Efficient and High-Performance Parallel Hardware Architectures for the AES-GCM

Mehran Mozaffari-Kermani, *Member*, *IEEE*, and Arash Reyhani-Masoleh, *Member*, *IEEE*

Abstract—Since its acceptance as the adopted symmetric-key algorithm, the Advanced Encryption Standard (AES) and its recently standardized authentication Galois/Counter Mode (GCM) have been utilized in various security-constrained applications. Many of the AES-GCM applications are power and resource constrained and require efficient hardware implementations. In this paper, different application-specific integrated circuit (ASIC) architectures of building blocks of the AES-GCM algorithms are evaluated and optimized to identify the high-performance and low-power architectures for the AES-GCM. For the AES, we evaluate the performance of more than 40 S-boxes utilizing a fixed benchmark platform in 65-nm CMOS technology. To obtain the least complexity S-box, the formulations for the Galois Field (GF) subfield inversions in $GF(2^4)$ are optimized. By conducting exhaustive simulations for the input transitions, we analyze the average and peak power consumptions of the AES S-boxes considering the switching activities, gate-level netlists, and parasitic information. Additionally, we present high-speed, parallel hardware architectures for reaching low-latency and high-throughput structures of the GCM. Finally, by investigating the high-performance $GF(2^{128})$ multiplier architectures, we benchmark the proposed AES-GCM architectures using quadratic and subquadratic hardware complexity $GF(2^{128})$ multipliers. It is shown that the performance of the presented AES-GCM architectures outperforms the previously reported ones in the utilized 65-nm CMOS technology.

Index Terms—Advanced encryption standard, Galois/Counter mode, high performance, low power.

1 Introduction

The Advanced Encryption Standard-Galois/Counter Mode (AES-GCM) provides authentication and confidentiality for sensitive data simultaneously. In the AES-GCM, data confidentiality is provided by the Advanced Encryption Standard (AES) [1]. The AES was accepted by the National Institute of Standards and Technology (NIST) in 2001 as the replacement for the previous cryptographic standards. Since then, it has been included in wireless standards of Wi-Fi [2] and WiMAX [3] and many other applications, ranging from the security of smart cards to the bitstream security mechanisms in FPGAs [4]. The authentication of the AES-GCM is provided by the Galois/Counter Mode (GCM) [5] using a universal hash function. The AES-GCM has been used for a number of applications such as the new LAN security standard WLAN 802.1ae (MACSec) [6] and Fibre Channel Security Protocols (FC-SP) [7]. Moreover, it has been utilized in a number of cores from industry, see, for example, [8], [9], and [10]. In addition, two AES-GCM software-based implementations have been presented in [11] and [12].

Among the transformations in the AES encryption, the *SubBytes* (S-boxes) is the only nonlinear one, requiring the highest area and consuming much of the AES power [13]. Therefore, the performance metrics of the S-boxes affect those for the entire AES encryption significantly. For low-complexity implementations, the S-box can be realized using logic gates in composite fields. These S-boxes can also be pipelined for achieving high performance. On the other hand, the S-boxes based on lookup tables (LUTs) could be area efficient when implemented utilizing the memory resources available on FPGAs. In some previous works such as [13], [14], [15], [16], [17], and [18], one specific S-box and in [19], three reported S-boxes have been synthesized on application-specific integrated circuit (ASIC). However, exhaustive search has not been performed for all suitable composite fields to evaluate their performance metrics using the same technology.

It is also noted that in some other works, see, for instance, [20], [21], [22], [23], [24], and [25], the hardware and timing complexities of different composite field S-boxes have been evaluated in terms of logic gates (in [26], software implementations have been performed). However, benchmarking the performance (including power consumptions through simulation-based approaches) of the S-boxes implementations on hardware platforms has not been performed in these works.

In this paper, we optimize logic gates and perform comprehensive ASIC syntheses of more than 40 different S-boxes for deriving their performance metrics. This benchmarking, which is done on the same platform, results in having a clear picture of the performance metrics of different designs.

Different GCM architectures have been presented in the literature. In [27], [28], and [29], the sequential method for the hardware implementation of the GCM function is adopted. Although this method of realization is area efficient, it needs many clock cycles (equal to the number of input blocks), reducing the performance of the

- *M. Mozaffari-Kermani is with the Department of Electrical Engineering, Princeton University, Princeton, NJ 08544, USA. E-mail: mozafari@princeton.edu.*
- *A. Reyhani-Masoleh is with the Department of Electrical and Computer Engineering, The University of Western Ontario, London, ON N6A 5B9, Canada. E-mail: areyhani@eng.uwo.ca.*

Manuscript received 14 Oct. 2010; revised 15 Mar. 2011; accepted 19 June 2011; published online 8 July 2011.
Recommended for acceptance by E. Antelo.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2010-10-0566.
Digital Object Identifier no. 10.1109/TC.2011.125.

architecture. Because of the low throughput of the sequential method, a parallel method is proposed in [30] which uses two $GF(2^{128})$ multipliers to perform this operation in parallel. This parallel implementation has been generalized in [31] and [32] so that q, $q \geq 2$, parallel $GF(2^{128})$ multiplications are performed concurrently. Very recently, a high-performance approach for computing the $GHASH_H$ function for long messages has been proposed in [33]. However, in this scheme, the hardware complexity is increased. In this paper, a high-performance parallel method for obtaining the GCM by relying on the low-complexity powers of the hash subkey is proposed. Without precomputing the hash subkey exponents, compact realizations of these exponents are obtained and implemented. This results in high-throughput and low-latency GCM hardware architectures, suitable for high-performance applications.

The contributions of this paper by referring to their corresponding sections are summarized as follows:

- Logic-gate minimizations for the polynomial basis (PB) inversion in $GF(2^4)$ are performed (Lemma 1). Moreover, we synthesize the structures of different AES S-boxes using the Synopsys Design Vision (which is the graphical user interface to Synopsys Design Compiler) [34] in STM 65-nm CMOS standard technology [35] (Section 3). Then, the areas and delays of these hardware architectures are derived and compared.
- To achieve the least dynamic power-consuming AES S-box, we obtain the average and peak power consumptions of the S-boxes through exhaustive searches considering the possible $2^8(2^8-1) = 65{,}280$ input transitions. These derivations are based on a timing simulation-based analysis using the switching activities of internal nodes with Synopsys PrimeTime PX [34] and ModelSim [36].
- Using a complexity reduction technique, the hardware complexities of different architectures for the subkey exponentiations in the GCM are reduced (Section 4). Then, by utilizing these low-complexity exponentiations, we propose efficient architectures for the GCM, yielding high throughput and low latency (Algorithm 1).
- Finally, the proposed hardware architectures for the AES-GCM are synthesized considering two types of $GF(2^{128})$ multipliers (Section 5). We investigate the performance of quadratic and six different subquadratic complexity $GF(2^{128})$ multipliers (Table 5). It is shown that the proposed architectures for the AES-GCM have higher throughput and efficiency and reach lower latency compared to the previously reported ones (Table 7 and Fig. 8).

The organization of this paper is as follows: in Section 2, preliminaries related to the AES-GCM are presented. In Section 3, logic-gate optimizations for the inversions in $GF(2^4)$ within the S-boxes are presented and we present the results of our syntheses for different S-boxes. Power consumption derivations and comparisons of the S-boxes through a simulation-based method are also presented in this section. In Section 4, the proposed high-performance architectures for implementing the GCM are presented. Section 5 presents the ASIC syntheses and comparisons of the proposed architectures and the previously reported ones. Finally, conclusions are made in Section 6.

2 The AES-GCM

In this section, we present preliminaries for the AES-GCM algorithm. In what follows, the AES (used for confidentiality) and the GCM (used for authentication) algorithms in the AES-GCM and their hardware architectures are presented.

2.1 The Advanced Encryption Standard

In the AES-GCM, only the AES encryption is utilized with the input and the output blocks of 128 bits. However, based on the security requirements, the key size could be determined as AES-128 (with 10 rounds), AES-192 (with 12 rounds), or AES-256 (with 14 rounds) [1]. In the AES encryption, all the rounds except for the last round have four transformations of *SubBytes*, *ShiftRows*, *MixColumns*, and *AddRoundKey*. For the last round, *MixColumns* is eliminated and only three transformations of *SubBytes*, *ShiftRows*, and *AddRoundKey* are used.

The transformation *SubBytes* (S-boxes) is implemented by 16 S-boxes. In the S-box, each byte of the input state is substituted by a new byte. In *ShiftRows*, the first row of the state remains intact and the four bytes of the last three rows of the input state are cyclically shifted. In the *MixColumns* transformation, each column is modified individually and in the final transformation, *AddRoundKey*, modulo-2 addition of the input state and the key of the corresponding round is performed [1].

For realizing the S-box, the irreducible polynomial of $P(x) = x^8 + x^4 + x^3 + x + 1$ is used to construct the binary field $GF(2^8)$. Let $I \in GF(2^8)$ and $O \in GF(2^8)$ be the input and the output of the S-box. Then, the S-box consists of finding the multiplicative inversion (MI), i.e., $I^{-1} \in GF(2^8)$ with the exception of mapping the zero input to the zero output, followed by the affine transformation in $GF(2^8)$ [1].

In what follows, we present the preliminaries regarding the hardware implementations of the S-boxes within the AES using LUTs and composite fields.

2.1.1 LUT-Based S-Boxes

The AES S-boxes can be implemented using LUTs. For this purpose, 256×8 memory cells are used to store the 256 possible 8-bit outputs of each S-box. The LUT-based implementation of the S-boxes is suitable for the FPGA platforms in which block memories are available, see, for example, [37], [38], and [39]. However, although this implementation reaches high-speed architectures, it is not suitable for applications requiring low-complexity AES ASIC implementations [40].

The usage of arithmetic in composite fields reduces the space complexity of the S-box. Moreover, it allows us to use pipelining and therefore the effective speed of the AES encryption is increased while processing independent messages. Consequently, the S-boxes implemented using composite fields can lead to area-efficient and high-performance structures [40]. In the following, the preliminaries on composite field realizations of the S-boxes are presented.

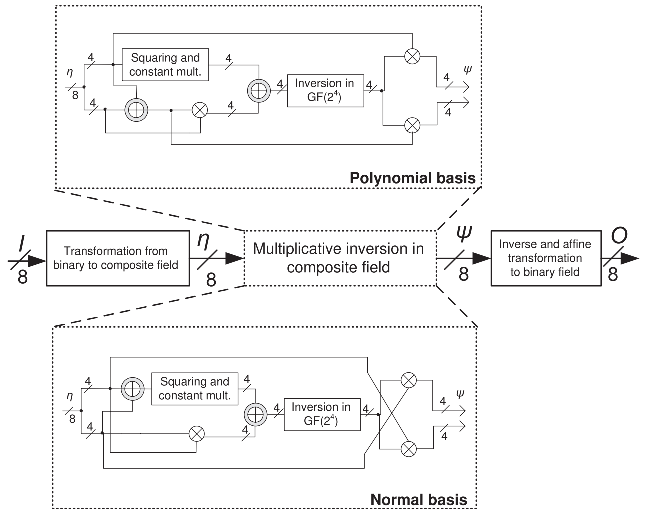

Squaring and constant mult.
Inversion in GF(2⁴)
Polynomial basis
Transformation from binary to composite field
Multiplicative inversion in composite field
Inverse and affine transformation to binary field
Normal basis

Fig. 1. The composite field S-box architecture using polynomial basis and normal basis.

