Improving the Performance of the IDEA Cryptographic Algorithm Using FPGAs

JOSE M. GRANADO, MIGUEL A. VEGA, JUAN M. SANCHEZ, JUAN A. GOMEZ
Univ. Extremadura. Dept. Informatica
Escuela Politecnica. Campus Universitario s/n. 10071 Caceres. SPAIN
FAX:+34-927-257-202

Abstract: - Nowadays, the cryptography field is on the increase in the telecommunications world, because of this there is a constant need of more secure and efficient cryptographic algorithms. Thus, a lot of research is being done in order to try to improve the current algorithm performance. At present, one of the alternatives under research is the implementation of these algorithms in FPGAs (Field-Programmable Gate Arrays), which offer excellent features. In this work, we present a detailed research of the IDEA cryptographic algorithm implementation in Virtex FPGAs. Nine different hardware implementations are presented, which are compared with each other and with the algorithm software implementation. In addition, the conclusions of this detailed research and the possible future work lines are shown. In short, the implementation of the IDEA algorithm using FPGAs offers advantages over software implementation thanks to the use of the intrinsic parallelism (pipelining and replication), resulting in a performance that surpasses in 16 times the software version.

Key-Words: - Computer Security and Cryptography, IDEA Cryptographic Algorithm, FPGA (Field-Programmable Gate Array), Performance, Experience using Handel-C

1 Introduction

Nowadays, the information security has achieved a great importance, both when information is sent through a non-secure network (as Internet) and when data are stored in massive storage devices. The cryptographic algorithms are used in order to guarantee the security of data sent or stored. Among them we find the IDEA algorithm [1], the one used in our research. This is one of the most popular algorithms; for example, due to its use in the PGP (Pretty Good Privacy) system [2].

In this work we present a total of 9 implementations of IDEA algorithm, using reconfigurable hardware, in order to study the performance improvement provided by the use of an FPGA in the cryptographic algorithm implementation. In section 2, the IDEA algorithm is briefly described. Then, all the implementations performed and their results are explained in detail. Finally, in section 4, both the conclusions obtained and the future research lines are presented.

2 The IDEA Algorithm

IDEA (International Data Encryption Algorithm) is a coding/decoding algorithm of 64-bit text blocks, using a key of 128 bits (it is an algorithm of private key) that is used to generate 52 subkeys of 16 bits. The algorithm consists of 9 phases, 8 identical phases (figure 1(a)) and one last phase of transformation (figure 1(b)). The 64-bit block is propagated through each phase, divided into four 16-bit sub-blocks. See [1] for a more detailed explanation of the algorithm.

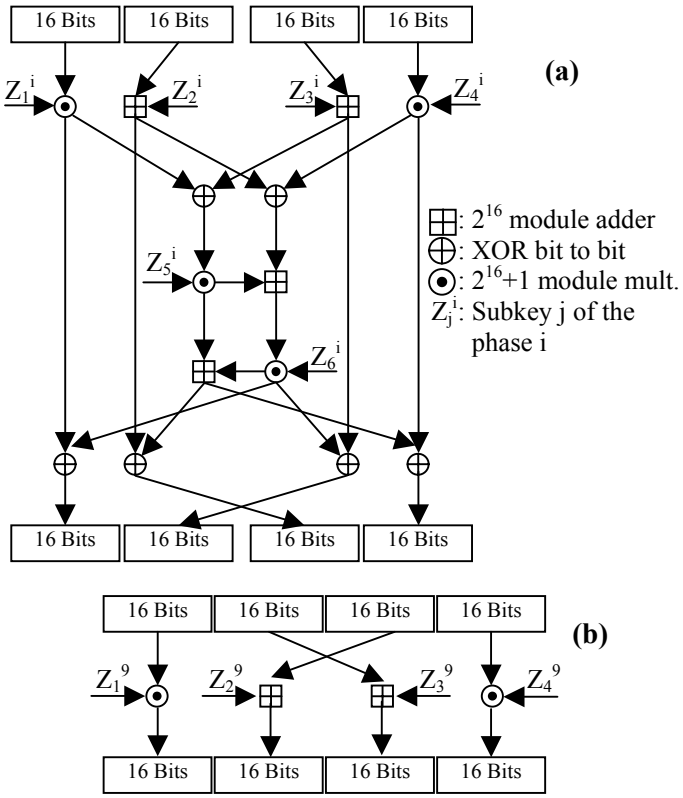

Fig. 1. (a) Structure of an algorithm's phase. (b) Structure of the algorithm's transformation phase.

