FPGA Implementation of Three IPSec Cryptographic Algorithms

JUAN M. DÍEZ, SLOBODAN BOJANIĆ*, CARLOS CARRERAS and OCTAVIO NIETO
Department of Electronic Engineering
Technical University of Madrid
Dpto. de Ingeniería Electrónica, E.T.S.I. de Telecomunicación, Ciudad Universitaria s/n, 28040 Madrid
SPAIN
{jmdiez, slobodan, carreras, nieto}@die.upm.es http://www.die.upm.es

Abstract: - The paper deals with FPGA implementation of three cryptographic algorithms compliant to IP Security Protocol (IPSec). The algorithms belong to three different classes of cryptographic primitives: block cyphers (CAST5 algorithm), stream cyphers (RC4) and hash functions (HMAC SHA-1). The target technology was FPGA family VirtexII. The different architectures were applied and their analysis is given. The throughput results for the given technology were 899.8 Mbps for SHA-1, 797.7 Mbps for CAST5 and 155.2 Mbps for RC4. Lower RC4 throughput was expected since it was particularly designed for software implementation. In applications with PCI board where PCI bus shares input and output lines, the throughput can be duplicated, thus reaching Gigabit rates for block cipher CAST5 (1.59 Gbps).

Key-Words: - FPGA, cryptography, IPSec, CAST5, RC4, HMAC SHA-1, VirtexII

1 Introduction

Applications such as electronic banking, electronic commerce, and Virtual Private Networks (VPNs) require an efficient and cost-effective way to protect the confidental data whose significantly increased traffic over the Internet have posed serious security problems. Internet Protocol Security standard (IPSec) protects the transferred data using cryptographic algorithms which are generally time-consuming thus causing a bottleneck in high speed networks. The implementation of a cryptographic algorithm must achieve high processing rate to fully utilize the available network bandwidth. To follow the variety and the rapid changes in algorithms and standards, a cryptographic implementation must also support different algorithms and be upgradeable in field.

Field Programmable Gate Arrays (FPGAs) constitute an appealing alternative for the implementation of encryption algorithms as they gather advantages from traditional software and hardware (ASIC) approaches. Software is considered to provide ease of use, ease of upgrade, portability, and flexibility. On the other hand, ASICs are considered more physically secure and have the potential to provide improved performance by means of specialized architectures. FPGAs also provide more physical security than software with the potential of improved performance, but the design and development costs are greatly reduced (in fabrication, ASICs remain cost efficient for high-volume productions). Besides, FPGAs are reconfigurable so they also support some features of software implementations like ease of upgrade and flexibility, very important in cryptographic applications [1].

In this work, three different types of cryptographic algorithms complient to IPSec were implemented in FPGA: stream cipher RC4, block cipher CAST5 and hash algorithm SHA-1. The algorithms are presented in Section 2; Section 3 describes the methodology; in Section 4, the results are given, and the conclusions are drawn in Section 5.

2 Cryptographic Algorithms

The IPSec is a set of protocols that use cryptographic algorithms within them. Two authentication (HMAC-MD5 and HMAC-SHA1) and seven encryption algorithms (DES, Triple DES, CAST-128, RC5, IDEA, Blowish and ARCFour as public implementation of RC4) have been specified to date [2]. The three algorithms implemented in this work belongs to threee different cryptographic classes: block ciphers (CAST5), stream ciphers (RC4) and hash algorithms (SHA-1).

Stream ciphers encrypt individual characters of a plaintext message one at a time, using an encryption transformation which varies with time. By contrast, block ciphers tend to simultaneously encrypt groups of characters of a plaintext message using a fixed encryption transformation. They are the most prominent and important elements in many cryptographic systems. Hash functions take a message as input and produce an output hash-value,

* sponsored by the Spanish Ministry of Science and Technology through the "Ramon y Cajal" program.

i.e. a hash function *h* maps bit-strings of arbitrary finite length to strings of fixed length, say *n*-bits [3].

