International Journal of
Advanced Science and Convergence

ISSN : 2619-8150
Volume 3, Number 1, March 2021
https://doi.org/10.22662/IJASC.2021.3.1.011

# The Hardware Implementation of NIST Lightweight Cryptographic Candidate SpoC for IoT Devices

**Dennis Agyemanh Nana Gookyi[1] and Kwangki Ryoo[2]**

[1]Student, Department of Information and Communication Engineering, Hanbat National University, Daejeon, 34155, South Korea

[2]Professor, Department of Information and Communication Engineering, Hanbat National University, Daejeon, 34155, South Korea

## *Abstract*

**Background/Objectives**: This paper presents the hardware implementation of SpoC Lightweight Cryptography (LWC) candidate for low-cost devices. **Methods/Statistical analysis**: The proposed hardware implementation of the SpoC Authenticated Encryption with Associated Data (AEAD) is capable of both encryption and decryption. The design was implemented on the Virtex-4 Field FPGA using Xilinx ISE Design Suite. The synthesis results reported 951 slices at 246.61 MHz maximum clock frequency. The encryption and decryption routines take 589 and 590 cycles. **Findings**: This work used 30% fewer LUTs for both encryption and decryption as compared to the existing work. The decrease in area in this work is a result of the optimization of the implementation for low-cost devices while the existing work implemented the basic iterative architecture without optimizing. This work achieved almost two times the frequency of the existing SpoC implementation. The existing SpoC implementation used 150 fewer cycles as compared to this work. This is because this work implemented the SpoC using extra registers in other to reduce the critical path which increases the frequency. **Improvements/Applications**: For future works, the proposed SpoC AEAD will be combined with other security protocols to form a lightweight cryptographic System-on-Chip (SoC).

## *Index Terms*

NIST, Lightweight Cryptography, AEAD, SpoC, FPGA

**Corresponding author : Kwangki Ryoo**
kkryoo@gmail.com

• Manuscript received January 28, 2021.
• Revised February 26, 2021 ; Accepted March 20, 2021.
• Date of publication March 31, 2021.

## I. INTRODUCTION

The Internet of Things (IoT) is said to be the fourth industrial revolution [1]. This is a technological revolution that has significantly changed the way we interact with others, live, and work. The backbone of IoT is composed of extremely low cost and resource-constrained devices that gather, process, store and transmit sensitive information. The devices are constrained in terms of computational complexity, power, and storage to reduce the cost of mass deployment. Since communication among IoT devices is mainly through insecure channels, security issues such as confidentiality, integrity, and authenticity [2] can be applied to the devices. Confidentiality deals with an adversary's ability to access private communication, integrity deals with the ability to detect a change in transmitted information while authenticity deals with the ability to identify the sender of a message. To reduce the cost of securing computing devices, a single algorithm can be used to ensure confidentiality, integrity, and authenticity. This algorithm is known as Authenticated Encryption with Associated Data (AEAD).

For the past decade, there has been a high need for AEAD schemes, and for that matter, NIST issued the Competition for Authenticated Encryption: Security, Applicability, and Robustness (CAESER) contest in January 2013 [3]. The CAESAR contest was to standardize Authenticated Encryption (AE) for conventional computing devices. The contest ended in March 2019 [4] of which six submissions were declared winners. During the CAESAR contest, NIST realized that most of the submissions were not tailored towards low-cost devices like sensors, smart cards, and RFID tags. This lead NIST to issue a call for lightweight AEAD schemes in August 2018 as part of the LWC standardization process [5]. The lightweight AEAD contest has now entered the second round with thirty-two candidates [6] announced in August 2019. The contest is due to end in the year 2021. The submissions for the lightweight AEAD contest are evaluated in resource-constrained environments using hardware metrics such as area, power consumption, memory, throughput, and latency. The NIST evaluation team of the contest has called for third-party hardware implementation of the second-round candidates using Application-Specific Integrated Circuit (ASIC) and FPGA tools.

The first step in the evaluation of a cryptographic algorithm for low-cost devices is to measure the hardware resources consumed by the algorithm. This paper, therefore, proposes and implements a hardware architecture for SpoC lightweight AEAD algorithm [7]. SpoC is one of the submissions that made it into the second round of the NIST lightweight AEAD competition. The rationale for selecting SpoC out of the thirty-two candidates can be summarized as below:

• Security: SpoC employs 128-bit security which is sufficient for low-cost devices. Moreover, several third-party cryptanalyses of the algorithm found no weakness or errors in the specification. SpoC also meets the minimum-security guarantees for confidentiality and integrity that NIST set for the lightweight AEAD competition.

• Underlying construction: SpoC mode of operation is permutation-based which has become popular over the last decade. The efficiency of the permutation-based mode of operation coupled with Keccak [8] (a permutation-based candidate) winning the Secure Hash Algorithm 3 (SHA-3) [9] competition has made algorithms with such constructions preferable.

• Estimated hardware footprint: Most of the algorithms that made it into the second round of the NIST lightweight AEAD competition make use of Substitution-Boxes (S-Boxes) which take up a lot of hardware resources and memory to implement. The SpoC algorithm, however, uses simple shift operations in place of S-Boxes to reduce the hardware area. Moreover, SpoC has a special feature known as inverse free (encryption and decryption routines are the same) which makes it very suitable for low-cost devices.

The main contributions of this work are as follows:

• A compact hardware architecture is implemented for both encryption and decryption routines of the SpoC lightweight AEAD algorithm. The hardware architecture is modeled to meet the resources requirement of low-cost devices.