2.1.2 *Composite Field S-Boxes*

Let us briefly explain the composite field arithmetics to calculate the multiplicative inversion over $GF(2^8)$. The composite field realizations of the S-box using polynomial and normal bases are presented in Fig. 1. As seen in this figure, a transformation matrix transforms a field element $I \in GF(2^8)$ to the corresponding representation in the composite field $GF(16^2)$, i.e., η. We consider the irreducible polynomial of $u^2 + u + \nu$, where ν is chosen over $GF(16)$ depending on the composite fields. Then, the multiplicative inversion generates the inverse as $\psi = \eta^{-1}$. Finally, as seen in Fig. 1, the inverse transformation matrix transforms the composite field element to the one in the binary field, i.e., $O \in GF(2^8)$.

Using polynomial basis constructed by the irreducible polynomial of $u^2 + u + \nu$, one can obtain the coordinates of ψ as $\psi_h = \eta_h(\eta_h^2\nu + \eta_h\eta_l + \eta_l^2)^{-1}$ and $\psi_l = (\eta_l + \eta_h)(\eta_h^2\nu + \eta_h\eta_l + \eta_l^2)^{-1}$ [14]. This multiplicative inversion in composite fields using polynomial basis is shown in the top part of Fig. 1 by a dotted rectangle. Similarly, for normal basis, the coordinates of ψ are obtained as $\psi_h = (\eta_h\eta_l + ({\eta_h}^2 + {\eta_l}^2)\nu)^{-1}\eta_l$ and $\psi_l = (\eta_h\eta_l + ({\eta_h}^2 + {\eta_l}^2)\nu)^{-1}\eta_h$ [20], shown in the dotted rectangle in the bottom of Fig. 1. One can refer to [14] and [20] for more details on the composite field S-box architectures. As seen in Fig. 1, the above multiplicative inversions consist of composite field multiplications, additions, and inversion in the subfield $GF(2^4)$. In this figure, the subfield multiplications are shown by crossed circles. Moreover, the circle with plus inside represents $GF(2^4)$ addition using four XOR gates.

2.2 The Galois/Counter Mode

Authenticated encryption and decryption are the two functions within the GCM. The authenticated encryption performs two tasks; encrypting the confidential data and computing an authentication tag. The authenticated decryption function decrypts the confidential data and verifies the tag [5]. The data flow of the authenticated encryption is shown in Fig. 2. As seen in this figure, the mechanism for the confidentiality of data is a variation of the block cipher counter mode of operation, denoted by $GCTR_K$ (Galois Counter with the key K) [5]. For the AES-GCM, the block cipher encryption with the specific key K is shown by AES_K in Fig. 2. Then, the function $GCTR_K$ performs the block cipher counter mode with the *Initial Counter Block (ICB)* and its increments ($CB_2 - CB_i$) and the plaintext blocks ($P_1 - P_i$) as the inputs.

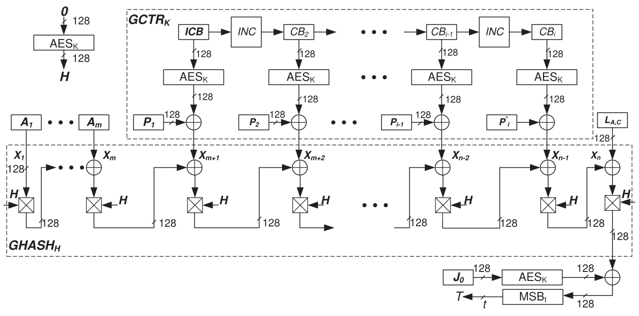


Fig. 2. The GCM authenticated encryption data flow [5].

As shown in Fig. 2, the Galois Hash ($GHASH_H$) function within the GCM provides the authentication for the confidential data. This function is constructed by $GF(2^{128})$ multiplications with a fixed parameter, called the hash subkey (H). The $GHASH_H$ function calculates

$$\sum_{j=1}^{n} X_j H^{n-j+1} = X_1 \cdot H^n \oplus X_2 \cdot H^{n-1} \oplus \ldots \oplus X_n \cdot H, \quad (1)$$

where X_1 to X_n are the n, 128-bit blocks of the input [5]. It is noted that the hash subkey is generated by applying the AES to the *zero* block, i.e., $0 = (0, 0, ..., 0) \in GF(2^{128})$. Then, the $GHASH_H$ function calculates (1) [5]. All the arithmetic operations in (1), i.e., additions, Galois Field (GF) multiplications, and exponentiations are performed over $GF(2^{128})$ constructed by the irreducible polynomial $P(x) = x^{128} + x^7 + x^2 + x + 1$. As seen in Fig. 2, the total number of input blocks to $GHASH_H$ is $n = m + i + 1$, where m and i are the number of blocks for the additional authenticated data (AAD) ($A_1 - A_m$) and the output of $GCTR_K$, respectively. Eventually, the authentication tag T with length of t bits is derived. In the authenticated decryption, the same $GHASH_H$ procedure is performed on the authenticated data and ciphertext blocks to verify the tag. For the entire description of the GCM, one can refer to [5].

3 Performance Evaluations and Comparisons of the AES S-Boxes

In this section, logic-gate optimizations for reducing the complexity of the S-boxes are presented. We also present the implementation results for benchmarking the performance of different S-boxes. Finally, power consumption derivations and comparisons of the S-boxes through a simulation-based method are also presented.

The implementation complexities of the S-boxes using composite fields are dependent on the choice of the coefficients $\nu \in GF(2^4)$ and $\Phi \in GF(2^2)$ in the irreducible polynomials $u^2 + u + \nu$ and $v^2 + v + \Phi$ used for the composite fields, respectively.

The composite fields $GF(((2^2)^2)^2)$ in polynomial basis use iterations to construct the S-box. For these composite fields, the constants $\nu \in GF(2^4)$ and $\Phi \in GF(2^2)$ are over $GF((2^2)^2)/v^2 + v + \Phi$ and $GF(2^2)/x^2 + x + 1$, respectively. According to [22], after exhaustive search for finding the possible choices for $\nu \in GF(2^4)$ and $\Phi \in GF(2^2)$, the following 16 combinations are obtained: $\Phi \in \{\{10\}_2, \{11\}_2\}$ and

$$\nu \in \{\{1000\}_2, \{1001\}_2, \{1010\}_2, \{1011\}_2, \{1100\}_2, \{1101\}_2, \{1110\}_2, \{1111\}_2\}.$$

Similarly, for normal basis, it can be derived that the only two acceptable values for Φ are $\Phi = \{10\}_2$ and $\Phi = \{01\}_2$. Furthermore, the following eight values of ν are acceptable:

$$\nu \in \{\{0100\}_2, \{0001\}_2, \{1000\}_2, \{0010\}_2, \{0111\}_2, \{1101\}_2, \{1011\}_2, \{1110\}_2\}.$$

Based on the possible values of ν and Φ in polynomial basis representation, the (inverse) transformation matrices can be constructed using the algorithm presented in [16]. In this algorithm, using an exhaustive search, the transformation matrix is constructed using eight base elements in $GF(((2^2)^2)^2)$, i.e., $1, \xi, \xi^2, \ldots, \xi^7$, to which eight base elements of $GF(2^8)$ are mapped. We note that for each combination of ν and Φ, there exist eight possible (inverse) transformation matrices. These are constructed according to the base element ξ and the conjugates of this base element, i.e., ξ^{2^i}, $i = 1, 2, \ldots, 7$. In this section, for each combination of ν and Φ, one of these possible matrices is considered. As suggested in [16], we have also used subexpression sharing for obtaining the low-complexity implementations for these matrices. We note that different (inverse) transformation matrices in normal basis are derived simply by reordering the columns.

3.1 Logic-Gate Optimizations

The field inversion in $GF(2^4)$ of the most compact composite field in [20] has been modified in [18] to decrease its hardware complexity. This field uses normal basis with $\Phi = \{10\}_2$ and $\nu = \{0001\}_2$. Now, we consider polynomial basis to further optimize the S-boxes using polynomial basis. We present the following lemma through which the hardware complexity of the composite field inversion in $GF(2^4)$ is decreased. This is performed by presenting low-complexity formulations for the inversion in $GF(2^4)$ through logic-gate minimization. Moreover, these formulations are implemented using NAND, NOR, and XOR gates for reducing the complexity.

Lemma 1. *Let $\gamma = (\gamma_3, \gamma_2, \gamma_1, \gamma_0)$ be the input and $\theta = (\theta_3, \theta_2, \theta_1, \theta_0)$ be the output of an inverter in $GF(2^4)$. Then, the formulations for the low-complexity inversion in $GF(2^4)$ using polynomial basis with $\Phi = \{11\}_2$ are as follows:*

$$\begin{aligned} \theta_3 &= \gamma_2\overline{\gamma_3\gamma_1} + \gamma_3\gamma_0, \\ \theta_2 &= \gamma_3\overline{\gamma_0} \vee \gamma_2(\gamma_3 \vee \gamma_1), \\ \theta_1 &= \gamma_2\overline{\gamma_0} \vee \gamma_3\overline{\gamma_1}\gamma_0 \vee \overline{\gamma_3}\gamma_1\overline{\gamma_2}, \\ \theta_0 &= \gamma_3 \vee \gamma_1\overline{\gamma_0} \vee \overline{\gamma_2}\gamma_0\overline{\gamma_1} + \gamma_1(\gamma_2 \vee \gamma_3\overline{\gamma_0}). \end{aligned} \tag{2}$$

Moreover, for $\Phi = \{10\}_2$, one reaches the following:

$$\begin{aligned} \theta_3 &= \gamma_2\overline{\gamma_3\gamma_1} + \gamma_3\overline{\gamma_0}, \\ \theta_2 &= \gamma_2\overline{\gamma_1} \vee \gamma_3(\gamma_2 \vee \gamma_0), \\ \theta_1 &= \gamma_3\gamma_1(\gamma_2 \vee \gamma_0) \vee \gamma_2\gamma_0 + \gamma_3 + \gamma_2 + \gamma_1, \\ \theta_0 &= \gamma_0 \vee \gamma_2\overline{\gamma_3} \vee \overline{\gamma_1}\gamma_3\overline{\gamma_2} + \gamma_2(\gamma_1 \vee \gamma_0\overline{\gamma_3}), \end{aligned} \tag{3}$$

where "+" and "$\vee$" represent the XOR and OR operations, respectively.

