Resource-Shared Crypto-Coprocessor of AES Enc/Dec With SHA-3

Dur-e-Shahwar Kundi, *Member, IEEE*, Ayesha Khalid, *Member, IEEE*, Arshad Aziz, Chenghua Wang, Máire O'Neill, *Senior Member, IEEE*, and Weiqiang Liu, *Senior Member, IEEE*

***Abstract*— Cryptographic co-processors are integral to the modern System-on-Chips. Flexibility in such designs serves dual purpose, i.e., it enables acceleration of different essential cryptographic primitives (Encryption/Authentication/Pseudo Random Number Generation (PRNG)) and also results in design compaction via resource sharing. In this context, a novel resource-shared crypto-coprocessor, named *AE$HA-3* is presented, which combines two National Institute of Standards and Technology (NIST) standardized algorithms, i.e., Advance Encryption Standard (AES) and Secure Hash Algorithm-3 (SHA-3). Due to algorithmic dissimilarities, so far no resource-shared implementation enabling AES key scheduling/ enc/dec and SHA-3 has been presented. *AE$HA-3* exploits resource-sharing for area reduction, i.e., integration of Look-Up-Tables *(I-Tables)* for AES enc/dec; logical optimization of Six Input Equation *(SixIE)* for SHA-3; a *Unified XOR Section* to carry out both key whitening in AES and SHA-3 transformations. Furthermore, the AES key scheduling was performed using the same resource-shared hardware. The proposed *AE$HA-3* on Xilinx Virtex FPGA family results in highest hardware efficiency in terms of Throughput per Slice (TPS), along with a 49.37% area consumption reduction, when compared against the smallest stand-alone implementations presented to date.**

***Index Terms*— Advance encryption standard (AES), cryptography, embedded system, secure hash algorithm-3 (SHA-3), cryptographic accelerator.**

I. Introduction

SECURITY is essential to today's digital systems. To enable multi-gigabit communication bandwidths, cryptographic *accelerators* (Application Specific Integrated Circuits (ASICs)) are increasingly used for public key exchange, authentication, encryption etc. However, orthogonal to the performance offered by these dedicated ASICs, another critically required feature is the design *flexibility*. Flexible accelerators, with a unified hardware architecture supporting various versions/types/classes of cryptographic primitives, is a well established practice with various significant works reported in literature [1]–[3]. We list some advantages of flexible crypto-copressors, taken up on ASICs/Field Programmable Gate Arrays (FPGAs) below.

Firstly, they provide a common implementation supporting various cryptographic operations as required by most (if not all) communication standards/protocols. With the rapid development of communication technologies such as Internet-of-Things (IoTs), Wireless Sensor Networks (WSNs), Software Defined Networking (SDN) etc., security is becoming an integral part of these systems. That in turn demands the implementation and adoption of variety of cryptographic primitives and protocols for multi-purpose security services such as confidentiality, integrity, authentication as well as resistance to quantum computing threats. Examples include Global System for Mobile (GSM), 3rd Generation Partnership Project (3GPP), Transport Layer Security (TLS)/Secure Sockets Layer (SSL), bluetooth, IEEE 802.11, ISO/IEC 29192 etc, which recommend usage of cryptographic algorithms belonging to classes of stream ciphers, block ciphers, Message Authentication Codes (MAC) as well as Public Key Cryptography (PKC). A novice approach to undertake a separate hardware realization for each of these security services will result in a high resource and power-hungry design. Instead, a resource-shared design enabling multi-purpose cryptographic solutions is more efficient, attractive especially to the systems with area and power constraints.

Secondly, flexible cryptographic implementations can continuously protect against evolving cryptanalytic vulnerabilities by switching to appropriate security levels or an alternate algorithm out of the multiple choices available. The impending realization of a scalable Quantum computer is an immediate threat to the currently deployed security infrastructure. Quantum algorithms will reduce the security strengths of today's Rivest, Shamir and Adelman (RSA) and Elliptic Curve Cryptography (ECC) families of PKC to zero by virtue of Shor's algorithm [4], while the Grover's search algorithm [5] reduces the complexity of the search space of a quadratic brute-force attack to half for symmetric key cryptography (i.e., AES-128 and SHA3-256 will result in marginal security strength of 64-bits contrary to 128-bits). Consequently, National Institute of Standards and Technology (NIST) called for Post Quantum Cryptography (PQC) competition in 2016 for the standardization of quantum-resilient algorithms for

Manuscript received February 15, 2020; revised May 4, 2020; accepted May 22, 2020. Date of publication June 3, 2020; date of current version December 1, 2020. This work was supported in part by the National Natural Science Foundation of China under Grant 61871216 and Grant 61771239, in part by the Fundamental Research Funds for the Central Universities China under Grant NE2019102 and Grant NG2020001, and in part by the Six Talent Peaks Project in Jiangsu Province under Grant 2018-XYDXX-009. This article was recommended by Associate Editor M. Mozaffari Kermani. *(Corresponding author: Weiqiang Liu.)*

Dur-e-Shahwar Kundi, Chenghua Wang, and Weiqiang Liu are with the College of Electronic and Information Engineering, Nanjing University of Aeronautics and Astronautics, Nanjing 211100, China (e-mail: dureshahwar@nuaa.edu.cn; chwang@nuaa.edu.cn; liuweiqiang@nuaa.edu.cn).

Ayesha Khalid and Máire O'Neill are with the Centre for Secure Information Technologies (CSIT), Queen's University Belfast, Belfast BT7 iNN, U.K. (e-mail: a.khalid@qub.ac.uk; m.oneill@ecit.qub.ac.uk).

Arshad Aziz is with the National University of Sciences and Technology (NUST), Karachi 24090, Pakistan (e-mail: arshad@pnec.nust.edu.pk).

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

Digital Object Identifier 10.1109/TCSI.2020.2997916

the PKC [6], while NSA's Information Assurance Directorate (IAD) announced a transition to Suite B of doubled up keys for symmetric key cryptography (including AES [7], SHA-3 [8] algorithms) [9].

Thirdly, a common hardware implementation of different algorithms, if required to execute exclusively, opens door to the possibility of clever resource sharing, often enabling drastic area/power reductions in hardware. Compared to multiple distinct cores, a clever unified design reusing the macro-blocks of individual algorithms based on their structural similarities has been shown to be more efficient [10]. *Finally*, the area reduction generally comes in hand with power requirement reduction, that is very welcome for the IoT applications [11]. Throughput/security requirements may also often require systems to switch between different versions of the same algorithm.

This work presents a unified FPGA based hardware design of a cryptographic core called *AE$HA-3* that is capable of performing two NIST standards (AES+SHA-3). The major contributions of our work are as follows:

1) A resource-shared architecture of AES encryption (enc) & decryption (dec) with SHA-3 is undertaken for providing multi-purpose cryptographic services (confidentiality, integrity, Authenticated Encryption (AE) etc.) as well as being utilized by PQC algorithms. To the best of our knowledge, such novel unified architecture endeavor has never been undertaken before, due to apparent structural dissimilarities in AES [12] and SHA-3 [13] designs, even a unified implementation of both the AES enc & dec processes is not a popular one due to their asymmetric architecture.
2) We undertake the idea of aggressively exploiting the macro block reuse within the two NIST standards to present detailed designs of: the *Base Module* (suitable for IoT applications) and *AE$HA-3* (for high performance, multi-gigabit applications). The *Base Module* accommodates a single AES-128 key scheduling, enc & dec core with Keccak-f[400], while *AE$HA-3* undertakes Keccak-$f$[1600] (SHA-3) with four AES-128 enc & dec cores.
3) A thorough benchmarking of *AE$HA-3* (on FPGAs) is carried out, comparing it with both the standalone algorithm implementations as well as unified designs undertaking block ciphers and hash functions. The proposed *AE$HA-3* on Xilinx Virtex FPGA family boosts the highest hardware efficiency in terms of Throughput per Slice (TPS), along with a 49.37% area consumption reduction.

The rest of the paper is organized as follows. The Section II provides the background including literature review of available resource-shared architectures. Section III presents our proposed resource-shared *AE$HA-3* design, followed by the detailed design description of *Base Module*, AES Key Scheduler and its scalability to incorporate SHA-3 with AES-128. Section IV benchmarks the implementation results and comparisons. Finally, conclusions are drawn in Section V.

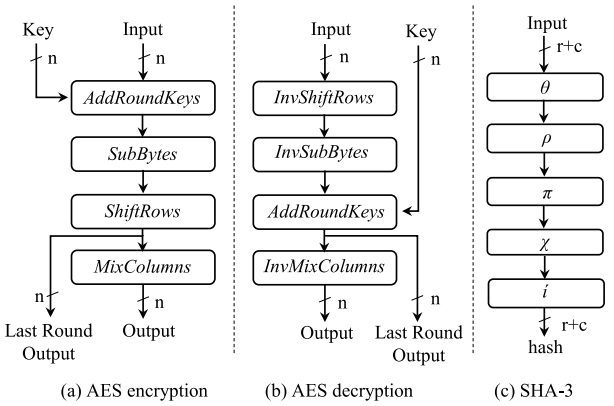

Fig. 1. Round function of AES enc/dec & SHA-3.

II. Background

A. Advance Encryption Standard (AES)

AES [7] is a Federal Information Processing Standard (FIPS) approved cryptographic algorithm that is most widely used encryption algorithms for the secrecy of information. It is a symmetric block cipher that supports data block of 128-bit with a variable key sizes of 128, 192 and 256 bits. The input data in AES, is arranged in a two dimensional array of 4×4 bytes called a *State*, consisting of 16 bytes in total. It uses a round function that is composed of four different byte-oriented transformations as shown in Fig. 1(a) & (b), where both enc & dec processes involve several asymmetries. The number of rounds to be executed depend on the length of the key size.

SubBytes is a non-linear byte substitution step that operates independently on each byte of the *State* using a substitution table. This substitution table which is invertible, is derived by the application of two transformations: First taking multiplicative inverse (MI) in GF(2^8) with element 00 being mapped to itself and then applying an affine transform (AF) over GF(2). The inverse transformation of this step during decryption process is *InvSubBytes*.

ShiftRows is a transposition step that shifts cyclically the last three rows of the AES state with a certain number of bytes towards left. The inverse of this during decryption process is *InvShiftRows* in which last three rows are shifted cyclically towards right.

MixColumns is a permutation step that operates on every column of the state. Each column is considered as a four-term polynomial over GF(2^8) and multiplied modulo $x^4 + 1$ with fixed polynomial $c(x)=\{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$. As a result of this multiplication, four bytes in a column are replaced by the following equations:

$$\begin{aligned} b'_{0,c} &= (\{02\} \bullet b_{0,c}) \oplus (\{03\} \bullet b_{1,c}) \oplus b_{2,c} \oplus b_{3,c} \\ b'_{1,c} &= b_{0,c} \oplus (\{02\} \bullet b_{1,c}) \oplus (\{03\} \bullet b_{2,c}) \oplus b_{3,c} \\ b'_{2,c} &= b_{0,c} \oplus b_{1,c} \oplus (\{02\} \bullet b_{2,c}) \oplus (\{03\} \bullet b_{3,c}) \\ b'_{3,c} &= (\{03\} \bullet b_{0,c}) \oplus b_{1,c} \oplus b_{2,c} \oplus (\{02\} \bullet b_{3,c}) \end{aligned} \quad (1)$$

In this step during the encryption each byte; $b_{i,c}$ is multiplied by constants; 01, 02, & 03 while in *InvMixColumns* step during the decryption process, each byte is multiplied by constants; 09, 0b, 0d & 0e.

