ECE 636/INFT 931

Final project report

Hardware implementation of IPSec cryptographic transformations

Issam Andoni, issama@microsoft.com
Pawel Chodowiec, pchodowi@gmu.edu
Jacek Radzikowski, jradziko@osf1.gmu.edu

George Mason University
Electrical and Computer Engineering
Spring 2001
1. Introduction

Internet Protocol Security (IPSec) protocol suite is used to provide privacy and authentication services at the IP layer. IPSec provides an open industry-standard alternative to proprietary IP encryption technologies. It is designed to provide interoperable, high quality, cryptographically-based security for IPv4 and IPv6. These services are provided at the IP layer below the transport layer, making it transparent to applications and users, meaning there is no need to change network applications on a user's desktop when IPSec is implemented in the firewall or router.

The set of security services offered includes:

- **Authentication**—Strong authentication services prevent the interception of data by using falsely claimed identities.
- **Confidentiality**—Confidentiality services prevent unauthorized access to sensitive data as it passes between communicating parties.
- **Data integrity**—IP authentication headers and variations of hash message authentication code ensure data integrity during communications.
- **Dynamic rekeying**—Dynamic rekeying during ongoing communications helps protect against attacks.

These objectives are met through the use of two traffic security protocols, the Authentication Header (AH) and the Encapsulating Security Payload (ESP), and through the use of cryptographic key management procedures and protocols. The IP Authentication Header (AH) provides connectionless integrity, data origin authentication, and an optional anti-replay service. The Encapsulating Security Payload (ESP) protocol may provide confidentiality (encryption), and limited traffic flow confidentiality. It also may provide connectionless integrity, data origin authentication, and an anti-replay service. Both AH and ESP are vehicles for access control, based on the distribution of cryptographic keys and the management of traffic flows relative to these security protocols.

**IPSec Encryption**

IPSec uses block ciphers for encryption of messages. Currently DES is required and 3DES is recommended in IPSec implementations. Other block ciphers may be supported. It is quite likely that AES will also be required. AES is already taken into account to be included as a future IPSec requirement. Also other AES finalists (MARS, RC6, Serpent and Twofish) may be recommended as optional ciphers.

DES and 3DES may work in CBC mode of operation. No other modes are suggested for those ciphers. In case of AES it is likely that CBC, and some type of counter mode will be required, since NIST is going to consider counter mode for new standard.
IPSec Authentication

The algorithms involved in authentication are the HMAC-MD5 Message Digest Algorithm or HMAC-SHA1, the Secure Hash Algorithm. These algorithms are intended to make it nearly impossible for anyone to alter the authenticated data in transit.

Security Association and Key exchange

Before IP packets can be transmitted from one computer to another, a security association (SA) must be established. An SA is a set of parameters that defines the services and mechanisms, such as keys, necessary to protect communications for a security protocol. An SA must exist between the two communicating parties using IP Security. Internet Security Association and Key Management Protocol (ISAKMP) defines a common framework to support the establishment of security associations. ISAKMP is not linked to one specific algorithm, key generation method, or security protocol. IPSec supports both manual as well as dynamic key assignments. For dynamic key generation IPSec uses Diffie-Hellman algorithm to establish pre-shared key. Key management is done using two phases:

1. Phase 1 is where the two ISAKMP peers establish a secure, authenticated channel with which to communicate. This is called the ISAKMP Security Association (SA). "Main Mode" and "Aggressive Mode".
2. Phase 2 is where Security Associations are negotiated on behalf of services such as IPSec or any other service, which needs key material and/or parameter negotiation.

Today’s computer networks transfer data in rates of gigabits per second. Handling such a big traffic on the IPSec layer requires performing cryptographic transformations at very high speed. This speed is currently available only to expensive super computers. Our project will focus on investigating possibilities to accelerate IPSec transformations using reconfigurable hardware, but our results may be easily extended to other hardware solutions.
2. Project background

This project is based on a yearlong research done by Pawel Chodowiec and Dr. Kris Gaj, which led to comparison of all AES candidates in terms of their implementations in reconfigurable hardware.

Currently Pawel Chodowiec is working on development of gigabit rate Rijndael implementation in FPGA for the purposes of GRIP (Gigabit Rate IP Security) project stimulated by USC – Information Sciences Institute.

Pawel is further interested in implementations of public key cryptography and elliptic curve cryptography in configurable hardware as a part of his Ph.D. thesis.
3. Main goals for the project

Our original goals for this project included:

1. **Hardware implementation of three block ciphers: DES, 3DES and Rijndael (AES).**
   
   - Implement encryption and decryption part for each algorithm. One Virtex 1000 has enough space to fit all of these block ciphers, therefore we assume at this point that no reconfiguration will be required during normal operation.
   