As we can guess, the major problem lies in the multipliers, since, aside from taking a great amount of computation, they are executed 4 times in each phase. The improvement of this component is one of the more observed approaches in the literature. In our case, we will use the improvement of the Low-High algorithm [3] proposed by Biham [4].

3 Implementations and Results

3.1 Software Implementation

A software implementation of the IDEA algorithm has been made by means of the programming language Visual C++ that will serve as base to make performance and functionality comparisons with hardware versions. This implementation is strictly sequential and consists of a loop that, for each block to code, will cross the different phases of the IDEA algorithm (figure 1). Table 1 shows the results obtained by this version. It is important to emphasize that advanced optimizations have not been included in this version: assembler code, MMX technology, …

Processor	(Clock) Frequency	Performance
Pentium IV	1.7 GHz	3.861 Mbits/s

Table 1. Software implementation results.

3.2 Hardware Implementation

In this study, a Celoxica RC1000 board with a Xilinx Virtex-2000E FPGA is used. The different hardware versions implemented result from combining three types of algorithms with three types of communications between host (computer) and the FPGA. This gives a total of nine hardware versions. First, the host-FPGA communication types used will be shown, following of the algorithm types. Then, the results obtained by each of the implementations will be presented.

3.2.1 Host-FPGA Communication Types

Communication through RC1000's Memory Banks

This communication type establishes the RC1000's memory banks as the only means of data transference between the host (computer) and the FPGA, and besides, none other data storage auxiliary structure is defined. This is the reason why in order to process (code/decode) a block is first necessary to read it from the board's memory and, after operating with it, to write the result in the memory again. This makes the operation time of this algorithm type be highly dependent on the memory access time. In addition, due to the fact that each block is of 64 bits and the RC1000 board's memory stores 32-bit data, it is necessary to perform two readings and two writings per block to process, and this limits still more the performance of the algorithm that uses this communication type.

As it is observed in figure 2, the memory is an intermediary way in the data transference between the host and the FPGA. The arrows represent data transit.

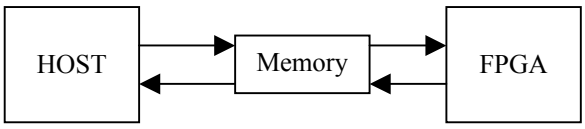

Fig. 2. Host-FPGA data transference using memory banks.

Communication by means of Control & Status Ports

In this communication type, the memory is eliminated and the information is transferred through the Control port (from the host to the FPGA) and Status one (from the FPGA to the host) of the RC1000 board. Due to the ports can only transmit 8-bit data, it is necessary to divide the blocks into eight segments of that size, and to perform for each block so many host→FPGA and FPGA→host transferences as segments into which the block is divided.

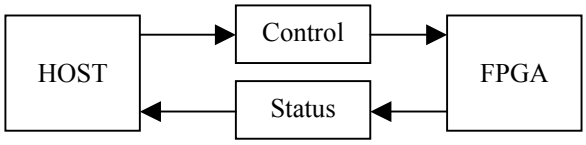

Fig. 3. Host-FPGA data transference using Control & Status ports.

The way in which the elements are distributed in the data communication is perfectly observed in figure 3, where arrows represent the information flow between the elements that they connect.

Like the previous case, the data transference and data processing are closely tied, being impossible to separate the transference and the processing. In conclusion, the processing (coding/decoding) of the blocks is penalized by the ports' transference speed.

Communication by means of Memory using an Internal Array

This type of communication, like the first type, uses the memory to perform the communication between the host and the FPGA, but with the proviso that, in this case, the FPGA should have an internal array implemented that will serve as auxiliary storage. Therefore, the data transference and data processing (coding/decoding) are totally separated.

As it can be observed in figure 4, the array serves as intermediary between the memory and the computation module, separating the processing of the communication. In this figure, the arrows represent data flows between elements. Therefore, for each *x* characters, where *x* stands for the size of the internal array (1496 for the sequential and sequential with replication versions, and 1200 for the pipelined version -see section 3.2.2-). The host sends data to the memory, and then the internal array receives those data coming from the memory. Afterwards, the computation module only uses the internal array, both to obtain the blocks to code/decode and to store the resulting blocks. Finally, the data pass from the array to the memory, and from this to the host. This process would be repeated if the size of the data to code/decode was larger than the array size.

