UNIVERSITY
OF TRENTO

DEPARTMENT OF INFORMATION AND COMMUNICATION TECHNOLOGY

38050 Povo – Trento (Italy), Via Sommarive 14
http://www.dit.unitn.it

A NOVEL METHODOLOGY FOR ANALYSIS OF THE COMPUTATIONAL COMPLEXITY OF BLOCK CIPHERS: RIJNDAEL, CAMELLIA AND SHACAL-2 COMPARED

Fabrizio Granelli and Giulia Boato

January 2004

Technical Report DIT-04-004

A novel methodology for analysis of the computational complexity of block ciphers: Rijndael, Camellia and Shacal-2 compared.

F. Granelli and G. Boato
DIT - University of Trento
Via Sommarive 14, I-38050 Trento (Italy)
E-mail: {granelli,boato}@dit.unitn.it

Abstract

The paper presents a methodology for the evaluation of the complexity and computational cost of different block ciphers, in order to be independent from the actual platforms they are implemented on. An analysis of three block ciphers selected by NESSIE (New European Schemes for Signatures, Integrity and Encryption) in 2003 - Rijndael, Camellia and Shacal-2 - is then provided using such methodology. First, the structure of all ciphers is described, so as to emphasize the different kinds of transformations required. Then, the schemes are compared in terms of basic operations (AND, OR, shifts) for each step, in such a way to evaluate their complexity and to provide effective guidelines regarding their implementation.

Keywords: *block ciphers, cipher complexity, performance evaluation, AES.*

1 Introduction

The NESSIE project (New European Schemes for Signatures, Integrity and Encryption) is an open competition for the crypto algorithms of the 21^{st} century, which aim is to select a strong portfolio of crypto algorithms that will protect the information society (on-line bank transactions, credit cards, personal informations, e-commerce,...). In February 2003 the NESSIE project announces the

finalists: MISTY1, Camellia, Shacal-2 and AES (Rijndael) are the four selected block ciphers [1]. The NESSIE call for primitives specified three security levels for block ciphers: high, normal and normal-legacy [2]. MISTY1 was selected for the normal-legacy security level, while the other three correspond to higher levels. For this reason we decided to take into consideration only Rijndael, Camellia and Shacal-2.

The deadline for submissions to the NESSIE project was just before NIST's announcement that the AES block cipher was to be Rijndael. The choice of Rijndael was primarily based on its efficiency and low memory requirements. In the same year (2000) NTT and Mitsubishi Electric jointly developed Camellia, a next-generation symmetric-key encryption algorithm, and presented a submission statement of it to NESSIE. Camellia has the same interface as AES and provides excellent efficiency [3, 4]. The distinguishing characteristic is the smallest 128-bit block cipher hardware in the world.

Shacal was submitted to NESSIE by Gemplus [5]. Shacal has two versions (Shacal-1 and Shacal-2) depending on the hash standard SHA-1 [6] and SHA-256 used in encryption mode. Only Shacal-2 was selected by NESSIE, while SHA-256 was added in the hash functions portfolio. SHA was not designed for encryption, however, its compression function can be used for encryption. No security flaws have been found in Shacal-2 and its performance is quite good [2].

NESSIE primitives will be used on a variety of platforms: PCs, smart cards, hardware, and in various other applications. Some application areas impose very high performance requirements (hard disk encryption) and protection of high speed communications. Performance evaluation is an essential part in determining the practicality of a cryptographic algorithm. In fact, an algorithm that performs well is more likely to be adopted for practical applications. The state of the art offers several performance evaluations, but all of them (being based on actual harware/software implementations) are strictly connected to the selected platforms. NESSIE also provided such kind of results [7].

Aim of the paper is to present a methodology for the evaluation of the complexity and computational cost of block ciphers, thus bridging the gap between the mathematical studies and algorithms implementation. The approach, in fact, allows complexity evaluation and effective comparison of block ciphers without being dependent on the actual platform they are implemented on. The method

is presented for the analysis and comparison of Rijndael, Camellia and Shacal-2. Section 2 briefly introduces the three algorithms. Section 3 reports detailed explanation of the methodology and the application on the selected block ciphers. The achieved results are presented in Section 4, while Section 5 provides some comparisons among the three algorithms. Finally, Section 6 concludes the paper.

2 Block ciphers

2.1 Rijndael

Rijndael is a iterated block cipher with a variable block length and a variable key length $(128, 192, 256$ bit$)$. The number of rounds is determined by a combination of key length and block size [8, 9]. While other block ciphers have Feistel structure (see Camellia), it has non-Feistel structure. Rijndael is a symmetric block cipher with a simple, elegant and easily understandable algebraic structure. It relies directly on algebraic constructs. Let $GF(2^8)$ be the Galois Field defined by the irreducible polynomial [6] $x^8 + x^4 + x^3 + x + 1$ and then view the block as set of elements of the field. The data are placed in an $4 \times N_b$ (block length/32) array of elements of $GF(2^8)$ [8, 9].

