Side Channel Resistance at a Cost: A Comparison of ARX-based Authenticated Encryption

Flora Coleman, Behnaz Rezvani, Sachin Sachin, William Diehl
Department of Electrical and Computer Engineering, Virginia Tech, Blacksburg, VA
{flora-coleman, behnaz, sachin255701, wdiehl}@vt.edu

***Abstract*—Lightweight cryptography offers viable security solutions for resource constrained Internet of Things (IoT) devices. However, IoT devices have implementation vulnerabilities such as side channel attacks (SCA), where observation of physical phenomena associated with device operations can reveal sensitive internal contents. The U.S. National Institute of Standards and Technology has called for lightweight cryptographic solutions to process authenticated encryption with associated data (AEAD), and is evaluating candidates for suitability in a Lightweight Cryptography (LWC) Standardization Process. Two Round 2 candidate variants, COMET-CHAM and SCHWAEMM, use Addition-Rotation-XOR (ARX) primitives. However, ARX ciphers are known to be costly to protect against certain SCA. In this work we implement side channel protected versions of COMET-CHAM and SCHWAEMM using register transfer level design. Identical protection schemes consisting of a threshold implementation (TI)-protected Kogge-Stone adder are adopted. Resistance to power side channel analysis is verified on an Artix-7 FPGA target device. Implementations comply with the Hardware API for Lightweight Cryptography, and use a custom-designed extension of the Development Package for the Hardware API for Lightweight Cryptography which enables test and evaluation of side channel resistant designs. We compare side channel protection costs of the two candidates against each other, against their unprotected counterparts, and against previous side channel protected AEAD implementations. COMET-CHAM is shown to consume less area and power, while SCHWAEMM has higher throughput and throughput to area ratio, and is more energy efficient. On average, the costs of protecting these ciphers against SCA are 32% more in area and 38% more in power, compared to the average protection costs for a large selection of previously-evaluated ciphers of similar implementation. Our results highlight the costs involved in implementing side channel protected ARX-ciphers, and help to inform NIST LWC late round and final portfolio selections.**

***Index Terms*—lightweight cryptography, NIST, authenticated encryption, side-channel, FPGA, ARX**

I. Introduction

With the continued growth of the Internet of Things (IoT), more attention is being directed towards ensuring the security of IoT devices. One way is through the use of lightweight cryptography, which has the potential to offer security at a lower cost than current cryptographic standards. This is valuable for the IoT because many devices do not have the necessary resources to support current standards. To engender suitable lightweight cryptography solutions, the National Institute of Standards and Technology (NIST) is holding the Lightweight Cryptography (LWC) Standardization Process from 2018 to 2021 [1]. NIST is evaluating algorithms for authenticated encryption with associated data (AEAD), which provide message confidentiality, integrity and authenticity in one algorithm [2]. To evaluate submissions, NIST suggested looking at a number of metrics, including performance, consumption of resources, and resistance to side channel attacks, including in reconfigurable hardware platforms such as FPGAs.

Cryptographic algorithms can be resistant to mathematical attacks yet still vulnerable to side channel attacks (SCA) due to the physical characteristics of implementations. SCA analyze these characteristics to expose sensitive information. For example, in Differential Power Analysis (DPA) [3], statistical analysis of power consumption data is used to reveal sensitive information about the target algorithm. One countermeasure used to guard against power analysis SCA is the threshold implementation (TI) method [4]. An advantage of TI is that it leverages the benefits of secret sharing [5] while also maintaining security in CMOS device implementations which are subject to data-dependent glitches. Although TI provides important protection against SCA, there are costs incurred with applying the method, including increased consumption of resources (e.g., FPGA look-up tables (LUTs)) and power consumption.

In this paper, we examine two AEAD cipher candidates which have advanced to Round 2 of the NIST LWC Standardization Project: COMET-CHAM [6] and SCHWAEMM [7]. COMET and SCHWAEMM are the only two remaining NIST-candidate families which use Addition-Rotation-XOR (ARX) primitives as their source of non-linearity. This provides an interesting basis for comparison, especially when considering how these algorithms can be protected against SCA. To better understand the cost associated with protecting these algorithms, we create several register transfer level (RTL) implementations of each candidate. We first build unprotected implementations (UnPr) of each candidate using basic-iterative, i.e. round-based, architecture, where modulo addition used in the ARX primitive is implemented with the FPGA fabric adder. As an incremental step towards the protected versions, we provide implementations which integrate a registered Kogge-Stone adder (KSA) in lieu of the fabric adder. These implementations were then modified to be protected (Pr) against SCA by using a 1st-order DPA-resistant, 3-share, TI KSA [8] scheme. Boolean masking is used to separate input

This work is partially funded by the U.S. Department of Commerce (NIST) Award 70NANB18H219 for Lightweight Cryptography in Hardware and Embedded Systems and the National Science Foundation under Grant Number 1303297.

data, such as public or secret key data, into three secret shares. A direct comparison of performance and costs in nearly cycle-equivalent unprotected and protected architectures is achieved using the incremental UnPr KSA implementations as a basis for comparison. We compare these ciphers against one another and we also compare results to previous work. This helps address a question not satisfactorily covered in the previous NIST SHA-3 competition, namely, assessment of the relative side channel protection costs of ARX ciphers [9].