AddRoundKeys performs an addition (bitwise XOR) of the state with RoundKey and each RoundKey is derived from input cipher Key using a Key Scheduler.

B. Secure Hash Algorithm-3 (SHA-3)

SHA-3 is a standard hash algorithm, announced by NIST in 2012 [14]. It is based on the 1600-bit instance of Keccak algorithm [8]. The permutation function of Keccak-f[b] as defined by its authors, operates on a fixed length of string called the width of permutation 'b' that can be chosen from a set of seven transformations {25, 50, 100, 200, 400, 800, 1600}. Further, 'b' is the sum of two parameters; rate 'r' & capacity 'c' and equals to $(r + c)$-bit. The value of 'c' is to be selected twice of the desired length of hash digest 'h'-bit,i.e., $c = 2h$. The input state of Keccak-f[b] is arranged in a three dimensional matrix of $5 \times 5 \times w$, where 'w' defines length of lane and equals to '$b/25$'. It also uses a round function that consists of five step mappings $(\theta, \rho, \pi, \chi, \iota)$ as shown in Fig. 1 (c), having totally different algorithmic structure from AES. The number of rounds to be executed is calculated by $12+2(log_2(b/25))$.

Theta (θ) Step: $(0 \le x, y \le 4, 0 \le z < w)$

$$C[x, y, z] = A[x, 0, z] \oplus A[x, 1, z] \oplus A[x, 2, z] \oplus A[x, 3, z] \oplus A[x, 4, z]; \quad (2)$$

$$D[x, z] = C[(x - 1), z] \oplus ROT(C[(x + 1), 1]); \quad (3)$$

$$A'[x, y, z] = A[x, y, z] \oplus D[x, z]; \quad (4)$$

Rho (ρ) Step: $(0 \le x, y \le 4)$

$$A[x, y, z] = ROT(A'[x, y, z], r[x, y]); \quad (5)$$

Pi (π) Step: $(0 \le x, y \le 4, 0 \le z < w)$

$$B[y, (2x + 3y), z] = A[x, y, z]; \quad (6)$$

Chi (χ) Step: $(0 \le x, y \le 4, 0 \le z < w)$

$$A[x, y, z] = B[x, y, z] \oplus (NOT(B[(x + 1), y, z]) \; AND(B[(x + 2), y, z])); \quad (7)$$

Iota (ι) Step: $(0 \le z < w)$

$$A'[0, 0, z] = A[0, 0, z] \oplus RC[z]; \quad (8)$$

The *A[x, y, z]* represents a particular lane of a state while *B[x, y]*, *C[x]* and *D[x]* are the intermediate results. The XOR ($\oplus$), *NOT* and *AND* are the bitwise logical operations while *ROT* is bitwise cyclic shift operator with value of *r[x, y]* and *RC* is the round constant.

C. Resource-Shared Architectures

The idea of integrated implementations of multiple cryptographic primitives into a single architecture, for area efficiency has been extensively taken up in scientific literature. We survey different approaches taken so far in this context, in an informal categorization.

A simplistic solution for a flexible hardware system is *a general purpose processor core with instruction set architecture (ISA) customization*, specific to all the individual security kernels needed to be supported. This approach, however straightforward, will generally fail to compete with the custom designed cryptographic ASICs, in terms of acceleration achieved. Noticeable examples include CryptoManiac [15], Cryptoraptor [16], ECC processors [17], [18], etc. The macro block commonalities were exploited in ARX based cryptographic kernels as Coarse-Grained Reconfigurable Architecture (CGRA) on FPGAs to achieve remarkable area saving [1].

The basic building blocks for major families of cryptography primitives include a very small subset of operations (used under different configurations), e.g., permutation, substitution, key whitening operations used in most of the block ciphers. These commonalities have been exploited in rapid prototyping by high level synthesis tools for block ciphers [19], stream ciphers [20], etc. *Unified resource shared architectures have been proposed for several cryptographic hashes/several block ciphers, or a combination of both*. Several unified cryptographic hashes hardware designs have been reported so far: MD-5+SHA-1 was provided in [21], [22], MD5+RIPEMD-160 was proposed in [23] while a joint multi-hash core of SHA-1+MD-5 +RIPEMD-160 was reported in [24]. A multi-purpose AES core supporting all three key lengths and various modes of operation such as Cipher Block Chaining (CBC), Electronic Code Book (ECB) was reported in [25], [26] whereas a joint accelerator for the symmetric key ciphers such as AES/SMS4/Camellia was provided in [27].

A unified architecture providing the security solution both of integrity and confidentiality has much more practical importance. Many implementations took up a NIST standard (or some prominent finalists) for a resource-shared area-efficient design, examples include ChaCha stream cipher + BLAKE (SHA-3 finalist) in [28], AES + ECHO and Fugue (SHA-3 round 2 finalists) in [29] and [30], respectively, AES + Grøstl (SHA-3 finalist) in [30]–[35]. The BLAKE algorithm was inspired by ChaCha stream cipher while ECHO, Fugue and Grøstl were highly inspired by AES. These hash algorithms either shared the common structure/features with these encryption algorithms or utilized them as a subroutine. For example, ECHO uses AES directly as a subroutine, therefore supports plain AES with negligible overhead [29] while Grøstl and Fugue shared the common structure such as SubBytes transform of AES without using it directly [30]–[35]. Therefore, these algorithms offered a unique opportunity to design a unified resource-shared implementation with AES and consequently provided different cryptographic services by a single hardware with significantly fewer resources and efficient performance. However, all these designs [29]–[35] were based on non-standard hash functions such as BLAKE, ECHO, Fugue and Grøstl for providing integrity.

III. The Proposed *AE$HA-3* Design

The design and optimization of *AE$HA-3*, for an efficient FPGA based resource-shared architecture of AES enc/dec with standard hash function; SHA-3 was undertaken in following steps.

1) First a combined AES enc & dec core is implemented as shown in Fig. 2. Initially the ShiftRows & InvShiftRows

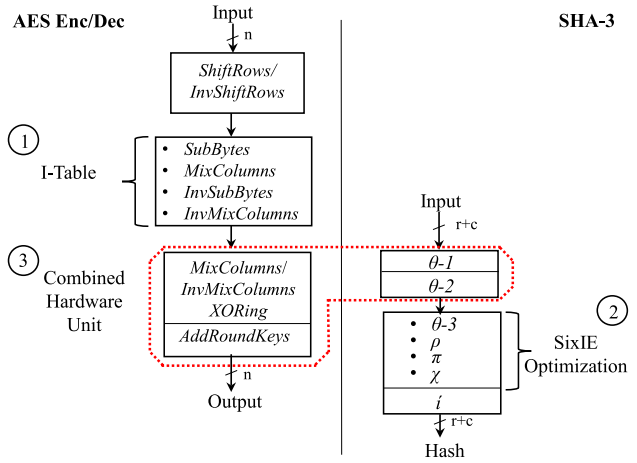

Fig. 2. Shared round function of AES Enc & Dec with SHA-3.

transforms are re-arranged and mixed together, then we design an Integrated-Table (**I-Table**) (point 1 in Fig. 2) in which the most computationally intensive transformations; SubBytes, MixColumns & InvSubBytes, InvMixColumns of respective AES enc & dec are combined together in the form of single unified Look-Up-Table, targeting Block RAMs (BRAMs) of FPGA having 9× better security against Differential Power Analysis (DPA) than LUTs [36].

2) Then we apply the logical optimization technique on SHA-3 algorithm named as Six Input Equation (**SixIE**) optimization (point 2 in Fig. 2). In which the selected step mappings (θ-3, ρ, π & χ excluding ι) of SHA-3 are logically grouped into a single equation comprising of six inputs; targeting LUT-6 technology of Xilinx FPGAs but can also be implemented using 8-input Adaptive LUT (ALUT) by Intel Altera FPGAs [37]. Whereas, the first two equations of θ transform, i.e., θ-1 & θ-2 are kept as it is.
3) We now undertake the architectural commonalities between AES enc/dec and SHA-3 for designing a resource-shared hardware unit as indicated by point 3 in Fig. 2. These are MixColumns/InvMixColumns XORing (1) & AddRoundKeys transform of AES enc/dec processes and the first two equations of θ transform of SHA-3, i.e., θ-1 (2) & θ-2 (3).

The architecture is first elaborated with the help of *Base Module*, in which a basic variant of Keccak algorithm, i.e., Keccak-f[400] is selected to incorporate with a single core of AES-128 enc/dec. The Keccak-f[400] has the internal state-size of 400-bit with 'r' and 'c' of 144 & 256-bits, respectively, as defined in [38]. With this *capacity*, it produces the message digest of 128-bit, i.e., c/2 similar to that of AES-128 state. The detailed diagram of our proposed *Base Module* is shown in Fig. 3(a), which comprises of three main sections. A single-unit **Look-Up-Table Section** to carry out AES enc & dec look-up operations. Resource-shared **Unified XOR Section** to realize the AES MixColumns/InvMixColumns *XORing* & AddRoundKeys transformations or SHA-3 θ-1 & θ-2 transformations. The last section **SixIE Network** to implement the SHA-3 integrated equation by a single hardware.

The input data-path (*In*) for the *Base Module* is selected as 128-bit according to the AES-128 while for Keccak-f[400], it is concatenated with additional 16-bit zeros to make 144-bit of 'r'. The resulting r-bit are then further concatenated to c-bit of zeros (representing the initializing phase) to form the internal data-path of 400-bit ('$r + c$'). In case of AES, the 128-bit input (*In*) to the proposed design is first *XORed* with the cipher *Key* and then applied to the proceeding modules, while in case of Keccak, there is no cipher *Key* hence assigned with zero input values. The design has two controlling input signals; *Enc/Dec* & *AES/Keccak* on the basis of which a selection is made to calculate either AES enc, AES dec or Keccak hash output. The *Enc/Dec* controls the selection between the corresponding enc and dec tables for the AES in **Look-Up-Table Section** while *AES/Keccak* selects between the AES and Keccak data-paths for the **Unified XOR Section**, as depicted in Fig. 3(b).

The data-path for AES enc & dec is highlighted by Fig. 3(c); it first goes through **Look-Up-Table Section**, where corresponding to each byte 32-bit encrypted or decrypted output is produced by the *I-Table* core on the basis of *Enc/Dec* and *T/S* (representing *T-box/S-box* Tables) signals. For complete 128-bit input we get an output of 512-bit, the upper half 256-bit are concatenated with upper half 64-bit *Round Key* while lower half 256-bit are concatenated with lower half 64-bit *Round Key* to make 320-bit each. The resulting 320-bit are applied in parallel to *N1* & *N2* XOR networks of **Unified XOR Section** as shown in Fig. 3. The AES input to each XOR network (*N1* & *N2*) is 320-bit and corresponding to each input it generates 64-bit output. The combination of both 64-bit output from XOR network in proper MSB and LSB order forms the final 128-bit AES enc/dec output.

