A Comparative Analysis of Lightweight Hash Functions Using AVR ATXMega128 and ChipWhisperer

Mohsin Khan[1]*, Dag Johansen[1], and Håvard Dagenborg[1]

UiT: The Arctic University of Norway, Tromsø, Norway
{mohsin.khan, havard.dagenborg, dag.johansen}@uit.no

Abstract. Lightweight hash functions have become important building blocks for security in embedded and IoT systems. A plethora of algorithms have been proposed and standardized, providing a wide range of performance trade-off options for developers to choose from. This paper presents a comparative analysis of 22 key software-based lightweight hash functions, including the finalist from the SHA-3 competition. We use a novel benchmark methodology that combines an AVR ATXMega128 microcontroller with the ChipWhisperer cryptanalysis platform and evaluate and compare the various hash functions along several dimensions, including execution speed, memory footprint, and energy consumption. Using the composite E-RANK metric, we provide new insight into the various trade-offs each hash function offers to system developers.

Keywords: Lightweight Hash Functions · NIST Cryptographic Algorithms · ISO Standardized Lightweight Hash Functions · Software Benchmarking · Performance Analysis · AVR Microcontroller.

1 Introduction

Hash functions are vital building blocks for many security mechanisms, like digital signatures, message authentication codes, and file checksums. Traditional hash functions are effective on devices with ample computational resources, but may fall short on resource-constrained devices with limited computational capacity, such as 8-bit AVR and PIC family of microcontrollers. In some computational environments, like IoT and embedded systems, devices might also be battery-powered and require energy-efficient algorithms.

DM-PRESENT [15], to our knowledge, was among the first hash functions developed specifically for devices with limited resources. From 2007 to 2012, NIST ran the SHA-3 competition to find a more lightweight and secure alternative to SHA-2. Keccak [12] won this competition, with BLAKE as one of the finalists. Both of these hash functions are actively used in cryptographic libraries today, such as OpenSSL and Python's hashlib. Additionally, Keccak is specifically utilized in blockchain technology and smart contracts. In 2016, ISO standardized

* Corresponding author: Mohsin Khan, email: mohsin.khan@uit.no

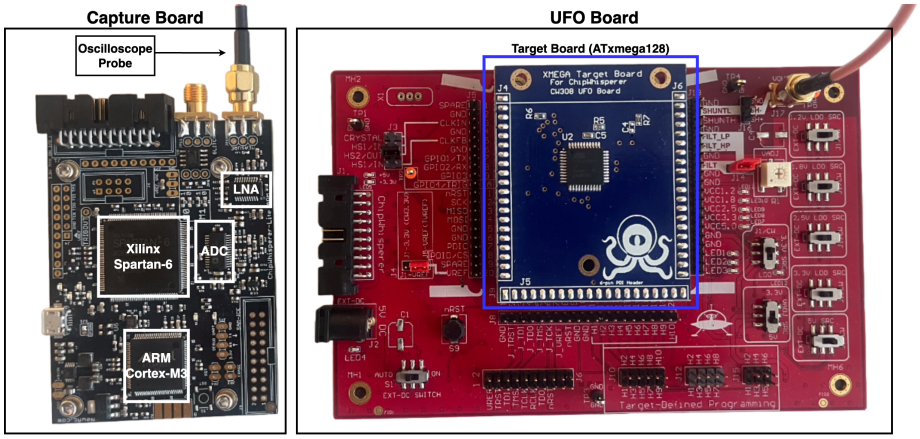

Fig. 1. ChipWhisperer hardware overview

PHOTON, SPONGENT [14], and Lesamnta-LW under ISO/IEC 29192-5:2016. Then, in 2019, NIST began the standardization of lightweight cryptographic algorithms and received 57 submissions, some of which included lightweight hash functions. After three rounds of evaluation and rigorous analysis of security and benchmarking results, ASCON [23] was announced as the winner in 2023. Consequently, a plethora of hashing functions have been proposed and developed, providing a wide range of performance and security trade-offs. However, few works exist that analyze and compare a larger selection of these hash functions.

This paper benchmarks and compares 22 key lightweight hash functions, including those submitted to NIST [38], those standardized by ISO [28], including PHOTON and Lesamnta-LW, along with the SHA-3 competition [18] finalist, BLAKE [2] and its latest variant, BLAKE3 [35], due to its wide usability in various well-known applications.

For each hash function, we measure several performance metrics, including Cycles per Byte (CPB), energy consumption, RAM usage, and ROM using a novel benchmark environment comprised of the AVR ATmega128 microcontroller in combination with the ChipWhisperer [34] cryptanalysis tool. We also calculate the composite E-RANK metric [29] to compare provide a more comprehensive overview of the performance trade-offs of each hash function.

2 ChipWhisperer

Our measurements were performed using the ChipWhisperer Level 2 Starter Kit from NewAE Technology, a toolkit originally designed for learning about side-channel attacks on embedded devices by analyzing power consumption. Its modular design allows for the integration of specialized modules, enabling precise measurements and advanced testing of cryptographic systems. ChipWhisperer

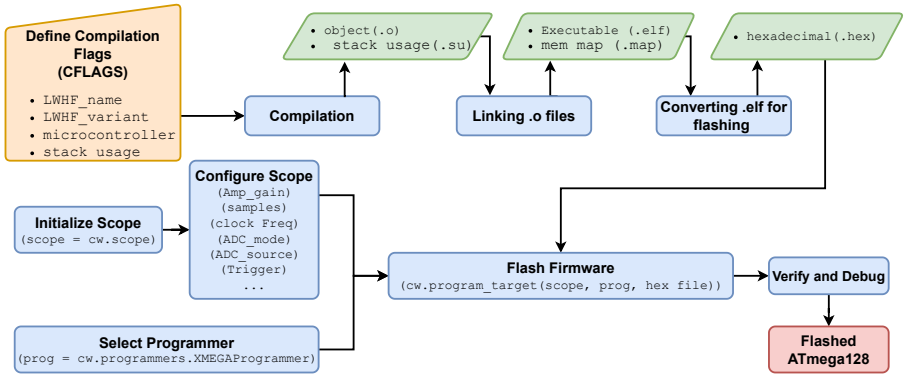

Fig. 2. Process flow for firmware build and deployment to target device.

comprises a *capture board* and a *target board*, connected via a serial connection, and an oscilloscope probe, as shown in Figure 1.

The ChipWhisperer capture board includes an FPGA microcontroller, an Analog-to-Digital Converter (ADC), and a Low-Noise Amplifier (LNA). The FPGA microcontroller serves as the central processing unit, managing communication, timing, and data acquisition processes. The LNA amplifies weak power signals to reduce noise and enhance the accuracy of the power analysis obtained from the target board. These amplified signals are then captured by the ADC for conversion into digital signals. The target board is mounted on a Universal Feature Observation (UFO) Board, which provides a standardized interface for power, clock, and data connections to the target board. Data is transferred between ChipWhisperer and the target board over a serial port, while the oscilloscope probe collects voltage traces as the target board executes specific processes triggered by this communication.

