Side-Channel Resistant Implementations of Three Finalists of the NIST Lightweight Cryptography Standardization Process: Elephant, TinyJAMBU, and Xoodyak

Abubakr Abdulgadir[1,2], Richard Haeussler[1], Sammy Lin[1], Jens-Peter Kaps[1], and Kris Gaj[1]

[1]Cryptographic Engineering Research Group, George Mason University, Fairfax, VA, U.S.A.
[2]PQSecure Technologies, Boca Raton, FL, U.S.A.

***Abstract*—Lightweight authenticated ciphers are crucial in many resource-constrained applications, including online payments and the Internet of Things. The US National Institute of Standards and Technology (NIST) coordinates a standardization process, currently in Round 3, to select lightweight algorithms for such applications. Although security is paramount, cost, performance, and resistance to side-channel attacks are among the most critical selection criteria. This paper investigates the effect of applying side-channel countermeasures on the cost and performance of three NIST Lightweight Cryptography Finalists: Elephant, TinyJAMBU, and Xoodyak. For all investigated algorithms, we apply Domain-Oriented Masking. We then compare the cost of protection in terms of resource utilization and performance. Our first-order protected designs of Elephant, TinyJAMBU, and Xoodyak occupy 5451, 1267, and 6431 LUTs and have a throughput of 93, 120, and 891 Mbps, respectively, when implemented on Xilinx Artix-7 FPGAs.**

***Index Terms*—lightweight cryptography; hardware; FPGA; side-channel attacks**

I. Introduction

Many applications are restricted in terms of area and power. For example, smart cards and many Internet-of-Things (IoT) devices have tight constraints on these metrics. Lightweight Cryptography (LWC) is a field that strives to provide solutions for such use cases.

Authenticated Encryption with Associated Data (AEAD) is a widely used cryptographic primitive. However, current AEAD standards, such as AES-GCM [1], are not suitable for lightweight applications, and the US National Institute of Standards (NIST) is coordinating a standardization process currently in Round 3 to select suitable LWC algorithms.

Side-Channel Analysis (SCA) [2] is a severe threat to secure deployment of cryptography, especially in LWC applications, which usually have little or no physical security. In SCA attacks, side-channel information is used to recover sensitive data. Many countermeasures have been proposed to protect implementations against SCA. For example, Domain Oriented Masking (DOM) [3] provides security in the presence of glitches, hence, suitable for hardware designs.

Although security against cryptanalysis is crucial to select standardization candidates, cost, performance, and side-channel resistance distinguish candidate ciphers that proceed to the final stages of the selection process.

The community has studied SCA protection of many LWC finalists. In [4], side-channel resistance for several NIST LWC candidates was investigated, including Xoodyak and Elephant. However, their work focuses on masking software implementations, while we focus on masking hardware designs. Additionally, their study concerns the primitives rather than the entire authenticated encryption algorithms.

A design space exploration study was performed for SCA-resistant implementations of TinyJAMBU in [5]. However, no direct comparison with other NIST LWC finalists was performed. In [6], side-channel resistant hardware implementations of NIST LWC candidates COMET-CHAM and SCHWAEMM were reported.

This paper describes and compares hardware-protected implementations of three out of ten finalists of the NIST LWC standardization process: Elephant, TinyJAMBU, and Xoodyak. To the best of our knowledge, we present the first protected hardware implementations for Xoodyak and Elephant.

Our main contribution is providing a timely comparison between a significant portion of the remaining candidates in the ongoing NIST LWC standardization process. NIST specifically sets ease of protection against side-channel attacks as a desirable criterion, so our results will be helpful to compare studied candidates.

II. Background

A. Elephant

Elephant is a lightweight permutation-based authenticated encryption scheme that contains three variants, with a different underlying hash function and security level.

Dumbo (Elephant-Spongent-π[160] with 112-bit security) is the primary variant of the submission package, making it the logical choice for hardware implementation. For the full specification of the algorithm, see [7].

Mask values come from an LFSR that is initialized with the output of $P(K||0^{n-k})$, where n is the number of bits in

the state, and k is the number of bits in the key (128). Once initialized, the following equation updates the LFSR:

$$(x_0, ..., x_{19}) \rightarrow (x_1, ..., x_{19}, x_0 \lll 3 \oplus x_3 \ll 7 \oplus x_{13} \gg 7)$$

Dumbo's permutation consists of 80 rounds of the Spongent lightweight hash function. Each round of Spongent consists of 3 layers: a) Xor with the ICounter, b) S-box Layer, and c) pLayer.

