Hardware Implementations of NIST Lightweight Cryptographic Candidates: A First Look

Behnaz Rezvani and William Diehl

Virginia Tech, Blacksburg, VA 24061, USA
email: {behnaz, wdiehl}@vt.edu

Abstract. The U.S. National Institute of Standards and Technology (NIST) has embarked on a multi-year effort called the lightweight cryptography (LWC) standardization process to evaluate lightweight (LW) authenticated encryption with associated data (AEAD) and optional hash algorithms for inclusion in U.S. federal standards. As candidates are evaluated for many characteristics including hardware resources and performance, obtaining results of hardware implementations as early as possible is preferable. In this research, we implement three NIST LWC Round 2 candidate ciphers, SpoC, Spook, and GIFT-COFB, in the Artix-7 FPGA. Implementations are compliant with the previously-validated CAESAR hardware applications programming interface (CAESAR HW API) for authenticated ciphers and are tested in actual hardware. Results indicate that SpoC is the smallest in terms of area, requiring 1344 look-up tables (LUTs), while GIFT-COFB has the highest throughput-to-area (TPA) ratio at 0.154 Mbps/LUT. The results also illustrate the hardware implementation challenges of integrating multiple cryptographic primitives into one design, as well as complexities due to padding and truncation.

Keywords: Lightweight cryptography · FPGA · Authenticated cipher · Encryption

1 Introduction

Devices in the Internet of things (IoT) are vulnerable to theft of privacy information, and are subject to potentially more destructive attacks such as replay or man-in-the-middle attacks. To guard against the range of such attacks, cryptographic solutions should ensure *confidentiality*, *integrity*, and *authenticity*. Authenticated encryption with associated data (AEAD) can ensure all of the above services in a single algorithm, while realizing savings in cost and performance, and by avoiding security pitfalls of interactions with separately-designed ciphers and hashes.

In August 2018, the U.S. National Institute of Standards and Technology (NIST) issued a call for specifications for lightweight AEAD and optional hashes, to be subjected to several rounds of evaluation as part of the lightweight cryptography (LWC) standardization process, and eventually incorporated into U.S.

federal information processing standards (FIPS) [17]. Submissions of specifications were permitted until February 2019, and 56 qualified Round 1 candidates were publicized in April 2019. In August 2019, 32 candidates were selected for Round 2, which is expected to last one year.

NIST LWC candidates are evaluated on several criteria, including cost (e.g., area) and performance (e.g., throughput (TP)) in resource-constrained environments representative of emerging IoT devices. All submissions were required to include software reference implementations. While several submissions included synthesis or implementation results from the authors' own hardware submissions in ASIC or FPGA, the NIST LWC evaluation process specifically assigns higher weight to 3rd-party implementations.

In this research, we provide an early evaluation of selected NIST LWC AEAD candidate submissions through hardware implementations; optional hash algorithms are not considered in this research. Given the large number of qualified submissions and the short period of time between initial publication of specifications and Round 2 selection, we selected three ciphers for evaluation: SpoC, Spook, and GIFT-COFB. These ciphers are representative of a large number of NIST LWC Round 1 submissions, since: at least 47 submissions are composed of block- or block-like primitives, out of which at least 20 are sponge-based; at least 22 ciphers (both block and stream) use 4-bit S-boxes, and at least 9 ciphers use a logical AND or multiplication for non-linear transformations; SpoC and Spook are sponge-based, while Spook and GIFT-COFB use 4-bit S-boxes, and SpoC uses a logical AND for non-linear transformations [18]. Further, none of the chosen ciphers included author FPGA implementations in their initial submissions.

We implemented candidate authors' primary recommended versions of selected ciphers using register transfer level (RTL) methodology in Verilog or VHDL and basic-iterative (i.e., round-based) architecture. Since each round of the ciphers is executed in one clock cycle, this implementation offers a reasonable latency and throughput with a medium area. Although latency and energy consumption are also important considerations in LWC, [2], there are certain trade-offs between them. As discussed in [2, 4], basic-iterative implementation is an efficient architecture and a good candidate for applications with low energy consumption.

All implementations are fully compliant with the existing CAESAR hardware applications programming interface (CAESAR HW API) [15], and use the CAESAR hardware developer's package (CAESAR HW DP) at [13]. We used CAESAR HW API and DP since a corresponding NIST LWC API and DP were not available at time of implementation. Artix-7 FPGA implementations were compared according to maximum frequency, area (look-up tables (LUTs)), TP (Mbps), and throughput-to-area (TPA) ratios.