The algorithm consists of an initial round-key addition, the required number of rounds and a final round. Rijndael performs encryption by an iterative transformation called the round transformation, of which inverse operation is used for decryption and the order of the round for decryption is reversed. The round transformation is composed of three distinct invertible uniform operations: a linear mixing layer (ShiftRows and MixColumns), a non-linear layer (SubBytes) and a key addition layer (AddRoundKey). Specific choices for the different layers are based on application of the wide trail strategy [8, 9], a design method to provide resistance against linear and differential cryptanalysis.

- SubBytes is a non-linear byte substitution, operating on each of the state bytes independently, composed by the multiplicative inverse in $GF(2^8)$ $(0$ is mapped into itself $)$ and a fixed affine transformation over $GF(2^8)$;
- ShiftRows cyclicly shifts the elements of the i^{th} row of the state C_i elements to the right, where 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 $03x^3+x^2+x+02$ to give elements of a new $4 \times N_b$ array;
- AddRoundKey is a XOR of the key (after the scheduling) with the array.

Decryption is realized by applying reverse transformation of each round. Since inverse cipher is similar in structure, but uses a MixColumns transformation with another polynomial ($0Bx^3 + 0Dx^2 + 09x + 0E$) a performance degradation is observed on 8-bit processors. Some observation by Barreto and [10] help to reduce the performance degradation of the decryption procedure (see Section 4).

The key schedule for 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 N_k (key length/32): there is a version for N_k equal to or below 6 and a version for N_k above 6. In both versions the first N_k columns of the array are filled with the cipher key. The following columns are defined recursively in terms of previously defined columns. The recursion uses the bytes of the previous column, the bytes of the column N_k positions earlier and round-constants. For additional details, see [8, 9].

2.2 Camellia

Camellia is a next-generation symmetric-key encryption algorithm, which supports 128-bit block size and 128, 192, 256-bit key, i.e. the same interface specifications as AES. Efficiency on both software and hardware platforms is a remarkable characteristic of Camellia in addition to its high level of security.

Camellia uses an 18-round Feistel structure for 128-bit key and a 24-round Feistel structure for 192 and 256-bit keys. There is additional input/output whitening and every six rounds two logical functions FL, FL^{-1} are used to provide some non-regularity across rounds. The key schedule generates from secrecy key 64-bit subkeys for round function and for FL/FL^{-1} functions. The F-function of Feistel structure presents the SPN structure, where the S-function is represented by four fixed S-boxes, used two times in every round. All of them are affine equivalent to the inversion function in $GF(2^8)$.

Camellia decryption procedure can be done in the same way as encryption procedure by reversing the order of subkeys, which is one of merits of Feistel networks. In Camellia, FL/FL^{-1} function layers are inserted every six rounds, but this property is still preserved.

Camellia key schedule varies for different key sizes. The procedure can be divided into two steps. First, derive one (or two) 128-bit key materials K_A (and K_B) from the original secret key K. Second, generate all round keys by rotating K, K_A, K_B by various amounts [4, 3].

2.3 Shacal-2

Shacal-2 is an encryption algorithm based on SHA-256, which was introduced by NIST in 2000, which is used in encryption mode. Shacal-2 is a 256-bit block cipher defined with a 512-bit secret key. Shorter key (not shorter than 128 bit) may be used by padding the key with zeros to a 512-bit string.

For the encryption procedure the 256-bit plaintext is put into eight 32-bit variables $A\,B\,C\,D\,E\,F\,G\,H$, which are subsequently elaborate by the compression function during 64 rounds. Each round $A\,B\,C\,D\,E\,F\,G\,H$ are redefined, adding (mod 2^{32}) constants and round keys, and using four functions ($Ch,\ Maj,\ \Sigma_0,\ \Sigma_1$) [5, 2].

In order for Shacal-2 to be invertible the final addition with the initial value, which occurs in hash mode for SHA-256, is omitted. As function over $GF((2^8)^4)^8$ each round step is invertible in the round variables.

In all Shacal-2 documentation there is no detailed description of the decryption procedure. Therefore, the following paragraphs present a procedure for decryption defined by the authors of the paper.

