Analytical Comparison of Cryptographic Techniques for Resource-Constrained Wireless Security

M. Razvi Doomun and KMS Soyjaudah
(Corresponding author: M. Razvi Doomun)

Faculty of Engineering, University of Mauritius
Reduit Campus, Mauritius
(Email: r.doomun@uom.ac.mu)

(Received May 29, 2007; revised and accepted Feb. 2 & Mar. 3, 2008)

Abstract

With the widespread growth in applications for resource-limited Wireless Sensor Networks (WSN), the need for reliable and efficient security mechanisms for them has increased manifold but its implementation is a non-trivial task. Limitations in processing speed, battery power, bandwidth and memory constrain the applicability of existing cryptography Algorithms for WSNs. In this paper, two potential block ciphers, namely the RC5 and AES-Rijndael, are analyzed and their suitability for resource-limited wireless network security are compared based on performance criteria such as computational complexity overhead and energy consumption. Using simulation tests and analytical models, we provide important analysis and considerations on practical feasibility of these cryptographic Algorithms in sensor networks to help designers predict security performance under a set of constraints for WSNs.

Keywords: Block cipher analysis, computational overhead, energy-efficient encryption, wireless sensor network, wireless security

1 Introduction

Resource-efficient cryptographic Algorithms are becoming a pre-requisite for security services in various network environments, such as Wireless Sensor Networks (WSN), in which low computational cost and low power consumptions are key performance requirements. Though, security in WSNs share many characteristics with traditional wireless ad-hoc networks, there are fundamental differences between the make-up and aims of the two types of networks. Although WSNs are economically viable, but at the same time their operational requirements prevent the direct application of existing highly reliable traditional wireless security techniques [11]. Traditional wireless network security standards are not well-suited for sensor networks as they are more demanding in memory, energy and are computationally intensive. An important aspect is that the sensor nodes in WSNs are unattended, deployed in hostile environment, have unreplenishing energy and network topology is unknown making them prone to different types of malicious attacks like eavesdropping, masquerading, traffic-analysis, etc. These devices are very limited in their energy (Typically 20-30 Joules of initial energy), computation processor, bandwidth and communication capabilities that impose strong restrictions on the processing capabilities and available memory for security mechanism executions. Thus, providing high degree of security in terms of data privacy and authentication is a challenging task and is at the same time of high priority. A number of security techniques have been proposed in recent years to cope with battery characteristics for designing better energy-efficient security protocol [8, 11, 17]. It has been argued that, using asymmetric cryptography on highly resource-constrained devices is often not feasible due to delay, energy and memory constraints. For comparison, an implementation of symmetric cryptography on an 8-bit micro-controller uses 7150 bytes of program memory [8], in contrast to 30 KBytes of program memory for the smallest available implementation of asymmetric cryptography [5]. As a result, symmetric cryptography is the choice for applications that cannot afford computational complexity. Efficient cryptographic Algorithms with optimized implementations, in terms of energy consumption and computational overhead, are imperative to maximize the battery lifetime of sensor nodes.

The aim of this paper is to present a methodology for the evaluation of the complexity or computational cost and energy efficiency of two block ciphers, Advance Encryption Standard (AES)- Rijndael and RC5, that have been published as potentially suitable for WSN security. Generally, the energy consumption by cryptographic Algorithms executed on a given microprocessor is expected to be proportional to the number of clock cycles needed by the processor for their execution. The main contributions of this work can be summarized as follows: (1) Compare the number of basic processing operations required by each encryption Algorithm and, analyze their capabilities with other similar well-established results and performance models. (2) Provide a simulation framework and mathematical model to assess the processing overhead and energy consumption of the cryptographic algorithms. (3) Evaluate analytically simulation results to understand requirements of optimized cryptosystems for resource-limited wireless device and the design adaptive encryption for future energy-efficient WSN security The rest of the paper is structured as follows: Section 2 presents the related work. Section 3 describes the RC5 and AES-Rijndael block cipher algorithms. Section 4 introduces our analytical models of cryptographic algorithms complexity in terms of primitive operations. In Section 5, the experimental methodology is discussed and, performance results and evaluation of cryptographic algorithms are presented in Section 6. Finally, Section 7 ends up with final considerations and future works.

2 Related Work

Security designers are facing the critical task of guarantying the security of increasingly more complex wireless network systems while dealing with very tight constraints. The ingredients of a security protocol are cryptographic algorithms, which are selected based on the security objectives that are to be achieved by the protocol. Many battery-powered wireless systems are constrained by the environments they operate in and the resources they possess, hence security designers should not address security considerations from a functional perspective only. One of the current foremost challenges is the mismatch between the energy and performance requirements of security processing, and the available battery and processor processing. These challenges have to be addressed to provide an efficient and reliable wireless security solution and they have been emphasized by Ravi *et al.* in [13] as proposed below. The **processing gap** highlights that current wireless network security system architectures are not capable of keeping up with the computational demands of security processing. The **battery gap** emphasizes that the current energy consumption overheads of supporting security on battery constrained wireless systems are very high. The **flexibility** imposes that a mobile wireless system is often required to execute multiple heterogeneous security protocols and standards. The **tamper resistance** emphasizes that secure mobile wireless systems are facing an increasing number of attacks from physical to software attacks. Side-channel attacks also represent an important threat for these systems. The **assurance gap** is related to reliability and stresses the fact that secure systems must continue to operate reliably despite attacks from intelligent adversaries who intentionally seek out undesirable failure modes.

However, security should not be overdone and efficient encryption algorithm' refers to one which requires little storage, makes optimal use of hardware resources and consumes less energy. The cost of encryption and decryption depend on a number of parameters: the size of the plaintext and ciphertext, respectively; the implementation complexity of the algorithm; the cipher mode adopted; and the key scheduling. Specifically, key length is important and the longer the key the higher the encryption time. Also, the cost for decryption depends on the cost of the checks needed to accept the decryption. In particular there are two main concerns about the implementation of AES-Rijndael on sensor nodes claimed in the literature [7, 15, 19]: (1) It is often considered too slow and (2) It requires more space in memory. In fact, the baseline version of AES-Rijndael uses over 800 bytes memory space for lookup tables, which is a high overhead on constrained WSN environment. However some researchers do believe that Rinjdael can be efficiently implemented on WSN platform and Vitaletti and Palombizio [19] presented an improved implementation of AES-Rijndael for wireless sensor networks running on Eyes sensor nodes. Their implementation of AES-Rijndael is smaller, from about 1/3 to 1/5 of the size of previous implementations and shows reasonable speed performance (slower than RC5 by a factor 2). Xiao *et al.* [9] analyzed the security overhead of AES-CCMP in IEEE 802.15.4 specification. The authors observed that processing cycles per block increases as key length increases, payload increases or MIPS decreases. AES-CCMP Decryption has much higher processing cycles than encryption and the increase of processing cycles over the key length and the payload size tends to be linear. Karlof *et al.* also designed a chain-block-cipher (CBC) security mechanism called TinySec [8], and argued why CBC is the most appropriate encryption scheme for sensor networks. Initially they found AES-Rijndael to be slow for sensor networks and their work established that RC5 is one of the most appropriate encryption methods for software implementation on embedded micro-controllers. The use of CBC mode with blocks of size 8-byte results in ciphertexts hat are multiples of 8 bytes causing message expansion hence increasing power consumption. Karlof *et al.* proposed the use of ciphertext stealing technique [8] to ensure that the ciphertext and plaintext are of same length. Encrypting data payloads of less than 8 bytes generates ciphertext of 8 bytes since it requires as a minimum one block of ciphertext. Nonetheless, the ciphertext stealing process involves additional computation, which degrades the sensors power life span. In [9], Law *et al.* evaluated the performance and energy characteristics of symmetric key algorithms on sensor node. Using the benchmark parameters, code, data memory and CPU cycles, the simulation results showed that AES-Rijndael is suitable for high security and energy-efficiency. However, the evaluation results in [10] which provide a comparative study on performance of symmetric key cryptography in sensor networks, disagreed with the work in [18] in which RC5 was selected as the encryption/decryption scheme. In the paper [2], the authors assess the feasibility of different encryption schemes for a range of embedded architectures, i.e. from low-end 8-bit width to high-end 32-bit width processors, and its impact on the performance of encryption. For a sequence initialization-encryption-decryption of plaintext size 16 bytes using RC5, 75% of execution time accounts for initialization step, 13% of execution time for encryption and 13% of execution time for decryption. The initialization overhead is significant for most encryption algorithms, in particular for small plaintext size. The AES encryption of a single 128-bit block of data using a 128-bit key requires approximately 2109 clock cycles on the Strong ARM platform, whereas the decryption of a single block can take 2367 cycles, and the key scheduling would require some 670 clock cycles [4]. Given a clock frequency of 133 MHz, the corresponding energy values are typically 5.7 μJ (for encryption), 6.4 μJ (decryption), and 1.8 μJ (key scheduling). An analysis of various ciphers is presented in [10, 20] and a summary of RC5 and AES-Rijndael qualitative performance comparison is shown in Table 1.