• The hardware architecture for the SpoC AEAD algorithm is implemented and tested on an FPGA device. The architecture is verified using simulation modules and test results are compared to the latency requirements of devices like RFID tags.

• Hardware synthesis and simulation results such as area, frequency, throughput, latency, and efficiency of the proposed SpoC implementation are compared to the other lightweight AEAD candidates. This is done to prove that SpoC is one of the ideal candidates for securing low-cost devices.

The rest of this paper is organized as follows: Section II provides background on AEAD algorithms and summarizes the candidates that made it into the second round of the lightweight AEAD competition, Section III discusses some related works, Section IV describes the hardware implementation of the SpoC AEAD algorithm, Section V discusses the hardware results of the algorithm, the paper is concluded in Section VI.

## II. BACKGROUND

The idea of an AEAD protocol is the provision of services such as confidentiality, integrity, and authenticity using a single algorithm. This is vital because AEAD protocols reduce cost, increase performance, and eliminates the security flaws of combing different algorithms to achieve confidentiality, integrity, and authenticity. The idea of AEAD was first introduced by Bellare et al [10] and has evolved over the years. The AEAD scheme usually consists of encryption and decryption routines. The encryption routine takes as inputs a fixed-length secret key (K), a fixed-length public nonce (N), a variable length associated data (AD), and a variable-length plaintext (P). The output consists of a variable-length ciphertext (C) and a fixed-length tag (T). The input to the decryption routine consists of a fixed-length key, variable length ciphertext, fixed-length tag, and variable-length nonce. The output consists of the plaintext or error symbol (⊥) depending on the verification tag matching the tag from the sender. The error symbol is invalid data that is outputted to prevent adversaries from accessing the plaintext. An overview of an AEAD scheme is illustrated in Fig. 1.

The majority of AEAD schemes use a block cipher primitive for encryption and various methods for authentication [11, 12, 13]. Some of these schemes have been analyzed and recommended by NIST [11, 12] while others have been standardized [13]. Recent developments have seen the rise in the use of permutation-based functions [14] for AEAD designs. This is mainly due to the fact that the winner of the Secure Hash Algorithm 3 (SHA-3) competition [9] is a permutation-based function known as Keccak [8]. The need for a unified scheme or a family of schemes motivated NIST to initiate the CAESAR competition that produced six winners in March 2019. The CAESAR competition produced schemes that are convenient for widespread adaptation. The CAESAR schemes are however not tailored towards resource-constrained devices like RFID tags and for that matter, NIST organized a lightweight standardization process to select AEAD schemes that are suitable for low-cost devices. The second round of the lightweight AEAD competition is underway and is set to last till the year 2021.

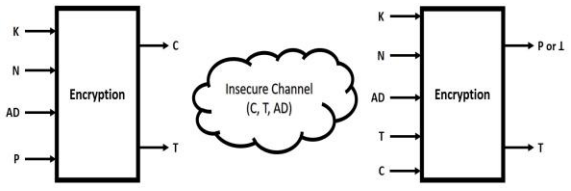


**Fig. 1.** Overview of AEAD Algorithm.

### *A. NIST Lightweight AEAD Competition*

The NIST lightweight AEAD competition was instituted to standardize algorithms for low-cost devices used at the end nodes of IoT platforms. The target devices of the competition consist of devices such as RFID tags, sensors, and embedded systems. The sensors and RFID tags are usually implemented using ASIC technology to meet their constrained features. Embedded systems are designed using microcontrollers with memory as little as16 bytes. For such devices, cryptographic protocols should be designed to consume a minimal amount of hardware resources.

The evaluation criteria [15] used by NIST to seed the lightweight AEAD algorithms include security, cost, performance, and third party analysis. The security evaluations of the algorithms were carried out against known attacks. The submissions were evaluated using cost metrics such as area, memory usage, and energy consumption. The performance metrics include throughput, latency, and power consumption. The third-party analysis metric favored submission with a greater number of third party cryptanalysis.

### *B. NIST AEAD Round 2 Candidates Specification*

Thirty-two algorithms have made it into the NIST lightweight AEAD competition [6]. The total number of researchers who contributed to the thirty-two algorithms totaled 111. The underlying constructions of the candidates include 9 block cipher based submission, 6 tweakable block cipher based submissions, 16 permutation-based submissions, and 1 stream cipher based submission. The thirty-two candidates use 21 unique cryptographic primitives which include AES, GIFT, PYJAMASK, SATURNIN, SKINNY, TweAES, TweGIFT, CLYDE, Grain-128a, Simeck, XOODOO, Welch-Gong, Subterranean, sLiSCP-light, SPARX, PHOTON, SimP, RECTANGLE, Keccap-p, ASCON-p, and GIMLI-24.

A total of 20 unique modes of operation were employed by the AEAD candidates. They include Counter (CTR), Ciphertext Feedback (CFB), Plaintext Feedback (PFB), Hybrid Feedback (HYFB), Output Feedback (OFB), Combined Feedback (COFB), Beetle, Minimally XORed Feedback (mixFeed), Offset Codebook (OCB), Small (Simple, Slim, Sponge based) AEAD from Block cipher (SAEB), Small Universal Deterministic Authenticated Encryption (SUNDAE), JAMBU, Parallel AE from a Forkcipher (PAEF), Cipher Block Chaining (FCBC), Offset Two-Round (OTR),

Sponge One-Pass (S1P), Linear Feedback Shift Register/Non-linear Feedback Shift Register (LFSR/NLFSR), Sponge, Duplex, and DrySponge. The most used mode of operation is the duplex mode which is used by 9 out of the 32 candidates.

