Authenticated Encryption on FPGAs from the Reconfigurable Part to the Static Part

Karim Moussa Ali Abdellatif

UPMC
SORBONNE UNIVERSITÉS

THÈSE DE DOCTORAT DE L'UNIVERSITÉ PIERRE ET MARIE CURIE

Spécialité: Informatique, Télécommunication et Électronique

École Doctorale Informatique, Télécommunication et Électronique

Présentée par:

Karim Moussa Ali Abdellatif

Pour obtenir le grade de

Docteur de l'Université Pierre et Marie Curie

CHIFFREMENT AUTHENTIFIÉ SUR FPGAs DE LA PARTIE RECONFIGURABLE À LA PARTIE STATIC

Soutenue le 07/10/2014

devant le jury composé de:

M.	Bruno Robisson	ENSM.SE/CEA	Rapporteur
M.	Lilian Bossuet	Laboratoire Hubert Curien	Rapporteur
M.	Jean-Claude Bajard	LIP6	Examinateur
M.	Hayder Mrabet	FLEXRAS	Examinateur
M.	Olivier Lepape	NanoXplore	Examinateur
M.	Habib Mehrez	LIP6	Directeur de thèse
Mme.	Roselyne.Chotin-Avot	LIP6	Encaderante de thèse

UPMC
SORBONNE UNIVERSITÉS

Ph.D. THESIS OF THE PIERRE AND MARIE CURIE UNIVERSITY

Departament: Electronics, Telecommunications and Computer Science

Presented by:

Karim Moussa Ali Abdellatif

Submitted to obtain the Ph.D. degree from:

University of Pierre and Marie Curie

Authenticated Encryption on FPGAs from the Reconfigurable Part to the Static Part

Defense date: 07/10/2014

Committee in charge:

M.	Bruno Robisson	ENSM.SE/CEA	Reviewer
M.	Lilian Bossuet	Laboratoire Hubert Curien	Reviewer
M.	Jean-Claude Bajard	LIP6	Examiner
M.	Hayder Mrabet	FLEXRAS	Examiner
M.	Olivier Lepape	NanoXplore	Examiner
M.	Habib Mehrez	LIP6	Thesis Advisor
Mme.	Roselyne.Chotin-Avot	LIP6	Thesis Co-Advisor

Abstract

Communication systems need to access, store, manipulate, or communicate sensitive information. Therefore, cryptographic primitives such as hash functions and block ciphers are deployed to provide encryption and authentication. Recently, techniques have been invented to combine encryption and authentication into a single algorithm which is called Authenticated Encryption (AE). Combining these two security services in hardware produces better performance compared to two separated algorithms since authentication and encryption can share a part of the computation. Because of combining the programmability with the performance of custom hardware, FPGAs become more common as an implementation target for such algorithms.

The first part of this thesis is devoted to efficient and high-speed FPGA-based architectures of AE algorithms, AES-GCM and AEGIS-128, in order to be used in the reconfigurable part of FPGAs to support security services of communication systems. Our focus on the state of the art leads to the introduction of high-speed architectures for slow changing keys applications like Virtual Private Networks (VPNs). Furthermore, we present an efficient method for implementing the GF(2^{128}) multiplier, which is responsible for the authentication task in AES-GCM, to support high-speed applications. Additionally, an efficient AEGIS-128 is also implemented using only five AES rounds. Our hardware implementations were evaluated using Virtex-5 and Virtex-4 FPGAs. The performance of the presented architectures (Thr./Slices) outperforms the previously reported ones.

The second part of the thesis presents techniques for low cost solutions in order to secure the reconfiguration of FPGAs. We present different ranges of low cost implementations of AES-GCM, AES-CCM, and AEGIS-128, which are used in the static part of the FPGA in order to decrypt and authenticate the FPGA bitstream. Presented ASIC architectures were evaluated using 90 and 65 nm technologies and they present better performance compared to the previous work.

Keywords– Authenticated Encryption, FPGAs, ASIC, Secure Reconfiguration.

To the memory of my father ..

Acknowledgements

After seemingly endless days of typing and correcting, the time has come to write the last part of this thesis. I will take this opportunity to thank several people who have greatly influenced the journey leading to this dissertation.

First of all, great appreciation goes to my supervisors Prof. Habib Mehrez and Dr. Roselyne Chotin-Avot for giving me the chance to do the PhD degree in LIP6. I am grateful for the freedom I have been given during my research and the guidance provided when needed.

I would like to extend my gratitude to Prof. Hesham Hamed, Prof. Hassan Aboushady, Dr. Alp Kilic and Vinod Pangracious for their valuable support along the way. A great thank you goes to Akram Malak, Dr. Raouf Khalil, Yasser Yousry, Tamer Badran, Alhassan Sayed, Mohamed Shaaban, and Mootaz Allam for making working hours more enjoyable.

Besides my colleagues, I appreciate the care and support of my good friends, who always managed to move my thoughts away from work when needed like Ahmed Nabil, Mahmoud Borcan, Ahmed Nadi, and Bahaa Elmasry.

Finally, I would like to thank my parents to whom I owe a great deal. To my late father Moussa Abdellatif, thank you for showing me that the key to immortality is to live a life worth remembering. Very special thanks and a lot of love go to my wife Nouranne Fahhim who always motivated and supported me with great enthusiasm.

For all our freedom martyrs, Egypt 25/01/2011

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter, we introduce the overall motivation for this work and describe the contributions. Furthermore, we outline the principal organization of the core chapters.

1.1 Motivation

1.1.1 Authenticated Encryption

In our growing world of technology, the amount of information that we share with the rest of the digital universe is constantly increasing. Our demands to conceal confidential data are therefore being strongly needed and become very important. The protection of a message includes the protection of confidentiality and authenticity. There are two main approaches to authenticate and encrypt a message:

1. The first approach is to treat the encryption and the authentication separately. A block cipher or stream cipher is used to encrypt the plaintext, and a Message Authentication Code (MAC) is used to authenticate the

message as shown in Fig. 1.1. For example, we may use Advanced Encryption Standard (AES) [4] for encryption, and then apply Pelican Message Authentication Code (Pelican-MAC) [5] or Hash Message Authentication Code (HMAC) [6] to the message to generate the MAC. The encryption algorithm encrypts the message using a certain key (key1) to provide the encrypted message. The authentication algorithm generates the MAC using another key (key2) to provide the receiver with the entity of the sender (authentication). The receiver decrypts the encrypted message with the same key (key1) and authenticates the message (MAC computation) using key2 to compare it with the sent MAC in order to detect if they are equal or not.

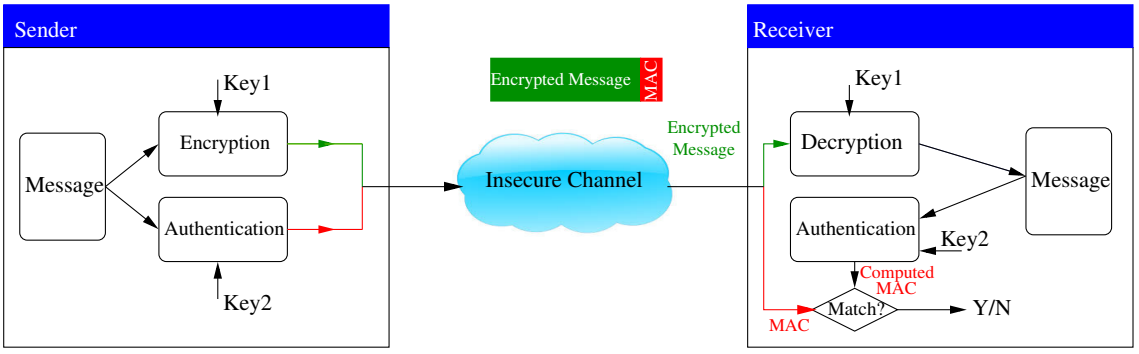

FIGURE 1.1: Two separated algorithms for encryption and authentication

2. Another approach is to apply an integrated Authenticated Encryption (AE) algorithm to the message to provide both encryption and authentication. One can expect that this is more efficient since encryption and authentication can share a part of the computation. AE algorithms use only one key for encryption and authentication as shown in Fig. 1.2. Therefore, the key exchange and storage issues are better compared to using two separated algorithms.

AE has been used in many widely standards such as Secure Shell (SSH) [7], Secure Sockets Layer / Transport Layer Security (SSL/TLS) [8], IPsec [9], and IEEE 802.11 (Wi-Fi) [10]. This has made AE very important in protocols to secure the fundamentals of the information and communication infrastructure.

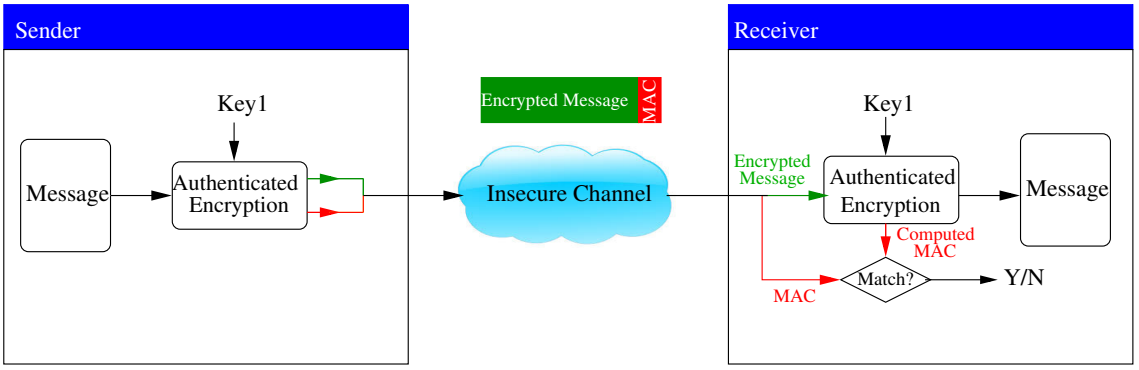

FIGURE 1.2: Authenticated encryption

There are now two NIST recommended modes of operation for authenticated encryption, namely, Counter with Cipher Block Chaining Mode (CCM) [11] and Galois Counter Mode (GCM) [12]. Quite some AE schemes have been proposed, and more are expected to join the ranks with the ongoing CAESAR[1]. The CAESAR competition is a move towards selecting a portfolio of AE schemes that should improve upon the state of the art. AEGIS [13] is considered one of the submitted proposals to CAESAR.

Software realizations of such algorithms have the advantage that they are portable to multiple platforms. In general, they have a fast time to market. However, they can be applied in systems with limited traffic at low encryption rates. Moreover, software-based applications are not power efficient compared to specialized hardware architectures. Speed and power are two major drawbacks that motivate the hardware design exploitation of cryptographic primitives.

1.1.2 FPGAs

The increasing costs of silicon technology have put considerable pressure on developing dedicated SoC solutions. This means that the technology will be used increasingly for high-volume or specialist markets. Recently, Field Programmable Gate Arrays (FPGAs) have been proposed as a hardware technology for communication systems as they offer the capability to develop the

[1]Competition for Authenticated Encryption: Security, Applicability, and Robustness.

most suitable circuit architecture of the application in a similar way to SoC systems. Compared to a full custom ASIC design, they are cost efficient, easier to manage and can immediately be put into operation. Furthermore, they can continuously be reprogrammed during and after the design.

In order to redefine the functionality of the FPGA, a bitstream configuration file is uploaded on the FPGA. The reconfiguration includes downloading this bitstream file which contains the new design on the FPGA. As shown in Fig. 1.3, the bitstream is processed by the static part (configuration logic) which is not programmable. After that, the Static Random Access Memory (SRAM) is programmed by the processed bitstream. By programming SRAM cells, the functionality on the FPGA reconfigurable part (user part) can be tailored to implement the new design.

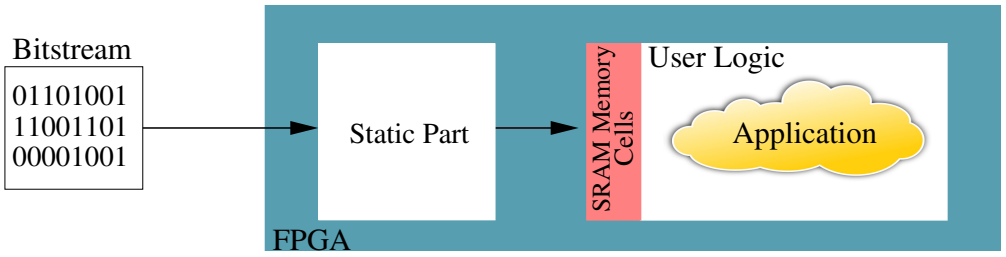

FIGURE 1.3: Reprogrammability using bitstream

Configurable Logic Blocks (CLBs), interconnections, and embedded components (like Block Rams (BRAMs) and Digital Signal Processing (DSP) units) shown in Fig. 1.4 are established by programming SRAM cells to connect fabricated routing wires together.

SRAM-based FPGAs such as those manufactured by Xilinx and Altera comprise the largest fraction of the overall market. Because SRAM memory is volatile, SRAM cells must be loaded with configuration data each time the device powers up. Configuration data is typically transmitted from an external memory source (Non Volatile Memory (NVM)) (see Fig. 1.5), such as a flash memory or a configuration device, to the FPGA.

In recent years, FPGAs manufacturers have come closer towards filling the performance gap between FPGAs and ASICs, enabling them, not only to serve as fast prototyping but also to become active players as components in

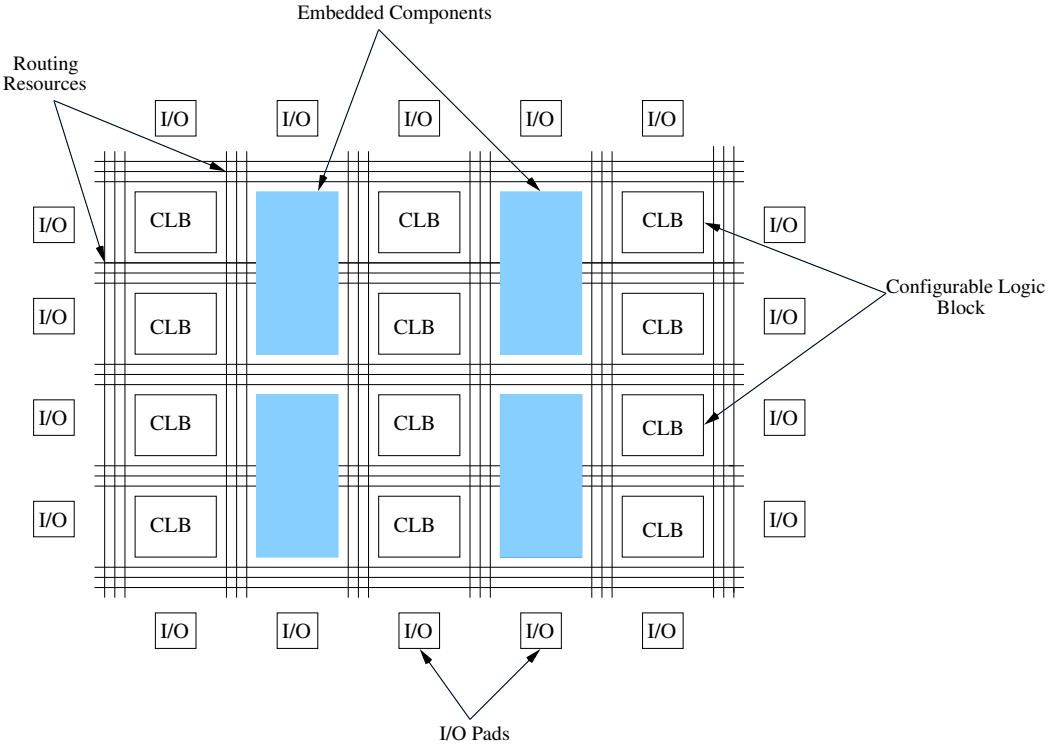

FIGURE 1.4: Reconfigurable part (user part) architecture

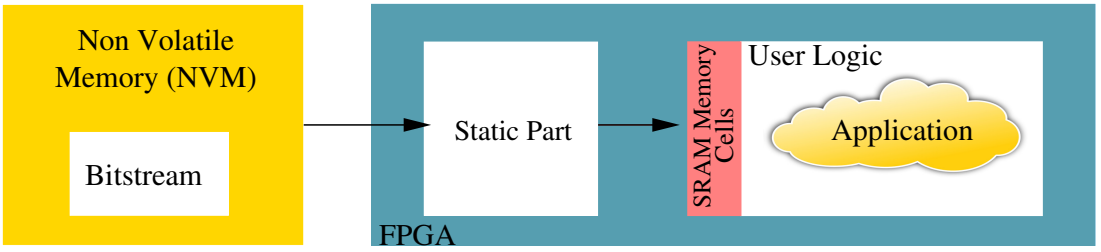

FIGURE 1.5: FPGA reconfiguration using NVM

embedded systems. As an example, Virtex-5 family [14] of FPGAs has 2400-25920 CLBs, 1152-11664 Kb BRAMs, 8-24 multi-Gigabit transceivers, 32-640 DSP slices, 4-12 digital clock managers, and 2-6 PLL clock generators. Thus, FPGAs are integral parts in embedded system design.

1.2 Contributions

This thesis deals principally with the design of efficient hardware implementations of cryptographic algorithms for encryption and authentication. We propose FPGA and application-specific ASIC implementations that target a wide range of different applications.

In particular, we focus on the hardware design of current AE algorithms. Fig. 1.6 shows the main contributions of this thesis. We investigate new efficient and high speed FPGA-based architectures of AE algorithms which will be in the user part (reconfigurable part) of the FPGA. Also, with the gathered knowledge, we define a framework to present different ranges of low cost ASIC architecture of AE algorithms which are inserted in the static part in order to protect the Intellectual Property (IP) of FPGA Bitstreams.

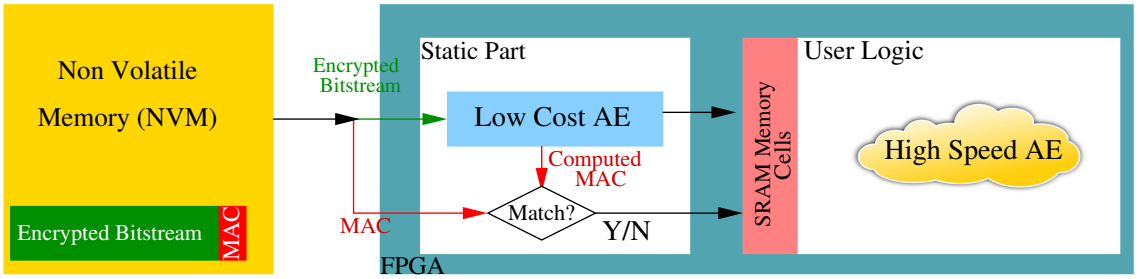

FIGURE 1.6: Thesis contribution

The contributions of this thesis can be summarized in the following points:

- **High speed AE for slow changing key applications on FPGAs**
  - Algorithms for authenticated encryption are the most suitable solution to reliably secure a network link. AES-GCM has been utilized in various security-constrained applications. VPNs are widely employed to connect private local area networks to remote locations. VPNs use AES-GCM for encryption and authentication. Current commercial security appliances of VPNs allow a throughput from 40 to 60 Gbps [15, 16]. We describe the benefits of VPNs as being a slow changing key environment in order to design efficient high speed architectures. In addition, we present a solution for the

parallelization of AES-GCM cores in order to support applications up to 100 Gbps. Moreover, we present a protocol to secure the reconfiguration of the proposed architectures on FPGAs.

- **FPGA-based high performance AES-GCM using Efficient $GF(2^{128})$ multiplier**
  - The performance of AES-GCM is always determined by the GF(2^{128}) multiplier (authentication part) because of the inherent computation feedback. We introduce an efficient pipelined Karatsuba Ofman Algorithm (KOA) for the GF(2^{128}) multiplier to support high speed applications. In particular, the computation feedback is removed by analyzing the complexity of the computation process. The proposed GF(2^{128}) multiplier is evaluated with the pipelined AES in order to support high speed applications.
- **Efficient hardware implementation for AEGIS**
  - Pushed by our involvement into the hardware design of efficient AE algorithms, we decided to follow the CAESAR competition by designing high speed architecture of AEGIS which was accepted to the first round of the CAESAR competition.

  All presented architectures in these contributions aim at applications with high demands for data throughput and performance. For these designs, we primarily employ Xilinx Virtex-4 and Virtex-5 FPGAs as an implementation target. Our proposed high speed architectures provide most savings in logic and routing resources with the highest data throughput on FPGAs compared to previous work reported in open literature.
- **Low cost solutions for IP protection of FPGA bitstreams**
  - IPs loaded on the FPGAs represent a kind of investment that requires protection especially in case of remote reconfiguration. Therefore, the FPGA must accept encrypted bitstreams from authorized entities. We present different low cost solutions for performing decryption and authentication of bitstreams. Efficient and

compact ASIC AE architectures, AES-CCM, AES-GCM, and AEGIS, are added in the static part of the FPGA to support encrypted and authenticated bitstreams.

Presented ASIC architectures were evaluated by using 90 and 65 nm technologies. Our comparison to previous work reveals that our architectures are more area-efficient.

1.3 Thesis organization

The outline of this thesis is as follows:

- **Chapter 2** gives detailed overview about current AE algorithms. AES-CCM, AES-GCM, and AEGIS are discussed in detail. Later in the chapter, we highlight the previous work that covers the hardware implementations of AE algorithms.
- **Chapter 3** is entirely dedicated to high speed AES-GCM architectures for slow changing key applications like VPNs. It presents an efficient method for the parallelization of AES-GCM cores on FPGAs. Furthermore, we propose a protocol to protect the bitstream of the proposed architectures on FPGAs.
- **Chapter 4** presents an efficient method for the $GF(2^{128})$ multiplier used in AES-GCM. By focusing on the drawbacks of the previous architectures of $GF(2^{128})$ multipliers, we propose an efficient method to remove the computation feedback in the $GF(2^{128})$ multiplier. This is accomplished by the presented pipelined Karatsuba Ofman Algorithm (KOA)-based $GF(2^{128})$. Finally, an efficient high speed architecture of AEGIS is also presented in this chapter.
- **Chapter 5** starts with an overview of security issues used in reconfiguration of FPGAs and analyze how well they are suited to IP protection of FPGA bitstreams. Later in this chapter, low cost implementations of AE algorithms are presented for FPGA bitstreams protection.

- **Chapter 6** summarizes the thesis and points out some future work.

Part I

High Speed FPGA-based AE Architectures

Chapter 2

Authenticated Encryption

2.1 Introduction

Originally, confidentiality and authenticity services have been implemented separately, by using two different algorithms. Encryption algorithms are used to ensure confidentiality while Message Authentication Codes (MACs) can be used to provide authentication. When two separated algorithms are used to provide independent security services, it is considered cryptographically secure to use separated keys for each algorithm. Recently, techniques have been invented to combine encryption and authentication into a single algorithm which is called Authenticated Encryption (AE). Combining these two security services in hardware might support the following advantages:

- Area requirement for a single algorithm could be smaller compared to two separated algorithms because encryption and authentication can share a part of the computation.

- A slight advantage regarding key management and key storage issues because AE needs only a single key for both encryption and authentication.

The following is a brief overview of the most popular AE algorithms that have been developed as a result of the current research. The aim is to provide the details of AE algorithms to allow the reader to understand and analyze each algorithm. Also, for each algorithm, we present the previous work in terms of the hardware implementation in order to highlight the current challenges of the state of the art.

The most common way of constructing an AE scheme is a mode of operation for a block cipher, like AES [4]. Therefore, in order to discuss AE algorithms clearly, it is useful to highlight AES because it is common in all AE schemes that we will present.

2.2 Advanced Encryption Standard (AES)

For the drawbacks of the previous symmetric-key cryptographic standards such as the DES and the 3DES, they have been replaced by the Advanced Encryption Standard (AES) [4]. In particular, the AES has overcome the drawbacks of the previous standards in terms of vulnerability to brute force attacks and slow software implementations. The AES was accepted by the National Institute of Standards and Technology (NIST) in 2001 and since its acceptance, it has been utilized in a variety of security-constrained applications like IEEE 802.11i standard [17]. It is considered by industry and government environments as the essential scheme to protect sensitive information.

2.2.1 Algorithm specifications

The AES algorithm is a symmetric-key cipher, in which both the sender and the receiver use a single key for encryption and decryption. The data block length is fixed to be 128 bits, while the key length can be 128, 192, or 256

bits. It is an iterative algorithm. Each iteration can be called a round, and the total number of rounds, N, is 10, 12, or 14, when the key length is 128, 192, or 256 bits, respectively.

The 128-bit data block is divided into 16 bytes. These bytes are mapped to a 4×4 array called the State, and all the internal operations of the AES algorithm are performed on the State. Each byte in the State is detonated by $S_{(i,j)}$ $(0 \leq i \leq 3, 0 \leq j \leq 3)$ and is considered as an element of $GF(2^8)$ with $P(x) = x^8 + x^4 + x^3 + x + 1$ as an irreducible polynomial. Fig. 2.1 shows the block diagram of the AES algorithm.

In the encryption of the AES algorithm, each round except the final round consists of four transformations in a fixed order:

1. **SubBytes** is a non-linear transformation that substitutes the bytes of the state independently, using a s-box built over two steps: computation of the multiplicative inverse in the Galois field $GF(2^8)$ followed by an affine transformation in GF(2). The SubBytes can be described as follows:

$$S(i,j) = MS_{i,j}^{-1} + C \tag{2.1}$$

   where M is an 8×8 binary matrix, S^{-1} is the multiplicative inverse of the input, and C is an 8-bit binary vector with only 4 nonzero bits.

2. **ShiftRows** is a simple shifting transformation. The first row of the State does not change, while the second, third and fourth rows cyclically shift one byte, two bytes and three bytes to the left, respectively

3. **MixColumns** multiplies in $GF(2^8)$ each column of the of the State (4 bytes) with a(x) modulo $x^4 + 1$ where

$$a(x) = 03x^3 + 01x^2 + 01x + 02. \tag{2.2}$$

4. **AddRoundKey** adds a round key to the state in GF(2).

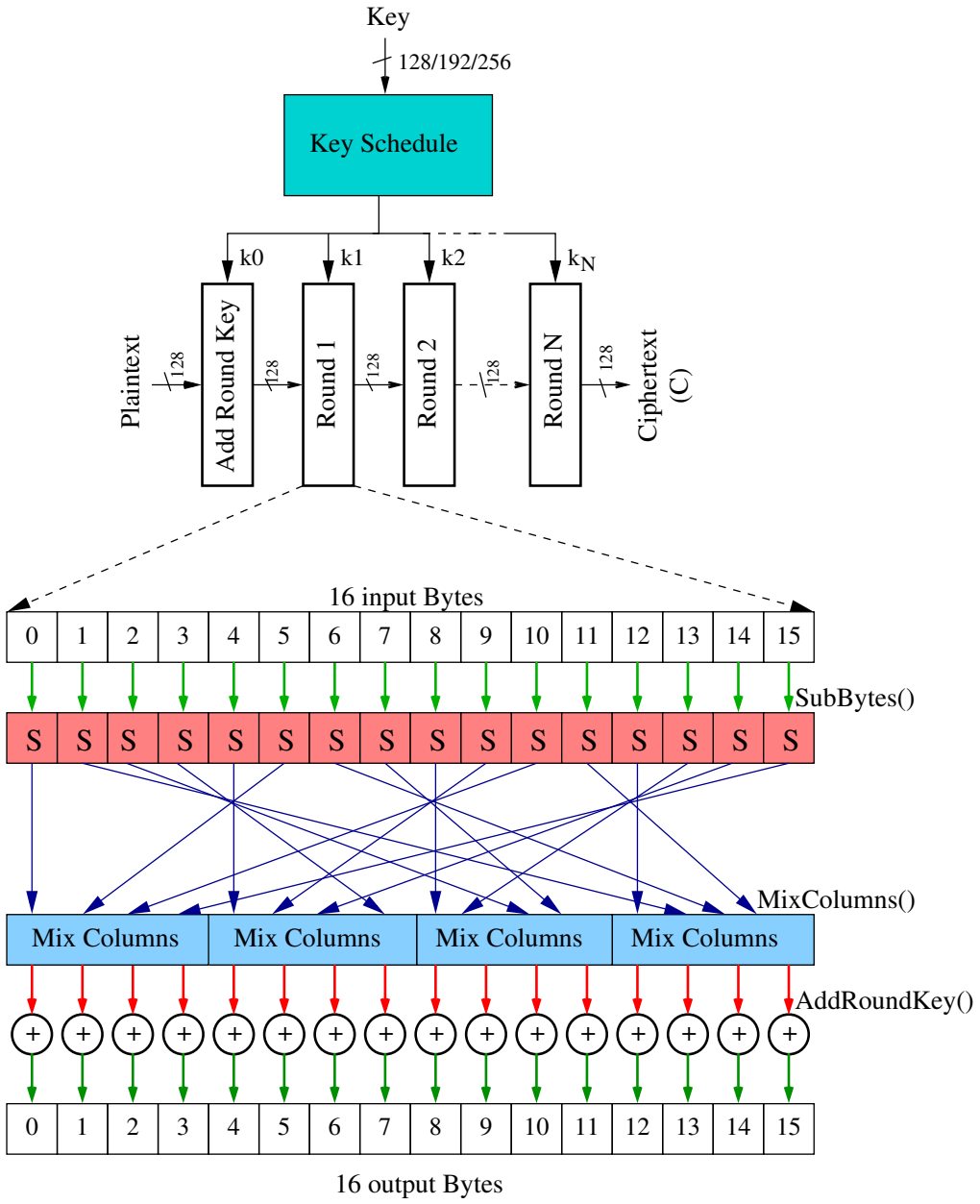

FIGURE 2.1: AES algorithm

The AES round is completed with the key expansion (see Fig. 2.2), which generates the round keys from the original key. This expansion is defined on 4-bytes words and mainly uses the transformations of the round. A single round key is made up by four words and is updated to the next round key using **g** function which combines the SubWord (s-box over a 4-byte input) and RotWord (word shift) functions. The rest of the key parts are generated using XOR function.

