Architecture Optimization and Performance Comparison of Nonce-Misuse-Resistant Authenticated Encryption Algorithms

Sandhya Koteshwara, *Student Member, IEEE*, Amitabh Das, *Senior Member, IEEE*, and Keshab K. Parhi, *Fellow, IEEE*

***Abstract*— This paper presents a performance comparison of new authenticated encryption (AE) algorithms which are aimed at providing better security and resource efficiency compared to existing standards. Specifically, these algorithms improve the security of existing AE standards by providing a critical property termed nonce-misuse resistance. This paper addresses algorithm to architectural mappings of several candidates from the ongoing Competition for AE: Security, Applicability, and Robustness as well as a submission from the Crypto Forum Research Group. Implementations of the architectures on both field-programmable gate arrays and application-specific integrated circuits platforms are provided and compared with the architecture of a popular standard: Advanced Encryption Standard in Galois Counter mode (AES-GCM). Optimizations that are applicable to AE, in general, and nonce-misuse-resistant architectures, in particular, are presented. A hardware–software codesign approach to optimization is also discussed. The implementations via proposed optimizations demonstrate that new AE algorithms can provide comparable performance as standard AES-GCM while enhancing security and resource utilization for specific use-case scenarios.**

***Index Terms*— Advanced Encryption Standard in Galois Counter mode (AES-GCM), AES-GCM-synthetic IV (SIV), authenticated encryption (AE), Competition for AE: Security, Applicability, and Robustness (CAESAR) competition, Deoxys, nonce-misuse resistance, pipelineable on-line encryption with authentication tag (POET), PRIMATE-APE.**

I. Introduction

With the advent of the Internet of Things (IoT) era, billions of devices will be connected to each other and to a common network. Hence, it is of utmost importance to ensure security and privacy of communication between the devices as well as between the device and the cloud server. Moreover, there is a growing trend to bring computing to the edge of the IoT rather than the cloud, termed as edge-centric computing [1]. Hence, resource efficient and strongly secure cryptographic algorithms which can ensure the security of communication and can be implemented on the hardware of the device itself have become critical. Also, algorithms that require smaller key sizes, less frequent change of keys, and better resilience are desired.

Authenticated encryption (AE) algorithms combine the process of authentication and encryption to create a single algorithm which is secure and resource efficient. It is well understood that confidentiality of data does not suffice, and it is important to ensure authentication of source as well as data integrity [2]. Hence, AE algorithms are growing in importance with the changing device scenarios and platforms. When data such as packet headers and message numbers are also included as part of the plaintext, the resulting algorithms are termed AE with associated data (AEAD) algorithms. These additional data require only authentication and are termed associated data.

The general equations of AEAD are expressed as follows:

$$\mathrm{Enc}_K(N, \mathrm{AD}, \mathrm{PT}) = (\mathrm{CT}, \mathrm{Tag}) \quad (1)$$

$$\mathrm{Dec}_K(N, \mathrm{AD}, \mathrm{CT}, \mathrm{Tag}) = (\mathrm{PT}, \perp). \quad (2)$$

In these equations, K represents the secret key, AD represents the associated data, PT refers to the plaintext, and N is a unique nonrepeating number termed *nonce*. These inputs are applied to the encryption algorithm Enc to produce the outputs: CT, which represents the encryption of the plaintext, and Tag, which are utilized for authentication purposes. The nonce is used to transmit more than one message block using the same secret key. For the decryption algorithm Dec, N, AD, CT, Tag, and K are used as inputs to retrieve PT. In addition, Tag is verified and a valid/invalid message is generated (represented as $\perp$ in the equation).

Some of the existing standards for AEAD algorithms include Advanced Encryption Standard in Galois Counter mode (AES-GCM), Advanced Encryption Standard in Counter with cipher block chaining message authentication code mode, encrypt-then-authenticate-then-translate, Offset Codebook (OCB) mode, etc. We do not delve into the details of these algorithms and refer the reader to the existing literature [3]–[6]. Instead, we focus on AES-GCM which has been deployed in most practical applications such as Transport Layer Security (TLS) and Secure Socket Layer. Using AES-GCM as a baseline architecture, we present the architectures of several new algorithms which have been proposed as a part of the ongoing Competition for AE: Security, Applicability, and

Manuscript received June 28, 2018; revised November 5, 2018; accepted December 31, 201 8. *(Corresponding author: Sandhya Koteshwara.)*

S. Koteshwara was with the Department of Electrical Engineering, University of Minnesota, Minneapolis, MN 55455 USA. She is now with the IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598 USA (e-mail: kotes001@umn.edu).

A. Das was with Intel Labs, Security and Privacy Research, Intel Corporation, Hillsboro, OR 97124 USA. He is now with AMD, Austin, TX 78735 USA.

K. K. Parhi is with the Department of Electrical Engineering, University of Minnesota, Minneapolis, MN 55455 USA.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2019.2894656

Robustness (CAESAR) competition as well as from Crypto Forum Research Group (CFRG). The chosen algorithms incorporate an important property termed nonce-misuse-resistance.

The presented nonce-misuse-resistant algorithms are mapped to hardware using both field-programmable gate array (FPGA) and application-specific integrated circuit (ASIC) platforms. The implementations are then compared to the AES-GCM standard. This paper also discusses important architectural differences that arise because of the nonce-misuse resistance property and optimizations to overcome these limitations. FPGA-based implementation of two of the algorithms discussed in this paper, namely, Deoxys and AES-GCM-synthetic IV (SIV) have been presented in [7] and [8], respectively. This paper moves beyond these implementations by providing several optimizations and comparisons with other candidates such as PRIMATE-APE and pipelineable on-line encryption with authentication tag (POET). Also, a hardware–software codesign approach to obtain resource efficient implementations applicable to modern IoT like platforms is presented. A final comparison of the selected candidates and AES-GCM with respect to performance and resource utilization is presented with a recommendation of the most suitable candidates for several use-case scenarios.

The rest of this paper is divided as follows. In Section II, we present related work and discuss the motivation of this paper in detail. Section III presents the flow diagram and architecture of several algorithms which provide nonce-misuse resistance. Optimizations applicable to these new AE algorithms are presented in Section IV. In Section V, we discuss the detailed experimental setup and present results on FPGA/ASIC platforms. An analysis of the side-channel vulnerabilities of these algorithms is presented in Section VI. Finally, we conclude with a recommendation of candidates based on applications.

II. Motivation and Related Work

In this section, we discuss the shortcomings of current AEAD algorithms and motivation for this paper.

A. New AEAD Schemes

Several issues have been identified in existing AE algorithms. Some of these are highlighted below.

1) Existing algorithms are still too large in terms of area or consume too much energy. This is especially true if the AE schemes are to be implemented on a device which has minimal resources.
2) Several security properties such as nonce-misuse resistance, decryption misuse resistance, robustness against leakage of plaintext, detection of forgery attempts, and so on are desirable and missing in existing AEAD algorithms.
3) Cryptanalytical efforts have found groups of weak keys in the most widely adopted AES-GCM algorithm [9].
4) Better performance while maintaining the same level of security of existing standards or better security with the same performance are both desirable. This is especially true due to increase in number and reduction in the size of devices in modern applications.

To alleviate some of these problems, a call for novel authenticated algorithms was put forth in the form of CAESAR competition. The goal of this competition is to identify a portfolio of AE algorithms which offer advantages over AES-GCM and are suitable for widespread adoption [10]. The competition is ongoing and currently in its final round with seven finalists. The first round had 54 submissions out of which 29 candidates were selected for the second round. In the third round, 15 potential candidates were recognized. These candidates have different properties with respect to security, resource consumption, and underlying constructions. A summary for the candidates and their important properties can be found in [11] and [12]. A candidate that we discuss in this paper, Deoxys, has been selected to the final round of the competition. Note that even though some of the candidates selected for this paper have not been selected for the final round of the CAESAR competition, these are useful candidates for several applications. Since the competition considers several parameters apart from security, these candidates have been excluded from the next round. We also consider an algorithm submitted to CFRG after the CAESAR competition first round submission deadline.

Several independent implementations of the algorithms in both hardware and software can be found in the literature. To bring these implementations under the same platform, there is an ongoing effort being carried out by the SUPERCOP software benchmarking team and the ATHENa GMU hardware platform team [13]. Our work specifically focuses on in-depth analysis of nonce-misuse resistance schemes whose details and significance will be discussed next.

B. Importance of Nonce-Misuse Resistance

From the description of AEAD schemes, it is observed that nonces are critical to the security of symmetric key cryptographic algorithms. The construction of these algorithms is such that using the same key, multiple messages can be encrypted by using different nonces. Hence, the nonce must not repeat under the same key. Even though this appears to be a simple requirement, it is not easy to satisfy as has been observed in many practical applications [14].

