Journal of Circuits, Systems, and Computers
Vol. 28, No. 12 (2019) 1930009 (25 pages)

DOI: 10.1142/S0218126619300095

World Scientific
www.worldscientific.com

ASIC and FPGA Comparative Study for IoT Lightweight Hardware Security Algorithms*

Nagham Samir[†,¶], Abdelrahman Sobeih Hussein[‡,‖], Mohaned Khaled[†,**], Ahmed N. El-Zeiny[†,††], Mahetab Osama[†,‡‡], Heba Yassin[†,§§], Ali Abdelbaky[†,¶¶], Omar Mahmoud[†,‖‖], Ahmed Shawky[†,***] and Hassan Mostafa[†,§,†††]

[†]*Electronics and Communications Engineering Department, Faculty of Engineering, Cairo University, Giza 12613, Egypt*

[‡]*Computers and Systems Engineering Department, Faculty of Engineering, Mansoura University, Mansoura 35516, Egypt*

[§]*Nanotechnology and Nanodevices Program, Zewail City of Science and Technology, Giza, Cairo, Egypt*

[¶]*naghamsamir2014@gmail.com*
[‖]*abdelrahman.sobeih@gmail.com*
[**]*MohanedEECE@gmail.com*
[††]*ahmed.nagy.elzeiny@gmail.com*
[‡‡]*mahetabo17@gmail.com*
[§§]*heba.m.yassin@gmail.com*
[¶¶]*alibaky92@gmail.com*
[‖‖]*omar.mahmoud.yahya@gmail.com*
[***]*ahmed.shawky.awwad@gmail.com*
[†††]*hmostafa@uwaterloo.ca; hmostafa@zewailcity.edu.eg*

Received 14 September 2018
Accepted 8 November 2018
Published 28 December 2018

Data security, privacy and authenticity are crucial in wireless data transmission. Low power consumption is the main requirement for any chip design targeting the Internet of Things (IoT) applications. In this research paper, a comparative study of eight authenticated encryption and decryption algorithms, selected from the "Competition for Authenticated Encryption: Security, Applicability and Robustness" (CAESAR), namely, ACORN, ASCON, CLOC, JOLTIK, MORUS, PRIMATEs, SCREAM and SILC, is presented. The FPGA and ASIC implementations of these eight algorithms are synthesized, placed and routed. Power, area, latency and throughput are measured for all algorithms. All results are analyzed to determine the most suitable algorithm for IoT applications. These results show that ACORN algorithm exhibits the lowest power consumption of the eight studied at the expense of lower throughput and higher

*This paper was recommended by Regional Editor Emre Salman.
[†††]Corresponding author.

latency. MORUS algorithm gives the highest throughput among the eight selected algorithms at the expense of large area utilization.

Keywords: Internet of Things (IoT); hardware security; authentication encryption with associated data (AEAD); Competition for Authenticated Encryption: Security, Applicability and Robustness (CASEAR); Advanced Encryption Standard (AES).

1. Introduction

In the last few decades, network firewall has been the best solution to overcome the insecurity attacks. However, the emergence of Internet of Things (IoT) applications has made the security issue more critical and complicated.[1] IoT makes use of data collected from IoT devices to optimize the observation and control of the world in domains such as logistics, retail, military and healthcare.[2] This huge and continuously increasing number of devices is leading to more attack vectors by hackers.[3] As a result, the security becomes one of the main challenges required by IoT stakeholders to deploy the IoT applications in the market.

One of the most important questions that is arising in the IoT field is that how to achieve the IoT security. Two main data security models exist, software security and hardware security. The most popular and cost-effective model is software security. Software security is achieved by a cryptography program which is responsible for securing all the data of the organization network. Software security provides good levels of security, however, it has a clear disadvantage. The security of operating system (OS) compromises the security of the cryptography software. Moreover, continuous updates are needed for the OS and the cryptography program, especially the current versions of the OS are well known for the hackers. This disadvantage of the software security leads to the rising demand for the hardware security. Hardware security is achieved by connecting a hardware security module (HSM) to the organization network. HSM is a physical device that provides extra security for sensitive data. The HSM is responsible for achieving all the cryptography aspects (i.e., data encryption and decryption in addition to data authentication).[4] Correspondingly, the security question is rephrased from how to achieve the IoT security to how to implement the HSM module.

The objective of this work is to provide a quantitative answer to the above question by carrying out a comprehensive comparison among different cryptography algorithms that are used to implement the HSM module, taking into consideration the power consumption of the HSM-implemented modules to match the power constraints imposed by the low-power IoT applications.[5] The main contribution of this work is to provide a quantitative comparison among different HSM modules and to recommend the HSM algorithm that is suitable for IoT applications.

The paper presents a quantitative comparison among eight different algorithms that have participated in the "Competition for Authenticated Encryption: Security, Applicability and Robustness" (CAESAR).[6] The selected algorithms

are Side-Channel Resistant Authenticated Encryption with Masking (SCREAM), JOLTIK, ASCON, MORUS, CLOC, Simple Lightweight CFB (SILC), PRIMATEs and ACORN. The eight algorithms are implemented using the application-specific integrated circuit (ASIC) flow with CMOS UMC 130-nm technology. The synthesis step is performed by using Synopsys Design Compiler (DC) tool and the place and route step is conducted by using Cadence System-on-Chip (SoC) Encounter tool. Moreover, all the eight algorithms are implemented using the field-programmable gate array (FPGA) flow with ZC702 evaluation board for the Zynq-7000 XC7Z020. The synthesis and the place and route steps are carried out by using Xilinx Vivado 2016.4 tool. The maximum frequency of all algorithms is set to 10 MHz that suits the low-power IoT applications and also to provide a fair comparison among the different HSM modules.

In the presented quantitative comparison, several aspects are studied such as throughput, latency, power consumption and area, for both the ASIC and the FPGA design flows. Moreover, other aspects are investigated in this comparison such as: the nature of the algorithm (i.e., iterative or serialized), the type of key scheduling (i.e., tweakable or not), algorithm capabilities (i.e., whether it includes decryption unit in addition to the encryption unit or not), number of rounds, data block size, size of the public message number (PMN) and the size and design of the substitution box.

The paper is organized as follows. Section 2 discusses the low-power IoT applications and the security challenges. Section 3 explains the CAESAR competition background and the hardware application programming interfaces (APIs) of the different algorithms used in this paper. Section 4 introduces the different features associated with the selected algorithms in this paper. Section 5 provides brief descriptions of the eight selected algorithms. Section 6 presents the hardware implementations of the eight selected algorithms in FPGA and ASIC flows. Section 7 provides the results and discussion of the quantitative comparison. Section 8 holds a comparison between existing cryptographic schemes and the selected lightweight algorithms from CASESAR competition. Section 9 concludes the work of this paper.

2. Internet of Things

With the growth of IoT arises several security issues. These security issues are categorized into two main classes: (i) the variety in information and (ii) secure communication among objects. These issues cause various challenges in the IoT applications security such as: authentication, privacy and power consumption.[7] These challenges are described as follows.

2.1. *Authentication*

Due to the large increase in the number of objects, authentication with traditional methods such as secret keys and public keys becomes a complex task. However,

authentication is a hard challenge in IoT to avoid the third-party manipulation of data. Therefore, other methods are presented to achieve strong-based authentication between main parties.[7]

2.2. *Privacy*

Privacy is one of the main challenges in IoT security. This is because all the IoT objects are connected to the Internet and their information are vulnerable to hackers. Accordingly, the objects information should be protected and their privacy should be maintained.

2.3. *Power consumption*

Power is a significant challenge in IoT applications. This is because most of the IoT modules harvest energy from photovoltaic, piezoelectric and thermal energies. Accordingly, low power consumption is a must to lengthen the battery lifetime and to push the IoT industry into the market.

