Hardware Benchmarking of Round 2 Candidates in the NIST Lightweight Cryptography Standardization Process

Kamyar Mohajerani, Richard Haeussler, Rishub Nagpal, Farnoud Farahmand, Abubakr Abdulgadir,
Jens-Peter Kaps, and Kris Gaj
Department of Electrical and Computer Engineering
George Mason University
Fairfax, VA, U.S.A.
{mmohajer, rhaeussl, rnagpal2, ffarahma, aabdulga, jkaps, kgaj}@gmu.edu

***Abstract*—Twenty five Round 2 candidates in the NIST Lightweight Cryptography (LWC) process have been implemented in hardware by groups from all over the world. All implementations compliant with the LWC Hardware API, proposed in 2019, have been submitted for hardware benchmarking to George Mason University's LWC benchmarking team. The received submissions were first verified for correct functionality and compliance with the hardware API's specification. Then, the execution times in clock cycles, as a function of input sizes, have been determined using behavioral simulation. The compatibility of all implementations with FPGA toolsets from three major vendors, Xilinx, Intel, and Lattice Semiconductor was verified. Optimized values of the maximum clock frequency and resource utilization metrics, such as the number of look-up tables (LUTs) and flip-flops (FFs), were obtained by running optimization tools, such as Minerva, ATHENa, and Xeda. The raw post-place and route results were then converted into values of the corresponding throughputs for long, medium-size, and short inputs. The results were presented in the form of easy to interpret graphs and tables, demonstrating the relative performance of all investigated algorithms. An effort was made to make the entire process as transparent as possible and results easily reproducible by other groups.**

***Index Terms*—Lightweight Cryptography, authenticated ciphers, hash functions, hardware, FPGA, benchmarking**

I. Introduction

A comprehensive framework for fair and efficient benchmarking of hardware implementations of lightweight cryptography was proposed in [1]. This framework was based on the idea of the Lightweight Cryptography Hardware API [2], which was published in October 2019, and has remained stable since then.

The corresponding LWC Development Package has been built as a major revision of the CAESAR Development Package [3], [4] by an extended team including representatives of the Technical University of Munich (TUM), Virginia Tech, and George Mason University. The first version of this package was published on October 14, 2019. Since then, this package was updated several times, including the most recent revision in October 2020. The first implementations of candidates in the Lightweight Cryptography Standardization process, compliant with the LWC Hardware API and using the new development package, were reported by members of the Virginia Tech Signatures Analysis Lab in [5].

Before the start of Round 2 of the NIST Lightweight Cryptography Standardization Process in September 2019, multiple submission teams developed hardware implementations non-compliant with the proposed LWC API. These implementations used very divergent assumptions, interfaces, and optimization goals. Only 8 out 32 teams (ACE, DryGASCON, ForkAE, ISAP, Romulus, SKINNY, Subterranean 2.0, and WAGE) made their HDL code public, either as a part of the corresponding Round 2 submission package or the candidate website. Preliminary results reported in the algorithm specifications were based on the use of about a dozen different FPGA families and about the same number of standard-cell ASIC libraries. Only results obtained using the same FPGA family or the same ASIC library can be fairly compared with one another. As a result, before the start of this benchmarking effort, at most 6 FPGA implementations and 4 ASIC implementations could possibly be compared with one another. However, even such a limited comparison would be highly unfair because of the use of different interfaces, assumptions, and optimization targets.

II. Methodology

A. LWC Hardware API

Hardware designers participating in the hardware benchmarking of Round 2 LWC candidates are expected to follow Hardware API for Lightweight Cryptography defined in detail at [2]. The major parts of this API include the minimum compliance criteria, interface, and communication protocol supported by the LWC core. The proposed API is intended to meet the requirements of all candidates submitted to the NIST Lightweight Cryptography standardization process, as well as all CAESAR candidates and current authenticated cipher and hash function standards. The main reasons for defining a common API for all hardware implementations of candidates submitted to the NIST Lightweight Cryptography standardization project are: a) Fairness of benchmarking, b) Compatibility

Partially supported by the US Department of Commerce (NIST) using grant no. 70NANB18H219.

among implementations of the same algorithm by different designers, and c) Ease of creating the supporting development package, aimed at simplifying and speeding up the design process.