To facilitate fair comparison, all implementations follow the guidelines defined by the Hardware Application Programming Interface (API) for Lightweight Cryptography (LWC API) [10] and use utility modules provided in the Development Package (DP HWAPI LWC) [11]. To support the protected implementations of COMET-CHAM and SCHWAEMM, we implement the LWC API standards for SCA-protected implementations by a custom extension of the DP HWAPI LWC. The updated DP HWAPI LWC supports the generation of multi-shared test vectors, input/output, and processing of shared implementations.

The main contributions of this paper include:

1) The first side channel resistant FPGA implementations of NIST LWC Round 2 ARX-based candidates COMET-CHAM and SCHWAEMM.
2) An extension to the Development Package for the Hardware API for Lightweight Cryptography to facilitate testing and verification of multi-share SCA-resistant implementations for all NIST LWC candidates.
3) A *direct* comparison of these two NIST LWC candidates in terms of resource and performance costs; and an *indirect* comparison between ARX and non-ARX based authenticated ciphers.

II. Background & Previous Work

There have been several FPGA implementations of NIST LWC Round 2 Candidates. RTL implementations of SpoC, GIFT-COFB, COMET-AES and Ascon are provided in [12]. These implementations were evaluated across several FPGAs including the Artix-7, Spartan-6 and Cyclone-V. In [13], authors evaluated a LOTUS implementation on the Virtex-6 FPGA, and a hardware evaluation of ESTATE was presented in [14]. Additionally, an ASIC-based evaluation of LWC Candidates ACE and WAGE was conducted in [15]. These hardware implementations provide valuable points of comparison for LWC candidates, but do not investigate costs associated with side channel protection. The NIST report on Round 1 results [16] does not focus heavily on hardware implementations and SCA resistance, suggesting that this will play a larger role in later round selections. Additionally, many previous HW implementations do not employ a standard protocol such as the LWC API, which makes direct comparisons difficult.

Threshold implementations (TI) provide provably secure protection against DPA, including protections against glitch propagation pertinent to CMOS device implementations [4]. When a glitch occurs in hardware, the power consumption associated with the glitch is comparatively large. Data-dependent differential power consumption can even be used to deduce internal values, as in [17]. In order to provide security, TI relies upon the three properties of non-completeness, correctness, and uniformity [4]. Non-completeness ensures that at least one share is omitted from each function calculation. Correctness is achieved when the summation of the output shares provides an accurate result. Uniformity guarantees that the input probability distribution of a function matches its output probability distribution. Finally, it is important to note that the minimum number of shares needed to uphold these properties for a given function is one more than the degree of the function, e.g., three shares are required to protect a second-degree primitive [4].

Costs of protecting cryptographic algorithms against SCA have been investigated in the past. Several have used TI as their method for providing protection. In [18], the authors produced a protected version of the NIST LWC Round 2 Candidate Ascon [19] using TI. They similarly compared the trade offs of protected versus unprotected implementations. A threshold implementation of the block cipher SPARX was presented in [20]. The permutation SPARKLE, which is used in SCHWAEMM, is based off of a variation of SPARX [7]. A large group of AEAD algorithms was evaluated in [21], providing metrics on the cost associated with TI-based SCA protection for multiple ciphers. Although there have been several works examining threshold implementations, none of these previous works compare the costs of multiple ARX ciphers, or compare a subgroup of ARX ciphers against a larger group of non-ARX ciphers.

ARX-based ciphers are known to offer fast performance in software when compared to Substitution-Permutation Network (SPN)-based ciphers [22]. Several works have investigated software implementations of ARX-based ciphers, including a performance evaluation of both SPARX and CHAM in [23]. However, good performance of ARX-ciphers in hardware is not certain [9], [24]. In this research, we choose to evaluate RTL implementations of COMET-CHAM and SCHWAEMM on an Artix-7 FPGA to gain a better understanding of the costs associated with ARX-ciphers on hardware, and the impact of using side channel resistant methodologies.

Previous SCA protection for ARX ciphers has been primarily investigated in software due to the fast speed of ARX structures in software [25], [26]. In general, there are two techniques for applying a masking scheme against SCA on ARX ciphers: 1) Using Boolean and arithmetic masking together [25], [26], and 2) using Boolean logic for arithmetic operations [27]. The fundamental challenge in the first method is the need for a secure conversion from the Boolean to arithmetic masking and vice versa. Furthermore, since these conversion methods are primarily designed and optimized for software platforms, they may not perform as efficiently in hardware [27], [28]. This problem is alleviated in the second technique, where we directly apply a masking method on the Boolean shares. Inspired by this, we focus on the protected hardware implementations of the ARX-based ciphers using purely Boolean masking in this paper.

III. Methodology

A. Register Transfer Level Implementations

The COMET family of submissions to the NIST LWC Standardization Process employs the COMET mode of operation [6] and a variety of possible block ciphers. The COMET mode of operation combines aspects of the Beetle [29] mode of operation and the CTR [30] mode of operation. COMET requires a 128-bit key and a 128-bit nonce to process data blocks and tag of size 128 bits. COMET has two ARX-based members: COMET-CHAM and COMET-SPECK. The SPECK block cipher is designed for optimal performance in software, while SIMON is a hardware-optimized block cipher [31]. However, CHAM is a new lightweight block cipher that has comparable results to SPECK in software, but smaller area than SIMON in hardware [32]. Therefore, we focus on the COMET-CHAM submission. The COMET state is made up of the concatenation of two parts: 128-bit Y-state and 128-bit Z-state. The contents of Y-state and Z-state are considered as the input state and key state of CHAM, respectively. The COMET implementation is shown in Figure 1. CHAM-128/128 uses 4 branches to process the 128-bit state, with each branch consisting of one 32-bit word of the state. CHAM consists of 80 rounds, and each ARX-round uses one of the 8 unique round keys generated using the 128-bit input key. The key schedule of CHAM is based on the stateless-on-the-fly, thus the key state is not updated during the 80 rounds. 80 total clock cycles per permutation call (one per round) are required using basic-iterative architecture.