3. CASEAR Competition

3.1. *Cryptographic competitions*

Many cryptographic competitions are held to gather the cryptanalysts and cipher designers from all over the world to share their knowledge and designs. Following each competition, a final portfolio is announced. These competitions provide a great boost to the cryptographic research community's understanding of block ciphers, and a tremendous increase in confidence in the security of block ciphers. The first competition was held in 1997 when the United States National Institute of Standards and Technology (NIST) announced an open competition for a new Advanced Encryption Standard (AES). Eventually, NIST selected Rijndael as the standard AES.[8]

During the Early Symmetric Crypto workshop in Mondorf-les-Bains in 2013, the CAESAR was announced.[9] The objective of the CAESAR competition is to announce a final portfolio that includes the hardware implementation of an authenticated cipher that is much robust to several hacking attacks and compatible with different communication protocols. The competition had attracted 55 block cipher submissions and was performed over three rounds. Each round contained a submission of software version of each algorithm, C or Python, and then, a submission of hardware version of the algorithm.[6] Only 15 block ciphers reached the final round. The 15 block ciphers are ACORN, AEGIS, AESOTR, AEZ, ASCON, CLOC–SILC, COLM, Deoxys, JUMBO, Katje, Keyak, MORUS, NORX, OCB and Tiaoxin.[6]

3.2. *Hardware API for authentication ciphers*

The hardware API for authenticated ciphers has been developed to meet all the requirements of all algorithms that have been submitted to the CAESAR competition. The top level of the API is the authenticated encryption with associated data (AEAD) core. The architecture of the AEAD core consists of three main blocks: pre-processor, cipher core and post-processor, as shown in Fig. 1. The main difference between the different algorithms is in the cipher core implementation, as it contains the hardware blocks that perform either encryption or decryption and authentication algorithm steps.[10] The George Mason University Application Programming Interface (GMU-API) blocks are described as follows.

3.2.1. *Pre-processor*

Pre-processor is the first block of the AEAD core which receives public and secret data and starts processing them. It is responsible for doing various processing functions, such as:

- parsing segment headers loading and activating keys,
- serial-in-parallel-out loading of input blocks,
- padding input blocks in case the input block size is not equal to the algorithm block size,
- keeping track of the number of data bytes left to process.

3.2.2. *Cipher core*

The cipher core is divided into two blocks: core data path and core controller. The core data path contains the hardware which is responsible for encryption or

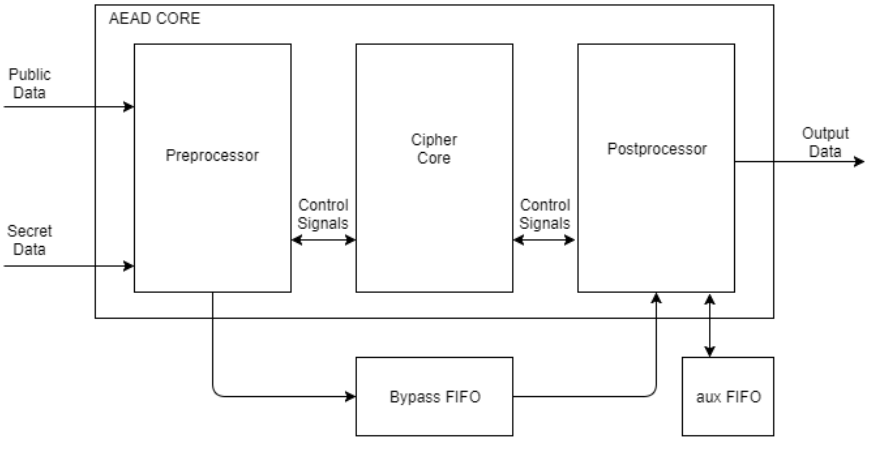


Fig. 1. CAESAR hardware API.[10]

decryption and processing the associative data to perform data (tag) generation, in addition to the hardware which is responsible for the key scheduling and the generation of round keys. The cipher core controller is an algorithmic state machine that takes some information signals from the pre-processor and generates control signals to the core data path.

3.2.3. *Post-processor*

The post-processor is the output stage of the API. It is responsible for:

- clearing any portions of the output blocks that are not belonging to the ciphertext or plaintext,
- parallel-in-serial-out conversion of output blocks into words,
- formatting output words into segments,
- generating the status block with the result of authentication. If the message is authenticated, it outputs its block through the DO port, else it discards it.

3.2.4. *Bypass first-in-first-out*

Small 4×24 first-word-fall-through (FWFT) first-in-first-out (FIFO) bypasses the tags, header, associated data (AD) and any data blocks that are used in the authentication process and will not be encrypted. It also bypasses any required data that the post-processor needs to operate with the maximum efficiency. For example, post-processor should know whether the last block needs to be unpadded or not. In addition, the post-processor should know whether the incoming data is ciphertext or plaintext, because if it is a ciphertext, then there is no need to temporarily store it as there is no tag verification needed in the encryption.[10]

3.2.5. *Auxiliary FIFO*

The memory used by the post-processor to temporarily store the decrypted message till the result of authentication is ready.[10]

4. Common Features for CASEAR Candidates

The selected CAESAR candidates support authentication, by using AEAD. The objective of the CAESAR competition is to submit algorithms that are supporting confidential and authenticated communication. Authentication operation happens at transmitter and receiver sides. At the transmitter side, the data is encrypted and the tag is generated using the message and associated data. At the receiver side, the message is decrypted and then the tag is generated by applying the same operations performed at the transmitter side. Original tag from transmitter side is compared with the tag from receiver side. If they are the same, then the message is authenticated, otherwise the receiver discards the message.

Some of the selected candidates are block ciphers, meaning that input data is processed in terms of data block. Data blocks are processed using various modes. In this section, the following operation modes are presented:

- Electronic code block (ECB): Encryption and decryption operations are conducted independently on each data block of plaintext and ciphertext, respectively. The advantage of ECB mode is the ability to accomplish the operation in parallel. However, ECB mode suffers from small bit diffusion which means that the order of the plaintext is suffering from few scattered positions in the ciphertext.
- Cipher feedback (CFB): The first message is XORed with encrypted initialization vector (IV) to produce ciphertext. Then the subsequent messages are processed by

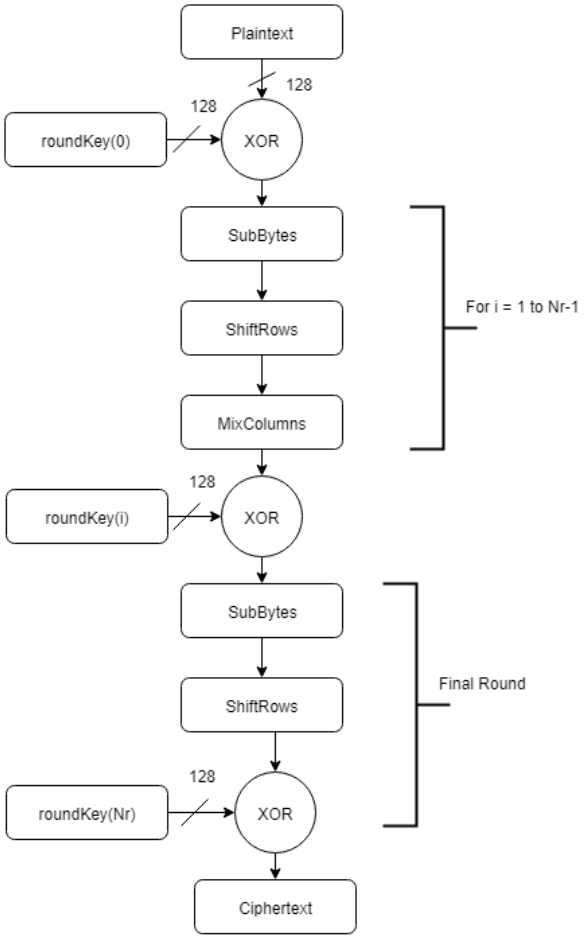


Fig. 2. The AES algorithm.[12]

considering the previous ciphertext as the current IV. CFB mode is unable to recover the whole message if a ciphertext is lost.

- Cipher block chaining (CBC): In order to produce the ciphertext, the same operations of CFB mode are carried out again. However the messages are XORed with IV, not encrypted IV.

