Experimental Testing of the Gigabit IPSec-Compliant Implementations of Rijndael and Triple DES Using SLAAC-1V FPGA Accelerator Board

Pawel Chodowiec[1], Kris Gaj[1], Peter Bellows[2], and Brian Schott[2]

[1] Electrical and Computer Engineering, George Mason University, 4400 University Drive, Fairfax, VA 22030
{kgaj, pchodow1}@gmu.edu
[2] University of Southern California - Information Sciences Institute
Arlington, VA 22203
{pbellows, bschott}@east.isi.edu

Abstract. In this paper, we present the results of the first phase of a project aimed at implementing a full suite of IPSec cryptographic transformations in reconfigurable hardware. Full implementations of the new Advanced Encryption Standard, Rijndael, and the older American federal standard, Triple DES, were developed and experimentally tested using the SLAAC-1V FPGA accelerator board, based on Xilinx Virtex 1000 devices. The experimental clock frequencies were equal to 91 MHz for Triple DES, and 52 MHz for Rijndael. This translates to the throughputs of 116 Mbit/s for Triple DES, and 577, 488, and 423 Mbit/s for Rijndael with 128-, 192-, and 256-bit keys respectively. We also demonstrate a capability to enhance our circuit to handle the encryption and decryption throughputs of over 1 Gbit/s regardless of the chosen algorithm. Our estimates show that this gigabit-rate, double-algorithm, encryption/decryption circuit will fit in one Virtex 1000 FPGA taking approximately 80% of the area.

1. Introduction

IPSec is a set of protocols for protecting communication through the Internet at the IP (Internet Protocol) Layer [15, 22]. One of the primary applications of this protocol is an implementation of Virtual Private Networks (VPNs). In IPSec Tunnel Mode, multiple private local area networks are connected through the Internet as shown in Fig. 1a. Since the Internet is an untrustworthy network, a secure tunnel must be created between security gateways (such as firewalls or routers) belonging to private networks involved in the communication. The information passing through the secure tunnel is encrypted and authenticated. Additionally, the original IP header, containing the sender's and receiver's addresses is also encrypted, and replaced by a new header including only information about the security gateway addresses. This way a limited resistance against the traffic control analysis is accomplished. A second use of IPSec is client-to-server or peer-to-peer encryption and authentication (see Fig. 1b). In IPSec Transport Mode , many independent pair-wise encryption sessions may exist

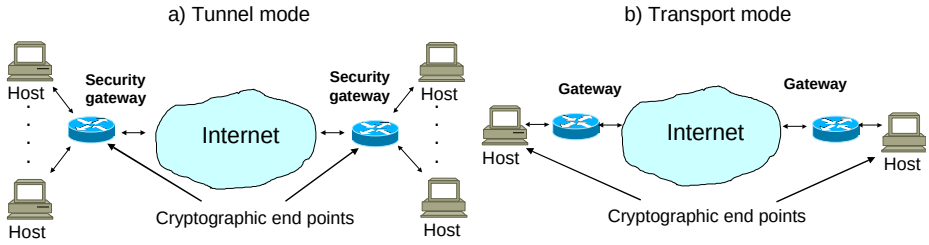

Fig. 1. IPSec Tunnel and Transport Modes

simultaneously. *The large number of connections and high bandwidth supported by a single security gateway or server suggests the use of hardware accelerators for implementing cryptographic transformations.*

The suite of cryptographic algorithms used for encryption and authentication in IPSec is constantly evolving. In the case of encryption, current implementations of IPSec are required to support DES, and have the option of supporting Triple DES, RC5, IDEA, Blowfish, and CAST-128. Since DES has been shown to be vulnerable to an exhaustive key-search attack using the computational resources of a single corporation [2], the current implementations of IPSec typically support Triple DES. In 1997, the National Institute of Standards and Technology (NIST) initiated an effort towards developing a new encryption standard, called AES (Advanced Encryption Standard) [1]. The development of the new standard was organized in the form of a contest coordinated by NIST. In October 2000, Rijndael was announced as the winner of the contest and a future Advanced Encryption Standard. In November 2000, a first Internet-draft was issued, proposing including AES-Rijndael as a required encryption algorithm in IPSec, with the remaining AES contest finalists as optional algorithms to be used in selected applications [11].

An encryption algorithm is not the only part of IPSec that is currently being extended and modified. Other modifications currently being considered include different modes of operation for encryption algorithms [18], hash functions used by authentication algorithms [21], type and parameters of public key cryptosystems used by a key management protocol, etc. *The fast and hard to predict evolution of IPSec algorithms leads naturally to prototype and commercial implementations based on reconfigurable hardware.*

An FPGA implementation can be easily upgraded to incorporate any protocol changes without the need for expensive and time-consuming physical design, fabrication, and testing required in case of ASICs. Additional capabilities appear when an FPGA accelerator supports a real-time partial reconfiguration. In this case, the accelerator can reconfigure itself on the fly to adapt to

- traffic conditions (*e.g.*, by changing the number of packet streams processed simultaneously),
- phase of the protocol (*e.g.*, by using the same FPGA with time sharing for implementing key exchange, encryption, and authentication),
- various key sizes and parameter values (*e.g.*, by adjusting the circuit architecture to different key sizes and different values of system parameters).

Additionally, several optional IPSec-compliant encryption, authentication, and key exchange algorithms can be implemented, and their bitstreams stored in the cache memory on the FPGA board. Algorithm agility accomplished this way can substantially increase the system interoperability.