Known $A^{i+1}\,B^{i+1}\,C^{i+1}\,D^{i+1}\,E^{i+1}\,F^{i+1}\,G^{i+1}\,H^{i+1}$ you get the values of A^i $B^i\,C^i\,E^i\,F^i\,G^i$, compute

$$R^i = \Sigma_1\left(E^i\right) + Ch\left(E^i, F^i, G^i\right) + K^i + W^i + \Sigma_0\left(A^i\right) + Maj\left(A^i, B^i, C^i\right)$$

and calculate its opposite $\overline{R^i}$ in $GF(2^8)^4$. Then you find the missing values of D^i and H^i as follows:

$$\begin{aligned} H^i &= A^{i+1} + \overline{R^i} \\ D^i &= E^{i+1} + \overline{H^i + R^i} \end{aligned}$$

In Shacal-2 the key schedule expands the 512-bit key to 2048 bits: the 512-bit key are considered as 16 32-bit words and are expanded with the function

$$W^i = \sigma_1(W^{i-2}) + W^{i-7} + \sigma_0(W^{i-15}) + W^{i-16}$$

where $16 \leq i < 64$ and σ_0, σ_1 are defined through XORs, ORs and binary shifts [2].

3 Methodology for comparison

The methodology proposed in this section provides a tool to evaluate a block cipher performance while achieving independence from the particular platform which it is implemented on. The core idea is to consider only the amount of relevant operations required for the implementation of the algorithms, reducing all involved transformations to *bytewise-AND* and *bytewise-OR*, and *shifts*. Cost and working of communication and logical circuits depend on technological progress, and consequently such a methodology remains valid over time and could also provide suggestions on the best platform to implement a given cipher.

The method is presented for the numerical evaluation of the computational complexity of Rijndael, Camellia and Shacal-2. In order to illustrate the methodology, details of the calculation of the computational cost of the transformations are reported. Despite the fact that Rijndael, Camellia and Shacal-2 have different structures, it is still possible to reduce these three algorithms to a set of successive logical operations - bytewise-AND, bytewise-OR and shifts of bytes - in order to enable effective comparison among them and to verify the effective contribution of the methodology.

The cost of the algorithms is estimated through a ”step by step” analysis. Since Rijndael, Camellia and Shacal-2 are iterated block chipers, it is possible to reconstruct the cost of the body algorithm through the analysis of only one round. Each round of a iterative block cipher is a function over $GF(2^{blocklength})$ and consists of composite transformations. As shown in the previous section, the ciphers have different structures, but using the following scheme the implementation computational cost can be calculated:

(a) Analysis of input/output whitening (for Rijndael and Camellia);

(b) Analysis of one round;

(c) Analysis of the last round (Rijndael) or FL/FL^{-1} (Camellia).

In order to reconduct all the transformations to logical operations and shifts, we exploit the rule that a simple bitwise-XOR of a and b is equal to $\overline{a}b + \overline{b}a$. Since negations are negligible compared to AND/OR logical operations, a bitwise-XOR is considered as the sum of two bitwise-ANDs and one bitwise-OR. Moreover, a circular shift operation of a 8-bit word by n position is considered as a bytewise-OR and 8 shifts.

3.1 Rijndael

All finite fields used in the description of Rijndael have a characteristic of 2, therefore the addition operation is a XOR. In the algorithm there are no multiplications of two variables in $GF(2^8)$, but only the multiplication of a variable with a constant (in MixColumns).

(a) AddRoundKey can be implemented with $8N_b$ bytewise-ANDs and $4N_b$ bytewise-ORs.

(b) One round of Rijndael consists of SubBytes, but we consider substitutions as table look ups and do not include them in the complexity evaluation [13] ; ShiftRows, which consists of $3N_b$ shifts of bytes and $3N_b$ bytewise-ORs; MixColumns (see further) and AddRoundKey. As explained in Section 2 in MixColumns the columns of the state are multipliqed modulo $x^4 + 1$ by $03x^3 + x^2 + x + 02$ to give elements of a new $4 \times N_b$ array. The multiplication modulo this polynomial can be written as a circulant matrix multiplication [9]: the first new element has calculated as follows, and similarly the others $4N_b - 1$:

$$b_0 = 02a_0 + 03a_1 + a_2 + a_3.$$

The addition can be implemented with the bitwise-XOR instruction. There is still multiplication complexity to calculate. As Rijndael authors, we consider bytes as polynomials. The multiplication is defined modulo $x^8 + x^4 + x^3 + x + 1$. Let's see how multiplication by the value 02 can be implemented. The polynomial associated with 02 is x. Therefore, if we multiply an element $z \in GF(2^8)$ with 02, we get:

$$z \cdot x = z_6x^7 + z_5x^6 + z_4x^5 + (z_3 \oplus z_7)x^4 + (z_2 \oplus z_7)x^3 + z_1x^2 + (z_0 \oplus z_7)x + z_7.$$

The multiplication by 02 can be implemented with 8 shifts, 8 bitewise-ORs and 3 bitwise-XORs. Since all elements of $GF(2^8)$ can be written as a sum of powers of 02, the complexity of multiplication by any constant value can be calculated easily. The multiplication by 03 can be implemented as follows:

$$z \cdot 03 = z \oplus z \cdot 02.$$

It needs 8 shifts, 8 bitwise-ORs, 3 bitwise-XORs and 1 bytewise-XOR. MixColumns transformation costs for every element 4 bytewise-XORs, 6 bitwise-XORs, 2 bytewise-ORs and 16 shifts, i.e. it can be implemented with $19N_b$ bytewise-XORs, $8N_b$ bytewise-ORs and $64N_b$ shifts.