Similarly, the data-path for Keccak is represented by Fig. 3(d); the 400-bit input goes into the **Unified XOR Section**, where it is processed sequentially. It first goes into *N1* XOR network and then after passing from *Padding* module it is applied to the *N2* XOR network. In case of Keccak we need *Padding* module in order to make the input of required length for *N2* XOR network by padding extra zeros because *N1* produces 80-bit Keccak output corresponding to 400-bit input. The 80-bit from **Unified XOR Section** along with this 400-bit input are now provided to **SixIE Network** that finally produce the 400-bit round output of Keccak. Out of this 400-bit, corresponding 128-bit hash output is selected after the last round. The **Unified XOR Section** is shared between AES enc & dec processes and Keccak-f[400]; therefore, corresponding input data-path to this section is selected through *AES/Keccak* signal as shown in Fig. 3(a).

The *Mux-1* and *Mux-2* as shown in Fig. 3(a), are the feedback multiplexers for AES and Keccak, respectively. Both the multiplexers at '0' select initial input to circuit while at '1' select the feedback input for the calculation of remaining rounds for each AES enc/dec and Keccak. AES-128 has total of 10 rounds while Keccak-f[400] has 20 rounds to produce the final output [7], [8].

A. Look-Up-Table Section

The enc & dec processes in AES algorithm comprise of totally different sequence of transformations and also different

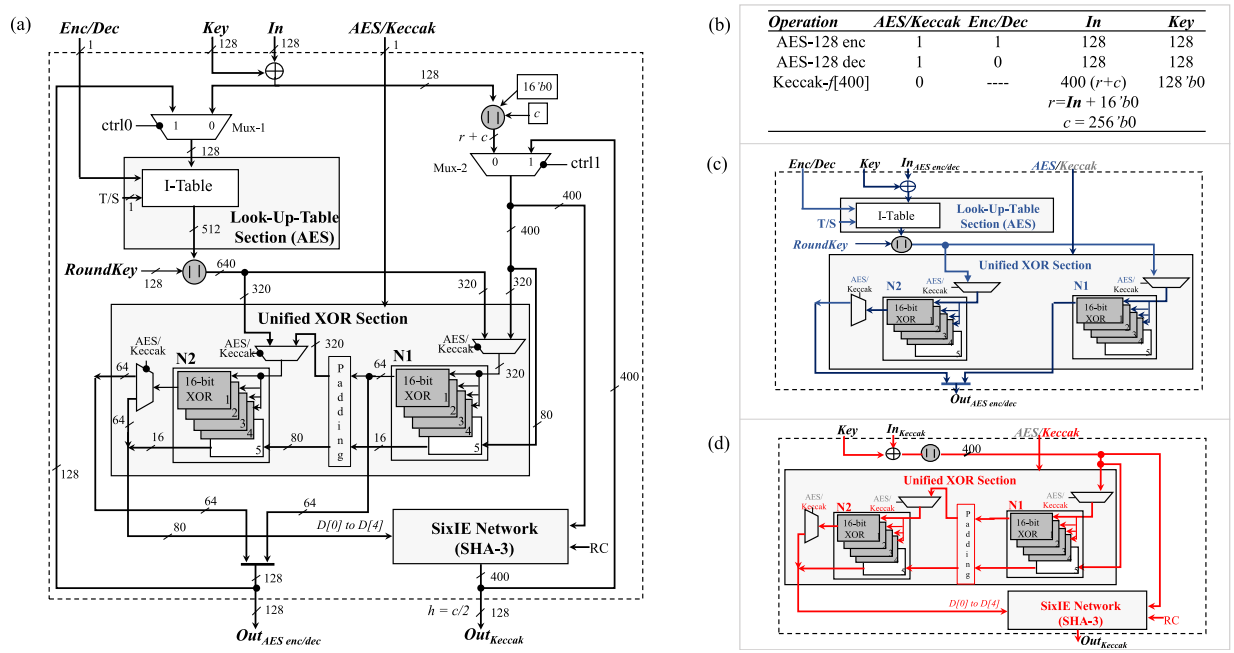

Fig. 3. (a) Proposed *Base Module* of AES-128 enc/dec and Keccak-*f* [400] (b) Inputs and control signals (c) AES data-path (d) Keccak data-path.

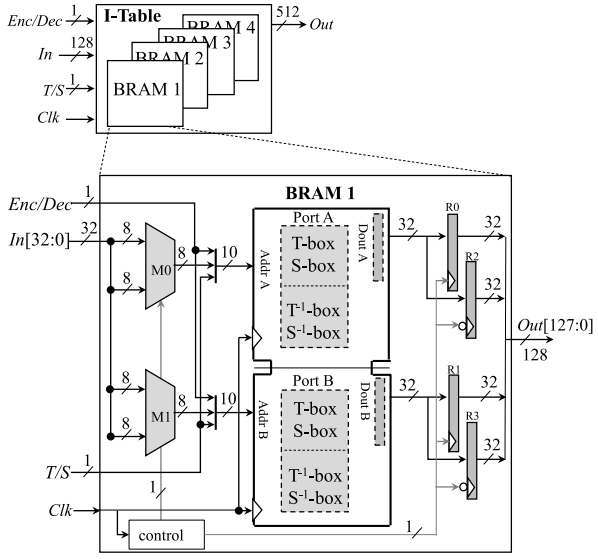

Fig. 4. Internal architecture of Look-Up-Table section.

set of multiplication constant; (*01*, *02*, *03*) during the MixColumns transform of encryption and (*09*, *0b*, *0d*, *0e*) during InvMixColumns step of decryption process. Therefore, this results in the requirement of two separate and unshared hardware unit along with different set of XOR networks for both the enc & dec data-paths [39]. To incorporate this asymmetry, we design a **Look-Up-Table Section** as a single-unit hardware that processes 128-bit AES input data either for enc or dec and at the same requires a single XOR network for both the processes.

This section consists of a single *I-Table* core that is implemented using four 32Kb Block RAMs (BRAMs) in a dual-port mode to process four 32-byte AES columns as shown in Fig. 4. Each 32Kb BRAM is configured with a unified Look-Up-Table comprising of all enc & dec multiplication constants in the form of T-box/S-box and T^{-1}-box/S^{-1}-box tables where T-box represents the combined Look-Up-Table for SubBytes and MixColumns transforms while S-box is the Look-Up-Table for the SubBytes only. During the first 9 rounds of AES encryption process, we require T-box with (*01*, *02*, *03*) multiplication constants whereas during the first 9 rounds of AES decryption process, we require T^{-1}-box with (*09*, *0b*, *0d*, *0e*) multiplication constants [40]. Whereas, in the last round of AES enc/dec processes, we require S-box/ S^{-1}-box tables, respectively, instead of T-box and T^{-1}-box tables.

To store all enc & dec multiplication constants and substitution values in a single BRAM, same aspect ratio, i.e., 32-bit width of BRAM is defined. Based on this each entry is adjusted to 32-bit format as the maximum entry in all these four tables is of 32-bit. This arrangement now transforms each table size into 8Kb. Further, to re-use the same XOR network during both the enc & dec processes, the multiplication constants in all these four tables are re-arranged. This arrangement is done in such a way that 32-bit output from BRAM corresponding to each 8-bit input during enc/dec follows the same symmetry for the next section. For T-box it is arranged in (*02*, *01*, *01*, *03*) order, for S-box (*01*, *00*, *00*, *00*) order, for T^{-1}-box (*0e*, *09*, *0d*, *0b*) order and for S^{-1}-box (*01*, *00*, *00*, *00*) order.

To access these four 8Kb tables from a single 32Kb BRAM, 10-bit addressing is used that is formed by using two 1-bit signals (*Enc/Dec*, *T/S*) as MSB in addition to 8-bit input data; *Addr[9:0]* = {*Enc/Dec*, *T/S*, *Din*}. The MSB *Enc/Dec*-bit selects between enc & dec Look-Up-Tables (i.e., T-box/S-box or T^{-1}-box/S^{-1}-box), respectively, whereas *T/S*-bit selects between the two tables, i.e., it selects the T-box during the first nine rounds of the encryption process and then selects S-box in its final round. Similarly during the decryption process, it selects the T^{-1}-box for the first nine rounds and then selects S^{-1}-box in the final round.

The input state of AES consists of four 32-bit columns and each column is applied to a single BRAM where it is processed in a time-multiplexed fashion by using multiplexers *M0* & *M1* as shown in Fig. 4. For selection, 1-bit counter as a control signal being synchronized with the main clock (*Clk*) is utilized to select the input bytes at multiplexers port and also to select the appropriate positive and negative-edge triggered registers at the output. At control bit '0', *M0* & *M1* select the first byte and applied it to address port of BRAM (*Addr A* & *Addr B*). The BRAM at positive-edge of *Clk* produces the desire 32-bit output at its *Dout A* & *Dout B* ports after clock-to-out time. These outputs are then stored in selected positive-edge triggered register (*R0* & *R1*) respectively. Similarly At control bit '1', the *M0* & *M1* select the next two bytes and are again applied to BRAM. The corresponding outputs are now stored in negative-edge triggered registers (*R2* & *R3*). To further enhance the throughput and performance of BRAMs, internal embedded registers (indicated by dash lines in Fig. 4) in combination with external registers are enabled at free of cost to process single column of an AES state in a two-stage pipelined architecture that takes two clock cycles to produce 128-bit output.

For the complete processing of four 32-bit columns (128-bit input), four parallel BRAMs are utilized that produce 512-bit output as shown in Fig. 4. This 512-bit AES output is then fed to next module, i.e., **Unified XOR Section** where it is *XORed* to produce the final AES encrypted or decrypted output. Thus the integration of AES enc & dec look-up operations into single joint architecture and accessing it from BRAM in time multiplexed fashion result in saving of 75% of BRAMs resources and utilizes only 4 BRAMs otherwise 8 BRAMs will be required by AES encryptor and another 8 BRAMs for the decryptor [41]–[43].

B. Unified XOR Section

Conventionally, we require two 128-bit separate and unshared XOR networks for the XORing of T-box and T^{-1}-box output [39] and another two 128-bit XOR networks for the operation of AddRoundKeys during the AES enc & dec processes. Similarly in case of Keccak-f[400] variant, we also require two 80-bit XOR networks for θ-1 & θ-2 transformations. Thus all of these transformations require total of 672-bit XOR network with maximum of 5-input per XOR gate. Now if we consider a LUT-6 of Xilinx FPGA, the implementation of either one 5-input XOR gate or 2-input XOR gate will utilize one LUT primitive. Hence 672-bit XOR network results in the utilization of total 672 LUTs.

To reduce down these XORing requirement, we proposed an **Unified XOR Section** of 160-bit only that is shared between all the above AES enc & dec and Keccak transformations by taking the following considerations.

1. Rearrangement of T-box outputs: The original 32-bit T-box output of each byte in a AES column ($b_{0,c}$, $b_{1,c}$, $b_{2,c}$, $b_{3,c}$) is XORed in a defined sequence as given by (1) of MixColumns transform, in order to produce the final AES column output. While the lanes of Keccak (*A[x,0,z], A[x,1,z], A[x,2,z], A[x,3,z], A[x,4,z]*) are XORed in a different sequence

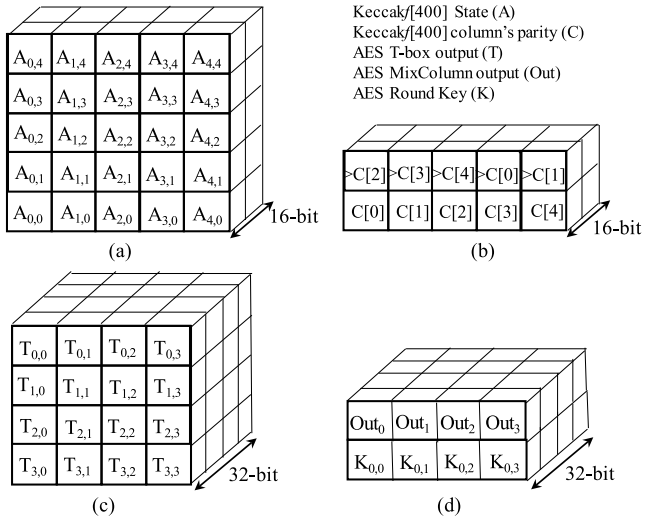

Fig. 5. (a) Keccak-f[400] input state for θ-1 and (b) Keccak-f[400] state for θ-2 (c) AES output state after T-box and (d) AES state for AddRoundKeys transform.

according to θ-1 transform as given by (2). Therefore, both required different XOR networks and to share a single XOR network between the two, we need modification in one of the sequence.