In this paper, we present the results of the first phase of our project aimed at implementing a full suite of IPSec cryptographic transformations using SLAAC-1V FPGA board. In this phase, two encryption algorithms AES-Rijndael and Triple DES were implemented and experimentally tested in our environment.

2. FPGA Board

The SLAAC-1V PCI board is an FPGA-based computation accelerator developed under a DARPA-funded project called Systems-Level Applications of Adaptive Computing (SLAAC). This project, led by USC Information Sciences Institute (ISI), investigated the use of adaptive computing platforms for open, scalable, heterogeneous cluster-based computing on high-speed networks. Under the SLAAC project, ISI developed several FPGA-based computing platforms and a high-level distributed programming model for FPGA-accelerated cluster computing [16]. About a dozen universities and research labs are using SLAAC-1V for a variety of signal and image processing applications.

The SLAAC-1V board architecture is based on three user-programmable Xilinx Virtex XCV-1000-6 FPGA devices. Each of these devices is composed of 12,288 basic logic cells referred to as CLB (Configurable Logic Block) slices, and includes 32 4-kbit blocks of synchronous, dual-ported RAM. All devices can achieve synchronous system clock rates up to 200 MHz, including input/output interface.

The logical architecture of SLAAC-1V is shown in Fig. 2. The three Virtex 1000 FPGAs (denoted as X0, X1, and X2) are the primary processing elements. They are connected by a 72-bit "ring" path as well as a 72-bit shared bus. The width of both buses supports an 8-bit control tag associated with each 64-bit data word. The direction of each line of both buses can be controlled independently. The processing elements are connected to ten 256K x 36-bit SRAMs (Static Random Access Memories) located on mezzanine cards. The FPGAs X1 and X2 are each connected to four SRAMs, while X0 is connected to two. The memory cards have passive bus switches that allow the host to directly access all memories through X0.

About 20% of the resources in the X0 FPGA are devoted to the PCI interface and board control module. The remaining logic of this device (as well as the entire X1 and X2 FPGAs) can be used by the application developer. The 32/33 control module release uses the Xilinx 32-bit 33MHz PCI core. The control module provides high-speed DMA (Direct Memory Access), data buffering, clock control (including single-stepping and frequency synthesis from 1 to 200 MHz), user-programmable interrupts, etc. The current 32/33 control module has obtained DMA transfer rates of over 1 Gbit/s (125 MB/s) from the host memory, very near the PCI theoretical maximum. The bandwidth for SLAAC-1V using the 64-bit 66MHz PCI controller (using the Xilinx 64-bit 66MHz core) has been measured at 2.2 Gbit/s. The user's design located in X0 is connected to the PCI core via two 256-deep, 64-bit wide FIFOs. The DMA

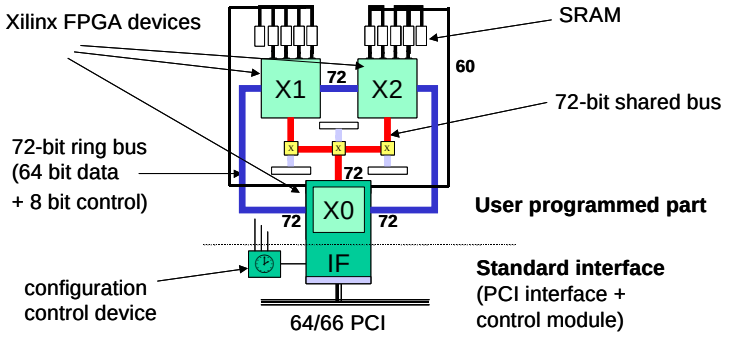

Fig. 2. SLAAC-1V Architecture

controller located in the interface part of X0 can transfer data to or from these FIFOs as well as to provide fast communication between the host and the board SRAMs. The DMA controller load balances input and output FIFOs and can process large memory buffers without host processor interaction. Current interface development includes managing memory buffer rings on the FPGA to minimize host interrupts on small buffers.

SLAAC-1V supports partial reconfiguration, in which part of an FPGA is reconfigured while the rest of the FPGA remains active and continues to compute. A small dedicated Virtex 100 configuration control device is used to configure all FPGAs and manages 6 MB of flash / SRAM as a configuration “cache”.

The work discussed in this paper was done in collaboration with the ISI Gigabit-Rate IPSec (GRIP) project, which is funded in the DARPA Next Generation Internet (NGI) program. The GRIP team has constructed a gigabit Ethernet daughter card which connects to SLAAC-1V in place of the crossbar connection of the X0 chip. To the host, the SLAAC-1V / GRIP system appears to be a gigabit Ethernet card with optional acceleration features. The GRIP team is currently customizing the TCP/IP stack for the Linux operating system to take advantage of the hardware acceleration in order to deliver fully-secure, fully-authenticated gigabit-rate traffic to the desktop.

3. Implementation of Rijndael

Rijndael is a symmetric key block cipher with a variable key size and a variable input/output block size. Our implementation supports all three key sizes required by the draft version of the AES standard, 128, 192, and 256 bits. Our key scheduling unit is referred to as 3-in-1, which means that it can process all three key sizes. Switching from one key size to the other is instantaneous, and is triggered by the appropriate control signals. Our implementation is limited to the block size of 128-bits, which is the *only* block size required by Advanced Encryption Standard. Implementing other block sizes, specified in the original, non-standardized description of Rijndael is not justified from the economical point of view, as it would substantially increase circuit area and cost without any substantial gain in the cipher security.