B. TinyJAMBU

TinyJAMBU [8] is a lightweight variant of JAMBU, a CAESAR Round-3 candidate. To update its state, TinyJAMBU uses a keyed permutation based on a Nonlinear Feedback Shift Register (NLFSR). To perform the permutation, the NLFSR is updated by calculating the *feedback* bit(s) as a function of the state and the key. Then, the state register is shifted to the right. Depending on how the permutation is implemented, multiple feedback bits can be calculated in parallel.

The TinyJAMBU-128 variant has a 128-bit state and uses a 128-bit key, 96-bit nonce, and 64-bit tag. The data (associated data and plaintext/ciphertext) is processed in 32-bit blocks. The specification defines 3-bit constants called *FrameBits* that are different for the nonce, associated data, plaintext/ciphertext, and finalization.

C. Xoodyak

Xoodyak [9] is a cryptographic primitive designed to perform authenticated encryption with associated data (AEAD), hashing. The permutation used in Xoodyak, called Xoodoo, is a Keccak-*f* inspired permutation that iteratively applies a round function to a 384-bit state. In Xoodyak, the number of rounds is 12.

The 384-bit state can be seen as a 3D array with three 4×32 horizontal planes A0, A1, and A2. The round function consists of the application of a mixing layer θ, plane shifting ρ_{west}, round constant addition ι, nonlinear layer χ, and a plane shifting ρ_{east}. The most critical operation in the Xoodoo permutation from the SCA countermeasure perspective is χ since it is the only operation that is nonlinear. We show the details of this operation below and omit other operations to save space. In the equations below, $\oplus$ is a bitwise XOR, $\overline{A}$ is a bitwise negation of A, and $\cdot$ is bitwise AND.

$$B_0 \leftarrow \overline{A_1} \cdot A_2$$
$$B_1 \leftarrow \overline{A_2} \cdot A_0$$
$$B_2 \leftarrow \overline{A_0} \cdot A_1$$
$$A_y \leftarrow A_y \oplus B_y \quad for\ y \in \{0, 1, 2\}$$

III. Methodology

Our implementations are fully compatible with the GMU LWC Hardware API [10]. This feature enables direct comparison with other LWC Hardware API compliant implementations. For protected implementations, we leverage the LWC API extension developed in [6]. This extension facilitates the development and verification of SCA-protected designs by supporting I/O of data in a shared format. For SCA-protected designs, we used the DOM countermeasure since it provides security in the presence of glitches with a relatively low area overhead and randomness requirements. To provide the random bits required for the DOM countermeasure, we utilize a Pseudo-Random Number Generator (PRNG) based on Trivium [11], which is seeded using data provided through the RDI port.

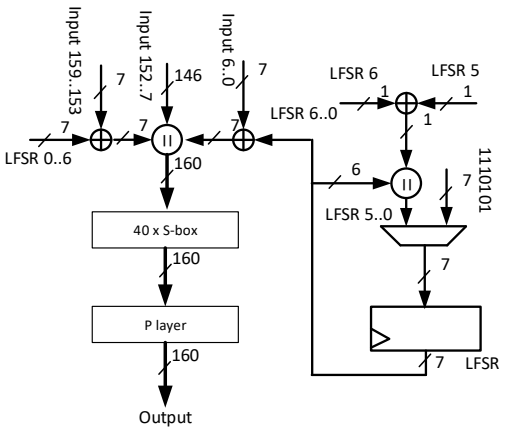

Fig. 1. Elephant: A round of the Spongent permutation

Protected implementations process data in shares that are not mixed unless proper precautions are taken. The first-order SCA resistance was confirmed using the TVLA methodology. All designs were benchmarked in an Artix-7 FPGA in terms of area in LUTs, Throughput in Mbps, power in mW, and energy-per-bit in nJ/bit.

We benchmarked all designs in Xilinx Artix-7 FPGA using Vivado 2020.1 for synthesis and implementation. Minerva [12] was used to search for optimized tool settings to achieve the maximum frequency.

IV. Baseline Unprotected Implementations

A. Elephant

The $mask_K$ values are written such that $mask_K^0$ is the previous mask, $mask_K^1$ is the current mask, and $mask_K^2$ is the next mask. In our implementation, these masks are calculated using a Linear Feedback Shift Register (LFSR).