2.1 RC4 algorithm

The alegged RC4 algorithm can be used with a variety of key lengths [4]. It specifically can be operated with 40-bit keys and with 128-bit keys. Key Setup:

1. Allocate an 256 element array of 8 bit bytes to be used as an S-box, label it S [0] .. S [255].
2. Initialize the S-box. Fill each entry first with it's index: S [0] = 0; S [1] = 1; etc. up to S [255] = 255;
3. Fill another array of the same size (256) with the key, repeating bytes as necessary:
   for (i = 0; i < 256; i = i + 1) S2 [i] = key [i % keylen];
4. Set j to zero and initialize the S-box like this:

```
for (i = 0; i < 256; i = i + 1)
    {
    j = (j + S [i] + S2 [i]) % 256;
    temp = S [i];
    S [i] = S [j];
    S [j] = temp;
    }
```

5. Initialize i and j to zero.

For either encryption or decryption, the input text is processed one byte at a time. A pseudorandom byte K is generated:

```
i = (i+1) % 256;
j = (j + S[i]) % 256;
temp = S [i];
S [i] = S [j];
S [j] = temp;
t = (S [i] + S [j]) % 256;
K = S [t];
```

To encrypt, XOR the value K with the next byte of the plaintext. To decrypt, XOR the value K with the next byte of the ciphertext.

2.2 CAST5 algorithm

CAST5 belongs to the class of Feistel ciphers thus it is similar to the Data Encryption Standard (DES). It is a 12- or 16-round cipher that has a blocksize of 64 bits and a keysize of up to 128 bits [5].

There are four encryption steps where plaintext $m_1...m_{64}$ and key $K= k_1...k_{128}$ are inputs and ciphertext $c_1...c_{64}$ is output:

1. (Key schedule) Compute 16 pairs of subkeys $\{K_{mi}, K_{ri}\}$ from K
2. (L_0,R_0) <-- $(m_1...m_{64})$. Split the plaintext into left and right 32-bit halves $L_0 = m_1...m_{32}$ and $R_0 = m_{33}...m_{64}$.
3. (16 rounds) for i from 1 to 16, compute L_i and R_i as follows: $L_i = R_{i-1}$; $R_i = L_{i-1} \wedge f(R_{i-1},K_{mi},K_{ri})$.
4. $c_1...c_{64}$ <-- (R_{16},L_{16}). Exchange final blocks L_{16}, R_{16} and concatenate to form the ciphertext.

Decryption is identical to encryption as explained above, except that the rounds are used in reverse order to compute (L_0,R_0) from (R_{16},L_{16}).

The algorithm allows a key size that can vary from 40 bits to 128 bits in 8-bit increments. For key sizes greater than 80 bits, the algorithm uses the full 16 rounds, otherwise it uses 12 rounds. For key sizes less than 128 bits, the key is padded with zero bytes.

2.3 SHA-1 algorithm

Secure Hash Algorithm (SHA-1) computes a condensed representation (160 bits) of a message or a data file of length $< 2^{64}$ [6, 7].

First, the message is padded to a multiple of 512 bits in length: append a 1 to the message, then add as many zeroes as necessary to reach the target length. This target length is the next possible length that is 64 bits less than a whole multiple of 512 bits. Finally, as a 64-bit binary number, append the original length of the message in bits.

A sequence of functions f_t, 0<= t <=79 is used that operate on three 32-bit words B, C, D and produce a 32-bit word as output:

f_t(B,C,D)= (B and C) or ((not B) and D) (0<= t<= 19)

f_t(B,C,D) = B xor C xor D (20 <= t <= 39)

f_t(B,C,D) =

(B and C) or (B and D) or (C and D) (40<=t <=59)

f_t(B,C,D) = B xor C xor D (60 <= t <= 79).

A sequence of constant words K(0), K(1), ... , K(79) is used: K = 5A827999 (0<= t <=19), K_t = 6ED9EBA1 (20 <= t <= 39), K_t = 8F1BBCDC (40 <= t <= 59), K_t = CA62C1D6 (60 <= t <= 79).

The digest is computed using the final padded message. The computation uses two buffers of five 32-bit words A,B,C,D,E and H_0,H_1,H_2,H_3,H_4, a sequence of eighty 32-bit words labeled W_0, W_1,..., W_{79} and a single word buffer TEMP.

To generate the message digest, the 16-word blocks M_1, M_2,..., M_n are processed in 80 steps. Before processing, the {Hi} are initialized: H_0 = 67452301, H_1 = EFCDAB89, H_2 = 98BADCFE, H_3 = 10325476, H_4 = C3D2E1F0.