Therefore, to implement one round we need $46N_b$ bytewise-ANDs, $31N_b$ + 12 bytewise-ORs and $64N_b$ + 96 binary shifts, i.e for 128-bit block encryption 184 bytewise-ANDs, 136 bytewise-ORs and 352 binary shifts; for 192-bit block encryption 276 bytewise-ANDs, 198 bytewise-ORs and 480 binary shifts; while for 256-bit block encryption 368 bytewise-ANDs, 260 bytewise-ORs and 608 binary shifts.

(c) The last round is equal to the round transformation, but with the MixColumns step removed. It can be implemented with $8N_b$ bytewise-ANDs, $7N_b$ bytewise-ORs and $3N_b$ shifts of bytes.

The number of rounds N_r is determined by a combination of key length and block size. Rijndael $32N_b$-bit encryption uses this scheme: (a), $(N_r - 1)$(b), (c).

The limitations of Rijndael are related with its inverse because the cipher and its inverse make use of different code and/or tables and the inverse cipher can only partially re-use the circuitry that implements the cipher. The number of operations changes only for InvMixColumns. InvMixColumns uses a MixColumns transformation with another polynomial: $0Bx^3 + 0Dx^2 + 09x + 0E$. The multiplication by bigger coefficients costs much more. We describe how they can be implemented.

$$\begin{aligned} z \cdot 09 &= z\,(01 \oplus 02^3) \\ z \cdot 0B &= z\,(01 \oplus 02 \oplus 02^3) \\ z \cdot 0D &= z\,(01 \oplus 02^2 \oplus 02^3) \\ z \cdot 0E &= z\,(02 \oplus 02^2 \oplus 02^3) \end{aligned}$$

where

$$zx^2 = z_5x^7 + z_4x^6 + (z_3 \oplus z_7)x^5 + (z_2 \oplus z_6 \oplus z_7)x^4 +$$

$$+(z_1 \oplus z_6)x^3 + (z_0 \oplus z_7)x^2 + (z_6 \oplus z_7)x + z_6$$

and

$$zx^3 = z_4x^7 + (z_3 \oplus z_7)x^6 + (z_2 \oplus z_6 \oplus z_7)x^5 + (z_1 \oplus z_5 \oplus z_6)x^4$$

$$+(z_0 \oplus z_5 \oplus z_7)x^3 + (z_6 \oplus z_7)x^2 + (z_5 \oplus z_6)x + z_5$$

Globally, InvMixColumns transformation need $134N_b$ bytewise-ANDs, $99N_b$ bytewise-ORs and $32N_b$ shifts of bytes to be implemented.

This asymmetry is due to the fact that the performance of the inverse cipher is considered to be less important than that of the cipher. In many applications of a block cipher, the inverse cipher operation is not used. This is the case for the calculation of Message Authentication Codes, but also when the cipher is used in Cipher Feedback mode or Output Feedback mode [8].

Following the key scheduling algorithm for all possible values of N_b and N_k (4, 6 and 8) we calculated how many bytewise-ANDs, bytewise-ORs and shifts do they need (see Section 4). If applications need to calculate frequently the round keys for decryption on-the-fly, their generation can be implemented in such a way that it outputs the round keys in the right order. To do this InvKeyExpansion [9] has to be used instead of KeyExpansion. With the proposed methodology we estimated that such kind of decryption round keys calculation need the same amount as KeyExpansion transformation with doubled shifts operations.

3.2 Camellia

Camellia was designed to provide high speed in software and hardware implementation as well as compactness of hardware chips. 32-bit integer addition and multiplications are not used. The reason is that - as we've already shown in 3.1 - multiplications need large hardware implementation requirements.

(a) Input/output whitening can be implemented with 32 bytewise-ANDs and 16 bytewise-ORs.

(b) One round of Camellia presents the Feistel structure in which half-plaintext enter in function F, suffered XOR and it is exchanged with other half-plaintext. The function F presents SPN-structure based on a XOR with subkey, a substitution and a permutation. As said in 3.1, we do not consider the S-box. XOR can be

implemented with 8 bytewise-XORs and permutations with 36 bytewise-XORs. The global estimate is 104 bytewise-ANDs and 52 bytewise-ORs.