Our contributions in this work are as follows: 1) We offer fully-functional 3rd-party hardware implementations of selected NIST LWC AEAD candidates, still early in Round 2; 2) We provide a comparison with selected past and present authenticated cipher implementations, in order to bridge the space between

CAESAR and NIST LWC processes, as well as API-compliant and API-non-compliant implementations; and 3) We show examples of features of cipher implementations which may be resource-intensive and reduce performance, in order to illustrate design pitfalls and help guide later-round tweaks.

2 Background

2.1 Authenticated encryption with associated data

In both CAESAR and the NIST LWC standardization process, two operations are defined on AEAD, authenticated encryption and authenticated decryption. In encryption, inputs consist of a public message number *Npub* usually defined as a "number used once" (nonce), a secret key K, plaintext PT, and associated data AD. The outputs of authenticated encryption include *Npub*, AD, ciphertext CT, and *Tag*, which provides for integrity and authenticity of all transmitted data. In authenticated decryption, the inputs are *Npub*, AD, K, CT, and *Tag*. CT is internally decrypted into PT, however, an internal Tag' is computed and checked against *Tag* prior to releasing PT, in a step called "tag verification."

2.2 Hardware API and developer's package

Since a hardware API has not yet been approved for the NIST LWC standardization process, we implemented ciphers using the CAESAR HW API. As NIST LWC AEAD software function prototypes are identical to those used in CAESAR, we expect similar compatibility with hardware implementations. Of note, an API definition of hash was not included in the CAESAR HW API, but is expected to be included in a NIST LWC HW API.

To facilitate the hardware designer's task of meeting CAESAR HW API requirements, a hardware developer's package is provided at [13]. The package includes an input processor (Pre-Processor) and output processor (Post-Processor), which are encapsulated in a top-level module called AEAD. A designer can place a custom design in a subordinate module called CipherCore, and use standardized interfaces to communicate with Pre- and Post-Processors. A set of Python scripts called `aeadtvgen` is used to generate representative test vectors directly from the software reference implementation, and an accompanying HDL test bench (AEAD_TB) automatically verifies test vectors against expected results. An implementer's guide to assist in using the CAESAR HW DP is also available at [14]. In this research, we use the CAESAR HW DP (v2.0), and develop RTL implementations inside CipherCore. The definitions of AEAD and subordinate modules, as well as internal signals are defined in [14].

3 Ciphers implemented in this research

3.1 SpoC

SpoC, described in [1], refers to "sponge with a masked capacity." In SpoC, capacity is masked with data blocks instead of rate which improves the security

and allows larger rate value per permutation call. We implement one of the authors' primary recommendations, SpoC-64, with capacity $c = 128$ bits, state size $b = 192$ bits, nonce size $n = 128$ bits, and tag size $t = 64$ bits. In a sponge-based cipher, the rate refers to the number of keystream bits generated per permutation call, and the capacity $c = b - r$.

This cipher is based around the sLiSCP-light[192] permutation, which uses a combination of a Type II Generalized Feistel Structure (GFS) and Simeck box (SB), and consists of 18 steps of 6 rounds each. Each step consists of three transformations, namely, SubstituteSubblocks (SSb), AddStepconstants (ASc), and MixSubblocks (MSb). The non-linear operations are applied in the SSb or SB. SBs consist of XORs, bitwise rotations, and a 48-bit logical AND.

The duplex sponge construction of SpoC is shown in Fig. 1. At each point in time, the state can be divided into a c-bit Y and r-bit Z, and represented as $Y \parallel Z$. The initial state $Y_0 \parallel Z_0$ is formed by interleaving Nonce and Key ($f(N_0, K)$), and performing a permutation. The tag is generated using an interleaved set of bytes extracted from across the entire 192-bit state. Control bits *ctrl* are 4-bit constants used for domain separation (i.e., to distinguish between authenticated encryption or decryption phases), such as *AD*, *PT*, and *Tag*, and to differentiate between full and partial blocks.

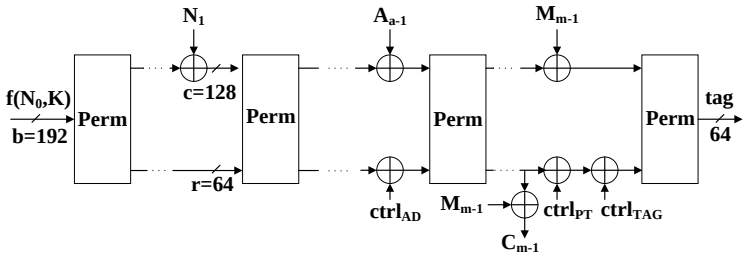

Fig. 1. SpoC duplex sponge construction.

