Software Benchmarking of NIST Lightweight Hash Function Finalists on Resource-Constrained AVR Platform via ChipWhisperer

Mohsin Khan [1][a], Håvard Dagenborg,[1][b] and Dag Johansen [1][c]
[1] *UiT The Arctic University of Norway, Tromsø, Norway*
{*mohsin.khan, havard.dagenborg, dag.johansen*}*@uit.no*

Keywords: NIST, Lightweight Hash Functions, Software Benchmarking

Abstract: This paper presents a novel performance evaluation of five key lightweight hash functions on an ATxmega128 microcontroller, using our E-RANK metric as a composite metric that integrates execution speed, memory footprint, and energy efficiency into a unified and balanced ranking. We leverage hardware-specific profiling techniques, where counter registers are accessed directly on the microcontroller to measure execution speed and analyze RAM and ROM footprints post-compilation to determine memory usage. Energy consumption is measured using the ChipWhisperer FPGA toolkit, capturing voltage traces via an oscilloscope probe. Our evaluations provide new insights into the trade-offs inherent in each lightweight hash function, providing guidance on which one is most suitable for various application-specific constraints.

1 INTRODUCTION

Cryptographic hash functions are essential components in many security mechanisms. They provide data integrity, verify authenticity, and prevent repudiation. Traditional hash functions are, however, not optimized for resource-constrained devices, such as those commonly used in Internet of Thing (IoT) and RFID applications. As an alternative, many specialized Lightweight Hash Functions (LWHFs), like PHOTON (Guo et al., 2011) and SPONGENT (Huang et al., 2021), have been proposed and standardized under ISO/IEC 29192-5 for use in resource-constrained environments.

In 2019, NIST called for the standardization of lightweight cryptographic algorithms, receiving 57 submissions for consideration. Among the submissions, many cryptographic algorithms included their associated LWHFs, including the winner ASCON (Dobraunig et al., 2021). The NIST reports on lightweight cryptographic algorithms include software implementation results for the NIST standardized LWHFs, evaluating execution time, memory footprint, and power consumption on ARM Cortex-M4, ESP32, and AVR ATmega328P (Turan et al., 2019; Turan et al., 2021; Turan et al., 2023). However, these benchmarking results lack fine-grained power profiling, trade-off analysis between performance and cost, and performance tuning for specific embedded platforms. Others (Windarta et al., 2022) have provided a comparative analysis of NIST LWHFs but compile results obtained through different methodologies without ensuring direct comparability or focusing on hardware implementations (Khan et al., 2023) without providing a unified benchmarking approach to standardize performance comparisons.

In this paper, we benchmark the LWHFs from the five finalists in the NIST lightweight cryptography standardization process, evaluating their performance based on Cycles per Byte (CPB), RAM and ROM footprint, and energy consumption. Our experimental setup includes a novel hardware profiling approach that combines the NewAE ChipWhisperer platform with an ATxmega128 microcontroller. This setup enables high-resolution power analysis and energy profiling, making it possible to accurately measure the computational efficiency of each LWHF. We use our novel E-RANK metrics to assess and compare the overall efficiency of the LWHFs, which balances the tradeoff between the various measurement types. Our measurements show that the selected LWHFs have different performance characteristics, and the selection of which one to use is highly application dependent.

[a] https://orcid.org/0000-0003-1815-8642
[b] https://orcid.org/0000-0002-1637-7262
[c] https://orcid.org/0000-0001-7067-6477

Table 1: Comparison of the structure of finalist LWHFs from the NIST lightweight cryptography competition.

LWHF	Internal Parameters			Structure (Sponge Construction)			Output Hash
	Rate (bits)	State (bits)	Capacity (bits)	Permutation	Round Structure	Rounds	(bits)
ASCON	64	320	256	Ascon-p	SPN	12/8	256
PHOTON-Beetle	32	256	224	PHOTON	AES-like SPN	12	256
Xoodyak	128	384	256	Xoodoo	SPN with ARX based Plane Diffusion	12	256
SPARKLE	128	384	256	SPARKLE-384	ARX-based SPN	var	256
ISAP	var	320 / 400	var	Ascon-p / Keccak-p[400]	SPN	var	256

2 BACKGROUND

This section provides an overview of the five NIST lightweight cryptography finalists that we study in this paper, summarized in Table 1. Note that all are based on a Sponge construction approach.