Fig. 1 shows the implementation of one round of the Spongent permutation. A 7-bit LFSR is initialized to 1111010, and then updated in each subsequent round. The ICounter layer takes the LFSR's output and XORs it with the lowest 7-bits of the input. Additionally, the LFSR's output bits are reversed and then XORed with the highest 7-bits of the input. The output of the ICounter layer proceeds into the S-box layer, which consists of forty 4-bit S-boxes. The S-boxes are implemented as lookup tables. The permutation's final layer is the pLayer, which shifts bit locations based on the following equation.

$$P_{160}(x) = \begin{cases} 40 \cdot x \bmod 159 & \text{if } x \in \{0, ..., 158\} \\ 159, & \text{if } x = 159 \end{cases} \quad (1)$$

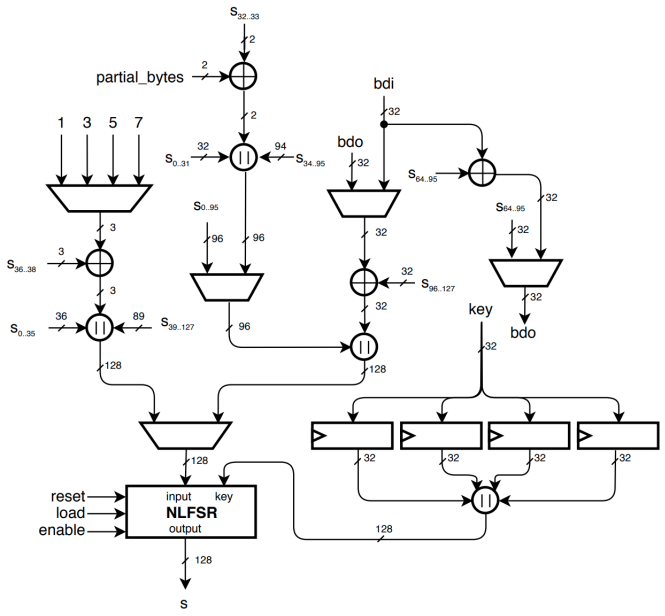

Fig. 2. Unprotected implementation of TinyJAMBU

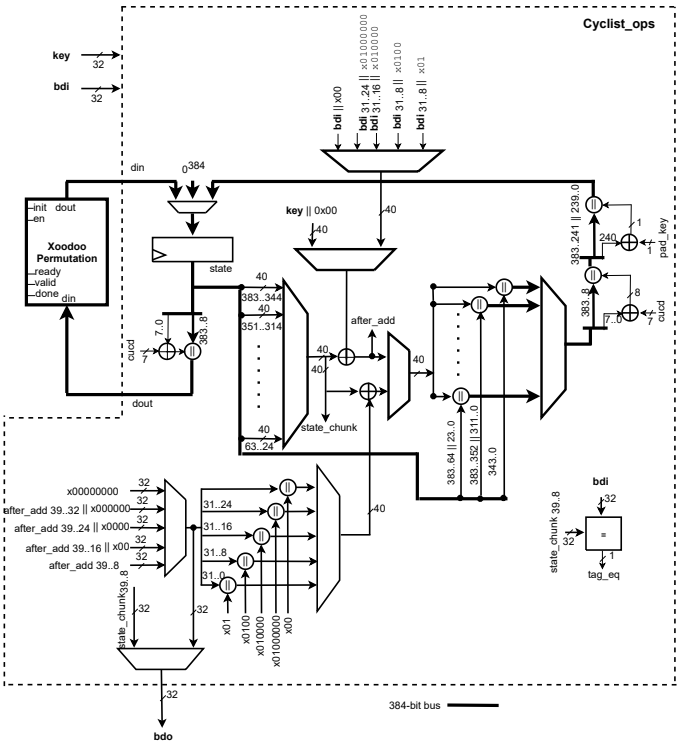

Fig. 3. Full-Width Unprotected Xoodyak

B. TinyJAMBU

The top-level block diagram of the unprotected implementation of TinyJAMBU is shown in Fig. 2.

The key is accepted 32 bits at a time and stored internally in a shift register. Next, the nonce is accepted, and the initialization phase is completed. The associated data is absorbed 32 bits at a time and used to update the state. During encryption, the plaintext is absorbed, and at the same time, the ciphertext is generated. Finally, the tag is generated. When performing decryption, the ciphertext is absorbed, and plaintext is produced. Finally, the tag is calculated and compared to the expected tag, and the status code is emitted based on the comparison outcome. The NLFSR is designed to be capable of computing between 1 and 32 feedback bits in one clock cycle. This value can be configured during synthesis using the parameter *N* which denotes the number of feedback bits computed in parallel.