Some special features make some submissions standout. The features include parallelizability, online, and inverse free. The inverse free feature reduces the hardware area while the parallelizability and online features reduce latency. From the thirty-two candidate algorithms, four candidates do not exhibit the parallelizability and online features while three candidates do not use the inverse free feature.

This work focuses on implementing a hardware architecture for one of the NIST lightweight AEAD candidates known as SpoC. This work is vital because third party implementation of the candidate algorithms are preferred by the NIST evaluation team. This is because algorithm designers tend to report false hardware results of their implementations.

### *C. SpoC Lightweight AEAD Description*

The SpoC lightweight AEAD algorithm is a sponge based protocol that operates on the Beetle mode [16]. The Beetle mode of operation is where the ciphertext is not part of the inputs to the permutation algorithm. This work focuses on the SpoC's primary recommendation which is SpoC-64. The SpoC-64 has a capacity (c) of 128-bit, state size (b) of 192-bit, nonce size (n) of 128-bit, tag size (t) of 64-bit, and key size (k) of 128-bit. The core primitive of the SpoC-64 AEAD is a lightweight version of the Lighter Sponge-specific Cryptographic Permutations (sLiSCP) [17] family known as sLiSCP-light-192 [18]. This section discusses the sLiSCP-light-192 primitive before the SpoC-64 AEAD algorithm.

### *D. sLiSCP-light-192 Permutation*

The SpoC-64 AEAD is designed using sLiSCP-light-192 permutation with a state size of 192-bit. The sLiSCP-light-192 uses a four-block (each of 48-bit labeled S0, S1, S2, and S3) Generalized Feistel Structure (GFS) [19] with a Simeck Box (SB) [20]. The permutation uses a total of 18 steps with each step using 6 rounds. Each of the 18 steps consists of three stages which include Substitute_Sub_block (SSb), Add_Step_constants (ASc), and Mix_Sub_blocks (MSb) as shown in Fig. 2. The permutation uses two SBs (SB1 and SB3) which are constructed using XOR gates, rotations, and logical AND gates. The SB takes a 48-bit input and produces a 48-bit output. Two 6-bit round constants rc0 and rc1 are applied to SB1 and SB3 respectively while two-step constants (sc0 and sc1) are applied during the ASc stage.

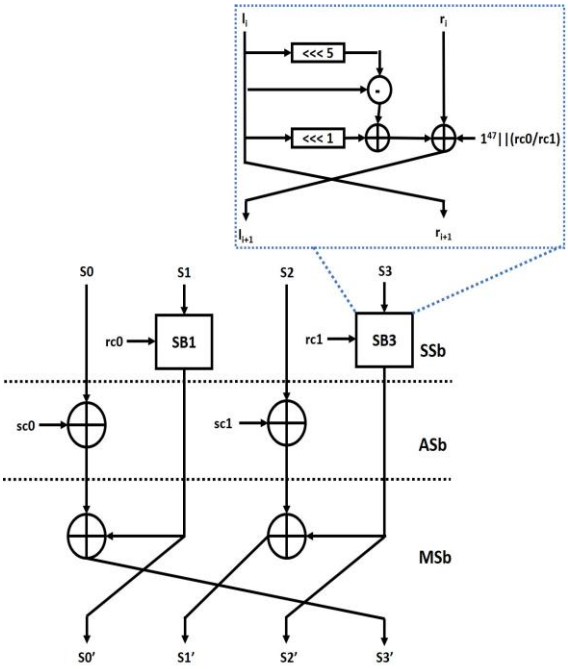


**Fig. 2.** sLiSCP-light-192 Permutation.

### *E. SpoC AEAD Specifications*

The Beetle sponge design of the SpoC-64 AEAD algorithm is shown in Fig. 3. At each step, the state is divided into a concatenation of Y and Z. Four main stages are involved in the construction which includes initialization (init), associated data processing (proc_ad), plaintext/ciphertext processing (proc_pt), and tag processing (proc_tag). A 4-bit control variable (ctrl) is used to distinguish among the various processes. The init process shown in Fig. 3(a) initialises the state using the nonce (N = N0||N1) and the secret key (K = K1||K2). The ctrl code for this process is a 4-bit binary 0000. At the end of the initialization process, Y0 and Z0 are generated. The proc_ad processes the associated data (AD) as shown in Fig. 3(b) using the ctrl code 0010. The proc_pt processes the plaintext (PT) using the ctrl code 0100. This process also generates the ciphertext (CT) as shown in Fig. 3(c). The last process is the proc_tag shown in Fig. 3(d). This has the responsibility of generating the tag using the ctrl code 1000. Detailed descriptions of each process can be found in the SpoC specification document [6].