1. Divide M_i into 16 words W_0, W_1, ... , W_{15}, where W_0 is the left-most word.
2. For t = 16 to 79 let Wt = S_1(Wt-$_3$ xor Wt$_{-8}$ xor Wt-14 xor Wt-16).
3. Let A = H_0, B = H_1, C = H_2, D = H_3, E = H_4.
4. For t = 0 to 79 do TEMP = S_5(A) + f_t(B,C,D) + E + W_t + K_t; E = D; D = C; C = S_{30}(B); B = A; A = TEMP;

5. Let $H_0 = H_0 + A$, $H_1 = H_1 + B$, $H_2 = H_2 + C$, $H_3 = H_3 + D$, $H_4 = H_4 + E$.

After processing Mn, the digest is the 160-bit string represented by the 5 words: H0 H1 H2 H3 H4.

3 Methodology

The design methodology starts with an in-depth analysis of each algorithm to determine the most appropriate architecture for its implementation. The different architectural alternatives considered in this study are explained in a following section. The analysis includes from a high-level evaluation of the different alternatives, to low-level timing analysis and resource allocation aimed to support fully utilized resources whenever possible, while maintaining maximum throughput (for example, in pipeline design or memory design).

From the result of architectural design, each algorithm is implemented as an RTL description using VHDL as specification language. These descriptions correspond to the iterative part of each algorithm, where a body of operations is repeated in a loop a number of times for each input data word. The goal is to obtain implementations that display the performance that can be obtained in FPGAs. Therefore, all circuitry aimed to support key modification and key scheduling in CAST-5 is left out of the evaluation (they are assumed to be preloaded in memory).

The same target FPGA, the Xilinx Virtex 2V3000, has been used for the three implementations [8]. The reason for this selection was to provide enough resources for the implementations. Therefore, feasibility, rather than low cost, has been the basis for the approach taken here. In consequence, it is expected that refining the current design at a lower level of abstraction (using the low level resources of the FPGA) can eventually improve the promising results presented here. It should be considered that all the designs only use part of the FPGA, so it is expected that design refinement will allow smaller FPGAs and, therefore, reduced cost.

The design process continues with the synthesis of the specifications using FPGA Compiler II, version 3.6.1 by Synopsys. The synthesis already provides estimates that can help to refine the RTL descriptions for improved performance. In the same vein, the synthesis tool is configured to optimize the throughput of the implementation. However, it should be noted that this tool performs a general optimization without taking into account the fine details of the particular FPGA architecture and the available resources. Therefore, although the implementations will provide useful results for evaluating the FPGA behavior with encryption algorithms, they are not expected to be quasi-optimal, thus leaving room for further improvements in perfomance.

Place and route is performed using ISE 4.1i by Xilinx. This provides accurate estimates of delays and FPGA occupation which are the basis of the results presented later for each implementation. Finally, timing and functional verification are performed through simulation.

As mentioned above, the speed of encryption (throughput) is the main objective of the designs. This speed is expressed as encrypted bits per second. The number of FPGA resources actually used in the implementation of each algorithm is also of great interest. This usage is expressed as the number of CLBs required and describes the occupation of the FPGA. A partially occupied FPGA can leave room for the implementation of other algorithms (or copies of the same algorithm) operating in parallel, thus allowing increased throughput with the same dedicated resources (the complete FPGA).

4 Architectural design

In general, these cryptographic algorithms are based on a loop with a simple function in the body which is executed multiple times. This is a characteristic which is common to other algorithms of digital signal processing [9]. They share a common analysis methodology which has been applied to the hardware implementation of a large number of applications with different optimizations depending of the characteristics of the particular algorithm [9,10]. The design based on FPGA's constitutes a flexible environment where it is possible to evaluate easily different architectures and algorithms.

One of the drawbacks of FPGAs is their limited clock frequencies, particularly when compared to the operating frequencies of today's processors executing software implementations. In this sense, the potential for improved performance mainly depends on the degree of parallelization allowed by the encryption algorithm. Since each algorithm behaves differently in software and in hardware, the implementations presented in this paper not only characterize the behavior of FPGAs, but also characterize the algorihtms in terms of their orientation towards hardware.