ASCON (Dobraunig et al., 2021) uses a 320-bit internal state, a 64-bit rate for input, and a 256-bit capacity for cryptographic strength. During initialization, a predefined IV is loaded, and a 12-round Ascon-p permutation is applied. Messages are processed in 64-bit blocks, XORed into the rate, followed by an 8 or 12-round permutation. Once all input is processed, a final 12-round permutation occurs before extracting the digest. For longer outputs, extra squeezing rounds are done.

PHOTON-Beetle (Bao et al., 2021) utilizes a sponge construction based on the PHOTON-256 permutation. The hash function has a 256-bit internal state, split into 32-bit rate and 224-bit capacity, balancing speed and security. During initialization, the state is set to zero, and the first 128-bit input block is absorbed. The absorption phase processes the message in 32-bit blocks, each XORed with the rate portion, followed by 12 rounds of the PHOTON-256 permutation, which includes four core operations: AddConstant, SubCells (4-bit S-box), ShiftRows, and MixColumnSerial, ensuring diffusion and cryptographic strength. The squeezing phase derives the final hash output by applying another round of PHOTON-256 and extracting two 128-bit parts from the state.

Xoodyak (Daemen et al., 2020) function is based on the Xoodoo permutation. Its internal state consists of 384 bits, split into 128 bits for the rate and 256 bits for capacity. Xoodyak employs a duplex sponge structure for hashing and authenticated encryption. Initialization sets the internal state to zero. During absorption, input blocks XOR with the state's rate portion, followed by 12 rounds of the Xoodoo permutation, which involves bitwise mixing, a non-linear layer, and diffusion transformations. Finally, in the squeezing phase, the hash output is taken from the state, with an option for extendable output length.

ESCH (Beierle et al., 2020) is based on a sponge construction and features an Addition-Rotation-XOR (ARX) design. ESCH processes messages iteratively, padding and absorbing them into an internal state before transforming them via SPARKLE-384 (Esch256) or SPARKLE-512 (Esch384). These permutations use the Alzette ARX-box, a lightweight cryptographic primitive that ensures non-linearity and diffusion while remaining efficient in constrained environments. The internal state undergoes multiple transformation rounds, combining modular addition, bitwise rotation, and XOR operations to enhance security. After absorbing all input data, a finalization step produces the 256-bit output (Esch256) or 384-bit output (Esch384).

ISAP (Dobraunig et al., 2017) employs a sponge-based construction approach, using Ascon-p or Keccak-p[400] permutation. The hash function features an iterative permutation structure where input is absorbed into a fixed-size internal state, processed via multiple rounds of the chosen permutation, and then squeezed to generate the final hash. The state begins with a predefined value. Input blocks are XORed into the state's rate portion during absorption, followed by a permutation round for diffusion. Finalization applies extra rounds before output extraction.

3 RESEARCH METHODOLOGY

3.1 Measurements

To obtain precise CPB, RAM, ROM, and energy consumption measurements, we are using the open-source ChipWhisperer[1] platform by NewAE Technology. ChipWhisperer is a suite of hardware and software tools created to find security vulnerabilities in embedded systems. Its modular structure

[1]https://www.newae.com/chipwhisperer

enables the integration of specialized modules, enabling precise measurements and advanced cryptographic testing. The Chipwhisperer uses features such as high-precision oscilloscope integration, triggering mechanisms, and clock-glitching capabilities. The ChipWhisperer-lite variant of the toolkit consists of a *capture* board and a *target* board.

The capture board consists of an Xilinx Spartan-6 LX9 FPGA, an ARM Cortex-M3 (ATSAM3U2C) microcontroller, an AD9215 BRUZ-105 Analog-to-Digital converter (ADC), and an AD8331ARQZ Low-Noise Amplifier (LNA). The FPGA serves as the core processing unit responsible for data acquisition and clock synchronization. The ARM Cortex-M3 manages USB communication between ChipWhisperer and a host PC to provide command execution and communication protocols. In our case, we use the Universal Asynchronous Receiver-Transmitter (UART) to program the target board and send trigger signals. The LNA amplifies weak power traces from the target board before forwarding them to the ADC, offering an adjustable gain from 0 to 55 dB.

In our experimental setup, we are using an ATxmega128 microcontroller as our target board, mounted on a Universal Feature Observation (UFO) board, which provides a standardized interface for power, clock, and data connections. The oscilloscope probe connects the capture board and the UFO Board, capturing voltage glitches upon triggering. The FPGA on the capture board controls and synchronizes the oscilloscope.