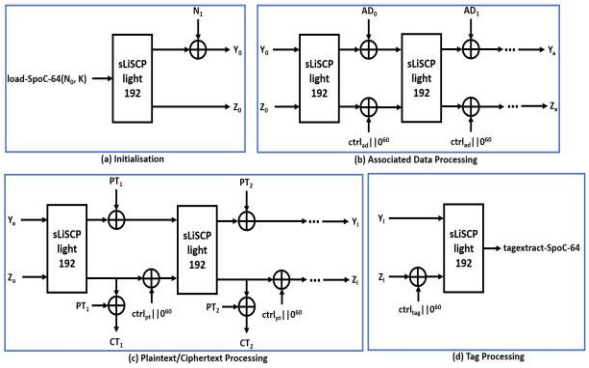


**Fig. 3.** Sponge Construction of SpoC AEAD.

## III. RELATED WORK

The first public cryptographic competition was the call for the Advanced Encryption Standard (AES) in the year 1997 [21]. Subsequently, new cryptographic competitions have an interest in third-party hardware related results as early as possible. Third-party hardware results for the submissions are difficult to come by because the number of submissions increases with each new competition while the evaluation periods are getting shorter. The NIST lightweight AEAD competition was announced in August 2018 with the deadline for submission set in February 2019. This means that cryptographers were given exactly six months to design algorithms with software implementation. The first-round candidates which totaled 51 in number were announced in April 2019 which means that the evaluators at NIST used only two months for their examination of the submissions. The 32 second-round candidates were announced in August 2019 which means that third-party implementers were given just four months to implement the 32 candidates using hardware tools. The third-round candidates are said to be announced in September 2020. Out of the 32 second-round candidates, less than half submitted hardware implementation results.

The very first third-party hardware implementations of some of the second-round candidates were presented by Behnaz et al. in November 2019 [22]. The authors implemented six of the candidates which include SpoC, GIFT-COFB, COMET-AES, COMET-CHAM, ASCON, and SCHWAEMM using FPGA devices. The authors provide hardware results such as power, energy-per-bit, frequency, area, throughput, and throughput-to-area. The paper concluded that SpoC used the smallest area while ASCON achieved the highest throughput-to-area.

This paper presents the hardware implementation of the SpoC AEAD algorithm. The advantages of the hardware implementation presented in this work as opposed to that in [22] include:

- This work gives detailed hardware architectures for each module used in the SpoC AEAD algorithm while that of [22] provides only the top module architecture.
- This work implements both the encryption and decryption routines of SpoC AEAD using one hardware architecture while [22] implements only the encryption routine.
- This work gives verification results in the form of simulation waveforms while [22] do not provide simulation results.
- Overall, this work achieved better results in terms of hardware area and throughput when compared to [22].

## IV. HARDWARE IMPLEMENTATION OF SPOC AEAD ALGORITHM

Fig. 4 shows the overall hardware architecture (datapath and control unit) for the SpoC AEAD algorithm. The architecture is capable of executing both the encryption and decryption routines. The CONTROL UNIT is responsible for generating select signals for the multiplexor in the DATAPATH. The DATAPATH consists of four main registers with include 64-bit PREG, 64-bit PCOUT, 64-bit TOUT, and 192-bit SPREG. The PREG is used during the decryption routine to store the generated plaintext. The PCOUT stores the generated plaintext/ciphertext while the TOUT stores the generated tag. The SPREG serves as an input to the sLiSCP-light-192 permutation module (SLISCP_PERM). The VERIFY module is used by only the decryption routine for verification of the tags. This section discusses each module in the overall architecture.

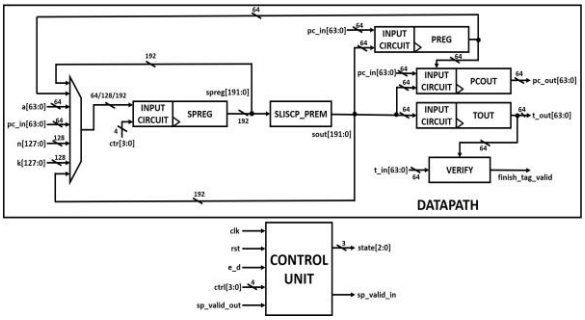


**Fig. 4.** Hardware Architecture for SpoC AEAD.

### *A. Registers*

The proposed SpoC AEAD hardware architecture uses four registers to store data. They include PREG, PCOUT, TOUT, and SPREG. Each register has an input circuit that computes the value to be stored

using XOR gates and concatenation operations.

The PREG is a 64-bit register that is only used during the decryption routine to store the generated plaintext. The input circuit computes the plaintext using two 32-bit XOR gates and concatenates the outputs of the XOR operation to get a 64-bit value as shown in Fig. 5. The inputs to the XOR gates consist of the ciphertext (pc_in[63:0]) and the output from the permutation module ({sout[191:160], sout[95:64]}). The PREG value is given to the PCOUT register after the decryption routine is complete.

The PCOUT is a 64-bit register that stores the generated plaintext or ciphertext depending on whether the encryption or decryption routine is executed. The inputs to the register use two 32-bit XOR gates, two multiplexors, and 64-bit concatenation to compute the value stored in the register as shown in Fig. 6. The inputs to the XOR gates consist of the plaintext/ciphertext input (pc_in[63:0]) and the output from the permutation module ({sout[191:160], sout[95:64]}). The outputs from the XOR gates are concatenated to get a 64-bit value. The multiplexor 1 selects between the 64-bit PREG output (preg[63:0]) and 64'd0 which indicates a null value when the decryption verification fails. The multiplexor 2 selects between the concatenated value and the output of multiplexor 1 when executing encryption and decryption respectively.

The TOUT register store the generated 64-bit during both encryption and decryption routines using only the concatenation operator as shown in Fig. 7. The input to the register concatenates two 32-bit part selects from the output of the permutation module (sout[47:16] and sout[143:112]).

The SPREG is a 192-bit register that serves as an input to the sLiSCP-light-192 permutation module. This has the most complicated input circuitry among all the registers as shown in Fig. 8. The circuit uses a total of six 32-bit input XORs, three multiplexors, and four concatenation operators. The inputs to the first concatenation are the key (k) and the nonce (n), this is selected during the initialization process. The second concatenation inputs include n and sout which is selected during the nonce processing. The third concatenation is selected during the tag processing while the last concatenation is selected during the plaintext/ciphertext processing.

