AUTHENTICATED ENCRYPTION IN HARDWARE

by

Milind M. Parelkar

A Thesis
Submitted to the
Graduate Faculty
of
George Mason University
in Partial Fulfillment of the
the Requirements for the Degree
of
Master of Science
Electrical and Computer Engineering

Committee:

______________________	Dr. Kris Gaj, Thesis Director
______________________	Dr. William Sutton
______________________	Dr. Peter Pachowicz
______________________	Andre Manitius, Chairman, Department of Electrical and Computer Engineering
______________________	Lloyd J. Griffiths, Dean, School of Information Technology and Engineering
Date: ______________________	Fall 2005 George Mason University Fairfax, VA

Authenticated Encryption in Hardware

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science at George Mason University

By

Milind M. Parelkar
Bachelor of Engineering
University of Mumbai(Bombay), India, 2002

Director: Dr. Kris Gaj, Associate Professor
Department of Electrical and Computer Engineering

Fall 2005
George Mason University
Fairfax, VA

Acknowledgments

I would like to thank Dr. Kris Gaj for helping me throughout the course of this research. Special thanks to Pawel Chodowiec, without whose help, it would have been very difficult to come up with good results, on time.

Table of Contents

List of Tables

List of Figures

Abstract

AUTHENTICATED ENCRYPTION IN HARDWARE
Milind M. Parelkar
George Mason University, 2005
Thesis Director: Dr. Kris Gaj

Traditionally, authenticated encryption was achieved using an independent algorithm for encryption, and a separate one for authentication. Recently, in response to the NIST's solicitation, new modes of operation of block ciphers have been developed. These modes allow for a joint implementation of encryption and authentication. This feature is especially beneficial in case of hardware implementations, as it allows for the substantial decrease in the circuit area and power compared to the traditional schemes.

In this thesis, three new modes of operation, OCB, CCM and EAX, and two versions of the traditional scheme, based on AES for encryption, and HMAC-SHA1 and SHA-512 for authentication, have been compared from the point of view of efficiency of hardware implementations. All schemes have been implemented targeting Xilinx Virtex 4 family of FPGAs and a standard-cell ASIC based on 90 nm and 130 nm technology. All schemes have been compared from the point of view of the circuit area, implementation speed, inclusion in the current generation of standards, and a potential for future improvements.

Two potential applications of the authenticated encryption have been analyzed in detail: FPGA bitstream security, and authenticated-encryption in wireless networks. In both cases, all authenticated encryption schemes seemed to be easily meet the current speed requirements. As a result, circuit area was used as a primary criterion for comparison.

The obtained results indicate that out of all five schemes, the CCM mode of operation has the smallest area requirements, and outperforms the traditional scheme based on AES and HMAC-SHA-1 by a factor of 2 in ASICs, and by a factor of 1.2 in FPGAs. For other applications, which require higher speed and optimization for a high speed to area ratio, the OCB mode of operation seems to offer the best potential for efficient hardware implementations.

Chapter 1: Authenticated Encryption - Introduction and Motivation

1.1 Cryptographic Goals

There are four security objectives which form the framework of cryptographic security services. These objectives are –

1. Confidentiality
2. Data Integrity
3. Authentication
4. Non-repudiation

Confidentiality is a service used to keep the content of information from all but those authorized to have access to it. Confidentiality is ensured if no one but the sender and the receiver is able to read the message. *Data integrity* is related to prevention of unauthorized alteration of data. In order to ensure data integrity, one must be able to detect modification of data. Data modifications include insertion of bits, deletion of bits or substitutions. *Authentication* of data or message source verifies that the data actually originates from the person who is purporting to be the sender. *Non-repudiation* of a message is a service which prevents both communicating parties from denying sending or receiving the message [1].

It is assumed that the authentication of the source of data provides message integrity also as a part of the service. Similarly, non-repudiation provides authentication and extends it with protection against cheating by any of the communicating parties [2].

1.2 What is Authenticated - Encryption?

The focus of this thesis is on authenticated encryption algorithms. In order to explain the details of such algorithms, some cryptographic concepts regarding encryption and authentication are explained below.

Historically, it was confidentiality of data that got most attention and authentication was considered a fringe issue. This was partly because no amount of message integrity or authentication will give you confidentiality. It is only message encryption which can protect data from eavesdroppers. Message encryption might provide some kind of authentication, but it is usually very weak and cannot be relied upon.

Various authentication techniques are explained in detail in Section 1.4. Typically, confidentiality and authentication services have been implemented separately, by using different algorithms. Encryption algorithms are used to ensure confidentiality while Message Authentication Codes (MACs) can be used to provide authentication. When two separate algorithms are used to provide independent security services, it is considered cryptographically secure to use separate keys for each algorithm [3]. Recently, techniques have been invented which can combine encryption and authentication into a single algorithm. Merging these two security services in hardware might possibly provide the following advantages –

1. Area requirement for a single algorithm could be smaller as compared to two separate algorithms. Reduction of real-estate requirement on chip translates directly to reduction of costs.

2. Designs with a smaller implementation area tend to consume less power than larger designs. This is an attractive solution for low-power applications like handheld or battery-powered devices.

3. Another advantage is that the combined algorithm needs only a single key as opposed to separate keys for authentication and encryption when separate algorithms are used. Hence the combined algorithms have a slight advantage with regards to key management and key storage issues over separate algorithms.

4. The newer algorithms have been designed with some target goals like speed, speed-to-area ratio etc. Since all the combined schemes are based on block ciphers, the designers have tried to be as efficient as possible with aspects such as the number of block cipher calls required for getting both confidentiality and authentication from the algorithm. Speed improvement cannot be guaranteed for all combined schemes, but some schemes have a scope for parallelization and can work at higher speeds than older techniques.

Confidentiality-Only Modes of Operations of Block Ciphers

A mode of operation of a block cipher is an algorithm that features the use of a symmetric key block cipher algorithm to provide a cryptographic service such as confidentiality or authentication. Previously standardized modes of operation provide only confidentiality. These are as follows –

1. Electronic Code Book (ECB)
2. Cipher Block Chaining (CBC)
3. Cipher Feedback (CFB)
4. Output Feedback (OFB)
5. Counter Mode (CTR)

The most basic mode of operation of block cipher is known as Electronic Code Book (ECB) mode. ECB mode of operation is shown in Fig. 1.1. The input message is broken down into blocks of length equal to the block size of the cipher. Each block is encrypted or decrypted independent of other blocks. As a result, if there are identical input blocks, the ECB mode would produce identical ciphertext blocks when encrypted with the same key. Thus, ECB reveals repetitions of data and patterns within the message [3].

The insecurity of ECB mode of operation led to the use of chaining or feedback modes of operation. Cipher Block Chaining (CBC) mode of operation is the most common feedback mode used. This is shown in Fig. 1.2. Input to the block cipher is xored with the ciphertext of the previous block. This reduces correlations between plaintext and ciphertext.

Both ECB as well as CBC modes of operation provide only confidentiality. If the encrypted message is changed during transit, then a part of the ciphertext would not decrypt properly. Nonetheless, without human inspection, this improperly decrypted message would be hard to detect using automated programs. Additionally, messages may be of the form of executable or binary files which are not suitable for human

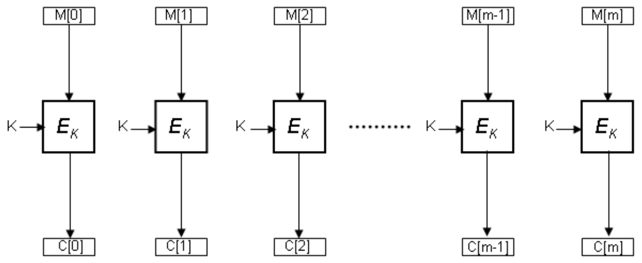


Figure 1.1: Encryption in ECB Mode of Operation – $M[i]$: i^{th} message block; $C[i]$: i^{th} ciphertext block; K: Secret Key; E_K: Encryption function

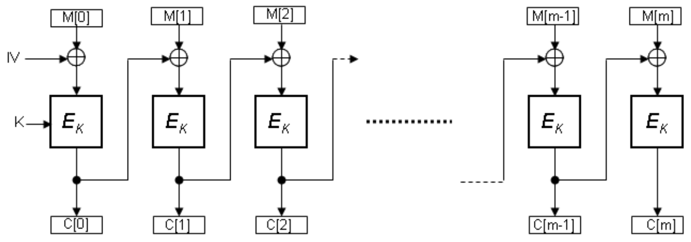


Figure 1.2: Encryption in CBC Mode of Operation – $M[i]$: i^{th} message block; $C[i]$: i^{th} ciphertext block; K: Secret Key; IV: Initialization vector; E_K: Encryption function; (Please note that secret key, K is an input to all E_K blocks)

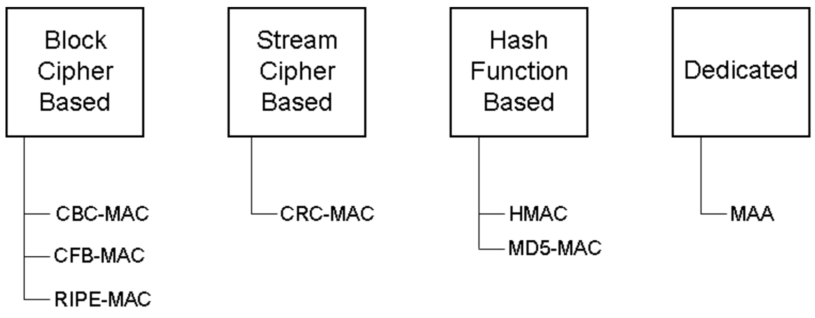


Figure 1.3: Types and examples of MACs

inspection.

Message Authentication Codes(MACs)

Message Authentication Code (MAC) is an algorithm used to provide authentication of messages. This algorithm generates an authentication tag, often referred to as a MAC, which is a function of a secret key and a message. Since, tag verification also requires possession of the secret key, only the intended recipient of the message can check its authenticity [2].

MACs can be divided into four categories as shown in Fig. 1.3 –

1. Block Cipher Based MACs
2. Stream Cipher Based MACs
3. Hash Function Based MACs
4. Dedicated MACs

Detailed explanation of MACs is covered in Section 1.4.2.

Authenticated-Encryption Modes of Operation

Authenticated-Encryption modes of operation of block ciphers provide both encryption as well as authentication using the underlying block cipher. In other words, the modes are designed in such a way that they produce a ciphertext as well as an authentication tag which can be verified by the receiver. Also, both the ciphertext as well as the tag are generated using the same key. Authenticated - Encryption is a relatively recent concept and the impetus to it's development was given by NIST's effort to standardize new Modes of Operations for block ciphers [4].

1.3 Target Applications for Authenticated - Encryption

The purpose of this research is to look at applications which would require a combined authentication and encryption scheme and determine the factors which would influence the selection of a particular mode of operation for that application. Two applications were selected as the basis for analysis. These are –

1. Encryption and Authentication of FPGA Bitstream
2. Authenticated Encryption in IEEE 802.11 wireless LANs

1.3.1 Encryption and Authentication of FPGA Bitstream

Field Programmable Gate Arrays (FPGAs) provide the flexibility to change the configuration after initial programming. In the past, FPGAs were used as glue-logic or as design testing platforms before fabricating the ASIC. Nowadays, FPGAs are used in applications where reconfigurability is a necessity. Advancement of FPGA technology has allowed FPGA manufacturers to allow a capability to reconfigure FPGAs on the fly. This is known as dynamic reconfiguration. Also, there are techniques to reconfigure only specific parts of the FPGA, which is known as partial reconfiguration.

Reconfiguration of FPGAs allows configuration updates in hardware which are similar to software updates or patches. If a portion of the design is not optimal or is no longer required, that part of the chip can be reconfigured with new logic. This reconfigurability brings in the issue of remote reconfiguration. For example, an FPGA array might be used on satellites, which can be remotely reconfigured as per the requirements, without the need for physical access to the FPGA.

Remote reconfiguration gives rise to an issue of FPGA bitstream security. FPGAs are configured by writing a bitstream to the internal configuration memory. In case of remote configuration, the bitstream has to be sent over some communication channel, which is probably insecure. In order to protect the Intellectual Property (IP) represented by the bitstream, confidentiality of the bitstream must be guaranteed. This is done by encrypting the bitstream before it is transmitted [5]. Another required security service is bitstream authentication. This is a requirement since the remote FPGA must discard all bitstreams which did not originate from an authentic source, or those which were damaged in transit [6].

In fact, an attacker might not only want to steal the IP but also might want to take control over the remote FPGA. Similar to installation of hardware patches, the attacker might try to install malicious spyware on the remote FPGA which could be later used to steal sensitive data, secret keys, etc. [7].

Xilinx FPGAs have a 3-DES decryption engine on their Virtex II Pro family

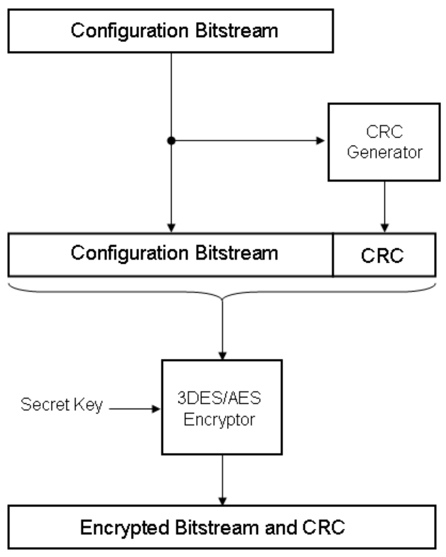


Figure 1.4: FPGA Bitstream Security

of FPGAs [5][8], whereas the latest Virtex-4 family uses an AES decryption engine with a 256-bit key. Currently, there is no secure bitstream authentication mechanism on FPGAs. Xilinx FPGAs use a 32-bit CRC for the bitstream, which can provide "limited" authentication. The main purpose of the CRC is to check for transmission errors. The "limit" of authentication provided by a 32-bit CRC can be looked at in this way - "The attacker would have to try only $2^{32}/2$ bitstreams, on average, in order to get one of the bitstreams to authenticate correctly." On the other hand, an approved authentication algorithm like, an HMAC would typically provide higher order of authentication security. For example, HMAC used in IPSec has 96 bits of security and this can be further increased by increasing the size of the authentication tag. IPSec is a set of protocols developed by the IETF to support secure exchange of packets at the IP layer.

Types of Security Engines on FPGAs

Selected families of current generation of FPGAs include decryption engines which take care of decrypting the incoming bitstream. As suggested earlier integrity checking engines must be added to the already existing engines for security. Currently available decryption engines are implemented on the FPGA fabric and do not consume any programmable FPGA resources. A few suggested implementations of authentication engines are as follows –

1. Implementation on the FPGA fabric – This would seem to be the most apparent technique if a combined encryption - authentication scheme is used. The entire programmable area on the chip remains available to the user. Also, in terms of security this type of authentication engine is the most secure since the integrity of the engine can be guaranteed by the manufacturer. The downside is the increase in cost. In fact, cost has been the main hindrance against adopting proper security measures in FPGAs. The general argument is that only a small percentage of users really care about security of designs and the remaining section of the user base would not really like to share the increased cost for security which they did no want in the first place.

2. Implementation using programmable FPGA resources – This option is suitable for adding security features to older generations of FPGAs. The disadvantage is that the available programmable area for the user reduces. This option is more feasible from the cost perspective than the option of implementing the engine on the FPGA fabric. Only users who need authentication would need to pay for the additional IP core from the manufacturer. Security of such an

implementation is comparable to the security of the engine on the fabric only if the integrity of the authentication engine can be guaranteed.

3. Implementation using Embedded Microprocessors on FPGA – This is an efficient option for implementing authentication functionality on select versions of FPGAs which have an on-chip embedded processor e.g. Xilinx Virtex II Pro. Embedded processors on Xilinx FPGAs include soft-core processors like Xilinx MicroBlaze or hardcore processors like PowerPC. Security is comparable to the security of authentication engine using programmable resources. This is a relatively better option if the design does not use any of the on-chip microprocessors to implement logic functionality.

1.3.2 Authenticated - Encryption in 802.11 Wireless LANs

Communication over an unsecured network would require the use of encryption and authentication. Wireless applications are justifiable candidates for authenticated-encryption modes of operations since in most applications, both confidentiality as well as authentication would be desired. Moreover, use of a combined authenticated encryption technique would be an advantageous option because of its "smaller area" appeal as opposed to a generic composition scheme using separate algorithms for encryption and authentication. Smaller implementation area translates to lower power which might be an appealing factor in battery operated devices.

Communication in wireless LANs has been selected as a case study since the IEEE 802.11i standard for wireless LANs is actually one of the first standards to incorporate a combined authenticated - encryption mode of operation for data security. Another security service sought in the 802.11i standard is protection against replay attacks.

IEEE 802.11i standard [9] for wireless LANs uses a variant of the CCM mode of operation which is called as CCM Protocol (CCMP). Similar security measures would also be required in wired networks.

1.4 Authentication Techniques

Some authentication techniques have been introduced in Section 1.2. These techniques are explained in detail in this section along with some other techniques. As mentioned earlier, the purpose of authentication is to provide a guarantee for the receiver that the message originated from the person who claims to be the author. Legacy authentication techniques include public key digital signatures and secret-key Message Authentication Codes(MACs). Public key cryptography is an expensive option in terms of computational cost as well as circuit area and power. Hence the use of MACs was the most common solution over the years.

1.4.1 Hash Functions

Cryptographically secure hash functions are widely used as components for MACs. A hash function, H is a transformation that takes a variable size input, m and returns a fixed-length string which is called the hash value, h (i.e. $h = H(m)$) [3]. The basic requirements of cryptographic hash functions are –

- It should be able to handle an arbitrarily long input message.
- The output must be of a fixed length.
- $H(x)$ is relatively easy to compute for any given x.
- $H(x)$ is a one-way function.

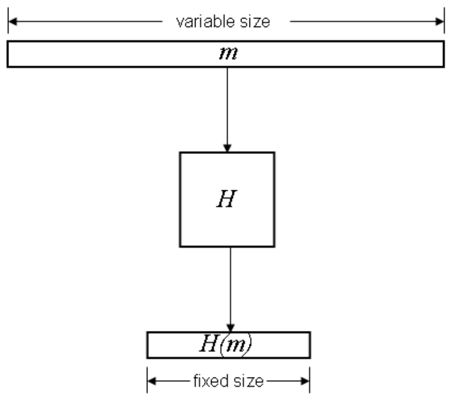


Figure 1.5: Hash Function – m: Message input; H: Hash function; $H(m)$: Message digest

- $H(x)$ is collision-free.

A hash function H is said to be one-way if it is hard to invert. More formally, *hard to invert* implies that given a hash value h, it is computationally infeasible to find any input x such that $H(x) = h$

If, given a message x, it is computationally infeasible to find a message y not equal to x such that $H(x) = H(y)$ then H is said to be a weakly collision-free hash function.

A strongly collision-free hash function H is one for which it is computationally infeasible to find any two messages x and y such that $H(x) = H(y)$.

As can be seen from Fig. 1.5, there is no secret parameter involved in hash computation. Hence, hash function cannot be used alone to provide message authentication.

1.4.2 Message Authentication Codes (MACs)

The concept of Message Authentication Codes was introduced in Section 1.2. Out of the four types of MACs introduced earlier, only two types – Block Cipher based and

Hash Function Based MACs, are of importance to us.

Block Cipher Based MACs

Block cipher based MACs are also known as authentication-only modes of operation of block cipher. An authentication-only mode of operation of block cipher generates an authentication tag as its output. This authentication tag is used by the receiver to check the authenticity of the message. CBC-MAC is a commonly used authentication-only mode of operation. Referring to Fig. 1.6, if all output blocks, except the last one, are discarded, then the last block can serve as an authentication tag. Each output block depends upon the current input block as well as all previous inputs. hence, even if a single bit of the input changes, the contents of the last block will change [2]. FIPS-113 standard for MACs specifies that the length of the MAC can be less than the complete block. Alternately, the FIPS-113 MAC value can be further processed as shown in Fig. 1.6 in order to generate an authentication tag.

Another authentication only mode of operation is One-Keyed MAC (OMAC) which is a variant of CBC-MAC and has proven security properties for an arbitrary sized message [10]. OMAC is a component of one of the authenticated-encryption schemes, EAX, which has been implemented and is covered in greater detail in the chapter on EAX mode of operation. Previously used block cipher based MAC schemes include MDC-2, MDC-4 etc.

Hash Function Based MACs

Hash function based or keyed-hash message authentication code (HMAC), is a type of message authentication code (MAC) computed using a cryptographic hash function

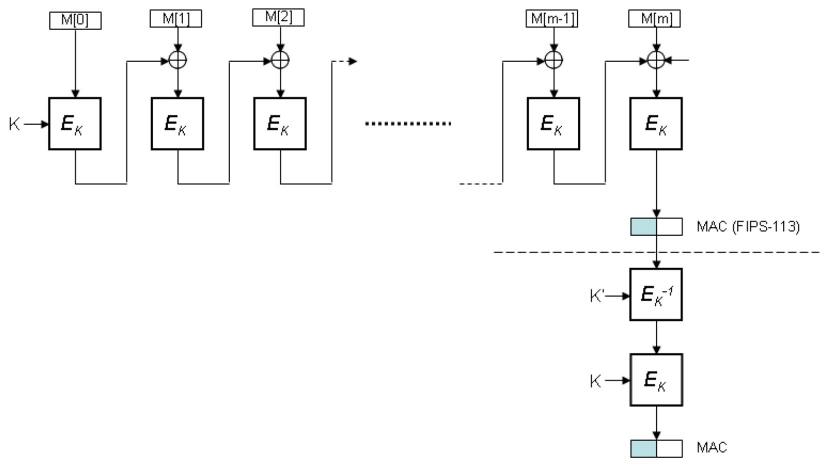


Figure 1.6: MAC Computation in CBC-MAC – $M[i]$: i^{th} message block; K: Secret Key; E_K: Encryption function; E_K^{-1}: Decryption function; (Please note that secret key, K is an input to all E_K blocks)

in combination with a secret key. HMACs were designed in order to use the speed efficiency of hash functions. Software implementations of hash functions are faster than block ciphers. Hence, HMACs which are based on hash functions are more efficient in terms of speed as compared to block cipher based MACs like CBC-MAC.

Any hashing function could be used with HMAC. FIPS PUB 198 standard for HMAC specifies its use only with cryptographically approved hash functions [11]. An example of a secure hash function (which is commonly used in HMAC implementations) is SHA-1. Other common hashing functions include MD5 and RIPEMD-160. Furthermore, there is a new generation of SHA hashing functions (SHA-256, SHA-384, and SHA-512), which provides message digests of longer lengths and thereby more security.

It would be worthwhile to point out the major difference between Fig. 1.5 and

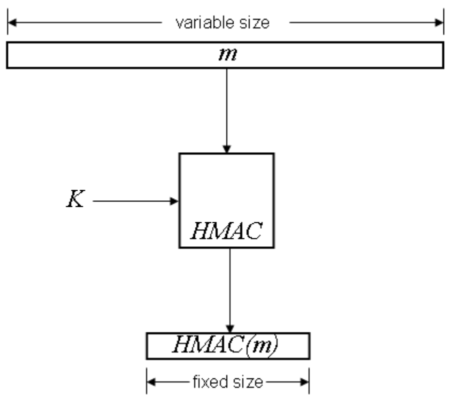


Figure 1.7: Keyed-Hash Message Authentication Code (HMAC) – m: Input message; K: Secret Key; $HMAC(m)$: Message Authentication Code

Fig. 1.7 – the presence of a secret key in the case of HMAC. The secret key is a necessary constituent in order to guarantee authentication.

1.4.3 Authenticated-Encryption using Modes of Operations of Block Ciphers

Authenticated - Encryption modes of operation use a secure block cipher in a particular way to generate both the ciphertext and an authentication tag. Integrity Aware Parallelizable Mode (IAPM) mode of operation developed by Charanjit Jutla from IBM is supposed to be the first publicly described authenticated - encryption scheme that combines confidentiality and authenticity at a small increment to the cost of providing confidentiality alone [12].

Authenticated - Encryption schemes can be further classified into two categories –

1. Single Pass Schemes

2. Two Pass Schemes

Single Pass Schemes

Single pass schemes like OCB, IAPM and XCBC have been put forth as proposals to NIST for standardization. The idea behind all these schemes is that a single pass is needed through the underlying block cipher in order to achieve encryption as well as perform some computations which can be used for integrity checking. The main hurdle with these schemes is patent restrictions. Also, existing single pass schemes do not provide authentication-only support for parts of a message; the entire message needs to be encrypted and authenticated. This is particularly important when some blocks of packet data like packet headers need not be encrypted.

Two Pass Schemes

Two pass Schemes like CCM, EAX etc. have become popular choices for authenticated encryption. They are referred to as *Two Pass Authenticated Encryption with Associated Data* (AEAD) schemes. In these schemes, authenticate-only data is called Associated Data. Associated data is sent in the clear, but its integrity can be checked at the receiver's end since the entire data block is authenticated. CCM is a required scheme in IEEE 802.11i Wireless LANs and IEEE 802.15.4 personal area networks.

1.5 Need and Criteria for Comparison of Hardware Implementations

Most of the implementation data for the modes of operations available in open literature is related to software implementations. The specifications of the modes of operations also have performance estimates in software. The numbers related to software implementations do not actually translate very well to hardware implementations of these modes. Also, whatever little data regarding hardware implementations is available in open literature, is disjointed and it might not be prudent to compare such disparate implementations and draw conclusions from those.