Therefore, 32-bit AES T-box output from BRAMs in **Look-Up-Table Section** before applying to this **Unified XOR Section** are rearranged in output registers *R0-R4* (as shown in Fig. 4) in a sequence as required by θ-1 transform of Keccak. The 32-bit T-box output from each BRAM during encryption is now stored in *R0* as (*02*, *01*, *01*,*03*) sequence, in *R2* as (*03*, *02*, *01*, *01*), in *R1* as (*01*, *03*, *02*, *01*) while in *R3* as (*01*, *01*, *03*, *02*) contrary to original sequence (*02*, *01*, *01*, *03*). This rearrangement of T-box outputs at the register level are done through the routing without utilizing any hardware resources and provides the opportunity to share the same XOR network with Keccak input.

2. Modification of AES and Keccak-f[400] State Matrices: Both the AES and Keccak have different sizes of state matrix that in turn required separate XOR network. In Keccak-f[400] variant, the input state is arranged in the form of 5×5 state matrix with each lane is of 16-bit wide while in AES after the **Look-Up-Table Section**, we received 4×4 state matrix with each lane is of 32-bit as shown in Fig. 5. The Fig. 5(a) represents the Keccak-f[400] state while Fig. 5(c) represents the AES state that have to XOR according to (2) & (1) respectively. Similarly we have different input states of Keccak-f[400] and AES for the calculation of θ-2 (3) and AddRoundKeys transformations as shown in Fig. 5 (b) & (d) respectively.

In order to mitigate the differences between the AES and Keccak states, we first modified the AES state matrix of Fig. 5(c). The 128-bit *Round Key* of Fig. 5(d) is concatenated with AES state, in such a way that it forms the fifth row of the AES state matrix being symmetric with fifth row of the Keccak state. Further, each 32-bit lanes of AES T-box state are divided into two 16-bit lanes in order to have symmetry with 16-bit lanes of the Keccak state.

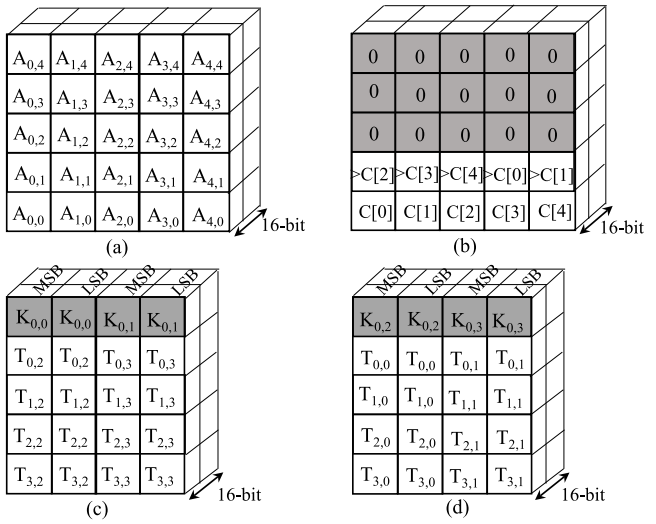

Fig. 6. (a) Keccak-f[400] input state for θ-1 and (b) modified Keccak-f[400] state for θ-2 (c,d) Modified AES states.

Similarly the Keccak state for θ-2 step as shown in Fig. 5(b) is also modified to have five rows. For this purpose it is padded with extra zeros by using the *Padding* module (as shown in Fig. 3(a)). The final modified input states of both AES and Keccak for our **Unified XOR Section** are shown in Fig. 6, where Fig. 6(a) represents the original Keccak input state while Fig. 6(b) represents the modified input state of Keccak for θ-2. Similarly Fig. 6(c & d) represent the modified AES state with 16-bit lanes that now provide us the opportunity to share the same XOR network between AES enc/dec and Keccak-f[400] and also the AddRoundKeys transform of AES enc/dec is realized by the same hardware.

Based on these finalized input states of Keccak-f[400] and AES, we therefore design our **Unified XOR Section**. It is implemented by two similar sets of XOR networks *N1* & *N2*, for both Keccak-f[400] and AES inputs. The AES data (either enc or dec) from **Look-Up-Table section** is applied to these XOR networks *N1* & *N2* simultaneously in the form of two streams to produce the final 128-bit AES output i.e *Out (AES enc/dec)* while Keccak input state is processed sequentially from *N1* to *N2* and 80-bit output of (3), i.e., *D[0]* to *D[4]* is produced. This Keccak output is then provided to **SixIE Network** for further processing. The selection between the input states for *N1* & *N2* are done through multiplexers using the control signal *AES/Keccak* as shown in Fig. 3(a).

Each XOR network (*N1* or *N2*) comprises of five 16-bit XOR instances for the respective five columns of Keccak state having lane size of 16-bit each. Out of these five, only four 16-bit XOR instances (as indicated by dark shade in Fig. 3(a)) are shared between the four columns of both AES enc/dec & Keccak states while the last 16-bit XOR instance processes only the fifth column of Keccak state. The input to XOR network *N1* is selected either from state in Fig. 6(a) or modified AES state in Fig. 6(c) in order to produce either column parity (*C[0]* to *C[4]*) of Keccak or LSB 64-bit (8 bytes) final output of AES enc/dec process respectively. Similarly the input to XOR network *N2* is selected either from the state in Fig. 6(b) or AES state in Fig. 6(d) to calculate either the output of (3) of Keccak-f[400]; *D[0]* to *D[4]* or final MSB 64-bit AES enc or dec output respectively.

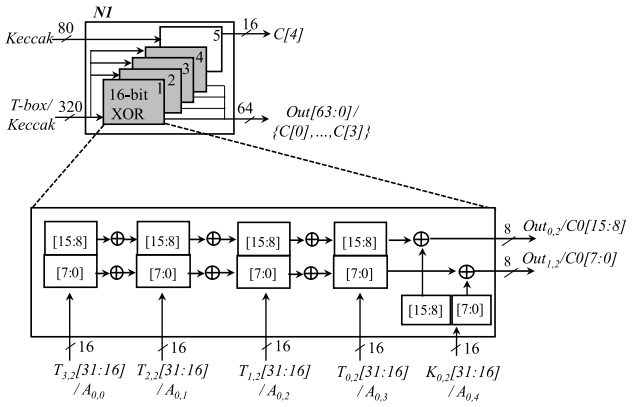

Fig. 7. Internal architecture of single 16-bit XOR instance.

The internal architecture of both *N1* and *N2* is same but both processes different set of inputs. For the simplicity a single 16-bit XOR instance # 1 of '*N1*' is elaborated in Fig. 7, in which the first column from Keccak-f[400] state ($A_{0,0}$, $A_{0,1}$, $A_{0,2}$, $A_{0,3}$, $A_{0,4}$) as shown in Fig. 6(a) or first MSB column from AES state ($T_{3,2}$, $T_{2,2}$, $T_{1,2}$, $T_{0,2}$, $K_{0,0}$) of Fig. 6(c) is processed. The network on the basis of *AES/SHA3* selection produces either first 16-bit column parity (*C[0]*) of Keccak or two bytes (16-bit) of AES output, i.e., $Out_{0,2}$, $Out_{1,2}$ respectively. All the other 16-bit XOR instances are same and in a similar way compute the output of either AES enc/dec or Keccak-f[400] transformations.

Thus our proposed resource-shared **Unified XOR Section** comprises of 160-bit single XOR hardware having two similar (80-bit + 80-bit) XOR networks *N1* & *N2*. This hardware is shared between AES enc/dec and Keccak transformations that utilized only 160 LUTs. Our resource-shared **Unified XOR Section** thus results in saving of 76.19% area resources as compared to 672 LUTs, that was conventionally required by all the transformations such as θ-1 & θ-2 of Keccak and MixColumns/InvMixColumns XORing & AddRoundKeys of AES enc & dec processes.

C. SixIE Network

The 80-bit Keccak output (*D[0]* to *D[4]*) from *N2* of **Unified XOR Section** is now provided to **SixIE Network** along with 400-bit input state (*A[x,y,z]*) according to (4). In this section the next four equations of Keccak from (4) to (7) are logically grouped into a single equation as given by (9) through our logical optimization technique.

$$\begin{aligned} A'[x,y,z] &= (ROT((A[x+3y,x,z] \oplus D[x+3y,z]), r[x+3y,x])) \\ &\quad \oplus (NOT(ROT((A[(x+1)+3y,(x+1),z] \\ &\quad \oplus D[(x+1)+3y,z]), r[(x+1)+3y,(x+1)])) \\ &\quad AND(ROT((A[(x+2)+3y,(x+2),z] \\ &\quad \oplus D[(x+2)+3y,z]), r[(x+2)+3y,(x+2)]))); \end{aligned} \quad (9)$$

In (9), *ROT* rotate number of bits of each lane (16-bit) by a particular offset (*r[x,y]*) on the basis of x & y. The values of *r[x,y]* for Keccak-f[400] are defined in [8]. Also the parameters x, y, z in (9) for Keccak-f[400] ranges from 0 $\leq x,y \leq 4$, $0 \leq z \leq 15$.

With the help of this single equation, we are now able to calculate the final output of any lane of Keccak directly there by accessing the six 16-bit inputs from both *A[x,y,z]* and *D[x,z]* depending on value of x & y. For example, to calculate the value of *A'[1,0]* lane, we put x=1 and y=0 in (9) with z ranges from 0 to 15, it directly selects the following inputs; (*A[1,1]*, *D[1]*), (*A[2,2]*, *D[2]*), and (*A[3,3]*, *D[3]*) with their respective offsets 12, 11 and 5 [8]. The resulted equation is as follows:

$$A'[x,y,z] = (ROT((A[1,1] \oplus D[1]),12)) \oplus (NOT(ROT((A[2,2] \oplus D[2]),11))AND(ROT((A[3,3] \oplus D[3]),5))); \quad (10)$$

This resulted six input equation (10) can easily be implemented by using LUT-6 of Xilinx FPGA that is why we apply this SixIE optimization to Keccak. They are available in all modern series of Xilinx FPGA such as Spartan-6, Virtex-5/6/7 etc. Each LUT-6 primitive is manually configured by INIT = 64'h60069ff99ff96006 as shown in Fig. 8, comprising of all the possible combinations of LUT inputs according to simplified representation of (10); *Out*=(*In1*⊕ *In2*)⊕ (*NOT*(*In3*⊕ *In4*) *AND* (*In5*⊕ *In6*).