Rijndael is a substitution-linear transformation cipher based on S-boxes and operations in the Galois Fields. Below we describe the way of implementing all component operations of Rijndael, and then present how these basic operations are combined together to form the entire encryption/decryption unit.

3.1 Component Operations

Implementation of the encryption round of Rijndael requires realization of four component operations: ByteSub, ShiftRow, MixColumn, and AddRoundKey. Implementation of the decryption round of Rijndael requires four inverse operations InvByteSub, InvShiftRow, InvMixColumn, and AddRoundKey.

ByteSub is composed of sixteen identical 8x8 S-boxes working in parallel. ***InvByteSub*** is composed of the same number of 8x8-bit inverse S-boxes. Each of these S-boxes can be implemented independently using a 256 x 8-bit look-up table.

A Virtex XCV-1000 device contains 32 4-kbit Block Select RAMs. Each of these memory blocks is a synchronous, dual-ported RAM with the data port width configurable to an arbitrary power of two in the range from 1 to 16. Each memory block can be used to realize two table look-ups per clock cycle, one for each data port.

In particular, each 4-kbit Block Select RAM can be configured as a 512 x 8-bit dual-port memory. If encryption or decryption are implemented separately, only the first 256 bytes of each memory block are utilized as a look-up table. If encryption and decryption are implemented together within the same FPGA, both uninverted and inverted 256 byte look-up tables are placed within one memory block. In each case, 16 data bits are processed by one memory block, which means that a total of 8 memory blocks are needed to process the entire 128-bit input.

ShiftRow and ***InvShiftRow*** change the order of bytes within a 16-byte (128-bit) word. Both transformations involve only changing the order of signals, and therefore they can be implemented using routing only, and do not require any logic resources, such as CLBs or dedicated RAM.

The ***MixColumn*** transformation can be expressed as a matrix multiplication in the Galois Field $GF(2^8)$:

$$\begin{bmatrix} B_0 \\ B_1 \\ B_2 \\ B_3 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} A_0 \\ A_1 \\ A_2 \\ A_3 \end{bmatrix} \quad (1)$$

Each symbol in this equation (such as A_i, B_i, '03') represents an 8-bit element of the Galois Field. Each of these elements can be treated as a polynomial of degree seven or less, with coefficients in {0,1} determined by the respective bits of the $GF(2^8)$ element. For example, '03' is equivalent to '0000 0011' in binary, and to

$$C(x) = 0 \cdot x^7 + 0 \cdot x^6 + 0 \cdot x^5 + 0 \cdot x^4 + 0 \cdot x^3 + 0 \cdot x^2 + 1 \cdot x + 1 \cdot 1 = x + 1 \quad (2)$$

in the polynomial basis representation.

The multiplication of elements of $GF(2^8)$ is accomplished by multiplying the corresponding polynomials modulo a fixed irreducible polynomial

$$m(x) = x^8 + x^4 + x^3 + x + 1. \quad (3)$$

For example, multiplying a variable element $A=a_7\ a_6\ a_5\ a_4\ a_3\ a_2\ a_1\ a_0$ by a constant element '03' is equivalent to computing

$$B(x) = b_7 x^7 + b_6 x^6 + b_5 x^5 + b_4 x^4 + b_3 x^3 + b_2 x^2 + b_1 x + b_0 = \quad (4)$$
$$=(a_7 x^7 + a_6 x^6 + a_5 x^5 + a_4 x^4 + a_3 x^3 + a_2 x^2 + a_1 x + a_0) \cdot (x+1)$$
$$\mod (x^8 + x^4 + x^3 + x + 1).$$

After several simple transformations

$$B(x) = (a_7 + a_6)\ x^7 + (a_6 + a_5)\ x^6 + (a_5 + a_4)\ x^5 + (a_4 + a_3 + a_7)\ x^4 + (a_3 + a_2 + a_7)\ x^3 +$$
$$+ (a_2 + a_1)\ x^2 + (a_1 + a_0 + a_7)\ x + (a_0 + a_7), \quad (5)$$

where '+' represents an addition modulo 2, *i.e.* an XOR operation.

Each bit of a product B, can be represented as an XOR function of at most three variable input bits, *e.g.*, $b_7 = (a_7 + a_6)$, $b_4 = (a_4 + a_3 + a_7)$, etc.

Each byte of the result of a matrix multiplication (1) is an XOR of four bytes representing the Galois Field product of a byte A_0, A_1, A_2, or A_3 by a respective constant. As a result, the entire MixColumn transformation can be performed using two layers of XOR gates, with up to 3-input gates in the first layer, and 4-input gates in the second layer. In Virtex FPGAs, each of these XOR operations requires only one lookup table (*i.e.*, a half of a CLB slice).