It is my belief that the results for hardware implementations of modes of operations can be compared only if all implementations adhere to some guidelines. Sticking to such guidelines generally ensures that the comparison is not subjective to a particular style of hardware implementation.

Also, the comparison of various authentication options should take into consideration the application. Each application has its own set of constraints. Suitability of a particular scheme to a certain application might not guarantee its appropriateness for some other application.

Let us take a look at the constraints to be considered for target applications mentioned in Section 1.3. There are two main constraints – implementation area and throughput. In case of FPGA bitstream security, it would be attractive to have the smallest possible footprint for the cryptographic circuit. Real estate on an integrated circuit is expensive and would tend to increase the cost of the FPGA. But this cryptographic unit must be able to provide the supported level of configuration throughput. Currently, the maximum configuration throughput using JTAG interface is 30 Mbps and using SelectMAP interface is 150 Mbps.

In case of network applications, selection would be based on the acceptable level of throughput that can be provided by these cryptographic units. Contemporary standards for wireless networks (802.11b/g) specify the maximum throughput with the use of MACs as 54 Mbps. IEEE is targeting a throughput of 100+ Mbps in the next generation of wireless LANs (802.11n). Another factor to be considered is whether parallelization can be used in a particular mode of operation to increase throughput beyond the value provided by non-parallelized implementation.

1.6 Previous Work

The authenticated - encryption schemes are comparatively new and there has been very little previous work regarding analysis and comparison of hardware implementations of these schemes. Even if there has been research going on in this field, there is very little information reported in open literature. Helion Technologies produces commercial IP cores for AES with CCM and OCB modes of operations, but very little implementation data is available for these implementations.

At the same time, there has been a significant research and analysis of stand-alone authentication functions like HMAC. Implementation concepts and analysis of results for HMAC is given in [13]. Hardware implementations of Secure Hash Algorithms (SHA) are covered in [14] and [15].

Chapter 2: Methodology for Comparison of Authentication Techniques

2.1 Scope of this Research

The range of applications for authenticated-encryption schemes is vast. Each of these applications would have a slightly different constraint. It would be practically impossible to perform an exhaustive search of probable applications and tailor the implementations depending upon the needs of that particular implementation. Hence, the implementations have been limited to be optimal for the representative case studies included in this research. The goal is to derive the necessary data for other applications from the currently available data generated during the course of this research.

Three authenticated-encryption modes of operations have been implemented. These include Offset Code Book (OCB), Counter with CBC-MAC (CCM), and EAX. Also, generic composition schemes using HMAC for authentication and AES for encryption have been implemented in order to compare the feasibility of using the authenticated-encryption modes of operations as opposed to older generic composition schemes.

Modes of operations are implemented with 3 underlying block ciphers, AES (Rijndael), Twofish and Serpent. These block ciphers were designed by another student from the same research group, Pawel Chodowiec, and used directly in the implementations of the modes of operation wrapper without any modifications. The block cipher implementations were designed in order to meet a certain clock frequency. Hence,

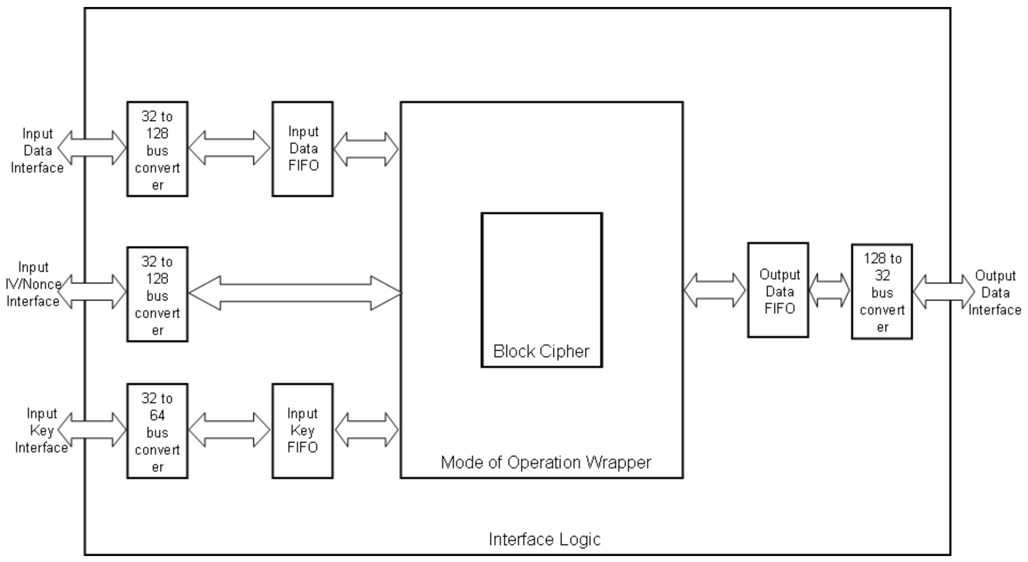


Figure 2.1: Top-Level View of Modes of Operation Wrapper with Input and Output FIFOs and Bus-Width Converters

each cipher has been designed with a different number of pipeline stages.

The following explanation is with reference to the design structure shown in Fig. 2.1. The innermost entity is the block cipher. The block cipher is surrounded by a wrapper which is specific to a particular mode of operation. The wrapper is interfaced to the outside world using 32-bit wide FIFOs. Since Data and IV/Nonce ports of the wrapper are 128 bits wide and the Key port is 64 bits wide, bus width converters have to be introduced on all input and output ports in order to connect to 32-bit wide FIFOs.

The implementations have been restricted with regards to the following criteria –

1. Pre-processing assumed to be done externally – Some modes of operation like

CCM have an input data format consisting of numerous parameters. It is extremely difficult and area inefficient to process plain input data inside the wrapper. Looking at typical requirements in the representative applications, it is safe to assume that such pre-processing can be practically implemented, external to the mode of operation wrapper, in software.

2. Padding of partially full blocks done externally – Each mode of operation has its own padding convention with regards to partially filled data blocks. It is assumed that all necessary padding is done externally.

3. Modes of operation wrappers built around a single block cipher instantiation – In all mode of operation wrappers only a single instantiation of the underlying block cipher is used in order to limit the implementation area. For modes of operation which are parallelizable, parallelization is implemented using the internal block cipher pipeline.

4. Limited use of block cipher pipeline – The block ciphers are designed in such a way that they can process only a single data block at a time. This adversely affects the throughput when there are more pipeline stages.

5. Extra space for non-required functionality – Both encryption as well as decryption functionality has been implemented in the block cipher design. There is no easy way to remove the decryption functionality from the block cipher implementation even for modes of operation which do not require decryption.

2.1.1 Implementation Goals for Modes of Operations

The following implementation goals influenced design decisions –

1. The mode of operation wrappers must be implemented in such a way that it should be easy to replace one block cipher with another by making minimal number of changes to the wrapper design. This requires that all block ciphers be exactly same in terms of top-level pin-out configurations and timing requirements of input and output ports.

2. The mode of operation wrappers must be designed in such a way that it should be easy to replace one mode of operation with the another without making any changes to the wrapper-cipher interfaces. This requires that all modes of operation wrappers be exactly similar in terms of pin-out configuration at the top-level and at the wrapper-cipher interface.

3. It should be easy to add multiple modes of operation to a higher level wrapper and select a particular mode of operation on-the-fly using a control input. This requirement arises from the fact that a sender might want to encrypt and authenticate a packet using one mode of operation, while he might want to use a different mode of operation to process the next packet. This requires designing a top-level wrapper which has the ability to accept individual wrapper plug-ins without any modifications whatsoever to the individual wrapper design.

4. It would also be favorable to have 32-bit interfaces to and from the wrapper instead of having 128-bit interfaces as required by the block ciphers. This necessity arises from the fact that there are only a limited number of I/O pins on the FPGA.

2.2 Tools, Design Process and Synthesis Parameters

All implementations have been coded in VHDL. I have coded all the designs at the RTL level, wherever possible, in order to implicitly guide the synthesis tools to infer the correct netlist from my VHDL code. I have implemented two versions of all designs, one is platform independent and the other is FPGA specific. In the FPGA specific design, I have refrained from using Xilinx primitive components unless it is absolutely unavoidable. This is done in order to maintain inter-platform compatibility with regards to various FPGA families. Xilinx primitives have been used for describing the following components –

1. Distributed RAM using Look-up Tables (LUTs)

2. Block RAMs

3. Variable length Shift Registers using LUTs (SRL16s)

Coding at the RTL level is considered to be optimal for synthesis. Also, limited use of Xilinx specific primitives leaves some scope for the synthesis tools to optimize the design.

Tools used in the Design Process

Aldec Active HDL 6.3 was used for design entry. This tool has a built-in text editor, a VHDL compiler and a simulator. It allowed me to test the designs using both functional and timing simulation under the control of testbenches.

For FPGA synthesis, I used Synplicity Synplify Pro 8.0. It has excellent support for Xilinx primitives and from my experience, produces better results than its counterpart, Xilinx XST. I did use Xilinx XST for synthesis of codes with a large number of Xilinx primitives, but in general the results from Synplify Pro were always better or at least equivalent to Xilinx XST. I set no other constraint except the clock period in Synplify Pro. I tried to get the designs to run at their maximum potential and hence had to make more than a few passes through the tool to get the optimal clock frequency. In some cases, I also tagged some paths in the design as multi-cycle paths and false paths in order to guide the synthesis tool into making favorable decisions regarding placement and routing.

For implementation, I used Xilinx ISE 7.1 which is extremely user friendly in terms of setting design parameters and optimization options. In order to get basic implementation results, I did not set any optimization option other than the desired clock frequency. I used the Static Timing Analyzer tool included in Xilinx ISE in order to get timing information about the designs. Static Timing Analyzer gives detailed information about the worst paths in the circuit including fan-out of each net and routing as well as logic delays for all components and nets along the path.

For ASIC synthesis, I used Synopsys Design Compiler. I performed synthesis for both 90 nm and 130 nm technologies. I used Tcl scripts to perform synthesis. I did not set any other constraints except the desired clock frequency. I used the platform independent version of the codes for performing ASIC synthesis. Design Compiler returned results about the implementation area and timing. Wireload model was used for estimation of routing delays. The ASIC synthesis design flow is shown in Fig. 2.3.

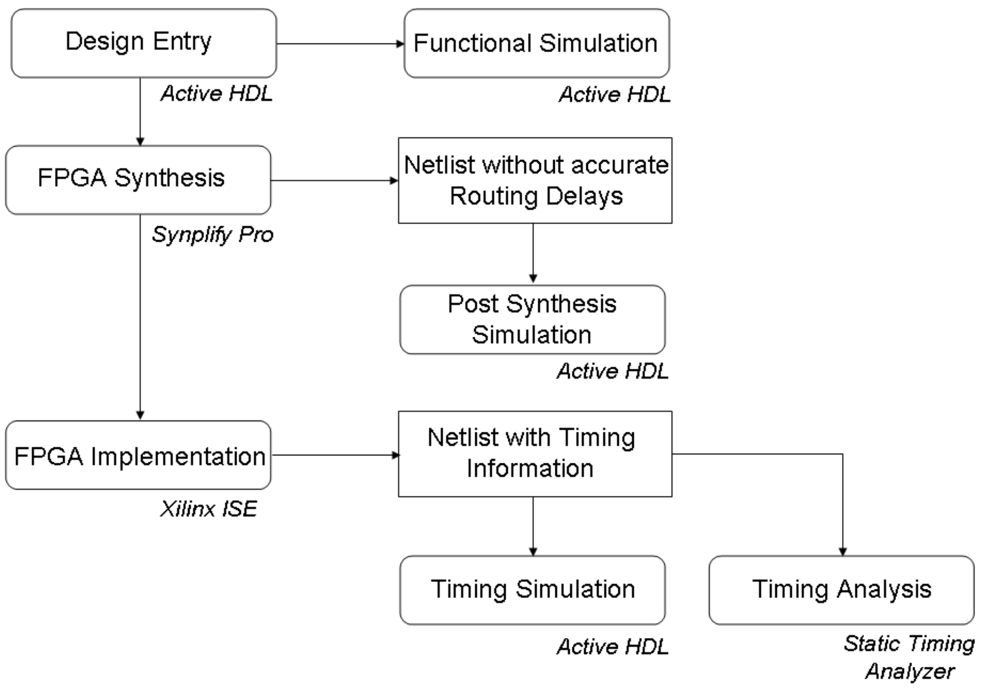


Figure 2.2: FPGA Design Flow

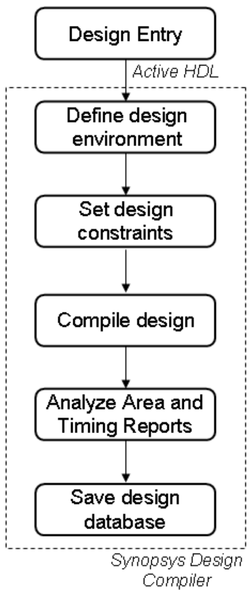


Figure 2.3: ASIC Synthesis Design Flow

Chapter 3: Authentication with Keyed HMACs

3.1 Secure Hash Standard - SHA

Cryptographic authentication techniques are based on two primitives, hash functions and message authentication codes (MACs). FIPS 180-2 publication specifies cryptographically secure hash functions. These form the basis of the Secure Hash Standard. All the SHA functions mentioned in the FIPS 180-2 standard [16] are similar in terms of the concept and construction. Table 3.1 shows comparison of the four secure hash algorithms.

These algorithms differ significantly in the level of security they provide. The level of security is directly related to the size of the message digest. In the context of hash functions, security refers to the fact that a birthday attack on a message digest of size n produces a collision with a work factor of approximately $2^{n/2}$. Collision is said to occur when multiple messages hash to the same value. In general, a hash function which produces a message digest of n bits has a security of $n/2$ bits [3].

Table 3.1: Comparison of Secure Hash Algorithms

	Maximum supported message size (bits)	Size of Message Digest (bits)	Security (bits)
SHA-1	$< 2^{64}$	160	80
SHA-256	$< 2^{64}$	256	128
SHA-384	$< 2^{128}$	384	192
SHA-512	$< 2^{128}$	512	256

Each of the Secure Hash Algorithms is described in 2 stages – Preprocessing and Hash Computation. Preprocessing involves padding a message, parsing the padded message into m-bit blocks, where m is the block size of the hash function and setting initialization vectors to be used during hash computation. The hash computation stage involves creation of a message schedule from the padded message and use of the message schedule along with functions, round constants and word operations to iteratively generate a series of hash values. The final hash value generated by the hash computation stage is the message digest.

3.1.1 Hardware Implementation of SHA-1

Hardware implementation of SHA-1 is the easiest to explain among all the SHA functions. Other SHA functions have a similar structure and can be easily described in a similar way. Fig. 3.1 shows a high-level block diagram for SHA-1. Explanation of various blocks is provided with reference to SHA-1.

Design of Message Scheduler Unit

Message Scheduler Unit in SHA-1 can be optimized very efficiently for FPGA implementations. This follows from the particular structure of the scheduler unit.

The Message Scheduler operates according to the conditions specified below and outputs a single 32-bit word, W_t per clock cycle, where t denotes the round number. Since the message digest in SHA-1 is computed over 80 rounds, the value of t ranges from 0 to 79. M_t denotes the 32-bit message word which is input to the scheduler in round t. $ROTL^1$ rotates its argument left by 1 bit position. W_{t-x} denotes the 32-bit word which was input to the scheduler x rounds before the current round. Details of

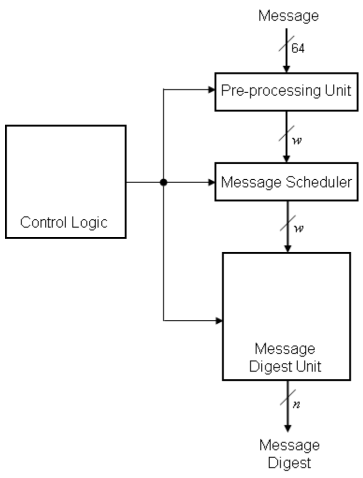


Figure 3.1: Block Diagram of SHA Function – w: Word Size; n: Size of Message Digest

SHA-1 can be found in [16].

$$W_t = \begin{cases} M_t, & for 0 \le t \le 15 \\ ROTL^1(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}), & for 16 \le t \le 79 \end{cases}$$

The diagram for message scheduler unit of SHA-1 is shown in Fig. 3.2. The data path is 32 bits wide, and the unit consists of 16 registers. Hence, the block size which can be handled by SHA-1 is 32*16=512 bits. In other words, SHA-1 accepts 512 bits of data at a time and compresses it down to 160-bits. These 160 bits act as initial values to registers A through E for hashing the next 512 bit block.

The multiplexer selects between the incoming message words and the words computed by the message scheduler unit. For the first 16 rounds, the multiplexer selects the message word as the input to the register bank. In the remaining rounds, internally generated words are fed back to the register bank. Accordingly, proper control

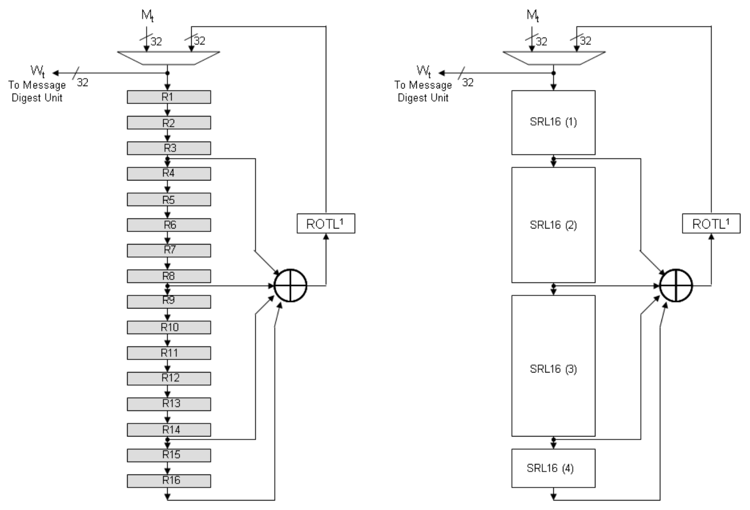


Figure 3.2: SHA-1 Message Scheduling Unit – M_t: Message word; W_t: Output word from Message Scheduling Unit; SRL16: LUT Shift Register primitive to selected Xilinx FPGA families

signal is generated for the multiplexer by the control unit depending upon the round number. The generation of the feedback word is a simple combinational logic with an XOR operation and bit rotation.

An optimization used for FPGA implementation can be seen in Fig. 3.1. Instead of using 16 registers in the message scheduler, a sequence of registers is grouped together and implemented as a shift register. Further performance improvement is possible by using primitive components specific to Xilinx family of FPGAs. Xilinx Virtex and Spartan families have a primitive variable length shift register based on LUTs. This specific shift register is named SRL16 and is a component in the Xilinx UNISIM library. The obvious advantage of using SRL16s instead of registers is in a situation where a small device is being targeted and there are limited number of flip-flops available. Generally, this will not be the case for any reasonable sized FPGA device. The main drawback of using a dedicated flip-flop is that the LUT which feeds the flip-flop might be utilized as a route-though thereby making it unavailable as a logic resource. Hence, the resource utilization on an FPGA for utilizing one FF might actually be the FF and one LUT.

A route-through LUT is used any time an external signal needs to reach a slice resource that can only be reached by using a LUT. The most common case is when it's necessary to reach a FF D-pin when the direct connection BX/BY is already used for something else, such as CIN or F5/F6MUX select [17].

Design of Message Digest Unit

SHA-1 Message Digest Unit is shown in Fig. 3.3. The unit includes five working registers, A, B, C, D, E and five output registers H_0 .. H_4. Each of these registers is 32 bits wide. The final value stored in the output registers is the message digest. Hence, in case of SHA-1, the size of message digest is 5*32=160 bits. Initially, the five working registers are initialized by the pre-processing unit. During every hash computation, the message digest unit takes a 512-bit input and processes it over 80 rounds. During each round, the contents of the working registers are updated. The final contents of the output registers after 80 rounds is the hash value for that 512-bit block. For processing the next 512-bit block, the hash value generated by the previous block is used as the initialization value. This process goes on till all input data blocks have been processed. The final contents of the output registers is the hash value for the entire message.

$ROTL^x$ operation does not require any logic resources, since it can be accomplished by using only routing resources. W_t is the output word generated by the

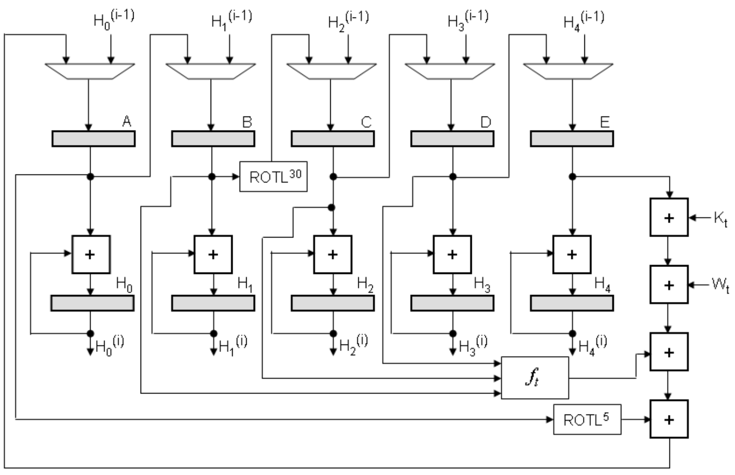


Figure 3.3: SHA-1 Message Digest Unit – K_t: Round constant; W_t: Output word from Message Scheduling Unit; $ROTL^x$: Rotate left by x bit positions; H^b_a: Hash word a, block b

message scheduling unit. K_t is a round dependent constant. This is implemented using a 4:1 multiplexer with inputs hardwired to proper values. A particular input of the multiplexer is selected according to the round number. Function f_t selects the output of one of three combinatorial logic functions, *Ch*, *Maj* and *Parity* depending upon the round number. *Ch*, *Maj* and *Parity* are simple functions of variables B, C and D. Please refer to [16] for details of operations.

As can be seen from Fig. 3.3, the critical path of the circuit passes through a sequence of adders. In order to reduce the critical path in the circuit, design methodology of the adders is critical. In this case, there are 5 operands to be added. Various implementations of adders were studied, and a comparative analysis influenced the decision to use ripple carry adders. An interesting alternative is to use a carry save adder tree in order to reduce the number of operands to be added from 5 to 3.

Three operands can then be reduced to two by using another carry save adder. To add two operands, fast adders based on parallel prefix networks like Brent-Kung or Kogge-Stone could be used. Implementation results show that there is no substantial performance improvement after using this scheme. Also, the use of fast adders to improve timing results in an increase in circuit area.

In case of an FPGA implementation, the use of ripple carry adders provides an added advantage. Xilinx family of FPGAs have dedicated resources for performing fast additions. These include carry chain logic and a dedicated AND gate per CLB slice [17]. The synthesis tool can recognize certain VHDL constructs as adders, and it uses carry chain logic in order to speed up the ripple addition.

Another interesting design consideration was the placement of adders required to find the final hash. This placement dictates a trade-off between number of clock cycles required to compute the output and the period of each clock cycle. If these adders are placed outside the loop, then they are used for addition after 80 rounds have been completed. In this case, the entire iteration of computing the hash takes 81 clock cycles. The number of clock cycles can be reduced to 80 if the adders are placed inside the loop. A control circuit is required which forces one of the inputs of the adders to a zero for all but the last cycle. The length of the critical path increases only slightly in this case. The control circuit uses up a little more area than the first option. Both the options are comparable in terms of performance, and it was arbitrarily chosen to use the adders outside the loop [14].

Design of Control Logic

The control unit for SHA-1 is extremely simple to design and is quite intuitive to understand. All major selections in the algorithm, depend upon the round in which the decision has to be taken. As stated earlier, the hash is computed over 80 rounds. These rounds are separated into 4 blocks, with each block consisting of 20 rounds.

The control logic is based on a round counter, which counts from 0 to 79 and flips back to 0 in order to prepare for the next data block to be hashed. The output of the counter denotes the current round number and this round number is used to generate `enable` signals for the registers and select signals for the multiplexers in modules producing K_t and f_t.

The hash function is designed in such a way that it expects the first word passed to it to be the length of the message in bytes. This is necessary because the control logic has to make a decision about when to stop the computations and determine the correct MAC. This control is implemented using another counter, which keeps track of the index of the data block passed to the hash function.