(c) Both FL and FL^{-1} work on a half-plaintext. In their structures are present bytewise-ANDs, bytewise-ORs, bytewise-XORs and rotations to the left or to the right by one bit. The total amount is 40 bytewise-ANDs, 32 bytewise-ORs and 8 shifts of bytes.

Camellia 128-bit encryption with 128-bit key follows this scheme: (a), 6(b), (c), 6(b), (c), 6(b), (a); with 192 or 256-bit key: (a), 6(b), (c), 6(b), (c), 6(b), (c), 6(b), (a).

The first step of scheduling for the key (see Section 2.2) is very similar to the algorithm but the number of rounds is changed and subkeys are fixed constants. For 128-bit key it follows the scheme (a), 2(b), (a), 2(b), while for 192 or 256-bit key (a), 2(b), (a), 2(b), (a), 2(b). On-the-fly subkey generation is computable in the same way in both encryption and decryption.

3.3 Shacal-2

Shacal-2 consists only of 64 round steps. No input/output addition is required. 64(b) is its corresponding scheme.

(b) One round of Shacal-2 consists of 7 additions in $GF(2^8)^4$. The amount of basic operations needed for such addition strictly depends on the 32-bit adder architecture which is implemented with. There are existing technologies characterized by different levels of complexity and performance. On the contrary, the number of logical ANDs and ORs normally required with a parallel addition of n bit is that of n full adders [15]. Since the aim of this work is to introduce a platform-independent methodology (see Section 1), we decided not to decompose the addition modulo 2^{32} but to present their number in separate tables.

For each involved function the amount of basic operations is: 16 bytewise-ANDs and 4 bytewise-ORs for Ch; 28 bytewise-ANDs and 8 bytewise-ORs for Maj; 16 bytewise-ANDs, 20 bytewise-ORs and 96 binary shifts for Σ_0; 16 bytewise-ANDs, 20 bytewise-ORs and 96 binary shifts for Σ_1.

Therefore, the total estimate for one round is 76 bytewise-ANDs, 52 bytewise-ORs, 24 shifts of bytes and 7 32-bit additions.

Regarding Shacal-2 decryption, the formulation that we have proposed in Sec-

tion 2.3 stresses that compared to the encryption procedure the only difference is the amount of 32-bit additions. To work out the opposite in $GF(2^8)^4$ an addition over the finite field is required. Therefore, one round of Shacal-2 decryption can be implemented with 76 bytewise-ANDs, 52 bytewise-ORs, 24 shifts of bytes and 10 32-bit additions.

The scheduling of 256-bit key is based on 48 times iterated function (see Section 2.3). Each round consists of 3 addition modulo 2^{32} and both σ_0 and σ_1 transformations, which can be implemented with of 32 bytewise-ANDs, 32 bytewise-ORs and 141 binary shifts.

4 Complexity evaluation

This Section provides the evaluation of the complexity of Rijndael, Camellia and Shacal-2 using the methodology described in Section 3.

4.1 Rijndael

The results regarding complexity evaluation of Rijndael encryption, decryption and key schedule are reported in Table 1, 2 and 3, respectively. It is interesting to observe (Table 3) that expansion of the N_k-bit secret key for 256-bit encryption requires an amount of basic operations which is inversely proportional to N_k. The number of round keys is equal for the three cases (14) and, with the KeyExpansion algorithm creating 14 round keys from a 256-bit secret key, their generation costs less than in the case of 128 or 192-bit ones.

As cited in Section 2.1, some observations can be done in order to improve Rijndael decryption procedure. We consider two ways to rewrite the InvMixColumns trasformation, one from Barreto presented in Section 4.1.3 of [9] and another proposed in [10] for an integrated desing of Rijndael. Figure 1 reports the achieved results of the comparison of 256-bit decryption (with 128, 192 or 256-bit key), by analysing also the two different transformations. Results corresponding to the decryption with the other possible sizes of block and key are almost identical. Clearly, the proposed techniques really improve the Rijndael decryption procedure, thus reducing the hardware complexity and saving operation resource.

Table 1: Rijndael encryption

	AND	OR	shifts (bytes)
128/128	1720	1268	408
128/192	2088	1540	496
128/256	2456	1812	584
192/128	3132	2310	744
192/192	3132	2310	744
192/256	3684	2718	876
256/128	4912	3624	1168
256/192	4912	3624	1168
256/256	4912	3624	1168

Table 2: Rijndael dencryption

	AND	OR	shifts (bytes)
128/128	5176	3860	1272
128/192	6312	4708	1552
128/256	7448	5556	1832
192/128	9468	7062	2328
192/192	9468	7062	2328
192/256	11172	8334	2748
256/128	14896	11112	3664
256/192	14896	11112	3664
256/256	14896	11112	3664