There are several approaches that can be used to ensure nonces do not repeat while using the same key. One solution is to update the keys frequently. However, regular exchange of secret keys between two parties is not an easy task and many applications will not have the capability to do so. Specifically, with systems involving IoT devices where millions of devices are interconnected with each other, the most practical implementation would program one secret key and utilize it for the lifetime of the device. The second approach to ensure nonces do not repeat is to derive the nonces from counters. With sufficiently large counters, the nonce values will be based on each increment of the counter and will not repeat. However, if the counter is made to overflow by injecting faults or other forms of attacks, the nonces start to repeat breaking the security of the algorithm. Finally, the nonce can

TABLE I
SUMMARY OF CANDIDATE FEATURES AND COMPARISON WITH AES-GCM

Parameters	AES-GCM	Deoxys	AES-GCM-SIV	POET	Primate-APE
Nonce-misuse resistance	No	Complete	Complete	Complete	Complete
Security level	t = 64 bits q = 64 bits	t = 128 bits q = 64 bits	t = 64 bits q = 64 bits	t = 128 bits q = 64 bits	t = 128 bits q >64 bits
Parallelizable	Yes	Yes	Yes	Partial	No
Comparison to AES-GCM	-	- Authentication first - Secure block cipher	- Similar components - Authentication first	- AES for both Auth & Enc - Flexible	- Lowest resource - Slower

*t=time complexity, q=query complexity

**The detailed security analysis of AES-GCM-SIV based on nonce-repetition frequency and message block length is beyond the scope of this paper and can be referred from [15], [16].

be made unique by using random number generators. However, the random number generator should be of high quality and sufficiently large. Ensuring this requirement is met again is a challenging task with the ever-shrinking sizes of devices.

Attacks due to nonce-misuse have been demonstrated in the literature in important applications such as the TLS [14]. Thus, algorithms which can inherently provide some form of nonce-misuse resistance have become favorable and will continue to increase in importance as IoT devices become more prevalent.

C. Algorithms With Nonce-Misuse Resistance

Nonce-misuse-resistant schemes ensure that even though the nonce is repeated, only limited knowledge about the encrypted message is given away, while the complete plaintext decryption is not possible [17]. There are two types of nonce-misuse-resistant algorithms: *complete* and *partially* nonce-misuse resistant. While the complete nonce-misuse resistance property is preferred, some applications may still benefit from partial nonce-misuse resistance.

In [12], candidates which provide both complete and partial nonce-misuse resistance have been identified and initial performance comparisons from several existing studies have been presented. Based on this analysis, candidates which provide complete nonce-misuse resistance and promise good performance from initial comparisons have been selected for our study. Among the finalists of the competition, only Deoxys, OCB, and ElmD (which has now been merged with another algorithm AES-COPA) provide complete nonce-misuse resistance. Since OCB is covered by some patents and ElmD shows high area consumption from our initial analysis, these were not selected for our study. AES-GCM-SIV, which is an independent submission from CFRG, is also included in our study as it promises complete nonce-misuse resistance while reusing the blocks of standard AES-GCM. Table I lists the basic properties of these candidates and their detailed algorithm and architecture are discussed next.

III. ALGORITHM AND ARCHITECTURAL DESCRIPTIONS

Next, we briefly discuss the algorithm and architecture of AES-GCM standard. The discussions of nonce-misuse-resistant algorithms and architectures will be based on this standard.

A. AES-GCM

The AES-GCM algorithm is represented using the block diagram of Fig. 1. This algorithm is a block cipher-based

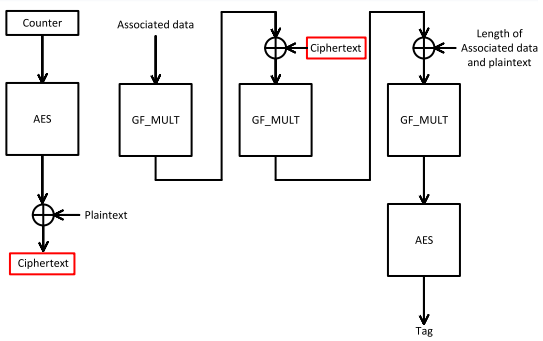

Fig. 1. Description of the algorithm of AES-GCM. Left: cipher text block generation using a counter. Right: associated data processing and processing of plaintext blocks to generate the tag.

AE scheme. This implies that a block cipher such as AES is used to perform all encryption operations. AES in the counter (CTR) mode of operation handles plaintext in a block by block manner to generate cipher text blocks. The first value of the counter is dependent on the nonce, and the counter is incremented to process subsequent message blocks. The authentication process is handled by a Galois field multiplier which performs polynomial multiplication on the associated data and the generated cipher text blocks. A final encryption results in the generation of the tag value.

Note that the counter value is encrypted using the AES block and the plaintext is just XORed with the output. This implies that the security of the algorithm is completely dependent on the nonce being different for each message. If the same nonce repeats for two different messages, a simple XOR operation can differentiate between them. Thus, information is leaked, and the system is not nonce-misuse resistant.

A serial implementation aimed at low area and power consumption is used to design the architecture of AES-GCM. This means that the architecture makes use of a single AES block and Galois field multiplier block. The blocks of the data path of AES-GCM are illustrated in Fig. 2. The plaintext blocks are processed serially and require correct control to direct data in and out of the blocks. All architectures created in this paper follow the serial implementations. Based on the different properties of the architecture, optimizations are then added on top of the serial architecture. This is discussed in Section IV.

The finite-state machines (FSMs) required for control are presented in Fig. 3. Note that after processing of a cipher text block by the encryption block, the first state machine generates a *ct_done* signal. The authentication FSM first processes the

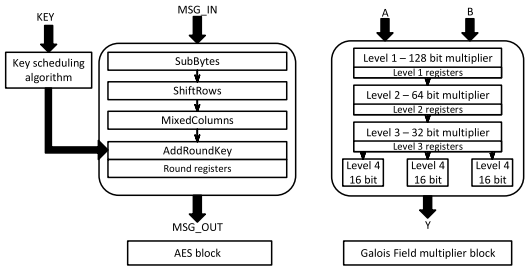

Fig. 2. Data path of AES-GCM consisting of AES block and Galois field multiplier block.

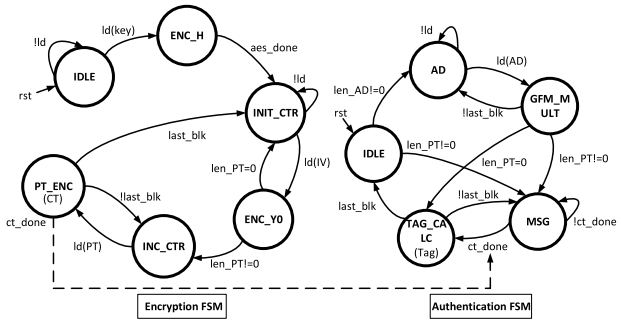

Fig. 3. FSMs of AES-GCM for encryption and authentication in a block-interleaved manner. Signal *ct_done* is used for synchronization between the two FSMs.

associated data blocks and waits for the *ct_done* signal. Upon receiving the signal, the processing of cipher text blocks is performed in an interleaved manner with the encryption process to generate the tag value. The ability to parallelly process multiple message blocks is an important property of the AES-GCM algorithm since it results in a good performance and resource utilization.

B. Deoxys

Deoxys is a block-cipher-based AE algorithm utilizing a tweakable block cipher, *Deoxys-BC*. The tweakable block cipher is based on AES but uses a key and a tweak value. It has more rounds than standard AES, claiming better security. The Deoxys algorithm has two modes: one for which nonce must not be reused and one which is nonce-misuse resistant. The nonce-misuse-resistant version of Deoxys provides full 128-bit security for unique nonces and birthday bound security when nonce is reused. Existing hardware and software implementation also shows that it is suitable for short messages having low precomputation overhead.

The basic steps of Deoxys authentication and encryption are described in Fig. 4. Since the block cipher in Deoxys is a custom built tweakable block cipher, the inputs include an additional tweak value which needs to be provided each time the encryption block is called. From the block diagram, we observe that the associated data blocks are processed first followed by the plaintext blocks. Finally, incorporating the nonce value into the tweak value, the tag is generated. Note that, for the encryption process, the nonce value is used as input to the block cipher. The tag value is incorporated into the tweak value. Thus, there is a dependence between the tag generation and cipher-text generation in Deoxys. This will lead

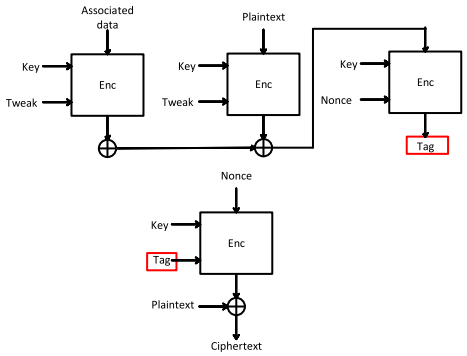

Fig. 4. Description of the algorithm of Deoxys. The encryption block in this figure is the tweakable Deoxys block cipher with a key and tweak as inputs. Top: tag generation. The generated tag is used for processing of plaintext blocks to generate cipher text blocks.

to important differences in the architectural implementation and optimizations.

Note that the nonce is incorporated as the tweak value in tag generation and as the message input in cipher text generation. Now, even if the nonce is reused, we observe that if the associated data and plaintext are different, the generated tag will be different. Since the tag forms a part of the tweak value for cipher text generation, the generated cipher text will also be different. Thus, no information is leaked if nonce is reused providing complete nonce-misuse resistance.

The Deoxys algorithm uses the Deoxys Block cipher for both authentication and encryption purposes. Thus, a hardware implementation requires only one block cipher as illustrated in Fig. 5. For the implementation of the block cipher, we use a standard implementation as used in AES. Since the rounds of Deoxys and AES are similar, the implementation is also similar. However, Deoxys has more number of rounds (14 rounds) compared to 10 rounds of AES. The key schedule is also different in the case of Deoxys and is based on the tweakey schedule.