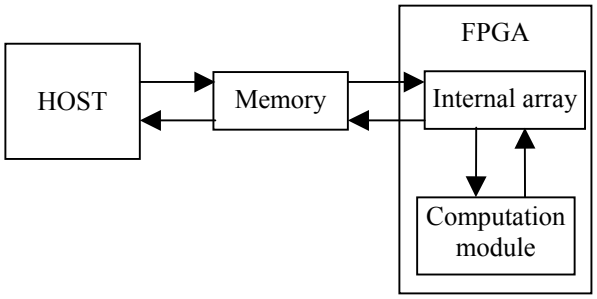

Fig. 4. Host-FPGA data transference by means of memory using an internal array.

This independence between communication and processing allows us to examine better the performance characteristics of the algorithm used, although the resources considerably increase because the internal array must be implemented within the FPGA.

3.2.2 Hardware Algorithm Types

Sequential Hardware Algorithm

This algorithm is similar to the software implementation, that is, it is strictly sequential, with the only advantage of being executed in a hardware device. In this algorithm type, before an operation is performed the previous one must have been finished, as it can be seen in figure 5. In this implementation type, the replication of the multipliers, significant elements of the design, is not made. However, the replication of both adders and XOR gates is allowed in order to simplify the design in Handel-C, although their executions are not performed in parallel.

Analyzing figure 5, the phase operation time (PT_{seq}) can be computed by means of equation 1, where *mt* represents the multiplier operation time, *at* refers to the adder operation time, *xt* stands for the XOR operation time, and *ct* represents the time that the signal takes going through the connections.

$$PT_{seq} = 4*mt + 4*at + 6*xt + ct \quad (1)$$

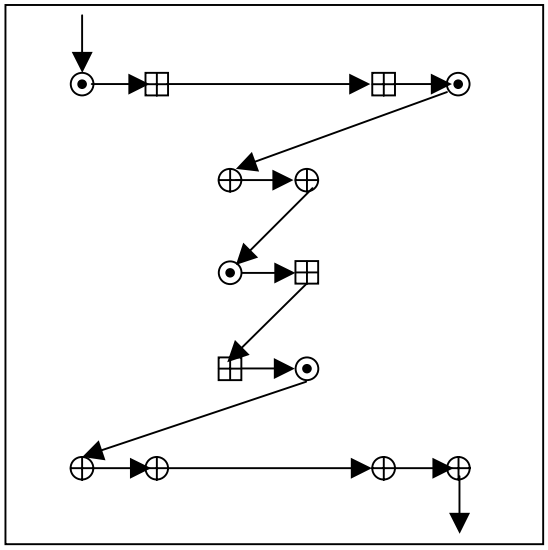

Fig. 5. Execution flow of sequential hardware algorithm.

Sequential Hardware Algorithm with Replication

In this algorithm type, the inherent parallelism of the IDEA algorithm phase is used. The idea is that, by means of hardware replication, some parts of each phase of the IDEA algorithm will be executed in parallel, as figure 6 shows.

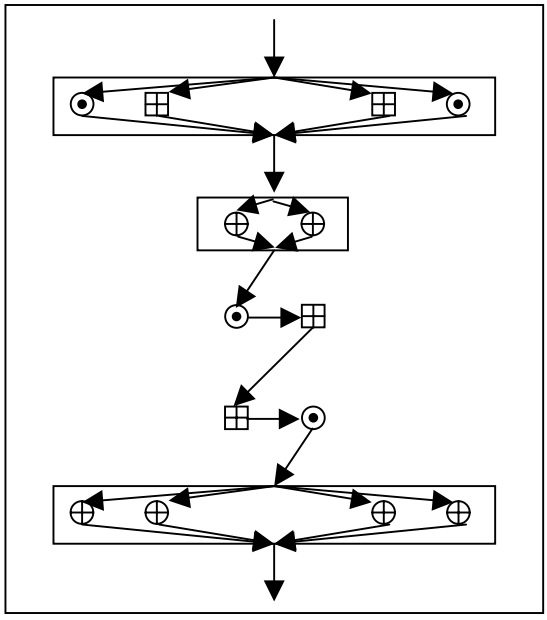

Fig. 6. Execution flow of sequential hardware algorithm with replication.

This produces a performance improvement compared with the previous version, in which all the operations are performed in sequential. As we can see

in figure 6, the operations in each box are executed in parallel. This is the reason why the operation time of each of these boxes is the major operation time of all its elements. Furthermore, we can verify that using two multipliers by phase is enough, since only two of them will be executed in parallel, the other two multiplications can be done by using the two existing multipliers.