Proof. For $\gamma = (\gamma_3, \gamma_2, \gamma_1, \gamma_0)$ as the input and $\theta = (\theta_3, \theta_2, \theta_1, \theta_0)$ as the output of an inverter in $GF(2^4)$, the formulations for the inversion in $GF(2^4)$ using the polynomial basis with $\Phi = \{11\}_2$ and $\Phi = \{10\}_2$ are obtained as follows, respectively, [16], [22]:

$$\begin{aligned} \theta_3 &= \gamma_3\gamma_2\gamma_1 + \gamma_3\gamma_0 + \gamma_2, \\ \theta_2 &= \gamma_3\gamma_2\gamma_1 + \gamma_3\gamma_2\gamma_0 + \gamma_3\gamma_0 + \gamma_2\gamma_1 + \gamma_3, \\ \theta_1 &= \gamma_3\gamma_2\gamma_1 + \gamma_3\gamma_1\gamma_0 + \gamma_3\gamma_0 + \gamma_3\gamma_1 + \gamma_2\gamma_0 \\ &\quad + \gamma_2\gamma_1 + \gamma_2 + \gamma_1, \\ \theta_0 &= \gamma_3\gamma_2\gamma_1 + \gamma_3\gamma_2\gamma_0 + \gamma_3\gamma_1\gamma_0 + \gamma_2\gamma_1\gamma_0 \\ &\quad + \gamma_2\gamma_0 + \gamma_3\gamma_0 + \gamma_2\gamma_1 + \gamma_3 + \gamma_1 + \gamma_0, \end{aligned} \tag{4}$$

$$\begin{aligned} \theta_3 &= \gamma_3\gamma_2\gamma_1 + \gamma_3\gamma_0 + \gamma_3 + \gamma_2, \\ \theta_2 &= \gamma_3\gamma_2\gamma_1 + \gamma_3\gamma_2\gamma_0 + \gamma_3\gamma_0 + \gamma_2\gamma_1 + \gamma_2, \\ \theta_1 &= \gamma_3\gamma_2\gamma_1 + \gamma_3\gamma_1\gamma_0 + \gamma_2\gamma_0 + \gamma_3 + \gamma_2 + \gamma_1, \\ \theta_0 &= \gamma_3\gamma_2\gamma_1 + \gamma_3\gamma_2\gamma_0 + \gamma_3\gamma_1\gamma_0 + \gamma_2\gamma_1\gamma_0 + \gamma_3\gamma_1 \\ &\quad + \gamma_3\gamma_0 + \gamma_2\gamma_1 + \gamma_2 + \gamma_1 + \gamma_0. \end{aligned} \tag{5}$$

One can obtain θ_3-θ_0 in (2) and (3) from those of (4) and (5), respectively. For performing this, we note that $\gamma_i + 1 = \overline{\gamma_i}$ and $\gamma_i + \gamma_j + \gamma_i\gamma_j = \gamma_i \vee \gamma_j$. For instance, we obtain θ_3 in (2) from that of (4) as $\theta_3 = \gamma_3\gamma_2\gamma_1 + \gamma_3\gamma_0 + \gamma_2 = \gamma_2(\gamma_3\gamma_1 + 1) + \gamma_3\gamma_0 = \gamma_2\overline{\gamma_3\gamma_1} + \gamma_3\gamma_0$. Using similar methods, some of which are also presented in [18], one can obtain (2). As another example, one can obtain θ_3 in (3) from that of (5) as $\theta_3 = \gamma_3\gamma_2\gamma_1 + \gamma_3\gamma_0 + \gamma_3 + \gamma_2 = \gamma_2(\gamma_3\gamma_1 + 1) + \gamma_3(\gamma_0 + 1) = \gamma_2\overline{\gamma_3\gamma_1} + \gamma_3\overline{\gamma_0}$. By verifying the 16 combinations of the input γ, same results are obtained for (2) and (4) ((3) and (5)). □

In what follows, we evaluate and compare the performance metrics of different S-boxes. The presented results confirm efficiency increase using (2) and (3).

3.2 Area and Delay Evaluations

In the following, we evaluate and compare the areas, delays, throughputs, and efficiencies of different S-boxes, including the ones presented in [13], [14], [15], [16], [18], [20], [21], [22], [23], and [25]. It is noted that the implementation in [24] is for the inversion in $GF(2^4)$ and does not provide the entire S-box architecture.

Using MATLAB [41], we have derived the low-complexity transformation and mixed inverse and affine transformation matrices for the syntheses. We have used the STM 65-nm CMOS standard technology and CORE65LPSVT standard cell library [35]. This library is optimized for using in low-power applications. The nominal junction temperature is 25 °C and VHDL has been used as the design entry to the Synopsys Design Vision [34]. We note that the presented results are post synthesis and do not consider the post layout routing.

TABLE 1
Evaluation of the Performance Metrics of the S-Boxes on ASIC Using the STM 65-nm CMOS Standard Technology

Structure	Specifications		Area		Delay [Frequency]	Throughput	Efficiency
	Φ	ν	(μm^2)	GE[a]	(ns) [MHz]	(Gbps)	($\frac{Mbps}{\mu m^2}$)
Polynomial basis	10	1000 [21], [22][b]	525.2	252.5	1.31 [763]	6.1	11.6
		1000 (proposed, using (3))	518.9	249.4	1.15 [869]	7.0	**13.5**[3]
		1001	537.2	258.2	1.43 [699]	5.6	10.4
		1010	535.6	257.5	1.36 [735]	5.9	11.0
		1011	540.8	260.0	1.43 [699]	5.6	10.3
		1100 [13], [14], [16][c]	540.3	259.7	1.37 [730]	5.8	10.8
		1101	548.6	263.7	1.34 [746]	6.0	10.9
		1110	524.7	252.3	1.40 [714]	5.7	10.9
		1110 (proposed, using (3))	**510.2**[3]	245.2	1.25 [800]	6.4	12.5
		1111	535.6	257.5	1.40 [714]	5.7	10.6
	11	1000	528.3	253.9	1.39 [719]	5.8	10.9
		1000 (proposed, using (2))	516.3	248.2	**1.11 [900]**[3]	**7.2**[3]	**13.9**[2]
		1001	534.6	257.0	1.56 [641]	5.1	9.6
		1010 [22][b]	519.0	249.5	1.42 [704]	5.6	10.9
		1010 (proposed, using (2))	**498.4**[2]	239.6	**1.11 [900]**[3]	**7.2**[3]	**14.4**[1]
		1011	531.0	255.2	1.45 [690]	5.5	10.4
		1100	548.1	263.5	1.49 [671]	5.4	9.8
		1101	546.5	262.7	1.42 [704]	5.6	9.8
		1110	542.8	260.9	1.52 [657]	5.3	9.7
		1111	542.9	261.0	1.52 [657]	5.3	9.7
Normal basis	10	0001 [20][b]	569.4	273.7	1.59 [629]	5.0	8.8
		0001 [20], [18][d]	511.7	246.0	1.45 [690]	5.5	10.7
		0010	564.2	271.2	1.42 [704]	5.6	10.0
		0100	576.7	277.2	1.57 [637]	5.1	8.8
		1000	579.3	278.5	1.46 [685]	5.5	9.4
		0111	575.1	276.5	1.56 [641]	5.1	8.9
		1011	589.2	283.2	1.55 [645]	5.2	8.7
		1101	572.0	275.0	1.60 [625]	5.0	8.7
		1110	588.6	282.0	1.57 [637]	5.1	8.6
	01	0001	564.2	272.2	1.58 [633]	5.1	9.0
		0010	577.2	277.5	1.51 [662]	5.3	9.2
		0100	564.2	271.2	1.59 [629]	5.0	8.9
		1000	570.9	274.4	1.58 [633]	5.1	8.9
		0111	583.9	280.7	1.49 [671]	5.4	9.2
		1011	572.5	275.2	1.58 [633]	5.1	8.9
		1101	583.9	280.7	1.48 [676]	5.4	9.3
		1110	571.9	274.9	1.59 [629]	5.0	8.8
Normal basis	-	0001 [23][e]	**403.2**[1]	193.8	1.64 [610]	4.9	12.1
Polynomial basis	-	1110 [15][f]	554.3	266.5	1.26 [794]	6.4	11.5
Mixed basis [25][g]	-	-	571.1	274.5	1.30 [769]	6.2	10.9
LUT/ROM [42], [19][h]	-	-	1,407.1	676.4	**0.60 [1666]**[1]	**13.3**[1]	9.5
LUT/ROM-MI [19][i]	-	-	1,434.6	689.4	**0.68 [1470]**[2]	**11.8**[2]	8.2

1, 2, and 3 are the best cases for each performance metric. a. Gate equivalent in terms of two-input NAND. b. Among all fields considered, the presented composite field has the least hardware complexities in terms of logic-gate counts. c. These are some works in which this composite field is used. d. The hardware complexity of this composite field, which is obtained in [20] as the most compact one, has been improved in [18]. e. This implementation is based on a minimization method resulting in low area at the expense of more timing complexity. f. This architecture is based on the composite field $GF((2^4)^2)$. g. Has been presented very recently based on mixed polynomial and normal bases and only focuses on decreasing the critical path delay. h. Using synthesized ROM-based LUTs. i. LUTs for the multiplicative inversion and logic gates for the affine transformation.

The results of our syntheses are presented in Table 1. As seen in this table, for different S-boxes, the areas (in terms of $\mu \mathrm{m}^2$), critical path delays (in terms of ns), maximum working frequencies (in terms of MHz), throughputs (in terms of Gbps), and efficiencies (in terms of $\frac{Mbps}{\mu m^2}$) have been obtained. According to the STM 65-nm standard cell library information, the lowest and nominal drive strength for the cells is two. It is noted that the area of a NAND gate in the utilized STM 65-nm CMOS library for the drive strength of two is 2.08 $\mu \mathrm{m}^2$. Then, using this area, we have also provided the gate equivalent (GE) measure for different S-boxes in the table. Note that if we increase the area effort, lower areas are usually achieved mostly at the expense of more delay overhead.

Memory macros tend to be expensive in hardware for implementing the S-boxes, resulting in high hardware complexity and power consumption. Therefore, this implementation is not considered in this paper. We have considered two different methods of realization of the LUT S-boxes. In these methods, read-only LUTs are used for implementing the S-box, see, for instance, [42] and the hw-lut/hybrid-lut architectures in [19]. This allows us to logic optimize the S-box architecture by synthesis of hardware description languages, leading to low-area implementations. In the first method (LUT/ROM), the entire S-box is implemented using LUTs. Moreover, we consider the S-boxes in which only the multiplicative inversion in $GF(2^8)$ is implemented using LUTs and the affine transformation is implemented separately (LUT/ROM-MI). This enables the designers to share the multiplicative inversion in $GF(2^8)$ for the S-box and the inverse S-box in the merged structures.

In some of the previous works such as [14], [15], [20], and [21], the area of the S-box has been presented in terms of gate equivalent. For instance, in [14] and [21], the areas of the implemented S-boxes have been provided as 294 GE and 272 GE using 0.11 and $0.18\ \mu\mathrm{m}$ technologies, respectively. Based on the information of the cell library in a $0.18\ \mu\mathrm{m}$ technology, the gate count of the S-box has been converted to gate equivalent as 180 GE in [20]. We note that although FPGA implementations have been performed, the results presented in [20] (unlike those in [14], [15], [21], and this paper) are the direct conversion of the gate count (without synthesis) to GE. In addition, the conversion factor of 1.75 has been used in [20] for obtaining the GE for XOR/XNOR and MUX21. However, in the cell library used in this paper, these conversion factors are 2.25 and 2, respectively [35]. Other parameters which cause different GE results are the type of the synthesis tools used and the map effort specified. In this paper, the synthesis results are obtained using VHDL as the design entry to the Synopsys Design Vision [34].

Using (2) and (3) of Lemma 1, we have also presented the results of the logic-gate optimized S-boxes in Table 1. Specifically, we have used (2) and (3) for two most compact S-boxes using polynomial basis for $\Phi = \{11\}_2$ and $\Phi = \{10\}_2$ in Table 1. It is also noted that for each of the evaluated performance metrics, the three best cases among different results for the S-boxes have been marked with superscripts 1, 2, and 3. As shown in Table 1, the areas for the composite field S-boxes range from 403.2-$589.2\ \mu\mathrm{m}^2$ (difference of 46.1 percent), the working frequencies from 625-900 MHz (difference of 44.0 percent), the throughputs from 5.0-7.2 Gbps (difference of 44.0 percent), and the efficiencies from 8.6-$14.4\ \frac{Mbps}{\mu m^2}$ (difference of 67.4 percent).

As seen in Table 1, the S-boxes using LUTs (last two rows) are the fastest S-boxes. However, their efficiencies are not the highest among other S-boxes in Table 1. Among the composite field S-boxes, the one using normal basis presented in [23] is the most compact one (see the area column in Table 1). However, it has the worst working frequency and throughput. The S-boxes using polynomial basis (optimized using (2)) have the highest frequency and throughput among the composite field S-boxes. Finally, the highest efficiency (see the last column in Table 1) is obtained for the one using polynomial basis with $\Phi = \{11\}_2$ and $\nu = \{1010\}_2$ (optimized using (2)).