We implement a basic-iterative architecture based on the sLiSCP permutation, where one round of the SSb transformation executes in a single clock cycle. This requires 108 clock cycles for the permutation. Our implementation of SpoC-64 is shown in Fig. 2.

3.2 Spook

Spook uses sponge-one-pass (S1P), and is based on duplex sponge construction [5]. The Spook AEAD algorithm uses two primitives, the Clyde-128 tweakable block cipher (TBC) and the Shadow-512 permutation. The Shadow-512 permutation uses the same definitions for L-boxes and S-boxes, and similar definition for round constants, as the TBC. However, it is performed across a b-bit state (e.g., $b = 512$ in our implementation), and employs encryption only, i.e., there are no inverse L-boxes or S-boxes. Additionally, there is a diffusion function, adapted from the Midori cipher, which acts across all b-bits of state, and is implemented in 32-bit words according to [5].

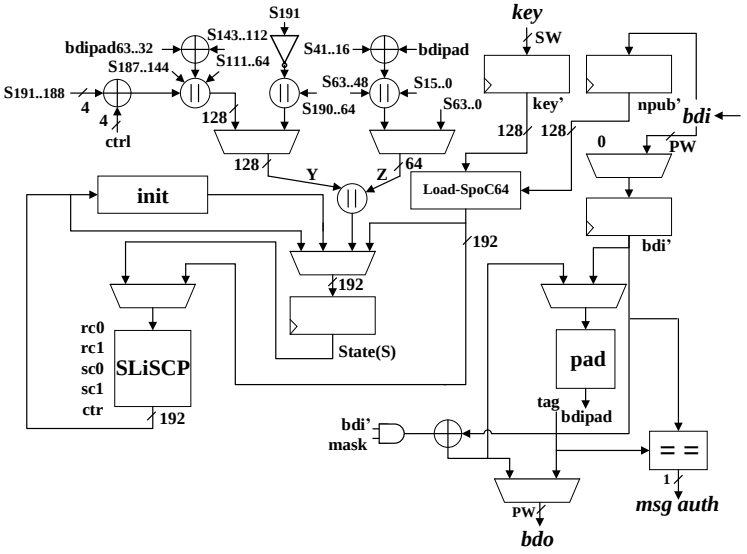

Fig. 2. Block diagram for SpoC-64.

We implement the authors' primary recommendation, Spook[128; 512; su], with parameters block size $n = 128$, rate $r = 256$, capacity $c = 256$, tag size $\tau = 128$, and state size $b = 512$. The "su" denotes "single user." We also used the recommended parameters for the TBC and Shadow-512, which consist of 6 steps of 2 rounds each. Since the rate determines the amount of plaintext converted to ciphertext in every block, we use an external block size of 256 bits for generation of test vectors and computation of throughput. There are m blocks of plaintext (or message) M, and a blocks of associated data A. There is a τ-bit nonce N (128 bits), a $key = K \parallel P$, which consists long term secret K (128 bits), and public tweak P. Following conventions in the software reference implementation, $P = 0$ in this hardware implementation. Of note, the authors' primary recommendation achieves Ciphertext Integrity and Misuse with Leakage in encryption and decryption (CIML2), which is an extension of ciphertext integrity in the presence of nonce misuse and side-channel leakages [6].

In TLS cipher constructions, linear transformations are computed using linear L-boxes consisting of rotations and XORs, and non-linear S-boxes. The L-box is an interleaved transformation applying jointly to pairs of 32-bit words, i.e., half of the 128-bit state is processed in each L-box. L-boxes can be implemented using look-up tables or by arithmetic calculations. We follow the authors' formula for L-box calculations described in [5]. The S-boxes are a variant of the 4-bit S-box used in the Skinny block cipher [4], and are implemented with look-up tables in this research.

The duplex sponge-based computational flow for authenticated encryption is shown in Fig. 3. The TBC is used twice – during initialization and tag generation. Initialization consists of loading the upper 256 bits of the state variable with $P \parallel 0^* \parallel N \parallel 0^*$, and computing $B = E_K^P(N)$, where N is a 128-bit nonce. Upon

TBC completion, the lower 256 bits of the state are loaded with $0 \parallel B$. *Tag* is formed as $Z = E_K^{V||1}(U)$, where U is the upper 128 bits of state after the last permutation, and V is the next highest 127 bits of state. During authenticated decryption, the supplied tag Z' must be decrypted as $U' = D_K^{V'||1}(Z')$, and compared to U (i.e., $U == U'$) for tag verification. Inverse L-boxes $Lbox^{-1}$ and S-boxes $Sbox^{-1}$ are required for decryption.