By examining the CPU cycles, authors in [10] conclude that AES-Rijndael is the most energy efficient cipher as its fewer CPU cycles is equivalent in less energy used. Conversely, RC5 has a noticeable advantage in both data memory and code memory at the expense of speed. The authors in [9] have developed some well-known cryptographic algorithm, including two implementations of RC5, in search for the best compromise in security and energy efficiency, on a typical sensor node. The reference RC5 implementation [14] and Schneier's RC5 implementation [15] with different coding style were compared to assess code size and speed optimization impact on energy used per byte.

Generally, it is agreed that AES-Rijndael has strong cryptographic properties, but at the cost of more memory and processing requirements compared to RC5. Furthermore, AES has higher mean values for energy-latency product, which means that it consumes the most energy per byte during encryption and decryption [6]. Most prior work in the literature has focused on efficient AES cipher implementations with throughput as main criterion. However, AES-Rijndael implementations in small and constrained environments require additional factors to be accounted for, such as complexity, memory and energy supply. In Potlapally et al. [12], the result for 128-bit AES encryption claimed is 151 nJ/bit for encryption energy usage. Execution time and energy consumption of key set up and encryption can differ in a factor of 1000. Hodjat *et al.* state in [7] that the AES encryption itself takes only 11 cycles, but the complete program with loading the data and key, AES encryption, and returning the result back to the software routine takes a total of 704 cycles. Thus, AES-Rijndael decryption was found to consume approximately up to 20-30% more energy than encryption [7]. Nevertheless, its performance is very good and seems likely to remain so on many 8-bit or 32-bit processors since it uses only efficient and commonly available instructions. Consequently, the performance of any symmetric key cryptography is essentially dependent on two factors, namely embedded data bus width and instruction set [21]. Some encryption algorithms are adapted to 32-bit word arithmetic, while most embedded processors generally use 8 or 16 bit data bus width.

In this section we have presented a detail literature review of the overhead computational costs to guarantee the cryptographic flexibility of secure wireless system. The need for studying AES-Rijndael and RC5 are also discussed. Based on all the related research, security performance and energy are closely related and have to be considered for the design of efficient and optimized wireless security mechanisms. Several questions are tackled in this research work: What are the performance and the potential cryptographic mechanisms? What are the overhead and computational costs to provide flexible and reliable wireless security? What are the relationships between the cryptographic primitive operations and the performance of the wireless security system? The complexity analysis of AES-Rijndael and RC5 in this study explores the impact of various parameters on the computational/processing cost, hence energy consumption, for secure communication in resource-constrained environment. Based on our analysis, we also discuss optimization opportunities for efficient implementation of the cryptographic algorithms.

3 Description of Block Ciphers

In this section, we provide an overview of RC5 and AES-Rijndael cryptographic algorithms.

3.1 Overview of RC5

RC5 is a fast symmetric block cipher with a two-word input block size plaintext and output block ciphertext. RC5 implementation with a 64-bit data block and 64-bit key uses the XOR, addition and rotation (circular shift) operations. Larger number of encryption rounds N_r presumably provides an increased level of security but at the expense of speed and more memory. RC5 uses an "expandable key table", **S**, that is derived from the secret key **K** and the size **t** of Table **S** also depends on N_r, with S has $\mathbf{t} = \mathbf{2}(N_r + \mathbf{1})$. The strength of RC5 depends heavily on the cryptographic properties of data-dependent rotations. The description of the encryption algorithm is given in the pseudo-code that follows. We assume that the input block is given in two w-bit registers A and B, and that the output is also placed in the registers A and B. During encryption, the plaintext is split into two 32-bit words and a series of XOR ($\oplus$), rotation ($<<<$) and addition ($+$) operations are performed on these words in conjunction with the array S to generate the ciphertext.

Table 1: Summary of AES and RC5 cipher performance

Performance by Key Setup						
RANK	**Size Optimised**			**Speed Optimised**		
	Code mem.	**Data mem.**	**Speed**	**Code mem.**	**Data mem.**	**Speed**
1	RC5	Rijndael	Rijndael	RC5	Rijndael	Rijndael
2	Rijndael	RC5	RC5	Rijndael	RC5	RC5
Performance by Encryption						
RANK	**Size Optimised**			**Speed Optimised**		
	Code mem.	**Data mem.**	**Speed**	**Code mem.**	**Data mem.**	**Speed**
1	RC5	RC5	Rijndael	RC5	RC5	Rijndael
2	Rijndael	Rijndael	RC5	Rijndael	Rijndael	RC5

The decryption process is similar and involves the above operations in a different order. The encryption routine is:

```
        A = A + S[0];
        B = B + S[1];
    for i = 1 to N_r do
        A = ((A ⊕ B) <<<B) + S[2i];
        B = ((B ⊕ A) <<< A) + S[2i + 1];
```

The decryption routine is easily derived from the encryption routine, as follows:

```
    for i = N_r downto 1 do
        B = ((B – S[2i+1] ) >>> A) ⊕ A;
        A = ((A – S[2i] ) >>> B) ⊕ B;
        B = B – S[1]; A = A – S[0];
```

RC5 is a parameterized algorithm, with a variable block size, a variable number of rounds, and a variable-length secret key. This provides good flexibility in both the performance characteristics and the level of security. RC5 is simple and easy to implement, and also more amenable to analysis than many other block ciphers. Hence, RC5 block cipher offers a computationally inexpensive way of providing secure encryption.