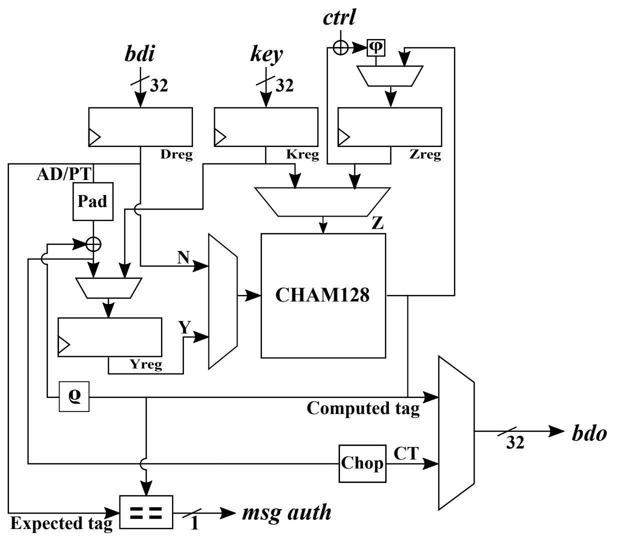

Fig. 1. COMET-CHAM Implementation compliant with the LWC API. All bus widths are 128 bits unless otherwise indicated.

SCHWAEMM256-128 is an authenticated cipher which employs the SPARKLE384 permutation [7]. The cipher requires a 256-bit nonce and a 128-bit key to process 256-bit data blocks. The ciphertext produced is the same size as the input plaintext, and the output tag has a size of 128 bits. SCHWAEMM uses a mode of operation which is a modification of Beetle, detailed in [29]. Several features of SCHWAEMM include a constant injected for domain separation, the combined feedback, and the rate whitening which modifies the state before the start of the SPARKLE permutation. Six instances of the ARX box *Alzette* are used (6 total branches) in SPARKLE384, each taking as input two of the 12 consecutive words of the state [7]. Each step of the permutation completes the 4 ARX rounds of *Alzette* before the application of the linear layer. Depending on the current operation, either 7 or 11 steps will be completed before the permutation terminates. This implementation allocates one clock cycle per ARX round, taking either 30 or 46 clock cycles to complete the permutation, including two additional clock cycles to store the input and output states. The implementation is depicted in Figure 2.

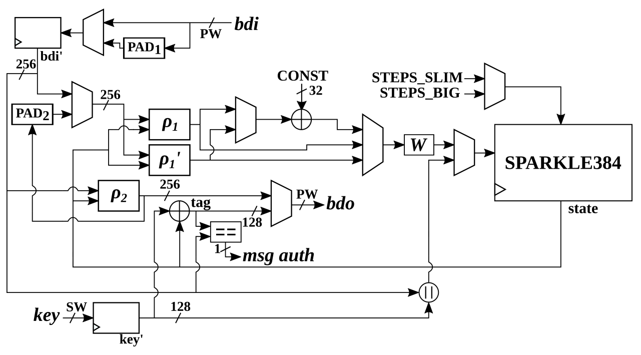

Fig. 2. SCHWAEMM Implementation compliant with the LWC API. All bus widths are 384 bits unless otherwise indicated.

B. SCA Protection Scheme

To make these RTL implementations side channel resistant, we first integrate an unprotected 32-bit Kogge-Stone adder (KSA) [8] into the ARX primitives. We chose the KSA since it supports the overlay of a Boolean masking scheme, and is previously demonstrated with good performance in hardware and software [27], [33]. Specifically, a registered version of a 32-bit KSA was used to ensure that the same number of clock cycles would be used for the ARX primitive in the UnPr (KSA) implementation and the Pr implementation. Next, the 3-TI KSA scheme explored in [27] was applied. Figure 3 depicts the 32-bit registered KSA, and shows its six levels of registers.

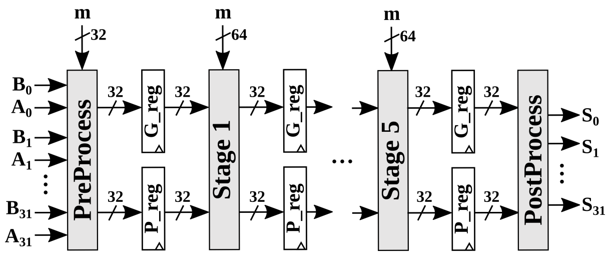

Fig. 3. 32-bit Kogge-Stone Adder with six layers of registers. Bus widths are 1 bit unless otherwise indicated. “m” denotes input for refreshing randomness.