B. LWC Hardware Development Package

To make the benchmarking framework more efficient in terms of the hardware development time, the designers are provided with the following resources, compliant with the use of the proposed LWC Hardware API:
a) VHDL code supporting the API protocol, common to all Lightweight Cryptography standardization process candidates, as well as all CAESAR candidates and AES-GCM (LWC_rtl)
b) Universal testbench, common for all API-compliant designs (LWC_TB)
c) Python app used to automatically generate test vectors (cryptotvgen)
d) Reference implementations of a dummy authenticated cipher and a dummy hash function (dummy_lwc)
e) Implementer's Guide, describing all steps of the development and benchmarking process, including verification, experimental testing, and generation of results.

It should be stressed that the *implementations of authenticated ciphers (with an optional hash functionality), compliant with the LWC Hardware API, can also be developed without using any of the aforementioned resources, by just following the specification of the LWC Hardware API directly.*

C. FPGA Platforms and Tools

For the purpose of this benchmarking study, the GMU group selected three benchmarking platforms representing FPGA families of three major vendors: Xilinx, Intel, and Lattice Semiconductor. The primary criteria for the selection of FPGA devices were as follows:

1) representing widely used low-cost, low-power FPGA families
2) capable of holding SCA-protected designs (possibly using up to four times more resources than unprotected designs)
3) supported by free versions of state-of-the-art industry tools.

These criteria led to the selection of the following FPGA devices:

1) From Xilinx
   Artix-7 : xc7a12tcsg325-3, including 8,000 LUTs, 16,000 FFs, 40 18Kbit BRAMs, 40 DSPs, and 150 I/Os.
2) From Intel
   Cyclone 10 LP : 10CL016-YF484C6, including 15,408 LEs, 15,408 FFs, 56 M9K blocks, 56 multipliers (MULs), and 162 I/Os, and
3) From Lattice Semiconductor
   ECP5 : LFE5U-25F-6BG381C, including 24,000 LUTs, 24,000 FFs, 56 18Kbit blocks, 28 MULs, and 197 I/Os.

The corresponding FPGA tools capable of processing HDL code targeting these (and many other FPGA devices) were:

1) From Xilinx: Xilinx Vivado 2020.1 (lin64)
2) From Intel: Intel Quartus Prime Lite Edition Design Software, ver. 20.1
3) From Lattice Semiconductor: Lattice Diamond Software v3.11 SP2.

D. Optimization Target

Our underlying assumption is that the implementation of an LWC algorithm *protected against side-channel attacks* should take no more than all look-up tables (LUTs) of the selected Xilinx FPGA device, Artix-7 : xc7a12tcsg325-3. Taking into account that protected implementations typically take up to 3-4 times more LUTs than unprotected implementations, our unprotected design should take no more than one-fourth of the total number of LUTs, i.e., 2000 LUTs. At the same time, we assume that the benchmarked implementations are not permitted to use any family-specific embedded resources, such as Block RAMs, DSP units, or embedded multipliers. Any storage should be implemented using either flip-flops or distributed memory, which, in the case of Xilinx FPGAs, is built out of LUTs. The number of Artix-7 flip-flops is limited to 4000, as in this FPGA family, each LUT is accompanied by two flip-flops. The designs are also prohibited from using any family-specific primitives or megafunctions.

This proposed optimization target has been clearly communicated to all LWC submission teams through the document titled Suggested FPGA Design Goals, posted on the LWC hardware benchmarking project website [2], as well as announcements on the lwc-forum, and private communication.

E. Functional Verification

All submitted implementations were first investigated in terms of compliance with the LWC Hardware API and the completeness of their deliverables, requested for benchmarking. Then, a comprehensive set of new test vectors, unknown in advance to hardware designers, was generated separately for each variant of each algorithm. These tests included multiple special cases, such as empty AD, empty plaintext, various widths of an incomplete last block, etc. If these test vectors passed, the implementation was judged functionally correct and compliant with the LWC Hardware API. If these test vectors failed, the source of failure was investigated in close collaboration with hardware designers. The designers were allowed to submit revised versions of their code.

F. Timing Measurements

The testbench LWC_TB, being a part of the LWC Development package, has been extended to include support for measurements of the execution times for authenticated encryption, authenticated decryption, and hashing. This testbench was used to measure the execution times for:

1) Input sizes used in the definitions of benchmarking metrics
2) All possible AD and plaintext lengths (in bytes) between 0 and 2 full input blocks, in increments of one byte.

Only the execution times obtained experimentally, using the timing measurements, were used to calculate values of the corresponding throughputs.

G. Synthesis, Implementation, and Optimization of Tool Options

The determination of the maximum clock frequency and the corresponding resource utilization was performed using tools specific for each FPGA vendor. For Artix-7 FPGAs, Minerva: An Automated Hardware Optimization Tool, described in [6], was used. The average time required to find the optimum requested clock frequency and the best optimization strategy was about 3.5 hours per algorithm variant. For Intel FPGAs, ATHENa – Automated Tool for Hardware EvaluatioN [7], was used. This tool supports all recent Intel FPGA families as well as older Xilinx FPGA families before Series 7. A new tool, Xeda [8], which stands for cross (X) electronic design automation, was developed. Xeda provides a layer of abstraction over simulation and synthesis tools and removes the difficulty associated with testing a design across multiple FPGA vendors. Additionally, Xeda allows user-made plugins, which can extend functionality to new tools or allow for post-processing of synthesis and simulation results. For Lattice Semiconductor FPGAs, Xeda and a plugin developed to find the maximum clock frequency were used. The synthesis was performed using both the Lattice Synthesis Engine (LSE) and Synplify Pro. Only the better of the two results was reported.

H. Performance Metrics

The following performance metrics have been evaluated as a part of this benchmarking project:

1) Resource utilization
   Number of LUTs for Artix-7 and ECP5 FPGAs, LEs for Cyclone 10 LP FPGAs, and flip-flops for all FPGAs, assuming no use of embedded memories (such as BRAMs), DSP units, and embedded multipliers.
2) Throughput in Mbits/s for the following sizes of inputs
   a) Long [with Throughput = $d \cdot$Block_size$/($Time$(N+d$ blocks$)-$ Time$(N$ blocks$))$]
   b) 1536, 64 bytes, and 16 bytes.

   All throughputs are calculated separately for plaintext (PT) only, Associated Data (AD) only, and hash message.

III. Hardware Designs

A total of 30 designs were received. These designs covered 25 out of 32 Round 2 candidates. Candidates implemented independently by two different groups included Ascon, COMET, Gimli, TinyJAMBU, and Xoodyak. The following submissions were provided by co-authors of algorithms submitted to the NIST LWC standardization process: ACE, ESTATE, ForkAE, Gimli, ISAP, KNOT, LOCUS-AEAD/LOTUS-AEAD, Oribatida, Romulus, Spook, Subterranean 2.0, TinyJAMBU, WAGE, and Xoodyak. The implementation of DryGASCON was developed by an independent researcher, Ekawat Homsirikamol, in close collaboration with the author of the algorithm. An additional implementation of Gimli was contributed by members of the Chair of Security in Information Technology at the Technical University of Munich, Germany.

Most groups used VHDL. Four design teams used exclusively Verilog for the implementation of the entire LWC unit. As a result, these implementations did not take advantage of the LWC Development Package, available only in VHDL. Algorithms implemented this way included Gimli, Romulus, Spook-v2, and Subterranean 2.0. The submission Xoodyak_GMU2-v1 is implemented purely in Bluespec SystemVerilog, depending on its own Bluespec LWC development package [9]. Three implementations modeled only the part unique to a given algorithm, its CryptoCore, in Verilog. These designs included DryGASCON, KNOT, and SpoC.

Ten submissions contained a single variant. In the remaining, the number of variants varied between 2 and 16, with an average of 2.7 per hardware design submission. Most of the variants of the same algorithm share a significant portion of the HDL source code and differ only in values of generics or constants. In some cases, a separate source code was provided for each variant.

The total number of implemented variants reached 88. We assigned to each variant a unique name. For algorithms implemented by a single group, this name consists of the name of the algorithm followed by a variant number. For algorithms implemented by two groups we add a group name abbreviation after the algorithm name. The abbreviations used are: CI for CINVESTAV-IPN, GMU for George Mason University, Graz for TU Graz, Austria, GT for Gimli Team, VT for Virginia Tech, TJT for TinyJAMBU Team, and XT for Xoodyak Team + Silvia. For Spook, exceptionally, the name of the variant is Spook-v2-v1. In this name, v2 indicates version 2 of Spook proposed in [10]. Features of all variants are summarized in [11].

For almost all candidates, decryption can be performed with exactly the same speed as encryption. As a result, in the Results section, we focus only on the timing metrics related to encryption. The following candidates process AD significantly faster than plaintext: Xoodyak, TinyJAMBU, ESTATE, LOCUS & LOTUS, Oribatida, and Romulus. The ratio of the hashing throughput to the plaintext processing throughput is 2.00 for Saturnin, 1.00 for ACE, DryGASCON, and Gimli, and the smallest for KNOT, PHOTON-Beetle, and Subterranean 2.0.

IV. Results and Their Analysis

A. Throughputs for Long Inputs

The two-dimensional graphs Throughput vs. Number of Used LUTs are shown in Figs. 1 and 2. The throughputs concern the cases of Plaintext (PT) only and Associated Data (AD) only, respectively. Both graphs present results for the Xilinx Artix-7 FPGA xc7a12tcsg325-3. The results apply to long inputs. We use the logarithmic scale on both axes. Dashed lines represent the same throughput over area ratio. In the legends of these figures, the algorithms are listed in the order of decreasing throughput. While the order of the symbols remains the same, the mapping of symbols to algorithms changes.

In these graphs, each candidate is represented by only one variant, selected according to the following rules. If a candidate has one or more variants with the area below 2520 LUTs (the area of the smallest implementation of AES-GCM available to us), the fastest variant meeting this criterion is selected. If a