   - Implement key schedule for each cipher. In case of the AES we plan to support 128-, 192- and 256-bit keys.
   
   - Support CFB, CBC, OFB and counter modes for all block ciphers.
   
   - Achieve a throughput of at least 1 Gbps regardless of the chosen algorithm and mode of operation.

2. **Hardware implementation of Diffie-Hellman algorithm for key exchange.**

   Investigate the possibility of implementing the Diffie-Hellman algorithm on single FPGA device. We are aware that this algorithm presents a challenge for hardware implementation, therefore we are planning to propose the architecture only, but will not develop any implementation for purposes of this project.

   As an additional task we will investigate the possibility of implementing a random number generator on the FPGA for purposes of Diffie-Hellman algorithm.

3. **Hardware implementation of hash functions HMAC-MD5 and HMAC-SHA**

   Implementation of hash functions does not seem to be a very difficult task for FPGA. However, we need to investigate more whether it is worth of hardware implementation at all. It seems that these functions are simple enough to be performed in software.

4. **Interface our hardware solutions with existing IPSec implementation.**

   Integrate our hardware accelerator with a FreeS/WAN IPSec implementation available under Linux operating system.

   Our original goals for this project were very broad. Unfortunately, we have greatly overestimated our potential and ability to work as a robust team and we have split all tasks among ourselves. As a result, we worked quite independently and have overlooked the need for better coordination of our tasks. Therefore we managed to achieve our goals only partially.
4. IPSec Hardware accelerators currently available on the market

With Internet growth, there is ever growing market for IPSec acceleration. Companies are rapidly deploying Internet based Virtual Private Networks (VPNs) to streamline communications between remote employees, branch offices, customers, business partners and suppliers. To provide fast cryptographic services that can meet the need of this grouping market many vendors have developed products to support IPSec accelerations. IPSec hardware acceleration increases communication throughput to support high bandwidth connections and frees host CPU resources to perform other operations. To support the effort of hardware acceleration Intel provides encryption co-processor. Intel 82594ED encryption co-processor supports all IPSec standard mandated cryptographic algorithms. It includes support for the following algorithms: SHA1-HMAC, MD5-HMAC, DES, and 3DES. During our project we investigated currently existing hardware IPSec accelerators. In this section we describe some of the most promising implementations and compare them.

Check Point VPN-1 Accelerator Card

The following tests were provided by Check Point and published in their Web site http://www.checkpoint.com/products/vpn1/vpn1accwp.html. We included these test results to provide an overview of the performance gain that can be achieved as a result of implementing cryptography using accelerators. The Check Point VPN-1 Accelerator Card is a PCI compatible card designed to extend Check Point VPN-1 Gateway software. Check-Point VPN-1 acceleration card is based on LUNA VPN from CHRYSALIS-ITS so its specification is provided under that section.

The following charts illustrate the impact of adding hardware acceleration to both Windows NT and Solaris VPN-1 gateways. In each case the accelerator card improved throughput significantly. As mentioned previously, actual performance will vary depending upon platform variations, traffic conditions, and encryption algorithms used.

The tests results for VPN-1 running on a Sun Sparc Ultra 30 both with and without the hardware accelerator card installed are shown on the Fig. 1. The tests were run using UDP traffic with both 576 byte and 1400 byte packets. The accelerator card improved throughput by a factor of five for 576 byte packets; and by a factor of nine for 1,400 byte packets.
Tests run by Check Point Software Technologies show that CPU processing load decreases significantly when the VPN-1 Accelerator Card is added to a VPN-1 Gateway Deployment. Fig. 2 shows results of these tests:

As the Fig. 2 indicates if one processor is 100% loaded, adding a second processor frees CPU resources by only 5%. Adding a VPN-1 Accelerator Card to a single processor frees up 35% for running other applications. The accelerator card is seven times more effective in freeing CPU resources than adding an additional processor.