3.2 Keyed-Hash Message Authentication Code

Message Authentication Codes (MACs) are used to authenticate both the source of the message and its integrity. MACs based on cryptographic hash functions are known as Keyed-Hash Message Authentication Codes (HMACs). HMAC function is used by the sender to compute an authentication tag (MAC) that is a complex function of the secret key and message input. The MAC is typically sent along with the message to the receiver. The receiver can then recompute the MAC value using the message and

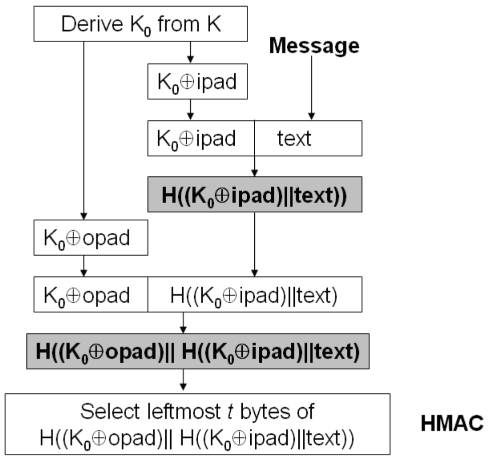

Derive K_0 from K
Message
$K_0 \oplus$ipad
$K_0 \oplus$ipad
text
H(($K_0 \oplus$ipad)||text))
$K_0 \oplus$opad
$K_0 \oplus$opad
H(($K_0 \oplus$ipad)||text)
H(($K_0 \oplus$opad)|| H(($K_0 \oplus$ipad)||text)
Select leftmost *t* bytes of H(($K_0 \oplus$opad)|| H(($K_0 \oplus$ipad)||text))
HMAC

Figure 3.4: HMAC Dataflow – K: Secret key; *ipad*: Inner pad; *opad*: Outer pad; H: Underlying hash function; t: Number of bytes of MAC; $||$ indicates concatenation of strings

the secret key. The MAC value computed at the receiver's side is then compared with the MAC value sent by the sender. If the MAC values match, then the receiver is assured that the message has come from the right sender and has not changed during transmission in the network.

Dataflow diagram for HMAC is shown in Fig. 3.4. Please refer [11] for details of HMAC computations. Two constants, named as inner pad (*ipad*) and outer pad (*opad*) are used in order to ensure that the inner and outer hashes behave as if they were different functions. The constants, *ipad* and *opad* are strings, `0x36` and `0x5C` respectively repeated B times, where B is the block size of the hash function used. The shaded operations in Fig. 3.4 are hash computations. It would be worthwhile to note the following with respect to HMACs –

- The procedure to generate the internal key K_0 from user key K depends upon the length of the user key. Let B be the block size of the hash function, and $l(K)$ be the length of the user key.

1. If $l(K) = B$, then $K_0 = K$.
2. If $l(K) < B$, then $B - l(K)$ zeros are appended to K in order to generate K_0.
3. If $l(K) > B$, then K is hashed to get an L byte string. $B - L$ zeros are appended to the hash to get K_0.

- The first hash operation takes an arbitrary length message as input, but the length of the input to the second hashing operation is constant for a given hash function and length of the key.
- The length of the HMAC is variable and depends upon a user defined parameter t. If $t < B$, then leftmost t bytes of the second hash are used as the MAC value.

3.2.1 Hardware Implementation of HMAC SHA-1

Hardware implementation of HMAC SHA-1 is shown in Fig. 3.5. The HMAC wrapper is designed in such a way that it can interface with any secure hash algorithm designed according to the specifications.

The user key is read 64 bits at a time and stored in a register. Padding signals are generated by xoring the key with parameters, *ipad* and *opad* of proper length. Data passed to the hash function must be scheduled in the correct sequence. The hash function expects the first data word which is passed to it to be the length of the message. The actual message begins from the second word onwards. As can be seen in Fig. 3.4, the input to the first hash operation is the message appended to a

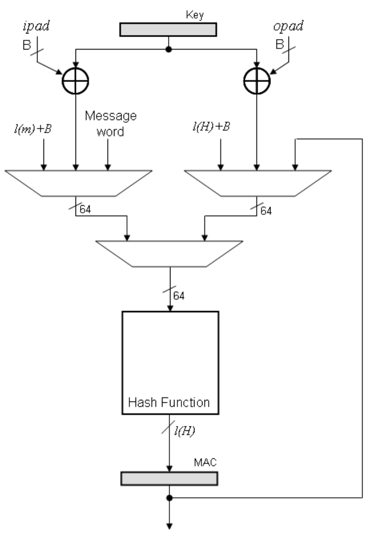


Figure 3.5: HMAC Datapath – *ipad*: Inner pad; *opad*: Outer pad; $l(m)$: Length of the message; B: Block size of hash function; $l(H)$: Non-truncated output size of hash function

string which has length equal to the length of *ipad*. The length of *ipad* is determined by the block size, B of the hash function. Hence the message length to the first hash operation increases by B bytes over the length of the original message. For the second hashing operation, the hash generated by the first hash operation is appended to a string derived from *opad*. Hence, the length of the message to the second hash operation is denoted as $l(H) + B$, where $l(H)$ is the length of the non-truncated output of the hash function. Following the message length block, padding blocks are passed to the hash function. These blocks are followed by the message which has to be hashed. The output of the MAC register can be truncated to include only the leftmost t bytes, where t is a user parameter. The control logic for the HMAC wrapper is a simple state machine based on a counter. The counter keeps track of the index of words being passed to the hash function. Control signals are generated from the state of the counter.

Chapter 4: OCB Mode of Operation

4.1 Introduction

Offset CodeBook (OCB) is a shared-key encryption mechanism which simultaneously provides both encryption and authentication. The OCB mode of operation can be used in conjunction with any 128 bit block cipher. It is classified as a single-pass authenticated-encryption scheme since only a single pass is required through the underlying block cipher in order to provide both the security services.

The keyspace is determined by the keyspace of the underlying block cipher. For a M-bit long message and cipher block size of n-bits, OCB requires only $\lceil M/n \rceil + 2$ block cipher invocations. This is particularly important for short messages since there is minimal overhead in terms of initialization block cipher calls. Another important feature of OCB is that it is parallelizable [18].

4.1.1 OCB Encryption and Decryption

A block diagram representation of OCB mode is shown in Fig. 4.1. OCB mode of operation uses two mutually invertible operations –

1. Encryption and Authentication Tag Generation

2. Decryption and Authentication Tag Verification

OCB encryption and tag generation uses only the forward (encryption) functionality of the underlying block cipher, while decryption and tag verification process uses both forward (encryption) as well as reverse (decryption) functionality of the cipher. This aspect of OCB is not optimal when it is used with block ciphers like AES which have a significantly different encryption and decryption function.

Important features of OCB mode of operation can be summarized as follows –

- **Message Partitioning** – The input message is partitioned into blocks of size 128-bits. These are denoted as $M[1], M[2], \ldots M[m]$ in Fig. 4.1.

- **Initialization** – Two block cipher calls are necessary to initialize the state of the OCB mode. The first block cipher call generates a parameter L by encrypting a string of n zeros, where n is the block size of the cipher. This block is denoted as 0^n in Fig. 4.1. The second block cipher call generates a parameter R which is an encryption of the value $N \oplus L$, where N is a nonce known to both sender and receiver, and typically transmitted in clear.

- **Computing Iteration Constants** – Each block cipher invocation in OCB after the first two invocations uses a constant $Z[i]$ which is xored with the the message input $M[i]$. Similarly, the output of the block cipher is also xored with $Z[i]$. $Z[i]$ is a 128-bit block which is referred to as iteration constant in this document.

- **Handling the last message block correctly** – The last message block is handled differently than other blocks in the OCB mode. This situation arises from the condition that the last block might not be completely full. The length of the block is computed and represented as a 128-bit value. This is denoted as $l(M[m])$ in Fig. 4.1. This length field is then xored with iteration constant $Z[m]$ and $(L \cdot x^{-1})$ which is denoted as L *inverse*. The resulting block is then

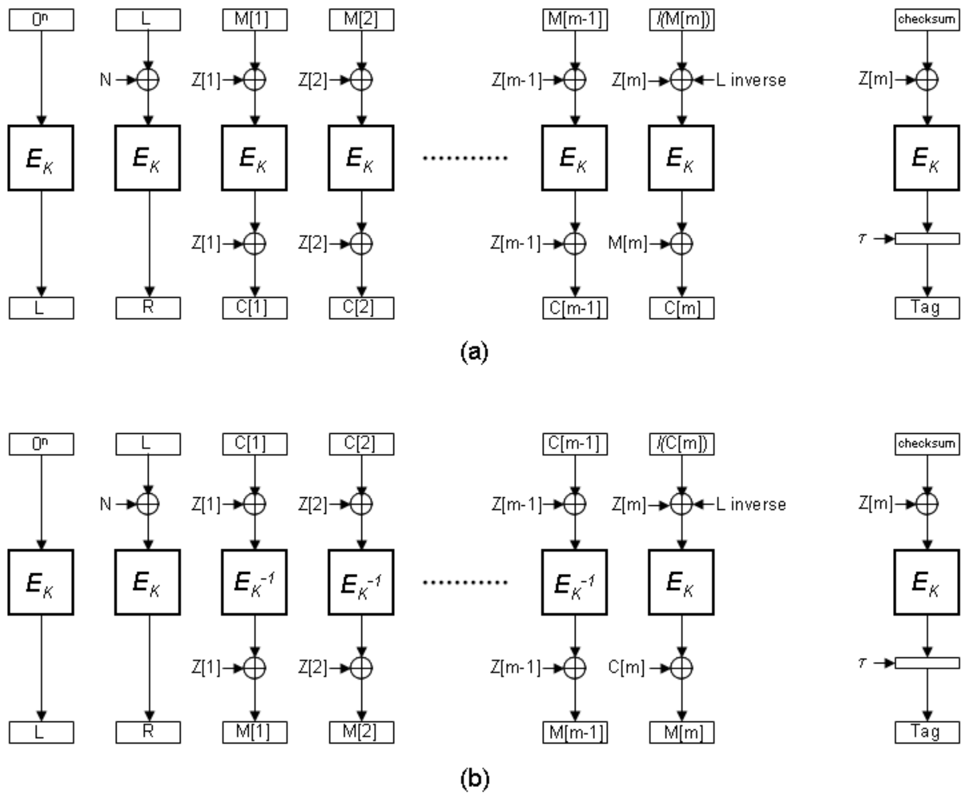


Figure 4.1: (a) OCB Encryption and Tag Generation; (b) OCB Decryption and Tag Verification; E_K: Encryption function of Block Cipher; E_K^{-1}: Decryption function of Block Cipher; L, R: OCB parameters; $L\ inverse$: OCB parameter derived from L; N: Nonce; $M[i]$: i^{th} message block; $M[m]$: Last message block; $C[i]$: i^{th} ciphertext block; $C[m]$: Last ciphertext block; $Z[i]$: i^{th} iteration constant; $l(M[m])$: Length of last message block encoded as a 128-bit string; $l(C[m])$: Length of last ciphertext block encoded as a 128-bit string; τ: Length of the authentication tag requested by the user

encrypted. At the output of the cipher, the original message block is xored with the cipher output to generate a block of the same length as the last message block.

- **Checksum** – OCB mode generates a ciphertext and an authentication tag as the output. In order to generate the authentication tag, checksum is computed by successively xoring message blocks. In case of an incomplete last block, it is padded with trailing zeros before xoring. The computed checksum is then xored with an iteration constant and encrypted to get a 128-bit authentication block. Authentication tag consists of the leftmost τ bits of the authentication block, where τ is a user input. τ determines the security measure of the OCB scheme.

- **Modifications for Decryption** – As can be seen from Fig. 4.1, the only difference between encryption and decryption for OCB mode of operation is the use of reverse cipher functionality (decryption) during block cipher invocations for all message blocks except first two initialization blocks, last message block and the checksum block. Specifically, in Fig. 4.1(b) note that some block cipher invocations use E_K blocks (encryption) while others use E_K^{-1} blocks (decryption).

4.2 Hardware Implementation of OCB Mode

A generic OCB wrapper was designed which can be interfaced with any 128-bit block cipher designed according to the certain specifications. This particular implementation assumes that all input blocks are completely full. Processing of partially filled input blocks is not supported by this design.

4.2.1 Datapath Design

The datapath design is shown in Fig. 4.2. There are two main factors which affect the datapath design in this mode.

1. Input message blocks are processed differently depending upon the type of the block. Various types of blocks and their features are listed below.
    - Initialization blocks are passed to the block cipher without any additional processing.
    - Message blocks (except the last block) are handled in a specific way. Message block $M[i]$ is xored with an iteration constant $Z[i]$ before it is passed to the cipher.
    - The last message block, $M[m]$ is handled differently from other message blocks, as can be seen from Fig. 4.1.
    - Checksum is processed in the same way as message blocks. But since the source of checksum and the message blocks is not the same a separate path has to be designed for passing the processed checksum to the cipher.
2. Output data from the block cipher must be handled differently depending upon the type of block which produced that particular output.
    - Encryptions of initialization blocks are not passed to the output of the OCB wrapper. Such blocks must be stored internally in the wrapper and used for further processing.

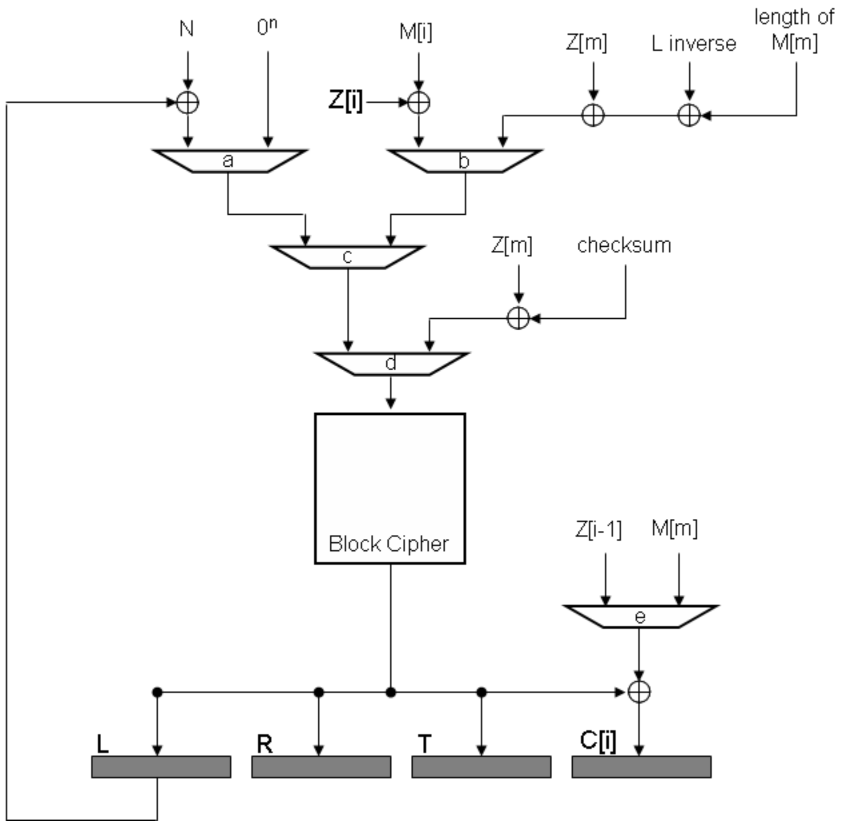


Figure 4.2: OCB Datapath – N:Nonce; $M[i]$:i^{th} message block; $M[m]$:Last message block; $Z[i]$:Iteration constant; $Z[m]$:Iteration constant for last message block; L, R:OCB Initialization parameters; L *inverse*: OCB Parameter derived from L; $C[i]$:i^{th} block of ciphertext; T:OCB Authentication Tag; 0^n:Block of n zeros; n:block size of cipher=128 bits

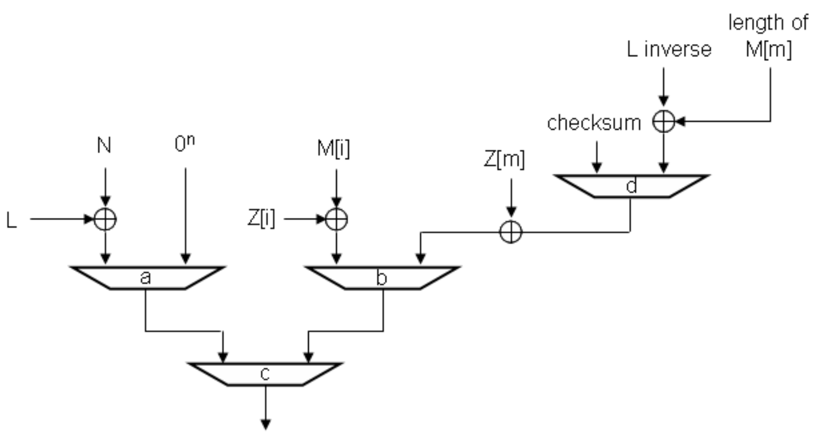


Figure 4.3: Alternate Design for Datapath at Input Side of the Cipher

- Block cipher outputs for non-initialization blocks (message and ciphertext) must be xored with iteration constant, $Z[i]$ before the blocks can exit the wrapper.
- The encrypted checksum block produces a 128-bit tag field. This has to be further truncated depending upon the size of the user input τ.

Datapath Design at Input Side of the Cipher

As can be seen from Fig. 4.2, the datapath at the input of the block cipher consists of a sequence of multiplexers. Each of these multiplexers is controlled using select signals generated by the control logic. The data block which is to be passed to the block cipher propagates through the set of multiplexers and arrives at the input of the cipher. The checksum could possibly be handled using multiplexer `‘b’` instead of using multiplexer `‘d’` but this would have required additional multiplexer at the input of multiplexer `‘b’`. The alternate design is shown in Fig. 4.3. In this design, multiplexer `‘c’` feeds data to the block cipher instead of multiplexer `‘d’` as in Fig. 4.2.

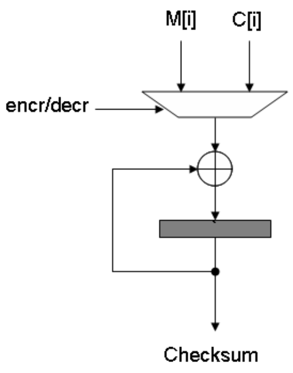


Figure 4.4: OCB Checksum – $M[i]$: Input message block; $C[i]$: Output of register $C[i]$ (Please refer Fig. 4.2 for naming convention)

Also, multiplexers `‘b’` and `‘d’` can be combined together to form a 4-to-1 multiplexer.

The naming convention used in Fig. 4.2 represents encryption. For decryption, ciphertext should be given at the input denoted by $M[i]$. The output during decryption is stored in the register denoted by $C[i]$.

Design of Checksum Module

The module used for computing the checksum is shown in Fig. 4.4. The register stores intermediate values of the checksum which are xored with the subsequent message block. It should be noted that the OCB checksum is computed on the message blocks (*not* input blocks) and hence during decryption data blocks fed to the checksum module are not the input blocks to the OCB wrapper but the output blocks generated by the cipher.

Design of Module for Computing Iteration Constants

Iteration constant, $Z[i]$ is xored with corresponding message block, $M[i]$. $Z[1]$ is computed from OCB parameters, L and R. Subsequent values of $Z[i]$ are computed from $Z[i-1]$. This can be represented using Eqn. 4.1 and 4.2.

$$Z[1] = (N \oplus L) \tag{4.1}$$

$$Z[i] = Z[i-1] \oplus L(ntz(i)) \;\ldots\; for\ i \geq 2 \tag{4.2}$$

In Eqn.4.2, $ntz(i)$ denotes the number of trailing zeros in the integer i. The integer i is the index used for numbering the message blocks. This design assumes that the size of the message is less than 2^{64} blocks i.e. the i can be represented using 64 bits. If the design is used to target a known application, with an upper limit on message size, i can be represented using lesser number of bits. This would improve the circuit timing since it would reduce the size of an adder which increments i by one every clock cycle. The basic building block for generating the value of $ntz(i)$ is a priority encoder as shown in Fig. 4.5 with an additional output `all_0s`. The truth table for this component is shown in Table. 4.1. Input bit 0 has the highest priority. The output `all_0s` is asserted LOW when all inputs are zeros. This output is used to cascade the module in order to handle an input i with length greater than 8.

As can be seen in Fig. 4.6, the `all_0s` outputs of the first level (leftmost column) of priority encoders acts as inputs to the next level. The encoder at the output side generates the most significant three bits of the six bit output. The least significant three bits are passed from the output of the appropriate first level encoder.

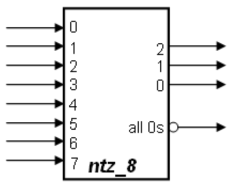


Figure 4.5: Modified priority encoder for computing $ntz(i)$ $(i \leq 8)$

Table 4.1: Truth Table for Priority Encoder ntz_8

Input [7..0]	Outputs	
	Output[2..0]	all_0s
0000 0000	000	0
XXXX XXX1	000	1
XXXX XX10	001	1
XXXX X100	010	1
XXXX 1000	011	1
XXX1 0000	100	1
XX10 0000	101	1
X100 0000	110	1
1000 0000	111	1

An alternate design implementing this functionality would be a memory which stores the values of $ntz(i)$ at address location i. For an n-bit index i, the size of the memory required would be $2^{n-1} \times n$. The memory requirement is reduced by half since odd indices have a value of 0 for number of trailing zeros. Implementation using a memory would consume substantially more resources than the chosen design method.

Also, $L(i)$ is computed from $L(i-1)$ using a shift and conditional xor.

$$L(x) = \begin{cases} L(x-1) \ll 1, & \text{if } msb\{L(x-1)\} = 0 \\ (L(x-1) \ll 1) \oplus 0^{120}10000111, & \text{if } msb\{L(x-1)\} = 1 \end{cases}$$

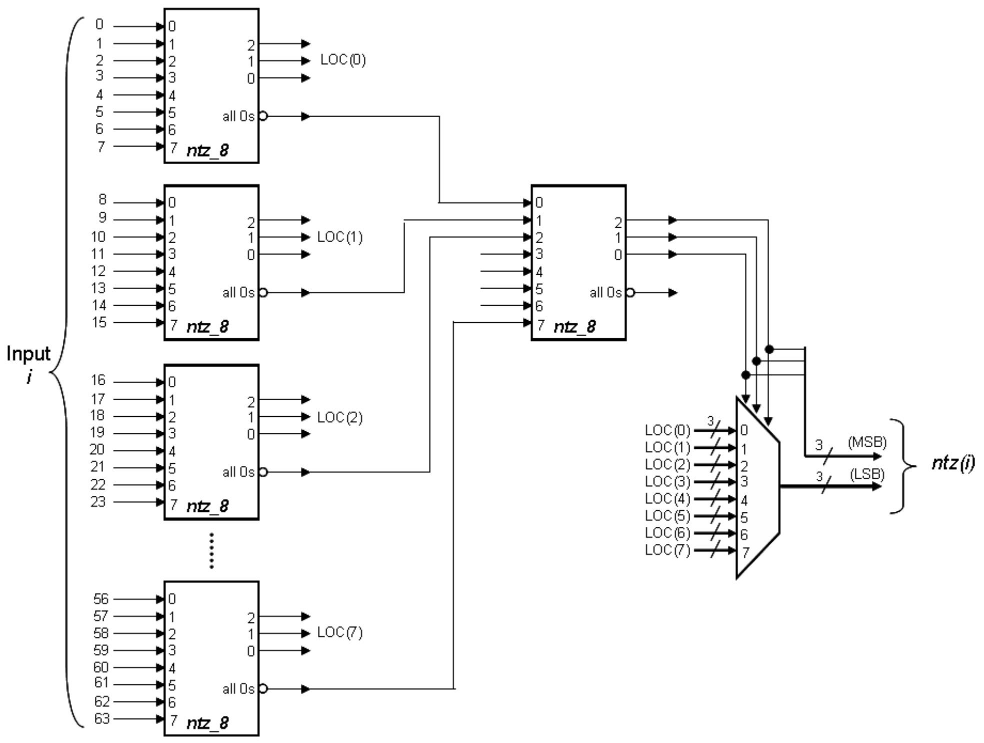


Figure 4.6: Cascaded priority encoders for computing $ntz(i)$ $(i \leq 64)$

When $ntz(i)$ is generated, in order to compute $L(ntz(i))$, the value of $L(ntz(i)-1)$ is required. Hence, all values of $L(i)$ are stored in a memory rather than computing them on the fly. In this particular design a dual-ported BlockRAM is used for implementing memory.

Fig. 4.7 shows the design of the module for computing values of $Z[i]$. Referring to Fig. 4.2, it can be seen that the iteration constants are required for processing at the input of the cipher and also at the output. A dual register pipeline holds the last two values of Z.