The operation time of each phase (PT_{s+r}) of this algorithm can be computed like equation 2 shows, where *mt*, *at*, *xt* and *ct* represent the same values as in the previous equation. As it can be observed, the operation time between the sequential algorithm and the sequential algorithm with replication is reduced in (mt+2*at+4*xt). This result is obtained from subtracting equations 1 and 2. Therefore, the performance has been increased notably with a little-significant increase of hardware resources (just one more multiplier).

$$PT_{s+r} = 3*mt + 2*at + 2*xt + ct \qquad (2)$$

Phase-Pipelined Hardware Algorithm

Until now, we have seen two algorithm types in which, in order to code a block, all the IDEA algorithm's phases must be executed before being able to begin with another block. In the phase-pipelined algorithm, when a block leaves a phase and goes through the following one, the next block to code will enter the phase that the present block leaves. This is the reason why parallelism degree is increased by making a phase-level pipelining, as figure 7(b) presents.

All that is possible thanks to an ECB (Electronic CodeBook) version of the IDEA algorithm is being used, version in which the coding of a block is totally independent of the rest. In conclusion, this implementation increases the performance in such a way that a coded block is obtained in every phase operation time.

In order to make the pipelining, apart from replicating all the hardware relative to each phase, it is necessary to add between phases pipelining registers of the same size as the block, that is, 64 bits. Obviously, in this implementation, there is a notable increase of hardware resources, however the performance improvement also is outstanding.

Figure 7(a) shows the execution flow of the phase-pipelined version. We can observe that it is very similar to the sequential algorithm with replication (figure 6), where the only difference is that one of the multipliers has been removed from the parallel execution zone of the beginning of the phase. This decision has been taken to reduce the hardware used, that is, a single multiplier will be used for each phase. The phase operation time of this algorithm (PT_{pipe}) can be seen in equation 3. If we compare this equation with equation 2 of the previous algorithm, we see now the phase time increases in *mt*.

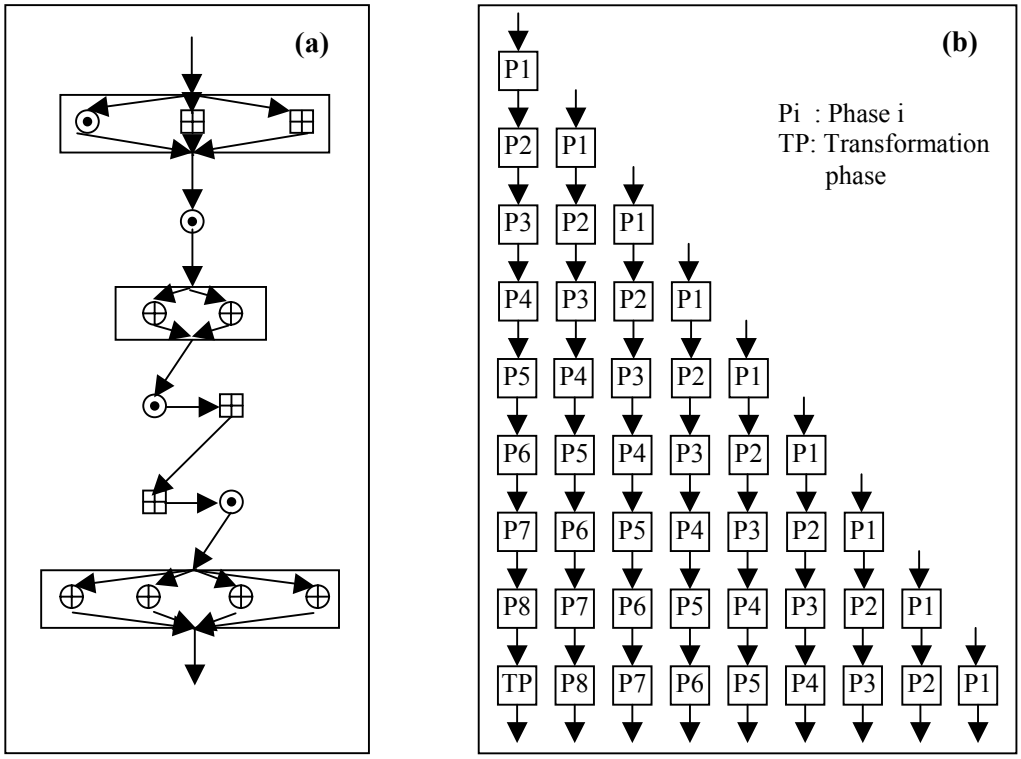

Fig. 7. (a) Execution flow of phase-pipelined hardware algorithm. (b) Pipeline of the hardware algorithm.

