SCA Evaluation and Benchmarking of Finalists in the NIST Lightweight Cryptography Standardization Process

Kamyar Mohajerani[1], Luke Beckwith[1,2], Abubakr Abdulgadir[2], Eduardo Ferrufino[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
{mmohajer, lbeckwit, aabdulga, eferruf, jkaps, kgaj}@gmu.edu

Abstract. Side-channel resistance is one of the primary criteria identified by NIST for use in evaluating candidates in the Lightweight Cryptography (LWC) Standardization process. In Rounds 1 and 2 of this process, when the number of candidates was still substantial (56 and 32, respectively), evaluating this feature was close to impossible. With ten finalists remaining, side-channel resistance and its effect on the performance and cost of practical implementations became of utmost importance. In this paper, we describe a general framework for evaluating the side-channel resistance of LWC candidates using resources, experience, and general practices of the cryptographic engineering community developed over the last two decades. The primary features of our approach are a) self-identification and self-characterization of side-channel security evaluation labs, b) distributed development of protected hardware and software implementations, matching certain high-level requirements and deliverable formats, and c) dynamic and transparent matching of evaluators with implementers in order to achieve the most meaningful and fair evaluation report. After the classes of hardware implementations with similar resistance to side-channel attacks are established, these implementations are comprehensively benchmarked using Xilinx Artix-7 FPGAs. All implementations belonging to the same class are then ranked according to several performance and cost metrics. Four candidates – Ascon, Xoodyak, TinyJAMBU, and ISAP – are selected as offering unique advantages over other finalists in terms of the throughput, area, throughput-to-area ratio, or randomness requirements of their protected hardware implementations.

Keywords: lightweight cryptography · side-channel analysis · authenticated ciphers · hash functions · hardware · software · benchmarking

1 Introduction

NIST has specified resistance to Side-Channel Analysis (SCA) as one of the primary criteria for evaluating candidates in the Lightweight Cryptography (LWC) Standardization Process [1]. To assist NIST in evaluating finalists in this process, we have developed the following three calls:

1. Call for Side-Channel Security Validation Labs
2. Call for Protected Hardware Implementations, targeting low-cost modern FPGAs
3. Call for Protected Software Implementations, targeting low-cost modern embedded processors.

The general idea was that no single group was likely to have resources and expertise to develop and evaluate SCA-protected implementations of all 10 finalists. Additionally, self-evaluation by developers might have been insufficient and/or error-prone. Therefore, it has been the collective responsibility of the cryptographic engineering community to contribute to the evaluation process and make it as transparent and fair as possible. Contributions by multiple groups have made:

- each group's workload more manageable;
- coverage of implementation platforms more complete;
- results more credible.

These contributions were strongly encouraged and justified by at least the following factors:

- The new LWC standard is likely to be used for decades. Choosing the right algorithm had a potential to save the community countless man-hours necessary to secure implementations of a hard-to-protect standard or start a new standardization process from scratch.
- It was a joint project that multiple experts in the field could focus on in the limited amount of time devoted to analysis. Most implementations have been, by nature, open-source. Most evaluations were transparent and reproducible. This process has revealed and highlighted some implementation and evaluation methods that rarely got fully disclosed and published in the past.
- Automated insertion of countermeasures was highly desirable (especially considering the very short period reserved for developing protected implementations). Insights gained through these developments may lead to tremendous progress in the field of Computer-Aided Design (CAD) tools for SCA.
- The developed protected implementations can become benchmarks for new attacks and leakage assessment methods that can be discovered and published in years to come.
- Research on NIST standards is highly visible. Participants have been rewarded with recognition by the cryptographic community that may translate to new collaboration, funding, and publication opportunities.

The results of this effort were presented to NIST on October 27, 2022. The corresponding slides were published on the George Mason University website titled "Lightweight Cryptography in Hardware and Embedded Systems"[1] under "Evaluation of Finalists in the NIST LWC Process" shortly after and announced on lwc-forum on November 1, 2022. Minor modifications and extensions were made on November 25, 2022. This report is a written record of these earlier presentations, providing additional details, numerical results, and additional commentary. It is published for archival purposes and to support NIST efforts on providing full justification and explanation regarding the choice of Ascon as a future federal lightweight cryptography standard.

2 Side-Channel Security Evaluation Labs

2.1 General Idea

We called for groups capable and willing to serve as side-channel security evaluation labs to identify their capabilities and contribute to the evaluation process. Our draft call was sent for comments to `lwc-forum` in December 2021. The final version of this call was published on January 18, 2022. The deadline for submitting lab specifications was initially set to February 28, 2022, and then extended to March 15, 2022, for groups that expressed initial interest.

The assumption was that submitters should have access to the equipment used for side-channel leakage assessment and/or attacks, experience, and human resources necessary to perform security analysis. Suggested devices used for evaluating hardware implementations were low-cost modern FPGAs, such as Artix-7 and Spartan-7 from Xilinx, Cyclone 10 LP from Intel, and ECP5 from Lattice Semiconductor. Suggested embedded processors used for evaluating software implementations were ARM Cortex-M4F, RISC-V (e.g., RV32IMAC), Microchip 8-bit AVR, and TI MSP430. A particular lab could specialize in evaluating only hardware implementations, only software implementations, or both.

[1] https://cryptography.gmu.edu/athena/index.php?id=LWC

Table 1: Side-Channel Security Evaluation Labs that Reported Experiments Targeting Hardware Implementations

No.	**Team**	**Evaluation Platform**	**Target FPGA Family**	**Target Boards**	**Leakage Assessment Methods**	**Attacks**
1	IAIK, TU Graz, Austria	NewAE ChipWhisperer	Artix-7	NewAE CW305	t-test	
2	CCSL, Shanghai Jiao Tong University, China	Riscure Inspector, NewAE ChipWhisperer, SAKURA	Kintex-7, Spartan-6	SAKURA-G, SAKURA-X	t-test, χ^2-test, DL-LA	CPA, TA, MIA, DL-based methods
3	HSCP Lab, Tsinghua University, Beijing, China	SAKURA	Kintex-7, Spartan-6	SAKURA-G, SAKURA-X	NICV, t-test, χ^2-test	SPA, DPA, CPA, MIA, TA, LRA,
4	Secure-IC, France	Secure-IC Analyzr, SAKURA	Spartan-6	SAKURA-G	Tests specified in ISO/IEC 17825:2016	
5	CERG, George Mason University, USA	FOBOS3	Artix-7	NewAE CW305	t-test	
6	Ruhr-Universitat Bochum, Germany	PROLEAD and other simulation-based probing security leakage-detection tools			simulation-based probing security evaluation	

Table 2: Side-Channel Security Evaluation Labs that Reported Experiments Targeting Software Implementations

No.	**Team**	**Evaluation Platform**	**Target Processors**	**Leakage Assessment Methods**	**Attacks**
1	CCSL, Shanghai Jiao Tong University, China	Riscure Inspector, NewAE ChipWhisperer	ARM Cortex-M4F, ATxmega128D4, ATmega128A	t-test, χ^2-test, DL-LA	CPA, TA, MIA, DL-based methods
2	HSCP Lab, Tsinghua University, Beijing, China		ARM Cortex-M4F, ARM Cortex-M3	NICV, t-test, χ^2-test	SPA, DPA, CPA, MIA, TA, LRA
3	CESCA Lab, Radboud University, the Netherlands	Riscure Inspector, NewAE ChipWhisperer, Jupyter notebook scripts	ARM Cortex-M4F, ATxmega128D4	t-test, χ^2-test, DL-LA	SPA, DPA, CPA TA; DEMA; DFA, FI attacks

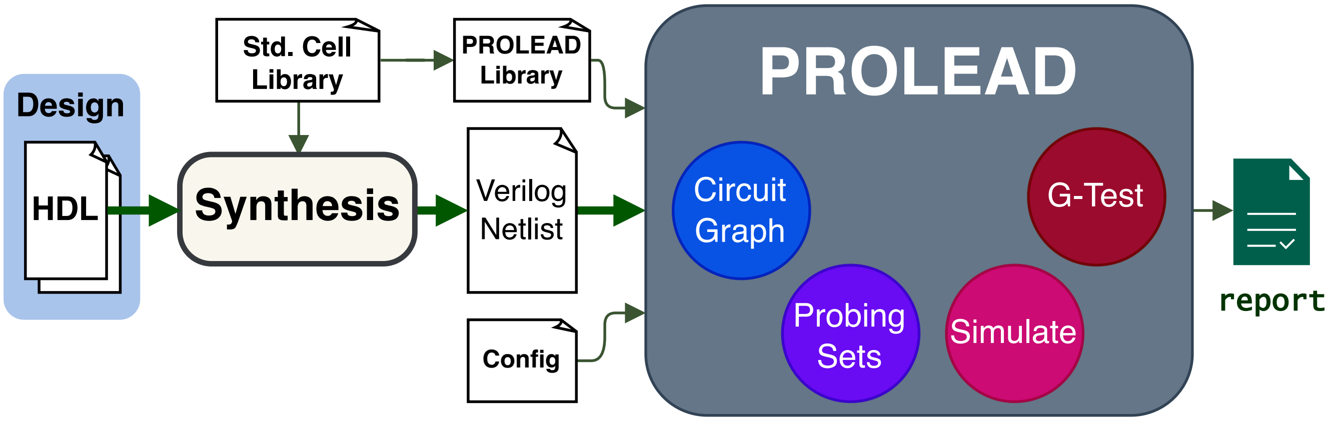

Figure 1: A simulation-based evaluation of protected hardware designs using PROLEAD

2.2 Lab Specifications

The specifications of labs that reported results of experiments targeting protected hardware and software implementations are summarized in Tables 1 and 2. These tables describe

- 2 labs that supported both software and hardware implementations,
- 4 labs that supported only hardware implementations, and
- 1 lab that supported only software implementations.

The detailed specifications are posted on our ATHENa Lightweight Cryptography web page at https://cryptography.gmu.edu/athena/index.php?id=LWC.

In Table 1, we summarize the major capabilities of the labs targeting hardware implementations in terms of the Evaluation Platform, Target FPGA Family, Target Board, Leakage Assessment Methods, and Key Recovery Attacks.

The most popular Evaluation Platforms were NewAE ChipWhisper and SAKURA, declared by 4 out of 6 labs. Riscure Inspector, Secure-IC Analyzr, and FOBOS3 were used by one lab each. PROLEAD [2], which is a leakage-detection tool based on simulation and probing security model, was used by one lab.

Four labs supported Xilinx 7 Series FPGA families, such as Artix-7 and Kintex-7, based on six-input Look-Up Tables (LUTs). Three labs supported Spartan-6 based on four-input Look-Up Tables (LUTs). Among the Target Boards, the most popular were SAKURA boards and NewAE CW305.

The most widely supported Leakage Assessment Method was Welch's t-test a.k.a. TVLA (Test Vector Leakage Assessment) [3]–[8]. Two labs supported a newer and supplementary Pearson's χ^2-test introduced in [9]. The team representing Secure-IC used tests specified in ISO/IEC 17825:2016. These tests were described and critically analyzed in [10]. One lab declared support for NICV: Normalized Inter-Class Variance for Detection of Side-Channel Leakage [11], [12]. One lab listed among their methods DL-LA: Deep Learning Leakage Assessment, defined in [13]. The team from Ruhr-Universitat Bochum relied on the simulation-based tool called PROLEAD [2]. The procedure for leakage assessment with PROLEAD is depicted in Fig. 1. After the full design is synthesized, the netlist is provided to the tool, along with a description of gates (PROLEAD library) and a configuration file with details about the design operation (e.g., random and fixed inputs for the simulation step) and tool settings (e.g., number of simulations). For a d order evaluation, PROLEAD analyzes the circuit graph, generates all possible sets of d glitch- and transition- extended probes [14], simulates the design using the configured fixed and random inputs, and then analyzes the observed value on the probes using statistical G-test. By using the extended robust probing model, PROLEAD can narrow down the choice of probing sets to only primary outputs and register inputs of the design while being able to detect implementation flaws, including those arising from physical defaults such as glitches and transitions.

Two labs supported attacks, such as Simple Power Analysis (SPA), Differential Power Analysis (DPA), Correlation Power Analysis (CPA), Template Attacks (TA), Mutual Information Analysis (MIA), and Deep Learning (DL)-based methods [15].

In Table 2, we summarize major capabilities of the labs targeting software implementations in terms of the Evaluation Platform, Target Processors, Leakage Assessment Methods, and Key Recovery Attacks. The most supported target processor was ARM Cortex-M4F, listed by all three labs. Two labs supported ATxmega128D4. ATmega128A and ARM Cortex-M3 were supported by one lab each. In terms of the Leakage Assessment Methods, all labs supported the t-test and χ^2-test. The third most popular test was the Deep Learning Leakage Assessment (DL-LA), supported by two out of three labs. The most supported attacks were Correlation Power Analysis (CPA) and Template Attacks (TA).

3 Protected Hardware Implementations

3.1 Introduction

We submitted a draft version of the Call for Protected Hardware Implementations to `lwc-forum` on December 13, 2021. After analyzing all received comments and incorporating the best-received suggestions, we posted a final version of this call on the GMU Lightweight Cryptography website on January 18, 2022. According to the call, the submitted designs were expected to demonstrate strong resistance against side-channel attacks when implemented on low-cost modern FPGAs, such as Artix-7 and Spartan-7 from Xilinx, Cyclone 10 LP from Intel, and ECP5 from Lattice Semiconductor. A potential for porting the designs to ASIC (Application-Specific Integrated Circuit) technology and demonstrating their resistance in this environment was highly desirable. All submitted implementations were planned to be investigated by one or more Side-Channel Security Evaluation Labs.

3.2 Requirements

Protected hardware implementations were required to follow the LWC Hardware API v1.2.0 or later. In this extended API, we assumed that inputs and outputs are split into shares, as shown in Fig. 2. Input that is not shared (e.g., an instruction or a segment header) is put into share 1, with the remaining shares being set to zeros. The updated interface is shown in Fig. 3. In unprotected implementations, the public data input PDI accepts data of size w. For protected implementations, we modified this input to accept pn shares of size w in parallel. The same holds for the data output DO, which now provides pn shares of size w. The number of shares on the secret data input SDI is denoted as sn, as it can differ from the number of shares on PDI.

A majority of common side-channel countermeasures require the consumption of randomness during cipher operations. Any randomness an LWC implementation needs can be provided by the random data input RDI, which is of size rw. This port, just like all the others, follows a simple FIFO protocol. Each read will provide rw bits. The value of rw can be arbitrary up to 2048 bits. Note that independent of how many random bits are actually used, our testbench assumes that all rw bits are used with each read.

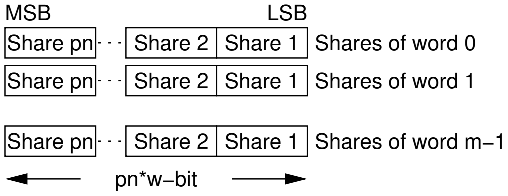

Figure 2: Pre-Shared Data

We also assume that a deterministic random bit generator (DRBG) used as a source of fresh randomness is located outside of the protected LWC core. The important advantages of this approach include:

- ability to share DBRG with other units (e.g., for the generation of nonces, protection of other units, e.g., those implementing public-key cryptography, etc.)
- ease of replacing the type of DBRG (e.g., due to compliance with other standards, validation requirements, evolving understanding of how cryptographically strong the DBRG used for refreshing randomness must be, etc.)

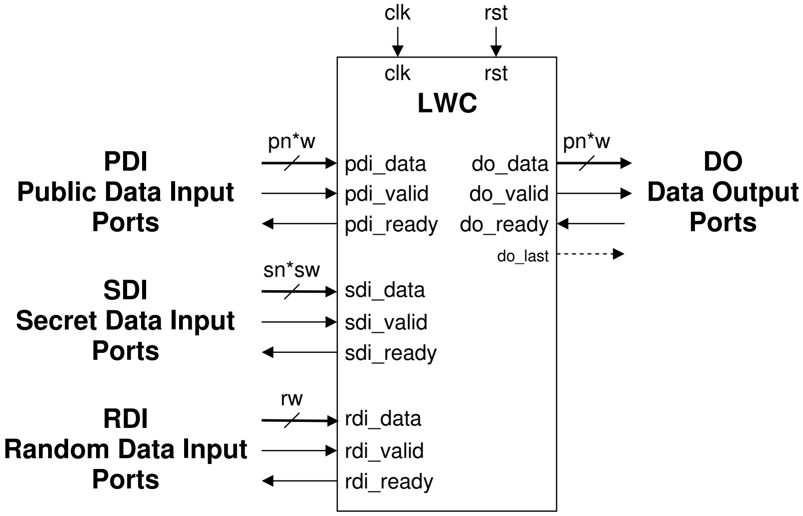

Figure 3: LWC API extended with Random Data Input (RDI)

- we are interested in evaluating/benchmarking LWC candidates and not DBRGs. The final NIST LWC selection itself could become the basis of future lightweight DBRGs.
- Concurrent operation of the DBRG circuit could introduce additional noise in the measurements and make leakage detection more difficult. There is no guarantee that this type of noise by itself could hinder an actual attack scenario, but it is likely to make the leakage evaluation more difficult (more traces, more computations, or more expensive measurement equipment).

Our testbench counted how many random bits were consumed by the protected implementation during its operation and used this information, together with the width of the rdi_data bus, to differentiate between various protected designs. Specifically, the total number of consumed fresh random bits was one of the major items on the list of reported evaluation metrics.

We proposed the following constraints on a first-order protected implementation of an LWC candidate: 8000 LUTs, 0 Block RAMs, and 0 DSP units of Artix-7 FPGAs. The number of LUTs corresponded to the smallest device of the Artix-7 family of FPGAs. This number is also consistent with the Round 2 limit on the number of LUTs, set to 2000 LUTs, and the observation that the first-order protected hardware implementations typically took 3-4x more hardware resources than the corresponding unprotected implementations. For the implementations of two-pass algorithms, the memory (FIFO) required for the second-pass processing is instantiated as Block RAM, but stays outside of the LWC boundary and is not accounted for in the reported resource utilization of these implementations.

Table 3: Proposed constraints on resource utilization

Type of Implementation	#LUTs	#BRAMs	#DSP units
Unprotected	≤ 2000	0	0
1st Order Protected	≤ 8000	0	0

3.3 Submissions

In response to our call for protected implementations, 42 protected hardware designs were received from 4 groups, covering 9 out of 10 LWC finalist schemes (all except Grain-128AEAD). These implementations are summarized in Table 4.

ISAP [16] provides mode-level robustness against a large class of implementation attacks (such as Differential Power Analysis (DPA) and fault attacks) through the usage of leakage-resilient re-keying and a two-pass construction. All other protected implementations use masking as a countermeasure against power and electromagnetic (EM) side-channel attacks. ISAP specification recommends two underlying cryptographic permutations: Ascon-p (same as Ascon-128a) for the primary variant ISAP-A-128a, and Keccak-p[400] for the secondary variant ISAP-K-128a. ISAP team's hardware submission [2] included 5 variants of ISAP-A-128a (32, 16, 8-bit interface, 2x unrolled, and StP-based tag verification) and 1 variant of ISAP-K-128a (only with 16-bit interface). Due to hardware similarities, only ISAP-A-128a with the 32-bit interface (ISAP-A_Graz_dn) and ISAP-K-128a (ISAP-K_Graz_dn16) were benchmarked. Masked implementations of ISAP provide side-channel resistance in hashing mode, as well as improved resistance against simple power analysis and template attacks.

All masked implementations are based on previously released unprotected hardware designs as listed in Table 5. Among the masked designs, 6 designs are manually protected. Three of them were developed for Xoodyak, two for Ascon, and one for TinyJAMBU.

Thirty masked implementations have been generated by utilizing AGEMA [17], a tool for the semi-automated generation of masked hardware. These implementations were generated by Ruhr-University Bochum. The flow for generating masked implementations using AGEMA is depicted in Fig. 4. AGEMA operates on a synthesized netlist, identifies the wires and gates that need to be secured, and replaces them with their masked versions. To ensure secure masking, AGEMA relies on the concept of Probe-Isolating Non-Interference (PINI) and composable gadgets. Due to the insertion of extra gadget registers, the control logic of the design needs to be modified accordingly, but AGEMA is not able to detect or make the necessary adjustments to the control logic. Additionally, portions of the design which handle protocol-level and handshaking details need to be manually modified. As a result, only the combinational cryptographic permutations were processed by AGEMA and were subsequently integrated into the designs through manual modification and the use of the updated LWC package with support for masked implementations. Out of the 30 AGEMA designs made available for benchmarking, three either failed verification or synthesis, mapping, placing, and routing.

The manually protected designs use Domain Oriented Masking (DOM) [18] (Ascon-128_Graz_d{1,2}, TinyJAMBU_GMU_d1, Xoodyak_GMU_d1, and Xoodyak_Tsinghua_d1DOM) and Threshold Implementation (TI) [19] (Xoodyak_Tsinghua_d1TI) masking schemes. The semi-automatically protected designs utilize HPC2 [20] composable gadgets. For security order d, TI-based implementations require $t \cdot d + 1$ shares, where $t \geq 2$ is the multiplicative complexity of the non-linear portion of the design (t=2 in case of Xoodyak_Tsinghua_d1TI). The DOM and HPC2 schemes require $d + 1$ shares.

A total of 63 designs, including 35 protected and 28 unprotected implementations, were benchmarked for performance and FPGA resource utilization (area).

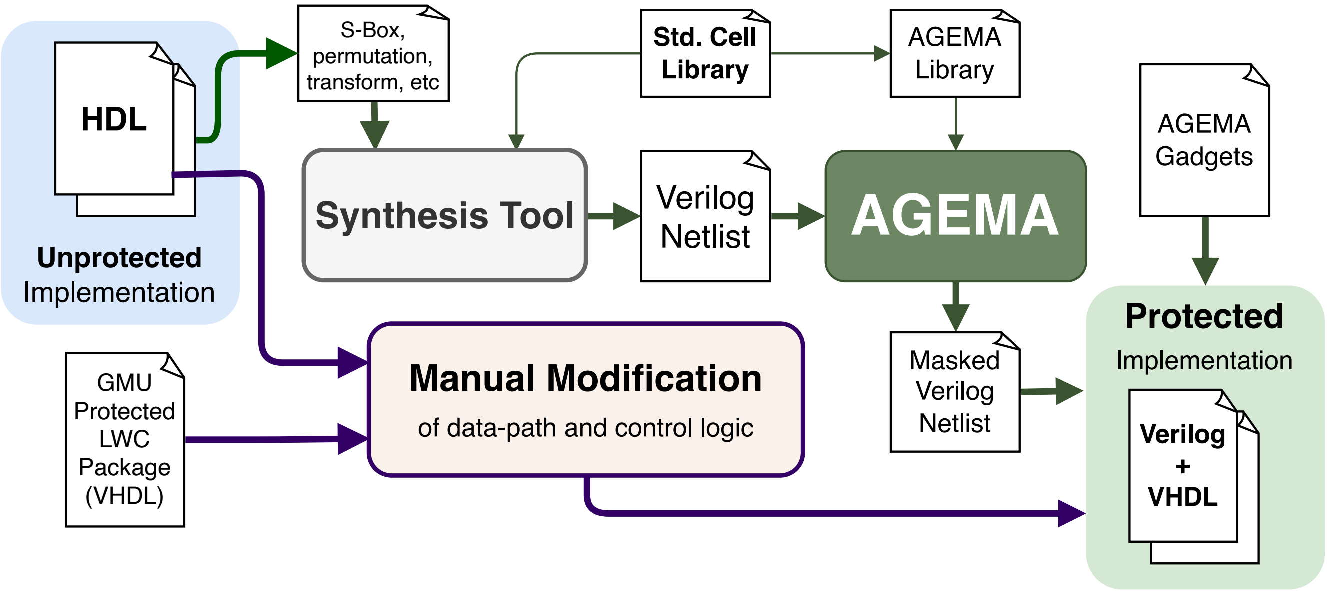

Figure 4: Development of protected hardware designs using AGEMA

[2] https://github.com/isap-lwc/isap-hardware-package

Table 4: Summary of protected and unprotected hardware designs used in this study (M: manually protected, A: protected using AGEMA)

Finalist	**Unprotected**	**Order 1**	**Order 2**	**Order 3**
Ascon	Graz GMU (2)	M: Graz A: Bochum (2)	M: Graz A: Bochum (2)	A: Bochum (2)
Elephant	GMU	A: Bochum	A: Bochum	A: Bochum
Grain-128AEAD	GMU			
GIFT-COFB	VT GMU	A: Bochum	A: Bochum	A: Bochum
ISAP (Masked)	Graz	A: Bochum	A: Bochum	A: Bochum
ISAP	Graz (mode-level protection) (6)			
PHOTON-Beetle	GMU	A: Bochum	A: Bochum	A: Bochum
Romulus	NTU	A: Bochum	A: Bochum	A: Bochum
SPARKLE	VT GMU	A: Bochum	A: Bochum	A: Bochum
TinyJAMBU	GMU TJ Team	M: GMU A: Bochum	A: Bochum	A: Bochum
Xoodyak	XT Team GMU (2)	M: Tsinghua (2) M: GMU A: Bochum	A: Bochum	A: Bochum

Table 5: Protected implementations and the corresponding unprotected designs they are based on. *Fastest* and *Smallest* columns specify whether the unprotected base has the highest encryption (PT) throughput and/or lowest area (number of LUTs) respectively among the benchmarked unprotected implementations of the same scheme. "OA" signifies that the unprotected base was the only implementation available and "OO" means that the unprotected base was the only implementation which the source code was publicly available.

Implementation	**Unprotected Base**	**Fastest**	**Smallest**
Ascon-128_Bochum_d{1,2,3}	Ascon-128_Graz-x1	✗	✗
Ascon-128_Graz_d{1,2}	Ascon-128_Graz-x1	✗	✗
Ascon-128a_Bochum_d{1,2,3}	Ascon-128a_Graz-x1	✗	✓
Elephant_Bochum_d{1,2,3}	Elephant_GMU	OA	OA
GIFT-COFB_Bochum_d{1,2,3}	GIFT-COFB_VT	✗	✗
ISAP-A_Bochum_d{1,2,3}	ISAP-A_Graz_dn	OA	OA
PHOTON-Beetle_Bochum_d{1,2,3}	PHOTON-Beetle_GMU	OA	OA
Romulus-N_Bochum_d{1,2,3}	Romulus-N_RT-x1	OA	OA
SPARKLE_Bochum_d{1,2,3}	SPARKLE_VT	✗	✗
TinyJAMBU_Bochum_d{1,2,3}	TinyJAMBU_GMU	OO	OO
TinyJAMBU_GMU_d1	TinyJAMBU_GMU	OO	OO
Xoodyak_Bochum_d{1,2,3}	Xoodyak_XT-x1	✗	✓
Xoodyak_GMU_d1	Xoodyak_GMU-x1	✗	✗
Xoodyak_Tsinghua_d1{DOM,TI}	Xoodyak_XT-x1	✗	✓

Table 6: Protected Hardware Implementations of LWC Finalists

Candidates	**Protection Order**	**Protection Method**	**HDL**	**Variants**	**Initial Evaluation**	**Primary Hardware Designers**	**Academic Advisors / Program Managers**
ISAP[3]	N/A	Mode-level robustness	VHDL	6	Analytical	Robert Primas	Stefan Mangard
Ascon **Elephant** **GIFT-COFB** **ISAP** **PHOTON-Beetle**[4] **Romulus** **SPARKLE** **TinyJAMBU** **Xoodyak**	1, 2, 3	HPC2	Verilog + VHDL	Ascon:6 Others: 3	PROLEAD [2]		Amir Moradi
TinyJAMBU[5] **Xoodyak**[6]	1	DOM	VHDL	1	t-test	TinyJAMBU: Sammy Lin, Abubakr Abdulgadir Xoodyak: Abubakr Abdulgadir, Richard Haeussler	Jens-Peter Kaps, Kris Gaj
Ascon[7]	1, 2	DOM	VHDL	1	CocoAlma [21]	Robert Primas, Rishub Nagpal	Stefan Mangard
Xoodyak[8]	1	DOM, TI	Verilog + VHDL	2	t-test	Shuohang Peng, Shuying Yin, Cankun Zhao	Leibo Liu, Bohan Yang, Wenping Zhu

[3] https://github.com/isap-lwc/isap-hardware-package

[4] https://github.com/Chair-for-Security-Engineering/LWC-Masking

[5] https://github.com/GMUCERG/TinyJAMBU-SCA

[6] https://github.com/GMUCERG/Xoodyak-SCA

[7] https://github.com/ascon/ascon-hardware

[8] https://github.com/ybhphoenix/THU_HWSec_LWC

4 Protected Software Implementations

We called for software implementations of finalists resistant against side-channel attacks such as power and electromagnetic analysis, using the same timeline as in the case of hardware implementations. The focus of our call was on the use of platform-independent algorithmic countermeasures. The submitted code was expected to demonstrate strong resistance against side-channel attacks when executed on low-cost modern embedded processors, such as ARM Cortex M4F, RISC-V (e.g., RV32IMAC), Microchip 8-bit AVR, and TI MSP430. This code could contain assembly language instructions specific to a given Instruction Set Architecture (ISA).

Protected software implementations were expected to use the standard NIST API defined in Submission Requirements and Evaluation Criteria for the Lightweight Cryptography Standardization Process, published in August 2018[9]. Protected implementations were not allowed to use `nsec`, beyond specifying it as an argument of `crypto_aead_encrypt()` and `crypto_aead_decrypt()` set to NULL.

5 GMU Team role

Our team was in communication with the evaluation labs and the implementation submitters aiming at the best match between both groups. The final matches are summarized at the GMU Lightweight Cryptography in Hardware and Embedded Systems web page[10].

Our team also performed a t-test on the protected implementations of Ascon, Elephant, PHOTON-Beetle, TinyJAMBU, and Xoodyak developed using AGEMA at the Ruhr University of Bochum.

Additionally, the GMU team benchmarked and ranked implementations with a comparable security level in terms of Throughput, Area, Throughput/Area, and the number of random bits per each byte of plaintext and associated data (AD). The benchmarking was performed using the Xilinx Artix-7 family of FPGA devices.

Our team has published the record of evaluations in progress and reports from the completed evaluations on the mentioned above website.

