FPGA-Based High Performance AES-GCM Using Efficient Karatsuba Ofman Algorithm

Karim M. Abdellatif, R. Chotin-Avot, and H. Mehrez

LIP6-SoC Laboratory, University of Paris VI, France
{karim.abdellatif, roselyne.chotin-avot, habib.mehrez}@lip6.fr

Abstract. AES-GCM has been utilized in various security applications. It consists of two components: an Advanced Encryption Standard (AES) engine and a Galois Hash (GHASH) core. The performance of the system is determined by the GHASH architecture because of the inherent computation feedback. This paper introduces a modification for the pipelined Karatsuba Ofman Algorithm (KOA)-based GHASH. In particular, the computation feedback is removed by analyzing the complexity of the computation process. The proposed GHASH core is evaluated with three different implementations of AES (BRAMs-based SubBytes, composite field-based SubBytes, and LUT-based SubBytes). The presented AES-GCM architectures are implemented using Xilinx Virtex5 FPGAs. Our comparison to previous work reveals that our architectures are more performance-efficient (Thr. /Slices).

Keywords: AES-GCM, FPGAs, GHASH, Karatsuba Ofman Algorithm (KOA).

1 Introduction

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 smaller area compared to two separate algorithms.

Galois Counter Mode (GCM) [1] mode 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 [2] and NIST 800-38D have considered employing GCM to enhance their performance. The reconfigurability of FPGAs offers major advantages when using them for cryptographic applications. Hence, they are commonly used as an implementation target.

Our Contribution: In this work, we present efficient FPGA-based architectures for AES-GCM by modifying the architecture of the pipelined KOA-based GHASH. Our focus on state of the art of KOA-based GHASHs leads to solve the algorithm complexity resulting from the inherent computation feedback. In addition, three different implementations of AES are evaluated and added to the proposed GHASH in order to increase the flexibility of the presented work.

The major features and the previous work of AES-GCM are described in **Section 2**. After that, our proposed architecture of GHASH is presented (**Section 3**). The overall architecture of AES-GCM is shown in **Section 4**. Implementation details and performance comparison are discussed in **Section 5**. **Section 6** concludes this work.

2 AES-GCM

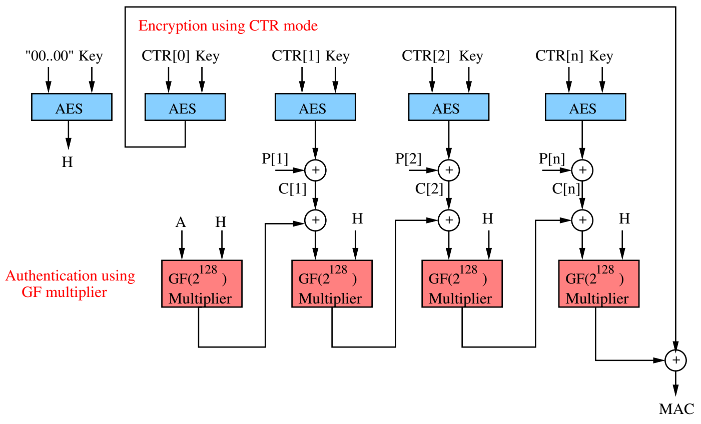

Fig. 1. AES-GCM: Encryption process is performed using counter mode and authentication is done using GF(2^{128}), A is an optional 128-bit Additional Authenticated Data which is authenticated but not encrypted

Recently, Galois Counter Mode (GCM) [1] was considered as a new mode of operation of Advanced Encryption Standard (AES). GCM simultaneously provides confidentiality, integrity and authenticity assurances on the data. It supports not only high speed authenticated encryption but also protection against bit-flipping attacks. It can be implemented in hardware to achieve high speeds with low cost and low latency. Software implementations can achieve excellent performance by using table-driven field operations. GCM was designed to meet the need for an authenticated encryption mode that can efficiently achieve speeds of 10 Gbps and higher in hardware. It contains an AES engine in counter mode and a Galois Hash (GHASH) module as presented in Fig. 1.

As shown in Fig. 1, the GHASH function (authentication part) is composed of chained GF(2^{128}) multipliers and bitwise exclusive-OR (XOR) operations. Because of the inherent computation feedback, the performance of the system is usually determined by the GF(2^{128}).