LUNA VPN from CHRYSSALIS-ITS

Luna VPN is a PCI cryptographic accelerator card. It provides cryptographic processing at wire speeds. It has support for many operating systems including Windows NT, Solaris, HP-UX, and FreeBSD. LUNA VPN features include performance acceleration up to 100 Mb/s. The card supports up to 8000 simultaneous connections. The card supports the following:
- PKCS#11, IKE, and IPSec protocols
- Symmetric algorithms DES and 3DES
- Hashing Algorithms SHA-1 and MD-5
• Message Authentication HMAC and HMAC-SHA-1
• Asymmetric Algorithms RSA(512-2048), Diffie-Hellman (512-2048) and DSA (1024)
• Web site reference is http://www.chrysalis-its.com/products/luna_vpn.html

Cisco Secure PIX Accelerator Card

Cisco Accelerator card is a PCI cryptographic accelerator card providing cryptographic processing at wire speeds. It intended to support only PIX OS v 5.3(1). Cisco card features include performance acceleration up to 100 Mb/s. The card supports up to 2000 simultaneous connections. The card supports the following:

• PKCS#11, IKE, and IPSec protocols.
• Symmetric algorithms 56-bit DES and 168-bit 3DES.
• Hashing Algorithms SHA-1 and MD-5.
• Asymmetric Algorithms RSA, Diffie-Hellman and DSA

NetSwift 1000 VPN Accelerator

NetSwift Accelerator card is a PCI cryptographic accelerator card providing cryptographic processing up to T-3 speeds. It supports Linux OS. The card supports the following:

• Symmetric algorithms DES, 3DES, and RC4.
• Hashing Algorithms SHA-1, HMAC, and MD-5.
• Asymmetric Algorithms RSA(384-4096), Diffie-Hellman and DSA
• Web site reference is http://www.ivea.com/netswift/

Guardian IPSec Accelerator

The Guardian IPSec Accelerator card is a PCI cryptographic accelerator card providing cryptographic processing up to 45 Mbps speeds. It supports Microsoft Windows NT operating system. The card supports the following:

• Symmetric algorithms DES and 3DES.
• IKE.
• X.509 v.3 Digital Certificate
• Web site reference is http://www.netguard.com/subpages/products_gvpn.htm

CTGI PowerCrypt

CTGI PowerCrypt Accelerator card is a PCI cryptographic accelerator card providing cryptographic processing up to 216 Mbits/sec speeds. It supports Windows NT, Linux, FreeBSD, and OpenBSD operating systems. The card supports the following:

• Symmetric algorithms DES, 3DES, and RC4.
• Hashing Algorithms SHA-1 and MD-5.
• http://www.powercrypt.com/
Notes on implementations

The above implementations are just a subset of the manufacturer that implemented or currently are looking to implement cryptographic hardware accelerators. None of them at the moment supports AES, but many initiated researches to add AES to their current modules. The speed that was reported reflects speed that can be obtained using DES-56 encryption and thus should only be seen as the best scenario. Some vendors just included support for symmetric technology while others decided to add support for public key cryptography as well.
5. Target platform for our hardware implementations

Hardware implementations are usually designed and coded either in the hardware description language, such as VHDL and Verilog HDL, or on the schematic. There exist two major implementation approaches for hardware designs: Application Specific Integrated Circuits (ASIC) and Field Programmable Gate Arrays (FPGA).

Application Specific Integrated Circuits are designed all the way from behavioral description to the physical layout. The design is very time consuming and engages a lot of manpower. On the other hand, Field Programmable Gate Arrays offer very unique features. They can be bought off the shelf and reconfigured to perform different functions. Each reconfiguration takes only a fraction of a second. FPGA consists of thousands of small universal building blocks, known as Configurable Logic Blocks (CLB) [30]. CLBs are connected using programmable interconnects. Some of the FPGA families contain dedicated memory blocks. These are called Block SelectRAM in Xilinx devices [30].

Figure 3 Structure of the Virtex FPGA
Starting our project we have had already some experience in designing digital circuits for FPGA implementation using VHDL. In our laboratory we have SLAAC-1V PCI board with Xilinx Virtex 1000 devices, therefore naturally we have chosen this board as our development platform.

SLAAC-1V board is an FPGA-based computation accelerator developed under a DARPA-funded project called Systems-Level Applications of Adaptive Computing (SLAAC). USC Information Sciences Institute (ISI) provided this board to us for the purposes of Gigabit Rate IP Security (GRIP) project.

**Figure 4.** SLAAC1-V board

The board architecture is based on three Xilinx Virtex XCV-1000 FPGA devices. Each of these devices is composed of 12,288 CLB Slices and 32 4-kbit Block SelectRAM modules. The logical architecture of the SLAAC1-V board is shown on the Fig. 5. The three devices, X0/IF, X1 and X2 are interconnected through 72-bit ring paths, and through 72-bit shared bus. All devices are available for reconfiguration. Device X0 is responsible for communication with PCI bus, therefore around 20% of its area is devoted to PCI interface. In our project we were using only device X0.