The following selected algorithms: SILC, JOLTIK and CLOC are based on AES to perform the encryption and the decryption processes. AES is a symmetric block cipher that uses several key sizes. AES has various standard versions: AES-128, AES-192 and AES-256.[11] Number of rounds for each version depends on the key size. It uses 10, 12 and 14 rounds for key sizes of 128, 192 and 256, respectively. Data to be encrypted is represented in 4×4 matrix of bytes denoted by state. The block cipher applies four permutation functions in each round which are: add round key, sub bytes, shift rows (SR) and mix columns (MC). Figure 2 shows a flowchart for AES encryption algorithm.[12]

Add round key function XORed the current state with the round key. The round keys are generated using a key scheduling algorithm which takes the original 128-bit or 192-bit or 256-bit key and generates the 10, 12 or 14 128-bit round keys. Sub bytes function replaces each byte in the current state using the substitution box (S-box). The shift rows function rotates the state rows right with different numbers of positions. First row is left unrotated, second row is rotated with one position, third row is rotated with two positions and fourth row is rotated with three positions. Mix columns function that is denoted by the bit diffusion layer considers the columns of the state as polynomials over Galois field (GF) and multiplied them with a fixed polynomial.[13]

5. The Selected Algorithms

5.1. *ACORN algorithm*

ACORN is a symmetric and authenticated encryption (AE) algorithm based on stream cipher. Stream cipher differs from block cipher in that the key (K) and input data are applied bit-wise. ACORN provides parallel operation for encryption and decryption processes. This parallelism is achieved through independent processing for each bit of either plaintext or ciphertext. This parallelism benefits lightweight hardware implementation as the control circuit for the hardware implementation is greatly simplified.[14] Number of bits that are being padded to the message is fixed which reduces the hardware implementation cost.[15] ACORN is an inverse-free authenticated encryption scheme. This type of schemes require low memory and area because they utilize one block to perform either encryption or decryption operations instead of separate blocks for each operation.[16]

ACORN has various standards that are the same in tag, key and initialization vector sizes, which is 128 bits, but differ in the number of steps. ACORN v2 provides

more protection than ACORN v1 by increasing the number of steps.[16] In this paper results of ACORN are carried out from ACORN v2 implementation with eight states processed in parallel. The hardware used to accomplish encryption and decryption processes are gathered in a single block with a control signal to select between encryption and decryption. Input of this block is multiplexed among plaintext or ciphertext or associated data, or even zero. Figure 3 represents the data path for cipher core of ACORN v2 for bit-wise case (not 8 bits).[15] It depends mainly on two blocks. First block is the state update function, which updates the internal state and then affects the output ciphertext and tag. Second block is the key stream generator (KSG) with current state as the input. It is used to generate a key stream that is used later to generate the tag and the ciphertext.

5.2. *ASCON algorithm*

ASCON is an AEAD algorithm. ASCON depends on duplex sponge mode, which produces a string that is based on the whole input string.[16] Sponge mode is a

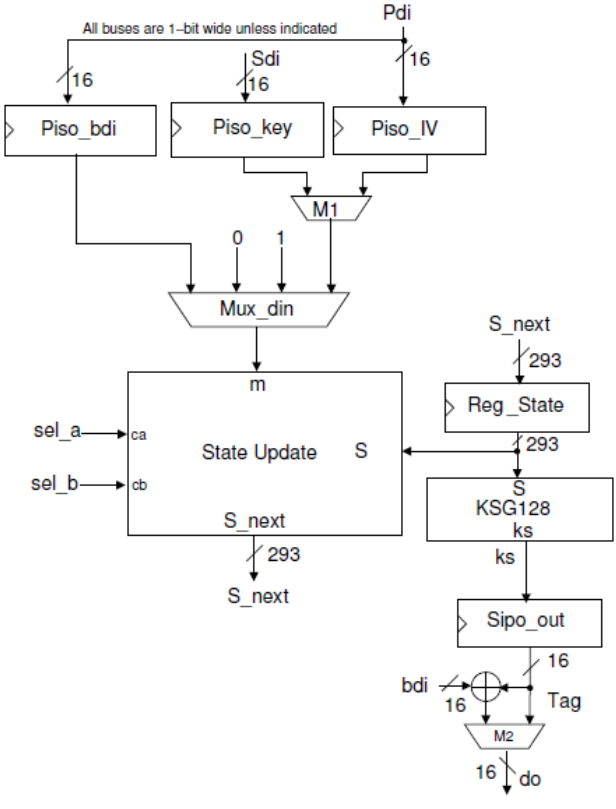


Fig. 3. ACORN v2 cipher core data path.[15]

permutation mode that produces an output evaluated from a state value which is updated using key, plaintext and associated data. The ASCON encryption process is carried out by using permutation blocks that perform iterations on predefined operations.[17] ASCON has several parameters used for encryption such as: secret key (K), associated data (A), public message number that is denoted by nonce (N) and IV, in order to encrypt a plaintext (P), according to the formula

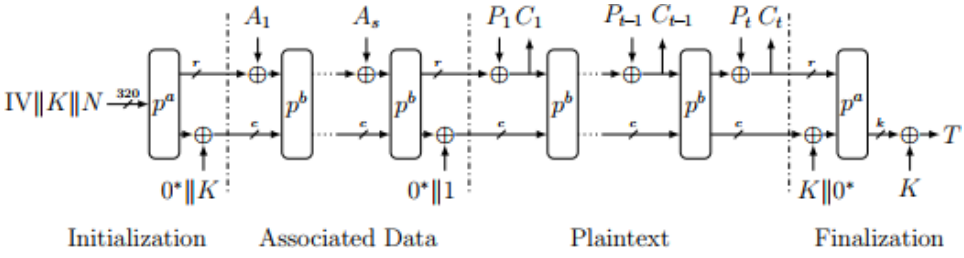
$$E_{a,b,k,r}(K, N, A, P) = (CT)\,, \tag{1}$$

where a, b are the numbers of permutation rounds, r is the state size and k is the secret key size. The outputs of this process are the ciphertext C and the authentication tag T.[18]

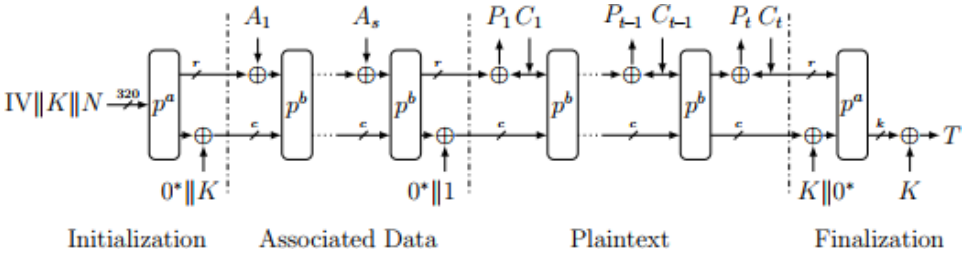
Figure 4 illustrates ASCON modes of operation which are the encryption mode and the decryption mode. P block is the main block in ASCON algorithm. P block has two flavors: one for carrying out the initialization/finalization process (P^{a}) and the other for performing the internal processes (P^{b}).[17]

5.3. *CLOC algorithm*

Compact Low-Overhead Cipher Feedback (CLOC CFB) is a block cipher mode of operation. CLOC ensures authentication and secure encryption through dealing with associated data. CLOC design aims to optimize some factors such as complexity,



(a) Eneryption



(b) Decryption

Fig. 4. ASCON's modes of operation.[18]

overhead and memory requirement. These factors are the drawbacks of previously implemented algorithms such as: Counter with Cipher Block Chaining Message Authentication Code (CCM), Encryption-then-Authentication-then-Translate (EAX) and Encryption-then-Authentication-then-Translate-Prime (EAX-prime). CLOC is designed to deal with small input data width such as 16 bytes which is suitable for small microprocessors with word size of 8 bits or 16 bits. CLOC uses two block cipher modes: CFB for encryption part and Cipher Block Chaining Message Authentication Code (CBC-MAC) for the authentication part. These modes are responsible for providing data in a confidential and authenticated manner as discussed in Refs. 19 and 20.

CLOC implementation consists of four main blocks which represent the cipher core as portrayed in Fig. 5. These blocks are HASH, pseudo-random function (PRF), encryption engine (ENC) and decryption engine (DEC). HASH block uses hash functions to generate the IV. PRF block generates tag which is used in authentication. ENC converts plaintext to ciphertext. DEC converts ciphertext back to plaintext as presented in Refs. 19 and 20. CLOC uses AES-128 to perform the encryption process.