3.2 Experiment Overview

Our experiment consists of three main stages.

Stage 1: Firmware Development: The firmware development for the target board is carried out in the C language, with the LWHF implementations sourced from the NIST website. A base C program is developed to enable simple serial communication with the ChipWhisperer Python API, enabling trigger signal handling and the execution of specific hash functions. The selected LWHFs are integrated into the base C program through macros, allowing for a flexible selection of different algorithms during compilation.

The firmware compilation process utilizes the AVR-GCC compiler, incorporating compiler flags for object files and macros that specify the hash function name and variant. During compilation, the C implementation files of the chosen LWHFs are translated into object files, which are then linked together to generate the `.elf` and `.map` files. The `.elf` is then converted to `.hex` file for flashing on the target. Also, the compiler is configured to generate `.su` files for each object file, providing stack usage. Finally, the ChipWhisperer Python API uses the generated `.hex` file to program the ATxmega128 target device.

Stage 2: Target Flashing: The ChipWhisperer capture board is initialized by configuring the scope and setting its key parameters, including gain, sample rate, trigger settings, and clock sources. Once the scope is initialized, the selected Programmer is used to flash the compiled `.hex` file onto the ATxmega128 target board. After flashing, the firmware is verified and debugged.

Stage 3: Evaluation Metrics Measurement This stage involves initializing and configuring simple serial communication and the scope, which are then used to initialize the target device by linking it with the scope and serial interface. Once the initialization is complete, the system determines the specific performance metric to be measured, selecting from RAM usage, ROM usage, CPB, or energy consumption.

3.3 Evaluation Metrics

The performance metrics used for benchmarking the selected LWHFs are as follows.

Cycles per Byte (CPB) represents the number of processor cycles required to process each byte of input data. The CPB measurement is conducted using a hardware-specific technique that utilizes the `TCC0.CTRLA` register (Control Register for Timer/Counter C0) on the ATxmega128 target board. It is responsible for configuring the clock source and prescaler settings.

To measure execution cycles, the `TCC0.CTRLA` register is initialized to zero at the start of the hashing operation, and as the selected LWHF processes the input data, the timer records the total clock cycles consumed. At the end of the hashing process, the register is accessed to retrieve the final cycle count, which is then divided by the number of input bytes to compute the CPB value. The calculated CPB is transmitted to the host system via simple serial communication.

RAM is determined by summing the initialized and uninitialized global/static variables along with the stack memory utilized by the LWHF. The initialized (`.data`) and uninitialized (`.bss`) segments are extracted from the `.map` file, which is generated during the linking stage of compilation using AVR-GCC.

Also, both dynamic and static stack usage are obtained from the `.su` file, which is created during the compilation of each object file, provides a detailed breakdown of memory allocation.

ROM is determined by adding the memory occupied by the program code `(.text)` and constant data `(.rodata)`. The `.map` file, generated during the linking stage of compilation, provides a detailed memory map of the program code and constant data, allowing precise measurement of the total ROM footprint.

Energy is measured using ChipWhisperer's API function `cw.capture_trace`, which returns an array of instantaneous voltage samples acquired through the ADC when a high-to-low trigger signal is detected from the target device. The obtained samples are normalized values in the range -0.5 to 0.5 and must be converted into actual voltages for energy calculations. Given a raw ADC voltage sample V_{raw}, as returned by the API, the ADC reference voltage V_{ref}, and the gain G_{amp} of the amplifier applied to the signal before digitization, the actual voltage V_{actual} is given as follows.

$$V_{\text{actual}} = V_{\text{raw}} \times \frac{V_{\text{ref}}}{G_{\text{amp}}} \tag{1}$$

Given the shunt resistance R_{shunt} and the supply voltage V_{sup}, Ohm's Law gives us the instantaneous power as follows.

$$P_{\text{trace}} = I_{\text{trace}} \times V_{\text{sup}} = \frac{V_{\text{actual}}}{R_{\text{shunt}}} \times V_{\text{sup}} \tag{2}$$

Because the ADC normalized samples may end up negative due to signal oscillations or AC coupling effects, we use the Root Mean Square (RMS) power of a trace to obtain average power and ensure precise energy calculations. Given N samples, the output RMS power P_{rms} of a trace is given as follows.

$$P_{\text{rms}} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} P_{\text{trace},i}^2} \tag{3}$$

The time taken by the microcontroller to execute a hash function is given by $T_{\text{exec}} = C/f_{\text{clk}}$, where C represents the total number of cycles utilized during the execution of the hash function, and f_{clk} denotes the clock frequency of the microcontroller. This gives us the energy consumption E as a function of power and time as follows.

$$E = P_{\text{rms}} \times T_{\text{exec}} = \frac{P_{\text{rms}} \times C}{f_{\text{clk}}} \tag{4}$$

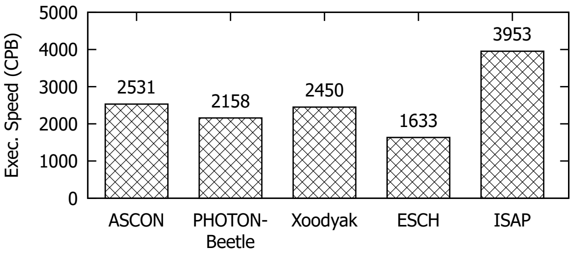

Figure 1: Execution speed (CPB) of selected LWHFs.

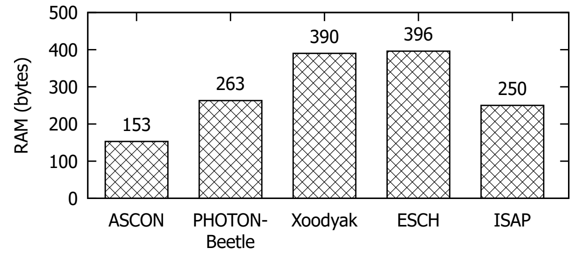

Figure 2: RAM usage of selected LWHFs.

E-RANK is our improved variant of the RANK metric (Beaulieu et al., 2015) that combines execution time, memory footprint, and energy consumption into a single metric. Although the RANK metric has previously been used as a performance trade-off metric to evaluate execution efficiency relative to resource consumption, it does not account for power dissipation. Our modified version incorporates energy consumption, as shown in Equation 5.

$$\text{E-RANK} = \frac{10^9/\text{CPB}}{(\text{ROM} + 2 \times \text{RAM}) \times E} \tag{5}$$

4 RESULTS AND ANALYSIS

The measured execution speeds, in terms of CPB, are shown in Figure 1. We found that ESCH runs fastest, followed by PHOTON-Beetle, Xoodyak, and ASCON. ISAP demonstrates the highest execution time by a large margin.

For memory consumption, ASCON exhibits the smallest RAM footprint followed by ISAP and PHOTON-Beetle, as can be seen in Figure 2. ESCH, with Xoodyak closely trailing, has the highest RAM usage. ROM usage is shown in Figure 3. PHOTON-Beetle requires the least ROM, followed by ASCON and Xoodyak, while ISAP and ESCH have the highest ROM allocation.

The measured energy consumption is shown in Figure 4. Each data point is the mean of 10 runs, and the plotted error bars indicate insignificant variability across executions. The figure highlights that ESCH and PHOTON-Beetle are the most energy-efficient

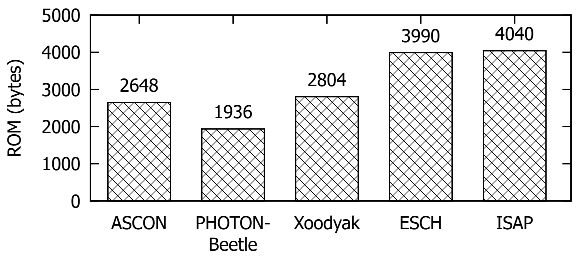

Figure 3: ROM usage of selected LWHFs.

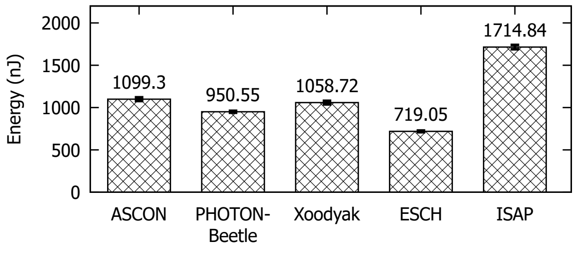

Figure 4: Energy consumption of selected LWHFs.

LWHFs, while ISAP registers the highest energy consumption. ASCON and Xoodyak have a moderate and similar level of energy consumption, with only a slight difference.

The calculated E-RANK is plotted in Figure 5. As can be seen in the figure, PHOTON-Beetle outperforms all other candidates, followed by ESCH. ASCON and Xoodyak are positioned in the middle with a slight difference in their E-Rank. On the lower end, ISAP has the lowest E-RANK values.

A comparative analysis of the LWHFs based on execution efficiency, memory footprint, energy consumption, and overall efficiency (i.e., E-RANK) can be found in Figure 6. Here, we normalize each metric to a common scale to prevent any single one from dominating the radar chart visualization. CPB is normalized by dividing the highest CPB value by each LWHF's CPB, ensuring faster execution yields a higher score. Energy, RAM, and ROM are normalized using their respective minimum values, so lower resource consumption translates to a higher normalized score. Since lower values for CPB, energy consumption, RAM, and ROM indicate better performance, these metrics are inverted and appropriately scaled. E-RANK is a positive metric, so it is scaled by dividing each value by the highest observed value.

From Figure 6, we observe clear trade-offs, correlations, and dominant trends among the benchmarked LWHFs. The ESCH LWHF offers the fastest execution speed and least energy consumption, but has higher RAM and ROM usage. ASCON and PHOTON-Beetle consume low memory footprints with moderate energy consumption and execution speed. Xoodyak consumes moderate amounts of en-

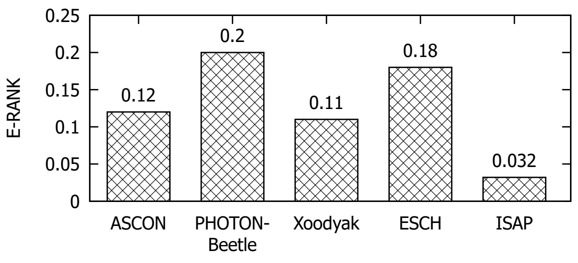

Figure 5: Calculated E-RANK of selected LWHFs (Higher values indicate better overall trade-off performance).

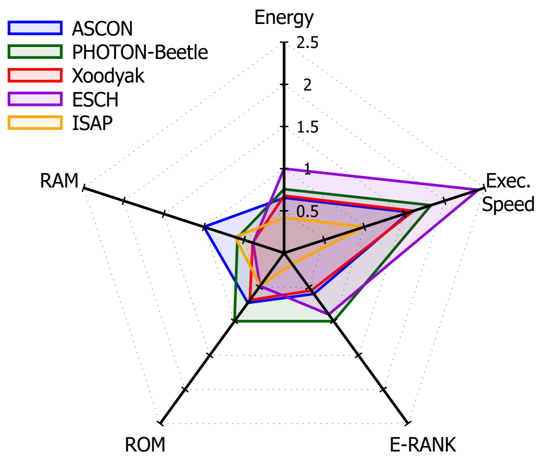

Figure 6: Multi-metric comparison of selected LWHFs.

ergy, execution speed, and RAM, but its RAM usage is higher than that of ASCON, PHOTON-Beetle, and ISAP. In addition, ISAP exhibits the highest CPB, energy, and ROM, highlighting its design as a security-focused function rather than one aimed at efficiency. This highlights a key trade-off: faster execution generally demands more memory, and designs that prioritize security execute slowly.

Among evaluated LWHFs, the best overall balance among speed, energy, and memory efficiency is achieved by PHOTON-Beetle, which has the highest E-RANK. PHOTON-Beetle is followed by ESCH due to its highest execution speed and lowest energy requirement. Although ESCH boasts a high execution speed, its E-RANK lies after PHOTON-Beetle because of its increased memory requirements. ASCON has a low memory requirement, but lies in third place in terms of E-RANK due to its low execution speed and moderate level of energy consumption. Xoodyak has a lower E-RANK because it offers a mix of moderate energy efficiency and execution speed, requiring more RAM than ASCON and PHOTON-Beetle. ISAP ranks the lowest in E-RANK, indicating that its focus on security comes at a trade-off, as explained earlier.

As is clear from Figure 6, there is not a single op-

Table 2: Application-specific suitability of LWHFs.

Application	Recommended LWHF
Low-Power (Battery-powered, energy-efficient)	ESCH
Memory-Constrained (Limited ROM/RAM availability)	PHOTON-Beetle ASCON
High-Speed Execution (Real-time, high-throughput processing)	ESCH
Security critical (Robust against side-channel attacks)	ISAP

timal LWHF for every scenario. This suggests that the choice of hash function is based on the specific application requirements. Also, different microcontroller (MCU) architectures impact the choice of an appropriate LWHF for real-world deployments. Table 2 outlines the most suitable LWHFs for various application scenarios best suited for each use case.