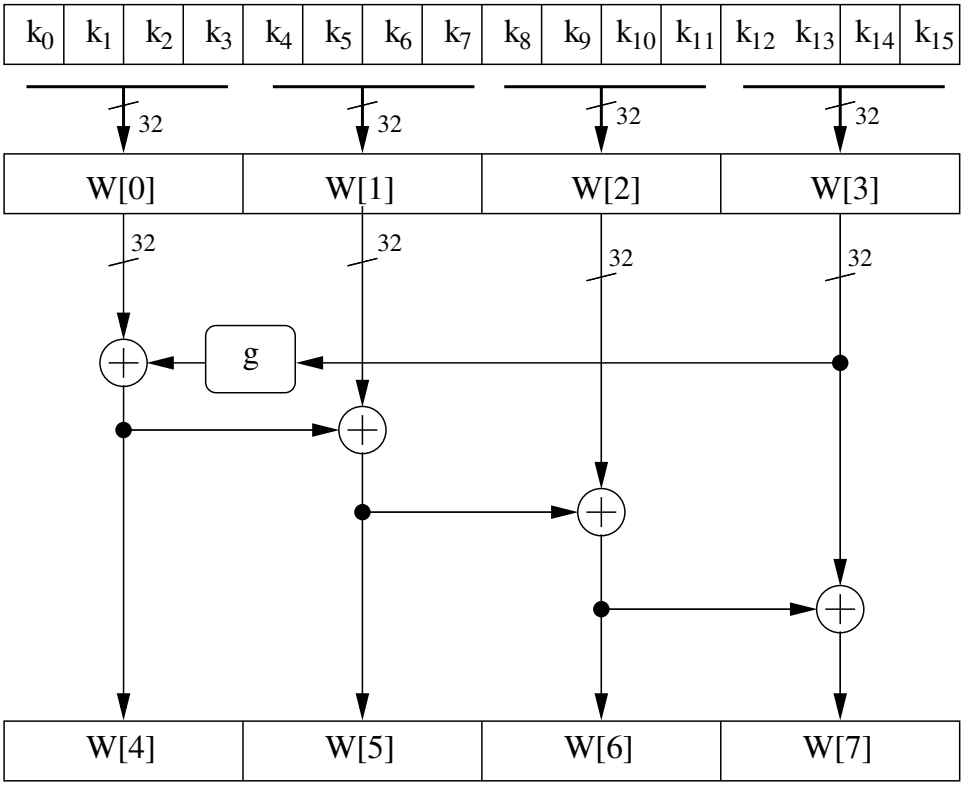

FIGURE 2.2: Key expansion of AES-128

The transformations in the decryption process perform the inverse of the corresponding transformations in the encryption process.

1. **InvSubBytes** performs the inverse of the SubBytes stage as follows:

$$S_(i,j) = (MS_{i,j}^{-1} + C)^{-1}. \tag{2.3}$$

2. **InvShiftRows** does not change the first row, while the rest of the rows are cyclically shifted to the right by the same offset as that in the ShiftRows.

3. **InvMixColumns** multiplies the polynomial formed by each column of the State with $a(x)^{-1}$ modulo $x^4 + 1$, where

$$a(x) = 0bx^3 + 0dx^2 + 09x + 0e. \tag{2.4}$$

4. **AddRoundKey** adds a round key to the state in GF(2), while the round keys are computed in the reverse order within the key expansion process.

2.2.2 Hardware implementation

After the standardization of AES, several research contributions focusing on hardware implementations of the algorithm have been presented. Nowadays, almost each possible AES architecture, covering the entire design space (both in ASIC and FPGA), has been investigated. Companies and federal organizations have plenty of choice in selecting the suitable design.

We list the major components and strategies that characterize the AES hardware design:

- **Datapath**: AES is defined by 128-bit state with 8-bit operations. As a result, the designer has the choice to adapt or shrink the datapath of his architecture according to the system specification. Typical datapath widths are 128, 64, 32, 16, or 8 bits. The datapath size affects directly the area occupation and the final throughput of the AES core. It represents a sort of area/speed trade-off. In [18], the authors analyze different-datapath implementations on FPGAs.

- **Round components:** The SubBytes transformation is the most costly (both in size and propagation delay) component of the AES round. In terms of the ShiftRows operation, it does not consume any logic because it is implemented as a straightforward routing of the signals without any specific hardware component, while the most efficient strategy to realize MixColumns is with the combinational logic as shown in [19]. Several approaches to design the SubBytes have been developed and implemented. We describe briefly three main solutions:

  1. **Block Random Access Memory (BRAMs)**
     The s-box is implemented as a Look Up Table (LUT) and stored in a dedicated memory. This approach is particularly suitable for

modern FPGAs, which contain BRAMs. This approach saves the logic area because the most consuming part (s-boxes) are implemented as BRAMs (see [20])

2. **Composite field**
   In [21], composite field arithmetic is employed to reduce the area requirements, and different implementations for the inversion in subfield $GF(2^4)$ are compared. This reduction to composite fields brings significant improvement in flexibility and in the area costs of SubBytes.

3. **FPGA LUT**
   The third approach is dedicated only to FPGA devices. The s-box can be directly instantiated in LUTs located in the FPGA logic elements. As example, the basic logic elements of Xilinx FPGAs are called slices and each slice comprises a different number of LUTs depending on the selected device. In a Xilinx Virtex-5 chip a single s-box is implemented on 32 6-input LUTs because of 6-input LUT technology (see [22]).

- **Rolled or unrolled**: The operation mode of AES influences remarkably the performance of the implementation. Feedback modes like Cipher Block Chaining (CBC) mode [23] lead to iterative designs, where single round is considered to be the most efficient solution. Fig. 2.3(a) shows the iterative design using only one round. An example of the iterative design is shown in [22]. Because of using only one round, the throughput of the iterative design of AES-128 presented by [22] is as follows:

$$Throughput(Mbps) = \frac{F_{max}(Mhz) \times 128}{10}. \tag{2.5}$$

Full unrolled architectures combined with pipelining (Fig. 2.3(b)) can be exploited only by the feedback-free modes [23] like Electronic Codebook (ECB) and Counter (CTR) mode. Unrolled designs with pipelining are able to achieve 20-30 Gbps, simply by putting a pipeline stage between the rounds. To further increase the speed, pipelining can also be implemented inside the round and the SubBytes operation (see [18]). The throughput of the pipelined design is calculated as follows:

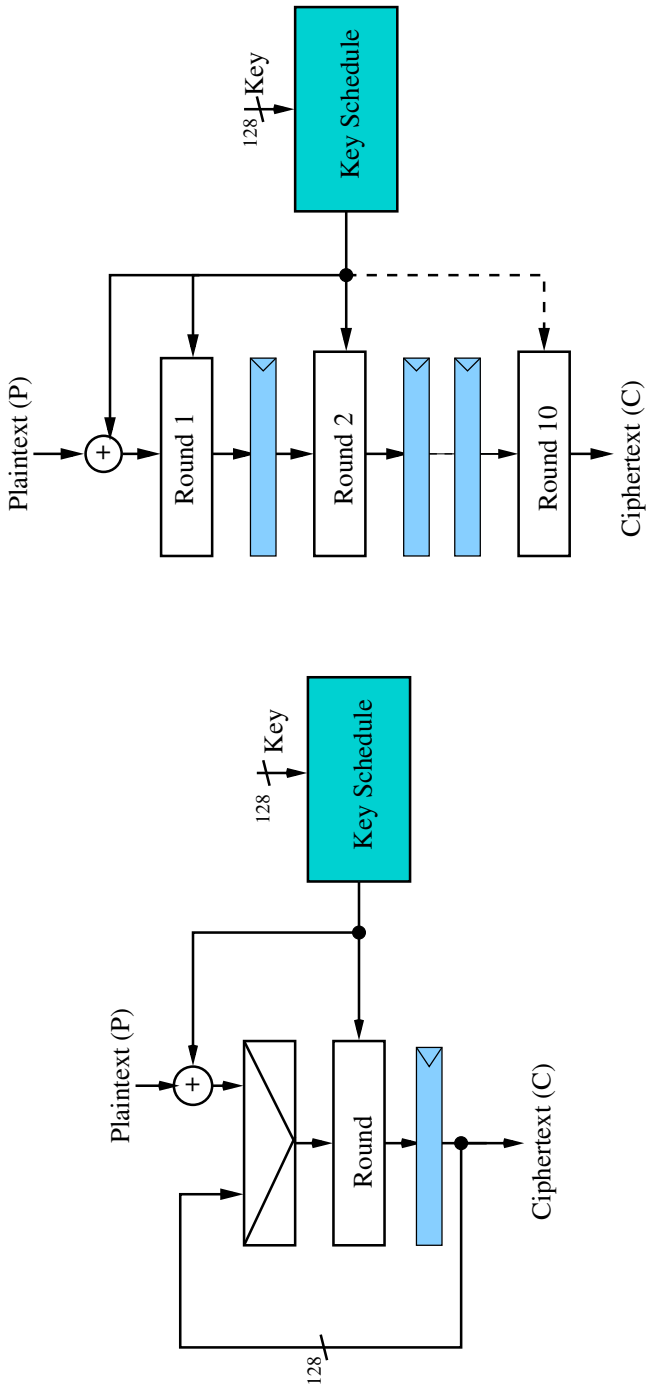

FIGURE 2.3: (a) Iterative design. (b) Pipelined design

$$Throughput(Mbps) = F_{max}(Mhz) \times 128. \quad (2.6)$$

Table 2.1 shows the hardware hardware comparison between the iterative design and the pipleined design. It is clear that the pipelined design consumes more area compared to the iterative design, while the speed is higher.

TABLE 2.1: Hardware comparison

Design	FPGA	Area (slices)	Frequency Mhz	Throughput Gbps
Iterative [22]	Vertex-E	2257	169	2.1
Pipelined [18]	Spartan-III	17425	196.1	25.1

After giving an overview of AES, the following sections present the current AE algorithms and how they are implemented on FPGAs and ASIC.

2.3 AES-CCM

Counter with Cipher Block Chaining Mode (CCM) [24] can be used in conjunction with any approved 128-bit block cipher like AES. 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. It is designed for packet environment, where all the data is available in storage before the processing. This implies that it is not online.

2.3.1 Algorithm specifications

Variables:

N : 128-bit Valid nonce

A : 128-bit Associated data

P : 128-bit Plaintext
C : 128-bit Ciphertext
MAC : 128-bit Message Authentication Code
L : Message length
Key:128-bit Key

Specifications:

AES-CCM has four inputs: an AES key, N, P, and A. AES-CCM generates two outputs: C and MAC. N and A are used to verify the correctness of the MAC. Fig. 2.4 shows the block diagram of CCM. It depends on the block cipher AES in two modes. Cipher Block Chaining (CBC) mode for the authentication, while the encryption is performed by Counter (CTR) mode. A and N are not encrypted but they are authenticated in order to verify the correctness of authentication.

Encryption and MAC generation:

1. The plaintext P is stored in a memory.
2. N, A, and P are loaded to AES in CBC mode to generate Y, this value is used for authentication.
3. The first value of the counter is encrypted (CTR mode) to be **S[0]** which is mixed with Y through XOR operation in order to generate the final MAC.
4. Encryption is performed by CTR mode.

Decryption and MAC verification:

1. Decryption is performed by XORing C[i] with the values of the encrypted counter S[i] to give P.
2. N, A, and P are loaded to AES in CBC mode to generate Y.
3. Y is XORed with S[0] to generate the MAC.
4. MAC verification is computed.

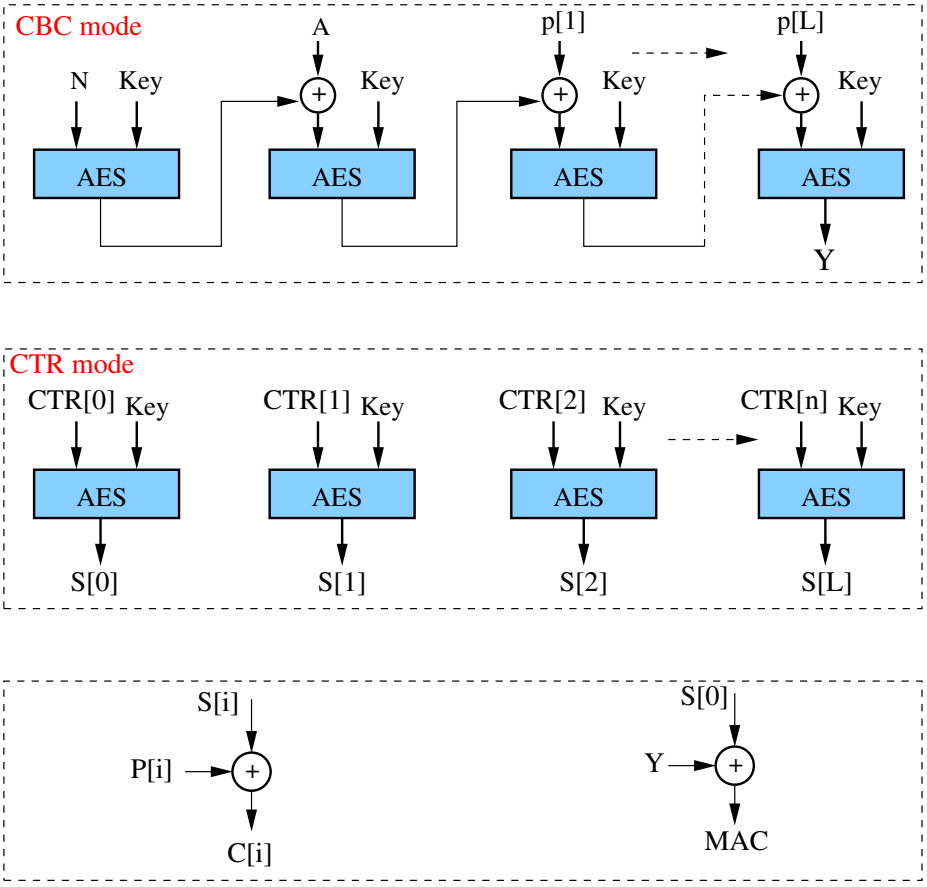

FIGURE 2.4: AES-CCM

2.3.2 Hardware implementation

There are some research contributions focusing on the hardware implementations of AES-CCM on FPGAs and ASIC. Because of being feedback-based algorithm (it depends on CBC mode), iterative design of AES is always used in the hardware implementation.

In [25], iterative AES (one round) was used to implement the architecture of AES-CCM on FPGAs and ASIC. The proposed architecture in [25] used one AES block for doing both authentication and encryption. This manner decreases the consumed area because there is only one AES architecture that is used for the two modes, CBC and CTR modes. As a result the speed of the architecture is reduced by 2 compared to the use of two AES architectures. Two

iterative components of AES were implemented for both encryption and authentication on FPGAs in [26]. In [27], FPGA-based AES-CCM was presented by using two blocks of AES and they are 32-based datapath architectures.

From the previous discussion, it is clear that AES-CCM is a feedback-based algorithm because it depends on the CBC mode in terms of performing the authentication task. Therefore, only iterative-based designs of AES are applicable to CCM, while using pipelined designs is not. As a result, the overall speed is limited as shown in Table 2.2.

TABLE 2.2: Hardware comparison of the previous AES-CCM architectures

Design	Implementation platform	Area	Frequency Mhz	Throughput Mbps
[25]	90 nm	0.057 mm^2	148	434
[26]	Spartan-3 (90 nm)	633 slices	100.08	1051.5
[27]	Spartan-3 (90 nm)	487 slices + 4 BRAMs	247	687.3

2.4 AES-GCM

Galois Counter Mode (GCM) [28] is an AE algorithm. It is well-suited for wireless, optical, and magnetic recording systems due to its multi-Gbps authenticated encryption speed, outstanding performance, minimal computational latency as well as high intrinsic degree of pipelining and parallelism. New communication standards like IEEE 802.1ae [29] and NIST 800-38D have considered employing GCM to enhance their performance.

It simultaneously provides confidentiality, integrity and authenticity assurances on the data. It can be implemented in hardware to achieve high speeds with low cost and low latency. It was designed to meet the need for an authenticated encryption mode that can efficiently achieve speeds of 10 Gbps and higher in hardware.

2.4.1 Algorithm specifications

Operations:
The following operations are used in AES-GCM:

$\oplus$: bit-wise exclusive OR
$\|$: concatenation

Variables:

IV: 96-bit Initialization Vector
A: 128-bit Associated data
P: 128-bit Plaintext
C: 128-bit Ciphertext
L: Message length
MAC: Message Authentication Code
Key:128-bit Key

Specifications:

AES-GCM accepts four inputs: an AES key, IV, P, and A. AES-GCM generates two outputs: C and MAC. A is used to verify the correctness of the MAC. Fig. 2.5 shows the block diagram of AES-GCM. It contains an AES engine in CTR mode for encryption, while the authentication is performed by the Galois Hash (GHASH) which is composed of chained GF(2^{128}) multipliers.

Encryption and MAC generation:

1. Generation of H by encrypting 0^{128} block

$$H = E(Key, 0^{128}). \tag{2.7}$$

2. Assigning the first value of the counter (CTR[0]) as follows:

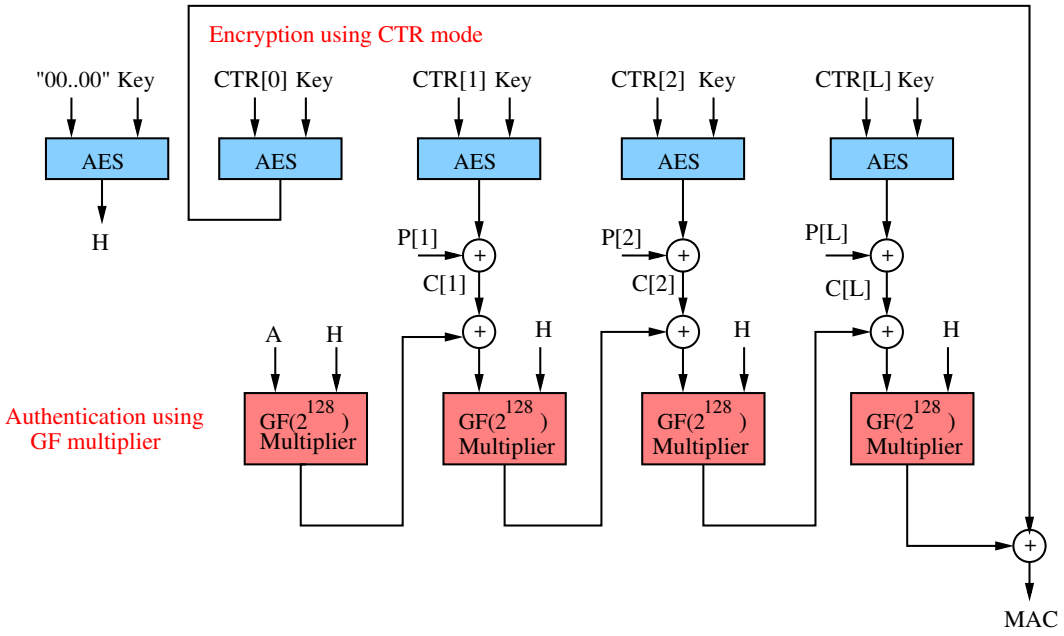

FIGURE 2.5: AES-GCM

$$CTR[0] = \begin{cases} IV \parallel 0^{31}1 & If(length(IV) = 96) \\ H \times IV & otherwise \end{cases}$$

where the multiplication between H and IV is a $GF(2^{128})$-based multiplication.

3. Encryption is performed by AES in CTR mode.

4. Authentication is computed by the $GF(2^{128})$ multipliers chain and the output is XORed with the encrypted value of CTR[0] in order to obtain the final MAC.

Decryption and MAC verification:

1. Generation of H by encrypting 0^{128} block (Equation 2.7).

2. Assigning the first counter value (step 2 in encryption stage)

3. Decryption is performed by XORing the encrypted values of the counter with C to obtain P.

4. MAC verification is accomplished by the same scenario of the encryption process.

2.4.2 Hardware implementation

As we presented before, two main components in AES-GCM are an AES engine and a $GF(2^{128})$ multiplier. Also, the main goal of AES-GCM is to support high speed applications (10 Gbps and higher in hardware). Therefore, most of the previous hardware architectures were proposed to support high speed applications. As a result, the pipelined AES (pipelined AES implementation has been discussed in Section 2.1) is used to perform the encryption task of AES-GCM [3, 30, 31]. The second important part of AES-GCM is the GHASH which depends on the inherent feedback operations chain of the $GF(2^{128})$ multipliers. Therefore, the system performance is always determined by the $GF(2^{128})$ multiplier.

The $GF(2^{128})$ multiplier uses irreducible polynomial $P(x) = x^{128} + x^7 + x^2 + x + 1$ to compute the multiplication between A and H as shown in **Algorithm 1**.

Algorithm 1 GF(2^{128}) multiplier

```
Input A, H ∈ GF(2^128), P(x) Field Polynomial
Output X
X=0
for i = 0 to 127 do
    if A_i = 1 then
        X ⟵ X ⊕ H
    end if
    if H_127 = 0 then
        H ⟵ rightshift(H)
    else
        H ⟵ rightshift(H) ⊕ P(x)
    end if
end for
Return X
```

In [32], [33], and [34], the multiplication was performed using bit-parallel, digit-serial, and hybrid multipliers. Supporting high speed applications motivates using the parallel version which performs the multiplication in only one

clock cycle. In **Algorithm 1**, if H is fixed, the multiplier is called a fixed operand GF(2^{128}) multiplier as shown by [1]. This design proposed by [1] can be used efficiently (smaller area) on FPGAs as the circuit is specialized for H. The drawback in [1] is the need for a new reconfiguration in case of changing the key.

Karatsuba Ofman Algorithm (KOA) [35] is used to reduce the complexity (consumed area) of the GF(2^{128}) multiplier. More precisely, the single step KOA algorithm splits two m bit inputs A and B into four terms A_h, A_l, B_h, B_l which are $m/2$ bit terms. The 1-step iteration of KOA shown in Fig. 2.6 can be described as:

$$\begin{cases} D_l &= A_l \times B_l \\ D_{hl} &= (A_h \oplus A_l) \times (B_h \oplus B_l) \\ D_h &= A_h \times B_h \\ D &= D_h X^m \oplus X^{m/2}(D_h \oplus D_{hl} \oplus D_l) \oplus D_l. \end{cases} \tag{2.8}$$

After the multiplication stage is processed using KOA, the binary field reduction step is used to convert the length of the vector from $2m-1$ to m as shown in Equation 2.9.

$$C(x) = \quad D \quad mod \quad P(x) \tag{2.9}$$

where P(x) is the field polynomial used for the multiplication operation.

KOA was used by [30] to reduce the complexity (consumed area) of the GF(2^{128}) multiplier as shown in Fig. 2.7(a). Because the GHASH is a chain of the $GF(2^{128})$ multipliers, the GHASH output (Fig. 2.7(a)) is calculated as follows:

$$Output = (C_i \oplus Z_{i-1}) \times H. \tag{2.10}$$

The drawback of the architecture presented in [30] is the critical delay resulting from the multiplication stage. In order to reduce the data path (critical delay)

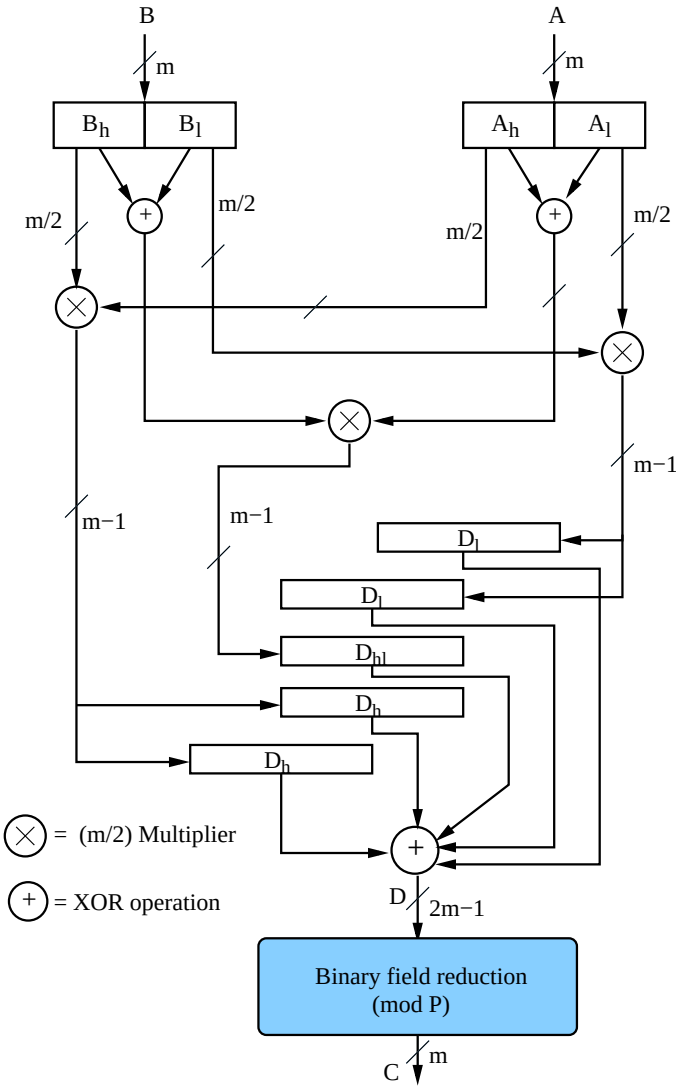

FIGURE 2.6: Polynomial Multiplication using KOA

of the KOA multiplier, pipelining concept was accomplished by [3] as shown in Fig. 2.7(b). Equation 2.10 was written by [3] as follows:

$$Output = Q_1 \oplus Q_2 \oplus Q_3 \oplus Q_4, where \tag{2.11}$$

$$Q1 = (((C_1 \times H^4 \oplus C_5) \times H^4 \oplus C_9) \times H^4 \oplus ....) \times H^4 \tag{2.12}$$

$$Q2 = (((C_2 \times H^4 \oplus C_6) \times H^4 \oplus C_{10}) \times H^4 \oplus ....) \times H^3 \tag{2.13}$$

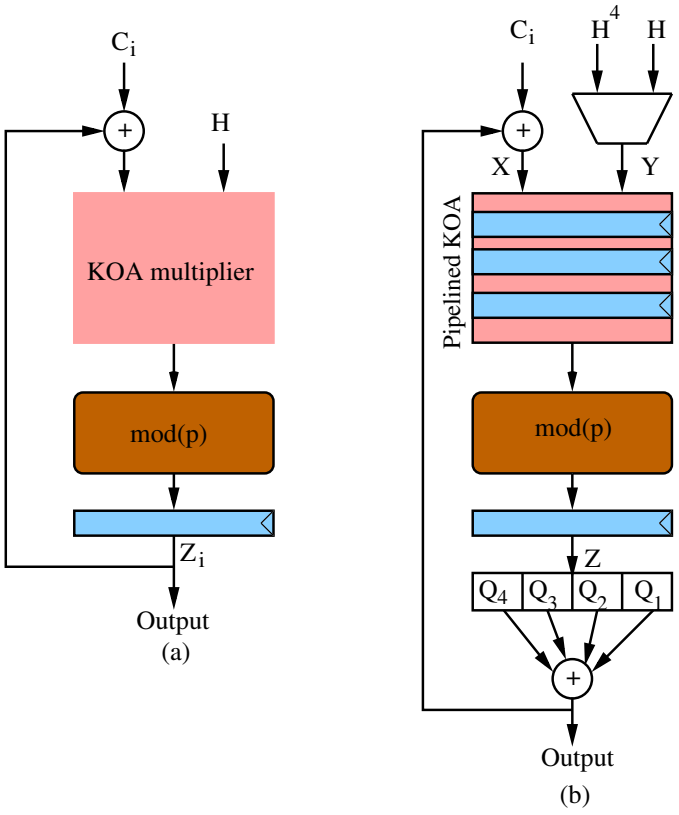

FIGURE 2.7: (a) KOA based GHAH. (b) Pipelined KOA based GHASH

$$Q3 = (((C_3 \times H^4 \oplus C_7) \times H^4 \oplus C_{11}) \times H^4 \oplus ....) \times H^2 \tag{2.14}$$

$$Q4 = (((C_4 \times H^4 \oplus C_8) \times H^4 \oplus C_{12}) \times H^4 \oplus ....) \times H. \tag{2.15}$$

The hardware architecture proposed by [3] (Fig. 2.7(b)) is a 4-stage pipelined KOA-based GHASH. An example of data flow control for the GHASH is shown in Table 2.3, where $C1$ .... C_8 is the input sequence and "-" denotes "don't care". At the beginning, H^4 is passed to port Y. After the input of C_6, H is passed to port Y. The partial GHASH values Q_1, Q_2, Q_3, and Q_4 are ready at the 9^{th}, 15^{th}, 18^{th}, and 12^{th} clock, respectively. As shown from Table 2.3, the multiplier output resulting from 8 frames of 128-bit is ready after 19 clock cycles. Therefore, the real throughput is calculated as follows:

TABLE 2.3: Data flow control for GHASH calculation by [3]

Clock	C_i	X	Y	Z	Comment
1	C_1	C_1	H^4	0	
2	C_2	C_2	H^4	0	
3	C_3	C_3	H^4	0	
4	C_4	C_4	H^4	0	
5	C_5	$(C_1 \times H^4) \oplus C_5$	H^4	$C_1 \times H^4$	
6	C_6	$(C_2 \times H^4) \oplus C_6$	H	$C_2 \times H^4$	
7	C_7	$(C_3 \times H^4) \oplus C_7$	H	$C_3 \times H^4$	
8	C_8	$(C_4 \times H^4) \oplus C_8$	H	$C_4 \times H^4$	
9	-	-	-	$((C_1 \times H^4) \oplus C_5)H^4$	$z = Q_1$
10	0	$((C_2 \times H^4) \oplus C_6) \times H$	H	$((C_2 \times H^4) \oplus C_6) \times H$	
11	0	$((C_3 \times H^4) \oplus C_7) \times H$	H	$((C_3 \times H^4) \oplus C_7) \times H$	
12	0	-	-	$((C_4 \times H^4) \oplus C_8) \times H$	$z = Q_4$
13	0	-	-	-	
14	0	$((C_2 \times H^4) \oplus C_6) \times H^2$	H	$((C_2 \times H^4) \oplus C_6) \times H^2$	
15	-	-	-	$((C_3 \times H^4) \oplus C_7) \times H^2$	$z = Q_2$
16	-	-	-	-	
17	-	-	-	-	
18	-	-	-	$((C_2 \times H^4) \oplus C_6) \times H^3$	$z = Q_3$
19	-	-	-	-	GHASH

$$Throughput(Mbps) = F_{max(MHz)} \times 128 \times (\frac{8}{19}). \tag{2.16}$$

The last component of Equation 2.16 is $(\frac{8}{19})$, it is called the reduction factor and the authors of [3] neglected this component in their throughput calculation. Therefore, their presented design of GHASH has not increased the throughput as they claimed.