3.2 Overview of AES – Rijndael

The Advanced Encryption Standard (AES- Rijndael) is a symmetric-key cipher with a simple and elegant algebraic structure, in which both the sender and the receiver use a single key for encryption and decryption. The data block length is fixed to be 128 bits, while the key length can be 128, 192, or 256 bits and the number of rounds $\mathbf{N_r}$ is 10, 12 or 14 respectively. The 128-bit data block is divided into 16 bytes that are mapped to a 4×4 array called the **State**, and all the internal operations of the Rijndael algorithm are performed on the State. In the encryption of the AES-Rijndael algorithm, each round except the final round consists of four iterative transformations: the SubBytes, the ShiftRows, the MixColumns, and the AddRoundKeys, while the final round does not have the MixColumns transformation. The data are placed in an $\mathbf{4{\times}N_b}$ block length array of elements of GF(2^8) which is Galois Field defined by the irreductible polynomial $x^8 + x^4 + x^3 + x + 1$. The encryption of each block of data involves an initialization phase, $(\mathbf{N_r}\textbf{-1})$ iterations of the basic encryption processing and a finalization round: **SubBytes** is a non-linear byte substitution, operating on each of the state bytes independently, composed by the multiplicative inverse in GF(2^8) (0 mapped onto itself) and a fixed affine transformation over GF(2^8); **ShiftRows** cyclically shifts the elements of the i^{th} row of the state $\mathbf{C_i}$ elements to the right, where $\mathbf{C_i}$ are fixed constants; In **MixColumns** the columns of the state are considered as polynomials over GF(2^8) and multiplied modulo $x^4 + 1$ by a fixed polynomial; and **AddRoundKey** is a XOR of the key (after the scheduling) with the array.

The key schedule for AES-Rijndael is a simple expansion using XOR and cyclic shifts, and consists of two components: **KeyExpansion** and round key selection. The application in the scheduling scheme of SubBytes ensures the non-linearity, without adding much more space requirements on an 8-bit processor. KeyExpansion depends on the value of $\mathbf{N_k}$(key length/32).

4 Cipher Analysis

In this section we model how the structure and functional primitives of cryptographic algorithms relate to the computational load, execution time and hence the energy consumption of cipher.

4.1 Computational Evaluation

In [3, 22, 23], the approach used to evaluate a block cipher computational complexity and energy-consumption performance is based on the number of basic or primitive operations required in the execution of the algorithm. All involved algorithmic transformations can be condensed to a set of logical operations: bytewise-AND, bytewise-OR, and shifts of bytes. For a simple bitwise-XOR it is considered as the sum of 2 bitwise-ANDs and 1 bitwise-OR.

Likewise, a circular shift (rotate) operation of an 8-bit word by $\boldsymbol{n}$ positions is taken as a bytewise-OR and 8 shifts.

4.2 AES-Rijndael computational analysis

AddRoundKey() Transformation: In the AddRoundKey() transformation, a Round Key is added to the State by a simple bitwise XOR operation. This is implemented with $\mathbf{8N_b}$ bytewise-ANDs and $4N_b$ bytewise-ORs, where is$\mathbf{N_b}$ = block length/32.

SubBytes() Transformation: The SubBytes() transformation is a non-linear substitution that operates independently on each byte of the State using a substitution table (S-box). The transformations are: multiplicative inverse in GF(2^8) and an affine mapping/inverse affine mapping over GF(2^8) for the encryption/decryption process. The ByteSub transformation can be realised using 16 look-up tables with 8bit-input/8bit-output. In this case, it is excluded in the complexity evaluation, but will have an impact on memory cost. But the S-box can also be implemented by mathematical calculations, then each SubByte operation incurs $3\mathbf{N_b}$ bytewise-ANDs and $2\mathbf{N_b}$ bytewise-ORs.

ShiftRows() Transformation: In the ShiftRows() transformation, the bytes in the last three rows of the State are cyclically shifted over different numbers of bytes (offsets). The first row, Row 0, is not shifted. The Shiftrow transformation can be implemented using cyclic left shift operation. Each ShiftRows consists of $3\mathbf{N_b}$ shifts of bytes and $3\mathbf{N_b}$ bytewise-ORs.

MixColumns() Transformation: The MixColumns() transformation operates on the State column-by-column, treating each column as a four-term polynomial. The columns are considered as polynomials over GF(2^8) and multiplied modulo $x^4 + 1$ with a fixed polynomial $\{03\}x^3 + \{01\}x^2 + \{01\}x^1 + \{02\}$ to given elements of a new $4 \times \mathbf{N_b}$ array. MixColumns transformation costs for every element 4 bytewise-XORs, 6-bitwise-XORs, 2 byteswise-ORs and 16 shifts, i.e. it can be implemented for each round operation with $19\mathbf{N_b}$ bytewise-XORs, $8\mathbf{N_b}$ bytewise-ORs and $64\mathbf{N_b}$ shifts; or $38\mathbf{N_b}$ bytewise-ANDs, $27\mathbf{N_b}$ bytewise-ORs and $64\mathbf{N_b}$ shifts. Hence, the number of operations required to execute **one** normal round (i.e. SubBytes (negligible), ShiftRows, Mix-Columns and AddRoundKey) are $46\mathbf{N_b}$ bytewise-ANDs, $31\mathbf{N_b} + 12$ bytewise-ORs and $64\mathbf{N_b} + 96$ binary shifts. The last $\mathbf{N_r}$ round transformation does not have the Mix-Columns step. Its equivalent implementation is with $8\mathbf{N_b}$ bytewise-ANDs, $7\mathbf{N_b}$ bytewise-ORs and $3\mathbf{N_b}$ shifts of bytes.

(I) SubBytes look-up table neglecting SubBytes operation cost

Neglecting SubBytes in complexity model, the computational effort required for AES-Rijndael encryption of one block of data is a function of the block size, the key size, and the number of processing cycles required for performing basic operations bytewise-AND ($\mathbf{T_a}$), bytewise-OR ($\mathbf{T_o}$), and bytewise shift ($\mathbf{T_s}$) and expressed in general terms as:

$$\begin{aligned} & T_{AES-ENCRYPT} \\ = \; & (46\mathbf{N_b N_r} - 30\mathbf{N_b})\mathbf{T_a} + [31\mathbf{N_b N_r} + 12(\mathbf{N_r} - 1) \\ & -20\mathbf{N_b}]\mathbf{T_o} + [64\mathbf{N_b N_r} + 96(\mathbf{N_r} - 1) \\ & -61\mathbf{N_b}]\mathbf{T_s}. \end{aligned}$$

A critical limitation of AES-Rijndael is related to its decryption because the cipher and its inverse make use of partially different code. The decryption code has InvMixColumns operation, which uses a transformation with another polynomial, $0\text{B}x^3 + 0\text{D}x^2 + 09x + 0\text{E}$. This leads to an additional complexity for decryption as multiplication by bigger coefficients costs much more. Overall, InvMixColumns transformation needs $134\mathbf{N_b}$ bytewise-ANDs, $99\mathbf{N_b}$ bytewise-ORs and $32\mathbf{N_b}$ shifts of bytes to be implemented. Hence, the difference in computation between one InvMixColumn and MixColumn operation is $[96\mathbf{N_b T_a} + 72\mathbf{N_b T_o} - 32\mathbf{N_b T_s}]$. Therefore, the total number of processing cycles in computational effort required for AES-Rijndael decryption of **one block of data** is given by:

$$\begin{aligned} & T_{AES-DECRYPT} \\ = \; & T_{AES-ENCRYPT} + \{[96\mathbf{N_b T_a} + 72\mathbf{N_b T_o} \\ & -32\mathbf{N_b T_s}] \times (\mathbf{N_r} - 1)\}. \end{aligned}$$

Table 2 shows the number basic operations AND, OR and SHIFT calculated for Encryption, Decryption and key Expansion in AES-Rinjdael with SubBytes look-up table.