In between the initial and final TBC operations, the Shadow-512 permutation is computed once to initialize the state, and once following the processing of each block of A or M. 10^* padding *pad* is applied to each final partial block of A, but padding is not directly applied to a final partial block of M. In this case, the resulting upper 256 bits of state are loaded as $C_{m-1} \parallel \{State_{512-|C_{m-1}|-1..256}\} \oplus 01 \parallel 0^*$. State truncation is performed by remembering the number of valid bytes loaded in the last block of plaintext and by applying variable masks to C_{m-1} and $\{State_{512-|C_{m-1}|-1..256}\} \oplus 01 \parallel 0^*$.

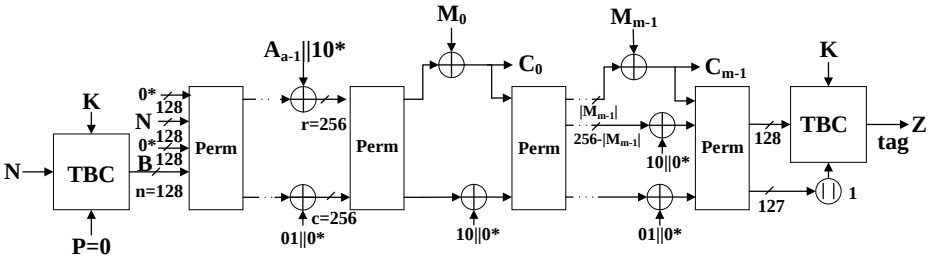

Fig. 3. Spook duplex sponge construction.

Domain separation between blocks of A and M is accomplished by $State_{255..0} \oplus \{01, 10, 11\} \parallel 0^*$, where the two-bit combination depends on whether the block is processing A or M, and whether or not the last block is partial or full.

We base our design strategy on an attempt to reuse components such as L-boxes, S-boxes, and internal state registers, and construct the equivalent of TBC or permutation calls using an arithmetic logic unit (ALU)-like microarchitecture approach. We use a basic-iterative architecture with reference to the TBC. This means that one round of TBC (encryption or decryption) will execute in one clock cycle, including one set of 128-bit L-boxes, 128-bit S-boxes, and round constants. The tweakey is updated at the end of each step, or every other round. This results in 12 clock cycles per TBC.

However, our target implementation is not strictly basic-iterative with respect to the permutation, since we instantiate only 128-bit L-boxes and S-boxes, but must call each module 4 times across a 512-bit state. This results in a total of 144 clock cycles per permutation. Our implementation is shown in Fig. 4.

3.3 GIFT-COFB

GIFT-COFB is based on the combined feedback (COFB) mode of operation with GIFT-128 as the underlying block cipher [3]. COFB mode is single-pass