To produce the final 400-bit Keccak output (*A'[x,y,z]*), 400 (25 × 16) such LUT-6 primitives are instantiated together for complete 25 16-bit lanes as shown in Fig. 8. The complete implementation of (9) for all values of *A[x,y,z]* and *D[x]* therefore utilized 400 LUT-6 primitives that incorporate θ-3, ρ, π & χ transformations of Keccak together. Further, the implementation of last step i.e ι given by (8) is also done in this same section that consumes 16 extra LUT-6 primitives as it is the XORing of 16-bit *RC* with the first lane of Keccak state, i.e., *A[0,0]* only.

However, individual implementation of θ-3 & χ will consume 400 LUT-6 and 400 LUT-6 primitives, respectively, whereas ρ and π steps will not utilize any area resources but they may result in increased critical paths. However the logical optimization of Keccak step mappings as proposed above, i.e., the **SixIE** optimization, utilized only 400 LUT-6 primitives. Furthermore, this proposed optimization also eliminates the requirement of 400-bit intermediate registers as needed before. This results in reduction of 50% of area resources as compared to the individual implementation of each step.

D. AES Key Scheduler

The ingredients of AES Key Scheduler, i.e., SubWord, RotWord and XORing etc. are almost the same as that of AES encryption, so the same resource-shared architecture of Fig. 3(a) is re-used to generate ten 128-bit *Round Keys* from 128-bit cipher *Key* ahead of enc/dec process and stored in 128-bit registers. The pre-computed *Round Keys* will remain same for the rest of AES enc/dec until the new cipher *Key* is initialized. It will always take 10 clock cycles to generate

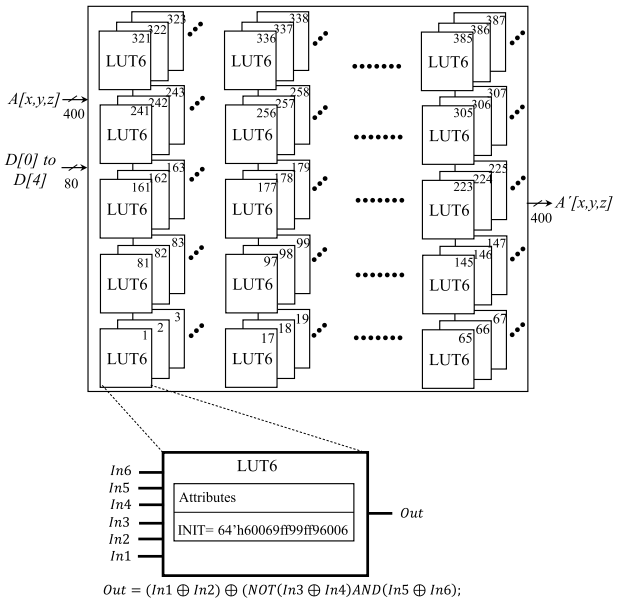

Fig. 8. Internal architecture of SixIE network.

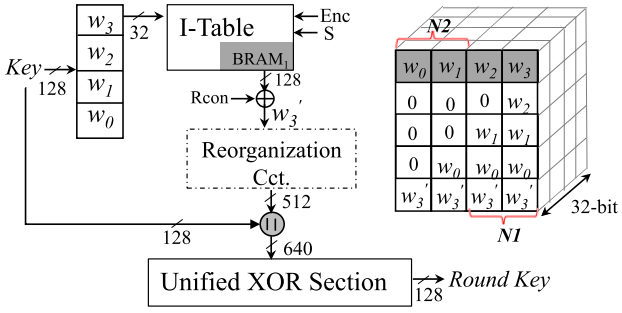

Fig. 9. AES key scheduler.

10 *Round Keys* after that AES enc/dec process starts with the performance being promised. The AES key scheduling process utilizing the same resource-shared hardware, along with extra circuitry to generate the state for **Unified XOR Section** is shown in Fig. 9. The reorganization circuitry and storing of 160 bytes of *Round Keys* cost just 30 Slices on Virtex-6 FPGA.

E. Scalability to AE$HA-3

So far we have presented a resource-shared architecture of Keccak-f[400] variant with a single core of AES-128 enc & dec to produce 128-bit hash digest or 128-bit AES encrypted/decrypted output. The proposed *Base Module* can be easily extend to SHA-3 that is Keccak-f[1600] variant of Keccak algorithm and can support either AES-192 or AES-256 depending upon the required security levels. For AES-192 and AES-256, the state-size/data-path will remain the same, i.e., 128-bit, only the number of keys as well as the rounds to be executed will change. Similarly for SHA3-256 and SHA3-512, the internal state-size of 1600-bit as well as number of rounds will remain the same, only the 'c'-bit as well as generated message digest will vary.

The main challenge was the difference between the internal state-sizes of both the algorithms such as 128-bit and 400-bit for AES-128 and Keccak-f[400], respectively. Whereas SHA-3

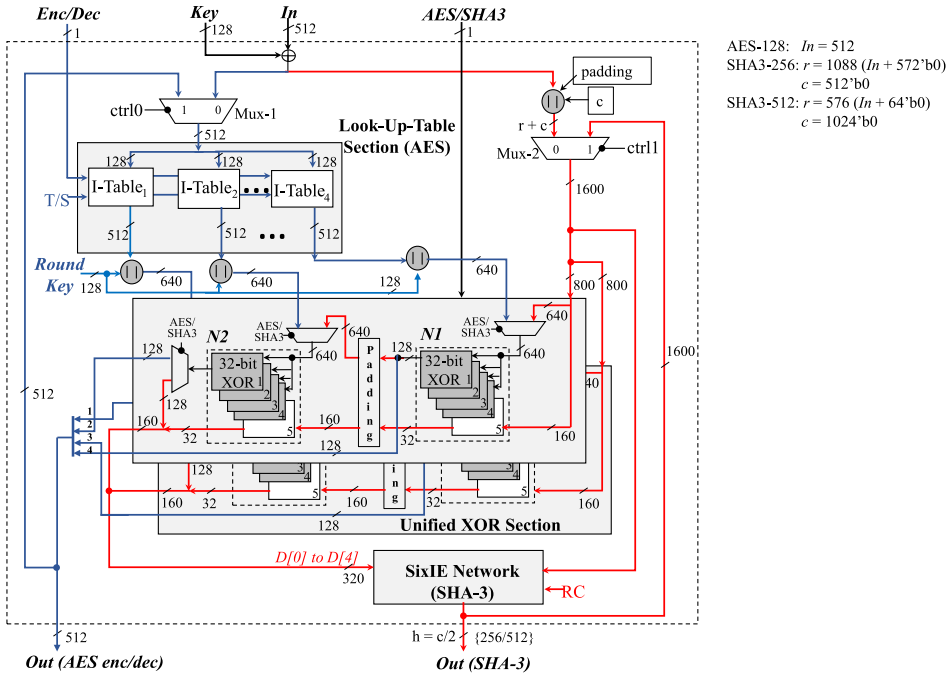

Fig. 10. *AE$HA-3*: Resource-shared architecture of AES Enc & Dec and SHA-3.

(Keccak-f[1600] variant) has four times the internal state-size of the basic module, i.e., Keccak-f[400]. To design a balanced architecture of AES-128 enc/dec with SHA-3 and to share the resources effectively, four parallel streams of AES-128 cores are accommodated with it.

The block diagram of resource-shared *AE$HA-3* design is shown in Fig. 10, the design is capable of producing either 512-bit of AES enc/dec output by processing four parallel block of 128-bit inputs or 256-bit/512-bit hash digest depending on the use of SHA-3 mode such as SHA3-256/SHA3-512 having *capacity c* being 512-bit/1024-bit, respectively. In both the SHA-3 modes, the c-bit, after concatenation with the input, makes the internal state-size of 1600-bit for the SHA-3.

The **Look-Up-Table Section** for the AES enc/dec now consists of four parallel I-Table cores (*I-Table*$_1$ to *I-Table*$_4$) each processing 128-bit of input data. The resulting 512-bit output from each I-Table cores is then concatenated to 128-bit *Round Key* before passing on to the **Unified XOR Section**. The final and modified input states of AES and SHA-3 for **Unified XOR Section** are shown in Fig. 11, where Fig. 11(a) and Fig. 11(b) represent the SHA-3 state for the θ-1 and θ-2, respectively, each having the lane size of 64-bit while Fig. 11(c) represents the four AES states each having 32-bit lanes. Based on these states, the **Unified XOR Section** is designed in such as way that two AES states are processed with half of SHA-3 states while other two AES states are processed with remaining half of SHA-3 states. Therefore, the **Unified XOR Section** of *AE$HA-3* consists of two identical sets of XOR networks each having *N1* & *N2* with five 32-bit XOR instances.

The working principle of this **Unified XOR Section** is also same to one explained in detail for the *Base Module*. This section either processes two I-Table states in parallel (Fig. 11(c)) or processes the 64-bit SHA-3 states (Fig. 11(a) & (b)) sequentially, depending on the control signal *AES/SHA3*.

For AES enc & dec, *N1* of the first set of XOR network produces the 128-bit output by processing the I-Table$_4$ data while *N2* produces 128-bit output by processing the I-Table$_2$ data that finally constitute to 256-bit AES output. Similarly for SHA-3 (using Keccak-f[1600]), *N1* of the first set of XOR network produces the 160-bit output of (2) by processing half of the SHA-3 state. The resulting 160-bit output after passing through the *Padding* module is then applied to *N2* that finally produces the first 160-bit output of (3). The second set of XOR network will produce the remaining 256-bit AES output and 160-bit θ-2 output of SHA-3, i.e., *D[0]* to *D[4]*.

The 320-bit output of θ-2 of SHA-3, i.e., *D[0]* to *D[4]* along with initial 1600-bit input state is then provided to 1600-bit **SixIE Network** that is four times replication of **SixIE Network** of *Base Module*. This network now produces 1600-bit SHA-3 output having the same logical optimized equation as given by (9). Here the rotational constant *ROT* and range of parameters x, y, z for (9) change according to the specification of Keccak-f[1600] with minimal hardware overhead. Again the ι step is also implemented within the same section.

Hence, scalability to AE$HA provides us the opportunity to support futuristic security applications (against quantum threats) since it enables higher security levels of the two NIST standards. Furthermore, several quantum-resilient algorithms in NIST PQC competition (e.g., CRYSTALS-Kyber) utilize both AES and SHA-3 (for Pseudo Random Number Generator (PRNG) or Extendable-output Function (XOF)) [44]

IV. Implementation Results & Analysis

The *Base Module* and the *AE$HA-3* designs were manually coded in Verilog HDL and tested using *Xilinx ISE Design Suit 14.7*. The design synthesis was carried our using the *Xilinx Synthesis Technology (XST)* on various FPGA families, results are presented for Virtex-6 (XC6VLX195T-3) and

TABLE I
IMPLEMENTATION RESULTS OF OUR PROPOSED RESOURCE-SHARED DESIGNS

Architecture	Cores	FPGA	Area (Slices + BRAM)	Equiv. Slices	Frq. (MHz)	Thr. (Mbps)
Base Module	Keccak-*f*[400] AES-128 enc/dec	Virtex-7	285 + 4*	797	376.88	2713.54
						2412.03
		Virtex-6	297 + 4*	809	332.98	2397.46
						2131.07
AE$HA-3	Keccak-*f*[1600] 4 × AES-128 enc/dec	Virtex-7	1250 + 16*	3298	364.26	16513.12
						9325.05
		Virtex-6	1380 + 16*	3428	328.15	14876.13
						8400.64

*1 BRAM equivalent to 128 Slices [42]

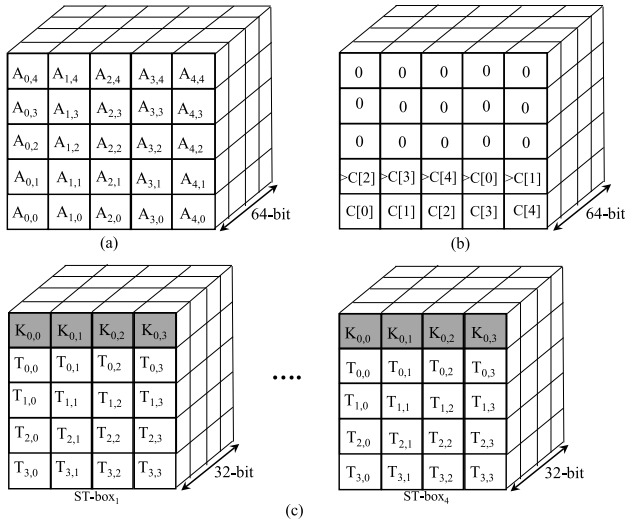

Fig. 11. (a) SHA-3 input state for θ-1 (b) SHA-3 input state for θ-2 (c) Modified AES output states after each I-Table.