6 Side-channel Evaluation Results

The parameters of security validation experiments and the corresponding results are shown in Tables 7 and 8 for hardware implementations and Tables 9 and 10 for software implementations. The tables capture the significant parameters of each experiment and results, and interested readers are referred to the detailed reports available on the ATHENa Lightweight Cryptography web page[11,12]. The goal of these tests was to provide confidence in the effectiveness of the countermeasures to achieve the stated security level. Additionally, feedback from evaluations was helpful for implementation teams to refine their implementations and fix bugs so that benchmarked designs were as close as possible to achieving the claimed side-channel resistance.

6.1 Hardware Implementations Result Summary

Table 7 shows that most of the tests are leakage assessment tests. Specifically, the Test Vector Leakage Assessment [3], [5] and χ^2-test [9] have been used. The attack performed was Correlation Power Analysis (CPA). In one instance, a template attack (TA) was attempted.

The most used targets (Evaluation Platforms) were NewAE ChipWhisperer CW305, SASEBO-GIII, and SAKURA boards using Xilinx Artix-7 and Kintex-7 FPGAs. These targets were clocked at 1-100 MHz, and side-channel information was measured using both shunt resistors and electromagnetic emanation (EM).

A wide range of oscilloscope settings has been used. The sampling rate varied from 22 MHz to 6.25 GHz, and the resolution from 8 to 12 bits. Most of the experiments used sampling clocks that were not synchronized to the target clock. The experiments performed by CERG use the FOBOS control board, which contains a version of OpenADC, as the oscilloscope.

[9] https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/final-lwc-submission-requirements-august2018.pdf

[10] https://cryptography.gmu.edu/athena/index.php?id=LWC

[11] https://cryptography.gmu.edu/athena/LWC/Lab_Implementation_Matching_HW.html

[12] https://cryptography.gmu.edu/athena/LWC/Lab_Implementation_Matching_SW.html

Table 7: Results of Side-channel Evaluation of Protected Hardware Implementations

Implementation	**Lab**	**Target**	**Oscilloscope**	**Freq.** [MHz]	**Sampl. Freq.** [MS/s]	**Reso-lution** [bits]	**Meas. Type**	**Test**
Ascon_Bochum_d1	CERG	CW305	FOBOS3 ADC	16	80	10	power	TVLA
Ascon_Bochum_d1	IAIK	CW305	PicoScope 6404C	1	22	8	power	TVLA
Ascon_Bochum_d1	CCSL	SAKURA-X	LeCroy 610Zi		1000	8	EM	TVLA
Ascon_Bochum_d1	CCSL	SAKURA-X	LeCroy 610Zi		1000	8	EM	χ^2-test
Ascon_Bochum_d1	CCSL	SAKURA-X	LeCroy 610Zi		1000	8	EM	CPA
Ascon_v1_Graz_d1	HSCP	SAKURA-G	WaveRunner 8404M	4	100	8	power	TVLA
Elephant_Bochum_d1	CERG	CW305	FOBOS3 ADC	10	50	10	power	TVLA
Elephant_Bochum_d1	IAIK	CW305	PicoScope 6404C	1	22	8	power	TVLA
GIFT_COFB_Bochum_d1	IAIK	CW305	PicoScope 6404C	1	22	8	power	TVLA
GIFT_COFB_Bochum_d1	CCSL	SASEBO-GIII			500	8	EM	TVLA
GIFT_COFB_Bochum_d1	CCSL	SASEBO-GIII			500	8	EM	χ^2-test
GIFT_COFB_Bochum_d1	CCSL	SASEBO-GIII			500	8	EM	χ^2-test
GIFT_COFB_Bochum_d1	CCSL	SASEBO-GIII			500	8	EM	CPA
ISAP_Bochum_d1	CCSL	Kintex 7	LeCroy 610Zi				EM	CPA
ISAP_Bochum_d1	CCSL	Kintex 7	LeCroy 610Zi				EM	TVLA
ISAP_Bochum_d1	CCSL	Kintex 7	LeCroy 610Zi				EM	χ^2-test
ISAP_Graz	CCSL	Kintex 7	LeCroy 610Zi				EM	CPA
Photon Beetle_Bochum_d1	CERG	CW305	FOBOS3 ADC	16	80	10	power	TVLA
Romulus_Bochum_d1	IAIK	CW305	PicoScope 6404C	1	22	8	power	TVLA
Romulus_Bochum_d1	CCSL	SASEBO-GIII			500	8	EM	TVLA
Romulus_Bochum_d1	CCSL	SASEBO-GIII			500	8	EM	TVLA
Romulus_Bochum_d1	CCSL	SASEBO-GIII			500	8	EM	χ^2-test
Romulus_Bochum_d1	CCSL	SASEBO-GIII			500	8	EM	χ^2-test
Romulus_Bochum_d1	CCSL	SASEBO-GIII			500	8	EM	CPA
Romulus_Bochum_d1	CCSL	SASEBO-GIII			500	8	EM	TA
TinyJAMBU_Bochum_d1	CERG	CW305	FOBOS3 ADC	10	50	10	power	TVLA
TinyJAMBU_GMU_d1	HSCP	SAKURA-G	WaveRunner 8404M	4	100	8	power	TVLA
Xoodyak_Bochum_d1	IAIK	CW305	PicoScope 6404C	1	22	8	power	TVLA
Xoodyak_GMU_d1	Secure-IC	Arty A7	Tektronix MSO64	100	6250	12	EM	TVLA
Xoodyak_Bochum_d1	CERG	CW305	FOBOS3 ADC	10	50	10	power	TVLA

Table 8: Results of Side-channel Evaluation of Protected Hardware Implementations

Implementation	Lab	Test	Num. of Traces [×10^6]	Thresh. Exc.	Notes
Ascon_Bochum_d1	**CERG**	**TVLA**	**10**	**Y(1.5M)**	6 out of 1000+ samples exceed the threshold
Ascon_Bochum_d1	IAIK	TVLA	10	N	
Ascon_Bochum_d1	CCSL	TVLA	1	N	
Ascon_Bochum_d1	CCSL	χ^2-test	1	N	
Ascon_Bochum_d1	CCSL	CPA	11	–	No bytes revealed
Ascon_v1_Graz_d1	HSCP	TVLA	10	N	
Elephant_Bochum_d1	**CERG**	**TVLA**	**7**	**Y(2.7M)**	3 out of 12,000+ samples exceed the threshold
Elephant_Bochum_d1	IAIK	TVLA	10	N	
GIFT_COFB_Bochum_d1	IAIK	TVLA	10	N	
GIFT_COFB_Bochum_d1	CCSL	TVLA	1	N	Classification based on a nonce bit. A similar test was also based on a bit in an intermediate value.
GIFT_COFB_Bochum_d1	**CCSL**	**χ^2-test**	**1**	**Y**	Classification based on a nonce bit: threshold exceeded
GIFT_COFB_Bochum_d1	CCSL	χ^2-test	1	N	Classification based on a bit in an intermediate value
GIFT_COFB_Bochum_d1	CCSL	CPA	1	–	Key not revealed
ISAP_Bochum_d1	CCSL	CPA		–	Key not revealed
ISAP_Bochum_d1	**CCSL**	**TVLA**		**Y**	Some samples exceeding the threshold observed
ISAP_Bochum_d1	**CCSL**	**χ^2-test**		**Y**	Some samples exceeding the threshold observed
ISAP_Graz	CCSL	CPA		–	Key not revealed
PHOTON-Beetle_Bochum_d1	CERG	TVLA	10	N	t-values crossed threshold briefly before returning below threshold
Romulus_Bochum_d1	IAIK	TVLA	10	N	
Romulus_Bochum_d1	**CCSL**	**TVLA**	**10**	**Y**	Case A: Few samples exceed the threshold at 1 M traces. Classification based on a nonce bit.
Romulus_Bochum_d1	CCSL	TVLA	1	N	Case B: No samples exceed the threshold at 1 M traces. Classification based on an intermediate bit.
Romulus_Bochum_d1	**CCSL**	**χ^2-test**	**1**	**Y**	Case A: Few samples exceed the threshold at 1 M traces. Classification based on a nonce bit.

Romulus_Bochum_d1	CCSL	χ^2-test	1	N	Case B: No samples exceed the threshold at 1 M traces. Classification based on an intermediate bit.
Romulus_Bochum_d1	CCSL	CPA	1	–	Key not revealed
Romulus_Bochum_d1	CCSL	TA	1	–	Key not revealed
TinyJAMBU_Bochum_d1	CERG	TVLA	10	N	One sample exceeded the threshold so the test repeated again. Another sample exceeded the threshold but at another location indicating a false positive.
TinyJAMBU_GMU_d1	HSCP	TVLA	10	N	
Xoodyak_Bochum_d1	IAIK	TVLA	10	N	
Xoodyak_GMU_d1	Secure-IC	TVLA	0.1	N	Classification based on an input plaintext bit
Xoodyak_Bochum_d1	**CERG**	**TVLA**	**10**	**Y(3.2M)**	10 out of 900 samples exceed the threshold

Table 9: Results of Side-channel Evaluation of Protected Software Implementations

Implementation	**Lab**	**Target**	**Oscilloscope**	**Freq.** [MHz]	**Sampl. Freq.** [MS/s]	**Resolution** [bits]	**Meas. Type**	**Test**
Ascon_Graz_d1	CESCA	STM32F407	Pico 3206D	168	100-1000	8	EM	CPA
Ascon_Graz_d2	CCSL	STM32F303	Pico 3203D		62.5	16	EM	TVLA
Ascon_Graz_d2	CCSL	STM32F303	Pico 3203D		62.5	16	EM	X2-test
Ascon_Graz_d2	CCSL	STM32F303	Pico 3203D		62.5	16	EM	CPA
GIFT_COFB_Adomnicai	CCSL	STM32F303			125	16	EM	TVLA
GIFT_COFB_Adomnicai	CCSL	STM32F303			125	16	EM	X2-test
GIFT_COFB_Adomnicai	CCSL	STM32F303			125	16	EM	CPA
GIFT-COFB_Adominicai	HSCP	STM32F303		8	25	8	power	TVLA
ISAP_ISAP_Team	CESCA	STM32F407		100	100-1000	8	EM	TVLA
ISAP_ISAP-team	CCSL	STM32F303	LeCroy 610Zi				EM	CPA
Romulus_Adominicai	HSCP	STM32F303	WaveRunner 8404M	8	25	8	power	TVLA
Romulus_Adomnicai	CCSL	STM32F303			125	16	EM	TVLA
Romulus_Adomnicai	CCSL	STM32F303			125	16	EM	TVLA
Romulus_Adomnicai	CCSL	STM32F303			125	16	EM	DL-LA
Romulus_Adomnicai	CCSL	STM32F303			125	16	EM	DL-LA
Romulus_Adomnicai	CCSL	STM32F303			125	16	EM	CPA
Romulus_Adomnicai	CCSL	STM32F303			125	16	EM	TA

Table 10: Results of Side-channel Evaluation of Protected Software Implementations

Implementation	**Lab**	**Test**	**Num. of Traces** [$\times 10^6$]	**Thresh. Exc.**	**Notes**
Ascon_Graz_d1	CESCA	CPA	15	–	Second order CPA. No bytes revealed.
Ascon_Graz_d2	CCSL	TVLA	0.06	N	
Ascon_Graz_d2	CCSL	χ^2-test	0.06	N	
Ascon_Graz_d2	CCSL	CPA	0.06	–	Key not revealed
GIFT_COFB_Adomnicai	CCSL	TVLA	0.02	N	Classification based on a nonce bit. Another test was done with classification based on an intermediate bit.
GIFT_COFB_Adomnicai	CCSL	χ^2-test	0.02	N	Classification based on a nonce bit. Another test was done with classification based on an intermediate bit.
GIFT_COFB_Adomnicai	CCSL	CPA	0.02	–	Key not revealed
GIFT-COFB_Adominicai	**HSCP**	**TVLA**	**0.1**	**Y**	Threshold exceeded. Report mentions possible causes.
ISAP_ISAP_Team	CESCA	TVLA	0.1	N	Fixed key vs random key test
ISAP_ISAP-team	CCSL	CPA		–	Key not revealed
Romulus_Adominicai	**HSCP**	**TVLA**	**0.1**	**Y**	Threshold exceeded. Report mentions possible causes.
Romulus_Adomnicai	CCSL	TVLA	1	N	Case A: No sample exceeded the threshold for 1 M traces. Classification based on a nonce bit.
Romulus_Adomnicai	CCSL	TVLA	1	N	Case B: No samples exceed the threshold for 1 M traces. Classification based on an intermediate bit.
Romulus_Adomnicai	CCSL	DL-LA		N	Case A: No sample exceed the threshold for 1 M traces. Classification based on a nonce bit.
Romulus_Adomnicai	CCSL	DL-LA		–	Case B: No samples exceed the threshold for 1 M traces. Classification based on an intermediate bit.
Romulus_Adomnicai	CCSL	CPA		–	Key not revealed
Romulus_Adomnicai	CCSL	TA		–	Key not revealed

Table 8 highlights the results of security evaluation experiments. In most leakage assessment tests, the predefined threshold was not crossed for the given test parameters. In some cases, the threshold was exceeded. In such cases, we report the number of traces at which the threshold was exceeded if mentioned in the report. Below, we provide notes on the experiments that exceeded the pre-defined threshold. It is noteworthy that having a leakage assessment test exceeding the threshold does not necessarily indicate an exploitable leakage, and there is a possibility of false positives.

- The tests on Ascon_Bochum_d1, Elephant_Bochum_d1, and Xoodyak_Bochum_d1 by the CERG lab: In all of these cases, the TVLA 4.5 threshold is exceeded at a few (3-10) samples. These tests use a sampling clock that is synchronized with the target clock, which results in more precise measurements. For all of these tests, t-values do not exceed the threshold until more than one million traces have been considered.
- The TVLA and χ^2-tests on GIFT_COFB_Bochum_d1, ISAP_Bochum_d1, and Romulus_Bochum_d1 by the CCSL lab: In these tests, the threshold has been exceeded. These test results were published in August 2022. Consequently, these implementations were updated in November 2022 when a bug related to providing randomness to masked gadgets was fixed.

The implementations from Bochum were generated using the AGEMA tool, which is used to convert the datapath to a masked design. The control logic, however, needs manual modification to provide the needed randomness to the masked gadgets in the proper cycles. In the first round of evaluations, leakage in the Bochum submissions was attributed to the fact that random data was not fed correctly to the masked gadgets. In other words, required fresh randomness was not provided in some clock cycles. The submitter corrected this issue by minor changes in the control logic that had a negligible effect on area and throughput. This negligible effect was confirmed for the cases of the TinyJAMBU_Bochum_d1 and Ascon_Bochum_d1 implementations, analyzed by CERG.

None of the reported CPA or template attacks attempted on the protected designs resulted in the reliable recovery of any key fragment. CPA attacks, as expected, attempted key recovery at the initialization phase when the nonce and the key are used to initialize the state and before intermediate values become a function of too many secret key bits.

All the tested hardware implementations use first-order masking except ISAP_Graz, which depends on mode-level protection. For mode-level resistance attack-based evaluation is more meaningful than leakage assessment which can show unexploitable leakage. As shown in Table 8, the CPA attack on the ISAP_Graz did not reveal the key.

We conclude that although some implementations show some leakage in the leakage assessment tests, these leakages are most likely fixable without significant changes in cost and performance. In many tests, we observed that leakage is significantly reduced by minor fixes in the control logic responsible for feeding randomness to the masked gadgets with no change to the datapath, which uses the majority of resources.

6.2 Software Implementations Result Summary

Tables 9 and 10 summarize the parameters and the results of the experiments performed on the protected software implementations. Similar to the case of hardware implementations, most of the experiments are leakage assessment tests in the form of TVLA, χ^2-test, and DL-LA. Correlation Power Analysis and template attacks (TA) were also attempted. All experiments used ARM Cortex-M4 as a target, and side-channel information was measured using shunt resistors and electromagnetic emanation (EM). None of the experiments reported the usage of sampling clocks that were synchronized to the target clock.

None of the reported CPA or template attacks attempted on the protected software designs resulted in the recovery of any part of the key. A second-order CPA attack by the CESCA lab could not reveal the key for Ascon_Graz_d1 using 15 million traces. In comparison, their CPA attack on the unprotected Ascon implementation can reliably reveal the key using 500 thousand traces.

The security analysis on ISAP by the CESCA lab concluded that DPA attacks were not an option except for the tag generation operation.

In most leakage assessment tests, the predefined threshold has not been crossed for the given test parameters. In some cases, the threshold has been exceeded. Below, we provide notes on the experiments that exceeded the predefined threshold.

- GIFT-COFB_Adominicai TVLA test by the HSCP lab: TVLA threshold has been exceeded in this test, and the submitted report attributes this to the unmasking of the state when performing encryption and decryption. The report shows no leakage in the key scheduling part of the algorithm.
- Romulus_Adominicai TVLA test by the HSCP lab: TVLA threshold has been exceeded. The evaluation report points out that this leakage is partially due to the associated data not being masked.

As noted in the reports by the HSCP team, these implementations could be leveled implementations where some part of the algorithm is protected against DPA while other parts are protected against SPA. In this case, direct application of leakage assessment tests will show a leakage.

In conclusion, in the outcomes of the tests on the protected software implementations, no leakage has been detected except in the two cases listed above. For these cases, further analysis is needed to see if the leakage is exploitable.

6.3 Target and Sampling Clock Synchronization

The effect of synchronizing the sampling and the target clocks on the number of traces needed for key recovery has been observed and discussed in the literature [22]. In this section, we show that performing sampling for leakage assessment using a synchronized clock is significantly more effective in leakage detection. In other words, using synchronized sampling and target clocks, one can detect leakage using significantly fewer traces than in setups using asynchronous clocks.

In the following tests, we use a masked implementation of the NIST LWC finalist Xoodyak_Bochum_d1. While the datapath of the design is masked, the control logic has an issue with providing random bits at some clock cycles, causing leakage.

We performed TVLA tests on the masked implementation of Xoodyak using an external oscilloscope at a sampling rate of 1 GS/s and 125 MS/s using 8-bit and 15-bit resolution, respectively. We repeated the same experiment, but this time, we used FOBOS 3 to capture traces at 50 MS/s with 10-bit resolution. In the FOBOS 3 case, the ADC clock is synchronized with the target clock, while the external oscilloscope sampling clock is not. Table 11 shows the details of each experiment and the corresponding results. In all cases, we used exactly the same Xoodyak implementation, which was instantiated in the NewAE C305 board and ran at 10 MHz, and we used the same fixed-vs-random test vectors.

Figure 5 shows the maximum t value in each experiment as a function of the number of traces processed. The red line marks the 4.5 threshold with t values exceeding this threshold, indicating leakage detection. The figure shows that in test C, which uses the synchronous clock, the t values exceed the threshold after processing 1.3 million traces. For experiment A, the test detects leakage after processing 1.6 million traces, while in experiment B, the leakage is detected after 8.7 million traces are processed.

When comparing experiment C, which uses synchronous clocks v.s, experiment B, we observe that although both sampling rate and resolution are higher in experiment B, the experiment with synchronized clocks detects the leakage using significantly fewer traces.

Table 11: TVLA results for the masked Xoodyak implementation depending on the measurement setup; all run for 10 Million traces

Oscilloscope	DUT	Resolution	Sync.	Sample Rate	DUT Freq.	Fails at Traces	Label
PicoScope 5244D	CW305	8 bit	No	1 GS/s	10 MHz	1.6 M	A
PicoScope 5244D	CW305	15 bit	No	125 MS/s	10 MHz	8.7 M	B
FOBOS 3	CW305	10 bit	Yes	50 MS/s	10 MHz	1.3 M	C

6.4 Qualitative Evaluation

In addition to quantitative evaluations, such as those summarized in this report, theoretical analysis of leakage properties and countermeasures are of extreme importance. An insightful analysis of the side-channel security of NIST LWC finalists was carried out by Verhamme et al. in [23]. Additionally, the authors investigated "leveled" implementations for Ascon, ISAP, Romulus-T, and Romulus-N, where only parts of the implementation require

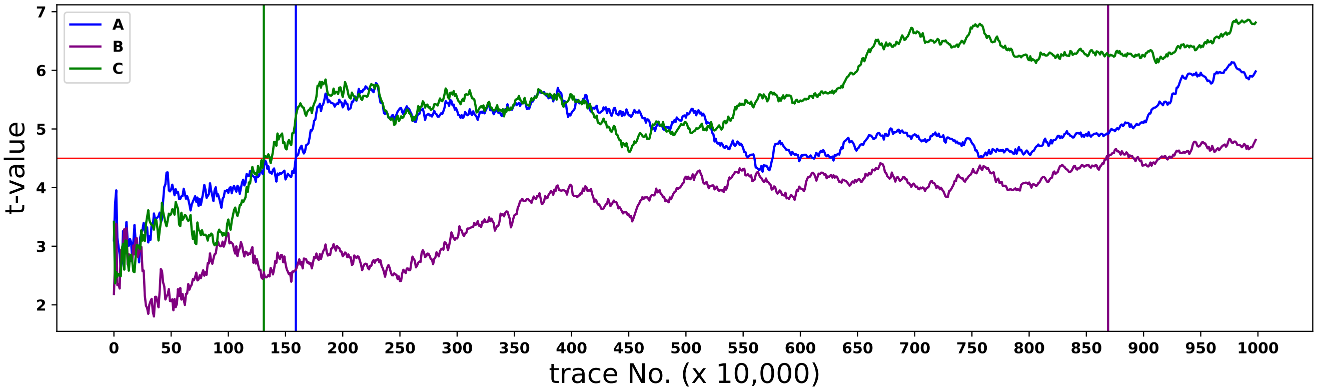

Figure 5: Maximum t value vs. the number of processed traces. Vertical lines indicate the point at which the t value exceeds the threshold in each experiment

masking (or higher order masking), reducing the performance and/or area overhead while maintaining the desired level of side-channel security. We believe that both qualitative and quantitative approaches have their merits and limitations and provide complementary information for evaluating cryptographic schemes and their implementations. Unfortunately, no implementations incorporating the "leveled" protection approach were available to us while conducting the evaluation, and therefore, they are not being represented in our results.

7 Benchmarking of Hardware Implementations

In this section, we compare the performance and area results for the protected hardware implementations. As a reference point, we also provide benchmarking results for the unprotected designs that were used as a *starting point for the protected designs* as well as the unprotected design with the *highest throughput over area ratio* for each algorithm. The results are generated for the Xilinx Artix-7 FPGA family. The target FPGA device is `XC7A100T-2FTG256L`, the main component of the NewAE CW305 board, which was used by many groups for the side-channel evaluation.

All evaluated designs are compatible with the GMU LWC API. The latency of major operations, expressed in clock cycles, is determined using simulation. The cycle count is determined for various lengths of plaintext, associated data, and hash input so that implementations can be compared for both short and long inputs. The area and maximum frequency were calculated using Xeda [24], a tool that automates simulation and synthesis for various FPGA and ASIC toolchains. Xeda can search for the maximum frequency for a specific target by sweeping through target frequency (through a heuristic variant of binary search) as well as different synthesis/implementation options and strategies (through evolutionary optimization). The maximum frequency is combined with the latency and input size to calculate the throughput of each design.

7.1 Protected vs. Unprotected Hardware Designs

A typical dependence between the throughput vs. area characteristics of unprotected and protected designs of various orders is shown in Fig. 6. Three protected implementations of orders 1, 2, and 3, respectively, are generated with the help of AGEMA. They are all based on a single unprotected implementation, Elephant_GMU. All protected designs operate with a very similar plaintext throughput. In the case of Elephant, this throughput is about 5 times smaller than in the case of an unprotected design. Additionally, the SCA countermeasures introduce area overhead (for area expressed in LUTs), which is dependent on the protection order. For Elephant, this overhead is about 3.6 for order 1, 7.2 for order 2, and 12.9 for order 3. Thus, the area of protected designs is almost exactly proportional to the protection order.

The same designs offer similar dependencies when used for processing ADs, as shown in Fig. 7. The only major difference is a significantly higher throughput for processing of AD vs. plaintext. The areas of all designs are exactly the same as in Fig. 7, as each design is capable of processing both plaintext and AD.

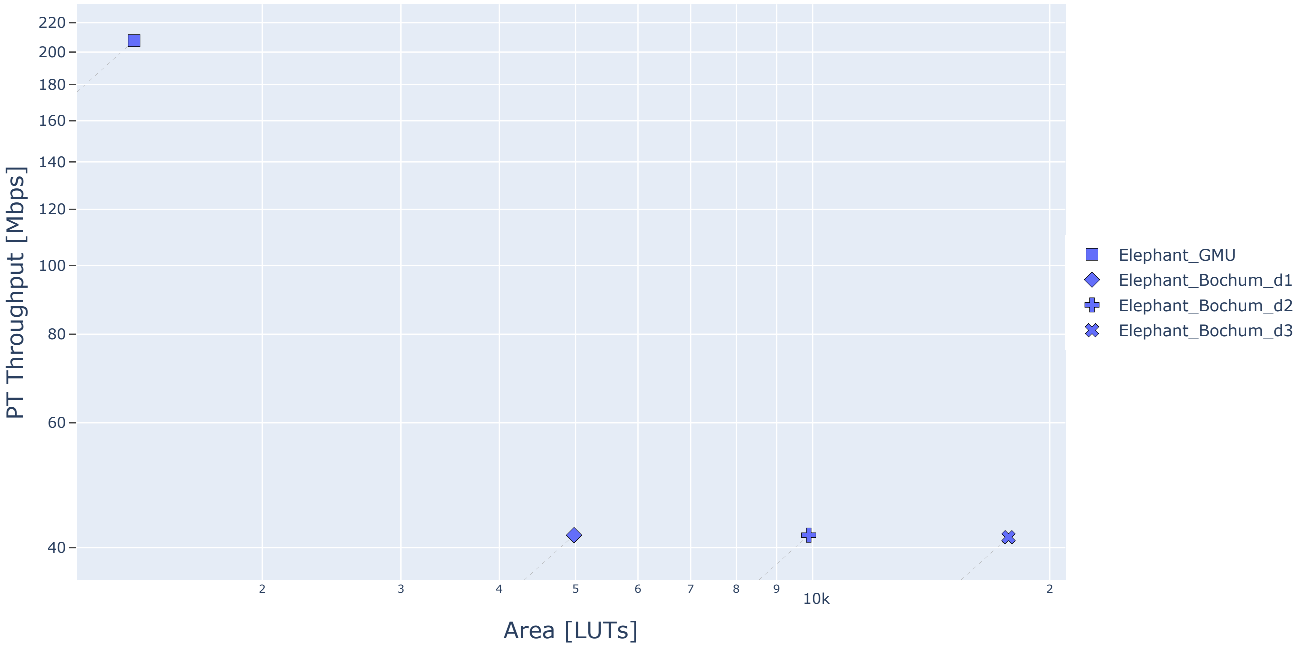

Figure 6: Elephant: Plaintext Throughput vs. Area for Unprotected and Protected Designs

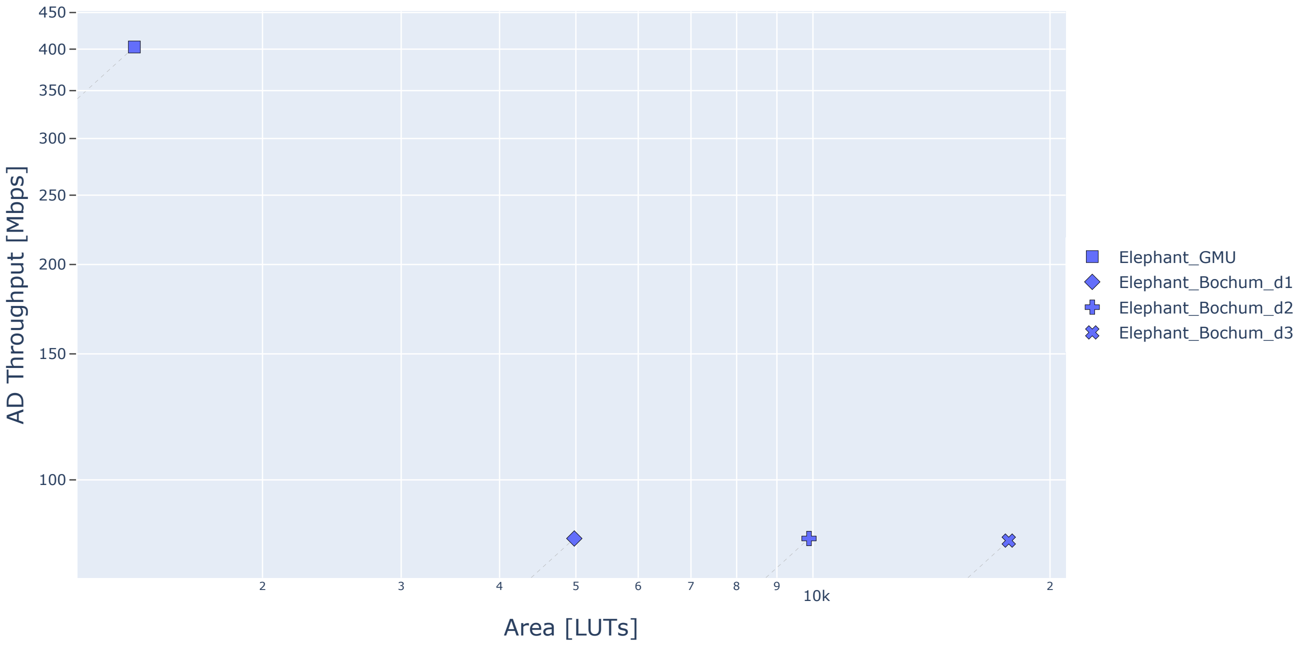

Figure 7: Elephant: AD Throughput vs. Area for Unprotected and Protected Designs

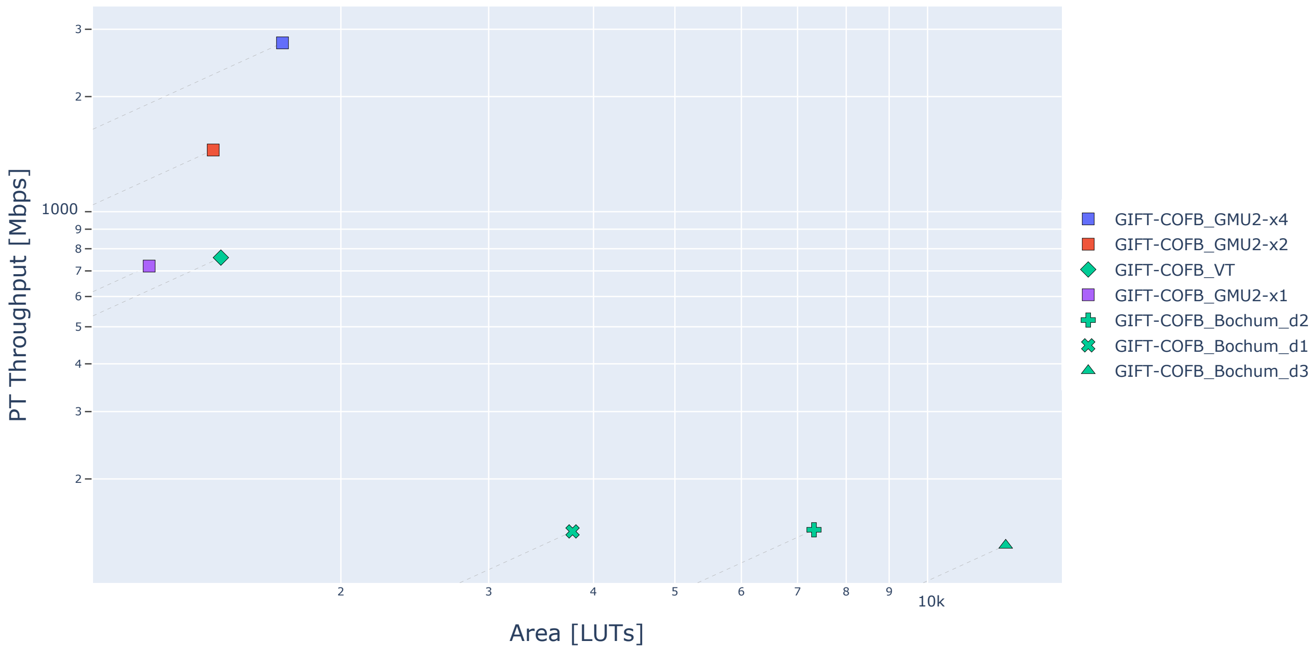

Figure 8: GIFT-COFB: Plaintext Throughput vs. Area for Unprotected and Protected Designs