Table 3: Rijndael key schedule

	AND	OR	shifts (bytes)
128/128	340	290	120
128/192	384	288	96
128/256	430	299	84
192/128	630	543	228
192/192	598	431	132
192/256	678	471	132
256/128	986	841	348
256/192	950	703	228
256/256	924	630	168

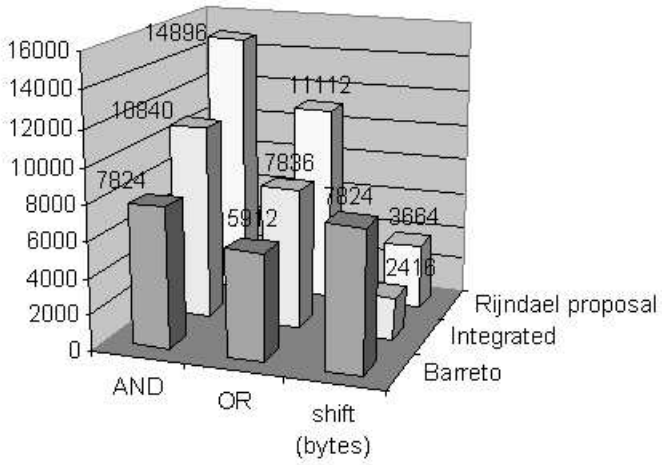

Figure 1: Different Rijndael decryption procedures

4.2 Camellia

In Table 4 we have reported the results regarding complexity evaluation of Camellia encryption/decryption procedure. Moreover, we propose also the results (marked by †) of a modification presented in Section C.2.6 of [3]. The P-function of the SPN-structure can be rewrote requiring an additional variable and 27 bytewise-XORs instead of 36. From a memory point of view this technique is worse than the standard one, but the computational cost gets much lower.

Table 4: Camellia encryption/decryption

	AND	OR	shifts (bytes)
128/128	2016	1032	16
128/128†	1692	870	16
128/192	2680	1376	24
128/192†	2248	1160	24
128/256	2680	1376	24
128/256†	2248	1160	24

Results concerning the key schedule are reported in Table 5. In addition, an optimization is proposed (marked by †), which combines the modification discussed above and a XOR cancellation property described in C.1.3 of [3].

Table 5: Camellia key schedule

	AND	OR	shifts (bytes)
128/128	448	576	352
128/128†	328	516	352
128/192	720	840	480
128/192†	580	770	480
128/256	720	840	480
128/256†	580	770	480

4.3 Shacal-2

Results about Shacal-2 are reported in Table 6, where "32-bit add." represents the addition over $GF(2^8)^4$.

Table 6: Shacal-2

	AND	OR	shifts (bytes)	32-bit add.
Encryption	4864	3328	1536	448
Decryption	4864	3328	1536	640
Key Expan	1536	1536	846	144

5 Comparisons

5.1 Camellia versus Rijndael

Some results regarding the comparison between Rijndael and Camellia are reported in Figures 2 and 3. In the complexity evaluation the shift operation entails lower cost in comparison with bytewise-AND/OR, therefore we focus on AND and OR operations. Considering both encryption/decryption and key schedule, it is possible to check Camellia's high speed (see Section 2.2). In particular, the comparison between hardware performance of Camellia and Rijndael reported in [4] is verified. For additional details on comparison between Camellia and Rijndael see [13].