In this paper, we consider two versions of the Deoxys algorithm presented in the literature, namely, versions v1.3 [18] and v1.41 [19]. One of the main differences between these two versions is the implementation of the tweakey key schedule algorithm. Briefly, Deoxys version v1.3 requires the implementation of simple substitution function h and a multiplication function g. Deoxys version v1.41 replaces the multiplication function g with simple linear-feedback shift registers (LFSRs). More details about the implementation can be found in the algorithm description of these two versions.

The state machine illustrated in Fig. 5 is used to route data to and from the block cipher to perform encryption and authentication process. Since the same block cipher is used for all operations, the implementation is completely serial in nature. Note that there are no multipliers required for this architecture, unlike AES-GCM which reduces resource requirement and eliminates the problem associated with security of polynomial multiplication blocks.

C. AES-GCM-SIV

AES-GCM-SIV combines the GCM building blocks into an SIV paradigm [20]. The basic steps of encryption and

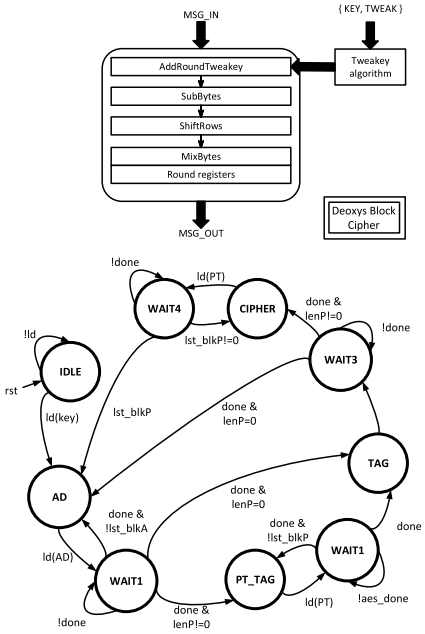

Fig. 5. Data path and control path of the Deoxys algorithm. Note that the block cipher is a tweakable block cipher and a single state machine is used for both authentication and encryption.

decryption are shown in Fig. 6. The main advantage of AES-GCM-SIV is its similarity in structure to AES-GCM algorithm. Mainly, the AES block cipher is still used in CTR mode in this algorithm. The POLYVAL construction used for building the multiplication is similar to the GHASH construction used in AES-GCM. The conversion between the two is expressed in the following equation:

$$\begin{aligned}&\text{POLYVAL}(H, X_1, X_2, .)\\ &\quad = \text{ByteReverse}(\text{GHASH}(\text{ByteReverse}\\ &\qquad \times(H) * x, \text{ByteReverse}(X_1), \text{ByteReverse}(X_2), .)). \quad (3)\end{aligned}$$

However, there are several differences in the overall construction of the algorithms. The tag generation occurs first in AES-GCM-SIV compared to the generation of cipher text blocks occurring first in the case of AES-GCM. Second, the plaintext needs to be processed twice in the case of AES-GCM-SIV instead of just once as in the case of AES-GCM. The occurrence of tag generation before cipher text and the processing of plaintext twice are common to both AES-GCM-SIV and Deoxys algorithms. We assume that the plaintext is stored in memory and can be accessed by the algorithms twice. Future studies will address the practicality and limitations of such an assumption.

In the AES-GCM-SIV algorithm, the nonce (not shown in the figure) is incorporated as part of the tag generation process. The tag then forms a part of the initial counter value for cipher text generation. Hence, similar to Deoxys, even if the nonce is reused, the generated tag will be different. No information will be leaked through the cipher text, providing nonce misuse resistance.

The architecture of AES-GCM-SIV consists of the AES block as well as the POLYVAL block. While the AES block is realized similar to the AES block used in AES-GCM,

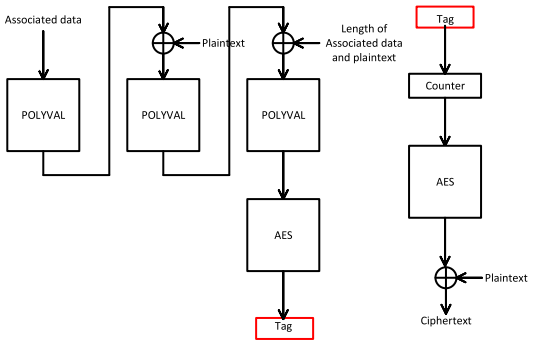

Fig. 6. Algorithm description of AES-GCM-SIV. The left part of the figure represents processing of associated data and message blocks using the POLYVAL and AES block to produce a Tag. The right half of the figure represents cipher text generation using the Tag as an initial value for the counter.

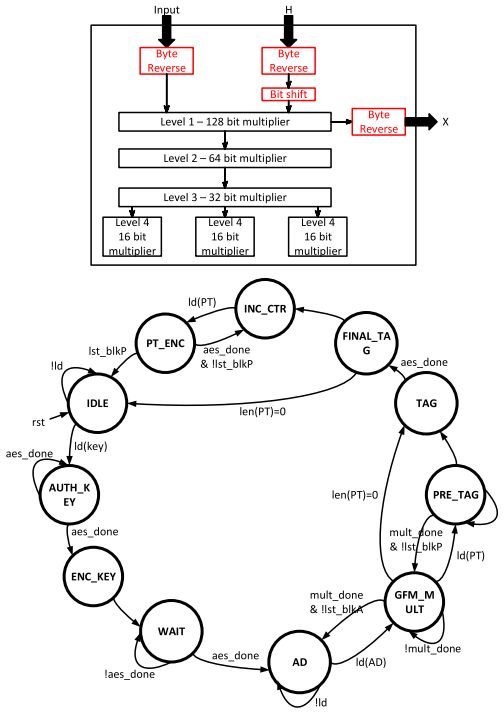

Fig. 7. Data path and control path of AES-GCM-SIV. Only the conversion of *GF_MULT* of AES-GCM to *POLYVAL* of AES-GCM-SIV is presented here. Also, note that a single FSM is used for both authentication and encryption.

the POLYVAL block is realized by making appropriate swap and shift operations to the GHASH composition of AES-GCM algorithm. This is described in detail in [20] and illustrated in Fig. 7. The components marked in red indicate the additional operations required for POLYVAL. The control path of AES-GCM-SIV is also illustrated in this figure. Note that due to the dependence of the cipher text generation on the tag generation process, only a single serial finite state machine just as in Deoxys can be used. The state machine routes data to and from the AES and POLYVAL blocks.

D. POET

POET is a block-cipher-based algorithm which provides both nonce misuse resistance as well as decryption misuse resistance. This means that if authentication fails, the adversary

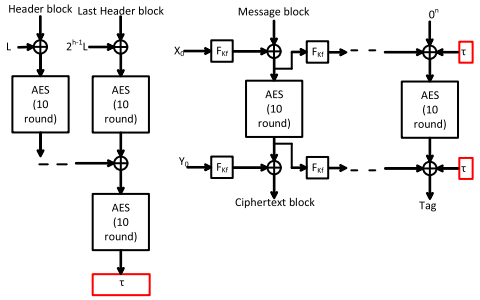

Fig. 8. Algorithm description of POET. The left part of the figure presents processing of associated data/header blocks to produce an intermediate value τ. Cipher text and Tag generation using the intermediate τ value are presented in the right half of this figure.

obtains no information about decrypted cipher texts. POET uses the AES block cipher itself for both encryption and authentication and does not require any Galois field multiplication operations. Thus, the algorithm can reuse available AES modules and provide required AE operations. The POET algorithm is also described as a flexible algorithm since it is not necessary to use AES block ciphers in this construction. It is shown that the AES can be replaced with a smaller four-round AES block cipher or other such lightweight block ciphers. This will reduce the resource overhead of POET while providing lower security which can be suitable for some applications. The authors also propose an encryption only version of the algorithm termed POE whose details can be referenced from [21].

The top-level algorithm description of POET is provided in Fig. 8. The associated data blocks are processed using a hashing key L and the AES block cipher to produce an intermediate value τ. This is represented in the first part of the figure. The processing of plaintext blocks occurs next. This occurs through the AES block ciphers and hash functions F_{K_F}. Here, we note that the output of the message processing blocks is used together with the value τ to generate the final tag. Hence, in POET, the cipher text generation occurs first similar to AES-GCM algorithm.

In POET, the nonce is a part of the header block shown in the figure. The message block is a direct input to the AES block for cipher text generation. Hence, even if the nonce is repeated, any change in the message block results in a completely different cipher text block. Thus, no information is leaked, and misuse resistance is achieved.

The top-level block diagram of POET is illustrated in Fig. 9. The architecture is constructed using one AES block cipher (which is utilized for both encryption and authentication) and two four-round AES blocks which serve as the hash function (F_{K_f}) blocks. We observe from the algorithm description of POET (Fig. 8) that an initial vector X is first processed by the F_{K_f} block and XORed with the message block. After processing of message block using the AES module, the Y vector processed by the F_{K_F} block is utilized to produce the cipher text block. However, the processing of the next message block has a dependence on the F_{K_F} output of the previous message block. Hence, the processing of Y vector and processing of previous message blocks is performed simultaneously using two F_{K_F} function blocks.