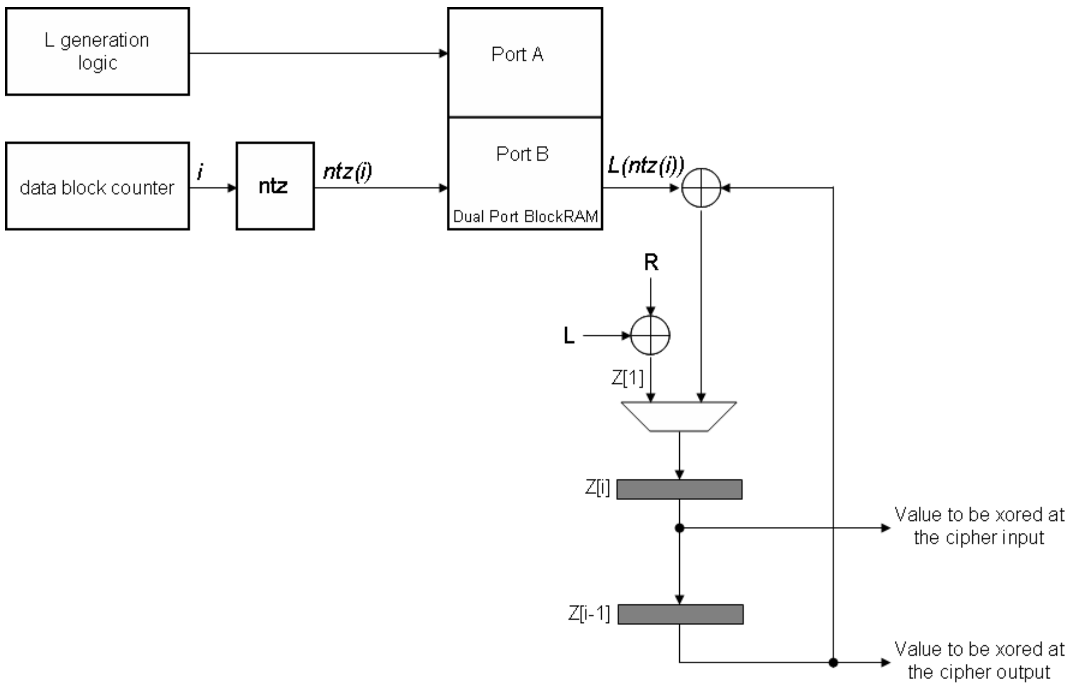


Figure 4.7: Computation of iteration constant ($Z[i]$)

Datapath Design at Output Side of the Cipher

The block cipher writes to four registers in the datapath at the output side of the cipher . These are –

- Register L – This register stores the value of $L(0)$, i.e. the encryption output generated by an input consisting of a block of all zeros.
- Register R – This register stores the value of R, i.e. the encryption output generated by an input block $(N \oplus L)$, where N is the nonce and $L = L(0)$.
- Register $C[i]$ – During encryption, this register is used to store the last generated ciphertext block before it is accepted by the output fifo. The same register is used to store the computed plaintext output during decryption.

- Register T – This register is used to store the complete 128-bit tag generated from the checksum.

The ciphertext can be written directly to the output fifo instead of buffering it in register $C[i]$. This would be feasible during encryption - tag generation operation. During decryption, the OCB output (plaintext) needs to be xored successively with partial checksum in order to compute the final checksum. Hence the output during decryption needs to be buffered inside the wrapper instead of passing it directly to the output fifo.

4.2.2 Design of Control Logic

Control logic for OCB wrapper is designed as separate modules controlling specific sections of the wrapper while exchanging minimal number of signals among control modules. This is preferable to having a single logic for controlling the wrapper since it infers a state machine with a large number of states. This in turn is difficult to design perfectly since there might be some states which are unaccounted for. A modular approach helps in this regard since smaller state machines can be completely specified and are easier to debug.

Communication Control Logic

Communication control logic acts like a scheduler and passes data blocks in the correct order to the block cipher. All communication between the wrapper and the block cipher takes place via a two-signal handshake mechanism; an input signal to the block cipher, `valid` is used to trigger the block cipher to read new data and the cipher acknowledges the data read by a pulse on an output signal, `read`. It is the job of the communication control logic to interpret the validity of the next scheduled data block and generate a `valid` signal to the block cipher.

Encryption/Decryption Control Logic

In the OCB mode of operation, both forward (encryption) and reverse (decryption) block cipher operations are required. Hence, the underlying block cipher has a control signal (`encr/decr`) to select between encryption and decryption functions. Referring to Fig. 4.1, it can be seen that during OCB encryption, only forward function of the cipher is used. But, during OCB decryption, some blocks are processed by the cipher using the forward function while some are processed using the reverse function. A control module generates the `encr/decr` control signal depending upon the operation being performed and the particular type of block being processed.

Design Simplification using Auxiliary Tags

A novel idea has been used to simplify the design of control logic for modes of operation. In order to distinguish among various block types, each type of block is marked with an auxiliary tag which is passed to the cipher module via an independent port along with the corresponding data block. The cipher does not process the tag but just passes it through the internal pipeline so that the auxiliary tag for a particular block exits the cipher at the same time as the encrypted/decrypted result. Also, the cipher provides a look-ahead capability which allows the wrapper to look at the exiting auxiliary tag up to three clock cycles before it appears at the cipher output. This gives the wrapper additional time to get ready for processing the exiting data block.

The control logic at the output reads the auxiliary tag exiting from the block cipher and performs one of the control functions below –

1. Generates control signals to enable proper datapath registers at the output of the block cipher.

2. Generates control signals at the output of the wrapper for output FIFO control

3. Generates indication signals at the wrapper output like *Start of Frame (SOF)* and *End of Frame (EOF)* indications.

Chapter 5: CCM Mode of Operation

5.1 Introduction

Counter with Cipher Block Chaining - Message Authentication Code, abbreviated as CCM is an authenticated-encryption mode of operation of block cipher. CCM can be used in conjunction with any approved 128-bit block cipher like AES [19].

CCM is designed for use in a packet environment, where all the necessary data is available in storage before CCM processing is applied to that data. This implies that CCM is not online i.e. it cannot handle stream processing. Another useful feature of CCM mode of operation is that it can handle associated data i.e. data which must be authenticated but not encrypted. This might be particularly useful in networking applications where data blocks like packet headers are usually sent in the clear, but the receiver must be able to ascertain their source. CCM has been specified in the draft IEEE 802.11i standard for wireless networks. It has also been specified in IEEE 802.15 (Wireless Personal Area Networks) and 802.16 (Broadband Wireless Metropolitan Area Networks) standards.

5.1.1 CCM Encryption and Decryption

CCM processing consists of two pairs of related processes: generation-encryption and decryption-verification. These processes combine two cryptographic primitives, encryption using counter mode and authentication using CBC-MAC. CCM provides

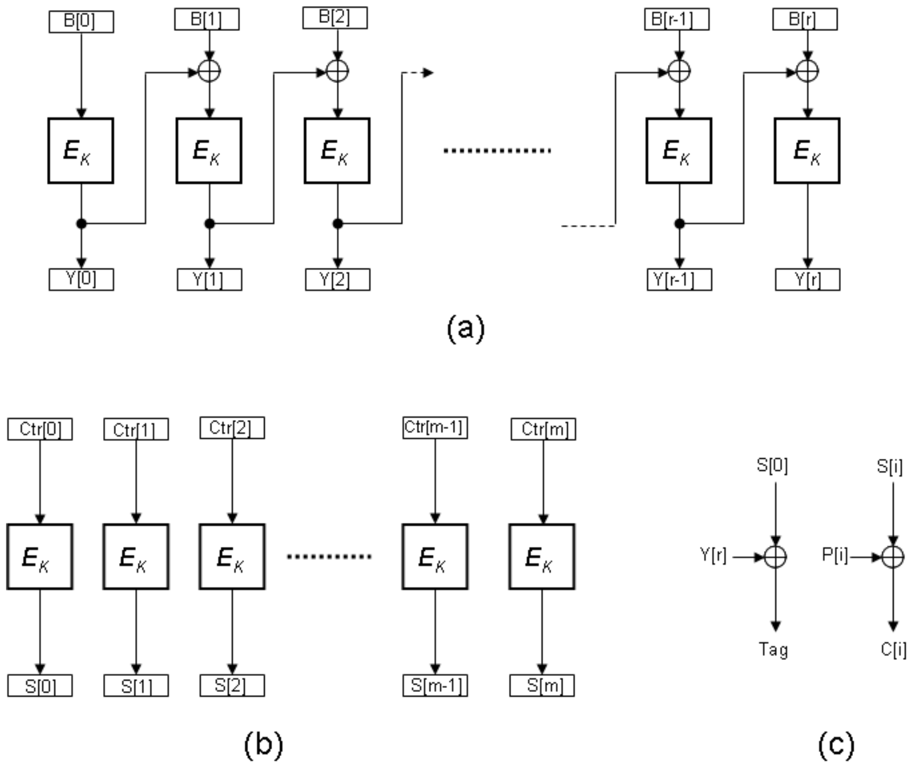


Figure 5.1: CCM Encryption – (a) Generation of CBC-MAC; (b)Encryption using Counter Mode; (c)Final CCM processing; E_K: Encryption function of block cipher; $B[i]$: Processed CCM input block; $Y[i]$: CBC output blocks; $Ctr[i]$: i^{th} input block in Counter mode; $S[i]$: Output blocks in Counter Mode; $P[i]$: i^{th} payload block

considerable advantage due to the fact that only the forward (encryption) function of the block cipher is used. This is useful in case of block ciphers like AES which have a significantly different forward and reverse function [20].

A block diagram representation of CCM Encryption is shown in Fig. 5.1. Naming conventions in CCM are a little complicated since the mode is based on many parameters. Detailed information about CCM can be found in [19]. In the simplest sense, CCM encryption processes two types of input blocks –

- **Payload Block** - This type of data block must be encrypted and authenticated. These blocks are denoted as $P[i]$.
- **Associated Data Block** - This type of data block must be authenticated but not encrypted.

Table 5.1: CCM Block Types and Operations

Block Type	Operations	
	Encryption	Authentication
Payload Blocks	✓	✓
Associated Data Blocks		✓

The naming convention used throughout this chapter is consistent with Fig. 5.1. Data blocks which are authenticated are denoted as $B[i]$. Hence blocks B include blocks P and additional associated data blocks.

Important steps performs in CCM processing can be summarized as follows –

- **Pre-processing of the Message** – In this step, the message is arranged to get it into a form suitable for CCM processing. The first block $B[0]$ uniquely determines the Nonce (N) and the length of the payload, $l(m)$ [19]. If associated data is present, the block(s) after $B[0]$ will consist of associated data until there is no more associated data left. If the last associated data block is not a complete block, then it is padded with zeros. After the associated data blocks, payload blocks follow.

- **Computing CBC-MAC** – Blocks $B[i]$ are encrypted using CBC mode and the output of final encryption is the Message Authentication Code (MAC) which can be used for authentication. The CBC-MAC is denoted as $Y[r]$ in Fig. 5.1.

- **Encryption using Counter Mode** – Payload blocks are encrypted using Counter (CTR) Mode of operation. The initial value of counter variable (denoted as $Ctr[0]$ in Fig. 5.1) is an Initialization Vector (IV) input to the CCM wrapper. Subsequent values of counter variable are computed by incrementing the current counter value by one.

- **Processing Counter Mode Encryption** – The blocks generated by Counter Mode encryption are denoted as $S[i]$ for corresponding $Ctr[i]$ counter variable. $S[0]$ is processed differently than the following $S[i]$ blocks.

- **Computing Authentication Tag** – Authentication tag (denoted as Tag in Fig. 5.1) is generated by xoring $S[0]$ with the computed CBC-MAC. The length of the authentication tag to be output by the CCM mode wrapper can be set by the user and it is just a truncation of the original 128-bit MAC.

- **Computing the Ciphertext** – Ciphertext is computed by xoring the result of counter encryptions, $S[i]$ with the corresponding payload blocks $P[i]$, for $i > 0$. The final output of the CCM mode of operation is $C\|Tag$, where $\|$ indicates concatenation of two strings.

5.2 Hardware Implementation of CCM Module

A generic CCM wrapper is designed which can be interfaced with any 128-bit block cipher designed according to certain specifications. This particular implementation assumes that all input data blocks are completely full. Partially filled blocks cannot be handled by the CCM wrapper. Also, all preprocessing described in Section 5.1.1 is done outside the CCM wrapper.

5.2.1 Datapath Design

The design of datapath for CCM mode of operation is relatively straightforward but controlling datapath elements is a complex task. Factors which affect CCM datapath design are –

1. As shown in Table 5.1, some blocks passed to the CCM wrapper must be only authenticated while others must be encrypted as well as authenticated. Authentication is done using CBC mode of operation while encryption is done using Counter mode. Depending upon the type of block, either of the two modes must be selected.

2. Since only one instantiation of block cipher is used in order to limit the implementation area, the same block cipher has to be used in both CBC and Counter modes. For blocks which must be authenticated as well as encrypted, alternate invocations of the block cipher are necessary, once in the CBC mode and then in the Counter mode.

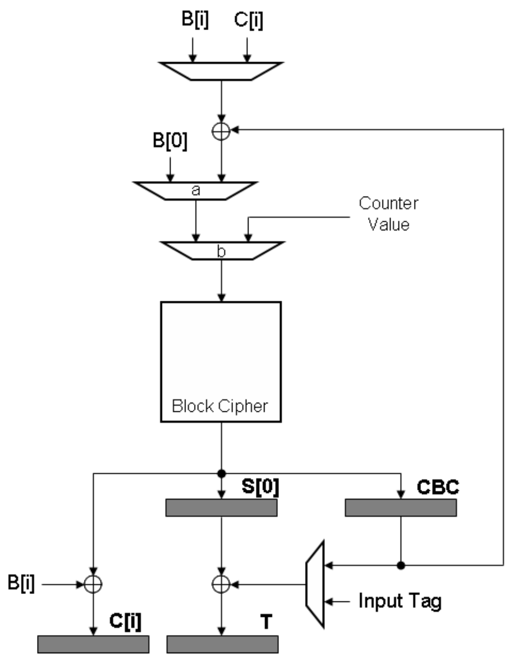


Figure 5.2: CCM Datapath – $B[i]$: Pre-processed input block; $S[0] = E_K(Ctr[0])$; $C[i]$: i^{th} block of ciphertext; CBC: Encryptions in CBC mode; T: Authentication Tag

3. The first block encrypted using Counter mode is buffered for later use. Subsequent counter encryption outputs are xored with corresponding payload blocks to produce CCM ciphertext.

4. Encryptions in CBC mode must be buffered since the output has to be later xored with the next incoming message block before it is passed to the block cipher. Also, consecutive CBC encryptions restrict paralellization of this scheme.

Datapath Design at the Input Side of the Cipher

As explained earlier, CCM processing involves two cryptographic primitives, encryption using Counter mode and authentication using CBC mode of operation. The following explanation is with respect to Fig. 5.2. Multiplexer `'b'` at the input of the block cipher selects the mode in the sense that it passes proper inputs to the block cipher depending upon required mode of operation. As such, the block cipher processes all data in the same way. It is the wrapper which actually processes the inputs and outputs of the block cipher that determines the mode of operation. The input to multiplexer `'b'` denoted as `Counter value` is passed to the cipher when the counter mode of operation is to be invoked, otherwise data from the other input is passed to the cipher. The XOR at the input of multiplexer `'a'` handles chaining in the CBC mode as shown in Fig. 5.1(a). The naming convention used in Fig. 5.2 represents CCM encryption. For decryption, the pre-processed ciphertext is input through the input $B[0]$ and the output plaintext is buffered in register $C[i]$.

Processing Counter Input to Block Cipher

In Counter mode of operation, the IV/Nonce input is used for the initial counter invocation and for subsequent invocations, the previous IV is incremented by one. A feature of CCM, which adds complexity to the design is support for variable size nonces. The format for CCM IV/Nonce is shown in Table. 5.2. Valid values for L are from 2 to 8. Therefore the size of the counter variable can range from 2 octets to 8 octets. Depending upon the size of the counter, only a specific number of least significant bits must be incremented. Rest of the IV remains the same throughout. This is shown in Fig. 5.3.

Table 5.2: CCM Nonce Format

Octet No.	0	$1 \ldots 15-L$	$16-L \ldots 15$
Contents	Flags	Nonce N	Counter $Ctr[0]$

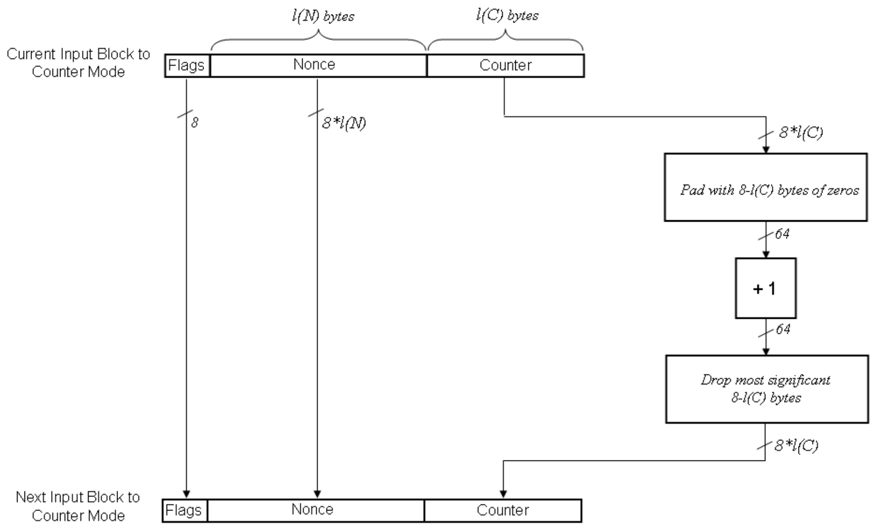

l(N) bytes
l(C) bytes
Current Input Block to Counter Mode
Flags
Nonce
Counter
8
8*l(N)
8*l(C)
Pad with 8-l(C) bytes of zeros
64
+1
64
Drop most significant 8-l(C) bytes
8*l(C)
Next Input Block to Counter Mode
Flags
Nonce
Counter

Figure 5.3: Counter Mode Input Processing in CCM Mode – $l(N)$: Length of Nonce field; $l(C)$: Length of Counter field; $l(N) + l(C) + 1 = 16$

Handling Associated Data Blocks

As mentioned earlier, if associated data is present in a frame then it should be passed to the block cipher immediately after initialization block $B[0]$. After all associated data blocks have been passed to the block cipher, only then payload blocks are passed. In order to correctly handle data, the CCM wrapper must keep track of the number of associated data blocks present in the frame. In order to determine the number of associated data blocks, two control fields must be checked.

1. Bit no. 6 in the Flags byte in frame $B[0]$. This field is denoted as *Adata* in

Table 5.3: CCM Block $B[0]$ Format

Octet No.	0	$1 \ldots 15 - L$	$16 - L \ldots 15$
Contents	Flags	Nonce N	Length of the payload $l(m)$

Table 5.4: CCM Flags Format

Bit No.	7	6	5	4	3	2	1	0
Contents	–	Adata	M			L		

Table. 5.4. This bit indicates whether associated data is present ($Adata = 1$) or not. If associated data is present, then block $B[1]$ must be checked in order to determine the length of associated data.

2. The format for block $B[1]$ is shown in Table. 5.5. The first two bytes of the block denote the size of associated data blocks in terms of octets. It must be mentioned here that specification for CCM mode of operation specifies variable length formats for encoding $l(Adata)$. The CCMP protocol specified in the IEEE 802.11i standard, which is a variant of the original CCM mode, reserves only the first two octets for specifying $l(Adata)$. The CCM wrapper has been designed to support this feature of the CCMP protocol.

Table 5.5: First Associated Data Frame Format (Block $B[1]$)

Octet No.	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Contents	$l(Adata)$		… Adata …													

Once, the length information about associated data is extracted from the blocks $B[0]$ and $B[1]$, it is straightforward to count the number of blocks being passed to the block cipher and determine the end of associated data blocks.

Datapath Design at the Output Side of the Cipher

The block cipher writes to four registers in the datapath at the output side of the block cipher as can be seen from Fig. 5.2. These are –

- Register $S[0]$ - This register stores the encrypted output of $Ctr[0]$.
- Register $C[i]$ - This register stores the i^{th} block of ciphertext/plaintext during encryption/decryption.
- Register CBC - This register acts as intermediate storage for the output of encryptions performed in CBC mode of operation.
- Register T - This register stores the 128-bit authentication tag which can be later truncated to user-defined length.

5.2.2 Design of Control Logic

The control logic for CCM wrapper is designed as separate modules controlling specific portions of the wrapper. In case of CCM mode no control logic is required for selecting between encryption and decryption functions, since CCM encryption as well as decryption uses only forward (encryption) function of the block cipher. Simplification of control logic design is achieved by using auxiliary tags for tagging different types of input blocks to the block cipher. The concept of auxiliary tags is explained in Section 4.2.2.

The design of control logic is based on Table. 5.6. The control logic acts as a scheduler and controls the sequence of operations. The main aspect of designing the control logic is to determine the optimal sequence of operations in hardware. The order of operations represented by the software-based pseudocode presented in [19] is shown in Fig. 5.1. The software-based pseudocode involves first performing a CBC-MAC operation on the entire data block and then selectively encrypting the payload. If the same order of operations is followed in the hardware implementation, then storage equal to the size of the payload would be required in order to store all payload blocks which have to pass through the cipher twice. The modification is to get the same block to pass through the cipher twice in successive cipher invocations; the first pass for encryption, the next for authentication.

Table 5.6: Order of Operations in Hardware Implementation of CCM – *Auth*: Authentication; *Encr*: Encryption; *Adata*: Associated Data; n: Number of Adata Blocks; m: Number of Payload blocks; r=m+n; All notations consistent with Fig. 5.1

Block Type	Notation	Input to Block Cipher	Mode		Operation	
			CBC	CTR	Auth	Encr
Initialization Block	$B[0]$	$B[0]$	✓		✓	
Adata Block[1]	$B[1]$	$B[1] \oplus Y[0]$	✓		✓	
Adata Block[2]	$B[2]$	$B[2] \oplus Y[1]$	✓		✓	
…	…	…	…	…	…	…
Adata Block[n-1]	$B[n\text{-}1]$	$B[n\text{-}1]\oplus Y[n\text{-}2]$	✓		✓	
Adata Block[n]	$B[n]$	$B[n]\oplus Y[n\text{-}1]$	✓		✓	
Payload Block[1]	P[1]				✓	✓
		$Ctr[0]$		✓		
		$Ctr[1]$		✓		
		$P[1]\oplus Y[n]$	✓			
Payload Block[2]	P[2]				✓	✓
		$Ctr[2]$		✓		
		$P[2]\oplus Y[n+1]$	✓			
Payload Block[3]	P[3]				✓	✓
		$Ctr[3]$		✓		
		$P[3]\oplus Y[n+2]$	✓			
…	…	…	…	…	…	…
…	…	…	…	…	…	…
Payload Block[m-1]	P[m-1]				✓	✓
		$Ctr[m\text{-}1]$		✓		
		$P[m\text{-}1]\oplus Y[n+m\text{-}2]$	✓			
Payload Block[m]	P[m]				✓	✓
		$Ctr[m]$		✓		
		$P[m]\oplus Y[n+m\text{-}1]$	✓			

Chapter 6: EAX Mode of Operation

6.1 Introduction

EAX mode of operation is a two pass "Authenticated Encryption with Associated Data" (AEAD) scheme based on any approved block cipher. EAX is an improvement over CCM mode of operation which has a few drawbacks like complex parametrization and inability to handle online data. EAX mode of operation supports streaming data input and does not require data to be in storage before applying EAX processing. EAX also avoids complex padding and pre-processing rules like CCM, thereby making it more efficient from an implementation perspective [21].

EAX does not impose any restrictions on the block size of the underlying block cipher. Message expansion is minimal; the length of the output is only τ bits more than the length of the message, where τ is the length of the authentication tag. Furthermore, implementation efficiency is improved by the fact that EAX uses only the forward (encryption) functionality of the underlying block cipher. This is especially appealing when EAX mode is used with block ciphers like AES (Rijndael) that have a significantly different forward and reverse function [20].

6.1.1 EAX Encryption and Decryption

EAX mode of operation uses two mutually invertible processes –

1. Encryption and authentication tag generation

2. Decryption and authentication tag verification

EAX mode of operation uses three parameters - Nonce, Header and Message. This combination is called a triple. Provable security is related to indistinguishability from random bits and the inability of an attacker to produce a new but a valid $< nonce, header, ciphertext >$ triple.

The mode uses the underlying block cipher in CBC and CTR modes of operation. Since both these modes need only the forward functionality of the cipher, it is sufficient to implement only the encryption function of the block cipher, as stated earlier. An interesting aspect of the mode is the tweaked version of OMAC implementation. It utilizes multiplication over $GF(2^{128})$. Parameter L is defined as $L = E_K(0^n)$ where K is the secret key of the underlying block cipher and n is the block size. The convention 0^n indicates a string of n zeros. Parameters B and P are defined as – $B = 2L$ and $P = 4L$, where multiplication is carried out over $GF(2^8)$. Multiplication by 2 over $GF(2^{128})$ is implemented as a single bit left shift and conditional XOR. A design related security issue is that both these parameters must be calculated in constant time in order to be secure against timing attacks.

Encryption and Authentication Tag Generation

Dataflow diagram for encryption and signature generation in EAX mode of operation is shown in Fig. 6.1. Note the subtle change in font used to differentiate between the input (normal font in caps) and output of OMAC function (italicized font in caps). The EAX mode of operation uses the underlying block cipher in CBC (Cipher Block Chaining) and CTR (Counter) modes of operation. CBC mode used is a variant of OMAC (One-Keyed MAC) and is referred to as “tweaked” OMAC in [21].