The ***InvMixColumn*** transformation can be expressed as a following matrix multiplication in $GF(2^8)$.

$$\begin{bmatrix} A_0 \\ A_1 \\ A_2 \\ A_3 \end{bmatrix} = \begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} B_0 \\ B_1 \\ B_2 \\ B_3 \end{bmatrix} \quad (6)$$

The primary differences, compared to MixColumn, are the larger hexadecimal values of the matrix coefficients. Multiplication by these constant elements of the Galois Field leads to the more complex dependence between the bits of a variable input and the bits of a respective product. For example, the multiplication $A='0E' \cdot B$, leads to the following dependence between the bits of A and B:

$$a_7 = b_7 + b_6 + b_5 + b_4 \quad (7)$$
$$a_6 = b_6 + b_5 + b_4 + b_3 + b_7 \quad (8)$$
$$a_5 = b_5 + b_4 + b_3 + b_2 + b_6 \quad (9)$$
$$a_4 = b_4 + b_3 + b_2 + b_1 + b_5 \quad (10)$$
$$a_3 = b_3 + b_2 + b_1 + b_0 + b_6 + b_5 \quad (11)$$
$$a_2 = b_2 + b_1 + b_0 + b_6 \quad (12)$$
$$a_1 = b_1 + b_0 + b_5 \quad (13)$$
$$a_0 = b_0 + b_7 + b_6 + b_5. \quad (14)$$

The entire InvMixColumn transformation can be performed using two layers of XOR gates, with up to 6-input gates in the first layer, and 4-input gates in the second layer. Because of the use of gates with the larger number of inputs, the InvMixColumn transformation has a longer critical path compared to the MixColumn transformation, and the entire decryption is more time consuming than encryption.

AddRoundKey is a bitwise XOR of two 128-bit words and can be implemented using one layer of 128 look-up tables, which translates to 64 CLB slices. Assuming that one operand of the bitwise XOR is fixed, this operation is an inverse of itself, so no special transformation is required for decryption.

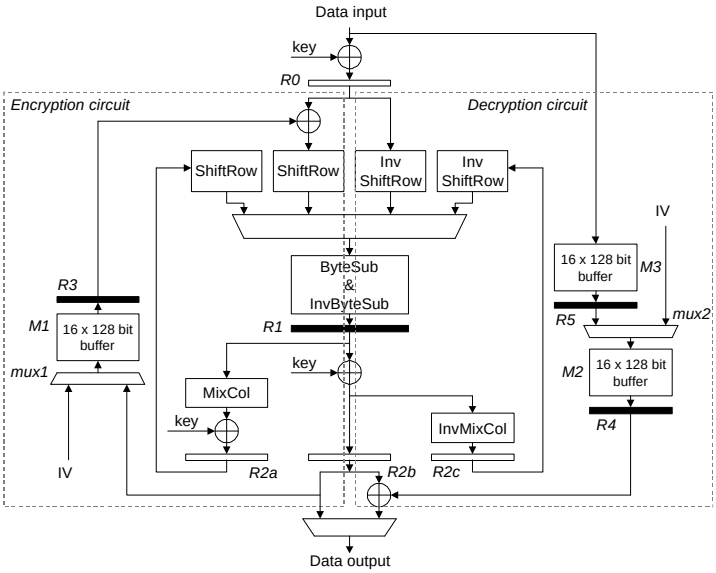

Fig. 3. The general architecture of Rijndael

3.2 General Architecture of the Encryption/Decryption Unit

The block diagrams of the encryption/decryption unit in the basic iterative architecture and in the extended pipelined architecture are shown in Fig. 3. Only registers R1, R3, R4, and R5 (shaded rectangles in Fig. 3) are present in the basic iterative architecture. The remaining registers (transparent rectangles in Fig. 3) have been added in the extended architecture based on the concept of inner-round pipelining.

The register R1 is a part of Block SelectRAM, the synchronous dedicated memory, used to implement ByteSub and InvByteSub transformations, so it was chosen as a basic register in the basic iterative architecture. In this architecture, 11, 13, and 15 clock cycles are required in order to process one block of data for 128-, 192-, and 256-bit keys respectively. The critical path is located in the *decryption circuit*, and includes AddRoundKey (an xor operation), InvMixColumn, InvShiftRow, multiplexer, and InvByteSub (memory read). It is important to note that our *decryption circuit* has a structure (order of operations) similar to the *encryption circuit*, but still does not require any additional processing of round keys (unlike the architecture suggested in [5] and adopted in [8, 9, 10]).

Introducing pipeline registers R2a-c and R0 allows the circuit to process two independent streams of data at the same time. Our architecture assumes the use of the Cipher Block Chaining (CBC) mode for processing long streams of data. The CBC mode is the only mode required by the current specification of IPSec to be used with DES and all optional IPSec encryption algorithms. It is also most likely to be the first mode recommended for use together with AES. The encryption and decryption

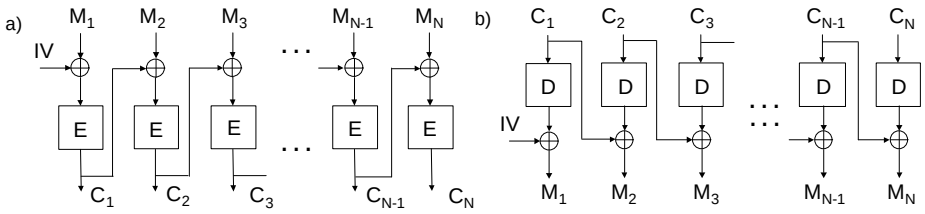

Fig. 4. Cipher Block Chaining Mode a) encryption, b) decryption

in the CBC mode are shown in Fig. 4. An initialization vector IV is different for each packet and is transmitted in clear as a part of the packet header. The CBC mode allows concurrent encryption of blocks belonging to different packets, but not to the same packet. This limitation comes from the fact that the encryption of any block of data cannot begin before the ciphertext of the previous block becomes available (see Fig. 4a). The same limitation does not apply to decryption, where all blocks can be processed in parallel.