**Figure 5.** Logical structure of SLAAC-1V board
6. Design approach

All our hardware designs have been encoded in VHDL. We used Active-HDL 4.1 as a design entry tool. This tool greatly supported the development of our codes permitting very accurate behavioral, post-synthesis and timing simulations under control of test benches. We have developed several test benches for testing different components of the entire design. Once the entire code was verified we used Xilinx Foundation Series 3.3i for design synthesis and implementation.

![Design flow diagram]

We have not set any constraints other than target clock frequency of 50 MHz. Xilinx implementation tools produced configuration bitstream for FPGA and timing model of implemented circuit. The timing model was fed back to Active-HDL for timing simulations, which also confirmed functionality of the circuit. Finally, we downloaded the configuration bitstream to the SLAAC-1V board for final experimental testing and design validation.
7. Architectures for IPSec cryptographic transformations

7.1 Diffie-Hellman Key Exchange

Diffie-Hellman key exchange Technique is a public key algorithm that allows two communicating entities to agree on a shared key over an insecure medium without any prior secrets. The protocol has two public system parameters \( p \) and \( g \). Parameter \( p \) is a prime number and parameter \( g \), the generator, is an integer less than \( p \), which is capable of generating every element from 1 to \( p-1 \) when multiplied by itself a certain number of times, modulo \( p \).

Suppose that Alice and Bob want to agree on a shared secret key using the Diffie-Hellman protocol. They proceed as follows:

1. Alice generates a random private value \( a \).
2. Bob generates a random private value \( b \).
3. Then they derive their public values using parameters \( p \) and \( g \) and their private values. Alice’s public value is \( g^a \mod p \) and Bob’s public value is \( g^b \mod p \).
4. They then exchange their public values.
5. Alice computes \( k_{ab} = (g^b)^a \mod p \), and Bob computes \( k_{ba} = (g^a)^b \mod p \).
6. Note that \( k_{ab} = k_{ba} = k \). So Alice and Bob now have a shared secret key \( k \).

The protocol security lies on the difficulty of solving the discrete logarithm problem. It assumes that it is computationally infeasible to calculate the shared secret key \( k = g^{ab} \mod p \) given the two public values \( g^a \mod p \) and \( g^b \mod p \) when the prime \( p \) is sufficiently large.

Authenticated Diffie-Hellman

In 1992 Diffie, van Oorschot, and Wiener developed a variation of the protocol called authenticated Diffie-Hellman key agreement protocol. The standard Diffie-Hellman key exchange is vulnerable to a middleperson attack. This vulnerability is due to the fact that Diffie-Hellman key exchange does not authenticate the participants. The authenticated Diffie-Hellman key exchange came in respond to that. The immunity is achieved by allowing the two parties to authenticate themselves to each other by the use of digital signatures and public-key certificates.

Before the protocol, the two parties Alice and Bob each possess a public/private key pair and a certificate for the public key. During the protocol, Alice computes a signature on certain messages and sends Bob a public value \( g^a \mod p \) together with her signature and her public-key certificate. Bob also proceeds in a similar way. Even though Charlie is still able to intercept messages between Alice and Bob, he cannot forge signatures without Alice’s private key and Bob’s private key. Hence, the enhanced protocol defeats the middleperson attack.
Challenges

IPSec protocol makes use of Diffie-Hellman Technique to agree upon a shared key. In both software and hardware implementation the computation will involve computing the value $g^a \mod p$. This computation is very expensive for large $p$ both in software as well as in hardware. The exponent is usually calculated by progressive squaring method and takes $2^N$ modular multiplication to complete. Thus the core arithmetic operation in Diffie-Hellman is modular multiplication. Several methods had been proposed to overcome these limitations. One of the most efficient algorithms that can be implemented in hardware is the one proposed by Peter Montgomery under the name Modular Multiplication without Trial Division. Unfortunately, we have been unable to investigate this algorithm closer and make use of it in this project.
7.2 Implementation of DES and Triple DES

Currently the only symmetric cipher required for IPSec implementations is DES working in CBC mode. 3DES-EDE in CBC mode is only recommended for stronger data protection. Since 3DES uses DES as a building block, we wanted to implement both ciphers in one circuit. We have agreed to the following design criteria for the circuit:

- DES should work in CBC, CFB, OFB and Counter modes of operation.
- 3DES should work in the same modes of operation as DES.
- 3DES should support 112- and 168-bit key sizes.
- The overall throughput of 1Gbps should be achievable for DES and 3DES regardless of chosen mode of operation. This throughput would be possible only when multiple distinct streams of data could be processed in parallel.