The initial approach is based on an architecture divided in a data-path and a controller. The data-path consists of the implementation of a single loop round and the control logic, which controls a simple counter to iterate the loop and some multiplexers to select the input data and the different subkeys or the necessary

data. This basic architecture (also called iterative looping) minimizes the hardware area but the controller must iterate several times to perform the complete loop and it may require a large number of clock cycles. The performance of this architectures can be considered low and the advantages of hardware implementation are wasted.

To improve the throughput of the architecture the following two basic options can be applied:

- loop unrolling and parallelization of resources,
- pipelining.

A loop unrolling architecture implements multiple rounds as a single combinational logic block. The number of required clock cycles is divided by the number of unrolled rounds. On the other hand, the delay of this combinational block is increased due to its larger complexity. However, depending on the operations in the body, this increment can be smaller than the sum of individual delays of the unrolled rounds. In this case, the product of the number of cycles by the delay of the block is reduced and, therefore the total time of loop execution is also reduced. That is possible when some of the body operations can be executed in parallel, and the improvement of performance depends on the degree or parallelization obtained. If parallelism can not be applied, this methodology does not provide any performance enhancement. The hardware area is larger than the non-optimized architecture, since the resources are replicated according to the number of unrolled rounds implemented in the combinational logic block.

A pipelined architecture is the other option. This technique is widely used because it is easy to apply and it offers the advantage of high throughput by increasing the number of data that are being simultaneously processed (parallelism in time). The implementation of a pipeline consists of a full unrolled loop divided in stages by register banks. Every stage can process different data. The hardware area is the largest of the tree analyzed architectures because there is a stage for every round of the loop.

Another important consideration about pipelining has to be taken into account. The processing of input data will finish several cycles after data have been put in the pipeline (depending of the number of stages used in the pipeline). Two input data must be independent in order to be processed simultaneously in the pipeline architecture. This drawback limits the use of pipelined implementations of a single round in algorithms where temporal results are fed back to following iterations, so they can not be executed independently. However, this methodology can be applied to the global loop when the algorithm has modes of operations which do not require feedback of the loop outputs, such as Electronic Code Book.

What follows is an analysis of the three algorithms which are examples of the architectural approaches commented in this section.

4.1 RC4 implementation

This algorithm is probably the simplest but this simplicity complicates the performance of its hardware implementation. In short, the algorithm performes five consecutive S-box accesses (3 Reads and 2 Writes) and the data obtained in an access are used to calculate the index for the next access. There is no possibility of parallelism and the initial iterative looping architecture without optimizations must be chosen.

The critical element of the design is the implementation of the S-box memory array and two design alternatives have been evaluated. The first one, is an implementation using a dual-port RAM block and tree clock cycles per stream generation. In the second one, a register array allowing one cycle per stream is implemented. The first alternative needs an excessive high clock frequency to obtain acceptable throughput so it was discarded. The large registers array used by the second option exceed the synthesis capacity of the FPGA Compiler II tool. A bit-slice description of the array was necessary to carry out the synthesis process. A RAM Block was used to store the key.

4.2 CAST5 implementation

This algorithm is a block cipher and a pipelined architecture was selected for the implementation. The decision was made assuming a non-feedback mode of operation. As commented in the previous section, a classic pipelined architecture requires the implementation of a resource for each operation. So for example, to allow 16 accesses per S-box, 8x4 dual-port ROM blocks are required. In our case, to get around this disadvantage, a modified pipelined architecture was used to reduce the number of required resources, specially the ROM blocks.

The basic idea is to implement a pipeline where the number of stages is multiplied by n and to introduce a new block data once every n cycles. Then, the pipeline is partially empty and their resources are used only every n cycles. Therefore, it is possible to reuse these resources in several pipeline stages, n-times at the most. An architecture with n=8 was implemented. The resulting number of shared resources versus the number in the original pipeline is shown in Table 1. The number of ROM blocks has

been minimized to one per S-box and the arithmetic operator have been reduced but not minimized, because a greater reduction complicates excessively the design and the synchronization of the shared resources.