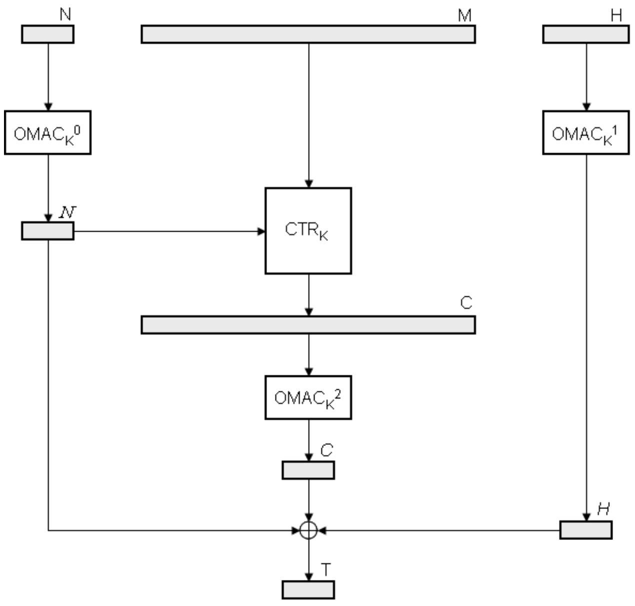


Figure 6.1: Encryption - Signature Generation in EAX Mode of Operation – M: Message; N: Nonce; H: Header; $N = OMAC_K^0$(N); $H = OMAC_K^1$(H); C= CTR_K(M); $C = OMAC_K^2$(C); T= $N \oplus C \oplus H$; Ciphertext=C$\|$T

6.1.2 Modified OMAC Operation

One-key CBC MAC is an improvement over security flaws in CBC-MAC. Bellare, Kilian, and Rogaway [22] proved the security of the CBC MAC for fixed message length mn bits, where n is the block length of the underlying block cipher E. However, it is well known that the CBC MAC is not secure unless the message length is fixed. OMAC is proved to be secure for an arbitrary length message and uses a single secret key, K [10].

The original OMAC algorithm has been “tweaked” a bit in order to incorporate it in the EAX mode of operation [21]. As can be seen in Fig. 6.3, OMAC is similar to CBC-MAC except for the processing of the last block. If the last block is completely

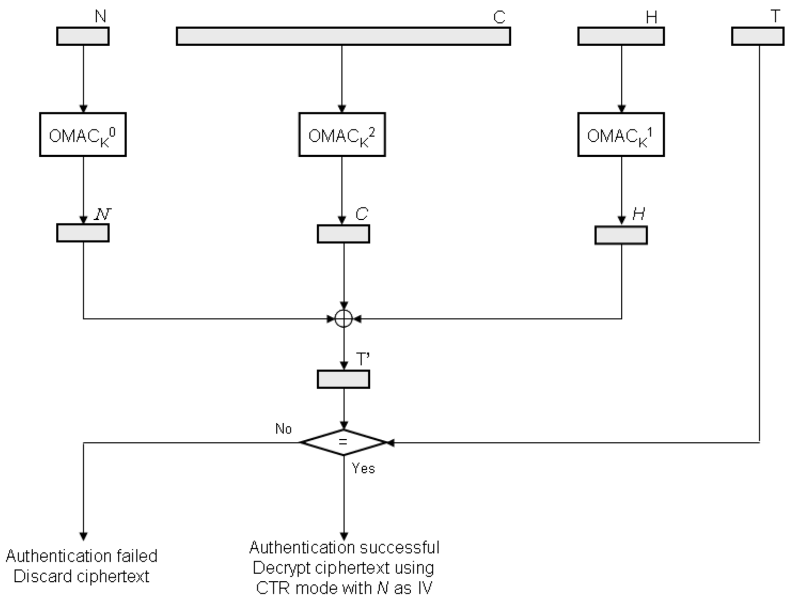
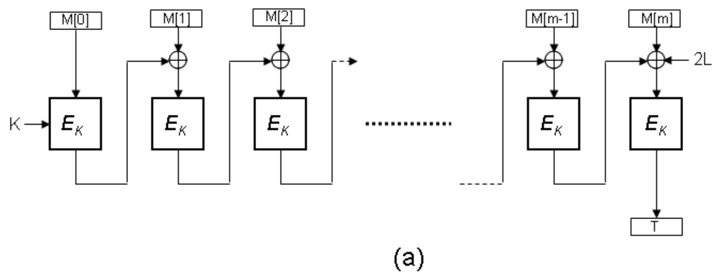


Figure 6.2: Decryption - Verification in EAX Mode of Operation – C: Ciphertext; N: Nonce; H: Header; $\mathcal{N} = OMAC_K^0$(N); $\mathcal{H} = OMAC_K^1$(H); $\mathcal{C} = OMAC_K^2$(C); T$' = \mathcal{N} \oplus \mathcal{C} \oplus \mathcal{H}$; T: Input Tag

full, then the last message block, $M[m]$ is xored with ciphertext of previous block and a constant, $2L$, where $L = E_K(0)$. If the last block is not completely filled, then it is padded with a string 10^i, where i is the number of zeros required to completely fill the block. The padded block, denoted as $MPad$ in Fig. 6.3, is then xored with the constant, $4L$. The ciphertext generated by the last block, denoted as T, is the output of OMAC.

In the specification of EAX mode of operation OMAC invocations are denoted as $\mathcal{M} = OMAC_K^t$(M), where K is the secret key for the block cipher, t is an integer constant and M is the input message. Note the difference in the fonts used for denoting the message (M) and the OMAC output ($\mathcal{M}$).



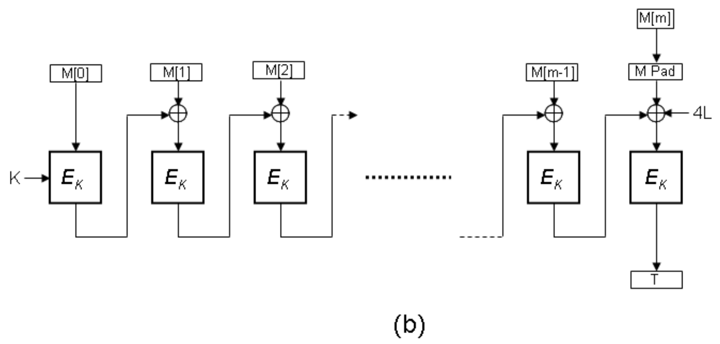


Figure 6.3: One-Key CBC MAC (OMAC) – (a) OMAC operation completely filled last block, (b) OMAC operation with partially filled last block; $M[i]$: i^{th} message block; $L = E_K(0)$ (with reference to EAX); T: Tag; $MPad = M\|10^i$; K: Secret key; E_K: Encryption function of block cipher; $\|$ denotes string concatenation

$$OMAC_K^t(\mathrm{M}) = OMAC_K([t]_n\|\mathrm{M}),$$

where $[t]_n$ is the representation of integer t as a n-bit number, and n is the block size of the block cipher.

6.2 Hardware Implementation of EAX mode

A generic EAX wrapper was designed which can be interfaced with any 128-bit block cipher designed according to specifications. This particular implementation assumes that all input data blocks are completely full. Partially filled blocks are not supported

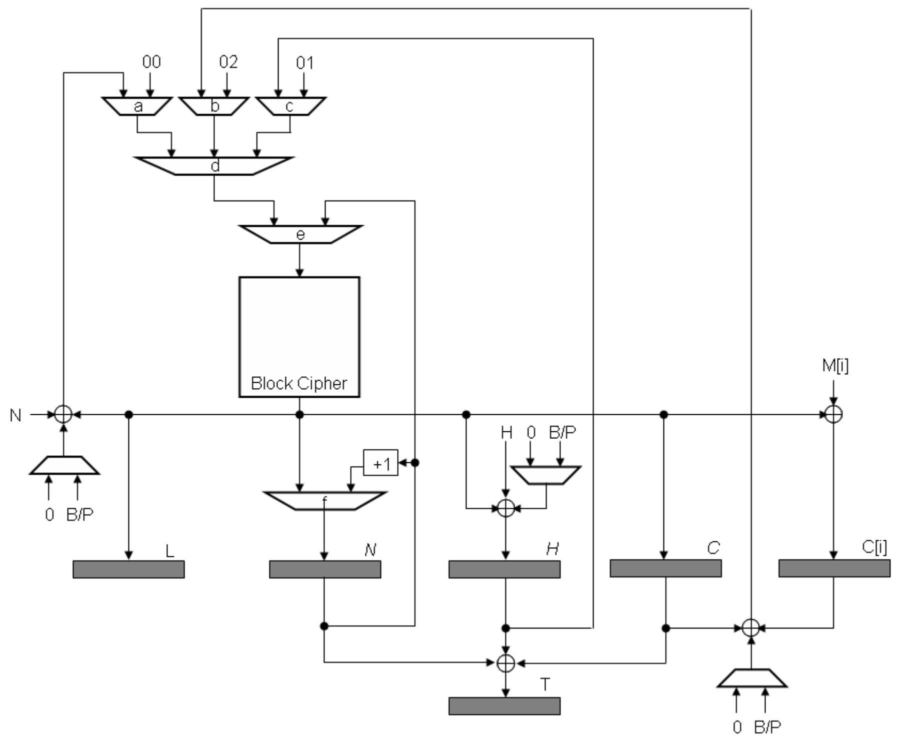


Figure 6.4: EAX Datapath – M[i]: i^{th} message block; C[i]: i^{th} ciphertext block; N: Input nonce; H: Input header; L=$E_K(0)$; B=2L; P=4L; T: Authentication tag

by this implementation.

6.2.1 Datapath Design

Datapath design for EAX is the most complex among the implemented authenticated encryption modes of operation. This is because of the interleaved CTR and OMAC invocations required in EAX mode. Factors which affect EAX datapath design are –

- EAX mode uses CTR mode for encryption and a variant of CBC mode for authentication. Single block cipher instantiation is used in order to limit implementation area.

- The design of the block which generates parameters B and P is quite straightforward. B and P are derived from the parameter L which is defined as $L = E_K(0)$, where E is the underlying block cipher and K is the secret key. $L \in GF(2^{128})$. $B = 2L$ and $P = 4L$, where $B, P \in GF(2^{128})$. If parameters B and P are computed on the fly then they must be computed in equivalent time in order to prevent timing attack. Hence, B and P are pre-computed (in variable time) and used in computations as required rather than computing them on the fly.

- Data processing using OMAC makes selections based on whether the last block is completely full or not. Since, this implementation always assumes complete data blocks, this decision is not required to be taken in hardware. Fig. 6.4 does have reference to a parameter B/P which is used to indicate the OMAC choice as stated in the original EAX specification.

Datapath Design at Input Side of the Cipher

Design aspects of EAX datapath are explained with respect to Fig. 6.4. Multiplexers `‘a’`, `‘b’`, `‘c’` and `‘d’` are used for implementation of OMAC mode. Referring to Fig. 6.1, $N = OMAC_K^0$(N). i.e $N = OMAC_K([0]_n \| \text{N})$. Hence, in order to compute N, two block cipher calls are required, assuming that the input nonce (N) is a 128-bit string. The first block cipher call is for encryption of $[0]_n$. The next block cipher call is for encryption of L$\oplus$B$\oplus$N, where L=$E_K([0]_n)$ and B=2L. In order to implement the OMAC operation with input nonce, N as the argument, multiplexer `‘a’` first passes the string $[0]_n$. During the subsequent pass through the block cipher, the select line for multiplexer `‘a’` is flipped and the string L$\oplus$B$\oplus$N is passed through it.

Multiplexer `‘b’` is used for processing $OMAC_K^2$(C), while multiplexer `‘c’` processes $OMAC_K^1$(H). Multiplexer `‘d’` uses a 2-bit select line and selects either the Nonce, Header or Ciphertext related blocks to be passed to the block cipher in the correct sequence. Interleaved invocations of block cipher in CBC (OMAC is a CBC MAC variant) and CTR modes is controlled by multiplexer `‘e’`.

Processing Counter Input to Block Cipher

As can be seen from Fig. 6.1, $N = OMAC_K^0$(N) acts as the IV for CTR mode of operation. This requires that the nonce, N be processed before any input message/ciphertext blocks can be processed in CTR mode. The processed nonce is stored in Register N at the output side of the block cipher. After every CTR mode invocation, the value of Register N is incremented and made ready for processing the next block in CTR mode.

Handling Message Header

Message Header, H is processed using OMAC algorithm. Header is analogous to associated data in CCM mode of operation. Headers are only authenticated but not encrypted.

Processing of header is independent of processing other blocks like nonce and message. Thus there is no restriction on the sequence in which the message header must be processed. The output of $OMAC_K^1$(H), is stored in register H and used in the final processing for computing the authentication tag, T.

Datapath Design at the Output Side of the Cipher

Output data from the block cipher is written to registers at the output side of the cipher. These are –

- Register N: This register is used to store the intermediate as well as the final value of the nonce processed using OMAC.
- Register L: As stated earlier, EAX mode of operation uses three parameters – L, B and P. L is buffered since it is later used for computation of parameters, B and P.
- Register H: This register is used to store intermediate as well as final values of the header processed using OMAC.
- Register C[i]: In CTR mode of operation, encryption of i^{th} CTR block is xored with i^{th} message block, M[i] in order to produce i^{th} block of ciphertext, C[i]. This is the ciphertext output of the EAX mode. The same register is used to store the output (plaintext) during decryption.
- Register C: This register is used to store the intermediate as well as final results of OMAC processing for ciphertext stored in register C[i].

Calculating EAX parameters, B and P

EAX parameters, B and P are calculated from parameter L. Computation of L is described in Section 6.2.1. Computations are performed over $GF(2^{128})$. The irreducible polynomial chosen for performing Galois Field computations is $\mathrm{x}^{127}+\mathrm{x}^{7}+\mathrm{x}^{2}+\mathrm{x}+1$.

Multiplication by 2 over $GF(2^n)$ is implemented as a single bit left shift and conditional xor as expressed in the following equation.

$$B = 2L = \begin{cases} L \ll 1, & \text{if } msb(L) = 0 \\ (L \ll 1) \oplus 0^{120}10000111, & \text{if } msb(L) = 1 \end{cases}$$

Similarly, $P = 4L$ is computed by multiplying B by 2 over $GF(2^{128})$.

6.2.2 Design of Control Logic

Although the datapath for EAX mode of operation is more complex as compared to other modes of operation, comprehensively defined auxiliary tags overcomes the requirement for a complex control logic. The control logic in case of EAX mode of operation acts like a task scheduler. It monitors the auxiliary tag entering the exiting the block cipher and generates proper logic signals for multiplexers at the input side of the block cipher and `enable` signals for registers at the output side of the cipher. Additionally, EAX mode of operation uses only the forward (encryption) function of block cipher. Therefore, no control is required for selecting between encryption and decryption functions of the block cipher.

Chapter 7: Implementation Results

The results for hardware implementation of selected authentication techniques are tabulated in this chapter. Area utilization and timing results are provided for FPGA implementations as well as ASIC synthesis.

7.1 Throughput Computations for Modes of Operations and Generic Composition Schemes

Selection of a particular authentication technique or mode of operation would depend upon two main factors, implementation area and throughput. For authenticated-encryption schemes, the throughput is not a static value for given parameters, but is dependent on the length of the message. Derivation of formulas, along with appropriateness of approximation techniques used is described in this section.

Let us define some parameters that will be used for throughput computations.

L : Length of the message in terms of blocks (1 block=128 bits)

t_p : Minimum clock period

n_{rounds} : Number of rounds in the underlying block cipher

$n_{pipeline\_stages}$: Number of pipeline stages per round in the underlying block cipher

n_{cipher_calls} : Number of block cipher calls required for processing input data

$n_{allowed\_blocks}$: Number of blocks allowed to pass through the internal pipeline of the block cipher at a time

Table 7.1: Number of Rounds and Pipeline Stages per Round used in the Hardware Implementations of Block Ciphers

Cipher	n_{rounds}	$n_{pipeline\_stages}$	Max. Clock Frequency for ASIC Synthesis (90 nm)
AES-128	11	2	187.9 MHz
AES-192	13	2	187.9 MHz
AES-256	15	2	187.9 MHz
Twofish	16	8	157.5 MHz
Serpent	16	2	142.5 MHz

The general formula for throughput computation is as follows –

$$\text{Throughput} = \frac{\text{Size of the message (in bits)}}{\text{Time required to compute the output}}$$

$$\therefore \text{Throughput} = \frac{128 \times L}{n_{cipher\_calls} \times n_{rounds} \times n_{pipeline\_stages} \times t_p} \times n_{allowed\_blocks}$$

Data about the number of rounds and pipeline stages per round in the hardware implementations of the block ciphers is given in Table 7.1. Please note that initial rounds are also considered as a complete round in case of AES.

In order to find the throughput values for individual modes of operation, it is necessary to find the number of block cipher calls required by each mode of operation to compute the output for a message consisting of L blocks. Also, in order to find the maximum throughput supported by each mode, it is necessary to determine the value of $n_{allowed\_blocks}$. All throughput values given in the tables are for the basic case i.e. assuming that only a single block is allowed to be present in the block cipher pipeline at any time ($n_{allowed\_blocks} = 1$).

7.1.1 ECB Mode of Operation

ECB mode of operation is the only mode considered here, that has a static throughput for given parameters. This condition arises from the fact that processing a message using ECB mode does not require any additional block cipher calls for pre-processing or post-processing the message.

The number of block cipher calls required for ECB mode of operation is L.

Hence, the throughput for ECB mode of operation can be written as –

$\text{Throughput}_{ECB} = \frac{128 \times L}{L \times n_{rounds} \times n_{pipeline\_stages} \times t_p} \times n_{allowed\_blocks}$

7.1.2 OCB Mode of Operation

Throughput of OCB mode of operation is determined by the number of block cipher calls and the clock frequency at which the wrapper can work. For a message of size L blocks, OCB mode of operation requires $L + 3$ block cipher calls. Two of these block cipher calls are for pre-processing OCB parameters and one is for processing the checksum. The dependence of the throughput value on L is pronounced for small values of L i.e. short messages. As L increases, the throughput saturates at a particular value. Throughput values shown in the tables presented in this chapter are for large messages.

Hence, the throughput for OCB mode of operation can be written as –

$\text{Throughput}_{OCB} = \frac{128 \times L}{n_{cipher\_calls} \times n_{rounds} \times n_{pipeline\_stages} \times t_p} \times n_{allowed\_blocks}$

$\therefore \text{Throughput}_{OCB} = \frac{128 \times L}{(L+3) \times n_{rounds} \times n_{pipeline\_stages} \times t_p} \times 1$

$\therefore \text{Throughput}_{OCB} \approx \frac{128}{n_{rounds} \times n_{pipeline\_stages} \times t_p}$ (... for long messages)

7.1.3 CCM Mode of Operation

For computing the throughput of CCM mode of operation, additional parameters need to be defined. Assuming, that L is the length of the message, and A is the length of additional associated data in terms of blocks, the number of block cipher calls required for CCM mode are $(L + 1) + (A + L) + 1$. Typically, the length of additional associated data, A would be very small as compared to L, i.e. $A \ll L$. Therefore, $n_{cipher\_calls} \approx 2L + 2$.

Hence, the throughput for CCM mode of operation can be written as –

$$\text{Throughput}_{CCM} = \frac{128\times L}{n_{cipher\_calls}\times n_{rounds}\times n_{pipeline\_stages}\times t_p} \times n_{allowed\_blocks}$$

$$\therefore \text{Throughput}_{CCM} = \frac{128\times L}{(2L+2)\times n_{rounds}\times n_{pipeline\_stages}\times t_p} \times 1$$

$$\therefore \text{Throughput}_{CCM} = \frac{128}{2\times n_{rounds}\times n_{pipeline\_stages}\times t_p} \text{ (\ldots for long messages)}$$

7.1.4 EAX Mode of Operation

For EAX mode of operation, we need to define two more parameters – H (Number of blocks of Header) and N (Number of blocks of Nonce). Header is similar to additional associated data, A in CCM mode of operation. For EAX, $n_{cipher\_calls} = 2L+(H+1)+(N+1)$. Assuming, $H, \ll L$, we can approximate, $n_{cipher\_calls} = 2L+N+2$. Typically, $N = 1$; therefore, $n_{cipher\_calls} = 2L + 3$

Hence, the throughput for EAX mode of operation can be written as –

$$\text{Throughput}_{EAX} = \frac{128\times L}{n_{cipher\_calls}\times n_{rounds}\times n_{pipeline\_stages}\times t_p} \times n_{allowed\_blocks}$$

$$\therefore \text{Throughput}_{EAX} = \frac{128\times L}{(2L+3)\times n_{rounds}\times n_{pipeline\_stages}\times t_p} \times 1$$

$$\therefore \text{Throughput}_{EAX} = \frac{128}{2\times n_{rounds}\times n_{pipeline\_stages}\times t_p} \times 1 \text{ (\ldots for long messages)}$$

7.1.5 Generic Composition Schemes - AES + HMAC

Throughput computation is based on sightly different parameters in case of generic composition schemes compared to authenticated - encryption modes of operation. Generic composition schemes considered here use AES for encryption and HMAC for authentication. HMAC have been used with SHA-1 and SHA-512 as underlying hash functions.

In the design of generic composition schemes, AES and HMAC are separate units and have their own critical paths. For comparative purposes, they have been grouped together as a single entity with all components inside the entity clocked with the same master clock. Hence, the clock period of the design is taken as the larger clock period among AES and HMAC. If HMAC has a greater clock period, it determines the clock frequency at which the generic composition scheme can run. This in turn also determines the throughput of the generic composition scheme.

Let us introduce additional parameters for deriving the throughput in case of HMAC. Block size of underlying hash function ($n_{hash\_block\_size}$) is the number of bits that the hash function can process at a time. SHA-1 has a block size of 512 bits, while SHA-512 has a block size of 1024 bits. Both SHA-1 and SHA-512 require 80 rounds for computing the output.

For n blocks of data, HMAC requires $n+1$ hash function calls. Hence, throughput for HMAC can be formulated as –

Throughput$_{HMAC} = \frac{n_{hash\_block\_size} \times n}{(n+1) \times n_{rounds} \times t_p}$

For HMAC SHA-1, Throughput$_{HMAC\ SHA-1} = \frac{512 \times n}{(n+1) \times 80 \times t_p}$

$\therefore$ Throughput$_{HMAC\ SHA-1} = \frac{512}{80 \times t_p}$ (. . . for long messages)

For HMAC SHA-512, Throughput$_{HMAC\ SHA-512} = \frac{1024 \times n}{(n+1) \times 80 \times t_p}$

$\therefore$ Throughput$_{HMAC\ SHA-512} = \frac{1024}{80 \times t_p}$ (. . . for long messages)

7.2 FPGA Implementation Results

Devices from the Xilinx Virtex-4 family of FPGAs were targeted for all implementations. In particular, part number '`xc4vlx60ff1148`' with a speed grade of '`-11`' was used for all implementations [17]. Results are provided for all modes of operations implemented using a particular block cipher. Authenticated-Encryption modes are compared with Electronic Code Book (ECB) mode of operation since ECB is the basic block cipher implementation without any additions or feedback mechanism. Also, results for generic composition schemes comprising of HMAC for authentication and AES for encryption are provided.

7.2.1 Implementation of Modes with AES

FPGA Implementation results for all modes of operations with AES are shown in Table 7.2. In case of AES, the number of rounds is different for different key sizes. Therefore, there are different values for throughput for 128, 192 and 256 bit keys. Number of rounds (including the initial rounds) for AES-128, AES-192 and AES-26 are 11, 13 and 15 respectively. Hardware implementation of AES has two pipeline stages per round. It is worthwhile to reiterate that the underlying block ciphers have been designed to handle only a single data block at a time.

Table 7.2: FPGA Implementation Results for Modes of Operation with AES

	Mode of Operation			
	ECB	OCB	CCM	EAX
CLB Slices	1,188	2,964	2,799	2,993
LUTs	2,072	3,523	3,308	3,128
Flip-Flops	881	3,100	2,830	2,775
BlockRAMs	14	18	14	14
Min Clock Period (ns)	8.2	9.6	11.4	10.1
Max Clock Freq. (MHz)	121.7	103.4	87.5	99.2
Throughput (Mbps)-AES 128	708	601	255	287
Throughput (Mbps)-AES 192	599	508	215	243
Throughput (Mbps)-AES 256	519	440	186	211

Table 7.3: Comparison of FPGA Implementation Results - AES-ECB and AES-OCB

	Mode		Increase	% Increase
	ECB	OCB		
CLB Slices	1,188	2,964	1,776	149.5
LUTs	2,072	3,523	1,451	70
Flip-Flops	881	3,100	2,219	251.8
BlockRAMs	14	18	4	28.6
Min Clock Period (ns)	8.2	9.6	1.4	17.7
Max Clock Freq. (MHz)	121.7	103.4	-18.3	-15
Throughput (Mbps)-AES 128	708	601	-107	-15.1
Throughput (Mbps)-AES 192	599	508	-91	-15.2
Throughput (Mbps)-AES 256	519	440	-79	-15.2

Implementation results for all modes of operation are compared with the basic block cipher functionality i.e. the ECB mode of operation. Table 7.3 shows the increase and percentage increase in the amount of resources and timing values for OCB mode of operation as compared to ECB mode. Negative values indicate a decrease in the corresponding parameter value. Similar comparisons of AES-ECB with AES-CCM and AES-EAX are shown in Table 7.4 and Table 7.5 respectively.