Our goals appeared to be too ambitious for one-semester project, and we had to put stricter limitations on our design. We managed to implement only two modes of operation: CBC and Counter. In order to achieve the speed of 1Gbps we would need to apply pipelining and possibly use multiple copies of the circuit. We have not got that far either.

Starting our DES/3DES design we first looked what is already available to us. We knew about a 3DES code developed by two former ECE students Po Khuon and Tanvir Joy. They implemented encryption/decryption core without key schedule. Using their code seemed to be a good idea, since we could implement our own key schedule and put both circuits together. Unfortunately, that code happened to be incorrect. Additionally, it was written in a very hard to debug way. Namely, ordering of bits was incompatible with cipher specification [], tables of S-boxes were very hard to follow. Therefore, we decided to develop our own code from scratch.

Our circuit consists of three parts:
- cipher body realized in basic architecture[]
- key schedule capable of computing keys on the fly
- wrapper logic permitting CBC and Counter modes of operation

Cipher body

Implementation of 3DES encryption and decryption requires implementing only one round of DES, which is shown on the Fig. 7.
Figure 7. One round of DES used to build Triple DES

Multiplexors $mux_1$ permit loading new data block or feed back the result of previous iteration. Only half of the data block is transformed in each iteration, and this transformation depends on round key coming from the key schedule. The DES-specific transformation function $F$ has been implemented as a combinational logic and directly follows the algorithm specification [9]. We did not make any attempts to optimize any of the components of function $F$. However, it could be informative to investigate optimal decompositions for S-boxes in the future, since they do not fit entirely in Virtex lookup tables.

Multiplexors $mux_2$ choose right feedback for consecutive iterations. In DES implementation these multiplexors would not be required, because feedback is always the same. However, data swapping is skipped at the end of last round of DES. This becomes important to handle correctly in 3DES when switching from DES-E1 to DES-D2, and from DES-D2 to DES-E3 – Fig. 8.
Performing 3DES encryption or decryption of one data block in CBC mode requires 48 clock cycles, exactly as many as the number of rounds. DES, however, needs only 16 clock cycles.

**Key schedule**

DES key schedule, which serves as a basis for 3DES key schedule, consists of very simple operations. Consecutive round keys are computed by rotating 28-bit halves of the main 56-bit key. The result of rotations goes to Permutation Choice-2 function, which selects 48 bits composing round key. Since computing keys requires much simpler circuit than cipher round, it can be easily performed on the fly. This way only three 56-bit keys need to be stored on-chip. The circuit we have come up with is shown on the Fig. 9.
Two banks of key memories are placed at the input to the key schedule circuit. The user supplies 64-bit keys to the circuit, but only 56-bits of each key are selected by Permutation Choice-1 function and stored in one of the memory banks. Each memory bank can hold all three keys required for performing 3DES. Both memory banks are built using single dual-port memory, and can be operated independently. They are organized in a way permitting writing new key to one of the banks, while the other is used for round key computations.

The output of key memory goes to two simple circuits, one computes keys for encryption, the other for decryption.

**Wrapper logic realizing modes of operation**

We investigated possibility of implementing four modes of operation: CBC, CFB, OFB and Counter. Our main goal was to come up with the circuit, which will not contribute much to the overall delay. Therefore it is very essential to maintain feedback as simple as possible. During the final presentation of our project we showed a circuit which was capable of operating in CBC and counter mode. In this report we present a revised version of that circuit - Fig. 10, which is both simpler, in terms of required logic complexity and controlling, and can operate in three modes CBC, OFB and Counter.

![Figure 10. Wrapper circuit adding CBC, OFB and Counter modes to underlying block cipher.](image)

AND gates present in the circuit permit disabling particular signals so that they do not affect the result of following XOR operation. Multiplexors are controlled by simple finite state machine, while AND gates are enabled or disabled basing only on the chosen mode of operation.
**Operation in CBC mode – encryption**

In the CBC encryption mode the circuit is configured in a way that it creates logical structure presented on Fig. 11.

![Diagram of CBC encryption](image)

**Figure 11. CBC mode – encryption**

In the initial state, when the circuit waits for the first block of data, \( \textit{mux1} \) selects Initialization Vector. After loading first data block \( \textit{mux1} \) switches to feedback, and remains in this position until new Initialization Vector will be loaded.

**Operation in CBC mode – decryption**

In the CBC decryption mode the circuit is configured in a way that it creates logical structure presented on Fig. 12.