The ChipWhisperer Python API allows interaction with the ChipWhisperer FPGA, enabling programming of the target board, configuring clock settings, and triggering the target board to start and stop cryptographic operations. It also facilitates the capturing and transferring of power traces. Also, the Python API enables the configuration of ADC and LNA parameters, such as sampling rate and trigger settings, as well as adjustments to gain for capturing power traces and amplifying weak power signals.

The target board is programmed by uploading firmware, which is developed in the C language. This firmware includes a base C program designed to manage simple serial communication with the ChipWhisperer Python API. The base program handles trigger signals from the Python API and incorporates macros. These macros are used for the hash function and its variants, with the implementations of these hash functions sourced from their original publications. The firmware is compiled using AVR-GCC, including specific compiler flags for object files and macros that define the hash function name and variant. During the compilation process, the C implementations of the selected hash functions are

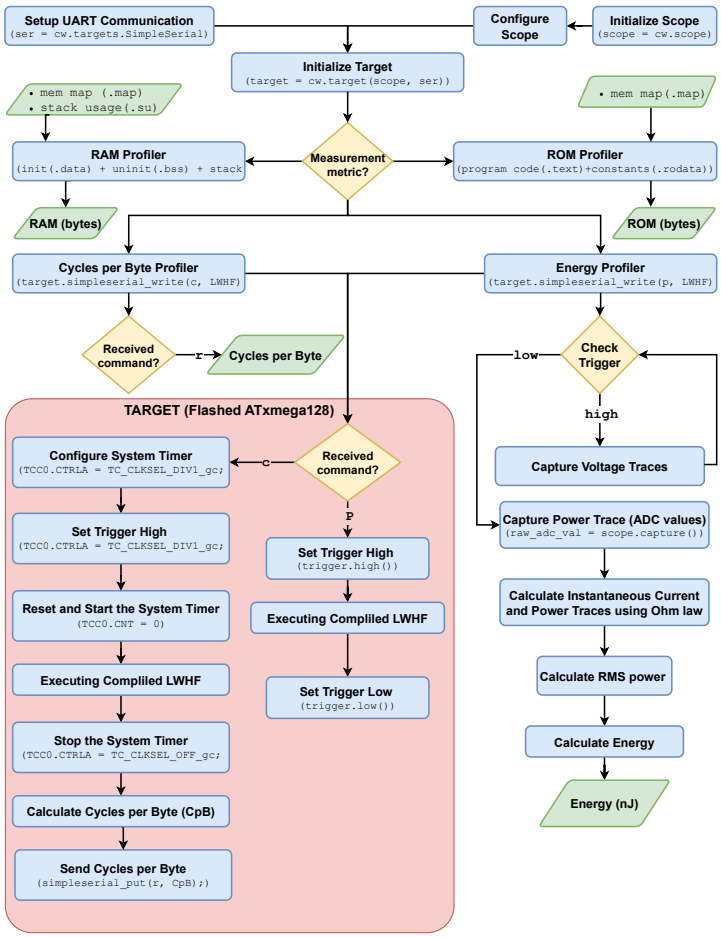

Fig. 3. Benchmarking framework.

converted into object files, and a corresponding `.su` file is generated to provide stack usage information. These object files are then linked together to create `.elf` and `.map` files. The `.elf` file is subsequently converted into a `.hex` file, which is used to flash the target device. The flashing is performed using the ChipWhisperer Python API, where the scope is initialized, and settings for amplifier gain, sampling rate, trigger conditions, and clock sources are configured. After flashing, the firmware is verified and debugged. The process for building firmware and deploying it on the target device using ChipWhisperer is illustrated in Figure 2.

3 Benchmarking Framework

We have opted for the AVR ATXMega128D4 microcontroller [32] as our target board for benchmarking the selected lightweight hash functions. ATXMega128D4 is a low-power 8-bit RISC microcontroller with 128 KB of flash memory, 8 KB of SRAM, 2 KB of EEPROM, a clock frequency of 32 MHz, and an operating voltage range of 1.6 to 3.6 V. These resource constraint specifications are sufficient to implement all the lightweight hash functions in this study and conduct performance tests. This setup also allows precise control over code execution and simplifies optimization at the assembly level.

Benchmarking begins by initializing the scope and configuring its settings. A 5 dB gain is applied to amplify low-amplitude power signals to minimize distortion, allowing clear capture of signal variations during hash function operations. A high-to-low trigger edge synchronizes the capture window with the start and the end of the hash function execution, and a sampling rate of 5 samples during the capture window offers moderate resolution for accurate trace alignment. The clock frequency is set to 7.3728 MHz using the ATXMega128D4's internal clock. Simultaneously, a simple serial communication is established using UART. After setting up the simple serial interface and the scope, the target device is initialized. Once the target initialization is complete, specific measurement metrics are selected for execution time, memory footprint, and energy profiling, as illustrated in the workflow of the benchmarking framework in Figure 3.

3.1 Execution Time Profiling

We measure execution time in terms of the number of processing cycles required to hash each byte of data, commonly known as Cycles per Byte (CPB). Our CPB profiler utilizes a control register associated with the timer/counter on the ATXMega128D4 target board. According to the hardware specifications for the ATXMega128D4 microcontroller, the `TCC0.CTRLA` register is responsible for configuring the clock source and pre-scaler settings. When CPB measurement is selected, the `TCC0.CTRLA` register is initialized and set to zero at the beginning of the hashing process.

Once the hashing process has completed processing the input data, the control register is stopped and the total number of cycles used is retrieved from the register. The resulting value is divided by the total number of input bytes processed to compute the final CPB value, before being transmitted to the host system. Note that the pre-scaler is set to `CLK_DIV1`, meaning that the timer/counter operates at the system clock frequency without any division.

3.2 Memory Footprint Profiling

In sponge constructions [13], the internal state is divided into two parts: rate and capacity. The rate indicates the number of bits of the state that can be directly read or written during the absorption and squeezing phases, determining the throughput of the function. The capacity is the portion of the state that remains

hidden from input and output, providing security against cryptanalytic attacks. Merkle–Damgård constructions [21] [31] do not explicitly separate the state into rate and capacity. Instead, the internal state is represented by the chaining value, while the rate is equal to the message block size. The overall state is updated by a compression function, and security is ensured through the design of that function and the padding scheme, rather than through a reserved hidden portion of the state.