Henzen et al. [31] proposed 4-parallel AES-GCM using pipelined KOA. Their design achieved the authentication of 18 frames of 128-bits in 11 clock cycles because of the latency resulting from the pipelined KOA. As a result, their throughput is calculated as follows:

$$Throughput(Mbps) = F_{max(MHz)} \times 128 \times \frac{18}{11}. \tag{2.17}$$

The authors of [31] neglected this component $(\frac{18}{11})$ in their throughput calculation and replaced it by 4. Hence, their presented parallel design of GHASH has not increased the throughput by 4 as shown in Equation 2.17.

Table 2.4 shows the hardware comparison of the current AES-GCM cores on FPGAs. It is clear that the maximum throughput resulting from implementing single AES-GCM core on FPGAs (Virtex-5) is 17.9 Gbps with 4628 slices ([3]). In terms of the parallel cores, the maximum throughput is 48.8 Gbps ([31]).

2.5 AEGIS

The field of AE has received more interest in the light of the recently announced CAESAR (Competition for Authenticated Encryption: Security, Applicability, and Robustness). CAESAR will define a portfolio of AE algorithms that offer advantages over AES-GCM. Secure and efficient algorithms for specific or possibly multiple environments will be presented.

There are some AE schemes have been proposed, and more are expected to join the ranks with the ongoing CAESAR. In this section, we present an overview on AEGIS [36] which is considered one of the candidates to CAESAR. It is constructed from the AES encryption round function (not the last round). AEGIS is a stream cipher with a large state which is updated continuously. Therefore, the attacks against a block cipher cannot be applied directly to it. The goal of AEGIS is to achieve high performance and strong security as outlined in [36]. AEGIS-128 processes a 16-byte message block with five AES-round functions, and AEGIS-256 uses 6 AES round functions. More precisely, we will highlight AEGIS-128 because the thesis concentrates on 128-based AE algorithms.

2.5.1 Algorithm specifications

Operations:

$\oplus$: bit-wise exclusive OR
$\&$: bit-wise AND
$\|$: concatenation

Variables and constants:

TABLE 2.4: Hardware comparison of the previous AES-GCM architectures on FPGAs

	FPGA type	Design	SubBytes	Area (slices)	BRAMs	Max-Freq MHz	Thr. Gbit/s	Thr./Slice Mbps/Slice
[3]	Virtex4	AES-GCM	BRAM	7712	82	285	15.4	1.99
[3]	Virtex4	AES-GCM	Comp.	14349	0	277	14.9	1.04
[3]	Virtex5	AES-GCM	BRAM	3533	41	314	16.9	4.78
[3]	Virtex5	AES-GCM	Comp	6492	0	314	16.9	2.60
[3]	Virtex5	AES-GCM	LUT	4628	0	324	17.5	3.77
[32]	Virtex4	AES-GCM	BRAM	13200	114	110	14	1.07
[32]	Virtex4	AES-GCM	Comp.	21600	0	90	11.52	0.53
[31]	Virtex5	4-parallel AES-GCM	BRAM	9561	450	233	48.8	5.1
[31]	Virtex5	4-parallel AES-GCM	Comp	18505	0	233	48.8	2.64
[31]	Virtex5	4-parallel AES-GCM	LUT	14799	0	233	48.8	3.29

The following variables and constants are used in AEGIS-128:

IV_{128}	:	The 128-bit initialization vector
P	:	128-bit plaintext
C	:	128-bit ciphertext
m_i	:	128-bit data block
const0	:	The first 16 bytes of const
const1	:	The second 16 bytes of const
K_{128}	:	The 128-bit key
msglen	:	The bit length of the plaintext

AEGIS depends on the **AESRound (A,B)**, where A is the 16-byte state, B is the 16-byte round key. It is performed using the AES round functions (ShiftRows, SubBytes, MixColumns, and AddRoundKey). With a 128-bit key and a 128-bit initialization vector, AEGIS-128 encrypts and authenticates a message with length less than 2^{64} bits. The authentication MAC length is less than or equal to 128 bits. It is strongly recommend the use of a 128-bit MAC.

The state update function of AEGIS-128:

The state update function shown in Fig. 2.8 updates the 80-byte state S_i with a 16-byte message block m_i.
$S_{i+1} = StateUpdate128(S_i, m_i)$ is given as follows:

$$\begin{aligned} S_{i+1,0} &= AESRound(S_{i,4}, S_{i,0} \oplus m_i) \\ S_{i+1,1} &= AESRound(S_{i,0}, S_{i,1}) \\ S_{i+1,2} &= AESRound(S_{i,1}, S_{i,2}) \\ S_{i+1,3} &= AESRound(S_{i,2}, S_{i,3}) \\ S_{i+1,4} &= AESRound(S_{i,3}, S_{i,4}). \end{aligned} \tag{2.18}$$

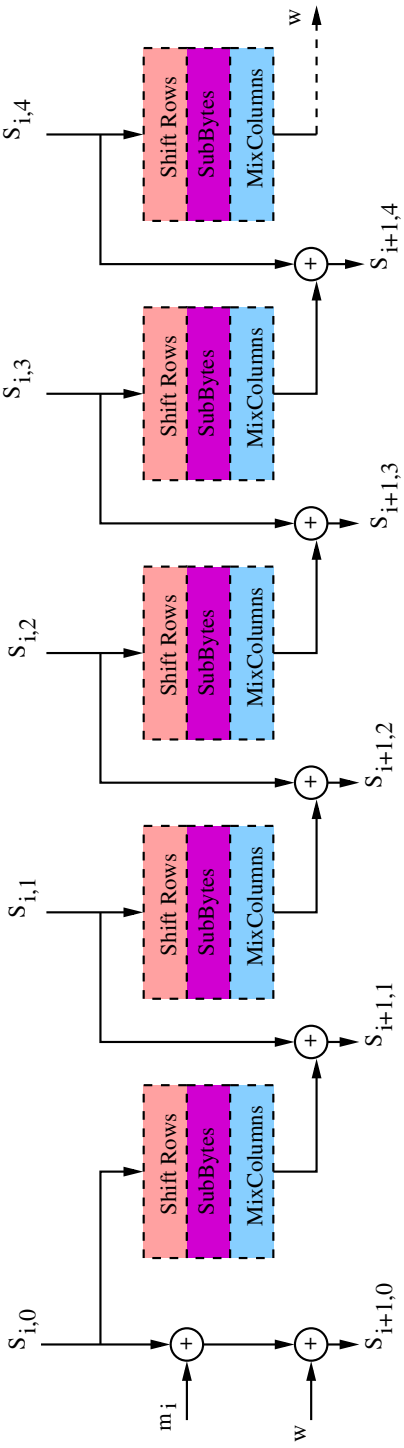

FIGURE 2.8: The state update function of AEGIS-128

The initialization of AEGIS-128:

The initialization of AEGIS-128 consists of loading the key and IV into the state, and running the cipher for 10 steps with the key and IV being used as message.

1. Load the key and IV into the state as follows:
   $$\begin{aligned} S_{-10,0} &= IV_{128} \\ S_{-10,1} &= Const1 \\ S_{-10,2} &= Const0 \\ S_{-10,3} &= K_{128} \oplus Const0 \\ S_{-10,4} &= K_{128} \oplus Const1. \end{aligned}$$

2. For i = -5 to -1, $m_{2i} = K_{128}$, $m_{2i+1} = K_{128} \oplus IV_{128}$.

3. For i = -10 to -1, $S_{i+1} = StateUpdate128(S_i, m_i)$.

The encryption of AEGIS-128:

After the initialization, at each step of the encryption, a 16-byte plaintext block P_i is used to update the state, and P_i is encrypted to C_i. If the size of the last message block is less than 128 bits, it is padded with 0 bits to a full block, and the padded full block is used to update the state.

1. If the last plaintext block is not a full block, use 0 bits to pad it to 128 bits.

2. For i = 0 to $(\frac{msglen}{128} - 1)$, the state is updated to perform encryption.

$$\begin{aligned} C_i &= P_i \oplus S_{i,1} \oplus S_{i,4} \oplus (S_{i,2} \& S_{i,3}) \\ S_{i+1} &= StateUpdate128(S_i, P_i). \end{aligned} \tag{2.19}$$

The finalization of AEGIS-128 (MAC generation):

After encrypting all the plaintext blocks, the authentication MAC is generated using seven more steps. The message being used at this stage is part of the state at the end of the encryption, together with the length of the associated data and the length of the message.

1. Let $tmp = lenA \| msglen$, where lenA and msglen are represented as 64-bit integers

2. For i= $(\frac{msglen}{128})$ to $(\frac{msglen}{128} + 6)$, $m_i = S_{\frac{msglen}{128},3} \oplus tmp$

3. For i= $(\frac{msglen}{128})$ to $(\frac{msglen}{128} + 6)$, the state is updated:
   $S_{i+1} = StateUpdate128(S_i, P_i)$

4. The authentication MAC is generated from the state $\frac{msglen}{128}+7$ as follows:

$$MAC \quad = \oplus_{i=0}^{4}(S_{(\frac{msglen}{128}+7),i}). \tag{2.20}$$

The decryption and verification of AEGIS-128:

The exact values of the key and the IV should be known to the decryption and verification process. The decryption is similar to encryption, and it is described below:

1. For i = 0 to $(\frac{msglen}{128} - 1)$, the state is updated to perform the decryption.

$$\begin{aligned} P_i \quad &= C_i \oplus S_{i,1} \oplus S_{i,4} \oplus (S_{i,2} \& S_{i,3}) \\ S_{i+1} \quad &= StateUpdate128(S_i, P_i). \end{aligned} \tag{2.21}$$

2. The finalization in the decryption process is the same as that in the encryption process (authentication).

In terms of the hardware implementation, to our knowledge, there is no any current hardware implementation for AEGIS.

2.6 Conclusion

In this chapter, we highlighted current AE algorithms, AES-CCM, AES-GCM, and AEGIS. Each algorithm was presented in detail and also followed by the current hardware implementations. Regarding AES-CCM, we showed the disadvantages of AES-CCM. It is not suitable for online applications as the message must be stored first before performing the authentication and the encryption. Also, it is not suited to high-speed implementations, because CBC-MAC is neither pipelinable nor parallelizable (see Table 2.2). In terms of AES-GCM, four different FPGAs-based architectures have been presented in the open literature ([32],[30],[3],[31]). It is clear that these contributions do generally have common challenge related to the throughput of their architectures (see Equation 2.16, Equation 2.17, and Table 2.4). Additionally, we presented an overview of AEGIS which is considered one of the candidates to CAESAR. Till now, there is no any hardware implementation for AEGIS.

Chapter 3

High Speed Authenticated Encryption for Slow Changing Key Applications Using FPGAs

3.1 Introduction

Virtual Private Networks (VPNs) offer an economic alternative to leased lines for building a private network. It is set up by allowing users to tunnel through the public network in a manner that manages the tunnel participants to enjoy a secure connection if they are on a typical private network. VPN tunnels can be either static (between two routers) or dynamic (between end-users and routers). VPNs use AES-GCM as a solution for protecting confidentiality and authenticity. Commercial security appliances of VPNs allow a throughput from 40 to 60 Gbps [37, 38]. Recently, the Cisco ASR 1000 Series Embedded Services Processors (ESPs) are used to support high throughput VPNs up to 200 Gbps [39]. The secret key used for encryption and authentication in these networks is changed weekly, monthly or yearly. Therefore, they are considered infrequent-key based applications.

This chapter describes the benefits of VPNs feature as an infrequent-key environment in order to design efficient and high speed AES-GCM. As the targeted platform is FPGA, FPGA-specific properties are considered for performance

improvement such as programmability, BRAMs, and LUT technology. The main contributions of this chapter are as follows:

1. As a first step towards an efficient high speed AES-GCM for VPNs, which are considered as an infrequent-key application, the key used for encryption and authentication is synthesized into the module structure of AES-GCM. This is achieved by combining the proposed key-synthesized AES (encryption) with the GF(2^{128}) multiplier (authentication) proposed by [1] in order to improve the hardware performance (Thr./Slice) of AES-GCM compared to [3, 30, 32].

2. We propose an efficient method for implementing parallel AES-GCM cores. The proposed method improves the performance (Thr./Slice) of the parallel hardware architectures compared to [31].

3. Because of being key-synthesized architectures, we present a protocol to secure the reconfiguration of the proposed architectures on FPGAs.

3.2 High Speed AES-GCM Architectures Using FPGAs

AES has a key expansion or key schedule operation, which takes the main key and derives from it subkeys K_r (10, 12, and 14 for AES-128, AES-192, and AES-256, respectively), where **r** denotes the corresponding round number. For our case, we concentrate on AES-128.

VPNs are considered as a slow key changing application. Therefore, implementing the key expansion is particularly expensive in terms of hardware cost in case of using FPGAs. By getting the benefit of being programmable devices, the key used for encryption in synthesized into the architecture in order to obtain better performance regarding the consumed area and the throughput.

In the proposed hardware architecture, constant key specialization in the FPGA is used. The precomputed keys are generated using a **C** code as shown in Table 3.1. After, these keys are synthesized into the architecture of AES.

As a result, the key expansion scheme is reduced from the architecture of AES. Because of the high throughput target, pipelined AES is used to obtain high throughput. Fig. 3.1 shows the proposed key-synthesized AES, where all keys are precomputed and synthesized into the architecture.

TABLE 3.1: Precomputed round keys

Main Key	000102030405060708090a0b0c0d0e0f
Precomputed k0	000102030405060708090a0b0c0d0e0f
Precomputed k1	d6aa74fdd2af72fadaa678f1d6ab76fe
Precomputed k2	b692cf0b643dbdf1be9bc5006830b3fe
Precomputed k3	b6ff744ed2c2c9bf6c590cbf0469bf41
Precomputed k4	47f7f7bc95353e03f96c32bcfd058dfd
Precomputed k5	3caaa3e8a99f9deb50f3af57adf622aa
Precomputed k6	5e390f7df7a69296a7553dc10aa31f6b
Precomputed k7	14f9701ae35fe28c440adf4d4ea9c026
Precomputed k8	47438735a41c65b9e016baf4aebf7ad2
Precomputed k9	549932d1f08557681093ed9cbe2c974e
Precomputed k10	13111d7fe3944a17f307a78b4d2b30c5
Precomputed H	c6a13b37878f5b826f4f8162a1c8d879

The SubBytes transformation can be implemented either by BRAMs, composite field or direct Look Up Tables (LUT). Modern FPGAs contain Block-RAMs. Therefore, implementing SubBytes using BRAMs decreases the consumed slices of the FPGA. The LUT approach is especially interesting on Virtex-5 devices because 6-input Look-Up-Tables (LUT) combined with multiplexors allow an efficient implementation of the AES SubBytes stage. Composite field approach uses the multiplicative inverse of GF(2^8) and it is efficient for memoryless platforms (see Fig. 3.2).

As a result of using key-synthesized AES, the operand H of the GHASH function ($GF(2^{128})$ multiplier) is also fixed because it is generated by applying the block cipher to the zero block. Therefore, the proposed multiplier by [1] is very suitable because it is a fixed operand multiplier. In [1], the multiplication was performed as follows (Fig. 3.3) :

1. **Algorithm 1** is divided into **Algorithm 2** and **Algorithm 3**.

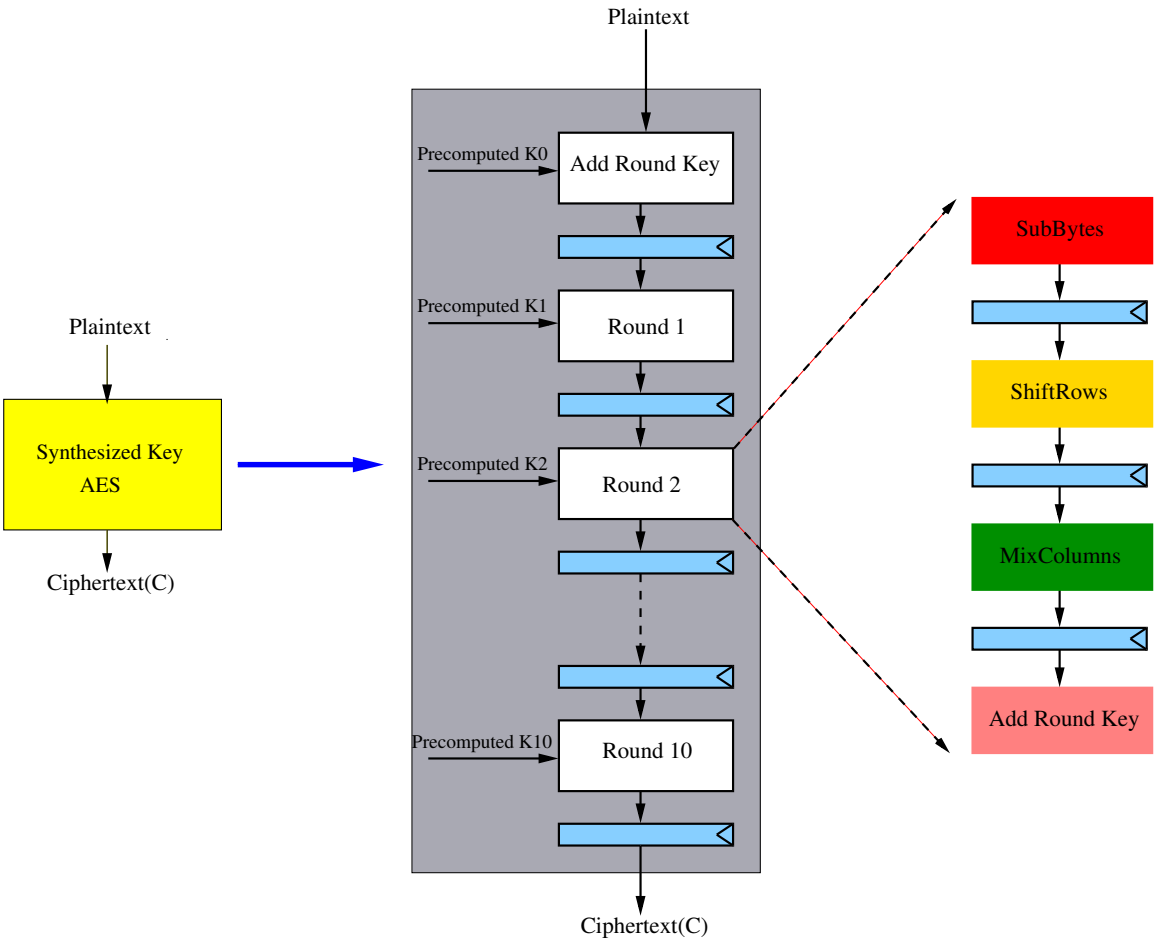

FIGURE 3.1: Key-synthesized AES

2. **Algorithm 2** is used to precompute the lookup table (T) which is based on a fixed H.

3. Performing the multiplication using the lookup table (T) (see **Algorithm 3**).

The lookup table generated by **Algorithm 2** contains 128 vectors of 128 bits. This table is synthesized into the architecture of the multiplier by **Algorithm 3** to compute the GF(2^{128}) multiplication. Synthesizing binary 1 values of table T directly perform logic and binary 0 values do not perform logic because of XOR operation as shown in **Algorithm 3**. Therefore, the consumed area and the datapath are reduced.

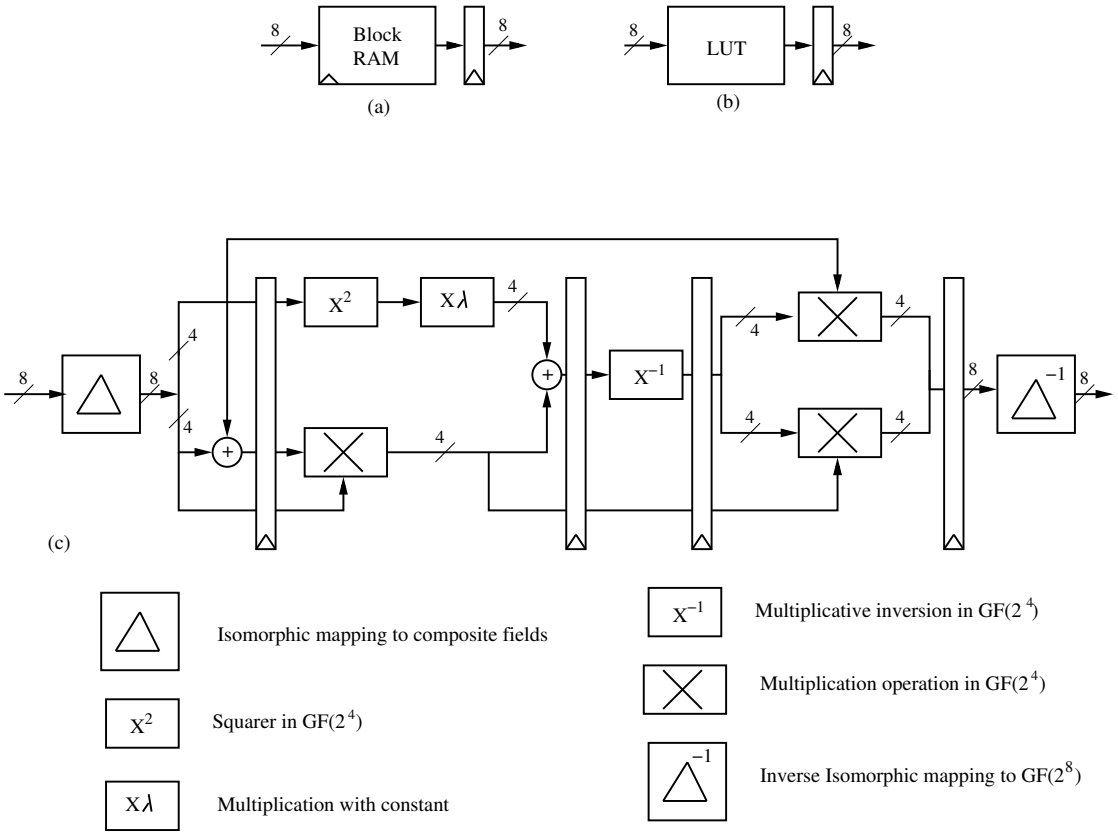

FIGURE 3.2: SubBytes implementation with BlockRAMs (a), with LUTs (b), with composite field approach (c)

The overall architecture of AES-GCM is presented in Fig. 3.4. The value of CTR[0] is encrypted and stored because it will be used for MAC generation. After that, the encryption process is performed using CTR mode, where the CTR values are encrypted and XORed with the Plaintext (P). Encrypted frames are then processed by the H-synthesized GHASH till the end of the data. After processing all the data frames, the last output of the GHASH is XORed with E(CTR[0]) in order to generate the MAC.

From Fig. 3.4, it is clear that the encryption and the authentication in GCM are performed using the AES in CTR mode and the H-synthesized GHASH, respectively. Therefore, the proposed architecture could also be tuned to handle the decryption and the authentication. Precisely, Ciphertext (C) is XORed with the output of the pipelined AES for performing the decryption and also is passed to the H-synthesized GHASH for MAC generation.

FIGURE 3.3: GF(2^{128}) multiplier proposed by [1]

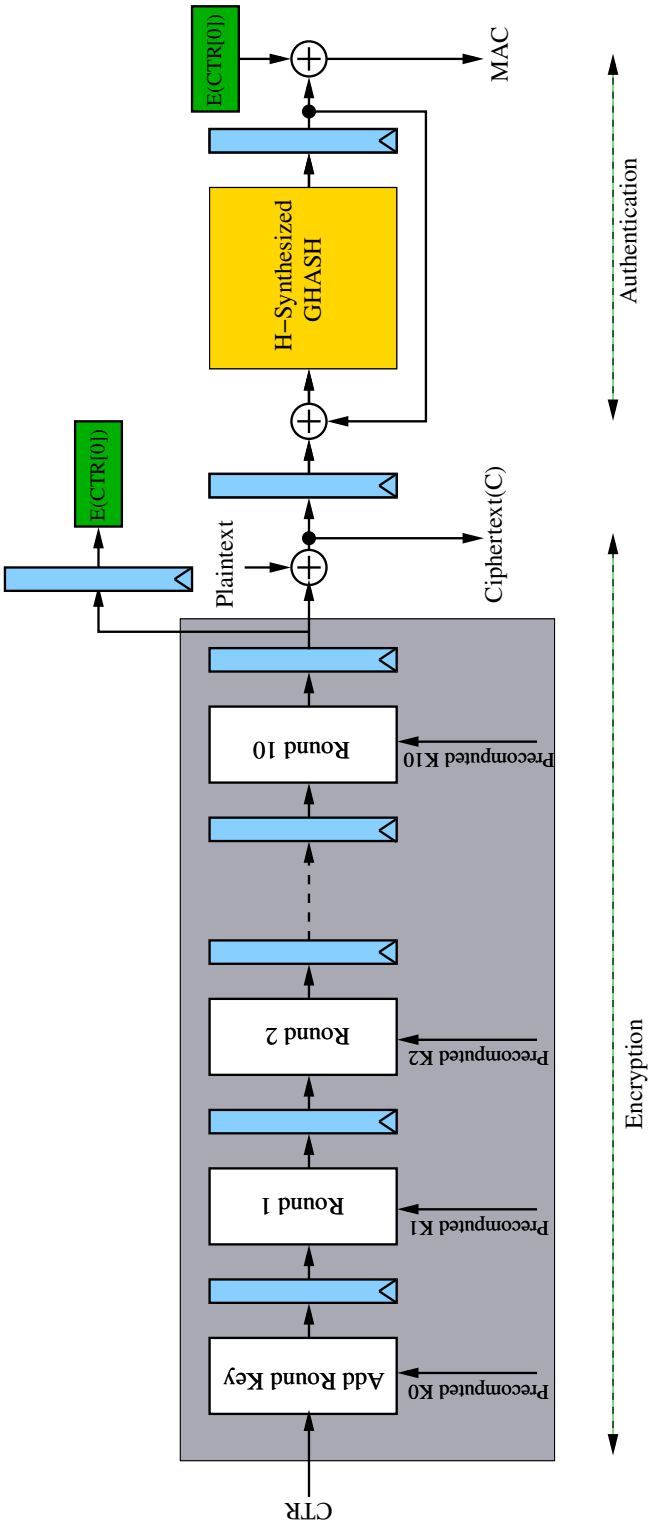

FIGURE 3.4: Proposed key-synthesized AES-GCM

The proposed architecture limits the logic utilization by specializing the core of AES-GCM on a per key. VPNs infrastructure can benefit from our key-synthesized AES-GCM implementation due to the nature of slow changing key operation.

3.2.1 Efficient Parallel AES-GCM cores

In order to implement parallel architectures of AES-GCM using key-synthesized method, parallel GHASHs must be constructed to meet the requirement of the key-synthesized nature (i.e, one of the two operands of each GHASH must be fixed).