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

```
Input A, H ∈ GF(2^128), F(x) Field Polynomial.
Output X
X=0
for $i = 0$ to 127 do
if $A_i = 1$ then
$X \longleftarrow X \oplus H$
end if
if $H_{127} = 0$ then
$H \longleftarrow rightshift(H)$
else
$H \longleftarrow rightshift(H) \oplus F(x)$
end if
end for
return X
```

Algorithm 1 describes the GF(2^{128}) multiplier. Serial implementation of **Algorithm 1** performs the multiplication process in 128 clock cycles. Parallel method can be implemented like [3] and it takes 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 [4]. This design proposed by [4] can be used efficiently (smaller area) on FPGAs as the circuit is specialized for **H**. We integrated this multiplier proposed by [4] with a key-synthesized AES engine in [5] in order to support slow changing key applications like Virtual Private Networks (VPNs). Also, in [5], we proposed a protocol to secure the FPGA reconfiguration to protect the bitstream because it is a key-based bitsream. The disadvantage of this method is the new reconfiguration which must be downloaded on the FPGA in case of changing the key.

Karatsuba Ofman Algorithm (KOA) is used to reduce the complexity (consumed area) of the GF(2^{128}) multiplier. 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 can be described as:

$$\begin{cases} D_l &= A_l \times B_l \\ D_{hl} &= (A_h \oplus A_l) \times (B_h \oplus B_l) \\ D_h &= A_h \times B_h \\ D &= D_h X^m \oplus X^{m/2}(D_h \oplus D_{hl} \oplus D_l) \oplus D_l \end{cases} \tag{1}$$

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.

$$C(x) = D \; mod \; P(x) \tag{2}$$

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

$$P(x) = x^{128} + x^7 + x^2 + x + 1 \tag{3}$$

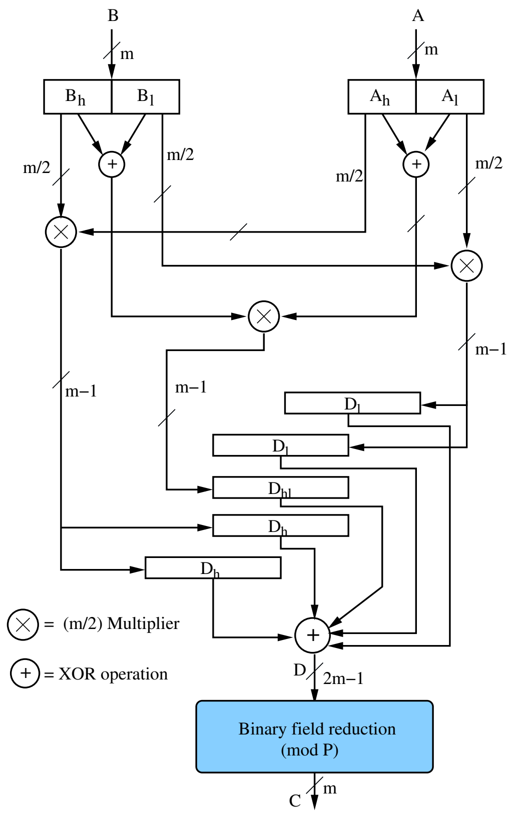

Fig. 2. Polynomial Multiplication using KOA

KOA was used by [6] to reduce the complexity (consumed area) of the GF(2^{128}) multiplier as shown in Fig. 3a. From Fig. 3a, the MAC calculation is as follows:

$$MAC = (C_i \oplus Z_{i-1}) \times H \tag{4}$$

The drawback of the architecture presented in [6] is the the critical delay resulting from the multiplication stage. In order to reduce the data path (critical delay) of the KOA multiplier, pipelining concept was accomplished by [7] as shown in Fig. 3b. Equation 4 was written by [7] as follows:

$$MAC = Q_1 \oplus Q_2 \oplus Q_3 \oplus Q_4, where \tag{5}$$

$$Q1 = (((C_1 \times H^4 \oplus C_5) \times H^4 \oplus C_9) \times H^4 \oplus ....) \times H^4 \tag{6}$$

$$Q2 = (((C_2 \times H^4 \oplus C_6) \times H^4 \oplus C_{10}) \times H^4 \oplus ....) \times H^3 \tag{7}$$