3.3 Power Consumptions and Comparisons

In the following, the power consumption results for different S-boxes are presented. We have derived the power consumptions of the S-boxes within the AES through a simulation-based analysis method. In what follows, we present the power derivation method as well as the results of our analysis and comparison.

3.3.1 Power Derivation Method

We use VHDL as the design entry to the Synopsys Design Vision. After obtaining the gate-level netlists of the S-boxes, timing simulations are performed using ModelSim SE $6.2d$ [36]. The testbench used for timing simulations covers all the $256 \times 255 = 65{,}280$ possible transitions for the 8-bit input of the S-box. This exhaustive input pattern assertion includes all the possible transitions between each two different pairs of the possible 256 inputs. Then, for each and every S-box, the results of the switching activities of all internal nodes have been logged in the Value Change Dump (VCD) files. We have set the resolution of the timing simulations to high so that the VCD files contain the switching activities of glitches (dynamic hazards) occurring in the logic gates. Then, as the final step, the power consumption of the circuit is computed from the VCD logs, gate-level netlists, cell information, and parasitics of the target ASIC library. We have utilized the Synopsys PrimeTime PX [34] to obtain the average power (including net switching power, cell internal power, and cell leakage power), peak, and instantaneous power consumption details. It is noteworthy that the power consumption results are for the working frequency of 50 MHz and for the high resolutions for both timing and power consumption.

3.3.2 Analysis and Comparison

The results of our simulation-based power computations are presented in Table 2. As depicted in this table, for different S-boxes, we have derived the average power (in terms of μW), peak power (in terms of mW), and the input pattern transition for which the peak power happens. As shown in Table 2, the average powers for the composite field S-boxes range from 44.39-$58.96\ \mu$W (difference of 32.8 percent) and the peak powers from 1.013-1.324 mW (difference of 30.7 percent). We have also marked (with superscripts 1, 2, and 3) the three cases for which the lowest power consumptions are achieved.

Comparing the results in Tables 1 and 2 shows that generally and with few exceptions, the S-boxes with more hardware complexities consume more power. As seen in Table 2, the highest and lowest average power consumptions are achieved for the LUT-based (using memories) S-box and the normal basis S-box presented in [23], respectively. Based on our results in Table 1, these two S-boxes have the highest and lowest hardware complexities, respectively. On the other hand, according to the results of Table 1, the normal basis S-box presented in [23] has the highest timing complexity among the composite field S-boxes.

The transitions of the inputs of the S-boxes when the peak powers occur have been also shown in Table 2. As shown in this table, most of the peak powers occur when the S-box input changes to the all-zero input.

4 High-Performance GCM Parallel Architecture

In this section, we propose high-performance parallel architectures for the GCM. These architectures improve the throughput and the latency of the structures presented in [31] and [32] for $GHASH_H$. They also remove the need for consecutive $GF(2^{128})$ multiplications with H for deriving (1). We also derive the hardware implementations of the exponentiations of the hash subkey to the powers of two, i.e., in the form of H^{2^j}, needing only XOR gates. Because of the low complexity of the implementations of these exponents, we take advantage of these low-cost hash subkey powers in the proposed high-performance architectures. We utilize the

TABLE 2
Evaluation of the Power Consumptions of the S-Boxes on ASIC Using the STM 65-nm CMOS Standard Technology and the Synopsys PrimeTime PX [34]

Structure	Specification		Average[a]	Peak[b]	
	Φ	ν	(μW)	(mW)	Transition
Polynomial basis	10	1000 [21], [22][c]	54.99	1.283	$78 \rightarrow 00$
		1001	55.77	1.184	$1C \rightarrow 00$
		1010	54.91	1.165	$F4 \rightarrow 58$
		1011	56.45	1.262	$79 \rightarrow 00$
		1100 [13], [14], [16][d]	55.98	1.283	$C0 \rightarrow 00$
		1101	56.63	1.324	$D5 \rightarrow 01$
		1110	55.28	1.313	$B5 \rightarrow 00$
		1111	55.87	1.161	$C3 \rightarrow 07$
	11	1000	54.69	**1.134**^{2}	$27 \rightarrow 00$
		1000 (proposed, using (2))	54.15	1.188	$B8 \rightarrow 00$
		1001	55.44	1.214	$54 \rightarrow 01$
		1010 [21][c]	54.12	**1.145**^{3}	$71 \rightarrow 00$
		1010 (proposed, using (2))	**53.78**^{2}	1.229	$C9 \rightarrow 00$
		1011	54.80	1.218	$55 \rightarrow 00$
		1100	55.13	1.178	$D7 \rightarrow 00$
		1101	56.51	1.244	$BA \rightarrow 00$
		1110	55.91	1.239	$3B \rightarrow 00$
		1111	55.40	1.185	$9C \rightarrow 00$
Normal basis	10	0001 [20][c]	58.02	1.268	$46 \rightarrow 00$
		0001 [20], [18][e]	**54.03**^{3}	1.189	$46 \rightarrow 0A$
		0010	58.51	1.291	$41 \rightarrow 00$
		0100	58.03	1.283	$46 \rightarrow 00$
		1000	58.15	1.290	$41 \rightarrow 00$
		0111	58.79	1.299	$F2 \rightarrow 00$
		1011	58.35	1.247	$91 \rightarrow 00$
		1101	58.81	1.300	$73 \rightarrow 00$
		1110	58.96	1.309	$91 \rightarrow 00$
	01	0001	58.19	1.323	$68 \rightarrow 00$
		0010	58.07	1.284	$92 \rightarrow 00$
		0100	57.90	1.292	$68 \rightarrow 00$
		1000	58.17	1.292	$92 \rightarrow 00$
		0111	58.54	1.291	$43 \rightarrow 00$
		1011	57.88	1.231	$E7 \rightarrow 00$
		1101	58.70	1.282	$42 \rightarrow 00$
		1110	58.23	1.246	$5B \rightarrow 00$
Normal basis	-	0001 [23][f]	**44.39**^{1}	**1.013**^{1}	$27 \rightarrow 00$
Polynomial basis	-	1110 [15][g]	55.48	1.208	$22 \rightarrow 00$
Mixed basis [25]	-	-	58.06	1.242	$46 \rightarrow 00$
LUT/ROM [42], [19]	-	-	63.18	1.337	$91 \rightarrow 00$
LUT/ROM-MI [19]	-	-	66.20	1.344	$91 \rightarrow 00$

1, 2, and 3 are the best cases for each performance metric. a. Includes net switching, cell internal, and cell leakage power. b. Obtained from the instantaneous power values for each case, including the power consumptions for the glitches. c. Among all fields considered, the presented composite field has the least hardware complexities in terms of logic-gate counts. d. These are some works in which this composite field is used. e. The power consumption of this composite field, which is obtained in [20] as the most compact one, has been improved in [18]. f. The lowest power yet the slowest composite field S-box. g. This architecture is based on the composite field $GF((2^4)^2)$.

powers in the form of H^{2^j} to obtain the other powers of the hash subkey with the least number of GF multiplications over $GF(2^{128})$ for proposed architectures. For instance, we derive $H^3 = H^2 \times H$ or $H^6 = H^4 \times H^2$.

4.1 High-Performance $GHASH_H$ Function

Algorithm 1 is used for obtaining the key formulation for the proposed $GHASH_H$ function. Although there is no restriction in choosing q, i.e., the number of parallel adder-multipliers, we use $q = 2^j$, $1 \leq j \leq \lfloor \log_2(n) \rfloor$. This leads to lower number of clock cycles and higher throughput needed for the implementations. In Algorithm 1, the output $GHASH(X, H)$ is obtained as follows:

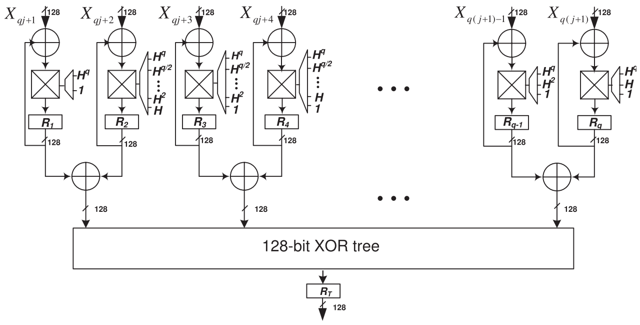


Fig. 3. The hardware architecture of the proposed high-performance GCM $GHASH_H$ function.

$$\begin{aligned} & X_1 \cdot \underbrace{H^q \times \cdots \times H^q}_{\frac{n}{q}\, times} \oplus X_2 \cdot \underbrace{H^q \times \cdots \times H^q}_{\frac{n}{q}-1\, times} \times H^{q-1} \oplus \cdots \\ & \oplus X_j \cdot \underbrace{H^q \times \cdots \times H^q}_{\frac{n}{q}-1\, times} \times H^{q-j+1} \oplus \cdots \\ & \oplus X_q \cdot \underbrace{H^q \times \cdots \times H^q}_{\frac{n}{q}-1\, times} \times H \oplus X_{q+1} \cdot \underbrace{H^q \times \cdots \times H^q}_{\frac{n}{q}-1\, times} \\ & \oplus X_{q+2} \cdot \underbrace{H^q \times \cdots \times H^q}_{\frac{n}{q}-2\, times} \times H^{q-1} \oplus \cdots \oplus X_n H, \end{aligned} \tag{6}$$

where all operations are performed over $GF(2^{128})$ constructed by the irreducible polynomial $P(x) = x^{128} + x^7 + x^2 + x + 1$ and $\bigoplus$ comprises 128 XOR gates.

Algorithm 1. The proposed high-performance approach for implementing the GCM.

```
Inputs: $X_p \in GF(2^{128}), 1 \leq p \leq n$, and $H^{2^j} \in GF(2^{128})$,
  $0 \leq j \leq \log_2(q)$.
Output: $GHASH(X, H) = \sum_{j=1}^{n} X_j H^{n-j+1}$.
 1: for $i = 1$ to $q$ do
 2:     $temp_i \leftarrow X_i$
 3:     for $j = 1$ to $\frac{n}{q} - 1$ do
 4:         $temp_i = (temp_i \times H^q \oplus X_{i+jq})$
 5:     end for
 6:     Let $q - i + 1 = ({a_0}^{(i)}, \ldots, {a_{\log_2(q)}}^{(i)})_2$
 7:     $temp_i = temp_i \times (H^{a_0^{(i)} q} \times H^{\frac{a_1^{(i)} q}{2}} \times \cdots \times H^{a_{\log_2(q)}^{(i)}})$
 8: end for
 9: $GHASH(X, H) = \sum_{i=1}^{q} temp_i$
10: return $GHASH(X, H)$.
```

One can rewrite (6) so that only the exponentiations of the hash subkey to the powers of 2 in the form of H^{2^j} are utilized. This method of exponentiation is based on the binary exponentiation, see, for example, [43]. As seen from this algorithm, for the exponentiations H^{q-i+1}, $1 \leq i \leq q$, one can use the binary representation of $q - i + 1$ as $({a_0}^{(i)}, \ldots, {a_{\log_2(q)}}^{(i)})_2$.