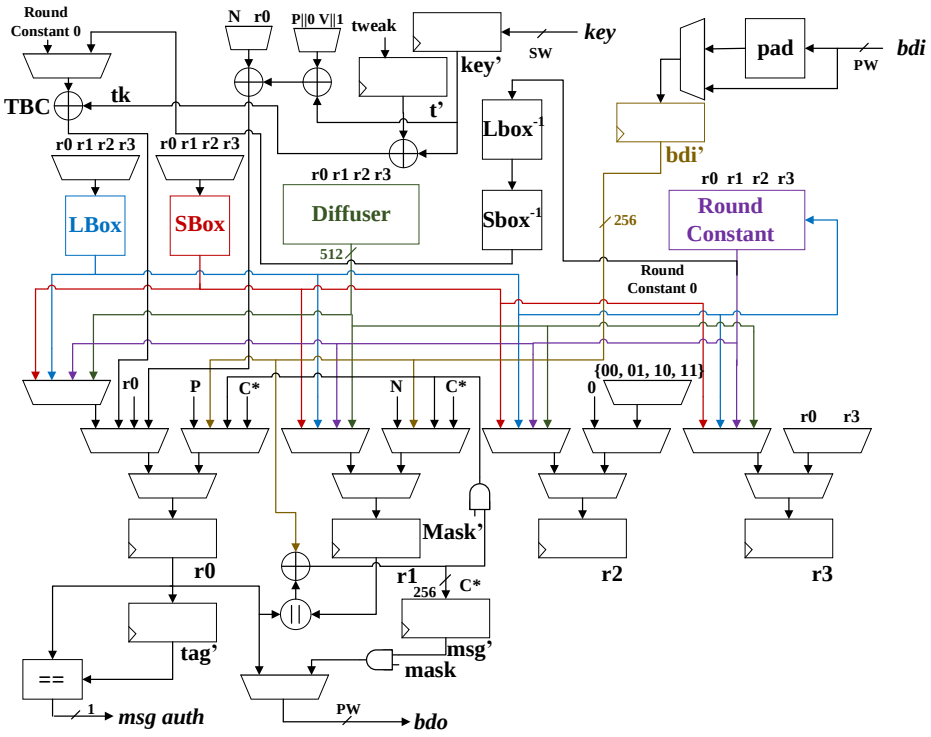

Fig. 4. Block diagram for Spook[128;512;su]. Bus widths are 128 bits unless indicated.

(one block cipher call per data block) and inverse-free (no need for block cipher decryption). The GIFT-COFB recommendations are data block size $n = 128$ bits, nonce size $|N| = 128$ bits, and tag size $|T| = 128$ bits.

GIFT-128 is a substitution-permutation network (SPN) with a 128-bit key length and a 128-bit cipher state length. This iterative block cipher has 40 rounds and each round consists of 3 transformations, namely, SubCells, PermBits, and AddRoundKey. The cipher state divides into four 32-bit words and the key state divides into eight 16-bit segments. In SubCells, 32 4-bit bitslice S-boxes are applied to every nibble of the state. Then, a 32-bit permutation is applied to every word of the state. In AddRoundKey, the round key is XORed to the second and third words of the state, and a round constant is added to the last word of the state. The round constants are generated by a 6-bit LFSR.

In Fig. 5, a simplified version of the encryption construction of GIFT-COFB is depicted. At the beginning of the encryption, the state is loaded by a nonce N and then, the upper 64 bits of the first E_K output L are considered as the delta state. Except for the last block of AD and M, the delta state is multiplied by 2 in GF(2^{64}) for every block of AD and M. For the last block of AD or M, the delta state is multiplied by 3^i or 3^{j-i}, where $i, j - i \leq 4$. The G function is defined as $G(Y) = (Y[2], Y[1] <<< 1)$ [3].

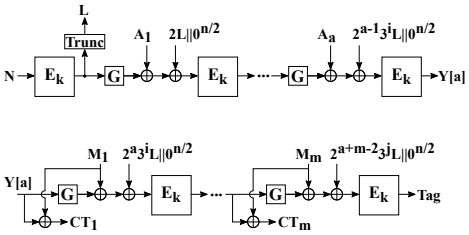

Fig. 5. GIFT-COFB encryption construction.

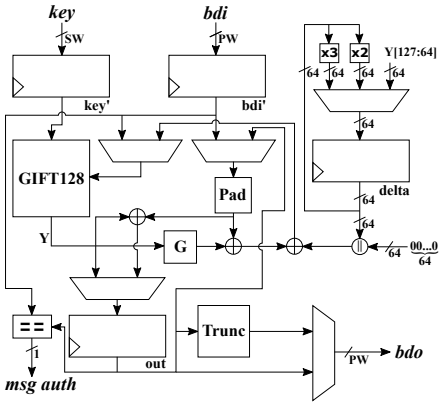

Fig. 6. Block diagram for GIFT-COFB. Bus widths are 128 bits unless indicated.

A basic-iterative (i.e., round-based) architecture is used here, i.e., every round of the GIFT round function is executed in one clock cycle. The GIFT-COFB authors also used a round-based design which is implemented in ASIC. GIFT has 40 rounds, thus it requires 40 clock cycles to process a block of the input data. However, for processing AD and M, we need additional clock cycles due to the delta state. As presented in [3], 4 clock cycles are required for the delta state. In this work, we use 4 clock cycles for processing an AD block, but 2 clock cycles in our finite state machine for processing the message blocks. The reason that we reduce the clock cycles for M is that the exponent in 3^{j-i} does not exceed 2 for an M block. As a result, we have 40, 44, and 42 clock cycles for processing nonce, an AD block, and a block of message, respectively. Note that these are the number of cycles that GIFT needs to process one block of data.