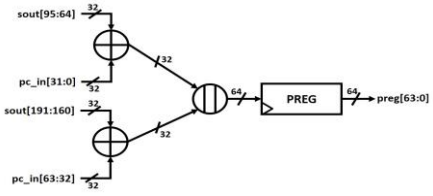


**Fig. 5.** PREG Register Input Circuit.

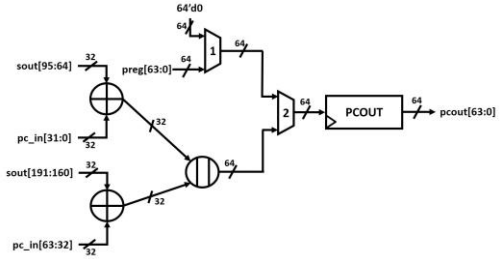


**Fig. 6.** PCOUT Register Input Circuit.

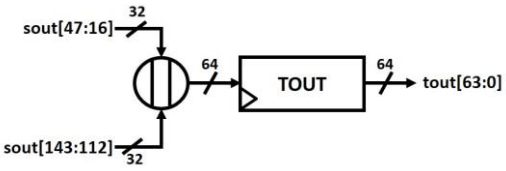


**Fig. 7.** TOUT Register Input Circuit.

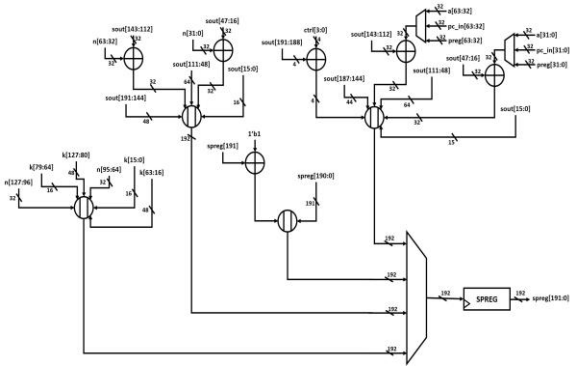


**Fig. 8.** SPREG Register Input Circuit.

### *B. Hardware Architecture for sLiSCP-light-192*

The proposed hardware architecture for the sLiSCP-light-192 permutation (illustrated in Fig. 2) is shown in Fig. 9. The architecture implements memory element RC_SC_RAM which stores the value of the round constants (rc0 and rc1) and step constants (sc0 and sc1). The constants are generated based on the rounds (rnd). During the SSb stage, the con[15:0] represents the concatenation of rc0 and rc1 while it represents the concatenation of sc0 and sc1 during the ASb stage. The RC_SC_RAM is a simple memory that stores a total of 512 bits of data.

The architecture input is the 192-bit register SPREG and produces 192-bit value stored in DREG register. It consists of 48-bit registers S0, S1, S2, S3, and 192-bit register DREG. SB1 and SB3 implement the Simeck box which will be discussed later in this section. The inputs to S0 and S2 registers are the outputs of SB1 (sout1) and SB3 (sout3) respectively. The input of register S1 is computed by the XORing of sout3, concatenation ({40'hFFFFFFFFFF, con[7:0]}), and either dreg[95:0] or spreg[95:48]. The input of register S3 is computed by the XORing of sout1, concatenation ({40'hFFFFFFFFFF,

con[15:8]}), and either dreg[191:144] or spreg[191:144]. The input to the DREG register is the concatenation of the four registers S0, S1, S2, and S3. The architecture uses a total of two XOR gates, four multiplexors, and three concatenation operators.

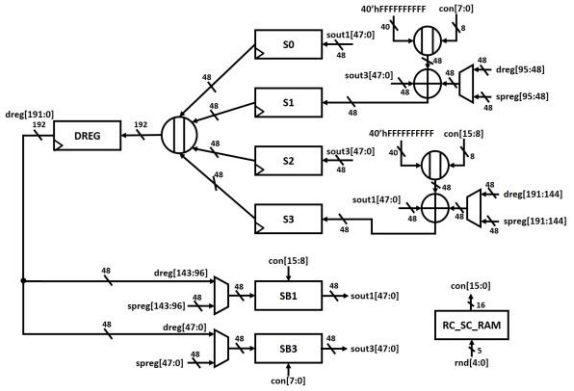


**Fig. 9.** SLiSCP-light-192 Hardware Architecture.

The input to SB1 consist of con[15:0] and either dreg[143:96] or spreg[143:96] while that of SB3 is made up of con[7:0] and either dreg[47:0] or spreg[47:0]. The hardware implementation of SB1 is shown in Fig. 10. The same architecture is also used by S3 by just changing the inputs. From the diagram, a 48-bit input din[47:0] is used to produce a 48-bit output sout1[47:0]. The architecture stores the shifted value of con[7:0] in the CREG register. The Least Significant Bit (LSB) of the CREG register is fed into the SRND module during every round. The SRND module implements the actual Simeck box using four concatenations, one XOR gate, and one AND gate as shown in the figure. The SRND module iterates for six rounds which means that the proposed architecture produces the output using exactly six clock cycles.

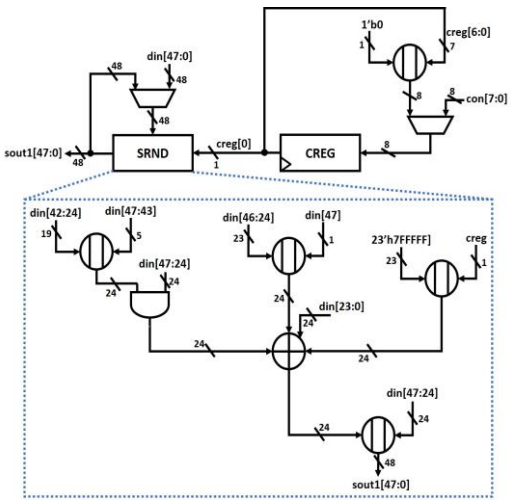


**Fig. 10.** Simeck Box (SB) Hardware Architecture.

### *C. Verification and Control Units*

The Verification Unit (VERIFY) is only activated during the decryption routine. It takes as its inputs the 64-bit tag (t_in) from the sender and the computed tag stored in the TOUT register (t_out) as shown the Fig. 4. The VERIFY module compares the t_in value to the t_out value. If the comparison is true, the computed plaintext is assigned to the output (pc_out). This indicates that the received tag is from a genuine sender. If the comparison is false, a null value is assigned to the pc_out to prevent adversaries from getting access to the plaintext.

The CONTROL unit generates select signals for the multiplexors in the overall hardware architecture. This is done by implementing a Finite State Machine (FSM) with a total of seven states. The transition of the states is dependent on the availability of the plaintext/ciphertext (PT) and the associated data (AD) as shown in Table 1. From the table, a 0 indicates empty PT/CT and AD while a 1 indicates full PT/CT and AD. The seven states include load_init, ad, pt_ct, xor_pt, tag, tag_extract, and finish. The load_init state initializes and loads the inputs during the start of operations. The ad state is responsible for associated data processing while the pt_ct state is for plaintext/ciphertext processing. The xor_pt state is used during the decryption process to compute the plaintext. The tag and tag_extract states are used to generate the verification tag while the finish state verifies the tag and halts the algorithm. The various state transitions for both the encryption and decryption routines are illustrated in Table 1.

**Table 1.** FSM STATE TRANSITIONS

| PT/ CT | AD | Enc State Transition | Enc State Transition |
|---|---|---|---|
| 0 | 0 | load_init → tag → tag_extract → finish | load_init → tag → tag_extract → finish |
| 1 | 0 | load_init → pt_ct → tag → tag_extract → finish | load_init → pt_ct → xor_pt → tag → tag_extract → finish |
| 0 | 1 | load_init → ad → tag → tag_extract → finish | load_init → ad → tag → tag_extract → finish |
| 1 | 1 | load_init → ad → pt_ct → tag → tag_extract → finish | load_init → ad → pt_ct → xor_pt → tag → tag_extract → finish |

## V. HARDWARE RESULTS AND VERIFICATION

The hardware architecture of the SpoC AEAD algorithm discussed in Section 4 was implemented on an FPGA device and developed using Verilog Hardware Description Language (HDL). The synthesis results were generated using Xilinx ISE 14.7 software with a Virtex-4 FPGA device (xc4vlx80-12ff1148). The synthesis results reported the maximum frequency (Freq) and area in terms of Look-up-Tables (LUTs). The higher the frequency of a design, the lower the delay (a combination of gate delay and wire delay). The lower the number of LUTs the better for resource-constrained devices. The latency of the design was calculated in terms of the number of cycles (cyc) using the equation below:

cyc=init_ld+N+(ADc*AD)+(PTCTc*PT_CT)+TG

Where init_ld represents the cycles for initialization and loading the permutation module while TG represents the cycles for tag processing. ADc and PTCTc represent the cycles for the associated data processing and the plaintext/ciphertext processing respectively while AD and PTCT represent the total number of associated data and plaintext/ciphertext blocks respectively. The lower the number of cycles the faster the design. The frequency, area, and cycle metrics are used to compute the throughput (TP) and Throughput-to-Area (TPA) metrics. The TP metric is computed by dividing the block size by the cycles and multiplying by the frequency (Freq x Blk / Cycles) while the TPA metric is computed by dividing the TP value by the area (TP/area). TPA metric is a very important parameter for comparing hardware implementations. This metric shows clearly the attempt of a hardware designer to significantly optimize one characteristic of a design (either area or throughput) at the expense of another.

The synthesis results for the SpoC AEAD algorithm are shown in Table 2. The hardware implementation consumed 951 LUTs at 246.6 MHz maximum clock frequency with 589 cycles for encryption (590 cycles for decryption), TP of 26.8, and TPA of 0.03. The results from this work are compared to that of [22] which implemented SpoC, ASCON, COMET-CHAM, COMET-AES, and GIFT-COFB. When compared to the SpoC implementation in [22], this work used 30% fewer LUTs for both encryption and decryption. The decrease in area in this work is a result of the optimization of the implementation for low-cost devices while [22] implemented the basic iterative architecture without optimizing. This work achieved almost two times the frequency of the SpoC implemented in [22]. The SpoC implementation in [22] used 150 fewer cycles as compared to this work. This is because this work implemented the SpoC using extra registers in other to reduce the critical path which increases the frequency. The increase in frequency and reduced area resulted in this work recording better results in terms of TP (Mbps) and TPA (Mbps/LUT) than the implementation in [22].