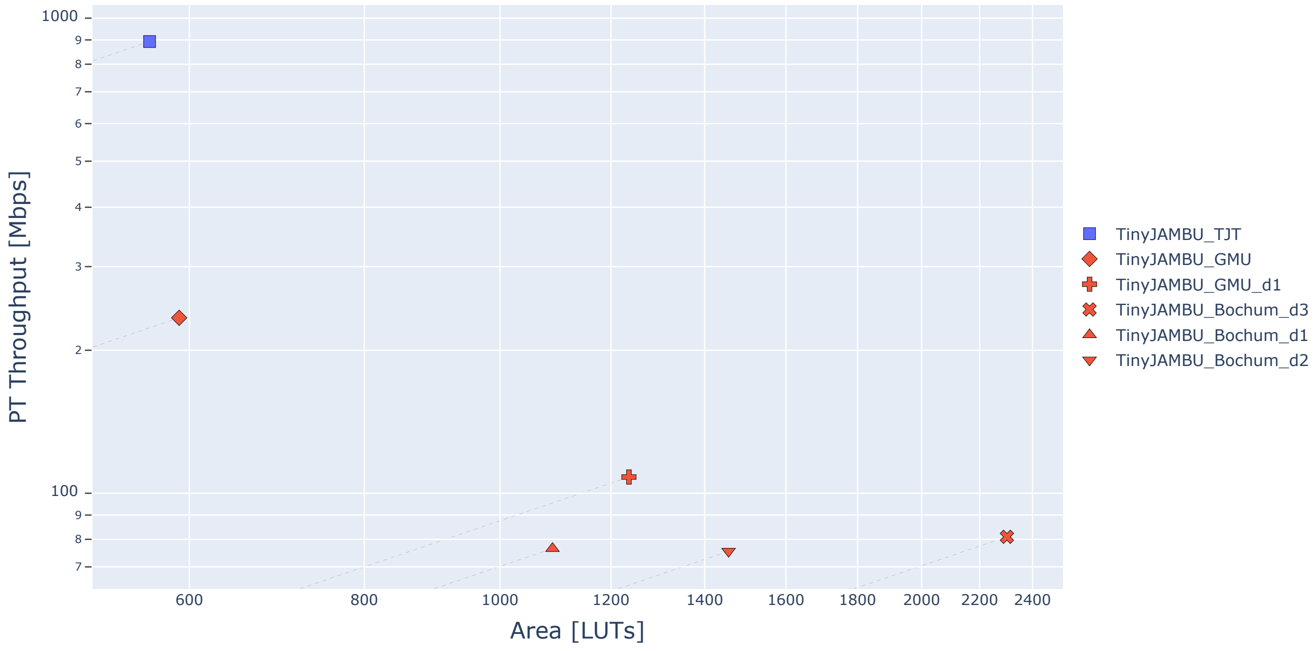

Figure 9: TinyJAMBU: Plaintext Throughput vs. Area for Unprotected and Protected Designs

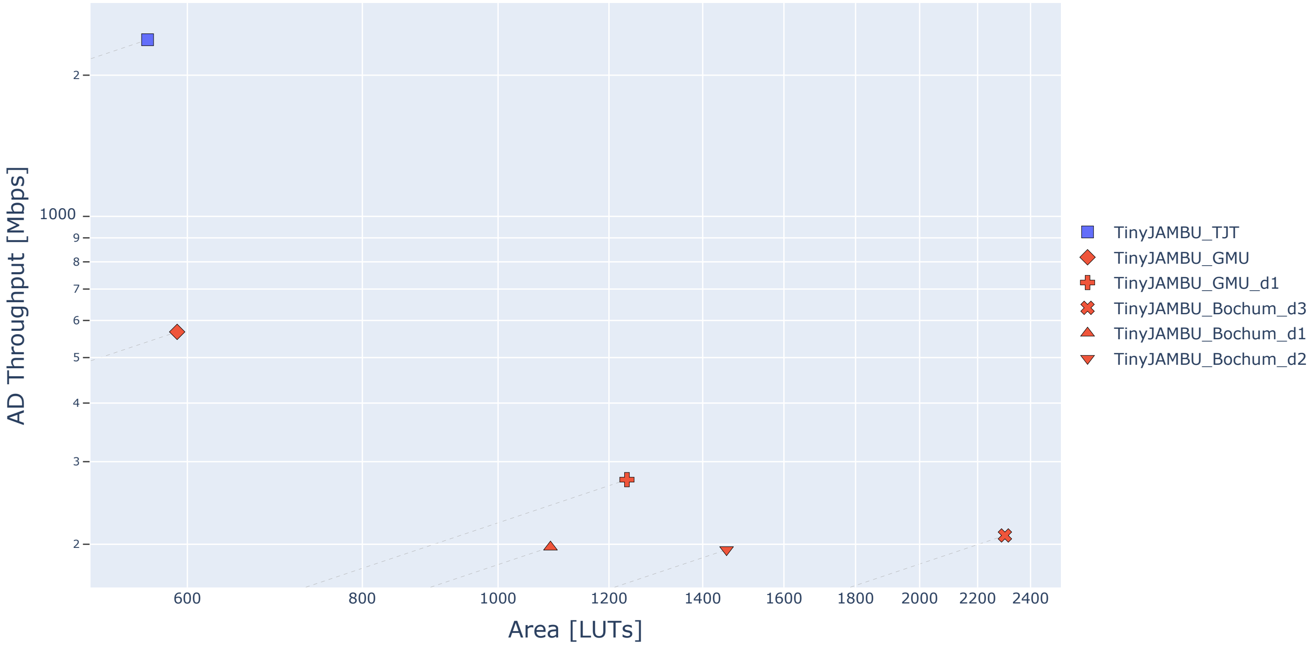

Figure 10: TinyJAMBU: AD Throughput vs. Area for Unprotected and Protected Designs

The case of GIFT-COFB is illustrated in Fig. 8. The primary difference as compared to Elephant is that there are now four unprotected implementations, GIFT-COFB_VT[13] from Virginia Tech, and GIFT-COFB_GMU2-x{1,2,4}[14] from GMU, where GIFT-COFB_GMU2-x2 and GIFT-COFB_GMU2-x4 are respectively 2x and 4x unrolled versions of GIFT-COFB_GMU2-x1. One interesting observation is that the maximum frequency of the 2x unrolled design GIFT-COFB_GMU2-x2 is not lower than GIFT-COFB_GMU2-x1, and unrolling results in substantially improved performance while incurring only a moderate overhead in the area. The protected designs are based on the GIFT-COFB_VT, which is neither the fastest nor the smallest design. The primary reason for choosing a sub-optimal design as a starting point for protected designs was most likely that GIFT-COFB_VT was written in VHDL, while GIFT-COFB_GMU2-x{1,2,4} designs are modeled in Bluespec SystemVerilog. The choice of the underlying design was made by the Bochum group. Consequently, it is fair to compare only the overheads of designs derived from GIFT-COFB_VT while keeping in mind that a more optimal starting point might have led to more efficient protected designs as well. The overhead in terms of throughput varies from 5.15 for order 2 to 5.65 for order 3. The small differences in throughputs of protected implementations are the result of the different clock frequencies, while the number of clock cycles remains the same. The overheads in terms of area are 2.6, 5.1, and 8.6, respectively. Similarly to the case of Elephant, the area of the protected implementation of order 2 is approximately twice as large as compared to the protected implementation of order 1. Protection order 3 leads to the increase in area by a factor larger than 3 as compared to the implementation of order 1.

The case of TinyJAMBU is somewhat similar to the case of GIFT-COFB. The protected designs are based on the less efficient of the two unprotected implementations, TinyJAMBU_GMU. The primary difference compared to the case of GIFT-COFB is the existence of the manually developed 1st-order protected implementation, TinyJAMBU_GMU_d1. As expected, the 1st-order manually protected design, TinyJAMBU_GMU_d1, is faster than the automatically generated design of the same order, TinyJAMBU_Bochum_d1. The throughput ratio is about 1.40. However, contrary to expectations, the manually developed design has a larger area. The area ratio is about 1.13. Thus, overall, the manual implementation is still more efficient in terms of the throughput-to-area ratio.

The case of Xoodyak, shown in Fig. 11 is exceptional, as this candidate has three manually developed protected implementations of order 1:

1. Xoodyak_Tsinghua_d1DOM – developed by the group from Tsinghua University using the Domain Oriented Masking (DOM) method

[13] https://github.com/vtsal/gift_cofb_lwc_v2
[14] https://github.com/kammoh/bluelight

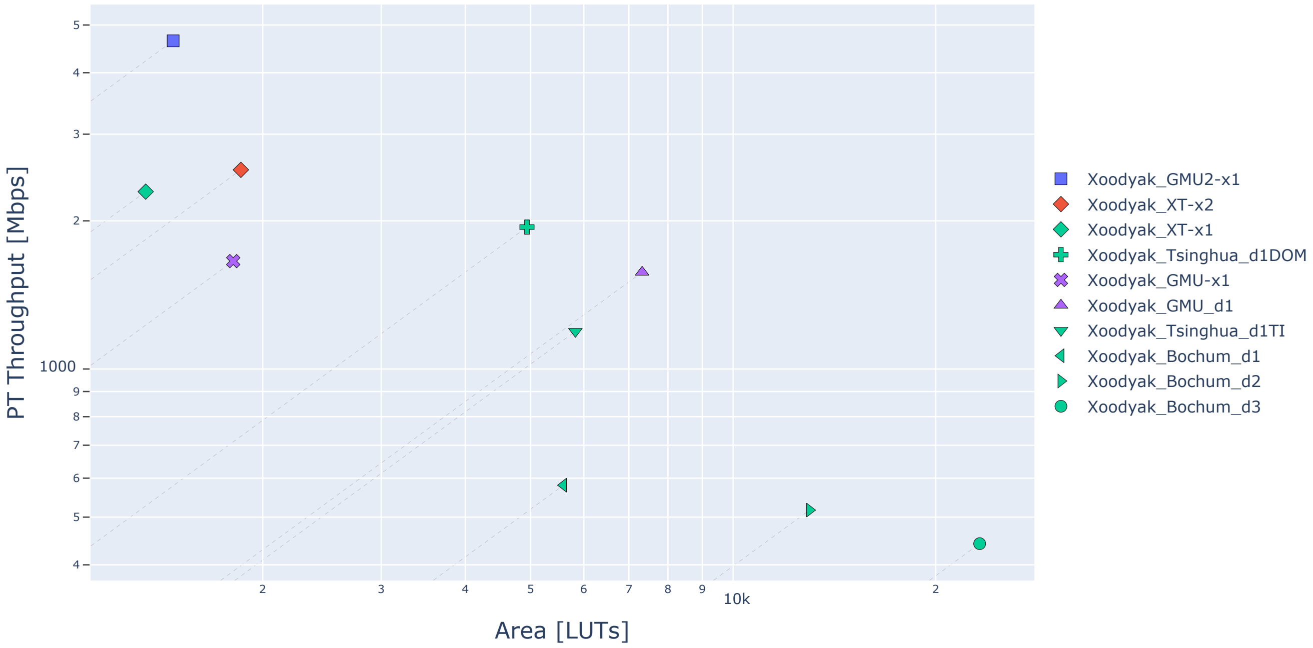

Figure 11: Xoodyak: Plaintext Throughput vs. Area for Unprotected and Protected Designs

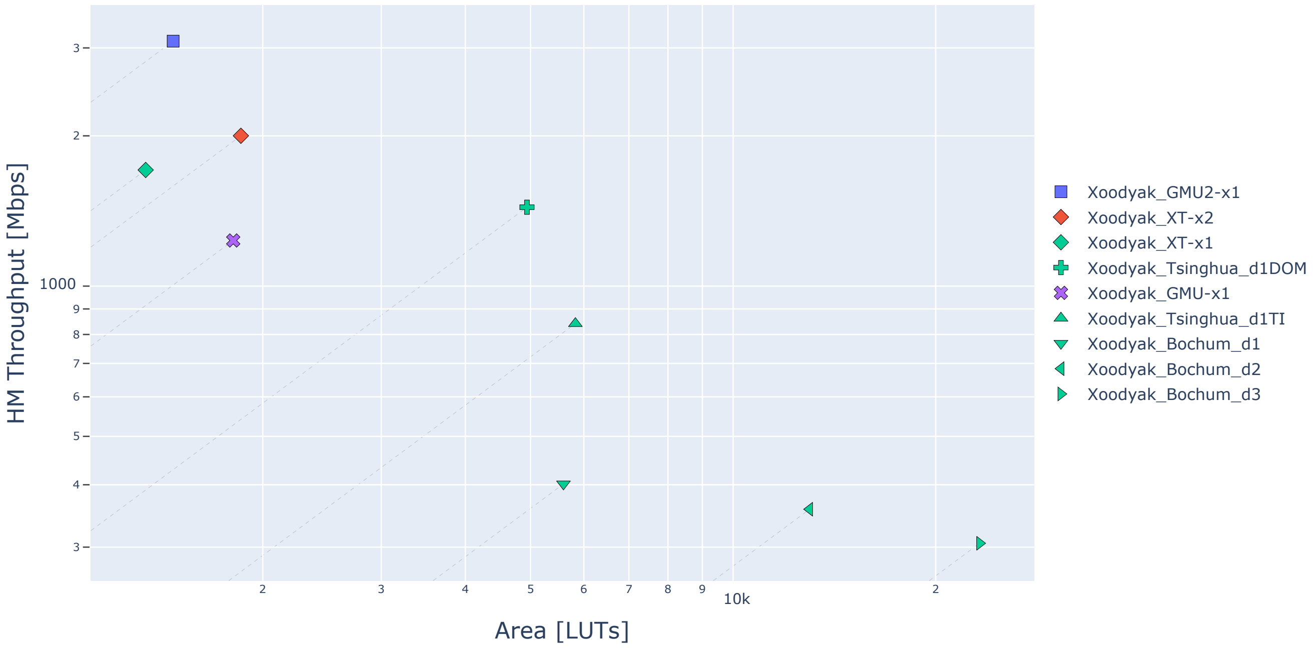

Figure 12: Xoodyak: Hashing Throughput vs. Area for Unprotected and Protected Designs

2. Xoodyak_Tsinghua_d1TI – developed by the group from Tsinghua University using the Threshold Implementation (TI) method, and

3. Xoodyak_GMU_d1 – developed by the group from George Mason University using the Domain Oriented Masking method.

All of these implementations offer only first-order protection. Additionally, there exist semi-automatically generated implementations of orders 1, 2, and 3, respectively, developed by the group from Ruhr University Bochum. The protected designs developed by Tsinghua University and Ruhr University Bochum used as a starting point the unprotected implementation developed by the Xoodyak Team, Xoodyak_XT-x1, based on the basic iterative architecture (a.k.a. the architecture with the unrolling factor x1). The protected design developed by George Mason University, used as a starting point a folded design, Xoodyak_GMU-x1, developed by the same team.

Apart from the mentioned above implementations, Fig. 11 also shows results for two fastest unprotected implementations: Xoodyak_GMU2-x1 developed using Bluespec SystemVerilog, and Xoodyak_XT-x2 – a two times unrolled variant of Xoodyak_XT-x1.

Both manually developed protected implementations, generated using DOM, produced a relatively small throughput overhead. The slowdown for the design from Tsinghua University, Xoodyak_Tsinghua_d1DOM was about 15% as compared to the corresponding unprotected design. The slowdown for the design from George Mason University, Xoodyak_GMU_d1, was only about 5%. The threshold implementation from Tsinghua University, Xoodyak_Tsinghua_d1TI, was both slower and larger than the corresponding DOM implementation, Xoodyak_Tsinghua_d1DOM.

Overall, Xoodyak_Tsinghua_d1DOM is the best-protected implementation of Xoodyak, both in terms of speed and area. It outperforms the semi-automatically created design from Bochum, Xoodyak_Bochum_d1, by a factor of about 3.3 in terms of throughput. It is also about 12% smaller in terms of the number of LUTs. The first-order protected implementations use between 3.7 and 4.35 more LUTs than the corresponding unprotected designs. The second-order implementation has an area overhead of about 9.7, and the third order 17.3.

The performance for hashing, illustrated in Fig. 12, is almost the same. The primary differences include smaller absolute values of throughput and no support for hashing in Xoodyak_GMU_d1. On the other hand, all areas are identical, as the designs supporting hashing use the same circuits for processing Plaintext, AD, and Hash Messages.

The case of Ascon, illustrated in Figs. 13 and 14 is, no doubt, the most complicated of all candidates. First, Ascon has two variants for authenticated encryption – Ascon-128 and Ascon-128a. They differ in terms of the data block size (64 bits for Ascon-128 and 128 bits for Ascon-128a). They also have a different number of rounds in the permutation used to process AD, plaintext, and ciphertext. This number of rounds is 6 for Ascon-128 and 8 for Ascon-128a. Ascon also has two different variants of a hash function: Ascon-Hash and Ascon-Hasha. They both use the same message block size of 64 bits. They differ in terms of the number of permutation rounds in the Absorb Message and Squeeze Hash phases. Ascon-Hash has 12 rounds, and Ascon-Hasha 8 rounds. In hardware, where the entire block is typically processed in parallel, and the rounds are executed sequentially, Ascon-128a and Ascon-Hasha are typically faster. Ascon-128 and Ascon-Hash are more conservative designs and are the primary recommendations of Ascon's authors.

Ascon has the following protected implementations:

1. Ascon-128_Graz_d1 and Ascon-128_Graz_d2 – two implementations of Ascon-128, of order 1 and 2, respectively, developed by the Ascon Team manually, using the Domain Oriented Masking (DOM) method. The starting point was the unprotected implementation from the same team – Ascon-128_Graz-x1.

2. Ascon-128_Bochum_d{1,2,3} – implementations of Ascon-128 of orders 1, 2, and 3, generated semi-automatically with the help of AGEMA, using Ascon-128_Graz-x1 as an underlying unprotected implementation, and

3. Ascon-128a_Bochum_d{1,2,3} – implementations of Ascon-128a of orders 1, 2, and 3, generated semi-automatically with the help of AGEMA, using Ascon-128a_Graz-x1 as an underlying unprotected implementation.

Each unprotected implementation from Graz University has a corresponding two-times unrolled implementation from the same group, with the name ending with x2. The two-times unrolled architectures are faster but bigger than the basic architectures. Because of the increased area, they were not used as a basis for any protected implementations.

The two fastest unprotected designs were developed by the GMU group. Ascon-128a_GMU[15] implements Ascon-128a. Ascon-128_GMU2-x1[16] and Ascon-128_GMU2-x2[16] implement Ascon-128 using the basic iterative and 2x unrolled architectures, respectively. All of them were modeled using Bluespec SystemVerilog.

Corresponding implementations of Ascon-128 and Ascon-128a developed by the Graz Team[17] (Ascon-128_Graz-x1 and Ascon-128a_Graz-x1) and generated semi-automatically by the Bochum Team (e.g., Ascon-128_Bochum_d1 and Ascon-128a_Bochum_d1) have the same areas. Within each pair, the implementation of Ascon-128a is approximately x1.3 faster than the implementation of Ascon-128 for unprotected designs and between 1.45 and 1.50 for protected designs.

The manually developed first-order protected implementation of Ascon-128 is 43% faster and 34% smaller than the corresponding semi-automatically generated design. The manually developed second-order protected implementation of Ascon-128 is about 9% faster and 25% smaller than the corresponding semi-automatically generated design.

The results for hashing, illustrated in Fig. 14 are similar. The primary differences are as follows:

- Unprotected GMU design, Ascon-128a_GMU is missing, as it does not support combining Ascon-128a with Ascon-Hasha
- Protected Bochum designs, Ascon-128a_Bochum_d{1,2,3} are missing, as they do not support combining Ascon-128a with Ascon-Hasha
- Manually developed protected implementations do not support hashing.

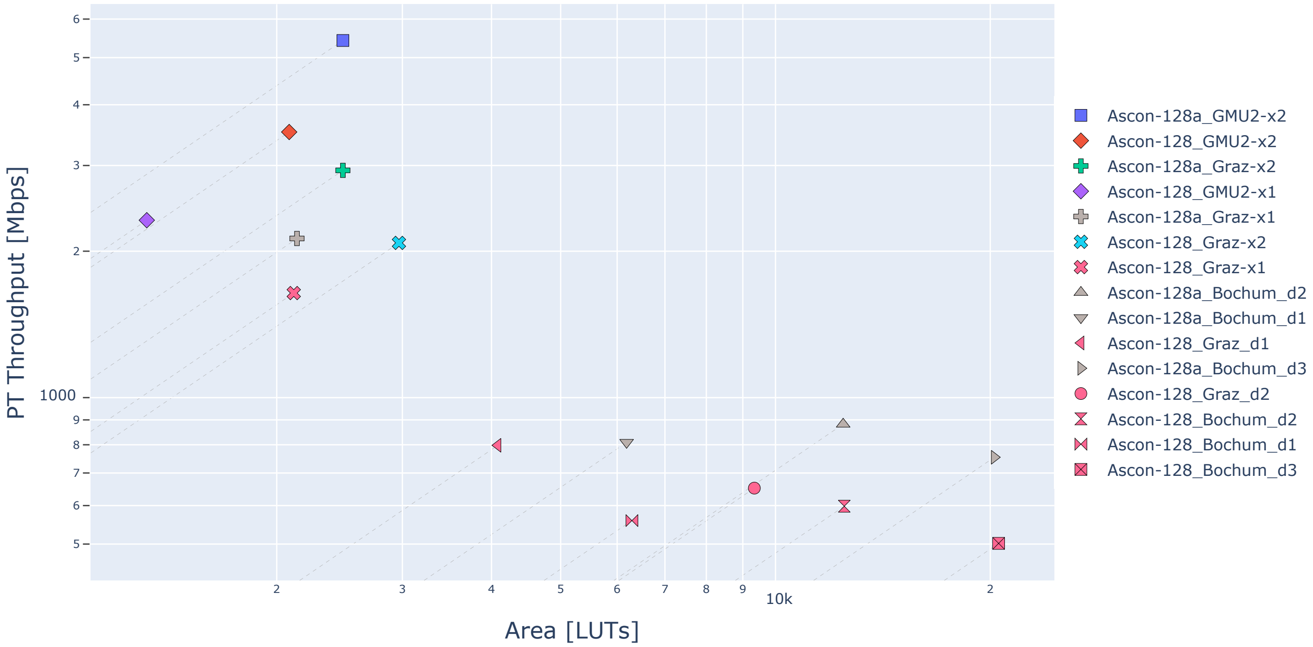

Figure 13: Ascon: Plaintext Throughput vs. Area for Unprotected and Protected Designs

In Fig. 15, we show the ratios between the numbers of LUTs (treated as units of area) for the available first-order protected and the corresponding unprotected implementations of 9 finalists (all except Grain-128AEAD). The candidates are ranked based on the smallest ratio among all protected implementations of a given candidate. The small ratio is desired as it indicates the smallest area overhead of adding masking to the unprotected implementation. The finalists with ratios below 2 include TinyJAMBU and Ascon. GIFT-COFB, Romulus, and ISAP have ratios between 2 and 3. Elephant, Xoodyak, and PHOTON-Beetle have ratios between 3 and 4. SPARKLE has the highest ratio, which exceeds a factor of 5.

[15] https://github.com/GMUCERG/Ascon

[16] https://github.com/kammoh/bluelight

[17] https://github.com/ascon/ascon-hardware

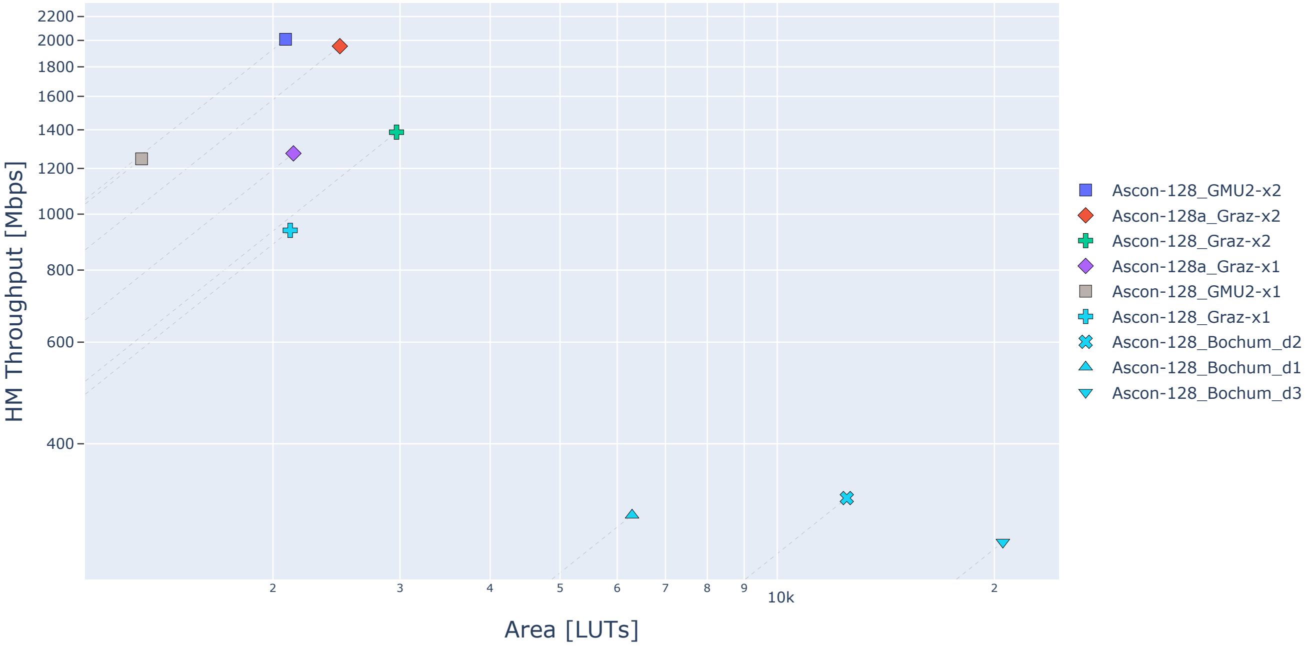

Figure 14: Ascon: Hashing Throughput vs. Area for Unprotected and Protected Designs

In Fig. 16, the same ratio is reported for the second-order protected implementations. The absolute values of ratios increase. The ranking of candidates remains mostly the same. TinyJAMBU has by far the smallest ratio. Ascon is second best. Xoodyak has the highest ratio that exceeds that of PHOTON-Beetle. The second-order protected implementation of SPARKLE is missing due to failing functional verification.

In Fig. 17, the same ratio is reported for the third-order protected implementations. The results for SPARKLE are missing due to the functional verification failure. The results for PHOTON-Beetle are not included due to the excessively long time required for synthesis, mapping, placing, and routing. The primary difference compared to the results for order 2 is that Romulus and GIFT-COFB have slightly smaller area overhead ratios than Ascon.

Fig. 18 illustrates the ratios of the throughputs of unprotected implementations to the throughputs of the corresponding protected designs. Small ratios are desirable as they indicate the small timing overhead of protected implementations. Xoodyak is the only candidate with a ratio close to 1. However, it should be stressed that this ratio is obtained for only one specific architecture and is accomplished at the cost of a substantial area overhead. Ascon and TinyJAMBU are the only candidates with ratios close to 2. The ratio is the highest for Romulus, for which it exceeds 8.

In Fig. 19, the same ratios are reported for the second-order protected implementations. Ascon, masked ISAP, and TinyJAMBU achieve timing overhead ratios smaller than 3. They are followed by Xoodyak, Elephant, and GIFT-COFB, with ratios between 4 and 5. The ratios for Romulus and PHOTON-Beetle exceed 8.

In Fig. 20, the same ratio is reported for the third-order protected implementations. The ratio for Ascon is the smallest, followed closely by the ratios for TinyJAMBU and the masked version of ISAP. All three mentioned above candidates have ratios close to 3. They are followed by Elephant, Xoodyak, and GIFT-COFB, with ratios between 5 and 6. The ratio for Romulus exceeds 9.