C. Xoodyak

We developed a full-width variant of Xoodyak where the 384-bit state register can be read or written in one clock cycle. All input and output ports are 32-bit wide. The datapath of this variant is depicted in Fig. 3.

The **key** and **bdi** ports are used to accept secret data (key) and public data (public message number, AD and plaintext/ciphertext), respectively. The **bdo** port is used to output ciphertext/plaintext and tag.

The design is composed of two major units: The Xoodoo permutation and Cyclist_ops unit. The Xoodoo permutation applies the round function to the state 12 times. The Xoodoo round function is implemented in combinational logic, and the whole permutation needs 12 clock cycles. The Xoodoo round function takes 384 bits as input and passes it through the $\theta, \rho_{west}, \iota, \chi$ and ρ_{east} layers discussed in Section II-C. For the ι layer, we use a 12×12 ROM to store the round constants.

The Cyclist_ops unit takes input (key or data) 32 bits at a time and applies padding when necessary. It also adds the *cu* and the *cd* constants in the `UP` and `DOWN` operations, respectively. For the `DOWN` operation, a multiplexer is used to extract the specific word where new data must be added, and another multiplexer to recombine the affected portion of the state with the unaffected part to be written back to the state.

V. Protected Implementations

A. Elephant

The S-box operations within the Spongent permutation are nonlinear. Protecting the S-box using DOM requires that the S-box be reduced to its algebraic normal form (ANF). The ANF of the Elephant S-box was obtained using software developed for decomposition of S-boxes [13], [14]. The S-box's ANF is as follows:

$$F[x, w, v, u] = [EDB0214F7A859C36]$$

$$F[0] = 0 + u + v + w \cdot v + x$$
$$F[1] = 1 + u + w \cdot v + x \cdot u + x \cdot v + x \cdot w + x \cdot w \cdot v$$
$$F[2] = 1 + v + w + x \cdot u + x \cdot w \cdot v$$
$$F[3] = 1 + v \cdot u + w + x + x \cdot u + x \cdot v + x \cdot v \cdot u + x \cdot w \cdot u$$

DOM protection of a three-input AND gate is required to implement this ANF. The protected implementation of the

three-input AND gate was derived as follows:

$$\begin{aligned} x \cdot y \cdot z =& (x_0 + x_1)(y_0 + y_1)(z_0 + z_1) = \\ =& (x_0y_0 + x_0y_1 + x_1y_0 + x_1y_1)(z_0 + z_1) = \\ =& x_0y_0z_0 + x_0y_1z_0 + x_1y_0z_0 + x_1y_1z_0 + \\ & x_0y_0z_1 + x_0y_1z_1 + x_1y_0z_1 + x_1y_1z_1 \end{aligned}$$

The S-box's ANF requires five instances of two-input AND gates and three cases of three-input AND gates. With these gates, 14-bits of randomness are needed per S-box. The Spongent permutation requires 40 S-boxes and therefore 560-bits of random data per cycle. To determine the most efficient way to express the same logic, a Satisfiability Modulo Theories (SMT) solver was used to minimize the number of AND gates. The resulting expression is more complex but more efficient using only three two-input AND gates and two three-input AND gates. Due to the expression's complexity, bold text is added after each instance of an AND gate. The first number in the text represents the number of input bits to the AND gate, and the second number identifies the gate's uniqueness. This expression reduces the number of random bits required per S-box to nine and a total of 360-bits per cycle for the Spongent permutation.

$$F[s3, s2, s1, s0] = [EDB0214F7A859C36]$$

$$\begin{aligned} F[0] =& s0 + s3 + ((s1 + s2) \cdot s1)\textbf{(2,1)} \\ F[1] =& not(s0 + s1 + ((s1 + s2) \cdot s1)\textbf{(2,1)} + \\ & (s3 \cdot (s0 + s1) \cdot (s0 + s2))\textbf{(3,1)} + \\ & ((s1 + s2) \cdot (s0 + s3) \cdot s3)\textbf{(3,2)}) \\ F[2] =& not(s1 + s2 + ((s1 + s2) \cdot (s1 + s3))\textbf{(2,2)} + \\ & ((s1 + s2) \cdot s1)\textbf{(2,1)} + \\ & (s3 \cdot (s0 + s1) \cdot (s0 + s2))\textbf{(3,1)} + \\ & ((s1 + s2) \cdot (s0 + s3) \cdot s3)\textbf{(3,2)}) \\ F[3] =& not(s0 + s2 + s3 + ((s1 + s2) \cdot (s1 + s3))\textbf{(2,2)} + \\ & ((s0 + s3) \cdot (s0 + s1))\textbf{(2,3)} + ((s1 + s2) \cdot s1)\textbf{(2,1)} + \\ & ((s1 + s2) \cdot (s0 + s3) \cdot s3)\textbf{(3,2)}); \end{aligned}$$