Table 7.4: Comparison of FPGA Implementation Results - AES-ECB and AES-CCM

	Mode		Increase	% Increase
	ECB	CCM		
CLB Slices	1,188	2,799	1,611	135.6
LUTs	2,072	3,308	1,236	59.7
Flip-Flops	881	2,830	1,949	221.2
BlockRAMs	14	14	0	0
Min Clock Period (ns)	8.2	11.4	3.2	39
Max Clock Freq. (MHz)	121.7	87.5	-34.1	-28.06
Throughput (Mbps)-AES 128	708	255	-453	-64
Throughput (Mbps)-AES 192	599	215	-384	-64.1
Throughput (Mbps)-AES 256	519	186	-333	-64.2

Table 7.5: Comparison of FPGA Implementation Results - AES-ECB and AES-EAX

	Mode		Increase	% Increase
	ECB	EAX		
CLB Slices	1,188	2,993	1,805	151.9
LUTs	2,072	3,128	1,056	51
Flip-Flops	881	2,775	1,894	215
BlockRAMs	14	14	0	0
Min Clock Period (ns)	8.2	10.1	1.8	22.7
Max Clock Freq. (MHz)	121.7	99.2	-22.5	-18.5
Throughput (Mbps)-AES 128	708	287	-421	-59.5
Throughput (Mbps)-AES 192	599	243	-356	-59.4
Throughput (Mbps)-AES 256	519	211	-308	-59.3

Table 7.6: FPGA Implementation Results for Modes of Operation with Twofish

	Mode of Operation			
	ECB	OCB	CCM	EAX
CLB Slices	3,269	5,213	5,042	5,263
LUTs	4,444	6,224	6,091	5,869
Flip-Flops	1,896	4,065	3,795	3,962
BlockRAMs	2	6	2	2
Min Clock Period (ns)	10	13.2	13.2	13.4
Max Clock Freq. (MHz)	100	75.5	75.8	74.6
Throughput (Mbps)	100	75	38	38

Table 7.7: Comparison of FPGA Implementation Results - Twofish-ECB and Twofish-OCB

	Mode		Increase	% Increase
	ECB	OCB		
CLB Slices	3,269	5,213	1,944	59.4
LUTs	4,444	6,224	1,780	40
Flip-Flops	1,896	4,065	2,169	114.4
BlockRAMs	2	6	4	200
Min Clock Period (ns)	10	13.2	3.2	32.5
Max Clock Freq. (MHz)	100	75.5	-24.5	-24.5
Throughput (Mbps)	100	75	-25	-25

7.2.2 Implementation of Modes with Twofish

FPGA implementation results for all modes of operation with Twofish as the underlying block cipher are shown in Table 7.6. Twofish consists of 16 rounds and the hardware implementation of Twofish consists of 8 pipeline stages per round. The throughput results shown in Table 7.6 assume that only a single block is allowed to pass through the pipeline at a time.

Table 7.8: Comparison of FPGA Implementation Results - Twofish-ECB and Twofish-CCM

	Mode		Increase	% Increase
	ECB	CCM		
CLB Slices	3,269	5,042	1,773	54.2
LUTs	4,444	6,091	1,647	37.1
Flip-Flops	1,896	3,795	1,899	100.1
BlockRAMs	2	2	0	0
Min Clock Period (ns)	10	13.2	3.2	32
Max Clock Freq. (MHz)	100	75.8	-24.2	-24.2
Throughput (Mbps)	100	38	-62	-63

Table 7.9: Comparison of FPGA Implementation Results - Twofish-ECB and Twofish-EAX

	Mode		Increase	% Increase
	ECB	EAX		
CLB Slices	3,269	5,263	1,773	61
LUTs	4,444	5,869	1,647	32.1
Flip-Flops	1,896	3,962	1,899	109
BlockRAMs	2	2	0	0
Min Clock Period (ns)	10	13.4	3.2	34.1
Max Clock Freq. (MHz)	100	74.6	-24.2	-25.4
Throughput (Mbps)	100	38	-62	-62

Table 7.10: FPGA Implementation Results for Modes of Operation with Serpent

	Mode of Operation			
	ECB	OCB	CCM	EAX
CLB Slices	15,849	17,737	17,129	17,265
LUTs	13,484	15,185	15,080	18,879
Flip-Flops	4,500	6,703	6,466	10,358
BlockRAMs	0	4	0	0
Min Clock Period (ns)	13.2	15	14.9	14.4
Max Clock Freq. (MHz)	75.8	66.7	67.2	69.4
Throughput (Mbps)	303	266	134	139

7.2.3 Implementation of Modes with Serpent

FPGA Implementation results for all modes of operation with Serpent as the underlying block cipher are shown in Table 7.10. Implementation of Serpent requires more resources as compared to AES and Twofish. This is because of the large number of round keys required during cipher rounds. All the round keys for encryption as well as decryption are stored in LUT-based memory on the FPGA. The memory for storing keys is the primary reason for the large implementation size. As a result of its inherently large size, the percentage increase in area for implementing the mode of operation wrapper is much smaller. Comparison of Serpent implementation in ECB mode of operation is compared with OCB, CCM and EAX modes in Table 7.11, Table 7.12 and Table 7.13 respectively.

7.2.4 Implementation of Generic Composition Schemes

In this section, implementation results for generic composition schemes combining HMAC for authentication and AES for encryption are provided. HMAC has been implemented with two underlying hash functions, SHA-1 and SHA-512. Table 7.14

Table 7.11: Comparison of FPGA Implementation Results - Serpent-ECB and Serpent-OCB

	Mode		Increase	% Increase
	ECB	OCB		
CLB Slices	15,849	17,737	1,888	11.9
LUTs	13,484	15,185	1,701	12.6
Flip-Flops	4,500	6,703	2,203	48.9
BlockRAMs	0	4	4	0
Min Clock Period (ns)	13.2	15	1.79	13.6
Max Clock Freq. (MHz)	75.8	66.7	-9.1	-11.9
Throughput (Mbps)	303	266	-37	-12.2

Table 7.12: Comparison of FPGA Implementation Results - Serpent-ECB and Serpent-CCM

	Mode		Increase	% Increase
	ECB	CCM		
CLB Slices	15,849	17,129	1,280	8.1
LUTs	13,484	15,080	1,596	11.8
Flip-Flops	4,500	6,466	1,966	43.7
BlockRAMs	0	0	0	0
Min Clock Period (ns)	13.2	14.9	1.67	12.6
Max Clock Freq. (MHz)	75.8	67.2	-8.5	-11.2
Throughput (Mbps)	303	134	-169	-55.8

Table 7.13: Comparison of FPGA Implementation Results - Serpent-ECB and Serpent-EAX

	Mode		Increase	% Increase
	ECB	EAX		
CLB Slices	15,849	17,265	1,416	8.9
LUTs	13,484	18,879	5,395	40
Flip-Flops	4,500	10,358	5,858	130.2
BlockRAMs	0	0	0	0
Min Clock Period (ns)	13.2	14.4	1.2	9.1
Max Clock Freq. (MHz)	75.8	69.4	-6.3	-8.3
Throughput (Mbps)	303	139	-164	-54.1

Table 7.14: FPGA Implementation Results for Generic Composition Schemes

	Generic Composition Scheme	
	AES + HMAC SHA-1	AES + HMAC SHA-512
CLB Slices	3,076	4,246
LUTs	4,367	6,411
Flip-Flops	3,092	4,635
BlockRAMs	16	16
Min Clock Period (ns)	11.8	14.6
Max Clock Freq. (MHz)	84.8	68.3
Throughput (Mbps)	542	708

shows results for two generic composition schemes. The results for these individual schemes are compared with the basic block cipher i.e. AES-ECB in Table 7.15 and Table 7.16.

As mentioned earlier, AES as well as HMAC are clocked using the same master clock. Also, the throughput of the combined system is the smaller of the individual throughput values. For a combined scheme using AES and HMAC SHA-1, the individual throughput values are 708 Mbps for AES and 542 Mbps for HMAC SHA-1. Hence, the throughput of the combined scheme is the lesser of the two values, i.e 542 Mbps. On the other hand, for the scheme using AES and HMAC SHA-512, the limit on throughput is set by the throughput of AES. The throughput of HMAC SHA-512 is 874 Mbps while the throughput of AES is only 708 Mbps. Hence, the resultant throughput of the combined scheme is 708 Mbps.

7.3 ASIC Synthesis Results

ASIC synthesis was performed targeting both 90 nm and 130 nm technologies using Synopsys Design Compiler. TSMC libraries were used for synthesis purposes. Implementation area as well as timing results are presented for both process technologies.

Table 7.15: Comparison of FPGA Implementation Results - AES-ECB and AES+HMAC SHA-1

	Scheme		Increase	% Increase
	ECB	AES+HMAC SHA-1		
CLB Slices	1,188	3,076	1,888	158.9
LUTs	2,072	4,367	2,295	110.8
Flip-Flops	881	3,092	2,211	250.9
BlockRAMs	14	16	2	14.3
Min Clock Period (ns)	8.2	11.8	3.6	43.6
Max Clock Freq. (MHz)	121.7	84.8	-37	-30.4
Throughput (Mbps)	708	542	-166	-23.5

Table 7.16: Comparison of FPGA Implementation Results - AES-ECB and AES+HMAC SHA-512

	Scheme		Increase	% Increase
	ECB	AES+HMAC SHA-512		
CLB Slices	1,188	4,246	3,058	8.9
LUTs	2,072	6,411	4,339	40
Flip-Flops	881	4,635	3,754	130.2
BlockRAMs	14	16	2	0
Min Clock Period (ns)	8.21	14.6	6.43	9.1
Max Clock Freq. (MHz)	121.7	68.3	-53.4	-8.3
Throughput (Mbps)	708	708	0	0

Table 7.17: ASIC Synthesis Results for Modes of Operation with AES (90 nm)

	Modes of Operation			
	ECB	OCB	CCM	EAX
Combinational Area	22,177	34,596	30,604	33,709
Non-Combinational Area	19,072	28,989	25,937	29,180
Total Cell Area	41,250	63,585	56,541	62,889
Min Clock Period (ns)	5.3	6.8	6.7	6.9
Max Clock Freq (MHz)	188	146.4	148.6	145
Throughput (Mbps)-AES 128	1,097	854	434	421
Throughput (Mbps)-AES 192	928	723	367	356
Throughput (Mbps)-AES 256	805	626	318	309

The unit for expressing area is a square micron.

7.3.1 Implementation of Modes with AES

Synthesis results for all modes of operation with AES as the underlying block cipher are presented. Results for 90 nm and 130 nm technologies are tabulated in different tables. Combined results for all modes synthesized targeting 90 nm and 130 nm technologies are shown in Table 7.17 and Table 7.21 respectively. Also, results for all modes of operation are compared with the basic block cipher functionality i.e. ECB mode of operation.

Throughput computations presented in Section 7.1 are used to compute the throughput values tabulated in the tables. These throughput values are for large messages, thereby eliminating the influence of extra pre-processing and post-processing on the throughput values.

Table 7.18: Comparison of ASIC Synthesis Results - AES-ECB and AES-OCB (90 nm)

	Mode		Increase	% Increase
	ECB	OCB		
Combinational Area	22,177	34,596	12,418	56
Non-Combinational Area	19,072	28,989	9,917	52
Total Cell Area	41,250	63,585	22,335	54.1
Min Clock Period (ns)	5.3	6.8	1.5	28.4
Max Clock Freq (MHz)	188	146.4	-41.5	-22.1
Throughput (Mbps)-AES 128	1,097	854	-243	-22.1
Throughput (Mbps)-AES 192	928	723	-205	-22
Throughput (Mbps)-AES 256	805	626	-179	-22.2

Table 7.19: Comparison of ASIC Synthesis Results - AES-ECB and AES-CCM (90 nm)

	Mode		Increase	% Increase
	ECB	CCM		
Combinational Area	22,177	30,604	8,426	38
Non-Combinational Area	19,072	25,937	6,865	36
Total Cell Area	41,250	56,541	15,291	37.1
Min Clock Period (ns)	5.3	6.7	1.4	26.5
Max Clock Freq (MHz)	188	148.6	-39.4	-21
Throughput (Mbps)-AES 128	1,097	434	-663	-60.4
Throughput (Mbps)-AES 192	928	367	-561	-60.4
Throughput (Mbps)-AES 256	805	318	-487	-60.5

Table 7.20: Comparison of ASIC Synthesis Results - AES-ECB and AES-EAX (90 nm)

	Mode		Increase	% Increase
	ECB	EAX		
Combinational Area	22,177	33,709	11,531	52
Non-Combinational Area	19,072	29,180	10,108	53
Total Cell Area	41,250	62,889	21,639	52.5
Min Clock Period (ns)	5.3	6.9	1.6	29.7
Max Clock Freq (MHz)	188	145	-43	-23
Throughput (Mbps)-AES 128	1,097	421	-676	-61.6
Throughput (Mbps)-AES 192	928	356	-572	-61.6
Throughput (Mbps)-AES 256	805	309	-496	-61.6

Table 7.21: ASIC Synthesis Results for Modes of Operation with AES (130 nm)

	Modes of Operation			
	ECB	OCB	CCM	EAX
Combinational Area	42,095	61,039	58,937	59,774
Non-Combinational Area	37,198	53,193	52,449	53,193
Total Cell Area	79,293	114,232	111,386	112,968
Min Clock Period (ns)	7.6	8.1	8.6	8.3
Max Clock Freq (MHz)	131.6	123.6	116	120
Throughput (Mbps)-AES 128	765	717	338	350
Throughput (Mbps)-AES 192	647	607	286	296
Throughput (Mbps)-AES 256	561	526	247	256

Table 7.22: Comparison of ASIC Synthesis Results - AES-ECB and AES-OCB (130 nm)

	Mode		Increase	% Increase
	ECB	OCB		
Combinational Area	42,095	61,039	18,943	45
Non-Combinational Area	37,198	53,193	15,994	43
Total Cell Area	79,293	114,232	34,938	44
Min Clock Period (ns)	7.6	8.1	0.5	6.5
Max Clock Freq (MHz)	131.6	123.6	-8	-6
Throughput (Mbps)-AES 128	765	717	-48	-6.2
Throughput (Mbps)-AES 192	647	607	-40	-6.2
Throughput (Mbps)-AES 256	561	526	-35	-6.2

Table 7.23: Comparison of ASIC Synthesis Results - AES-ECB and AES-CCM (130 nm)

	Mode		Increase	% Increase
	ECB	CCM		
Combinational Area	42,095	58,937	16,842	40
Non-Combinational Area	37,198	52,449	15,250	41
Total Cell Area	79,293	111,386	32,092	40.1
Min Clock Period (ns)	7.6	8.6	1	13.4
Max Clock Freq (MHz)	131.6	116	-15.6	-11
Throughput (Mbps)-AES 128	765	338	-427	-55.8
Throughput (Mbps)-AES 192	647	286	-361	-55.8
Throughput (Mbps)-AES 256	561	247	-314	-56

Table 7.24: Comparison of ASIC Synthesis Results - AES-ECB and AES-EAX (130 nm)

	Mode		Increase	% Increase
	ECB	EAX		
Combinational Area	42,095	59,774	17,679	42
Non-Combinational Area	37,198	53,193	15,994	43
Total Cell Area	79,293	112,968	33,674	42.5
Min Clock Period (ns)	7.6	8.3	0.7	9.5
Max Clock Freq (MHz)	131.6	120	-11.4	-8.7
Throughput (Mbps)-AES 128	765	350	-415	-54.3
Throughput (Mbps)-AES 192	647	296	-351	-54.3
Throughput (Mbps)-AES 256	561	256	-305	-54.4

Table 7.25: ASIC Synthesis Results for Modes of Operation with Twofish (90 nm)

	Modes of Operation			
	ECB	OCB	CCM	EAX
Combinational Area	66,856	79,252	76,786	79,388
Non-Combinational Area	62,411	72,367	69,689	72,534
Total Cell Area	129,268	151,620	146,476	151,923
Min Clock Period (ns)	6.4	7.4	7.4	7.5
Max Clock Freq (MHz)	157	135.1	135.1	133
Throughput (Mbps)	156	135	67	66

7.3.2 Implementation of Modes with Twofish

Combined synthesis results for all modes of operations with Twofish as the underlying block cipher are shown in Table 7.25 and Table 7.29 for 90 nm and 130 nm technologies respectively. As can be seen from the results, throughput is comparatively low for all modes of operation with Twofish. This is because of a comparatively greater number of pipeline stages in Twofish. Twofish has been designed with eight pipeline stages per round and since the pipeline is not fully utilized, it reduces the throughput by a factor of 8 as compared to a fully utilized pipeline. Sub-optimal number of pipeline stages were used in Twofish in order to meet a clock frequency of 100 MHz. Twofish has a complex round and hence up to 4 pipeline stages are not sufficient to make it run at the desired frequency.

7.3.3 Implementation of Modes with Serpent

Synthesis results for all modes of operation with Serpent are shown in Table 7.33 and Table 7.37 for 90 nm and 130 nm technology respectively. Serpent is implemented with a two pipeline stages per round and hence the throughput compares favorably to other ciphers. On the other hand, Serpent implementation requires a significantly

Table 7.26: Comparison of ASIC Synthesis Results - Twofish-ECB and Twofish-OCB (90 nm)

	Mode		Increase	% Increase
	ECB	OCB		
Combinational Area	66,856	79,252	12,395	18.5
Non-Combinational Area	62,411	72,367	9,955	16
Total Cell Area	129,268	151,620	22,351	17.3
Min Clock Period (ns)	6.4	7.4	1	16.5
Max Clock Freq (MHz)	157	135.1	-21.9	14.2
Throughput (Mbps)	156	135	-21	13.5

Table 7.27: Comparison of ASIC Synthesis Results - Twofish-ECB and Twofish-CCM (90 nm)

	Mode		Increase	% Increase
	ECB	CCM		
Combinational Area	66,856	76,786	9,930	14.9
Non-Combinational Area	62,411	69,689	7,277	11.7
Total Cell Area	129,268	146,476	17,207	13.3
Min Clock Period (ns)	6.4	7.4	1	16.5
Max Clock Freq (MHz)	157	135.1	-22.3	-14.2
Throughput (Mbps)	156	67	-89	-57.1

Table 7.28: Comparison of ASIC Synthesis Results - Twofish-ECB and Twofish-EAX (90 nm)

	Mode		Increase	% Increase
	ECB	EAX		
Combinational Area	66,856	79,388	12,531	18.7
Non-Combinational Area	62,411	72,534	10,122	16.2
Total Cell Area	129,268	151,923	22,654	17.5
Min Clock Period (ns)	6.4	7.5	1.2	18.4
Max Clock Freq (MHz)	157	133	-24.5	-15.6
Throughput (Mbps)	156	66	-90	-57.7

Table 7.29: ASIC Synthesis Results for Modes of Operation with Twofish (130 nm)

	Modes of Operation			
	ECB	OCB	CCM	EAX
Combinational Area	128,729	147,756	146,685	147,361
Non-Combinational Area	121,697	137,753	137,059	137,690
Total Cell Area	250,426	285,509	283,745	285,051
Min Clock Period (ns)	7.2	7.9	7.9	8.1
Max Clock Freq (MHz)	138.7	126.1	126.1	123.5
Throughput (Mbps)	138	126	63	61

Table 7.30: Comparison of ASIC Synthesis Results - Twofish-ECB and Twofish-OCB (130 nm)

	Mode		Increase	% Increase
	ECB	OCB		
Combinational Area	128,729	147,756	19,027	14.8
Non-Combinational Area	121,697	137,753	16,055	13.2
Total Cell Area	250,426	285,509	35,082	14
Min Clock Period (ns)	7.2	7.9	0.7	10
Max Clock Freq (MHz)	138.7	126.1	-12.6	-9.1
Throughput (Mbps)	138	126	-12	-8.7

Table 7.31: Comparison of ASIC Synthesis Results - Twofish-ECB and Twofish-CCM (130 nm)

	Mode		Increase	% Increase
	ECB	CCM		
Combinational Area	128,729	146,685	17,955	13.9
Non-Combinational Area	121,697	137,059	15,362	12.6
Total Cell Area	250,426	283,745	33,318	13.3
Min Clock Period (ns)	7.2	7.9	0.7	9.4
Max Clock Freq (MHz)	138.7	126.7	-12	-8.6
Throughput (Mbps)	138	63	-75	-54.3

Table 7.32: Comparison of ASIC Synthesis Results - Twofish-ECB and Twofish-EAX (130 nm)

	Mode		Increase	% Increase
	ECB	EAX		
Combinational Area	128,729	147,361	18,632	14.5
Non-Combinational Area	121,697	137,690	15,992	13.1
Total Cell Area	250,426	285,051	34,624	13.8
Min Clock Period (ns)	7.2	8.1	0.9	12.3
Max Clock Freq (MHz)	138.7	123.5	-15.2	-11
Throughput (Mbps)	138	61	-77	-55.8

Table 7.33: ASIC Synthesis Results for Modes of Operation with Serpent (90 nm)

	Modes of Operation			
	ECB	OCB	CCM	EAX
Combinational Area	197,583	210,015	207,789	208,665
Non-Combinational Area	185,056	196,059	192,427	197,192
Total Cell Area	382,640	406,075	400,217	405,858
Min Clock Period (ns)	7	8.2	8.3	8.3
Max Clock Freq (MHz)	142.5	121.7	120.5	121.1
Throughput (Mbps)	569	486	240	242

larger area than other ciphers because of storage necessary to store a large number of round keys.

7.3.4 Implementation of Generic Composition Schemes - AES + HMAC

Results for ASIC synthesis of generic composition schemes using AES for encryption and HMAC for authentication are presented. Two versions of HMACs are considered, one with SHA-1 as the underlying hash function and the other with SHA-512. Results for 90 nm and 130 nm technologies are shown in Table 7.41 and Table 7.44 respectively.

Throughput supported by the combined scheme is restricted by the smaller of

Table 7.34: Comparison of ASIC Synthesis Results - Serpent-ECB and Serpent-OCB (90 nm)

	Mode		Increase	% Increase
	ECB	OCB		
Combinational Area	197,583	210,015	12,431	6.3
Non-Combinational Area	185,056	196,059	11,002	5.9
Total Cell Area	382,640	406,075	23,434	6.1
Min Clock Period (ns)	7	8.2	1.2	17.1
Max Clock Freq (MHz)	142.5	121.7	-20.8	-14.6
Throughput (Mbps)	569	486	-83	-14.6

Table 7.35: Comparison of ASIC Synthesis Results - Serpent-ECB and Serpent-CCM (90 nm)

	Mode		Increase	% Increase
	ECB	CCM		
Combinational Area	197,583	207,789	10,205	5.2
Non-Combinational Area	185,056	192,427	7,371	4
Total Cell Area	382,640	400,217	17,577	4.6
Min Clock Period (ns)	7	8.3	1.3	18.2
Max Clock Freq (MHz)	142.5	120.5	-22	-15.4
Throughput (Mbps)	569	240	-329	-57.8

Table 7.36: Comparison of ASIC Synthesis Results - Serpent-ECB and Serpent-EAX (90 nm)

	Mode		Increase	% Increase
	ECB	EAX		
Combinational Area	197,583	208,665	11,081	5.6
Non-Combinational Area	185,056	197,192	12,136	6.6
Total Cell Area	382,640	405,858	23,217	6.1
Min Clock Period (ns)	7	8.3	1.2	17.7
Max Clock Freq (MHz)	142.5	121.1	-21.4	-15
Throughput (Mbps)	569	242	-327	-57.5

Table 7.37: ASIC Synthesis Results for Modes of Operation with Serpent (130 nm)

	Modes of Operation			
	ECB	OCB	CCM	EAX
Combinational Area	376,877	396,209	394,809	393,518
Non-Combinational Area	336,416	352,664	354,786	354,172
Total Cell Area	713,293	748,874	749,595	747,691
Min Clock Period (ns)	7.7	8.9	9	8.9
Max Clock Freq (MHz)	130.7	112.2	111.4	112.4
Throughput (Mbps)	522	448	222	224

Table 7.38: Comparison of ASIC Synthesis Results - Serpent-ECB and Serpent-OCB (130 nm)

	Mode		Increase	% Increase
	ECB	OCB		
Combinational Area	376,877	396,209	19,332	5.1
Non-Combinational Area	336,416	352,664	16,247	4.8
Total Cell Area	713,293	748,874	35,580	5
Min Clock Period (ns)	7.7	8.9	1.2	16.5
Max Clock Freq (MHz)	130.7	112.2	-18.5	-14.1
Throughput (Mbps)	522	448	-74	-14.2

Table 7.39: Comparison of ASIC Synthesis Results - Serpent-ECB and Serpent-CCM (130 nm)

	Mode		Increase	% Increase
	ECB	CCM		
Combinational Area	376,877	394,809	17,931	4.8
Non-Combinational Area	336,416	354,786	18,369	5.5
Total Cell Area	713,293	749,595	36,301	5.1
Min Clock Period (ns)	7.7	9	1.3	17.4
Max Clock Freq (MHz)	130.7	111.4	-19.4	-14.8
Throughput (Mbps)	522	222	-300	-57.5

Table 7.40: Comparison of ASIC Synthesis Results - Serpent-ECB and Serpent-EAX (130 nm)

	Mode		Increase	% Increase
	ECB	EAX		
Combinational Area	376,877	393,518	16,641	4.4
Non-Combinational Area	336,416	354,172	17,755	5.3
Total Cell Area	713,293	747,691	34,397	4.8
Min Clock Period (ns)	7.7	8.9	1.3	16.3
Max Clock Freq (MHz)	130.7	112.4	-18.4	-14
Throughput (Mbps)	522	224	-298	-57.1

the individual throughput values for AES and HMAC. In case of a combined scheme using AES and HMAC SHA-1, the throughput is restricted due to the throughput of HMAC SHA-1. HMAC SHA-1 supports a throughput of 842 Mbps, while AES supports almost 1.1 Gbps. On the other hand, for a combined scheme consisting of AES and HMAC SHA-512, the throughput is restricted by the throughput of AES. HMAC SHA-512 supports a throughput of almost 1.3 Gbps. Hence the throughput of the combined scheme is restricted to 1097 Mbps, which is the throughput of AES. The above resuts are for implementations targeting 90 nm technology. Similar techniques are used in order to determine the throughput for implementations targeting 130 nm technology. For 130 nm technology, the throughput both both HMAC SHA-1 (820 Mbps) and HMAC SHA-512 (1280 Mbps) is better than AES (765 Mbps).

