp.w Rn,Rn`	1	2.97	2.87
`cmp.w Rn,X(Rn)`	4	3.00	2.91
`cmp.w X(Rn),Rn`	3	3.00	2.91
`cmp.w X(Rn),X(Rn)`	6	2.99	2.90
`jc L`	2	2.96	2.89
`jeq L`	2	2.96	2.89
`jge L`	2	2.97	2.89
`jl L`	2	2.96	2.89
`jmp L`	2	2.97	2.89
`jnc L`	2	2.97	2.89
`jne L`	2	2.97	2.89
`mov.b #C,Rn`	2	2.98	2.89
`mov.b #C,X(Rn)`	5	2.97	2.89
`mov.b &L,Rn`	3	3.00	2.89
`mov.b @Rn,Rn`	2	3.00	2.89
`mov.b @Rn,X(Rn)`	5	3.00	2.90
`mov.b @Rn+,Rn`	2	3.03	2.91
`mov.b @Rn+,X(Rn)`	5	2.99	2.91
`mov.b Rn,Rn`	1	2.94	2.85
`mov.b Rn,X(Rn)`	4	2.97	2.89
`mov.b X(Rn),Rn`	3	2.99	2.89

Instruction	c	$I_{\texttt{Flash}}$	$I_{\texttt{RAM}}$
`mov.b X(Rn),X(Rn)`	6	2.99	2.89
`mov.w #C,Rn`	2	2.98	2.89
`mov.w #C,X(Rn)`	5	2.98	2.90
`mov.w @Rn,Rn`	2	2.99	2.89
`mov.w @Rn,X(Rn)`	5	2.98	2.90
`mov.w Rn,Rn`	1	2.93	2.85
`mov.w Rn,X(Rn)`	4	2.97	2.89
`mov.w X(Rn),Rn`	3	2.99	2.89
`mov.w X(Rn),X(Rn)`	6	2.98	2.90
`push.b Rn`	3	2.99	2.91
`push.w #C`	4	2.97	2.89
`push.w Rn`	3	3.00	2.91
`push.w X(Rn)`	5	2.98	2.90
`rla.b Rn`	1	2.93	2.85
`rla.w Rn`	1	2.94	2.85
`rlc.b Rn`	1	2.94	2.85
`rlc.w Rn`	1	2.94	2.85
`rra.b Rn`	1	2.93	2.85
`rra.w Rn`	1	2.93	2.85
`rrc.b Rn`	1	2.93	2.85
`rrc.w Rn`	1	2.93	2.85
`sub.b Rn,Rn`	1	2.95	2.86
`sub.w #C,Rn`	2	3.00	2.91
`sub.w @Rn,Rn`	2	3.02	2.89
`sub.w Rn,Rn`	1	2.95	2.87
`sub.w X(Rn),Rn`	3	3.01	2.90
`subc.b #C,Rn`	2	2.99	2.89
`subc.b Rn,Rn`	1	2.95	2.86
`subc.w Rn,Rn`	1	2.95	2.87
`subc.w X(Rn),Rn`	3	3.00	2.91
`swpb Rn`	1	2.94	2.85
`sxt Rn`	1	2.93	2.85
`xor.b #C,Rn`	2	2.99	2.90
`xor.b @Rn,Rn`	2	3.00	2.90
`xor.b @Rn,X(Rn)`	5	2.99	2.91
`xor.b @Rn+,Rn`	2	3.03	2.91
`xor.b Rn,Rn`	1	2.94	2.86
`xor.b Rn,X(Rn)`	4	2.98	2.92
`xor.b X(Rn),Rn`	3	3.00	2.91
`xor.b X(Rn),X(Rn)`	6	2.99	2.90
`xor.w #C,Rn`	2	3.00	2.90
`xor.w #C,X(Rn)`	5	2.99	2.91
`xor.w @Rn,Rn`	2	3.00	2.91
`xor.w Rn,Rn`	1	2.95	2.87
`xor.w Rn,X(Rn)`	4	3.00	2.91
`xor.w X(Rn),Rn`	3	3.00	2.90