A survey of Lightweight Cryptographic Hash Function

Baraa Tareq Hammad, Norziana Jamil, Mohd Ezanee Rusli and Muhammad Reza Z`aba

Abstract- Lightweight cryptography is developed to enhance the security level in pervasive computing applications such as those characterized by smart but resource-constrained devices. The two main primitives of lightweight symmetric cryptographic are lightweight block cipher and lightweight hash algorithm. In this paper, a comprehensive survey of some lightweight cryptographic hash functions will be described from both hardware and software perspectives. Apart from covering the analyses of these algorithms, the differences of these primitives in terms of throughput, number of cycle, comprehensive area, power and energy will be outlined. We will provide a classification of these lightweight hash functions as well.

Index Terms— Lightweight Cryptography, Hash Function, Resource-Constrained.

1 Introduction

Lightweight cryptography is a new branch of cryptography which is designed to cope with the rapid expansion of advanced technologies. The implementation of classical cryptography in these devices is impractical due to the mathematical complexity associated with cryptographic primitives [1]. Typically, classical cryptography requires high processing power and large memory space. Efforts developed to decrease the execution time of traditional cryptographic primitives have been reported [2, 3, 4, 5, 6]. However, the overall implementation cost increases due to the additional hardware involved.

Lightweight cryptography is developed to reduce key size, cycle rate, throughput rate, power consumption and area (which are measured in Gate Equivalence (GE)).

Recently, Hirotaka Yoshida proposed MAME [7], which is a lightweight cryptographic hash function. It takes a 256-bit message block and a 256-bit chaining variable as inputs and produces a 256-bit output. Its weight is 8.1 kg. Bogdanov et al. [8] developed Spongent lightweight hash functions based on the sponge construction instantiated with permutations [9]. This hash function has been implemented in the 4-bit S-box, and it fulfills the PRESENT design criteria (e.g. differential and linear properties). Spongent has 13 variants for various levels of collision/(second) preimage resistance and implementation constraint. In 2010, Aumasson [10] developed Quark. It is the first lightweight hash function designed based on a single security level. In order to minimize the memory requirements, sponge construction was implemented. Quark adopts a permutation P based on the stream ciphers Grain [11] and block cipher KATAN [12]. There are different variants of Quark, e.g. U-Quark (64-bit security), D-Quark (80-bit security) and T-Quark (112-bit security). Poschmann et al. [13] developed the PHOTON lightweight hash function. It uses a sponge-like construction and an AES-like primitive as internal unkeyed permutation. Therefore, it is a compact hash function with 1120 GE for 64-bit collision resistance security. Also, PHOTON employs two types of S-boxes, i.e. 4-bit PRESENT S-box and 8-bit AES S-box. Kavun et al. [14] developed Keccak, which is indeed a variant of SHA-3 hash function. Again, Keccak is designed based on the sponge construction. Its basic component is Keccak-f permutation, which consists of numerous simple rounds associated with logical operations and bit permutations.

Various algorithms have been proposed based on other constructions. By using Merkle–Damgård construction, Badel et al. [15] proposed ARMADILLO2 (advanced version of ARMADILLO), which is a multi-application primitive. It is applied in MAC and digital signatures (PRNG and PRF). As reported by [15], ARMADILLO2 requires 2,923 GE. In order to complete one computation, a total of 176 clock cycles is required (consuming 44 μW). A new compression function Q has been embedded in ARMADILLO2, which is more compact and secure [16].

DM-PRESENT80, DM-PRESE128 and H-PRESENT128 [17] are some of the lightweight designs of hash function which are developed based on Davies–Meyer mode and the block cipher PRESENT [9]. The feed forward feature of compression function and the reversible components that can be used as a block cipher have been removed.

Panasenko et al. [65] designed the lightweight cryptographic primitives and highlighted some guidelines for implementing these primitives. John [66] reviewed those lightweight cryptographic primitives with two block ciphers and stream ciphers. The security features and the hardware performances of these primitives were analyzed. Katagi et al. [67] highlighted the standardization status of lightweight cryptography primitives. Batina et al. [63] compared the requirements of some lightweight block ciphers and the AES algorithm. Juels [41] examined the approaches for privacy protection and integrity assurance in RFID systems. Lata et al. [68] reviewed some lightweight primitives and their potential applications. Apart from discussing the lightweight stream cipher and lightweight block cipher primitives, Arora et al. [69] studied the hybrid model of Hummingbird [70] and other lightweight cryptography primitives. Mohd et al. [71] classified the implementations of lightweight block cipher and showed that energy metrics is the most important metric in low constrained devices. In this paper, a more comprehensive survey of lightweight hash algo-

rithm will be given.