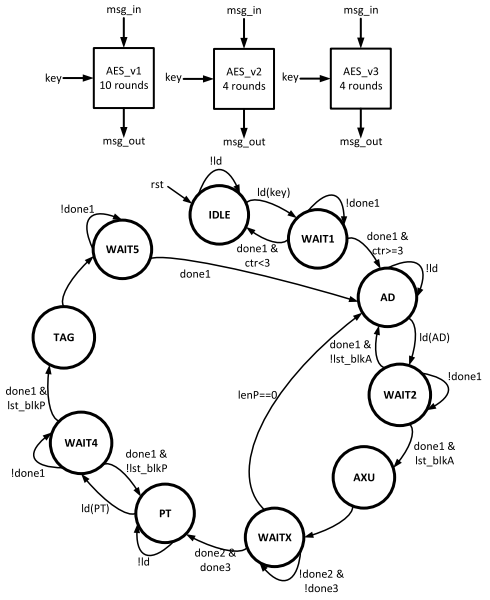

Fig. 9. Data path and control path of the POET algorithm. Note that the data path consists of a regular AES ten-round cipher and two AES four-round ciphers used as hash function blocks.

E. PRIMATE-APE

Unlike the block-cipher-based algorithms, the PRIMATE-APE is a sponge-/duplex-based construction termed permutation-based AE. The sponge construction is based on a fixed-length permutation block which operates on a fixed number of bits. These bits are divided into two parts: r also known as the bit-rate or rate and c termed as capacity. The r bits of the input are absorbed by multiple permutation function blocks in an iterative manner and squeezed out in subsequent stages. The capacity bits contribute to the permutations but are never changed or released as an output. Using sponge constructions in different ways, both ciphers and hashing algorithms can be created. The duplex construction is a variation of the sponge construction in that absorption and squeezing of data occur in an alternating fashion.

The PRIMATE-APE algorithm utilizes a duplex-based construction to create an AE algorithm. The permutation block is termed as PRIMATE in this algorithm. A high-level overview of the construction is presented in Fig. 10. We observe from this figure that the nonce/associated data blocks are absorbed in an iterative manner by the permutation block followed by the absorption of the message blocks. Finally, the cipher text block and tag are generated from the last permutation block. The input at the top which forms the rate r is of size 40 bits and the input at the bottom which forms the capacity c is of size 240 bits. Hence, a 240-bit key is required, and processing of associated data/message blocks occurs in blocks of 40 bits each. The details of the PRIMATE block are discussed in Section III-E.

In PRIMATE-APE, the nonce forms an input to the PRIMATE block early in the algorithm. The message block is also an input to the PRIMATE block. Thus, even if nonce is repeated, since the generated cipher text will be different, no information about the plaintext is leaked. Thus, nonce-misuse resistance is achieved.

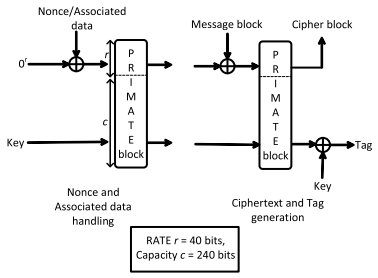

Fig. 10. Algorithm description of PRIMATE-APE. After initial absorption of the nonce and associated data by the PRIMATE block, for every message block that is input, the cipher text block is produced. Finally, the tag is generated.

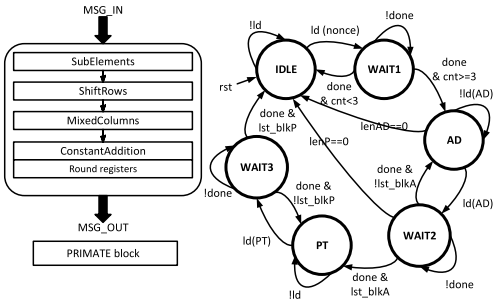

Fig. 11. Data path and control path of PRIMATE-APE algorithm. Note that the implementation of the PRIMATE block is similar to AES block but is much smaller and utilizes nibbles for processing.

The architecture of PRIMATE-APE mainly consists of the PRIMATE block as illustrated in Fig. 11. The PRIMATE block used in this paper is a version termed PRIMATE-120 as described in [22]. The structure of PRIMATE is similar to the AES block cipher and consists of rounds: SubElements, ShiftRows, MixColumns, and ConstantAddition. Hence, the same architecture as used for construction of AES block cipher can be used. However, the state elements of PRIMATE are 5-bit elements arranged in a 7×8 state matrix. The first row of the state is the rate and the rest is the capacity. Also, the key schedule algorithm is not required in case of PRIMATE. The 5-bit LFSRs are used to create the constants required for ConstantAddition round.

We observe from the architecture that the PRIMATE-APE is a lightweight AE algorithm because of the use of only one PRIMATE block. However, it can also be observed that all transactions with this block occur through the rate bits which are only 40 bits wide. Hence, this architecture will be significantly slower than other architectures presented in this paper. However, for short messages, this architecture will serve as a good lightweight option providing nonce-misuse resistance.

IV. Optimizations

In this section, we discuss some of the optimizations that can be applied to the discussed AEAD schemes. These optimizations are either targeted to utilize the properties of the FPGA platforms or the properties of the algorithm itself. In Section V, we present the results of optimization on the algorithms and discuss which of the optimizations are most suitable for each algorithm.

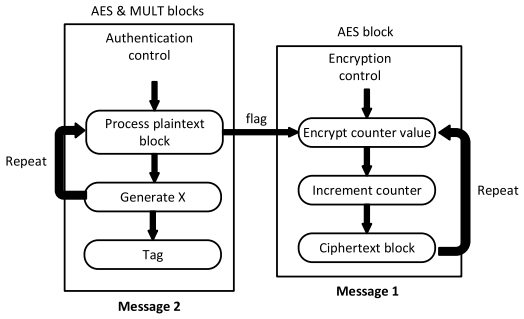

Fig. 12. Parallel processing of messages in AES-GCM-SIV.

A. Parallel Processing of Messages

From the architecture of AES-GCM, we have observed that it is inherently parallel in nature. This means that when a cipher text block is being generated by the encryption algorithm, parallel processing of data by the authentication algorithm can occur. This is an important property which results in low cycles per byte for AES-GCM. However, by modifications, such as pipelining and parallelism, this property can be utilized to optimize other algorithms.

AES-GCM-SIV utilizes AES and multiplier blocks for encryption and authentication operations. Hence, similar to AES-GCM, the two operations can be separated and performed in parallel. However, in AES-GCM-SIV, authentication occurs first and there is a data dependence between authentication and encryption. This means that first the authentication process must be performed completely to generate the tag and then the cipher text blocks can be generated through encryption. To break this dependence, we can process two messages in parallel and pipeline the design such that the two processes occur using two different FSMs. This concept is illustrated in Fig. 12. Synchronization is necessary for the parallel processing of messages. This can be achieved by using flags as indicated in the figure. After the associated data blocks of the second message are processed, the processing of plaintext blocks using the multiplier begins. At this time, the processing of plaintext blocks of message 1 can be carried out by the second state machine. Note that the tag of the first message needs to be stored for the encryption block to access during processing of message 2.

Deoxys uses a single block cipher for both authentication and encryption. Hence, to apply parallel processing to the Deoxys algorithm, two block ciphers are required. This process is similar to loop unrolling or unfolding technique [23] where the resources are replicated to process more data in the same time frame. Now, the authentication process of message 2 can occur in parallel with the encryption of message 1 blocks. The two processes can be completely separated since there is no contention of resources between the two processes. The concept of parallel processing in Deoxys is illustrated in Fig. 13. The *tag_done* signal which indicates that the tag processing has been completed can itself be used as a flag for the processing of cipher text blocks. Parallelly, the authentication state machine can begin the processing of message 2. Note that this results in reduction in time for

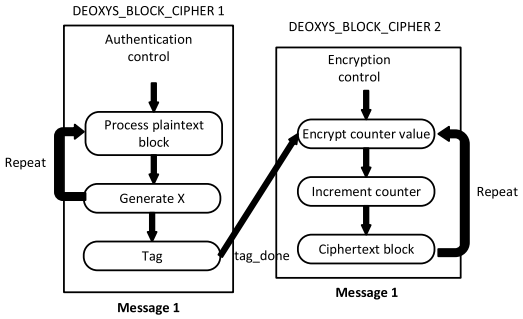

Fig. 13. Parallel processing of messages in Deoxys.

processing of messages but will also increase the resource consumption because of duplication.

B. Implementation of S-Box in Memory

The discussed algorithms utilize lookup tables (LUTs) in the form of S-Boxes of block ciphers. Each S-Box maps 1 byte of a message to 1 byte of the substitution value. Since each message block is of size 16 bytes (128 bits), 16 such S-Boxes are necessary to process a complete message block. These LUTs can be implemented using a straightforward implementation where the LUTs are synthesized using pure combinational logic elements. However, we note that most modern-day device platforms such as FPGAs and microcontrollers have some form of memory available on board. This memory can be used to port the LUTs of S-Boxes. This results in reduction of logic elements and utilization of the available memory blocks of the device platform.