5.4. *JOLTIK algorithm*

JOLTIK is a symmetric and authenticated encryption algorithm. JOLTIK utilizes AES-based permutation layers. JOLTIK parameters such as message and associated data are represented in blocks of size 64 bits. Each block is organized as 4×4 matrix of nibbles.[21] JOLTIK-BC supports two modes of operation: encryption mode and decryption mode. ECB is the used mode to accomplish encryption and decryption

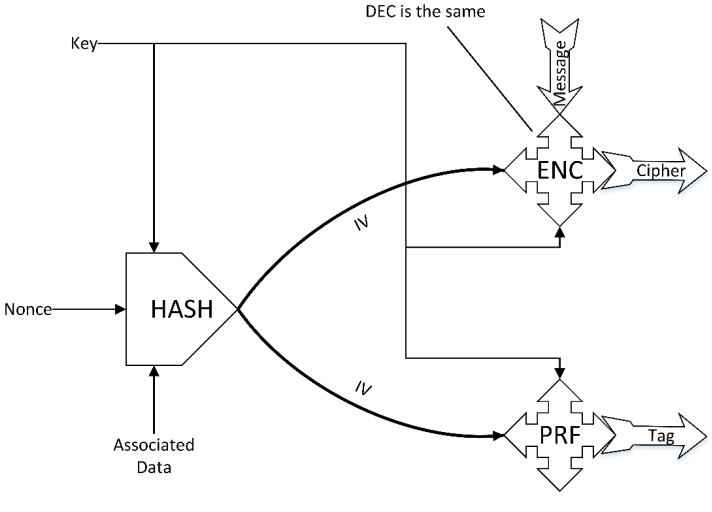


Fig. 5. Block diagram of CLOC.[19]

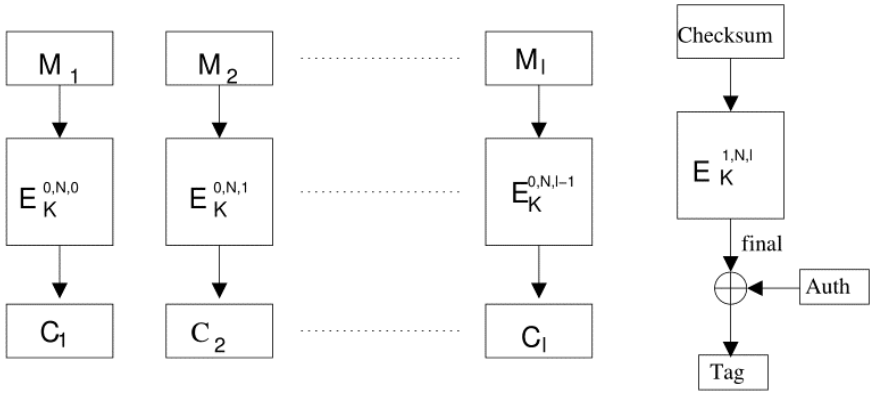


Fig. 6. JOLTIK block diagram of processing the message.[22]

processes. JOLTIK modes of operation require the implementation of a RAM to store all round keys because they are used in reversed order for decryption. This RAM implementation increases decryption latency overhead and consumes extra area.[13] For encryption process, the message is split into 64-bit blocks, each block is processed independently as shown in Fig. 6. For authentication, the associated data is processed to generate the authentication signal, which is XORed with the message blocks to generate the tag.[21]

JOLTIK uses tweakable block cipher (TBC) that uses an extra input denoted by tweak. The internal structure of the tweakable block cipher is similar to the AES block cipher. The TWEAKEY is defined as the concatenation of the key and the tweak. JOLTIK has more than one standard version that uses variety of key and tweak sizes, as given in detail in Ref. 21. JOLTIK-192 is the conducted version in this paper, where it has 128-bit key size and 64 bits for tweak. JOLTIK-192 gives a higher performance against brute force attack (BFA) as the number of attack iterations gets larger.[13] For processing a single block, JOLTIK-192 needs 32 rounds[21] and each round consists of four permutation layers: add round key, substitution layer, shift rows and diffusion layer.

5.5. *MORUS algorithm*

MORUS algorithm achieves encryption and authentication simultaneously.[22] MORUS is designed to target high-speed hardware implementation as it uses only shift, AND and XOR in its operation. MORUS has two internal state sizes: 640 bits and 1,280 bits, where the state is the unit of data that is initialized with the key. MORUS supports two key sizes of 128 bits and 256 bits.[23]

State update function is the main block in MORUS algorithm. It updates the state in each phase of operation and is denoted by State Update (S, M), where S is the state and M is the message block. This function has five rounds where two state

elements are modified in each round: one with left rotation and the other with ROTL function. ROTL function is denoted as ROTL $XY(A, B)$ where A is the block with X-bit size and block A is divided into four Y-bit words and rotates each word left by B bits.[24]

MORUS performs data processing in four phases: initialization, processing the associated data, encryption and finalization, as depicted in Fig. 7. Initialization is the first phase where the key and IV are loaded into state, then the state update function is run for 16 times and finally the key is XORed with the output. Initialization phase is designed to run 80 rounds so that it makes sure the IV and the key are kept secret by mixing them inside the state. In addition, each key and IV are used only once in the protection of any message. Therefore, the IV is not used for the same key to avoid any attacks on recovering the state. Second phase is processing the AD where the associated data is processed using the state update function which is run u times where u is AD length divided by 256. Encryption is the third phase where the plaintext is encrypted into blocks of size 256 bits and the state is updated for v times where v is the message length divided by 256. The last phase is finalization where authentication tag is generated by running state update function eight times.[24]

5.6. *PRIMATEs algorithm*

PRIMATEs family of authenticated ciphers is a sponge-based AE scheme that operates in a low resource environment and provides resistance to forgery attacks and differential power analysis (DPA) side-channel attacks (SCAs). There are three modes of operation in PRIMATEs: APE, HANUMAN and GIBBON. GIBBON has two security levels: 120 bits and 80 bits, where the size of the key is equal to the security level. This paper used GIBBON-80 as it is intended for lightweight applications[25] because of supporting low memory feature by storing only one intermediate state without revealing the key to the attacker.[26] GIBBON consumes less area than the other AEAD algorithms that are based on a block cipher, because it does not

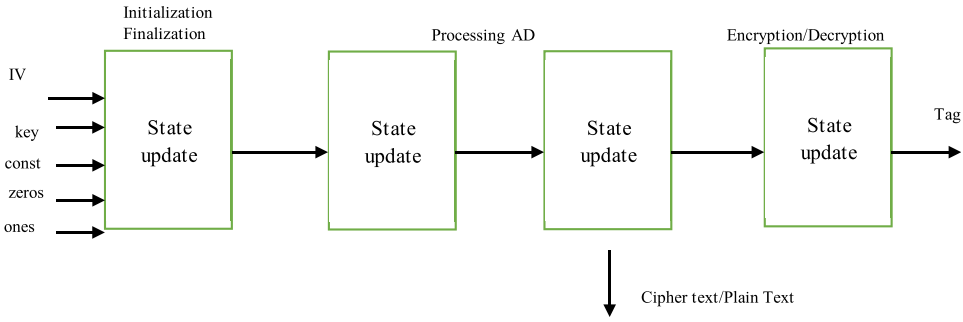


Fig. 7. Block diagram illustrates MORUS implementation.[24]

contain a key schedule, uses smaller S-box (5 bits instead of 8 bits) and uses a more compact, recursive maximum distance separable (MDS) matrix implementation.

PRIMATEs-80 operates on a 5×8 state matrix of 5-bit elements: the first row of the state is the rate of the state (40 bits), whereas the rest of the state is the capacity (160 bits). PRIMATEs family consists of four permutations: P1, P2, P3 and P4 which are defined by different numbers of rounds and different initial round constants (RCs), generated by a 5-bit linear-feedback shift register (LFSR). GIBBON uses six-round permutations P2 and P3 to process the associated data and the message, respectively, and 12-round permutation P1 for initialization and finalization. Each round of the four permutations contains four transformations that update the state matrix. The first transformation is sub-elements (SE) transformation, which substitutes every (5-bit) element in the state matrix with an element from S-box. The second transformation is SR, which shifts each row in the state matrix with a different value. The third transformation, MC, which follows a wide trail strategy, is a left multiplication by a 5×5 matrix in GF as shown in Fig. 8. The fourth transformation is constant addition (CA) transformation, which XORed the second element of the second row with a predefined round constant. This round constant is generated using a Fibonacci linear-feedback shift register with an initial value that differs for each permutation. GIBBON runs P2 if at least one block of AD is present, or when it does not execute any permutations for the AD segment. The state matrix is 200 bits, which leaves a small memory footprint during execution and minimizes processor usage.[25] However, most of the power is consumed in this matrix as it is updated continuously after each round.