For our purpose, we assess the memory footprint of each hash function by measuring its RAM and ROM consumption, which involves evaluating the size of the various memory segments used and summing up the total byte count. For RAM, we include the sizes of both initialized and uninitialized global and static variables, along with both dynamic and static stack usage, as shown below.

$$\text{RAM} = \texttt{.data} + \texttt{.bss} + \texttt{.su} \tag{1}$$

The initialized `.data` segment and the uninitialized `.bss` segment are extracted from the `.map` file that was generated by the AVR-GCC compiler during the linking stage. Both dynamic and static stack usage data are obtained from the `.su` files created during the compilation of individual object files, providing a comprehensive breakdown of memory allocation. Heap memory allocated via `malloc` and `calloc` calls are excluded because dynamic memory allocation is not utilized in the C implementations of the selected hash functions.

For ROM, we sum the memory occupied by the program code `.text` segment and constant data in the `.rodata` segment, as shown below.

$$\text{ROM} = \texttt{.text} + \texttt{.rodata} \tag{2}$$

The generated `.map` file provides a detailed memory map of the program code and constant data, allowing precise measurement of the total ROM footprint.

3.3 Energy Profiling

To profile energy consumption, we use the `cw.capture_trace` API function of the ChipWhisperer toolkit. The API call returns an array of instantaneous voltage samples collected by the ADC, normalized to the range -0.5 to +0.5. The voltage samples are measured when a high-to-low trigger signal is detected from the target device via serial communication.

While normalization ensures consistency across different hardware configurations, it requires us to convert the samples back to actual voltages. For this, let $\hat{V}$ be the normalized ADC voltage sample, V_{ref} the reference voltage of the ADC, which is in our case is 1 V, and G the gain of the amplifier applied to the signal before digitization, which in our case is 5 dB, then the actual voltage V is given by

$$V = \frac{V_{\text{ref}}}{G} \times \hat{V} = 0.2\hat{V} \tag{3}$$

Furthermore, for the ATXMega128D4 microcontroller, we have the shunt resistance $R_{\text{shunt}} = 49.9\,\Omega$ and the supply voltage $V_{\text{sup}} = 3.3\,\text{V}$. The instantaneous

current I of a sample $\hat{V}$ is then given by Ohm's law as follows

$$I = \frac{V}{R_{\text{shunt}}} = \frac{V}{49.9} = \frac{0.2\hat{V}}{49.9} = 4.0 \times 10^{-3}\hat{V} \tag{4}$$

This gives us the instantaneous power P for sample $\hat{V}$ as follows.

$$P = I \times V_{\text{sup}} = 4.0 \times 10^{-3}\hat{V} \times 3.3 = 1.3 \times 10^{-2}\hat{V} \tag{5}$$

The values in the trace array can sometimes be negative due to signal oscillations present in the original voltage trace. To obtain accurate power and energy calculations, we therefore calculate the Root Mean Square (RMS) of the trace. Given a trace $[\hat{V}_1, \hat{V}_2, \ldots, \hat{V}_N]$ of N samples, the RMS power P_{rms} is given as follows.

$$P_{\text{rms}} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} P_i^2} = \sqrt{\frac{1.7 \times 10^{-4}}{N}\sum_{i=1}^{N} \hat{V}_i^2} \tag{6}$$

To calculate an accurate energy estimation, execution time needs to be considered, which represents the total duration the microcontroller spends executing the hash function. The execution time is determined using Equation 7, where T_{exec} denotes the execution period, C is the total number of processing cycles used during the hash function's execution, and f_{clk} indicates the microcontroller's clock frequency, which in our case is set to 7.3728 MHz. Once the execution period has been measured, the energy consumption can be calculated using Equation 8.

$$T_{\text{exec}} = \frac{C}{f_{\text{clk}}} = \frac{C}{7.3728\,\text{MHz}} \tag{7}$$

$$E = T_{\text{exec}} \times P_{\text{rms}} = \frac{C}{7.3728\,\text{MHz}} \times \sqrt{\frac{1.7 \times 10^{-4}}{N}\sum_{i=1}^{N} \hat{V}_i^2} \tag{8}$$

3.4 Performance Comparison and Ranking

While single-dimensional metrics, such as throughput, CPB, and energy consumption, are useful when comparing hash functions with similar optimization goals, they are not suitable for evaluating tradeoff between various performance dimensions. For hardware implementations, Figure of Merit (FOM) [3] is commonly used as compound metric that measures performance as the ratio of throughput to the square number of logical gates (i.e., GE). FOM also captures energy requirements, as power consumption is proportional to the number of logic gates, represented by the Gate Equivalent (GE) factor. Later variants of FOM also normalize to the hardware clock frequency [26], as shown in Equation 9.

$$\text{FoM} = \frac{\text{throughput}}{\text{clk} \times \text{GE}^2} \tag{9}$$

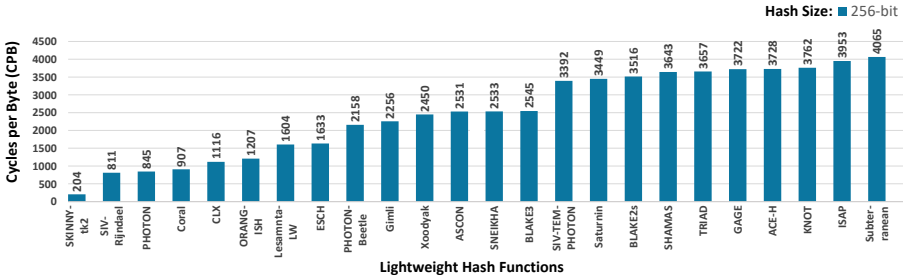

Fig. 4. Execution time (CPB) of lightweight hash functions.

In our case, for software implementations, the RANK metric [8] is commonly used to measure the tradeoff between execution efficiency, CPB, relative to memory usage (i.e., RAM and ROM). However, unlike FOM, the included memory footprint of RANK does not offer any insights into power dissipation. To mitigate this limitation, we instead use the more recent E-RANK metric [29], as defined in Equation 10, which incorporates energy consumption E for more detailed and accurate metric.

$$\text{E-RANK} = \frac{10^9/\text{cpb}}{(\text{ROM} + 2 \times \text{RAM}) \times E} \tag{10}$$

4 Results and Analysis

In this section, we present the results of our benchmarking and performance analysis of the selected hash functions. The list of all benchmarked hash functions is provided in Table 1, including an overview of their internal parameters and structure. State refers to the total amount of internal memory allocated for use during the hash function's operation.

The experimental results are visualized using graphs arranged in ascending order for execution time (CPB), memory footprint (both RAM and ROM), and energy consumption, while the composite E-RANK metric is presented in descending order. This arrangement places the most efficient hash function at the top, allowing for quick identification of optimal candidates. Performance values are analyzed in each results' subsection across defined intervals or ranges, enabling a more detailed comparison. By segmenting the data this way, the analysis highlights which hash function perform best within each operational range.