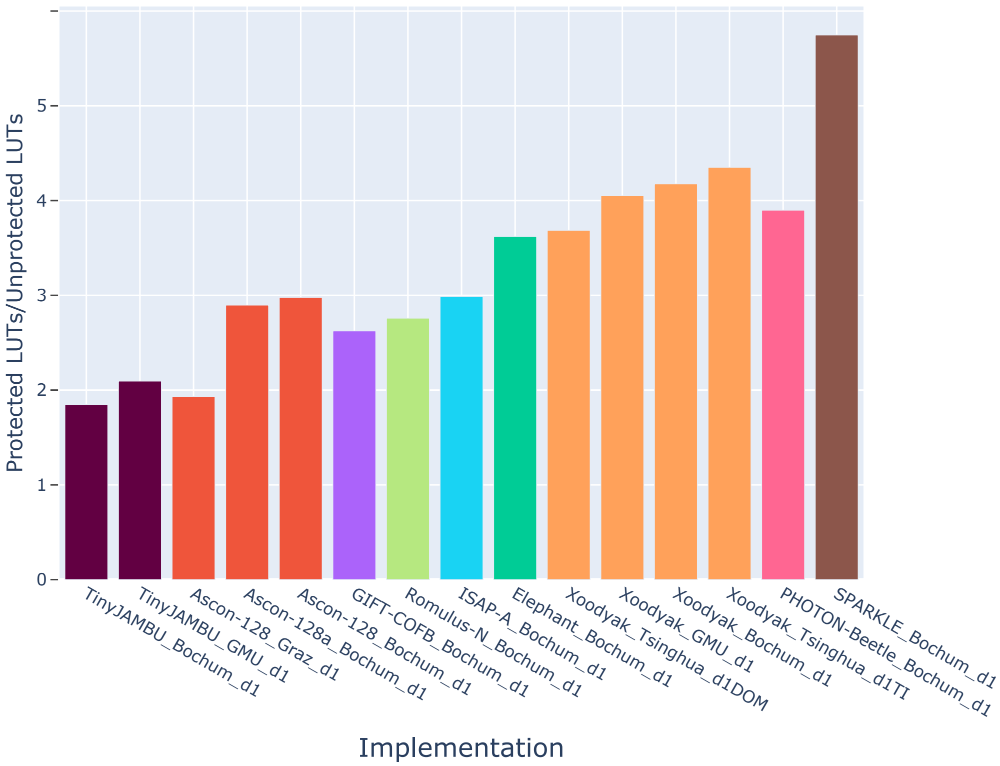

Figure 15: 1st order protected area over unprotected base area

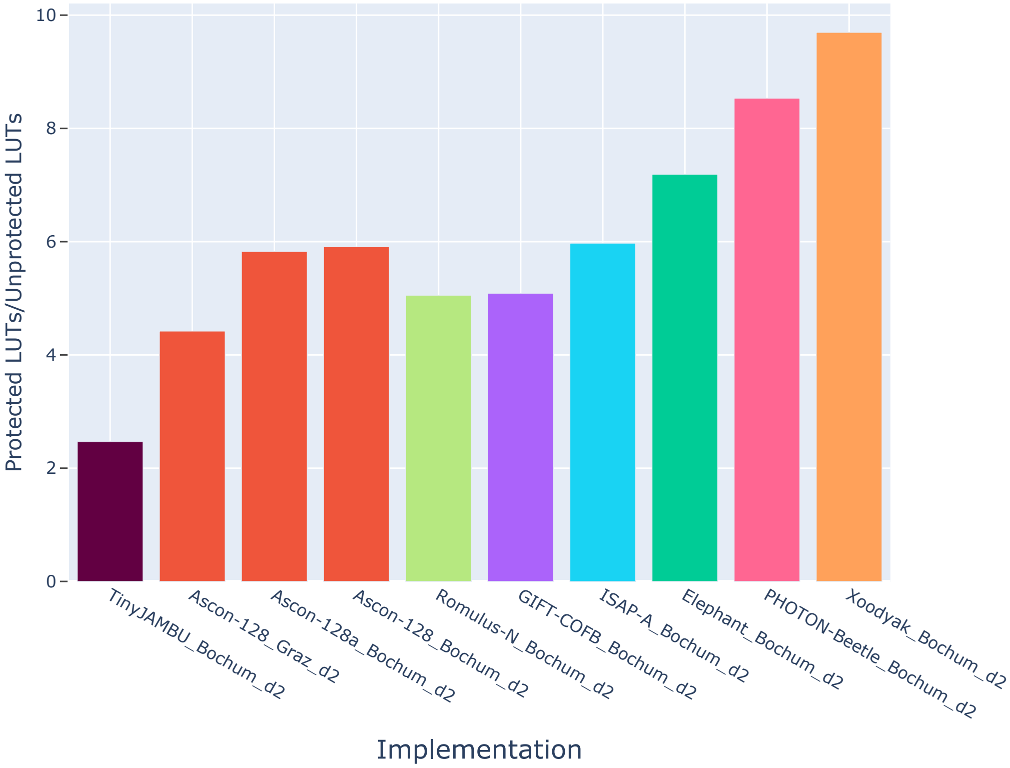

Figure 16: 2nd order protected area over unprotected base area

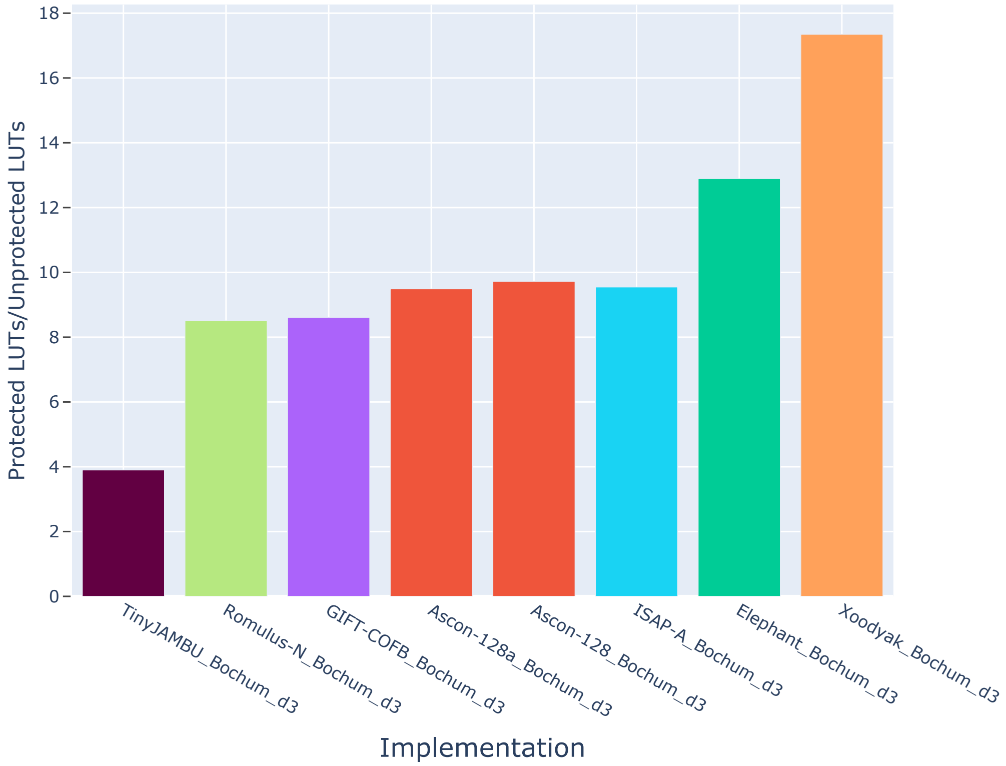

Figure 17: 3rd order protected area over unprotected base area

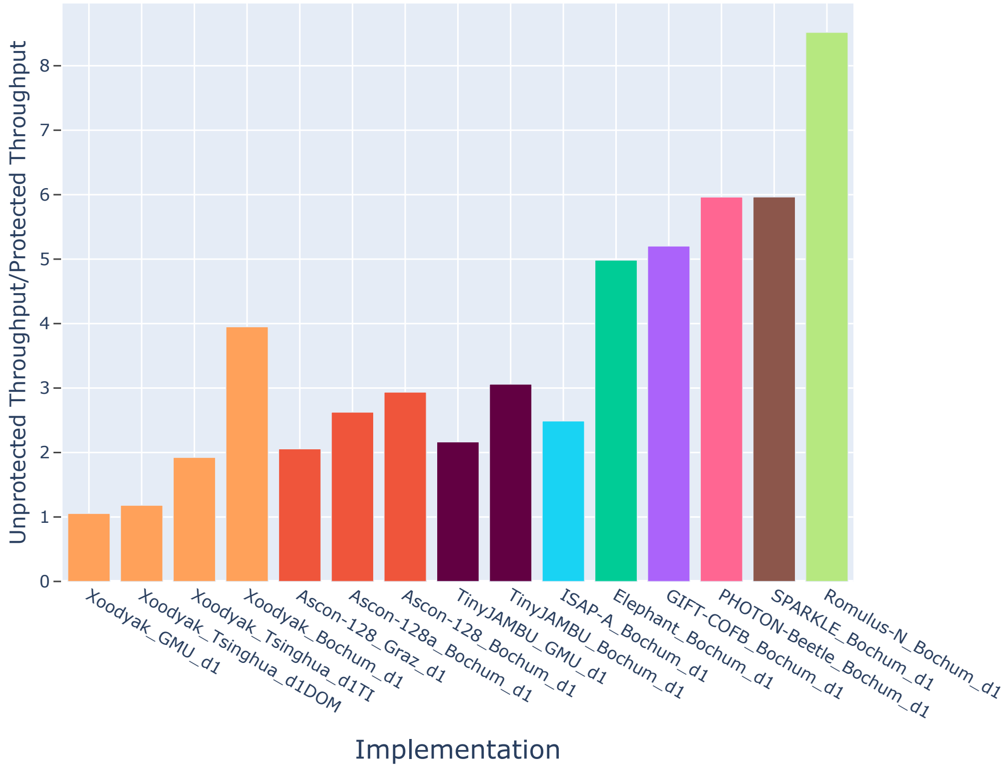

Figure 18: Long-message encryption throughput of unprotected base design over 1st order protected implementation

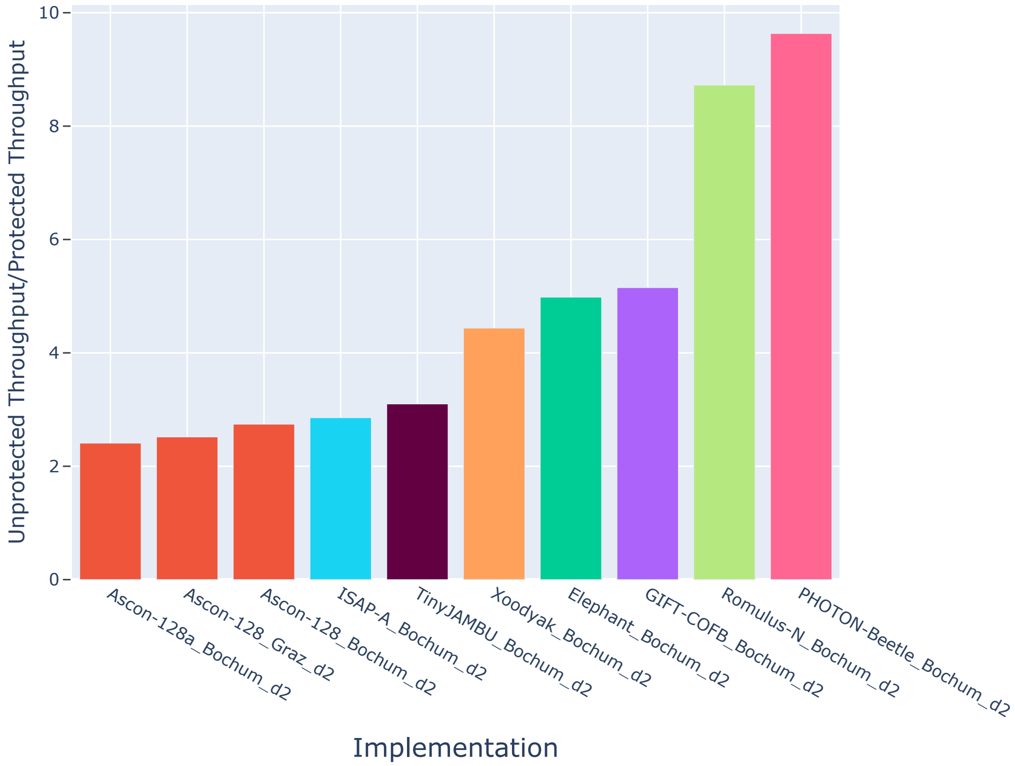

Figure 19: Long-message encryption throughput of unprotected base design over 2nd order protected implementation

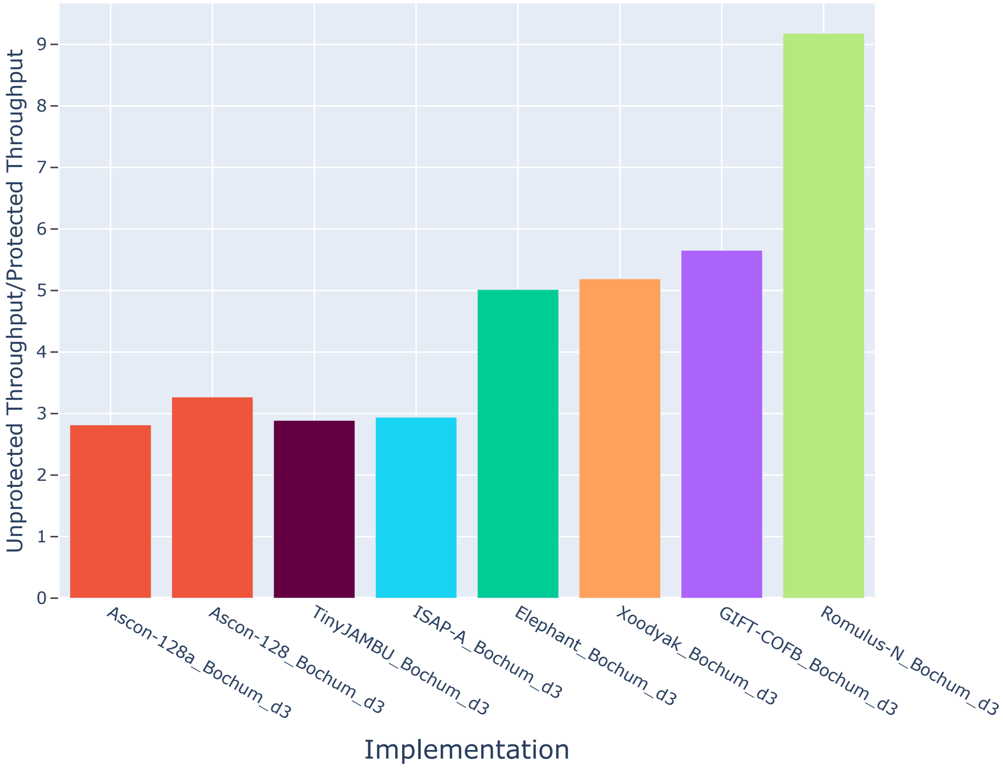

Figure 20: Long-message encryption throughput of unprotected base design over 3rd order protected implementation

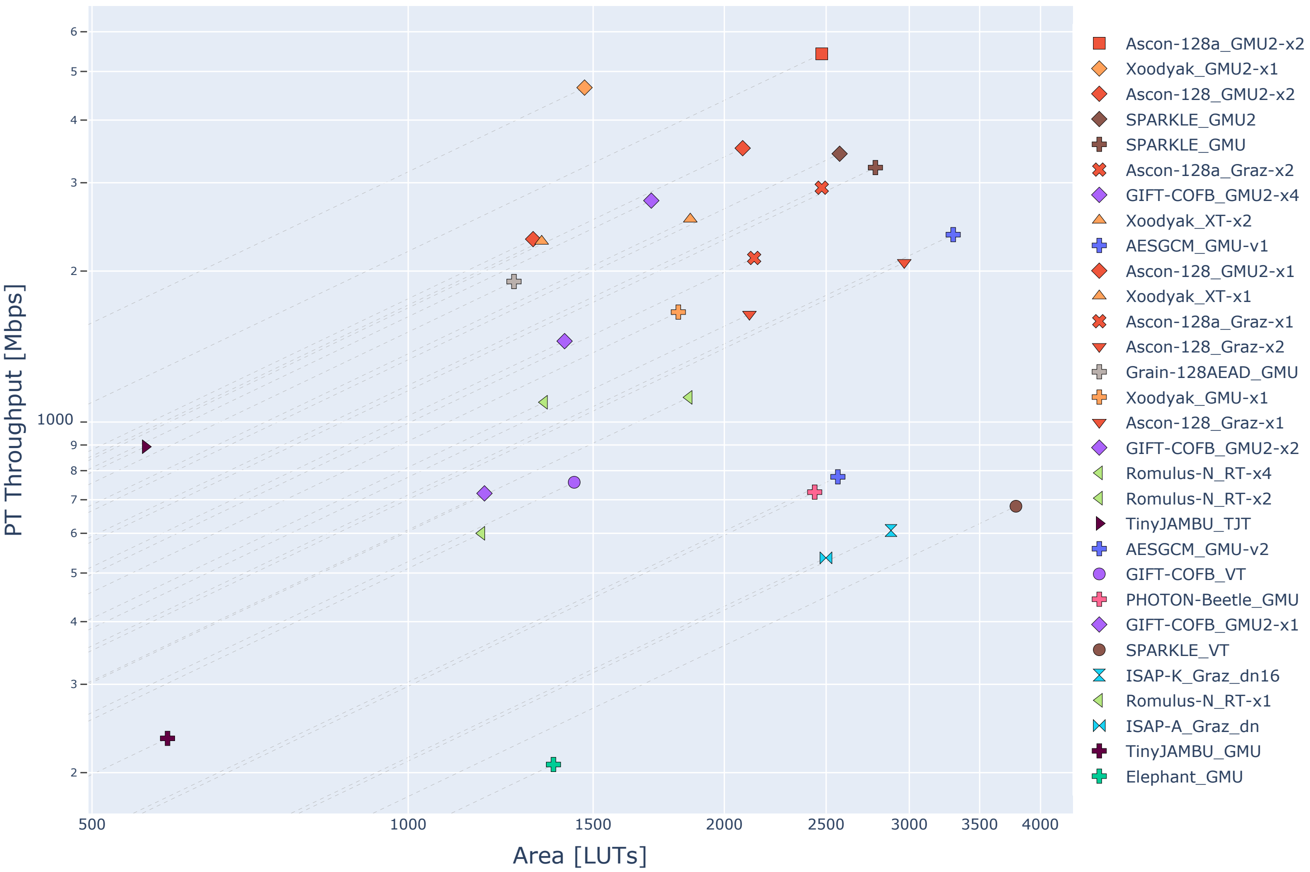

Figure 21: Encryption throughput vs LUTs for long messages (Unprotected)

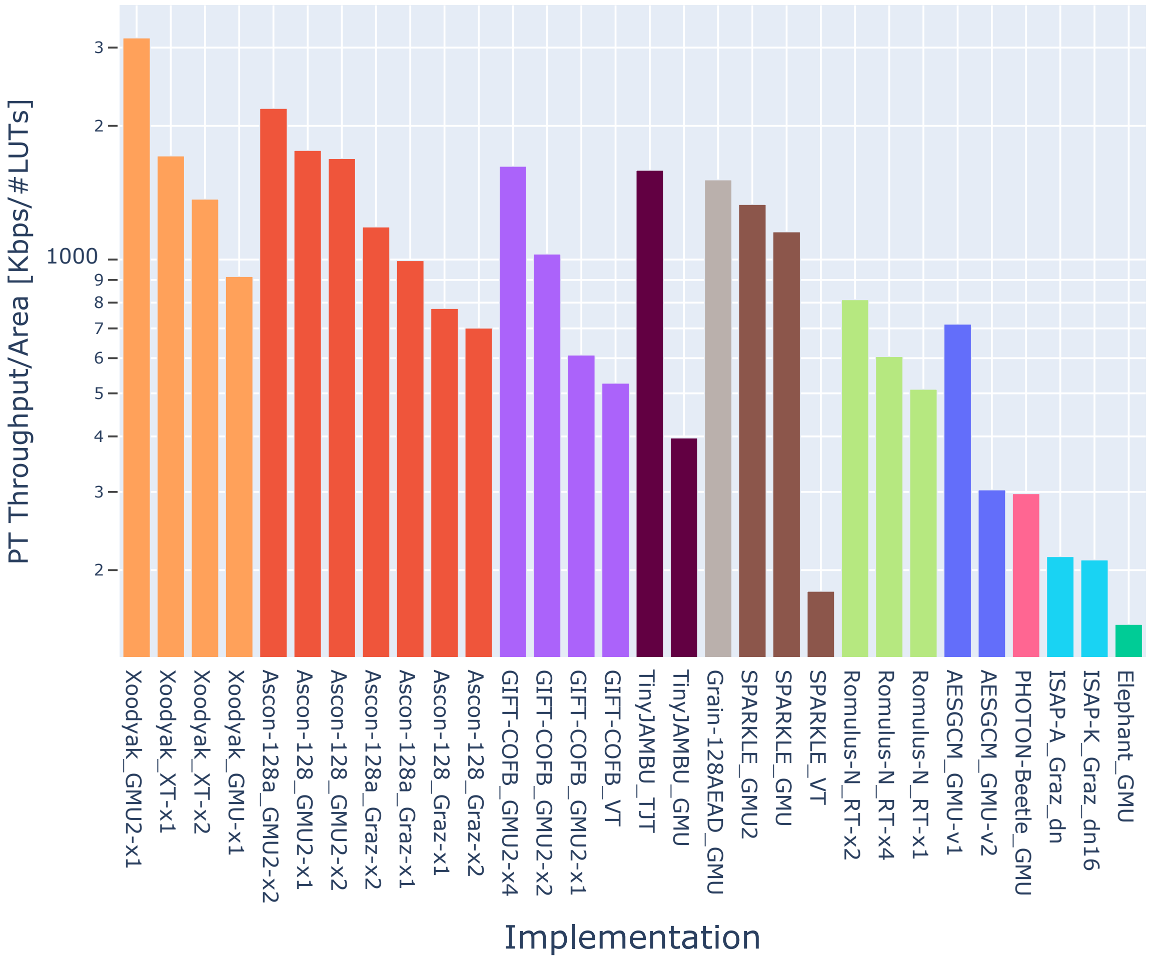

Figure 22: Encryption throughput over area for long messages (Unprotected)

Table 12: Unprotected implementations: Encryption throughput in Mbit/s for *Long*, *1536 Byte*, *64 Byte* and *16 Byte* messages, along with throughput ratios of different message types and sizes.

Implementation	$\mathbf{Thr^{PT}_{Long}}$	$\mathbf{Thr^{PT}_{1536}}$	$\mathbf{Thr^{PT}_{64}}$	$\mathbf{Thr^{PT}_{16}}$	$\mathbf{Thr\%^{PT}_{1536/Long}}$	$\mathbf{Thr\%^{PT}_{64/Long}}$	$\mathbf{Thr\%^{PT}_{16/Long}}$	$\mathbf{Thr\%^{AD/PT}_{Long}}$	$\mathbf{Thr^{AD}_{16}}$	$\mathbf{Thr^{AD+PT}_{16}}$
Ascon-128a_GMU2-x2	5421	5004	1807	602	92.3	33.3	11.1	1.0	602	1084
Xoodyak_GMU2-x1	4643	4385	1829	649	94.4	39.4	14.0	1.8	649	1298
Ascon-128_GMU2-x2	3516	3367	1705	670	95.8	48.5	19.0	1.0	639	1082
SPARKLE_GMU2	3427	3141	1075	319	91.6	31.4	9.3	1.0	319	499
SPARKLE_GMU	3216	2919	936	274	90.8	29.1	8.5	1.0	274	436
Ascon-128a_Graz-x2	2932	2790	1321	499	95.2	45.1	17.0	1.0	460	795
GIFT-COFB_GMU2-x4	2764	2647	1345	529	95.8	48.6	19.1	1.0	622	1058
Xoodyak_XT-x2	2539	2439	1240	489	96.1	48.8	19.3	1.3	500	917
AESGCM_GMU-v1	2364	2202	1048	403	93.1	44.3	17.1	1.2	508	833
Ascon-128_GMU2-x1	2316	2239	1271	540	96.7	54.9	23.3	1.0	499	821
Xoodyak_XT-x1	2293	2209	1150	461	96.4	50.2	20.1	1.5	468	880
Ascon-128a_Graz-x1	2124	2036	1040	411	95.8	49.0	19.4	1.0	364	622
Ascon-128_Graz-x2	2080	1989	994	387	95.6	47.8	18.6	1.0	370	628
Grain-128AEAD_GMU	1906	1789	739	261	93.8	38.8	13.7	1.0	250	442
Xoodyak_GMU-x1	1657	1582	749	284	95.5	45.2	17.1	1.5	284	545
Ascon-128_Graz-x1	1640	1588	913	392	96.8	55.7	23.9	1.0	360	590
GIFT-COFB_GMU2-x2	1450	1406	828	362	97.0	57.1	25.0	1.0	468	725
Romulus-N_RT-x4	1120	1091	689	320	97.5	61.5	28.6	1.6	320	640
Romulus-N_RT-x2	1095	1074	746	381	98.1	68.1	34.8	1.7	381	762
TinyJAMBU_TJT	893	880	661	371	98.6	74.0	41.6	2.7	511	650
AESGCM_GMU-v2	778	753	483	229	96.9	62.1	29.5	1.0	260	389
GIFT-COFB_VT	758	741	485	233	97.7	63.9	30.7	1.0	324	454
PHOTON-Beetle_GMU	725	714	532	295	98.5	73.3	40.7	1.2	307	431
GIFT-COFB_GMU2-x1	721	702	443	205	97.4	61.4	28.5	1.0	281	410
SPARKLE_VT	679	654	355	124	96.3	52.2	18.2	1.2	129	174
ISAP-K_Graz_dn16	608	543	168	54	89.3	27.6	8.9	1.7	87	106
Romulus-N_RT-x1	600	591	438	242	98.5	73.0	40.4	1.8	242	484
ISAP-A_Graz_dn	536	492	171	56	91.8	31.9	10.4	1.6	93	105
TinyJAMBU_GMU	234	230	186	117	98.3	79.7	50.0	2.4	169	194
Elephant_GMU	208	204	131	80	98.3	63.2	38.5	1.9	64	130
Minimum	**208**	**204**	**131**	**54**	**89.3**	**27.6**	**8.5**	**1.0**	**64**	**105**
Average	**1841**	**1747**	**849**	**341**	**95.7**	**52.0**	**23.2**	**1.3**	**360**	**606**
Maximum	**5421**	**5004**	**1829**	**670**	**98.6**	**79.7**	**50.0**	**2.7**	**649**	**1298**

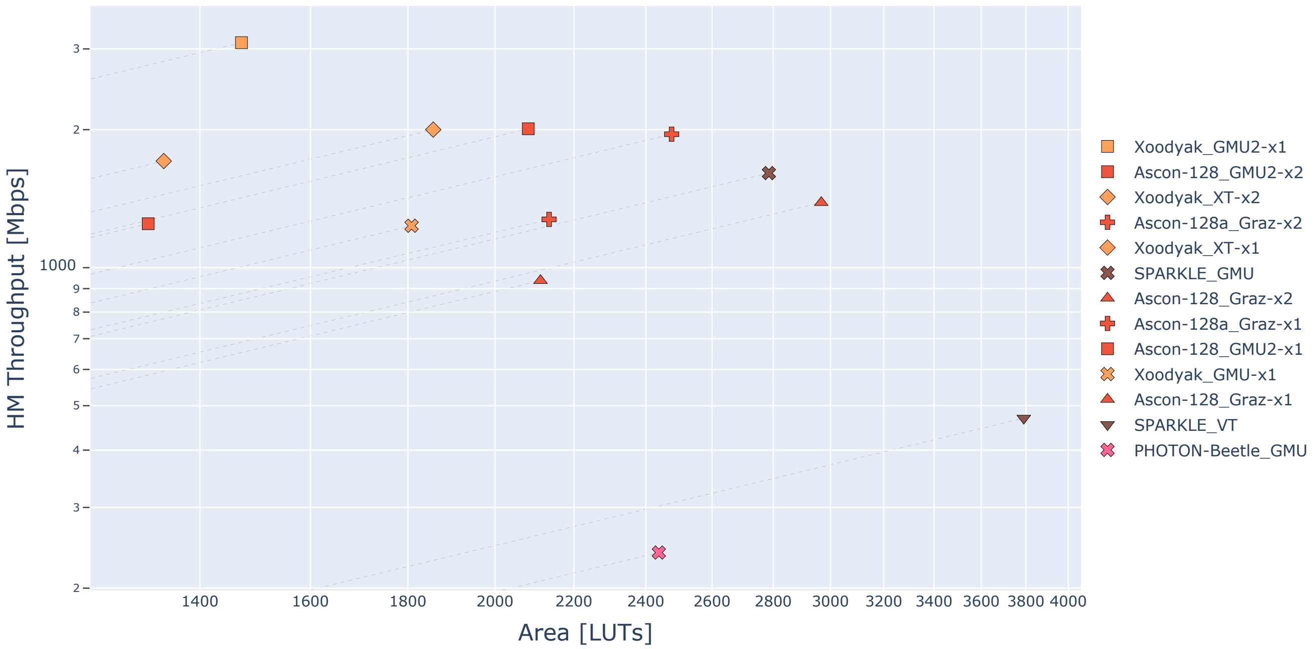

Figure 23: Hashing throughput vs LUTs for long messages (Unprotected)

Table 13: Unprotected implementations: Hashing throughput in Mbit/s for *Long*, *1536 Byte*, *64 Byte* and *16 Byte* messages, along with throughput ratios of different message sizes.