If $\mathbf{S_d}$ is the size of an unencrypted user data packet in bits, i.e. plaintext, the number of operations required to do the encryption, $\mathbf{OP_{AES}(S_d)}$:

$$\mathbf{OP_{AES}(S_d)} = T_{AES-ENCRYPT} \times \lceil \mathbf{S_d}/blocksize \rceil,$$

where $\lceil \; \rceil$ denotes ceil function.

Assuming, a processor executes $\mathbf{C_p}$ Millions Instructions Per Second (MIPS), and $TIME_{AES}(\mathbf{S_d}, \mathbf{C_p})$ refer to time required by the processor for encrypting one user packet of length $\mathbf{S_d}$ bits with AES-Rijndael. Then,

$$\begin{aligned} TIME_{AES}(\mathbf{S_d}, \mathbf{C_p}) \; = \; & T_{\text{AES-ENCRYPT}} \\ & \times \lceil \mathbf{S_d}/blocksize \rceil / \mathbf{C_p}. \end{aligned}$$

(II) For SubBytes with mathematical operation

$$\begin{aligned} & Tm_{AES-ENCRYPT} \\ = \; & (46\mathbf{N_b N_r} - 30\mathbf{N_b T_a}) + [31\mathbf{N_b N_r} + 12(\mathbf{N_r} - 1) \\ & -20\mathbf{N_b}]\mathbf{T_o} + [64\mathbf{N_b N_r} + 96(\mathbf{N_r} - 1) \\ & -61\mathbf{N_b}]\mathbf{T_s} + [3\mathbf{N_b T_a} + 2\mathbf{N_b T_o}](\mathbf{N_r} - 1). \end{aligned}$$

And

$$\begin{aligned} & Tm_{AES-DECRYPT} \\ = \; & Tm_{AES-ENCRYPT} + [96\mathbf{N_b T_a} + 72\mathbf{N_b T_o} \\ & -32\mathbf{N_b T_s}] \times (\mathbf{N_r} - 1). \end{aligned}$$

Table 2: Rijndael complexity with subByte lookup table

Key size/	Rijndael Encryption				Rijndael Decryption		
block size	OR	AND	Shift (Bytes)		OR	AND	Shift (Bytes)
128/128	1268	1720	408		3860	5176	1272
128/192	1540	2088	496		4708	6312	1552
128/256	1812	2456	584		5556	7448	1832
192/128	2310	3132	744		7062	9468	2328
192/192	2310	3132	744		7062	9468	2328
192/256	2718	3684	876		8334	11172	2748
256/128	3624	4912	1168		11112	14896	3664
256/192	3624	4912	1168		11112	14896	3664
256/256	3624	4912	1168		11112	14896	3664

Table 3 shows the number basic operations AND, OR and SHIFT calculated for Encryption, Decryption and key Expansion in AES-Rinjdael with SubBytes computations.

4.3 RC5 Computational Analysis

RC5 cipher implementation uses three basic operations (and their inverse for decryption): namely, modulo $\mathbf{2^w}$ addition (or its inverse, subtraction), bitwise XOR of words and left-rotation (or its inverse operation, right-rotation). By examining the operations in RC5 algorithm, we determine the number of processing cycles required to perform RC5 encryption and decryption using primitive operations bitwise-AND ($\mathbf{T_a}$), bitwise-OR ($\mathbf{T_o}$) and bitwise circular shift or rotate ($\mathbf{T_s}$).

$\mathrm{T_{RC5-ENCRYPT}} = 2\ (w)\ \mathbf{T_a} + [2(2w\mathbf{T_a} + w\mathbf{T_o}) + 2w\mathbf{T_s} + 2w\mathbf{T_a}] \times \mathbf{N_r}$

TRC5-DECRYPT $= 2\ (w)\ \mathbf{T_a} + [2(2w\mathbf{T_a} + w\mathbf{T_o}) + 2w\mathbf{T_s} + 2w\mathbf{T_a}] \times \mathbf{N_r}$

Neglecting required constant values computations; the number of processing cycles needed to initialize the array S and mixing the secret key for key expansion is expressed as:

TKEY-X $= \{[2(2w\mathbf{T_a} + w\mathbf{T_o}) + 3\mathbf{T_s} + 2(2w\mathbf{T_a} + w\mathbf{T_o}) + (2w\mathbf{T_a} + w\mathbf{T_o}) + w\mathbf{T_s} + 2\mathbf{T_a}] \times 6(\mathbf{N_r} + 1)\} + [2(\mathbf{N_r} + 1)\ (2\mathbf{T_a} + \mathbf{T_o})]$.

Using same notations as for AES computational analysis, $\mathbf{OP_{RC5}(S_d)}$ = TRC5-ENCRYPT × $\lceil \mathbf{S_d}/\text{block size} \rceil$, where $\lceil\ \rceil$ denotes ceil function.

TIME-RC5$(\mathbf{S_d}, \mathbf{C_p}) = \mathrm{T_{RC5-ENCRYPT}} \times \lceil \mathbf{S_d}/\text{block size} \rceil / \mathbf{C_p}$.

Table 4 summarizes the number of basic AND, OR and SHIFT operations in RC5. For comparison, the cryptographic algorithms analysis in [2] has remarkably high execution times, e.g. the execution of a single AES operation on an array of 1kByte lasted approximately 1.67s. Performing adequate security operations for sensor communication is very expensive since multiple encryption/decryption operations are needed for data aggregation or in-operation network. An approximate performance model of execution time of encryption algorithm is defined in [2] as:

$\mathbf{T_{exec}} = \{\boldsymbol{A} + \boldsymbol{B} * \textbf{msg\_}\textbf{\textit{length}} \ / \ \textbf{\textit{block\_size}}\} / \{\textbf{\textit{processor\_freq}} * \textbf{\textit{bus\_width}}\}$

Where $\lceil\ \rceil$ is the ceiling function, ***msg_length*** is the size of the plaintext in bytes, ***processor_freq*** and ***bus_width*** are the frequency and bus width of the microcontroller, respectively. ***Block_size*** is the size of the blocks in the algorithm. Parameter ***A*** includes all the initialization overheads while ***B*** captures the time spent in operations repeated for each block and Table 5 gives typical values for A and B.

From this analytical model, RC5 is faster compared to AES-Rijndael and therefore more energy-efficient under memory constraints for both encryption and decryption, but it suffers from a relatively costly key expansion. Hence, RC5 is a potential candidate for encryption of large amounts of data since in this case the costly key expansion does not fall into account. Conversely, AES-Rijndael will normally be preferred for the encryption of small amounts of data or medium-sized messages such as in intermittent communication and has a faster key expansion.

5 Experimental Methodology

Evaluations of security architectures are usually based on analytical modelling and performance evaluations are usually based on simulation models.