4.1 Execution Time Analysis

The execution time is quantified using the CPB metric. A lower CPB value indicates higher computational efficiency and faster throughput, whereas a higher CPB corresponds to slower processing and lower throughput. Figure 4 illustrates that SKINNY-tk2 exhibits the lowest CPB value, making it the most efficient

Table 1. Overview of selected lightweight hash function

Name	Rate (bits)	Capacity (bits)	State (bits)	Structure	Internal Primitive	Rounds
PHOTON-256 [25]	32	256	288	Extended Sponge	AES-like permutation	12
Lesamnta-LW [27]	128	—	256	Merkle–Damgård (MD)	AES-based block cipher (LW1 mode)	64
BLAKE2s [2]	512	—	256	HAIFA (MD variant)	ChaCha-inspired G function	10
BLAKE3 [35]	512	—	256	Binary Tree	BLAKE2s compression function	7
ASCON [23]	64	256	320	Sponge	Bit-sliced permutation	12/8
PHOTON-Beetle [5]	32	224	256	Sponge	PHOTON-256 permutation	12
Xoodyak [19]	128	256	320	Duplex (Cyclist mode)	3×32-bit slices, XOR/rotate/shift	12
KNOT [44]	32	224	256	Sponge/ Duplex	SPN-style substitution and diffusion	68
ORANGISH [17]	128	128	256	Sponge (JH mode)	PHOTON256 permutation	12
SHAMAS [36]	64	256	320	Sponge/ Duplex	Bit-sliced permutation, linear matrix mixing, byte-wise rotations	12
SIV-Rijndael [6]	32	224	256	Modified Sponge	Rijndael256 permutation	14
SIV-TEM-PHOTON [7]	32	224	256	Modified Sponge	PHOTON-256 permutation	20
SKINNY-tk2 [10]	32	224	256	Sponge	SKINNY-128-256 TK Cipher	48
SNEIKHA [37]	256	256	512	Sponge (BLNK2)	SNEIK f512 ARX Permutation	8
TRIAD [4]	32	224	256	Extended Sponge	Triad-P permutation	1024
Coral [33]	32	224	256	Sponge	πI permutation	10
Gimli [11]	128	256	384	Sponge	Gimli permutation	24
CLX [43]	32	256	288	Sponge	$P'_{288,n}$ NLFSR permutation	var
ACE-H [1]	64	256	320	Sponge (sLiSCP-light)	ACE Permutation (Simeck-style)	48
ESCH [9]	128	256	384	Modified Sponge	ARX-based Sparkle384	var
Subterranean [20]	32 (out) 9 (in)	224	257	Flat Sponge (Duplex)	Bitwise round function	1
Saturnin [16]	256	—	256	MD construction	Saturnin Block Cipher	32
ISAP [22]	144	256	320 / 400	Sponge	Keccak-p[400] and Ascon-p	var
GAGE [24]	8	224	232	Sponge	Custom SPN permutation	32

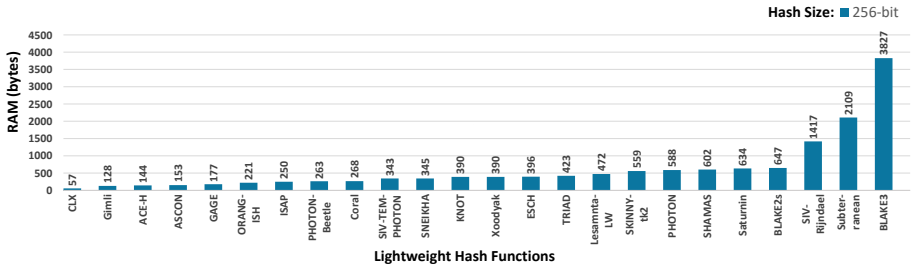

Fig. 5. RAM usage of the lightweight hash function.

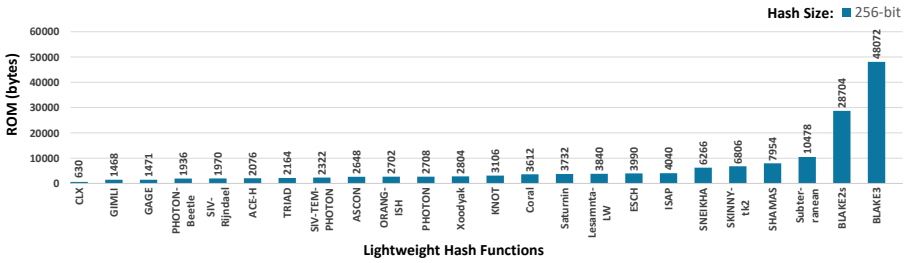

Fig. 6. ROM usage of the lightweight hash functions.

hash function regarding execution speed. It is followed by SIV-Rijndael, which also demonstrates a highly optimized execution time. Within the $\leq$ 1000 CPB range, the ISO-standardized PHOTON-256 also appears just after SIV-Rijndael with a slight margin. In the 1000–2000 CPB range, the NIST-submitted candidates, including CLX and ORANGISH, appear first, while Lesamnta-LW, followed by another ISO-standardized hash function, Lesamnta-LW, with a significant margin. In the 2000–3000 CPB segment, the execution time increases, with PHOTON-Beetle, Gimli, and Xoodyak preceding ASCON, followed by BLAKE3 with a slightly higher margin. In the 3000–4000 CPB range, SIV-TEM-PHOTON appears first, followed by Saturnin, then the SHA-3 competition finalist BLAKE2s. Toward the upper limit, KNOT and ISAP exhibit higher CPB. Finally, Subterranean shows the highest CPB, indicating the lowest execution efficiency among the evaluated lightweight hash functions.

4.2 Memory Footprint Analysis

The memory footprint provides an estimate of the resource consumption of each hash function, evaluated in terms of RAM and ROM requirements.

RAM Consumption: Figure 5 demonstrates that CLX, Gimli, and ACE-H exhibit the lowest RAM usage, making them highly suitable for devices with tight memory constraints. Within the $\leq$ 500 byte range, the ISO-standardized

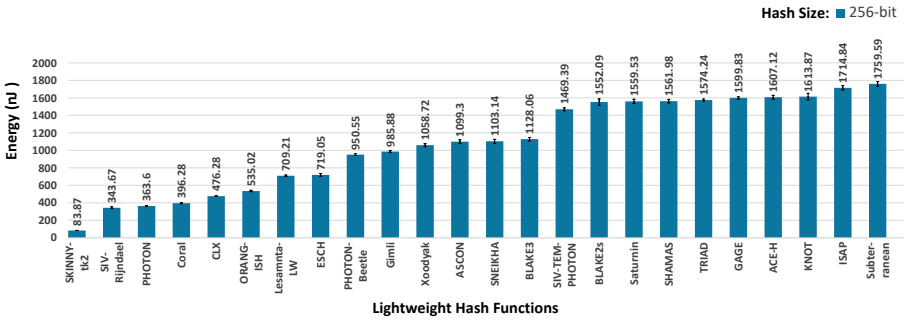

Fig. 7. Energy consumption of selected lightweight hash function.