4 Results

4.1 Implementations in this work

FPGA implementations in this research are developed in Verilog or VHDL using RTL design methodology and based on a basic-iterative architecture. They are compliant with the CAESAR HW API and include modules in the CAESAR HW DP (v2.0). Results are implemented in Xilinx Vivado 2018.3 for the Xilinx Artix-7 FPGA (xc7a100tcsg324-3), and optimized for throughput-to-area (TPA) ratio using the Minerva automated hardware optimization tool [11]. Our implementations are also verified in actual hardware (xc7a100tftg256-3) using FOBOS [8]. All implemented ciphers have nonce and key size of 128 bits. Post-optimization results of implementations in this work (TW) are shown in Table 1. Additionally, cipher implementations are available for inspection at [19].

Results indicate that SpoC has the highest maximum frequency of 265 MHz (1.9× greater than Spook), i.e., lowest sum of logic and routing delays, followed

by GIFT-COFB at 172 MHz (1.2× greater than Spook) and Spook at 141 MHz. In terms of area in FPGA LUTs, SpoC is the smallest with 1344 LUTs (19% of area of Spook), followed by GIFT-COFB with 2695 LUTs (39% of area of Spook), and Spook with 7082 LUTs. GIFT-COFB has the highest TP at 415.4 Mbps (2.7× greater than SpoC), followed by Spook at 248.9 Mbps (1.6× greater than SpoC) and SpoC at 152.8 Mbps. In terms of TPA ratio, GIFT-COFB is the highest at 0.154 Mbps/LUT (4.4× greater than Spook), followed by SpoC at 0.114 Mbps/LUT (3.3× greater than Spook), and Spook at 0.035 Mbps/LUT. Additionally, the latency of our GIFT-COFB implementation is only 50% of the latency of our SpoC implementation. This provides GIFT-COFB an advantage for the processing of very short messages, which is a desirable characteristic for LWC candidates as discussed in [17].

4.2 Comparison with selected previous authenticated cipher implementations

Previous hardware implementations during CAESAR and those provided as part of NIST LWC submissions provide some basis for comparison with cipher implementations in this research. However, most CAESAR implementations, even those that were compliant with the CAESAR HW API, used an earlier version of the CAESAR HW DP designed for high speed (HS) implementations. The HS package included functionality not used by many ciphers, and exacted a larger toll on area overhead. The CAESAR HW DP for LW implementations only appeared at the end of 2017, and thus there are fewer available examples. Some CAESAR API-compliant examples from [21, 12, 10], implemented using the LW CAESAR HW DP, are included in Table 1 for purpose of comparison.

A full-scale comparison with NIST LWC candidate author implementations is premature, since authors reported results for implementations not compliant with the CAESAR API, and using a variety of FPGA platforms. Some representative examples of block and sponge cipher FPGA implementations, e.g., ESTATE (ESTATE-TweGIFT-128), SAEAES, and Oribatida (Oribatida-256-64), as well as a non-CIML2 author implementation of Spook, are included in Table 1.

All CAESAR and NIST LWC implementations provided for comparison use a 128-bit key; TP is computed based on the processing rate of a large number of blocks of plaintext into ciphertext. The range of TPA ratios (0.017 to 0.088) for the CAESAR candidates, all Round 3 contenders or better, is somewhat analogous to the range of TPA ratios for our implementations (0.035 to 0.154). In contrast, the range of TPA ratios of sampled NIST LWC candidates (0.547 to 1.028) is noticeably higher. A judgement as to whether or not these implementations are "better" than either our implementations, or previous CAESAR implementations, is premature, since no uniform standards have been established for benchmarking of hardware implementations in the NIST LWC standardization process. For instance, our implementations of the AEAD top-level module, incorporating the required features in the CAESAR HW DP, were an average of 11% larger than basic CipherCore implementations of respective ciphers. Additionally, an implementation "compliant with the CAESAR API" is required to include

hardware necessary for input and output of AEAD data in specified protocol, and must realize "corner cases" (e.g., null blocks, partial blocks, padding, truncating, etc.) which often involve significant resources.

Table 1. Results of implementations in this work (TW), and comparison with CAESAR lightweight and NIST LWC candidates. The units of Freq, Area, TP, and TPA are MHz, LUTs, Mbps, and Mbps/LUT, respectively.

