Benchmarking Software Implementations of 1st Round Candidates of the NIST LWC Project on Microcontrollers

Sebastian Renner[1,2] Enrico Pozzobon[1] Jürgen Mottok[1]

{sebastian1.renner, enrico.pozzobon, juergen.mottok}@othr.de

[1]OTH Regensburg
[2]Technical University of Munich

Abstract

This paper introduces a custom framework for benchmarking software implementations from the National Institute of Standards and Technology (NIST) Lightweight Cryptography (LWC) project on embedded devices. We present the design and core functions of the framework and apply it to various NIST LWC authenticated encryption with associated data (AEAD) ciphers. Altogether, we evaluate the speed of 213 submitted algorithm variants on four different microcontroller units (MCUs), including 32 bit ARM and 8 bit AVR architectures. To allow a more meaningful comparison, we also conduct code size tests on all four boards and RAM utilization tests on one test platform.

1 Introduction

In the era of rising numbers of interconnected computing devices and highly frequent cyber attacks, an increased need for secure communication exists. Standard cryptosystems often cannot be applied in areas like sensor networks, since the here used devices typically consist of low-performance hardware components. To aid in the process of development, evaluation and standardization of suitable lightweight cryptography algorithms, the NIST has initiated the Lightweight Cryptography Project with the final goal to standardize lightweight hash functions and cryptosystems, which support authenticated encryption with associated data. NIST received and published 56 algorithm proposals, which include more than 200 AEAD cipher implementation variants.

In this paper, we focus on benchmarking speed, ROM and RAM usage of software implementations of at least one variant of each cipher on different mi-

crocontroller platforms. The goal of this work is to provide a broad overview over the ciphers' "weight". The following section will describe related work in the field of performance evaluations of cryptosystems. In section 3, we present a custom NIST LWC benchmarking framework and its features. Furthermore, the test setup, test cases and evaluated MCUs are shown. Section 4 introduces the benchmarking results, before we conclude the evaluation outcome in section 5 and end with a paragraph on possible future work.

2 Related Work

This work is about software performance analysis of the NIST LWC project candidates. Ankele et al. published software benchmarks of 2nd round submissions of the CAESAR AEAD competition on Intel desktop processors [1][2]. Cazorla et al. compared implementations of 17 block ciphers on a 16 bit MCU from Texas Instruments [3]. Similar research was conducted by Hyncica et al. in 2011. They evaluate 15 symmetric cryptographic primitives regarding throughput, code size and storage utilization on three different embedded platforms [4]. Tschofenig et al. analyzed the performance of cryptographic algorithms, also on MCUs. Their work focuses on asymmetric elliptic curve ciphers executed on ARM Cortex-M cores [5]. An evaluation of 19 block and stream ciphers was published by Dinu et al. in 2015. A previous paper written by the same authors, introduces a benchmark framework for cryptographic ciphers, which focuses on fair performance testing [6] [7]. The frameworks eBacs and SUPERCOP are additional examples for popular software written for evaluating implementations of cryptographic algorithms [8]. Built to extend SUPERCOP, XBX and XXBX enhance the testing framework to support the evaluation of hash functions and AEAD ciphers on embedded devices [9] [10].

The research presented in this paper focuses on the evaluation of 1st round candidates of the NIST LWC project. The software implementations are benchmarked on multiple different MCU platforms and architectures. We test the performance (speed), RAM and ROM utilizations for each one of the 56 submitted ciphers and provide results for 213 software variants obtained on four different evaluation boards for the first test case. Furthermore, we introduce a flexible and highly automated test framework and methodology for evaluating the NIST LWC ciphers.

3 Methodology

The NIST stated the delivery of a software implementation to be mandatory for each submitted AEAD cipher in its call for submissions. Besides requirements concerning the cryptographic primitive itself, the set of guidelines included some formal regulations. For example, the static directory structure within submissions and the use of a predefined software Application Programming Interface (API) for cryptographic functions are mentioned. Before developing

the methodology and test procedures for the software benchmarks, an analysis of these formal requirements was conducted. The goal was to extract the basic guidelines for the creation of a test setup, which is completely compliant to the defines of NIST and yet flexible in terms of expandability.

3.1 Framework

After reviewing existing performance benchmark frameworks for AEAD ciphers, a decision was made towards the development of a custom test tool. That was because our focus regarding the hardware architecture was set on various instruction sets, typically found on microcontrollers. Since an intensive study of an existing framework and probably programming a manual extension would have been necessary to execute our test cases on the selected MCUs, the decision to built test routines from scratch was considered to be more suitable in our case.

Our framework consists of a couple of C, Python and Bash scripts, which are communicating with each other in a mostly automated manner. The `compile_all.py` script is responsible for compiling each cipher implementation available from the NIST website for the target platform. Note, that our routine always tries to compile each submitted cipher (variant) as it was provided in the ZIP file; no changes are made to the reference implementation. `compile_all.py` fetches the source files of the `crypto_aead` directories and merges them into the target template structure one after the other. The MCU-specific template implements a basic runtime environment and utilizes the NIST API when calling the encryption/decryption functions. The `compile_all.py` script merges the sources of each cipher into a fitting C(++)/Arduino template depending on the desired target. For each cipher, it outputs binary files ready to be flashed onto the MCU. Moreover, a `test_all.sh` file is generated, which can later be used to start the performance benchmarks.

After the compilation has terminated, calling `test_all.sh` starts the performance benchmark for each successfully compiled implementation on the MCU under test. For each cipher test, the generic benchmark script `test.py` is called. `test.py` is responsible for preparing the message buffers and coordinating the execution of the core encrypt/decrypt functions. The script implements a small message passing protocol which standardizes the communication in between the test software and the target board. `test.py` represents the generic test wrapper to be used in each performance benchmark, independent from the target platform. It sends the test vector input data to `adapter.py` via `stdin`. The `adapter.py` script is different for each MCU platform. It connects to the target via a serial interface and exchanges data with `test.py` over `stdin` and `stdout`. `adapter.py` can basically be seen as an abstract middleware, which receives and forwards data from both the host (over std*) and the target (over serial).

To perform the speed benchmarks of the NIST LWC submissions, the `test_all.sh` script needs to be executed. It contains one command line chunk for each successfully compiled cipher variant. Since the prepared binary images

for the target platform are of course unique for every algorithm, they need to be flashed and tested sequentially on the MCU. `test_all.sh` creates an instance of `test.py`, which gets tied to an instance of the `adapter.py` script. Moreover, the binary to be flashed and the corresponding test vectors are passed as command line arguments. With such a setup, fully automated flash and performance test procedures can be conducted. `test.py` first connects to `adapter.py`, which is then flashing the target firmware. Afterwards, the first test vector is sent from the host to a serial line connected to the device under test. In parallel, `test.py` starts up and configures an instance of the Saleae Logic measurement software. One GPIO pin and the reset line of the MCU are being monitored using a Saleae logic analyzer during the encryption/decryption routines and the reflashing of new firmware. The GPIO pin gets toggled before and after a cryptographic function is executed. `test.py` monitors the processing of each test vector and saves and closes the Logic capture upon the termination of the test of one cipher. `test_all.sh` then automatically starts the flash and test procedures for the next pre-compiled algorithm variant. The test results can later be interpreted by parsing the high and low states of the GPIO and reset pin, which were captured with the logic analyzer.

The architecture of the performance evaluation framework allows testing all compiled cipher variants in a completely automated manner. Once the compilation has finished, calling the `test_all.sh` scripts starts the sequential flashing, testing and result collection for each binary. Writing basic log files of the processing of the test vectors hereby ensures an easier traceability in case of communication or cryptographic errors. When all cipher variants for one target have been tested, the results can be interpreted and put into context using a parser script. The test procedure is the same for any supported MCU, only the template which interacts with the NIST API and the MCU's hardware and the `adapter.py` script need to be adjusted. The main test script remains unchanged regardless of the test platform and solely connects to the adapter script via stdin and stdout. The communication from `adapter.py` to the MCU could for example also be implemented using a non-serial interface like Ethernet, if it is available on the MCU under test and the execution time of the complete test procedure wants to be reduced. Moreover, the integration of new target devices requires little effort and no generic test routines need to be reconfigured – only a specific `adapter.py` and the runtime environment for the encrypt/decrypt functions on the MCU have to be provided.