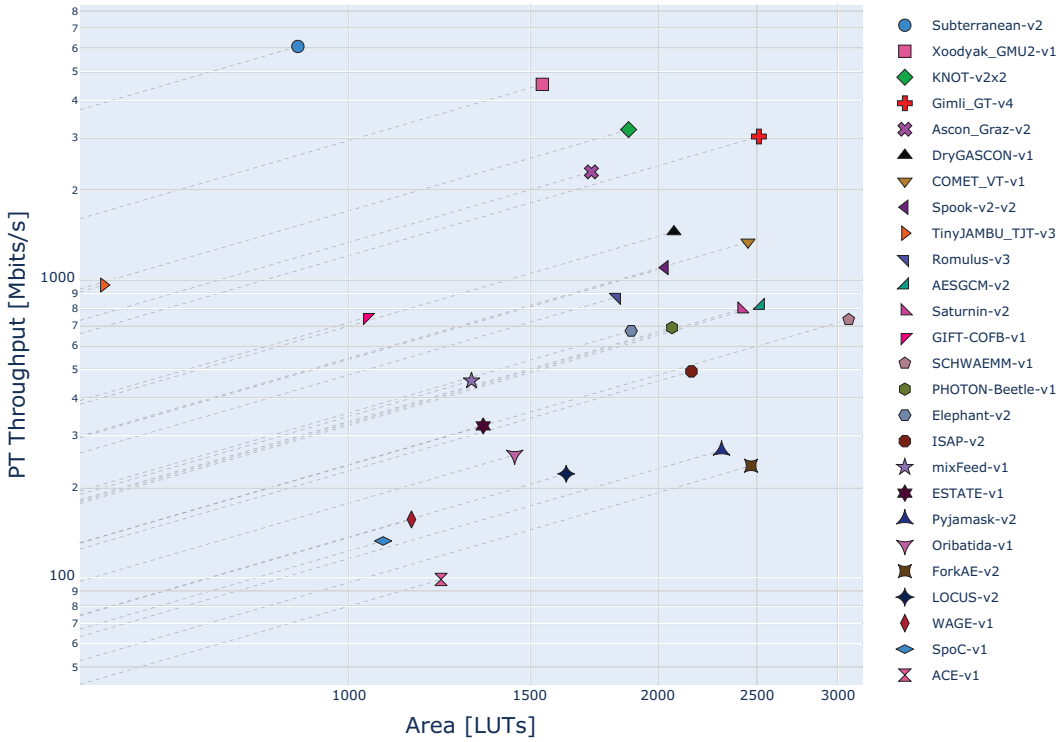

Fig. 1: Artix-7 Encryption PT Throughput for Long Messages vs LUTs

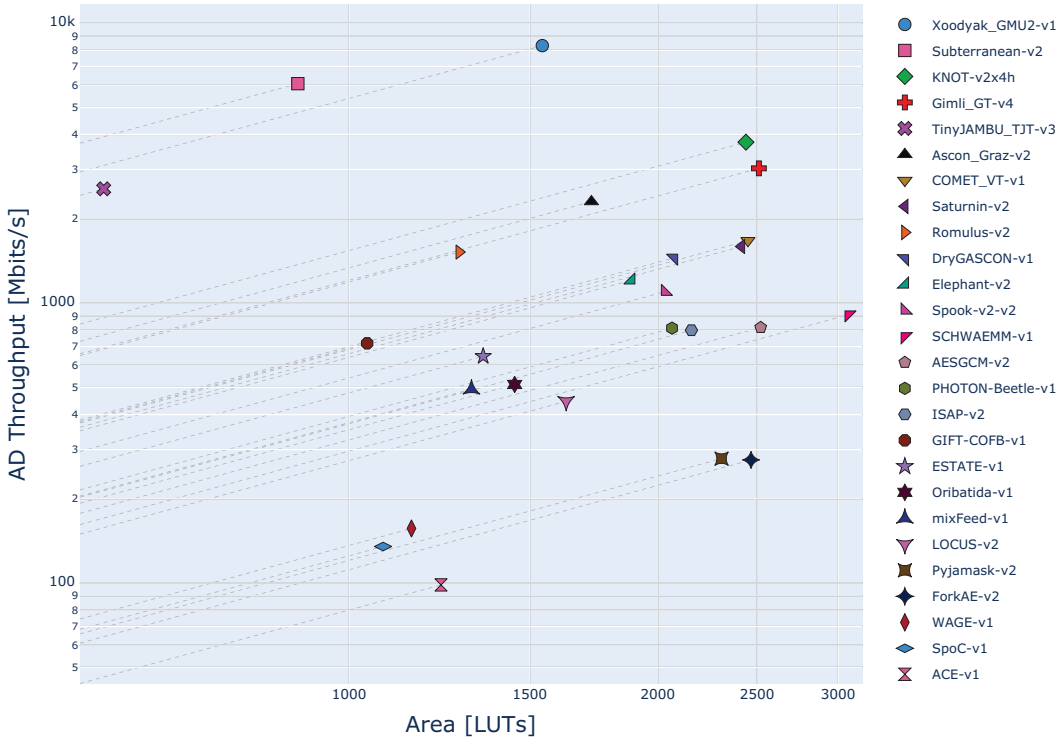

Fig. 2: Artix-7 Encryption AD Throughput for Long Messages vs LUTs

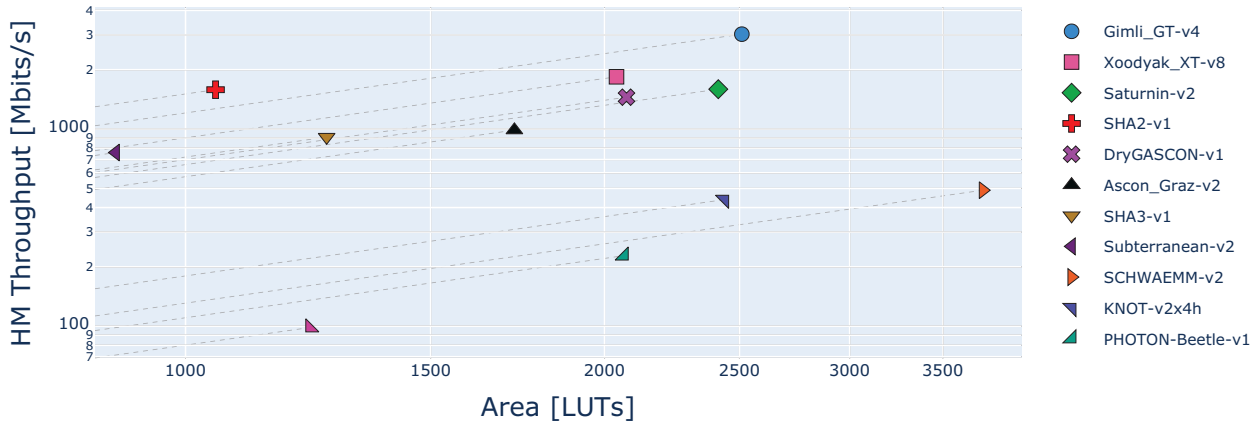

Fig. 3: Artix-7 Hashing Throughput for Long Messages vs LUTs

candidate does not have a variant with the area below 2520 LUTs, a variant with the smallest area is selected.

The threshold of 2520 LUTs (26% more than the intended target of 2000 LUTs) was selected because many designers tried to aggressively use close to 2000 LUTs to achieve the highest possible speed. As a result many of them ended up with designs taking between 2000 and 2520 LUTs. Additionally, the exact number of LUTs may depend on the exact options of tools, providing different trade-offs between the area and speed.

The winner for the PT only is Subterranean 2.0. Its implementation reaches the throughput of 6 Gbit/s and is the second smallest in terms of the number of LUTs. The second fastest is Xoodyak, with the throughput about 4.5 Gbit/s.

The next group includes three algorithms, KNOT, Gimli, and Ascon, with the throughputs between 2.3 and 3.2 Gbits/s. Out of these three, the implementation of KNOT is the fastest and the implementation of Ascon the smallest. The third group includes three algorithms with throughputs between 1 and 2 Gbits/s: DryGASCON, COMET, and Spook-v2. Their areas are in the range between 2000 and 2500 LUTs. The implementation of DryGASCON is the fastest, and the implementation of Spook-v2 the smallest in this group. The next algorithm in the ranking is TinyJAMBU, which reaches a speed very close to 1 Gbit/s and at the same time has by far the smallest area, around 600 LUTs. The first dozen candidates in terms of throughput, with