Virtex-7 (XC7V585T-3) devices. Design implementation on FPGA devices followed all intermediate steps, i.e., mapping, translation and place & route (PAR) with no timing violations. Each design was tested and verified by ISim simulator with functional and post-PAR simulations, final occupied Slices and operational frequency are reported here after post-PAR step.

The implementation results of our proposed resource-shared architectures are shown in Table I. The *Base Module* incorporates a single core of AES-128 enc/dec with Keccak-*f*[400] and produces either 128-bit enc/dec output or 128-bit hash digest in 20 clock cycles. On Virtex-7, it results in a operating frequency of 376.88 MHz and a throughput of 2713.54 Gbps ($\frac{144\times376.88}{20}$) for hashing while 2412.03 Gbps ($\frac{128\times376.88}{20}$) for enc/dec. The design utilizes 285 Slices and 4 BRAMs that correspond to total of 797 Equivalent (Equiv.) Slices where 1 32Kb BRAM is taken as equivalent to 128 Slices that holds equally for Virtex-5/6/7 FPGA. These Xilinx families have unified architecture based on LUT-6 technology, where one 256 × 32-bit table fits into 128 LUTs that correspond to 32 Slices as provided in [42].

The proposed unified *AE$HA-3* design integrates four cores of AES-128 enc/dec with SHA-3. The core generates either four 128-bit enc/dec output in 20 clock cycles (using the key size of 128-bit) or 256-bit hash digest (using SHA3-256 mode) in 24 clock cycles providing the security level of 128-bit. On Virtex-7, it has a throughput of 16.513 Gbps ($\frac{1088\times364.26}{24}$) for hashing while 9.325 Gbps ($\frac{512\times364.26}{20}$) for enc/dec. The design utilizes total of 3298 Equivalent Slices (almost 4× the *Base Module*) and results in an operating frequency of 364.26 MHz.

To best of authors' knowledge, reported *AE$HA-3* is the first resource-shared design for AES enc/dec with SHA-3 and at present no implementation results are available in the open literature for the comparisons purposes. Due to the unavailability of implementation results of unified architecture for AES with SHA-3, we therefore first provide the comparison of our proposed design with stand-alone implementations of both the algorithms, i.e., AES and SHA-3. Then we compare our implementation results with the available resource-shared designs of AES and other SHA-3 hash finalist such as Grøstl and Fugue.

A. Comparison With Stand-Alone Implementations

A fair comparison of *AE$HA-3* with stand-alone dedicated implementations of AES and SHA-3 has some inherent difficulties. These dedicated *cryptographic accelerators* are designed with the goal of achieving high performance and often have little or no flexibility. Flexibility is a desirable feature of cryptographic cores, which is however, *orthogonal* to the performance offered by these dedicated designs. Unification of multiple cryptographic functions that must coexist and exclusively (one at a time) executed are more area efficient, considering the area of the flexible core with the sum of the area of stand-alone cores. This is due to the possibility of aggressively exploiting the structural commonalities between these functions to enable resource-sharing techniques providing a compact solution and contributing to performance efficiency as well.

The comparison of our proposed designs with available stand-alone implementations is given in Table II. After a detailed literature survey, we report here the FPGA-based designs with Virtex-5/6/7 implementations. In the absence of availability of source codes for these designs, it is not possible to convert results from one type of devices to another. However, Xilinx Virtex-5/6/7 are based on similar architecture and consequently, from area perspective Equiv. Slices for designs will be almost same but frequency may vary with the choice of FPGA device; Virtex-5 has 550 MHz as maximum operating frequency while Virtex-6 has 601 MHz. For that reason, we have calculated the normalized throughput for Virtex-5 implementations by multiplying a factor of $1.09 = \frac{601}{550}$. Further, we have also provided our own stand-alone implementations of Keccak-*f*[400], AES-128 enc/dec and SHA-3

TABLE II
RESULTS COMPARISON OF PROPOSED DESIGNS WITH STAND-ALONE ALGORITHM IMPLEMENTATIONS

	Function	**FPGA**	**Area (Slices+BRAM)**	**Equiv. Slices**	**Thr. (Mbps)**	**Power (mW)**
Newe *et al.* [45]	AES-128 enc	Virtex-6	407 + 8	1431	637.14	–
Hussain and Jamal [46]	AES-128 enc/dec	Virtex-7	2444 + 0	2444	5306	–
Bouhraoua [47]	AES-128 enc/dec	Virtex-6	6318 LUTs + 0	2106[a]	4967	–
Granado-Criado *et al.* [48]	AES-128 enc	Virtex-6	415 + 16	2463	1815	–
Moreira *et al.* [49]	AES-128 enc	Virtex-5	280 + 9	1432	1830/1994*	–
	Keccak-f[400]	Virtex-5	482 + 0	482	1748/1905*	–
Jungk *et al.* [50]	Keccak-f[400]	Virtex-5	289 + 0	289	1135/1237*	–
Jungk *et al.* [51]	Keccak-f[400]	Virtex-5	319 + 0	319	1836/2001*	–
This work						
Ref_Keccak_400	Keccak-f[400]	Virtex-6	358 + 0	358	2610	134
Ref_AES	AES-128 enc/dec	Virtex-6	215 + 4	727	1856	184
Base Module	Keccak-f[400]	Virtex-6	297 + 4	809	2397.46	190
	AES-128 enc/dec				2131.07	
Newe *et al.* [45]	Keccak-f[1600]	Virtex-6	1048 + 0	1048	8830	–
Sharif *et al.* [52]	Keccak-f[1600]	Virtex-5	1338 + 1	1466	11252/12265*	–
		Virtex-5	1352 + 0	1352	13536/14754*	–
Gaj *et al.* [53]	Keccak-f[1600]	Virtex-6	1086 + 0	1086	11839	–
		Virtex-5	1369 + 0	1369	13337/14537*	–
Jungk *et al.* [50]	Keccak-f[1600]	Virtex-5	1215 + 0	1215	5054/5509*	–
This work						
Ref_SHA-3	Keccak-f[1600]	Virtex-6	1345 + 0	1345	13963	790
AE$HA-3	Keccak-f[1600]	Virtex-6	1380 + 16	3428	14876.13	1070
	4×AES-128 enc/dec				8400.64	

[a] Estimated Slices from LUTs considering 75% Slice utilization, * Normalized Thr. w.r.t Virtex-6 = Virtex-5 × 1.09

as *Ref_Keccak_400*, *Ref_AES* and *Ref_SHA-3*, respectively for the purpose of power consumption estimation.

Comparing to the reported stand-alone designs in Table II, our *Base Module* utilizes 57.73% less area with additional capability of decryption than the stand-alone implementation of AES enc and Keccak-f[400] by Moreira *et al.* [49] having 1,914 Equiv. Slices (1432 + 482). Moreover, to compare the area resources of our proposed *Base Module* with the smallest reported AES-128 core by Newe *et al.* [45] and Keccak-f[400] design by Jungk *et al.* [50], our *Base Module* saved 52.97% area resources than 1,720 Equiv. Slices (1431 + 289). Furthermore, in comparison to our own stand-alone implementations, *Base Module* saved 25.44% of area resources and 40.25% of power with respect to the combined Equiv. Slices and power consumption of 1085 (358 + 727) and 318 (134 + 184) mW respectively. The *Ref_Keccak_400* is a novice implementation having more area than *Base Module* as the combined θ-3, ρ, π & χ into a single SixIE on LUT-6, reduced 50% of area as well as eliminated the need of 400-bit intermediate registers between the Keccak-f[400] transformations.

In addition to reduced area utilization, our *Base Module* out performs many stand-alone designs in terms of throughput. It provides 20% better throughput for Keccak-f[400], i.e., 2.40 Gbps as compared to [49]–[51] designs whereas for AES-128 enc/dec, results in throughput of 2.13 Gbps greater than many stand-alone AES implementations [45], [48], [49] as shown in Table II. The achieved throughput figures are reasonable for the resource-constrained applications.

Similarly our proposed resource-shared *AE$HA-3* incorporating both SHA-3 and four cores of AES-128 enc/dec, utilizes only 3428 Slices. Comparing the area utilization with the smallest Keccak-f[1600] and 4 × AES enc/dec cores by [45], our core signifies 49.37% resource saving. Whereas, comparing to our own stand-alone implementations such as *Ref_SHA-3* and 4×*Ref_AES*, the area and power savings are 19.40% and 29.88% respectively. At the same time our proposed *AE$HA-3* core have highest throughput figures for both AES-128 enc/dec and Keccak-f[1600] (SHA3-256) by considering either the compatible Virtex-6 or normalized Virtex-5 devices. Our proposed designs are clearly either faster/smaller or both.

B. Comparison With Resource-Shared Implementations

Till date, no resource-shared hardware architecture that incorporates the two NIST standardized algorithms i.e., AES and SHA-3, has been presented in open literature. Therefore, we provide the comparisons of our work with other reported resource-shared architectures of AES with SHA-3 finalists such as Grøstl and Fugue. Most of these shared cores results are presented on Altera FPGA and consequently, an equivalence conversion is necessary. A single Logic Element (LE) on Altera Cyclone-III FPGA is taken as equivalent to 0.12 Slices w.r.t Virtex-5 [54], which is almost same for Virtex-6 FPGA.