The software design of the framework satisfies some common requirements regarding test automation. Test data is provided and collected through a standard interface, which communicates with exchangeable and modular scripts. Once the performance test has been started, no user intervention is necessary until all suitable cipher variants have been evaluated. Moreover, a basic logging functionality is included, continuous checks of the transmitted data ensure the recognition and reporting of communication errors.

To conlude the introduction to the test framework, Figure 1 visualizes its communication model and its previously described parts.

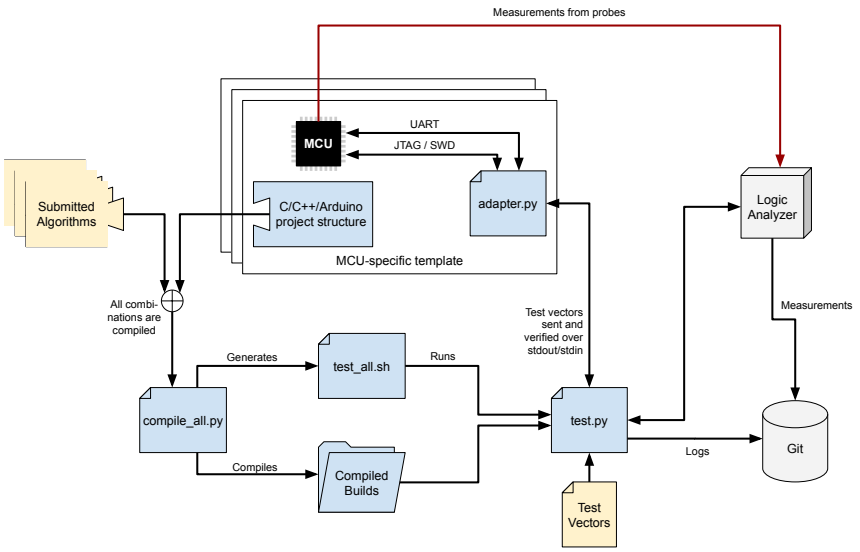

Figure 1: Core components and data flow of the test framework

3.2 Test Setup

Performing the speed benchmarks on the selected ciphers requires some hardware besides the actual MCU board. For our measurements, a Saleae logic analyser was used to probe the states of the necessary pins. In general, any other logic analyzer or an oscilloscope can be used to obtain this data. However, the automated test program starts the Saleae logic software, which expects a fitting logic analyzer connected via USB. Using a different device would require some modifications of the test software. As a flashing device, a SEGGER J-Link debugger is attached to the target MCU via a JTAG/SWD header. The J-Link has the advantage of supporting a broad range of MCUs, so a similar script for flashing different targets can be used. However, as with the logic analyzer, other hardware to flash the target could be appointed. To complete the hardware setup, a USB-to-serial cable needs to be attached to the configured UART pins in the MCU template. The 5V line from the USB connection can sometimes directly be used to power the test board. As mentioned in the previous section, the target-to-host communication can also be realized using a solution other than UART. This has to be planned and taken into account, when the template and the adapter script are developed for the MCU.

The appropriate software to compile and run the performance tests, including its underlying functions, concludes the test environment. Table 1 briefly shows the tools which have been used. The makefiles for the building of the MCU firmwares specify the recommended compiler flags from NIST.

Software type	Tool	Version
Compiler	arm-none-eabi-gcc	9.1.0
IDE	PlatformIO	4.0.0
IDE	STM32CubeMX	5.2.1
Interpreter	Python	3.7.3
Measurement utility	Saleae Logic	1.2.18
MCU flasher	SEGGER J-Link Commander	6.30

Table 1: Overview of used software tools

3.3 Test Cases

In this work, we introduce three different basic test cases, which are of relevance when assessing how lightweight a software implementation of a cipher is, the performance (speed), the size of the binary and the utilization of RAM. Of course, the test results of each cipher variant can be compared to its competitors within the NIST LWC project. However, we decided to include two more test candidates: A what we call *nocrypt* algorithm, which simply does a `memcpy` operation in its encrypt and decrypt functions and an implementation of one of the current state-of-the-art AEAD algorithms, AES-GCM. The results of the nocrypt benchmarks deliver an approximate minimal, respectively best case value for all test cases. This allows us to give a statement on for example the code size or execution time of the generated MCU template, even without integrating any cryptographic algorithm. AES-GCM implementations represent the state-of-the-art in the field of symmetric AEAD ciphers. It is a well-tested and standardized cipher. Comparing the candidates from the NIST LWC project to GCM shows how they perform against the actual standard in the different test cases. We stripped the GCM implementation inside mbed TLS to fit into our project environment and added wrapper encrypt/decrypt functions according to the NIST API. mbed TLS (formally known as PolarSSL) is part of the popular IoT operating system mbed OS and is compliant to NIST SP800-38D [11]. The flags MBEDTLS_AES_ROM_TABLES and MBEDTLS_AES_FEWER_TABLES were added to the configuration of mbed TLS since they are commonly used flags on embedded devices with a small amount of RAM and ROM. MBEDTLS_AES_ROM_TABLES places the SBOX and RCON tables and their inverses in the ROM instead of initializing them in the RAM on the first utilization of the AES algorithm. The MBEDTLS_AES_FEWER_TABLES reduces the binary size by avoiding the inclusion of some optimizations, bringing it closer to the one from other LWC entries.

When conducting the benchmarks, we aimed on including as many cipher variants per platform as possible. The reference implementations were treated with highest priority, however, also multiple variations of the cipher or optimized implementations have been included, when possible. Of course, an AVR-optimized program would not run on an ARM-based processor and vice versa, but on average more than one cipher variant has been tested on every test platform. In the performance test case, we ended up having results for 213

variants on at least one of the four test boards. Our tests refer to the code base available at NIST's LWC webpage in the beginning of July 2019. We did not change any of the submitted implementations, so every cipher can be judged solely on its reference code. To evaluate the performance of each cipher, the soft- and hardware setup was prepared as explained in the previous sections. The tests included processing the test vectors available in the submitted ZIP archive. The vectors for AES-GCM have been created using the `genkat_aead.c` file to ensure a fair evaluation.

The speed benchmark measures the time for the encryption and decryption of the message per test vector. If the vector contains associated data, its signing and verification is also taken into consideration. The time span is determined directly on the target, to avoid high latency, e.g. on the serial line. The logic analyzer gathers each encryption/decryption cycle from the GPIO pin toggle and saves the captured data to a text file upon the processing of the last test vector. The correct behavior of the cipher is checked by comparing the calculated plain- and ciphertext to the values in the test vector file. All measurements are later processed by a dedicated parser (`parse_logic.py`). This script does some basic checks to verify the input measurements and generates various textual reports and plots.

To compare the code size of the cipher variants, the GCM implementation and the nocrypt routine have also been included in the ROM usage test case. We integrated each implementation into the template sources and compiled a flashable binary for each cipher and test platform. The size of the nocrypt image can be seen as the minimal code size, when the template projects are applied. The compilation of each algorithm included the use of NIST's provided flags. After the `compile_all.py` script finished, the code size of the binaries was determined with a small bash script utilizing the `du` system command on Linux.

The volatile memory (SRAM) utilization was only measured on the STM32 F746ZG chip. To measure the RAM usage, the memory of the chip was filled with a known pattern, the test vectors were run, and the memory was dumped afterwards. By checking the differences between the memory dumps before and after the algorithm has been executed, it is possible to determine how many memory locations have been written during the execution of the encryption and decryption algorithms. The largest number of consecutive untouched memory locations between the end of the BSS segment and the beginning of the stack is considered the "unused memory". The number of additional bytes used by each algorithm when compared to the nocrypt implementation is considered to be the memory utilization of the examined algorithm.

3.4 Tested Platforms

The benchmarking framework currently supports four different platforms, featuring one 8 bit- and three 32 bit-MCUs and three different architectures. By choosing this initial set of supported boards, we aim to cover a wide range of microcontrollers, which are frequently used in IoT development. Also, the

afterwards described platforms are real-world low-cost-targets for the NIST LWC candidates. Directly providing templates for different architectures should show the simple expansion of the framework on the one hand. On the other hand, the diversity of the test platforms amplifies a fair evaluation of various cipher optimizations for low-performance MCUs. The following paragraphs introduce the key features of each test platform briefly.

Arduino Uno R3 The Arduino Uno features an 8 bit ATmega328P MCU from Atmel/Mircochip. The AVR-based controller has a clock speed of 16 MHz and provides 32 KB flash. The ATmega chip represents a simple low-end/low-cost processor, which is very popular in the community.

STM32F1 "bluepill" The "bluepill" or "blackpill" boards are cheap 32 bit evaluation platforms based on a STM32F103C8T6 MCU. The ARM Cortex-M3 core provides a clock frequency of 72 MHz and 64 KB of flash memory.

STM32 NUCLEO-F746ZG The F746ZG NUCLEO board is considered a high-power 32 bit device. It features 1 MB of flash memory and an ARM Cortex-M7 core which clocks at a frequency of up to 216 MHz. In contrast to the "bluepill", this chip is already better suited for more resource-intensive IoT products.

Espressif ESP32 WROOM The Espressif ESP32 WROOM evaluation kit is based on a dual-core 32 bit Xtensa LX6 MCU. With a maximum clock frequency of 240 MHz and a flash memory size of 4 MB, it is currently the most powerful platform supported in the test framework. The ESP32 and its predecessor ESP8266 are widely used for various IoT and automation projects.

4 Results

In this section, we provide visualizations of the test results. Since the data set for each test case contains measurements for 140-200+ ciphers, depending on the evaluation platform, extensive statistics are moved to the appendix section of this paper. The focus of the results paragraph is to provide a quick and brief overview over the best performing ciphers on each MCU and for each metric. To visualize the output of the performance (speed) measurements, as described in section 3, box plots of the test data of the ten average fastest cipher implementations per platform are shown (see figures 2, 4, 6 and 9). Note, that we also tried to obtain results for as many implementations as possible and the plots show the top ten cipher variants in ascending order, regardless of their relation to a specific candidate. That means, multiple variants of one candidate can be present in the ranking.

The results of the code size measurements are also presented per platform. As explained beforehand, the nocrypt "algorithm" is used as a reference size for the MCU template. To determine the actual needed storage for the cipher,

the reference size can be subtracted from the compiled flashable binary. Again, the ten best NIST LWC candidates are extracted into a diagram, this time in a simple bar plot (see figures 3, 5, 7 and 10).

The RAM utilization measurements have only been conducted on the STM32 F746ZG MCU (see figure 8). We expect similar results for this use case on every platform, that is why these tests have only been executed on one out of four chips. However, with this approach, we of course only obtain results for the cipher variants, that have been managed to be flashed onto the specific MCU.

In the following, we plot the average results for the respective top ten cipher variants for each test case and platform. To gather information about other implementations, see the provided table in the appendix. Due to various limitations, which can for example be linked to the cipher implementation itself, the (number of) test vectors or the hardware platform, not all variants could be evaluated. We have observed, that the code size of some algorithms exceeds the size of the flash memory on some of the tested boards. Moreover, optimized variants for specific Intel architectures or Single Instruction Multiple Data (Single Instruction Multiple Data) extensions can often not be supported on embedded platforms. Also, the reasons for the failure of a couple of test cases for some ciphers is still to be determined.

However, the vast majority of the available cipher implementations was evaluated and the full results are shown in the appendix.

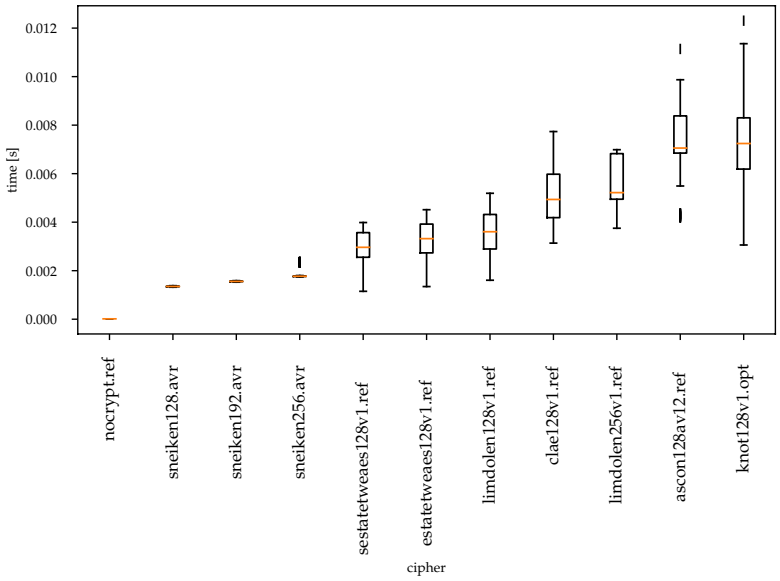

Figure 2: Ten fastest implementations on the Arduino Uno

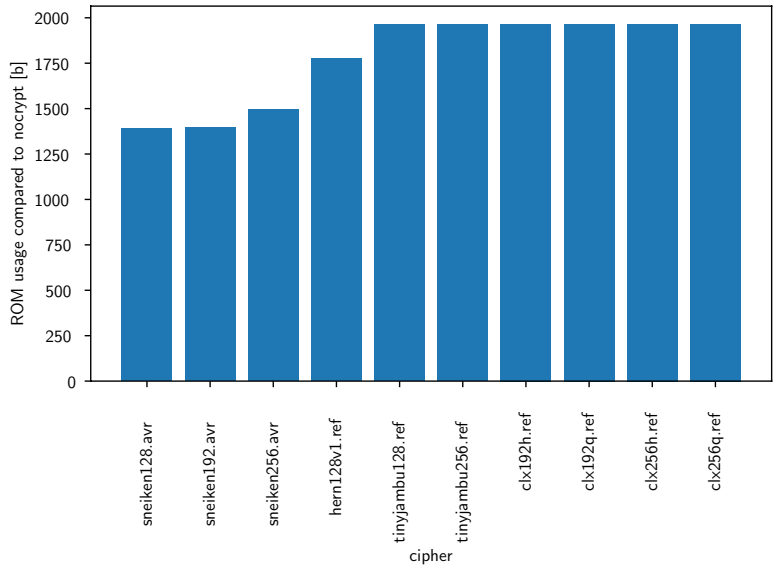

Figure 3: Ten smallest implementations on the Arduino Uno

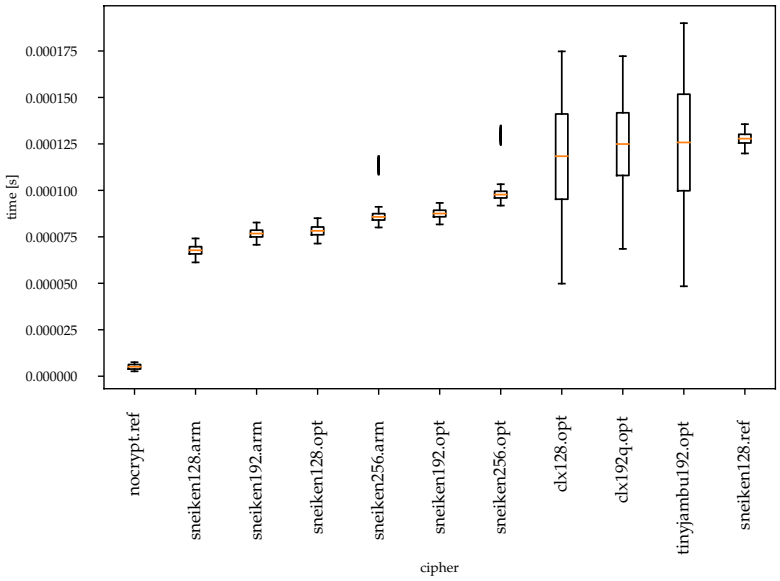

Figure 4: Ten fastest implementations on the “bluepill”

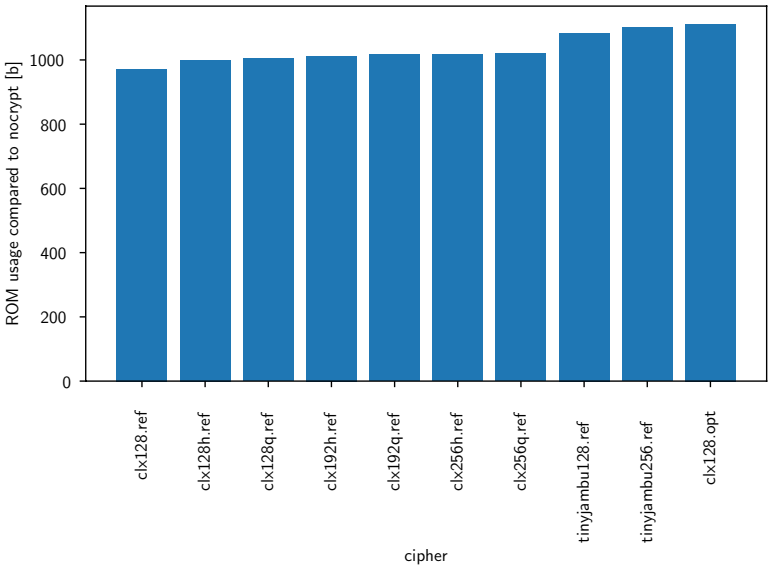

Figure 5: Ten smallest implementations on the “bluepill”

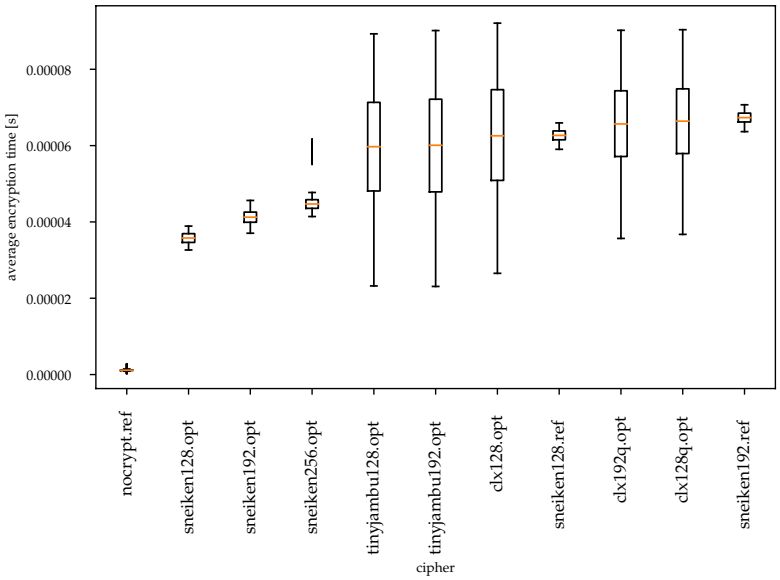

Figure 6: Ten fastest implementations on the STM32 F746ZG

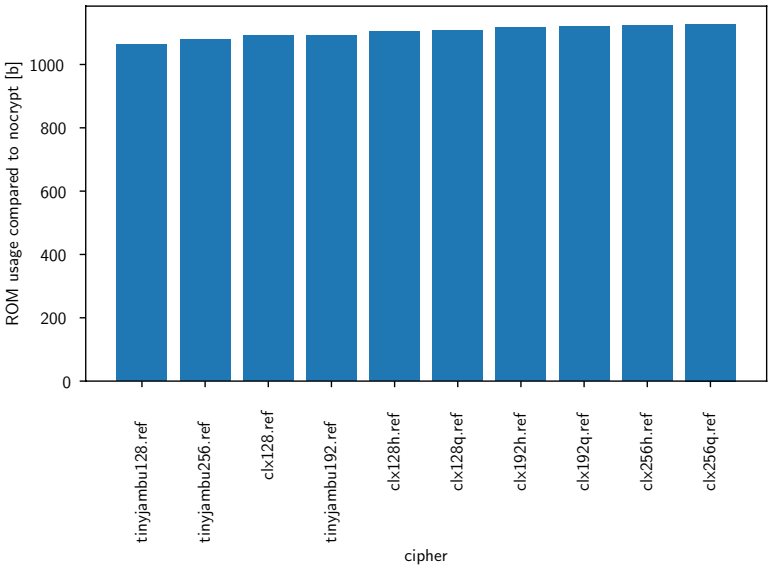

Figure 7: Ten smallest implementations on the STM32 F746ZG

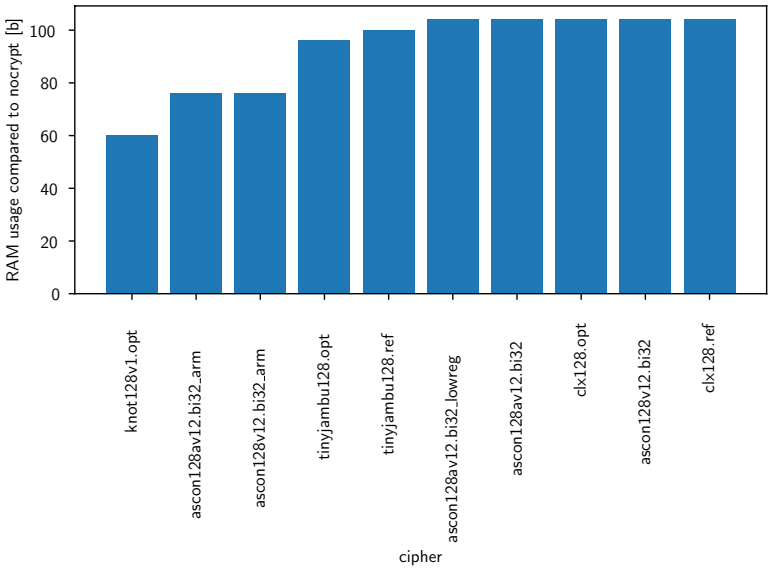

Figure 8: Ten least RAM-intensive implementations on the STM32 F746ZG

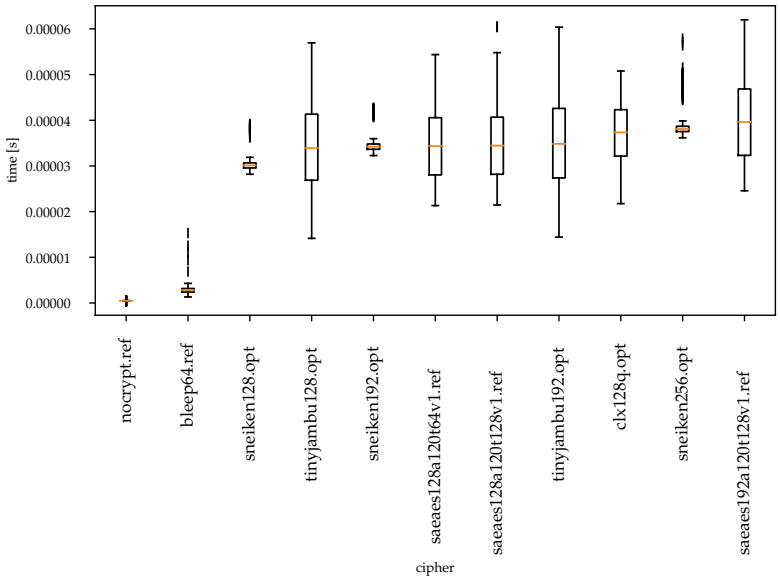

Figure 9: Ten fastest implementations on the Espressif ESP32

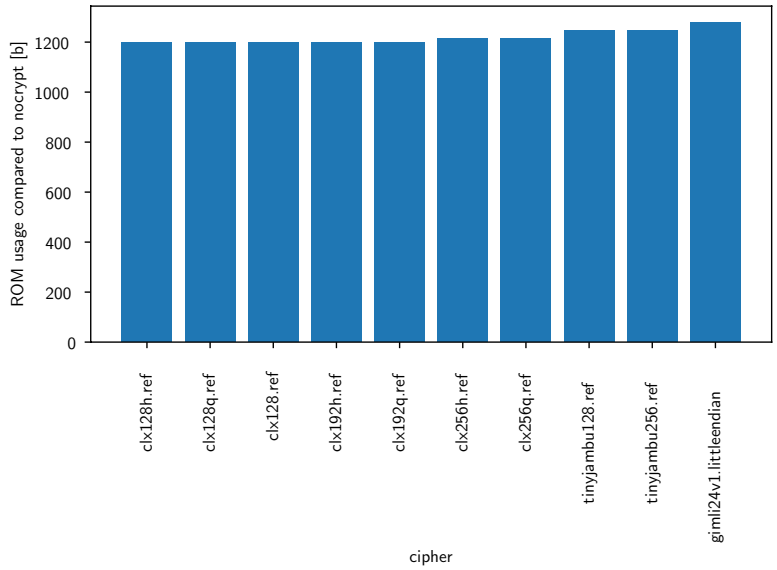

Figure 10: Ten smallest implementations on the Espressif ESP32

To provide a graphic representation of the best cipher variant in each test case, we pick the number one implementation from the speed, ROM and ROM test results on the STM32 F746ZG platform. Figure 11 shows how these candidates perform in comparison to the particularly best cipher in the other test categories. Unfortunately, we could not acquire speed results for knot128v1.opt on the MCU, that's why the radar chart uses the speed value of knot128v1.ref. Moreover, knot128v1.opt might be optimized regarding the performance according to its good test results on the Atmel chip. This could explain the less optimal results in the ROM test case.

Note, that the radar plot shows the speed values in milliseconds, the RAM values in kilobytes and the ROM values in 10 kilobyte blocks in order to support easier reading.

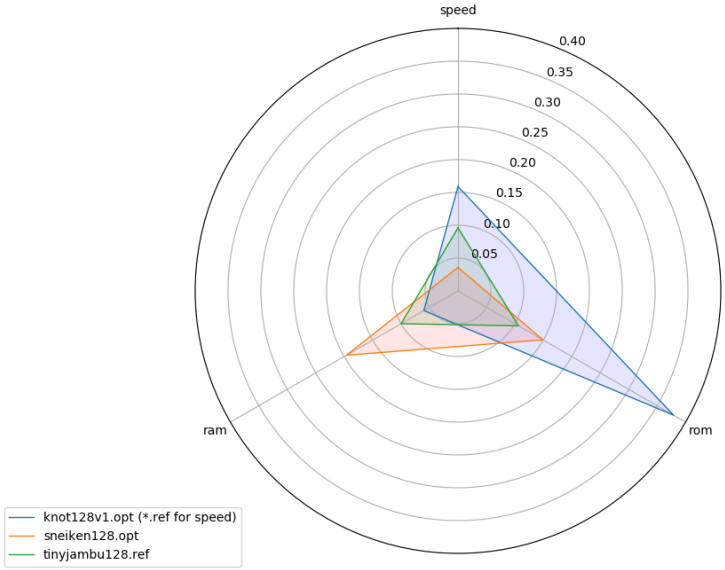

Figure 11: Test results for top ciphers per test case on the STM32 F746ZG

5 Conclusion

In this paper, we introduced a framework for benchmarking cipher software implementations of the NIST LWC project on various MCUs. We gave an overview over the its architecture, the core components and the communication channels. Multiple test cases and the corresponding methodology to gather results have been described.

Moreover, test data for more than 200 cipher variants was presented, featuring at least one variant from the 56 1st round candidates on at least one out of four supported MCU platforms. This work can be used as a reference when looking for performance figures of software implementations of NIST LWC candidates and should help in evaluating the fitness of AEAD ciphers for lightweight applications. Furthermore, the results can be viewed in comparison to other benchmarks in the context of the NIST LWC project, which aids in validating the test results and can contribute to the decision making process, e.g. for 2nd round candidates.

Our results show that multiple NIST LWC candidates perform better than the benchmarked AES-GCM implementation (aes128k96n.ref) stripped out of mbed TLS. On the one hand, this encourages the further effort in standardizing one or several candidates, as some of them seem to be more suitable for embedded devices regarding the performance test cases. However, additional criteria like security against side-channel attacks need to be taken into account when comparing the ciphers to each other. Moreover, it could be argued that the LWC submissions are mostly build to deliver good performance figures in resource-constrained environments and that there exist far more efficient GCM implementations than the standard mbed TLS variant.

6 Future Work

Besides the already conducted tests, more extensive test cases apart from processing e.g. the NIST test vectors for the performance test could be integrated in the test framework. Also, adding support for different MCUs is a sane and feasible approach for future research. By implementing more test platforms, the number of untested cipher variants (ca. 20) could maybe be reduced. In this context, the causes for the failure of these algorithms could be evaluated by debugging the compiling and test process. Lastly, the extension of the test framework such that it also supports the evaluation of the corresponding hash functions submitted to the NIST LWC project might be a reasonable research goal for the future.

7 Acknowledgements

This work is supported by the German Federal Ministry of Economic Affairs and Energy on the basis of a decision by the German Bundestag (grant 0350042A).

References

[1] Ralph Ankele and Robin Ankele. *Software Benchmarking of the 2nd round CAESAR Candidates*. Sept. 2016. DOI: 10.13140/RG.2.2.28074.26566.

[2] Daniel J. Bernstein. *Caesar: Competition for authenticated encryption: Security, applicability, and robustness*. https://competitions.cr.yp.to/caesar.html (accessed 2019-07-28). 2014.

[3] Mickaël Cazorla et al. "Survey and Benchmark of Lightweight Block Ciphers for MSP430 16-bit Microcontroller". In: *Sec. and Commun. Netw.* 8.18 (Dec. 2015), pp. 3564–3579. ISSN: 1939-0114. DOI: 10.1002/sec.1281. URL: http://dx.doi.org/10.1002/sec.1281.

[4] O. Hyncica et al. "Performance evaluation of symmetric cryptography in embedded systems". In: *Proceedings of the 6th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems*. Vol. 1. 2011, pp. 277–282. DOI: 10.1109/IDAACS.2011.6072756.

[5] H. Tschofenig and M. Pegourie-Gonnard. "Performance of state-of-the-art cryptography on ARM-based microprocessors". en. In: *NIST Workshop on Lightweight Cryptography* (2015).

[6] Daniel Dinu et al. "Triathlon of lightweight block ciphers for the Internet of things". In: *Journal of Cryptographic Engineering* (July 2015). DOI: 10.1007/s13389-018-0193-x.

[7] Daniel Dinu et al. "FELICS - Fair Evaluation of Lightweight Cryptographic Systems". en. In: *NIST Workshop on Lightweight Cryptography* (2015).

[8] Daniel J. Bernstein and Tanja Lange. *eBACS: ECRYPT Benchmarking of Cryptographic Systems*. http://bench.cr.yp.to (accessed 2019-07-28).

[9] Christian Wenzel-Benner and Jens Gräf. "XBX: eXternal Benchmarking eXtension for the SUPERCOP Crypto Benchmarking Framework". In: *Cryptographic Hardware and Embedded Systems, CHES 2010*. Ed. by Stefan Mangard and François-Xavier Standaert. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 294–305. ISBN: 978-3-642-15031-9.

[10] Jens-Peter Kaps. *eXtended eXternal Benchmarking eXtension (XXBX)*. SPEED-B - Software performance enhancement for encryption and decryption, and benchmarking. Utrecht, Netherlands, invited talk. 2016.

[11] Morris J. Dworkin. *NIST. No. Special Publication (NIST SP)-800-38D: Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*. 2007.

A Extended result table

Cipher	F7 avg. time [μs]	ESP32 avg. time [μs]	F1 avg. time [μs]	Uno avg. time [μs]	F7 ROM [B]	ESP32 ROM [B]	F1 ROM [B]	Uno ROM [B]	F7 RAM [B]
aceae128v1.ref	7043.028	17169.360	12154.701	109074.672	2756	3216	3084	4398	516
aes128k96n.ref	142.316	55.314	383.474		7284	9216	7460	9968	908
ascon128av12.bi16	284.207	9099.200			152900	241296			292
ascon128av12.bi32		1353.355			34052	55280	32844	59538	104
ascon128av12.bi32_arm					30696		30968		76
ascon128av12.bi32_lowreg		1262.480			31892	53072	32760	60412	104
ascon128av12.bi8	752.251				337376				480
ascon128av12.opt64		2250.001			43876	69216	63576		116
ascon128av12.ref	74.831	2432.839		7129.782	43444	74704	60016	8432	148
ascon128v12.bi16	328.641	8367.572			128984	200432			236
ascon128v12.bi32		1411.887			27536	44000	25968	50304	104
ascon128v12.bi32_arm					24896		25076		76
ascon128v12.bi32_lowreg		1336.083			25436	43360	26272	51078	104
ascon128v12.bi8	877.510				287400				460
ascon128v12.opt64		1951.799			37860	57008	54080		112
ascon128v12.ref	83.480	2584.507		8402.332	39320	67424	55228	6980	124
ascon80pqv12.opt64		1949.223			38028	57024	55032		124
ascon80pqv12.ref	84.052	2686.555		8507.308	39380	69280	56572	7968	140
bleep64.ref		2.882			3172	2336	3188	6418	
cilipadi128v1extrahot.ref	36002.369	11490.202	56410.952	327408.155	2916	3264	3336	7460	548
cilipadi128v1hot.ref	35413.662	12068.194	59374.403	343928.942	3704	3344	4160	7194	548
cilipadi128v1medium.ref	28650.856	9005.136	38666.609	256616.182	3524	3168	3988	6756	484

Cipher	F7 avg. time [μs]	ESP32 avg. time [μs]	F1 avg. time [μs]	Uno avg. time [μs]	F7 ROM [B]	ESP32 ROM [B]	F1 ROM [B]	Uno ROM [B]	F7 RAM [B]
cilipadi128v1mild.ref	29819.911	10030.453	49181.859	285771.829	2672	3088	3136	6288	468
clae128v1.ref	142.246	83.672	329.775	5076.378	2400	2656	2908	5480	180
clx128.opt	62.541	47.502	117.966		1240	1376	1112	2224	104
clx128.ref	86.110	46.817	139.627		1092	1200	972	1978	104
clx128h.opt	77.519	44.202	170.090		1488	1616	1384	2470	124
clx128h.ref	237.619	104.345	434.414		1104	1200	1000	2004	112
clx128q.opt	66.325	37.284	142.313		1492	1616	1388	2470	124
clx128q.ref	197.847	86.480	360.577		1108	1200	1004	2004	112
clx192h.opt	74.856	47.048	143.229		1312	1424	1180	2214	144
clx192h.ref	520.535	136.294	570.491		1116	1200	1012	1966	120
clx192q.opt	65.672	40.614	124.963		1316	1424	1184	2214	144
clx192q.ref	442.384	115.836	485.224		1120	1200	1016	1966	120
clx256h.opt	103.401	62.564	205.733		1488	1648	1344	2450	176
clx256h.ref	324.405	171.855	720.935		1124	1216	1016	1966	136
clx256q.opt	97.536	57.365	192.175		1492	1648	1348	2450	176
clx256q.ref	282.468	149.102	626.238		1128	1216	1020	1966	136
comet128aesv1.ref	219.047	118.647	529.351		7280	8096	7792	7214	571
comet128chamv1.ref	2400.781	915.416	2926.009	20299.364	2964	3680	3452	6010	416
comet64chamv1.ref	1013.083	505.276	1534.297	13398.122	2620	3632	3168	5846	372
comet64speckv1.ref	817.311	417.836	1824.362	11504.740	2668	3488	3240	5808	576
drygascon128.le32	542.165	212.386		66777.671	6564	10864	15056	5494	352
drygascon128.ref	1926.271	762.502	3148.628	114483.205	5072	2704	7356	5606	364
drygascon256.le32	669.085	331.173		101699.133	5916	14512	18080	6070	380
drygascon256.ref	3130.675	1237.274	5927.809	176608.901	5212	3072	7708	6970	476
elephant160v1.ref	20909.156	11075.056	39345.962	1504155.272	3916	3488	3700	8274	1396
elephant176v1.ref	22714.283	13009.892	44010.036	1761403.177	3984	3552	3728	8388	1432

Cipher	F7 avg. time [μs]	ESP32 avg. time [μs]	F1 avg. time [μs]	Uno avg. time [μs]	F7 ROM [B]	ESP32 ROM [B]	F1 ROM [B]	Uno ROM [B]	F7 RAM [B]
elephant200v1.ref	2531.479	1187.344	4437.237	130712.124	3236	2752	3072	8116	452
estatetweaes128v1.ref	377.285	155.710	644.732	3524.408	2840	3184	2512	5824	436
estatetwegift128v1.ref	19129.064	6936.047	34897.136		2888	3376	2664	6388	1820
flexaead128b064v1.ref	947.754	326.396	1506.920	38912.561	10724	12288	10676	16784	316
flexaead128b128v1.ref	1583.428	538.362	2611.327	67167.777	10792	12384	10788	17846	436
flexaead256b256v1.ref	3339.996	983.778	5261.696	130341.935	10868	12448	10792	18090	716
Fountain128v1.ref	21242.796	4278.897	37542.719	235287.741	3120	3632	2984	6222	804
giftcofb128v1.ref	1846.168	802.828	3495.506	316451.027	1880	2384	1788	2934	192
gimli24v1.littleendian		56.094	198.699	11552.569		1280	1252	3260	
gimli24v1.ref		152.467	547.546	16766.448		1936	1692	3380	
grain128aead.opt32	130.788		289.246		3056	3440	3696	11740	8521
grain128aead.ref	16633.033	7340.962	35956.574	171825.031	3244	3600	3800	10540	705
hern128v1.ref	54626.940	14197.733	47872.160		1328	1616	1292	1776	348
hyenav1.ref	10356.078	5042.519	25426.377		2968	3552	2844	6276	1720
ingage1k128n096c224r008.ref	18131.577	12871.094	48753.118	1764995.870	1488	1808	1448	2260	144
ingage1k128n096c224r016.ref	9931.223	7788.460	26533.425	1015137.899	2300	2560	2196	3262	176
ingage1k128n096c224r032.ref	6018.361	2534.505	10875.638	89522.925	2136	2480	2072	3490	152
ingage1k128n128c256r064.ref	5315.357	3521.188	13211.554	542680.743	2224	2688	2208	4184	196
ingage1k256n096c448r064.ref	7135.222	3337.617	15018.068	118839.111	2196	2512	2160	4202	232
ingage1k256n128c448r064.ref	7263.414	3337.933	15017.772	118837.610	2188	2512	2108	4202	236
isapa128av20.ref	4426.059	2254.737	9554.568	240035.998	5352	3728	7636	5438	372
isapa128v20.ref	7665.641	6298.894	22817.292	791234.642	5352	3728	7636	5334	372
isapk128av20.ref	10451.679	4681.281	18508.977	621471.148	5608	2784	7660	3484	408
isapk128v20.ref	87341.875	38963.683	154443.122		5612	2784	7660	3484	408
knot128v1.opt		61.494		7352.878	3788	6144	4416	11652	60
knot128v1.ref	158.999	90.480	412.492	9626.300	1884	2048	1908	3412	212

Cipher	F7 avg. time [μs]	ESP32 avg. time [μs]	F1 avg. time [μs]	Uno avg. time [μs]	F7 ROM [B]	ESP32 ROM [B]	F1 ROM [B]	Uno ROM [B]	F7 RAM [B]
knot128v2.opt		95.050			6148	10896	7296	19028	204
knot128v2.ref	171.477	115.009	498.147		3092	2672	3160	4768	348
knot192.opt		140.547			5164	9104	6124	17112	216
knot192.ref	251.881	170.777	731.390		3308	2928	3408	6274	348
knot256.opt		249.563			7396	14640	10228	32158	144
knot256.ref	478.119	281.495	1456.351	67656.377	3008	3376	3180	8764	444
laemsimon128v1.ref	271.938	494.904	597.329		11684	14352	13164	41574	728
laemsimon192v1.ref	273.293	496.912	601.856		11748	14368	13220	41580	744
laemsimon256v1.ref	293.464	500.929	644.603		11764	14384	13252	41724	760
lilliputaei128v1.add_felicsref	1386.624	523.586	2581.831	14150.531	2616	3280	2556	3822	540
lilliputaei128v1.add_tweakeyloop	1780.566	627.962	2864.693	19904.161	2888	3584	2828	4056	572
lilliputaei128v1.ref	1759.047	633.841	2782.727	17488.023	2856	3632	2800	4342	556
lilliputaei192v1.add_felicsref	1946.347	643.157	3066.085	17277.658	2728	3376	2680	3916	592
lilliputaei192v1.add_tweakeyloop	2206.565	794.368	3499.348	25774.090	2904	3600	2848	4058	624
lilliputaei192v1.ref	2205.845	807.014	3576.489	22045.678	2872	3632	2820	4344	604
lilliputaei256v1.add_felicsref	2172.646	837.917	4103.997	21923.028	2860	3536	2804	4044	668
lilliputaei256v1.add_tweakeyloop	3017.399	1036.652	4644.923	35004.687	2920	3584	2860	4062	688
lilliputaei256v1.ref	2963.059	1054.081	4480.594	28661.233	2888	3632	2832	4348	672
lilliputaeii128v1.add_felicsref	2001.667	631.969	3182.153	19742.594	2372	2928	2256	3454	564
lilliputaeii128v1.add_tweakeyloop	2170.776	749.157	3563.199	24735.962	2720	3344	2616	3370	596
lilliputaeii128v1.ref	2135.359	758.002	3411.995	21420.795	2688	3392	2588	3656	580
lilliputaeii192v1.add_felicsref	2867.808	802.351	3812.611	24670.591	2464	3056	2352	3646	612
lilliputaeii192v1.add_tweakeyloop	2359.150	963.406	4289.756	30918.083	2736	3360	2624	3374	644
lilliputaeii192v1.ref	2346.207	972.953	4376.689	27175.153	2704	3408	2596	3660	628
lilliputaeii256v1.add_felicsref	2705.196	1021.793	5103.570	31886.921	2576	3136	2468	3864	680
lilliputaeii256v1.add_tweakeyloop	3526.443	1264.759	5711.381	41582.334	2752	3360	2636	3374	712

Cipher	F7 avg. time [μs]	ESP32 avg. time [μs]	F1 avg. time [μs]	Uno avg. time [μs]	F7 ROM [B]	ESP32 ROM [B]	F1 ROM [B]	Uno ROM [B]	F7 RAM [B]
lilliputaeii256v1.ref	3451.389	1285.108	5550.820	35589.623	2720	3392	2608	3660	692
limdolen128v1.ref	243.757	117.167	443.012	3573.647	1460	1552	1332	3542	292
limdolen256v1.ref	556.976	222.193	982.636	5831.568	1592	1648	1456	4680	460
mixfeed.ref	1150.036	447.074	2008.397	12872.813	2492	2704	2404	4034	452
nocrypt.ref	1.148	0.484	5.080	14.687	0	0	0	0	0
orangezestv1.ref	17504.825	5824.106	24355.806		3656	3952	3580	4644	264
oribatida192v11.ref	18380.310	6787.658	30071.750	776284.449	5096	4848	5604	10210	904
oribatida256v11.ref	25412.086	11112.096	47023.975	1265930.986	5136	4912	5652	10192	912
paefforkskinnyb128t192n48v1.ref	3571.160	1180.759	5810.102	38592.654	7432	6112	8540	14728	460
paefforkskinnyb128t256n112v1.ref	3572.702	1181.510	5683.577	38590.783	7420	6112	8500	14676	468
paefforkskinnyb128t288n104v1.ref	5947.970	1968.977	9232.277	61641.124	7388	6256	8452	15300	532
paefforkskinnyb64t192n48v1.ref	7245.789	2397.866	11716.134	73602.829	6984	5792	8016	14642	444
photonbeetleaead128rate128v1.ref	25746.004	7196.612	40575.950	355116.438	4928	2816	6872	3712	280
photonbeetleaead128rate32v1.ref	66774.791	17899.066	87720.702	883322.814	4888	2736	6844	3608	280
pyjamask128aeadv1.ref	2000.806	735.478	3153.497	148612.689	2548	3024	2408	3890	604
pyjamask96aeadv1.ref	1255.104	665.301	2976.854	131952.916	2404	2816	2264	3624	536
qameleon128128128tcgpv1.ref	2544.762	1086.633	4509.733		7280	8208	7408	12254	1176
qameleon12812864gpv1.ref	2536.231	1028.097	4375.957		5984	6944	6076	11302	1064
qameleon12812896gpv1.ref	2495.770	1029.175	4375.997		6552	7568	6392	11552	1108
qameleon1286464mev1.ref					4900	5808	4868	7352	
qameleon6464nnmev1.ref					4116	4864	4160	6746	
qameleon6464tcmev1.ref					5248	6048	5272	7818	
Quartet128v1.ref	113.157	73.063	344.835		3160	3696	3912	9198	188
remusm1v1.ref	3701.481	1284.638	5714.744	38416.497	3932	4704	3688	5446	432
remusm2v1.ref	4366.065	1526.519	6689.271	45612.362	4216	4880	3940	5864	471
remusn1v1.ref	2684.551	940.644	4181.677	28047.692	3528	4064	3272	3246	368

Cipher	F7 avg. time [μs]	ESP32 avg. time [μs]	F1 avg. time [μs]	Uno avg. time [μs]	F7 ROM [B]	ESP32 ROM [B]	F1 ROM [B]	Uno ROM [B]	F7 RAM [B]
remusn2v1.ref	3372.022	1180.632	5228.450	35186.766	3732	4208	3436	3568	424
remusn3v1.ref	4358.761	1468.448	6906.212	43666.916	3492	3952	3228	3420	332
romulusm1v1.ref	6783.511	2192.651	10134.507	84618.714	5216	5520	4920	7882	528
romulusm2v1.ref	6814.861	2276.410	10257.003	87851.608	5972	6128	5780	8130	524
romulusm3v1.ref	4515.148	1521.942	6903.088	42393.937	6116	6304	5916	7976	524
romulusn1v1.ref	5436.593	1764.565	8015.440	68023.202	4164	4528	3828	4176	496
romulusn2v1.ref	4732.402	1765.100	8152.434	68020.708	4148	4576	3896	4198	476
romulusn3v1.ref	3195.939	1178.074	5347.701	32732.699	4168	4576	3924	4078	460
saeaes128a120t128v1.ref	117.800	35.522	180.444		7932	8624	7908	13070	312
saeaes128a120t64v1.ref	118.838	35.286	179.499		7928	8640	7912	13070	312
saeaes128a64t128v1.ref	137.596	41.186	209.096		8020	8736	8016	13070	336
saeaes128a64t64v1.ref	135.789	40.980	207.832		8016	8752	8020	13070	336
saeaes192a120t128v1.ref	136.813	40.706	209.073		8636	9328	8532	14930	352
saeaes192a64t128v1.ref	158.538	47.301	240.697		8724	9424	8640	14930	376
saeaes192a64t64v1.ref	158.953	47.251	239.753		8720	9440	8644	14930	376
saeaes256a120t128v1.ref	159.668	46.904	248.851		9248	9952	9068	16490	392
saeaes256a64t128v1.ref	182.645	54.406	288.894		9336	10048	9176	16490	416
saeaes256a64t64v1.ref	181.855	54.356	287.839		9332	10096	9180	16490	416
saefforkskinnyb128t192n56v1.ref	3351.887	1181.417	5687.258	38490.654	5280	5792	5296	13018	476
saefforkskinnyb128t256n120v1.ref	3560.404	1182.093	5690.846	38515.224	5300	5856	5344	13018	484
saturninctrcascadev2.bs32	157.524	99.515	320.447	9834.233	3440	4592	3644	10558	480
saturninctrcascadev2.bs32x	344.648	180.624	587.491	16402.347	11612	15088	11544	27722	972
saturninctrcascadev2.bs64	634.912	326.670	2032.773	28795.123	10456	13952	17632	26870	984
saturninctrcascadev2.ref	559.485	228.980	880.751	8484.003	2304	2464	2240	3210	512
saturninshortv2.ref					2680	2912	2580	3808	324
schwaemm128128v1.opt	122.009	46.685	248.978		2472	2576	2364	4734	120