In our implementation, the memory buffers M1, M2, and M3 are used to store the last (*i.e.*, the most recently processed) ciphertext blocks for up to 16 independent streams of data. Before the processing of the given stream begins, the corresponding memory location is set to the initialization vector used during the encryption or decryption of the first block of data.

Our architecture allows the simultaneous encryption of two blocks belonging to two different packets, and the simultaneous decryption of two blocks belonging to the same packet or two different packets.

The new secret-key block cipher modes, currently under investigation by NIST, are likely to allow unlimited parallel encryption and decryption of blocks belonging to the same packet [18]. An example of such a mode, likely to be adopted by NIST in the near future, is a counter mode [17]. Our implementation will be extended to permit such new modes as soon as they become adapted as draft standards.

Our architecture can be extended by adding additional outer-round pipeline stages, or implementing multiple instantiations of the same encryption/decryption unit, and using them for parallel processing of data. The total throughput in these extended architectures is directly proportional to the amount of resources (CLB slices, dedicated RAMs) devoted to the cryptographic transformations.

3.3 Round Key Module

The round key module consists of the 3-in-1 key scheduling unit and 16 banks of round keys. The banks of round keys are implemented using 8 Block SelectRAMs configured as two memories 256 x 64 bits. These memories permit storing up to 16 different sets of round keys, with 16 consecutive memory locations reserved for each set. Each set of subkeys may correspond to a different main key and a different security association.

The 3-in-1 key scheduling unit of Rijndael is shown in Fig. 5a. The operation of the circuit is described by formulas given in Fig. 5b. The unit is capable of computing two 32-bit words of the key material (w_i and w_{i+1}) per one clock cycle, independently

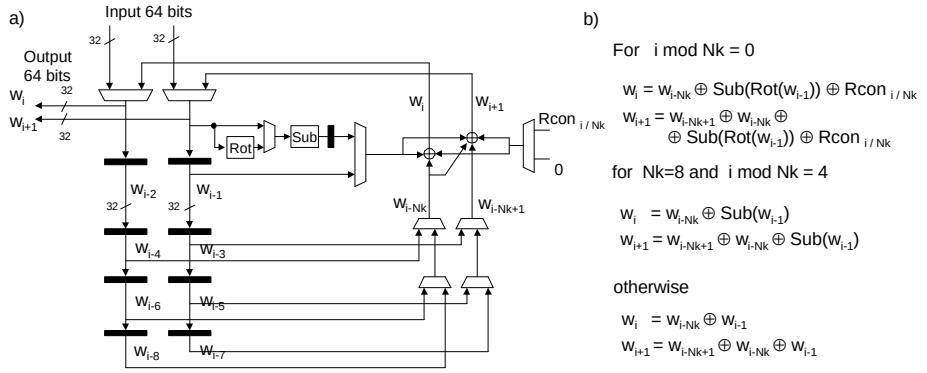

Fig. 5. The 3-in-1 key scheduling unit of Rijndael: a) the main circuit, b) formulas describing the operation of the circuit

of the size of the main key. Since each round key is 128 bit long (the size of the input block), two clock cycles are required to calculate each round key. Therefore, our key scheduling unit is not designed for computing subkeys on the fly. Instead, all round keys corresponding to the new main key are computed in advance and stored in one of the memory banks. This computation can be performed in parallel with encrypting data using previous main key, therefore key scheduling does not impose any performance penalty.

4 Implementation of Triple DES

4.1 Basic Architecture

In order to realize the Triple DES encryption and decryption it is sufficient to implement only one round of DES, as shown in Fig. 6a. The multiplexers *mux1* and *mux2* permit loading new data block or feed back the result of the previous iteration. Only half of the data block is transformed in each iteration, and this transformation depends on a round key coming from the key module. The DES-specific transformation function *F* has been implemented as a combinational logic and directly follows the algorithm specification. The multiplexers *mux3* and *mux4* choose the right feedback for consecutive iterations. In the single DES implementation, these multiplexers would not be required, because the feedback is always the same. However, this is not the case for Triple DES because of the data swapping at the end of the last round of DES. This feature becomes important when switching between the first and the second, and between the second and the third DES encryption in Triple DES. Performing the Triple DES encryption or decryption of one data block in the CBC mode requires 48 clock cycles, exactly as many as the number of the cipher rounds.

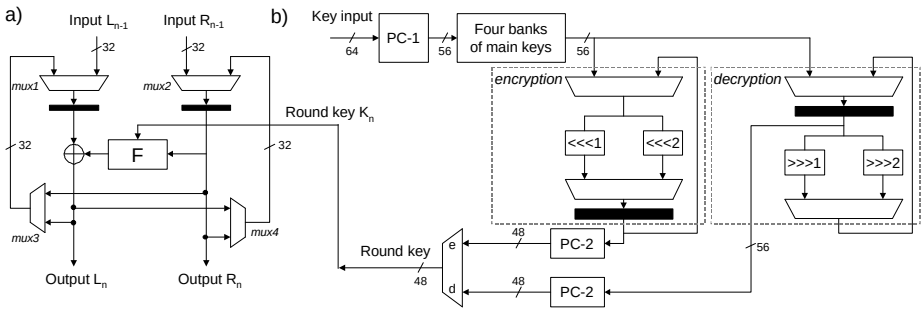

Fig. 6. Basic iterative architecture of Triple DES a) encryption/decryption unit, b) key scheduling unit

4.2 Round Key Module

The DES key schedule, which serves as a basis for the Triple DES key schedule, consists of very simple operations. Consecutive round keys are computed by rotating two halves of the main 56-bit key by one or two positions depending on the number of the round. The result of each next rotation goes through the Permuted Choice-2 function (PC-2), which selects 48 bits of a round key. Since DES key scheduling requires much simpler operations than encryption/decryption unit, it can be easily performed on the fly. This way only three 56-bit keys need to be stored on-chip. Our Triple DES key scheduling unit is shown in Fig. 6b.