Table 7.41: ASIC Synthesis Results for Generic Composition Schemes(90 nm)

	Generic Composition Scheme	
	AES + HMAC SHA-1	AES + HMAC SHA-512
Combinational Area	49,429	73,181
Non-Combinational Area	65,247	109,333
Total Cell Area	114,676	182,514
Min Clock Period (ns)	7.6	9.9
Max Clock Freq (MHz)	131.2	101.2
Throughput (Mbps)	842	1097

Table 7.42: Comparison of ASIC Synthesis Results - AES-ECB and AES+HMAC SHA-1(90 nm)

	Scheme		Increase	%Increase
	AES-ECB	AES + HMAC SHA-1		
Combinational Area	22,177	49,429	27,251	122.9
Non-Combinational Area	19,072	65,247	46,174	242.1
Total Cell Area	41,250	114,676	73,426	178
Min Clock Period (ns)	5.3	7.6	2.3	43.3
Max Clock Freq (MHz)	187.9	131.2	-56.7	-30.2
Throughput (Mbps)	1097	1097	0	0

Table 7.43: Comparison of ASIC Synthesis Results - AES-ECB and AES+HMAC SHA-512(90 nm)

	Scheme		Increase	%Increase
	AES-ECB	AES + HMAC SHA-512		
Combinational Area	22,177	73,181	51,003	230
Non-Combinational Area	19,072	109,333	90,260	473.3
Total Cell Area	41,250	182,514	141,264	375.5
Min Clock Period (ns)	5.3	9.9	4.6	85.7
Max Clock Freq (MHz)	187.9	101.2	-86.8	-46.2
Throughput (Mbps)	1097	1293	196	67.3

Table 7.44: ASIC Synthesis Results for Generic Composition Schemes(130 nm)

	Generic Composition Scheme	
	AES + HMAC SHA-1	AES + HMAC SHA-512
Combinational Area	88,345	151,238
Non-Combinational Area	140,579	225,827
Total Cell Area	228,924	377,065
Min Clock Period (ns)	7.8	10
Max Clock Freq (MHz)	128.2	100
Throughput (Mbps)	765	765

Table 7.45: Comparison of ASIC Synthesis Results - AES-ECB and AES+HMAC SHA-1(130 nm)

	Scheme		Increase	%Increase
	AES-ECB	AES + HMAC SHA-1		
Combinational Area	42,095	88,345	46,249	109.9
Non-Combinational Area	37,198	140,579	103,380	277.9
Total Cell Area	79,293	228,924	149,630	188.7
Min Clock Period (ns)	7.6	7.8	0.2	2.6
Max Clock Freq (MHz)	131.6	128.2	-3.4	-2.6
Throughput (Mbps)	765	765	0	0

Table 7.46: Comparison of ASIC Synthesis Results - AES-ECB and AES+HMAC SHA-512(130 nm)

	Scheme		Increase	%Increase
	AES-ECB	AES + HMAC SHA-512		
Combinational Area	42,095	151,238	109,142	259.3
Non-Combinational Area	37,198	225,827	188,628	507.1
Total Cell Area	79,293	377,065	297,771	375.5
Min Clock Period (ns)	7.6	10	2.4	31.3
Max Clock Freq (MHz)	131.6	100	-31.4	-23.9
Throughput (Mbps)	765	765	0	0

Chapter 8: Analysis of Results

Authentication schemes were implemented according to the design methodology presented in Chapter 2. Results for FPGA implementation and ASIC synthesis of the designs were tabulated in Chapter 7. The analysis of those results is presented here.

8.1 Analysis of Results of FPGA Implementations

8.1.1 Comparison of Authenticated-Encryption Modes of Operation with Generic Composition Schemes

In order to make a decision about the suitability of authenticated-encryption modes of operation versus generic composition schemes, it is necessary to compare and contrast the results obtained for these schemes. Graphical comparison is shown in Fig. 8.1. Comparison is based on authentication schemes using only AES-128. Both the generic composition schemes have significantly high throughput; 542 Mbps for AES + HMAC SHA-1 and 708 Mbps for AES + HMAC SHA-512. Throughput of OCB mode of operation is slightly less than generic composition schemes. CCM and EAX modes, being two-pass schemes support approximately half the throughput of a single-pass scheme like OCB mode of operation. Referring to the target throughput goal of $\sim$150 Mbps for the analyzed applications, mentioned in Chapter 1, we can see that all authentication options considered here easily meet the goals.

Since throughput requirements are satisfied, a selection would be done based upon

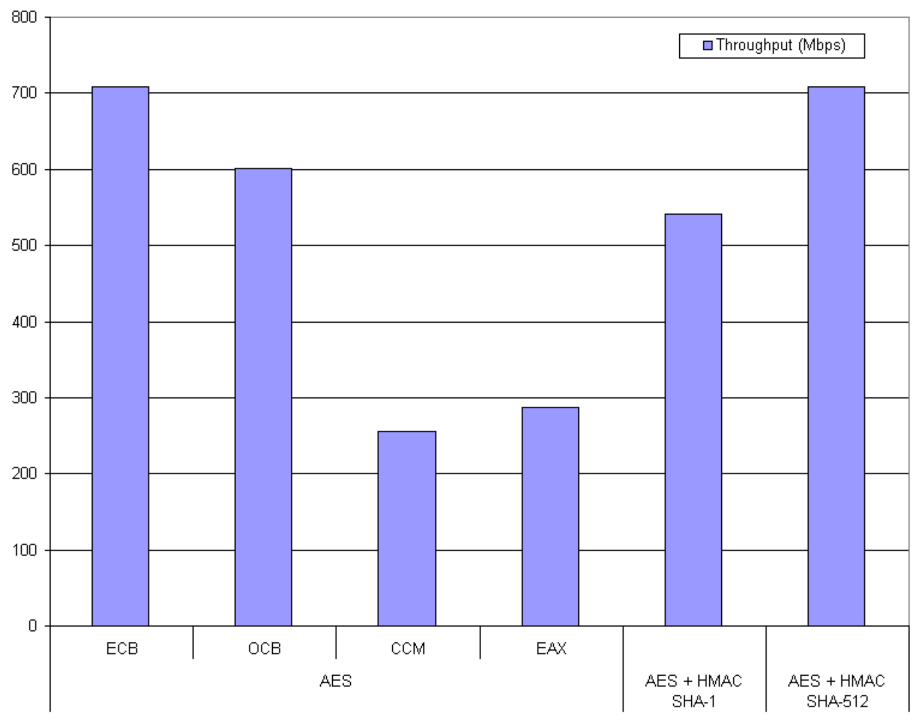


Figure 8.1: Throughput Comparison for FPGA Implementation – Authenticated-Encryption Modes of Operation and Generic Composition Schemes

the area required. For FPGA bitstream security application, the available area budget is quite small. Even for authentication applications in wireless networks, small area implementation of cryptographic algorithms would be preferred since this would leave more space on the FPGA for other logic components, or possibly allow migration to a smaller and cheaper FPGA device.

In order to compare resource utilization on an FPGA, some background about FPGA structure is introduced very briefly. Details about FPGA structure can be found in [17].

The basic logic block on an FPGA is known as Configurable Logic Block (CLB)

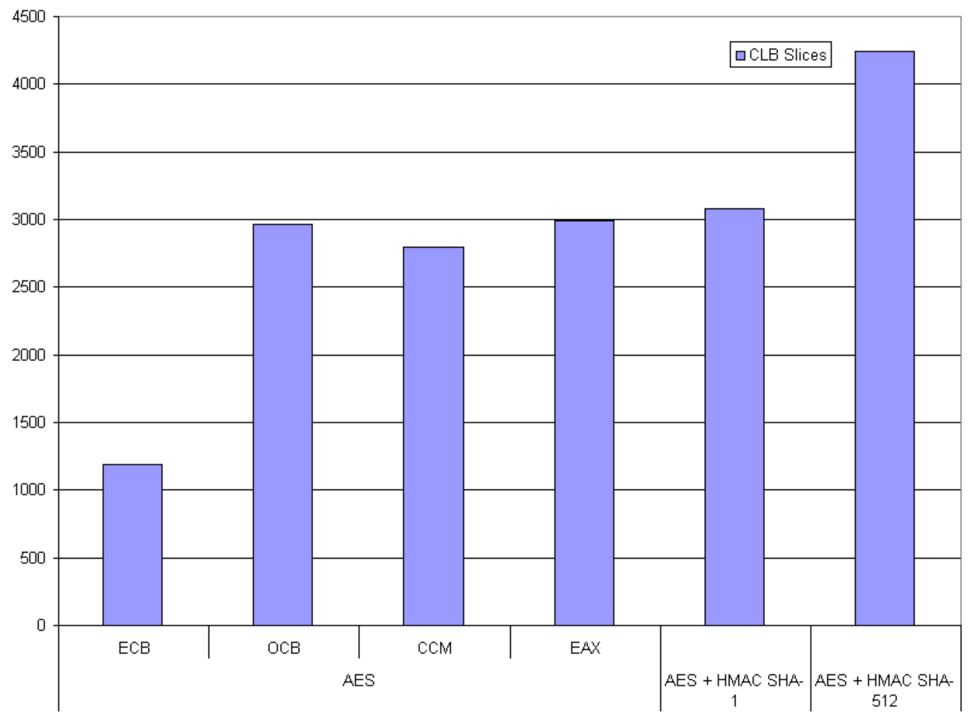


Figure 8.2: Comparison of Resource Utilization for FPGA Implementation – Authenticated-Encryption Modes of Operations and Generic Composition Schemes

Each CLB consists of two identical sub-blocks known as CLB Slices. Resource utilization on an FPGA is generally expressed in terms of CLB Slices. Comparison of Resource Utilization for AES-128 in all modes of operations with generic composition schemes is shown in Fig. 8.2. As can be seen, AES + HMAC SHA-1 utilizes almost the same amount of resources as authenticated-encryption schemes. The faster generic composition scheme, AES + HMAC SHA-512 is approximately 1.4 times larger than the largest authenticated-encryption scheme (4200 CLB Slices v/s 3000 CLB Slices). CCM uses the least amount of resources among all authenticated-encryption schemes and would be a feasible option to be used for both target applications.

8.1.2 Comparison of Modes of Operations Based on Different Ciphers

All implemented modes of operations are compared in Fig. 8.3 on the basis of the underlying block cipher used. It can be seen that throughput value for any mode of operation is actually set by the throughput supported by ECB mode of operation. Throughput in OCB mode approaches the throughput of ECB mode, whereas CCM and EAX modes support almost half the throughput of ECB mode.

If target throughput is known, then by looking at the throughput values for ECB mode of operation for a particular cipher, it is possible to make a rough estimation about the throughput supported by other modes of operation. For example, if the throughput requirement is 100 Mbps, and ECB mode of that block cipher provides a throughput of little over 100 Mbps, then it is evident that CCM and EAX modes of that cipher cannot be used in order to support the throughput requirement. OCB mode might be able to support the requirement, but that needs to be verified by actual analysis.

8.1.3 Comparison of Throughput/Area Ratio for FPGA Implementations

Throughput/Area ratio is an important comparison criteria. This is evident from the design goal that the implementation should support sufficient throughput and should have the least possible circuit area. Higher the Throughput/Area ratio, better the design.

For FPGA implementations, the area is expressed in terms of CLB Slices. Therefore, the unit of Throughput/Area ratio is {Mbps/CLB Slice}. A comparative graph

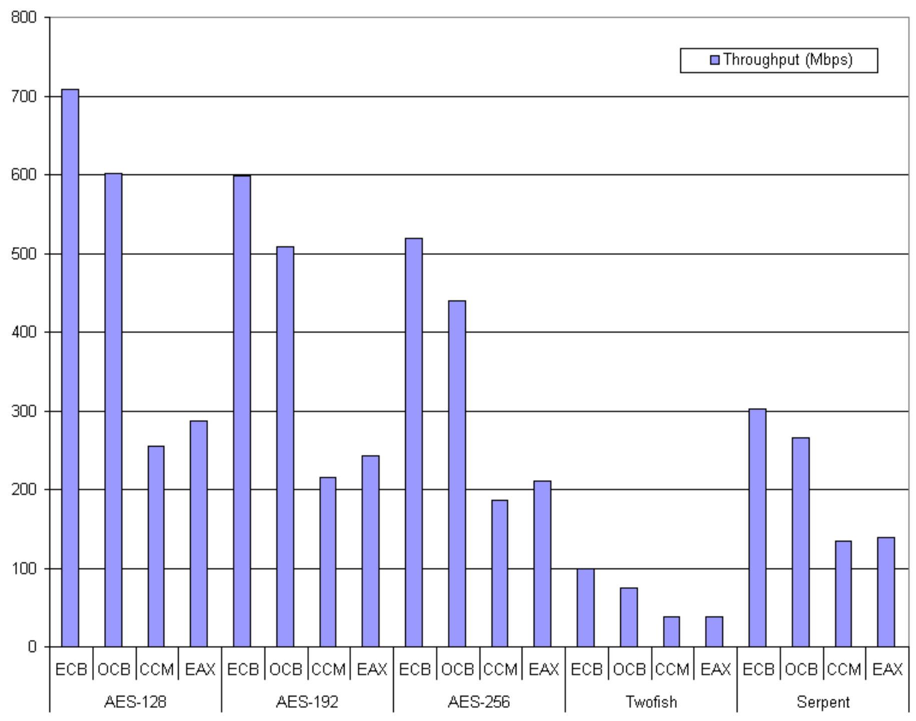


Figure 8.3: Throughput Comparison for FPGA Implementation – Authenticated-Encryption Modes of Operation based on Different Ciphers

is shown in Fig. 8.4. It is clearly evident that OCB is almost twice as good as CCM and EAX. This can be attributed to the single-pass and two-pass aspects of the schemes. Also, AES implementations have a better Throughput/Area ratio than Twofish and Serpent. Twofish suffers due to under-utilized pipeline stages, whereas Serpent has a drawback of occupying a relatively large circuit area.

Surprisingly, generic composition schemes compare favorably against almost all modes of operations, even against AES modes. This can be attributed to a significantly large number of Block RAMs (16) used in the implementation of generic composition schemes. When Throughput/Area ratio is computed, only area occupied

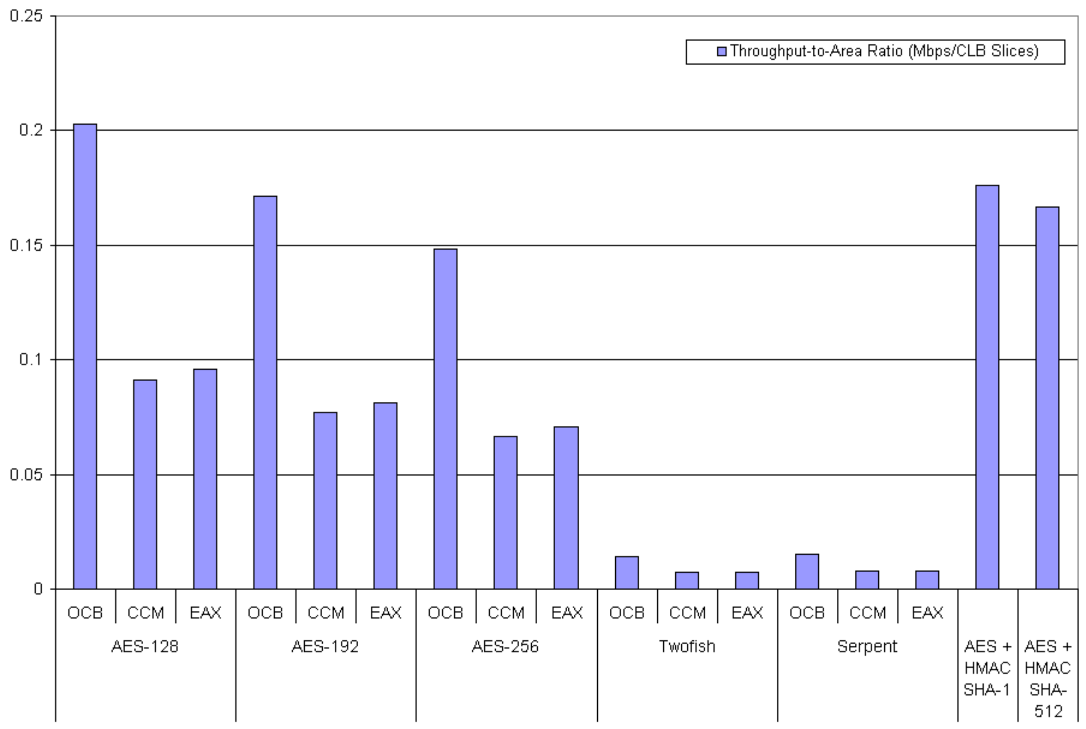


Figure 8.4: Throughput/Area Ratio for FPGA Implementation of Authentication Schemes

by CLB Slices is considered. Block RAMs are not included as a part of CLB Slice structure. Also, it is extremely difficult to find the equivalent number of CLB Slices for the area occupied by a single Block RAM. The results of comparison seem to be favorable towards generic composition schemes due to the Block RAM factor.

8.2 Analysis of Results of ASIC Synthesis

ASIC synthesis results are presented in Chapter 7 for designs targeting both 90 nm and 130 nm technologies. Those results are analyzed in this section with regards to circuit area and supported throughput.

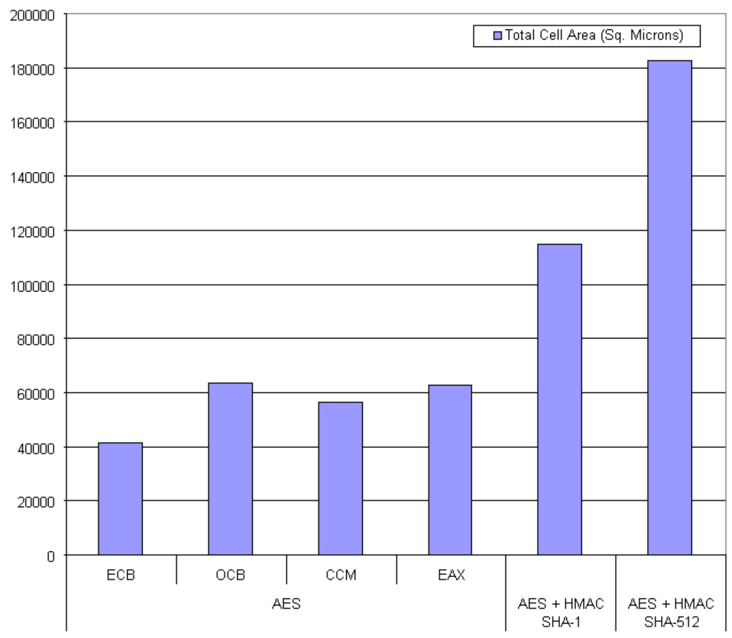


Figure 8.5: Total Cell Area for ASIC Synthesis - Modes of Operations and Generic Composition Schemes (90 nm)

Total cell area in ASIC synthesis reports is divided into Combinational area and non-combinational circuit area. Routing area is not specifically mentioned since the routing model (wireload) which has been used assumes zero routing area. The wireload model uses pin capacitances based on fan-out values for computing routing delays. Area estimation in ASIC synthesis is a reliable scale for comparing circuit areas for different designs.

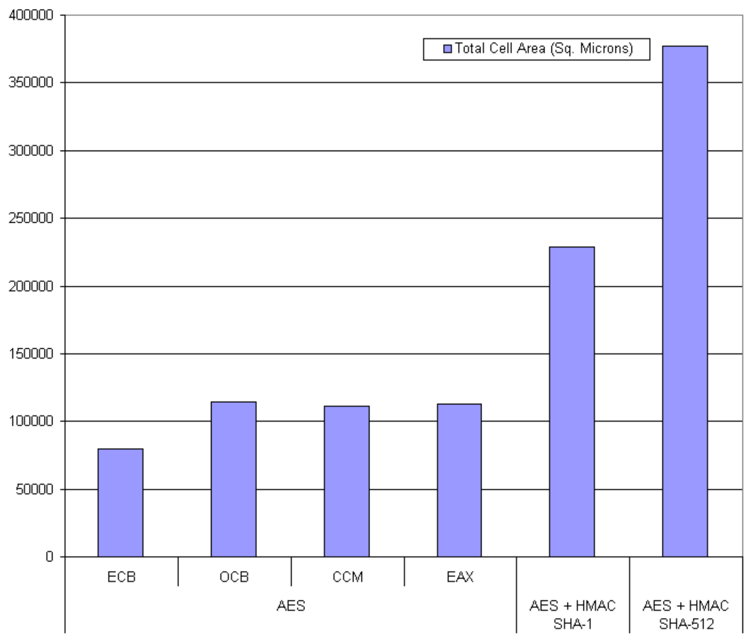


Figure 8.6: Total Cell Area for ASIC Synthesis - Modes of Operations and Generic Composition Schemes (130 nm)

8.2.1 Comparison of Authenticated-Encryption Modes of Operation with Generic Composition Schemes

Comparative graphs related to resource utilization are shown in Fig. 8.5 and Fig. 8.6 for 90 nm and 130 nm technology respectively. From the graphs it is evident that all three authenticated-encryption mode of operations, OCB, CCM and EAX require approximately the same amount of resources. Actually, CCM is the smallest design by a very small margin. ECB is not considered in this analysis since it is not an authentication scheme; it is shown in the graph as a benchmark value. Generic composition schemes occupy significantly more cell area than any mode of operation. For both technologies, AES + HMAC SHA-1 is bigger than authenticated-encryption modes approximately by a factor of 2, whereas AES + HMAC SHA-512 is bigger by a factor of approximately 3. In applications where circuit area is crucial, authenticated-encryption modes would be preferred over generic composition schemes, if they meet the required throughput.

Throughput supported by the above mentioned schemes is compared in Fig. 8.7 and Fig. 8.8 for 90 nm and 130 nm technology respectively. As can be seen, throughput supported by generic composition schemes is slightly greater than single pass scheme like OCB. CCM and EAX, being two-pass schemes, have approximately half the throughput of OCB. Even then, a throughput of 400 Mbps for 90 nm (250 Mbps for 130 nm) should be sufficient for applications related to bitstream security and wireless communications. Throughput requirements for both these applications are quite modest, around 100 to 150 Mbps. Details about the throughput requirements for target applications are given in Chapter 1.

Since all schemes satisfy the throughput requirements, choice of a particular scheme would be based on the scheme with minimum cell area. Therefore, CCM would probably be a proper choice according to the results obtained. Additionally, we can see that the results of both FPGA as well as ASIC synthesis agree in this regard. ASIC synthesis results are more favorable towards the use of CCM mode since the resource utilization is lesser by a factor of 2 when compared with AES + HMAC SHA-1 as opposed to FPGA results where both the schemes use comparable resources.

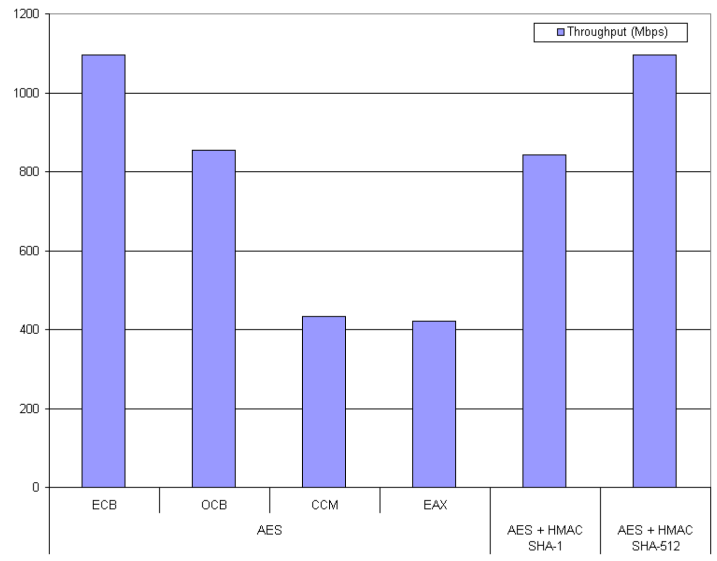


Figure 8.7: Throughput for ASIC Synthesis - Modes of Operations and Generic Composition Schemes (90 nm)

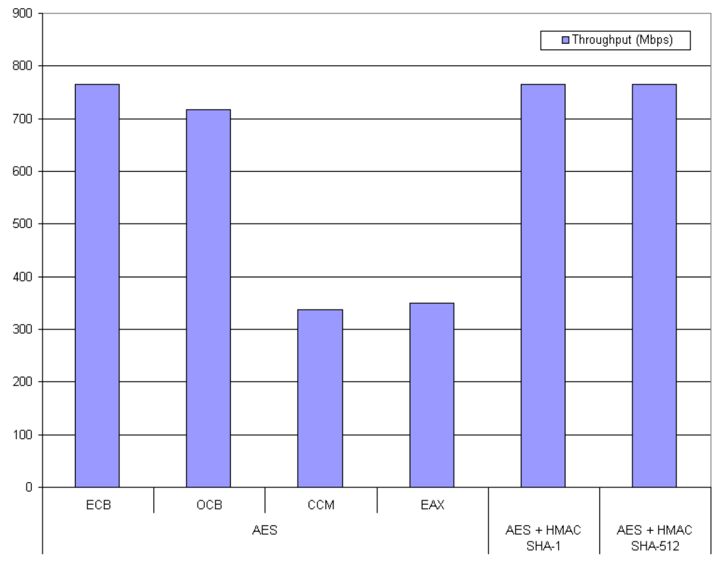


Figure 8.8: Throughput for ASIC Synthesis - Modes of Operations and Generic Composition Schemes (130 nm)

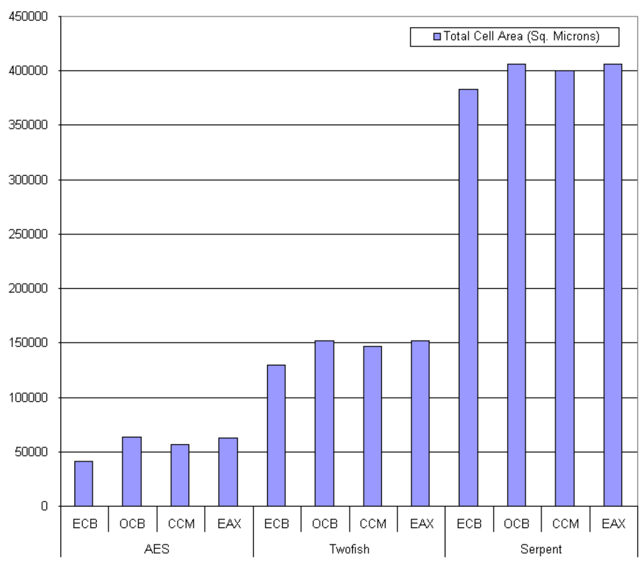


Figure 8.9: Total Cell Area for ASIC Synthesis - Modes of Operations (90 nm)

8.2.2 Comparison of Modes of Operations Based on Different Ciphers

Comparison of cell areas required by mode of operation wrappers for all implemented block ciphers is shown in Fig. 8.9 and Fig. 8.10 for 90 nm and 130 nm technologies respectively. We can get only a general indication of the comparative size of the ciphers form these figures. Modes of operations based on relatively large ciphers like Serpent would be difficult to fit into the area budget for most applications.