The 3-TI KSA scheme requires replacing all AND gates within the Kogge-Stone adder with 3-share TI AND gates, and

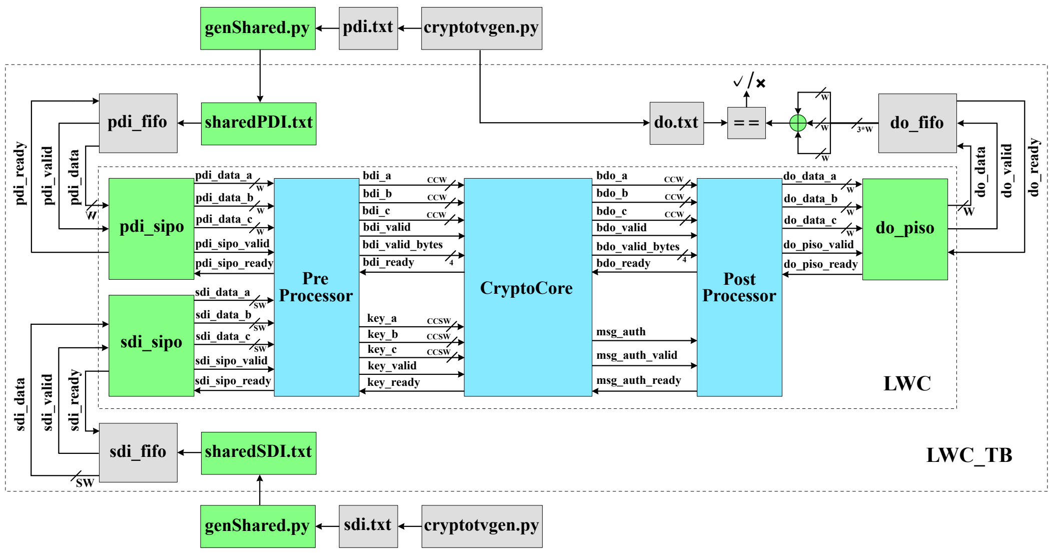

Fig. 4. Overview of the Extension to the Development Package for the Hardware API for Lightweight Cryptography (DP HWAPI LWC). Blue indicates modified modules, green indicates new additions, and gray indicates unchanged components.

registering the output at each stage of the adder. We introduce refreshing randomness to meet the uniformity property of TI implementations. This randomness is generated by an external pseudo random number generator (PRNG) based on the Trivium stream cipher [4]. The remainder of our implementations are protected by separating data into three Boolean masked shares. To prevent possible data leakage, our modified version of the DP HWAPI LWC ensures that these shares are not combined together anywhere within the LWC module (see Figure 4).

C. Extension of DP HWAPI LWC

The Development Package for the LWC API [11], fielded in 2019, supports easy implementations of AEAD and hash designs compliant with the LWC API [10]. Although sharing protocols for side channel resistant implementations were defined in the LWC API, they were not implemented in the development package. To support SCA-resistant implementations of COMET-CHAM and SCHWAEMM, we extend the DP HWAPI LWC to introduce an SCA-protected DP HWAPI LWC, as depicted in Figure 4. This modification will additionally facilitate LWC API-compliant SCA-resistant implementations of any NIST LWC candidate.

The baseline DP HWAPI LWC includes a PreProcessor, CryptoCore and PostProcessor. The PreProcessor and PostProcessor handle the processing of test vector inputs to and outputs from CryptoCore. Designers place their implementations within the CryptoCore module. The DP HWAPI LWC additionally includes a test bench (LWC_TB) and cryptotvgen, a Python module which generates test bench compatible test vectors. To implement the SCA-protected DP HWAPI LWC, we modify the PreProcessor and PostProcessor to simultaneously act on 3 shares using the previously defined control signals, and expand the input and output data signals `bdi` and `bdo` to and from the CryptoCore. We also include serial-in parallel-out (SIPO) modules to capture share-separated data arriving at the LWC, and a parallel-in serial-out (PISO) module to reserialize processed data departing from the LWC. Further, we provide additional test vector generation utilities, coded in Python, which reformat test vectors generated by cryptotvgen for use in the protected DP HWAPI LWC. The reformatted test vectors share-separate public and key data using initial randomness provided in Python.

Figure 4 shows our additions and modifications to the DP HWAPI LWC. Modified portions are color-coded blue; new additions are green, and unchanged portions are grey.

IV. Results

To confirm that the protected implementations of COMET-CHAM and SCHWAEMM were resistant to side channel analysis, t-tests were completed using the Flexible Open-source workBench fOr Side-channel analysis (FOBOS) [34]. Our instance of FOBOS uses the Digilent Nexys A7 as a control board, the NewAE CW305 Artix-7 FPGA (xc7a100tftg256-3) target board to instantiate designs under test (DUT), and collects power traces using the Picoscope 5000 Oscilloscope. Leakage analysis is performed using the Test Vector Leakage Assessment (TVLA) method detailed in [35]. Figures 5 and 6 show the results of t-tests on the UnPr and Pr versions of COMET-CHAM and SCHWAEMM respectively. With 2000 power traces collected at 1 MHz DUT frequency, the tests show leakage across the time domain (x-axis), indicated by t-values that exceed a threshold of $|4.5|$ (shown with blue lines). T-values for the protected t-tests are contained within the threshold, indicating that the protected implementations have less propensity to data-dependent leakage which could be exploited through SCA attacks.