The hardware implementation of Algorithm 1 has been presented in Fig. 3. For implementing Algorithm 1 in hardware, in total, $\frac{n}{q} + \log_2(q)$ clock cycles are needed. For the first $\frac{n}{q} - 1$ clock cycles, the $GF(2^{128})$ multiplications by H^q are performed. This is achieved by a simple control unit selecting H^q. Then, for the next $\log_2(q)$ clock cycles, the other exponentiations are used. These include the powers of the hash subkey in the form of H^{2^j} and a number of field elements $1 = (0, 0, \ldots, 1) \in GF(2^{128})$ for bypassing the

TABLE 3
Performance Analysis and Comparison of $GHASH_H$ within the GCM for n Blocks and q Parallel Structures

Approach	Latency	Throughput
Sequential [27], [28], [29]	n	$\frac{128}{(T_{mul}+T_X)n}$
[31], [32]	$\frac{n}{q}+q-1$	$\frac{128}{(T_{mul}+T_X)(\frac{n}{q}+q-1)}$
Proposed	$\frac{n}{q}+\log_2(q)$	$\frac{128}{(T_{mul}+T_X)(\frac{n}{q}+\log_2(q))}$

$GF(2^{128})$ multiplication operations. We note that if n is not a multiple of q, one needs to add $q - \mathrm{mod}\,(n,q)$ blocks containing $0 = (0,0,\dots,0) \in GF(2^{128})$ to the beginning of the n blocks to make the total blocks processed multiple of q. Performing this, the hash computation can be done normally based on the presented procedure. Finally, in one clock cycle, the result becomes $\sum_{j=1}^{n} X_j H^{n-j+1}$. As seen in Fig. 3, q adder-multipliers are required and multiplexers are also utilized to select different exponentiations.

To illustrate the proposed scheme, we use the case with $n = 16$ and $q = 8$. In the first clock cycle ($j = 1$), the outputs of all the multiplexers in Fig. 3 are H^8 for this case. Then, according to the following, the outputs of the multiplexers in the other cycles can be found.

$$\begin{aligned}
&\underbrace{\overbrace{\underbrace{(\overbrace{X_1H^8}^{j=1} \oplus X_9)H^8}_{j=2} \times 1}^{j=3} \times 1}_{j=4} \oplus \underbrace{\overbrace{\underbrace{(\overbrace{X_2H^8}^{j=1} \oplus X_{10})H^4}_{j=2} \times H^2}^{j=3} \times H}_{j=4} \oplus \\
&\cdots \oplus \underbrace{\overbrace{\underbrace{(\overbrace{X_iH^8}^{j=1} \oplus X_{i+8})H^{4a_1^{(i)}}}_{j=2} \times H^{2a_2^{(i)}}}^{j=3} \times H^{a_3^{(i)}}}_{j=4} \oplus \\
&\cdots \oplus \underbrace{\overbrace{\underbrace{(\overbrace{X_8H^8}^{j=1} \oplus X_{16})H}_{j=2} \times 1}^{j=3} \times 1}_{j=4},
\end{aligned} \tag{7}$$

where $(a_1, a_2, a_3)_2$ is the binary representation of $q - i + 1 = 9 - i$, $1 \le i \le 8$. Five cycles are required to implement (7); four cycles are shown in (7) with $j = 1$ to $j = 4$ and the last one is used for the addition of the results of the registers $R_1 - R_8$ to have the final result in R_T.

According to Fig. 3, the working frequency of the proposed scheme is obtained as $T_{mul} + T_X$ (we note that this delay is larger than that of the XOR tree). It is noted that T_{mul} is the time delay of the used multiplier and T_X is the time delay of one set of modulo-2 additions in the critical path. Furthermore, according to Algorithm 1, the number of clock cycles needed for the $GHASH_H$ function is $\frac{n}{q} + \log_2(q)$. Latency and throughput of the proposed scheme are compared with the ones presented in [27], [28], [29], [31], and [32] in Table 3. As seen in this table, the sequential approach has the least throughput which leads to low-performance hardware implementations. The throughput of the proposed scheme, i.e., $\frac{128}{(T_{mul}+T_X)(\frac{n}{q}+\log_2(q))}$, is higher than that of the scheme in [31] and [32], i.e., $\frac{128}{(T_{mul}+T_X)(\frac{n}{q}+q-1)}$, especially for high values of parallel structures, i.e., high values of q. For example, for the case presented in (7), the proposed architectures of this paper need $\frac{n}{q} + \log_2(q) = 2 + 3 = 5$ clock cycles to obtain the result. This can be compared with the linear relation of the scheme in [31] and [32] with q, leading to $\frac{n}{q} + q - 1 = 2 + 8 - 1 = 9$ clock cycles needed. The complete comparison in terms of hardware and timing complexities of the proposed architectures with the previous ones is presented in Section 5 using ASIC syntheses.

4.2 High-Speed Structures for Hash Subkey Powers

In the following, using squaring operations, we present three methods for implementing the hash subkey exponentiations. Using a complexity reduction algorithm, we also derive their hardware-optimized architectures.

According to [5], it is less likely that the GCM is invoked with the same key on distinct sets of input data. Thus, a new hash subkey and its powers need to be obtained in each invocation. It is known that the squaring operation in binary extension fields leads to a linear structure, see, for example, [44]. In other words, implementing squaring in hardware is less costly than $GF(2^{128})$ multiplications. The squaring of a field element over $GF(2^{128})$ in the GCM uses the irreducible polynomial $P(x) = x^{128} + x^7 + x^2 + x + 1$. Utilizing $P(x)$, we have obtained the formulations for the squaring after performing modular reduction. It is noted that MATLAB [41] has been utilized to verify the formulations used for squaring. For the GCM, the critical path delay of squaring is obtained as $3T_X$, where T_X is the XOR gate delay. Moreover, it requires 202 XOR gates.

To implement H^{2^j}, $2 \le j \le \lfloor \log_2(q) \rfloor$, one can cascade j squaring architectures or use a feedback for deriving them. We refrain using the feedback structure because of its low throughput and high latency. According to the hardware and timing complexities of squaring derived in this section, for H^{2^j}, the cascade structure yields to the hardware and timing complexities of $202j$ XOR gates and $3jT_X$, respectively. This leads to low-speed implementations which are not desirable in applications requiring high performance. It is possible to reduce the delay of the implementations of these exponentiations for the high-performance hardware implementations. To achieve this, we do not cascade the squaring implementations. Instead, we find the squaring exponentiations separately so that their derivations become in parallel. This reduces the critical path delay of the realizations. We present the following lemma for obtaining the exponentiations of the hash subkey within the GCM:

Lemma 2. *The squaring exponentiations of the hash subkey, i.e., H^{2^j}, $2 \le j \le \lfloor \log_2(q) \rfloor$, are obtained using the following:*

$$H^{2^j} \mod P(x) = d + \sum_{i=1}^{2^j-1} \tilde{e}_i, \tag{8}$$

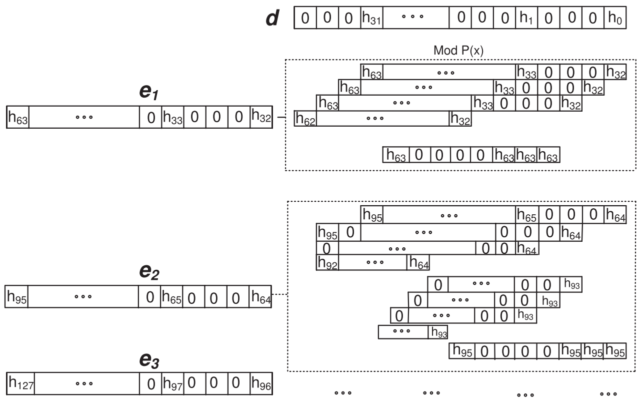


Fig. 4. The derivation of H^4 of the GCM hash subkey.

where d and $\tilde{e}_i$, $1 \le i \le 2^j - 1$, are field elements in $GF(2^{128})$ defined as follows:

$$d = \sum_{s=0}^{\frac{128}{2^j}-1} h_s x^{2^j \times s}, \quad \tilde{e}_i = \left(\sum_{s=\frac{128i}{2^j}}^{\frac{128(i+1)}{2^j}-1} h_s x^{2^j \times s} \right) \bmod P(x).$$

Proof. Let $H = \sum_{s=0}^{127} h_s x^s \in GF(2^{128})$ be the hash subkey of the *GHASH* function. Then, we have

$$\begin{aligned} H^{2^j} &= \left(\sum_{s=0}^{127} h_s x^{2^j \times s} \right) \bmod P(x) \\ &= \sum_{s=0}^{\frac{128}{2^j}-1} h_s x^{2^j \times s} + \left(\sum_{s=\frac{128}{2^j}}^{127} h_s x^{2^j \times s} \bmod P(x) \right) \\ &= d + \left(\sum_{i=1}^{2^j-1} \sum_{s=\frac{128i}{2^j}}^{\frac{128(i+1)}{2^j}-1} h_s x^{2^j \times s} \right) \bmod P(x) = d + \sum_{i=1}^{2^j-1} \tilde{e}_i \end{aligned}$$

and the proof is complete. □

For clarifying the method, we present the structure for deriving H^4 in Fig. 4. We obtain the polynomials d and e_1-e_3 in (8) as $d = h_{31}x^{124} + h_{30}x^{120} + \cdots + h_0, e_1 = h_{63}x^{252} + h_{62}x^{248} + \cdots + h_{32}x^{128}$, $e_2 = h_{95}x^{380} + h_{94}x^{376} + \cdots + h_{64}x^{256}$, and $e_3 = h_{127}x^{508} + h_{126}x^{504} + \cdots + h_{96}x^{384}$. As seen in this figure, the coefficients of d are added with the reduced coefficients of e_1-e_3 using $P(x)$.

The complexity reduction techniques use different methods for decreasing the number of gates needed in the implementations, see, for example, the ones in [45] and [46]. Because it is not guaranteed that the delay of the method in [45] is maintained, we have implemented the complexity reduction algorithm presented in [46] using a C code. In our program, the procedure suggested in [46] (to find the shared XOR terms) has been utilized for the case study of $q = 8$, which requires implementing H^2, H^4, and H^8. It is noted that through the employed technique, we reach low hardware complexities without changing the critical path delays.

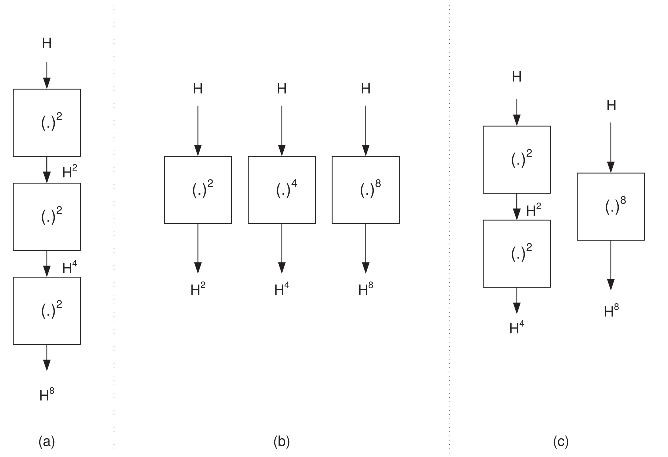


Fig. 5. (a) Cascade, (b) parallel, and (c) hybrid realization methods for the hash subkey exponentiations.

We have performed three experiments for implementing H^2, H^4, and H^8. These are shown in Fig. 5. As seen in Fig. 5a, in the cascade method, three identical squaring architectures are used consecutively. This method has the lowest hardware complexity and the highest timing complexity. In Fig. 5b, the parallel method of implementation of the hash subkey exponentiations is utilized. Compared to the other methods, this method has the lowest critical path delay while its hardware complexity is the highest. On the other hand, in the hybrid method which is shown in Fig. 5c, a compromise between hardware and timing complexities is achieved.