5.7. *SCREAM algorithm*

SCREAM is an iterative algorithm which includes masking scheme in order to be robust against the SCA in addition to BFA. Masking is a scheme of performing operations on random bits of an input vector such that the individual bits of the vector are not analyzed subsequently to recover the encrypted data.

The confidentiality mode for SCREAM is ECB which helps parallel computation of multiple blocks.[27] Moreover, it is a TBC with 128-bit key and 128-bit tweak.

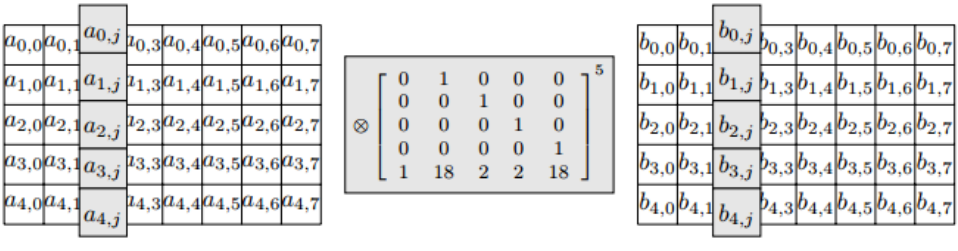


Fig. 8. Mix columns transformation for PRIMATEs-80. The state matrix is multiplied column by column with the 5×5 matrix.[25]

In TBC, the encryption algorithm uses the key as an input in addition to a new one called the tweak (T). Changing T is cheaper and easier than changing the key and makes the algorithm more secure.[28] T is updated in each step of the block cipher as a function of several variables including secret key, block number (j), mode of operation (i.e., message, associated data or tag) and PMN.[28] The PMN is an 88-bit number which acts as the IV to the tweak update block.

Figure 9 shows the cipher core of SCREAM algorithm where the main block of it is the encryption step (E_k), where $k = 0, 1, 2, \ldots,$ [Number of Steps (NS)-1], and NS varies from 8 steps to 12 steps. Each round is an LS cipher (L for L-box and S for S-box), which consists of linear diffusion box, bit slice substitution box and round constant table. The round constant and the L-box are represented by look-up tables. This LS cipher is employed in order to retain linearity for efficient masking and reduce the computational load (i.e., reduce the clock cycle needed for a step).[29]

5.8. *SILC algorithm*

SILC is an authenticated cipher. SILC uses CFB and CBC-MAC modes of operation. CBC-MAC is used for processing associated data and ciphertext, and CFB is used for

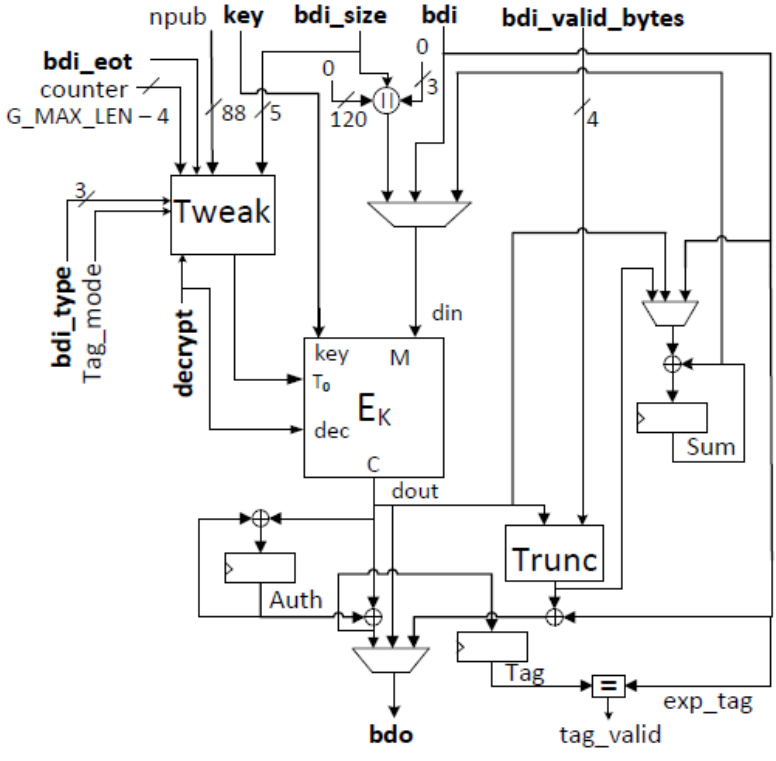


Fig. 9. SCREAM algorithm cipher core.[28]

generating ciphertext, as discussed in Ref. 30. SILC uses AES-128 block cipher which improves latency and memory utilization more than LED and PRESENT block ciphers.[22] SILC provides provable security against birthday attack (i.e., type of cryptographic attack that employs the mathematics of the birthday paradox) because of the pseudo-randomness of the block cipher (i.e., S-box in AES allows this pseudo-randomness).[30,31] The encryption and decryption operations are done using only the encryption function. Both encryption and decryption are online operations which means that every output block depends on all the previous input blocks. The only pre-computation in SILC is the round keys for key scheduling of the block cipher.

For this reason, no extra hardware register is needed for storing the pre-computed result. SILC avoids using a GF multiplier except in AES encryption function which reduces the area as GF multiplier requires large number of gates.[32] The SILC power consumption is dominated by the S-box block due to its large size ($16 \times 16 \times 8$) in look-up table implementation.[13,33]

SILC implementation is based on four subroutines: HASH, encryption (ENC), PRF and decryption (DEC). These subroutines process the data sequentially, however, ENC and PRF perform their function in parallel.[32] The order of these subroutines is based on whether the operation is encryption or decryption as shown in Fig. 10. HASH generates an intermediate tag that is used in the other subroutines.

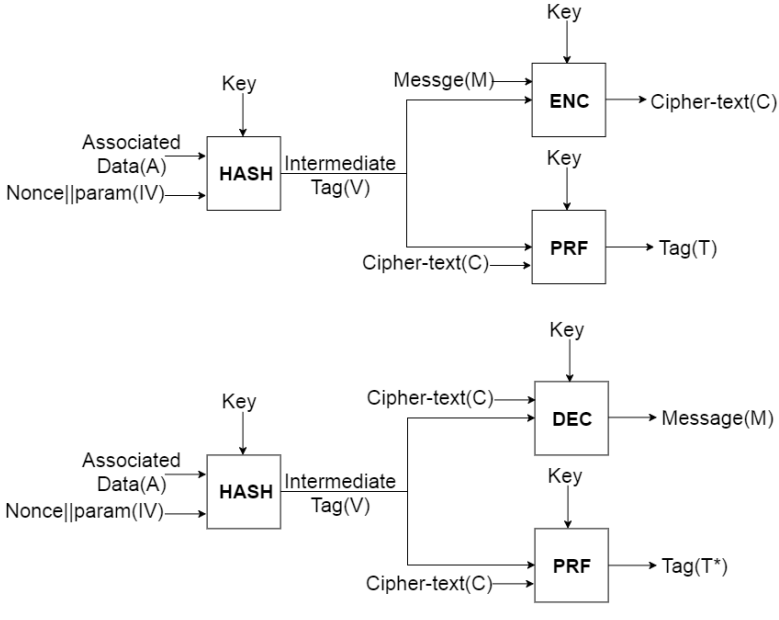


Fig. 10. SILC encryption and decryption block diagrams.[30]

PRF generates two different tags that are used in encryption and decryption operations. In decryption mode, these two tags are compared in order to determine the authentication result. In case of a mismatch, the output message is discarded. Each subroutine has different inputs to send to the AES encryption block as discussed in Ref. 30.

6. FPGA/ASIC Implementations

The difficulty of comparing the hardware performances of the different candidates of CAESAR competition arises due to the existence of various hardware platforms. As a result of this complexity, unified methods are used to give more accurate results for fair comparison. Very-high-speed integrated circuit Hardware Description Language (VHDL) is the used hardware description language (HDL) to implement algorithms in register transfer level (RTL). For low-power IoT applications, the operating frequency is chosen to be 10 MHz for all algorithms. Algorithms are verified using ModelSim functional simulator.