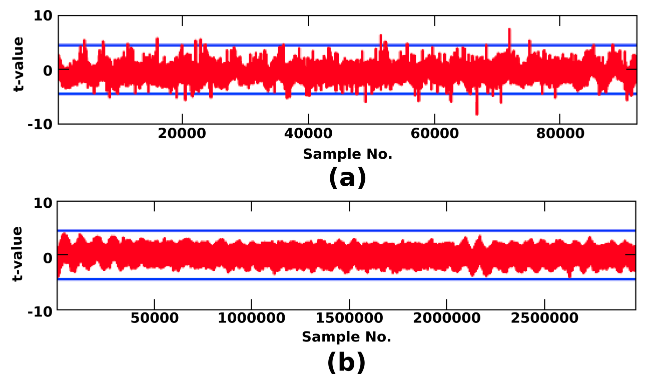

Fig. 5. COMET-CHAM t-test results run at 1MHz. (a) COMET-CHAM UnPr. (b) COMET-CHAM Pr.

Our implementations are optimized for best throughput-to-area (TPA) ratio using the Minerva automated tool for hardware optimization [36]. Results from this work (TW) are shown in Table I. Please note that the area of the PRNG is not considered in these results. Results of other selected instances of corresponding unprotected and protected cipher hardware implementations are included for comparison.

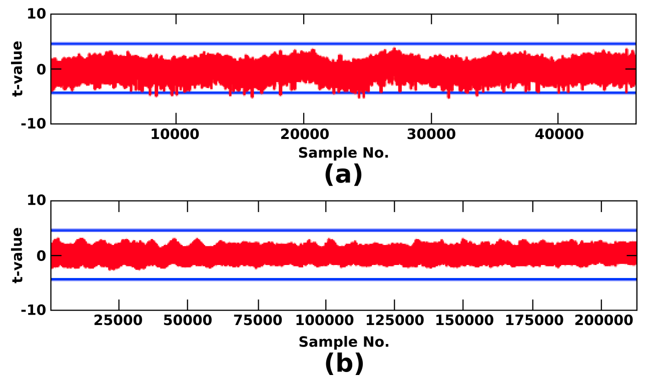

Fig. 6. SCHWAEMM t-test results run at 1MHz. (a) SCHWAEMM UnPr. (b) SCHWAEMM Pr.

A. Direct Comparison

Of the two ARX-based candidates, SCHWAEMM Pr consumed the most area at 12,531 LUTs, which is 4.4× larger than SCHWAEMM UnPr (KSA) and 5.4× larger than SCHWAEMM UnPr. By contrast, the area of COMET-CHAM Pr is 8,760 LUTs, which is 3.7× that of COMET-CHAM UnPr (KSA), and 4.0× larger than COMET-CHAM UnPr. The increased area of the protected 3-share TI implementation, resistant to 1st-order DPA, results from an approximate tripling of linear transformations (e.g., rotations) and associated datapath (e.g., registers, multiplexers, and XOR operations), and a quadratic increase in non-linear components (e.g., AND operations) associated with the Kogge-Stone Adder performing the 32-bit modulo addition. An additional increase of area results from our extension to the DP HWAPI LWC, as it includes additional buffers to split data into shares on input to the PreProcessor and recombine shares upon output from the PostProcessor.

However, SCHWAEMM offers higher throughput than COMET-CHAM. While the combinational logic critical paths of COMET-CHAM and SCHWAEMM are similar (e.g. the maximum frequency is 205 MHz for COMET-CHAM and 207 MHz for SCHWAEMM Pr), SCHWAEMM processes 256-bit message blocks whereas COMET-CHAM processes 128-bit blocks. Further, assuming basic-iterative architectures, CHAM UnPr requires 80 clock cycles to process one block of state while SCHWAEMM UnPr needs only 44 clock cycles. In the KSA-based implementations UnPr (KSA) and Pr, 6 clock cycles per addition are required versus only 1 cycle per FPGA fabric addition in UnPr. Therefore, the number of required clock cycles becomes 480 for CHAM UnPr (KSA) and Pr, and 264 for SCHWAEMM UnPr (KSA) and Pr. This leads to a SCHWAEMM Pr throughput (TP) of 231.4 Mbps, which is 11% less than SCHWAEMM UnPr (KSA), and a COMET-CHAM TP of 44.2 Mbps, which is 22% less than COMET-CHAM UnPr (KSA). The TP reductions from UnPr (KSA) to Pr result from additional routing delay due to area growth in the FPGA implementation; there is negligible increase in the critical path.

In terms of throughput-to-area ratio (TPA), SCHWAEMM Pr is the greatest of the protected implementations at 0.019 Mbps/LUT, which is a 4.8× reduction from SCHWAEMM UnPr (KSA), while the TPA ratio of COMET-CHAM is 0.005 Mbps/LUT, which is likewise a 4.8× reduction from COMET-CHAM UnPr (KSA).