the area below 2520 LUTs, also include Romulus, Saturnin, and GIFT-COFB. The design of SCHWAEMM is by far the largest, above 3000 LUTs, yet still only average (rank 13) in terms of throughput. More effort is required to demonstrate the competitiveness of this algorithm with the first 12 candidates mentioned above. All remaining algorithms have throughputs below 700 Mbits/s. Out of them, ISAP, Pyjamask, and ForkAE already have areas exceeding 2000 LUTs.

For AD only, the following changes in the rankings are the most significant. Subterranean 2.0 and Xoodyak swap their positions. Xoodyak is the fastest, with a speed exceeding 8 Gbit/s. The next group includes KNOT and Gimli, with the throughputs around 4 Gbit/s and 3 Gbits/s, respectively. TinyJAMBU moves from position 9 for processing plaintext only to position 5 for AD only, followed closely by Ascon at position 6. The new algorithms with throughputs in the range between 1 and 2 Gbit/s include Saturnin, Romulus, and Elephant. Among the first dozen algorithms in the ranking, there is only one change, GIFT-COFB is replaced by Elephant. All first 12 algorithms have throughputs for AD only greater than 1 Gbit/s.

Only 10 out of 25 investigated candidates support hashing. The two-dimensional graph, Throughput vs. Area for hashing long messages on Artix-7 FPGA, is shown in Fig. 3.

The two fastest designs are Gimli and Xoodyak, with throughputs approximately equal to 3 and 2 Gbits/s, respectively. Very close behind are Saturnin and DryGASCON, with the throughputs between 1.3 and 1.5 Gbits/s. They are followed by Ascon at about 1 Gbit/s and Subterranean at around 750 Mbits/s. SCHWAEMM (ESCH) reaches slightly less than 500 Mbit/s. The three remaining algorithms, KNOT, PHOTON-Beetle, and ACE have throughputs below 450 Mbits/s.