Previous parallel schemes of GHASH [31, 34] are not suitable because the two operands of each GHASH are varied during the running time operation. As a result, their architectures are not suitable for key-synthesized approach. Also, they have the same common problem in the throughput reduction factor (described in Chapter 2). Therefore, constructing parallel GHASHs which have a fixed operand for each GHASH multiplier is very important for high speed applications.

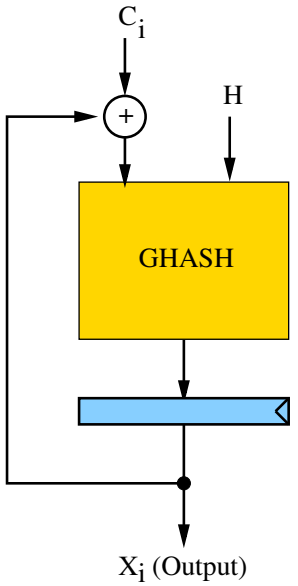

FIGURE 3.5: GHASH operation

Fig. 3.5 shows the GF(2^{128}) multiplication (GHASH) between **H** and a 128-bit input value C_i. $GHASH_H$ function for block i is defined in Equation 3.1.

$$X_i = (C_i \oplus X_{i-1}) \times H \tag{3.1}$$

In order to construct parallel GHASHs which have a fixed operand for each GHASH multiplier, we have to first analyze the processing equation of the GHASH. As shown in Equation 3.1, the the GHASH output is calculated by the multiplication between H and the result of XORing the input C_i with the previous output X_{i-1}. We propose writing Equation 3.1 as follows:

$$\begin{aligned} X_i &= (C_i \oplus X_{i-1}) \times H \\ &= (C_i \times H) \oplus (X_{i-1} \times H) \\ &= (C_i \times H) \oplus [(C_{i-1} \oplus X_{i-2}) \times H^2] \\ &= (C_i \times H) \oplus (C_{i-1} \times H^2) \oplus [(C_{i-2} \oplus X_{i-3}) \times H^3] \\ &= (C_i \times H) \oplus (C_{i-1} \times H^2) \oplus (C_{i-2} \times H^3) \\ &\oplus [(C_{i-3} \oplus X_{i-4}) \times H^4] \\ &= (\underbrace{(C_i \times H)} \oplus \underbrace{(C_{i-1} \times H^2)} \oplus \underbrace{(C_{i-2} \times H^3)} \\ &\oplus \underbrace{(C_{i-3} \times H^4)} .... \oplus \underbrace{(C_2 \times H^{i-1})} \oplus \underbrace{(C_1 \times H^i)}. \end{aligned} \tag{3.2}$$

It is clear from Equation 3.2 that each input from C_1 to C_i is multiplied with a fixed value from H^i to H. For example (see Fig. 3.6), $GHASH_i$ has H as an operand, $GHASH_{i-1}$ has H^2, ..., and $GHASH_1$ has H^i .

Unlike previous work, this method makes the parallel architecture of GHASHs suitable for the key-synthesized approach as we can get these values (H^i, H^{i-1}, ..., H) synthesized into the parallel architecture.

In terms of the throughput, from Fig. 3.6, it is obvious that only one clock cycle is needed for performing the parallel GHASHs output (there is no pipeline). Therefore, the throughput is calculated as follows:

$$Throughput(Mbps) = F_{max(MHz)} \times 128 \times i \tag{3.3}$$

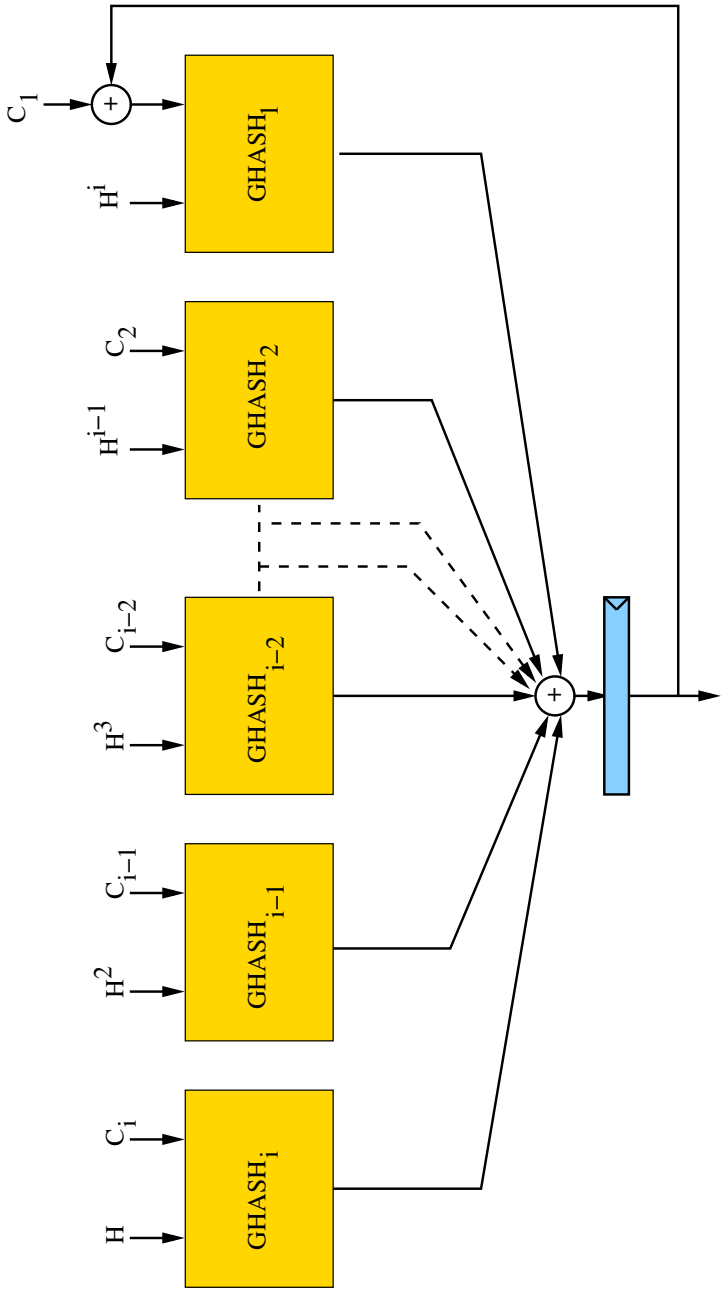

FIGURE 3.6: Proposed parallel GHASHs with fixed operand during running time operation

where i is the number of the parallel GHASHs.

From Equation 3.3, it is clear that there is no reduction factor in the throughput calculation compared to [31, 34].

In order to support high speed requirements up to 100 Gbps, we are forced to construct 4-parallel AES-GCM cores. Fig. 3.7 shows the presented 4-parallel AES-GCM using key-synthesized method.

In terms of the parallel pipelined AES, we instantiated four parallel AES cores which have the same key. Therefore, the round keys are precomputed and synthesized into the four cores. The resulting multi-core design is thus able to process a 512-bit block (4128 bits) of plaintext every clock cycle. The ciphertext is generated directly by XORing the four 128-bit blocks (P_{4i+1}, P_{4i+2}, P_{4i+3}, and P_{4i+4}) with the output strings resulting from encrypting the counter values.

In order to combine the authentication core with the multi-core AES, a design solution based on four parallel GHASHs has been investigated. From Equation 3.2, in order to construct 4-parallel GHASHs, the operands (H, H^2, H^3, H^4) are precomputed and synthesized into the architecture. The multi-core GHASH is able to process a 512-bit block (4128 bits) every clock cycle. Therefore the final overall throughput of the 4-parallel AES-GCM is as follows:

$$Throughput(Mbps) = F_{max(MHz)} \times 128 \times 4. \tag{3.4}$$

The presented architecture does not contain any reduction factor compared to [31, 34]. Also, it could be tuned to handle the decryption and authentication as we described before in the single architecture.

All the previous proposed architectures in this chapter are considered as FPGA-based architectures because of using the programmability of FPGAs in order to change the key. More precisely, in case of changing the key, a new bitstream is uploaded on the FPGA with the new key-synthesized specialization. Therefore, it is important to present how these architectures are uploaded on the FPGA in a secure manner in case of changing the key because the bitstream is considered a key-based bitstream.

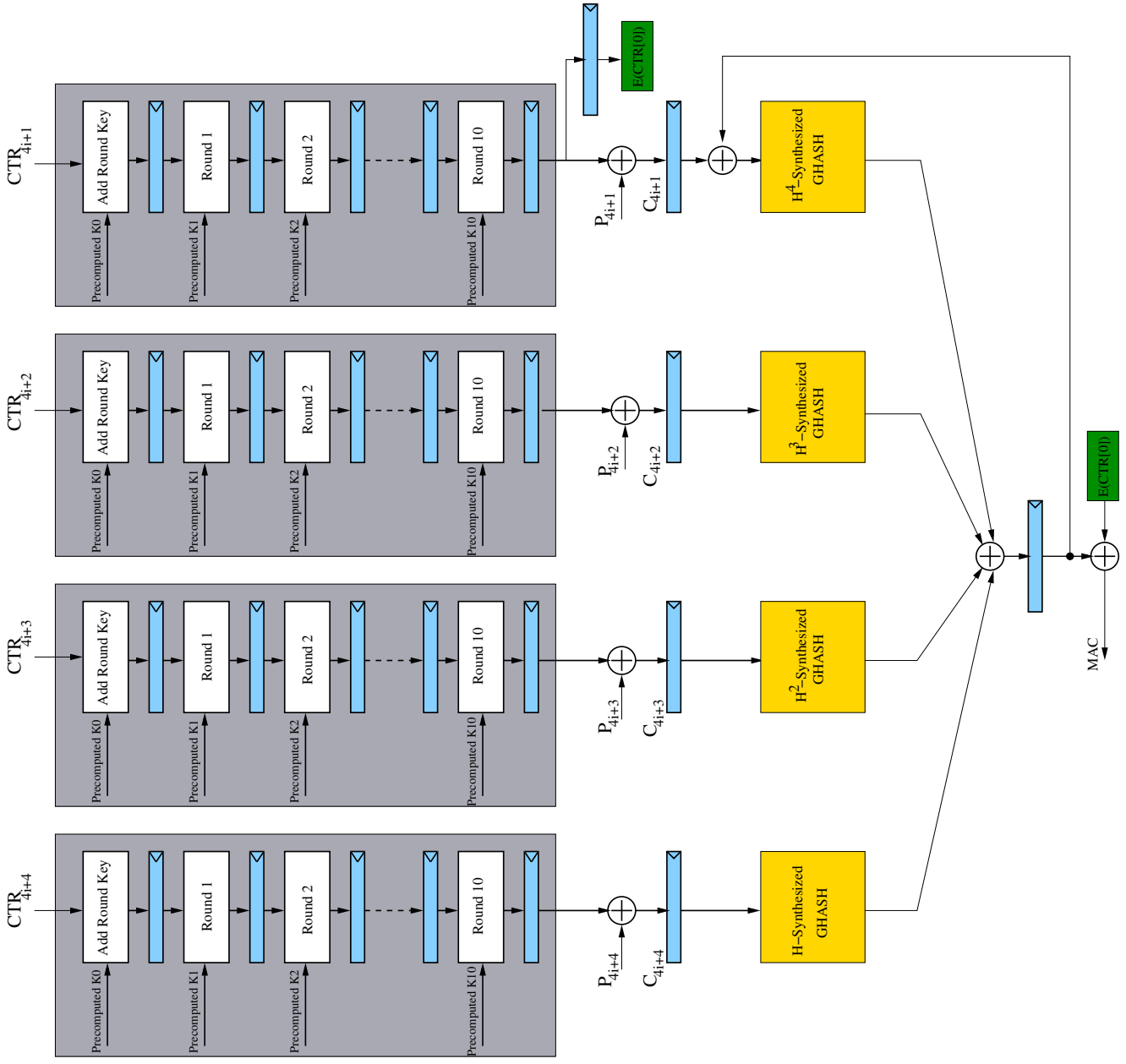

FIGURE 3.7: 4-parallel AES-GCM using key-synthesized method

3.3 Bitstream security of the proposed architectures

As a result of synthesizing the key into the architecture, the generated bitstream is a key-dependent. Hence, the bitstream must be sent in a secure way to the FPGA in case of changing the key. Our analysis focuses on securing the key exchange and the implementation of the key-dependent bitstream on the FPGA.

Fig. 3.8 shows the proposed protocol which is used to perform the key exchange and the reconfiguration process between two FPGAs in a secure way. Our scheme assumes that two FPGAs in two different networks are involved in a communication and the proposed AES-GCM is implemented. In case of changing the key to be k1, the protocol will be as follows:

1. The two servers are communicating in order to initialize the key (k1) of AES-GCM. This initialization is performed using public key cryptography.
2. The two servers generate the bitstream file which is k1-synthesized.
3. Server1 shares k3 with FPGA1 for secure reconfiguration. This key (k3) is used to encrypt the k1-synthesized bitstream. The same will be with server2 which shares k2 with FPGA2. Thanks to Xilinx because Virtex-5 and Virtex-4 contain an AES engine for supporting secure reconfiguration, in case of Virtex-6, the bitstram can be also authenticated because Virtex-6 has an on chip MAC for supporting authentication.
4. The two FPGAs decrypt the encrypted bitstream.
5. The k1-synthesized bitstream which defines the new key-based AES-GCM is implemented on the user logic.
6. The communication between the two FPGAs is achieved with the new key k1.

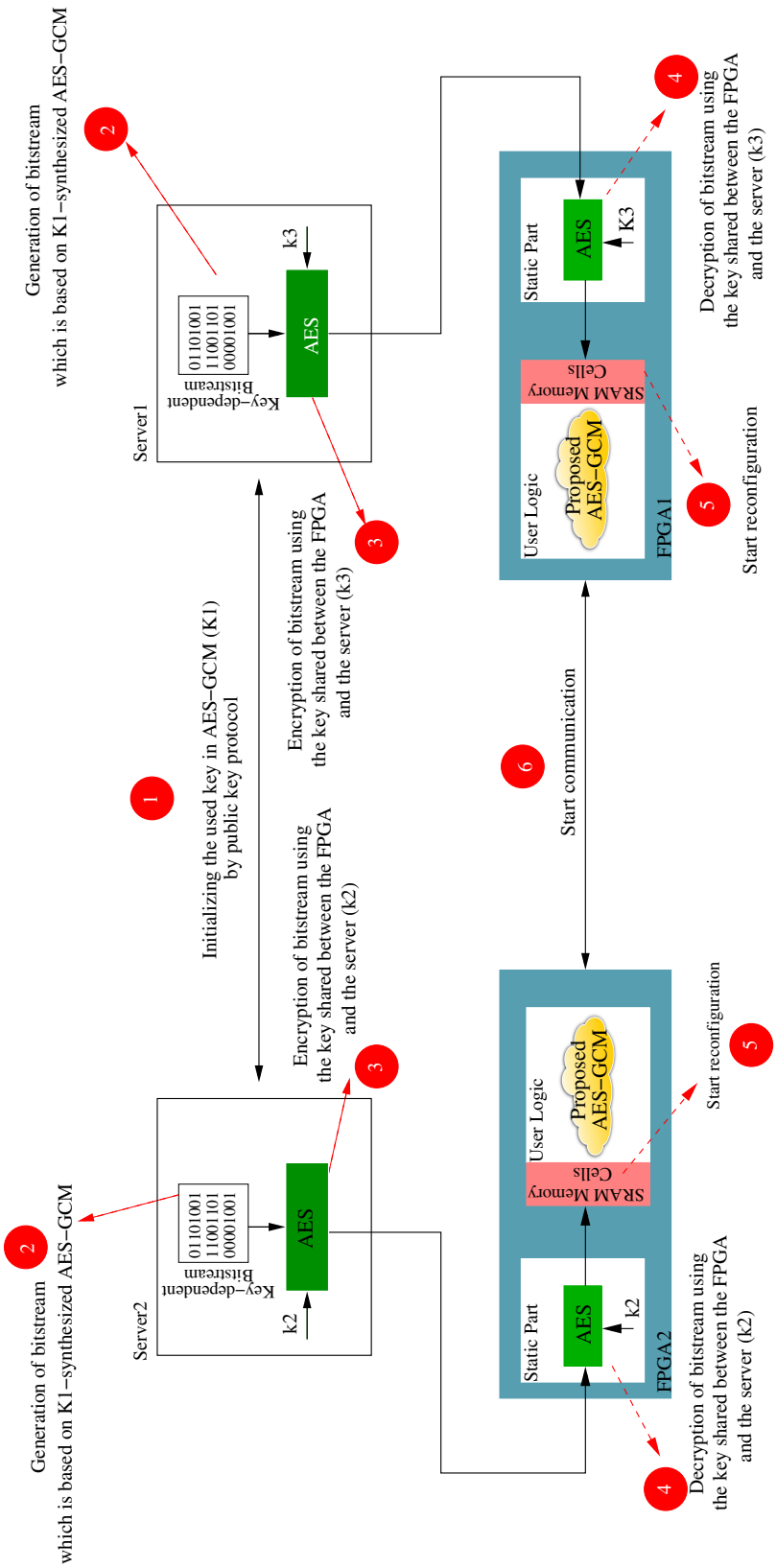

FIGURE 3.8: Secure bitstream communication

3.4 Hardware comparison

We coded our proposed schemes (AES-GCM and 4-parallel AES-GCM) in VHDL and targeted to Virtex-4 (V4LX60ff668-11) and Virtex-5 (XC5VLX220). ModelSim 6.5c was used for simulation. Xilinx Synthesize Technology (XST) is used to perform the synthesize and ISE9.2 was adopted to run the Place And Route (PAR).

Table 3.2 shows the hardware comparison between our results and previous work. Note the filled dots in the "Key" column. Key is synthesized into the architecture when denoted by ∘, otherwise, the key schedule is implemented when denoted by •.

On Virtex-4 platform, our key-synthesized AES-GCM core reaches the throughput of 27.7 Gbps with the area consumption of 4652 slices and 80 BRAMs. In case of using composite field SubBytes, it consumes twice more slices, however no BRAMs are required. On Virtex5, the most efficient implementation reaches the throughput 30.9 Gbps with 2478 slices and 40 BRAMs. Our implementations are technology independent and can be implemented to other FPGA devices.

By comparing our results of AES-GCM to the previous work, the comparison shows that our performance (Thr./Slice) is better than [30],[3],[32]. The operating frequency presented by [3] is better than ours because they used pipelined KOA but the overall throughput is lower than ours because their GHASH performed the multiplication of 8 frames of 128-bits in 19 clock cycles. Therefore, their throughput is calculated as follows (highlighted before in Chapter 2):

$$Throughput(Mbps) = F_{max(MHz)} \times 128 \times \frac{8}{19}. \tag{3.5}$$

We motivate using LUT method for parallel AES-GCM in case of using Virtex-5 to avoid the limit of BRAMs, especially the AES-GCM is always considered as a part of the system. The proposed 4-parallel AES-GCM module operates at 200 MHz on Virtex-5. In total, it consumes 12152 Slices. This implementation achieves throughput that reaches to 102.4 Gbps. Henzen et al. [31]

TABLE 3.2: Hardware comparison

	FPGA type	Design	Key	SubBytes	Slices	BRAM	Max-Freq MHz	Thr. Gbit/s	Thr./Slice Mbps/Slice
This work	Virtex4	AES-GCM	○	BRAM	4652	80	216.3	27.7	5.95
This work	Virtex4	AES-GCM	○	Comp.	10316	0	239	30.6	2.96
This work	Virtex5	AES-GCM	○	BRAM	2478	40	242	30.9	**12.5**
This work	Virtex5	AES-GCM	○	Comp.	5512	0	232	29.7	5.38
This work	Virtex5	AES-GCM	○	LUT	3211	0	216.3	27.7	8.62
This work	Virtex5	4-parallel AES-GCM	○	LUT	12152	0	200	102.4	8.42
[3]	Virtex4	AES-GCM	●	BRAM	7712	82	285	15.4	1.99
[3]	Virtex4	AES-GCM	●	Comp.	14349	0	277	14.9	1.04
[3]	Virtex5	AES-GCM	●	BRAM	3533	41	314	16.9	4.78
[3]	Virtex5	AES-GCM	●	Comp	6492	0	314	16.9	2.60
[3]	Virtex5	AES-GCM	●	LUT	4628	0	324	17.5	3.77
[30]	Virtex4	AES-GCM	●	Comp.	16378	0	161	20.61	1.26
[32]	Virtex4	AES-GCM	●	BRAM	13200	114	110	14	1.07
[32]	Virtex4	AES-GCM	●	Comp.	21600	0	90	11.52	0.53
[31]	Virtex5	4-parallel AES-GCM	●	LUT	14799	0	233	48.8	3.29

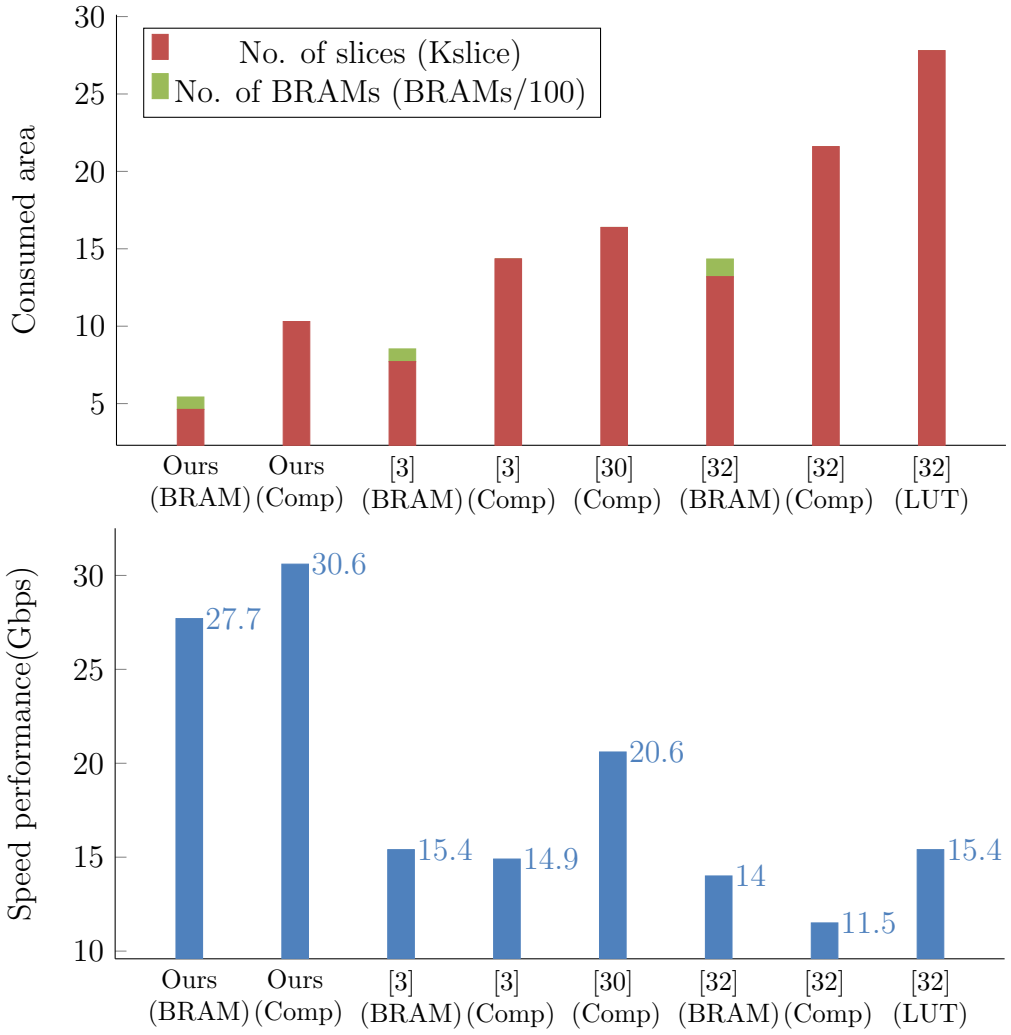

FIGURE 3.9: Hardware comparison on Virtex4

proposed 4-parallel AES-GCM using pipelined KOA. Their design achieved the authentication of 18 frames of 128-bits in 11 clock cycles because of the latency resulting from the pipelined KOA. As a result, their throughput is calculated as follows (highlighted before in Chapter 2):

$$Throughput(Mbps) = F_{max(MHz)} \times 128 \times \frac{18}{11}. \tag{3.6}$$

Fig. 3.9 presents the comparison between our proposed architectures and previous work on Virtex-4. It is clear that our work outperforms the previously reported ones. Therefore, proposed architectures can be used efficiently for slow changing key applications like VPNs and embedded memory protection.

3.5 Conclusion

In this chapter, we presented the performance improvement of AES-GCM by key-synthesized method. We integrated this concept using three methods of SubBytes implementation in order to increase the flexibility of the presented work. With our proposed parallel AES-GCM, each multiplier has a fixed operand. Therefore, the presented parallel AES-GCM is suitable for key-synthesized method with higher throughput compared to the previous work. The bitstream of the proposed architectures contains information related to the used key. Hence, we presented a protocol to protect the bitstream of the proposed architectures. Our presented AES-GCM architectures can be used for slow changing key applications like VPNs and they outperform the previous architectures regarding the hardware performance.

Chapter 4

Efficient and High Speed Key-Independent AES-Based Authenticated Encryption Architectures Using FPGAs

4.1 Introduction

AES-GCM has been utilized in various security applications as a solution for achieving both confidentiality and integrity. As we described before, it consists of two components: an AES engine and a GHASH core. The performance of the system is always determined by the GHASH architecture because of the inherent computation feedback.

The previous chapter invested the slow changing key applications like VPNs in order to design efficient and high speed architectures of AES-GCM by synthesizing the used key for encryption and authentication into the architecture (single and parallel). A new reconfiguration is loaded on the FPGA in case of changing the key. However, this solution presented in Chapter 3 is only for slow changing key applications.

By mid-December 2017, CAESAR, which is funded by NIST, will define a portfolio of AE algorithms that offer advantages over AES-GCM. The due date of the first-round submission for CAESAR was January 15th, 2014. AEGIS is considered one of the candidates to the first round. First, it was presented in SAC-2013 (Selected Area of Cryptography) and then becomes one of the candidates of CAESAR.

In this chapter we are involved in designing high speed architectures of AES-GCM in order to overcome the current challenges regarding the GHASH architecture. Also, we focus on the hardware implementation of AEGIS-128. The contributions of this chapter are as follows:

1. This chapter introduces an efficient method for implementing the pipelined Karatsuba Ofman Algorithm (KOA)-based GHASH on FPGAs to support high speed applications. In particular, the computation feedback is removed by analyzing the complexity of the computation process.
2. The proposed GHASH core is evaluated with three different implementations of AES (BRAMs-based SubBytes, composite field-based SubBytes, and LUT-based SubBytes) in order to perform the encryption and the authentication.
3. Because of being involved in designing high speed AE architectures, an efficient AEGIS-128 architecture is presented using only five AES rounds.

4.2 Efficient KOA-Based GHASH

Four different architectures of FPGAs-based AES-GCM were presented in the open literature ([32],[30],[3],[31]). It is clear that these contributions do generally have a common challenge related to the hardware performance (Thr./Slice) as follows:

- In [32], the authors used the parallel method for implementing the GHASH ($GF(2^{128})$) and the hardware performance is not sufficient to support high speed applications because of using the parallel method.

- Because of the poor results of the hardware performance, the authors of [30] proposed using KOA-based GHASH in order to reduce the hardware complexity and improve the overall performance and their throughput was as follows:

$$Throughput(Mbps) = F_{max(MHz)} \times 128. \tag{4.1}$$

- For better throughput, [3] claimed the throughput improvement to their previous KOA-based GHASH by using the pipelining concept (described before in detail (see Equation 4.2)).

$$Throughput(Mbps) = F_{max(MHz)} \times 128 \times \frac{8}{19} \tag{4.2}$$

- The same happened with the parallel architecture proposed by [31] (see Equation 4.3).

$$Throughput(Mbps) = F_{max(MHz)} \times 128 \times \frac{18}{11} \tag{4.3}$$

Also, in our previous architectures in Chapter 3, all the presented architectures were key-dependent architectures because they are only for slow changing key applications.

In this section, in order to improve the performance of AES-GCM, an efficient pipelined KOA-based GHASH is presented to obtain key-independent AES-GCM. The target is to present an efficient method that solves the problems of the previous work.

In order to reduce the complexity (consumed area), KOA is selected to perform the multiplication process in the GHASH. In terms of the throughput challenge, we present a new method to obtain a feedback-free multiplier. As the targeted platform is FPGA, FPGA-specific properties are considered for performance improvement.