C. Tower Field Implementation of Block Cipher

To implement the AES algorithm using a compact implementation, the S-Box of AES is replaced using tower field implementations. Basically, the S-Box lookup can be considered as an inversion in the Galois field (2^8). By decomposing this field into subfields, more optimized and compact implementations can be obtained. These decomposed fields are termed tower fields. Several papers discuss the tower field implementations [24]–[27]. In this paper, we adopt two implementations in the form of $GF(((2^2)^2)^2)$ and $GF((2^4)^2)$ decompositions.

D. Hardware–Software Codesign

Next, we discuss a design style which can serve as a potentially powerful tool in designing for low-resource applications in the form of a hardware/software codesign approach. Typically, hardware/software codesign is used as a method to increase execution speed of pure software algorithms by offloading work to dedicated hardware blocks. However, here, we look at hardware–software codesign from the perspective of reducing area and optimizing resources. The rationale behind this idea is that while blocks such as AES and Galois field multiplication perform efficiently in hardware, the design and optimization of FSMs on hardware are a challenging task. Implementing this in software simplifies execution while providing for a more flexible and reusable implementation.

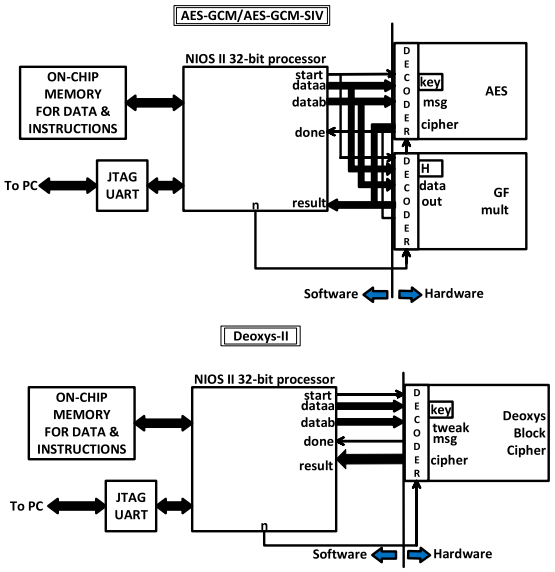

Fig. 14. Block diagram of a codesign implementation using NIOS II soft processor. The top part of the figure represents codesign on an AES-GCM/AES-GCM-SIV module. The bottom part presents codesign on Deoxys.

With modern development platforms typically including a processor, memory, and hardware blocks, this approach completely utilizes and optimizes resources in the best possible manner.

1) Codesign of AES-GCM: A codesign approach is deployed by retaining the AES blocks and Galois field multiplier in hardware, while implementing the top-level state machines in software. The software runs on a processor which could be a specialized processor or reused from one existing on the board. For example, when using an Altera FPGA board, we can make use of a 32-bit NIOS II soft processor core by running a C program on it. A top-level block diagram of the codesign with the hardware and software boundaries is shown in Fig. 14. For accessing data to and from the hardware blocks, we need address decoders which are built as a wrapper around the hardware blocks converting them into *custom instructions*. These instructions can then be called as macros in the software program. It is to be noted that data transfer occurs in blocks of 32 bits. However, our design uses 128-bit data requiring a stream of data to be transferred between the hardware and software boundary and synchronized using start and done signals. The decoders at the hardware/software boundary accept a 4-bit opcode n from the NIOS II processor. Values of n and their corresponding operations are shown in Table II. For each transaction, the NIOS II processor sends out a start signal and waits for a done signal from the decoder block.

2) Optimization of the Codesign: While the codesign approach reduces area and lowers power consumption, it suffers from an increase in latency due to the transfer of data between the processor and AES/Galois field multiplier blocks. Thus, there is a tradeoff between resource consumption versus speed of execution. However, optimizations can be made to reduce this latency by minimizing communications between the processor and hardware blocks by making some

TABLE II
TABLE OF OPCODES AND THEIR CORRESPONDING OPERATIONS FOR DECODERS

Opcode n	Operations in AES-GCM and AES-GCM-SIV	Operations in Deoxys
0	Write key/H[31:0], key/H[63:32]	Write key[31:0], key[63:32]
1	Write key/H[127:96], key/H[95:64]	Write key[127:96], key[95:64]
2	No operation	Write tweak[31:0], tweak[63:32]
3	No operation	Write tweak[127:96], tweak[95:64]
4	Write data[31:0], data[63:32]	Write data[31:0], data[63:32]
5	Write data[127:96], data[95:64]	Write data[127:96], data[95:64]
6	Perform operation on hardware and wait for done signal	
7	Read result [31:0]	
8	Read result [63:32]	
9	Read result [95:64]	
10	Read result [127:96]	
11-15	No operations	

critical observations. It is seen that the AES block repeatedly uses the secret key while the Galois field multiplier block needs access to the H value calculated as part of initialization. The two values are marked in Fig. 14. Thus, by transferring these values only once and storing them on the hardware blocks itself, latency per execution of hardware blocks can be reduced.

The approach of using S-Box in memory blocks can also be utilized since the AES algorithm is still completely present on hardware. This further reduces the area requirement of hardware blocks and utilizes the available memory resources more efficiently. In fact, the data memory accessed by the processor can be shared for the storage of S-Boxes resulting in better resource utilization of the complete system.

3) Application of Codesign Techniques to AES-GCM-SIV and Deoxys: The AES-GCM-SIV algorithm matches the AES-GCM algorithm except for the modifications to Galois field multiplier blocks (which are necessary to convert them to POLYVAL blocks) and top-level state machines. Hence, the mapping from hardware to software is the same as that of AES-GCM. All optimizations are equally applicable, and the resulting benefits are similar to the AES-GCM algorithm. An alternate approach to codesign of AES-GCM-SIV is to reuse the hardware blocks of AES-GCM and move all changes required by the POLYVAL hashing operations to software. This results in complete flexibility of use where a nonce-misuse-resistant and secure AES-GCM-SIV algorithm or the faster AES-GCM algorithm can be chosen based on the application.

The Deoxys algorithm differs by using the Deoxys block cipher for both encryption and authentication. Hence, the Deoxys block cipher is built on hardware similar to the AES block and repeatedly accessed for both encryption and authentication. Because of the requirement of only one hardware block, the overall area of this design will be the lowest. This indicates that the Deoxys algorithm benefits the most due to codesign approach. Also, note that the Deoxys algorithm has an additional tweak input which needs to be provided along with the key and message value. Unlike the key, the tweak input is not constant and cannot be stored on the hardware. This requires two additional opcodes (2 and 3) as presented in Table II. Further discussion and comparison between the three algorithms will be presented in Section V.

V. RESULTS

In this section, we describe the experimental setup adopted for all implementations and discuss the corresponding results obtained after applying optimizations discussed in Section IV. Results are reported both in terms of FPGA and ASIC implementations wherever applicable. This ensures platform obliviousness while making the final comparisons between different algorithms. Note that the described results mainly compare the encryption operation of the AEAD algorithms. Comparison of the decryption or a combination of both is scope for future work.

A. Experimental Setup

All results on the FPGA platform are obtained using Altera's Cyclone V family of FPGA which are built on TSMC's 28-nm low-power process technology. Specifically, Altera Cyclone 5CSEMA4U23C6 incorporated in an ATLAS SoC board is utilized. The Cyclone V family of FPGAs allows for low-area, low-cost implementations of algorithms. The implementations are written in Verilog Hardware Descriptive Language and simulated using the ModelSim tool. Synthesis is performed using the Altera Quartus Prime tool and timing measurements are performed using the TimeQuest timing analyzer. While running synthesis, area optimization is enabled, and a target frequency of 50 MHz is used. Power measurements are performed using the PowerPlay power analyzer tool.

For every experiment performed, the following measurements are reported.

1) The area consumption is reported both in terms of LUTs, which are the basic combinational elements of FPGAs, as well as register count. The resource utilization of the cipher as a percentage of the total adaptive logic modules (ALMs) and registers of the FPGA is also presented.
2) The output of power play power analyzer tool is reported in terms of power in milliwatt. Before running the tool, appropriate inputs in terms of value change dump files of the simulation of modules is provided. Only dynamic power is considered since the leakage power reported by the tool is for the entire fabric of the FPGA and does not correctly represent the leakage power of the design. Details of the generation of power values can be referenced from [7].
3) We report the time of operation in terms of cycles per byte which is defined as the number of cycles required to process each byte of the message. This measure is independent of the FPGA platform and is an important measure of the performance. It is calculated using the

TABLE III
AREA, THROUGHPUT, AND POWER OF AES-GCM AND OTHER ALGORITHMS USING DIRECT IMPLEMENTATION (FREQUENCY = 50 MHz)

	FPGA results							ASIC results	
Algorithm	**Area**	**Resource Utilization**	**Power in mW**	**Cycles per byte**	**Throughput in Mbps**	**Energy in pJ/bit**	**Throughput /Area**	**Area in GE**	**Total power in mW**
AES-GCM	LUTs: 4087 Regs: 2470	ALMs: 23% Regs: 14%	19.52	1.058	417. 16	51.63	0.101	40784	3.317
Deoxys - II (v 1.3)	LUTs: 3134 Regs: 2084	ALMs: 19% Regs: 13%	10.20	1.82	348	46.41	0.111	33422	2.795
Deoxys - II (v 1.4.1)	LUTs: 3046 Regs: 2081	ALMs: 20% Regs: 13%	9.61	1.82	384.24	43.73	0.126	32616	2.764
AES-GCM-SIV	LUTs: 4273 Regs: 2572	ALMs: 25% Regs: 15%	17.59	1.46	286.88	61.31	0.067	42995	2.965
POET	LUTs: 4824 Regs: 3012	ALMs: 31% Regs: 16%	16.14	1.37	476	55.28	0.098	61618	5.031
PRIMATE-APE	LUTs: 1699 Regs: 925	ALMs: 14% Regs: 10%	3.63	3	194	27.22	0.114	11292	1.276

following equation:

$$\text{Cycles\_per\_byte} = \frac{\text{Cycles\_per\_msg}}{\text{Size\_of\_msg\_in\_bytes}}. \quad (4)$$

The value for the number of cycles is obtained through simulation.