We also consider differences in power consumption and energy efficiency (energy-per-bit or E/bit) of COMET-CHAM and SCHWAEMM. Power is measured on FOBOS at the Artix-7 V_{cc} input across a 1 Ohm resistor for test vectors consisting of two authenticated encryption and decryption operations. Table II shows mean power and E/bit results at 40 MHz, and power gradient (mW/MHz) identified by the linear interpolation of measurements at multiple frequencies (10, 25, and 40 MHz). E/bit is computed as $MeanPwr(mW)_{40MHz}/TP(Mbps)_{40MHz}$. SCHWAEMM Pr is the most energy efficient at 1.21 nJ/bit, while COMET-CHAM consumes 5.44 nJ/bit at 40 MHz. As a large percentage of Artix-7 power is static, the power gradient (increase of mW per increase of 1 MHz) improves the metric for comparison of dynamic power consumption. COMET-CHAM Pr has the lower gradient 0.4823 mW/MHz, which is 4.8× greater than COMET-CHAM UnPr (KSA), while SCHWAEMM Pr has a gradient of 0.7293 mW/MHz, which is 4.6× greater than SCHWAEMM UnPr (KSA). We note that the gradients for COMET-CHAM UnPr (KSA) and SCHWAEMM UnPr (KSA) are 30% and 46% less than their respective UnPr gradients; this shows that 32-bit modulo additions performed by the fabric adder in a single clock cycle pull more current than the equivalent modulo addition divided over 6 clock cycles using the Kogge-Stone adder.

TABLE I
IMPLEMENTATION RESULTS

Cipher	Impl	FPGA	Freq (MHz)	Area (LUT/GE)	Area (Ratio)	TP (Mbps)	TP (Ratio)	TPA (Mbps/Area)	TPA (Ratio)	Ref
COMET-CHAM	UnPr	Artix-7	201.0	2214	0.92	282.73	4.99	0.128	5.41	TW
	UnPr (KSA)		255.0	2399	1.00	56.67	1.00	0.024	1.00	
	Pr		205.0	8760	3.65	44.18	0.78	0.005	0.21	
SCHWAEMM	UnPr	Artix-7	169.0	2321	0.82	920.51	3.56	0.397	4.33	TW
	UnPr (KSA)		189.0	2824	1.00	258.74	1.00	0.092	1.00	
	Pr		207.0	12531	4.44	231.41	0.89	0.019	0.20	
Ascon	UnPr	-	-	7950	1.00	5524	1.00	0.694	1.00	[18]
	Pr		-	30420	3.82	3774	0.68	0.124	0.18	
Ascon	UnPr	Spartan-6	195.5	2048	1.00	255.4	1.00	0.125	1.00	[21]
	Pr		103.1	6364	3.11	134.6	0.53	0.021	0.17	
Acorn	UnPr	Spartan-6	226.6	549	1.00	906.2	1.00	1.651	1.00	[21]
	Pr		142.7	2732	4.98	570.6	0.63	0.209	0.13	
JAMBU-AES	UnPr	Spartan-6	163.1	1073	1.00	50.9	1.00	0.048	1.00	[21]
	Pr		122.4	2869	2.67	38.2	0.75	0.013	0.28	
AES-GCM	UnPr	Spartan-6	176.0	1947	1.00	103.4	1.00	0.053	1.00	[21]
	Pr		116.8	4828	2.48	68.57	0.66	0.014	0.27	

TABLE II
POWER AND ENERGY MEASURED ON ARTIX-7 FPGA

Cipher	Impl	Mean Pwr (mW) @40 MHz	E/bit (nJ/Bit) @40 MHz	Gradient (mW/ MHz)	Gradient Ratio
COMET-CHAM	UnPr	31.8	0.56	0.1417	1.42
	UnPr (KSA)	30.6	3.47	0.0997	1.00
	Pr	46.9	5.44	0.4823	4.84
SCHWAEMM	UnPr	39.2	0.18	0.2910	1.85
	UnPr (KSA)	32.6	0.59	0.1573	1.00
	Pr	54.0	1.21	0.7293	4.64

B. Indirect Comparison

An exhaustive comparison with all previously published results is beyond the scope of this research. Selected results from previous works (e.g. [18], [21]), which performed comparisons of DPA-protected and unprotected cipher implementations, are included in Table I. An interesting comparison, however, is our subgroup of NIST LWC ARX-based cipher implementations versus the group of 11 ciphers (10 CAESAR candidates plus AES-GCM) reported in [21]. Though none of the external selection of ciphers included in Table I are ARX-based, the protected implementations of the ciphers use the TI method to provide 1st-order resistance to DPA and the FPGA implementations use the CAESAR Hardware API [37], which is similar to the LWC API. This provides a close basis for comparison. Implementations in [21] had an average area 3.1$\times$ larger than their unprotected counterparts, while COMET-CHAM Pr and SCHWAEMM Pr are an average of 4.1$\times$ larger than their respective UnPr (KSA) implementations. Although the referenced study did not provide power gradients, power consumption for Spartan-6 FPGA protected implementations at 10 MHz increased by an average factor of 3.4$\times$ over unprotected implementations, while power gradients of protected ARX ciphers in this research increased by 4.7$\times$ compared to their UnPr (KSA) counterparts. Finally, TPA ratios of protected ciphers in [21] decreased by 5.6$\times$ compared to unprotected implementations, while the ARX subgroup decreased by only 4.8$\times$. Thus we find that ARX hardware implementations have a higher area and power side channel protection cost than the "average" authenticated cipher, while there is no statistical reduction in TPA ratios.