In order to solve the problem of the inherent computation feedback in the multiplier, we have to first analyze the processing equation of the GHASH (Equation 4.4).

$$Output(Z_i) \quad = (C_i \oplus Z_{i-1}) \times H \tag{4.4}$$

As shown in Equation 4.4, the generation of the GHASH output is calculated by the multiplication between H and the result of XORing the input C_i and the previous output Z_{i-1}. We propose writing Equation 4.4 as follows:

$$\begin{aligned} Output \quad &= (C_i \oplus Z_{i-1}) \times H \\ &= (C_i \times H) \oplus (Z_{i-1} \times H) \\ &= (C_i \times H) \oplus [(C_{i-1} \oplus Z_{i-2}) \times H^2] \\ &= (C_i \times H) \oplus (C_{i-1} \times H^2) \oplus [(C_{i-2} \oplus Z_{i-3}) \times H^3] \\ &= (C_i \times H) \oplus (C_{i-1} \times H^2) \oplus (C_{i-2} \times H^3) \\ &\oplus [(C_{i-3} \oplus Z_{i-4}) \times H^4] \\ &= (\underbrace{(C_i \times H)} \oplus \underbrace{(C_{i-1} \times H^2)} \oplus \underbrace{(C_{i-2} \times H^3)} \\ &\oplus \underbrace{(C_{i-3} \times H^4)} .... \oplus \underbrace{(C_2 \times H^{i-1})} \oplus \underbrace{(C_1 \times H^i)} . \end{aligned} \tag{4.5}$$

According to Equation 4.5, the feedback resulting from XORing the input C_i and the previous output Z_{i-1} could be removed because the final output is calculated from the last two lines of the equation. More precisely, if the values from H to H^i exist, the feedback which affects the performance of the multiplier will be removed.

Assume that there is a packet of data which contains 64 frames of 128-bit (i=64, 1024 bytes) and the generation of the output (MAC) is required. Therefore, Equation 4.5 will be as follows:

$$\begin{aligned} Output \quad &= (\underbrace{(C_{64} \times H)} \oplus \underbrace{(C_{63} \times H^2)} \oplus \underbrace{(C_{62} \times H^3)} \\ &\oplus \underbrace{(C_{61} \times H^4)} .... \oplus \underbrace{(C_2 \times H^{63})} \oplus \underbrace{(C_1 \times H^{64})}. \end{aligned} \tag{4.6}$$

If the values form H to H^{64} are stored and multiplied to the input C_i as shown in Equation 4.6, the pipelined architecture can be simply performed. The architecture proposed for the pipelined KOA-based GHASH is shown in Fig. 4.1. The architecture shown in Fig. 4.1 is presented according to calculating the output (MAC) of 64 frames of 128-bit. For area reduction,

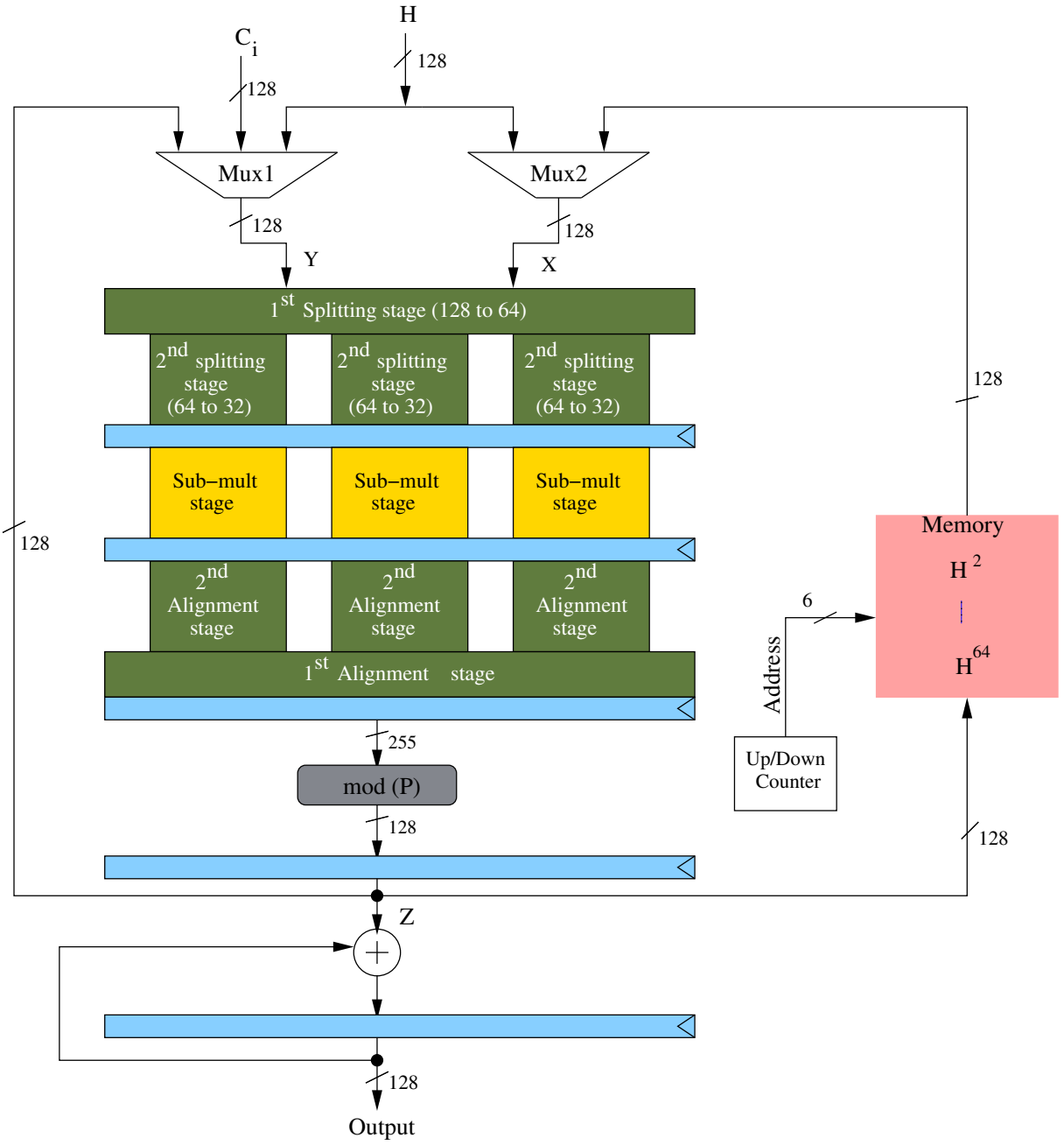

FIGURE 4.1: Proposed pipelined KOA-based GHASH

we propose using 2-step KOA. The two 128-bit inputs (C_i and H) are split two times in order to use 32-bit multipliers. Each sub-mult stage has three 32-bit multipliers. As a result, the complexity of the multiplier is reduced. Additionally, Pipelining approach is used for reducing the datapath of the multiplier. In total, the proposed architecture combines the following parts:

- 2-step pipelined KOA for performing the GF multiplication.
- A memory for storing the values from H^2 to H^{64}.
- 6-bit Up/Down counter for addressing the memory.
- Mux1 and Mux2 are used for switching between H, c_i, Z, and the memory output.

We divide the process of the output (MAC) generation into two steps:

1. Initialization stage:
   The first step includes storing the H values in the memory. At the beginning, H is passed to X and Y ports. The counter counts up and H^2 will appear on port Z after 4 clock cycles because we use 4-stage pipelined KOA. After, the memory stores H^2 and H^2 is passed to port Y and H to port X in order to generate H^3 and store it in the memory. This process is repeated till filling the memory with the values from H^2 to H^{64}. Filling the memory takes $63 \times 4 = 252$ clock cycles. This is called the initialization stage.

2. Output generation:
   After initializing the memory, the second step concerns with the output generation as presented in Equation 4.6. The counter starts counting down with the first input. The first input C_1 is passed to port Y and the memory passes H^{64} to port X. After one clock cycle, the second input C_2 is passed to port Y and the memory passes H^{63} to port X. This scenario is completed by passing C_{64} to port Y and H to port X. The output is calculated by XORing Z values (Equation 4.6). In terms of the time taken to generate the output, it is 64 clock cycles with 5 additional clock cycles as a latency (4 clock cycles because of the 4-stage pipelined KOA and one cycle because of the last register). Therefore, the throughput of the proposed architecture is as follows:

$$Throughput(Mbps) = F_{max(MHz)} \times 128 \times \frac{64}{69}. \tag{4.7}$$

Because of targeting our architecture on Xilinx Virtex-5 FPGAs, we recommend using CLBs for memory implementation because of 6-input Look-Up-Tables (LUT). Otherwise, using BRAMs is another solution.

The proposed architecture offers the following advantages over the previous designs:

- It reduces the reduction factor compared to [3] from $\frac{8}{19}$ to $\frac{64}{69}$ (see Equation 4.7). Therefore, the developed architecture presents the throughput improvement compared to [3].
- In case of changing the key, 252 clock cycles are needed to initialize the memory. Hence, no new reconfiguration is needed in case of changing the key unlike our previous architectures in Chapter 3.

4.3 High speed AES-GCM

This section describes adding the proposed GHASH to the pipelined AES in order to perform the encryption and the authentication of the input message. Fig. 4.2 shows the proposed high throughput architecture for AES-GCM. In contrast to the previously presented AES-GCM architectures in Chapter 3, we use the key-independent pipelined AES core which has the key schedule that generates the sub-keys (k0 to k10) during the running time (no new FPGA reconfiguration). Three SubBytes implementations (BRAMs-based SubBytes, composite field-based SubBytes, and LUT-based SubBytes) are used in order to increase the flexibility of the proposed architecture.

The proposed core performs both encryption and authentication as follows:

1. The pipelined AES engine generates H by encrypting "00..00" frame.
2. The first value of the counter CTR[0] is encrypted.
3. The proposed GHASH needs 252 clock cycles in order to initialize the memory after the generation of H.

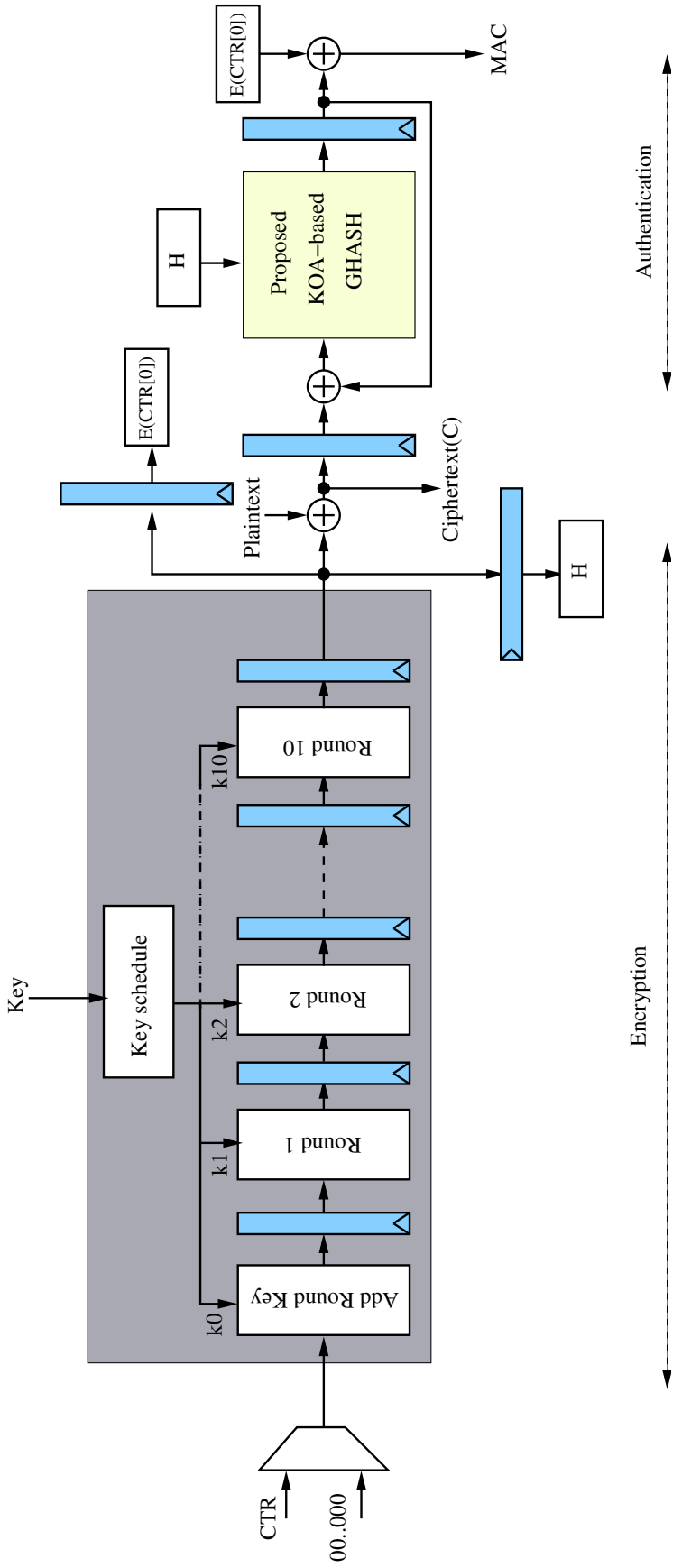

FIGURE 4.2: Proposed AES-GCM architecture

4. The AES engine changes its mode to be in CTR mode for performing encryption and delivering C_i to the proposed GHASH.

5. The last value resulting from the proposed GHASH is XORed with E(CTR[0]) in order to achieve the final MAC.

The presented architecture of AES-GCM perfectly suits the needs of GCM mode which performs the encryption and the authentication of the input message. As we described before, the encryption and the authentication in GCM are performed using the pipelined AES in CTR mode and the proposed GHASH, respectively. Therefore, the proposed architecture could also be tuned to handle the decryption and the authentication. Indeed, C_i is XORed with the output of the pipelined AES for performing the decryption and also is passed to the proposed GHASH for MAC generation.

4.4 Efficient hardware implementation for AEGIS-128

The goal of this section is to present our efficient and high speed architecture of AEGIS-128. As shown in Equations 4.8, 4.9, and 4.10, encryption, authentication and decryption tasks are performed by the five values of S (from $S_{i,0}$ to $S_{i,4}$). In case of using one AES-round, the values from $S_{i,0}$ to $S_{i,4}$ are updated every five clock cycles.

$$C_i = P_i \oplus S_{i,1} \oplus S_{i,4} \oplus (S_{i,2} \& S_{i,3}) \tag{4.8}$$

$$MAC = \oplus_{i=0}^{4}(S_{(\frac{msglen}{128}+7),i}) \tag{4.9}$$

$$P_i = C_i \oplus S_{i,1} \oplus S_{i,4} \oplus (S_{i,2} \& S_{i,3}) \tag{4.10}$$

In our proposed architecture shown in Fig. 4.3, five AES-rounds are used in order to generate the values from $S_{i+1,0}$ to $S_{i+1,4}$ in one clock cycle. In terms

of the hardware implementation, 80 s-boxes and 20 Mixcolumns are used. SubBytes stage is implemented by using LUT and composite field methods because using BRAMs adds a register (see Fig. 3.2) which is not be suitable with the iterative nature of AEGIS. Multiplexers from M1 to M5 are used to switch between the initialization mode to the encryption and the finalization modes. M6 is to support m_i to the state update function where:

$$m_i = \begin{cases} K_{128} & Initialization(i = even) \\ K_{128} \oplus IV_{128} & Initialization(i = odd) \\ P & Encryption \\ S_{\frac{msglen}{128},3} \oplus tmp & Finalization. \end{cases}$$

1. The initialization of AEGIS-128:
   As we described before, the initialization of AEGIS-128 consists of loading the key and IV into the state in 10 steps (i=-10 to -1). The initialization stage is performed as follows:

   - The used multiplexers (M1 to M5) pass the values $S_{init,0}$, $S_{init,1}$, $S_{init,2}$, $S_{init,3}$, and $S_{init,4}$ to the five AES-rounds during the first step.
   - During the next 9 steps, the values $S_{i+1,0}, S_{i+1,1}, S_{i+1,2}, S_{i+1,3}, S_{i+1,4}$ are fed to the multiplexers (M1 to M5). Also, M6 passes the values $m_{2i} = K_{128}$ or $m_{2i+1} = K_{128} \oplus IV_{128}$ according to the step number (even or odd). This takes 10 clock cycles as an initialization stage.

2. The encryption of AEGIS-128:
   After completing the initialization stage, the encryption is performed as shown in Equation 4.8 when M6 passes the plaintext (P) to be mixed with $S_{i,0}$. Encrypting 128-bit input takes only one clock cycle because of using five AES-rounds.

3. The finalization of AEGIS-128:
   After processing all the input message encryption, the finalization starts

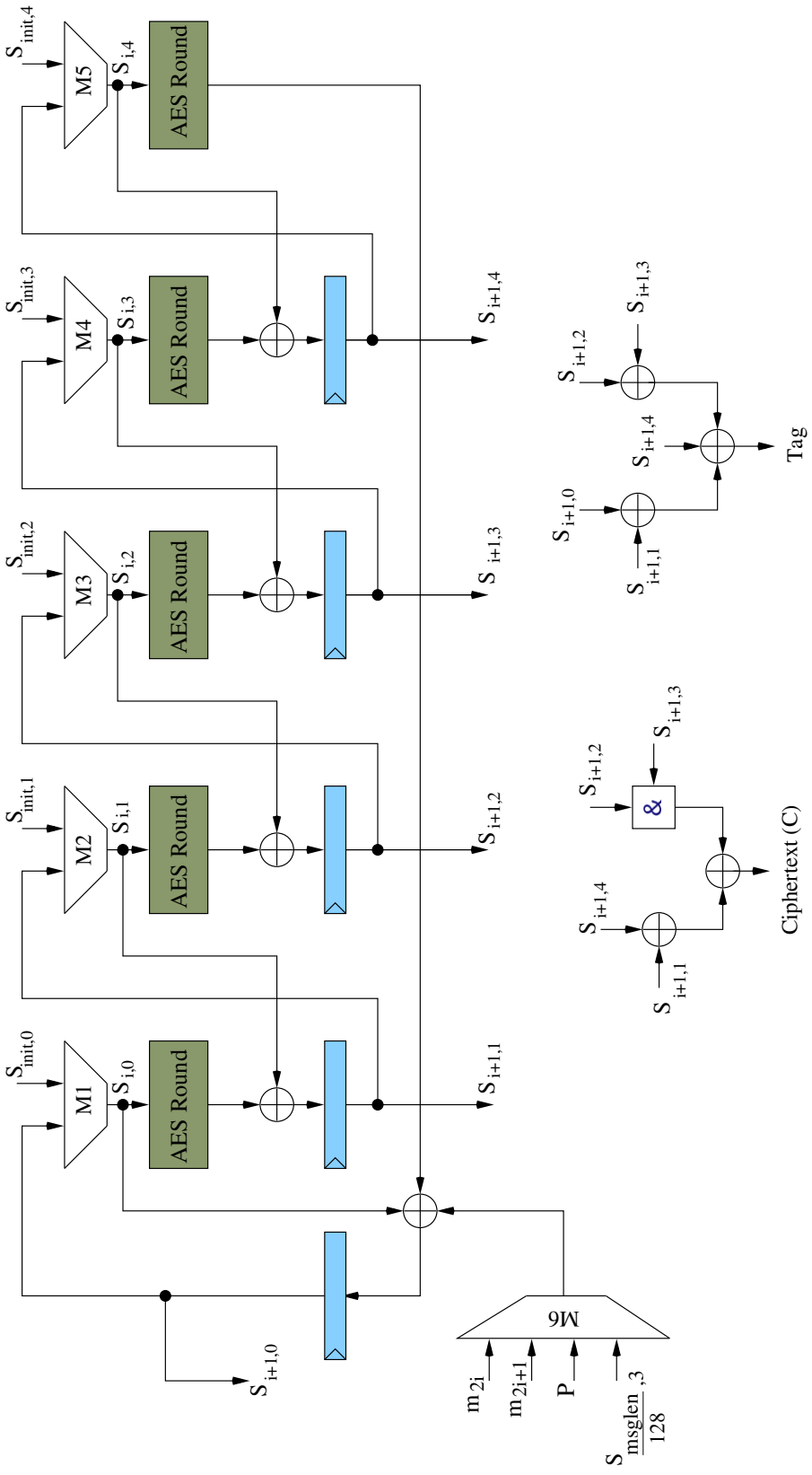

FIGURE 4.3: Proposed high speed AEGIS-128 architecture

in order to achieve the authentication task. M6 passes the value $S_{\frac{msglen}{128},3} \oplus tmp$. This starts from i= $(\frac{msglen}{128})$ to $(\frac{msglen}{128} + 6)$. As a result, the finalization stage takes 6 clock cycles.

It is clear that any process of AEGIS-128 (encryption, decryption, or authentication) depends on the generation of the values from $S_{i+1,0}$ to $S_{i+1,4}$ which takes only one clock cycle. Hence, the overall throughput is calculated as follows:

$$Throughput(Mbps) = F_{max(MHz)} \times 128 \tag{4.11}$$

The proposed architecture of AEGIS-128 could also be tuned to handle the decryption and the authentication by processing C_i into Equation 4.10 for performing the decryption and there is no change in terms of the authentication stage.

4.5 Hardware comparison

We coded our proposed schemes in VHDL and targeted to Virtex-5 (XC5VLX220). ModelSim 6.5c was used for simulation. Xilinx Synthesize Technology (XST) is used to perform the synthesize and ISE9.2 was adopted to run the Place And Route (PAR).

Table 4.1 shows the hardware comparison between our results and previous work. Note the filled dots in the "Key" column. The key is synthesized into the architecture when denoted by ∘ which requires a new reconfiguration in case of changing the key. Otherwise, the key schedule is implemented when denoted by • and no new reconfiguration is needed in case of changing the key.

On Virtex-5 platform, our proposed AES-GCM core reaches the throughput of 32.46 Gbps with the area consumption of 3836 slices and 50 BRAMs. In case of using composite field SubBytes, it consumes 7475 slices, however no BRAMs are required. In terms of using LUT SubBytes, the proposed architecture

TABLE 4.1: Hardware comparison

	FPGA	Design	key	SubBytes	Slices	BRAMs	Max-Freq MHz	Thr. Gbit/s	Thr./Slice Mbps/Slice
This work	Virtex5	AES-GCM	●	BRAM	3836	50	273.4	32.46	**8.46**
This work	Virtex5	AES-GCM	●	Comp.	7475	0	264.2	31.36	**4.19**
This work	Virtex5	AES-GCM	●	LUT	4770	0	311	36.92	**7.74**
This work	Virtex5	AEGIS-128	●	Comp.	2729	0	84.9	10.87	3.98
This work	Virtex5	AEGIS-128	●	LUT	1391	0	156.5	20.03	14.39
[3]	Virtex5	AES-GCM	●	BRAM	3533	41	314	16.9	4.78
[3]	Virtex5	AES-GCM	●	Comp	6492	0	314	16.9	2.60
[3]	Virtex5	AES-GCM	●	LUT	4628	0	324	17.5	3.77
[31]	Virtex5	AES-GCM	●	BRAM	9561	450	233	48.8	5.1
[31]	Virtex5	AES-GCM	●	Comp	18505	0	233	48.8	2.64
[31]	Virtex5	AES-GCM	●	LUT	14799	0	233	48.8	3.29
This work (Chapter 3)	Virtex5	AES-GCM	○	BRAM	2478	40	242	30.9	12.5
This work (Chapter 3)	Virtex5	AES-GCM	○	Comp.	5512	0	232	29.7	5.38
This work (Chapter 3)	Virtex5	AES-GCM	○	LUT	3211	0	216.3	27.7	8.62

occupies 4770 and reaches the throughput of 36.92 Gbps. Also, by using using LUT, our presented AEGIS-128 architecture consumes 1391 and reaches the throughput of 20.03 Gbps. Regarding composite field, it consumes 2729 with 10.87 Gbps as a throughput.

By comparing our results of AES-GCM to [3], the comparison shows that our performance (Thr./Slice) is better. This improvement results from reducing the reduction factor in the equation of throughput as shown in Equation 4.2 and Equation 4.7. Also, our presented AEGIS-128 architectures performance is better than [3].

In terms of the 4-parallel AES-GCM by [31], our area consumption of AES-GCM and AEGIS-128 is smaller compared to them. Also, our performance is better because the throughput presented by [31] is calculated as shown in Equation 4.3.

Our previous chapter presented architectures for slow changing key applications like VPNs and the FPGA needs a new reconfiguration when the key changes. Therefore, the proposed architectures in this chapter present better performance compared to our previous AES-GCM (Chapter 3) because the FPGA does not need a new reconfiguration when the key changes but it needs 252 clock cycles for the memory initialization or 10 clock cycles in case of AES-GCM and AEGIS-128, respectively.

We believe that the design presented by [3] is the closest one to our specifications, performance goals, technology and methodology. It is not a parallel design like [31]. Also, it supports the key variation without any new reconfiguration unlike our previous architectures in Chapter 3. Therefore, Fig 4.4 shows the overall comparison between our architectures and [3]. In terms of the consumed area, our designs are slightly more than [3]. However, It clear that the presented throughput is better than [3]. In general, the overall performance of our architectures is always better than [3].

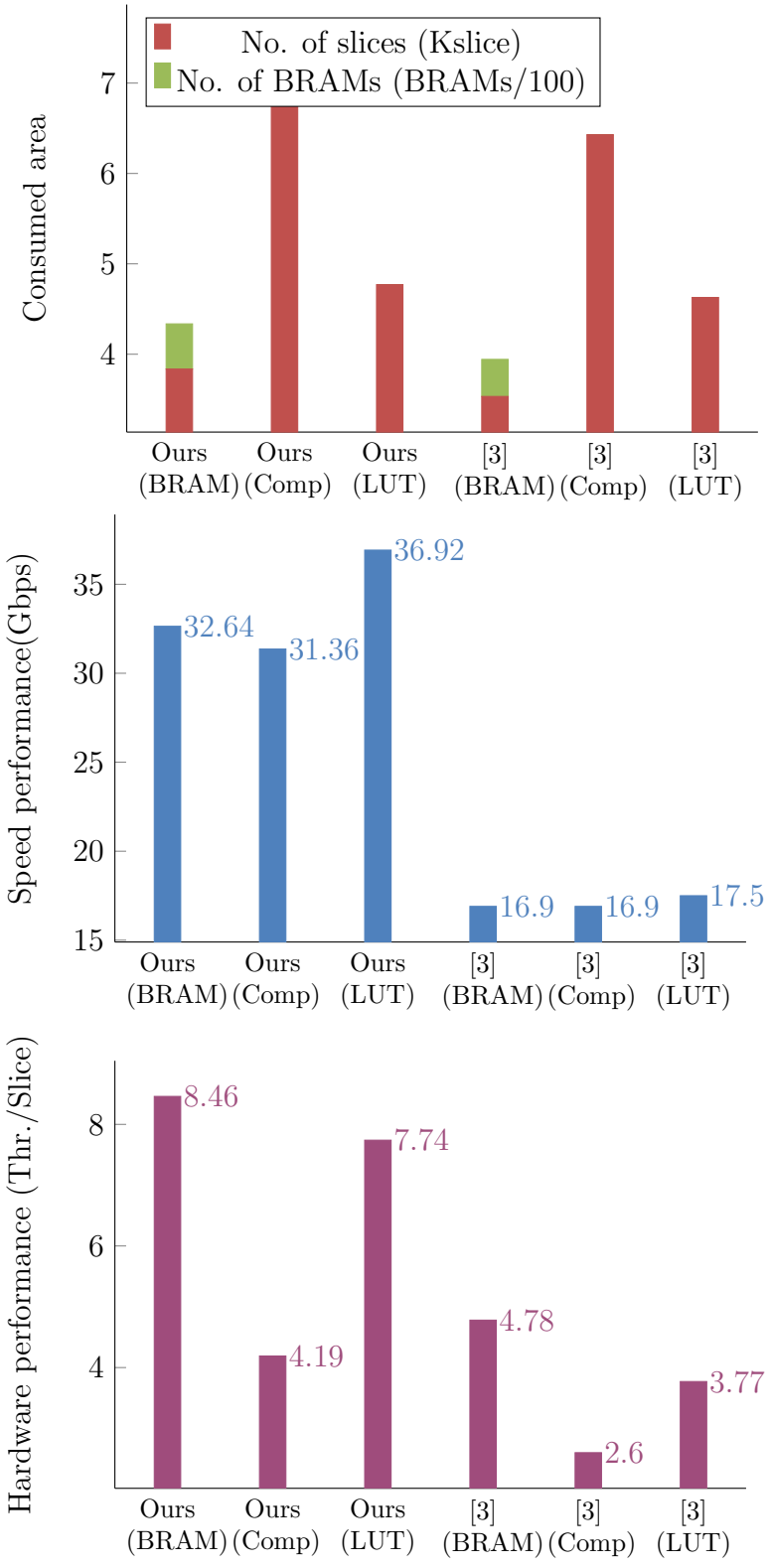

FIGURE 4.4: Hardware comparison on Virtex-5

4.6 Conclusion

In this chapter, we presented the performance improvement of AES-GCM (Thr./Slice). This was achieved by proposing an efficient pipelined KOA-based GHASH. With our proposed GHASH, the throughput reduction factor is decreased because of the feedback-free design. The presented multiplier was evaluated with three different AES implementations (BRAMs-based SubBytes, composite field-based SubBytes, and LUT-based SubBytes) in order to increase the flexibility of the presented AES-GCM. Furthermore, we designed an efficient architecture for AEGIS-128 which is considered one of the candidates of CAESAR by only five AES rounds with better performance compared to the previous work. The throughput of the presented cores ranges from 10.87 to 36.92 using Xilinx Virtex-5 FPGAs. It is shown that the performance of the presented architectures outperforms the previously reported ones.

Part II

Low Cost Solutions for Secure FPGA Reconfiguration

Chapter 5

Low Cost Solutions for Secure FPGA Reconfiguration

5.1 Introduction

When FPGAs were first introduced in the 1980s, this was a revolutionary step from static ASIC and VLSI solutions to flexible and maintainable hardware applications. With FPAGs, it has become possible to avoid the static designs of standard VLSI technology, and instead to compile electrical circuits for arbitrary hardware functions into configuration bitstreams which are used to program a fabric of reconfigurable logic.