B. TinyJAMBU

Building on the unprotected design described in Section IV-B, we developed a first-order SCA-resistant TinyJAMBU implementation using the DOM countermeasure. Fig. 4 depicts the datapath of the first-order protected design. The NLFSR_pr unit is the protected version of the NLFSR used for the permutation. The state register is duplicated along with the logic needed to compute the feedback bits.

The rest of the datapath, which is responsible for implementing the mode of operation and performing data selection (multiplexing), FrameBits bit addition, etc., is denoted TinyJAMBU_ops in Fig. 4. This component is duplicated twice, with each instance residing in a separate domain. Note that constants (FrameBits and partial block length) are added to the state only in Domain0 to avoid negating the operation.

The only nonlinear components in TinyJAMBU are the NAND gates in the NLFSR. We instantiate a first-order DOM-dep multiplier instead of the AND gates in the protected design.

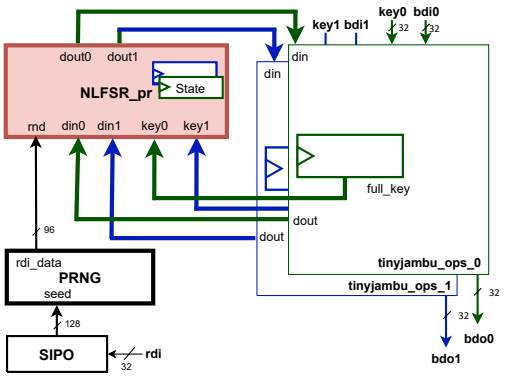

Fig. 4. Protected TinyJAMBU: Top-level Block Diagram

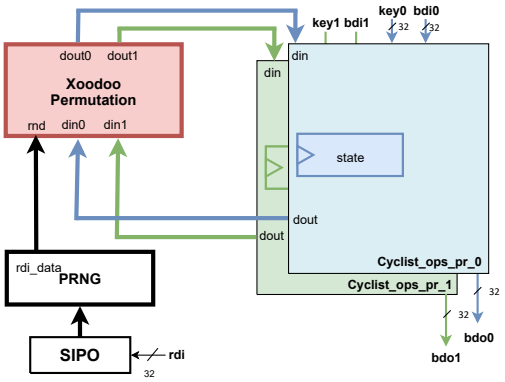

Fig. 5. Protected full-width implementation of Xoodyak – simplified block diagram

We utilized the DOM-dep multiplier [3] to avoid any leakage related to dependently shared variables. Since the multiplier is the only nonlinear part of the design, it is the only component not residing purely in a separate share domain. To turn the AND gates to NAND, we negate Domain0 output of each AND gate. Due to the mandatory synchronization registers in the DOM-dep multiplier, the NLFSR takes two clock cycles to compute N feedback bits. This has the effect of doubling the number of clock cycles needed to perform the permutation. The rest of the NLFSR logic and the state are duplicated to process the two state shares.

C. Xoodyak

Based on the unprotected Xoodyak design discussed in Section IV-C, we developed a first-order SCA-protected design using the DOM countermeasure. We use the optional **rdi** port specified in the LWC API to provide the PRNG seed. The data is initially split into two shares and processed separately in two domains, *Domain0* and *Domain1*. A high-level block diagram of our protected designs is shown in Fig. 5.

The 384-bit state register and the Cyclist_ops unit are duplicated with each instance existing in one domain. We do not perform constant addition in *Domain1*, i.e., the *cu* and *cd* constants are not added when performing the `UP` and `DOWN` operations, and the x01 padding is not performed.

The protected full-width Xoodoo round is shown in Fig. 6. This unit accepts two shares of input and produces two shares of output. The components θ, ρ_{east}, and ρ_{west} are linear operations. These operations are simply duplicated and operate on corresponding shares in *Domain0* and *Domain1* in parallel.