Nevertheless, the performance increase is obtained thanks to a block is coded in each PT_{pipe} (from the ninth block, since the pipeline is filled in the 8 previous cycles), whereas in the sequential algorithm with replication (as well as in the sequential one) a coded block is obtained from each 8 PT plus the time of the transformation phase. In short, the latency is nine times smaller.

$$PT_{pipe} = 4*mt + 2*at + 2*xt + ct \quad (3)$$

3.2.3 Obtained Hardware Results

In this section we study the results obtained by the different implemented hardware versions. The parameters of the elements used in our experiments are the following:

- Computer: 1.7-GHz Pentium IV with 768 MB of RAM.
- FPGA: Virtex XCV2000E with a –8 speed grade.
- FPGA clock: 20 MHz (always).
- Total number of FPGA slices: 19200.
- Coded data size: 31880 characters.

Resource use, Maximum frequency and Minimum period columns of table 2 have been obtained from the reports generated by the Flow Engine tool of Xilinx Foundation Series 4.1, after the synthesis of each hardware module. Therefore, they are real measurements on implementations already carried out, and not estimations. The Performance column of this table has been computed obtaining the average value after several (around 10) practical experiments. The nomenclature of the Algorithm column is the following. The prefix I represents the cryptographic algorithm used, in this case IDEA; then the type of algorithm used is indicated: S (Sequential), SR (Sequential with Replication) and P (Pipelined by phase); finally, the communication type used appears: MEM (memory banks), PORT and ARRAY.

Let us analyze the data of table 2 in detail. The first we can observe is the resource use of the FPGA. As it can be seen, both in the sequential implementation and sequential implementation with replication, if the communication is made by means of ports or memory banks, the occupation is small enough, since none expensive element is used. This does not happen in their respective phase-pipelined versions, where the resource use is tripled, due to the replication of the hardware necessary for each pipelining stage. In the versions where the internal array is used, we observe that all the implementations surpass a 90% of the FPGA occupation. This is due to the great size of the internal array. There is little difference between the sequential version and sequential version with replication as opposed to the phase-pipelined one, since this last contains a smaller array.

Algorithm	Resource Use (Slices)	Max. Frequency (MHz)	Min. Period (ns)	Perform. (Mbits/s)
I_S_MEM	2424 (12%)	24.311	41.134	5.175
I_SR_MEM	2654 (13%)	23.630	42.319	5.175
I_P_MEM	7760 (40%)	20.621	48.494	5.175
I_S_PORT	2449 (12%)	23.782	42.048	0.145
I_SR_PORT	2677 (13%)	21.180	47.215	0.145
I_P_PORT	7748 (40%)	20.443	48.917	0.145
I_S_ARRAY	18043 (93%)	23.408	42.721	7.846
I_SR_ARRAY	18297 (95%)	22.100	45.248	15.202
I_P_ARRAY	19198 (99%)	21.427	46.609	62.102

Table 2. Results obtained by hardware implementations.

As far as the maximum frequency and minimum period are concerned, it is not possible to emphasize anything, since in all the versions the data are very similar. The only thing to indicate here is that, in the phase-pipelined cases, a nine times smaller latency than in the versions without pipelining is obtained, increasing significantly the algorithm performance.

Finally, let us analyze the data relative to the performance. It can be observed that, in the cases where the communication is performed through the RC1000's ports or memory banks, the times of the three types of algorithms are equal. This is because the input/output time is so high that conceals the computation times (we should remember that 4 memory accesses per block are necessaries in the versions with communication through the memory banks, and 16 port accesses per block in the versions that use the RC1000's ports to communicate). It must be emphasized that the performance of the hardware versions that use the memory banks for communication (5.175 Mbits/s) is already greater than the performance of the software version (3.861 Mbits/s).

When we study the versions that use the internal array, we observe how the performance varies among the different algorithms, also seeing that the performances of the internal-array versions are much greater than those of the other versions. Furthermore, the performances of the internal-array versions grow in an exponential way when changing from an algorithm type to another, until reaching a performance in the FPGA pipelined version 16.084 times greater than in the software version. In addition, the performance for I_P_ARRAY implementation (62.102 Mbits/s) offered by the FPGA is higher than the one obtained by other authors, both using MMX technology [5] (32.9 Mbits/s) and by means of FPGAs [6] (0.447 Mbits/s) and [7] (1 Mbits/s), VLSI [8] (44 Mbits/s), and

diverse DSPs: Motorola 56166 [9] (1.25 Mbits/s), DEC SA-110 [10] (32 Mbits/s) and TI TMX320C6x [10] (53.1 Mbits/s).