Designing using FPGAs allows faster design cycles than ASICs because they enabled early functionality testing. Therefore, the use of FPGAs has been expanding from its traditional role in prototyping to mainstream production. The reconfiguration of FPGAs includes downloading the bitstream file which contains the new design on the FPGA. Reconfiguration of FPGAs is becoming increasingly popular particularly in networking applications. The option to reconfigure the FPGAs dynamically opens up the threat of stealing the Intellectual Property (IP) of the design. Since the configuration is usually stored in external memory, this can be easily tapped and read out by an eavesdropper.

The main goals and achievements of this chapter are as follows:

1. Giving an overview of security issues used in the reconfiguration of FPGAs.
2. Analyzing how well encryption and authentication are very important for trusted designs on FPGAs.
3. Proposing low cost hardware implementations to support and authenticate the encrypted bitstreams.

5.2 FPGA reconfiguration

Programmability manages the user to modify the functionality of a device outside of the founder. By applying this concept on gate arrays gives us Field programmable Gate Arrays (FPGAs). As we described before, the functionality of the FPGA is modified when the new bitstream is uploaded into the FPGA. The static part processes the bitstream and delivers it to the SRAM cells which reload the reconfigurable part with the new design.

Networked FPGA-based systems gain particular flexibility if remote reconfiguration updates are possible. It is attractive as it is used in such systems to offer new multimedia features or to repair eventual security issues remotely. It requires transmitting the bitstream file which contains the hardware IP from the development location over public (insecure) channels and thus introduces new security issues (see Fig. 5.1) .

The developer is faced with several problems resulting from sending the bitstream file through insecure network. An adversary attacker can detect the hardware IP to sell illegal copies or leak it to the public domain. There are several types of attacks could be occurred to the bitstream file:

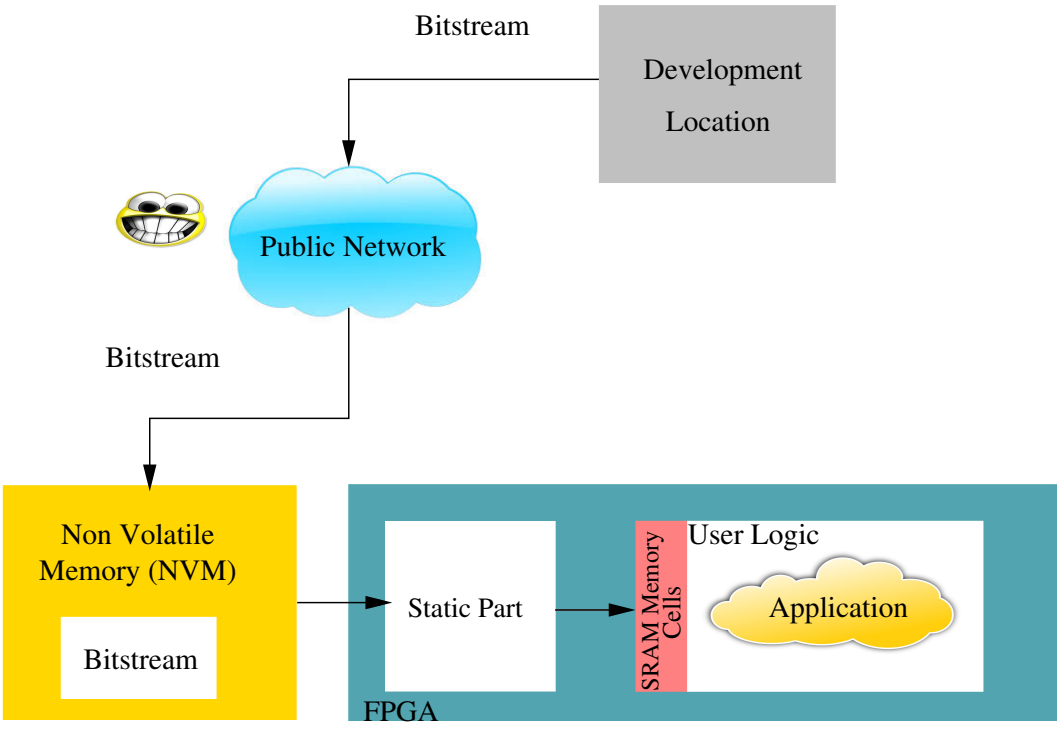

FIGURE 5.1: Remote reconfiguration

1. **Cloning attack:**

   The bitstream generated for one FPGA can be implemented on other of the same family and size. Therefore, attackers can clone the bitstream by copying it during sending it to the FPGA in order to sell the IP illegally (see Fig. 5.2).

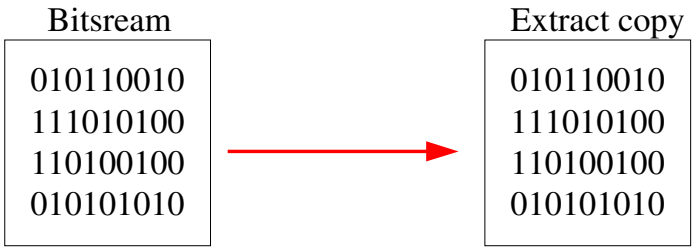

FIGURE 5.2: Cloning attack

2. **Reverse engineering attack:**

   It is the transformation of an encoded bistream into a logical functional description which is equivalent to the original design as shown in Fig. 5.3. The attacker extracts data from the bitstream such as keys and BRAMs content. An example of how the original design is recovered from the bitstream is shown in [40].

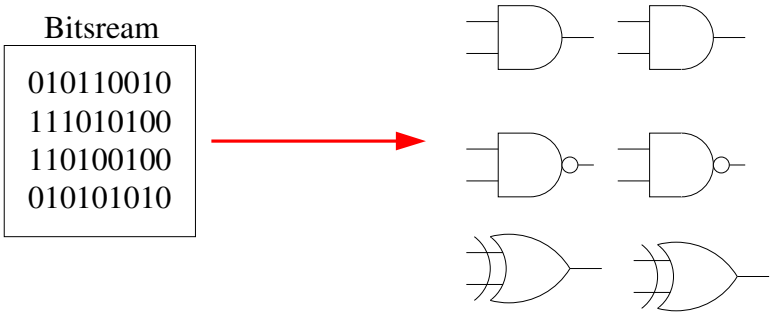

FIGURE 5.3: Reverse engineering attack

3. **Tampering attack:**

   In this type of attack, the attacker changes the design and sends it to the FPGA (see Fig. 5.4). In [41], the authors presented how an FPGA is damaged by malicious tampering of bitstreams.

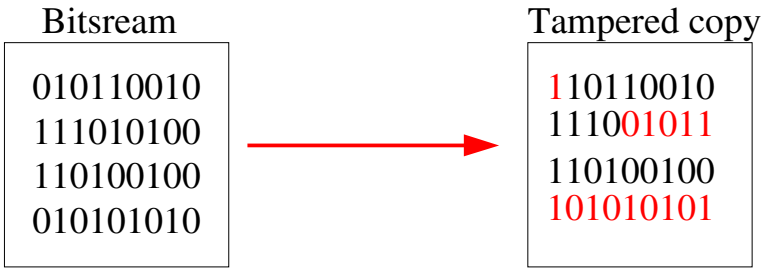

FIGURE 5.4: Tampering attack

5.3 Previous work

This section describes the previous methods used to secure the bitstream file during the reconfiguration process.

- **Bitstream confidentiality**:
  Encrypting the bistream by the developer and decrypting it within the FPGA protects against cloning and reverse engineering attacks. Encrypting the reconfiguration of the programmable devices was first introduced by [42]. Kean [2] proposed encrypting and decrypting the bitstream using embedded keys (programmed by the FPGA developer). Both encryption and decryption are performed within the FPGA itself. As shown in Fig. 5.5, the first step is performed in a trusted facility where the bitstream is encrypted by the FPGA using its embedded key K, and then stored in NVM. While the system is deployed in the field, the bitstream is decrypted with the same embedded key using a hard-core decryptor which is part of the static logic. The advantage of this method is that the key is embedded in the static part and it never leaves the FPGA. The disadvantage is that the need for a trusted environment in case of encrypting a new bitstream. Also, both encryptors and decryptors must be used because the authors used AES-128 in CBC mode, prohibitive cost. In [43], the authors proposed using the partial reconfiguration to encrypt and decrypt the bitsream. This means that the encryption and decryption circuits leave the static logic to be in the user logic. As a result, the static area is free. However, the key used for encryption and decryption is accessible to the user logic, where any one can read it out and decrypt the bitstream.

  Current FPGAs include hardwired mechanisms that ensure bitstream confidentiality [44]. Bitstream encryption, introduced by Xilinx on a production level with Virtex II FPGAs to prevent device cloning and to protect the confidentiality of the design data. Each Virtex-4, Virtex-5, and Virtex-6 device have an on-chip advanced encryption standard (AES) [45] decryption engine to support encrypted bitstreams.

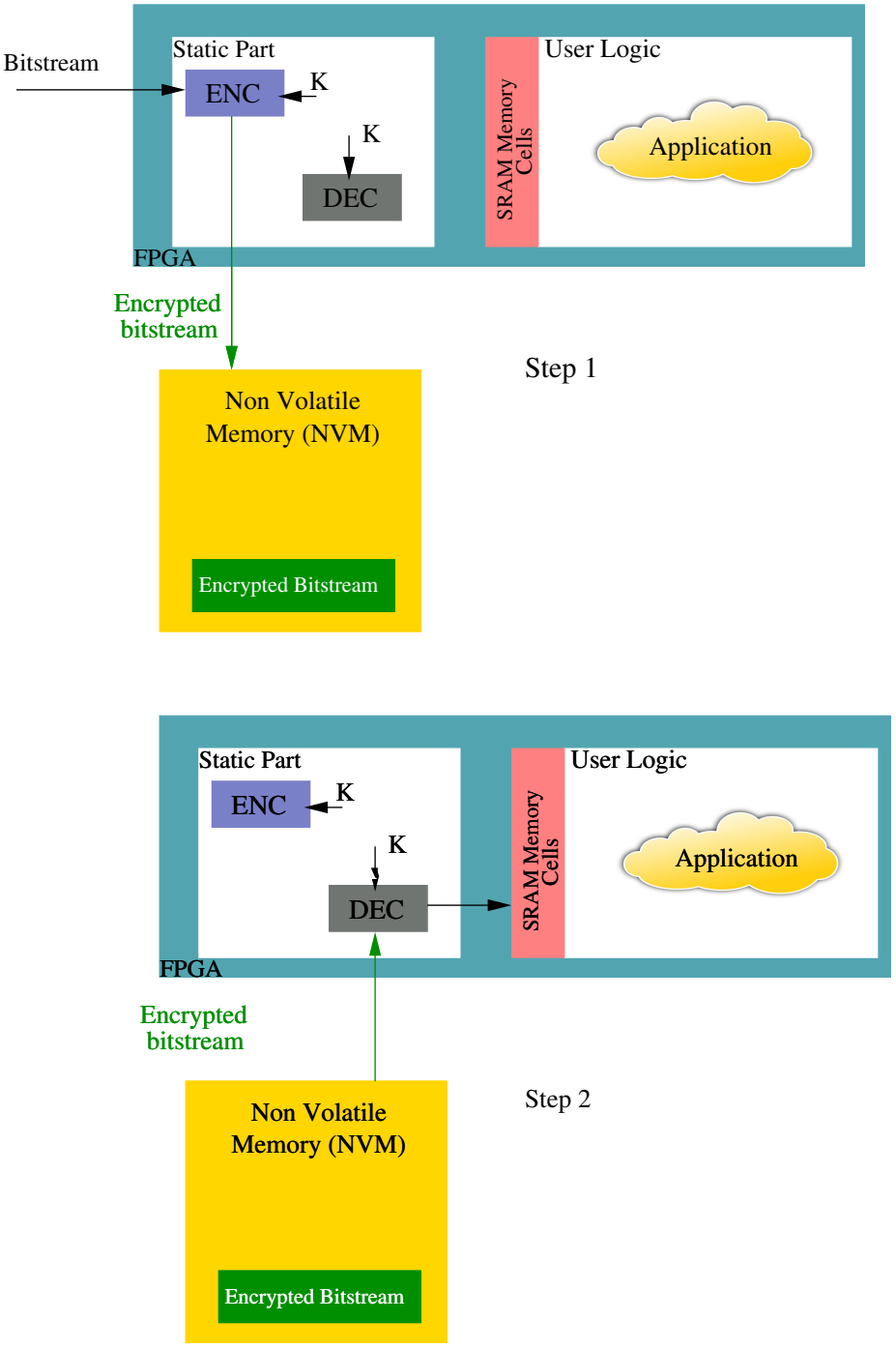

FIGURE 5.5: Bitstream encryption/decryption by [2]

The Xilinx bitstream encryption system consists of two parts:

- Software-based bitstream encryption.
- On-chip bitstream decryption.

By using the Xilinx ISE software, the user generates both the encryption key and the encrypted bitstream. The encryption key can only be programmed into the device via the JTAG port by the developer. These devices store the encryption key internally in either dedicated RAM, backed up by a small externally connected battery, or in one-time-programmable (OTP) fuses.

After programming the FPGA with the key which is used for encryption and decryption, as outlined in Fig. 5.6, the bitstream is delivered securely to the FPGA as follow:

1. The bitstream is encrypted by developer using the assigned key K.
2. The encrypted bitstream is sent to the NVM.
3. The NVM supports the FPGA with the encrypted bitstream in order to be decrypted using the static logic which contains the AES core.
4. The decrypted bitstream is passed to the SRAM cells to implement the new design.

Although this mechanism allows for protection of the system designer's IP against cloning as well as reverse engineering, it does not ensure the authenticity of the bitstream. Therefore, this solution is not enough to prevent attackers from destroying the FPGA using certain malicious bitstreams. As a result, the FPGA should accept only bitstreams from an authenticated source.

- **Bitstream integrity**:
  Tampering attack is based on the modification of the bitstream. The authors of [41] showed the effect of malicious tampering of bitstreams. They presented how an FPGA is damaged due to shorts caused by manipulating configuration bits that control pass gates. Therefore, the FPGA must be smart enough to detect the concept of **Who is the sender?**, to accept the correct bitstream sent by the trusted sender.

  Some FPGA vendors implement Cyclic Redundancy Checks (CRC) [46]. However, the purpose of CRC is to detect transmission errors, not to check the integrity of data in the cryptographic sense. This is why Xilinx

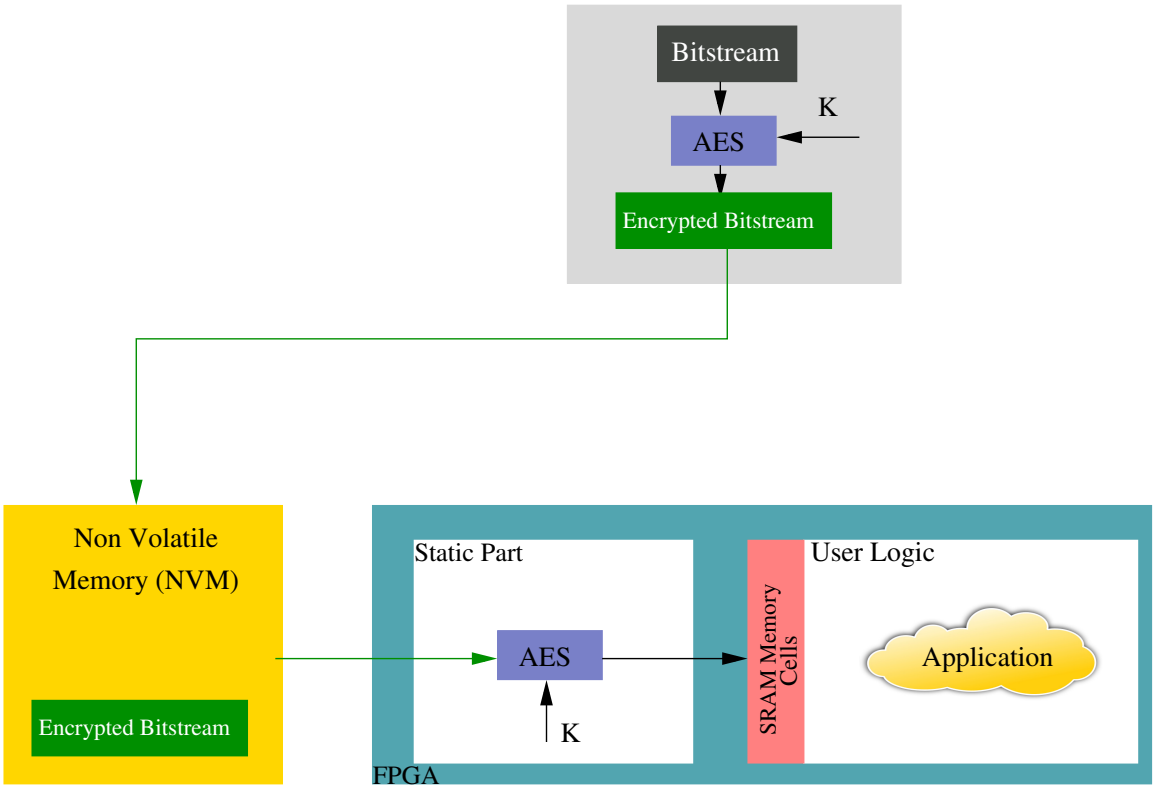

FIGURE 5.6: Bitstream encryption

[47] suggested using Message Authentication Code (MAC) function to ensure the integrity of the bitstream. Virtex-6 FPGAs are an example of the programmable devices that offer cryptographically strong bitstream authentication.

An on-chip bitstream keyed-MAC algorithm implemented in hardware provides additional security beyond that of using AES bitstream encryption alone [47]. Without knowledge of the AES and HMAC keys, the bitstream cannot be loaded, modified, or cloned.

As shown in Fig. 5.7, the developer sends the bitstream securely to the FPGA as follows:

1. The developer processes the bitstream via the AES and HMAC using k1 and k2 respectively in order to encrypt and authenticate the bitstream.
2. The encrypted bitsream and the MAC are sento to the NVM.

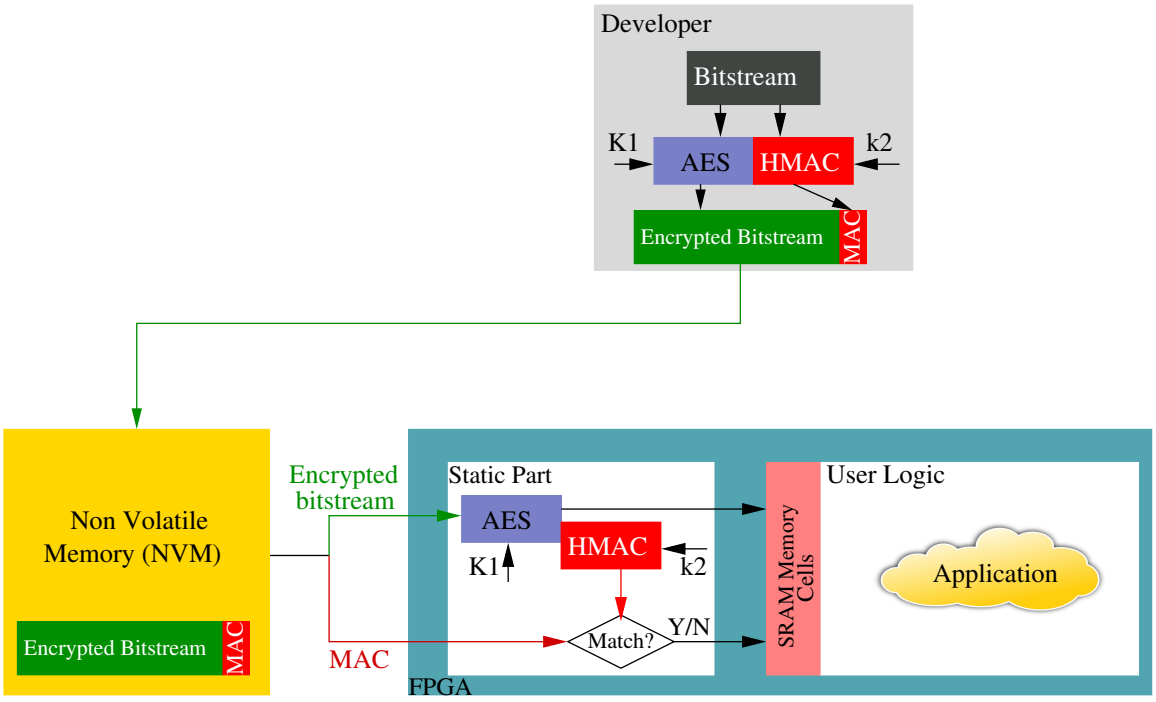

FIGURE 5.7: Bitstream encryption and authentication in Virtex6

3. The NVM is used to store the encrypted bitstream with its MAC and send them to the FPGA.
4. The FPGA static part is responsible for decrypting and authenticating the contents of the NVM.
5. The computed MAC is compared to the bitstream's MAC. If they are equal, the FPGA will continue to the start-up sequence. Otherwise, configuration will abort and the cells be cleared.

Drimer [48] proposed an FPGA bitstream encryption-authentication mechanism based on two parallel AES engines, in CTR and CBC modes, for performing both encryption and authentication. His solution used the architectures presented by Parelkar [25] who noted that generic composition of authentication and encryption (AES+HMAC) required more circuit area than AE algorithms. Therefore, Parelkar [25] presented CCM mode for achieving both authentication and encryption. Table 5.1 shows the difference between the hardware implementation of CCM mode and AES+HMAC. It is clear that CCM needs smaller area than AES combined with HMAC. In [49], the authors proposed using ALE

algorithm for performing both decryption and authentication but this algorithm was attacked by [50, 51].

TABLE 5.1: Hardware comparison

Design	architecture	Technology	Area mm^2	Frequency MHz	Throughput Mbps
Parelkar et al.[25]	AES-CCM	90 nm	0.057	148	434
Parelkar et al.[25]	AES+HMAC	90 nm	0.183	101.2	1293

As mentioned above, most of the previous work used the static part in order to perform only decryption or both decryption and authentication of the encrypted bitstream [2, 48, 52]. However, in order to leave the static part free for the developed application, Bossuet et al. [43] used the user part to decrypt the encrypted bitstream. Table 5.2 summarizes and compares between the previous work.

Our presented approach in the following section utilizes the static part in order to perform both decryption and authentication by an embedded key in the static part. As a result, the used key will never leave the static part (more secure). In order to reduce the area of the proposed solution in the static part, low cost AE algorithms will be implemented.

5.4 Low cost AE architecture for secure reconfiguration

Our goal is to design low cost solutions to be used for decryption and authentication of the encrypted bitstream. The reason of the low cost implementation is to reduce the used area of the static part which performs the security task. The used key is always embedded in the static part to avoid any user access. Efficient hardware implementations for AE are presented and compared with previous work. Presented architectures include AES-CCM, AES-GCM, and AEGIS-128 which can be used efficiently for secure reconfiguration of FPGAs.

TABLE 5.2: Previous work summery

Design	Bitstream confidentiality	Bitstream integrity	Key access storage	Core existence	Trusted environment for reconfiguration	Area requirement
[2]	Yes	No	Static part	Static part	Yes	AES-based encryptor and decryptor
[43]	Yes	No	User Part	User Part	No	AES-based encryptor and decryptor
[52]	Yes	No	Static part	Static part	No	AES-based decryptor
[47]	Yes	Yes	Static part	Static part	No	AES-based decryptor +HMAC
[48]	Yes	Yes	Static part	Static part	No	AES-CCM

The ASIC implementation is the main target, since the bitstream decryption and authentication modules are meant to be implemented as an independent IP in the static part (FPGA silicon).

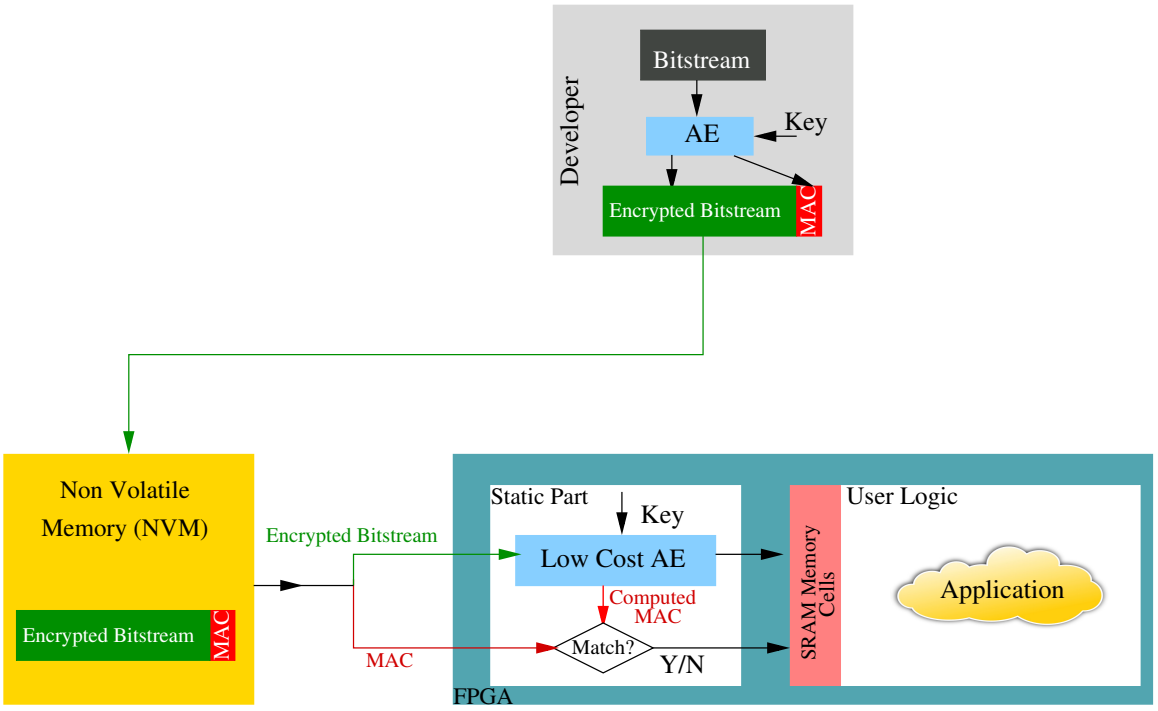

FIGURE 5.8: our Proposed approach

As shown in Fig. 5.8, the developer encrypt and authenticate the bitstream using AE (AES-CCM, AES-GCM, or AEGIS-128). After, the encrypted bitsream and the MAC are sent to the NVM. The NVM sends the encrypted bitstream with its MAC at the power-up. The low cost AE inserted in the static part decrypts and verifies the MAC for performing the secure reconfiguration.

5.4.1 Proposed AES-CCM

In the implementation of our AES-CCM core, we have focused on minimizing the area. Since AES-CCM uses two blocks of AES in CTR and CBC modes for performing both encryption and authentication, the proposed architecture uses only one AES block for both encryption/decryption and authentication in order to obtain an area-efficient architecture.

In [48], the authors motivated using the presented AES-CCM proposed by [25] in order to secure the reconfiguration process. In [25], one round-based AES (iterative design) was used to implement AES-CCM. As a result, the complexity of their design was sixteen SubBytes with four Mixcolumns.

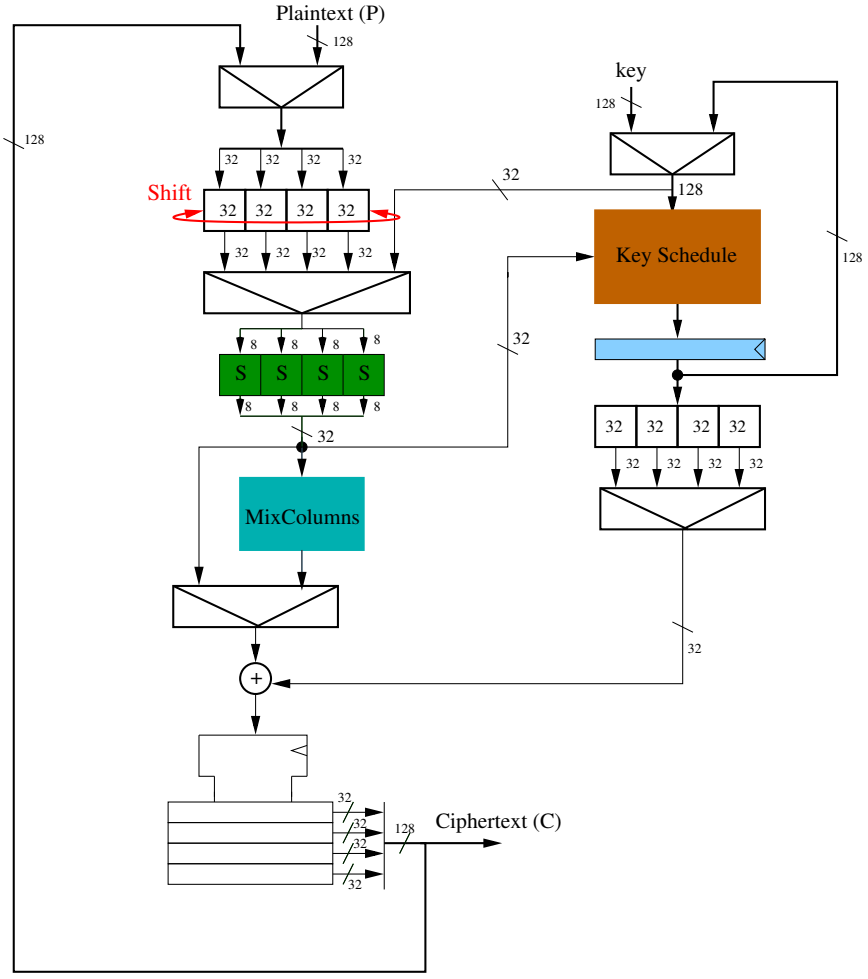

FIGURE 5.9: 1/4 round-based AES

In the proposed AES-CCM, we use 1/4 round-based AES that has an advantage of reducing the consumed area with a suitable throughput which is able to support applications lower than 1Gbps. Fig. 5.9 shows the architecture of 1/4 round-based AES. The data path is reduced from 128-bit to 32-bit. Hence, only four SubBytes (s-boxes) and one MixClolumn are used. The key schedule shares the four s-boxes stage with the main data path to avoid using another four s-boxes. Additionally, the SubBytes stage is implemented using composite field for reducing the consumed area.