Lesamnta-LW has the highest RAM usage. In the 500–1000 byte range, the NIST-submitted SKINNY-tk2 and the ISO-standardized PHOTON exhibit moderate RAM requirements, with BLAKE2s near the upper bound of this range and slightly higher than Saturnin. The most memory-intensive functions are BLAKE3, Subterranean, and SIV-Rijndael, which show significantly higher RAM consumption than the rest, with clear separation in their memory requirements.

ROM Consumption: Figure 6 demonstrates that CLX and Gimli lead with the lowest ROM consumption as well. Within the $\leq$ 2000 byte range, SIV-Rijndael exhibits the highest ROM utilization, followed closely by PHOTON-Beetle. In the 2000–4000 byte range, TRIAD and SIV-TEM-PHOTON consume the least, whereas ESCH and the Lesamnta-LW consume the most in this interval. PHOTON lies approximately at the midpoint of this range. Due to fewer candidates beyond this, the next interval spans 4000–10000 byte, where ISAP and SNEIKHA show relatively modest ROM usage. In contrast, SHAMAS exhibits the highest usage in this range. Beyond 10 000 byte, Subterranean has ROM consumption lower than both BLAKE2s and BLAKE3. Both variants of BLAKE are the most resource-intensive in terms of ROM.

These results demonstrate that CLX and Gimli offer highly optimized memory footprints, while BLAKE3 is the most memory-demanding in both RAM and ROM.

4.3 Energy Consumption Analysis

Figure 7 plots the measured energy consumption of the evaluated hash functions, arranged in ascending order. Each data point is the mean of 10 runs; error bars indicate variability across executions. The figure reflects the total energy required to process a fixed-length input and is particularly relevant for battery-powered or energy-constrained embedded systems.

The lowest energy consumption we observed is for SKINNY-tk2, followed by SIV-Rijndael. While both are among the most energy-efficient, SKINNY-tk2 shows a notable margin of efficiency over SIV-Rijndael. In the $\leq$ 500 nJ range,

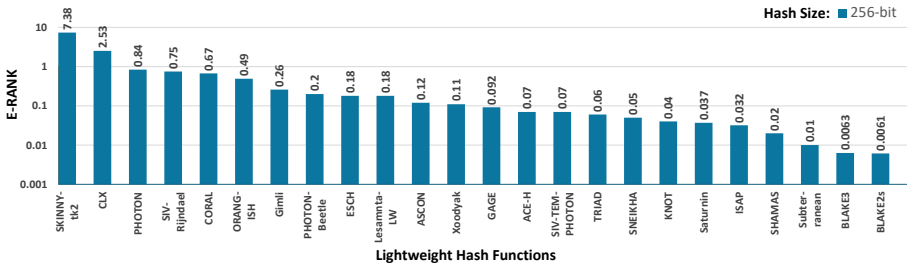

Fig. 8. E-RANK of the studied lightweight hash functions.

CLX and Coral appear at the higher end, whereas PHOTON-256 demonstrates moderate energy consumption, occupying a middle position within this range. Within the 500–1000 nJ range, ORANGISH and Lesamnta-LW exhibit lower energy usage, whereas Gimli and PHOTON-Beetle consume the most energy. In the 1000–1500 nJ interval, Xoodyak and ASCON are positioned as the most efficient, whereas SIV-TEM-PHOTON consumes the highest energy in this range. For hash functions consuming more than 1500 nJ, BLAKE2s and Saturnin are positioned at the lower end of this high-energy group. In contrast, Subterranean and ISAP exhibit the highest energy consumption across all evaluated candidates.

4.4 E-RANK Analysis

Figure 8 presents the E-RANK of the evaluated hash functions. The analysis reveals that SKINNY-tk2 achieves the highest E-RANK by a substantial margin. It is followed by CLX, which also attains an E-RANK greater than 1, demonstrating similarly strong efficiency characteristics. All remaining hash functions fall within the sub-unitary range (E-RANK < 1). Among these, PHOTON-256 exhibits the highest E-RANK, closely followed by SIV-Rijndael, both reflecting favorable trade-offs under constrained-resource settings. Lesamnta-LW and ASCON achieve moderate E-RANK values, suggesting a reasonable balance, although not as optimized as the leading candidates. At the lower end of the spectrum, the BLAKE variants, specifically BLAKE2s, and BLAKE3, record the lowest E-RANK values among all evaluated hash functions.

4.5 Comparative Analysis

Next, we present a comparative analysis of the benchmarked hash functions using a heatmap, which visually represents the efficiency of each metric. For this, we first normalize memory, execution time, and energy consumption to a range between 0 and 1, using the min-max normalization method below, where the value x is a single measurement and X the set of measurements.

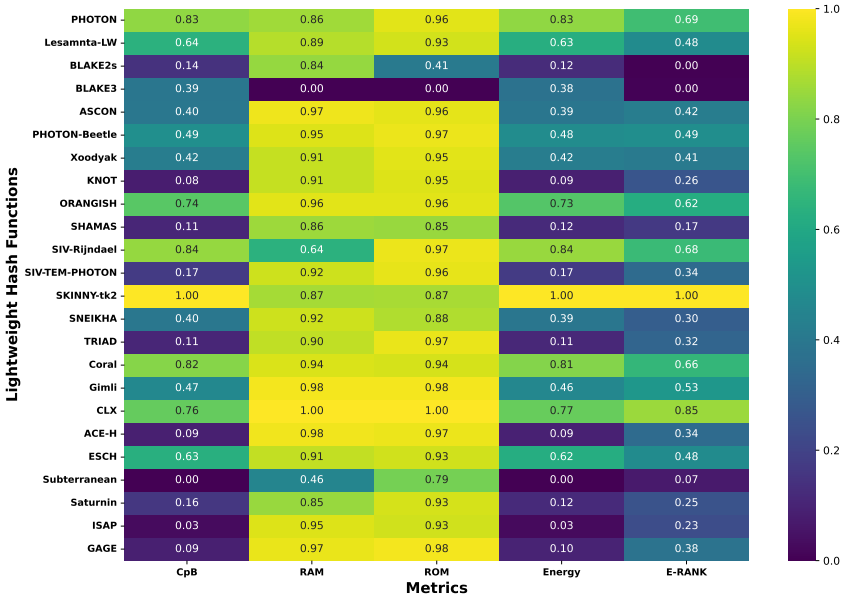

Fig. 9. Normalized Heatmap of selected lightweight hash functions (Higher = Better).