Implementation	$\mathbf{Thr}^{\mathbf{HM}}_{\mathbf{Long}}$	$\mathbf{Thr}^{\mathbf{HM}}_{\mathbf{1536}}$	$\mathbf{Thr}^{\mathbf{HM}}_{\mathbf{64}}$	$\mathbf{Thr}^{\mathbf{HM}}_{\mathbf{16}}$	$\mathbf{Thr\%}^{\mathbf{HM}}_{\mathbf{1536/Long}}$	$\mathbf{Thr\%}^{\mathbf{HM}}_{\mathbf{64/Long}}$	$\mathbf{Thr\%}^{\mathbf{HM}}_{\mathbf{16/Long}}$
Xoodyak_GMU2-x1	3095	3013	1872	856	97.3	60.5	27.7
Ascon-128_GMU2-x2	2009	1949	1160	511	97.0	57.7	25.5
Xoodyak_XT-x2	2000	1965	1397	733	98.2	69.8	36.7
Ascon-128a_Graz-x2	1955	1891	1078	460	96.7	55.2	23.5
Xoodyak_XT-x1	1708	1683	1249	692	98.5	73.1	40.5
SPARKLE_GMU	1608	1559	919	402	97.0	57.1	25.0
Ascon-128_Graz-x2	1387	1346	802	354	97.1	57.8	25.5
Ascon-128a_Graz-x1	1274	1235	718	311	96.9	56.3	24.4
Ascon-128_GMU2-x1	1247	1212	741	334	97.2	59.4	26.8
Xoodyak_GMU-x1	1234	1216	902	500	98.5	73.1	40.5
Ascon-128_Graz-x1	937	912	561	255	97.3	59.9	27.2
ISAP-A_Graz_dn	841	819	499	224	97.4	59.4	26.6
SPARKLE_VT	470	462	331	175	98.3	70.5	37.4
PHOTON-Beetle_GMU	239	240	261	357	100.3	109.0	149.3
Minimum	**239**	**240**	**261**	**175**	**96.7**	**55.2**	**23.5**
Average	**1429**	**1393**	**892**	**440**	**97.7**	**65.6**	**38.3**
Maximum	**3095**	**3013**	**1872**	**856**	**100.3**	**109.0**	**149.3**

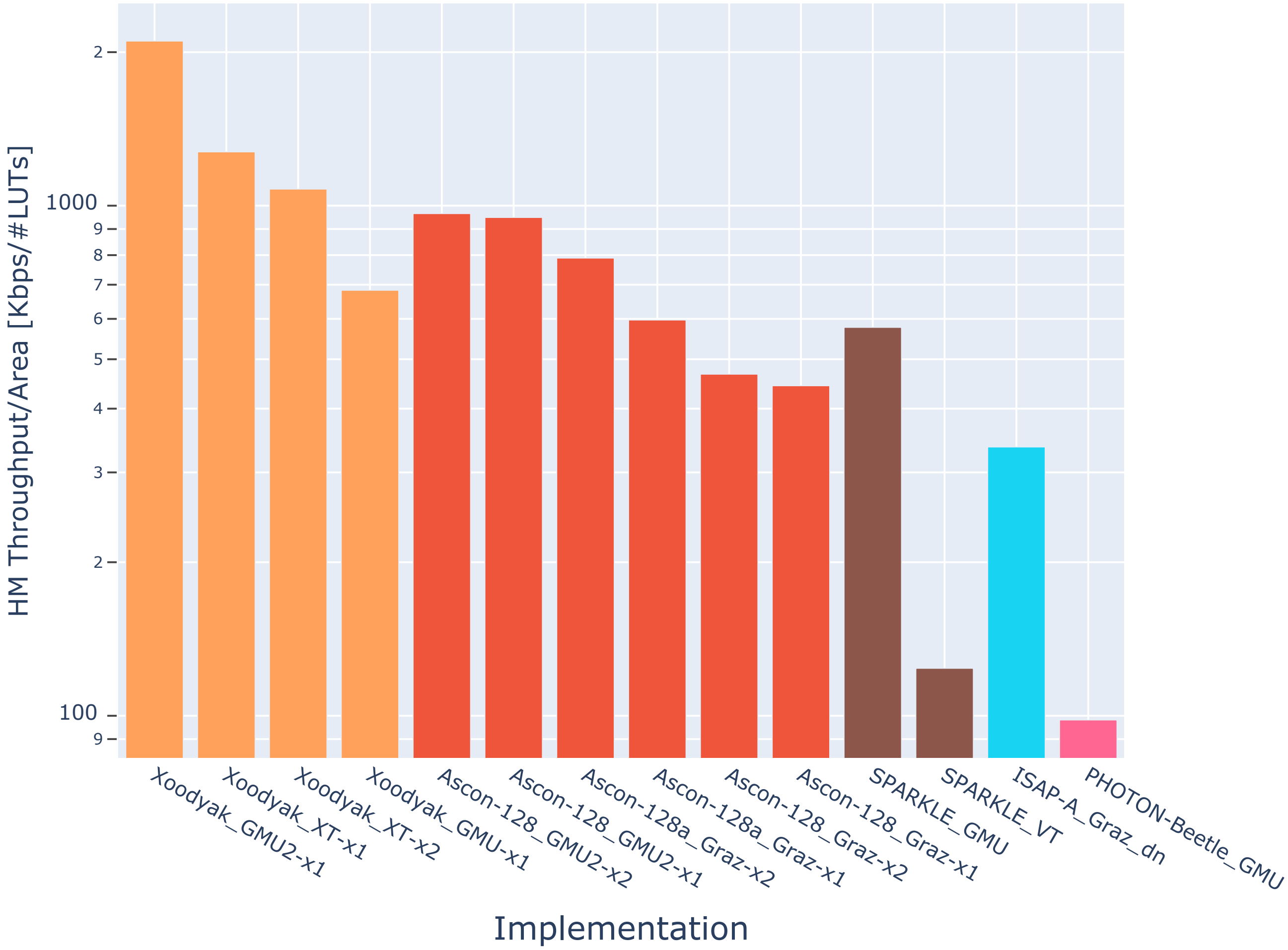

Figure 24: Hashing throughput over area for long messages (Unprotected)

Table 14: Unprotected implementations: throughput-over-area (Kbit/s/#LUTs) for encryption of long messages, resource utilization and maximum frequency.

Implementation	$\mathbf{Thr}^{\mathbf{PT}}_{\mathbf{Long}}$ / **LUTs**	$\mathbf{Thr}^{\mathbf{AD}}_{\mathbf{Long}}$ / **LUTs**	**LUTs**	**FFs**	$\boldsymbol{f}_{\mathbf{max}}$
Xoodyak_GMU2-x1	3154	5783	1472	1251	314
Ascon-128a_GMU2-x2	2190	2190	2476	973	212
Ascon-128_GMU2-x1	1761	1761	1315	863	253
Xoodyak_XT-x1	1711	2484	1340	505	227
Ascon-128_GMU2-x2	1689	1689	2082	864	220
GIFT-COFB_GMU2-x4	1622	1622	1704	873	194
TinyJAMBU_TJT	1589	4235	562	429	223
Grain-128AEAD_GMU	1512	1512	1261	841	238
Xoodyak_XT-x2	1368	1811	1856	512	172
SPARKLE_GMU2	1331	1331	2575	1381	107
Ascon-128a_Graz-x2	1184	1184	2476	587	183
SPARKLE_GMU	1155	1155	2785	1215	100
GIFT-COFB_GMU2-x2	1029	1029	1409	875	238
Ascon-128a_Graz-x1	995	995	2135	586	199
Xoodyak_GMU-x1	916	1330	1808	942	164
Romulus-N_RT-x2	812	1393	1348	502	205
Ascon-128_Graz-x1	776	776	2113	582	205
AESGCM_GMU-v1	716	855	3303	1402	199
Ascon-128_Graz-x2	701	701	2967	588	130
GIFT-COFB_GMU2-x1	610	610	1182	876	231
Romulus-N_RT-x4	605	941	1851	502	122
GIFT-COFB_VT	527	506	1439	691	278
Romulus-N_RT-x1	511	936	1175	480	206
TinyJAMBU_GMU	397	962	590	428	248
AESGCM_GMU-v2	303	301	2565	1510	199
PHOTON-Beetle_GMU	297	350	2438	813	187
ISAP-A_Graz_dn	215	337	2499	1060	184
ISAP-K_Graz_dn16	211	355	2882	1226	176
SPARKLE_VT	179	222	3790	1549	125
Elephant_GMU	151	293	1375	912	222
Minimum	**151**	**222**	**562**	**428**	**100**
Average	**1007**	**1322**	**1959**	**861**	**199**
Maximum	**3154**	**5783**	**3790**	**1549**	**314**

Table 15: Unprotected implementations: throughput-over-area (Kbit/s/#LUTs) for hashing of long and short messages, resource utilization and maximum frequency.

Implementation	$\frac{\mathbf{Thr}^{\mathbf{HM}}_{\mathbf{Long}}}{\mathbf{LUTs}}$	$\frac{\mathbf{Thr}^{\mathbf{HM}}_{\mathbf{16}}}{\mathbf{LUTs}}$	**LUTs**	**FFs**	$f_{\mathbf{max}}$
Xoodyak_GMU2-x1	2103	582	1472	1251	314
Xoodyak_XT-x1	1275	516	1340	505	227
Xoodyak_XT-x2	1078	395	1856	512	172
Ascon-128_GMU2-x2	965	246	2082	864	220
Ascon-128_GMU2-x1	948	254	1315	863	253
Ascon-128a_Graz-x2	789	186	2476	587	183
Xoodyak_GMU-x1	683	276	1808	942	164
Ascon-128a_Graz-x1	597	146	2135	586	199
SPARKLE_GMU	577	144	2785	1215	100
Ascon-128_Graz-x2	467	119	2967	588	130
Ascon-128_Graz-x1	444	121	2113	582	205
ISAP-A_Graz_dn	337	90	2499	1060	184
SPARKLE_VT	124	46	3790	1549	125
PHOTON-Beetle_GMU	98	146	2438	813	187
Minimum	**98**	**46**	**1315**	**505**	**100**
Average	**749**	**233**	**2220**	**851**	**190**
Maximum	**2103**	**582**	**3790**	**1549**	**314**

7.2 Performance of Unprotected Hardware Designs

The throughput for long inputs (plaintexts, associated data, ciphertexts, and hash messages) was determined by measuring the latency for two different-length inputs, calculating the difference, inverting it, and multiplying the result by the difference in input lengths. The input sizes are selected as integer multiples of the algorithm's block size for the input type. Doing that gives the ideal performance, where initialization and finalization latencies are assumed to be negligibly small. However, some applications may benefit from high performance for short inputs.

Thus, we also discuss the ranking of throughputs for 16-byte inputs. Additionally, results are reported for medium-size inputs with 64 and 1536 bytes. However, for simplicity, our focus is on the results for very long and very short inputs.

Table 12 shows throughputs for unprotected designs processing plaintexts of different sizes (long, 1536, 64, and 16 bytes). Table 13 presents the analogous throughputs for processing hash messages of different sizes. These two tables show results for various input lengths and the corresponding ratios (expressed in percentages) of the throughputs for medium and short inputs over throughputs for long inputs. The reported ratios provide insight into the overhead cost of initialization and finalization as a function of an algorithm and input size.

For all investigated unprotected designs, the throughput for 1536-byte plaintexts is equal to at least 90% of the throughput for long plaintexts. For 64-byte plaintexts, the ratio varies between 27.6% for ISAP-K_Graz_dn16 to 79.7% for TinyJAMBU_GMU. For 16-byte plaintexts, this ratio varies between 8.5% for SPARKLE_GMU[18] and 50% for TinyJAMBU_GMU[19].

The implementations of TinyJAMBU, Romulus, PHOTON-Beetle, and Elephant offer some of the smallest overheads for processing short messages. For processing 16-byte plaintexts, all these algorithms have implementations reaching at least 35% of the throughput for long messages. The smallest relative throughput, below 10%, is observed for the implementations of SPARKLE and ISAP.

An additional column in Table 12 shows the ratio of the AD throughput to PT throughput. For designs where the value is 1, the throughputs for AD and PT are the same. When the ratio is greater than 1 that means the design has higher performance for AD than for PT. Among the investigated designs, the ratio varies between 1.00 (for multiple designs) to 2.7 for TinyJAMBU_TJT.

The final two columns show throughput for inputs with 1) 16 bytes of AD and 2) 16 bytes of AD followed by 16 bytes of PT. In terms of absolute values of these throughputs, the highest values are obtained for Xoodyak, Ascon, and GIFT-COFB. For the processing of 16-byte ADs, these are the only algorithms with a throughput exceeding 600 Mbits/s. For the processing of 16-byte ADs and 16-byte plaintexts, these are the only algorithms with a throughput exceeding 1 Gbit/s.

Some algorithms require a permutation or other operation for the AD or PT stage even when no AD or PT is being ingested. Thus the throughput is higher for AD+PT than AD or PT alone since the implementation must perform some operations even if there is no AD or PT.

Overall, Ascon-128a, Xoodyak, Ascon, and SPARKLE are the only algorithms with throughput for long plaintexts exceeding 3 Gbit/s. For short plaintexts of the size of 16 bytes, the following algorithms exceed 500 Mbits/s: Ascon-128, Xoodyak, Ascon-128a, and GIFT-COFB.

The performance for hashing is summarized in Table 13. The ranking of algorithms (based on the throughput of the most efficient unprotected designs processing long messages) is Xoodyak, Ascon-128 (Ascon-Hash), Ascon-128a (Ascon-Hasha), SPARKLE, ISAP, and Photon-BEETLE. For short hash messages, Photon-BEETLE outperforms ISAP.

One unusual result to note is that the throughput for PHOTON-Beatle_GMU has higher throughput for short messages than long messages. This is a result of the way PHOTON-Beatle-Hash processes the message. In hash mode, the first 128-bit block of a message is absorbed in one shot, and subsequently, the remainder of the message is absorbed in chunks of r = 32 bits. Each absorption is followed by 12 rounds of the PHOTON permutation. As the first 128 bits of the message are processed effectively 4 times faster, PHOTON-Beatle's hashing throughput decreases as the input message grows larger. This behavior is different from all other LWC finalists, in which both AEAD and hashing throughputs generally increase with the size of the inputs.

The throughput vs. area graph for unprotected designs performing authenticated encryption for long plaintexts is summarized graphically in Fig. 21. The ratio of throughput-over-area (with area expressed using LUTs) is shown graphically in Fig. 22. The detailed results in terms of throughput-over-area (separately for plaintexts and

[18] https://github.com/kammoh/sparkle

[19] https://github.com/GMUCERG/TinyJAMBU

ADs), number of LUTs, number of flip-flops (FFs), and maximum clock frequency are shown in Table 14. For throughput-over-area, Xoodyak is the winner, with performance substantially higher than the next best algorithm, which is Ascon-128a. The individual implementation does impact the overall ranking as the other implementations of Xoodyak and Ascon rank substantially lower on the list. As shown in Fig. 21, the best implementation of Ascon-128a has slightly higher throughput than that of Xoodyak, but Xoodyak has a substantially lower area which gives it the best ratio of performance to the area. The best implementation of Xoodyak also has the highest hashing throughput and hashing throughput over area ratio of all designs, as shown in Figs. 23 and 24. In terms of the throughput-to-area ratios, Ascon-128a and Ascon-128 are the second and third for processing long plaintexts. For processing of hash messages, they swap places. Out of the remaining algorithms, the following finalists have throughput over area ratio exceeding 1000 kbps/#LUTs for processing of long plaintexts: GIFT-COFB, Tiny_JAMBU, Grain-128AEAD, and SPARKLE. For hashing of long messages, the remaining three algorithms (other than Xoodyak, Ascon-128, and Ascon-128a) in terms of the throughput over area ratio are ranked: SPARKLE, ISAP, and PHOTON-Beetle.

The smallest design supporting authenticated encryption and decryption is TinyJAMBU_TJT with an area of only 562 LUTs and 429 flip-flops (FFs), as shown in Table 14. The smallest designs that support encryption, decryption, and hashing are Ascon-128_GMU2-x1[20] , with 1315 LUTs and 505 FFs, and Xoodyak_XT-x1[20], with 1340 LUTs and 505 FFs.

7.3 Performance of First-Order Protected Hardware Designs

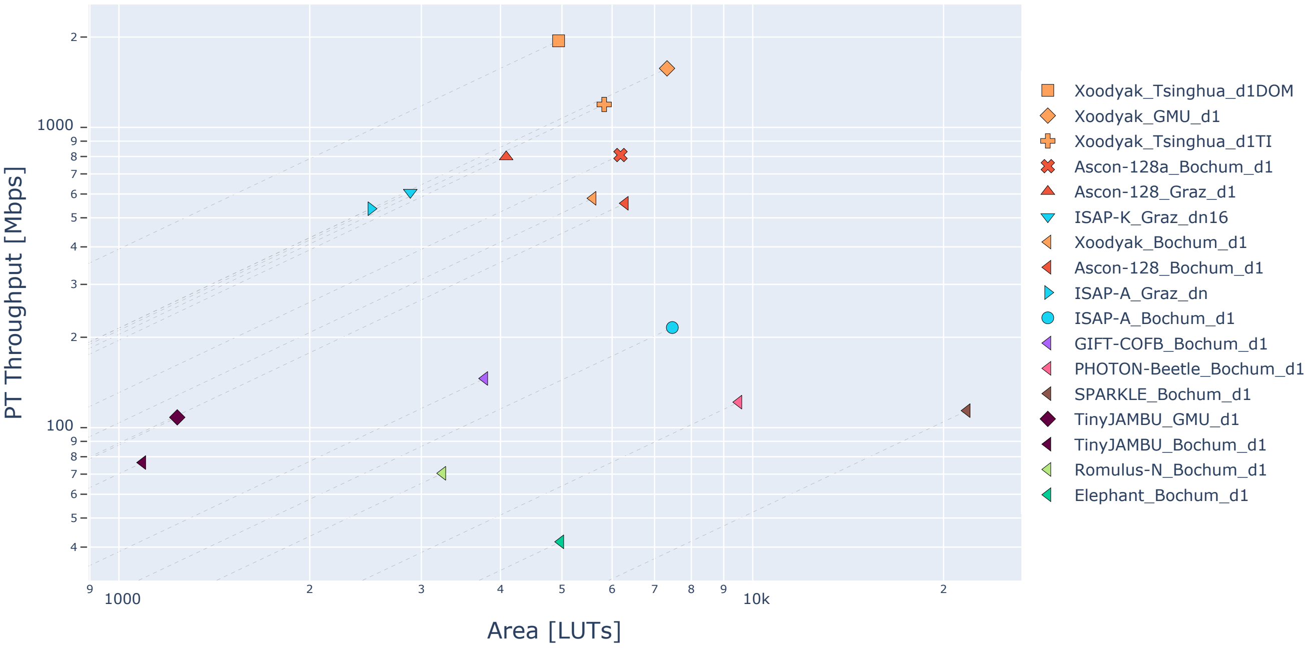

Figure 25: Encryption throughput vs LUTs for long messages (1st order protected)

For the first order protected implementations, results are illustrated in Figs. 25, 26, and 27 for processing plaintexts; in Figs. 28, 29, and 30 for processing ADs; and in Figs. 31, 32, and 33 for processing hash messages. For each input type, the first graph presents results of the dependence throughput vs. area, the second illustrates ratios of throughput over area, and the third shows the number of random bits per input byte.

The detailed numerical results are summarized in Tables 16 and 17, which show all throughput results, and Tables 18 and 19, which show throughput over area ratios, number of LUTs, number of FFs, maximum clock frequency, and the number of random bits required to process each byte of the plaintext and AD.

The first-order protected implementations include both manually and automatically protected designs. The manually generated implementations typically achieve better performance and area results than the corresponding designs generated automatically using AGEMA. The best design overall is Xoodyak_Tshinghua_d1DOM, which has

[20] https://github.com/kammoh/bluelight

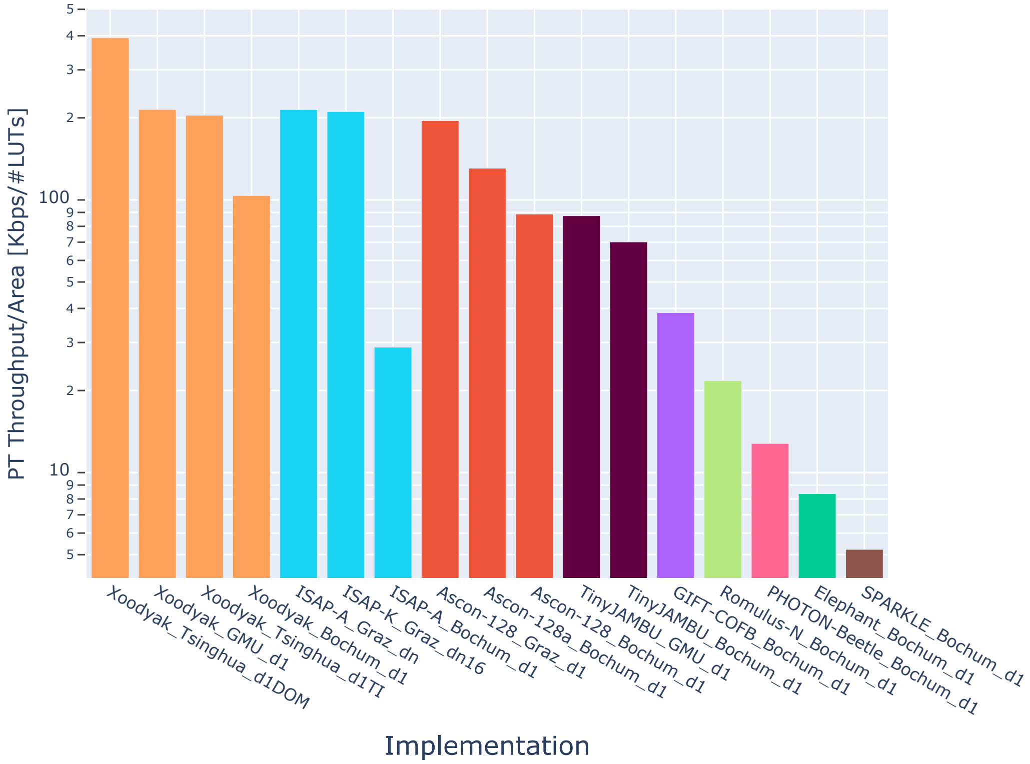

Figure 26: Encryption throughput over area for long messages (1st order protected)

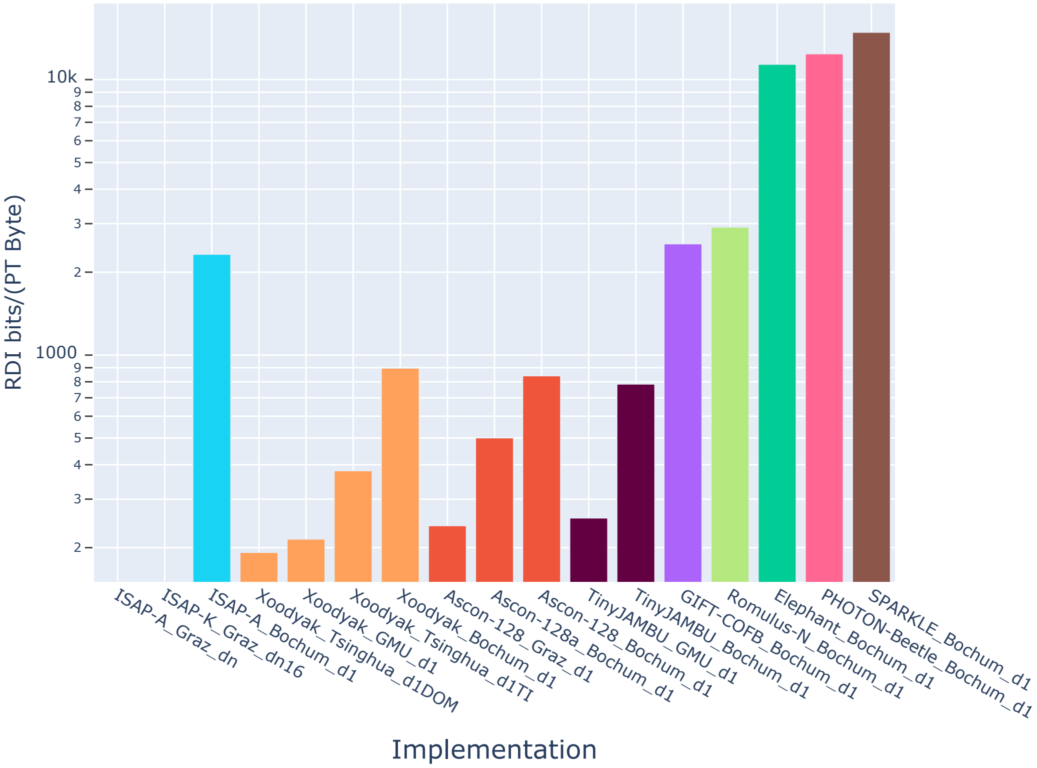

Figure 27: Random bits per plaintext byte (1st order protected)

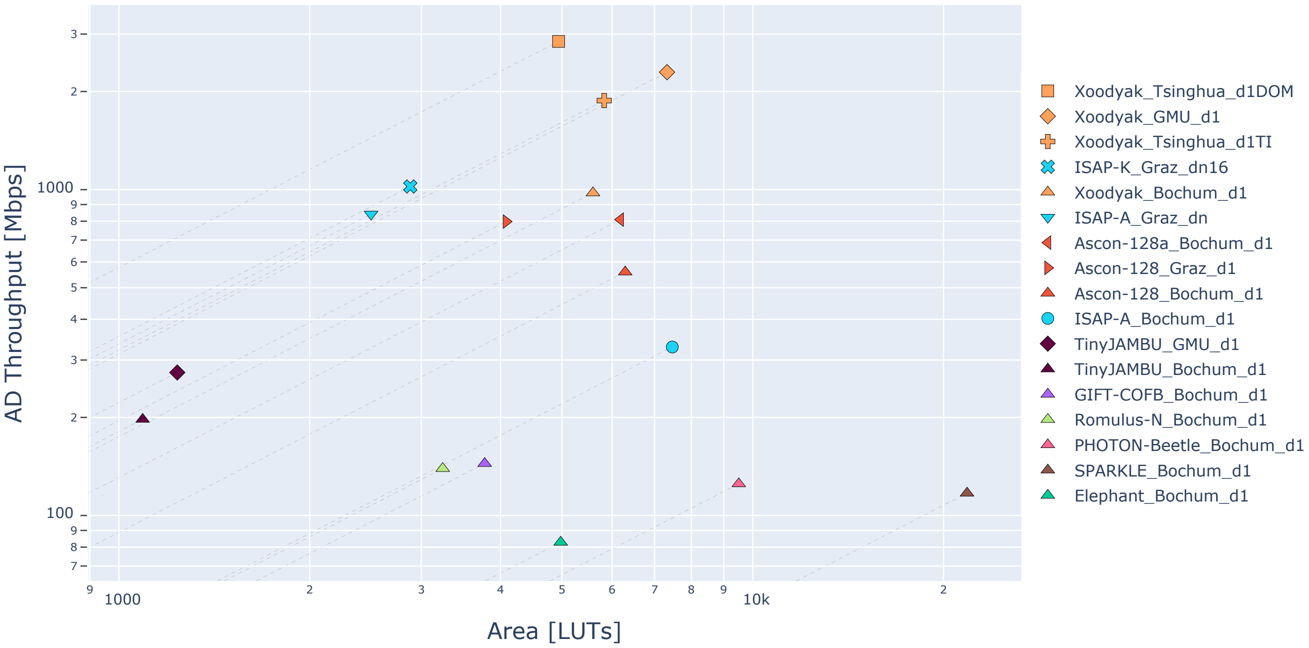

Figure 28: AD throughput vs LUTs for long messages (1st order protected)

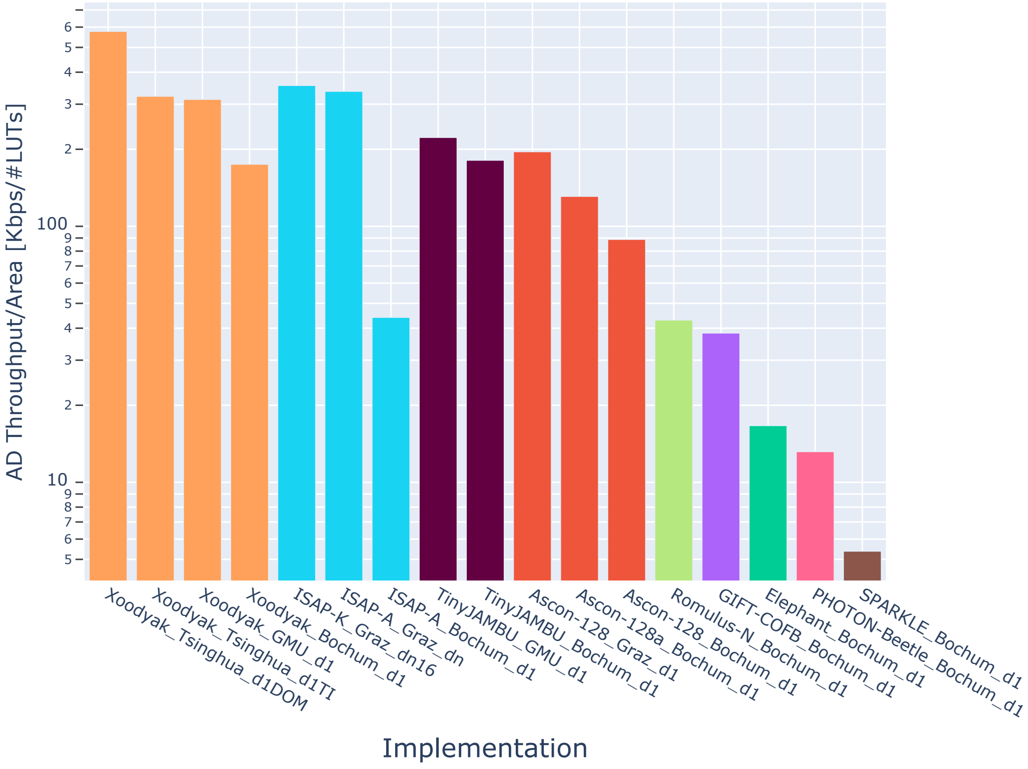

Figure 29: AD throughput over area for long messages (1st order protected)