The nonlinear operation χ, is the only place where data from *Domain0* and *Domain1* mix. To implement χ, we instantiate 384 DOM-protected AND gates, each taking two shared variables x and y along with one random bit. Each AND gate uses four registers. Two of these are required for synchronization, and the other two to fully pipeline the gate, as suggested in [3]. This method requires 384 bits of randomness per clock cycle. These bits are generated using a PRNG. The whole χ operation is implemented in two pipeline stages. The permutation needs 13 clock cycles, compared to 12 cycles for the fully combinational Xoodoo round function used in the unprotected design due to the pipeline delay.

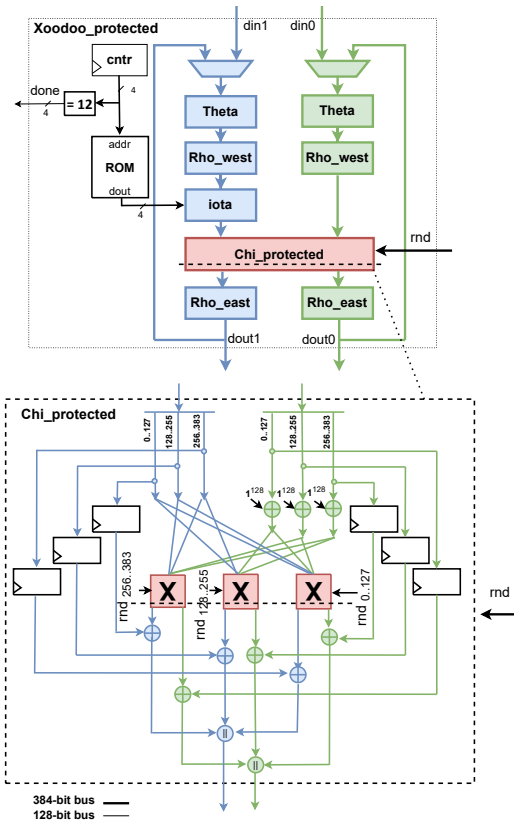

Fig. 6. Full-Width Protected Xoodoo Permutation

VI. Leakage Assessment

We utilized the Test Vector Leakage Assessment (TVLA) methodology [15] to validate the first-order SCA resistance of our protected designs.

The Flexible Opensource workBench fOr Side-channel analysis (FOBOS) [16] platform was used with the NewAE CW305 SCA board, which is based on the Artix-7 FPGA. We ran the FPGA at 1.25 Mz and used a USB3-based oscilloscope (Picoscope 5000) to collect traces at a sampling rate of 125 MS/s.

TABLE I
Area and Throughput on Xilinx Artix-7

Design	Area		Freq.	TP	LUT	TP
	LUTs	FFs	MHz	Mbps	Ratio	Ratio
Unprotected						
Elephant	1291	910	229	214.3	1.00	1.00
TinyJAMBU	591	428	266	250.4	1.00	1.00
Xoodyak	1808	851	170	1717.9	1.00	1.00
Protected						
Elephant	5451	2970	200	93.5	4.22	0.43
TinyJAMBU	1267	1271	247	119.8	2.14	0.48
Xoodyak	6431	4210	158	891.4	4.02	0.42

The TVLA results for the unprotected variants of Elephant, TinyJAMBU, and Xoodyak are shown in Fig. 7. In all cases, the t-value exceeds the threshold, which indicates information leakage as expected. On the other hand, the TVLA of the protected variants, illustrated in Fig. 8 shows no observable leakage even when processing 1 million traces, validating the effectiveness of the applied countermeasures.

VII. Cost of Protection

It is expected that protection against side-channel analysis will come at a price in the area, throughput, power consumption, and energy per bit. Quantifying this cost is critical for a fair evaluation of candidates since the cost of protection varies among candidates.

Benchmarking results for area and throughput of the protected designs are summarized in Table I, and the results for average power and energy per bit are reported in Table II.

The throughput of protected designs decreases by factors greater than 2. For Elephant and TinyJAMBU, the reduction by this factor happens because the permutation requires twice as many clock cycles to complete. Xoodyak's reduction is because the protected implementation operates at a lower max frequency, requires an additional clock cycle to complete the permutation, and is slowed down by the need to load all shares serially. The time to load the shares has less impact for the remaining candidates since the permutation takes significantly more cycles than the time to load shares.

Due to the small size of the non-linearity, the cost of protecting TinyJAMBU is the lowest among the investigated three algorithms. For example, the area of the protected TinyJAMBU is only $2.14\times$ the area of the unprotected baseline variant. Notably, the control logic and constant additions in the datapath are not duplicated in the protected design. For Xoodyak and Elephant, the same factor exceeds 4.