The timing and hardware complexities of these methods and the results of the complexity reduction technique utilized for them are depicted in Table 4. In this table, for three methods presented in Fig. 5, the hardware complexities before and after complexity reduction are derived. The timing complexity is remained unchanged after applying the complexity reductions. As seen in Table 4 in bold face, the least hardware complexity is achieved for the cascade method after the complexity reduction, i.e., 594 XOR gates. However, the timing complexity of this method is the highest among the three methods as depicted in this table. On the other hand, the timing complexity of the parallel method is the lowest, i.e., $5T_X$. As shown in Table 4, this is at the expense of higher hardware complexity which is 1,099 XOR gates after about 45 percent complexity reduction.

TABLE 4
Complexities of the Realizations of the Hash Subkey Exponentiations for $q = 8$ Parallel Architectures for $GHASH_H$

Method	Hardware Complexity	Hardware Complexity after complexity reduction	Complexity reduction (%)	Timing Complexity
Cascade (Fig. 5a)	606 XORs	**594 XORs**	$\approx 2\%$	$9T_X$
Parallel (Fig. 5b)	1986 XORs	1099 XORs	$\approx \mathbf{45}\%$	$\mathbf{5T_X}$
Hybrid (Fig. 5c)	1627 XORs	1062 XORs	$\approx 35\%$	$6T_X$

TABLE 5
Hardware and Timing Complexities Analysis of the Utilized Bit-Parallel Multipliers for the GCM

Multiplier		GE[a]	Delay	Efficiency[b]
Complexity	Type			$(10^3 \times \frac{Throughput}{GE})$
Quad. [52]	-	56,957	$T_A + 10T_X$	$0.21/T_X$
Sub-quad. [53]	KO_1	44,338	$T_A + 12T_X$	$0.23/T_X$
	KO_2	34,660	$T_A + 14T_X$	$0.25/T_X$
	KO_3	28,195	$T_A + 16T_X$	$0.27/T_X$
	KO_4	24,517	$T_A + 18T_X$	$0.28/T_X$
	KO_5	23,443	$T_A + 20T_X$	$0.27/T_X$
	KO_6	24,961	$T_A + 21T_X$	$0.24/T_X$

a. Gate equivalent in terms of two-input NAND. b. Considering $T_X = 1.99T_A$ according to the utilized technology.

4.3 $GF(2^{128})$ Multipliers for the GCM

Different types of $GF(2^{128})$ multipliers are utilized in the literature for implementing the $GF(2^{128})$ multiplications in (1). In [27], [31], and [32], the multiplications have been performed using bit-parallel, digit-serial, and hybrid multipliers in composite fields. Furthermore, in [28] and [47], the efficiency of different multipliers, including the subquadratic ones, is compared. Moreover, in [48], a high-speed AES-GCM core has been presented. It is noted that the considered $GF(2^{128})$ multipliers in these works include the Mastrovito multiplier [49] with quadratic space complexity, the Karatsuba-Ofman multiplier [50], and the $GF(2^{128})$ multiplier in [51].

We have considered the bit-parallel $GF(2^{128})$ multiplier presented in [52] which has quadratic hardware complexity. It is noted that this $GF(2^{128})$ multiplier has lower timing complexity compared to the subquadratic hardware complexity $GF(2^{128})$ multipliers. However, we note that according to the latency of the proposed architectures, i.e., $\frac{n}{q} + \log_2(q)$, increasing the number of parallel structures (q) results in having higher throughputs. On the other hand, having higher values for q increases the hardware complexities of $GHASH_H$. Therefore, for reducing the hardware complexity, using subquadratic hardware complexity $GF(2^{128})$ multipliers is beneficial when high values of q are utilized.

For reducing the hardware complexity of the AES-GCM, we have also used the efficient realization of the Karatsuba-Ofman multiplier presented in [53] as the subquadratic hardware complexity $GF(2^{128})$ multiplier. It is noted that the gate count of different steps for one Karatsuba-Ofman multiplier has been presented in [53]. Based on our technology hardware and timing specifications, we have presented the performance of the $GF(2^{128})$ multipliers in Table 5. As shown in this table, six different steps for the Karatsuba-Ofman multipliers are considered. We denote these realizations by KO_1 (for the case that only one step is performed) to KO_6 (for which the 128-bit $GF(2^{128})$ multiplier is broken all the way to 2-bit multiplications using Karatsuba-Ofman method). Applying the Karatsuba-Ofman method recursively to obtain KO_i, $2 \leq i \leq 6$ for the GCM would result in low-area implementations with higher timing complexities. As seen from this table, although the subquadratic multiplier KO_5 is the most compact implementation, the subquadratic multiplier KO_4 reaches the best efficiency. In the next section, we present the synthesis results of these subquadratic multipliers for our proposed architectures. We also compare the power consumptions and the efficiencies of different methods for realizing these multipliers.

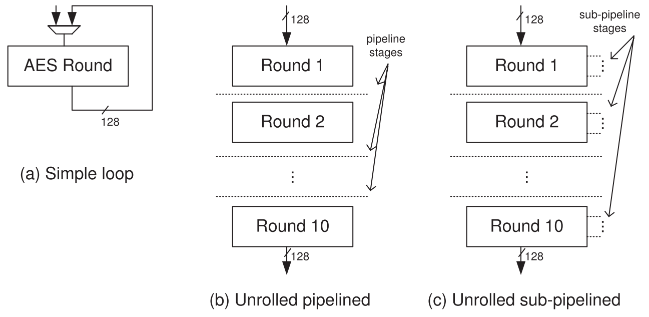


Fig. 6. The AES-128 structure for (a) simple loop, (b) unrolled pipelined, and (c) unrolled subpipelined architectures (MixColumns is bypassed in the last round).

5 AES-GCM Performance Comparisons

In this section, first, different AES architectures are presented and then we present and compare the ASIC synthesis results of the proposed and the previously presented architectures for the AES-GCM function.

We have presented different AES-128 architectures in Fig. 6. As seen in the AES simple loop structure (Fig. 6a), the AES rounds are executed serially (in the last round, MixColumns is bypassed). This architecture is the most compact AES architecture and has been used in the literature, see, for instance, [14]. However, it suffers from low throughput. In Fig. 6b, the AES unrolled pipelined structure is shown in which the pipeline stages are shown by dotted lines (see, for instance, [40]). As seen in this figure, 10 AES rounds are duplicated, with the last round without the MixColumns transformation. Although this architecture needs 10 AES rounds to be implemented, it allows the designers to use pipelining and hence process multiple inputs sequentially for achieving high throughput. For further increasing the throughput, subpipelining of the AES transformations can be used as depicted in Fig. 6c.

Subpipelining is useful in increasing the frequency of the AES at the expense of more area used for the pipeline registers; however, it increases the latency. For instance, the latency of a three-stage subpipelined AES is three times more than that of the unrolled pipelined. We also note that if the critical path delay is determined by the multipliers in the GCM architecture, subpipelining of the AES transformations cannot increase the frequency. Although both pipelined and subpipelined AES architectures can be utilized, in this paper, for the syntheses and comparisons, we use pipelined AES architecture presented in Fig. 6b. Moreover, for analyzing the effect of subpipelining, we have used subpipelined AES for two AES-GCM architectures. The details of our implementations are presented in this section.

According to Table 6, we use the most efficient S-box presented in Table 1, i.e., the one using polynomial basis based on (2), to reach the AES-GCM with the highest performance. The AES-128 encryption is considered as the

TABLE 6
The Proposed Architecture for the AES-GCM

AES (Unrolled pipelined)	GCM (Proposed using Algorithm 1)	
	Exponents	Multiplier
PB S-box ($\Phi = \{11\}_2$, $\nu = \{1010\}_2$) optimized using (2)	Complexity-reduced parallel method (Table 4 and Fig. 5b)	Quad. and six sub-quad. (KO_1 -KO_6) in Table 5

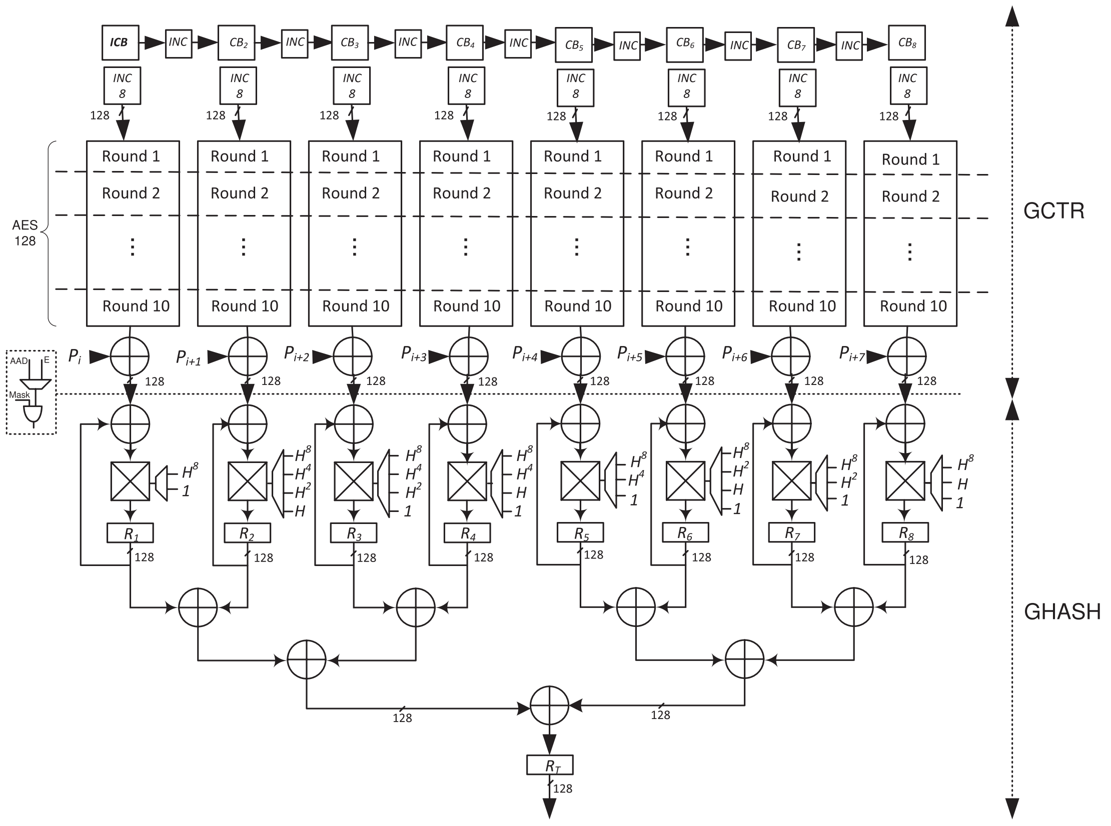


Fig. 7. The proposed AES-GCM high-performance architecture for $q = 8$ $(mod(n, q) = 0)$.

block cipher for the GCM (refer to Fig. 2) and as indicated in Table 6, the 10 rounds of the AES-128 are unrolled and pipelined. Moreover, as seen in Table 6, we use the proposed Algorithm 1 for the GCM and utilize the parallel method in Fig. 5b for hash subkey exponentiations (hardware optimized through complexity reduction methods in the previous section). Finally, as seen in this table, we use both quadratic and subquadratic multipliers presented in Table 5.