6.1. *Implementation on FPGA*

FPGA implementation of the candidates is performed using Xilinx Vivado 2016.4 design suite. The algorithms are synthesized using Zynq-7000 XC7Z020 FPGA device. Vivado tool is used to perform the logic synthesis, mapping, placing and routing. Vivado results report the areas and power consumptions of the algorithms. For power consumption measurements using FPGA, three parameters are defined namely the effective load capacitance of resources, the switching activity of resources and the thermal information. A method is proposed to adopt all parameters with different cases. The selected cases are chosen to provide fair comparison and analysis for low-power IoT applications.

6.2. *Implementation on ASIC*

Synthesis step is done using Synopsys DC B-2008.09 for Linux. CMOS UMC 130-nm technology with eight metal layers is the used technology for synthesis and place and route steps. DC takes RTL codes, technology libraries and constraints file as an input and produces the gate-level netlist as an output. The switching activity file generated from Vivado is included for accurate power consumption results. After the synthesis step is completed, auto place and route (APR) is carried out such that the standard cells are placed and routed and connected to input/output (I/O) pins, and the clock tree synthesis is performed.[34] APR is achieved using Cadence SoC Encounter 8.1 tool. SoC Encounter converts the gate-level netlist into layout. Figure 11 shows the SoC Encounter chip layouts of the eight cryptographic algorithms.

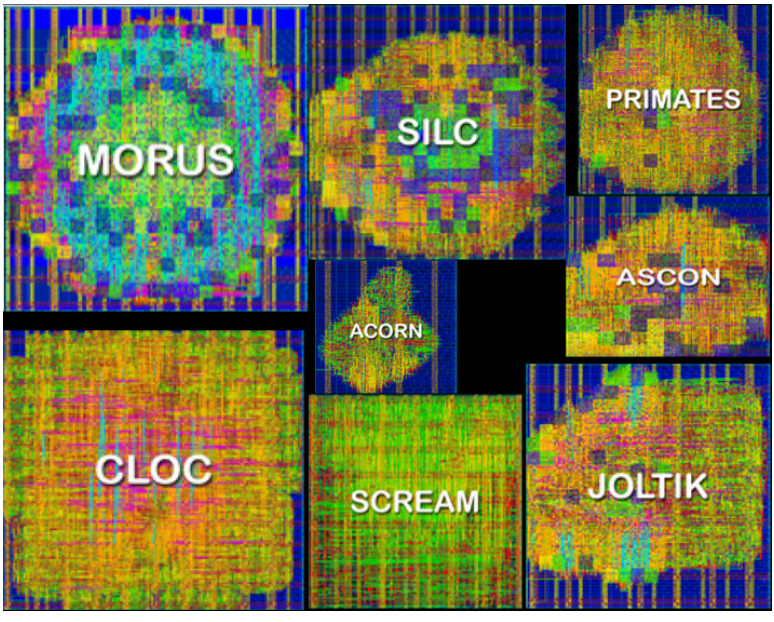


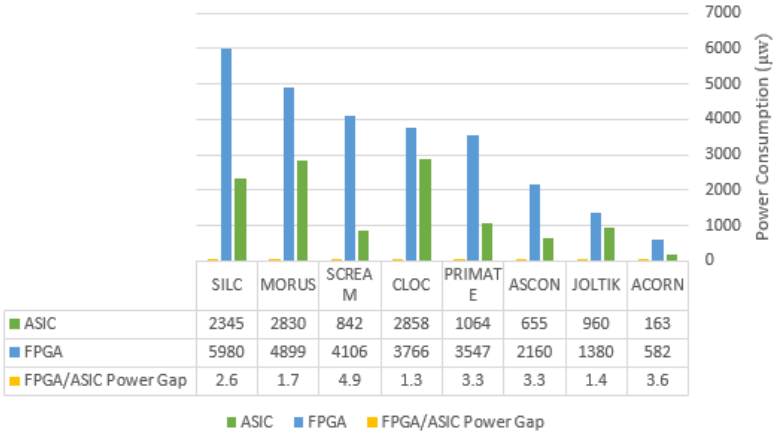
Fig. 11. SoC Encounter chip layouts of the eight cryptographic algorithms.

7. Results and Discussion

Various parameters are compared in order to find out the most suitable algorithm for low-power IoT applications. These parameters are: FPGA power, slice LUTs utilization, throughput, ASIC power, ASIC area and latency.

7.1. *FPGA and ASIC powers*

Figure 12 shows the power consumptions of the FPGA and ASIC implementations for the eight algorithms as well as the FPGA/ASIC power gap. FPGA/ASIC power gap is defined as the ratio between FPGA power and the ASIC power. It is found that CLOC algorithm has the smallest power gap, while the SCREAM algorithm has the largest power gap. FPGA and ASIC powers should be theoretically linear with each other. However, the reason behind this difference is ASIC power takes into consideration the interconnection and other physical effects that reflect on the dynamic power. Moreover, the technology used for ASIC flow differs from the one used for Zynq FPGA board. It should be noted that a design that is implemented using ROMs consumes much larger power for FPGA, such as the SCREAM algorithm. Figure 12 shows that SILC consumes the largest power for FPGA, and CLOC consumes the largest power for ASIC. Also, it shows that ACORN algorithm consumes the minimum power for both ASIC and FPGA implementations, as it is the only candidate that is based on stream ciphers.

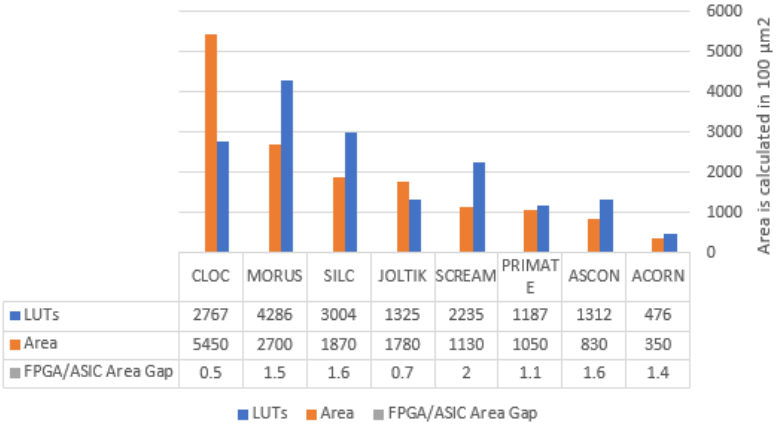


	SILC	MORUS	SCREAM	CLOC	PRIMATE	ASCON	JOLTIK	ACORN
ASIC	2345	2830	842	2858	1064	655	960	163
FPGA	5980	4899	4106	3766	3547	2160	1380	582
FPGA/ASIC Power Gap	2.6	1.7	4.9	1.3	3.3	3.3	1.4	3.6

Fig. 12. FPGA and ASIC power estimation results.

7.2. *Area and slice LUTs*

Figure 13 shows the FPGA utilizations, ASIC areas and FPGA/ASIC area gaps. Several parameters affect area estimation such as block size, key size, tag size, number of rounds and bus width. MORUS algorithm has the largest LUTs utilization because of large block size, while the CLOC algorithm consumes the largest area in the ASIC implementation. ACORN algorithm has the smallest area in both FPGA and ASIC implementations because of the small bus width. There is an area gap between FPGA and ASIC implementations because of the different hardware implementations of LUTs and the standard cells in the FPGA and ASIC flows, respectively. Figure 13 illustrates that the CLOC algorithm has the smallest area gap, while the SCREAM algorithm has the largest area gap.



	CLOC	MORUS	SILC	JOLTIK	SCREAM	PRIMATE	ASCON	ACORN
LUTs	2767	4286	3004	1325	2235	1187	1312	476
Area	5450	2700	1870	1780	1130	1050	830	350
FPGA/ASIC Area Gap	0.5	1.5	1.6	0.7	2	1.1	1.6	1.4

Fig. 13. FPGA utilization and ASIC area results.