V. CONCLUSIONS

In this research we performed a detailed comparison of side channel protected FPGA implementations of the two authenticated encryption candidate families in the NIST Lightweight Cryptography (LWC) Standardization Process which use Addition-Rotation-XOR (ARX) primitives: COMET-CHAM and SCHWAEMM. To implement and functionally verify implementations, and confirm side channel resistance of protected implementations, we extended an existing hardware development package to enable multi-share implementations compatible with the LWC Application Programming Interface. In a comparison of implementations with basic-iterative architecture and 1st-order DPA resistance, the protected COMET-CHAM is shown to use fewer FPGA resources, and consume less power than SCHWAEMM, while protected SCHWAEMM has a higher throughput-to-area (TPA) ratio, and is more energy efficient. Additionally, we considered the costs of side channel protection of this subgroup of ARX-based authenticated ciphers compared to a larger group of similarly-implemented ciphers from a previous study. We found that this subgroup of ARX ciphers is more costly to protect against SCA in terms of area and power consumption than the average cipher in the larger study, however, the protected ARX authenticated ciphers are generally not worse in terms of throughput or TPA ratio than the average authenticated cipher. Future work could include affirmation of these findings through a larger-scale study of protected authenticated ciphers enabled by our extension to the hardware development package.

REFERENCES

[1] National Institute for Standards and Technology, "Submission Requirements and Evaluation Criteria for the Lightweight Cryptography

Standardization Process," Tech. Rep., Aug. 2018. [Online]. Available: https://csrc.nist.gov/projects/lightweight-cryptography

[2] P. Rogaway, "Authenticated-Encryption with Associated-Data," in *Proceedings of the 9th ACM Conference on Computer and Communications Security*, ser. CCS '02. New York, NY, USA: Association for Computing Machinery, 2002, p. 98–107.

[3] P. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," in *Advances in Cryptology — CRYPTO' 99*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 388–397.

[4] S. Nikova, C. Rechberger, and V. Rijmen, "Threshold Implementations Against Side-Channel Attacks and Glitches," in *Information and Communications Security*, P. Ning, S. Qing, and N. Li, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 529–545.

[5] L. T. Brandão, N. Mouha, and A. Vassilev, "Threshold schemes for cryptographic primitives: challenges and opportunities in standardization and validation of threshold cryptography," National Institute of Standards and Technology, Gaithersburg, MD, Tech. Rep. NIST IR 8214, Mar. 2019. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/ir/2019/NIST.IR.8214.pdf

[6] S. Gueron, A. Jha, and M. Nandi. (2019, Nov.) COMET: COunter Mode Encryption with authentication Tag. [Online]. Available: https://csrc.nist.gov/Projects/lightweight-cryptography/round-2-candidates

[7] C. Beierle, A. Biryukov, L. Cardoso dos Santos, J. Großschädl, L. Perrin, A. Udovenko, V. Velichkov, and Q. Wang. (2019, Sep.) Schwaemm and Esch: Lightweight Authenticated Encryption and Hashing using the Sparkle Permutation Family. [Online]. Available: https://csrc.nist.gov/Projects/lightweight-cryptography/round-2-candidates

[8] P. M. Kogge and H. S. Stone, "A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations," *IEEE Transactions on Computers*, vol. C-22, no. 8, pp. 786–793, Aug. 1973.

[9] S. Chang, R. Perlner, W. E. Burr, M. S. Turan, J. M. Kelsey, S. Paul, and L. E. Bassham, "Third-Round Report of the SHA-3 Cryptographic Hash Algorithm Competition," 2012.

[10] J.-P. Kaps, W. Diehl, M. Tempelmeier, E. Homsirikamol, and K. Gaj. (2019, Oct.) Hardware API for Lightweight Cryptography. [Online]. Available: https://cryptography.gmu.edu/athena/index.php?id=LWC

[11] M. Tempelmeier, F. Farahmand, E. Homsirikamol, W. Diehl, J.-P. Kaps, and K. Gaj, "Implementer's Guide to Hardware Implementations Compliant with the Hardware API for Lightweight Cryptography," Nov. 2019. [Online]. Available: https://cryptography.gmu.edu/athena/index.php?id=LWC

[12] B. Rezvani, F. Coleman, S. Sachin, and W. Diehl, "Hardware Implementations of NIST Lightweight Cryptographic Candidates: A First Look," Cryptology ePrint Archive, Report 2019/824, pp. 1–26, Feb. 2020.

[13] A. Chakraborti, N. Datta, A. Jha, C. M. Lopez, M. Nandi, and Y. Sasaki, "LOTUS and LOCUS AEAD: Hardware Benchmarking and Security," Nov. 2019. [Online]. Available: https://csrc.nist.gov/Events/2019/lightweight-cryptography-workshop-2019

[14] ——, "ESTATE: Hardware Benchmarking and Security Analysis," Nov. 2019. [Online]. Available: https://csrc.nist.gov/Events/2019/lightweight-cryptography-workshop-2019

[15] M. D. Aagaard, M. Sattarov, and N. Zidaric, "Hardware Design and Analysis of the ACE and WAGE Ciphers," *arXiv:1909.12338 [cs]*, Jan. 2020, arXiv: 1909.12338. [Online]. Available: http://arxiv.org/abs/1909.12338

[16] M. S. Turan, K. A. McKay, C. Çalık, D. Chang, and L. Bassham, "Status report on the first round of the NIST lightweight cryptography standardization process," National Institute of Standards and Technology, Gaithersburg, MD, Tech. Rep. NIST IR 8268, Oct. 2019. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/ir/2019/NIST.IR.8268.pdf

[17] S. Mangard, N. Pramstaller, and E. Oswald, "Successfully Attacking Masked AES Hardware Implementations," in *Cryptographic Hardware and Embedded Systems – CHES 2005*, J. R. Rao and B. Sunar, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 157–171.