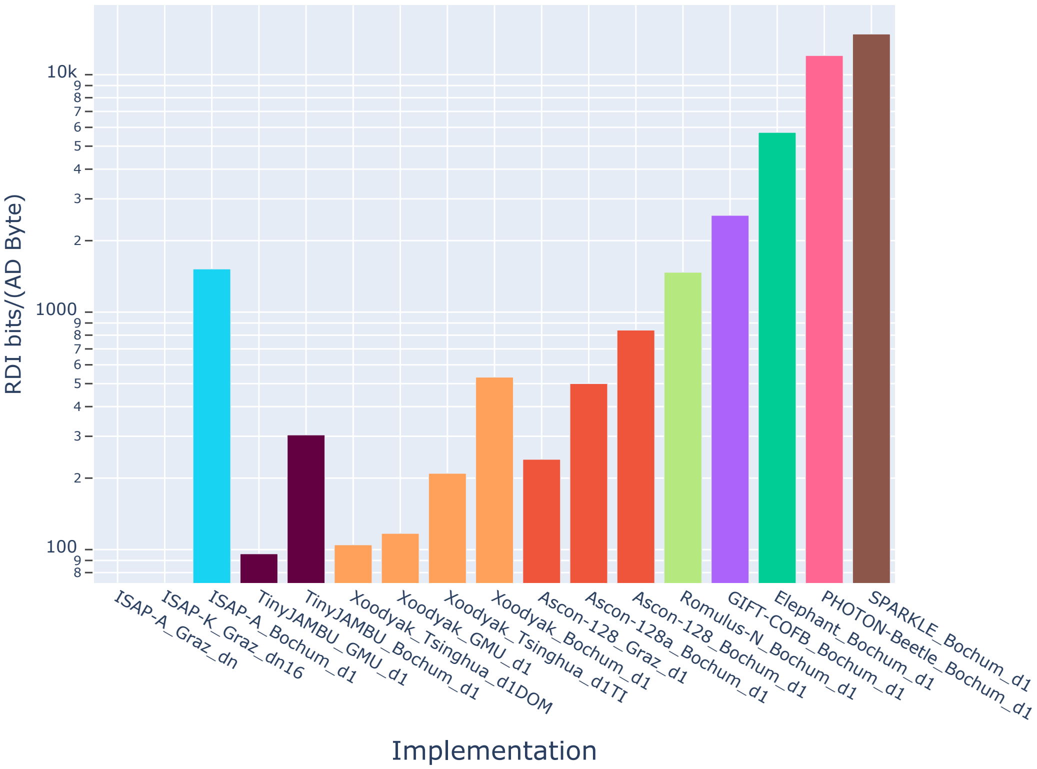

Figure 30: Random bits per AD byte (1st order protected)

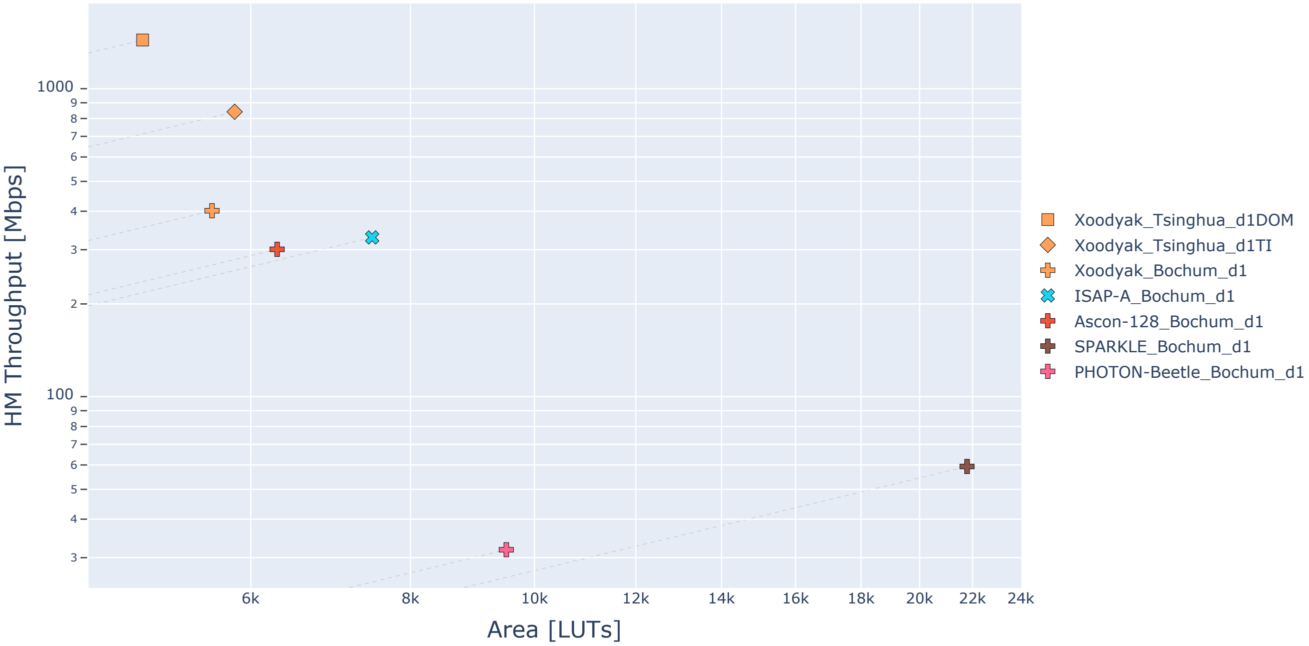

Figure 31: Hashing throughput vs LUTs for long messages (1st order protected)

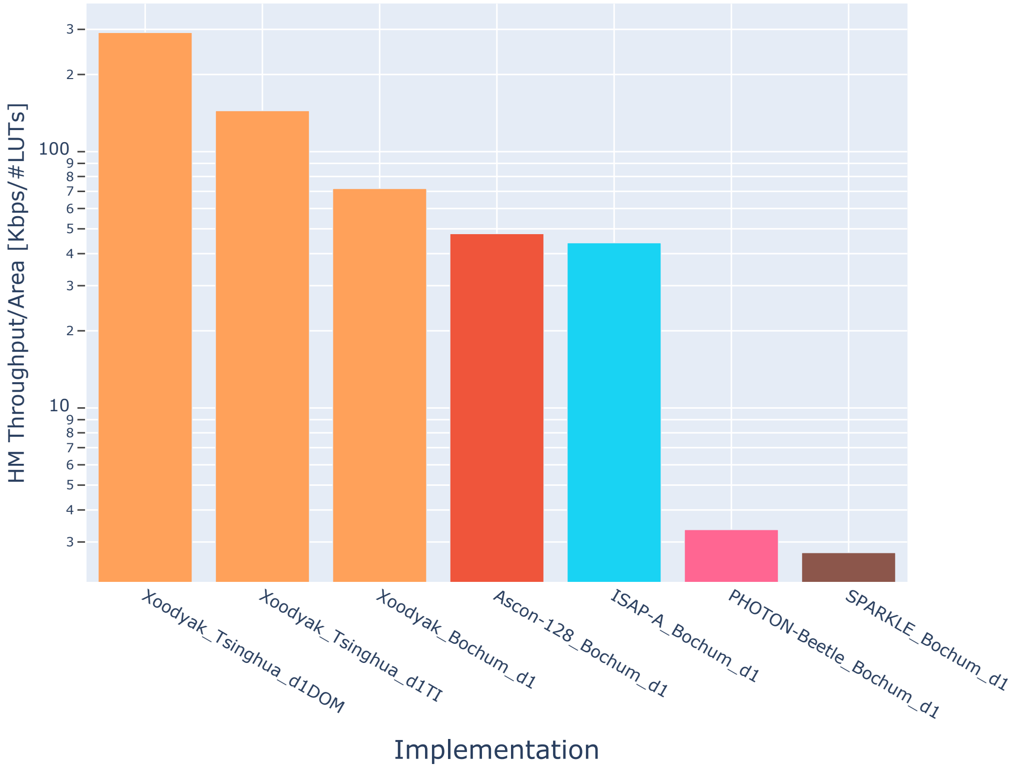

Figure 32: Hashing throughput over area for long messages (1st order protected)

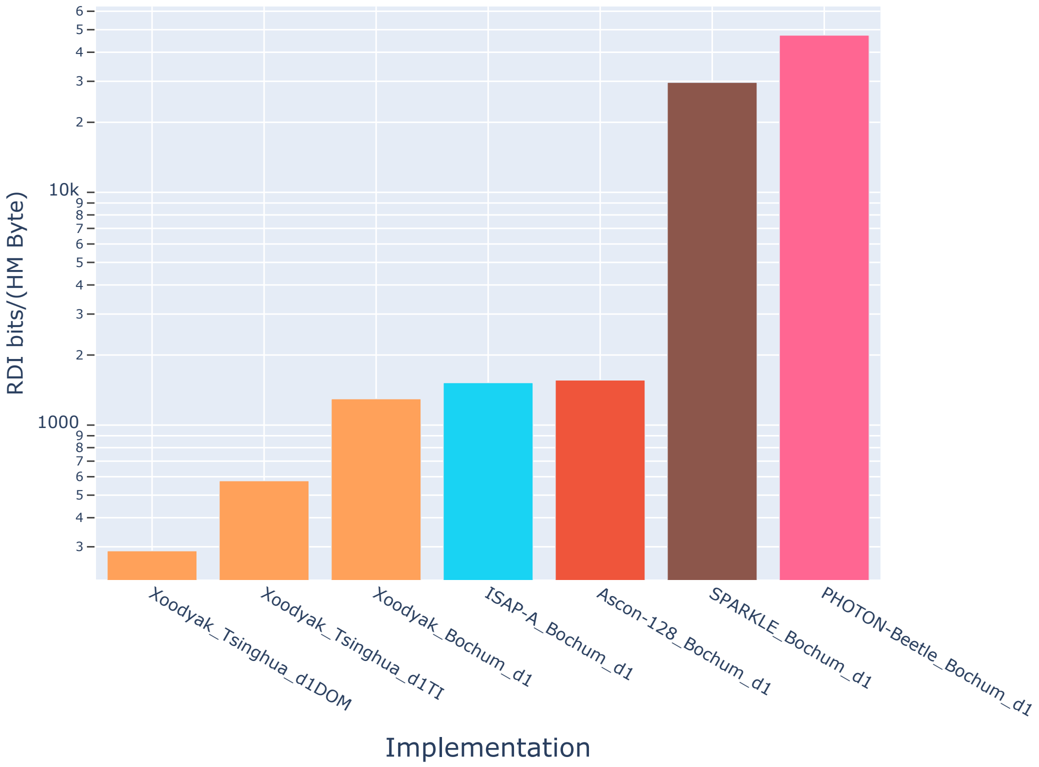

Figure 33: Random bits per HM byte (1st order protected)

Table 16: First-order protected implementations: Encryption throughput in Mbit/s for *Long*, *1536 Byte*, *64 Byte* and *16 Byte* messages, along with throughput ratios of different message types and sizes.

Implementation	$\mathbf{Thr^{PT}_{Long}}$	$\mathbf{Thr^{PT}_{1536}}$	$\mathbf{Thr^{PT}_{64}}$	$\mathbf{Thr^{PT}_{16}}$	$\mathbf{Thr\%^{PT}_{1536/Long}}$	$\mathbf{Thr\%^{PT}_{64/Long}}$	$\mathbf{Thr\%^{PT}_{16/Long}}$	$\mathbf{Thr\%^{AD/PT}_{Long}}$	$\mathbf{Thr^{AD}_{16}}$	$\mathbf{Thr^{AD+PT}_{16}}$
Xoodyak_Tsinghua_d1DOM	1943	1871	968	387	96.3	49.8	19.9	1.5	392	740
Xoodyak_GMU_d1	1573	1532	714	268	97.4	45.4	17.0	1.5	268	515
Xoodyak_Tsinghua_d1TI	1193	1142	605	245	95.8	50.7	20.6	1.6	247	476
Ascon-128a_Bochum_d1	809	780	431	179	96.4	53.3	22.1	1.0	151	254
Ascon-128_Graz_d1	798	777	478	217	97.4	59.9	27.1	1.0	194	313
ISAP-K_Graz_dn16	608	543	168	54	89.3	27.6	8.9	1.7	87	106
Xoodyak_Bochum_d1	581	562	301	123	96.7	51.9	21.2	1.7	124	242
Ascon-128_Bochum_d1	559	545	348	163	97.5	62.2	29.2	1.0	144	229
ISAP-A_Graz_dn	536	492	171	56	91.8	31.9	10.4	1.6	93	105
ISAP-A_Bochum_d1	215	196	65	21	91.2	30.1	9.7	1.5	36	39
GIFT-COFB_Bochum_d1	146	143	96	48	97.9	66.0	32.7	1.0	70	95
PHOTON-Beetle_Bochum_d1	122	120	96	58	98.9	78.6	47.8	1.0	59	79
SPARKLE_Bochum_d1	114	109	56	18	95.8	48.8	16.2	1.0	19	25
TinyJAMBU_GMU_d1	108	107	88	56	99.0	81.0	51.7	2.5	82	93
TinyJAMBU_Bochum_d1	77	76	62	40	99.1	81.3	52.1	2.6	59	66
Romulus-N_Bochum_d1	70	70	56	34	98.9	79.1	48.6	2.0	34	68
Elephant_Bochum_d1	42	41	27	17	98.4	63.8	39.7	2.0	13	26
MINIMUM	**42**	**41**	**27**	**17**	**89.3**	**27.6**	**8.9**	**1.0**	**13**	**25**
AVERAGE	**558**	**536**	**278**	**117**	**96.3**	**56.5**	**27.9**	**1.5**	**122**	**204**
MAXIMUM	**1943**	**1871**	**968**	**387**	**99.1**	**81.3**	**52.1**	**2.6**	**392**	**740**

Table 17: First-order protected implementations: Hashing throughput in Mbit/s for *Long*, *1536 Byte*, *64 Byte* and *16 Byte* messages, along with throughput ratios of different message sizes.

Implementation	$\mathbf{Thr^{HM}_{Long}}$	$\mathbf{Thr^{HM}_{1536}}$	$\mathbf{Thr^{HM}_{64}}$	$\mathbf{Thr^{HM}_{16}}$	$\mathbf{Thr\%^{HM}_{1536/Long}}$	$\mathbf{Thr\%^{HM}_{64/Long}}$	$\mathbf{Thr\%^{HM}_{16/Long}}$
Xoodyak_Tsinghua_d1DOM	1439	1417	1046	576	98.5	72.7	40.0
Xoodyak_Tsinghua_d1TI	841	830	635	366	98.7	75.5	43.5
Xoodyak_Bochum_d1	401	397	310	184	98.8	77.1	45.8
ISAP-A_Bochum_d1	329	320	200	92	97.4	60.7	27.8
Ascon-128_Bochum_d1	301	293	183	84	97.4	60.8	28.0
SPARKLE_Bochum_d1	59	58	42	23	98.4	71.6	38.7
PHOTON-Beetle_Bochum_d1	32	32	36	60	100.5	113.4	189.9
Minimum	**32**	**32**	**36**	**23**	**97.4**	**60.7**	**27.8**
Average	**486**	**478**	**350**	**198**	**98.5**	**76.0**	**59.1**
Maximum	**1439**	**1417**	**1046**	**576**	**100.5**	**113.4**	**189.9**

the highest throughput and throughput over area ratio for PT, AD, and HM as well as one of the lowest costs in terms of randomness. The next best three algorithms are Ascon-128a, Ascon-128, and ISAP. Their exact ranking varies depending on the performance metrics. For example, for encryption throughput, the ranking is Ascon-128a, Ascon-128, and ISAP. However, already for encryption throughput-over-area, the order is reversed and becomes ISAP, Ascon-128, and Ascon-128a. For AD throughput, the order is still different: ISAP, Ascon-128a, and Ascon. For AD throughput-over-area, ISAP is ahead and separated from both variants of Ascon by TinyJAMBU.

It should be stressed that ISAP has **mode-level protection** in place of traditional countermeasures, such as masking. Consequently, ISAP does not require any randomness for authenticated encryption and decryption (including the processing of AD). However, the mode-level protection does not apply to hash functions. Hence, only the masked implementation of ISAP, ISAP-A_Bochum_d1, can be counted in this case.

Thus, for processing of plaintext and AD, ISAP has zero randomness requirements. In terms of the number of random bits per plaintext byte, ISAP (with 0) is followed by Xoodyak, Ascon-128, and TinyJAMBU. In terms of the number of random bits per AD byte, ISAP (with 0) is followed by TinyJAMBU, Xoodyak, and Ascon-128. In terms of the number of random bits per hash message byte, Xoodyak is the best, followed by ISAP and Ascon-128, which are in a virtual tie with one another.

Most algorithms only have designs generated automatically using AGEMA. For these designs, developed at Bochum, the ranking by throughput over area ratio for PT is Ascon-128a, Xoodyak, Ascon-128, TinyJAMBU, GIFT-COFB, ISAP-A, Romulus-N, PHOTON-Beetle, Elephant, and then SPARKLE. These designs typically have a higher cost in terms of area and randomness, as well as lower throughput than the corresponding manually protected designs.

Similarly to the unprotected implementations, the smallest design is a semi-automatically generated implementation of TinyJAMBU, TinyJAMBU_Bochum_d1, with an area of 1090 LUTs and 1157 FFs. The manually protected implementation of TinyJAMBU, TinyJAMBU_GMU_d1, has more LUTs but fewer FFs.

Table 18: First-order protected implementations: throughput-over-area (Kbit/s/#LUTs) for encryption of long messages, resource utilization, maximum frequency and the number of required fresh random bits for encrypting 1 Byte of message.

Implementation	$\mathbf{Thr}^{\mathbf{PT}}_{\mathbf{Long}}$/**LUTs**	$\mathbf{Thr}^{\mathbf{AD}}_{\mathbf{Long}}$/**LUTs**	**LUTs**	**FFs**	$\boldsymbol{f}_{\mathbf{max}}$	$\mathbf{Rnd}^{\mathbf{PT}}_{\mathbf{Long}}$	$\mathbf{Rnd}^{\mathbf{AD}}_{\mathbf{Long}}$
Xoodyak_Tsinghua_d1DOM	393	577	4939	2582	202	192	105
Xoodyak_GMU_d1	215	313	7324	3379	159	214	117
ISAP-A_Graz_dn	215	337	2499	1060	184		
ISAP-K_Graz_dn16	211	355	2882	1226	176		
Xoodyak_Tsinghua_d1TI	205	322	5829	3379	197	380	209
Ascon-128_Graz_d1	195	195	4083	2185	175	240	240
Ascon-128a_Bochum_d1	131	131	6185	5746	183	500	500
Xoodyak_Bochum_d1	104	175	5596	6193	169	896	532
Ascon-128_Bochum_d1	89	89	6292	5752	183	840	840
TinyJAMBU_GMU_d1	88	222	1236	946	223	256	96
TinyJAMBU_Bochum_d1	70	181	1090	1157	234	784	304
GIFT-COFB_Bochum_d1	39	38	3776	3702	240	2532	2556
ISAP-A_Bochum_d1	29	44	7466	6477	195	2320	1520
Romulus-N_Bochum_d1	22	43	3242	2940	200	2912	1472
PHOTON-Beetle_Bochum_d1	13	13	9505	12118	168	12390	12040
Elephant_Bochum_d1	8	17	4977	5488	211	11354	5712
SPARKLE_Bochum_d1	5	5	21783	24030	146	14832	14832
Minimum	**5**	**5**	**1090**	**946**	**146**	**192**	**96**
Average	**119**	**180**	**5806**	**5198**	**191**	**3376**	**2738**
Maximum	**393**	**577**	**21783**	**24030**	**240**	**14832**	**14832**

Table 19: First-order protected implementations: throughput-over-area (Kbit/s/#LUTs) for hashing of long and short messages, resource utilization, maximum frequency and the number of required fresh random bits for hashing 1 Byte of message.

Implementation	$\frac{\textbf{Thr}^{\textbf{HM}}_{\textbf{Long}}}{\textbf{LUTs}}$	$\frac{\textbf{Thr}^{\textbf{HM}}_{\textbf{16}}}{\textbf{LUTs}}$	**LUTs**	**FFs**	$f_{\textbf{max}}$	$\textbf{Rnd}^{\textbf{HM}}_{\textbf{Long}}$
Xoodyak_Tsinghua_d1DOM	291	117	4939	2582	202	288
Xoodyak_Tsinghua_d1TI	144	63	5829	3379	197	576
Xoodyak_Bochum_d1	72	33	5596	6193	169	1296
Ascon-128_Bochum_d1	48	13	6292	5752	183	1560
ISAP-A_Bochum_d1	44	12	7466	6477	195	1520
PHOTON-Beetle_Bochum_d1	3	6	9505	12118	168	47320
SPARKLE_Bochum_d1	3	1	21783	24030	146	29664
Minimum	**3**	**1**	**4939**	**2582**	**146**	**288**
Average	**86**	**35**	**8773**	**8647**	**180**	**11746**
Maximum	**291**	**117**	**21783**	**24030**	**202**	**47320**

7.4 Performance of Second-Order Protected Hardware Designs

The second-order designs were primarily generated using AGEMA with the exception of Ascon-128_Graz_d2, which was implemented manually.

Results are illustrated in Figs. 34, 35, and 36 for processing plaintexts; in Figs. 37, 38, and 39 for processing ADs; and in Figs. 40, 41, and 42 for processing hash messages.

The detailed numerical results are summarized in Tables 20, 21, 22 and 23.

The protected designs maintain the same cycle latency for all levels of protection but increase in area and have slightly different maximum frequencies. Thus the throughput for a higher-order protected design may be slightly higher than the corresponding implementation for a lower-order of protection, but the area will increase.

The automatically protected Ascon-128a has the best throughput for processing long plaintexts, followed by the manually implemented implementation of Ascon-128. Mode-protected ISAP has the highest throughput for processing AD and the highest throughput-over-area ratio for processing both plaintexts and ADs. The overall ranking in terms of PT throughput-over-area is ISAP, Ascon-128a, Ascon-128, TinyJAMBU, Xoodyak, GIFT-COFB, Romulus-N, Elephant, and PHOTON-Beetle.

In terms of randomness requirements, they are none for the mode-protected implementations of ISAP. Among the masked implementations, they are by far the lowest in the manual design of Ascon-128 (Ascon-128_Graz_d2). Among automatically protected designs, the smallest number of random bits per plaintext byte is required for Ascon-128a, Ascon-128, TinyJAMBU, and Xoodyak. In terms of the number of random bits per AD byte, the order changes to ISAP (mode-protected), Ascon-128, TinyJAMBU, Ascon-128a, and Xoodyak.

TinyJAMBU achieves the lowest area and fourth-best throughput over area ratio for processing plaintexts (after ISAP, Ascon-128a, and Ascon-128).

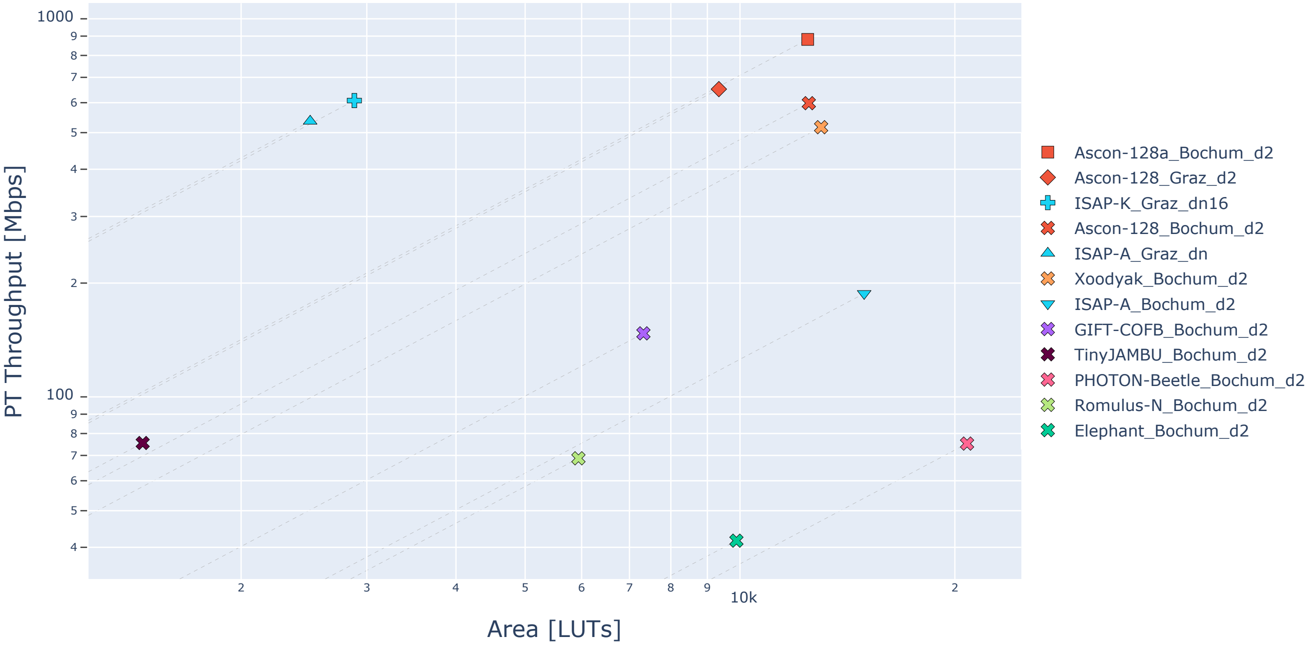

Figure 34: Encryption throughput vs LUTs for long messages (2nd order protected)

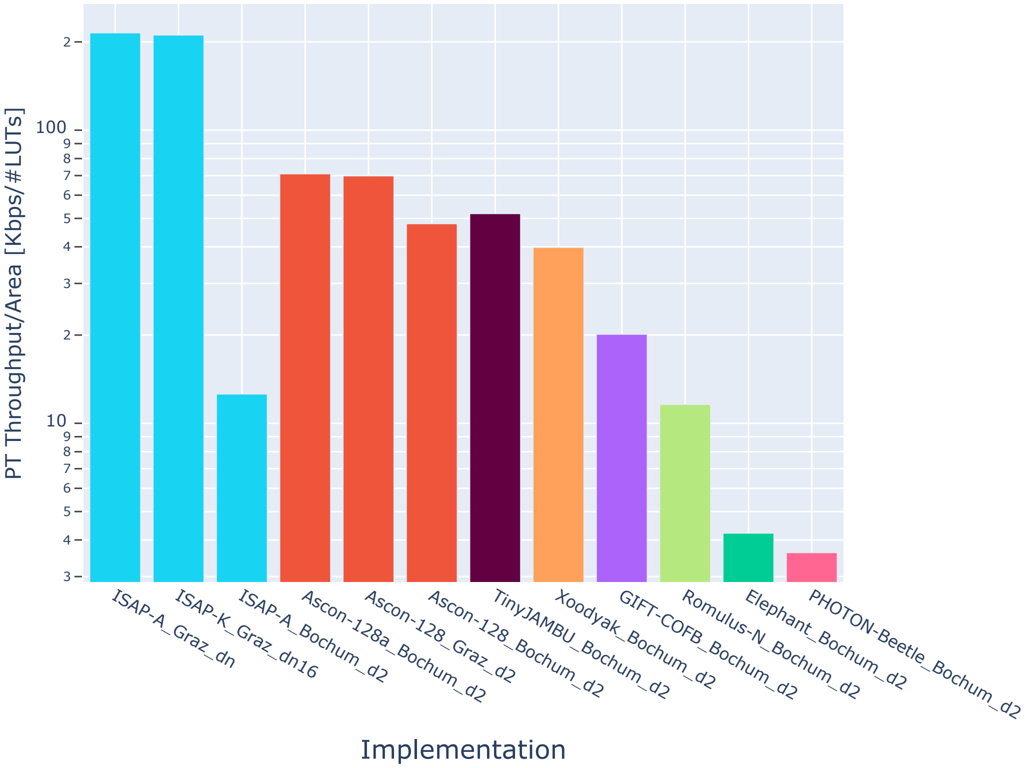

Figure 35: Encryption throughput over area for long messages (2nd order protected)

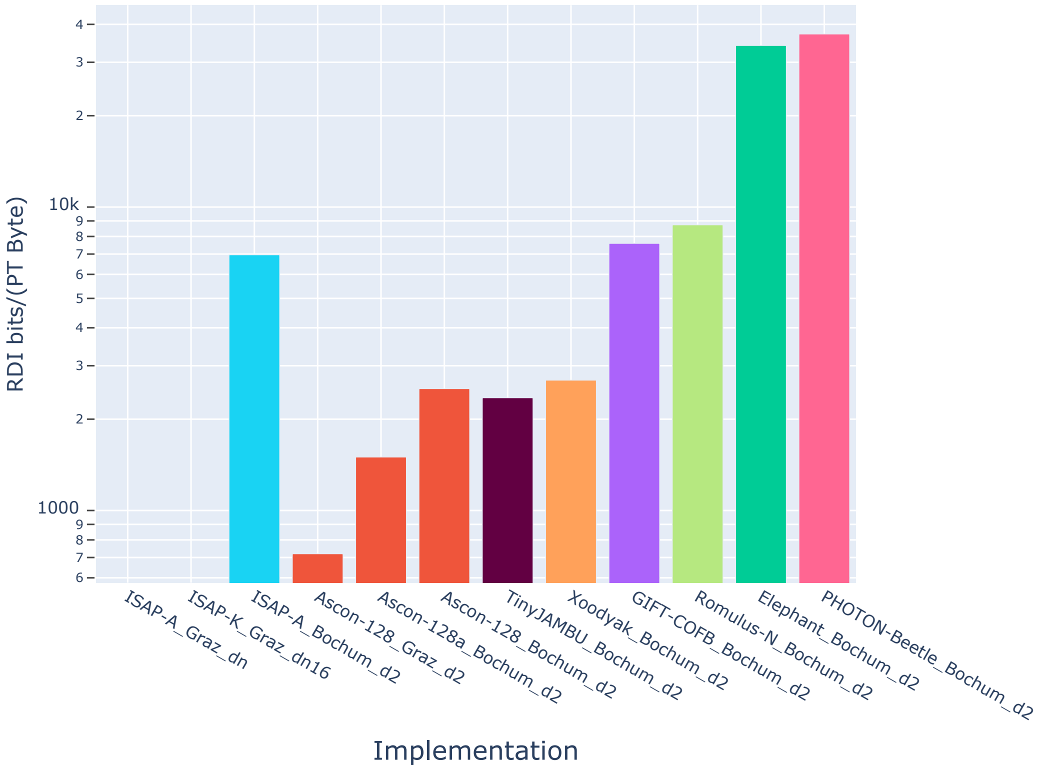

Figure 36: Random bits per plaintext byte (2nd order protected)