Four banks of the key memories are placed at the input to the key scheduling circuit. Each bank contains three DES keys used by Triple DES. The user supplies 64-bit keys to the circuit, but only 56-bits of each key are selected by the Permuted Choice-1 function (PC-1) and stored in one of the memory banks. Each memory bank can hold all three keys required for performing Triple DES. All memory banks are built using dual-port memory, and can operate independently. They are organized in a way that permits writing new key to one of the banks, while any other bank may be used for the round key computations. The output of the round key memory goes to two simple circuits, one computes keys for encryption, the other for decryption.

4.3 Extended Architecture

We are currently in the process of developing an extended pipelined architecture of Triple DES. Our goal is to obtain throughput over 1 Gbit/s. Our approach is to fully unroll single DES and introduce pipeline registers between cipher rounds, as shown in Fig. 7. This leads to a capability of processing up to 16 independent data streams, which gives a throughput of around 1.5 Gbit/s. We should be able to maintain clock frequency at the similar or even greater level, since this architecture permits significant simplifications compared to the basic iterative architecture. Namely, multiplexers *mux3* and *mux4* are no longer required in any of the stages (see Fig. 6b), and key scheduling can be greatly simplified as shown in Fig. 7b.

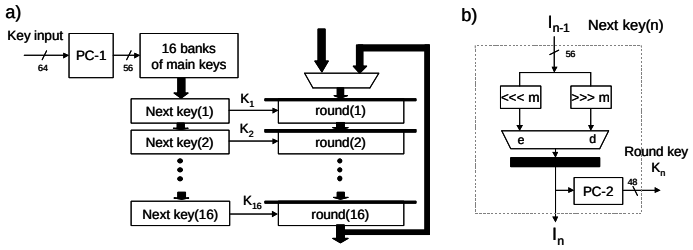

Fig. 7. Extended architecture of Triple DES: a) main circuit, b) next key module; the number of rotations m depends on the round number n, and can be equal to 0, 1, or 2

5. Testing Procedure

Our testing procedure is composed of three groups of tests. The first group is aimed at verifying the circuit functionality at a single clock frequency. The goal of the second group is to determine the maximum clock frequency at which the circuit operates correctly. Finally, the purpose of the third group is to determine the limit on the maximum encryption and decryption throughput, taking into account the limitations of the PCI interface.

Our first group of tests is based on the NIST Special Publication 800-20, which defines testing procedures for Triple DES implementations in ECB, CBC, CFB and OFB modes of operation [20]. This publication recommends two classes of tests for verification of the circuit functionality: Known Answer Tests (KATs), and the Monte-Carlo tests. Since, the Known Answer Tests are algorithm specific, we implemented them only for Triple DES. The Monte Carlo test is algorithm independent, so we implemented it for both Triple DES and Rijndael. The operation of this test is shown in Fig. 8. The test consists of 4,000,000 encryptions with keys changed every 10,000 encryptions. The ciphertext block obtained after each sequence of 10,000 encryptions is compared with the corresponding block obtained using software implementation. Software implementations of Triple DES and Rijndael from publicly available Crypto++ 4.1 library were used in our experiments.

The second group of tests was developed based on the principle similar to the Monte-Carlo tests. One megabyte of data is sent to the board for encryption (or decryption), the result is transferred back to the host, and downloaded again to the board as a subsequent part of input. The procedure is repeated 1024 times, which corresponds to encrypting/decrypting a 1 GB stream of data using CBC mode. Only

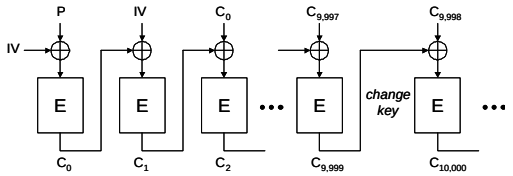

Fig. 8. Monte Carlo Test recommended by NIST in the CBC mode

the last megabyte of output is used for verification, as it depends on all previous input and output blocks. The transfer of data is performed by the DMA unit, so it takes place simultaneously with encryption/decryption. If the test passes, it is repeated at the increased clock frequency. The highest clock frequency at which no single processing error has been detected is considered the maximum clock frequency. In our experiments, this test was repeated 10 times with consistent results in all iterations.

The third group of tests is an extension of the second group. After determining the maximum clock frequency, we measure the amount of time necessary to process 4 GB of data, taking into account the limitations imposed by the 32 bit/33 MHz PCI interface. Since data is transmitted through the PCI interface in both directions (input and output), the maximum encryption/decryption throughput that can be possibly measured using this test is equal to 528 Mbit/s. This is a half of the maximum throughput in the regular operation of the FPGA accelerator, where only input data are transferred from the host to the accelerator card through the PCI interface, and the output is transferred from the FPGA card to the Ethernet daughter card.

6. Results

The results of static timing analysis and experimental testing for Rijndael and Triple DES are shown in Fig. 9.