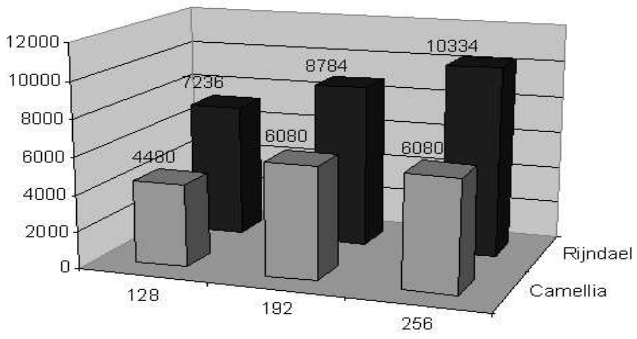

Figure 2: Comparison of AND operations

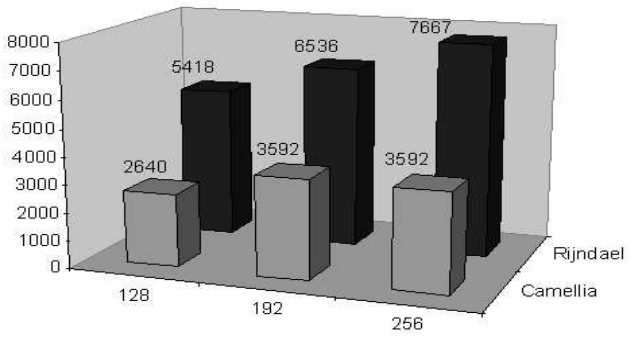

Figure 3: Comparison of OR operations

5.2 Shacal-2 versus Rijndael

Results regarding the comparison between Rijndael and Shacal-2 are presented in Figures 4, 5 and 6. Concerning the encryption procedure, Rijndael and Shacal-2 need a number of operations of the same order of magnitude, although the former is slightly better because of the amount of 32-bit additions. The comparison of the decryption procedures is more difficult. The overall complexity of Shacal-2, in fact, strongly depends on the actual implementation selected for the 32-bit adder, making the amount of required operations difficult to evaluate. Key schedule of Rijndael is much better than that of Shacal-2, Shacal-2 key schedule being quite expensive due to the use of the 512-bit key.

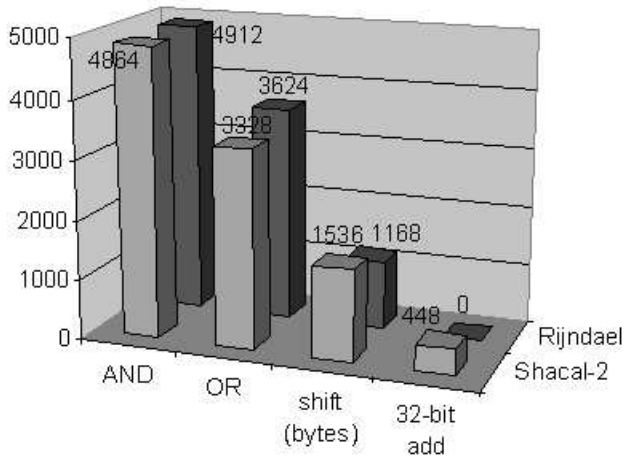

Figure 4: Rijndael vs Shacal-2 encryption

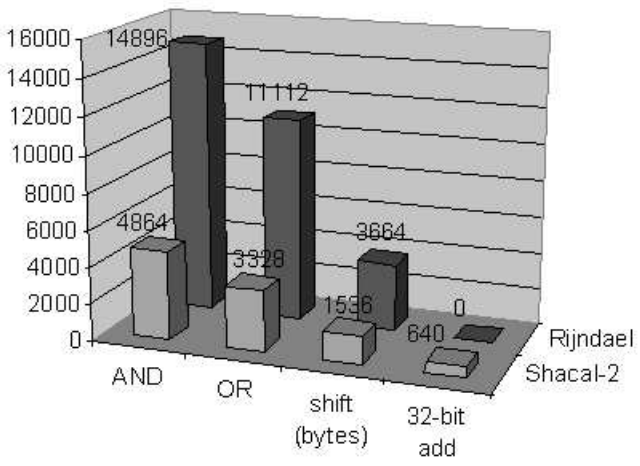

Figure 5: Rijndael vs Shacal-2 decryption

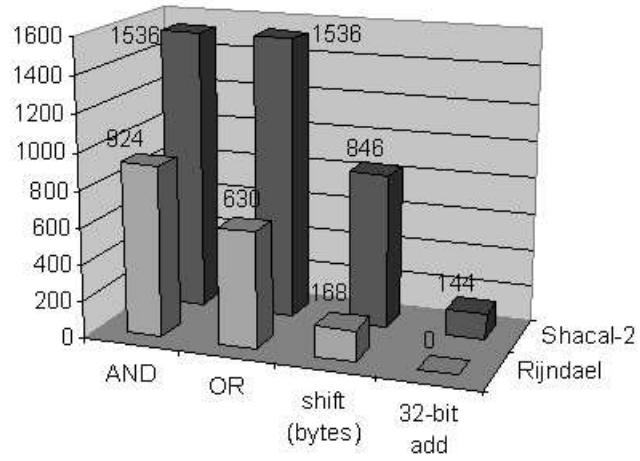

Figure 6: Rijndael vs Shacal-2 key expansion

5.3 Camellia, Rijndael and Shacal-2

In order to compare Rijndael, Camellia and Shacal-2, which support different block size, we consider a 256-bit block and, due to the lack of a specific 256-bit version of Camellia, we consider to use Camellia on both the 128-bit halves.

As written in 3.1, performance of the direct cipher is considered more im-

portant than that of the inverse (if different). In the following graph (Figure 7), the amount of bytewise-ANDs, bytewise-ORs, shifts of bytes and 32-bit additions needed by a 256-bit encryption and key schedule with the three algorithms is presented.

It is important to notice that the achieved results satisfy NESSIE considerations about finalists performances as reported in [14]: Camellia turns out to be close to Rijndael, Shacal-2 is ”quite good” , while Rijndael is ”very good and therefore it can effectively serve as a benchmark” for other block ciphers.