5 CONCLUSION

This paper presents a comprehensive benchmarking of the five finalist LWHFs from the NIST standardization process using the ChipWhisperer platform with an ATxmega128 microcontroller as the target. We evaluate key performance metrics including CPB, RAM usage, ROM footprint, and energy consumption, using hardware-specific profiling and optimized compilation techniques that ensure precise and reliable measurements. Our measurements reveal inherent trade-offs among the different LWHFs, where achieving faster execution often comes at the cost of higher memory and energy consumption. This emphasizes that the selection of an optimal LWHF is highly application-dependent.

ACKNOWLEDGEMENTS

The authors acknowledge the use of AI tools in preparing this paper, including Grammarly for grammar and style correction and OpenAI ChatGPT and Microsoft CoPilot for refining certain sections. The final content has been reviewed for accuracy and alignment with the research contributions.

REFERENCES

Bao, Z., Chakraborti, A., Datta, N., Guo, J., Nandi, M., Peyrin, T., and Yasuda, K. (2021). Photon-beetle authenticated encryption and hash family (2020). *Submission to the NIST Lightweight Competition.*

Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., and Wingers, L. (2015). The Simon and Speck block ciphers on AVR 8-bit microcontrollers. In Eisenbarth, T. and Öztürk, E., editors, *Lightweight Cryptography for Security and Privacy*, pages 3–20, Cham. Springer International Publishing.

Beierle, C., Biryukov, A., Cardoso dos Santos, L., Großschädl, J., Perrin, L., Udovenko, A., Velichkov, V., and Wang, Q. (2020). Lightweight aead and hashing using the sparkle permutation family. *IACR Transactions on Symmetric Cryptology*, 2020(S1):208–261.

Daemen, J., Hoffert, S., Peeters, M., Van Assche, G., and Van Keer, R. (2020). Xoodyak, a lightweight cryptographic scheme. *IACR Transactions on Symmetric Cryptology*, pages 60–87.

Dobraunig, C., Eichlseder, M., Mangard, S., Mendel, F., and Unterluggauer, T. (2017). Isap–towards side-channel secure authenticated encryption. *IACR Transactions on Symmetric Cryptology*, pages 80–105.

Dobraunig, C., Eichlseder, M., Mendel, F., and Schläffer, M. (2021). Ascon v1. 2: Lightweight authenticated encryption and hashing. *Journal of Cryptology*, 34:1–42.

Guo, J., Peyrin, T., and Poschmann, A. (2011). The photon family of lightweight hash functions. In Rogaway, P., editor, *Advances in Cryptology – CRYPTO 2011*, pages 222–239, Berlin, Heidelberg. Springer Berlin Heidelberg.

Huang, Y., Li, S., Sun, W., Dai, X., and Zhu, W. (2021). HVH: A lightweight hash function based on dual pseudo-random transformation. In *Security, Privacy, and Anonymity in Computation, Communication, and Storage*, pages 492–505, Cham. Springer International Publishing.

Khan, S., Lee, W.-K., Karmakar, A., Mera, J. M. B., Majeed, A., and Hwang, S. O. (2023). Area–time efficient implementation of nist lightweight hash functions targeting iot applications. *IEEE Internet of Things Journal*, 10(9):8083–8095.

Turan, M. S., McKay, K. A., Çalik, Ç., Chang, D., Bassham, L., et al. (2019). 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.

Turan, M. S., Turan, M. S., McKay, K., Chang, D., Bassham, L. E., Kang, J., Waller, N. D., Kelsey, J. M., and Hong, D. (2023). *Status report on the final round of the NIST lightweight cryptography standardization process*. US Department of Commerce, National Institute of Standards and Technology.

Turan, M. S., Turan, M. S., McKay, K., Chang, D., Calik, C., Bassham, L., Kang, J., and Kelsey, J. (2021). Status report on the second round of the nist lightweight cryptography standardization process.

Windarta, S., Suryadi, S., Ramli, K., Pranggono, B., and Gunawan, T. S. (2022). Lightweight cryptographic hash functions: Design trends, comparative study, and future directions. *IEEE Access*, 10:82272–82294.