![Diagram of CBC decryption](image)

**Figure 12. CBC mode – decryption**
In the initial state, when the circuit waits for the first block of data, \textit{mux2} selects Initialization Vector. First data block is being loaded simultaneously to \textit{reg1} and to the cipher core, while IV is being loaded to \textit{reg2}. After loading the first data block, \textit{mux2} switches to \textit{reg1}, and remains in this position until new Initialization Vector will be loaded. Both registers \textit{reg1} and \textit{reg2} remain enabled for loading arriving consecutive data blocks.

\textit{Operation in Counter mode}

Counter mode operates identically in both encryption and decryption schemes. The circuit is configured as shown on the Fig. 13.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{counter_mode.png}
\caption{Counter mode – encryption and decryption}
\end{figure}

Each time when new data block arrives it is loaded to register \textit{reg1}, current state of IV counter is being loaded to the cipher core, and counter state is being updated.

\textit{Operation in OFB mode}

OFB mode operates identically in both encryption and decryption schemes. The circuit is configured as shown on the Fig. 14.
Figure 14. OFB mode – encryption and decryption

In the initial state, when the circuit waits for the first block of data, \( \text{mux1} \) selects Initialization Vector. After loading of the first data block \( \text{mux1} \) switches to feedback position and remains in this position until new Initialization Vector will be loaded.

**Interface to SLAAC-1V circuits**

The SLAAC-1V board is connected to the PCI bus of the host computer and requires special circuit for communication. The designers of the board have created a special interface to the PCI bus. This interface creates communication model to the external world. There are four 64-bit FIFO buffers used for data transfers – Fig. 15.

Figure 15. Communication model in SLAAC-1V board
FIFO A1 is a 256-deep input buffer used typically for high throughput data transfers. FIFO B1 is an output buffer similar to A1. FIFOs A0 and B0 are 1-deep buffers and their main purpose is to supply user defined control signals. Both A0 and B0 FIFOs are accompanied with 4-bit TAG.

We decided to use FIFO A1 and B1 to stream data designated for encryption or decryption through the board. FIFO A0 is used to supply keys, IV vectors and control information. The destination of data coming from FIFO A0 is selected using TAG port. Keys are loaded directly to one of the memory banks described in previous section. IV vector is loaded to a special loadable counter. We have also defined a special control register which stores information about encryption process, chosen cipher, mode of operation and selected memory bank. FIFO B0 remains unused in our design.

Testing of the DES/3DES implementation

NIST has issued a Special Publication 800-20 [13], which defines testing procedures for 3DES implementations in ECB, CBC, CFB and OFB modes of operation. Counter mode is not included in any federal standard, therefore NIST did not provide test vectors for it. Following tests are defined by NIST for each mode of operation:

- Variable Plaintext Known Answer Test
- Inverse Permutation Known Answer Test
- Variable Key Known Answer Test
- Permutation Operation Known Answer Test
- Substitution Table Known Answer Test
- Monte Carlo Test

Known Answer Tests are constructed in a way permitting testing the specific components of the algorithm, as well as, exercising the entire algorithm implementation. We have chosen to use only Variable Plaintext KAT and Variable Key KAT. We also looked at the Monte Carlo Test and developed our own testing procedure basing on ideas from this test. We plan to implement more NIST tests in the future.

Variable Plaintext KAT

In the Variable Plaintext KAT all three keys are set to 0x0101010101010101 value. However, the main key is first filtered through PC-1 permutation which removes all parity bits, therefore each 56-bit key consists of 0’s only. IV vector is set to 0x0000000000000000 value, and plaintext vector has only one bit set. The test gives ciphertexts for all 64 possible plaintexts. Since the key vector is cleared, all round keys have the same value 0x0000000000000000 and do not affect transformations of plaintext in the main cipher body. This construction of test helps greatly in tracing mistakes especially made in permutation functions.

When we applied this test to our implementation for the first time we found out that some of the computed ciphertexts are different. Knowledge about how this test works
allowed us to quickly locate the mistake made in one of the S-boxes. After the mistake was corrected all vectors passed the test.

**Variable Key KAT**

In the Variable Key KAT plaintext and IV vector are set to 0x0000000000000000 value. All three 56-bit keys are set to the same value and each one has only one bit set. The test gives ciphertexts for all 56 possible keys. The purpose of this test is to discover incorrect behavior in the key schedule module provided there is no mistake in main cipher transformation.

When we applied this test for the first time the circuit responded always with the same ciphertext. This behavior indicated that keys were not loaded to the memory bank at all. However, all simulations at every level of description showed that the circuit was correct and should work properly. It turned out that we were testing the circuit at 1 MHz clock frequency, which for some reason is too low for dual-port memory. All vectors passed when we have rerun the test with 50 MHz clock frequency.