4) The throughput, which is defined as the number of bits that can be processed every second of the time of operation, is measured using the following equation:

$$\text{Throughput} = \frac{\text{Num\_of\_blks} \times \text{Blk\_length}}{\text{Cycles} \times \frac{1}{\text{Max\_freq}}}. \quad (5)$$

The value of Max_freq is obtained from synthesis. Hence, there is a dependence of the throughput on the FPGA platform. This is different from the cycles per byte measure which is not dependent on the hardware platform used.

5) The energy consumed to process every bit of the message is an important measure and can be calculated by measuring power and time of operation. For the measurement of energy consumption, the following equation is used:

$$\text{Energy} = \frac{\text{Power}}{\text{Frequency}} \times \frac{\text{Cycles}}{\text{Num\_of\_blks} \times \text{Blk\_length}}. \quad (6)$$

The throughput/area which defines a tradeoff between the speed of operation of the design versus the area consumption of the design is measured using the following equation:

$$\text{Throughput/Area} = \frac{\text{Throughput\_in\_Mb/s}}{\text{Area\_in\_LUTs}}. \quad (7)$$

The Throughput/Area is a good measure which allows comparison between implementations of the different algorithms.

6) For measurement of the performance of parallel designs (after application of optimization described in Section IV-A), the above-mentioned measurements are repeated for 100 messages.

All ASIC implementations are synthesized using the *Design Compiler* tool using a 65-nm technology library and target frequency of 50 MHz. The area results are reported in terms of gate equivalent (GE), calculated using the following equation:

$$\text{Gate\_Equivalent (GE)} = \frac{\text{Total\_area\_of\_algorithm}}{\text{Area\_of\_NAND\_gate}}. \quad (8)$$

The total power which is a sum of the dynamic, internal, and leakage power is reported in milliwatt. For a 65-nm technology, the dynamic power dominates over other power values. This gives a validation for the power numbers reported by FPGA implementations.

B. Implementation 1: Direct Implementation of the Algorithms Using Combinational Logic Only

The architectural descriptions provided in Section III are directly mapped to hardware. This results in implementations which are unoptimized but provide a good first comparison between all the algorithms. The results are tabulated in Table III for a clock speed of 50 MHz. Note that ASIC implementations are also included in the same table for comparison.

From the results, we observe that in terms of area, PRIMATE-APE is the smallest. This can be attributed to the fact that the PRIMATEs are sponge-based designs and, hence, are expected to be lightweight. The next smallest architecture with respect to area is the Deoxys algorithm. With the modifications of the algorithm as applied to version v 1.4.1, the area of the algorithm has reduced further. Both versions of Deoxys have the advantage of utilizing just one block cipher for both authentication and encryption processes. This avoids multiplication which is a resource-intensive operation. AES-GCM-SIV and POET are slightly more resource consuming in terms of area compared to AES-GCM. The power consumption trends are almost similar to the area consumption trends with PRIMATE and Deoxys having the least FPGA power consumption values.

With respect to performance, AES-GCM has the lowest cycles per byte value. This is because of the inherently parallel nature of this algorithm which is reflected even in the unoptimized implementation of its architecture. With respect to the reported throughput, POET and AES-GCM architectures are the fastest with PRIMATE being the slowest among all. With respect to energy consumption, PRIMATE consumes the

TABLE IV

AREA, THROUGHPUT, AND POWER OF AES-GCM AND OTHER ALGORITHMS USING PARALLEL PROCESSING OF MESSAGES (FREQUENCY = 50 MHz)

	FPGA results							ASIC results	
Algorithm	**Area**	**Resource Utilization**	**Power in mW**	**Cycles per byte**	**Throughput in Mbps**	**Energy in pJ/bit**	**Throughput/Area**	**Area in GE**	**Power in mW**
AES-GCM	LUTs: 4087 Regs: 2470	ALMs: 23% Regs: 14%	19.52	1.058	417. 16	51.63	0.101	40784	3.317
Deoxys - II (v 1.3)	LUTs: 4773 Regs: 2773	ALMs: 26% Regs: 16%	16.98	1.27	497.63	53.91	0.104	49894	4.361
Deoxys - II (v 1.4.1)	LUTs: 5004 Regs: 2859	ALMs: 27% Regs: 16%	16.61	1.27	548.03	52.73	0.110	48687	4.332
AES-GCM-SIV	LUTs: 4262 Regs: 2668	ALMs: 25% Regs: 16%	20.89	1.11	398.55	53.65	0.093	43278	4.170

*Note that for FPGA implementations only dynamic power is reported. This is because the static power reported by the tool is for the entire FPGA fabric and does not correctly represent the static power for the design under consideration.

**Note that for ASIC implementations the total power which is a sum of the dynamic, internal and leakage power is reported. For a 65 nm technology library, the dynamic power dominates the total power.

least energy while AES-GCM-SIV consumes the most energy. The throughput/area indicates that overall, Deoxys has the best ratio. In other words, it has area consumption and performance values in between that of other algorithms making it a good architecture overall. The ASIC implementation results in terms of both area and power are reflective of the FPGA results with PRIMATE and Deoxys consuming the least resources.

Note that both versions of Deoxys have the same cycles per byte value. This is because the major change in the two versions is with respect to the tweakey key schedule of the Deoxys block cipher and initialization method. These affect the maximum frequency reported and, hence, the throughput of the algorithm. They do not affect the cycles per byte value which is a platform independent number.

C. Implementation 2: Optimization Using Parallel Processing

The optimization described in Section IV-A is applied to both Deoxys and AES-GCM-SIV and the resulting implementation values are reported in Table IV. We note that the unoptimized implementation of AES-GCM is inherently parallel. Hence, there is no change in the reported values for area, power, and timing of AES-GCM.

From these results, we observe that all algorithms have improved performance benefits after application of the parallel processing technique. The cycles per byte value is significantly reduced and the throughput is significantly increased for all algorithms. However, for parallel processing of Deoxys algorithm, two block ciphers are required. This results in significant increase in area consumption for Deoxys compared to AES-GCM-SIV, which, in fact, sees a slight reduction in area consumption. This is because the same blocks of AES-GCM-SIV as used in the serial version are also utilized in the parallel version of the algorithm. The overall throughput/area is high for all algorithms and is comparable to AES-GCM. However, for Deoxys, there is a reduction in the throughput/area number because of the increase in area even though the throughput also increases significantly. For AES-GCM-SIV, there is improvement in the throughput/area value compared to the serial version. The ASIC area numbers are similar to the area results obtained for FPGA implementation while the ASIC power numbers are more reflective of the results of energy consumption.

D. Implementation 3: Optimization of the S-Box of the Block Ciphers Using Memory Blocks

For the optimization described in Section IV-B, we consider only Deoxys and AES-GCM-SIV. However, it is to be noted that the optimization is equally applicable to all the algorithms under study. For Deoxys, the serial implementation is considered since it has a better throughput/area compared to its parallel implementation. However, for AES-GCM-SIV, the parallel implementation of the algorithm is used instead of the serial version for application of the S-Box optimization.

The result of optimization using memory blocks is provided in Table V. The reported results show the memory requirements in kilobytes for storing the S-Box values. We observe from these results that all algorithms map to architectures with lower area and power requirements than their direct or optimized versions. This is because of the hardware requirement shifts from using purely combinational logic elements to the memory blocks present on board the FPGA. For AES-GCM and AES-GCM-SIV, S-Boxes are required for both key expansion and substitution steps. However, for Deoxys, only the substitution step requires S-Boxes. Hence, the memory requirement is also lower. For all the algorithms, the LUT consumption is reduced by almost 20%. This occurs at a cost of decrease in the throughput and increase in the energy consumption for almost all algorithms. The modified version of Deoxys reports better improvements compared to the older version since the power consumption is not as high resulting in lower energy consumption values.