5.1 Experimental Framework

The simulation framework to test efficiency of encryption algorithm consists of measuring the performance metrics such as execution time, number of instructions executed (type of operations involved in the algorithm), and energy usage by the encryption algorithm. Parameters of the algorithm that are varied include key length, word size and number of rounds. Both cryptographic algorithm, RC5 and AES are run on different hardware platforms such as laptops and PDAs. Using the Advanced Configuration and Power Interface (operating on Linux OS), the voltage

Table 3: Rijndael Complexity with SubBytes mathematical operation

Key size / block size	**Encrypt**			**Decrypt**			**Key Expansion**		
	OR	**AND**	**SHIFT**	**OR**	**AND**	**SHIFT**	**OR**	**AND**	**SHIFT**
128/128	1340	1828	3180	3932	5284	2028	290	340	120
128/192	1956	2742	4338	5844	7926	2610	288	384	96
128/256	2572	3656	5496	7756	10568	3192	299	430	84
192/128	1628	2220	3884	4796	6444	2476	543	630	228
192/192	2376	3330	5298	7128	9666	3186	431	598	132
192/256	3124	4440	6712	9460	12888	3896	471	678	132
256/128	1916	2612	4588	5660	7604	2924	841	986	348
256/192	2796	3918	6258	8412	11406	3762	703	950	228
256/256	3676	5224	7928	15208	11164	4600	630	924	168

Table 4: RC5 complexity

Cipher parameters	Key expansion – bitwise op.			Encryption – bitwise op.			Decryption – bitwise op.		
	AND	OR	SHIFT	AND	OR	SHIFT	AND	OR	SHIFT
RC5 w=32/$\mathbf{N}_r$ 12	25168	12506	2730	2368	768	768	2368	768	768
RC5 w=64 bits/16 rounds	65688	32742	6834	6272	2048	2048	6272	2048	2048

Table 5: Parameters for performance model [2]

Algorithm	***A***	***B***	***Block_size (bytes)***
RC5 -Init/Encrypt	352114	40061	8
RC5 - Init/Decrypt	352114	39981	8

and current drawn by the operating device are measured and the power consumed for each algorithm are determined by subtracting the idle power consumption. The processing times of different encryption stages are determined by including different timers in the source codes. Using the Oprofile tool [1], the number of processor cycles used by each algorithm are determined since it is well know that processor cycles is linearly correlated to energy usage. Each functional block of the cryptographic algorithm, such as initialization, encryption and decryption, is executed **many** times with the same input, and the modal value of results are taken over these runs. The above measurements will be repeated by varying the block size to be encrypted, key size and number of rounds. Then the correlation to energy consumption and processor cycle usage can be determined. The methodology adopted in the analysis of cryptographic computational complexity and energy-efficiency for wireless sensor network security is shown in Figure 1. The main components are the analytic computational load and energy cost models. Two components are included; a model for computational complexity that derives the computational overhead of a particular encryption algorithm in terms of IPS (instructions per second) or processor cycles per second and an energy cost model that estimates energy consumption in terms of Joules/bit. In order to derive results from these models (either in the form of one optimal design for a given set of parameters or more general design tradeoffs), a number of input parameters are specified: computational complexity parameters (e.g., key length, number of encryption rounds, processor speed etc.) and energy cost parameters (e.g., memory requirements, CPU utilization, throughput etc.). The cryptographic algorithm simulation environment is also used to verify the accuracy of the analytical model.

6 Execution Time Measurement

ANSI C implementations of the AES and RC5 block ciphers are used to evaluate their computational performance and energy cost characteristics. For encryption the libraries Crypto++ and OpenSSL providing implementation of RC5 and AES in C Language have been adapted. Sim-Panalyzer [16], a cycle accurate instruction set simulator, Oprofile and Wattch [1] are used to measure both the execution time and the energy consumption of software running on processor. In order to evaluate the performance of the cryptographic algorithms, the following simulation tests are carried out:

1) Encrypting data arrays of different size up to 8192 byte with RC5/AES. For each size, the test was executed multiple times and the median execution time was calculated.

2) Encrypting different size data up to 8192 byte with RC5/AES while periodically measuring the battery voltage/power consumption for different rounds.

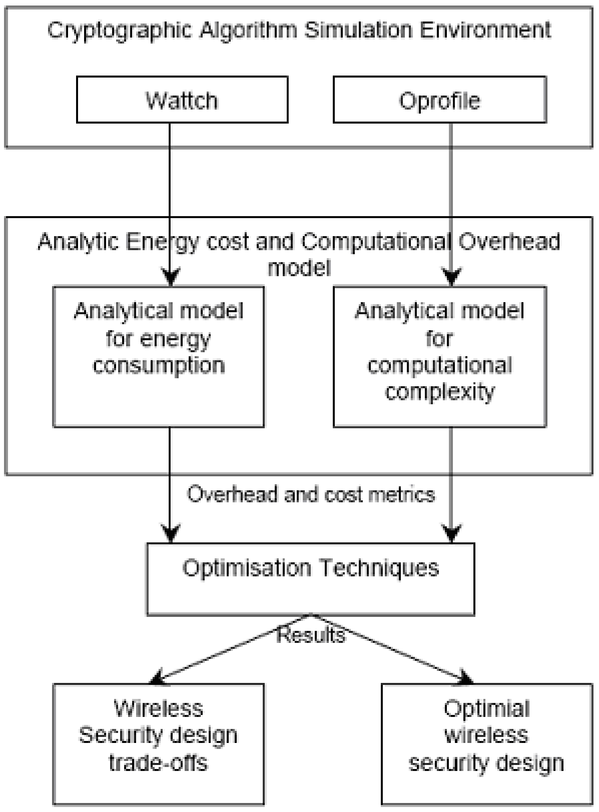
Cryptographic Algorithm Simulation Environment
Wattch
Oprofile
Analytic Energy cost and Computational Overhead model
Analytical model for energy consumption
Analytical model for computational complexity
Overhead and cost metrics
Optimisation Techniques
Results
Wireless Security design trade-offs
Optimial wireless security design

Figure 1: Cryptographic analysis model and optimization methodology

Figures 2 and 3 illustrates the execution time measurement in the cipher simulation procedure. In Figure 2, Initialization of data array, and different bit pattern (files such as image, audio etc) are considered for the AES-Rijndael encryption. Time measurement is a complex operation on real sensor nodes, and timestamp T_1 describes start time of encryption in the simulation operation. Different block sizes up to 1024 bytes are considered during execution of AES-Rinjdael (and RC5). Intermediate execution times of encryption (decryption) modules are captured after each rounds to determine the execution time of each encryption (decryption) stage and timestamp T_2 represents finishing time of process. The battery voltage was also periodically measured. For each block size, the test was executed multiple times to achieve an adequate confidence interval and the mean execution time was taken. $T_b[i]$, $T_s[i]$, $T_m[i]$, $T_k[i]$ are the time recorded for the ith round, Byte substitution, Shift row, Mix column and Key addition steps.

7 Performance Analysis

7.1 Simulation results and Costs analysis

The simulation performance of AES-Rijndael and RC5 cryptographic techniques are assessed in terms of computational overhead, complexity, execution time and energy cost. The computation times are linearly dependent to the amount of data being processed. The amount of computational energy consumed by a security function on a given microprocessor is primarily determined by the **number of clocks** needed by the processor to compute the security function, which in turn depends largely on the code efficiency of the cryptographic algorithm.