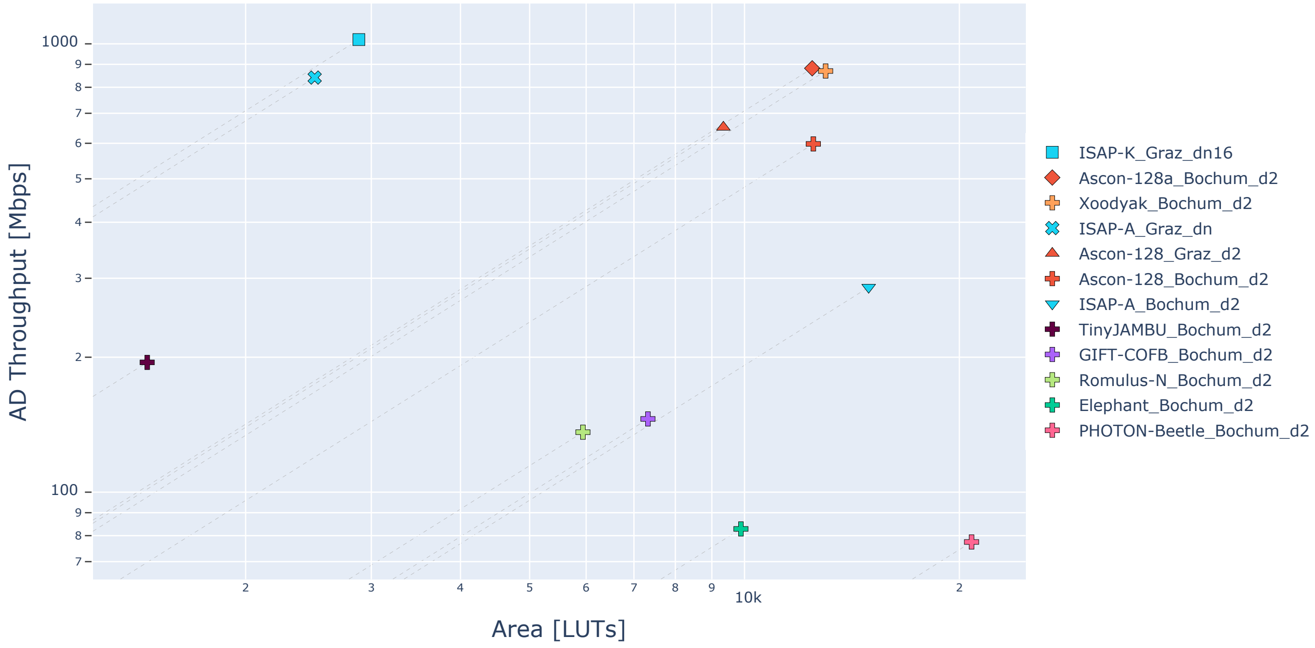

Figure 37: AD throughput vs LUTs for long messages (2nd order protected)

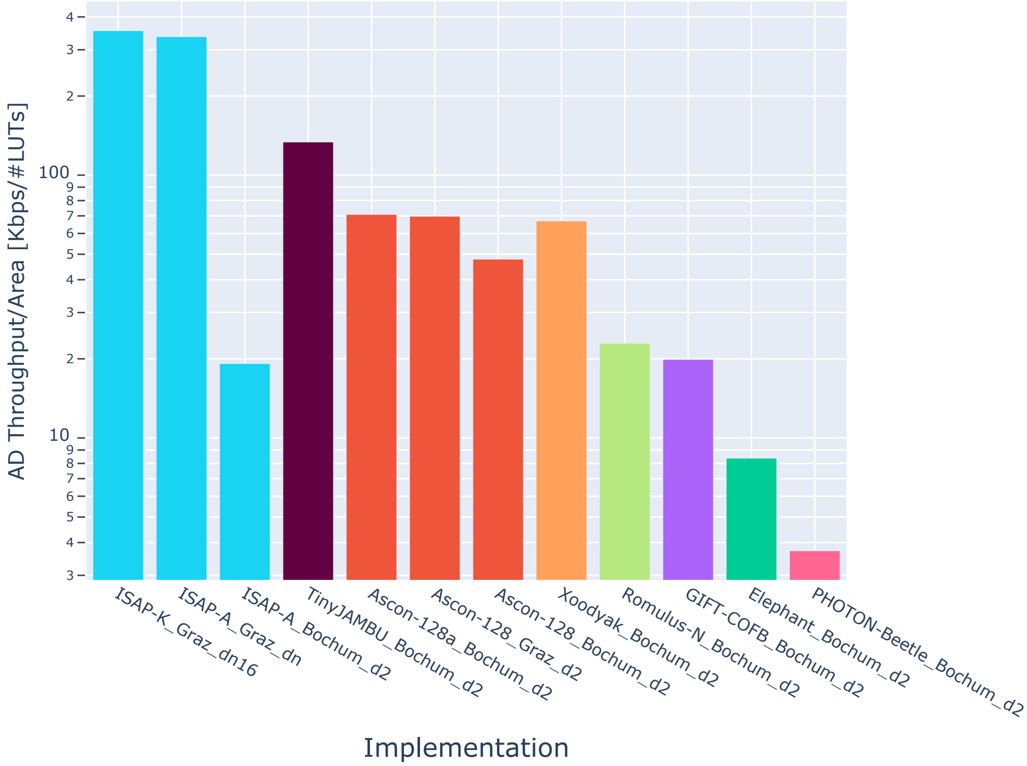

Figure 38: AD throughput over area for long messages (2nd order protected)

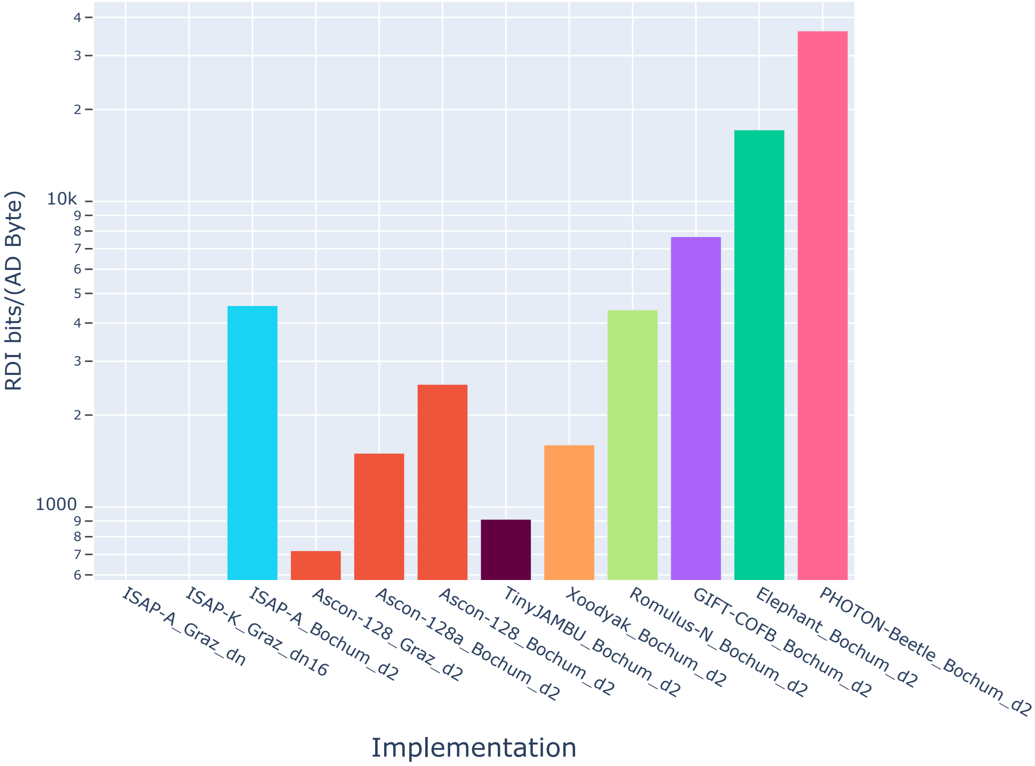

Figure 39: Random bits per AD byte (2nd order protected)

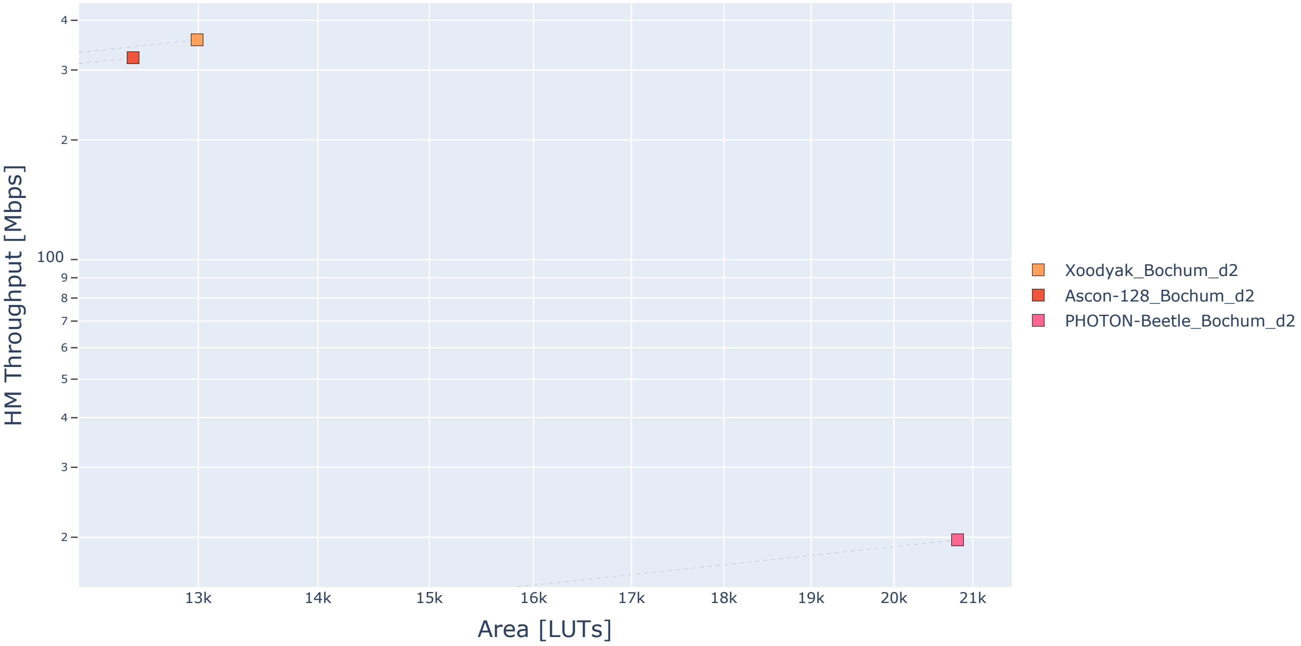

Figure 40: Hashing throughput vs LUTs for long messages (2nd order protected)

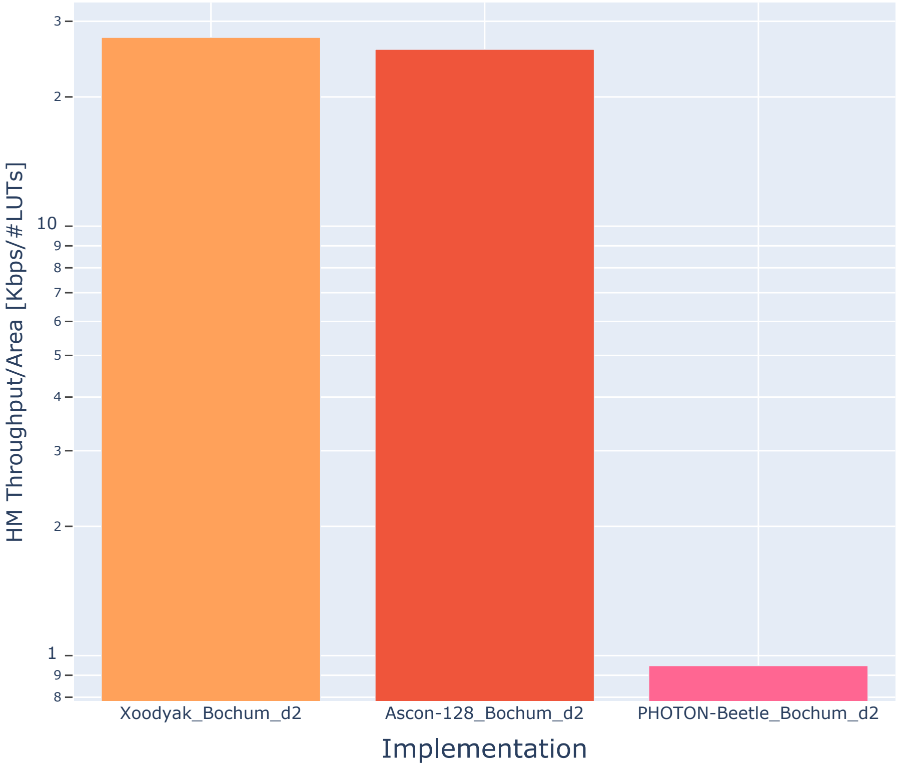

Figure 41: Hashing throughput over area for long messages (2nd order protected)

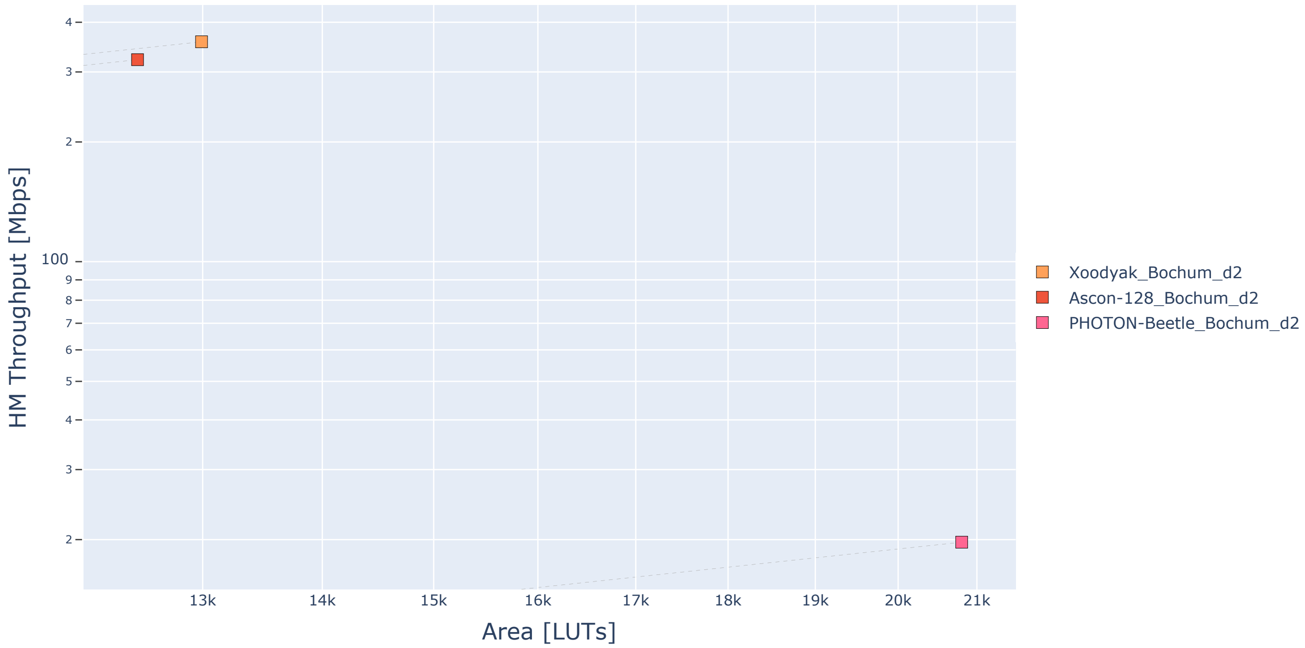

Figure 42: Random bits per HM byte (2nd order protected)

Table 20: Second-order protected implementations: Encryption throughput in Mbit/s for *Long*, *1536 Byte*, *64 Byte* and *16 Byte* messages, along with throughput ratios of different message types and sizes.

Implementation	$\mathbf{Thr^{PT}_{Long}}$	$\mathbf{Thr^{PT}_{1536}}$	$\mathbf{Thr^{PT}_{64}}$	$\mathbf{Thr^{PT}_{16}}$	$\mathbf{Thr\%^{PT}_{1536/Long}}$	$\mathbf{Thr\%^{PT}_{64/Long}}$	$\mathbf{Thr\%^{PT}_{16/Long}}$	$\mathbf{Thr\%^{AD/PT}_{Long}}$	$\mathbf{Thr^{AD}_{16}}$	$\mathbf{Thr^{AD+PT}_{16}}$
Ascon-128a_Bochum_d2	882	850	470	195	96.4	53.3	22.1	1.0	165	276
Ascon-128_Graz_d2	652	635	390	177	97.4	59.9	27.1	1.0	159	256
ISAP-K_Graz_dn16	608	543	168	54	89.3	27.6	8.9	1.7	87	106
Ascon-128_Bochum_d2	598	583	372	174	97.5	62.2	29.2	1.0	154	245
ISAP-A_Graz_dn	536	492	171	56	91.8	31.9	10.4	1.6	93	105
Xoodyak_Bochum_d2	517	500	268	110	96.7	51.9	21.2	1.7	110	216
ISAP-A_Bochum_d2	188	171	56	18	91.2	30.1	9.7	1.5	31	34
GIFT-COFB_Bochum_d2	147	144	97	48	97.8	65.9	32.6	1.0	71	95
TinyJAMBU_Bochum_d2	76	75	61	39	99.1	81.3	52.1	2.6	58	66
PHOTON-Beetle_Bochum_d2	75	74	59	36	98.9	78.6	47.8	1.0	36	49
Romulus-N_Bochum_d2	69	68	54	33	98.9	79.1	48.6	2.0	33	67
Elephant_Bochum_d2	42	41	27	17	98.4	63.8	39.7	2.0	13	26
Minimum	**42**	**41**	**27**	**17**	**89.3**	**27.6**	**8.9**	**1.0**	**13**	**26**
Average	**366**	**348**	**183**	**80**	**96.1**	**57.1**	**29.1**	**1.5**	**84**	**128**
Maximum	**882**	**850**	**470**	**195**	**99.1**	**81.3**	**52.1**	**2.6**	**165**	**276**

Table 21: Second-order protected implementations: Hashing throughput in Mbit/s for *Long*, *1536 Byte*, *64 Byte* and *16 Byte* messages, along with throughput ratios of different message sizes.

Implementation	$\mathbf{Thr^{HM}_{Long}}$	$\mathbf{Thr^{HM}_{1536}}$	$\mathbf{Thr^{HM}_{64}}$	$\mathbf{Thr^{HM}_{16}}$	$\mathbf{Thr\%^{HM}_{1536/Long}}$	$\mathbf{Thr\%^{HM}_{64/Long}}$	$\mathbf{Thr\%^{HM}_{16/Long}}$
Xoodyak_Bochum_d2	357	353	276	163	98.8	77.1	45.8
Ascon-128_Bochum_d2	322	314	196	90	97.4	60.8	28.0
PHOTON-Beetle_Bochum_d2	20	20	22	37	100.5	113.4	189.9
Minimum	**20**	**20**	**22**	**37**	**97.4**	**60.8**	**28.0**
Average	**233**	**229**	**165**	**97**	**98.9**	**83.8**	**87.9**
Maximum	**357**	**353**	**276**	**163**	**100.5**	**113.4**	**189.9**

Table 22: Second-order protected implementations: throughput-over-area (Kbit/s/#LUTs) for encryption of long messages, resource utilization, maximum frequency and the number of required fresh random bits for encrypting 1 Byte of message.

Implementation	$\mathbf{Thr^{PT}_{Long} \mid LUTs}$	$\mathbf{Thr^{AD}_{Long} \mid LUTs}$	**LUTs**	**FFs**	$\boldsymbol{f_{\mathrm{max}}}$	$\mathbf{Rnd^{PT}_{Long}}$	$\mathbf{Rnd^{AD}_{Long}}$
ISAP-A_Graz_dn	215	337	2499	1060	184		
ISAP-K_Graz_dn16	211	355	2882	1226	176		
Ascon-128a_Bochum_d2	71	71	12441	12374	200	1500	1500
Ascon-128_Graz_d2	70	70	9341	4061	143	720	720
TinyJAMBU_Bochum_d2	52	134	1456	2019	231	2352	912
Ascon-128_Bochum_d2	48	48	12486	12381	196	2520	2520
Xoodyak_Bochum_d2	40	67	12991	14046	151	2688	1597
GIFT-COFB_Bochum_d2	20	20	7323	8068	242	7587	7668
ISAP-A_Bochum_d2	13	19	14926	13314	170	6960	4560
Romulus-N_Bochum_d2	12	23	5938	6224	196	8736	4416
Elephant_Bochum_d2	4	8	9885	11898	211	34062	17136
PHOTON-Beetle_Bochum_d2	4	4	20803	30544	104	37170	36120
Minimum	**4**	**4**	**1456**	**1060**	**104**	**720**	**720**
Average	**63**	**96**	**9414**	**9768**	**184**	**10430**	**7715**
Maximum	**215**	**355**	**20803**	**30544**	**242**	**37170**	**36120**

Table 23: Second-order protected implementations: throughput-over-area (Kbit/s/#LUTs) for hashing of long and short messages, resource utilization, maximum frequency and the number of required fresh random bits for hashing 1 Byte of message.

Implementation	$\frac{\mathbf{Thr}^{\mathbf{HM}}_{\mathbf{Long}}}{\mathbf{LUTs}}$	$\frac{\mathbf{Thr}^{\mathbf{HM}}_{\mathbf{16}}}{\mathbf{LUTs}}$	**LUTs**	**FFs**	$f_{\mathbf{max}}$	$\mathbf{Rnd}^{\mathbf{HM}}_{\mathbf{Long}}$
Xoodyak_Bochum_d2	28	13	12991	14046	151	3888
Ascon-128_Bochum_d2	26	7	12486	12381	196	4680
PHOTON-Beetle_Bochum_d2	1	2	20803	30544	104	141960
MINIMUM	**1**	**2**	**12486**	**12381**	**104**	**3888**
AVERAGE	**18**	**7**	**15427**	**18990**	**150**	**50176**
MAXIMUM	**28**	**13**	**20803**	**30544**	**196**	**141960**

7.5 Performance of Third-Order Protected Hardware Designs

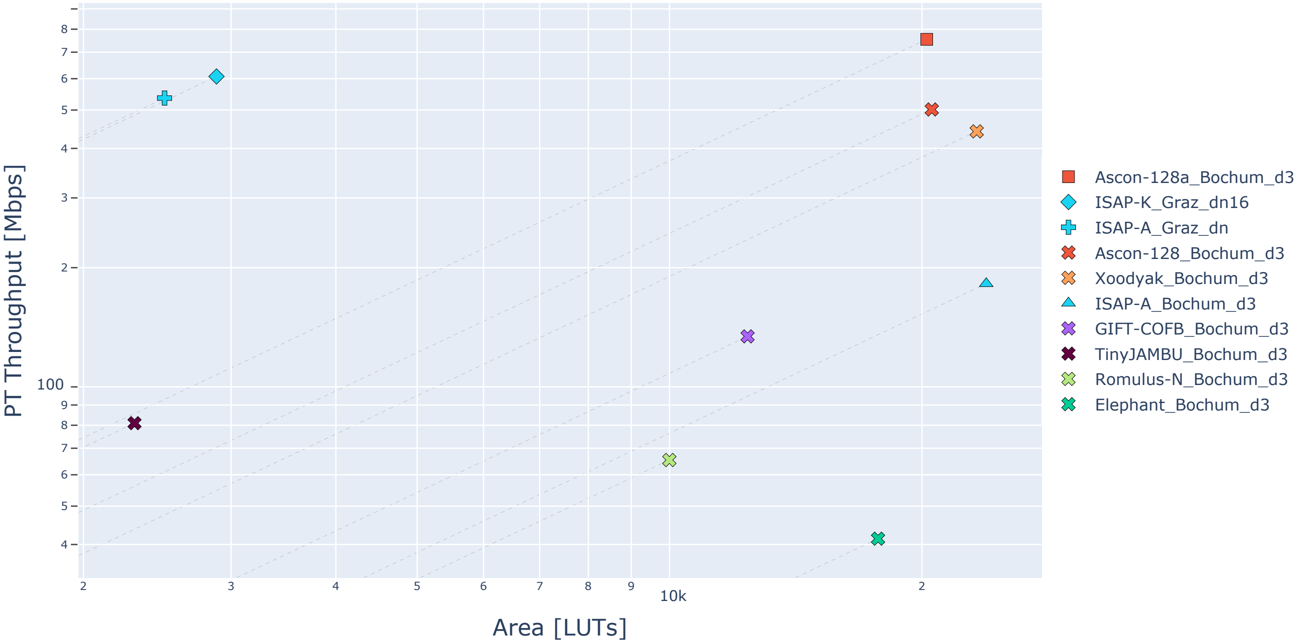

Figure 43: Encryption throughput vs LUTs for long messages (3rd order protected)

We see the same trend in the third-order protected designs which were all generated using AGEMA. These results are shown in Tables 24, 25, 26, and 27. Ascon-128a has the highest throughput for processing plaintexts. However, it is outperformed by a mode-protected ISAP in terms of the throughput for ADs. Ascon-128 and Xoodyak are close third and fourth in terms of the throughput for processing plaintexts. They swap places for the processing of ADs.

In terms of the throughput-to-area ratios, unmasked, mode-level protected ISAP is unbeatable due to its small area. For processing plaintext, it is followed by Ascon-128a, TinyJAMBU, Ascon-128, and Xoodyak. For the processing of AD, ISAP is followed by TinyJAMBU, Ascon-128a, Xoodyak, and Ascon-128.

Only three 3rd-order protected designs support hashing. Out of them, Xoodyak is the fastest, followed by a masked ISAP and Ascon-128. For the throughput vs. area ratio, Ascon-128 (Ascon-Hash) and Xoodyak are in a virtual tie, followed relatively closely by the masked ISAP.

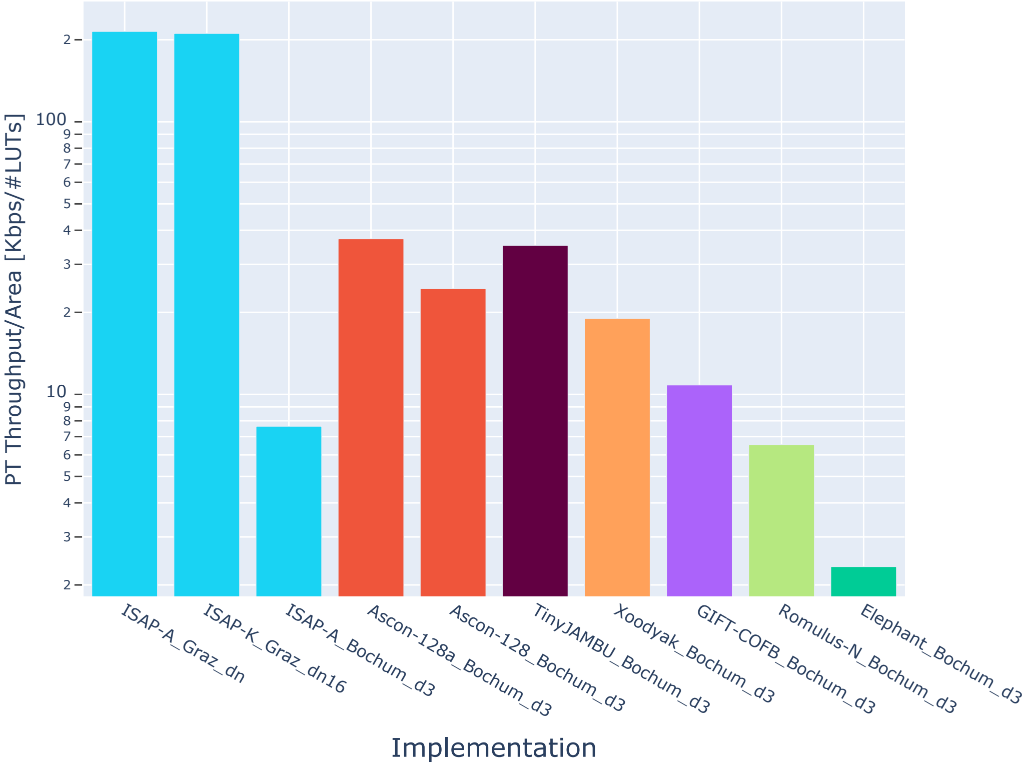

Figure 44: Encryption throughput over area for long messages (3rd order protected)

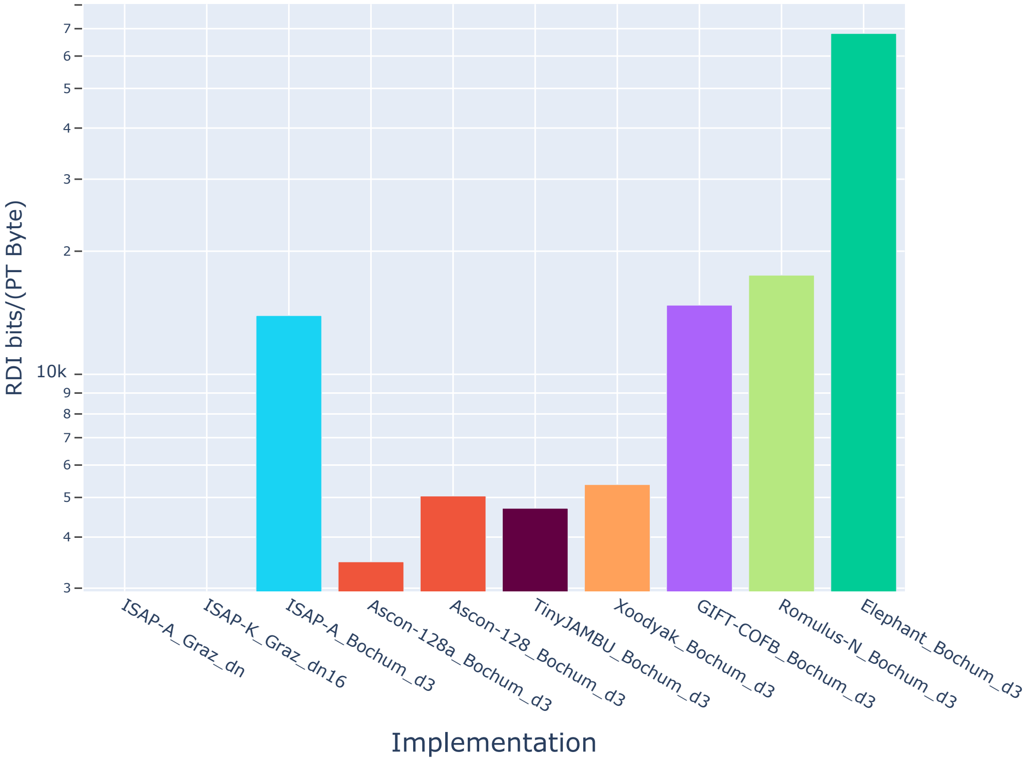

Figure 45: Random bits per Plaintext byte (3rd order protected)