2. HASH FUNCTION

Hash function takes messages of arbitrary input sizes and produces output messages with a fixed size. The associated computation involves Message Authentication Code (MAC), data integrity and digital signatures. A hash function H is collision-free if it maps messages of any length to strings of fixed length. The output is known as fingerprint [18, 19, 20]. One would like to preserve collisions or (second) preimage to be computationally difficult for the attacker. For an n-bit ideal hash function, an attacker performs $2^{n\backslash 2}$ and 2n computations to obtain a collision and a (second) preimage, respectively [13, 19].

The two main parts of a hash function are construction and compression functions. The construction function is designed to iterate the compression function. An ideal hash function must contain the properties of random oracle. Even though the random oracle does not exist, a hash function construction should meet the security criterion. A cryptographic hash function H with an n-bit output is expected to possess the following main security properties [21]:

1. Collision-resistance: It is difficult to find two different messages m_0 and m_1 such that $H(m_0) = H(m_1)$ and this requires at least $2^{n/2}$ work.
2. Preimage-resistance: Given a hash value $H(m)$, it is difficult to find m, and this requires at least 2^n work.
3. Second preimage-resistance: Given m_0, it is difficult to find a different input m1 such that $H(m_0) = H(m_1)$, and this requires at least 2^n work.

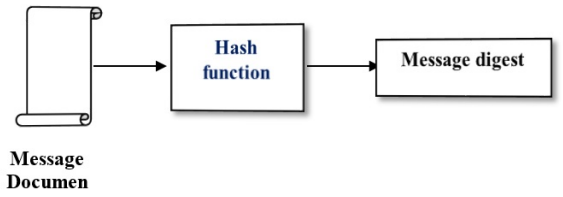

Figure 1. Hash Function

3 GENERIC CONSTRUCTION FOR LIGHTWEIGHT CRYPTOGRAPHIC HASH FUNCTIONS

In hash function, many construction have been used, e.g. Sponge construction [22], Merkle Damgård [18], Davies-Mayer [23], and Haifa [25]. Here, we focus on lightweight cryptographic hash function as it is not fully explored by previous researchers [24].

3.1 Sponge Construction

The challenge in designing lightweight hash functions lies on achieving the balance between security and memory requirements. Producing an output of size > 256 bits to prevent any collision is a common approach; however, it is computationally expensive. Sponge construction [26] is commonly adopted to solve this issue by reducing the (second) preimage security for the same internal state size [8].

This construction is developed based on b-bit permutation P with capacity c bits and bit rate r. m_i is the r-bit message block and Z_i is part of the hash value (with output length n). Its width is determined from the size of its internal state $b = r + c \geq n$. Initially, the bits of the state are set to zero. Then, the input message is padded and divided into blocks of r-bit. The construction consists of two phases, i.e. absorbing and squeezing phases. In the absorbing phase, the r-bit input message blocks are XORed with the first r-bit of the state before inserting the function P. After processing all message blocks, the squeezing phase starts. The first r-bit of the state is returned as an output block, followed by the inclusion of function P as shown in Figure 2. The number of output blocks is determined by the user [10, 14].

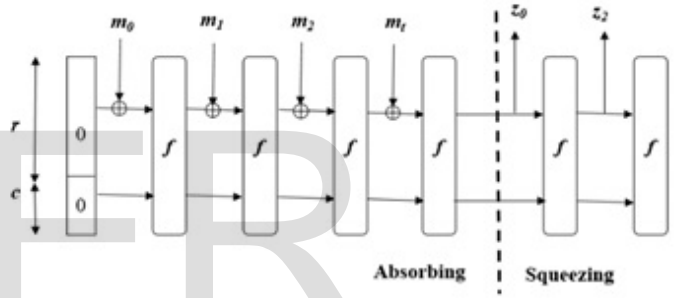

Figure 2. Sponge Construction

Seemingly, the sponge construction [27] is the only alternative to the classical Merkle–Damgård construction due to its lightweight design. This process depends only on a single permutation, and the message blocks are then combined with a simple XOR with the internal state.

In contrast with Davies–Meyer construction, storage of message blocks and "feed forward" intermediate values are not required in sponge construction. