AN ABSTRACT OF THE THESIS OF

Ahmed Al Faresi for the degree of Master of Science in

Electrical & Computer Engineering presented on June 13, 2005.

Title: Hardware Realization of OCB mode for Efficient Authenticated Encryption.

Signature redacted for privacy.

Abstract approved: _____

Çetin Kaya Koç

Authenticated-Encryption modes of operation have recently received great attention amongst researchers. Such modes of operation provide both privacy and authenticity. A proposed mode in this category is the Offset Codebook mode (OCB) by Rogaway et al. This mode shows great substantial advantages over conventional modes. In the past when one wanted a shared-key mechanism that provided both privacy and authentication one would first encrypt separately and then use a Message Authentication Code (MAC). The cost of such a mechanism is equal to the cost of encryption plus the cost of producing the MAC, usually done with different keys for each operation. The OCB mode however uses one key and provides privacy and authenticity simultaneously and with lower costs and higher speed then conventional methods.

Since this mode is relatively new, the proposed work provides a synthesizable hardware implementation of the OCB encryption algorithm. Furthermore an efficient hardware realization of the Advanced Encryption Standard (AES) to be incorporated in the OCB mode is provided. The architectural designs presented are analyzed and synthesized in terms of performance and overall throughput. The results are evaluated against other conventional modes in this area.

# Hardware Realization of OCB mode for Efficient Authenticated Encryption

by

Ahmed Al Faresi

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented June 13, 2005
Commencement June 2006

Master of Science thesis of Ahmed Al Faresi presented on June 13, 2005

APPROVED:

Signature redacted for privacy.

Major Professor, representing Electrical & Computer Engineering

Signature redacted for privacy.

Director of the School of Electrical Engineering & Computer Science

Signature redacted for privacy.

Dean of the Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Signature redacted for privacy.

Ahmed Al Faresi, Author

ACKNOWLEDGMENTS

In the name of Allah, the All merciful, the All compassionate. All praise is due to Allah. I would like to thank my parents for their support during my studies overseas. I would like to thank my advisor Dr Koç for giving me the opportunity to work on this interesting project. Dr Koç's directions, reviews and valuable comments helped me accomplish this work.

Last but not least I would like to thank my brothers and sisters and my friends, who supported me all the way through.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF TABLES

# HARDWARE REALIZATION OF OCB MODE FOR EFFICIENT AUTHENTICATED ENCRYPTION

## 1. INTRODUCTION.

With the advent of new block ciphers, such as the Advanced Encryption Standard (AES), there is a need to update long-standing modes of operation and an opportunity to consider the development of new modes [7]. Highly motivated by NIST (National Institute of Research and Technology) researchers had a great strive to achieve an authenticated-encryption scheme that is low in cost and high in efficiency. In the past the primary form of authenticated-encryption was actually the one of generic composition, where one would encrypt a message and compute a message authentication code (MAC) separately. With such a scheme the cost is calculated to be the cost to encrypt plus the cost to MAC. Such schemes suffered weaknesses and eventually where misused. Recently two schemes that offered a low cost authenticated-encryption have been proposed by Jutla [8], namely Integrity aware cipher block chaining (IACBC) and Integrity aware parallelizable mode (IAPM). A refinement of IAPM gave birth to the OCB (offset codebook mode) proposed by Rogaway [2]. This mode provides both authenticity and privacy at a cost almost equivalent to getting privacy alone. Moreover OCB is fully parallelizable a feature that would show great promise in hardware implementation. One other important feature of this algorithm is its ability to process a message of arbitrary length, meaning the message size need not be known in advance.

This thesis presents an attempt to provide a hardware realization of the OCB-AES encryption engine. The design will target high efficiency and performance while maintaining the ability of the online feature that is the ability to read variable length messages. This work will describe the scalable hardware design and analyze the synthesis results. The results are compared to other proposed authenticated-encryption designs.

The comparison is done in terms of performance (bits/cycle), total computational time, throughput and complexity.

This chapter is further divided into 4 sections. In the first section I describe the AES standard as an example of the block cipher of choice in this study. The next section describes the message authentication codes and provides a brief introduction to authenticity. The third section gives a general description of authenticated-encryption schemes as a form of literature background to this thesis. The fourth and last section details the outline for the rest of the thesis.

## 1.1. Advanced Encryption Standard (AES)

OCB is a block-cipher mode of operation. Where mode of operation, or mode, for short, is an algorithm that features the use of a symmetric key block cipher algorithm to provide an information service, such as confidentiality or authentication [7]. The OCB mode requires the use of a block cipher, and gives freedom of choice of the block cipher to be incorporated. AES seems to be the logical choice, with a key of 128 bits as the minimum recommended in todays cryptographic applications. It is worth mentioning that choosing a block length of less than 128 bits could result in a lower security bound. Furthermore it would seem illogical to use a modern algorithm with an old block cipher such as DES. The implementation in the thesis is made with AES being the block cipher of choice; therefore we would like to give a brief description of the AES algorithm.

AES refers to an algorithm proposed by Rijndael and accepted by NIST as the new standard to replace DES. Rijndael then became attributed to the algorithm of AES. AES is designed for use with keys of length s 128, 192 and 256 bits. Since the implementation uses a 128-bit key we will explain the algorithm in terms of this key choice. The algorithm consists of 10 rounds. Each round has a round key, derived from the original key. A round starts with an input of 128 bits and produces an output of 128 bits [9].

There are four basic steps, called layers that are used to form the rounds:

1. The ByteSub Transformation: This non-linear layer is for resistance to differential and linear cryptanalysis attacks.
2. The ShiftRow Transformation: This linear mixing step causes diffusion of the bits over multiple rounds.
3. The MixColumn Transformation: This layer has a purpose similar to ShiftRow.
4. AddRoundKey: The round key is XORed with the result of the above layer [9].

The algorithm is depicted in Figure 1.1. One can see how the plaintext is processed through the 4 layers to produces the corresponding ciphertext. There are basic features that should be considered when implementing the Rijndael AES.

### *1.1.1. Hardware Implementation*

Hardware implementations can be optimized for speed and size. In hardware most often increase in size means higher cost, which is a feature that may not be in that advantage of parallel implementations. Rijndael shows the second highest throughput for none feedback modes like OCB. However for fully pipelined implementations the area requirement increases but the throughput remains unaffected.

### *1.1.2. Encryption versus decryption*

The encryption and decryption functions in Rijndael differ. FPGA study reports that the implementation of both encryption and decryption takes about 60 percent more space than the implementation of encryption alone. Rijndael's speed does not vary significantly between encryption and decryption, although the key setup performance is slower for decryption than encryption [1].

### *1.1.3. Potential for instruction-level parallelism*

This refers to the ability to exploit ILP features in current and future processors.

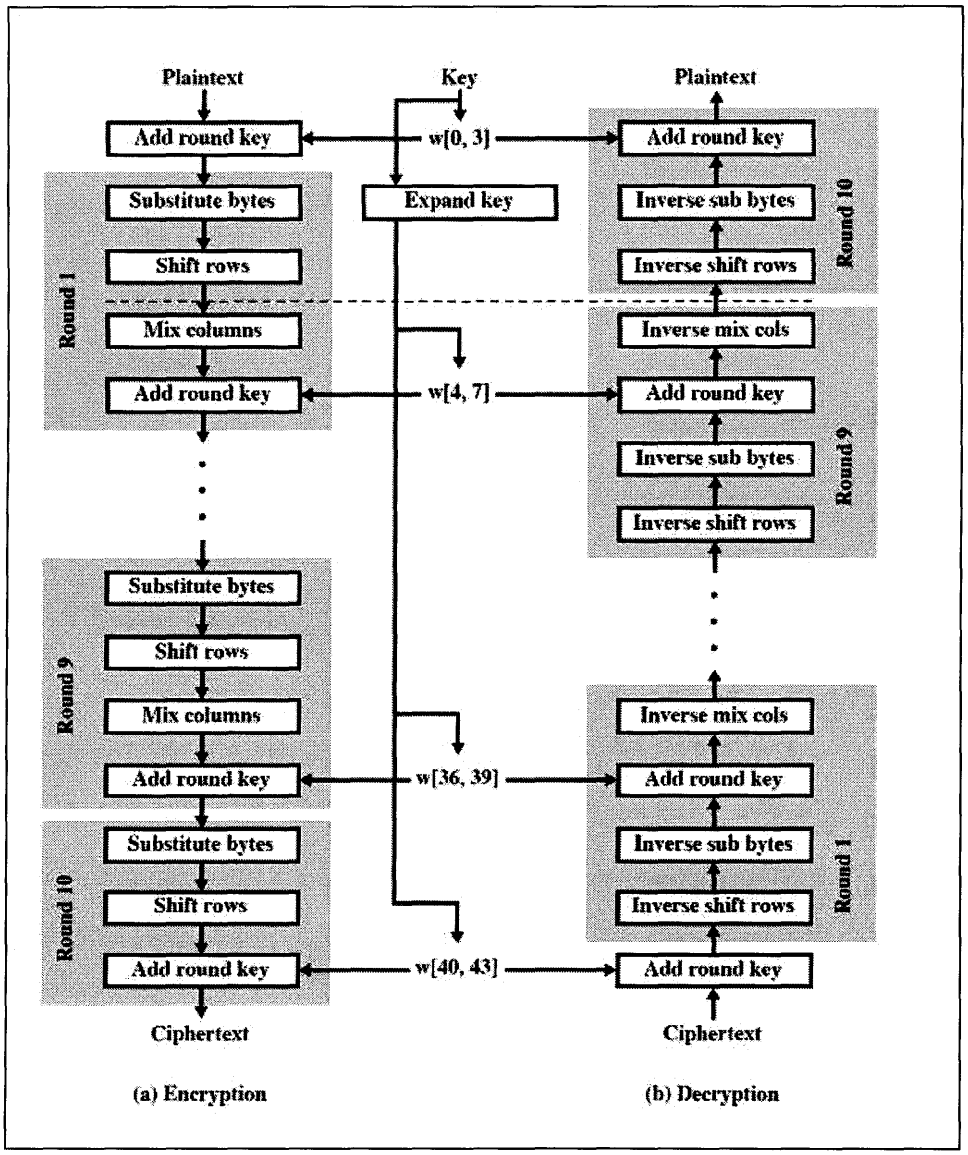


FIGURE 1.1: AES Encryption and Decryption [1].

## 1.2. Authentication Code (MAC)

In an authenticated-encryption scheme, the privacy factor is given by encryption and the MAC provides the authenticity factor. In contrast the OCB mode uses the AES block cipher to supply the privacy factor, the authenticity is provided by a TAG value, which will be further discussed in detail. A brief introduction of the MAC would pave the way to understand the authenticity factor in the OCB later on.

A MAC, also known as a cryptographic checksum, is generated by a function C of the form: MAC=$C_K(M)$, where M is a variable-length message, K is a secret key shared only by the sender and receiver, and $C_K(M)$ is the fixed-length authenticator. The MAC is appended to the message at the source at a time when the message is assumed or known to be correct. The receiver authenticates that message by re-computing the MAC [1].

A MAC function is similar to encryption. One difference is that the MAC algorithm need not be reversible, as it must for decryption. In general, the MAC function is a many-to-one function. The domain of the function consists of messages of some arbitrary length, whereas the range consists of all possible MACs and all possible keys. If an n-bit MAC is used, then there are $2^n$ possible MACs, whereas there are N possible messages with $N >> 2^n$ [1].

In Fig1.2 (a) and 1.2 (b) the first cases the MAC is calculated with the message as input and is then concatenated to the message. The entire block is then encrypted. In the second case, the message in encrypted first. Then the MAC is calculated using the resulting ciphertext and concatenated to the ciphertext to form the transmitted block.

Earlier in the introduction we stated that authenticated-encryption schemes relied on separately encrypting and then MACing to get both privacy and authenticity, however such schemes would prove costly and inefficient since each operation incurs its own cost. The idea of combining privacy and authenticity raised some concerns. There was a great concern among researchers that a scheme that provides a privacy-authenticity combo

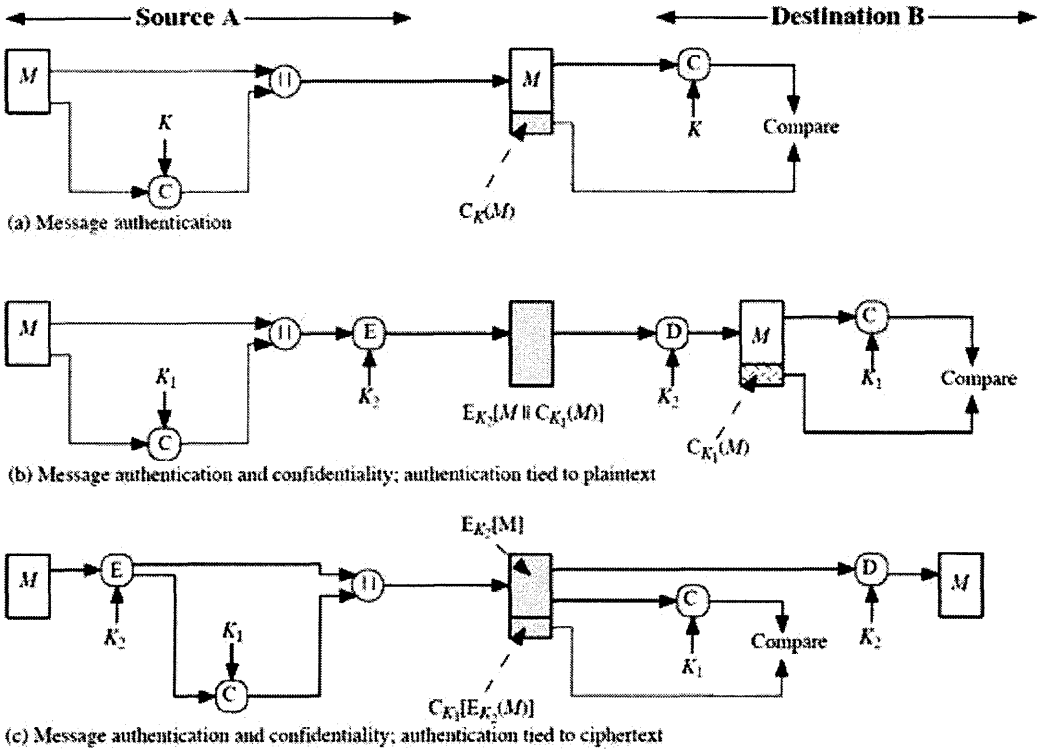


FIGURE 1.2: Basic Uses of Message Authentication Code (MAC) [1].

may lack the expected degree of authenticity. This concern was driven by the countless failed attempts to correctly add authenticity to encryption schemes. However OCB provides privacy and authenticity more like in an encapsulation scheme.

Authenticity in the context of an encapsulation scheme is a more general concept than that of a MAC. A MAC makes explicit a particular mechanism, namely the attachment of a tag to the transmission. (The tag, computed using the key, is created by the sender and checked by the receiver.) An encapsulation scheme may use a MAC, or may not, and consideration of authenticity for such a scheme cannot make assumptions about the presence of any type of mechanism. But there is a deeper difference between a MAC and a general authentication scheme. In formalizing the security of a MAC the adversary makes a number of queries to a MAC-generation oracle, with each

query mapping the message $M_i$ to its tag $t_i$. After that the adversary has to come up with a new message M and a tag t such that the receiver will deem (M; t) authentic. In particular, the adversary must "know" the message M that is being forged, insofar as the adversary outputs it along with t. In contrast, an adversary attacking an authentication scheme in the general sense we are defining wins even if he/she does not know what is the message M which is being forged. All that is required is that there is such a message underlying C -that is, the receiver will recover something in the message space M (and not an indication that C is bogus) [10].

## 1.3. Authenticated Encryption Scheme

This section provides insight into authenticated encryption schemed. We start by defining the concept then we go on to describe the current approaches applied to obtain authenticated encryption and we conclude with a justification for choosing OCB as the candidate for this goal.

### *1.3.1. Notion of authenticated encryption*

The concept of authenticated encryption thats is namely combining privacy with authenticity is one of traditional nature and has been misunderstood in the cryptographic arena for some time. The approach of the problem was incorrect and was leading to many miserable failures in trying to achieve the goal of privacy + authenticity. Afterwards the concept of trying to combine privacy with authenticity was rendered unsafe, due to the fear that such scheme would not provide an efficient amount of authenticity. However this problem was in large due to the common belief amongst security scientists that redundancy added in encryption schemes implied authenticity as well. The misunderstanding comes from the idea that enciphering was thought to equal encrypting, which is not true!

Enciphering is applying a permutation $P_k$ on a plaintext M where k is a shared key. However in order to achieve good encryption that is encryption in the semantic sense, (i.e. encryption that could withstand a chosen plaintext attack and beyond) enciphering is rendered insufficient. Only when a message has enough entropy in it then maybe enciphering will do the trick and can achieve a close value of semantic encryption.

I quote the following paper that justifies some old intuition to the problem At some level it would seem to be folklore that enciphering strings which employ nonces or redundancy makes for good encryption. In the security literature one sees many statements to the effect that we assume that messages to be encrypted employ adequate redundancy, or we avoid replay attacks by including a nonce in the messages we encrypt. Our results help formalize what such authors may have had in mind, since the statements above become meaningful and true when encryption means enciphering and when the roles of nonces and redundancy are formally defined [10].

### *1.3.2. Generic Composition*

The term authenticated encryption scheme refers to a shared-key based transform whose goal is to provide both privacy and authenticity of the encapsulated data. In such a scheme the encryption process applied by the sender takes the key and a plaintext to return a ciphertext, while the decryption process applied by the receiver takes the same key and a ciphertext to return either a plaintext or a special symbol indicating that it considers the ciphertext invalid or unauthentic.

The design of such schemes has attracted a lot of attention historically. The early schemes were typically based on adding redundancy to the message before encrypting, and many of these schemes were broken. Today authenticated encryption schemes continue to be the target of design and standardization efforts. A popular modern design paradigm is to combine MACs with standard block cipher modes of operation. The goal of symmetric encryption is usually viewed as privacy, but an authenticated encryption

scheme is simply a symmetric encryption scheme meeting additional authenticity goals [3].

In order to analyze the security of authenticated encryption schemes, we need consider the two notions of authenticity for symmetric encryption namely integrity of plaintexts and integrity of cipher-texts.

- Integrity of plaintext: It is computationally impossible to produce a cipher-text that decrypts to a message never encrypted by the sender.
- Integrity of ciphertext: It is computationally impossible to produce a ciphertext that was not produced previously by the sender.

Generic composition is making a black-box use of a given symmetric encryption scheme and a given MAC. The following are the required tools to achieve this goal.

- Encryption schemes for privacy.
- Message authentication schemes for authenticity.
- Provable security analysis.

Combining the above tools achieves the goals of authenticated encryption. Following is an account of the symmetric key based encryption scheme:

- Constructions: CBC-mode encryption, CTR-mode encryption, OFB mode.
- Security notions: Authenticity and Privacy.
- Authenticity: Integrity of both plaintexts and ciphertexts.
- Privacy: Indistinguishability and Non-malleability under either chosen- plaintext attacks or adaptive chosen-ciphertext attacks where those are defined as follows:
  - Chosen plaintext attack To gain further secretive information by choosing arbitrary plaintexts to be encrypted and obtaining the corresponding ciphertexts.

– Adaptive chosen-ciphertext attack To choose subsequent ciphertexts based on the information received from previous requests

| | Privacy | | | Integrity | |
|---|---|---|---|---|---|
| Composition Method | IND-CPA | GIND-CCA | NM-CPA | INT-PTXT | INT-CTXT |
| Encrypt-and-MAC | Insecure | Insecure | Insecure | Secure | Insecure |
| MAC-then-Encrypt | Secure | Insecure | Insecure | Secure | Insecure |
| Encrypt-then-MAC | Secure | Insecure | Insecure | Secure | Insecure |

TABLE 1.1: Summary of security results, under the assumption that the MAC scheme is weakly unforgeable [3].

| | Privacy | | | Integrity | |
|---|---|---|---|---|---|
| Composition Method | IND-CPA | GIND-CCA | NM-CPA | INT-PTXT | INT-CTXT |
| Encrypt-and-MAC | Insecure | Insecure | Insecure | Secure | Insecure |
| MAC-then-Encrypt | Secure | Insecure | Insecure | Secure | Insecure |
| Encrypt-then-MAC | Secure | Secure | Secure | Secure | Secure |

TABLE 1.2: Summary of security results, under the assumption that the MAC scheme is strongly unforgeable [3].

We consider three composition methods where E is an encryption function; T is a tagging algorithm for some message authentication scheme. $K_e$ is a key for encryption and $K_m$ is a key for message authentication, and where || denotes appending.

1. Encrypt-and-MAC

   $E_{K_e,K_m}(M) = E_{K_e}(M)||T_{K_m}(M)$

2. MAC-then-Encrypt

   $E_{K_e,K_m}(M) = E_{K_e}(M)||T_{K_m}(M)$

3. Encrypt-then-MAC

$$E_{K_e,K_m}(M) = E_{K_e}(M)||T_{K_m}(E_{K_e}(M))$$

Observing the results from Tables 1.1 and 1.2 it seems Encrypt-then-MAC is the most secure method provided that the MAC scheme is strongly unforgeable. Formal security goals for authenticated encryption

- Authenticity: Integrity of ciphertexts (INT-CTXT), Integrity of plaintext (INT-PTXT)
- Privacy: Indistinguishability and non-malleability each of which can be considered either under chosen-plaintext or (adaptive) chosen-ciphertext attacks (IND-CPA, IND-CCA, NM-CPA, NM-CCA)

Secure: The composite encryption scheme is secure, assuming:

- The component encryption scheme is IND-CPA secure and the base MAC scheme is UF-CMA (Unforgeable under chosen-message attack) secure.

Insecure: The composite scheme is insecure:

- The exists some IND-CPA secure symmetric encryption and some MAC UF-CMA such that the composite scheme based on them does not meet the security requirement in question

Any pseudorandom function is a strongly unforgeable MAC, and most practical MACs seem to be strongly unforgeable. Therefore, analyzing the composition methods under this notion is a realistic and useful approach. The use of a generic composition method secure in the above sense is advantageous from both performance and of security architecture point of view. The performance benefit arises from the presence of fast MACs such as HMAC and UMAC.

The architectural benefits arise from the stringent notion of security being used. To be secure, the composition must be secure for all possible secure instantiations of

its constituent primitives. If it is secure for some instantiations but not others, we declare it insecure. An application can thus choose a symmetric encryption scheme and a message authentication scheme independently and then appeal to some fixed and standard composition technique to combine them. No tailored security analysis of the composed scheme is required [11].

Generic composition is one of many approaches for authenticated encryption designs. Other general approaches include encryption with redundancy that is redundancy is appended to the message and then passes through a block cipher mode of operation for encryption. However such schemes where under so many attacks that eventually they got broken. Another form of encryption with redundancy in the style of [10] involves adding randomness/redundancy and then enciphering instead of encrypting, which has proven to work, provided that has a variable-length pseudorandom permutation, which is very expensive. Another scheme is the RPC mode of [12] however it inefficient in terms of space and performance compared to generic composition. Another scheme is IACBC mode of Jutla [8] and a refinement of Julta's work is an elegant mode namely OCB by Rogaway [2] which uses $[|M|/n] + 2$ block cipher invocations. Implementation and testing of that scheme is the scope of this thesis, as it is required to compare its speed with that of generic composition methods that use fast MACs (cf. [13, 14, 15]).

## 1.4. Thesis Outline

Chapter 2 describes the alternative options and implementations of authenticated encryption. Chapter 3 describes the OCB mode in terms of features, notation, and properties. Then there will be an elaborate description of the encryption scheme itself. Chapter 4 describes the security considerations of OCB and the implementation options available. Then it describes the challenges and the application of such a mode. Chapter 5 deals with the implementation results and analysis. There we present the summary

of findings and display a comparison of these findings with other hardware implemented authenticated-encryption designs, in addition recommendations for future work are given.

# 2. OVERVIEW OF GOAL

NIST plans to develop new standard modes that address symmetric block cipher algorithms, which need to be independent in terms of key size and block length. In addition the four DES modes (ECB, CBC, ECB, OFB) that were defined in Federal Information Processing Standard (FIPS) 81. are to be included. The goal arises because new modes are required to address block ciphers such as AES, which are replacing DES in many applications.

## 2.1. Encryption Modes with Almost Free Message Integrity

Charanjit Jutla of IBM presented two new modes, each of which provides both confidentiality and message integrity: Integrity Aware Cipher Block Chaining Mode (IACBC) and Integrity Aware Parallelizable Mode (IAPM) [8]. He asserted that almost all encryption applications need message security; the new modes provide the additional service at a much smaller cost in performance than can be achieved when encryption and message integrity are provided separately.

Both IACBC and IAPM modes have proofs of security for both confidentiality and message integrity, assuming that the underlying block cipher algorithm is secure. The proofs of integrity are equivalent to those available for the CBC mode, and the proofs of message integrity are equivalent to those available for CBC-MAC, which is a message authentication code (MAC) based on the CBC mode. A paper containing these proofs is available at the ePrint archive at [16]. IACBC is a non-parallelizable mode that is similar to the CBC mode, except that IACBC also specifies whitening of the output blocks with a pairwise independent random sequence. Two methods for generating this random sequence are provided in the paper. An implementation of IACBC using a DES engine had a throughput of over 90 percent of the throughput of

a standard CBC implementation. Thus, the cost of message integrity is relatively small compared to the cost of supplementing the CBC mode with a separate MAC. IAPM is a parallelizable mode that specifies both input and output whitening with a pairwise independent sequence. Thus, IAPM is similar to the ECB mode in its form, but similar to CBC in its proofs of security, which are not available for the ECB mode. Although the IAPM mode has not yet been implemented, similar performance is expected for serial implementation as that achieved for the IACBC mode. [17]

## 2.2. Fast Encryption and Authentication: XCBC Encryption and XECB Authentication Modes

Virgil Gligor of VDG, Inc. presented two papers written by himself and Pompiliu Donescu. In the first paper, On Message Integrity in Symmetric Encryption, [18] different notions of message integrity for block-oriented symmetric encryption are explored, along with their relationships. These notions are expressed as a combination of integrity goals (e.g., protection against existential forgery and assurance of plaintext integrity) to be achieved in the face of different types of attacks (e.g., chosen-plaintext and ciphertext-only attacks). The integrity notions are partially ordered by a dominance relation. Defining the notions of integrity in terms of this dominance relation enables a characterization of the relative strengths of various symmetric encryption modes.

In the second paper, Fast Encryption and Authentication: XCBC and XECB Authentication Mode, two mode types are proposed: the XCBC mode and the XECB-MAC mode. These families of modes are similar to the IACBC and IAPM modes discussed in Section 2.2, but the whitening sequences are not required to be pairwise independent; this allows better performance at the cost of relaxing the security bounds that can be proven. The XCBC modes provide both confidentiality and integrity protection in a single pass over the data. These modes detect integrity violations at a low cost in performance, power and implementation, and can be executed in a parallel or pipelined

manner. The performance and security of these modes depends on the performance and security of the underlying block cipher algorithm (e.g., AES). Both stateful and stateless variants are provided. In addition to message integrity, these modes have the following properties:

- Support for real-time message authentication.
- Support for multiple encryption modes (i.e., modes other than CBC could be used).
- Support for interleaved-parallel or pipelined encryption.
- Incremental updates of encrypted data (i.e., the incremental update of data structures is possible).
- Support for architecture-dependent parallel encryption, since there is no ciphertext chaining or requirement for an a priori knowledge of the number of processors.
- Resistance to key attacks can be implemented, if required, in a manner similar to that of DESX.

The paper also provides evidence for the security of the XCBC modes against both adaptive chosen-plaintext and message-integrity attacks. The performance of the XCBC modes in software implementations is only minimally degraded in comparison to the CBC mode, and is superior to the CBC mode and other similar modes that are used to provide message integrity. The XECB-MAC modes provide message authentication, can be operated in a fully parallel or pipelined manner, and support incremental updates and out-of-order verification. These modes are intended for use either stand-alone to protect the integrity of plaintext messages, or with encryption modes that have similar properties, whenever separate secret keys are used to provide confidentiality and integrity. Both stateless and stateful variants of XECB-MAC are provided. XECB-MACs properties include:

- The XECB-MAC modes are intended to be secure against adaptive chosen-plaintext attacks.

- Parallel or pipelined operation is possible.
- The XECB-MAC modes are incremental with respect to block placement.
- Verification of the authentication code can proceed even if the blocks are received out of order.

The number of block encryption computations for XECB-MAC is the same as the number of block encryption computations for CBC-MAC. In sequential implementations, the performance of XECB-MAC is slightly lower than that of CBC-MAC because of additional processing. On the other hand, the ECB-MAC mode can take advantage of parallelism or pipelining to improve its performance. A third mode was presented: the PM-XOR mode. This mode is a stateless fully parallel mode that is similar to Jutlas IAPM mode (see Section 2.2); however, the $S_i$ elements are not pairwise independent [17].

## 2.3. OCB Mode: Parallelizable Authenticated Encryption and PMAC: A Parallelizable Message Authentication Code

Phillip Rogaway of the University of California at Davis proposed two new modes: the Offset Codebook Mode (OCB) and the Parallelizable MAC mode. The OCB mode is based on the work of Jutla (see Section 2.2) and Gligor and Donescu (see Section 2.3). This mode provides both confidentiality and integrity in a manner that is parallelizable (i.e., different blocks can be processed at the same time). Other properties of this mode include:

- The data to be processed need not be an even multiple of the block length (e.g., if AES is used, the block length is 128 bits; the data need not be forced to a multiple of 128 bits in length).

- Only two extra cipher calls beyond that needed for encryption alone are required to process the data.
- While a non-repeating nonce is required, it need not be unpredictable (e.g., a simple counter may be used).
- The offset (i.e., the whitening) values used in the OCB mode depends only on the key it only needs to be computed once at the beginning of a keys crypto period.
- Only a single key is used for this mode, as opposed to separate keys for encryption and authentication, as is done in current systems.
- Three variants of this mode are possible.

Proofs of the security properties of the OCB mode are under construction. PMAC is similar to the XECB mode proposed by Gligor and Donescu (see Section 2.3). This mode also uses an offset. The PMAC mode is fully parallelizable and achieves existential unforgeability under an adaptive chosen-plaintext attack; a proof of this security claim is currently being prepared. Other properties of this mode include:

- No nonces or random values are required.
- A minimum number of invocations of the block cipher algorithm are required: one per data block.
- The length of the data need not be a multiple of the block size of the cipher algorithm.
- Only one key is required.
- Only one invocation of the block cipher algorithm is required to compute the initial offset.
- Three variants of this mode are possible.

Rogaway indicated that the algorithm descriptions for OCB and PMAC are intended to allow for various implementation tricks [17].

# 3. OCB MODE

## 3.1. OCB Algorithm

OCB stands for offset codebook, the name gives evidence to the operational description inside the mode. One starts with a number of message blocks, these blocks are offset, and then the block cipher is applied after which the result is offset again. OCB is fully parallelizable and packs excellent features some of which follows:

- Arbitrary-length messages + minimal-length cipher-texts. Any string $M \in \{0,1\}$ can be encrypted; $|M|$ need not be a multiple of the block length n. What is more, plaintexts are not padded to strings of length a multiple of n, and thus cipher-texts are as short as possible.

- Nearly optimal number of block-cipher calls: OCB uses $[|M|/n] + 2$ block-cipher invocations. (This count does not include a block-cipher call assumed to be made during session setup.) It is possible to make do with $[|M|/n] + 1$, but such alternatives scheme would be more complex or would require a random IV. Keeping low the number of block-cipher calls is especially important when messages are short. In many domains, short messages dominate.

- Minimal requirements on nonces: Like other encryption modes, OCB requires a nonce. The nonce must be non-repeating (the entity that encrypts chooses a new nonce for every message with the only restriction that no nonce is used twice) but it does not have to be unpredictable. Requiring of a nonce only that it be non-repeating is less error prone, and often more efficient, than requiring it to be unpredictable.

- Improved offset calculations: As with [8, 18], we require a sequence of offsets. We generate these in a particularly cheap way, each offset requiring just a few machine

cycles. We avoid the use of extended-precision addition, which would introduce endian dependency and might make the scheme less attractive for dedicated hardware.

- Single underlying key: The key used for OCB is a single block-cipher key, and all block-cipher invocations are keyed by this one key, saving space and key-setup time.

The above features are achieved since they where addressed from the start to make OCB work where other modes have failed [19].

### *3.1.1. OCB security properties*

OCB is proven to be secure, in the sense of reduction-based cryptography. Specifically, its proved in terms of indistinguishability under chosen-plaintext attack and authenticity of cipher-texts as shown in [20], this combination implies indistinguishability under the strongest form of chosen-ciphertext attack (CCA) (which, in turn, is equivalent to nonmalleability under CCA. The proof of privacy assumes that the underlying block cipher is good in the sense of a pseudorandom permutation (PRP), while the proof of authenticity assumes that the block cipher is a strong PRP. The actual results are quantitative; the security analysis is in the concrete security paradigm. The proofs use standard techniques, but pushed quite far. OCB has stronger security properties than standard modes. In particular, non-malleability and indistinguishability under CCA are not achieved by CBC, or by any other standard mode, but these properties are achieved by OCB. We believe that the lack of strong security properties has been a problem for the standard modes of operation, because many users of encryption implicitly assume these properties when designing their protocols. For example, it is common to see protocols, which use symmetric encryption in order to bind together the parts of a plaintext, or which encrypt related messages as a way to do a handshake. Standard modes do not

support such practices. This fact has sometimes led practitioners to invent or select peculiar ways to encrypt (a well-known example being the use of PCBC mode. It is believed that a mode like OCB is less likely to be misused in applications because the usual abuses of privacy-only encryption become correct cryptographic techniques [19].

### *3.1.2. Notation and Basic Operations*

The following are the basic notations, which are used to describe the OCB algorithm throughout this document:

- String: is a finite sequence of symbols, each symbol being 0 or 1, where $\{0,1\}^*$ Defines the set of all strings.
- **ntz**(i) is the number of trailing 0-bits in the binary representation of i where $i \geq 1$.
- If $L \in \{0,1\}^*$ then $|L|$ denotes the length of L, in bits, while $||L||_n = max\{1, [|L|/n]\}$ denotes the length of L in n-bit blocks, where the empty string counts as one block.
- Len (i) is the integer i written in binary as an n bit string.
- $\mathbf{zpad}_n(L)$ or $L0^*$ define the number of minimum number of 0-bits padded to L to get a length of 128 bits provided $|L| \leq n$, n being the block length.
- $L \gg 1$ is a right shift of L by 1 bit.
- $L \ll 1$ is a left shift of L by 1 bit.
- If $A, B \in \{0,1\}^*$ then $A \oplus B$ is the bitwise xor of A [first$l$ bits] and B [first $l$ bits], where $l = min\{|A|, |B|\}$ so, for example, $1001 \oplus 11 = 01$.

To multiply $a \in \{0,1\}^n$ by x over the field $GF(2^n)$ where n = 128

$$a \cdot x = \begin{cases} a \ll 1 & \text{if firstbit(a)= 0} \\ (a \ll 1) \oplus 0^{120}10000111 & \text{if firstbit(a) = 1} \end{cases}$$

To multiply $a \in \{0,1\}^n$ by x over the field $GF(2^n)$ where n = 128

$$a \cdot x^{-1} = \begin{cases} a \gg 1 & \text{if firstbit(a)= 0} \\ (a \gg 1) \oplus 0^{120}1000011 & \text{if firstbit(a) = 1} \end{cases}$$

## 3.2. OCB Scheme

OCB requires the use of a block cipher and a tag length. The block cipher is an encryption function typically AES. The n value which refers to the block length must be $\leq 64$ although $n \leq 128$ is highly discouraged. The tag length is an integer $\tau \in [0..n]$. This means an adversary would be able to forge a valid ciphertext with probability $2^{-\tau}$. [2] suggests using a default tag length of $\tau = 64$. OCB-$X$ denotes OCB mode with an $X$ block cipher for AES 128 the naming becomes OCB AES128. If a tag length is specified the notation becomes OCB $[X, \tau]$. In addition to the above, the OCB mode requires the use of a nonce. The nonce need not be secret or random however its use is only limited during one single session. The session being the time period in which the encryption key is used. A good example of a nonce is a counter. The responsibility lies in the hands of the user, to not repeat the nonce during a session. If a nonce is repeated the authenticity for all future messages will be broken and the privacy of messages that used the repeated nonce will be broken too. It is the user's responsibility to communicate the nonce to the party that will decrypt but the notable thing in that feature it could be communicated in the clear. The OCB mode scheme is depicted in Figure 3.1. The OCB mode encrypts-and- authenticates a non-empty message $M\{0,1\}^*$ using $[|M|/n]+2$ block-cipher invocations. The first and last steps namely the computation of the offset R

and the tag T denotes the 2-block cipher invocations required. The in between parallel steps are determined by the size of the message.

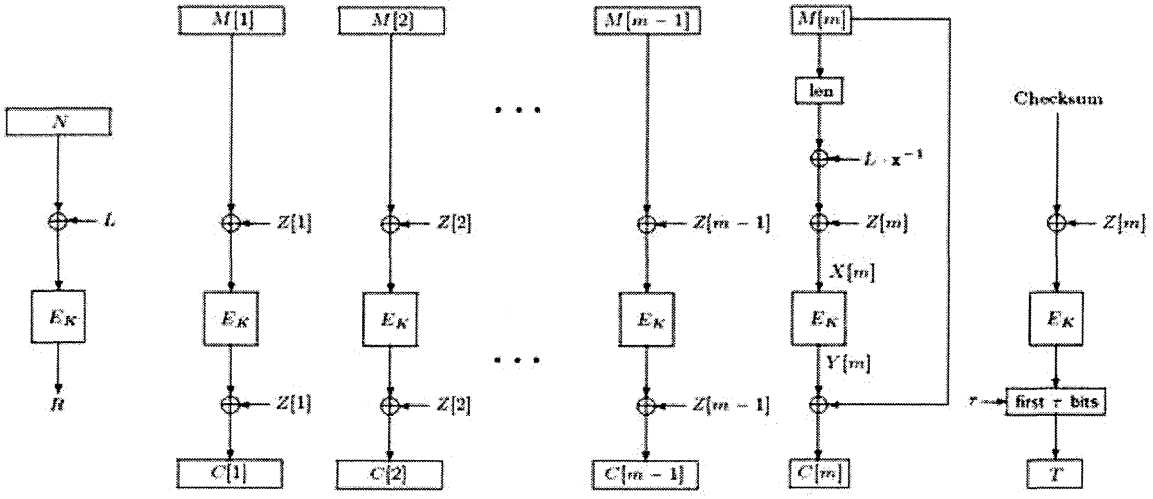


FIGURE 3.1: OCB Encryption Scheme [2].

### *3.2.1. Key setup*

The key is randomly generated and made available for both the encryption and decryption parties. For both parties the necessary key setup associated to block-cipher enciphering and deciphering is made. The following would be the operations preformed to create string values that will be used to create the offset values i.e. the $Z[1], \ldots, Z[m]$ depicted in Figure 3.1. we define the sting L (i) to be $L.x^i$ for $L \in \{0,1\}^n$ and $i \geq -1$.

Let m denote the maximum number n-bit blocks a message can have. Do the following

1. Define a string L by applying $E_k$ to a fixed string $0^n$, $L \leftarrow E_k(0^n)$
2. $L \leftarrow L(0)$
3. Let $\mu \leftarrow log_2 mfori \in [1 \ldots \mu]$, compute $L(1)..L(\mu)$, where $L(i) \leftarrow L(i) \cdot x$ is computed using a shift and a conditional xor as described in section 3.2
4. Compute $L(-1) \leftarrow Lx^{-1}$ using a shift and a conditional xor
5. Store the values $L(-1), L(0), L(1), \ldots (u)$

OCB makes use of the canonical Gray code $\gamma = \gamma^n$ constructed by $\gamma^1 = (01)$ and for $l \geq 0$. Where for $i = 0$ we have $\gamma_i \cdot L = 1 \cdot L = L$ and for $i \geq 2$ we have $y_i \cdot L = (y_{i-1} \cdot L) \oplus L(ntz(i))$

Example:

$\gamma_2 \cdot L = (\gamma_{2-1} \cdot L) \oplus (L(ntz(2))) =?$

We know $(\gamma_{2-1} \cdot L) = (\gamma_1 \cdot L) = 1 \cdot L = L$ and ntz (2) is the number of trailing zeros in the binary representation of 2 which is 1. Therefore, we have $\gamma_2 \cdot L = L \oplus L(1)$ so that means the ith word is obtained by xoring the previous word with $L(ntz(i))$.

### *3.2.2. OCB encryption*

To encrypt a message $M \in \{0,1\}^*$ with a key K and Nonce $N \in \{0,1\}^n$ obtaining a ciphertext C, follow the below algorithm [2].

Algorithm OCB.$Enc_K$ (N,M)

Partition M into $M[1]\ldots M[m]$

1. $L \leftarrow E_K(0^n)$
2. $R \leftarrow E_K(N \oplus L)$
3. for $i \leftarrow 1$ to m do $Z[i] = \gamma_i \cdot L \oplus R$
4. for $i \leftarrow 1$ to $m-1$ do $C[i] \leftarrow E_K(M[i] \oplus Z[i]) \oplus Z[i]$
5. $X[m] \leftarrow len(M[m]) \oplus L \cdot x^{-1} \oplus Z[m]$
6. $Y[m] \leftarrow E_K(X[m])$
7. $C[m] \leftarrow Y[m] \oplus M[m]$
8. $C \leftarrow C[1]\ldots C[m]$
9. $Checksum \leftarrow M[1] \oplus \ldots \oplus M[m-1] \oplus C[m]0^* \oplus Y[m]$
10. $T \leftarrow E_K(Checksum \oplus Z[m])$ [first $\tau$ bits]
11. $return \leftarrow C||T$

The algorithm refers to Figure 3.1 in terms of the procedure. First the message M is partioned into m blocks where $m = max\{1, [|M|/n]\}$ and n is the block size in bits. The string value L is computed by encrypting 0 n bits using the block cipher of choice in the OCB mode. The offset R is then computed by xoring the nonce N with the string value L. Offset $Z[1] = L \oplus R$ and for any $i \geq 2, Z[i] = Z[i-1] \oplus L(ntz(i))$. By $C[m]0^*$ we mean $C[m]$ padded with zero bits to the right to get to length $n$.

### *3.2.3. OCB decryption*

To decrypt a message C$\{0,1\}^*$ with a key K and Nonce $N \in \{0,1\}^n$ obtaining a plaintext M do the reverse process of section 3.3.1, making sure that the presented Tag is as expected (if not, regard the presented ciphertext as invalid). The below algorithm [2] depicts the decryption process.

Algorithm OCB.$Dec_K$ (N,M)

Partition C into $C[1] \ldots C[m]$

1. $L \leftarrow E_K(0^n)$
2. $R \leftarrow E_K(N \oplus L)$
3. for $i \leftarrow 1$ to m do $Z[i] = \gamma_i \cdot L \oplus R$
4. for $i \leftarrow 1$ to $m-1$ do $M[i] \leftarrow E_K^{-1}(M[i] \oplus Z[i]) \oplus Z[i])$
5. $X[m] \leftarrow len(M[m]) \oplus L \cdot x^{-1} \oplus Z[m]$
6. $Y[m] \leftarrow E_K(X[m])$
7. $M[m] \leftarrow Y[m] \oplus C[m]$
8. $M \leftarrow M[1] \ldots M[m]$
9. $Checksum \leftarrow M[1] \oplus \ldots \oplus M[m-1] \oplus C[m]0^* \oplus Y[m]$
10. $T \leftarrow E_K(Checksum \oplus Z[m])$ [first $\tau$ bits]
11. if T = T" then return M else return INVALID

### OCB properties

OCB was designed to address certain properties, which makes it one of the most efficient authenticated-encryption schemes. These properties seem to be what many such attempts lacked in creating a high performance authenticated encryption. Some properties of OCB taken from [2] are listed below. The summary of these properties are depicted in table 3.1.

### Arbitrary-length messages and no ciphertext expansion

Any message M can be encrypted, yielding a ciphertext C of length $|M| + \tau$. That is, the length of the ciphertext namely the portion $C = C[1] \ldots C[m]$ of the ciphertext that excludes the tagis the same as the length of the message M. This is better, by up to n bits, than what one gets with conventional padding.

### Single block-cipher key

OCB makes use of just one block-cipher key, K. Thus only one block-cipher key needs to be setup, saving on storage space and key-setup time.

### Weak nonce requirements

Modes of operation that requires a random IV are error-prone. As an example, consider CBC mode, where $C[i] = E_K(M[i] \oplus C[i-1])$ and $C[0] = IV$. It is sometimes suggested that a mode, which needs a random IV, is preferable to one that needs a nonce. First, a random value of sufficient length can always be used as a nonce, but a nonce cannot be used as a random value. Second, the manner in which systems provide random IVs is invariably stateful anyway: unpredictable bits are too expensive to harvest for each IV, so one does this rarely, using state to generate pseudorandom bits from unpredictable bits harvested before. Third, the way to generate pseudorandom bits needs to use cryptography, so the prevalence of non-cryptographic pseudorandom number generators routinely results in implementation errors. Next, nonce-based schemes make it possible for the receiver to implement replay-detection with no added cryptography. Finally, nonces can be communicated using fewer bits, without any additional cryptography.

### On-line

OCB encryption and decryption are on line in the sense that one does not need to know the length of the message in advance of encrypting or decrypting it. Instead, messages can be processed as one goes along, using constant memory, continuing until there is an indication that the message is over. An incremental interface (in the style popular for cryptographic hash functions) would be used to support this functionality.

| | |
|---|---|
| Security Function | Authenticated encryption. Provides both privacy and authenticity. It achieves a strong form of privacy: what cryptographers call "indistinguishability under chosen-ciphertext attack" and "non-malleability under chosen-ciphertext attacks". These strong properties make OCB easier to correctly use in protocols than standard privacy modes. |
| Error Propagation | If the ciphertext is corrupted in any manner then the received ciphertext will almost certainly (probability 1 - 2-t) be rejected. |
| Synchronization | Optional. If the nonce N is transmitted along with each ciphertext, there are no synchronization requirements. If it is not sent (to save transmission bits) the receiver must maintain the corresponding value. |
| Parallelizability | Fully parallelizable. Both encryption and decryption are fully parallelizable. Thus it will have ever faster implementations as machines offer up more and more parallelism, and it is good for encrypting messages in hardware at the highest network speeds. |
| Keying Material | One block-cipher key. One needs a single key, K, which keys all invocations of the underlying block cipher. |
| Ctr/IV/Nonce Requirements | Single-use nonce. The encrypting party must supply a new nonce with each message it encrypts. The nonce need not be unpredictable or secret. The nonce is n bits long (but it would typically be communicated using fewer bits, as determined by the application). |
| Memory Requirements | Any bit string allowed. Any string M 0, 1 may be encrypted, including the empty string and strings which are not an integral number of bytes. The length of the string does need not be known in advance. |
| Ciphertext Expansion | Minimal possible (for a scheme meeting the desired privacy notion). Expansion is 0n bits for the tag plus 0n bits for the nonce. The former depends on a user-specified parameter t, with 3280 bits being typical. Messages, which are not a multiple of the block size, do not receive additional expansion due to padding. |
| Other Characteristics | Efficiency: Uses [— M — /n] + 2 block-cipher calls and very efficient offset calculations. Endian neutrality: Can be implemented equally efficiently on big-endian and little-endian machines. Provable security: The mode provably meets its goals, assuming the underlying block cipher meets now-standard cryptographic assumptions. |

TABLE 3.1: Summary of OCB properties [2].

# 4. IMPLEMENTATION CRITERIA

## 4.1. Proof of Concept

The goal of this thesis is to develop a functional OCB-AES encryption engine in hardware. The OCB mode requires a block cipher mode, and AES 128 is the candidate chosen. Furthermore to fully produce a high performance and low cost implementation it becomes then necessary to also implement a fast Rijnadel-AES engine for that purpose. OCB is a fairly new algorithm and the only available implementation is written in C code, which is for illustration purposes only. I have yet to come across an implementation of OCB in hardware for either ASIC or FPGAs, this to me seems to be the first attempt in doing so.

## 4.2. OCB Implementation Considerations

With network speeds on the rise, pushing the envelope so high that we are currently witnessing many products operating at speeds of 10-gigabits and much faster ones under development, it becomes inevitable to create a standard for high-speed authenticated encryption. Encrypting data at such high speeds in unachievable in current personal computers and requires the use of hardware accelerators. However current cryptographic standards do not provide methods to secure traffic at such high rates. Many standard block cipher modes have fundamental performance limitations and require the use of message authentication codes to achieve authenticity goals. In recognition of this OCB was designed as a candidate of high performance authenticated encryption that could be used to secure traffic at such high speeds. The features in OCB make it the most prominent amongst its caliber. It utilizes high parallelism, which translates to higher speeds when allowed by the environment. Pipelining is also a very important feature

that could give rise to higher performance. We present certain criteria that provides for a high performance authenticated-encryption mode.

### *4.2.1. Parallelism*

Parallelism can be found in the encryption algorithm when certain transformations can be preformed simultaneously. It can also be exploited outside the algorithm by augmenting a number of processing units to work simultaneously on independent blocks. Parallelizability is important for obtaining the highest speeds from special-purpose hardware, and it may become useful on commodity processors. For special-purpose hardware, one may want to encrypt-and-authenticate at speeds near 10 Gbits/second an impossible task, with todays technology, for modes like CBC encryption and the CBC MAC. (One could always create a mode that interleaves message blocks fed into separate CBC encryption or CBC MAC calculations, but that would be a new mode, and one with many drawbacks. For commodity processors, there is an architectural trend towards highly pipelined machines with multiple instruction pipes and lots of registers. Optimally exploiting such features necessitates algorithms with plenty to do in parallel [2].

### *4.2.2. Pipelining*

Pipelining is critical to efficiency at high speeds. OCB gives the freedom of choice to use any block cipher mode. However AES is highly encouraged. A pipelined implementation of AES consists of ten separate rounds. At each clock cycle, plaintext data enters the first round, the intermediate data moves from one round to the next, and ciphertext data leaves the final round. While it takes ten clock cycles to encrypt any given plaintext, the circuit completes one encryption per clock, once the pipeline is full. In a mode of operation that requires block chaining (such as CBC, CBC-MAC, OFB, and CFB), each block of input to AES depends on the previous block of output. These

modes cannot be pipelined, and thus suffer either a factor of ten reduction of speed or a similar increase in circuit size [21].

### *4.2.3. Simplicity*

Simplicity has been a central design goal. Some of OCB's characteristics that contribute to simplicity from [2] are :

- Short and full final-message-blocks are handled without making a special case: the treatment of all messages is uniform, regardless of their length.
- Only the simplest form of padding is used: append a minimal number of 0-bits to make a string whose length is a multiple of n. This method is computationally fastest and helps avoid a proliferation of cases in the analysis.
- Only one algebraic structure is used throughout the algorithm: the finite field $GF(2^n)$.
- In forming the sequence of offsets, Gray-code coefficients are taken monotonically, starting at 1 and stopping at m. One never goes back to some earlier offset, uses a peculiar starting point, or forms more offsets than there are blocks.

### *4.2.4. Efficiency*

A significant criterion that has been neglected in many authenticated encryption modes but was addressed from the start in OCB is the number of Block-cipher invocations. Reducing this number increases the efficiency of the whole algorithm. It may not seem much to shave off a few block-cipher invocations making for shorter cipher-texts, but most of many cryptographic applications deal with short messages. Roughly a third of the messages on the Internet backbone are 43 bytes big [2]. Encrypting messages of

such length makes it even trickier to deal with message expansion and computational overhead, since comparatively the inefficiencies could be large. Another efficacy measure is the circuit depth of an encryption scheme as measured in terms of blockcipher gates. For OCB encryption, this number is three: a call to form R; calls to form the ciphertext core; and a call to compute the tag. Block-cipher circuit-depth serves as a lower bound for latency in an aggressively parallel environment. Reducing the block-cipher circuit-depth to one or two is possible, but the benefit does not seem worth the associated drawbacks. Depending on padding conventions and the optional processing done to the final block in order to ensure security across messages of varying lengths. So the total will be as few as 2 $[|M|/n] + 1$ or as many as 2 $[(|M| + 1)/n] + 4$ block-cipher calls. Thus OCB saves between $[|M|/n] - 1$ and $[|M|/n] + 3$ block-cipher calls compared to separate CBC encryption and CBC MAC computation. As with any mode, there is overhead beyond the block-cipher calls. Per block, this overhead is about four n-bit xor operations, plus associated logic. The work for this associated logic will vary according to whether or not one precomputed $L(i)$-values and many additional details [2].

### *4.2.5. Other Factors*

There are other important goals for an authenticated encryption mode suitable for use at high data rates. It should be possible to use the same pipeline to process successive data packets without stalling. There should be a minimal circuit depth to the algorithm, so that the per-packet pipeline stall is minimal or nonexistent. Of course, the algorithm should also be implemental in as small a circuit as possible. The overhead of the message authentication component should be small relative AES counter mode. Additionally, while hardware requirements are paramount for high-speed cryptography, the need for interoperability dictates that software performance should also be good [21].

## 4.3. Hardware vs. Software Implementations

Cryptographic applications or products can be implemented in both hardware and software. However the determining factor is the desired speed and cost of the encryption/decryption implementations.

Software implementations have become wide spread, thats due to its accessibility, ease of use and low cost. It is also not labor consuming, since people could achieve correct implementations with basic skills in programming. Most software implementations are done in high programming languages such as $C$, $C^{++}$ and Java and are made to operate on personal computers or smart cards. Software implementations are ideal for low power, speed and security requirements. However in order to achieve higher speeds the use of hardware implementations is inevitable.

Hardware implementations are usually designed and coded using hardware description languages, such as VHDL and Verilog or using schematic capture. There are currently two major implementation approaches for hardware designs, one being Application Specific Integrated Circuits (ASIC) and Field Programmable Gate Arrays (FPGA).

Application Specific Integrated Circuits (ASIC)

- This approach requires the design from the behavioral description to the physical layout
- Cost and time consuming
- High manpower

Field Programmable Gate Arrays (FPGA)

- Reconfigurable to perform different functions
- Comparatively cheap since they are purchased off-the-shelf
- Not optimal and slower then ASIC

Every type of implementation has its advantages and disadvantages. Their basic features are summarized in table 4.1 [4].

## 4.4. OCB Applications

The IEEE is actually defining a brand new encapsulation protocol. This new protocol is expected to use a stronger cipher the Advanced Encryption Standard (AES) in Offset Codebook (OCB) mode. It has been stated that OCB-AES does not have the weaknesses that RC4, the current WEP encryption, has.This could mean that OCB will be able to provide the industrial-strength data integrity and privacy for 802.11 wirelesses. AES-OCB is touted as being much stronger than WEP/TKIP [22].

Other applications of the OCB mode include Internet security many popular Internet protocols rely on authenticated encryption schemes for privacy and authenticity. Examples of these include: SSL, TLS, SSH, IPSEC. Other applications on the Internet require both privacy and integrity examples of these include: online banking, retail, and auctions, secure file transfer.

| | ASICs | FPGAs | Software |
|---|---|---|---|
| Speed | Very fast | Fast | Moderately fast |
| Design Cost | Very expensive | Moderately expensive | Inexpensive |
| Design Cycle | Long | Moderately long | Short |
| Design Tools | Very expensive | Inexpensive | Inexpensive |
| Maintenance and Upgrades | Expensive | Inexpensive | Inexpensive |
| Tamper Resistance | Strong | Limited | Weak |
| Key Protection | Strong | Limited | Weak |
| Algorithm Agility | No | Yes | Yes |

TABLE 4.1: Characteristic features of implementations of cryptographic transformations in ASICs, FPGAs and software [4].

# 5. IMPLEMENTATION RESULLTS AND ANALYSIS

This Chapter describes the design process followed in order to achieve the hardware realization of the OCB-AES mode. The architecture and logic design are described in detail presenting the functional blocks, and the results of our findings.

## 5.1. Design methodology

The Algorithmic description of OCB [2] served as the basis of our hardware design. It was thoroughly analyzed in an attempt to exploit the features in it. The algorithm was divided into portions and each tackled separately, considering many data path options and settling on the efficient routes foreseen. One important feature was the use of AES as our block cipher. It was then necessary to do a literature review on Rijndael AES and find the most efficient way to implement it, since its performance will be vital to the whole design. Once the architecture was specified the components of the design where described in VHDL code. The design was simulated in ModelSim for Functional correctness. Later the design was synthesized targeting Virtex 2 Pro-ff896 family FPGA and an ASIC ammi05_typ auto ($0.5\mu$ CMOS with hierarchy presrved) as the target technologies.

## 5.2. Overall Architectural Design

The top-level architectural design of the OCB-AES encryption engine is shown in Figure 5.1. The implemented version is depicted in Figure 5.2. The main functional blocks are Generate_offset & Offset_initialization, cipher_computation and two other control blocks.

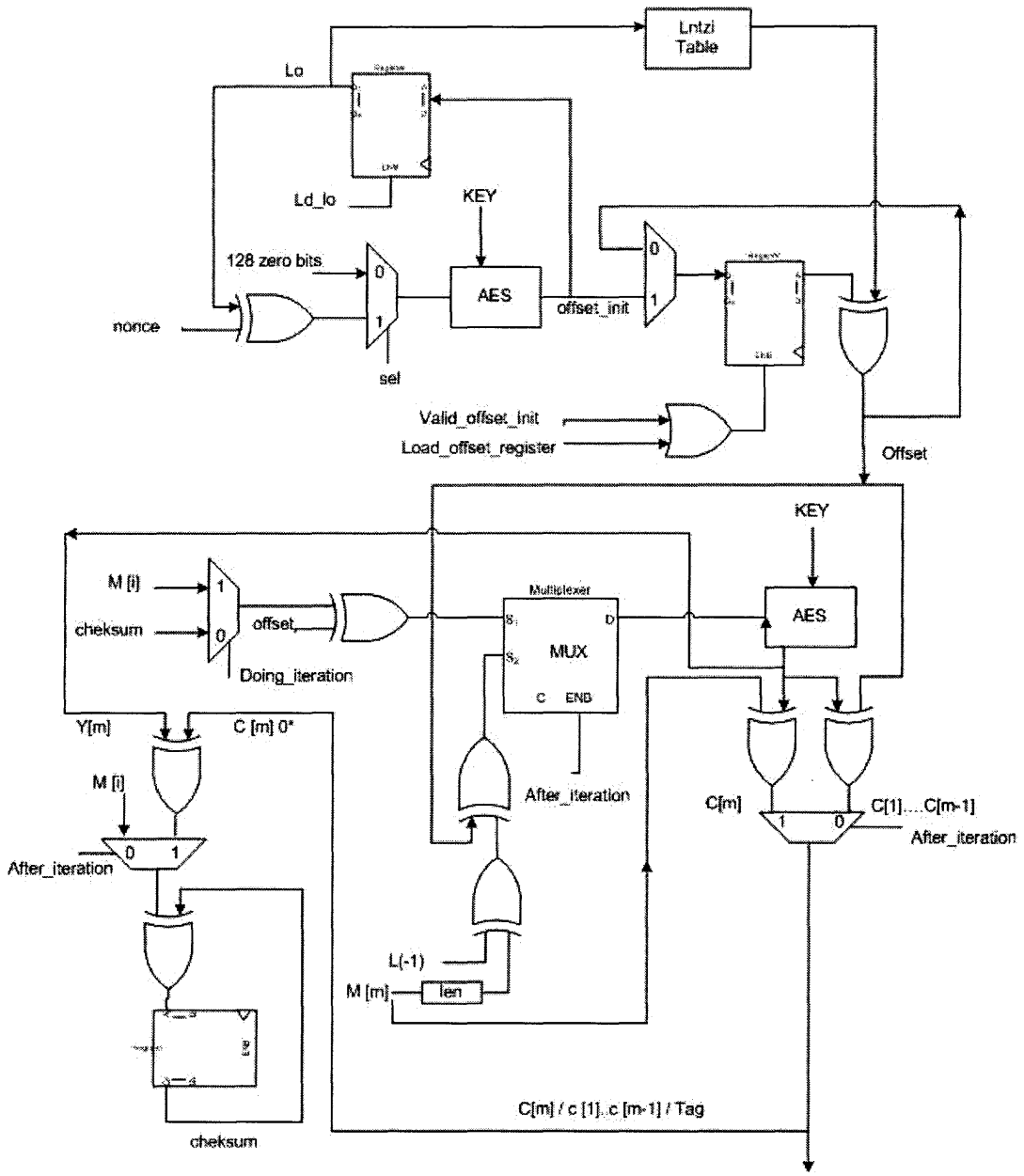


FIGURE 5.1: System Level Diagram of OCB-AES Encryption Engine

The AES block in Figure 5.1 refers to the Rijndael encryption engine. There are three major inputs: M[i] which is a 128-bit block of the plaintext, nonce, and the key.

The nonce is xored with a stride L and then is fed through a mux into the AES engine to produce the initial offset. Since the computation of the initial offset would be done just once for that encryption session, it seems wasteful to dedicate one AES block just for that purpose, since it adds to the area unnecessarily. The L(i) values required to produce the intermediate offsets for the remaining blocks are pre-computed and saved in a table. This is done by a shift register and an xor. The values then are saved to the ntzi table. This implementation alleviates the need to compute L by using block ciphers lots of different times. Once the offset is generated it passes through the section where the cipher texts are computed. Here another instance of the AES engine is implemented. The key along with the offset and the message block are processed to produce the cipher text block for that message. The last block of the plaintext is computed once a signal indicating that the block before the last in the message was done. Once the last block is processed it is used to compute the checksum. The final round is when the checksum passes through the AES engine to produce the Tag. As you can observe the design uses minimum instances of the AES block, hence reducing area dramatically. Further more the checksum could not be produced until the value C[m] (last 128-bit cipher block) is computed. This means a parallel structure dedicating a sole AES engine for that purpose would be deemed unresourceful.

The interface of the overall design is as follows:

| | |
|---|---|
| Input: | clock, reset_n |
| Control: | encrypt, no_block_minus1 |
| Data: | Plain_text, Key, nonce |
| Output: | Cipher_text, done |

An IO & Memory block provides the interface between the user and the memory elements for the operands and the sparsing of the message. I did provide a $C^{++}$ software code as an interface. For hardware the requirement would be to meet the timing specifications of the Generate_offset & Offset_initialization block. There are many different

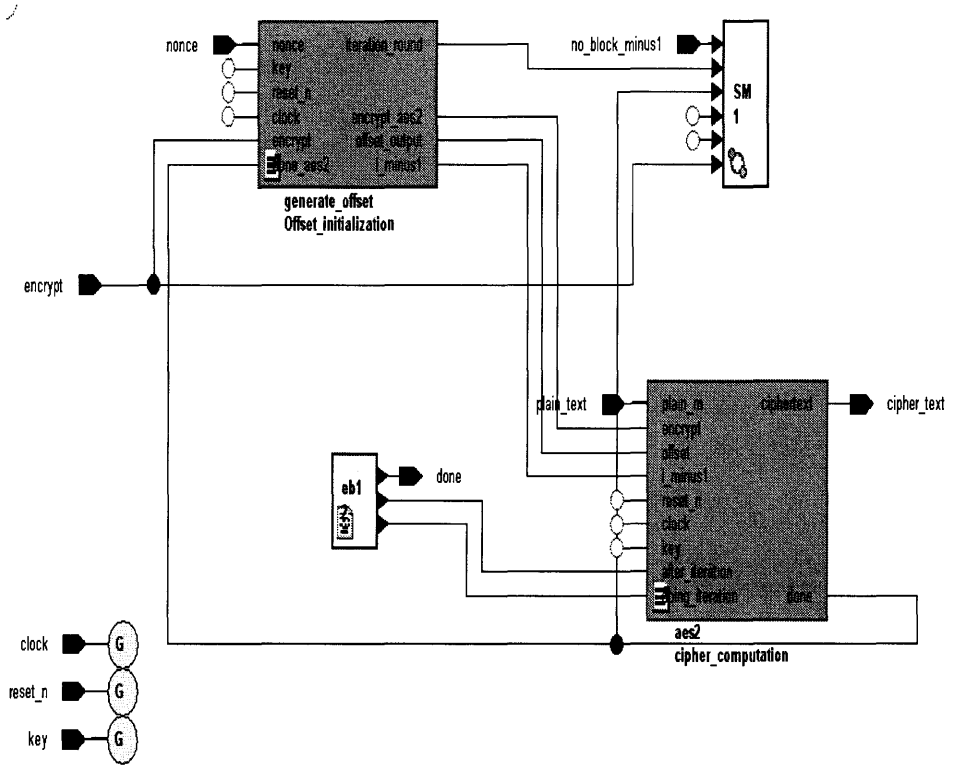


FIGURE 5.2: Implemented Block Diagram of OCB-AES Encryption Engine

flexible solutions to implement this block, depending on the target system requirements in which it will be integrated. Therefore, the architecture of this functional unit is out of scope of this work. The input takes in serial data, that are passed through a counter. The counter then outputs the iteration round value which in turn is passed to the sm1 block where it is compared against the value of no_block_minus1 (the block before the last) if the iteration round is smaller or equal to the no_block_minus1 value, it triggers a signal doing_iteration = 1, that is it keeps processing more rounds until the value of the counter reaches the last block, then it triggers a signal after_iteration =1 indicating the phase of encrypting the last block of data. We also use an active low asynchronous reset.

After the computation of the L strings and the offset values they are passed to the cipher_compuation block, responsible for creating the C[1] ... C[m] values of the cipher

text and computing the tag. A data block of 128 bits can be processed every 13-clock cycles since that is the number of cycles that the AES encryption engine can operate at. The total operation takes 26 clock cycles once we start encryption, we have 26 because we need first 13 clock cycles for computing L[0] and pre-computing other values of L, and the 13 clock cycles is for computation of offset_initialization. The cipher computation for each block from there on takes 13 clock cycles. The number of clock cycles to encrypt m blocks = $(3 + m) \times 13$.

The data path of the overall design is mainly split into two sub-blocks one is generate_offset and other is AES 2. The control blocks are also divided further. We present an account of the function of each of these blocks in detail.

## 5.3. Control Blocks

Sm1 and Eb1 depicted in Figure 5.2 are small control blocks, Eb1 is a state machine which makes a transition from state and indicates generation of cipher (C[1],..., C[m-1]), cipher_ last(C[m]), tag followed by a done state. It also generates the three control signals: doing_iteration when in state cipher, after_interation when in state cipher_last and done in state done. Sm1 on the other hand handles the control signals for the Rijndael system. Sm1 and eb1 combined together forms the control block on higher level internally we split each block (offset_initaliztion and AES2 block) and they have their own control section. So in this regards our control structure is distributed rather than centralized. It is worth mentioning that the generation of the no_block_minus_1 value is done via a software interface that we implemented in $C^{++}$ code.

## 5.4. Offset Initialization and Generation Architecture

This functional design of this block is to generate the offset values to be used for offsetting the message blocks before and after encryption. The design is further

subdivided into blocks or tasks. The first task is to generate the initial offset from the nonce this is the offset_intialization sub-block. The other sub-block is the generation of the L string values. The detailed description of these sub-blocks follows.

### *5.4.1. The Offset Initialization Block*

This block computes initial value of offset in-order to start the encryption. This block also computes the intermediate values of offset based on iteration level. Figure 5.3 displays the architectural design of this unit.

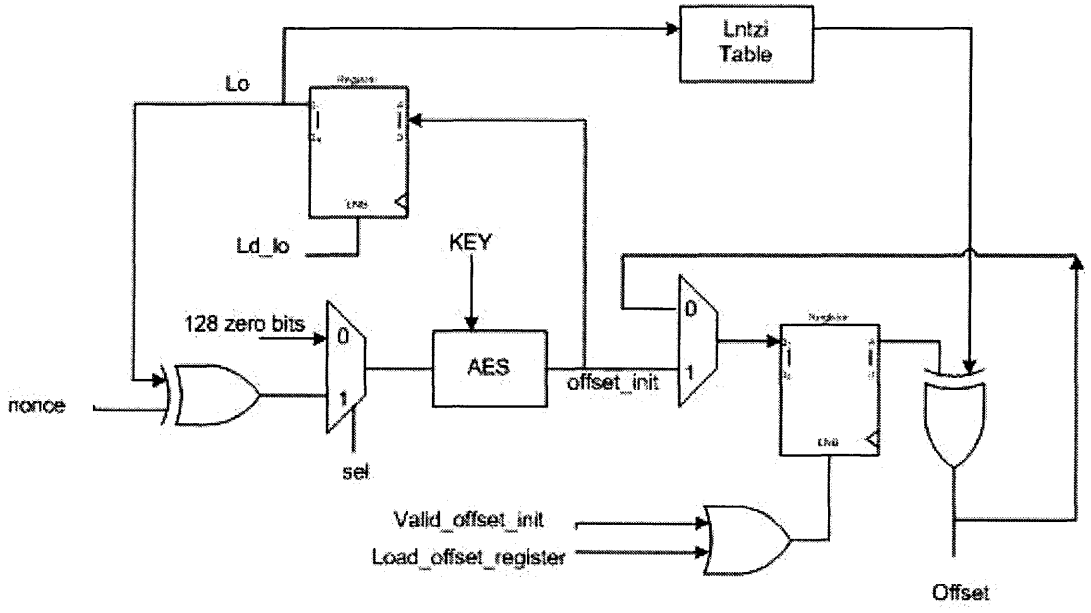


FIGURE 5.3: Architectural Design of the Offset_initialization Block

### *5.4.2. The L_generate Block*

This block from the value of L [0], pre-computes all the values of L i.e. L[1], L[2], ..., L[9]. We are limiting the number of block to $m = 2^{10} - 1$ (based on assumption that each file is of size less than 10 MB) Hence we compute only until L [9]. Also based on iteration round which is computed using the increment counter, which increments its output whenever the input to its incr signal goes high. This iteration count input is then passed on through a 10 to 4 bit priority encoder whose functional operation is as depicted in Table 5.1.

| Input bit position | Output In decimal | Output in binary bit position | L[ntzi] |
|---|---|---|---|
| 9 8 7 6 5 4 3 2 1 0 | | 3 2 1 0 | |
| 0 0 0 0 0 0 0 0 0 0 | 0 | 0 0 0 0 | L[0] |
| x x x x x x x x x 1 | 0 | 0 0 0 0 | L[0] |
| x x x x x x x x 1 0 | 1 | 0 0 0 1 | L[1] |
| x x x x x x x 1 0 0 | 2 | 0 0 1 0 | L[2] |
| x x x x x x 1 0 0 0 | 3 | 0 0 1 1 | L[3] |
| x x x x x 1 0 0 0 0 | 4 | 0 1 0 0 | L[4] |
| x x x x 1 0 0 0 0 0 | 5 | 0 1 0 1 | L[5] |
| x x x 1 0 0 0 0 0 0 | 6 | 0 1 1 0 | L[6] |
| x x 1 0 0 0 0 0 0 0 | 7 | 0 1 1 1 | L[7] |
| x 1 0 0 0 0 0 0 0 0 | 8 | 1 0 0 0 | L[8] |
| 1 0 0 0 0 0 0 0 0 0 | 9 | 1 0 0 1 | L[9] |

TABLE 5.1: Functional Operation of Priority Encoder

Since we cannot have a 0th iteration the condition where the counter's output is all zeros is insignificant, hence that one selection does not really effect the design. The incr signal goes high whenever aes2 produces a done signal. Figure 5.4 depicts the L & ntzi computational block. Shown in the figure, 3 blocks, namely the priority encoder we just described along with the incr-counter and the ntzi generator.

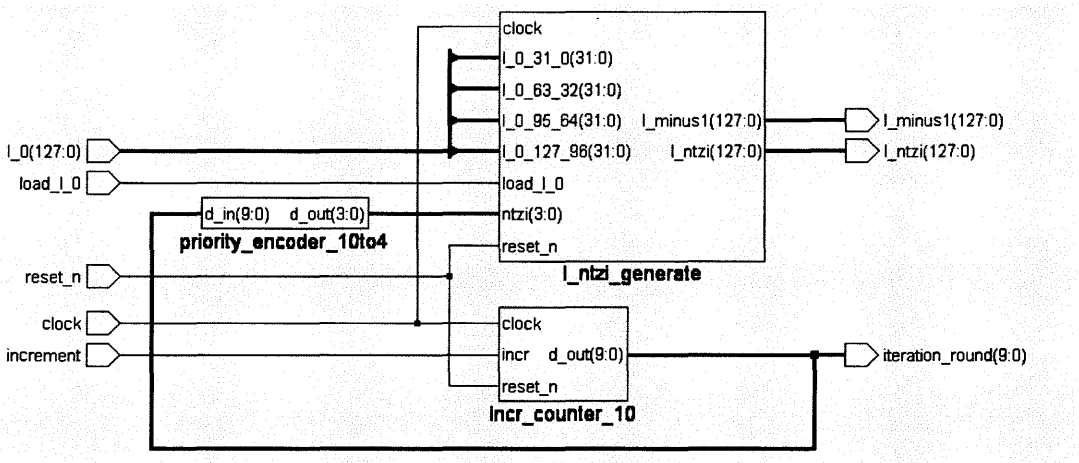


FIGURE 5.4: L & ntzi Computational Block

We compute L [i + 1] using the following function:

$$Outputl\_plus1 = \begin{cases} \ll L(i)(Left - shift) & \text{when the MSB of input is zero} \\ \ll L(i) & \text{XOR with MSB of 87 otherwise} \end{cases}$$

The above function is translated using the RTL schematic depcited in Figure 5.5. The XOR gate here is nothing but a MUX translation of the above function.

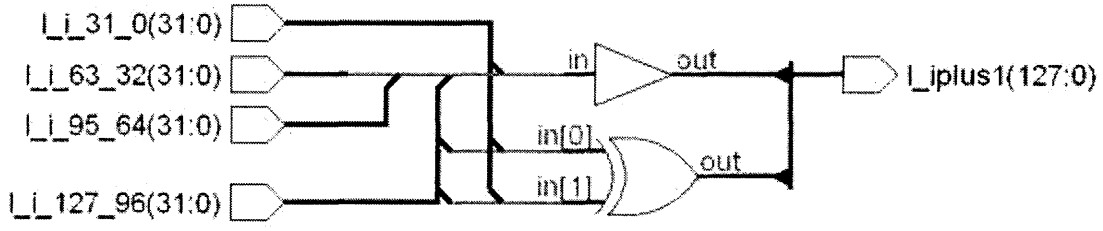


FIGURE 5.5: RTL Schematic of the I_plus1 Output

L_minus1 is computed using the following binary operation:

$l\_minus1(127DOWNTO7) \Leftarrow l\_0(0)|\&l\_0(127DOWNTO8)$

$l\_minus1(6DOWNTO0) \Leftarrow |\_0(7DOWNTO1)WHEN(l\_0(0) =' 0')ELSE$

$l\_0(7DOWNTO1)XOR$"1000011";

$$Outputl\_minus1 = \begin{cases} \gg L(0)(Right - shift) & \text{when the LSB of L(0) is zero} \\ \gg L(i)) & \text{NOT L(MSB) and L(0) XOR LSB\_43 otherwise} \end{cases}$$

## 5.5. Cipher Computation Architecture

The functional design of this block is to produce the ciphertexts and the tag value from the plaintext and the offset values. This design includes an implementation of a Rijndael AES core engine and some xors and muxes to produce the required results.

### *5.5.1. AES Engine*

The architectural strucure of the Rijndael AES engine is depicted in Figure 5.6. The design features a more serial approach by cloning only necessary units. Many criteria where taken into consideration when developing this Implementation. The key generation was produced in parallel with the encryption round, rather than recomposing the keys in advance this method provides better efficiency and saves a vast amount of buffer space. One set of round key is computed per round.

A module containing S-Box, ShiftRow, MixColumn, and AddRoundKey was designed and the same components were reused each round. All the necessary computations required are computed in only one cycle. The main operations preformed in our AES encryption engine implementation are as follows:

- S-Box : Designed as a lookup table with 8-bit address input and 8-bit output. 16 copies of S-Boxes are duplicated to meet the one clock cycle specification.

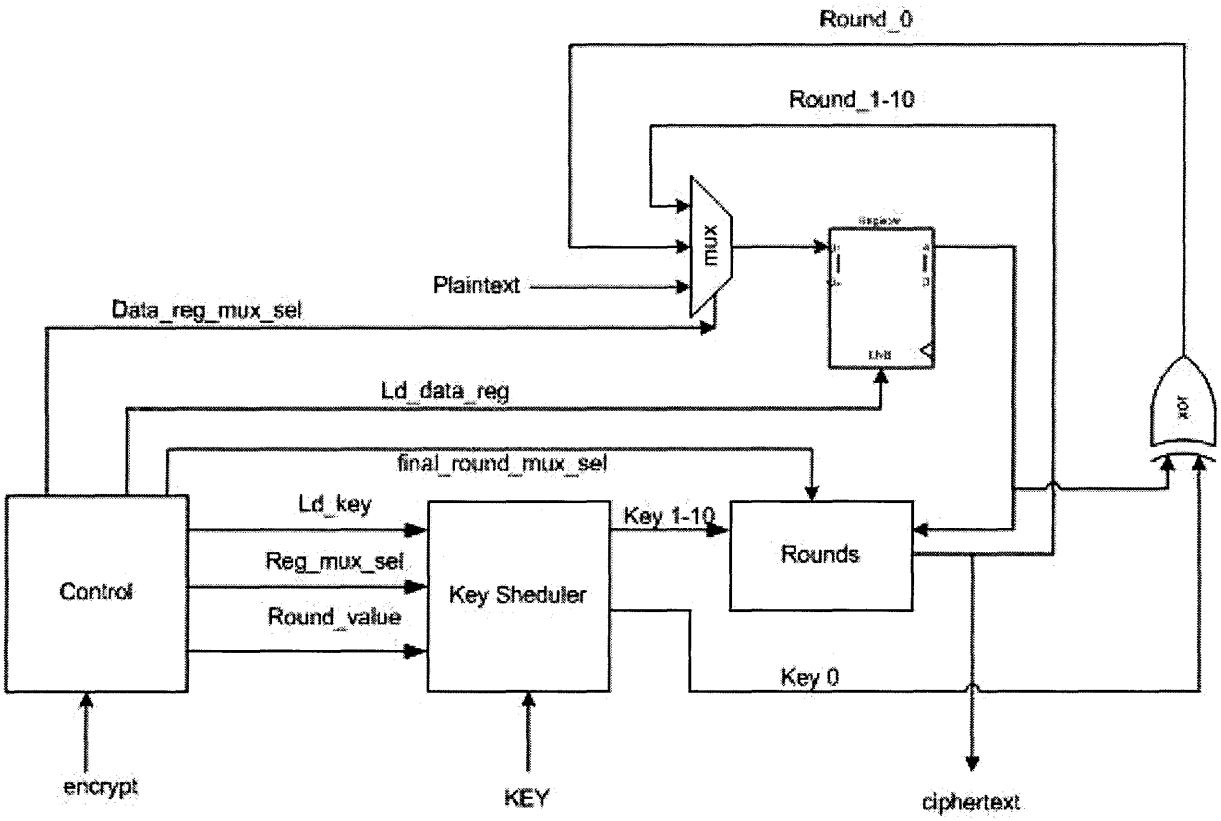


FIGURE 5.6: A top level system design of the AES engine

- Shift Row : Using a shift register
- Mix Column : This is implemented by simple shift and/or xor operations. For results of more than 8-bits, irreducible polynomial 100011011 is used to reduce the result.
- Key Schedule : Four S-Boxes are duplicated such that computations for each round complete in one clock cycle. The next round's sub-key can be derived from buffering each sequence of sub-keys
- Add Round Key : Composed of xor gates.

- Control : A finite state machine, which provides register loads and multiplexer select signals. Moreover, round constants are stored in a lookup table and provided to the key schedule block.

There are a total of 13 rounds, out of which 3 rounds are for initialization, loading inputs and output generation. The architectural process of one round of AES encryption is depicted in Figure 5.7. Each round takes one clock cycle to execute. Making the total of cycles from when we start encryption till we get the results 13 clock cycles.

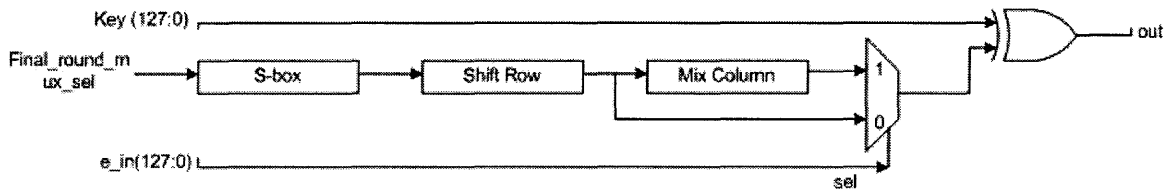


FIGURE 5.7: System Level Design of an AES Round

## *5.5.2. The Cipher Computation Block*

This block is responsible for computing the ciphertexts and the tag value. AES 1 refers to the instance of the Rijndael encryption engine used to produce the offset values. Whereas aes2 refers to an instance of the same engine used to compute all cipher text blocks along with the tag. Figure 5.8 depicts the architectural structure of aes2 where as 5.9 depicts the implemented Logic diagram of the aes2 engine.

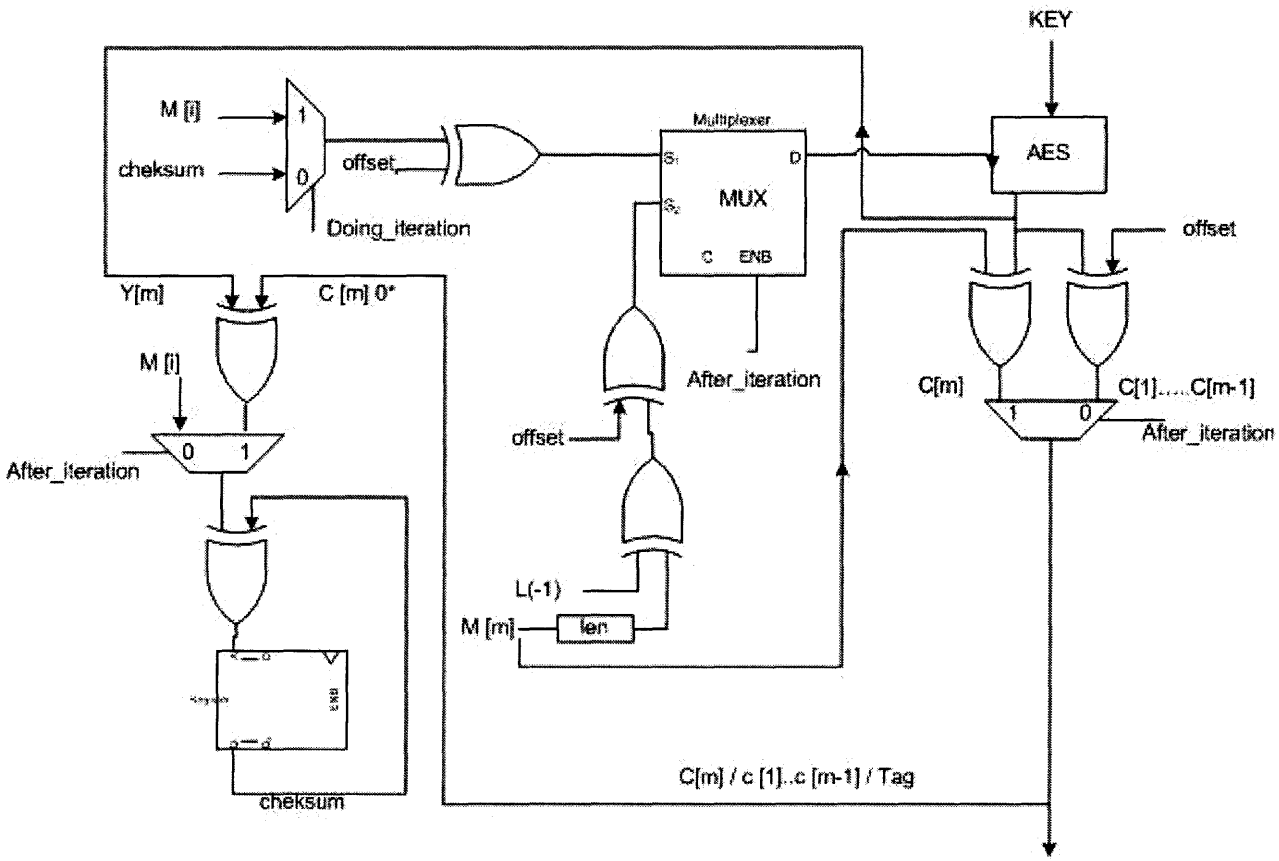


FIGURE 5.8: Architectural Design of The aes2 Engine

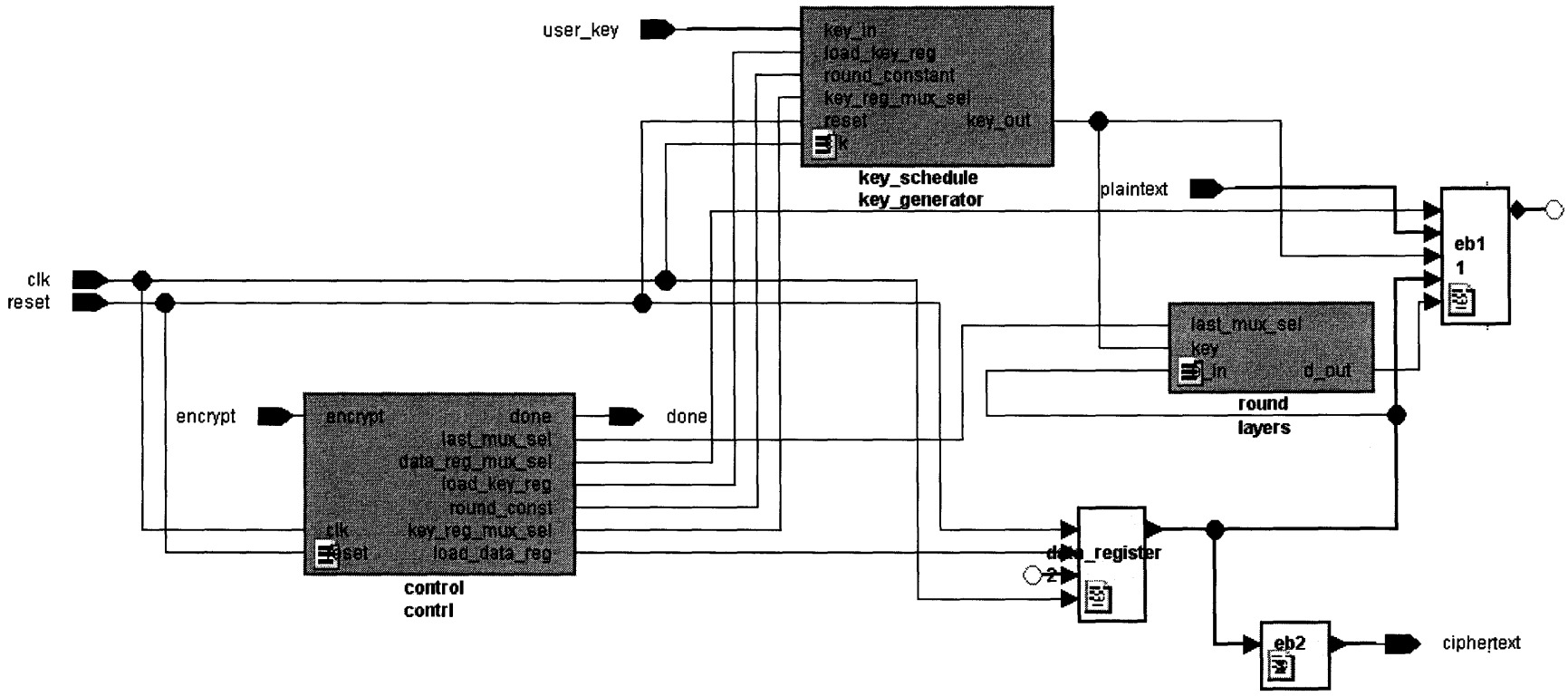


FIGURE 5.9: Implemented Block Diagram of OCB-AES Encryption Engine

## 5.6. Synthesis and Timing Results

The designs where synthesized in two environments. The first using the Xilinx package software the second, is that for the ASIC using Leonardo synthesis tool. It is worth mentioning that latter synthesis environment is intended for educational purposes and do not reflect the same environment for commercial ASICs. However it provides a reasonable environment for comparison between designs.

### *5.6.1. Area and Timing results for FPGA & ASIC Implementations*

Table 5.2 depicts the synthesis results for the Xilinx target technology. The first section describes the area usage of the chip. The bottom part gives the maximum clock frequency that our OCB-AES engine can operate at. The same results where obtained for the 0.5 u CMOS ASIC desgin after synthesis.These values are provided in Table 5.3.

| Device Utilization for 2VPX20ff896 | | | |
|---|---|---|---|
| Resource | Used | Avail | Utilization |
| IOs | 524 | 556 | 94.24% |
| Global Buffers | 2 | 16 | 12.50% |
| Function Generators | 9951 | 18560 | 53.62% |
| CLB Slices | 4976 | 9280 | 53.62% |
| Dffs or Latches | 940 | 20228 | 4.65% |
| Timing Report | | | |
| Clock Frequency | 145.138 MHz | | |

TABLE 5.2: Timing Report for The Virtex 2 Pro FPGA

Observing the values in the above tables we find the ASIC Implemnation to consume a large number of gates. The clock frequency is also slower on the ASIC desgin wich is usually not the case since ASIC's tend to be faster. In the case of ASIC desgins

| ASIC ami_0.5 | |
|---|---|
| Number of ports | 526 |
| Number of netss | 47569 |
| Number of instances | 46782 |
| Number of references to this view | 0 |
| Total accumulated area | |
| Number of gates | 167802 |
| Number of accumulated instances | 46782 |
| Clock Frequency | 89.2 MHz |

TABLE 5.3: Device Utilization Results 0.5u ASIC

every additional flip-flop or latch introduced to the circuit requires additional resources. However in FPGA technology that is not the case. Neverthless the size and speed ratio seems reasonable from chips of that callibar. Since most other AES modes opearte at almost half such frequency.