Each round processes the 128-bit frame in five clock cycles (four clock cycle for the main frame + one clock cycle because the key schedule shares the s-boxes with the main data path). Because of using AES-128, 10 rounds must be computed to achieve the encryption/decryption process. In order to encrypt/decrypt a 128-bit frame, 50 clock cycles are needed (5×10). Therefore the throughput of the 1/4 round-based AES is as follows:

$$Throughput(Mbps) = \frac{F_{max(MHz)} \times 128}{50}. \tag{5.1}$$

Our AES-CCM architecture uses one 1/4 round-based AES for both encryption and authentication as shown in Fig. 5.10. Encryption and authentication are computed as follows:

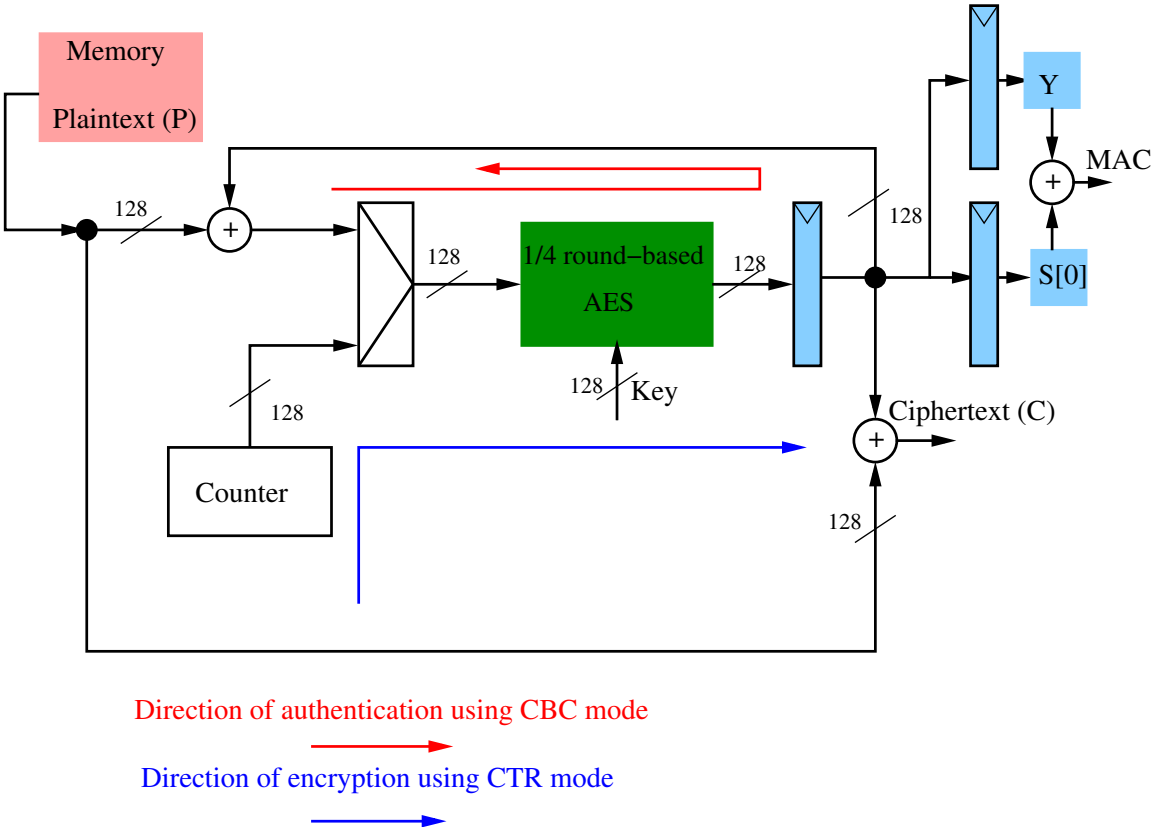

FIGURE 5.10: Proposed AES-CCM (encryption and authentication)

1. The plaintext (P), which is stored in a memory (not online), is processed by 1/4 round-based AES in CBC mode in order to generate **Y**.

2. CTR[0] is encrypted to achieve S[0] which is XORed with **Y** to generate the MAC.

3. Encryption is then performed by 1/4 round-based AES in CTR mode.

The same architecture could be tuned to handle both decryption and authentication as follows (see Fig. 5.11):

1. CTR[0] is encrypted to obtain S[0].

2. Decryption is computed by XORing the encrypted counter values with the ciphertext (C) to obtain the plaintext (P).

3. The plaintext (P) is processed by 1/4 round-based AES in CBC mode to perform the authentication by the generation of Y which is XORed with S[0] to obtain the MAC.

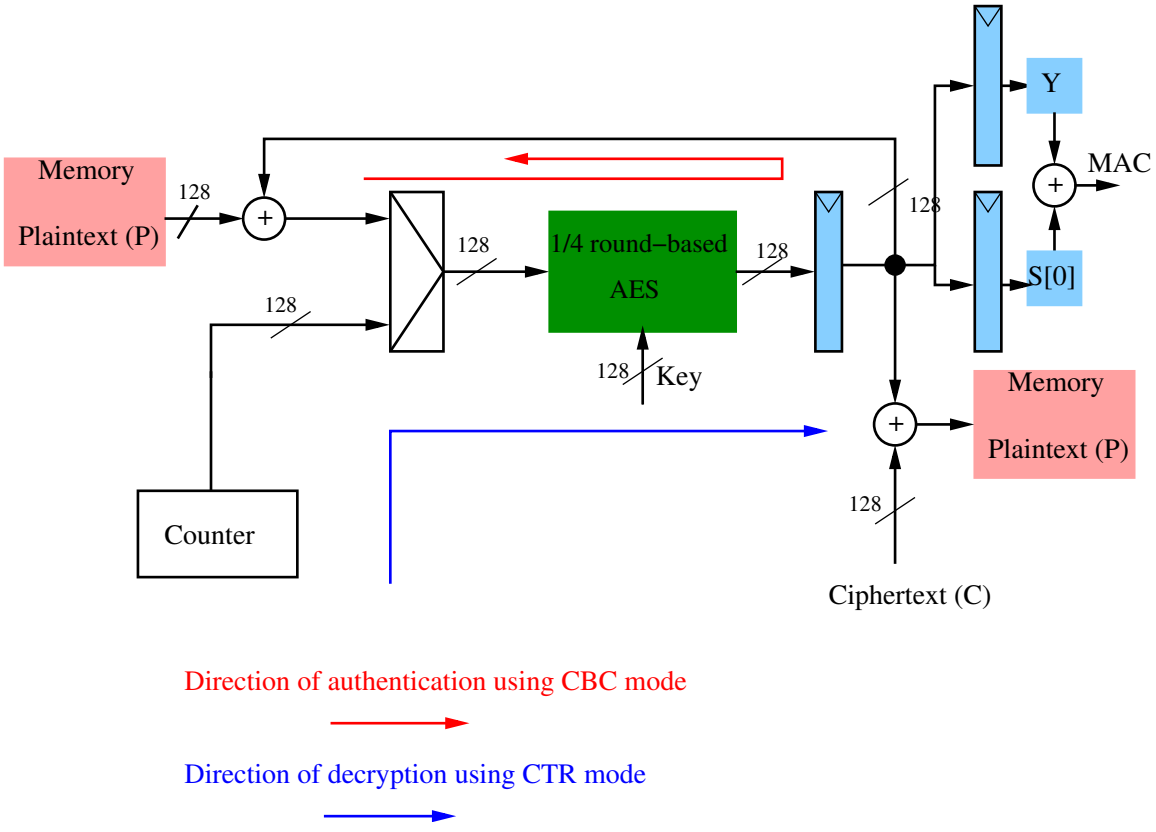

FIGURE 5.11: Proposed AES-CCM (decryption and authentication)

Compared to the solution of [48], the proposed architecture consumes only four s-boxes and one MixColumn instead of sixteen s-boxes and four MixColumns. A 128-bit frame takes 50 clock cycles to be encrypted/decrypted or added to MAC queue. Therefore, the achieved throughput of our presented AES-CCM is calculated as follows:

$$Throughput(Mbps) = \frac{128 \times F_{max(MHz)}}{50 \times 2}. \tag{5.2}$$

5.4.2 Proposed AES-GCM

AES-GCM uses two components: an AES engine and a GF(2^{128}) multiplier. The target is directed to optimize the overall architecture which includes the encryption/decryption part (AES) and the authentication part (GF(2^{128})).

Our proposed architecture uses the 1/4 round-based AES for area optimization. Previous architectures of GF(2^{128}) such as [30, 34] were used for high speed applications. In terms of the hardware implementation, they are expensive. Hence, it is important to design a multiplier which can be used efficiently with the 1/4 round-based AES.

Serial GF(2^{128}) multiplier is described in **Algorithm 1** [12], where *A,H* are inputs to the multiplier and *F(x)* is the field polynomial, $F(x) = x^{128} + x^7 + x^2 + x + 1$. The output C needs 128 clock cycles to be ready in case of using serial multiplier.

As shown in **Algorithm 1**, it is possible to design one round for achieving the multiplication in 128 clock cycles. Four rounds are used together to reduce the number of clock cycles needed to perform the multiplication from 128 to 32 (128/4) clock cycles in order to be suitable for the 1/4 round-based AES architecture as shown in Fig. 5.12.

Our proposed AES-GCM architecture shown in Fig. 5.13 uses 1/4 round-based AES with the hybrid GF(2^{128}) multiplier to accomplish the task of encryption and authentication. In case of performing both encryption and authentication, the sequence is as follows:

Algorithm 1 : GF(2^{128}) Multiplier

Input A,H, F(x) Field Polynomial

Output C

For i=0 to 127 do

IF H_i =1 Then

C ⟵ C ⊕ A

End IF

IF A_{127}= 0 Then

A = rightshift (A)

Else

A = rightshift (A) ⊕ F(x)

End If

End For

Return C

One Round

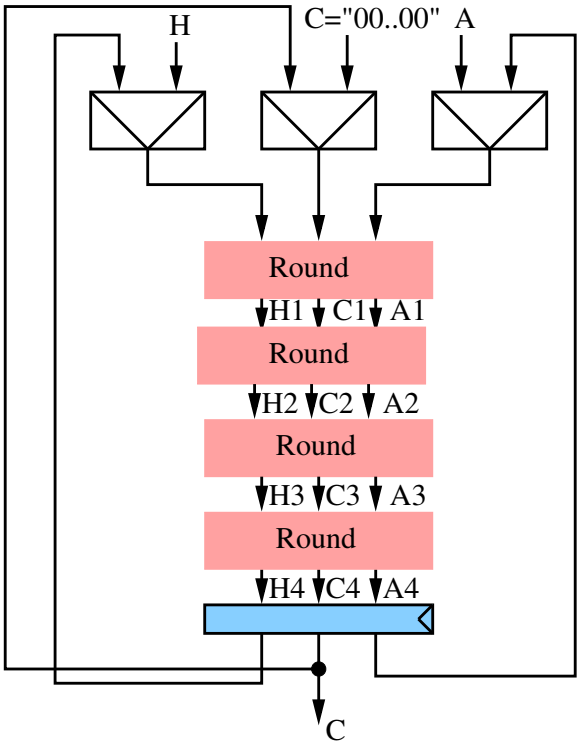

FIGURE 5.12: Proposed GF(2^{128}) multiplier

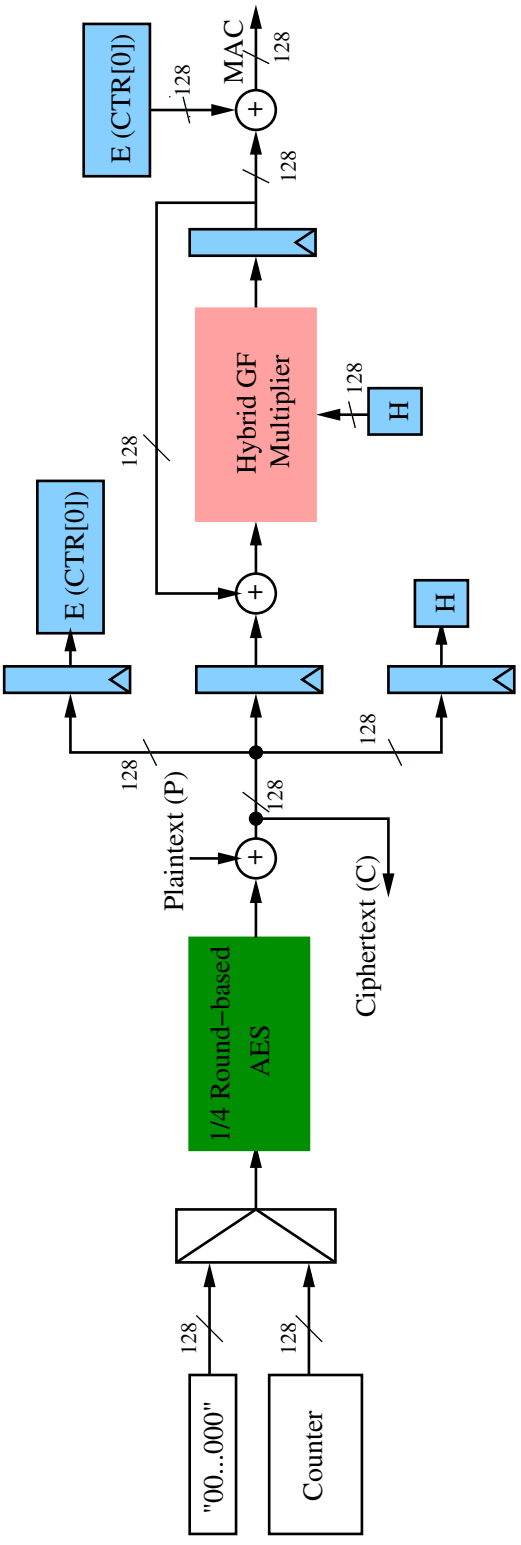

FIGURE 5.13: Proposed AES-GCM (encryption and authentication)

1. ”00..00” frame is encrypted to obtain H.
2. CTR[0] is encrypted (it is used later for authentication).
3. Encryption is performed by 1/4 round-based AES in CTR mode.
4. Encrypted frames are passed to the hybrid multiplier for performing the authentication.

The presented architecture could be also tuned for performing both decryption and authentication as follows (see Fig. 5.14):

1. ”00..00” frame is encrypted to obtain H.
2. CTR[0] is encrypted (it is used later for authentication).
3. Decryption is computed by the 1/4 round-based AES in CTR mode, where the ciphertext (C) is XORed with the encrypted counter values to obtain the plaintext (P).
4. The ciphetext (C) is passed to the hybrid multiplier for performing the authentication.

It is clear from Fig. 5.13 and Fig. 5.14 that each frame is encrypted/decrypted in 50 clock cycles and added to MAC queue in 32 clock cycles for authentication. As a result, the throughput of the proposed AES-GCM is calculated as follows:

$$Throughput(Mbps) = \frac{128 \times F_{max}}{50}. \tag{5.3}$$

The advantage of AES-GCM over AES-CCM is that AES-GCM supports online applications because of the external hashing using the GF(2^{128}). Hence, there is no need for the memory to store the plaintext (P). Also, the throughput is higher (see Equations 5.2 and 5.3).

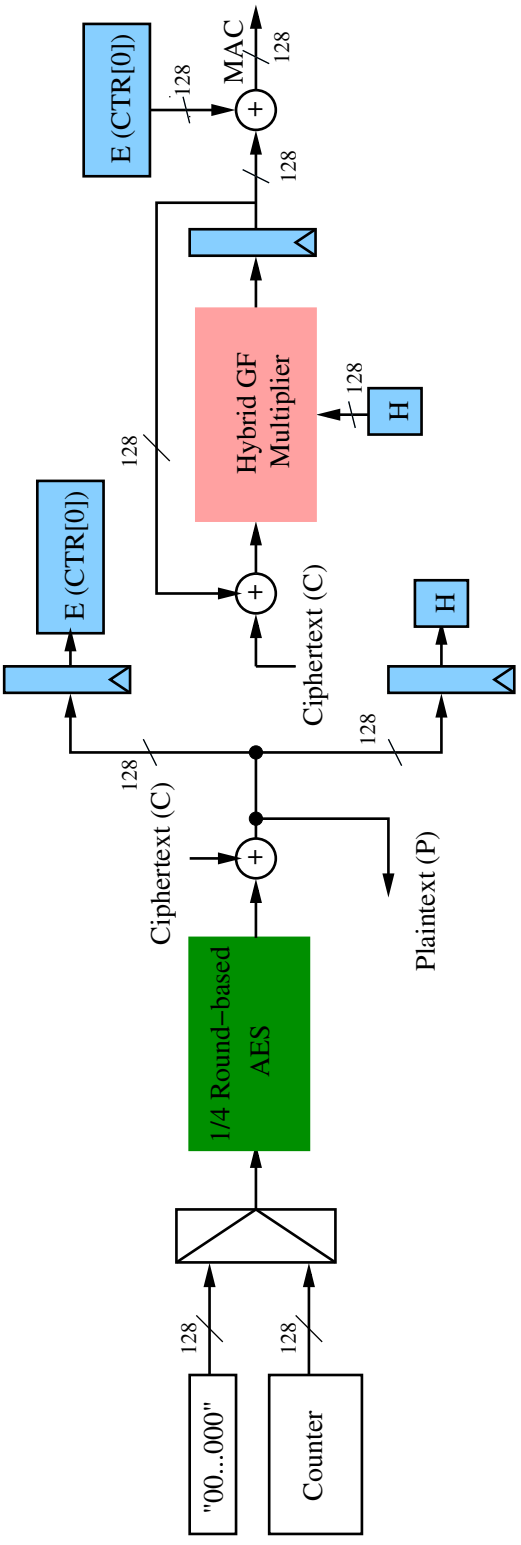

FIGURE 5.14: Proposed AES-GCM (decryption and authentication)

5.4.3 Proposed AEGIS-128

As shown in the following Equations (5.4, 5.5, and 5.6), encryption, authentication, and decryption tasks are performed by $S_{i,0}$, $S_{i,1}$, $S_{i,2}$, $S_{i,3}$, and $S_{i,4}$. The values from $S_{i,0}$ to $S_{i,4}$ are generated by five AES-rounds. This means that using five AES-rounds generates the values from $S_{i,0}$ to $S_{i,4}$ in only one clock cycle. In case of using one AES-round, the values from $S_{i,0}$ to $S_{i,4}$ are updated every five clock cycles.

$$C_i = P_i \oplus S_{i,1} \oplus S_{i,4} \oplus (S_{i,2} \& S_{i,3}) \tag{5.4}$$

$$MAC = \oplus_{i=0}^{4}(S_{(\frac{msglen}{128}+7),i}) \tag{5.5}$$

$$P_i = C_i \oplus S_{i,1} \oplus S_{i,4} \oplus (S_{i,2} \& S_{i,3}) \tag{5.6}$$

One AES-round (Fig. 5.15 (a)) contains ShiftRows stage, SubBytes (16 s-boxes) stage, and MixColumns (4 Mixcolumns). In terms of the hardware implementation, the most consuming parts are SubBytes and MixColumns. Therefore, we propose using only four s-boxes followed by only one MixColumn stage in order to reduce the consumed area. As a result, the round complexity is reduced to be 1/4 round (Fig. 5.15 (b)) by using a full ShiftRows module, 4-input multiplexer, four s-boxes, one MixColumn and First-In-First-Out (FIFO) register. The data path becomes 32-bit instead of 128-bit because of using four s-boxes and one MixColumn. Four clock cycles are needed to update each state S_i. Therefore, updating values from $S_{i,0}$ to $S_{i,4}$ takes 20 clock cycles (4×5).

The overall architecture of the proposed AEGIS-128 is shown in Fig. 5.16. The 1/4 AES-round (Fig. 5.15 (b)) is used to reduce the hardware complexity of the proposed architecture. Multiplexers from M1 to M9 switch between the initialization mode to the encryption and the finalization modes. M10 is to support m_i to the state update function where:

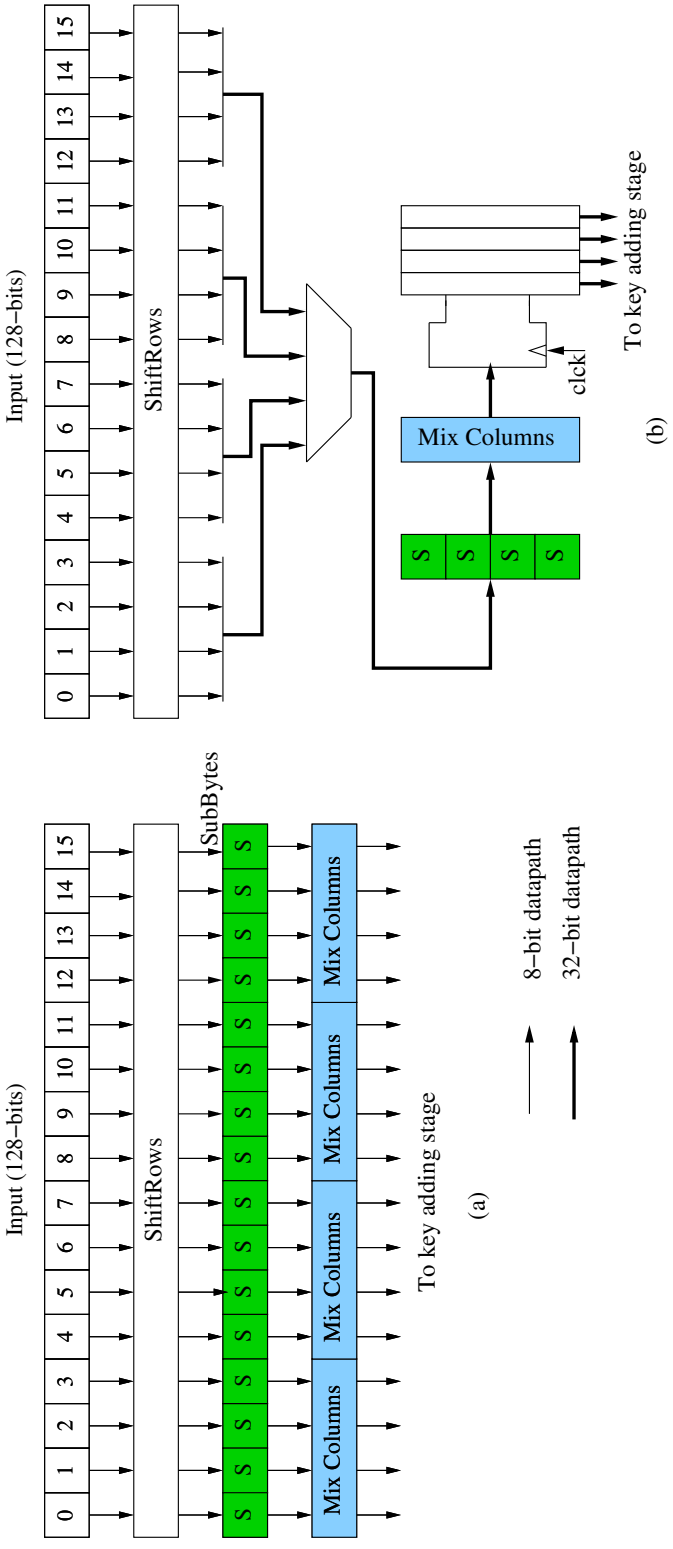

FIGURE 5.15: (a) Full AES-round. (b) (1/4) AES-round

$$m_i = \begin{cases} K_{128} & Initialization(i = even) \\ K_{128} \oplus IV_{128} & Initialization(i = odd) \\ P & Encryption \\ S_{\frac{msglen}{128},3} \oplus tmp & Finalization \end{cases}$$

Both encryption and authentication are performed as follows:

1. The initialization of AEGIS-128:
   As we described before, the initialization of AEGIS-128 consists of loading the key and IV into the state in 10 steps (i=-10 to -1). Therefore, the used multiplexers (M1 to M9) pass the values $S_{init,0}$, $S_{init,1}$, $S_{init,2}$, $S_{init,3}$, and $S_{init,4}$ to the 1/4 AES-round sequentially during the first step. During the next 9 steps, the values $S_{i+1,0}$, $S_{i+1,1}$, $S_{i+1,2}$, $S_{i+1,3}$, and $S_{i+1,4}$ are fed to the multiplexers (M1 to M9). Also, M10 passes the values $m_{2i} = K_{128}$ or $m_{2i+1} = K_{128} \oplus IV_{128}$ according to the step number (even or odd).

2. The encryption of AEGIS-128:
   After completing the initialization stage, the encryption is performed as shown in Equation 5.4 when M10 passes the plaintext (P) to be mixed with $S_{i,0}$.

3. The finalization of AEGIS-128:
   After processing all the input message encryption, the finalization starts in order to achieve the authentication task. M10 passes the value $S_{\frac{msglen}{128},3} \oplus tmp$. This starts from i= $(\frac{msglen}{128})$ to $(\frac{msglen}{128} + 6)$.

With the same scenario, both decryption and authentication are performed as outlined in Fig. 5.17. The plaintext (P) is obtained according to Equation 5.6. It is passed by M10 to calculate $S_{i,0}$, which will be used later for the authentication as shown in Equation 5.5.