The estimated energy-per-bit of both the protected and unprotected implementations is reported in Table II.

The average power and energy per bit (E/bit) estimations were performed using Xeda [17], which provides an abstraction layer to run flows on different Electronic Design Automation (EDA) tools. For result accuracy, we performed vector-based estimation, and the Switching Activity Interchange Format (SAIF) files were generated while encrypting 1536 bytes of plaintext using post-route timing simulation. Average dynamic power has been estimated at 75 MHz.

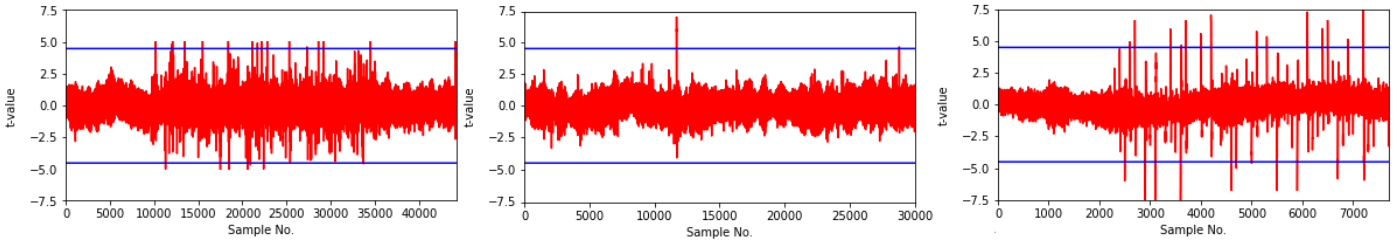

Fig. 7. TVLA Results for Unprotected Designs (10,000 traces). From left to right: Elephant, TinyJAMBU and Xoodyak

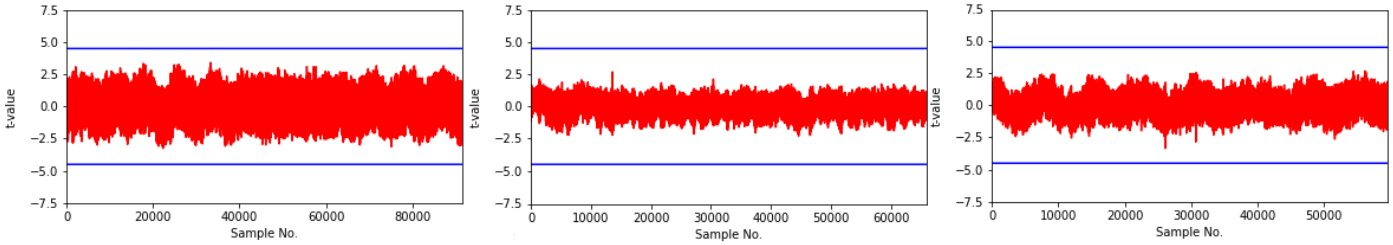

Fig. 8. TVLA Results for Protected Designs (1 million traces) From left to right: Elephant, TinyJAMBU and Xoodyak

TABLE II
ESTIMATED AVERAGE POWER, THROUGHPUT, AND ENERGY-PER-BIT ON XILINX ARTIX-7 (AT 75 MHZ)

Design	Total Power mW	Total Power Ratio	TP @75MHz Mbps	E/bit nJ/bit	E/bit Ratio Ratio
Unprotected					
Elephant	167	1.00	70.2	2.3	1.00
TinyJAMBU	68	1.00	70.6	1.0	1.00
Xoodyak	179	1.00	757.9	0.4	1.00
Protected					
Elephant	205	1.22	35.1	5.8	2.52
TinyJAMBU	127	1.87	36.4	3.5	3.50
Xoodyak	309	1.73	423.2	0.7	1.75

The protected Xoodyak implementation is 5× more efficient than that of TinyJAMBU and 8× more efficient than that of Elephant. This difference comes primarily from processing more bits per clock cycle.

VIII. CONCLUSIONS AND FUTURE WORK

We developed and compared SCA-protected implementations of three finalists in the NIST LWC standardization process. To the best of our knowledge, we present the first protected hardware implementations for Xoodyak and Elephant. The protection overhead is similar for all three algorithms in terms of throughput at the maximum clock frequency. TinyJAMBU excels in area and power in terms of absolute values, while Xoodyak is the best in terms of throughput and energy per bit. Elephant is not the best of three in any category. It is much bigger than TinyJAMBU in terms of area, much slower than Xoodyak in terms of throughput, and the least efficient of all three algorithms in terms of energy per bit.