Table 1. The resources in two architectures

	Partially empty	Original pipeline
adders	8	22
substracters	8	21
decrementers	4	16
ROM blocks	4	32

4.3 SHA-1 implementation

Since message digest algorithm SHA-1 operates in a feedback mode, a pipelined structure can not be applied. Therefore, a partial unrolling architecture was chosen for the implementation.

The critical path of the SHA-1 round is the calculation of the new data A following the next equation: $A_{new}= S_5(A)+F_t(B,C,D)+E+W_t+K_t$. The first operand represents the feedback between the round and the non-optimizable path of the unrolled rounds. The other operands are available before and their addition can be made in parallel with the calculation of data A of the previous internal round. The cost of unrolling a new round is only the delay of one addition and not the four additions of a complete round.

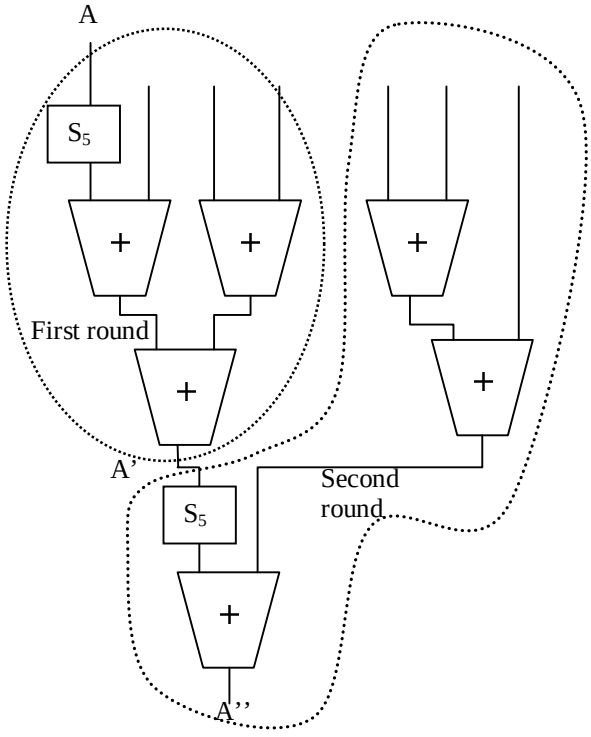

Figure 1. Structure of two unrolled SHA-1 rounds.

Figure 1. explains graphically the application of parallelism in a structure of two unrolled rounds.We select a combinational block of 4 rounds for the implementation. The design process is similar to the case shown in the figure.

5 Synthesis results

The designs were synthesized, placed and routed on the VirtexII family. The FPGA implementation results are shown in Table 2, and the obtained performance in Table 3. These data were obtained from the report files of the Xilinx tools.

Table 2. Implementation resources

	IOB´s	Slices	RAM blocks	Equivalent gates
RC4	46	6591	1	163,406
CAST5	133	5370	4	345,791
SHA-1	72	1550	0	29,900

The throughput is calculated as [10]:
Throughput = (number of bits processed per loop) x (Clock Frequency) / (number of cycles per loop)

Table 3. Implementation performances

	bits	cycles	clock freq. (MHz)	throughput (Mbps)
RC4	8	1	19.4	155.2
CAST5	64	8	99.7	797.7
SHA-1	512	22	38.6	899.8

The performance of the implementation of RC4 exhibits lower rates (155.2 Mbps throughput). This result was expected because the structure of this algorithm is not adequate for a hardware solution. The critical elements of RC4 are the output selection multiplexers of the register array. The synthesis reports show a complex and slower logic due to the large number of inputs (256) of the multiplexers.

Generally, stream ciphers are faster than block ciphers in hardware and have less complex hardware circuitry. While LSFR-based stream ciphers are well suited for hardware implementation, they are not especially amenable for software implementation. This has led to several proposals of stream ciphers (RC4, SEAL) designed particularly for fast software implementation on general purpose processors, not for programmable logic [3].

Therefore, RC4 is not efficient in hardware ie. it is measurably faster in software [11]. RC4 contains many structural and data hazards, forcing

serialization. It also contains many register adds, which are slow due to carry propagation.