$$\hat{x} = 1 - \frac{x - \min(X)}{\max(X) - \min(X)} \tag{11}$$

For E-RANK, values spans several orders of magnitude with many hash functions clustered closely together around the lower end of the scale. To avoid having these values skewed and compressed by min-max normalization, making them hard to visualize in a heatmap, we here apply a logarithmic transformation using the normalization formula below. The resulting heatmap is shown in Figure 9.

$$\bar{x} = \frac{\log\left(x \times \min(X)\right)}{\log\left(\max(X)/\min(X)\right)} \tag{12}$$

In terms of CPB and energy consumption, the evaluated hash functions SKINNY-tk2, SIV-Rijndael, PHOTON, and Coral exhibit the lowest CPB and energy usage. CPB is directly proportional to energy consumption. These hash functions utilize byte-aligned sponge constructions with fixed and non-branching permutations. They avoid runtime key scheduling, and the round functions consist of lightweight operations, such as 4-bit or 8-bit S-boxes, simple XORs, and linear diffusion layers. In contrast, ISAP, KNOT, and Subterranean have the lowest execution speeds and the highest energy consumption. This is primarily because these designs depend on bit-sliced or bit-level permutations, which consist of complex permutation structures. While these strategies enhance security and hardware optimization, they result in a higher CPB in software, illustrating the trade-off between security and efficiency on constrained devices.

In terms of memory footprint, CLX and Gimli have the least memory usage. This is because these hash functions eliminate the need for dynamic memory allocation by utilizing fixed-size, byte-aligned sponge states, and their permutation cores rely on components such as XOR, rotations, and Boolean logic instead of large S-boxes, lookup tables, or modular arithmetic. In contrast, the hash functions with the highest memory footprints, including BLAKE3, BLAKE2s, and Subterranean 2.0, exhibit structural complexity. BLAKE3 and BLAKE2s utilize complex compression schedules involving numerous constants and key-dependent initialization values, along with tree-based or parallel chunk processing and extendable output. Meanwhile, Subterranean employs bitwise transformations and supports multiple operational wrappers, all of which contribute to high memory requirements.

Some hash functions such as SKINNY-tk2, CLX, PHOTON, and SIV-Rijndael achieve high E-RANK scores due to their low CPB, ROM, RAM, and energy consumption. Note that SKINNY-tk2, SIV-Rijndael, and PHOTON benefit specifically from low CPB and energy usage. In contrast, CLX does not have the fastest execution speed but has the lowest memory footprint among these functions.

Other hash functions, like BLAKE3, BLAKE2s, and Subterranean, show very low E-RANK scores due to their large memory footprints and high energy consumption. For example, the ROM and RAM requirements of BLAKE3 contribute to its lower E-RANK, while Subterranean suffers from poor CPB and energy efficiency combined with high RAM usage, which further reduces its E-RANK. These findings emphasize that overall balanced performance is not determined only by execution speed, but rather by a balanced integration of processing rate, memory, and power metrics. This also suggests that faster execution generally necessitates more memory.

5 Related Work

Few studies focus on benchmarking of software-based lightweight hash functions, other than the status reports published by NIST. There is a particular lack of comparative analysis with other standardized lightweight hash functions such as PHOTON, Lesamnta-LW, or BLAKE. To our knowledge, no other studies provide a quantitative evaluation or analytical assessment of the performance-to-cost tradeoff.

Khan et al. [30] performed benchmarking emphasizing hardware implementations. The study evaluates the throughput-to-area (TP/A) ratio, hardware area utilization, and execution time but does not provide insights into the power analysis or software implementations. Windarta et al. [42] provide a detailed comparative study of NIST submitted hash functions, but the study concludes the results of multiple sources, such as previous research papers and internal evaluations by NIST. However, as a comparative study, it lacks a unified benchmarking approach to standardize performance comparisons.

The NIST status reports on lightweight cryptographic algorithms present software implementation results for NIST lightweight hash functions in detail,

Table 2. Performance metrics calculated values of lightweight hash function.

LWHF	CPB	RAM (B)	ROM (B)	Energy (nJ)	E-RANK
PHOTON	845	588	2708	363.6	0.84
Lesamnta-LW	1604	472	3840	709.21	0.18
BLAKE2s	3516	647	28704	1552.09	0.0061
BLAKE3	2545	3827	48072	1128.06	0.0063
ASCON	2531	153	2648	1099.3	0.12
PHOTON-Beetle	2158	263	1936	950.55	0.2
Xoodyak	2450	390	2804	1058.72	0.11
KNOT	3762	390	3106	1613.87	0.04
ORANGISH	1207	221	2702	535.02	0.49
SHAMAS	3643	602	7954	1561.98	0.02
SIV-Rijndael	811	1417	1970	343.67	0.75
SIV-TEM-PHOTON	3392	343	2322	1469.39	0.07
SKINNY-tk2	204	559	6806	83.87	7.38
SNEIKHA	2533	345	6266	1103.14	0.05
TRIAD	3657	423	2164	1574.24	0.06
Coral	907	268	3612	396.28	0.67
Gimli	2256	128	1468	985.88	0.26
CLX	1116	57	630	476.28	2.53
ACE-H	3728	144	2076	1607.12	0.07
ESCH	1633	396	3990	719.05	0.18
Subterranean	4065	2109	10478	1759.59	0.01
Saturnin	3449	634	3732	1559.53	0.037
ISAP	3953	250	4040	1714.84	0.032
GAGE	3722	177	1471	1599.83	0.092

evaluating execution time, memory footprint, and power consumption on ARM Cortex-M4, ESP32, and AVR ATmega328P [40, 41, 39]. However, these benchmarking results lack fine-grained power profiling, tradeoff analysis between performance and cost, and performance tuning for specific embedded platforms.

6 Conclusions

This study presents a detailed methodology for conducting software benchmarking of lightweight hash functions on an AVR microcontroller using the ChipWhisperer platform. The experimental setup is designed to obtain precise measurements of key performance metrics. CPB are accurately captured by reading the on-chip hardware cycle counter. The memory footprint is derived from post-compilation analysis using the AVR-GCC toolchain. Energy consumption is measured using a power measurement probe connected to the ChipWhisperer's integrated oscilloscope.

A comprehensive comparison between ISO-standardized lightweight hash functions, such as PHOTON and Lesamnta-LW, and NIST-submitted lightweight hash functions, including SHA competition finalists from the BLAKE family, reveals several trends. The hash functions from the ISO and NIST-LWC portfolios show that faster execution generally requires more memory (e.g., SKINNY-tk2),

while designs that focus on security tend to execute more slowly (e.g., ISAP). ISO-standardized designs like PHOTON and Lesamnta-LW were initially created with hardware implementation in mind, but the results show that these hash functions are well-suited for software implementation on 8-bit platforms. Also, many NIST-LWC submissions, such as SKINNY, CLX, and SIV-Rijndael, strike a strong balance across performance metrics. The BLAKE2s and BLAKE3 variants of the BLAKE family have a strong structure for security but tend to perform poorly on AVR-constrained platforms due to their high memory requirements and considerable amount of execution speed and energy consumption. Thus, there is currently no single optimal lightweight hash function for all scenarios; the best choice depends on specific application requirements and the architecture of different microcontrollers for real-world deployments.

References

1. Aagaard, M., AlTawy, R., Gong, G., Mandal, K., Rohit, R.: Ace: An authenticated encryption and hash algorithm. Submission document, NIST Lightweight Cryptography Project (Mar 2019), https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/round-1/spec-doc/ace-spec.pdf, round 1 Submission for NIST Lightweight Cryptography Standardization Process
2. Aumasson, J.P., Meier, W., Phan, R.C.W., Henzen, L.: BLAKE2, pp. 165–183. Springer Berlin Heidelberg, Berlin, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44757-4_9, https://doi.org/10.1007/978-3-662-44757-4_9
3. Badel, S., Dağtekin, N., Nakahara Jr, J., Ouafi, K., Reffé, N., Sepehrdad, P., Sušil, P., Vaudenay, S.: Armadillo: a multi-purpose cryptographic primitive dedicated to hardware. In: International Workshop on Cryptographic Hardware and Embedded Systems. pp. 398–412. Springer (2010)
4. Banik, S., Isobe, T., Meier, W., Todo, Y., Zhang, B.: Triad v1 – a lightweight aead and hash function based on stream cipher. Submission document, NIST Lightweight Cryptography Project (Mar 2019), https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/round-1/spec-doc/TRIAD-spec.pdf, round 1 Submission for NIST Lightweight Cryptography Standardization Process
5. Bao, Z., Chakraborti, A., Datta, N., Guo, J., Nandi, M., Peyrin, T., Yasuda, K.: Photon-beetle authenticated encryption and hash family (2020). Submission to the NIST Lightweight Competition (2021)
6. Bao, Z., Guo, J., Iwata, T., Song, L.: Siv-rijndael256: Authenticated encryption and hash family. Submission document, NIST Lightweight Cryptography Project (Feb 2019), https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/round-1/spec-doc/SIV-Rijndael256-Spec.pdf, round 1 Submission for NIST Lightweight Cryptography Standardization Process
7. Bao, Z., Guo, J., Iwata, T., Song, L.: Siv-tem-photon: Authenticated encryption and hash family. Submission document, NIST Lightweight Cryptography Project (Feb 2019), https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/round-1/spec-doc/SIV-TEM-PHOTON-Spec.pdf, round 1 Submission for NIST Lightweight Cryptography Standardization Process

8. Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., Wingers, L.: The Simon and Speck block ciphers on AVR 8-bit microcontrollers. In: Eisenbarth, T., Öztürk, E. (eds.) Lightweight Cryptography for Security and Privacy. pp. 3–20. Springer International Publishing, Cham (2015)
9. Beierle, C., Biryukov, A., dos Santos, L.C., Großschädl, J., Perrin, L., Udovenko, A., Velichkov, V., Wang, Q.: Schwaemm and esch: Lightweight authenticated encryption and hashing using the sparkle permutation family. Submission document, NIST Lightweight Cryptography Project (Mar 2019), https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/round-1/spec-doc/SPARKLE-spec.pdf, round 1 Submission for NIST Lightweight Cryptography Standardization Process
10. Beierle, C., Jean, J., Kölbl, S., Leander, G., Moradi, A., Peyrin, T., Sasaki, Y., Sasdrich, P., Sim, S.M.: Skinny-aead and skinny-hash: Authenticated encryption and hashing using the skinny block cipher. Submission document, NIST Lightweight Cryptography Project (2019), https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/round-1/spec-doc/SKINNY-spec.pdf, round 1 Submission for NIST Lightweight Cryptography Standardization Process
11. Bernstein, D.J., Kölbl, S., Lucks, S., Massolino, P.M.C., Mendel, F., Nawaz, K., Schneider, T., Schwabe, P., Standaert, F.X., Todo, Y., Viguier, B.: Gimli: A cross-platform permutation for hashing and authenticated encryption. Submission document, NIST Lightweight Cryptography Project (Mar 2019), https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/round-1/spec-doc/gimli-spec.pdf, round 1 Submission for NIST Lightweight Cryptography Standardization Process
12. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Keccak. In: Johansson, T., Nguyen, P.Q. (eds.) Advances in Cryptology – EUROCRYPT 2013. pp. 313–314. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
13. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: Sponge functions. In: ECRYPT Hash Workshop (2007), https://keccak.team/files/SpongeFunctions.pdf
14. Bogdanov, A., Knežević, M., Leander, G., Toz, D., Varıcı, K., Verbauwhede, I.: spongent: A lightweight hash function. In: Preneel, B., Takagi, T. (eds.) Cryptographic Hardware and Embedded Systems – CHES 2011. pp. 312–325. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
15. Bogdanov, A., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y.: Hash functions and rfid tags: Mind the gap. In: Oswald, E., Rohatgi, P. (eds.) Cryptographic Hardware and Embedded Systems – CHES 2008. pp. 283–299. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)
16. Canteaut, A., Duval, S., Leurent, G., Naya-Plasencia, M., Perrin, L., Pornin, T., Schrottenloher, A.: Saturnin: A suite of lightweight symmetric algorithms for post-quantum security. Submission document, NIST Lightweight Cryptography Project (Mar 2019), https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/round-1/spec-doc/SATURNIN-spec.pdf, round 1 Submission for NIST Lightweight Cryptography Standardization Process
17. Chakraborty, B., Nandi, M.: ORANGE: Algorithm specifications and supporting document. Submission document, NIST Lightweight Cryptography Project (Mar 2019), https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/round-1/spec-doc/orange-spec.pdf, round 1 Submission for NIST Lightweight Cryptography Standardization Process