The average ratios of the number of Cyclone 10 LP LEs and ECP5 LUTs to the number of Artix-7 LUTs, calculated over all major designs, are 1.9 and 1.7, respectively. The average decrease in the frequency is by a factor of 1.6 for Cyclone 10 LP and 2.6 for ECP5. The ranking of candidates remains approximately the same [11].

B. *Throughputs for Short Inputs*

For 1536-byte plaintexts, the throughputs are very close to throughputs for long inputs. The average percentage is 97%, the minimum 89% (Subterranean-v1). Multiple algorithms reach 99%. For 64-byte plaintexts, this ratio varies from 25% for Subterranean-v2 to 99% for ForkAE-v1, with an average of 61%. For 16-byte plaintexts, the ratio varies from 8% for Subterranean-v1 to 98% for ForkAE-v1, with an average of 32%. All mentioned above percentages are dependent only on the algorithm and its hardware architecture. They do not depend on a particular FPGA device.

In Table I, we summarize the relative changes in rankings for Artix-7. For the processing of PT only, the following algorithms rank higher for short messages than for long

TABLE I: Xilinx Artix-7 Encryption PT Throughput Rankings

Rank	1536 Byte	64 Byte	16 Byte
1	Subterranean-v2	Xoodyak_GMU2-v1	Xoodyak_GMU2-v1
2	Xoodyak_GMU2-v1	Subterranean-v2	Subterranean-v2
3	KNOT-v2x2	KNOT-v2x2	Ascon_VT-v1
4	Gimli_GT-v4	Ascon_Graz-v2	COMET_VT-v1
5	Ascon_Graz-v2	DryGASCON-v1	DryGASCON-v1
6	DryGASCON-v1	Gimli_GT-v4	KNOT-v2x2
7	COMET_VT-v1	COMET_VT-v1	TinyJAMBU_TJT-v3
8	Spook-v2-v2	TinyJAMBU_TJT-v3	Romulus-v2
9	TinyJAMBU_TJT-v3	Romulus-v2	Gimli_GT-v4
10	Romulus-v3	Spook-v2-v2	PHOTON-Beetle-v1
11	Saturnin-v2	PHOTON-Beetle-v1	Elephant-v2
12	GIFT-COFB-v1	GIFT-COFB-v1	GIFT-COFB-v1
13	SCHWAEMM-v1	Elephant-v2	ESTATE-v1
14	PHOTON-Beetle-v1	SCHWAEMM-v1	Spook-v2-v2
15	Elephant-v2	Saturnin-v2	ForkAE-v2
16	ISAP-v2	ESTATE-v1	SCHWAEMM-v1
17	mixFeed-v1	mixFeed-v1	Oribatida-v1
18	ESTATE-v1	ForkAE-v2	LOCUS-v2
19	Pyjamask-v2	Oribatida-v1	Saturnin-v2
20	Oribatida-v1	LOCUS-v2	mixFeed-v1
21	ForkAE-v2	ISAP-v2	SpoC-v1
22	LOCUS-v2	Pyjamask-v2	ISAP-v2
23	WAGE-v1	SpoC-v1	Pyjamask-v2
24	SpoC-v1	WAGE-v1	WAGE-v1
25	ACE-v1	ACE-v1	ACE-v1

messages: Xoodyak, Ascon, DryGASCON, COMET, TinyJAMBU, Romulus, PHOTON-Beetle, Elephant, ESTATE, Oribatida, ForkAE, LOCUS, and SpoC. The opposite is true for the following candidates: Subterranean 2.0, KNOT, Gimli, Spook, SCHWAEMM, Saturnin, ISAP, Pyjamask, and WAGE. The following 9 algorithms remain among the best 10, independently of the size of inputs: Subterranean 2.0, Xoodyak, KNOT, Gimli, Ascon, DryGASCON, COMET, TinyJAMBU, and Romulus. For the shortest considered plaintext of the size of 16 bytes, Spook-v2 drops to position 14. Out of these 9 algorithms, the following 6 also support hashing: Gimli, Xoodyak, DryGASCON, Ascon, Subterranean 2.0, and KNOT (with the first four at least two times faster than KNOT). A candidate particularly fast in hashing but not so good for processing small plaintexts is Saturnin. Details of all results are available in [11].