RC5 requires that a pre-computed key schedule to be stored in memory taking up significant bytes memory for each key. While, Rijndael needs another code for decryption because of the asymmetry of encryption and decryption and it takes almost twice memory for code since most part of it cannot be shared. The execution time and the battery consumed are proportional to the complexity of operations per round as proved by complexity equations derived and the number of data moves required making it difficult to affirm how much effect each type of low-level operation has in the performance of the schemes. It is mainly because data transfers are difficult to analyze with different register allocations that can be done by the compiler and state of the operation system. From above metrics and results, although it seems that an increase in power consumption is directly proportional to the increased operations, the increase is less amplified. This can be attributed to the fact that the data access from the file over which the operations are performed. However, the increased power consumption of larger key size represents a compromise that should be considered before choosing the size of the key. For normal applications, 128 bits key is considered very secure hence going for higher key sizes would mean unnecessary wastage of resources

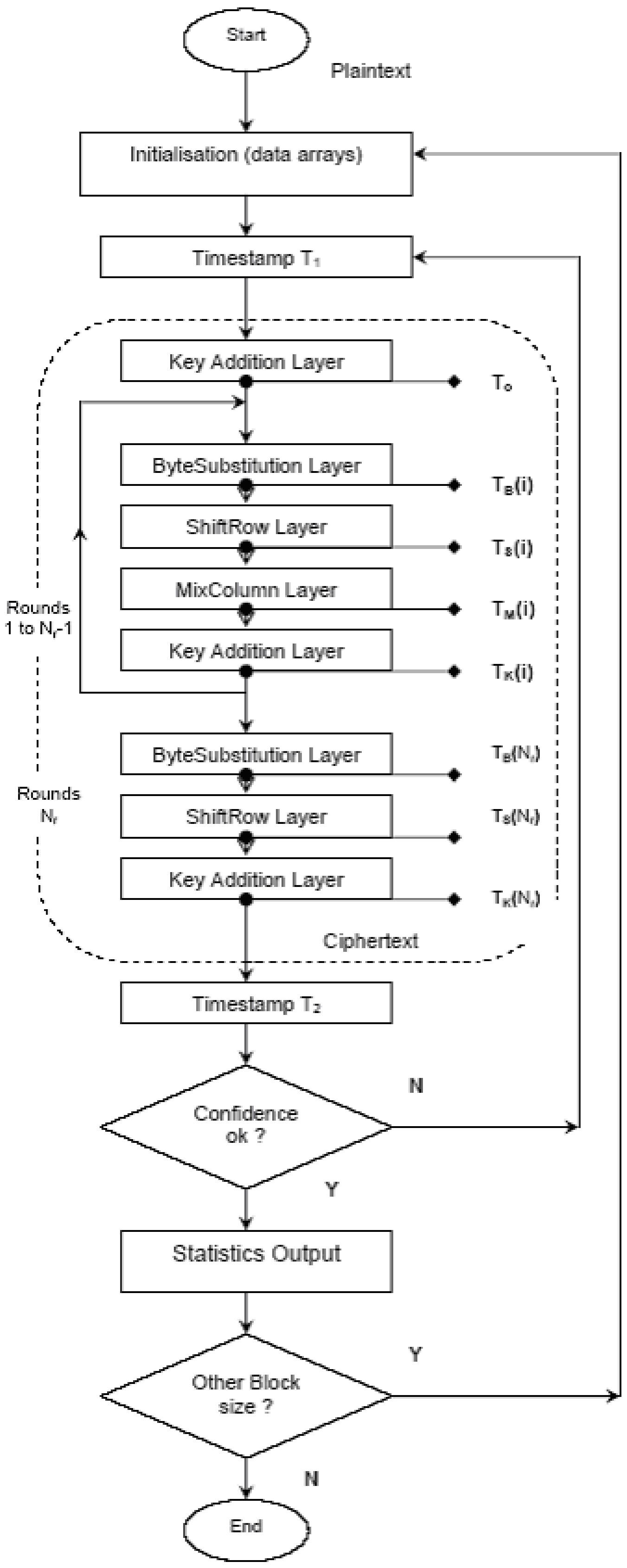

Figure 2: Timing of different stages in AES-Rijndael

Table 6: Overall performance of AES-Rijndael/RC5 block ciphers

Scheme	Encryption code size	Decryption code size	Encryption +Key expansion clock cycles	Decryption +Key expansion clock cycles	Energy for Key Expansion and encryption for 128-bit block	Energy for Key Expansion and encryption for 128-bit block
AES - 128	2390 bytes	2752 bytes	3638	4235	23.6μJ	36.5 μJ
RC5 -128	756 bytes	992 bytes	140452	140452	42.5μJ	42.5μJ

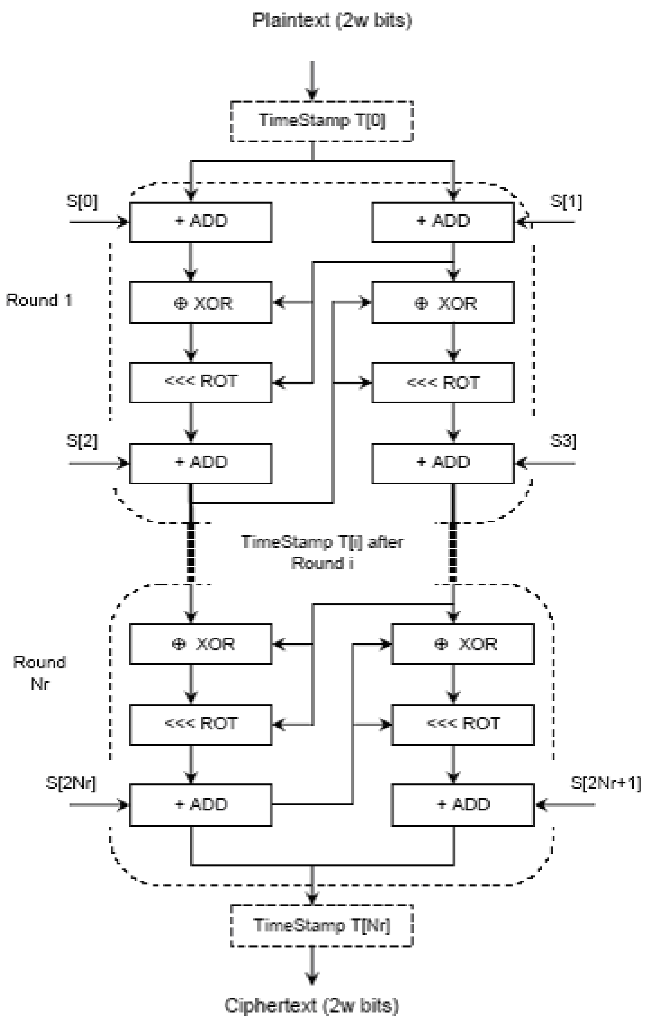

Figure 3: Timing of RC5 encryption algorithm

for the added security that is actually not required. Thus it is necessary to define what would constitute as adequate security which in turn varies from one application to another. RC5 encryption algorithm is very compact, and is coded efficiently. The Table S is relatively small size and accessed sequentially, thus minimizing issues of cache size.

Key observations for design choice of general-purpose cryptographic algorithm for resource constrained wireless sensor network in the near future:

AES-Rinjdael has lower power consumption and is faster to compute than RC5, as well as being cryptographically more secure. Although the *fastest* cryptographic engines are still nearly twice as fast performing RC5 than AES, we believe this is primarily a matter of the newness of AES and fully expect that, in the next few years, the fastest encryption engines will be targeting AES. For devices of the laptop variety (>=100,000J batteries) and larger, power usage from cryptographic algorithms is not a *huge* concern. For PDA devices and smaller ones, cryptographic algorithm selection should be taken seriously; run time differences of 2:1 can result from improper choices [12].

Low-power software design is effectively constrained to coming up with more efficient instruction sequences to implement the algorithm in question, as most processors and/or the operating system perform all system power sequencing. This does, however, allow hand-coded assembly language software to execute both faster and with lower power than software written in a high-level language. High-level compilers do have many avenues to optimize code, though; the power of such optimizers should not be underestimated.