The comparison of our proposed designs with available unified architectures is provided in Table III. The *Base Module* is compared with unified low-area designs whereas *AE$HA-3* design is compared with available unified high-speed designs. It is evident from the comparison table that our *Base Module* provides a balance solution such as utilized optimal number of area resources and offered highest throughput. Although area resources of our proposed *Base Module* is greater than these low-area designs [30]–[32], but enables much higher throughput figures. It is a known fact that area of design can be reduced by folding techniques at the cost of high latency

TABLE III
RESULTS COMPARISON OF PROPOSED RESOURCE-SHARED DESIGNS WITH REPORTED UNIFIED DESIGNS

	Cores	FPGA	Area (LE/Slices+BRAMs)	Equiv. Slices	Fr (MHz)	Thr. (Mbps)	TPS (Mbps/Slices)
Jarvinen [30]	Fuge-256	Cyclone-III	4520 LE**	552	60.75	972	0.22
	AES-128 enc					778	0.17
At *et al.* [31]	Grøstl-256	Virtex-7	185+1*	313	415	98	0.31
	AES-128 enc/dec					229	0.73
Pelnar *et al.* [32]	Grøstl-256	Virtex-6	302 + 0	302	80	13.24	0.05
	AES-128 enc					13.8	0.05
	AES-128 dec					9.99	0.03
Base Module	Keccak-f[400]	Virtex-6	297 + 4	809	332.98	2397.46	2.96
	AES-128 enc/dec					2131.07	2.63
Guo *et al.* [35]	Grøstl-256	Cyclone-IV	15135 LE**	1847	242.3	3877	0.26
	4×AES-128 enc						
Järvinen [30]	Grøstl-256	Cyclone-III	13723 LE**	1675	56.03	1434	0.10
	4×AES-128 enc					2869	0.21
Rogawski *et al.* [33]	Grøstl-256	Cyclone-III	23758 LE**	2851	144	2378	0.83
	4×AES-128 enc/dec						
Rogawski *et al.* [34]	Grøstl-256	Virtex-6	2447 + 0	2447	255	4212	1.72
	4×AES-128 enc/dec						
AE$HA-3	SHA3-256	Virtex-6	1380 + 16*	3428	328.15	14876.13	4.34
	4×AES-128 enc/dec					8400.64	2.45

*1 BRAM equivalent to 128 Slices [42], **1 LE equivalent to 0.12 Slices [54]

that eventually degrades the performance. It is worth noting here that both of these algorithms, i.e., Grøstl and Fugue are AES-inspired hash algorithms in their spirit and share major common cryptographic primitives/structures with AES, such as SubBytes transform, etc. Consequently, their resource-shared architectures with AES are not only much more straightforward to design but are more likely to win in terms of hardware efficiency. Both of these were SHA-3 finalists (and not winners) and were not taken up for NIST standardization.

For a better comparison, Throughput-per-Slice (TPS) is calculated for each design to compare the hardware efficiency of all these designs because it represents the ratio of throughput by the hardware resources (to achieve the resulted throughput). It is clear from Table III, that our proposed *Base Module* shows highest design efficiencies for both hash and enc/dec processes.

Similarly, our proposed *AE$HA-3* design, as shown in Table III, demonstrates the highest hardware efficiency (i.e., TPS) and proves to be the most efficient unified implementation of a standard hash function; SHA-3 with AES-128 enc & dec reported till date. The area resources of our proposed core are in the range of all these high-speed designs with added advantage of high performance. Worth noting point here is that the unified designs given by [30], [35] have incorporated only the AES encryption with Grøstl-256, incorporating the AES decryption in the same design would have required additional area resources. Furthermore, Grøstl-256 has the internal state-size of 512-bit whereas our proposed *AE$HA-3* design has the internal state-size of 1600-bit for the SHA-3. In addition, *AE$HA-3* core can be easily reconfigured to operate various flavors of AES and SHA-3 (including AES-192/256) and trivially tweaked to operate several modes of operation including AES Counter (CTR) mode, SHA-3 Hash-MAC (HMAC) etc. This tweaking will require external circuitry/control logic, that will not significantly increase area budget or critical path of the design.

V. CONCLUSION

This work undertakes *AE$HA-3*, a unified hardware implementation of AES+SHA-3, both of which are NIST standards for data confidentiality and integrity, respectively. This core not only finds its practical application in a wide number of modern communication system's security protocols for acceleration, but also can be used by PQC algorithms. Macro block singularities between the two standards are aggressively exploited to achieve logic/memory minimization to achieve area efficiency, highly suitable especially for low-power applications with multi-purpose security requirements. The *Base Module* incorporates Keccak-f[400] with AES-128 enc/dec and is well suited for IoT applications whereas *AE$HA-3* integrates SHA-3 with AES-128 enc/dec and is ideal for high throughput multi-gigabit applications. Both the proposed designs result in highest hardware efficiency (TPS) as compared to all available resource-shared architectures and are flexible to address the different security requirements. Study of Side-channel Analysis (SCA) vulnerabilities of *AE$HA-3* and incorporation of generic countermeasures is next on the agenda of this project.

REFERENCES

[1] K. Shahzad, A. Khalid, Z. E. Rákossy, G. Paul, and A. Chattopadhyay, "CoARX: A coprocessor for ARX-based cryptographic algorithms," in *Proc. 50th ACM/EDAC/IEEE Design Automat. Conf. (DAC)*, May/Jun. 2013, pp. 1–10.

[2] R. Buchty, N. Heintze, and D. Oliva, "Cryptonite—A programmable crypto processor architecture for high-bandwidth applications," in *Proc. Int. Conf. Archit. Comput. Syst.* Augsburg, Germany: Springer, 2004, pp. 184–198.

[3] A. Khalid, G. Paul, and A. Chattopadhyay, "RC4-AccSuite: A hardware acceleration suite for RC4-like stream ciphers," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 3, pp. 1072–1084, Mar. 2017.

[4] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM Rev.*, vol. 41, no. 2, pp. 303–332, Jan. 1999, doi: 10.1137/S0036144598347011.

[5] L. K. Grover, "Quantum mechanics helps in searching for a needle in a haystack," *Phys. Rev. Lett.*, vol. 79, no. 2, pp. 325–328, Jul. 1997. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.79.325

[6] D. Moody, "Post-quantum cryptography: NIST's plan for the future," in *Proc. Talk PQCrypto Conf.*, Fukuoka, Japan, Feb. 2016. [Online]. Available: https://pqcrypto2016.jp/data/pqc2016_nist_announcement.pdf

[7] *Advanced Encryption Standard*, National Institute of Standards and Technology (NIST), Gaithersburg, MD, USA, Standard FIPS-197, Nov. 2001.

[8] *SHA-3 Standard: Permutation-Based Hash Extendable-Output Functions*, National Institute of Standards and Technology (NIST), Gaithersburg, MD, USA, Standard FIPS PUB 202, Aug. 2015.

[9] *Commercial National Security Algorithm Suite*, National Security Agency, Maryland, MD, USA, Aug. 2015. [Online]. Available: https://www.iad.gov/iad/programs/iad-initiatives/cnsa-suite.cfm

[10] S. Sen Gupta, A. Chattopadhyay, and A. Khalid, "Designing integrated accelerator for stream ciphers with structural similarities," *Cryptogr. Commun.*, vol. 5, no. 1, pp. 19–47, Mar. 2013.

[11] S. S. Gupta, A. Chattopadhyay, and A. Khalid, "HiPAcc-LTE: An integrated high performance accelerator for 3GPP LTE stream ciphers," in *Proc. Int. Conf. Cryptol*. Chennai, India: Springer, 2011, pp. 196–215.

[12] X. Zhang and K. K. Parhi, "High-speed VLSI architectures for the AES algorithm," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 12, no. 9, pp. 957–967, Sep. 2004.

[13] D.-E.-S. Kundi and A. Aziz, "A low-power SHA-3 designs using embedded digital signal processing slice on FPGA," *Comput. Electr. Eng.*, vol. 55, pp. 138–152, Oct. 2016.

[14] *NIST Selects Winner of Secure Hash Algorithm (SHA-3) Competition*, National Institute of Standards and Technology (NIST), Gaithersburg, MD, USA, Oct. 2012. [Online]. Available: http://www.nist.gov/itl/csd/sha-100212.cfm

[15] L. Wu, C. Weaver, and T. Austin, "CryptoManiac: A fast flexible architecture for secure communication," in *Proc. 28th Annu. Int. Symp. Comput. Archit.*, Jun./Jul. 2001, pp. 110–119.

[16] G. Sayilar and D. Chiou, "Cryptoraptor: High throughput reconfigurable cryptographic processor," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2014, pp. 155–161.

[17] K. Ananyi, H. Alrimeih, and D. Rakhmatov, "Flexible hardware processor for elliptic curve cryptography over NIST prime fields," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 17, no. 8, pp. 1099–1112, Aug. 2009.

[18] A. Satoh and K. Takano, "A scalable dual-field elliptic curve cryptographic processor," *IEEE Trans. Comput.*, vol. 52, no. 4, pp. 449–460, Apr. 2003.

[19] A. Khalid, M. Hassan, A. Chattopadhyay, and G. Paul, "RAPIDfeinSPN: A rapid prototyping framework for feistel and SPN-based block ciphers," in *Proc. Int. Conf. Inf. Syst. Secur.* Kolkata, India: Springer, 2013, pp. 169–190.

[20] A. Khalid *et al.*, "RunStream: A high-level rapid prototyping framework for stream ciphers," *ACM Trans. Embedded Comput. Syst.*, vol. 15, no. 3, p. 61, 2016.

[21] D. Cao, J. Han, and X.-Y. Zeng, "A reconfigurable and ultra low-cost VLSI implementation of SHA-1 and MD5 functions," in *Proc. 7th Int. Conf. ASIC*, Guilin, China, Oct. 2007, pp. 862–865.

[22] K. U. Järvinen, M. Tommiska, and J. Skyttä, "A compact MD5 and SHA-1 co-implementation utilizing algorithm similarities," in *Proc. Int. Conf. Eng. Reconfigurable Syst. Algorithms (ERSA)*. Las Vegas, NV, USA: CSREA Press, Jun. 2005, pp. 48–54.

[23] C.-W. Ng, T.-S. Ng, and K.-W. Yip, "A unified architecture of MD5 and RIPEMD-160 hash algorithms," in *Proc. IEEE Int. Symp. Circuits Syst.*, vol. 2. Vancouver, BC, Canada, May 2004.

[24] T. S. Ganesh, M. T. Frederick, T. S. B. Sudarshan, and A. K. Somani, "Hashchip: A shared-resource multi-hash function processor architecture on FPGA," *Integration*, vol. 40, no. 1, pp. 11–19, Jan. 2007.

[25] M.-Y. Wang, C.-P. Su, C.-L. Horng, C.-W. Wu, and C.-T. Huang, "Single- and multi-core configurable AES architectures for flexible security," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 18, no. 4, pp. 541–552, Apr. 2010.

[26] Helion Tech., "High performance AES (Rijndael) cores for Xilinx FPGAs," Cambridge, U.K., Tech. Rep. Rev 2.4.0, 2007. [Online]. Available: http://www.heliontech.com/downloads/aes_xilinx_helioncore.pdf

[27] S. Satpathy *et al.*, "220MV-900MV 794/584/754 GBPS/W reconfigurable GF(2^4)2 AES/SMS4/camellia symmetric-key cipher accelerator in 14NM tri-gate CMOS," in *Proc. IEEE Symp. VLSI Circuits*, Jun. 2018, pp. 18–22.

[28] N. At, J.-L. Beuchat, E. Okamoto, I. San, and T. Yamazaki, "Compact hardware implementations of ChaCha, BLAKE, threefish, and skein on FPGA," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 61, no. 2, pp. 485–498, Feb. 2014.

[29] J.-L. Beuchat, E. Okamoto, and T. Yamazaki, "A low-area unified hardware architecture for the AES and the cryptographic hash function ECHO," *J. Cryptograph. Eng.*, vol. 1, no. 2, pp. 101–121, Aug. 2011.