E. Implementation 4: Optimization Using Tower Field Implementations

Optimization using tower field implementations are applied to the algorithms which are based on AES. Mainly, optimizations as applied to AES-GCM, AES-GCM-SIV, and POET are presented in Table VI. For the generation of these results, we employ two decompositions: $GF((2^4)^2)$ and $GF(((2^2)^2)^2)$ for all the algorithms. Note that the reduction in resources due to compact S-box implementation is not reflected in the implementations on Cyclone V platforms. However, implementations on the ASIC platform reflect this reduction. This can be attributed to the fact that modern FPGA platforms having routing strategies which might result in some optimizations

TABLE V

AREA, THROUGHPUT, POWER, AND MEMORY REQUIREMENTS USING OPTIMIZATION OF S-BOX USING MEMORY BLOCKS (FREQUENCY = 50 MHz)

Algorithm	Area	Resource Utilization	Memory	Throughput (Mbps)	Power (mW)	Energy pJ/bit	Throughput/Area
AES-GCM	LUTs: 3272 Regs: 2413	ALMs: 18% Regs: 13.6%	5 KB	402.72	21.35	53.01	0.123
Deoxys - II (v 1.3)	LUTs: 2482 Regs: 1821	ALMs: 15% Regs: 13.6%	4 KB	334.64	16.27	74.02	0.135
Deoxys - II (v 1.4.1)	LUTs: 2362 Regs: 2081	ALMs: 16% Regs: 13.5%	4 KB	373.62	13.79	62.74	0.158
AES-GCM-SIV	LUTs: 3497 Regs: 2444	ALMs: 20% Regs: 15.4%	5 KB	308.89	22.18	71.80	0.088

TABLE VI

RESULTS OF OPTIMIZATION USING TOWER FIELD IMPLEMENTATIONS OF AES

	FPGA results							ASIC results	
Algorithm	**Area**	**Resource Utilization**	**Power in mW**	**Cycles per byte**	**Throughput in Mbps**	**Energy in pJ/bit**	**Throughput/Area**	**Area in GE**	**Power in mW**
AES-GCM GF(2^4)^2)	LUTs: 4572 Regs: 2407	ALMs: 22% Regs: 13%	15.47	1.058	275.69	40.92	0.060	34827	2.997
AES-GCM GF(2^2)^2)	LUTs: 4735 Regs : 2407	ALMs: 23% Regs: 13%	17.80	1.058	279.47	47.08	0.059	35307	3.095
AES-GCM-SIV GF(2^4)^2)	LUTs: 4664 Regs: 2566	ALMs: 23% Regs: 14%	19.63	1.11	288.28	54.47	0.062	35138	3.750
AES-GCM-SIV GF(2^2)^2)	LUTs: 4808 Regs: 2566	ALMs: 25% Regs: 14%	21.81	1.11	252.46	60.52	0.052	35634	3.868
POET GF(2^4)^2)	LUTs: 5328 Regs: 2878	ALMs: 31% Regs: 16%	23.51	1.37	317.95	80.52	0.059	57989	4.569
POET GF(2^2)^2)	LUTs: 5424 Regs: 2878	ALMs: 31% Regs: 16%	24.26	1.37	301.31	83.09	0.055	58966	4.688

*Note that for FPGA implementations only dynamic power is reported. This is because the static power reported by the tool is for the entire FPGA fabric and does not correctly represent the static power for the design under consideration.

**Note that for ASIC implementations the total power which is a sum of the dynamic, internal and leakage power is reported. For a 65 nm technology library, the dynamic power dominates the total power.

TABLE VII

RESULTS OF HARDWARE/SOFTWARE CODESIGN IMPLEMENTATION

Algorithm	Area	Cycles per byte	Memory footprint	Component FPGA Power in mW
AES-GCM	LUT: 2562 Registers:1899	19.19	10.507 KB	5.13 (AES), 6.95 (Multiplier)
Deoxys - II (v 1.4.1)	LUT: 1507 Registers: 1080	22.96	7.247 KB	6.00 (Deoxys block cipher)
AES-GCM-SIV	LUT: 2613 Registers: 1666	24.35	9.900 KB	5.43 (AES), 7.34 (Multiplier)
NIOS II processor: 699 LUTs, 584 Registers (32-bit interface) Power consumption = 4.93 mW				

*Note that for FPGA implementations only dynamic power is reported. This is because the static power reported by the tool is for the entire FPGA fabric and does not correctly represent the static power for the design under consideration.

**Note that for ASIC implementations the total power which is a sum of the dynamic, internal and leakage power is reported. For a 65 nm technology library, the dynamic power dominates the total power.

not producing the desired effects. Hence, these optimizations might be more applicable to ASIC platforms or on other FPGA platforms such as Cyclone IV.

F. Implementation 5: Optimization Using Hardware–Software Codesign Approach

The results of implementation of hardware/software codesign approach are reported in Table VII. In this table, the area and power of the components of the design built into hardware are shown. The hardware requirements of the NIOS II soft processor are reported separately. Note that this overhead is common for all designs and is a platform specific value. From the implementations, we observe that the area requirements for all designs are reduced significantly. This is because only the hardware modules are now implemented using LUTs and registers. This reduction is obtained at an increased cycle per byte count. Note that there is an almost 10× increase in this value. Also, we observe that the component power is quite low compared to the power requirements if the complete module is built using LUTs.

Among the three algorithms, Deoxys benefits the most with a hardware–software codesign approach since the Deoxys algorithm utilizes only one Deoxys hardware block. Both AES-GCM and AES-GCM-SIV exhibit similar reduction in area and power because of similar requirements of cipher block and multiplier block.

VI. ANALYSIS OF SIDE-CHANNEL ATTACKS AND COUNTERMEASURES

AES-based or block-cipher-based schemes are susceptible to side-channel attacks targeting the nonlinear S-Box operation or the key schedule/tweak-key schedule. Classical algorithmic and design-oriented side-channel attack countermeasures such as masking or randomization

schemes [28], [29] for simple power analysis (SPA) and differential power analysis (DPA) and fault attacks on AES are applicable for most block-cipher-based AE schemes. In addition, circuit-level countermeasures such as using dual-rail logic style, power supply noise detector, or laser attack shields are useful for all the AE schemes. Sponge-based AE schemes are more secure against side-channel attacks compared to block-cipher-based candidates as they do not have a vulnerable key schedule which is the target of several attacks.

In general, cryptographic implementations can be protected via mechanisms involving frequent rekeying as a countermeasure to DPA attacks. Rekeying achieves this by limiting the number of processed inputs per key for the cryptographic primitive. Each plaintext to be encrypted for the underlying block cipher is provided with a new session key. This session key is derived from a preshared master secret and a nonce that is randomly generated on the tag. This inherently prevents DPA on the session key of the block cipher. However, for the session key derivation, a rekeying function that maps the master secret key to the session key and is easy to protect against both SPA and DPA attacks is required. In this context, a new AE candidate, called ISAP [30] has been designed based on the Keyak sponge-based CAESAR candidate, which provides inherent side-channel attack resistance by design. It uses a secure rekeying operation to protect against DPA attacks. In ISAP, several options are proposed for the rekeying function. A masking scheme involving polynomial multiplication of the secret key and the nonce which is strengthened using learning parity with leakage and learning with rounding. Other schemes that are presented are based on the GCM construction that mixes the secret key with the nonce using a tree-based approach. Another scheme also reuses GCM but combines it with a leakage-resistant pseudorandom function.

VII. Conclusion

In this paper, we have presented the implementation of several nonce-misuse-resistant algorithms using both FPGA and ASIC platforms. All the candidates presented offer nonce-misuse resistance compared to AES-GCM algorithm. This arises from the fact that in AES-GCM, the nonce was directly used by the AES algorithm in CTR mode. The plaintext was only XORed to this result. Thus, any repetition of the nonce leaked information about the plaintext. In both Deoxys and AES-GCM-SIV, the cipher text is dependent on the tag which changes even if the nonce is repeated. In POET and PRIMATE, the message block is fed as an input to the block cipher/sponge block, removing the direct dependence of the cipher text on the nonce. Thus, nonce-misuse resistance is achieved in all these algorithms compared to AES-GCM.

Important architectural differences that arise because of adding nonce-misuse resistance were discussed and optimizations which can be used to overcome the limitations were also presented. Based on the results from Section V, we have provided recommendations for the candidates most suitable for each application scenario as presented in Table VIII. Future work, in this direction, will involve exploring further architecture optimizations of these algorithms. A preliminary discussion on side-channel analysis was presented in this paper. Experimental attacks using side-channels and proposal of countermeasures are topics of further research.

TABLE VIII
Recommendations of Candidates Based on Observed Results

Application	Parameter	Candidate
Resource-constrained	Energy/Power	PRIMATE-APE
High performance	Throughput	POET
Efficient trade-off	Throughput/Area	Deoxys-II
Reusability	Reuse of AES-GCM blocks	AES-GCM-SIV
Small messages	Initialization cycles	Deoxys-II
Software performance	cpb	AES-GCM
Co-design low area requirement	Area/Memory	Deoxys-II

References

[1] P. G. Lopez *et al.*, "Edge-centric computing: Vision and challenges," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 5, pp. 37–42, Oct. 2015.

[2] M. Bellare and C. Namprempre, "Authenticated encryption: Relations among notions and analysis of the generic composition paradigm," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur.* Berlin, Germany: Springer, 2000, pp. 531–545.

[3] J. Salowey, A. Choudhury, and D. McGrew, *AES Galois Counter Mode (GCM) Cipher Suites for TLS*, document RFC 5288, 2008.

[4] D. McGrew and D. Bailey, *AES-CCM Cipher Suites for Transport Layer Security (TLS)*, document RFC 6655, 2012.

[5] M. Bellare, P. Rogaway, and D. Wagner, "The EAX mode of operation," in *Proc. Int. Workshop Fast Softw. Encryption*. Berlin, Germany: Springer, 2004, pp. 389–407.

[6] P. Rogaway, M. Bellare, and J. Black, "OCB: A block-cipher mode of operation for efficient authenticated encryption," *ACM Trans. Inf. Syst. Secur.*, vol. 6, no. 3, pp. 365–403, Aug. 2003.