Table 7: Execution Time profile of encrypt() simulation

AES-128 Encrypt()	Mean % computational time
SubBytes	1.9 %
ShiftRows	33 %
MixColumns	63 %
AddRoundKey	2.1 %

Results from Tables 7 and 8 on the different stages of the AES-Rijndael cipher algorithm code optimization revealed that the MixColumns() function execution was consuming substantially more time than other sub-

Table 8: Percentage of operations with encrypt() computation model

AES 128/10 Encrypt()	**Bytewise-AND**	**Bytewise-OR**	**Shift (bytes)**
SubBytes	120Ta (6.6%)	80To (6.0%)	0 (0.0%)
ShiftRows	0 (0.0%)	120To (9.0%)	120Ts (5.0%)
MixColumns	1368Ta (75.7%)	972To73.0%	2304Ts (95.0%)
AddRoundKey	320Ta (17.7%)	160To (12.0%)	0 (0.0%)

modules combined in encrypt() function. The main reason is due to the multiplication operation in the MixColumns() function, which is used to perform the GF(2^8) field multiplication on the data operands. Comparing the simulation values and computation model values, in Tables 7 and 8 reveal that cyclic shift (rotation) operations are more costly than bytewise-AND or bytewise-OR, hence adding extra computational load on microprocessor. Results in Table 6 confirms that decrypt () has more computational overhead than encrypt(), approximately 20-35% more clock cycles, because of the additional complexity of the GF multiplication in InvMixColumns() of decrypt (). The InvMixColumns() needs to perform four multiplications while MixColumns() performs only two multiplications per each byte of the State. This complies with the mathematical model analysis where the difference in computation between one InvMixColumn and MixColumn operation is $[96\mathbf{N_b T_a} + 72\mathbf{N_b T_o} - 32\mathbf{N_b T_s}]$.

As expected, the energy values are almost proportional to the corresponding execution times. Going from AES-128 bits key to AES-192 bits causes increase in power and time consumption by about 10% - 15% and to 256 bit key causes an increase of 16% - 25%. AES-128 has 3396, AES-192 has 6186, and AES-256 has 9704 different byte operations, which implies 35% and 45% more operations for AES-192 and AES-256 when compared to AES-128. Although RC5 algorithm seems well suitable algorithm for sensor networks, there are two considerations about why this choice may not be optimal for all resource-limited architecture. First, the key expansion step of the algorithm may not be integrated in the encryption process. This results in the need to have round keys stored for the whole encryption process which leads to a slightly higher memory requirement than desired (at least 112 bytes for RC5-32/12/8 when the original key needs to be kept). Second, the RC5 algorithm extensively uses circular shifts. This operation is often not supported by low cost processors and must be emulated by single step shifts, which leads to bad runtime behaviour. The AES-Rijndael algorithm can thereby offer the best performance, but requires large lookup-tables. Also this algorithm is not constructed for a key length shorter than 128-bit, which are often used in sensor networks

7.2 Optimisation Scope

The cipher algorithms need to perform highly complex sets of arithmetic functions, which can be simplified using look-up tables, caching and bit-manipulation. The SubBytes() in AES-Rijndael can be implemented using mathematical formula but consumes higher processor cycles and more energy. Hence, it is more efficient to implement it using a look-up table. But, look-up tables consume a lot of memory and efficient algorithms should generate it before encryption or decryption starts. Although, the code size will increase, it certainly improves overall cryptographic performance by minimising the dormant memory occupation. The basic operations of each block cipher can be implemented in different ways; however, implementations that are suitable for resource-constrained nodes are favoured. The programming codes are made more flexible for module reusability in the application. The Mulitplication () function is made common to both MixColumns() and InvMixColumns(). Finally, the MixColumns() and InvMixColumns() functions are the high-cost critical paths in the encryption and decryption program, respectively. It is more important to optimise the critical paths to a higher extent than the less critical paths. If the finite field inversion, InvMixColumns(), were implemented using Boolean functions only, then the required execution time and the energy cost would be very high. A moderate sized look-up table would be more efficient even on memory-restricted devices. To provide variable security strengths with limited amount of components, code efficiency, reusability and flexibility are important factors when choosing the components. In addition, the communication overhead of the algorithm, e.g., additional bytes we need to add to each data packet, should also be taken into consideration. RC5 is a highly efficient and flexible cryptographic algorithm, for which many parameters (key size, block size, number of rounds) can be adjusted to tradeoff security strength with power consumption. The comparison of RC5 implementations is provided in Table 9.

Table 9: Comparison of RC5 implementation gain

RC5 Implementation	**Operation cycles**	**Execution Time**
C version	5750	1.70 ms
SPINS (C+ ASM)	2775	0.75 ms
TinySec (C+ ASM)	1775	0.50 ms

Table 10 gives the average speed performance gain from AES optimization. Encrypt() and Decrypt() achieve up to 21% and 11% execution time gain, respectively. ShiftRows() are optimized better than all other sub-

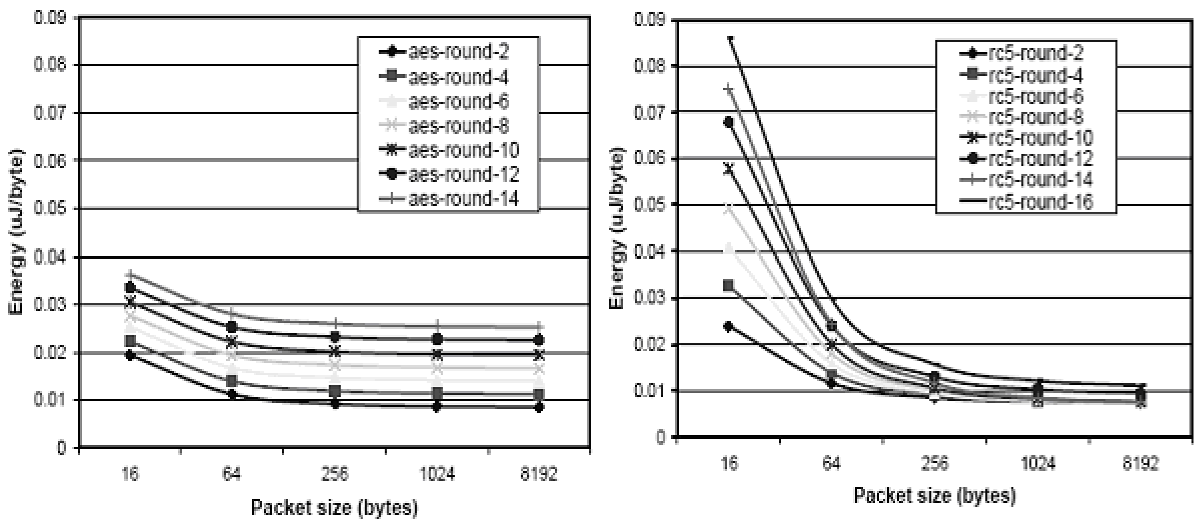

Figure 4: Energy consumption for AES-Rijndael and RC5 for different rounds

Table 10: Speed performance gain with optimization

AES Percentage Speed Performance (%)	
Main ()	12
Encrypt()	21
Decrypt()	11
SubBytes()	26
ShiftRows()	50
MixColumns()	1.4
AddRoundKey()	29

functions, whereas MixColumns() are the least improved due to the implementation of look-up table.

Optimizing block cipher implementations towards low energy consumption and low computational overhead is of paramount importance for virtually any battery-powered wireless device, especially sensor nodes. Memory utilization is also a critical concern since RAM and cache size is a precious resource in sensor nodes. Hence, code size optimization and lightweight block cipher software implementations that meet operational requirements of WSNs are important.