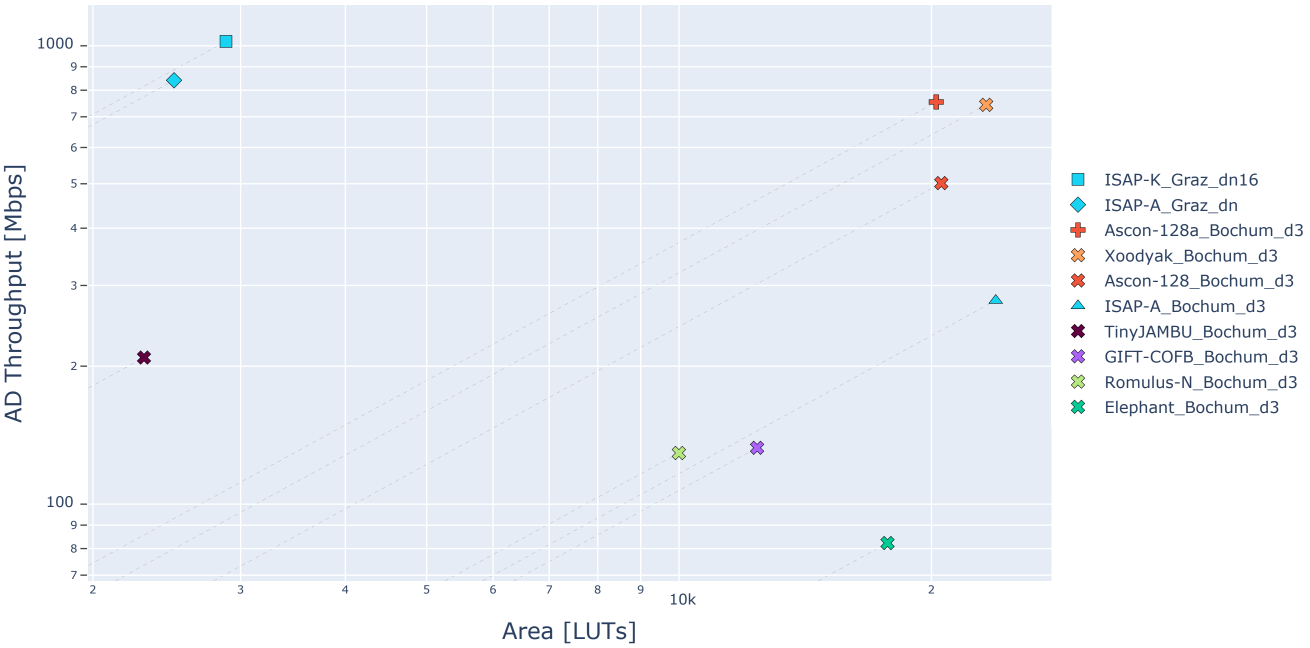

Figure 46: AD throughput vs LUTs for long messages (3rd order protected)

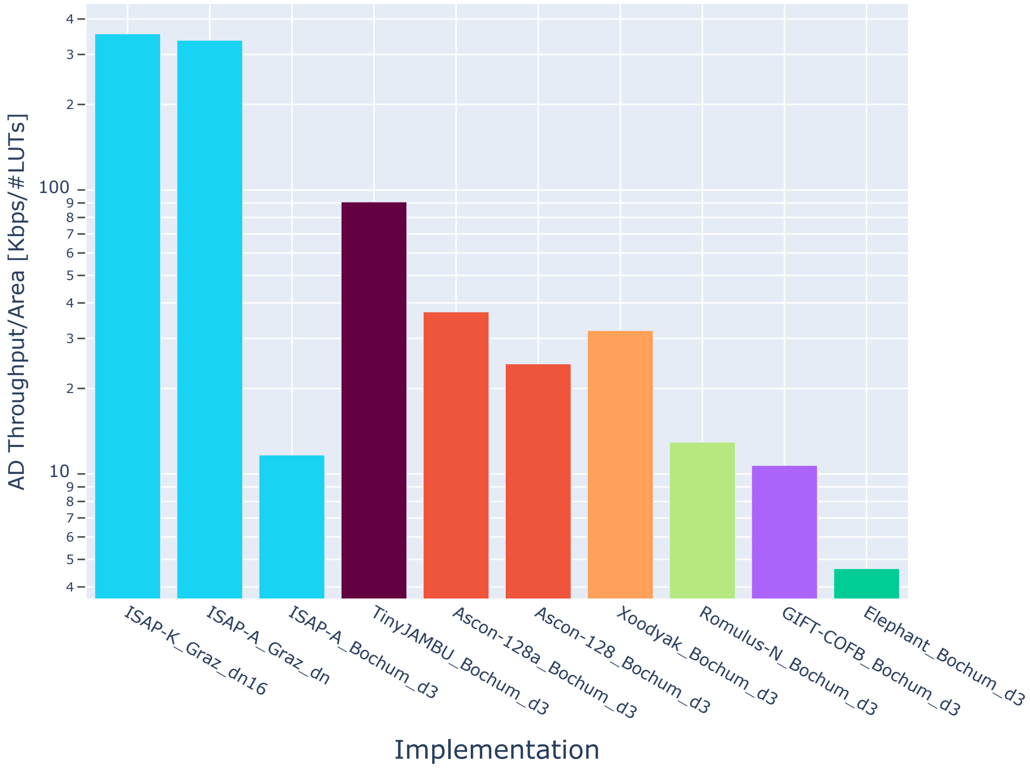

Figure 47: AD throughput over area for long messages (3rd order protected)

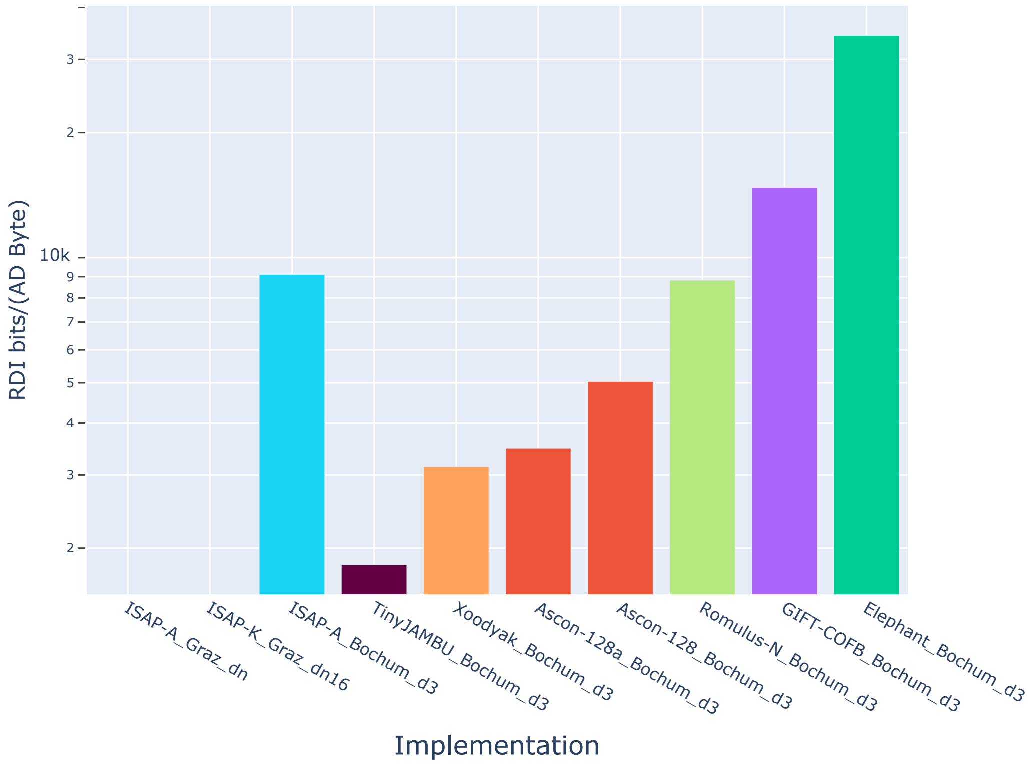

Figure 48: Random bits per AD byte (3rd order protected)

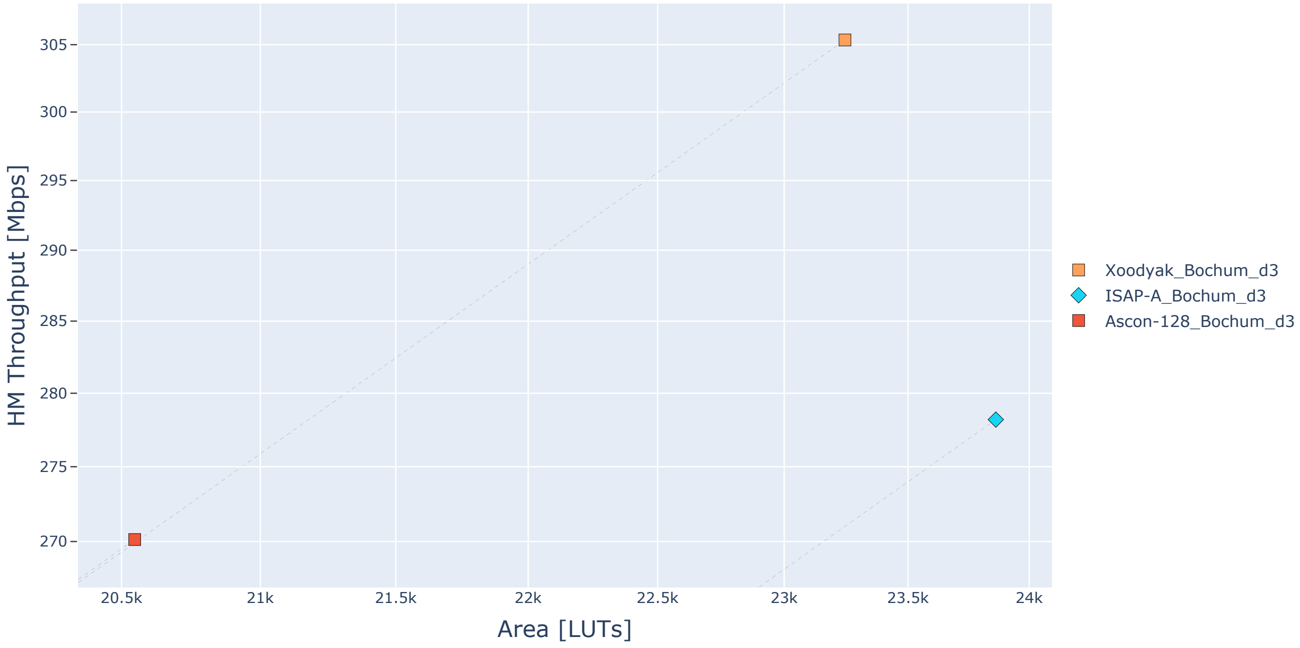

Figure 49: Hashing throughput vs LUTs for long messages (3rd order protected)

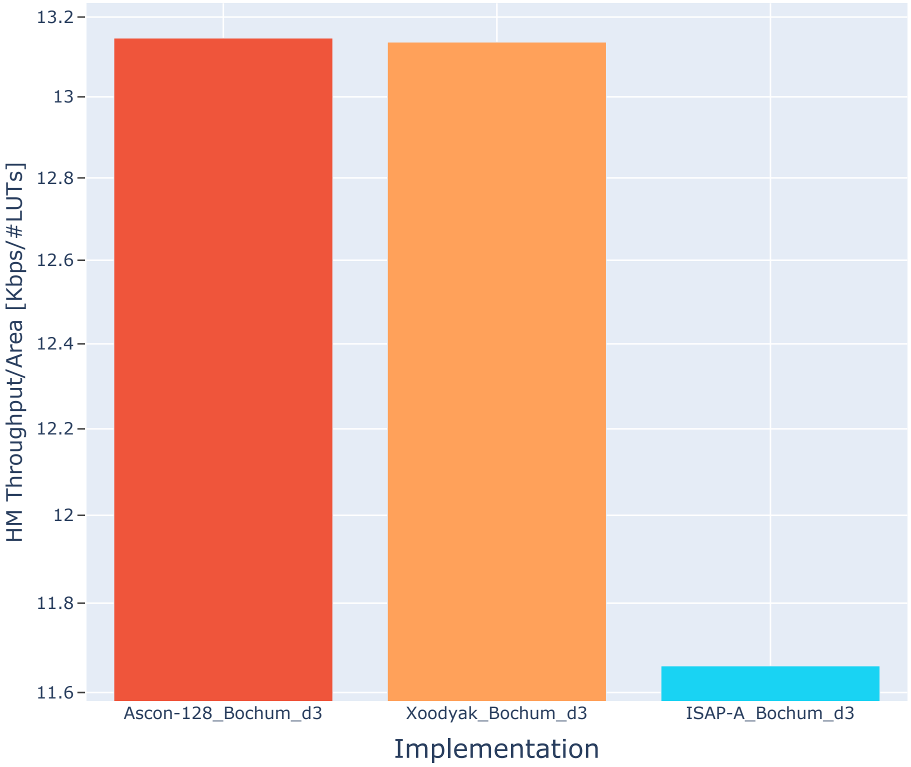

Figure 50: Hashing throughput over area for long messages (3rd order protected)

Table 24: Third-order protected implementations: Encryption throughput in Mbit/s for *Long*, *1536 Byte*, *64 Byte* and *16 Byte* messages, along with throughput ratios of different message types and sizes.

Implementation	$\mathbf{Thr^{PT}_{Long}}$	$\mathbf{Thr^{PT}_{1536}}$	$\mathbf{Thr^{PT}_{64}}$	$\mathbf{Thr^{PT}_{16}}$	$\mathbf{Thr\%^{PT}_{1536/Long}}$	$\mathbf{Thr\%^{PT}_{64/Long}}$	$\mathbf{Thr\%^{PT}_{16/Long}}$	$\mathbf{Thr\%^{AD/PT}_{Long}}$	$\mathbf{Thr^{AD}_{16}}$	$\mathbf{Thr^{AD+PT}_{16}}$
Ascon-128a_Bochum_d3	754	727	401	167	96.5	53.2	22.1	1.0	140	236
ISAP-K_Graz_dn16	608	543	168	54	89.3	27.6	8.9	1.7	87	106
ISAP-A_Graz_dn	536	492	171	56	91.8	31.9	10.4	1.6	93	105
Ascon-128_Bochum_d3	502	489	312	146	97.5	62.2	29.2	1.0	129	206
Xoodyak_Bochum_d3	442	427	229	94	96.7	51.9	21.2	1.7	94	184
ISAP-A_Bochum_d3	182	166	55	18	91.2	30.1	9.7	1.5	30	33
GIFT-COFB_Bochum_d3	134	131	88	44	97.8	65.9	32.6	1.0	64	87
TinyJAMBU_Bochum_d3	81	80	66	42	99.1	81.3	52.1	2.6	62	70
Romulus-N_Bochum_d3	65	65	52	32	98.9	79.1	48.6	2.0	32	64
Elephant_Bochum_d3	41	41	26	16	98.4	63.8	39.7	2.0	13	26
Minimum	**41**	**41**	**26**	**16**	**89.3**	**27.6**	**8.9**	**1.0**	**13**	**26**
Average	**335**	**316**	**157**	**67**	**95.7**	**54.7**	**27.5**	**1.6**	**75**	**112**
Maximum	**754**	**727**	**401**	**167**	**99.1**	**81.3**	**52.1**	**2.6**	**140**	**236**

Table 25: Third-order protected implementations: Hashing throughput in Mbit/s for *Long*, *1536 Byte*, *64 Byte* and *16 Byte* messages, along with throughput ratios of different message sizes.

Implementation	$\mathbf{Thr^{HM}_{Long}}$	$\mathbf{Thr^{HM}_{1536}}$	$\mathbf{Thr^{HM}_{64}}$	$\mathbf{Thr^{HM}_{16}}$	$\mathbf{Thr\%^{HM}_{1536/Long}}$	$\mathbf{Thr\%^{HM}_{64/Long}}$	$\mathbf{Thr\%^{HM}_{16/Long}}$
Xoodyak_Bochum_d3	305	302	236	140	98.8	77.1	45.8
ISAP-A_Bochum_d3	278	271	169	77	97.4	60.7	27.8
Ascon-128_Bochum_d3	270	263	164	76	97.4	60.8	28.0
MINIMUM	**270**	**263**	**164**	**76**	**97.4**	**60.7**	**27.8**
AVERAGE	**285**	**279**	**190**	**98**	**97.8**	**66.2**	**33.9**
MAXIMUM	**305**	**302**	**236**	**140**	**98.8**	**77.1**	**45.8**

Table 26: Third-order protected implementations: throughput-over-area (Kbit/s/#LUTs) for encryption of long messages, resource utilization, maximum frequency and the number of required fresh random bits for encrypting 1 Byte of message.

Implementation	$\mathbf{Thr^{PT}_{Long}/LUTs}$	$\mathbf{Thr^{AD}_{Long}/LUTs}$	**LUTs**	**FFs**	$\boldsymbol{f_{\max}}$	$\mathbf{Rnd^{PT}_{Long}}$	$\mathbf{Rnd^{AD}_{Long}}$
ISAP-A_Graz_dn	215	337	2499	1060	184		
ISAP-K_Graz_dn16	211	355	2882	1226	176		
Ascon-128a_Bochum_d3	37	37	20263	21565	171	3480	3480
TinyJAMBU_Bochum_d3	35	91	2301	3184	248	4704	1824
Ascon-128_Bochum_d3	24	24	20547	21567	165	5040	5040
Xoodyak_Bochum_d3	19	32	23244	24836	129	5376	3142
GIFT-COFB_Bochum_d3	11	11	12390	13801	221	14760	14760
ISAP-A_Bochum_d3	8	12	23861	23013	165	13920	9120
Romulus-N_Bochum_d3	7	13	9995	10236	186	17472	8832
Elephant_Bochum_d3	2	5	17727	20395	210	68124	34272
MINIMUM	**2**	**5**	**2301**	**1060**	**129**	**3480**	**1824**
AVERAGE	**57**	**92**	**13571**	**14088**	**185**	**16610**	**10059**
MAXIMUM	**215**	**355**	**23861**	**24836**	**248**	**68124**	**34272**

Table 27: Third-order protected implementations: throughput-over-area (Kbit/s/#LUTs) for hashing of long and short messages, resource utilization, maximum frequency and the number of required fresh random bits for hashing 1 Byte of message.

Implementation	$\mathbf{Thr^{HM}_{Long}}$ / **LUTs**	$\mathbf{Thr^{HM}_{16}}$ / **LUTs**	**LUTs**	**FFs**	$\boldsymbol{f}_{\mathbf{max}}$	$\mathbf{Rnd^{HM}_{Long}}$
Ascon-128_Bochum_d3	13	4	20547	21567	165	9360
Xoodyak_Bochum_d3	13	6	23244	24836	129	7776
ISAP-A_Bochum_d3	12	3	23861	23013	165	9120
Minimum	**12**	**3**	**20547**	**21567**	**129**	**7776**
Average	**13**	**4**	**22551**	**23139**	**153**	**8752**
Maximum	**13**	**6**	**23861**	**24836**	**165**	**9360**

8 General Conclusions

The cryptographic community developed SCA-protected hardware implementations of 9 out of 10 finalists in the NIST Lightweight Cryptography standardization process. Most of these designs were generated semi-automatically using a novel software tool developed at Ruhr University Bochum, called AGEMA. Typically, one design was developed for each protection order between 1 and 3. In the case of Ascon, separate designs were developed for two distinct variants, Ascon-128 and Ason-128a. Six out of 42 protected designs were developed manually. They covered only 3 out of 10 candidates (Xoodyak, Ascon, and TinyJAMBU). As expected, they were typically better than automatically protected designs, at least in terms of the throughput-to-area ratio. One algorithm, ISAP, was claimed to provide mode-level protection, offering resistance against Differential Power Analysis of arbitrary order when used for authenticated encryption/decryption with associated data. This protection did not extend to the keyed hash modes, such as that used in HMAC.

Selected protected hardware designs were evaluated by six Side-Channel Security Evaluation Labs. These evaluations led to the detection of some minor implementation errors. Most of these errors were fixed during or shortly after the time devoted to the evaluation of hardware implementations. Consequently, it was determined that all of these implementations could be fairly benchmarked and compared with one another, assuming the protection levels claimed by their authors. The assumption was made that any potential SCA security fixes, even if performed after the benchmarking process, would have a negligible effect on the absolute values of performance metrics (such as the throughput in Mbits/s and area in LUTs), not the mention the ranking of candidates.

Hardware benchmarking was performed by the GMU Team using a popular FPGA family, Xilinx Artix-7. The following major conclusions were reached. Overall, Xoodyak and Ascon performed the best for the majority of implementation categories (such as unprotected, 1st-order protected, 2nd-order protected, and 3rd-order protected designs) and the majority of possible input types (plaintext, associated data, hash message). These candidates offer high-speed, high throughput-to-area ratio, moderate randomness requirements for protected designs, and support for hashing. Ascon offers the added flexibility of choosing between two closely-related variants, Ascon-128 and Ascon-128a, and related hash functions Ascon-Hash and Ascon-Hasha.

Among the manually protected 1st-order designs available for Xoodyak (3), Ascon (1), and TinyJAMBU (1), the DOM implementation of Xoodyak was the best in terms of all performance metrics. However, there was no manually protected implementation of Ascon-128a, and among the automatically protected designs, Ascon-128a had the best throughput and throughput-to-area ratio for the processing of plaintext. TinyJAMBU was another candidate that clearly distinguished itself from other candidates. Most TinyJAMBU implementations were substantially smaller than those of other candidates for the same protection order. TinyJAMBU excelled in the throughput-to-area ratio, especially for the processing of AD and for processing of both plaintext and AD at the higher protection orders. The area of its implementations increased the least as a function of the protection order. For example, its 1st-order protected implementation is only about two times larger than the corresponding unprotected implementation. For the 3rd protection order, this ratio does not exceed 4. Additionally, TinyJAMBU was typically among the best candidates in terms of the moderate randomness requirements of its masked implementations. Drawbacks include no support for hashing and a relatively small throughput for processing long plaintext (as compared to Xoodyak and Ascon).

ISAP is unique with its two-pass design and mode-level, arbitrary-order protection against DPA. Assuming that this algorithm design provides the same (or higher) level of protection as the masked implementations of other candidates, ISAP ranks particularly high for the 2nd and 3rd protection order. For these orders, it ranks number 1 for the throughput-to-area ratio for plaintext and throughput and throughput-to-area ratio for AD. However, the mode-level algorithm design does not fully protect against simple power analysis. It also does not provide protection in the keyed hash mode, such as that used in HMAC.

When masking is applied, ISAP is typically outperformed by both Xoodyak and Ascon. Additionally, a two-pass authenticated encryption creates its own implementation challenges, such as the need for additional storage or at least sharing storage between the intermediate and final results. This sharing may potentially introduce additional security vulnerabilities.

The limitations of this study included a relatively small number of protected software implementations. The submitted software implementations covered only 5 out of 10 candidates. Additionally, the implementations of two candidates failed a basic leakage assessment test, and the mode-level protection of the third candidate (ISAP) could not be verified experimentally. Consequently, robust software implementations were developed only for Ascon and Xoodyak, confirming the large community interest in these candidates. The developers of the Ascon implementation (Ascon Team) claimed the second protection order. They also developed 6 variants of the implementation. The authors of the Xoodyak implementation (Hardware Security and Cryptographic Processor Lab at Tsinghua University) claimed the first protection order. Their submission included a single variant. The implementations of Ascon were evaluated by two independent labs and passed all leakage assessment tests and attack attempts. The protected software implementation of Xoodyak was not evaluated by any independent lab. None of the labs volunteered to perform independent benchmarking of protected software implementations. Thus, the study of protected software implementations primarily confirmed the community trust in Ascon and the difficulty of developing and evaluating SCA-protected implementations for other candidates developed later than Ascon.

In terms of the qualitative analysis reported in [23] and earlier publications from the same group, we believe that this evaluation is extremely valuable, especially for higher protection orders that cannot be practically assessed experimentally. The qualitative and quantitative analyses complement each other and should be both carefully considered when choosing new cryptographic standards and developing their SCA-secure implementations.

References

[1] NIST, "Submission Requirements and Evaluation Criteria for the Lightweight Cryptography Standardization Process," Tech. Rep., Aug. 2018.

[2] N. Müller and A. Moradi, "PROLEAD: A Probing-Based Hardware Leakage Detection Tool," *TCHES*, vol. 2022, no. 4, pp. 311–348, Aug. 31, 2022.

[3] 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.

[4] G. Becker, J. Cooper, E. DeMulder, *et al.*, "Test Vector Leakage Assessment (TVLA) methodology in practice," in *International Cryptographic Module Conference*, 2013, p. 13.

[5] T. Schneider and A. Moradi, "Leakage Assessment Methodology: A Clear Roadmap for Side-Channel Evaluations," in *Cryptographic Hardware and Embedded Systems – CHES 2015*, T. Güneysu and H. Handschuh, Eds., ser. Lecture Notes in Computer Science, Berlin, Heidelberg: Springer, 2015, pp. 495–513.

[6] A. A. Ding, C. Chen, and T. Eisenbarth, "Simpler, Faster, and More Robust T-Test Based Leakage Detection," in *Constructive Side-Channel Analysis and Secure Design*, F.-X. Standaert and E. Oswald, Eds., ser. Lecture Notes in Computer Science, Cham: Springer International Publishing, 2016, pp. 163–183.

[7] A. Moradi, *How to Evaluate Side-Channel Leakages*, Workshop Talk, Bochum, Germany, Jun. 2017.

[8] F.-X. Standaert, “How (Not) to Use Welch’s T-Test in Side-Channel Security Evaluations,” in *Smart Card Research and Advanced Applications*, ser. Lecture Notes in Computer Science, Cham: Springer International Publishing, 2019, pp. 65–79.

[9] A. Moradi, B. Richter, T. Schneider, and F.-X. Standaert, “Leakage Detection with the $X2$-Test,” *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 209–237, 2018.

[10] C. Whitnall, E. Oswald, C. Whitnall, and E. Oswald, “A Critical Analysis of ISO 17825 (‘Testing Methods for the Mitigation of Non-invasive Attack Classes Against Cryptographic Modules’),” in *Advances in Cryptology – ASIACRYPT 2019*, vol. 11923, Cham: Springer International Publishing, 2019, pp. 256–284.

[11] S. Bhasin, J.-L. Danger, S. Guilley, and Z. Najm, “NICV: Normalized inter-class variance for detection of side-channel leakage,” in *2014 International Symposium on Electromagnetic Compatibility*, Tokyo, Japan: IEEE, May 2014, p. 4.

[12] D. B. Roy, S. Bhasin, S. Guilley, A. Heuser, S. Patranabis, and D. Mukhopadhyay, “CC Meets FIPS: A Hybrid Test Methodology for First Order Side Channel Analysis,” *IEEE Transactions on Computers*, vol. 68, no. 3, pp. 347–361, Mar. 2019.

[13] T. Moos, F. Wegener, and A. Moradi, “DL-LA: Deep Learning Leakage Assessment: A modern roadmap for SCA evaluations,” *TCHES*, vol. 2021, no. 3, pp. 552–598, Jul. 2021.

[14] S. Faust, V. Grosso, S. Merino Del Pozo, C. Paglialonga, and F.-X. Standaert, “Composable Masking Schemes in the Presence of Physical Defaults & the Robust Probing Model,” *TCHES*, vol. 2018, no. 3, pp. 89–120, Aug. 14, 2018. (visited on 01/30/2022).

[15] M. Randolph and W. Diehl, “Power Side-Channel Attack Analysis: A Review of 20 Years of Study for the Layman,” *Cryptography*, vol. 4, no. 2, p. 15, May 2020.

[16] C. Dobraunig, M. Eichlseder, S. Mangard, *et al.*, *ISAP v2.0 Submission to NIST*, May 17, 2021. [Online]. Available: https://isap.iaik.tugraz.at/files/isapv20.pdf.

[17] D. Knichel, A. Moradi, N. Müller, and P. Sasdrich, “Automated Generation of Masked Hardware,” *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2022, no. 1, pp. 589–629, 2022.

[18] H. Gross, S. Mangard, and T. Korak, “Domain-Oriented Masking: Compact Masked Hardware Implementations with Arbitrary Protection Order,” in *Proceedings of the 2016 ACM Workshop on Theory of Implementation Security*, Vienna Austria: ACM, Oct. 24, 2016, pp. 3–3. (visited on 03/12/2022).

[19] S. Nikova, C. Rechberger, and V. Rijmen, “Threshold Implementations Against Side-Channel Attacks and Glitches,” in *Information and Communications Security, ICICS 2006*, ser. LNCS, vol. 4307, Springer Berlin Heidelberg, 2006, pp. 529–545. (visited on 06/30/2019).

[20] G. Cassiers, B. Grégoire, I. Levi, and F.-X. Standaert, “Hardware Private Circuits: From Trivial Composition to Full Verification,” *IEEE Transactions on Computers*, vol. 70, no. 10, pp. 1677–1690, Oct. 2021.

[21] V. Hadžić and R. Bloem, “CocoAlma: A Versatile Masking Verifier,” in *Formal Methods in Computer-Aided Design 2021, FMCAD 2021*, in collab. with T. Wien and T. Wien, TU Wien, Oct. 2021. (visited on 01/30/2022).

[22] C. O’Flynn, “A Framework for Embedded Hardware Security Analysis,” Ph.D. dissertation, Dalhousie University, Halifax, Nova Scotia, Jun. 2017.

[23] C. Verhamme, G. Cassiers, and F.-X. Standaert, “Analyzing the Leakage Resistance of the NIST’s Lightweight Crypto Competition’s Finalists,” in *Smart Card Research and Advanced Applications*, ser. Lecture Notes in Computer Science, Cham: Springer International Publishing, 2023, pp. 290–308.

[24] K. Mohajerani and R. Nagpal, *Xeda: Cross EDA Abstraction and Automation*, XedaHQ, Jan. 13, 2023. [Online]. Available: https://github.com/XedaHQ/xeda.