[18] H. Gross, E. Wenger, C. Dobraunig, and C. Ehrenhöfer, "Ascon hardware implementations and side-channel evaluation," *Microprocessors and Microsystems*, vol. 52, pp. 470–479, Jul. 2017. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0141933116302721

[19] C. Dobraunig, M. Eichlseder, F. Mendel, and M. Schläffer, "Ascon," V1.2, Nov. 2019. [Online]. Available: https://csrc.nist.gov/Projects/Lightweight-Cryptography/ Round-2-Candidates

[20] S. M. Ramesh and H. AlKhzaimi, "Side Channel Analysis of SPARX-64/128: Cryptanalysis and Countermeasures," in *Progress in Cryptology – AFRICACRYPT 2019*, J. Buchmann, A. Nitaj, and T. Rachidi, Eds. Cham: Springer International Publishing, Jun. 2019, pp. 352–369.

[21] W. Diehl, A. Abdulgadir, F. Farahmand, J.-P. Kaps, and K. Gaj, "Comparison of Cost of Protection Against Differential Power Analysis of Selected Authenticated Ciphers," in *2018 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, Apr. 2018, pp. 147–152.

[22] N. Mouha and B. Preneel, "A Proof that the ARX Cipher Salsa20 is Secure against Differential Cryptanalysis," *IACR Cryptology ePrint Archive*, pp. 1–18, 2013.

[23] B. Seok and C. Lee, "Fast implementations of ARX-based lightweight block ciphers (SPARX, CHAM) on 32-bit processor," *International Journal of Distributed Sensor Networks*, vol. 15, no. 9, p. 10, Sep. 2019. [Online]. Available: http://journals.sagepub.com/doi/10.1177/1550147719874180

[24] N. Mouha, "ARX-based Cryptography," Jun. 2011. [Online]. Available: https://www.cosic.esat.kuleuven.be/ecrypt/courses/albena11/slides/nicky_mouha_arx-slides.pdf

[25] L. Goubin, "A Sound Method for Switching between Boolean and Arithmetic Masking," in *Cryptographic Hardware and Embedded Systems — CHES 2001*, Ç. K. Koç, D. Naccache, and C. Paar, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 3–15.

[26] B. Debraize, "Efficient and provably secure methods for switching from arithmetic to boolean masking," in *Cryptographic Hardware and Embedded Systems – CHES 2012*, E. Prouff and P. Schaumont, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 107–121.

[27] T. Schneider, A. Moradi, and T. Güneysu, "Arithmetic Addition over Boolean Masking," in *Applied Cryptography and Network Security*, T. Malkin, V. Kolesnikov, A. B. Lewko, and M. Polychronakis, Eds. Cham: Springer International Publishing, 2015, pp. 559–578.

[28] J. D. Golic, "Techniques for random masking in hardware," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 54, no. 2, pp. 291–300, 2007.

[29] A. Chakraborti, N. Datta, M. Nandi, and K. Yasuda, "Beetle Family of Lightweight and Secure Authenticated Encryption Ciphers," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2018, no. 2, pp. 1–24, 2018.

[30] M. Dworkin, "Recommendation for Block Cipher Modes of Operation - Methods and Techniques," Dec. 2001. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38a.pdf

[31] R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers, "The SIMON and SPECK Families of Lightweight Block Ciphers," Cryptology ePrint Archive, Report 2013/404, 2013, https://eprint.iacr.org/2013/404.

[32] B. Koo, D. Roh, H. Kim, Y. Jung, D.-G. Lee, and D. Kwon, "CHAM: A Family of Lightweight Block Ciphers for Resource-Constrained Devices," in *Information Security and Cryptology – ICISC 2017*, H. Kim and D.-C. Kim, Eds. Cham: Springer International Publishing, 2018, pp. 3–25.

[33] J.-S. Coron, J. Großschädl, M. Tibouchi, and P. K. Vadnala, "Conversion from Arithmetic to Boolean Masking with Logarithmic Complexity," in *Fast Software Encryption*, G. Leander, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 130–149.

[34] A. Abdulgadir, W. Diehl, and J.-P. Kaps, *Flexible, Opensource workBench fOr Side-channel analysis (FOBOS), User Guide*, v2.0 ed., Cryptographic Engineering Research Group, Dec 2019.

[35] G. Becker, J. Cooper, E. DeMulder, G. Goodwill, J. Jaffe, G. Kenworthy, T. Kouzminov, A. Leiserson, P. Rohatgi, and S. Saab, "Test Vector Leakage Assessment (TVLA) methodology in practice," vol. 1001, Gaithersburg, MD, Nov. 2013, pp. 1–13.

[36] F. Farahmand, A. Ferozpuri, W. Diehl, and K. Gaj, "Minerva," Dec. 2017. [Online]. Available: https://cryptography.gmu.edu/athena/index.php?id=Minerva

[37] E. Homsirikamol, W. Diehl, A. Ferozpuri, F. Farahmand, P. Yalla, J. Kaps, and K. Gaj, "CAESAR Hardware API," Cryptology ePrint Archive, Report 2016/626, 2016, http://eprint.iacr.org/2016/626.pdf.