4 Conclusions and Future Work

In this work, diverse hardware implementations of the IDEA cryptographic algorithm, using a Virtex-2000E FPGA and Handel-C, have been analyzed. The study has verified that the theoretical times computed in equations 1, 2 and 3 are fulfilled by the performance data of table 2.

Furthermore, we have demonstrated that, when we use hardware (the FPGA), the IDEA algorithm performance increases. Concretely, the obtained data indicate that the algorithm implementation by means of reconfigurable hardware (FPGAs) surpasses in more than 16 times the software version.

On the other hand, it is necessary to say that, we can still make more advanced improvements in the elements that compose the circuits of our hardware algorithms. More concretely, in the multipliers, critical elements of the design. In fact, although our performance results are better than those of other authors [5-10], some even higher result can be found in the literature, as the one obtained by CryptoBoost III [11] (200 Mbits/s).

This is one of the research lines that will be followed in the future, that is, to implement better multipliers at lower level, for example by using partial reconfiguration in order to design multipliers by constant. We also intend to increase the pipelining degree by dividing the IDEA algorithm phases into substages, so that we can augment the number of blocks that are computed in parallel, and thus increasing the performance. In this line, we also intend to make hardware implementations duplicating the data path, that is, having several separated execution lines that are executed in parallel, so that the performance increases notably too.

Finally, in the future, studies with other cryptographic algorithms will be done, as well as with other operation modes, such as the CBC (Cipher Block Chaining), the CFB (Cipher-FeedBack) and the OFB (Output-FeedBack).

5 Acknowledgments

This work has been supported in part by the Spanish Government under Grant TIC2002-04498-C05-01.

References:

[1] Schneier, B.: "Applied Cryptography". *John Wiley & Sons, 2nd edition,* 1996.

[2] Garfinkel, S.: "PGP: Pretty Good Privacy". *O'Reilly*, 1995.

[3] Lai, X.: "On the Design and Security of Block Ciphers". *ETH Series in Information Processing*, nº 1, Hartung-Gorre Verlag, Konstanz, 1992.

[4] Biham, E.: "Optimization of IDEA". *Technical Report, NESSIE document NES/DOC/TEC/WP6/026/1*, Computer Science Department, Technion - Israel Institute of Technology, Haifa, Israel, January 2002.

[5] Lipmaa, H.: "IDEA: A Cipher For Multimedia Architectures?", *Selected Areas in Cryptography'98, LNCS 1556*, Springer Verlag, August 1998, pp. 248-263.

[6] Caspi, E.; Weaver, N.: "IDEA as a Benchmark for Reconfigurable Computing". *Technical Report*, BRASS Research Group. University of Berkeley, December 1996.

[7] González, I.; Gómez, F. J.; Martínez, J.: "A HW/SW Co-Design Case Study: Implementing a Cryptographic Algorithm in a Reconfigurable Platform". *Proc. of the XVI Conference on Design of Circuits and Integrated Systems (DCIS'2001)*, Porto, Portugal, November 2001, pp. 504-509.

[8] Bonnenberg, H.; Curiger, A.; Felber, N.; Kaeslin, H.; Lai, X.: "VLSI Implementation of a New Block Cipher". *Proc. of the International Conference on Computer Design: VLSI in Computer and Processors*, Washington, USA, IEEE Computer Society Press, 1991, pp. 510-513.

[9] Madhusudan-Sastry, T. R.; Ganesan, T.; Madhukar, B.; Srinivasa, N.: "Time is Right for a Good, Secure IDEA". *Electronic Engineering Times*, October 1995.

[10] Mencer, O.; Morf, M.; Flynn, M. J.: "Hardware Software Tri-Design of Encryption for Mobile Communication Units". *Proc. of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, USA, vol. 5, May 1998, pp. 3045-3048.

[11] Beuchat, J. L.; Haenni, J. O.; Teuscher, C.; Gomez, F. J.; Restrepo, H. F.; Sanchez, E.: "Approches Matérielles et Logicielles de l'Algorithme IDEA". *Technique et Science Informatiques*, vol. 21, nº 2, 2002, pp. 203-204.