[30] K. Järvinen, "Sharing resources between AES and the SHA-3 second round candidates Fugue and Grøstl," in *Proc. 2nd SHA Candidate Conf.*, Santa Barbara, CA, USA: Univ. California, Aug. 2010. [Online]. Available: https://csrc.nist.gov/events/2010/the-second-sha-3-candidate-conference

[31] N. At, J.-L. Beuchat, E. Okamoto, I. San, and T. Yamazaki, "A low-area unified hardware architecture for the AES and the cryptographic hash function Grøstl," *J. Parallel Distrib. Comput.*, vol. 106, pp. 106–120, Aug. 2017.

[32] M. Pelnar, M. Muehlberghuber, and M. Hutgter, "Putting together what fits together–GrÆStl," in *Smart Card Research and Advanced Applications* (Lecture Notes in Computer Science), vol. 7771. Berlin, Germany: Springer 2013, pp. 173–187.

[33] M. Rogawski and K. Gaj, "A high-speed unified hardware architecture for AES and the SHA-3 candidate Grøstl," in *Proc. 15th Euromicro Conf. Digit. Syst. Design*, Izmir, Turkey, Sep. 2012, pp. 568–575.

[34] M. Rogawski, K. Gaj, and E. Homsirikamol, "A high-speed unified hardware architecture for 128 and 256-bit security levels of AES and the SHA-3 candidate Grøstl," *Microprocessors Microsyst.*, vol. 37, nos. 6–7, pp. 572–582, Aug. 2013.

[35] K. Guo and H. Heys, "A pipelined implementation of the Grøstl hash algorithm and the advanced encryption standard," in *Proc. 26th IEEE Can. Conf. Elect. Comput. Eng. (CCECE)*, Regina, SK, Canada, May 2013, pp. 1–4.

[36] S. Shah, R. Velegalati, J.-P. Kaps, and D. Hwang, "Investigation of DPA resistance of block RAMs in cryptographic implementations on FPGAs," in *Proc. Int. Conf. Reconfigurable Comput. FPGAs*, Quintana Roo, Mexico, Dec. 2010, pp. 274–279.

[37] Intel Corp., "Stratix 10 logic array blocks and adaptive logic modules user guide," Santa Clara, CA, USA, Tech. Rep. UG-S10LAB, 2020. [Online]. Available: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literatur e/hb/stratix-10/ug-s10-lab.pdf

[38] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche, "The Keccak SHA-3 submission," STMicroelectron., Geneva, Switzerland, Tech. Rep. Ver 3.0, 2011. [Online]. Available: http://Keccak.noekeon.org/Keccak-reference-3.0.pdf

[39] S. Morioka and A. Satoh, "A 10 gbps full-AES crypto design with a twisted-BDD S-Box architecture," in *Proc. IEEE Int. Conf. Comput. Design: VLSI Comput. Processors*, Freiburg, Germany, Sep. 2002, pp. 98–103.

[40] D.-S. Kundi, A. Aziz, and N. Ikram, "A high performance ST-box based unified AES encryption/decryption architecture on FPGA," *Microprocessors Microsyst.*, vol. 41, pp. 37–46, Mar. 2016.

[41] S. Drimer, T. Güneysu, and C. Paar, "DSPs, BRAMs, and a pinch of logic: Extended recipes for AES on FPGAs," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 3, no. 1, p. 3, Jan. 2010.

[42] P. Bulens, F.-X. Standaert, J.-J. Quisquater, P. Pellegrin, and G. Rouvroy, "Implementation of the AES-128 on Virtex-5 FPGAs," in *Progress in Cryptology* (Lecture Notes in Computer Science), vol. 5023. Berlin, Germany: Springer 2008, pp. 16–26.

[43] A. Aziz and N. Ikram, "A look-up-table implementation of AES," in *Proc. Int. Conf. High Perform. Comput., Netw. Commun. Syst. (HPCNCS)*, Orlando, FL, USA, 2007, pp. 187–191.

[44] R. Avanzi *et al.*, "CRYSTALS-Kyber algorithm specification (version 2.0)," NIST PQC round 2, Gaithersburg, MD, USA, Tech. Rep. Ver 2.0, 2019. [Online]. Available: https://pq-crystals.org/kyber/data/kyber-specification-round2.pdf

[45] T. Newe, M. Rao, D. Toal, G. Dooly, E. Omerdic, and A. Mathur, *Efficient and High Speed FPGA Bump in the Wire Implementation for Data Integrity and Confidentiality Services in the IoT* (Smart Sensors, Measurement and Instrumentation), vol. 22. Cham, Switzerland: Springer, 2017, pp. 259–285.

[46] U. Hussain and H. Jamal, "An efficient high throughput FPGA implementation of AES for multi-gigabit protocols," in *Proc. 10th Int. Conf. Frontiers Inf. Technol.*, Dec. 2012, pp. 215–218.

[47] A. Bouhraoua, "Design feasibility study for a 500 Gbits/s advanced encryption standard cipher/decipher engine," *IET Comput. Digit. Techn.*, vol. 4, no. 5, pp. 48–334, 2010.

[48] J. M. Granado-Criado, M. A. Vega-Rodríguez, J. M. Sánchez-Pérez, and J. A. Gómez-Pulido, "Hardware security platform for multicast communications," *J. Syst. Archit.*, vol. 60, no. 1, pp. 11–21, Jan. 2014.

[49] N. Moreira, A. Astarloa, U. Kretzschmar, J. Lazaro, and E. Molina, "Securing IEEE 1588 messages with message authentication codes based on the KECCAK cryptographic algorithm implemented in FPGAs," in *Proc. IEEE 23rd Int. Symp. Ind. Electron. (ISIE)*, Istanbul, Turkey, Jun. 2014, pp. 1899–1904.
[50] B. Jungk, M. Stottinger, and M. Harter, "Shrinking KECCAK hardware implementations," in *Proc. SHA Workshop*. Santa Barbara, CA, USA: Univ. California, Aug. 2014. [Online]. Available: http://csrc.nist.gov/groups/ST/hash/sha-3/Aug2014/documents/jungk_paper_sha3_2014_workshop.pdf
[51] B. Jungk and M. Stottinger, "Among slow dwarfs and fast giants: A systematic design space exploration of KECCAK," in *Proc. 8th Int. Workshop Reconfigurable Communication-Centric Systems-on-Chip (ReCoSoC)*, Darmstadt, Germany, Jul. 2013, pp. 1–8.
[52] M. U. Sharif, R. Shahid, M. Rogawski, and K. Gaj, "Use of embedded FPGA resources in implementations of five round three SHA-3 candidates," in *Proc. ECRYPT II Hash Workshop*, Tallinn, Estonia, May 2011.
[53] K. Gaj, E. Homsirikamol, M. Rogawski, R. Shahid, and M. U. Sharif, "Comprehensive evaluation of high-speed and medium-speed implementations of five SHA-3 finalists using Xilinx and Altera FPGAs," in *Proc. 3rd SHA-3 Candidate Conf.*, Washington, DC, USA, Mar. 2012, pp. 1–19.
[54] Intel Corp., "Stratix III FPGAs vs. Xilinx Virtex-5 devices: Architecture and performance comparison," White Paper WP01007-2.1, Oct. 2007.

Dur-e-Shahwar Kundi (Member, IEEE) received the M.Sc. and Ph.D. degrees in electrical engineering from the National University of Sciences and Technology (NUST), Karachi, Pakistan, in 2010 and 2016, respectively. She is currently a Post-Doctoral Fellow with the College of Electronic and Information Engineering, Nanjing University of Aeronautics and Astronautics (NUAA), Nanjing, China, under the scheme of China Postdoctoral Science Foundation. Her research interests include hardware security, cryptographic engineering, and reconfigurable computing.

Ayesha Khalid (Member, IEEE) received the B.E. degree in computer systems engineering from the National University of Sciences and Technology (NUST), Pakistan, and the M.S. degree in electrical engineering from the Center for Advanced Studies in Engineering (CASE), University of Engineering and Technology, UET-Taxila. She has been a Lecturer with the Queen's University of Belfast, Belfast, U.K., since 2018. Her research interests include lattice-based cryptography, embedded systems security, side channel attacks, and cryptographic hardware. She was a recipient of the DAAD Scholarship Award for her Ph.D. studies at RWTH Aachen, Germany.

Arshad Aziz received the Ph.D. degree in electrical engineering from NUST, Pakistan, in 2007. He is currently an Associate Professor with the Electrical Engineering Department, National University of Sciences and Technology, Karachi, Pakistan. He has published over 90 technical papers in journals and conference proceedings. His research interests include network security, cryptography, and FPGA-based system design.

Chenghua Wang received the B.Sc. and M.Sc. degrees from Southeast University, Nanjing, China, in 1984 and 1987, respectively. In 1987, he joined the College of Electronic and Information Engineering, Nanjing University of Aeronautics and Astronautics, Nanjing, where he became a Full Professor in 2001. He has published six books and over 100 technical papers in journals and conference proceedings. His current research interests include testing of integrated circuits and systems for communications. He was a recipient of more than ten teaching and research awards at the provincial and ministerial level.

M*á*ire O'Neill (Senior Member, IEEE) received the M.Eng. (Hons.) and Ph.D. degrees in electrical and electronic engineering from Queen's University Belfast (QUB) in 1999 and 2002, respectively. She is currently a Regius Professor in electronics and computer engineering, an Acting Director of the Institute of Electronics, Communications and Information Technology (ECIT), and the Director of the UK Research Institute in Secure Hardware and Embedded Systems (RISE). She has authored two research books and has over 190 international peer-reviewed conference and journal publications. She is a fellow of Royal Academy of Engineering, a fellow of the Irish Academy of Engineering, and a member of Royal Irish Academy. She has been a technical program committee member for many international conferences, including DAC, CHES, DATE, SOCC, ISCAS, IET ISSC, and RFIDSec. She has received numerous awards for her research to date which include 2014 UK Royal Academy of Engineering Silver Medal, Women's Engineering Society (WES) prize at 2006 IET Young Woman Engineer of Year awards and named as British Female Inventor of the Year in 2007. She is an Associate Editor for the IEEE TRANSACTIONS ON COMPUTERS and the IEEE TRANSACTIONS ON EMERGING TOPICS IN COMPUTING and a Guest Editor for several journals, including *IET Information Security* in 2005 launch issue and a special issue on Cryptography in the coming decade in *ACM Transactions on Embedded Computing* in 2015.

Weiqiang Liu (Senior Member, IEEE) received the B.Sc. degree in information engineering from the Nanjing University of Aeronautics and Astronautics (NUAA), Nanjing, China, and the Ph.D. degree in electronic engineering from Queen's University Belfast (QUB), Belfast, U.K., in 2006 and 2012, respectively. He is currently an Associate Professor and the Vice Dean of the College of Electronic and Information Engineering, NUAA. He has published one research book by Artech House and over 90 leading journal and conference papers. His research interests include computer arithmetic, hardware security, and VLSI design for digital signal processing and cryptography. He is a member of the IEEE Circuits and Systems Society. He is the program Co-Chair of ARITH and program members of international conferences: ARITH, DATE, ASP-DAC, ISCAS, ASAP, ISVLSI, GLSVLSI, AsiaHOST, NANORACH, AICAS, SiPS, and ICONIP. He has served as an Associate Editor for several IEEE journals such as the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS I: Regular Papers from 2020 to 2022, the IEEE TRANSACTIONS ON EMERGING TOPICS IN COMPUTING from 2019 to 2021, and the IEEE TRANSACTIONS ON COMPUTERS from 2015 to 2019, and a Guest Editor for the PROCEEDINGS OF THE IEEE from 2019 to 2020.