Cipher	Type	FPGA	Freq	Area	TP	TPA	Ref
CAESAR							
Ascon-128	Sponge	Spartan-6	216.0	684	60.1	0.088	[21]
Ascon-small	Sponge	Spartan-6	146.1	1640	114.0	0.070	[10]
CLOC-AES	Block	Spartan-6	101.9	1604	68.7	0.043	[12]
SILC-AES	Block	Spartan-6	115.1	872	15.1	0.017	[12]
NIST LWC (AEAD)							
SpoC	Sponge	Artix-7	265.0	1344	152.8	0.114	TW
Spook	Sponge	Artix-7	141.0	7082	248.9	0.035	TW
Spook	Sponge	Artix-7	181.8	3771	3878.4	1.028	[20]
GIFT-COFB	Block	Artix-7	172.0	2695	415.4	0.154	TW
ESTATE	Block	Virtex-7	580.1	1413	928.3	0.657	[9]
SAEAES	Block	Virtex-7	145.9	348	263.3	0.757	[16]
Oribatida	Sponge	Virtex-7	554.2	940	514	0.547	[7]

4.3 Observations

The use of two primitives in Spook, i.e., the Clyde-128 TBC and the Shadow-512 primitive, increases the area of our Spook FPGA implementation. While we have employed a strategy to reuse shared components among the primitives, such as S-box and L-box, the increased use of control structures necessary to tie together all components likely outweighs any advantages gained over the use of separate TBC and Shadow permutation primitives in a "black box" approach. The fact that the TBC requires encryption and decryption in the CIML2 case adds again to implementation complexity. Results in [20] show that significant improvements in area and performance are possible when a fully reversable TBC is not required.

The necessity of applying one-zero (10^*) padding to input words of AD and PT is a known factor in increasing complexity. However, cipher algorithms which can make padding application features as similar as possible for both AD and PT can reduce hardware complexity. While this is the case in SpoC and GIFT-COFB, it is not the case in Spook, since the padding in Spook occurs on words of state vice words of input when processing PT.

Additionally, truncation is also a well-known feature in cryptographic algorithms, where $|CT|$ should equal $|PT|$ in order to assure invertibility and not leak information. While truncation in software looks innocuous enough, it often catches cryptographers by surprise in hardware. We employed a non-resource intensive method in SpoC, to remove one unwanted byte at a time from ciphertext output in a serial fashion. However, in Spook, truncated ciphertext is required to be passed to the state input for subsequent permutations, in addition to being routed directly to cipher output, which increases implementation complexity.

5 Conclusion

We provided an early look at fully-functional FPGA implementations of selected NIST LWC standardization process Round 2 candidates. Candidates examined in this research, SpoC, Spook, and GIFT-COFB, are representative of many NIST LWC candidates. Implementations are fully-compliant with the existing CAESAR HW API for authenticated ciphers, use the associated CAESAR HW developer's package, are optimized using the Minerva automated hardware optimization tool, and are verified to operate in actual hardware using the FOBOS test bench.

Our results show that SpoC has the highest maximum frequency of 265 MHz, and has the lowest area, in terms of LUTs, with 1344 LUTs. GIFT-COFB has the highest throughput (TP) at 415.4 Mbps, and the highest throughput-to-area (TPA) ratio. However, differences in security assumptions among authors' primary recommendations (e.g., nonce misuse or leakage resistance) can heavily influence results. The TPA ratio results of the implemented ciphers are similar to results reported for CAESAR HW API compliant late-round CAESAR candidates, but have TPA ratios which are significantly less than TPA ratios reported for a selected group of NIST LWC submission author implementations of ciphers of similar construction. However, no conclusion can be drawn regarding the relative hardware merits of candidates implemented according to different compliance standards, which reinvigorates the need for a standardized hardware API and minimum compliance criteria for the NIST LWC standardization process.

Finally, we show that implementation complexities resulting from the need to integrate two cryptographic primitives (e.g., a block cipher and sponge permutation) into one authenticated cipher, as well as padding and truncation strategies, can affect area and performance of resulting implementations, and should be considered by algorithm designers.

6 Acknowledgements

The authors would like to thank the Spook Team for their comments and insights.