On the other hand, the results of SHA-1 and CAST5 are much more promising, reaching 899.8 Mbps and 797.7 Mbps respectively. The CAST5 throughput shown in Table 3. is the bitrate of input flow. In some applications, for example in PCI-boards, the sum of the input and the output flow have to be considered because the PCI bus shares the input and output lines. In this case, the total throughput becomes 1.59 Gbps.

The critical components of the CAST5 implementation are the 32-bit circular shifters. Synthesis results show a delay due to logic cells of 4 ns, but after the placement and routing processes, the global delay nets is increased to 10 ns. This different are due to routing nets, which represent 60% of the critical path delay. These elements are not taken into account by the synthesizer, thus causing the reported difference.

The comparison with other implementations can be based on some reported results of 3.3 MBytes/sec on a 150 MHz Pentium processor for CAST cipher [5], and of 87 MB/s at 88 MHz for SHA-256 on a Xilinx Virtex XCV300E-8 FPGA [13].

6 Conclusion

Three cryptographic algorithms compliant to IP Security Protocol (IPSec) were implemented in FPGA (Field Programmable Gate Array). The algorithms belong to three different classes of cryptographic primitives: block cyphers (CAST5 algorithm), stream cyphers (RC4) and hash functions (HMAC SHA-1). The target technology was FPGA family VirtexII which is compliant with PCI-X 133 and PCI 66 MHz standars allowing the design of applications based on coprocessor boards. Different architectures like loop unrolling with parallel resources and pipelining were applied and their analysis has been provided.

The throughput results for the given technology were 899.8 Mbps for SHA-1, 797.7 Mbps for CAST5 and 155.2 Mbps for RC4. The lower RC4 throughput was expected since this algorithm was designed particularly for fast software implementations, and, therefore, it is not expected to be efficient in hardware. In applications with PCI boards where the PCI bus shares input and output lines, the computed throughput can be duplicated, thus reaching Gigabit rates for block cipher CAST5 (1,59 Gbps).

Acknowledgements

This work has been partially supported by the "Comisión Interministerial de Ciencia y Tecnología" (CICYT) under project number TIC2000-1395-C02-01.

References:

[1] R. Doud, "Hardware Crypto Solutions Boost VPN", *Electronic Engineering Times*, pp. 57-64, April 12 1999.

[2] IP Security Protocol (IPSec) Chapter – Latest RFCs and Internet Drafts for IPsec, http: //ietf.org/htmlchapters/ipsec-chapter.html

[3] A. J. Menezes, P. C. van Oorshot and S. A. Vanstone, *Handbook of Apllied Cryptography*, CRC Press, 1996.

[4] B. Schneier, *Applied Cryptography*, New York, USA: John Wiley Sons Inc., 2nd ed., 1996.

[5] C. Adams, *CAST-128 Encryption Algorithms*, RFC 2144, 1997.

[6] NIST, Secure Hash Standard, FIPS PUB 180-1, 1995.

[7] National Insitute of Standards and Technology, *The keyed-Hash Message Authentication Code (HMAC)*, Federal Information Processing Standards Publication # HMAC, 2001.

[8] "Virtex-II Platform FPGA Data Sheet". Xilinx, Inc. 2001.

[9] K. K Parhi. "VLSI Digital Signal Processing Systems: Design and Implementation". John Wiley & Sons, Inc. 1999.

[10] A. J. Elbirt, W. Yip, B. Chetwynd and C. Paar. "An FPGA-Based Performance Evaluation of the AES Block Cipher Candidate Algorithm Finalists". IEEE Transactions on VLSI, Volume 9 Issue 4 Aug 2001 pag 547-557.

[11] I. Goldberg and D. Wagner, "Architectural Considerations for Cryptanalytic Hardware", CS252 Report <http://www.cs.berkeley.edu/iang/isaac/hardware/>, May 1996.

[12] M. Riaz and H. Heyes, "The FPGA Implementation of RC6 and CAST-256 Encryption Algorithms", *Proc. IEEE 1999 Canadian Conference on Electrical and Computer Engineering*, (Edmonton, Alberta, Canada), March 1999.

[13] K. T. Ting, S. C. L. Yuen, K. H. Lee and P. H. W. Leong, "An FPGA based SHA-256 Processor", *12th International Conference on Field Programmable Logic and Application*, Montpellier, France, September 2002.