For Triple DES in the basic iterative architecture, the maximum clock frequency is equal to 72 MHz according to the static analyzer, and 91 MHz according to the experimental testing using the SLAAC-1V board.

For Rijndael in the basic iterative architecture, the results for encryption and decryption are different, with decryption slower than encryption by about 13% in experimental testing. According to the timing analyzer, the maximum clock frequency for the entire circuit is equal to 47 MHz, with the critical path determined by the decryption circuit. In experimental testing, decryption works correctly up to 52 MHz, and encryption up to 60 MHz. However, we do not intend to change the clock frequency on the fly, therefore 52 MHz sets the limit for the entire circuit. The differences between the static timing analysis and experimental testing are caused by conservative assumptions used by the Xilinx static timing analyzer, including the worst case parameters for voltage and temperature prorating.

In Fig. 9b, the maximum throughputs corresponding to the analyzed and experimentally tested clock frequencies are estimated based on the equation:

$$Maximum\_Throughput = (Block\_size / \#Rounds) \cdot Maximum\_Clock\_Frequency. \quad (15)$$

Using formula (15), the maximum throughput of Rijndael in the basic iterative architecture for a 128-bit key is 521 Mbit/s based on the static timing analysis, and 577 Mbit/s based on the experimentally measured clock frequency. This result is expected to be further improved by optimizations of placement and routing. Taking into account our result, parallel processing of only two streams of data should be sufficient to obtain the speed over 1 Gbit/s. As a result, one stage of additional registers, R2a-c, was added to the basic iterative architecture in the extended

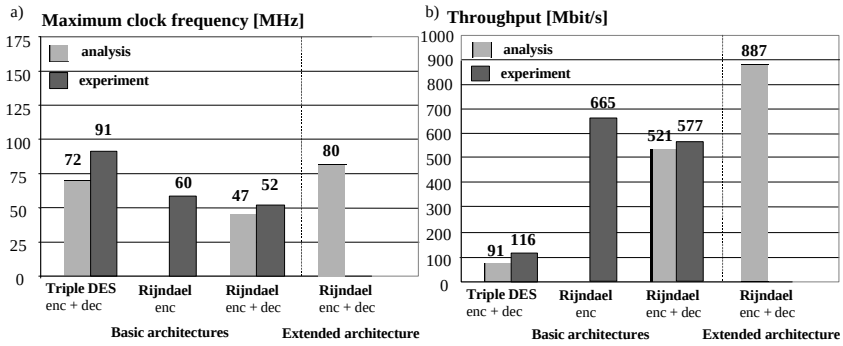

Fig. 9. Results of the static timing analysis and experimental testing for Rijndael and Triple DES a) maximum clock frequency, b) corresponding throughput

architecture as shown in Fig. 3. At this moment, we have been able to obtain a throughput of 887 Mbit/s for this extended pipelined architecture. Nevertheless, further logic and routing optimizations are expected to improve this throughput over 1 Gbit/s without the need of introducing any additional pipeline stages.

The worst-case throughput of Triple DES in the basic iterative architecture is 91 Mbit/s based on the static timing analysis, and 116 Mbit/s based on the experimentally measured maximum clock frequency, which translates to the 27% speed-up in experiment. Sixteen independent streams of data processed simultaneously should easily exceed 1 Gbit/s, leading to the extended architecture shown in Fig. 7.

The actual encryption and decryption throughputs, taking into account the limitations imposed by the PCI interface were measured using the third group of tests described in Section 5. The actual throughputs for DES, were equal to 102 Mbit/s for encryption, and 108 Mbit/s for decryption. The experimentally measured throughput for Rijndael was equal to 404 Mbit/s, and was the same independently of the key size, which means that this throughput was limited by the PCI interface. It should be noted that during the regular operation of the card, when no output is transferred back to the host memory, this throughput can be easily doubled and reach at least 808 Mbit/s.

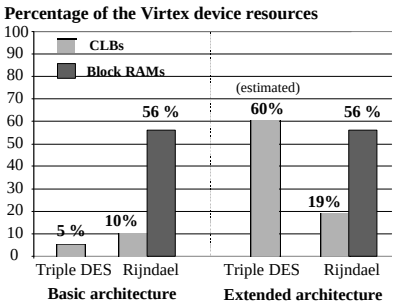

Fig. 10. Percentage of the Virtex device resources devoted to the encryption circuits

The total percentage of the FPGA resources used for the *basic* iterative architectures of Rijndael and Triple DES is 15% of CLB slices and 56% of BlockRAMs. The *extended* architectures of both ciphers, capable of operating over 1 Gbit/s, will take approximately 80% of CLB slices, and 56% of Block SelectRAMs. Only one Virtex XCV-1000 FPGA is necessary to assure the throughput of both ciphers in excess of 1 Gbit/s. Using two additional Virtex devices, and more complex architectures, the encryption throughput in excess of 3 Gbit/s can be accomplished. Our 64-bit/66 MHz PCI module will support this bandwidth.

7. Related Work

Several research groups developed VHDL implementations of Rijndael in Xilinx FPGAs [3, 6, 7, 12, 14], and Altera FPDs [8, 9, 10, 19]. A survey and relative comparison of results from various groups is given in [13]. All major results described in the aforementioned papers are based on the static timing analysis and simulation, and have not yet been confirmed experimentally.

The first attempt to validate the simulation speed of Rijndael through experimental testing is described in [9]. The test was performed using especially developed PCI card. Nevertheless, since the operation of the system appeared to be limited by the PCI controller, no numerical results of the experimental tests were reported in the paper.