References

1. AlTawy, R., Gong, G., He, M., Jha, A., Mandal, K., Nandi, M., Rohit, R.: SpoC: An Authenticated Cipher Submission to the NIST LWC Competition (Feb 2019), https://csrc.nist.gov/Projects/Lightweight-Cryptography/Round-1-Candidates
2. Andres Lara-Nino, C., Daz-Prez, A., Morales-Sandoval, M.: FPGA-Based Assessment of Midori and Gift Lightweight Block Ciphers: 20th International Conference, ICICS 2018, Lille, France, October 29-31, 2018, pp. 745–755 (10 2018)
3. Banik, S., Chakraborti, A., Iwata, T., Minematsu, K., Nandi, M., Peyrin, T., Sasaki, Y., Sim, S.M., Todo, Y.: GIFT-COFB: An Authenticated Encryption and Hash Algorithm Submission to the NIST LWC Competition (Mar 2019), https://csrc.nist.gov/Projects/Lightweight-Cryptography/Round-1-Candidates

4. Beierle, C., Jean, J., Kölbl, S., Leander, G., Moradi, A., Peyrin, T., Sasaki, Y., Sasdrich, P., Sim, S.M.: The skinny family of block ciphers and its low-latency variant mantis. In: CRYPTO 2016. pp. 123–153
5. Bellizia, D., Berti, F., Bronchain, O., Cassiers, G., Duval, S., Guo, C., Leander, G., Leurent, G., Levi, I., Momin, C., Pereira, O., Peters, T., Standaert, F.X., Wiemer, F.: Spook: Sponge-Based Leakage-Resilient Authenticated Encryption with a Masked Tweakable Block Cipher (Mar 2019), https://csrc.nist.gov/Projects/Lightweight-Cryptography/Round-1-Candidates
6. Berti, F., Pereira, O., Standaert, F.X.: Reducing the cost of authenticity with leakages: a ciml2-secure ae scheme with one call to a strongly protected tweakable block cipher. In: AFRICACRYPT 2019. pp. 229–249
7. Bhattacharjee, A., List, E., Lpez, C.M., Nandi, M.: The Oribatida Family of Lightweight Authenticated Encryption Schemes (Mar 2019), https://csrc.nist.gov/Projects/Lightweight-Cryptography/Round-1-Candidates
8. CERG: Flexible Open-source workBench fOr Side-channel analysis (FOBOS) (Oct 2016), https://cryptography.gmu.edu/fobos/
9. Chakraborti, A., Datta, N., Jha, A., Lopez, C.M., Nandi, M., Sasaki, Y.: ESTATE (Mar 2019), https://csrc.nist.gov/Projects/Lightweight-Cryptography/Round-1-Candidates
10. Diehl, W., Farahmand, F., Abdulgadir, A., Kaps, J.P., Gaj, K.: Face-off between the caesar lightweight finalists: Acorn vs. ascon. 2018 International Conference on Field-Programmable Technology (Dec 2018)
11. Farahmand, F., Ferozpuri, A., Diehl, W., Gaj, K.: Minerva: Automated Hardware Optimization Tool. In: 2017 International Conference on ReConFigurable Computing and FPGAs (ReConFig)
12. Farahmand, F., Diehl, W., Abdulgadir, A., Kaps, J.P., Gaj, K.: Improved lightweight implementations of caesar authenticated ciphers. pp. 29–36 (04 2018)
13. George Mason University: Development Package for the CAESAR Hardware API, v2.0 (Dec 2017), https://cryptography.gmu.edu/athena/index.php?id=CAESAR
14. Homsirikamol, E., Diehl, W., Ferozpuri, A., Farahmand, F., Gaj, K.: Implementer's Guide to the CAESAR Hardware API v2.0 (Dec 2017)
15. Homsirikamol, E., Diehl, W., Ferozpuri, A., Farahmand, F., Yalla, P., Kaps, J.P., Gaj, K.: CAESAR Hardware API. Cryptology ePrint Archive, Report 2016/626
16. Naito, Y., Matsui, M., Sakai, Y., Suzuki, D., Sakiyama, K., Sugawara, T.: SAEAES (Feb 2019), https://csrc.nist.gov/Projects/Lightweight-Cryptography/Round-1-Candidates
17. National Institute of Standards and Technology: Submission Requirements and Evaluation Criteria for the Lightweight Cryptography Standardization Process (Aug 2018), https://csrc.nist.gov/projects/lightweightcryptography
18. Rezvani, B., Diehl, W.: Detailed Characteristics of NIST Lightweight Cryptography Project Round 1 Submissions (v2) (Jul 2019), https://rijndael.ece.vt.edu/wdiehl/
19. SAL: NIST Lightweight Cryptography Project (Jul 2019), https://rijndael.ece.vt.edu/wdiehl/
20. Spook Team: Spook (unprotected) implementation of encryption. email of Jul. 30, 2019
21. Yalla, P., Kaps, J.P.: Evaluation of the CAESAR Hardware API for Lightweight Implementations. In: International Conference on Reconfigurable Hardware (ReConFig 2017). pp. 1–6 (Dec 2017)