V. Conclusions and Future Work

For processing of long plaintexts, with the budget of 2520 Artix-7 LUTs or less, 10 candidates outperform the current standard AES-GCM. These candidates, in the order of Throughput, include: Subterranean 2.0, Xoodyak, KNOT, Gimli, Ascon, DryGASCON, COMET, Spook-v2, TinyJAMBU, and Romulus. All these algorithms, as well as Saturnin and Elephant, outperform AES-GCM also for processing of long ADs, while meeting the area limit. Out of them, only Gimli, Xoodyak, and Saturnin support hashing faster than SHA-2. Two additional ones, DryGASCON and Ascon, perform hashing faster than the folded implementation of SHA-3. Future work will include ASIC benchmarking and energy per bit evaluation in FPGAs and ASICs.

References

[1] J.-P. Kaps, W. Diehl, M. Tempelmeier, F. Farahmand, E. Homsirikamol, and K. Gaj, "A Comprehensive Framework for Fair and Efficient Benchmarking of Hardware Implementations," Cryptology ePrint Archive 2019/1273, Nov. 2019.

[2] Cryptographic Engineering Research Group (CERG) at George Mason University. (2020). "Hardware Benchmarking of Lightweight Cryptography," [Online]. Available: https://cryptography.gmu.edu/athena/index.php?id=LWC.

[3] ——, (2019). "Hardware Benchmarking of CAESAR Candidates," [Online]. Available: https://cryptography.gmu.edu/athena/index.php?id=CAESAR.

[4] P. Yalla and J.-P. Kaps, "Evaluation of the CAESAR hardware API for lightweight implementations," in *2017 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, Cancun: IEEE, Dec. 2017.

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

[6] F. Farahmand, W. Diehl, and K. Gaj, "Minerva: Automated hardware optimization tool," in *International Conference on ReConfigureable Computing and FPGAs (ReConfig)*, Cancun, Mexico, 2017, pp. 1–8.

[7] K. Gaj, J.-P. Kaps, V. Amirineni, M. Rogawski, E. Homsirikamol, and B. Y. Brewster, "ATHENa - Automated Tool for Hardware EvaluatioN: Toward Fair and Comprehensive Benchmarking of Cryptographic Hardware Using FPGAs," in *2010 International Conference on Field Programmable Logic and Applications, FPL 2010*, Milan, Italy: IEEE, Aug. 2010, pp. 414–421.

[8] K. Mohajerani and R. Nagpal, *Xeda: Cross-EDA Abstraction and Automation*, Dec. 9, 2020. [Online]. Available: https://github.com/XedaHQ/xeda.

[9] K. Mohajerani, *BlueLight: Bluespec implementations of Lightweight Cryptography Candidates*, Dec. 9, 2020. [Online]. Available: https://github.com/kammoh/bluelight.

[10] D. Bellizia, F. Berti, O. Bronchain, G. Cassiers, S. Duval, C. Guo, G. Leander, G. Leurent, C. Momin, O. Pereira, T. Peters, F.-X. Standaert, B. Udvarhelyi, and F. Wiemer, "Spook: Sponge-Based Leakage-Resistant Authenticated Encryption with a Masked Tweakable Block Cipher," *IACR Transactions on Symmetric Cryptology*, vol. 2020, no. S1, pp. 295–349, 2020.

[11] K. Mohajerani, R. Haeussler, R. Nagpal, F. Farahmand, A. Abdulgadir, J.-P. Kaps, and K. Gaj, *FPGA Benchmarking of Round 2 Candidates in the NIST Lightweight Cryptography Standardization Process: Methodology, Metrics, Tools, and Results*, Cryptology ePrint Archive, Report 2020/1207, 2020. [Online]. Available: https://eprint.iacr.org/2020/1207.