**Monte Carlo Test**

Monte Carlo Test is a long running test with arbitrarily chosen keys, plaintext and IV vector. It consists of 400,000 encryptions where keys are slightly changed every 1,000 encryptions. It is sufficient to compare only result of the last encryption, however NIST recommends verifying ciphertexts every 1,000 encryptions.

![Figure 16. Monte Carlo Test in CBC mode recommended by NIST](image)

It is clear that performing one encryption requires knowing the ciphertext block computed two iterations before. This means that we would have to constantly read and write back single data blocks to the board through PCI interface. This action would definitely greatly impair the speed of test. We were concerned that Monte Carlo Test might run too long and we decided to apply our own test, which bases on similar idea.
Our long running testing procedure

Instead of encrypting a single block of data and feeding the result back we have chosen to encrypt 256 blocks (the size of FIFO buffers) and then feed them back – Fig. 17.

![Diagram of the encryption process](image)

**Figure 17.** Long running test applied to our implementation

We were repeating this procedure 1023 times. We used 3DES implementation from publicly available Crypto++ 4.1 library to verify the results. If the results of all encryptions were correct, all keys, IV vector and entire plaintext buffer were reinitialized randomly, and entire test was repeated at increased clock frequency. If the test failed we assumed that circuit failed at too high clock frequency.

Parameters of DES/3DES implementation

**Area**

Our circuit turned out to be very small. Implemented alone took 596 CLB Slices which is only 5% of the available area. When implemented together with the PCI core the area requirements increased to 2948 CLB Slices (23% of the available area).

**Throughput**

We have obtained the maximum throughput information from timing analyzer and from experiments with SLAAC-1V board. Timing analyzer performs analysis basing only on the model of FPGA delays. Therefore, the circuit performance reported by timing analyzer was quite conservative.
Performance reported by timing analyzer

- Max. clock frequency: 71.9 MHz
- DES performance: 274.3 Mbps
- 3DES performance: 91.4 Mbps

Performance measured on the SLAAC-1V board

- Max. clock frequency: 91 MHz
- DES performance: 347.1 Mbps
- 3DES performance: 115.7 Mbps

**Figure 18.** Performance of DES and 3DES in CBC mode
7.3 Implementation of AES

AES is not a federal standard yet, but NIST already announced which algorithm is going to play this role [15]. It is very likely that AES will be required in future IPSec implementations, most likely in CBC mode and some version of Counter mode. Our initial design criteria were very similar to those accepted for DES and 3DES design:

- AES should work in CBC, CFB, OFB and Counter modes of operation
- AES requires only 128-bit data blocks
- AES should support 128-, 192- and 256-bit key sizes
- The implementation should achieve throughput of at least 1 Gbps

We planned to use Pawel’s previous work focused on implementing AES candidates in FPGA devices [16, 21, 22, 23]. Pawel has developed several architectures for Rijndael cipher; none of them was equipped with key schedule. As a starting point we have chosen basic architecture with S-boxes implemented on Block SelectRAM modules. Following parameters characterize chosen architecture:

- Area: 986 CLB Slices + 16 Block SelectRAMs
- Max. clock frequency: 50.4 MHz
- Throughput: 615.2 Mbps

As it can be seen above, AES performs very well and exceeding a throughput of 1 Gbps seemed to be a fairly easy task. Introducing one or two pipeline stages should be sufficient. The only circuit that needed to be designed from scratch was key schedule capable of computing all three key sizes required by AES specification. As about modes of operation, we planned to adapt the circuit already developed for DES/3DES. However, Rijndael requires separate circuits for encryption and decryption and implementing special circuit for modes of operation may turn out to be more area and performance efficient than just adapting solution from DES/3DES.

We have not finished implementing entire AES algorithm. Currently only we have designed and tested key schedule circuit, but it was not put together with cipher body yet. Therefore, all performance characteristics presented for AES come from timing analyzer only.

Key schedule

Rijndael’s key schedule is far more complicated than key schedule in DES. In basic algorithm entire key material is computed in chunks of 32-bits. Since every round requires 128-bit keys, the straightforward implementation of key schedule will be four times slower than the cipher itself and cannot support computing keys on the fly. It is possible to compute keys on the fly in fairly simple circuit for 128-bit keys only, and such circuit would not be capable of supporting any other key sizes. Computing subkeys for 192- and 256-bit keys is far more challenging when the data block is 128-bit wide. After close analysis we managed to come up with the circuit which remains fairly simple
and is capable of computing 64-bits of key material per one clock cycle – Fig. 19. This is still too few to support computation of keys on the fly, and all computed subkeys need to be computed in advance and stored in some memory in order to achieve full cipher performance.