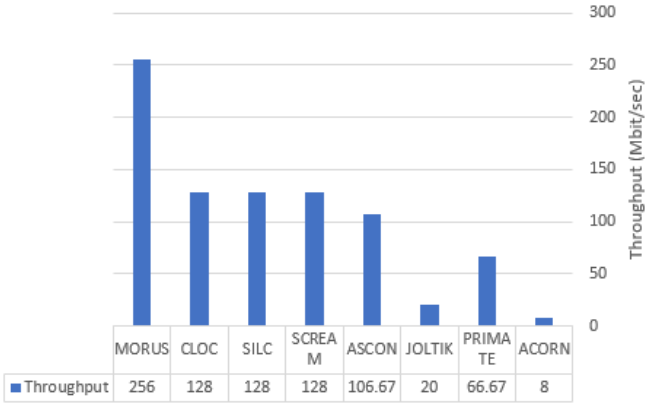


Fig. 14. Throughput results.

7.3. *Throughput*

Figure 14 shows the throughput results for the candidates in Mbits/s. The formula that is used to calculate the throughput is given by[35]

$$\text{Throughput} = \frac{\text{Block size}}{\text{Number of rounds} + C} * \text{Frequency}\,, \tag{2}$$

where C takes two values, either 0 or 1, and it represents the additional clock cycle used for initialization at the beginning of each round. Block size, number of rounds and frequency are the parameters which determine the throughput and these parameters are constant for each of the selected algorithm. MORUS algorithm gives the highest throughput among the selected algorithms because of the large block size. However, ACORN has the smallest throughput.

7.4. *Latency*

In order to provide fair comparison among the selected algorithms which have various message and associated data lengths, a formula is developed to calculate the latency. The latency is calculated starting from the beginning of applying the message till the first output data appears. The pipeline of pre-processor and post-processor blocks makes the latency and the throughput not proportional to each other, and then the throughput cannot be evaluated using the developed formula. Figure 15 illustrates the average latency between encryption and decryption processes. The formulas used to calculate latency of encryption and decryption processes are given by

$$\text{Latency}_{\text{ENC}} = \frac{T_{\text{ENC}}/T\text{Clk}}{(M + A) * \text{Block bytes}}\,, \tag{3}$$

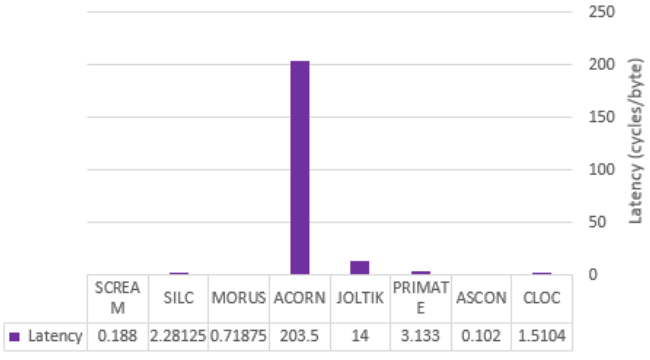


Fig. 15. Latency results.

$$\text{Latency}_{\text{DEC}} = \frac{T_{\text{DEC}}/T\text{Clk}}{(C + A) * \text{Block bytes}}, \tag{4}$$

where T_{ENC} and T_{DEC} are the required intervals to accomplish encryption and decryption processes, respectively, block bytes are the number of bytes per block and A, M and C are calculated using the ceil function for number of bytes of associated data, plaintext, ciphertext divided by block bytes, respectively. Pre-synthesis and post-synthesis results of latency for FPGA and ASIC flows are the same, which is because the timing constraints are satisfied in addition to the used clock cycle of 100 ns providing enough time to process data even with external LUTs or standard cells delays. Figure 15 shows that ACORN algorithm gives the largest latency because of small block size, while the ASCON algorithm provides the smallest latency.

8. Comparison

Among all AES-based candidates, the algorithm that uses large S-box consumes relatively large power. For the case of JOLTIK and SILC algorithms, both of them are AES-based permutation layers that include the substitution layer. The S-box size for SILC is the same as the case in AES, which is 256×8,[13] and for JOLTIK the size is 16×4.[21] This large S-box makes SILC to consume larger power and area than JOLTIK. ACORN, which is the only stream cipher among the selected algorithms, consumes the minimum power and area, while giving a small throughput in the range of Kbits/s. For low-power IoT applications, ACORN algorithm is suggested to keep the data secure in case when the used protocols in the system do not require high data rates.

In order to evaluate the submissions of CAESAR competition, comparisons are carried out among existing encryption algorithms, and the presented algorithms in this paper. Table 1 holds the comparison in ASIC flow, in terms of frequency, area,

Table 1. Comparison between the studied designs and previous designs in ASIC flow.

Design	Frequency (MHz)	Area (mm^2)	Power (mW)	Technology (nm)
AES-128[36]	10	0.148	0.724	UMC 130
AES-192[36]	10	0.149	0.729	UMC 130
AES-256[36]	10	0.149	0.727	UMC 130
RSA[36]	10	1.236	9.38	UMC 130
3DES[36]	10	0.217	0.968	UMC 130
Twofish[36]	10	0.101	0.675	UMC 130
ACORN	10	0.035	0.163	UMC 130
JOLTIK	10	0.178	0.96	UMC 130
ASCON	10	0.083	0.655	UMC 130
PRIMATEs	10	0.106	1.064	UMC 130
CLOC	10	0.544	2.858	UMC 130
SCREAM	10	0.114	0.842	UMC 130
MORUS	10	0.27	2.83	UMC 130
SILC	10	0.187	2.345	UMC 130

dynamic power and technology used. The ASIC implementation of RSA algorithm introduced in Ref. 36 consumes the largest area and power, while the ACORN algorithm is the lowest in power and area.

Table 2 compares the same aspects in FPGA flow among the selected algorithms and previous cryptography algorithms. This comparison is based on frequency, LUTs, power and type of FPGA used. The implementation of 3DES presented in Ref. 36 consumes large power compared to the selected ACORN algorithm which has the smallest power results. The Twofish algorithm introduced in Ref. 36 has the largest LUT utilization, while the presented ACORN algorithm consumes the smallest number of LUTs. Both tables highlights that ACORN has the lowest in power, area and resources utilization, compared to the other existing algorithms at the expense of low speed and high latency.

Table 2. Comparison between the studied designs and previous designs in FPGA flow.

Design	Frequency (MHz)	LUTs	Power (mW)	FPGA
AES[36]	10	961	246	Zynq-7000 XC7Z020
RSA[36]	10	1,178	255	Zynq-7000 XC7Z020
3DES[36]	10	1,191	125	Zynq-7000 XC7Z020
Twofish[36]	10	556	121	Zynq-7000 XC7Z020
ACORN	10	476	0.582	Zynq-7000 XC7Z020
JOLTIK	10	1,325	1.38	Zynq-7000 XC7Z020
ASCON	10	1,312	2.16	Zynq-7000 XC7Z020
PRIMATEs	10	1,187	3.547	Zynq-7000 XC7Z020
CLOC	10	2,767	3.766	Zynq-7000 XC7Z020
SCREAM	10	2,235	4.106	Zynq-7000 XC7Z020
MORUS	10	4,286	4.899	Zynq-7000 XC7Z020
SILC	10	3,004	5.980	Zynq-7000 XC7Z020

9. Conclusion

This paper presents the comparative study of FPGA and ASIC implementations of security algorithms selected from CAESAR competition that are served for IoT applications. The chosen algorithms are ACORN, MORUS, JOLTIK, PRIMATEs, SILC, CLOC, SCREAM and ASCON. The FPGA implementation is conducted using ZC702 evaluation board for the Zynq-7000 XC7Z020. The synthesis and the place and route steps are performed using Xilinx Vivado 2016.4 tool, while the ASIC approach is done using CMOS UMC 130-nm technology. The synthesis step is performed using Synopsys Design Compiler tool. Cadence SoC Encounter tool is utilized for place and route step. The comparative study analyzes several issues that are critical for low-power IoT applications such as power consumption, area, throughput and latency. Also, a comprehensive investigation about CAESAR competition and the GMU-API is conducted.

ACORN algorithm is recommended for low-power IoT applications because it consumes the minimum power among the eight selected algorithms. The AES-based algorithms SILC, CLOC and JOLTIK consume extra power according to the size of S-box. Furthermore, ACORN utilizes the smallest LUTs/area in the FPGA/ASIC implementations. In terms of throughput, ACORN gives the minimum throughput in range of Kbits/s which makes ACORN not preferred for high data rate applications. While the MORUS algorithm gives the highest throughput because of the large block size. MORUS algorithm is recommended for high data rate applications. ASCON algorithm gives the smallest latency, however, ACORN algorithm gives the largest latency because of small block size. This work concludes that ACORN algorithm is the most suitable algorithm for low-power IoT application because the power consumption is the main concern.