Fig. 7 presents the proposed architecture for the AES-GCM for $q = 8$ parallel structures. The AES-128 pipeline registers are shown by dashed lines in Fig. 7. As seen in this figure, 10 clock cycles are needed for obtaining the ciphertext. After these first 10 clock cycles, the results are obtained after each clock cycle. According to Fig. 7, eight parallel AES-128 structures are implemented as part of $GCTR_K$ to provide inputs to $GHASH_H$. As seen in this figure, the function $GCTR_K$ performs the AES counter mode with the *Initial Counter Block* and its one-increments (CB_i). Moreover, $q = 8$ increments (using *INC 8* module) and the plaintext blocks (P_i) are used as the inputs. It is assumed that the data are encrypted and the *IV* in the GCM is 96 bits which is recommended for high-throughput implementations [5].

The architecture shown in Fig. 7 assumes that the number of blocks n is a multiple of the number of parallel structures q and there is no additional authenticated data. In case that n is not a multiple of q, one can append $q - \bmod(n, q)$ zero blocks at the beginning of the blocks for which hash is computed. This is done by adding a masking gate along the dotted line as shown in Fig. 7. Moreover, in this case, the counter blocks and accordingly P_is in Fig. 7 start from the $q - \bmod(n, q) + 1$ column, i.e., the first actual input block. This is similar to the method used in [32]. We also note that in case AAD is present, additional multiplexers are placed at the output of the $GCTR$ block in Fig. 7 along the dotted line so that instead of encrypted data, the AAD is fed to the architecture. Such a scheme which is similar to the one presented in [32], also needs more flexibility in the counter blocks and accordingly P_is so that only when the AAD is done, the counter blocks provide the encrypted data. Finally, in Fig. 7 and as the last processed block, the output of the $GCTR$ block in the rightmost column is masked and $L_{A,C}$ (number for n) is fed (using an extra multiplexer which is not shown in Fig. 7 for the sake of brevity). $AES_K(J_0)$ and $H = AES_K(0)$ (see Fig. 2) can be also obtained or precomputed in Fig. 7 (similar to [32]), the details of which are not presented in Fig. 7.

The results of our syntheses for the AES-GCM using the STM 65-nm CMOS technology [35] are presented in Table 7. The architectures have been coded in VHDL as the design entry to the Synopsys Design Vision [34]. The proposed architectures in this paper and the ones in [27], [28], [29], [31], and [32] have been synthesized. The syntheses are based on the case for $q = 8$ parallel addition-multiplications using the bit-parallel $GF(2^{128})$ multiplier presented in [52] which has quadratic hardware complexity. For achieving low hardware complexity for the AES-GCM, we have also synthesized six different steps for the Karatsuba-Ofman multipliers. As seen in Table 7, areas, power consumptions, and maximum working frequencies are tabulated. From the

TABLE 7
ASIC Synthesis Comparisons of the AES-GCM

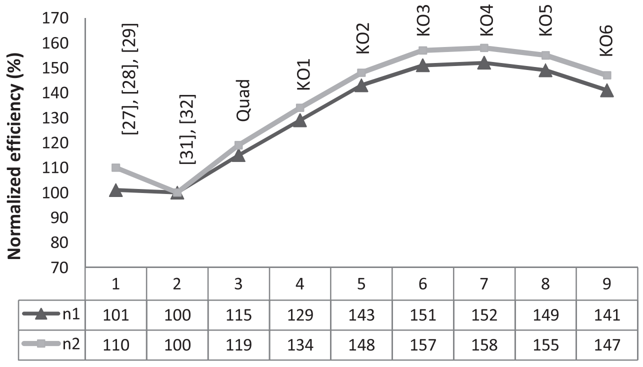
Scheme[a]	Total Area [AES][b]		Power	Freq.	Thro.	Eff.
	(mm^2)	K-GE[c]	(mW)	(MHz)	(Gbps)	($\frac{Gbps}{mm^2}$)
[27], [28], [29]	0.23 [0.12]	110 [57]	19.6	568	$\frac{72.7}{n}$	$\frac{316.0}{n}$
[31], [32][d]	1.86 [1.02]	894 [490]	144.3	568	$\frac{72.7}{\frac{n}{8}+7}$	$\frac{39.1}{\frac{n}{8}+7}$
Proposed (quad.)[e]	1.82 [0.92]	875 [442]	142.5	641	$\frac{82.0}{\frac{n}{8}+3}$	$\frac{45.1}{\frac{n}{8}+3}$
Proposed (KO_1)[e]	1.62 [0.92]	779 [442]	124.6	641	$\frac{82.0}{\frac{n}{8}+3}$	$\frac{50.6}{\frac{n}{8}+3}$
Proposed (KO_2)[e]	1.46 [0.92]	702 [442]	113.0	641	$\frac{82.0}{\frac{n}{8}+3}$	$\frac{56.2}{\frac{n}{8}+3}$
Proposed (KO_3)[e]	1.34 [0.92]	644 [442]	104.8	621	$\frac{79.4}{\frac{n}{8}+3}$	$\frac{59.3}{\frac{n}{8}+3}$
Proposed (KO_4)[e]	1.31 [0.92]	630 [442]	101.2	613	$\frac{78.4}{\frac{n}{8}+3}$	$\frac{59.8}{\frac{n}{8}+3}$
Proposed (KO_5)[e]	1.30 [0.92]	625 [442]	102.0	595	$\frac{76.1}{\frac{n}{8}+3}$	$\frac{58.5}{\frac{n}{8}+3}$
Proposed (KO_6)[e]	1.33 [0.92]	639 [442]	105.2	578	$\frac{73.9}{\frac{n}{8}+3}$	$\frac{55.6}{\frac{n}{8}+3}$

a. For the case of $q = 8$ parallel structures. b. The area of the AES is shown inside brackets. c. 10^3 gate equivalent in terms of two-input NAND. d. The least complexity scheme in [31] and [32] has been synthesized. e. The quad./subquad. multipliers used in the AES-GCM.

discussions in Section 4, for n input blocks and q parallel structures, the latency for the architecture in [27], [28], and [29] is n, for the one in [31] and [32] is $\frac{n}{q} + q - 1$, and for our proposed architectures is $\frac{n}{q} + \log_2(q)$. According to these, for different architectures presented in Table 7, throughputs and efficiencies are also presented.

As presented in Table 7, the sequential approach in [27], [28], and [29] has the lowest hardware complexity compared to other approaches. However, it has the least throughput leading to low-performance hardware implementations. As depicted in Table 7, lower areas and power consumptions are achieved for the subquadratic hardware complexity $GF(2^{128})$ multipliers used in our proposed architectures compared to the one in [52]. As seen in this table, the maximum working frequency is decreased as we increase the number of multiplication steps. However, this trend is not observed for the hardware complexity, i.e., it is decreased up to KO_5 as the optimum value and then rises for KO_6.

The highest throughput is achieved for the proposed architectures in this paper, i.e., $\frac{82.0}{\frac{n}{8}+3}$ Gbps using quadratic and KO_1/KO_2 subquadratic multipliers. As seen in Table 7, the highest efficiency is derived for KO_4, i.e., $\frac{59.8}{\frac{n}{8}+3}\frac{Gbps}{mm^2}$. As seen in this table, the working frequencies and throughputs for KO_1 and KO_2 are similar. We have observed that this is because for these two multipliers, the critical path delay is dominated by the AES rounds and not the subquadratic multiplier. Inner round pipelining can be performed to increase the working frequencies of the implementations. Nevertheless, this subpipelining increases the area and latency of the AES-GCM architectures. We have performed experiments by subpipelining the AES rounds for the architectures using KO_1 and KO_2 multipliers. This is achieved by adding one pipeline stage after ShiftRows and right before MixColumns. The results of our experiments show no major difference in



	1	2	3	4	5	6	7	8	9
n1	101	100	115	129	143	151	152	149	141
n2	110	100	119	134	148	157	158	155	147

Fig. 8. Comparison of the efficiencies of nine different AES-GCM architectures for $n_1 = 2^{32} - 2$ and $n_2 = 2^{10}$.

the maximum working frequency of the design utilizing KO_2 multiplier and increase in its hardware complexity. However, for the architectures using KO_1 multipliers, the working frequency of 689 MHz with the increased area of $1.70\ mm^2$ is achieved. Therefore, for this architecture which uses KO_1 multipliers, the best speed is obtained compared to the results in Table 7. However, its efficiency is obtained as $\frac{51.9}{\frac{n}{8}+3}\frac{Gbps}{mm^2}$ which is less than that of the architectures with KO_4 multipliers (see Table 7).

For comparing the efficiencies of the schemes presented in Table 7, we have presented Fig. 8. Based on the derived values for efficiencies in the last column of Table 7, two different graphs for two values of n are presented in Fig. 8. We consider two different values of n, i.e., $n_1 = 2^{32} - 2$ (the largest encrypted message size allowed) and $n_2 = 2^{10}$. It is noted that considering the normalized efficiency (percent) of the scheme in [31] and [32] as 100, the relative efficiencies for different architectures are presented in this figure. As seen in Fig. 8, for $n_1 = 2^{32} - 2$ and $n_2 = 2^{10}, KO_4$ has the highest efficiencies (51 and 44 percent more than the sequential method, respectively).

6 Conclusions

In this paper, we have obtained optimized building blocks for the AES-GCM to propose efficient and high-performance architectures. For the AES, through logic-gate minimizations for the inversion in $GF(2^4)$, the areas of the S-boxes have been reduced. We have also evaluated and compared the performance of different S-boxes using an ASIC 65-nm CMOS technology. Furthermore, through exhaustive searches for the input patterns, we have performed simulation-based power derivations for different S-boxes to reach more accurate results compared to the statistical methods. We have also proposed high-performance and efficient architectures for the GCM. For the case study of $q = 8$ parallel structures in $GHASH_H$, we have performed a hardware complexity reduction technique for the hash subkey exponentiations, having their timing complexities intact. The ASIC comparison results show that better efficiencies are achieved for the proposed architectures. Based on the available resources and performance goals to achieve, one can choose the proposed AES-GCM architectures to fulfill the constraints of different applications.

ACKNOWLEDGMENTS

The authors would like to thank the reviewers for their comments. This work has been supported in part by an NSERC Discovery grant awarded to Arash Reyhani-Masoleh. They would like to thank Canadian Microelectronics Corporation (CMC) Microsystems for providing the required infrastructure and CAD tools that have been used in this work. They would also like to thank Christopher Kennedy who assisted in their software implementation of the complexity reduction algorithm for hash subkey exponentiations.