**Table 2.** SYNTHESIS RESULTS AND COMPARISONS

| Algorithm | Blk (Bit) | Rnd | Freq (MHz) | Area LUT | Cyc | TP | TPA |
|---|---|---|---|---|---|---|---|
| GIFT-COBE [22] | 128 | 40 | 134.3 | 1960 | 205 | 83.9 | 0.04 |
| COMET-AES [22] | 128 | 10 | 128.6 | 3058 | 73 | 225 | 0.07 |
| COMET-CHAM [22] | 128 | 80 | 145.5 | 2338 | 357 | 52.2 | 0.02 |
| ASCON [22] | 64 | 12 | 174.4 | 1913 | 68 | 164 | 0.08 |
| SpoC [22] | 64 | 108 | 131 | 1364 | 439 | 19.1 | 0.01 |
| SpoC [This Work] | 64 | 108 | 246.6 | 951 | 589 | 26.8 | 0.03 |

The hardware implementation of the SpoC AEAD algorithm was verified through simulation using Xilinx ISE Simulator (ISIM). The test vectors for the simulation were obtained from [23]. For the encryption routine, the key and the nonce use the same 128-bit value (128'h000102030405060708090A0B0C0D0E0F0) while the associated data (ad) and plaintext also use the same 64-bit value (64'h0001020304050607). The generated 64-bit ciphertext (ct) and tag are 64'h01DCF91B896496FC and 64'h0CCCDBEC61B55102 respectively as shown in Fig. 11. The encryption routine uses a total of 589 clock cycles. For the decryption routine, the key, nonce, associated data use the same values as the encryption routine with a 64-bit ciphertext (64'01DCF91B896496FC) and a 64-bit tag (64'h0CCCDBEC61B55102). The generated 64-bit plaintext (pt) and tag are 64'h0001020304050607 and 64'h0CCCDBEC61B55102 respectively as shown in Fig. 12. The encryption routine uses a total of 590 clock cycles.

A popular platform for the execution of lightweight cryptographic algorithms is the

Electronic Product Code (EPC) [24] RFID tags. These tags are part of the target devices of the NIST lightweight AEAD competition. The EPC tags are passively powered and for that matter, many research works [25, 26, 27] consider a clock frequency of 100 kHz as the upper bound for the tags. It is a common notion that the authentication of the EPC tags should take less than 150 msec for the whole process [28]. With the 150 msec authentication time and the 100 kHz clock frequency, the maximum number of cycles required for the authentication of a tag is set at 15,000 clock cycles. From our simulation results, the encryption and decryption routines of the proposed SpoC AEAD hardware architecture used 589 and 590 clock cycles respectively. This indicates that the proposed SpoC hardware architecture meets the latency requirements of the real-world low-cost hardware devices.


**Fig. 11.** Simulation of SpoC AEAD Enc Routine.


**Fig. 12.** Simulation of SpoC AEAD Dec Routine.

## VI. CONCLUSION

The search for lightweight AEAD algorithms by NIST for resource-constrained devices like RFID tags has entered into the second round. The examiners of the second-round candidates are advocating for third party hardware implementations (ASIC or FPGA) of the 32 second-round candidates. This paper, therefore, provides a hardware implementation of the SpoC AEAD algorithm that made it into the second round of the NIST competition. The implemented version is the SpoC-64 which is made up of a 128-bit key, 128-bit nonce, 64-bit plaintext, 64-bit associated data, 64-bit tag, and a 64-bit ciphertext. The SpoC AEAD is sponge-based which uses the Beetle mode of operation with the sLiSCP-light-192 permutation as the primary primitive for the algorithm.

The proposed SpoC AEAD algorithm hardware architecture was designed using Verilog HDL with Xilinx ISE 14.7 used for synthesis (using Virtex-4 FPGA device) and ISIM for simulation. The synthesis and simulation reports provided the maximum frequency, area, cycles, throughput (TP), and throughput-to-area (TPA). The proposed SpoC AEAD hardware architecture consumed 951 LUTs, achieved a maximum frequency of 246.61 MHz, used 589 and 590 clock cycles for encryption and decryption respectively, recorded a TP of 26.8 Mbps, and a TPA ratio of 0.03.

The results from this work were compared to an existing third-party hardware implementation of the SpoC algorithm. This work used 30% fewer LUTs for both encryption and decryption as compared to the existing work. The decrease in area in this work is a result of the optimization of the implementation for low-cost devices while the existing work implemented the basic iterative architecture without optimizing. This work achieved almost two times the frequency of the existing SpoC implementation. The existing SpoC implementation used 150 fewer cycles as compared to this work. This is because this work implemented the SpoC using extra registers in other to reduce the critical path which increases the frequency. The increase in frequency and reduced area resulted in this work recording better results in terms of TP and TPA than the existing implementation.

The latency results (clock cycles) used by the proposed SpoC hardware design were compared to the real-world EPC RFID tags authentication time. These tags are part of the target devices of the NIST lightweight AEAD competition. The maximum number of cycles required for the authentication of a tag is set at 15,000 clock cycles. From our simulation results, the encryption and decryption routines of the proposed SpoC AEAD hardware architecture used 589 and 590 clock cycles respectively. This indicates that the proposed SpoC hardware architecture meets the latency requirements of the real-world low-cost hardware devices.

For future works, the proposed SpoC AEAD will be combined with other security protocols to form a lightweight cryptographic System-on-Chip (SoC).

## REFERENCES


[1] Schwab, K. (2020). *The fourth industrial revolution.* Retrieved from https://www.weforum.org/about/the-fourth-industrial-revolution-by-klaus-schwab

[2] Choi, S., Yang, C., & Kwak, J. (2018). System hardening and security monitoring for IoT devices to mitigate IoT security vulnerabilities and Threats. *KSII Transaction on Internet and Information Systems*, 12(2), 906-918.