Acknowledgment

This work was supported by the Egyptian Information Technology Industry Development Agency (ITIDA) under ITAC Program.

References

1. R. Roman, P. Najera and J. Lopez, Securing the Internet of Things, *Computer* **44** (2011) 51–58.
2. J. Gubbi, R. Buyya, S. Marusic and M. Palaniswami, Internet of Things (IoT): A vision, architectural elements, and future directions, *Fut. Gener. Comput. Syst.* **29** (2013) 1645–1660.
3. K. Sha, R. Errabelly, W. Wei, T. A. Yang and Z. Wang, EdgeSec: Design of an edge layer security service to enhance IoT security, *Proc. IEEE 1st Int. Conf. Fog and Edge Computing (ICFEC)* (2017), pp. 81–88.
4. A. Asaduzzaman, M. F. Mridh and M. N. Uddin, An inexpensive plug-and-play hardware security module to restore systems from malware attacks, *Proc. 2013 Int. Conf. Informatics, Electronics and Vision (ICIEV)* (2013), pp. 1–5.

5. D. Minoli, K. Sohraby and B. Occhiogrosso, IoT Security (IoTSec) mechanisms for e-health and ambient assisted living applications, *Proc. 2017 IEEE/ACM Int. Conf. Connected Health: Applications, Systems and Engineering Technologies (CHASE)* (2017), pp. 13–18.
6. International Cryptologic Research Community, CAESAR Submissions (2017), https://competitions.cr.yp.to/caesar-submissions.html.
7. Z.-K. Zhang, M. C. Y. Cho, C.-W. Wang, C.-W. Hsu, C.-K. Chen and S. Shieh, IoT security: Ongoing challenges and research opportunities, *Proc. 2014 IEEE 7th Int. Conf. Service-Oriented Computing and Applications (SOCA)* (2014), pp. 230–234.
8. International Cryptologic Research Community, Introduction to Cryptographic Competitions (2014), https://competitions.cr.yp.to/index.html.
9. International Cryptologic Research Community, CAESAR Timeline (2018), https://competitions.cr.yp.to/caesar.html.
10. E. Homsirikamol, W. Diehl, A. Ferozpuri, F. Farahmand, M. U. Sharif and K. Gaj, GMU hardware API for authenticated ciphers (2015), IACR Cryptology ePrint Archive, https://eprint.iacr.org/2015/669.pdf.
11. NIST, Announcing the Advanced Encryption Standard (AES) (2001), Federal Information Processing Standards Publication No. 197, https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.197.
12. S. M. Soliman, B. Magdy and M. A. A. El Ghany, Efficient implementation of the AES algorithm for security applications, *Proc. 2016 29th IEEE Int. System-on-Chip Conf. (SOCC)* (2016), pp. 206–210.
13. C. Paar and J. Pelzl, *Understanding Cryptography: A Textbook for Students and Practitioners* (Springer Science & Business Media, 2009).
14. H. Wu, ACORN: A lightweight authenticated cipher (v2) (2015), https://competitions.cr.yp.to/round2/acornv2.pdf.
15. U. Mamidi, Lightweight authenticated encryption for FPGAs, Master's thesis, George Mason University (2016).
16. H. Wu, ACORN: A lightweight authenticated cipher (v3) (2016), https://competitions.cr.yp.to/round3/acornv3.pdf.
17. G. Bertoni, J. Daemen, M. Peeters and G. Van Assche, Cryptographic sponge functions (2011), Submission to NIST (Round 3), https://keccak.team/files/CSF-0.1.pdf.
18. M. Fivez, Energy efficient hardware implementations of CAESAR submissions, Master's thesis, KU Leuven (2016) [https://www.esat.kuleuven.be/cosic/publications/thesis-279.pdf].
19. C. Dobraunig, M. Eichlseder, F. Mendel and M. Schläffer, ASCON v1.2: Submission to the CAESAR Competition (2016), https://competitions.cr.yp.to/round3/asconv12.pdf.
20. T. Iwata, K. Minematsu, J. Guo and S. Morioka, CLOC: Authenticated encryption for short input, *Proc. Int. Workshop Fast Software Encryption* (Springer, Berlin, 2015), pp. 149–167.
21. T. Iwata, K. Minematsu, J. Guo and S. Morioka, CLOC: Compact low-overhead CFB (2014), https://competitions.cr.yp.to/round1/clocv1.pdf.
22. J. Jean, I. Nikolić and T. Peyrin, JOLTIK v1.3 (2015), https://competitions.cr.yp.to/round2/joltikv13.pdf.
23. H. Wu and T. Huang, The authenticated cipher MORUS (v1) (2014), https://competitions.cr.yp.to/round1/morusv1.pdf.
24. A. Mileva, V. Dimitrova and V. Velichkov, Analysis of the authenticated cipher MORUS (v1), *Proc. Int. Conf. Cryptography and Information Security in the Balkan* (2015), pp. 45–59.

25. E. Andreeva *et al.*, PRIMATEs v1: Submission to CAESAR Competition (2014), https://competitions.cr.yp.to/round1/primatesv1.pdf.
26. M. Agrawal, D. Chang and S. K. Sanadhya, A new authenticated encryption technique for handling long ciphertexts in memory constrained devices, *Int. J. Appl. Cryptogr.* **3** (2017) 236–261.
27. M. Vaidehi and B. J. Rabi, Design and analysis of AES-CBC mode for high security applications, *Proc. 2014 2nd Int. Conf. Current Trends in Engineering and Technology (ICCTET)* (2014), pp. 499–502.
28. J. Jean, I. Nikolić and T. Peyrin, Tweaks and keys for block ciphers: The TWEAKEY framework, *Proc. Int. Conf. Theory and Application of Cryptology and Information Security* (2014), pp. 274–288.
29. W. Diehl and K. Gaj, RTL implementations and FPGA benchmarking of three authenticated ciphers competing in CAESAR round two, *Proc. 2016 Euromicro Conf. Digital System Design (DSD)* (2016), pp. 91–98.
30. V. Grosso, G. Leurent, F.-X. Standaert and K. Varıcı, LS-designs: Bitslice encryption for efficient masked software implementations, *Proc. Int. Workshop Fast Software Encryption* (2014), pp. 18–37.
31. T. Iwata, K. Minematsu, J. Guo, S. Morioka and E. Kobayashi, SILC: Simple Lightweight CFB (2014), https://competitions.cr.yp.to/round1/silcv1.pdf.
32. P. S. Barreto, H. Y. Kim and V. Rijmen, Toward secure public-key blockwise fragile authentication watermarking, *IEE Proc., Vis. Image Signal Process.* **149** (2002) 57–62.
33. T. Iwata, K. Minematsu, J. Guo, S. Morioka and E. Kobayashi, CLOC and SILC (2016), https://competitions.cr.yp.to/round3/clocsilcv3.pdf.
34. S. Morioka and A. Satoh, An optimized S-box circuit architecture for low power AES design, *CHES 2002: Cryptographic Hardware and Embedded Systems* (Springer, Berlin, 2002), pp. 172–186.
35. H. B. Kommuru and H. Mahmoodi, ASIC Design Flow Tutorial Using Synopsys Tools, Tutorial, Nano-Electronics & Computing Research Lab, School of Engineering, San Francisco State University, San Francisco (2009) [available at: http://read.pudn.com/downloads188/ebook/884100/ASIC_Design_Flow_Tutorial_with_synopsys.pdf].
36. E. Homsirikamol, W. Diehl, F. Farahmand, A. Ferozpuri and K. Gaj, C vs. VHDL: Benchmarking CAESAR candidates using high-level synthesis and register-transfer level methodologies (2015), Directions in Authenticated Ciphers (DIAC), https://cryptography.gmu.edu/athena/presentations/DIAC_2015_gaj_HLS.pdf.
37. M. A. Bahnasawi, K. Ibrahim, A. Mohamed, M. Khalifa, A. Moustafa, K. Abelmonim, Y. Ismail and H. Mostafa, ASIC-oriented comparative review of hardware security algorithms for Internet of Things applications, *Proc. IEEE Int. Conf. Microelectronics (ICM 2016)* (IEEE, 2016), pp. 285–288.