REFERENCES

[1] Nat'l Inst. of Standards and Technologies "Announcing the Advanced Encryption Standard (AES)," Fed. Information Processing Standards Publication, no. 197, Nov. 2001.
[2] Wi-Fi, http://standards.ieee.org/getieee802/download/802.11-2007.pdf, 2011.
[3] WiMAX, http://standards.ieee.org/getieee802/download/802.16e-2005.pdf, 2011.
[4] S. Trimberger, "Security in SRAM FPGAs," *IEEE Design and Test of Computers,* vol. 24, no. 6, p. 581, Nov./Dec. 2007.
[5] M. Dworkin, "Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC," *NIST SP,* 800-38D, 2007.
[6] IEEE Standard for Local and Metropolitan Area Networks, Media Access Control (MAC) Security, 2006.
[7] Fibre Channel Security Protocols (FC-SP), http://www.t10.org/ftp/t11/document.06/06-157v0.pdf. 2006.
[8] Algotronics Ltd.: GCM Extension for AES G3 Core, 2007.
[9] Helion Technology: AES-GCM Cores, 2007.
[10] Elliptic Semiconductor Inc.: CLP-15: Ultra-High Throughput AES-GCM Core-40 Gbps, 2008.
[11] E. Käsper and P. Schwabe, "Faster and Timing-Attack Resistant AES-GCM," *Proc. Int'l Workshop Cryptographic Hardware and Embedded Systems (CHES '09),* pp. 1-17, 2009.
[12] K. Jankowski and P. Laurent, "Packed AES-GCM Algorithm Suitable for AES/PCLMULQDQ Instructions," *IEEE Trans. Computers,* vol. 60, no. 1, pp. 135-138, Jan. 2011.
[13] S. Morioka and A. Satoh, "An Optimized S-Box Circuit Architecture for Low Power AES Design," *Proc. Int'l Workshop Cryptographic Hardware and Embedded Systems (CHES '02),* pp. 172-186, Aug. 2002.
[14] A. Satoh, S. Morioka, K. Takano, and S. Munetoh, "A Compact Rijndael Hardware Architecture with S-Box Optimization," *Proc. Int'l Conf. Theory and Application of Cryptology and Information Security: Advances in Cryptology (ASIACRYPT '01),* pp. 239-254, Dec. 2001.
[15] J. Wolkerstorfer, E. Oswald, and M. Lamberger, "An ASIC Implementation of the AES SBoxes," *Proc. Cryptographers Track at the RSA Conf. (CT-RSA '02),* pp. 67-78, Jan. 2002.
[16] X. Zhang and K.K. Parhi, "High-Speed VLSI Architectures for the AES Algorithm," *IEEE Trans. Very Large Scale Integration (VLSI) Systems,* vol. 12, no. 9, pp. 957-967, Sept. 2004.
[17] T. Good and M. Benaissa, "692-nW Advanced Encryption Standard (AES) on a $0.13-\mu\mathrm{m}$ CMOS," *IEEE Trans. Very Large Scale Integration (VLSI) Systems,* vol. 18, no. 12, pp. 1753-1757, Dec. 2010.
[18] M. Mozaffari-Kermani and A. Reyhani-Masoleh, "A Low-Cost S-box for the Advanced Encryption Standard Using Normal Basis," *Proc. IEEE Int'l Conf. Electro/Information Technology (EIT '09),* pp. 52-55, 2009.
[19] S. Tillich, M. Feldhofer, T. Popp, and J. Großschädl, "Area, Delay, and Power Characteristics of Standard-Cell Implementations of the AES S-Box," *J. Signal Processing Systems,* vol. 50, pp. 251-261, 2008.
[20] D. Canright, "A Very Compact S-Box for AES," *Proc. Int'l Workshop Cryptographic Hardware and Embedded Systems (CHES '05),* pp. 441-455, Sept. 2005.
[21] N. Mentens, L. Batina, B. Preneel, and I. Verbauwhede, "A Systematic Evaluation of Compact Hardware Implementations for the Rijndael S-Box," *Proc. Cryptographers Track at the RSA Conf. (CT-RSA '05),* pp. 323-333, 2005.
[22] X. Zhang and K.K. Parhi, "On the Optimum Constructions of Composite Field for the AES Algorithm," *IEEE Trans. Circuits and Systems II: Express Briefs,* vol. 53, no. 10, pp. 1153-1157, Oct. 2006.
[23] J. Boyar and R. Peralta, "A New Combinational Logic Minimization Technique with Applications to Cryptology," *Proc. Int'l Symp. Experimental Algorithms (SEA '10),* pp. 178-189, 2010.
[24] S. Nikova, V. Rijmen, and M. Schläffer, "Using Normal Bases for Compact Hardware Implementations of the AES S-Box," *Proc. Int'l Conf. Security and Cryptography for Networks (SCN '08),* pp. 236-245, 2008.
[25] Y. Nogami, K. Nekado, T. Toyota, N. Hongo, and Y. Morikawa, "Mixed Bases for Efficienct Inversion in $F_{((2^2)^2)^2}$ and Conversion Matrices of SubBytes of AES," *Proc. Int'l Workshop Cryptographic Hardware and Embedded Systems (CHES '10),* pp. 234-247, Aug. 2010.
[26] D. Canright and D.A. Osvik, "A More Compact AES," *Selected Areas in Cryptography,* pp. 157-169, Springer-Verlag, 2009.
[27] S. Lemsitzer, J. Wolkerstorfer, N. Felbert, and M. Braendli, "Multi-Gigabit GCM-AES Architecture Optimized for FPGAs," *Proc. Int'l Workshop Cryptographic Hardware and Embedded Systems (CHES '07),* pp. 227-238, 2007.
[28] P. Patel, "Parallel Multiplier Designs for the Galois/Counter Mode of Operation," Master of Applied Science thesis, The Univ. of Waterloo, 2008.
[29] B. Yang, S. Mishra, and R. Karri, "High Speed Architecture for Galois/Counter Mode of Operation (GCM)," *Cryptology ePrint Archive: Report 2005/146* June 2005.
[30] D.A. McGrew and J. Viega, "The Galois/Counter Mode of Operation (GCM)," *NIST Modes Operation Symmetric Key Block Ciphers,* http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/gcm/gcm-revised-spec.pdf, 2005.
[31] A. Satoh, "High-Speed Parallel Hardware Architecture for Galois Counter Mode," *Proc. Int'l Symp. Circuits and Systems (ISCAS),* pp. 1863-1866, 2007.
[32] A. Satoh, T. Sugawara, and T. Aoki, "High-Performance Hardware Architectures for Galois Counter Mode," *IEEE Trans. Computers,* vol. 58, no. 7, pp. 917-930, July 2009.
[33] N. Meloni, C. Nègre, and M.A. Hasan, "High Performance GHASH Function for Long Messages," *Proc. Int'l Conf. Applied Cryptography and Network Security (ACNS '10),* pp. 154-167, 2010.
[34] Synopsys, http://www.synopsys.com/, 2011.
[35] STMicroelectronics, http://www.st.com/, 2011.
[36] ModelSim, http://www.model.com/, 2011.
[37] M. McLoone and J.V. McCanny, "High Performance Single-Chip FPGA Rijndael Algorithm Implementations," *Proc. Int'l Workshop Cryptographic Hardware and Embedded Systems (CHES '01),* pp. 65-76, 2001.
[38] F.X. Standaert, G. Rouvroy, J.J. Quisquater, and J.D. Legat, "Efficient Implementation of Rijndael Encryption in Reconfigurable Hardware: Improvements and Design Tradeoffs," *Proc. Int'l Workshop Cryptographic Hardware and Embedded Systems (CHES '03),* pp. 334-350, Sept. 2003.
[39] P. Bulens, F.-X. Standaert, J.-J. Quisquater, P. Pellegrin, and G. Rouvroy, "Implementation of the AES-128 on Virtex-5 FPGAs," *Proc. Cryptology in Africa First Int'l Conf. Progress in Cryptology (AFRICACRYPT '08),* pp. 16-26, 2008.
[40] A. Hodjat and I. Verbauwhede, "Area-Throughput Trade-Offs for Fully Pipelined 30 to 70 Gbits/s AES Processors," *IEEE Trans. Computers,* vol. 55, no. 4, pp. 366-372, Apr. 2006.
[41] Mathworks, http://www.mathworks.com/, 2011.
[42] S.-Y. Lin and C.-T. Huang, "A High-Throughput Low-Power AES Cipher for Network Applications," *Proc. Asia and South Pacific Design Automation Conf. (ASP-DAC '07),* pp. 595-600, 2007.
[43] D.E. Knuth, *The Art of Computer Programming: Semi-Numerical Algorithms,* vol. 2, pp. 441-466. Addison-Wesley, 1981.
[44] R. Lidl and H. Niederreiter, *Introduction to Finite Fields and Their Applications.* Cambridge Univ. Press, 1994.
[45] O. Gustafsson and M. Olofsson, "Complexity Reduction of Constant Matrix Computations over the Binary Field," *Proc. Int'l Workshop Arithmetic of Finite Fields (WAIFI '07),* pp. 103-115, 2007.
[46] H. Yi, J. Song, S. Park, and C. Park, "Parallel CRC Logic Optimization Algorithm for High Speed Communication Systems," *Proc. Int'l Conf. Comm. Systems (ICCS '06),* pp. 1-5, 2006.
[47] G. Zhou, H. Michalik, and L. Hinsenkamp, "Improving Throughput of AES-GCM with Pipelined Karatsuba Multipliers on FPGAs," *Proc. Int'l Workshop Reconfigurable Computing: Architectures, Tools and Applications (ARC '09),* pp. 193-203, 2009.

[48] J. Lázaro, A. Astarloa, U. Bidarte, J. Jiménez, and A. Zuloaga, "AES-Galois Counter Mode Encryption/Decryption FPGA Core for Industrial and Residential Gigabit Ethernet Communications," *Proc. Int'l Workshop Reconfigurable Computing: Architectures, Tools and Applications (ARC '09)*, pp. 312-317, 2009.

[49] E.D. Mastrovito, "VLSI Architectures for Computation in Galois Fields," PhD thesis, Linköping Univ., 1991.

[50] A. Karatsuba and Y. Ofman, "Multiplication of Multidigit Numbers on Automata," *Soviet Physics Doklady*, vol. 7, pp. 595-596, 1963.

[51] H. Fan and M.A. Hasan, "A New Approach to Subquadratic Space Complexity Parallel Multipliers for Extended Binary Fields," *IEEE Trans. Computers*, vol. 56, no. 2, pp. 224-233, Feb. 2007.

[52] A. Reyhani-Masoleh and M.A. Hasan, "Low Complexity Bit Parallel Architectures for Polynomial Basis Multiplication over $GF(2^m)$," *IEEE Trans. Computers*, vol. 53, no. 8, pp. 945-959, Aug. 2004.

[53] G. Zhou, H. Michalik, and L. Hinsenkamp, "Complexity Analysis and Efficient Implementations of Bit Parallel Finite Field Multipliers Based on Karatsuba-Ofman Algorithm on FPGAs," *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, vol. 18, no. 7, pp. 1057-1066, July 2010.

Mehran Mozaffari-Kermani received the BSc degree in electrical and computer engineering from the University of Tehran in 2005, and the MESc and PhD degrees from the Department of Electrical and Computer Engineering at The University of Western Ontario in 2007 and 2011, respectively. After the completion of his PhD, he worked at the Advanced Micro Devices (AMD) as a senior ASIC/layout designer, integrating sophisticated security/cryptographic capabilities into a single chip. Dr. Mozaffari-Kermani was awarded a Natural Sciences and Engineering Research Council of Canada (NSERC) postdoctoral fellowship in 2011. Currently, he is an NSERC postdoctoral research fellow at the Electrical Engineering Department of Princeton University. His research interests include developing security/privacy measures for emerging technologies, cryptographic systems, fault diagnosis and tolerance in cryptographic hardware and embedded systems, VLSI reliability, and low-power secure and efficient FPGA and ASIC designs. He is a member of the IEEE.

Arash Reyhani-Masoleh received the BSc degree in electrical and electronic engineering from Iran University of Science and Technology in 1989, the MSc degree in electrical and electronic engineering from the University of Tehran in 1991, both with the first rank, and the PhD degree in electrical and computer engineering from the University of Waterloo in 2001. From 1991 to 1997, he was with the Department of Electrical Engineering, Iran University of Science and Technology. From June 2001 to September 2004, he was with the Centre for Applied Cryptographic Research, University of Waterloo, where he was awarded a Natural Sciences and Engineering Research Council of Canada (NSERC) Postdoctoral Fellowship in 2002. In October 2004, he joined the Department of Electrical and Computer Engineering, University of Western Ontario, London, Canada, where he is currently a tenured associate professor. His current research interests include algorithms and VLSI architectures for computations in finite fields, fault-tolerant computing, and error-control coding. He has been awarded a NSERC Discovery Accelerator Supplement (DAS) in 2010. Currently, he serves as an associate editor for *Integration, the VLSI Journal* (Elsevier). He is a member of the IEEE and the IEEE Computer Society.