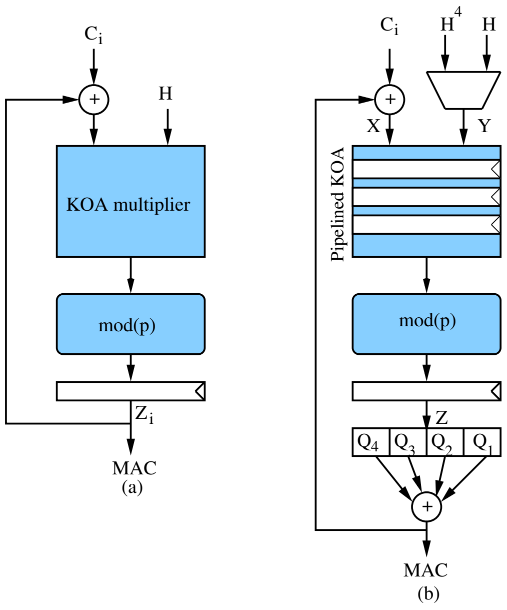

Fig. 3. (a) KOA based GHAH; (b) Pipelined KOA based GHASH

$$Q3 = (((C_3 \times H^4 \oplus C_7) \times H^4 \oplus C_{11}) \times H^4 \oplus ....) \times H^2 \tag{8}$$

$$Q4 = (((C_4 \times H^4 \oplus C_8) \times H^4 \oplus C_{12}) \times H^4 \oplus ....) \times H \tag{9}$$

The hardware architecture proposed by [7] (Fig. 3b) is a 4-stage pipelined KOA-based GHASH. An example of data flow control for the GHASH is shown in Table 1, 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 1, the generated MAC resulting from 8 frames of 128-bit is ready after 19 clock cycles. Therefore, the real throughput is calculated as follows:

$$Throughput(Mbps) = F_{max(MHz)} \times 128 \times (\frac{8}{19}) \tag{10}$$

The last component of Equation 10 is $(\frac{8}{19})$, it is called the reduction factor and the authors of [7] neglected this component in their throughput calculation. Therefore, their presented design of GHASH has not increased the throughput.

Table 1. Data flow control for GHASH calculation by [7]

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

Henzen et al. [8] 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:

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

The authors of [8] 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 11.

3 Efficient KOA-Based GHASH

Four different architectures of FPGAs-based AES-GCM have been presented in the open literature ([5],[7],[6],[8]). It is clear that these contributions do generally have different challenges related to the performance of their architectures. The performance of the architecture presented by [5] is limited because a new reconfiguration is needed in case of changing the key. Also, Zhou et al.[7] claimed the throughput improvement to their previous KOA-based GHASH [6] by using pipelined KOA but we discussed how their method is not efficient for throughput improvement as shown in Equation 10. Also, in [8], the authors claimed that their parallel architecture increased the throughput by 4 because they presented four parallel AES-GCM but we proved that their design is not efficient in terms of increasing the speed as shown in Equation 11.

In this work, in order to improve the performance of AES-GCM, an efficient pipelined KOA-based GHASH is presented. As the targeted platform is FPGA, FPGA-specific properties are considered for performance improvement.

The KOA is selected to reduce the complexity (consumed area) of the classic school multiplication as presented by [7]. Therefore, our presented GHASH uses the KOA for performing the $GF(2^{128})$ multiplication.

As shown in Equation 4, The generation of the MAC 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 as follows:

$$\begin{aligned} MAC &= (C_i \oplus Z_{i-1}) \times H \\ &= (C_i \times H) \oplus (Z_{i-1} \times H) \\ &= (C_i \times H) \oplus [(C_{i-1} \oplus Z_{i-2}) \times H^2] \\ &= (C_i \times H) \oplus (C_{i-1} \times H^2) \oplus [(C_{i-2} \oplus Z_{i-3}) \times H^3] \\ &= (C_i \times H) \oplus (C_{i-1} \times H^2) \oplus (C_{i-2} \times H^3) \\ &\oplus [(C_{i-3} \oplus Z_{i-4}) \times H^4] \\ &= (\underbrace{(C_i \times H)} \oplus \underbrace{(C_{i-1} \times H^2)} \oplus \underbrace{(C_{i-2} \times H^3)} \\ &\oplus \underbrace{(C_{i-3} \times H^4)} .... \oplus \underbrace{(C_2 \times H^{i-1})} \oplus \underbrace{(C_1 \times H^i)} \end{aligned} \tag{12}$$

According to Equation 12, the feedback resulting from XORing the input C_i and the previous output Z_{i-1} is removed because the final MAC is calculated from the last two lines of the equation.