Extending protections to a higher order and investigating other protection methods are interesting topics for future research.

REFERENCES

[1] NIST, "Recommendation for Block Cipher Modes of Operation: GCM and GMAC," https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38d.pdf, Nov. 2007.

[2] P. Kocher, J. Jaffe, and B. Jun, "Diferential Power Analysis," in *CRYPTO '99 - 19th International Conference on Cryptology*, Santa Barbara, CA, Aug. 1999.

[3] H. Gross, S. Mangard, and T. Korak, "Domain-Oriented Masking: Compact Masked Hardware Implementations with Arbitrary Protection Order," Cryptology ePrint Archive 2016/486, Nov. 2016.

[4] S. Belaïd, P.-É. Dagand, D. Mercadier, M. Rivain, and R. Wintersdorff, "Tornado: Automatic Generation of Probing-Secure Masked Bitsliced Implementations," in *Advances in Cryptology – EUROCRYPT 2020*, vol. 12107. Cham: Springer International Publishing, 2020, pp. 311–341.

[5] A. Abdulgadir, S. Lin, F. Farahmand, J.-P. Kaps, and K. Gaj, "Side-Channel Resistant Implementations of a Novel Lightweight Authenticated Cipher with Application to Hardware Security," in *Proceedings of the 2019 on Great Lakes Symposium on VLSI - GLSVLSI '21*, Virtual, Jun. 2021, p. 6.

[6] F. Coleman, B. Rezvani, S. Sachin, and W. Diehl, "Side Channel Resistance at a Cost: A Comparison of ARX-based Authenticated Encryption," in *2020 30th International Conference on Field-Programmable Logic and Applications (FPL)*. Gothenburg, Sweden: IEEE, Aug. 2020.

[7] T. Beyne, Y. L. Chen, C. Dobraunig, and B. Mennink, "Elephant v1.1," p. 48, 2019.

[8] H. Wu and T. Huang, "TinyJAMBU: A Family of Lightweight Authenticated Encryption Algorithms," *Submission to the NIST Lightweight Cryptography Standardization Process*, Mar. 2019.

[9] J. Daemen, S. Hoffert, M. Peeters, G. V. Assche, and R. V. Keer, "Xoodyak, a lightweight cryptographic scheme," Submission to the NIST Lightweight Cryptography Process, 2019.

[10] J.-P. Kaps, W. Diehl, M. Tempelmeier, E. Homsirikamol, and K. Gaj, "Hardware API for Lightweight Cryptography," GMU, Fairfax, VA, GMU Report, Oct. 2019.

[11] C. De Canniere and B. Preneel, "TRIVIUM Specifications," Tech. Rep., 2005.

[12] F. Farahmand, A. Ferozpuri, W. Diehl, and K. Gaj, "Minerva: Automated hardware optimization tool," in *2017 International Conference on ReConFigurable Computing and FPGAs, ReConFig 2017*. Cancun: IEEE, Dec. 2017, pp. 1–8.

[13] B. Bilgin, S. Nikova, V. Nikov, and V. Rijmen, "TI toolkit." [Online]. Available: http://homes.esat.kuleuven.be/šnikova/ti_tools.html

[14] B. Bilgin, S. Nikova, V. Nikov, V. Rijmen, and G. Stütz, "Threshold implementations of all 3 ×3 and 4 ×4 s-boxes," in *Cryptographic Hardware and Embedded Systems – CHES 2012*, E. Prouff and P. Schaumont, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 76–91.

[15] G. Goodwill, B. Jun, J. Jaffe, and P. Rohatgi, "A testing methodology for side-channel resistance validation," in *NIST Non-Invasive Attack Testing Workshop*, Nara, Japan, 2011.

[16] A. Abdulgadir, W. Diehl, and J.-P. Kaps, "An Open-Source Platform for Evaluating Side-Channel Countermeasures in Hardware Implementations of Lightweight Authenticated Ciphers," Tech. Rep., Nov. 2019.

[17] K. Mohajerani, R. Haeussler, R. Nagpal, F. Farahmand, A. Abdulgadir, J.-P. Kaps, and K. Gaj, "FPGA Benchmarking of Round 2 Candidates in the NIST Lightweight Cryptography Standardization Process: Methodology, Metrics, Tools, and Results," Cryptology ePrint Archive 2020/1207, Oct. 2020.