8 Conclusions and Future Research

The performance evaluation of cryptographic algorithms is vital for the safe and efficient development of cryptosystem in devices with low computational power. In this paper, we presented a systematic model of cryptographic algorithms complexity and in particular analyzed the suitability of RC5 and AES-Rijndael encryption techniques to provide efficient link layer security. The approach presented in this paper and its respective tools allow efficiency estimation of the algorithms in resource-constrained devices. The results' evaluations bring the possibility of verifying the viability of the algorithm application in resource-limited device. The simulation results presented quantify the differences in computational complexity overhead and energy cost of AES-Rijndael and RC5 algorithm with varying operational parameters. Besides, a simple and low overhead analytic model has been presented to compare the execution time and number of processing required for RC5 and AES-Rijndael. These outcomes can be helpful for security designers in adopting a specific cryptographic scheme for resource constrained wireless environment. AES-Rijndael is suitable on all aspects and future research in optimization techniques will definitely make it the de facto encryption for resource-constrained wireless networks. RC5 is good on the code point of view, but the key schedule consumes more time. Resource and cost constraints remain a challenge in designing efficient security mechanisms for WSNs. Several directions for future research arise from work. One future research is to explore adaptive cryptographic mechanisms to optimize energy consumption by varying cipher parameters with timely acquisition of resource-context in WSN environment. The adaptability of the security system will improve sensor nodes battery's lifetime.

References

[1] D. Brooks, and et. al, "Wattch: A framework for architectural-level power analysis and optimizations", *ISCA*, pp. 83-94, 2000.

[2] P. Ganesan, R. Venugopalan, and P. Peddabachagari, "Analyzing and modelling encryption overhead for sensor network nodes," *ACM in Proceedings of WSNA'03*, pp. 151-159, Sep. 2003.

[3] F. Granelli and G. Boato, "A novel methodology for analysis of the computational complexity of block ciphers: Rijndael, Camellia and Shacal-2 compared," *Third Conference on Security and Network Architectures (SAR'04)*, pp. 1-7, June 2004.

[4] J. Grobschadl, S. Tillich, C. Rechberger, M. Hofmann, and Marcel Medwed, "Energy evaluation of software implementations of block ciphers under memory constraints," *Proceedings of the 10th Conference on Design, Automation and Test in Europe*, pp. 1110-1115, 2007.

[5] V. Gupta, M. Millard, S. Fung, Y.Zhu, N. Gura, and S. Shantz. "Sizzle: a standards-based end to end security architecture for the embedded internet," *Proceedings of third IEEE International Conference on Pervasive Computing and Communications, PerCom 2005*, pp. 247-256, Kaua, Huwaii: IEEE, Mar. 8-12, 2005.

[6] C. T. R. Hager, S. F. Midkiff, J. M. Park, and T. L. Martin, "Performance and energy efficiency of block ciphers in personal digital assistants," *Third IEEE International Conference on Pervasive Computing and Communications*, pp. 127-136, Mar. 8-12, 2005.

[7] A. Hodjat and I. Verbauwhede, "Interfacing a high speed crypto accelerator to an embedded CPU," *Proceedings of the 38th Asilomar Conference on Signals, Systems, and Computers*, vol. 1, pp. 488-492, IEEE Press, 2004.

[8] C. Karlof, N. Sastry, and D. Wagner. "TinySec: a link layer security architecture for wireless sensor networks," *ACM SenSys 2004 in Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems*, pp. 162-175, Nov. 2004.

[9] Y. W. Law, S. Dulman, S. Etalle, and P. J. M. Havinga. "Assessing security-critical energy-efficient sensor networks," *Proceedings 18th IFIP TC11 International Conference in Information Security, Security and Privacy in the Age of Uncertainty (SEC)*, pp. 459-463, Athens, Greece, May 2003.

[10] Y. Law, J. Doumen, and P. Hartel. "Benchmarking block ciphers for wireless sensor networks (extended abstract)," *1st IEEE International Conference Mobile Ad-hoc and Sensor Systems, IEEE Computer Society Press*, pp. 447-456, Oct. 2004.

[11] A. Perrig, J. Stankovic, and D. Wagner, "Security in wireless sensor networks," *Communication ACM*, vol. 47, no. 6, pp. 53-57, 2004.

[12] N. R. Potlapally, S. Ravi, A. Raghunathin, and N. K. Jha, "Analyzing the energy consumption of security protocols," *Proceedings. 2003 International Symposium on Low Power Electronics and Design*, pp. 25-27, Aug. 2003.

[13] S. Ravi, A. Raghunathan, P. Kocher, and S. Hattangady, "Security in embedded systems: Design challenges," *ACM Transactions on Embedded Computing Systems*, vol. 3, no. 3, pp. 461-491, Aug. 2004.

[14] R. L. Rivest, "The RC5 encryption algorithm", *Proceedings of the 1994 Leuven Workshop on Fast software Encryption*, pp. 86-96, Springer-verlag, 1995.

[15] B. Schneier, *Applied Cryptography:Protocols, Algorithms and Source Code in C*, John Wiley & Sons Inc. 2nd Edition, 1996.

[16] Sim-Panalyzer Project, Last Accessed, May 2007. http://www.eecs.umich.edu/~panalyzer/.

[17] H. S. Soliman, and M. Omari, "Application of synchronous dynamic encryption system (DES) in wireless sensor networks," *International Journal of Network Security*, vol. 3, no. 2, pp. 160-171, Sep. 2006.

[18] A. P. R. Szewczyk, V. Wen, D. Culler, and J. D. Tygar, "SPINS : security protocols for sensor networks," *Wireless Networks*, vol. 8, no. 5, pp. 521-534, Sep. 2002.

[19] A. Vitaletti, "Gianni palombizio: Rijndael for sensor networks: is speed the main issue?," *Electronic Notes Theorem Computer Science*, vol. 171, pp. 71-81, 2007.

[20] J. P. Walters, Z. Liang, W. Shi, and V. Chaudhary, "Wireless sensor network security: Survey," *Security in Distributed, Grid, and Pervasive Computing*, pp. 367-403, CRC Press, Yang Xiao, Editor, Auerbach publications, 2006.

[21] Y. Wang, G. Atterbury, and B. Ramamurthy, "A survey of security issues in wireless sensor networks," *IEEE Communications Surveys & Tutorials*, vol. 8, no. 2, pp. 2-23, 2006.

[22] C. Xenakis, N. L. Merakos, and I. Stavrakakis, "A generic characterization of the overheads imposed by IPsec and associated cryptographic algorithms," *Elsevier Journal of Computer Networks*, pp. 3225-3241, 2006.

[23] Y. Xiao, H. Chen, B. Sun, R. Wang, and S. Sethi. "MAC security and security overhead analysis in IEEE 802.15.4 wireless sensor networks," *EURASIP Journal on Wireless Communication and Networking*, pp. 1-12, 2006.

M. Razvi Doomun holds a B.Eng (Hons) in Electronic and Communication Engineering from University of Mauritius and an MSc in Multimedia Communications from University of Surrey, UK. He is at present lecturer in the Department of Computer Science and Engineering at University of Mauritius and currently doing his PhD in Wireless Security. He has published articles on Wireless security and multimedia communication.

KM Sunjiv Soyjaudah received his BSc. (Hons.) degree in Physics from Queen Mary College, University of London in 1982, his M.Sc. degree in digital electronics from King's College, University of London in 1991 and his PhD degree from University of Mauritius in 1998. He is a chartered engineer and a member of the IEEE. He is presently Professor in the department of electrical and electronic engineering of the University of Mauritius. His interests are communication theory, cryptography and wireless network security.