Assume that there are 64 frames of 128-bit and the generation of MAC is required. Therefore, Equation 12 will be as follows:

$$\begin{aligned} MAC_{64} = (&\underbrace{(C_{64} \times H)} \oplus \underbrace{(C_{63} \times H^2)} \oplus \underbrace{(C_{62} \times H^3)} \\ &\oplus \underbrace{(C_{61} \times H^4)} .... \oplus \underbrace{(C_2 \times H^{63})} \oplus \underbrace{(C_1 \times H^{64})} \end{aligned} \tag{13}$$

If the values form H to H^{64} are stored and multiplied to the input C_i as shown in Equation 13, the pipelined architecture can be simply performed. Indeed, the architecture developed for pipelined KOA-based GHASH is in Fig. 4. 4-stage pipelined KOA is used. In terms of the complexity, we used 2-step KOA like [7]. The description of Fig. 4 is presented according to the assumption of calculating the MAC of 64 frames of 128-bit. We divide the process of MAC generation into two steps:

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 initialization stage.

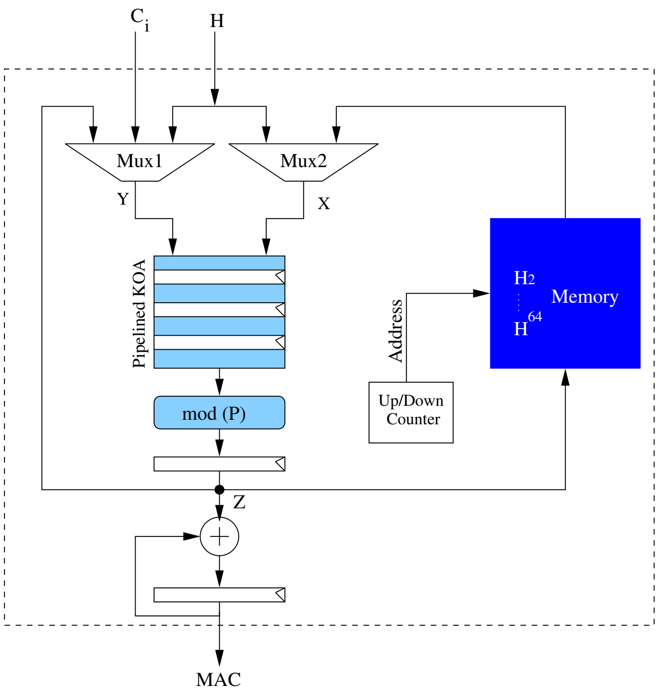

Fig. 4. Proposed pipelined KOA-based GHASH

After initializing the memory, the second step concerns with MAC generation as presented in Equation 13. 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 MAC is calculated by XORing Z values (Equation 13). In terms of the time taken to generate the MAC, 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:

$$Throughput(Mbps) = F_{max(MHz)} \times 128 \times \frac{64}{69} \qquad (14)$$

The proposed architecture reduces the reduction factor compared to [7] from $\frac{8}{19}$ to $\frac{64}{69}$. Therefore, the developed architecture presents the throughput improvement compared to [7]. 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 compared to [4].

Because of targeting our architecture on Xilinx Virtex5 FPGAs, we recommend using CLBs for memory implementation because of 6-input Look-Up-Tables (LUT). Otherwise, using BRAMs is another solution.

4 High Throughput 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. 5 shows the proposed high throughput architecture for AES-GCM. First, the pipelined AES engine generates H by encrypting "00..00" frame. Second, the proposed GHASH needs 252 clock cycles in order to initialize the memory as we described before. Third, the AES engine changes its mode to be in counter mode for performing encryption and delivering C_i to the proposed GHASH.