8.2.3 Comparison of Throughput/Area Ratio for ASIC Synthesis

Comparison of Throughput/Area ratios for ASIC synthesis would tend to give a fair estimation of the relative rankings of authentication schemes based on the ratio under

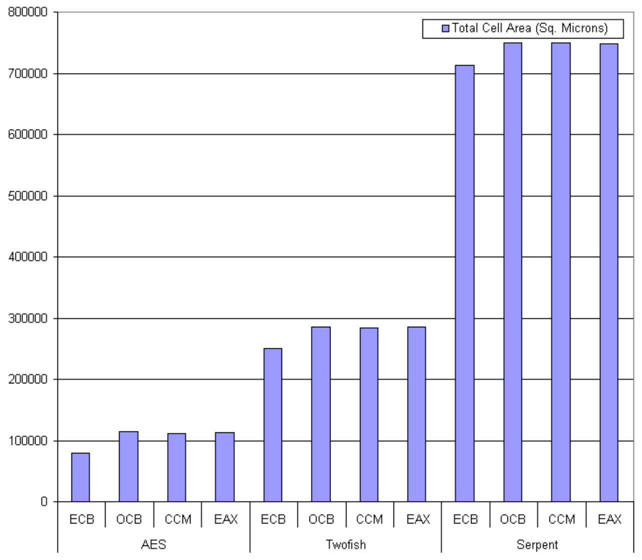


Figure 8.10: Total Cell Area for ASIC Synthesis - Modes of Operations (130 nm)

consideration. This is because, there is no parameter unaccounted for, as is the case with Block RAMs in FPGA implementations.

Graphical comparison of Throughput/Area ratios of authentication schemes for 90 nm and 130 nm technology are shown in Fig. 8.11 and Fig. 8.12 respectively. It would be fair to assume that a particular design would fit in a smaller area if synthesized targeting 90 nm technology than 130 nm. Also, 90 nm designs would run at a higher clock speed than 130 nm thereby resulting in a higher throughput.

Results for 90 nm technology are comparable to corresponding results for FPGA implementations. On the other hand, for generic composition schemes synthesized for 130 nm, throughput decreases and area increases as compared to 90 nm. This is the main reason for the significantly better Throughput/Area ratio for AES modes of operation as compared to generic composition schemes.

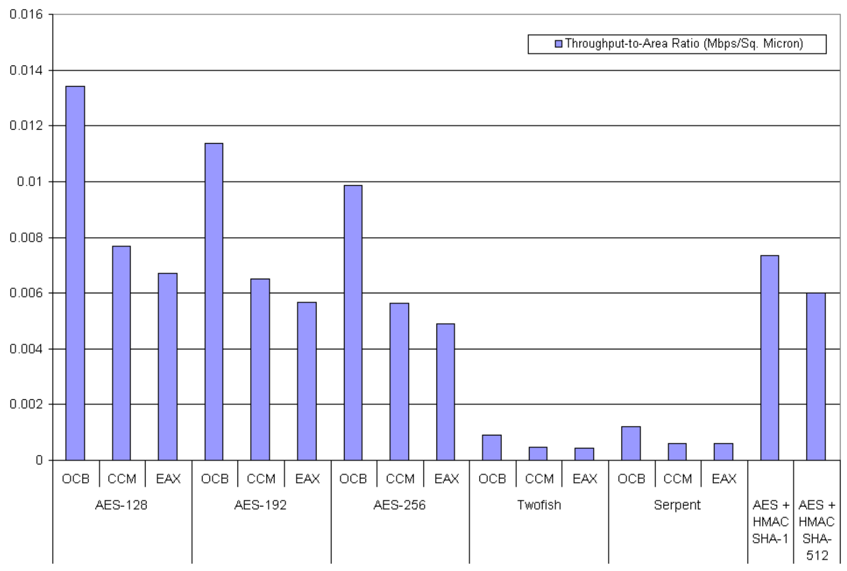


Figure 8.11: Throughput/Area Ratio for ASIC Synthesis of Authentication Schemes (90 nm)

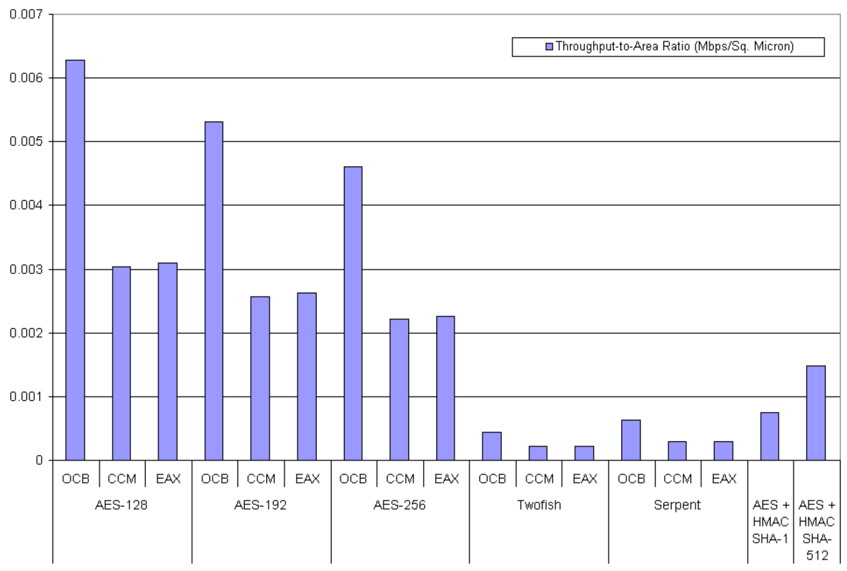


Figure 8.12: Throughput/Area Ratio for ASIC Synthesis of Authentication Schemes (130 nm)

Chapter 9: Modifications, Optimizations and Future Work

Implementation results for authentication schemes have been presented in Chapter 7. Analysis of the obtained results for both ASICs and FPGAs is performed in Chapter 8. This chapter serves as a natural follow-up to analysis of results and shows how the currently obtained results can be improved by introducing certain optimizations. Small but non-trivial modifications to the existing designs can improve performance of some authentication schemes.

Two main aspects of the design improvement will be analyzed in this chapter –

1. Improvement in throughput

2. Reduction of circuit area

9.1 Modifications for Improving Throughput

Mathematical formulas for computing throughput of the authentication schemes under consideration are provided in Chapter 7. The general formula is stated here for reference.

Throughput = $\frac{128 \times L}{n_{cipher\_calls} \times n_{rounds} \times n_{pipeline\_stages} \times t_p} \times n_{allowed\_blocks}$

By looking at the above equation, we can enumerate the following probable ways to improve throughput.

1. Length of the message (L), number of cipher calls required by the mode of operation (n_{cipher_calls}) and number of rounds in the block cipher (n_{rounds}) are parameters which are beyond the control of the hardware designer. Number of block cipher calls for a mode of operation and number of rounds of the block cipher are specified by the respective algorithms.

2. Maximum clock period (t_p) can be controlled up to a certain extent by the hardware designer. As clock period decreases, the value of throughput would increase. Hence we must look at ways by which the clock period of the circuit can be decreased. One of the ways to do that is to introduce pipelining in the circuit.

3. Number of pipeline stages in the block cipher ($n_{pipeline\_stages}$) and maximum number of blocks allowed to pass through the pipeline at any time ($n_{allowed\_blocks}$) are important parameters to be considered when pipelining is introduced in the circuit. The pipeline is fully utilized, when $n_{pipeline\_stages} = n_{allowed\_blocks}$. If the ratio $n_{allowed\_blocks}/n_{pipeline\_stages}$ is less than 1, then it leads to under-utilization of the pipeline, thereby decreasing the throughput.

In order to analyze the throughput improvements, it is necessary to analyze the optimal number of pipeline stages that must be introduced in each block cipher. Irrespective of the number of pipeline stages, a necessary modification is to modify the block cipher such that it can handle multiple data blocks at the same time and fully utilize the pipeline. This change would require some modification in the mode of operation wrappers in order to handle multiple data blocks.

Determining the Optimal Number of Pipeline Stages

Discussion about determination of optimal number of pipeline stages is done assuming that modifications are done to the block cipher and the wrapper to fully utilize the pipeline subject only to restrictions imposed by the algorithms themselves.

OCB mode of operation is a parallelizable single-pass scheme [18]. Theoretically, except for a couple of pre-processing block cipher calls and the final post-processing call, all other block cipher calls can be performed in parallel. Putting in more pipeline stages typically decreases the clock period of the circuit, but this decrease in period saturates after a certain number of stages. In case of AES-OCB with two pipeline stages, the current throughput will get doubled to 1.4 Gbps for FPGAs and more than 2 Gbps for 90 nm ASICs. Currently, FPGA implementation of Twofish-OCB can support a throughput of only 75 Mbps. This is a perfect example of the fact that a sub-optimal ratio $n_{allowed\_blocks}/n_{pipeline\_stages}$ adversely affects the throughput. Currently the ratio $n_{allowed\_blocks}/n_{pipeline\_stages} = 1/8$. If the pipeline were to be fully utilized, then the throughput would increase by a factor of 8.

Analysis of CCM and EAX modes of operation is a little more interesting with regards to the optimal number of pipeline stages. The specifications of both the modes [19][21] state that the modes are not parallelizable. In fact, hardware implementation of these modes of operations is parallelizable to a certain extent. The modes have been designed in such a way that the wrappers use the block cipher alternately, in CBC and Counter (CTR) modes. Subsequent CBC and CTR invocations are mutually exclusive and, in theory, could be parallelized. Therefore, if there are 2 pipeline stages in the block cipher, and if the pipeline is fully utilized, the throughput of CCM and EAX could be doubled.

Let us look at the effect of having more than 2 pipeline stages with CCM and EAX mode. Assume that the underlying block cipher has n pipeline stages, such that $n \geq 2$. Also, the block cipher supports full utilization of the pipeline. The restriction on utilization of the pipeline is imposed by the structure of CCM and EAX modes. The CCM and EAX modes can be optimized to handle at most 2 blocks being in the pipeline at any time. This would lead to under-utilization of the pipeline by a factor of $n/2$. Thus, the optimal value of n would most probably be 2. Throughput for $n = 4$ should be analyzed since, the sub-optimal value of $n_{allowed\_blocks}/n_{pipeline\_stages}$ would decrease the throughput by 2. On the other hand, additional 2 pipeline stages might actually decrease the clock period, t_p. If the clock period decreases by a factor of 2, then the resultant throughput will remain unchanged.

9.2 Modifications for Reducing the Circuit Area

When area is the main concern, sacrificing performance will generally lead to a smaller area. Another way to minimize area is sharing of resources in the design. Examples of practically reducing area by resource sharing in hardware implementation of cryptography can be found in [23]. Resource sharing might increase the size of the control logic in a small way in order to incorporate the controls for passing data from various modules to the common module and redistributing output data from the common module to its proper destination.

The focus of this section is to utilize in-built features of modes of operations in order to reduce the circuit area. CCM and EAX modes of operations are designed in order to use only the forward (encryption) function of the block cipher for all operations. Reverse (decryption) function of the block cipher is not required at all in CCM and EAX mode of operation. Hence one of the optimizations in case of CCM and EAX modes would be to implement the block cipher with only the forward functionality rather than both functionalities as is currently done.

I tried to modify the existing cipher cores such that they would support only encryption and was able to get about 15% improvement in area for AES and Twofish and about 20% improvement for Serpent. We can safely assume that complete redesigning of the cipher blocks would result in much more savings in area.

Using Preprocessing to Reduce Circuit Area

Looking at the comparison of resource utilization for various modes of operation, it is quite surprising to find that CCM is the most area-efficient among the 3 implemented modes of operation. Comments and critiques of CCM mode [21][24] point to the fact that the mode is inefficient due to unnecessary parametrization. Bellare, Rogaway and Wagner [21] specify that they designed the EAX mode in order to overcome the shortcomings of CCM mode. The question then is – Why is this discrepancy in the results?

I analyzed my implementations of both CCM and EAX mode in order to find the reason for the seemingly out-of-place results. The reason for area-efficient nature of CCM is that the inefficiencies mentioned in [21] do not add to the circuit area because of the pre-processing done in CCM. Since CCM is not on-line, it is safe to assume that such pre-processing is practically possible. OCB and EAX modes require very minimal pre-processing and hence I have not used any pre-processing scheme with these modes. Possible preprocessing includes one-time processing of static headers in case of EAX [21]. Headers might be static over the course of communication session and might contain information such as IP address of sender, receiver and fixed cryptographic parameters related to the session. There are no straightforward pre-processing techniques evident for OCB mode of operation.

9.3 Projected Benefits from Optimizations

While there are possible techniques to optimize the hardware implementations of authenticated-encryption modes of operations, there seems to be no straightforward improvement for generic composition schemes. Results for implementations of HMAC are comparable to previously published results. Some optimizations put forth by [14], if incorporated in the design can possibly improve the performance by a small margin. It would be prudent to assume that supported throughput for HMACs does actually saturate at the current levels unless some novel architecture is found. The algorithm is too simple to derive novel schemes for implementing it. On the other hand, there is a vast potential for improvement with the authenticated-encryption modes of operations. For single pass schemes like OCB, multi-gigabit throughput can be easily supported with a properly designed block cipher. The current throughput requirements of applications under consideration are modest enough to be satisfied by all implemented schemes. But thinking ahead, current network speeds might get obsolete very fast considering the rate at which technology is progressing. It did not take long to move from 11 Mbps supported by 802.11b wireless protocol to 54 Mbps supported by 802.11g protocol. Standards have already been put forth for 100+ Mbps support in 802.11n wireless standard. If considerations are stretched beyond wireless networking, then multi-gigabit ethernet is one application where high throughput rate would be required. From analysis of results and considered optimizations, we can see that generic composition schemes would soon become incapable handling future application requirements. On the other hand, the newer schemes are much better designed to provide improved performance for emerging needs.

9.4 Summary

Possible optimizations explained in this chapter are summarized below –

1. The hardware implementation of block cipher must be modified to fully utilize its internal pipeline. Also, similar modifications must be made to the mode of operations wrapper to fully support the modified block cipher functionality.

2. Throughput-to-area ratio must be analyzed in order to determine the optimal number of pipeline stages. Theoretical analysis seems to indicate that 2 pipeline stages should be optimal for CCM and EAX mode of operation. Further analysis of throughput-to-area ratio is necessary to estimate the optimal number of pipeline stages for OCB mode of operation.

3. Reduction in circuit area is possible in CCM and EAX mode by removing the decryption functionality from the underlying block cipher.

4. Use of preprocessing wherever possible, (maybe in future implementations of other modes of operations) can greatly reduce the circuit area as demonstrated by my implementation of CCM mode of operation.

Chapter 10: Summary

Combined authenticated-encryption modes of operation for block ciphers have been implemented targeting Xilinx Virtex 4 family of devices. Additionally, these designs have been adapted for synthesis targeting 90 nm and 130 nm standard cell ASIC technologies based on TSMC libraries. For the sake of comparison, generic composition schemes, consisting of an independent encryption algorithm and a separate authentication algorithm have also been implemented.

The authentication schemes have been compared with regards to a several applications, with primary focus on FPGA bitstream security and authenticated-encryption in wireless networks. The current state of these applications has a throughput requirement of about 150 Mbps. Additionally, both applications would typically have a low area budget. Therefore, design effort was concentrated on optimizing the designs for low area and a high Throughput/Area ratio. Analysis of possible future optimizations has been provided in order to come up with a more general and fair comparison, expandable to future requirements and other more demanding applications.

Results of both FPGA implementation and ASIC synthesis are consistent in the ranking of authentication schemes with regards to resource utilization and throughput. Generic composition schemes provide comparable throughput to authenticated-encryption schemes for both FPGA as well as ASIC implementations, at a cost of higher circuit area. As expected, ASIC synthesis results give a better throughput than FPGA implementation results and this can be primarily attributed to support for higher clock frequency in ASIC designs. Requirement of larger circuit area for generic composition schemes is more pronounced in ASICs than in FPGAs. This is at least partially because of the lack of unified measure of resource utilization in FPGAs. In case of ASICs, the total cell area in square microns covers all circuit resources. In case of FPGAs, multiple sources of resource utilization exist. These measures, number of CLB Slices, number of LUTs, number of Block RAMs, etc. are hard to unify and convert among one another.

In terms of applications, FPGA bitstream security engine is required to support a modest throughput requirement of 150 Mbps. Such an engine would be implemented on the FPGA fabric. Hence, ASIC synthesis results should be considered for choosing an authentication scheme which would be optimal for this application. From the results presented earlier, it is evident that all authentication schemes which have been considered in this thesis can support the required throughput. Therefore, the deciding criteria is minimum area. CCM mode of operation has the smallest implementation area and hence is the best candidate for providing bitstream security services. Additionally, CCM has been standardized by NIST as a mode of operation for use with AES. FPGA vendors would generally prefer to use standardized security schemes as it would be easier to convince customers about security services offered by their products.

In applications related to wireless networking, throughput requirements are easily satisfied by both FPGA as well as ASIC implementations of all authentication schemes. The decisive parameter is again the implementation area of the designs. CCM is the winner on the basis of that criteria. It is also significant that the IEEE

802.11i standard for wireless networks uses a subset of CCM mode for data authentication. Analysis shows that the throughput values reported in this thesis can be doubled by implementation of a two-stage pipeline in the underlying cipher and providing a mechanism to fully utilize this pipeline. With some optimizations, CCM should be able to support a throughput of about 500 Mbps for FPGA implementation and about 900 Mbps for 90 nm ASIC technology. This level of throughput would be sufficient to meet the requirements of wireless networks well into the future.

For other network applications, with higher throughput requirements, some tradeoffs have will have to be considered for selection of the best candidate. OCB and the generic composition scheme comprising of AES and HMAC SHA-512 have comparable throughput values. Additionally, there is a vast scope for improvement of throughput with OCB mode of operation as opposed to the generic composition scheme. Implementing a fully utilizable four-stage pipeline would increase the throughput supported by OCB-AES well beyond 3 Gbps for 90 nm ASICs and beyond 2 Gbps for FPGAs. If such high levels of throughput are not required, then implementations with better Throughput/Area ratios should be considered. For FPGA implementations, the choice would be between AES-OCB and generic composition scheme using AES and HMAC SHA-512. For ASIC implementations, AES-OCB outperforms other schemes based on Throughput/Area ratio. One of the possible issues which would lead to avoidance of OCB mode is patent restrictions. In that case, CCM mode is a good choice if the throughput requirements can be met.

Bibliography

Bibliography

[1] W. Stallings, *Cryptography and Network Security: Principles and Practice.* Pearson Education, 2002.

[2] A. J. Menezes, S. A. Vanstone, and P. C. V. Oorschot, *Handbook of Applied Cryptography.* Boca Raton, FL, USA: CRC Press, Inc., 1996.

[3] B. Schneier, *Applied Cryptography: Protocols, Algorithms, and Source Code in C.* New York, NY, USA: John Wiley & Sons, Inc., 1993.

[4] "Proposed modes of operation, *http://csrc.nist.gov/CryptoToolkit/modes/ proposedmodes/index.html.*"

[5] A. Telikepalli, "Is your FPGA design secure?" *XCell Journal*, 2003.

[6] I. Hadzic, S. Udani, and J. M. Smith, "FPGA viruses," in *FPL '99: Proceedings of the 9th International Workshop on Field-Programmable Logic and Applications.* London, UK: Springer-Verlag, 1999, pp. 291–300.

[7] "Security Scenarios, Actel Corporation, *www.actel.com/documents/ SecurityScenarios.pdf* ."

[8] S. Trimberger, R. Pang, and A. Singh, "A 12 Gbps DES encryptor/decryptor core in an FPGA," in *CHES '00: Proceedings of the Second International Workshop on Cryptographic Hardware and Embedded Systems.* London, UK: Springer-Verlag, 2000, pp. 156–163.

[9] "IEEE Standard for Information Technology - Medium Access Control Security Enhancements," IEEE Computer Society, Tech. Rep. IEEE 802.11i Part 11, Amendment 6, 2004.

[10] T. Iwata and K. Kurosawa, "OMAC: One-Key CBC MAC," 2002.

[11] "The Keyed Hash Message Authentication Code," National Institute of Standards and Technology (NIST), Tech. Rep. FIPS 198, 2002.

[12] C. S. Jutla, "Encryption Modes with Almost Free Message Integrity," Cryptology ePrint Archive, Report 2000/039, 2000.

[13] M.-Y. Wang, C.-P. Su, C.-T. Huang, and C.-W. Wu, “An HMAC processor with integrated SHA-1 and MD5 algorithms,” in *ASP-DAC '04: Proceedings of the 2004 conference on Asia South Pacific design automation.* Piscataway, NJ, USA: IEEE Press, 2004, pp. 456–458.

[14] R. Lien, T. Grembowski, and K. Gaj, “A 1 Gbit/s partially unrolled architecture of hash functions SHA-1 and SHA-512,” pp. 324 – 338, Jan 2004.

[15] L. Dadda, M. Macchetti, and J. Owen, “An ASIC design for a high speed implementation of the hash function SHA-256 (384, 512),” in *GLSVLSI '04: Proceedings of the 14th ACM Great Lakes symposium on VLSI.* New York, NY, USA: ACM Press, 2004, pp. 421–425.

[16] “Secure Hash Standard,” National Institute of Standards and Technology (NIST), Tech. Rep. FIPS 180-1, 1995.

[17] “Xilinx Virtex-4 datasheet,” 2005, http://direct.xilinx.com/bvdocs/ publications/ds112.pdf.

[18] P. Rogaway, M. Bellare, J. Black, and T. Krovetz, “OCB: A block-cipher mode of operation for efficient authenticated encryption,” in *ACM Conference on Computer and Communications Security*, 2001, pp. 196–205.

[19] D. Whiting, N. Ferguson, and R. Housley, “Counter withCBC-MAC (CCM), submission to NIST,” http://csrc.nist.gov/CryptoToolkit/modes/proposedmodes.

[20] “Specification for the advanced encryption standard (AES),” Federal Information Processing Standards Publication 197, 2001. [Online]. Available: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[21] M. Bellare, P. Rogaway, and D. Wagner, “The EAX Mode of Operation,” in *FSE*, 2004, pp. 389–407.

[22] M. Bellare, J. Kilian, and P. Rogaway, “The security of the cipher block chaining message authentication code,” *Journal of Computer and System Sciences*, vol. 61, no. 3, pp. 362–399, 2000.

[23] P. Chodowiec, “Comparison of the hardware performance of AES candidates using reconfigurable hardware,” Master's thesis, George Mason University, 2002.

[24] P. Rogaway and D. Wagner, “A critique of CCM,” Cryptology ePrint Archive, Report 2003/070, 2003.

Curriculum Vitae

Milind M. Parelkar received his Bachelor of Electronics degree from the University of Mumbai (Bombay), India in 2002. He was ranked among the top 10 students in the University. He was awarded the Vision Award for Academic Excellence at George Mason University in 2004. His publication, "*Implementation of EAX Mode of Operation for FPGA Bitstream Encryption and Authentication*" was accepted as a poster at FPT 2005 held at Singapore. He has been involved with teaching various undergraduate and graduate courses at George Mason University, both as a teaching assistant and a lab instructor.