Cipher	F7 avg. time [μs]	ESP32 avg. time [μs]	F1 avg. time [μs]	Uno avg. time [μs]	F7 ROM [B]	ESP32 ROM [B]	F1 ROM [B]	Uno ROM [B]	F7 RAM [B]
schwaemm128128v1.ref	173.971	64.153	268.205		5408	2960	7440	4968	220
schwaemm192192v1.opt	203.037	65.055	359.055		2904	2240	2804	4002	168
schwaemm192192v1.ref	197.555	85.908	353.307		5840	2608	7880	4308	268
schwaemm256128v1.opt	166.211	58.600	328.541		2548	2528	2436	5760	184
schwaemm256128v1.ref	179.176	77.291	321.097		5484	2912	7512	6066	284
schwaemm256256v1.opt	227.445	79.985	446.923		2336	2352	2336	5412	208
schwaemm256256v1.ref	263.317	104.040	404.160		5276	2736	7412	5758	308
sestatetweaes128v1.ref	341.770	127.214	594.986	3174.513	2884	3216	2592	5788	436
shamashv1.opt64	133.900	88.778	371.430		13252	16320	17200	42218	140
shamashv1.ref	157.255	85.185	384.596	7950.752	4012	4496	4716	9584	156
simple128aes10.ref	333.066	127.319	676.862		7000	7488	7472	6722	619
simple128gift.ref	40272.066	13744.350	52398.740	445537.719	2868	3248	3348	5708	864
simple128speck.ref	1721.942	898.552	4281.735	27911.181	2368	2752	2868	5148	904
simple64gift.ref	26572.177	11271.329	44377.340	351339.646	2956	3408	3504	5780	692
simple64present.ref	21708.020	6724.769	30421.479	224531.057	2692	3200	3208	5526	1064
simple64speck.ref	1966.768	796.116	3539.544	23794.035	2544	2992	3048	5304	576
sivrijndael256aead128v1.ref	1621.036	874.424	2803.449		3096				1648
sivtemphotonaeadv1.ref	27586.642	9296.271	45904.922	342048.845	5072	3072	7068	2538	368
skinnyaeadtk296128v1.ref	3964.238	1519.284	7004.319	42884.681	5420	5744	5452	10694	456
skinnyaeadtk29664v1.ref	3985.951	1519.344	7002.497	42877.123	5436	5776	5452	10702	456
skinnyaeadtk3128128v1.ref	6331.048	2258.720	10006.620	65270.758	5716	6304	5680	10216	460
skinnyaeadtk312864v1.ref	6332.436	2258.105	10536.461	65263.320	5728	6320	5692	10230	460
skinnyaeadtk396128v1.ref	5846.707	2258.399	10129.828	65268.957	5724	6304	5704	10220	456
skinnyaeadtk39664v1.ref	5820.733	2258.092	10009.874	65261.571	5740	6320	5712	10234	456
sneiken128.arm			67.754				1276		
sneiken128.avr				1350.041				1390	

Cipher	F7 avg. time [μs]	ESP32 avg. time [μs]	F1 avg. time [μs]	Uno avg. time [μs]	F7 ROM [B]	ESP32 ROM [B]	F1 ROM [B]	Uno ROM [B]	F7 RAM [B]
sneiken128.opt	35.786	30.388	78.218	14373.459	1492	1824	1464	5756	196
sneiken128.ref	62.679	41.485	127.839	14981.479	1780	2304	1700	6612	268
sneiken192.arm			76.772				1308		
sneiken192.avr				1557.860				1394	
sneiken192.opt	41.242	34.529	87.548	16731.597	1528	1824	1496	5760	204
sneiken192.ref	67.342	46.231	141.645	17376.493	1808	2304	1728	6618	276
sneiken256.arm			86.483				1404		
sneiken256.avr				1786.968				1496	
sneiken256.opt	45.042	38.759	98.557	19286.015	1620	1920	1592	5862	220
sneiken256.ref	78.314	51.378	152.561	19968.699	1816	2304	1732	6618	284
spix128v1.ref	4529.562	9763.821	7404.409	63942.546	2964	3616	3340	4842	444
spoc128sliscplight256v1.ref	2247.954	5056.624	3794.102	33292.171	2372	2864	2832	4022	428
spoc64sliscplight192v1.ref	2941.583	8670.760	5396.709	39682.020	2360	2928	2820	4032	412
spook128mu384v1.ref	739.610	259.851	987.533	41278.285	3140	3264	3040	6464	468
spook128mu512v1.ref	739.610	259.995	987.533	41278.284	3140	3264	3040	6464	468
spook128su384v1.ref	684.107	259.661	991.980	41275.597	3080	3248	2984	6396	452
spook128su512v1.ref	684.107	259.651	991.979	41275.597	3080	3248	2984	6396	452
subterraneanv1.add_mem_compact	940.932	521.063	1380.673	12629.421	9108	11760	8584	12378	496
subterraneanv1.ref	3647.831	1687.627	6350.099	36530.360	4284	5056	4336	5620	884
sundaegift0v1.ref	2407.938	1086.074	4658.570	430499.917	2440	2992	2916	4304	376
sundaegift128v1.ref	2921.141	1288.033	5528.094	510744.040	2460	3008	2924	4366	408
sundaegift64v1.ref	2673.655	1190.956	5106.517	471840.387	2472	3008	2932	4370	392
sundaegift96v1.ref	2353.894	1239.887	5317.298	491292.066	2476	3008	2936	4370	404
syconaer64128v1.ref	8051.422	2657.666	12888.710	108264.248	4512	5664	4736	6256	300
syconaer96128v1.ref	7835.576	2615.087	12331.872	106560.268	5496	5840	8308	6610	308
tgifm1128v1.ref	409.503	197.940	954.543		4316	5472	4156	6192	640

Cipher	F7 avg. time [μs]	ESP32 avg. time [μs]	F1 avg. time [μs]	Uno avg. time [μs]	F7 ROM [B]	ESP32 ROM [B]	F1 ROM [B]	Uno ROM [B]	F7 RAM [B]
tgifm2128v1.ref	505.697	239.750	1180.245		4600	5632	4408	6610	680
tgifn1128v1.ref	309.606	144.558	697.955		3912	4816	3740	3992	576
tgifn2128v1.ref	422.515	184.453	907.182		4116	4960	3904	4314	632
tinyjambu128.opt	59.421	34.026	127.869		1232	1424	1248	2124	96
tinyjambu128.ref	96.105	54.527	163.764		1064	1248	1084	1964	100
tinyjambu192.opt	59.894	35.186	125.577		1364	1536	1344	2224	108
tinyjambu192.ref	127.328	68.450	218.971		1092	1280	1120	2042	108
tinyjambu256.opt	73.820	45.089	150.111		1280	1456	1276	2170	116
tinyjambu256.ref	88.360	63.000	188.512		1080	1248	1100	1964	116
triadaev1.ref	46748.111	19094.936	118120.957	581589.825	4164	5072	4108	8994	756
triflev1.ref	36894.092	13101.582	63284.362		2960	3504	2704	6336	2140
twegift64locusaeadv1.ref	13318.046	6485.124	27400.572		4240	4880		6554	1332
twegift64lotusaeadv1.ref	15648.897	6483.968	27741.335		4908	5872		8458	1332
wageae128v1.ref	1733.381	1880.893	7538.703	40367.267	6612	5792	7720	6862	313
xoodyakv1.ref	472.481	252.050	890.169	41096.504	5068	2848	7128	2820	340
yarara128v1.opt64	121.562	4637.719			62360	109632	95884		108
yarara128v1.ref	247.856	141.179	619.524	17209.393	2228	2672	2576	5910	272