[3] National Institute of Standards and Technology. (2020). *CAESAR call for submission.* Retrieved from https://competitions.cr.yp.to/caesar-call.html

[4] National Institute of Standards and Technology. (2020). *CAESAR submission.* Retrieved from https://competitions.cr.yp.to/caesar-submissioans.html

[5] National Institute of Standards and Technology. (2020). *Announcing request for nomination for lightweight cryptographic algorithms.* Retrieved from https://csrc.nist.gov/News/2018/requesting-nominations-for-lightweight-crypto-algs

[6] National Institute of Standards and Technology. (2020). *Round 2 candidates.* Retrieved from https://csrc.nist.gov/Projects/lightweight-cryptography/round-2-candidates

[7] AITawy, R., Gong, G., He, M., Jha, A., Mandal, K., Nandi, M., & Rohit, R. (2020). *SpoC: An authenticated cipher submission to the NIST LWC competition.* Retrieved from https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-2/spec-doc-rnd2/spoc-spec-round2.pdf

[8] Bertoni, G., Daemen, J., Peeters, M., & Assche, G. V. (2020). *Keccak specification.* Retrieved from https://keccak.team/obsolete/Keccak-specifications.pdf

[9] National Institute of Standards and Technology. (2020). *Cryptographic hash algorithm competition.* Retrieved from https://www.nist.gov/programs-projects/cryptographic-hash-algorithm-competition

[10] Bellare, M. & Namprempre, C. (2008). Authenticated encryption: Relations among notions and analysis of generic composition paradigm. *Journal of Cryptology*, 21(4), 469-491.

[11] Whiting, D., Housley, R., & Ferguson, N. (2020). *Counter with CBC-MAC (CCM).* Retrieved from https://tools.ietf.org/rfc/rfc3610.txt

[12] McGrew, D. A. & Viega, J. (2004). The security and performance of the Galois/Counter Mode (GCM) of operation. *Progress in Cryptology – INDOCRYPT 2004*, 343-355.

[13] Rogaway, P., Bellare, M., & Black, J. (2003). OCB: A block-cipher mode of operation for efficient authenticated encryption. *ACM Transaction on Information and System Security*, 6(3), 365-403.

[14] Bertoni, G., Daemen, J., Peeters, M., & Assche, G. V. (2011). Duplexing the sponge: Single-pass authenticated encryption and other applications. *Selected Areas in Cryptography – SAC 2011*, 320-337.

[15] National Institute of Standards and Technology. (2020). *Submission requirement and evaluation criteria for the lightweight cryptography submission process.* Retrieved from https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/final-lwc-submission-requirements-august2018.pdf

[16] Chakraborti, A., Datta, N., Nandi, M., & Yasuda, K. (2018). OCB: Beetle family of lightweight and secure authenticated encryption ciphers. *IACR Transaction on Cryptographic Hardware and Embedded Systems*, 2018(2), 218-241.

[17] AITawy, R., Rohit, R., He, M., Mandal, K., Yang, G., & Gong, G. (2017). sLiSCP: Simec-based permutations for lightweight sponge cryptographic primitives. *Selected Areas in Cryptography – SAC 2017*, 129-150.

[18] AITawy, R., Rohit, R., He, M., Mandal, K., Yang, G., & Gong, G. (2018). sLiSCP-light: Towards hardware optimized sponge-specific cryptographic permutations. *ACM Transaction on Embedded Computing Systems*, 17(4), 1-32.

[19] Bogdanov, A. & Shibutani, K. (2012). Generalized Feistel networks revisited. *Design, Codes, and Cryptography*, 66(3), 75-97.

[20] Yang, G., Zhu, B., Suder, V., Aagaard, M. D. & Gong, G. (2015). The Simeck family of lightweight block ciphers. *Cryptographic Hardware and Embedded Systems – CHES 2015*, 13-16.

[21] National Institute of Standards and Technology. (2020). *Announcing request for candidate algorithm nominations for the advanced encryption standard.* Retrieved from https://csrc.nist.gov/news/1997/requesting-candidate-algorithm-nominations-for-aes

[22] Rezvani, B., Coleman, F., Sachin, S., & Diehl, W. (2019). Hardware implementation of NIST lightweight cryptographic candidates: A first look. *Lightweight Cryptography Workshop 2019*, 1-12.

[23] Renner, S., Pozzobon, E., & Mottok, J. (2020). *spoc64sliscplight192v1.* Retrieved from https://lwc.las3.de/cipher.php?variant=spoc64sliscplight192v1

[24] GSI. (2020). *EPC UHF Gen2 air interface protocol.* Retrieved from https://www.gs1.org/standards/epc-rfid/uhf-air-interface-protocol

[25] Bogdanov, A. & Shibutani, K. (2019). Generalized Feistel networks revisited. *International Association of Cryptologic Research, ePrint Archive*, 2009, 1-197.

[26] Feldhofer, M., Dominikus, S., & Wolkerstorfer, J. (2004). Strong authentication for RFID systems using the AES algorithm. *Cryptographic Hardware and Embedded Systems*, 357-370.

[27] Peris-Lopez, P., Hernandez-Castro, J. C., Estevez-Tapiador, J. M., & Ribagorda, A. (2007). LAMED – A PRNG for EPC Class-1 Generation-2 RFID specification. *Computer Standards and Interfaces*, 31(2009), 88-97.

[28] Armknecht, F., Hamann, M., & Mikhalev, V. (2014). Lightweight authentication protocols on ultra-constrained RFIDs – Myths and facts. *Radio Frequency Identification: Security and Privacy Issues – RFIDSec 2014*, 1-18.