[7] S. Koteshwara, A. Das, and K. K. Parhi, "FPGA implementation and comparison of AES-GCM and Deoxys authenticated encryption schemes," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2017, pp. 1–4.

[8] S. Koteshwara, A. Das, and K. K. Parhi, "Performance comparison of AES-GCM-SIV and AES-GCM algorithms for authenticated encryption on FPGA platforms," in *Proc. Asilomar Conf. Signals, Syst. Comput.*, Oct. 2017, pp. 1331–1336.

[9] H. Handschuh and B. Preneel, "Key-recovery attacks on universal hash function based MAC algorithms," in *Proc. Annu. Int. Cryptol. Conf.* Berlin, Germany: Springer, 2008, pp. 144–161.

[10] D. J. Bernstein. (2014). *CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness*. [Online]. Available: http://competitions.cr.yp.to/caesar.html

[11] F. Abed, C. Forler, and S. Lucks, "General overview of the first-round CAESAR candidates for authenticated encryption," IACR Cryptol. ePrint, Tech. Rep. 2014/792, 2014.

[12] S. Koteshwara and A. Das, "Comparative study of authenticated encryption targeting lightweight IoT applications," *IEEE Design Test*, vol. 34, no. 4, pp. 26–33, Aug. 2017.

[13] *ATHENA: Automated Tool for Hardware Evaluation*. Accessed: Feb. 7, 2019. [Online]. Available: https://cryptography.gmu.edu/athenadb/

[14] H. Böck, A. Zauner, S. Devlin, J. Somorovsky, and P. Jovanovic, "Nonce-disrespecting adversaries: Practical forgery attacks on GCM in TLS," in *Proc. USENIX WOOT*, 2016, pp. 1–11.

[15] T. Iwata and Y. Seurin, "Reconsidering the security bound of AES-GCM-SIV," *IACR Trans. Symmetric Cryptol.*, vol. 2017, no. 4, pp. 240–267, 2017.

[16] *Webpage for the AES-GCM-SIV Mode of Operation*. Accessed: Feb. 7, 2019. [Online]. Available: https://cyber.biu.ac.il/aes-gcm-siv/

[17] P. Rogaway and T. Shrimpton, "Deterministic authenticated-encryption," in *Advances in Cryptology—EUROCRYPT*, vol. 6. Springer, 2007.

[18] Deoxys v1.3. (2015). *Second-Round Submission to the CAESAR Competition*. [Online]. Available: http://competitions.cr.yp.to/-caesar-submissions.html.

[19] J. Jean, I. Nikolic, T. Peyrin, and Y. Seurin, "Deoxys v1. 41," Submitted to CAESAR, Oct. 2016.

[20] S. Gueron, A. Langley, and Y. Lindell, "AES-GCM-SIV: Specification and analysis," IACR Cryptol. ePrint Arch., Tech. Rep. 2017/168, 2017. [Online]. Available: https://eprint.iacr.org/2017/168
[21] F. Abed *et al.*, "Pipelineable on-line encryption," in *Proc. Int. Workshop Fast Softw. Encryption* Berlin, Germany: Springer, 2014, pp. 205–223.
[22] PRIMATEs v1.02. (Sep. 2014). CAESAR submission.
[23] K. K. Parhi, *VLSI Digital Signal Processing Systems: Design and Implementation*. New York, NY, USA: Wiley, 1999.
[24] D. Canright, "A very compact S-box for AES," in *Proc. Int. Workshop Cryptograph. Hardw. Embedded Syst.* Berlin, Germany: Springer, 2005, pp. 441–455.
[25] A. Satoh, S. Morioka, K. Takano, and S. Munetoh, "A compact Rijndael hardware architecture with S-box optimization," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur.* Berlin, Germany: Springer, 2001, pp. 239–254.
[26] A. Rudra, P. K. Dubey, C. S. Jutla, V. Kumar, J. R. Rao, and P. Rohatgi, "Efficient Rijndael encryption implementation with composite field arithmetic," in *Proc. Int. Workshop Cryptograph. Hardw. Embedded Syst.* Springer, 2001, pp. 171–184.
[27] X. Zhang and K. K. Parhi, "On the optimum constructions of composite field for the AES algorithm," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 53, no. 10, pp. 1153–1157, Oct. 2006.
[28] Z. Yuan, Y. Wang, J. Li, R. Li, and W. Zhao, "FPGA based optimization for masked AES implementation," in *Proc. IEEE 54th Int. Midwest Symp. Circuits Syst. (MWSCAS*, Aug. 2011, pp. 1–4.
[29] P. Maistri, S. Tiran, P. Maurine, I. Koren, and R. Leveugle, "Countermeasures against EM analysis for a secured FPGA-based AES implementation," in *Proc. Int. Conf. Reconfigurable Comput. FPGAs (ReConFig)*, 2013, pp. 1–6.
[30] C. Dobraunig, M. Eichlseder, S. Mangard, F. Mendel, and T. Unterluggauer, "ISAP–towards side-channel secure authenticated encryption," *IACR Trans. Symmetric Cryptol.*, vol. 2017, no. 1, pp. 80–105, 2017.

Sandhya Koteshwara (S'16) received the B.E. degree in electronics and communication from Visvesvaraya Technological University, Belgaum, India, in 2010, and the M.S. degree in electrical engineering from the University of Minnesota, Minneapolis, MN, USA, in 2014, where she is currently working toward the Ph.D. degree at the Department of Electrical and Computer Engineering.

Her current research interests include hardware security, low power architectures for cryptographic algorithms, and approximate computing.

Amitabh Das (M'09–SM'17) received the Ph.D. degree in hardware security and cryptography from the Computer Security and Industrial Cryptography Research Group, Electrical Engineering Department, KU Leuven, Leuven, Belgium.

He was a Security Researcher with the Security Center of Excellence Group, Intel Corporation, Hillsboro, OR, USA, where he is currently a Research Scientist at Intel Labs, Security and Privacy Research. He has authored or coauthored several peer-reviewed international IEEE/ACM/Springer conference and journal papers in the area of hardware security and cryptography. His current research interests include hardware cryptography, field-programmable gate array security, memory security, system-on-chip hardware security, embedded security, and hardware/software codesign of cryptographic algorithms.

Dr. Das serves as a reviewer for several IEEE and Springer transactions, journals, and conferences. He also serves as a Technical Program Committee Member for IEEE Internal Verification and Security Workshop and an industry Liaison and Technical Advisory Board Member for Semiconductor Research Corporation General Research Collaboration Trustworthy and Secure Semiconductors and Systems effort.

Keshab K. Parhi (S'85–M'88–SM'91–F'96) received the B.Tech. degree from IIT, Kharagpur, India, in 1982, the M.S.E.E. degree from the University of Pennsylvania, Philadelphia, PA, USA, in 1984, and the Ph.D. degree from the University of California at Berkeley, Berkeley, CA, USA, in 1988.

Since 1988, he has been with the University of Minnesota, Minneapolis, MN, USA, where he is currently a Distinguished McKnight University Professor and Edgar F. Johnson Professor at the Department of Electrical and Computer Engineering. He is involved in intelligent classification of biomedical signals and images, for applications such as seizure prediction and detection, schizophrenia classification, biomarkers for mental disorders, brain connectivity, and diabetic retinopathy screening. He has authored or coauthored more than 600 papers, has invented or coinvented 29 patents, has authored the textbook *VLSI Digital Signal Processing Systems* (New York, NY, USA: Wiley, 1999), and coedited the reference book *Digital Signal Processing for Multimedia Systems* (Boca Raton, FL, USA: CRC Press, 1999). His current research interests include the VLSI architecture design and implementation of signal processing, communications and biomedical systems, error control coders and cryptography architectures, high-speed transceivers, stochastic computing, secure computing, and molecular computing.

Dr. Parhi served as a Board of Governors Elected Member of the IEEE Circuits and Systems Society from 2005 to 2007. He is a fellow of the AAAS. He was a recipient of numerous awards including the 1999 Golden Jubilee Medal, 2012 Charles A. Desoer Technical Achievement Award, and the 2017 Mac Van Valkenburg Award, from the IEEE Circuits and Systems Society, the 2001 IEEE W. R. G. Baker Prize Paper Award, the 2003 IEEE Kiyo Tomiyasu Technical Field Award, the 2004 F. E. Terman Award from the American Society of Engineering Education, the 2013 Distinguished Alumnus Award from IIT Kharagpur, and the 2013 Graduate/Professional Teaching Award from the University of Minnesota. He has served as the Technical Program Co-Chair for the 1995 IEEE VLSI Signal Processing Workshop and the 1996 Application Specific Systems, Architectures, and Processors conference, and as the General Chair for the 2002 IEEE Workshop on Signal Processing Systems. He has served on the Editorial Boards for the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS Part I And Part II, the IEEE TRANSACTIONS ON VLSI SYSTEMS, the IEEE TRANSACTIONS ON SIGNAL PROCESSING, IEEE SIGNAL PROCESSING LETTERS, and the *IEEE Signal Processing Magazine*. He served as the Editor-in-Chief for the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS Part I from 2004 to 2005. He currently serves on the Editorial Board of the *Journal of Signal Processing Systems* (Springer). He was the Distinguished Lecturer of the IEEE Circuits and Systems Society from 1996 to 1998.