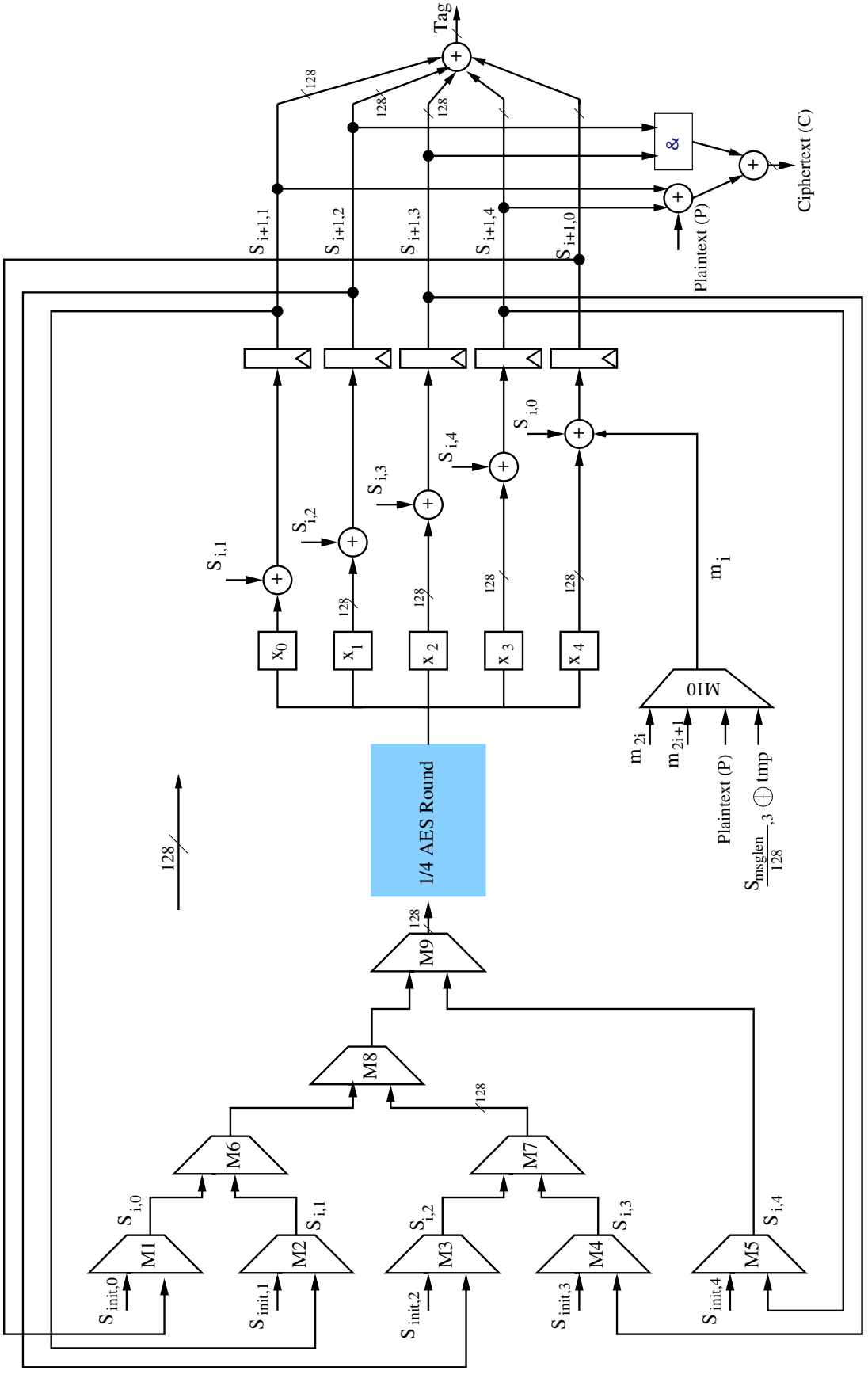

FIGURE 5.16: Proposed AEGIS-128 architecture (encryption and authentication)

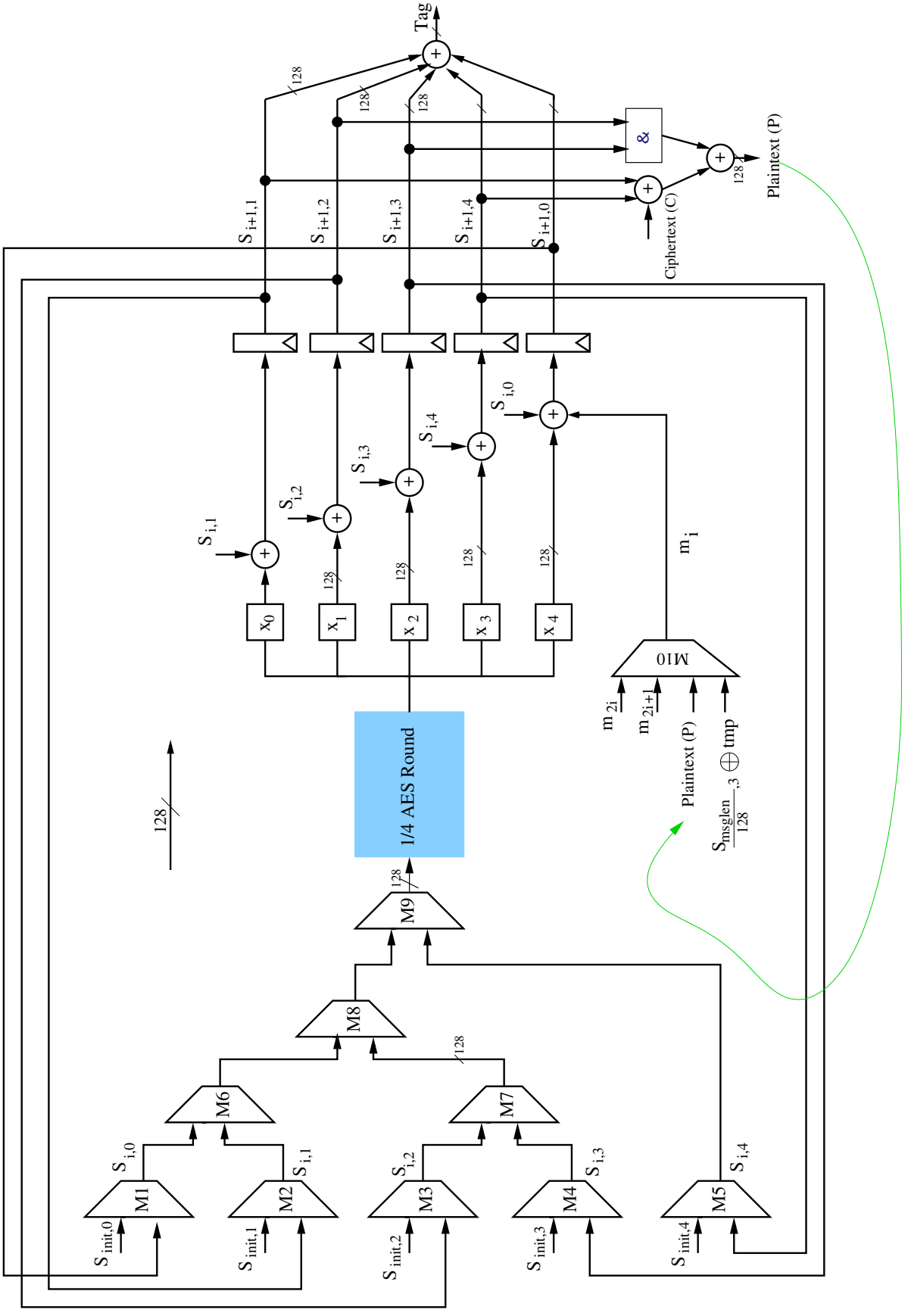

FIGURE 5.17: Proposed AEGIS-128 architecture (decryption and authentication)

Because of using (1/4) AES-round, four clock cycles are needed to process each of $S_{i+1,0}$, $S_{i+1,1}$, $S_{i+1,2}$, $S_{i+1,3}$, or $S_{i+1,4}$. Therefore, the time needed to update the overall state (achieving all S_{i+1}) is 20 clock cycles (4×5).

The initialization stage is performed from i=-10 to -1. Each stage needs the generation of five states (from $S_{i+1,0}$ to $S_{i+1,4}$). Hence, the time taken in the initialization stage is 200 clock cycles (10×20). The encryption/decryption stage requires only the generation of all S_{i+1} values (five values). As a result, 20 clock cycles are needed for encrypting 128-bit input. Authentication process requires 120 clock cycles (6×20). It is clear that any process (encryption, decryption, or authentication) depends on the generation of the values from $S_{i+1,0}$ to $S_{i+1,4}$ which takes 20 clock cycles. Hence, the overall throughput is calculated as follows:

$$Throughput(Mbps) = \frac{F_{max(MHz)} \times 128}{20}. \tag{5.7}$$

Because of supporting online applications, no memory is needed compared to AES-CCM. In comparison to AES-GCM and AES-CCM, the throughput is higher according to Equations 5.2, 5.3, and 5.7.

5.5 Hardware comparison

This section compares our presented architectures with the previous work. The ASIC implementation is the target, since the proposed architectures are implemented as an independent IP on the FPGA silicon. Presented architectures (decryption and authentication) have been implemented using 90 and 65 nm CMOS standard cell library and its performances are compared with the prior art in Table 5.3. The reason of using two different libraries (90 and 65 nm) in the evaluation of our proposed architectures is to present different estimations for the consumed area using different technologies.

In case of using 90nm technology, the proposed AES-CCM occupies 0.045 mm^2. GCM needs 0.063 mm^2 and AEGIS-128 takes 0.062 mm^2. Regarding 65 nm technology, AES-CCM occupies 0.023 mm^2, GCM needs 0.034 mm^2,

TABLE 5.3: Hardware comparison

Design	Architecture	Technology	Area	Memory	Frequency	Throughput	Function
			mm^2		MHz	Mbps	
This work	AES-CCM	90 nm	0.045	Yes	150	192	Decryption and authentication
This work	AES-GCM	90 nm	0.066	No	150	384	Decryption and authentication
This work	AEGIS-128	90 nm	0.062	No	150	960	Decryption and authentication
This work	AES-CCM	65 nm	0.023	Yes	150	192	Decryption and authentication
This work	AES-GCM	65 nm	0.034	No	150	384	Decryption and authentication
This work	AEGIS-128	65 nm	0.032	No	150	960	Decryption and authentication
[53]	AES	110 nm	0.099	No	222.2	526.7	Encryption/Decryption
[25]	AES-CCM	90 nm	0.057	Yes	148	434	Encryption and authentication
[25]	AES+HMAC	90 nm	0.183	No	101.2	1293	Encryption and authentication
[54]	Skein-1c	90 nm	0.064	No	286	1018	Authentication
[54]	Blake	90 nm	0.191	No	96	4475	Authentication

and AEGIS-128 takes 0.032 mm^2. Although AES-CCM presents the smallest architecture, it does not support online applications. Therefore, it is always combined with a memory in order to store the bitstream after the decryption process and deliver it again to the core for performing the authentication. However Parelkar et al. [25] presented one round-based AES for both encryption and authentication, the area consumption of our AES-CCM is smaller because we used 1/4 round-based AES.

In contrast to AES-CCM, AES-GCM and AEGIS-128 support online applications. Therefore, no memory is needed to support the data flow. It is clear that AES-GCM and AEGIS-128 are very close regarding the consumed area. However, the throughput of AEGIS-128 is 2.5 times faster than AES-GCM (see Equations 5.3 and 5.7). Although the AES-HMAC by [25] presented higher throughput, it consumed more area compared to ours (2.95 times of AEGIS-128 and 2.77 times of AES-GCM).

In order to enhance the comparison, we considered SHA-3 candidates like Skein and Blake. These architectures are used to support only the authentication stage. Hence, a decryption engine must be added to perform the decryption and the authentication. Our presented architectures of AES-GCM and AEGIS-128 present both decryption and authentication and their area consumption is close to Skein candidate which supports only the authentication. In case of Blake, it is clear that the consumed area of AES-GCM and AEGIS-128 is smaller.

The target of our architectures is to present a low cost solution which is added in the static part for performing the decryption and the authentication. Fig. 5.16 shows the area comparison between the proposed low cost solutions and the previous work. The hardware complexity of our presented area-efficient architectures ranges from 0.045 to 0.066 mm^2 using 90 nm technology and from 0.023 to 0.034 mm^2 using 65 nm technology.

By using the proposed architectures, the encrypted bitstream is decrypted using AE. Also, it is used to compute the MAC and compare it with the bitstream's MAC. If they are equal, the FPGA will continue to the startup sequence. Otherwise, configuration will abort and the cells be cleared. The

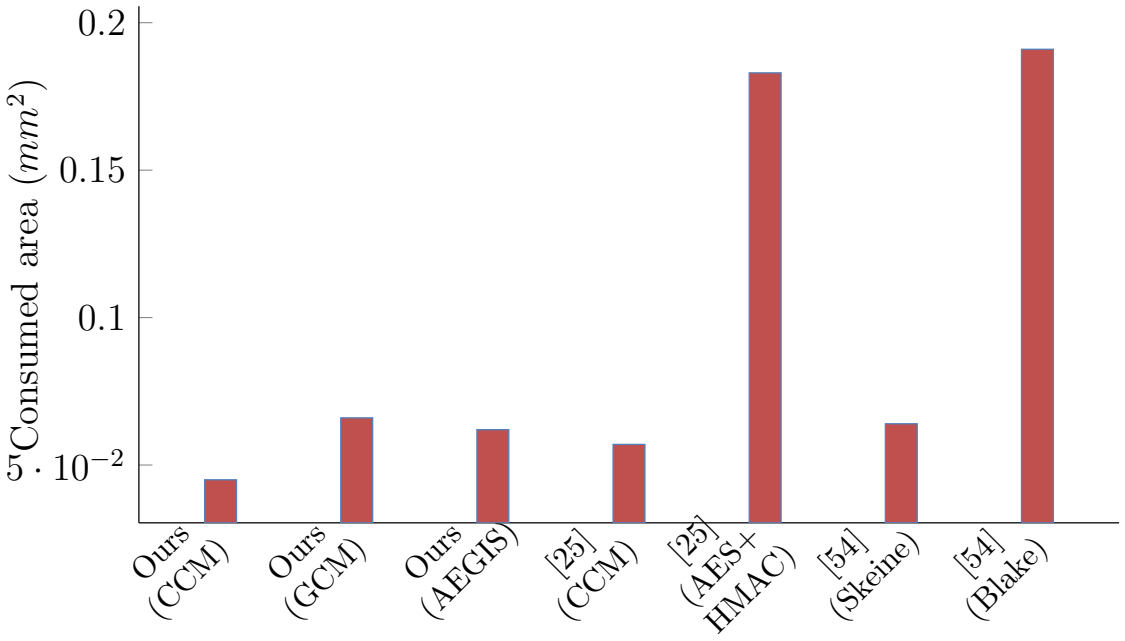

FIGURE 5.18: Area Comparison using 90 nm technology

adopted solutions meet the current configuration throughput. Table 5.4 shows the maximum throughput of the largest family members of recent FPGAs.

TABLE 5.4: Configuration throughput of some FPGA family members

FPGA	device	Technology	Throughput
Virtex-5[55]	LX330T	65-nm	800 Mbps
Stratix-III [56]	L340	65-nm	200 Mbps
Spartan-3 [57]	5000	90-nm	400 Mbps

5.6 Conclusion

This chapter proposes low cost solutions for bitstream security. This is achieved by proposing compact architectures for AE algorithms, AES-CCM, AES-GCM, and AEGIS-128. In order to reduce the hardware complexity of CCM mode, one 1/4 round-based AES is used for both decryption and authentication. Also, GCM mode uses the 1/4 round-based AES for decryption and the

hybrid GF(2^{128}) multiplier for authentication. In terms of AEGIS-128, 1/4 round is used for performing both decryption and authentication. Presented architectures were evaluated through ASIC implementation. Our comparison to the previous work reveals that the proposed designs are more resource-efficient.

Chapter 6

Summary and Future Work

6.1 Thesis Summary

The integration of security and privacy into embedded systems is an active research area that needs to keep track of technological developments.

In this thesis, we have proposed efficient hardware implementations of cryptographic algorithms for encryption and authentication. We presented FPGA and ASIC implementations that target a wide range of different applications. More precisely, we focused on the hardware design of current AE algorithms, AES-CCM, AES-GCM, and AEGIS. We investigated efficient and high speed FPGA-based architectures for these AE algorithms which were implemented in the user part (reconfigurable part) of the FPGA. In addition, we defined a framework to present different ranges of low cost ASIC architecture of AE algorithms which are inserted in the static part in order to protect the Intellectual Property (IP) of FPGA Bitstreams. The following summarizes the contributions of this work.

1. In chapter 3, we presented the performance improvement of AES-GCM by the key-synthesized method in order to support slow changing key applications like VPNs. We presented this concept using three methods of SubBytes implementation. By our proposed parallel AES-GCM, each multiplier has a fixed operand. Therefore, presented parallel AES-GCM

is suitable for key-synthesized method with higher throughput compared to the previous work. In addition, we proposed a protocol to protect the bitstream of the proposed architectures. Our presented AES-GCM architectures can be used for slow changing key applications like VPNs and they outperform the previous architectures regarding the hardware performance.

2. In chapter 4, an efficient and high speed independent-key AES-GCM was proposed. This was achieved by presenting an efficient pipelined KOA-based GHASH. The throughput reduction factor is decreased because of the feedback-free design. The presented multiplier was evaluated with three different AES implementations (BRAMs-based SubBytes, composite field-based SubBytes, and LUT-based SubBytes) in order to increase the flexibility of the presented AES-GCM. Furthermore, we designed an efficient architecture for AEGIS-128 which is considered one of the candidates of CAESAR by only five AES rounds. It is shown that the hardware performance of the presented architectures are better than the previously reported ones.

3. In chapter 5, We proposed low cost solutions for bitstream security. This is achieved by compact architectures for AE algorithms (AES-CCM, AES-GCM, and AEGIS-128). In order to minimize the hardware size of CCM mode, one 1/4 round-based AES is used for both decryption and authentication. Also, AES-GCM mode uses the 1/4 round-based AES for decryption and the hybrid GF(2^{128}) multiplier for authentication. In terms of AEGIS-128, 1/4 round is used for performing both decryption and authentication. Presented architectures were evaluated through ASIC implementation. Our comparison to the previous work reveals that the proposed designs are more resource-efficient.

6.2 Future work

As a future work for this thesis, the followings can be pursued.

1. Side-channel attacks are a class of physical attacks in which an adversary tries to exploit physical information leakages such as timing information, power consumption, or electromagnetic radiation. As a future work for this thesis, we propose studying such attacks on the proposed architectures and presenting countermeasures for these attacks.

2. Another future work for AE research area can be explored by following the final portfolio of CAESAR competition.

3. As an extension for this thesis, one can integrate a public key cryptography into the static part of the FPGA in order to change the key used for decryption and authentication of the bitstream remotely.

List of Publication

International Journals

- Karim M. Abdellatif, R. Chotin-Avot, and H. Mehrez "Ecient and High Speed AES-Based Authenticated Encryption Architectures Using FPGAs", under review in ACM Transactions on Recongurable Technology and Systems (TRETS) (Invited Paper).
- Karim M. Abdellatif, R. Chotin-Avot, and H. Mehrez "Authenticated Encryption on FPGAs from the Static Part to the Recongurable Part", Journal of Microprocessors and Microsystems: Embedded Hardware Design (MICPRO), Elsevier, 2014.
- Karim M. Abdellatif, R. Chotin-Avot, and H. Mehrez "Low cost Solutions for Secure Remote Reconguration of FPGAs", International Journal of Embedded Systems, 2013.

International Conferences

- Karim M. Abdellatif, R. Chotin-Avot, and H. Mehrez "FPGA-Based High Performance AES-GCM Using Ecient Karatsuba Ofman Algorithm", Recongurable Computing: Architectures, Tools and ApplicationProceedings of the International Symposium on Applied Recongurable Computing (ARC 2014), Springer Lecture Notes in Computer Science (LNCS), Portugal, 2014.
- Karim M. Abdellatif, R. Chotin-Avot, and H. Mehrez "Efficient Parallel AESGCM Architectures Using FPGAs", IEEE International Conference on Reconfigurable Computing and FPGAs (ReConFig), Mexico, 2013.

International Conferences

- Karim M. Abdellatif, R. Chotin-Avot, and H. Mehrez "Lightweight and Compact Solutions for Secure Reconguration of FPGAs", IEEE International Conference on Reconfigurable Computing and FPGAs (ReConFig), Mexico, 2013.
- Karim M. Abdellatif, R. Chotin-Avot, and H. Mehrez "High Speed Authenticated Encryption for Slow Changing Key Applications Using Recongurable Devices", IEEE Wireless Days, Spain, 2013.
- Karim M. Abdellatif, R. Chotin-Avot, and H. Mehrez "Protecting FPGA Bitstreams Using Authenticated Encryption", IEEE International Conference of New Circuits and Systems (NEWCAS), Paris, 2013.
- Karim M. Abdellatif, R. Chotin-Avot, and H. Mehrez "Efficient Parallel-Pipelined GHASH for Message Authentication", IEEE International Conference on Reconfigurable Computing and FPGAs (ReConFig), Mexico, 2012.

Posters

- Karim M. Abdellatif, R. Chotin-Avot, and H. Mehrez "Towards High Performance GHASH for Pipelined AES-GCM Using FPGAs", accepted in ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA), USA, 2014.
- Karim M. Abdellatif, R. Chotin-Avot, and H. Mehrez, " The Effect of S-box Design on Pipelined AES Using FPGA ", GDR SOC-SIP, Paris, 2012.

Presentations

- Karim M. Abdellatif, R. Chotin-Avot, and H. Mehrez "High Performance AES-GCM Using Feedback-Free Karatsuba Ofman Algorithm", Directions in Authenticated Ciphers (DIAC) [1], USA, 2014.

[1] The purpose of DIAC 2014 is to evaluate the state of the art in authenticated encryption and gather community input regarding desired future directions.

Bibliography

[1] J. Crenne, P. Cotret, G. Gogniat, R. Tessier, and J.P. Diguet. Efficient Key-Dependent Message Authentication in Reconfigurable Hardware. *International Conference on Field-Programmable Technology (FPT)*, pages 1–6, 2011.

[2] Tom Kean. Secure Configuration of Field Programmable Gate Arrays. In *Field-Programmable Logic and Applications*, pages 142–151. Springer, 2001.

[3] G. Zhou, H. Michalik, and L. Hinsenkamp. Improving Throughput of AES-GCM with Pipelined Karatsuba Multipliers on FPGAs. *Reconfigurable Computing: Architectures, Tools and Applications*, pages 193–203, 2009.

[4] National Institute of Standards and Technology. 197: Advanced encryption standard (AES). *Federal Information Processing Standards Publication*, 197:441–0311, 2001.

[5] Joan Daemen and Vincent Rijmen. The Pelican MAC Function. *IACR Cryptology ePrint Archive*, page 88, 2005.

[6] National Institute of Standards and Technology. The Keyed-Hash Message Authentication Code (HMAC). *FIPS PUB*.

[7] Mihir Bellare, Tadayoshi Kohno, and Chanathip Namprempre. Authenticated Encryption in SSH: Provably Fixing the SSH Binary Packet Protocol. pages 1–11, 2002.

[8] Tim Dierks. The Transport Layer Security (TLS) Protocol Version 1.2. 2008.

[9] Russell Housley. Using Advanced Encryption Standard (AES) CCM Mode with IPsec Encapsulating Security Payload (ESP). 2005.

[10] Sheila Frankel, Bernard Eydt, Les Owens, and Karen Scarfone. Establishing Wireless Robust Security Networks: A Guide to IEEE 802.11 i. *National Institute of Standards and Technology*, 2007.

[11] Morris J Dworkin. SP 800-38C. Recommendation for Block Cipher Modes of Operation: the CCM Mode for Authentication and Confidentiality. 2004.

[12] David McGrew and John Viega. The Galois/Counter Mode of Operation (GCM). *Submission to NIST*, 2004.

[13] Hongjun Wu and Bart Preneel. AEGIS: A Fast Authenticated Encryption Algorithm. 2013. http://eprint.iacr.org/.

[14] Xilinx. Virtex-5 Family Overview. 2009. URL http://www.xilinx.com/support/documentation/data_sheets/ds100.pdf.

[15] Cisco Corporation. Cisco ASA 5500 Series Adaptive Security Appliances. 2011. URL http://www.cisco.com/en/US/prod/collateral/vpndevc/ps6032/\ps6094/ps6120/prod-brochure0900aecd80285492.pdf.

[16] Stonesoft. Security Engine Firewall/VPN. 2011. URL http://www.stonesoft.com/export/download/pdf/datasheet-stonesoft-3206.pdf.

[17] Sheila Frankel, Bernard Eydt, Les Owens, and Karen Scarfone. Establishing Wireless Robust Security Networks: A Guide to IEEE 802.11 i. *National Institute of Standards and Technology*, 2007.

[18] Tim Good and Mohammed Benaissa. AES on FPGA from the Fastest to the Smallest. pages 427–440, 2005.

[19] Solmaz Ghaznavi, Catherine Gebotys, and Reouven Elbaz. Efficient Technique for the FPGA implementation of the AES MixColumns Transformation. pages 219–224, 2009.

[20] Chi-Jeng Chang, Chi-Wu Huang, Hung-Yun Tai, Mao-Yuan Lin, and Teng-Kuei Hu. 8-bit AES FPGA Implementation Using Block RAM. In *Industrial Electronics Society, 2007. IECON 2007. 33rd Annual Conference of the IEEE*, pages 2654–2659. IEEE, 2007.

[21] Xinmiao Zhang and Keshab K Parhi. High-Speed VLSI Architectures for the AES Algorithm. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 12(9):957–967, 2004.

[22] Francois-Xavier Standaert, Gael Rouvroy, Jean-Jacques Quisquater, and Jean-Didier Legat. Efficient Implementation of Rijndael Encryption in Reconfigurable Hardware: Improvements and Design Tradeoffs. In *Cryptographic Hardware and Embedded Systems-CHES 2003*, pages 334–350. Springer, 2003.

[23] Morris Dworkin. Recommendation for Block Cipher Modes of Operation. Methods and Techniques. Technical report, DTIC Document, 2001.

[24] Morris J Dworkin. SP 800-38C. Recommendation for Block Cipher Modes of Operation: the CCM Mode for Authentication and Confidentiality. 2004.

[25] Milind M Parelkar. *Authenticated Encryption in Hardware*. PhD thesis, George Mason University, 2005.

[26] Emmanuel Lopez-Trejo, F Rodrıguez Henrıquez, and Arturo Dıaz-Pérez. An Efficient FPGA Implementation of CCM mode using AES. In *International Conference on Information Security and Cryptology*, volume 3935, pages 208–215, 2005.

[27] Arshad Aziz and Nassar Ikram. An FPGA-based AES-CCM Crypto Core For IEEE 802.11 i Architecture. *IJ Network Security*, 5(2):224–232, 2007.

[28] David McGrew and John Viega. The Galois/Counter Mode of Operation (GCM). 2004.

[29] IEEE Standard for Local and Metropolitan Area Networks–Media Access Control (MAC) Security Amendment 1: Galois Counter Mode–Advanced Encryption Standard– 256 (GCM-AES-256) Cipher Suite. *IEEE*.

[30] G. Zhou, H. Michalik, and L. Hinsenkamp. Efficient and High-Throughput Implementations of AES-GCM on FPGAs. *International Conference on Field-Programmable Technology (FPT)*, pages 185–192, 2007.

[31] L. Henzen and W. Fichtner. FPGA Parallel-Pipelined AES-GCM Core for 100G Ethernet Applications. *Proceedings of the ESSCIRC*, pages 202–205, 2010.

[32] S. Lemsitzer, J. Wolkerstorfer, N. Felber, and M. Braendli. Multi-Gigabit GCM-AES Architecture Optimized for FPGAs. *Cryptographic Hardware and Embedded Systems-CHES*, pages 227–238, 2007.

[33] A. Satoh. High-Speed Hardware Architectures for Authenticated Encryption Mode GCM. *IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 4–pp, 2006.

[34] A. Satoh. High-Speed Hardware Architectures for Authenticated Encryption Mode GCM. *IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 4–pp, 2006.

[35] Anatolii Karatsuba and Yu Ofman. Multiplication of Multidigit Numbers on Automata. 7:595, 1963.

[36] Hongjun Wu and Bart Preneel. Aegis: A fast authenticated encryption algorithm. 2013. http://eprint.iacr.org/.

[37] Cisco Corporation. Cisco ASA 5500 Series Adaptive Security Appliances. 2011. URL http://www.cisco.com/en/US/prod/collateral/vpndevc/ps6032/ps6094/ps6120/prod-brochure0900aecd80285492.pdf.

[38] Stonesoft. Security Engine Firewall/VPN, 2011. URL http://www.stonesoft.com/export/download/pdf/datasheet-stonesoft-3206.pdf.

[39] Cisco Corporation. Cisco ASR 1000 Series Embedded Services Processor, 2013. URL http://www.cisco.com/c/en/us/products/collateral/routers/asr-1000-series-aggregation-services-routers/data_sheet_c78-450070.html.

[40] Jean-Baptiste Note and Éric Rannaud. From the Bitstream to the Netlist. In *FPGA*, volume 8, pages 264–264, 2008.

[41] Ilija Hadžić, Sanjay Udani, and Jonathan M Smith. Fpga viruses. In *Field Programmable Logic and Applications*, pages 291–300. Springer, 1999.

[42] Kenneth Austin. Data Security Arrangements for Semiconductor Programmable Devices, February 7 1995. US Patent 5,388,157.

[43] Lilian Bossuet, Guy Gogniat, and Wayne Burleson. Dynamically Configurable Security for SRAM FPGA Bitstreams. *International Journal of Embedded Systems*, 2(1):73–85, 2006.

[44] A. Lesea. IP security in FPGAs. *Xilinx http://direct.xilinx.com/bvdocs/whitepapers/wp261.pdf*, 2007.

[45] N.F. Pub. 197: Advanced encryption standard (AES). *Federal Information Processing Standards Publication*, 197:441–0311, 2001.

[46] C.W. Tseng. Lock your designs with the virtex-4 security solution. *XCell Journal, XILINX, Spring*, 2005.

[47] Xilinx. Virtex-6 FPGA Configuration User Guide. URL http://www.xilinx.com/support/documentation/user_guides/ug360.pdf.

[48] Saar Drimer. Authentication of FPGA Bitstreams: Why and how. In *Reconfigurable Computing: Architectures, Tools and Applications*, pages 73–84. Springer, 2007.

[49] Andrey Bogdanov, Amir Moradi, and Tolga Yalcin. Efficient and side-channel resistant authenticated encryption of fpga bitstreams. In *ReConFig*, pages 1–6, 2012.

[50] Shengbao Wu, Hongjun Wu, Tao Huang, Mingsheng Wang, and Wenling Wu. Leaked-State-Forgery Attack Against the Authenticated Encryption Algorithm ALE. pages 377–404, 2013.

[51] Dmitry Khovratovich and Christian Rechberger. The LOCAL attack: Cryptanalysis of the authenticated encryption scheme ALE. pages 174–184, 2014.

[52] Xilinx. Virtex-4 FPGA Configuration User Guide. 2009. URL http://www.xilinx.com/support/documentation/user_guides/ug071.pdf.

[53] Akashi Satoh, Sumio Morioka, Kohji Takano, and Seiji Munetoh. A Compact Rijndael Hardware Architecture with S-box Optimization. In *Advances in CryptologyASIACRYPT 2001*, pages 239–254. Springer, 2001.

[54] AH Namin and MA Hasan. Hardware Implementation of the Compression Function for Selected SHA-3 Candidates. *CACR*, 28:2009, 2009.

[55] Xilinx. Virtex-5 FPGA Data Sheet:DC and Switching Characteristics. URL http://www.xilinx.com/support/documentation/data$_$sheets/ds202.pdf.

[56] Altera. Stratix III Device Handbook. URL http://www.altera.com/literature/hb/stx3/stratix3_handbook.pdf.

[57] Xilinx1. Spartan-3 FPGA family:Complete data sheet. URL http://www.xilinx.com/support/documentation/data$_$sheets/ds099.pdf.