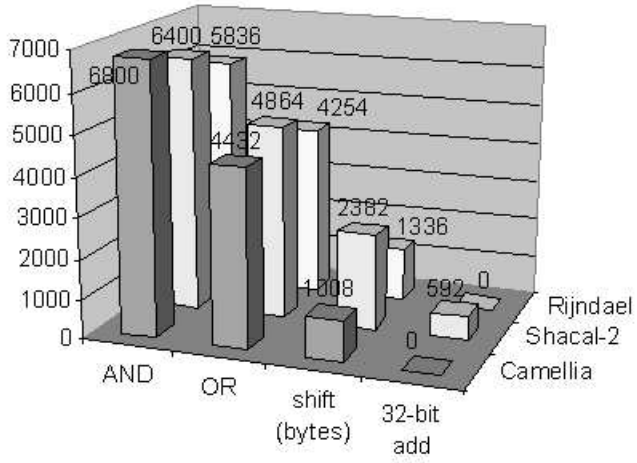

Figure 7: Rijndael, Camellia and Shacal-2

6 Conclusions

The paper presents a methodology for the evaluation of the complexity and computational cost of different block ciphers, in order to achieve independence from the platforms they are implemented on. An analysis of three of the block ciphers selected by NESSIE is provided: the complexity of Rijndael, Camellia and Shacal-2 is estimated through a ”step by step” analysis. The global amount of basic logical operations required for encryption and decryption are computed and some comparisons are presented.

Achieved results are consistent with other performance comparisons based on

actual cipher implementations reported in literature. In particular, NESSIE considerations about finalists performances are verified: Camellia is close to Rijndael, Shacal-2 is ”quite good”, while Rijndael is ”very good and therefore it can be effectively used as a benchmark”.

In conclusion, the presented work could be useful for the evaluation of the implementation complexity of such algorithms on small processors with limited processing power and, more in general, for comparison among block ciphers.

Acknowledgements The authors wish to thank Dr. Elisabetta Tecchio (University of Trento) for fruitful discussion and support about Camellia.

References

[1] *“NESSIE project announces final selection of crypto algorithms”*. February 27, 2003. www.cryptonessie.org.

[2] *“NESSIE Security Report”*. vers 2.0, February 2003. www.cryptonessie.org.

[3] K. Aoki, T. Ichikawa, M. Kanda, M. Matsui, S. Moriai, J. Nakajima, and T. Tokita. *“Specification of Camellia: A 128-bit Block Cipher”*. Nippon Telegraph and Telephone Corporation and Mitsubishi Electric Corporation, 2000.

[4] K. Aoki, T. Ichikawa, M. Kanda, M. Matsui, S. Moriai, J. Nakajima, and T. Tokita. *“Camellia: A 128-Bit Block Cipher Suitable for Multiple Platforms - Design and Analysis”*. Selected Areas in Cryptography, pag 39-56, 2000.

[5] H. Handschuh and D. Naccache *“Shacal”*. In B. Preneel, Ed., First Open NESSIE Workshop, Leuven, Belgium, November 13-14, 2000.

[6] A.J. Menezes, P.C. van Oorschot and S.A. Vanstone *“Handbook of Applied Cryptography”*. CRC Press, 1997.

[7] *“Performance of Optimized Implementations of the NESSIE Primitives”*. vers 2.0, February 2003. www.cryptonessie.org.

[8] J. Daemen and V. Rijmen. *“AES Proposal: Rijndael”*. version 2, 1999.

[9] J. Daemen and V. Rijmen. *“The Design of Rijndael”*. Springer, 2002.

[10] C. Lu and S. Tseng. *" Integrated Design of AES (Advanced Encryption Standard) Encrypter and Decrypter"*. Proc. of the IEEE International Conference on Application-Specific Systems, Architectures, and Processors (ASAP'02), 2002.

[11] R. Lidl and H. Niederreiter. *"Introduction to finite fields and their applications"*. Cambridge University, 1986.

[12] J. Worley, B. Worley, T. Christian, and C. Worley. *"AES Finalists on PA-RISC and IA-64: Implementations & Performance"*. csrc.nist.gov/encryption/aes/round2/˜conf3/papers/05-jworley.pdf.

[13] G. Boato, F. Granelli and E. Tecchio *"A methodology for numerical evaluation of the computational complexity of AES and Camellia"*. Proceedings of the Ninth International Conference on Distributed Multimedia Systems, Sept. 2003, Miami (USA), pp. 628-631.

[14] *"Update on the selection of algorithms for further investigation during the second round"*. March 2002. www.cryptonessie.org.

[15] R. Williams *"Computer Systems Architecture. A Networking Approach"*. Addison-Wesley, 2001.