18. Chang, S.j., Perlner, R., Burr, W.E., Turan, M.S., Kelsey, J.M., Paul, S., Bassham, L.E.: Third-round report of the sha-3 cryptographic hash algorithm competition. NIST Interagency Report **7896**, 121 (2012)
19. Daemen, J., Hoffert, S., Peeters, M., Van Assche, G., Van Keer, R.: Xoodyak, a lightweight cryptographic scheme. IACR Transactions on Symmetric Cryptology pp. 60–87 (2020)
20. Daemen, J., Massolino, P.M.C., Rotella, Y.: The subterranean 2.0 cipher suite. Submission document, NIST Lightweight Cryptography Project (Mar 2019), `https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/round-1/spec-doc/subterranean-spec.pdf`, round 1 Submission for NIST Lightweight Cryptography Standardization Process
21. Damgård, I.B.: A design principle for hash functions. In: Brassard, G. (ed.) Advances in Cryptology — CRYPTO' 89 Proceedings. pp. 416–427. Springer New York, New York, NY (1990)
22. Dobraunig, C., Eichlseder, M., Mangard, S., Mendel, F., Mennink, B., Primas, R., Unterluggauer, T.: Isap v2.0: Submission to the nist lightweight cryptography standardization process. Submission document, NIST Lightweight Cryptography Project (Mar 2019), `https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/round-1/spec-doc/ISAP-spec.pdf`, round 1 Submission for NIST Lightweight Cryptography Standardization Process
23. Dobraunig, C., Eichlseder, M., Mendel, F., Schläffer, M.: Ascon v1. 2: Lightweight authenticated encryption and hashing. Journal of Cryptology **34**, 1–42 (2021)
24. Gligoroski, D., Mihajloska, H., Otte, D.: Gage and ingage v1.0: Submission to the nist lightweight cryptography standardization process. Submission document, NIST Lightweight Cryptography Project (Mar 2019), `https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/round-1/spec-doc/GAGEandInGAGE-spec.pdf`, round 1 Submission for NIST Lightweight Cryptography Standardization Process
25. Guo, J., Peyrin, T., Poschmann, A.: The PHOTON family of lightweight hash functions. In: Rogaway, P. (ed.) Advances in Cryptology – CRYPTO 2011. pp. 222–239. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
26. Guo, J., Peyrin, T., Poschmann, A., Robshaw, M.: The LED block cipher. In: Preneel, B., Takagi, T. (eds.) Cryptographic Hardware and Embedded Systems – CHES 2011. pp. 326–341. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
27. Hirose, S., Ideguchi, K., Kuwakado, H., Owada, T., Preneel, B., Yoshida, H.: A lightweight 256-bit hash function for hardware and low-end devices: Lesamnta-lw. In: Rhee, K.H., Nyang, D. (eds.) Information Security and Cryptology - ICISC 2010. pp. 151–168. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
28. International Organization for Standardization: Iso/iec 29192-5:2016 information technology — security techniques — lightweight cryptography — part 5: Hash-functions (2016), `https://www.iso.org/standard/67173.html`, accessed: 2024-10-22
29. Khan, M., Johansen, D., Dagenborg, H.: Performance evaluation of lightweight cryptographic ciphers on ARM processor for IoT deployments. In: Zhao, J., Meng, W. (eds.) Science of Cyber Security. pp. 254–272. Springer Nature Singapore, Singapore (2025)
30. Khan, S., Lee, W.K., Karmakar, A., Mera, J.M.B., Majeed, A., Hwang, S.O.: Area–time efficient implementation of nist lightweight hash functions targeting iot applications. IEEE Internet of Things Journal **10**(9), 8083–8095 (2023). https://doi.org/10.1109/JIOT.2022.3229516

31. Merkle, R.C.: A certified digital signature. In: Brassard, G. (ed.) Advances in Cryptology — CRYPTO' 89 Proceedings. pp. 218–238. Springer New York, New York, NY (1990)
32. Microchip Technology Inc.: ATxmega16D4/32D4/64D4/128D4 8/16-bit AVR Microcontroller Datasheet. Atmel Corporation (Jun 2014), https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-8135-8-and-16-bit-AVR-microcontroller-ATxmega16D4-32D4-64D4-128D4_datasheet.pdf, document No. Atmel-8135, Revision H
33. Montes, M., Penazzi, D.: Yarará and coral v1: Lightweight authenticated encryption and hash function algorithms. Submission document, NIST Lightweight Cryptography Project (Mar 2019), https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/round-1/spec-doc/yarara_and_coral-spec.pdf, round 1 Submission for NIST Lightweight Cryptography Standardization Process
34. NewAE Technology Inc.: Newae hardware product documentation (2025), https://rtfm.newae.com/, accessed: 2025-03-03
35. O'Connor, J., Aumasson, J.P., Neves, S., Wilcox-O'Hearn, Z.: BLAKE3: one function, fast everywhere. https://github.com/BLAKE3-team/BLAKE3-specs/blob/master/blake3.pdf (Jan 2020), accessed: 2025-03-24
36. Penazzi, D., Montes, M.: SHAMASH & SHAMASHASH: Lightweight authenticated encryption and hash function algorithms. Submission document, NIST Lightweight Cryptography Project (2019), https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/round-1/spec-doc/ShamashAndShamashash-spec.pdf, round 1 Submission for NIST Lightweight Cryptography Standardization Process
37. Saarinen, M.J.O.: Sneiken and sneikha: Authenticated encryption and cryptographic hashing. Submission document, NIST Lightweight Cryptography Project (Mar 2019), https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/round-1/spec-doc/sneik-spec.pdf, round 1 Submission for NIST Lightweight Cryptography Standardization Process
38. of Standards, N.I., Technology: Lightweight cryptography project. https://csrc.nist.gov/projects/lightweight-cryptography (nd), accessed: 2024-10-22
39. Turan, M.S., McKay, K.A., Çalik, Ç., Chang, D., Bassham, L., et al.: Status report on the first round of the nist lightweight cryptography standardization process. National Institute of Standards and Technology, Gaithersburg, MD, NIST Interagency/Internal Rep.(NISTIR) **108** (2019)
40. Turan, M.S., Turan, M.S., McKay, K., Chang, D., Bassham, L.E., Kang, J., Waller, N.D., Kelsey, J.M., Hong, D.: Status report on the final round of the NIST lightweight cryptography standardization process. US Department of Commerce, National Institute of Standards and Technology (2023)
41. Turan, M.S., Turan, M.S., McKay, K., Chang, D., Calik, C., Bassham, L., Kang, J., Kelsey, J.: Status report on the second round of the nist lightweight cryptography standardization process (2021)
42. Windarta, S., Suryadi, S., Ramli, K., Pranggono, B., Gunawan, T.S.: Lightweight cryptographic hash functions: Design trends, comparative study, and future directions. IEEE Access **10**, 82272–82294 (2022). https://doi.org/10.1109/ACCESS.2022.3195572
43. Wu, H., Huang, T.: Clx: A family of lightweight authenticated encryption algorithms. Submission document, NIST Lightweight Cryptography Project (Mar 2019), https://csrc.nist.gov/CSRC/media/Projects/

Lightweight-Cryptography/documents/round-1/spec-doc/CLX-spec.pdf, round 1 Submission for NIST Lightweight Cryptography Standardization Process

44. Zhang, W., Ding, T., Yang, B., Bao, Z., Xiang, Z., Ji, F., Zhao, X.: KNOT: Algorithm specifications and supporting document. Submission document, NIST Lightweight Cryptography Project (2019), https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/round-1/spec-doc/KNOT-spec.pdf, round 1 Submission for NIST Lightweight Cryptography Standardization Process