## 5.7. Comparison with Generic Authenticated-Encryption Modes

Every digital circuitry can be implemented differently. Depending on the design goal. Since the objective is to demonstrate that OCB is both lower in cost and higher in speed than generic authenticated-encryption modes, it becomes essential then to take these requirements into consideration. Area most of the time effects cost. We tried to balance both area and speed in the design, in order to provide a suitable architecture in terms of both cost and operational speed. Throughput is a good measure of the circuit speed. One way to increase throughput is to use high-speed circuitry, thats why it was essential in the desgin to implement a fast Rijndael AES engine.

Another way to increase speed is to exploit parallelism. I do believe there is large amount of parallelism that could be exploited form the algorithm however it was hindered by the fact that the message processing circuitry would be complex. In order to process

a message of arbitrary length that would require large amounts of memory and probably the use of timing multiplexer. This translates into higher costs.

The design takes on an iterative architecture permitting only one block data at a time. In some design computing one block of data takes almost the same number of clock cycles as the number of cipher rounds. However in this design this is not the case. Since we are computing offsets "whitening keys", the relation here is the that number of clock cycles is indicated by the number of total rounds required for encryption .A summary of results in area, maximum clock rate, throughput, latency of an OCB-AES are depicted in Table 5.4

| Summary of The OCB Implemenation Results | | | | |
|---|---|---|---|---|
| Technology | Area (gates) | Clock rate (Mhz) | Throuhput (Mbps) | Latency (cycles) |
| FPGA | 4976 CLBS | 145 | 10.8 | 13 |
| ASIC | 167802 | 89.2 | 6.2 | 13 |

TABLE 5.4: Summary of results in area, maximum clock rate, throughput, latency of an OCB-AES encryption

The critical path of the OCB architecture is determined by the cipher_compuataion block. After the first 13 clock cycles The AES implemented has a latency of 13 clock cycles. The circuit depth of OCB is essentially 3 times that of the block cipher, a call to form the initial offset; calls to form the ciphertext core, and a call to compute the tag. Reducing the block-cipher circuit-depth to one or two is possible, but the benefit does not seem worth the associated drawbacks. Throughput is defined by the following

$$\text{throughput} = \frac{\text{block size}}{\text{clock period} \times \text{number of clock cycles}}$$

The OCB architecture reaches a throughput of 10 Mbps

$$10\text{Mbps} = \frac{145\text{MHz} \times 128\text{bits}}{1703\text{cc}}$$

For comparison purposes we report the area and timing results of a fast AES implementation using a Xilinx FPGA from [5] in Table 5.5. This AES implementation intended to be used with CCM mode (CBC-MAC), this implmenation was done on a Xilinx Spartan IIE, running on 50 MHz platform. The implementation is not that of a CBC-MAC rather of a fast AES to be used in CCM mode. The throuhphut achieved in that design is about 11 Mbps, where as our design produced almost the same throuhput with additional authentication goals !

| Device Utilization Summary | |
|---|---|
| Selected Device | 2s200pq208-5 |
| Number of Slices | 2035 out of 2352 86 percent |
| Number Slice Flip Flops | 787 out of 4704 16 percent |
| Number of 4 input LUTs | 3921 out of 4707 8 percent |
| Number of bonded IOBs | 24 out of 144 16 percent |
| Number of GCLKs | 1 out of 4 25 percent |
| Timing Report | |
| Speed Grade | -5 |
| Minimum Period | 23.075ns |
| Maximum Frequency | 43.337 MHz |
| Minimum input arrival time before clock | 16.783ns |
| Maximum output required time after clock | 16.756ns |
| Maximum combinational path delay | No path found |

TABLE 5.5: Experimental Results of AES on an FPGA, from David Zier[5].

| Algorithm | Message Size | | |
|---|---|---|---|
| | 64 B | 256 B | 1 KB |
| OCB encrypt | 5.68 (91) | 15.4 (247) | 53 (851) |

TABLE 5.6: Performance results, in cycles per byte (cycles per 16-byte clock) on a Xilinx Virtex-II Pro FPGA

In contrast to the above results fast AES implementation on small FPGAs is restricting. Larger chips could provide for faster AES implementations. Reducing the clock cycle time is always an architectural aim for faster processes. By instancing more AES engines we could produce a faster design, however that would results in higher costs, since these FPGAs are small in comparison to high-end ones. I do believe our results show high throughput in terms of cost endured.

## 5.8. Discussion

Performance of an architectural design can be evaluated in many ways, depending on the target environment for its use. The general consensus is that the faster a design performs the better it is. However many applications that regard low power and small chip size of major importance, we find that $area \times time$ is a great measure for the application performance. There is always this compromise in order to deal with the specifics of the goals intended. The overall speed of the architecture in this work is dependant on the number of clock cycles needed to perform an encryption in the OCB mode for a given message size, but also on the clock period that can be achieved. The number of cycles required to process an m-bits using OCB is $C(m) = [[m/n] + 3] \times 13$ at a clock rate of 145 MHz, where n is the block size (128). The number of cycles for different size message was computed and the summary of these calculations are depicted in Table 5.6.

For the sake of comparison we report in Table 5.7, some experimental results by Helger Lipmaa on a Pentium III [6].

| Algorithm | 64 B | 256 B | 1 KB | 4 KB |
|---|---|---|---|---|
| OCB encrypt | 24.7 (395) | 18.5 (296) | 16.9 (271) | 16.7 (267) |
| ECB encrypt | 15.1(241) | 15.0 (239) | 14.9 (238) | 14.9(238) |
| CBC encrypt | 15.9 (254) | 15.9 (254) | 15.9 (255) | 15.9 (256) |
| CBC mac | 19.2 (307) | 16.3 (261) | 15.5 (248) | 15.3 (246) |

TABLE 5.7: Performance results, in cycles per byte (cycles per 16-byte clock) on a Pentium III. The Block cipher is AES128 [6].

It is actually quite difficult to compare the results of our implementation to the ones above sine the environment is not unified. These values are produced via a Pentium III, optimized assembly implementation. Nevertheless our results show a vast improvement in the encryption of short messages, however when messages get larger the efficiency drops dramatically. This in large was expected as a serial implementation does not handle high volumes of data like a parallel architecture would. However such variations in the way the OCB should be implemented should be dictated by the goals of the target applications that need it. Future improvements could show higher promise for OCB.

## 5.9. Conclusion & Future work

Certain criteria constrict the exploitation of the parallelism present in the OCB algorithm. The online feature where the input could be a message of an arbitrary length hinders the implementation from exploiting the parallel features in the algorithm. Space in hardware is limited; to be able to process a message of any length we face the task of saving such a message in hardware. This Requires the message be saved in memory until there is an indication the message is over. Since memory is always limited and

retrieving data from and to memory presents amounts of delay, it seems an inefficient method to deploy. A serial process in this case makes more sense since the message gets processed with no limitations on length as we go along. Another constriction in saving the message in memory is the addressing of the data, which requires that the number of blocks in the message be known in advance. Where the number of blocks is defined as follows: $m = max\{1, [|M|/n]\}$ and n is the block size in bits. We where able to produce software to compute the number of blocks in a message however we found difficulty in doing so in hardware. We attempted to use a counter to count the number of bits in order to compute the number of blocks and then created a $640 \times 128$bit memory unit to store the message into 128-bit vectors, however this took cycle time and space so the whole process seemed inefficient. The limited numbers of inputs in hardware also created an obstacle in achieving the parallelism inherit in the algorithm. To have m processes execute at the same time where m is the number of blocks means we require m inputs plus any additional inputs for the design. Since the m value depends on the length of the message it means we have an upper bound in the number of simultaneous processes achieved in hardware. This translates into a limited exploitation of parallelism meaning we actually could not process a message of arbitrary length in the absolute sense of the manner but we could process a range of arbitrary messages. That is certainly the case in a parallel implementation however a serial implementation exonerates itself from such limitations.

A novel architecture for OCB-AES encryption has been proposed. Unfortunately it is hard to target any performance or efficiency measures, since there arent any previous architectural implementations to compare to. However based on the throughput values achieved and the clock cycles per byte computations, it is safe to assume that the results show higher speeds and lower costs then most generic composition architectures that serve the same purpose.

Although the design is still in its early stages and there are plenty of improvements needed to reach peak performance. The objective was to present that OCB-AES

implementation that could provide a better throughput and lower cost than generic composition models of the same category. I believe the objective was met and the results do testify. The architecture has been implemented in VHDL, synthesized and tested successfully. The design takes on a more serial approach, however a parallel approach may or may not provide a better throughput. The challenge in the parallel approach is reducing the delay form the message preprocessing and trying to achieve a low area implementation.

The architecture presented in this thesis is an early prototype and there are some issues to be addressed to further better performance:

1 investigate the AES engine if we can improve its current limitation of handling one block for 13 clock cycles which increases latency of OCB and reduces the throughput.

2 Further optimization in-terms of area and maximum operating frequency.

3 Try to develop the complete interface of accepting input data and be able to save the cipher on-fly.

# BIBLIOGRAPHY


1. W. Stallings, *Cryptography and network security: principles and practice*, Prentice-Hall, Inc., Upper Saddle River, NJ 07458, USA, third edition, 2002.

2. 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.

3. M. Bellare and C. Namprempre, "Authenticated encryption: Relations among notions and analysis of the generic composition paradigm," in *ASIACRYPT '00: Proceedings of the 6th International Conference on the Theory and Application of Cryptology and Information Security*, London, UK, 2000, pp. 531–545, Springer-Verlag.

4. P. R. Chodowiec, "Comparison of the Hardware Performance of the AES Candidates Using Reconfigurable Hardware," Master thesis, George Mason University, Fairfax, VA,USA, 2002.

5. K. Vu and D. Zier, "FPGA implementation aes for ccm mode encryption using xilinx spartan-ii," Tech. rep., Oregon State University, April 2003.

6. H. Lipmaa, "Personal communications," Jul 2001, www.tcs.hut.fi/ helger.

7. "Modes of operation," 2005, http://csrc.nist.gov/CryptoToolkit/modes.

8. C. S. Jutla, "Encryption modes with almost free message integrity," in *EUROCRYPT '01: Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques*, London, UK, 2001, pp. 529–544, Springer-Verlag.

9. L. C. Washington and W. Trappe, *Introduction to Cryptography: With Coding Theory*, Prentice Hall PTR, Upper Saddle River, NJ, USA, 2002.

10. M. Bellare and P. Rogaway, "Encode-then-encipher encryption: How to exploit nonces or redundancy in plaintexts for efficient cryptography," in *ASIACRYPT '00: Proceedings of the 6th International Conference on the Theory and Application of Cryptology and Information Security*, London, UK, 2000, pp. 317–330, Springer-Verlag.

11. C. Namprempre , "Thesis slides: Relations among notions and analysis of the generic composition paradigm," 2005, http://www.nr.no.

12. J. Katz and M. Yung, "Unforgeable encryption and adaptively secure modes of operation," *Fast Software Encryption00, Lecture Notes in Computer Science*, vol. ??, 2000.

13. M. Bellare, A. Desai, E. Jokipii, and P. Rogaway, "A concrete security treatment of symmetric encryption," in *FOCS '97: Proceedings of the 38th Annual Symposium on Foundations of Computer Science (FOCS '97)*, Washington, DC, USA, 1997, p. 394, IEEE Computer Society.

14. C. Dwork, D. Dolev, and M. Naor, "Non-malleable cryptography," in *STOC '91: Proceedings of the twenty-third annual ACM symposium on Theory of computing*, New York, NY, USA, 1991, pp. 542–552, ACM Press.

15. O. Goldreich, S. Goldwasser, and S. Micali, "How to construct random functions," *J. ACM*, vol. 33, no. 4, pp. 792–807, 1986.

16. E. Petrank and C. Rackoff, "CBC MAC for real-time data sources," Cryptology ePrint Archive, Report 1997/010, 1997, http://eprint.iacr.org/.

17. NIST, National Institute for Standards and Technology, "Symmetric key block cipher modes of operation workshop," Tech. rep., FIPS PUB, October 2000.

18. V. Gligor and P. Donescu, "Fast encryption and authentication: XCBC encryption and XECB authentication modes," *Contribution to NIST*, Oct 2000.

19. 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.

20. 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.

21. D. A. McGrew, J. Viega, and R. Housley, "The need for speed: authenticated encryption and gcm," work in progress, Network Working Group, October 2004.

22. Ars Technica, "Wireless Security Blackpaper," *arstechnica*, July 2002.