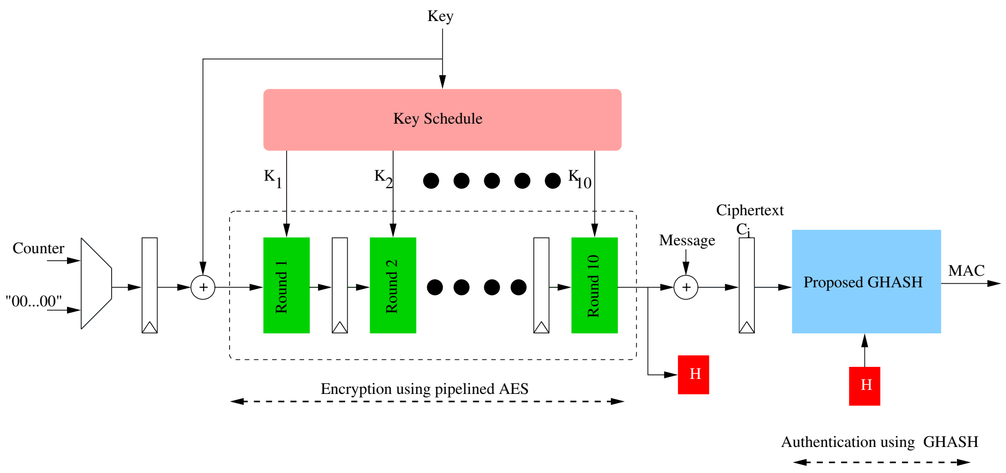

Fig. 5. Proposed AES-GCM architecture

The SubBytes transformation of AES can be implemented either by BRAMs, composite field approach, or direct LUT approach as shown in Fig. 6. Modern FPGAs contain BRAMs. Therefore, implementing SubBytes using BRAMs decreases the consumed slices of the FPGA. The LUT approach is especially interesting on Virtex5 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.

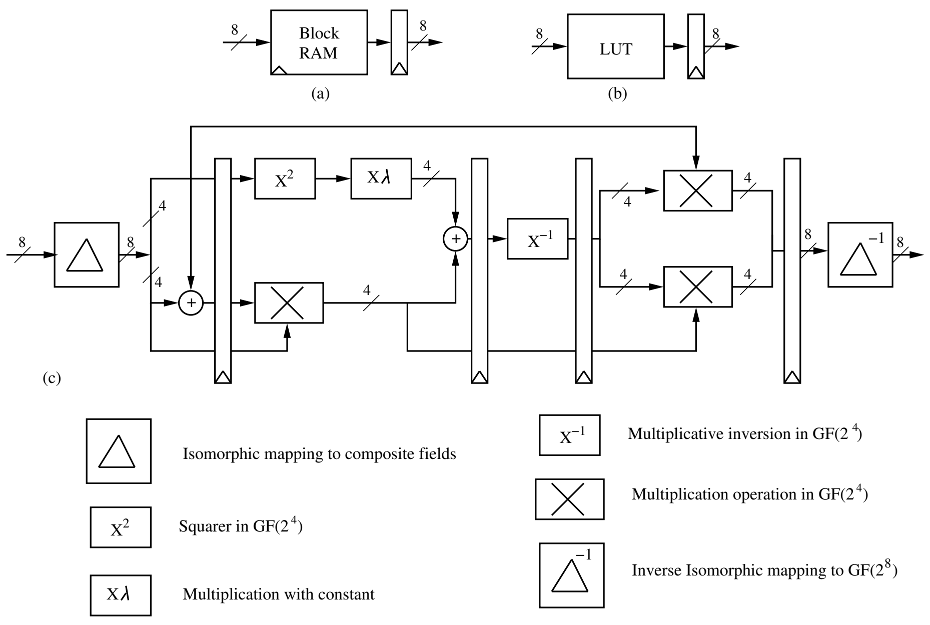
Block RAM
LUT
(a)
(b)
(c)
8
4
Isomorphic mapping to composite fields
Squarer in GF(2^4)
Multiplication with constant
Multiplicative inversion in GF(2^4)
Multiplication operation in GF(2^4)
Inverse Isomorphic mapping to GF(2^8)

Fig. 6. SubBytes implementation with BlockRAMs (a), with LUTs (b), with composite field approach (c)

The proposed 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 counter mode and the proposed GHASH respectively. Therefore, the proposed architecture could also be tuned to handle the decryption and authentication. Indeed, C_i is XORed with the output of the pipelined AES for performing the decryption and also passed to the proposed GHASH for MAC generation.