As a result, our paper is the first one that describes the successful experimental testing of Rijndael and directly compares the experimental results with simulation.

8. Summary and Possible Extensions

The IPSec-compliant encryption/decryption units of the new Advanced Encryption Standard - Rijndael and the older encryption standard Triple DES have been developed and tested experimentally. Both units support the Cipher Block Chaining mode. Our experiment demonstrated up to 27% differences between the results obtained from testing and results of the static timing analysis based on Xilinx tools. These differences confirmed that the results based on the static analyzer should be treated only as the worst-case estimates.

The experimental procedure demonstrated that the total encryption and decryption throughput of Rijndael and Triple DES in excess of 1 Gbit/s can be achieved using a single FPGA device Virtex 1000. Only up to 80% of resources of this single FPGA device are required by all cryptographic modules. The throughput in excess of 3 Gbit/s can be accomplished by using two remaining FPGA devices present on the SLAAC-1V accelerator board. The alternative extensions include the implementation and experimental testing of other security transformations of IPSec, such as HMAC and the Internet Key Exchange protocol.

References

1. Advanced Encryption Standard Development Effort. http://www.nist.gov/aes
2. Blaze M., Diffie W., Rivest R., Schneier B., Shimomura T., Thompson E., and Wiener M.: Minimal Key Lengths for Symmetric Ciphers to Provide Adequate Commercial Security http://www.counterpane.com/keylength.html
3. Chodowiec P., Khuon P., Gaj K.: Fast Implementations of Secret-Key Block Ciphers Using Mixed Inner- and Outer-Round Pipelining. Proc. ACM/SIGDA Ninth International Symposium on Field Programmable Gate Arrays, FPGA'01, Monterey, Feb. 2001, 94-102
4. Davida G. I. and Dancs F. B.: A crypto-engine, Proc. CRYPTO 87, (1987) 257-268
5. Daemen J. and Rijmen V.: Rijndael: Algorithm Specification. http://csrc.nist.gov/encryption/aes/rijndael/
6. Dandalis A., Prasanna V. K., Rolim J. D.: A Comparative Study of Performance of AES Final Candidates Using FPGAs. Proc. Cryptographic Hardware and Embedded Systems Workshop, CHES 2000, Worcester, MA, Aug 17-18, 2000
7. Elbirt A. J., Yip W., Chetwynd B., Paar C.: An FPGA implementation and Performance Evaluation of the AES Block Cipher Candidate Algorithm Finalists. Proc. 3rd Advanced Encryption Standard (AES) Candidate Conference, New York, April 13-14, 2000
8. Fischer V.: Realization of the Round 2 AES Candidates Using Altera FPGA. Submitted for 3rd Advanced Encryption Standard (AES) Candidate Conference, New York, April 13-14, 2000; http://csrc.nist.gov/encryption/aes/round2/conf3/aes3papers.html
9. Fisher V.: Realization of the RIJNDAEL Cipher in Field Programmable Devices. Proc. of DCIS 2000, Montpellier, Nov. 2000, 312-317
10. Fisher V., Drutarovský M.: Two methods of Rijndael implementation in reconfigurable hardware. Proc. of CHES 2001, Paris, 2001
11. Frankel S., Kelly S., Glenn R.: The AES Cipher Algorithm and Its Use with IPSec. Network Working Group Internet Draft, November 2000, (work in progress) available at http://ietf.org/html.charters/ipsec-charter.html
12. Gaj K., Chodowiec P.: Comparison of the Hardware Performance of the AES Candidates Using Reconfigurable Hardware. Proc. 3rd Advanced Encryption Standard (AES) Candidate Conference, New York, April 13-14, 2000
13. Gaj K. and Chodowiec P.: Hardware performance of the AES finalists survey and analysis of results, Technical Report available at http://ece.gmu.edu/crypto/publications.htm
14. Gaj K. and Chodowiec P.: Fast Implementation and Fair Comparison of the Final Candidates for Advanced Encryption Standard Using Field Programmable Gate Arrays, Proc. RSA Security Conference - Cryptographer's Track, April 2001
15. IP Security Protocol (ipsec) Charter - Latest RFCs and Internet Drafts for IPSec, http://ietf.org/html.charters/ipsec-charter.html
16. Jones M., Athanas P. *et al.*: Implementing an API for Distributed Adaptive Computing Systems. IEEE Workshop on Field-Programmable Custom Computing Machines, Napa Valley, CA, Apr. 1999, 222-230
17. Lipmaa H., Rogaway P., Wagner D.: CTR-Mode Encryption, Public Workshop on Symmetric Key Block Cipher Modes of Operation. Oct. 2000, Baltimore, MD, http://csrc.nist.gov/encryption/modes/workshop1/
18. Modes of Operation. http://csrc.nist.gov/encryption/modes/
19. Mroczkowski P.: Implementation of the Block Cipher Rijndael Using Altera FPGA. Public Comments on AES Candidate Algorithms - Round 2. http://csrc.nist.gov/encryption/aes/round2/pubcmnts.htm.
20. NIST Special Publication 800-20, Modes of Operation Validation System for the Triple Data Encryption Algorithm, National Institute of Standards and Technology (2000)
21. Secure Hash Standard Home Page. http://csrc.nist.gov/cryptval/shs.html
22. Smith R. E.: Internet Cryptography, Addison-Wesley (1997)