**Figure 19. AES key schedule**

**Operation in 128-bit mode**

Transformation $\text{SubByte( RotByte( k_i ) )}$ is performed every second clock cycle.
**Operation in 192-bit mode**

\[
\begin{align*}
 k_i &= k_{i:Nk} + k_{i-1} \\
 k_{i+1} &= k_{i:Nk+1} + k_{i:Nk} + k_{i-1}
\end{align*}
\]

**Figure 21.** Computing keys in 192-bit mode

Transformation SubByte( RotByte( k_i ) ) is performed every three clock cycles.

**Operation in 256-bit mode**

\[
\begin{align*}
 k_i &= k_{i:Nk} + k_{i-1} \\
 k_{i+1} &= k_{i:Nk+1} + k_{i:Nk} + k_{i-1}
\end{align*}
\]

**Figure 22.** Computing keys in 256-bit mode
Applying transformation involving SubByte is performed every second clock cycle, but it differs from previous schemes. SubByte is applied in an interleaved manner. If the clock cycle number is not divisible by 4, only SubByte(\(k_i\)) is computed. If clock cycle number is divisible by 4, the entire transformation SubByte(\(\text{RotByte}(k_i)\)) is used.

The entire key schedule circuit has been simulated and its correctness was verified using test vectors available in draft AES description[]. Since we did not put together the entire cipher, we have not tested anything on the SLAAC-1V board. However, to get some idea how the key schedule circuit performs we have synthesized it and extracted information from timing analyzer. We have obtained following circuit characteristics:

- Area: 259 CLB Slices + 2 Block SelectRAMs
- Max. clock frequency: 51.5 MHz

It turns out that our design is quite compact and performs with similar clock frequency as main cipher circuit, therefore it should not set limits on maximum throughput of the entire cipher.
8. Conclusions

When we look back at the original project goals we now realize how broad spectrum of complex tasks we tried to accomplish. Certainly these project goals could be split into two or three good class projects.

We were able to completely design, implement and test DES/3DES circuit. The circuit is very small – takes only 5% of Virtex XCV1000 resources and has good performance characteristics. 3DES implementation demonstrated encryption throughput in excess of 100 Mbit/s. DES implementation exceeds the throughput of 300 Mbit/s. This result is better than any of the commercial product which we were looking at. The speed of 115 Mbit/s for 3DES is completely sufficient to encrypt entire traffic in 100 Mbit Ethernet networks. Since the circuit is so small, it can be implemented in fairly inexpensive FPGAs. For example Spartan II XC2S150 would be enough. It could be also possible to fit entire DES/3DES implementation in smaller Spartan II XC2S50 if Block SelectRAM modules would be used to implement S-boxes. We do not know the exact cost of these devices, but it is expressed in tens of dollars only.

Although, we have not got to the point where we could demonstrate 1 Gbit DES/3DES implementation, it looks quite easy to achieve this speed range by using multiple units. One Virtex XCV1000 could hold more than 10 of them, which is completely sufficient.

AES design did not even reach the stage of experimental testing of basic architecture, but current results show potentially very good performance. We also demonstrated a key schedule circuit, which is capable of computing 64 bits of key material in one clock cycle regardless of the key size. Therefore, computing keys takes twice as much time as encrypting one block of data. Nevertheless, most of the messages transmitted over the network consist of more than 32 bytes. Taking into account the fact that computing keys can be performed in background while encryption of previous message is being processed; key schedule should be able to supply keys to the circuit without any need to wait for it.

AES in basic architecture performs with a speed of over 0.5 Gbit/s. This fact indicates that the throughput of 1 Gbit/s could be achieved by simply introducing one stage of pipeline. One additional stage will not increase circuit area significantly.

Implementation of Diffie-Hellman algorithm appears to be way more complex task than we originally anticipated. This subject could easily be enough for another project.

We cannot demonstrate any integration of our hardware solution with any existing IPSec implementation. Our attempt to do it with FreeS/WAN failed, because all cryptographic transformations are compiled into Linux kernel. Unfortunately, there is no possibility to use SLAAC-1V API functions in kernel, therefore we would have to implement entire communication with the board by ourselves.
Literature

IP and IPSec documentation


DES and 3DES documentation


AES/Rijndael documentation


Montgomery modular multiplication


[25] Thomas Blum and Christof Paar, *Montgomery Modular Exponentiation on Reconfigurable Hardware*


Other