5 Hardware comparison

We coded our proposed scheme in VHDL and targeted to Virtex5 (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 2. Hardware comparison

	FPGA type	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**
[7]	Virtex5	AES-GCM	●	BRAM	3533	41	314	16.9	4.78
[7]	Virtex5	AES-GCM	●	Comp	6492	0	314	16.9	2.60
[7]	Virtex5	AES-GCM	●	LUT	4628	0	324	17.5	3.77
[8]	Virtex5	AES-GCM	●	BRAM	9561	450	233	48.8	5.1
[8]	Virtex5	AES-GCM	●	Comp	18505	0	233	48.8	2.64
[8]	Virtex5	AES-GCM	●	LUT	14799	0	233	48.8	3.29
[5]	Virtex5	AES-GCM	○	BRAM	2478	40	242	30.9	12.5
[5]	Virtex5	AES-GCM	○	Comp.	5512	0	232	29.7	5.38
[5]	Virtex5	AES-GCM	○	LUT	3211	0	216.3	27.7	8.62

Table 2 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 Virtex5 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 occupies 4770 and reaches the throughput of 36.92 Gbps.

By comparing our results of AES-GCM to [7], 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 10 and Equation 14.

In terms of the 4-parallel AES-GCM by [8], our area consumption (Slices + BRAMs) is smaller compared to them. Also, our performance is better because the throughput presented by [8] is calculated as shown in Equation 11.

Our previous work [5] presented architectures for slow changing key applications like VPNs and the FPGA needs the new reconfiguration when the key changes. Therefore, the proposed architecture in this paper presents better performance compared to [5] because the FPGA does not need a new reconfiguration when the key changes but it needs 252 clock cycles for the memory initialization as described in **Section 3**.

6 Conclusion

In this paper, we presented the performance improvement of AES-GCM (Thr. /Slice). This was achieved by modifying the architecture of the pipelined KOA-based GHASH. With our proposed GHASH, the throughput reduction factor is decreased. Therefore, the throughput of the proposed AES-GCM architectures is increased. In addition, three AES implementations (BRAMs-based SubBytes, composite field-based SubBytes, and LUT-based SubBytes) were evaluated in order to increase the flexibility of the presented work. The throughput of the presented AES-GCM cores ranges from 31.36 to 36.92 using Xilinx Virtex5 FPGAs. It is shown that the performance of the presented AES-GCM architectures outperforms the previously reported ones.

References

1. D. McGrew and J. Viega, "The Security and Performance of the Galois/Counter Mode (GCM) of Operation," *Progress in Cryptology-INDOCRYPT 2004*, pp. 377–413, 2005.
2. "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.*
3. A. Satoh, "High-Speed Hardware Architectures for Authenticated Encryption Mode GCM," *IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 4–pp, 2006.
4. J. Crenne, P. Cotret, G. Gogniat, R. Tessier, and J. Diguet, "Efficient Key-Dependent Message Authentication in Reconfigurable Hardware," *International Conference on Field-Programmable Technology (FPT)*, pp. 1–6, 2011.
5. K. M. Abdellatif, R. Chotin-Avot, and H. Mehrez, "High Speed Authenticated Encryption for Slow Changing Key Applications Using Reconfigurable Devices ," *IEEE Wireless Days*, 2013.
6. G. Zhou, H. Michalik, and L. Hinsenkamp, "Efficient and High-Throughput Implementations of AES-GCM on FPGAs," *International Conference on Field-Programmable Technology (FPT)*, pp. 185–192, 2007.
7. G. Zhou and H. Michalik, "Improving Throughput of AES-GCM with Pipelined Karatsuba Multipliers on FPGAs," *Reconfigurable Computing: Architectures, Tools and Applications*, pp. 193–203, 2009.
8. L. Henzen and W. Fichtner, "FPGA Parallel-Pipelined AES-GCM Core for 100G Ethernet Applications," pp. 202–205, 2010.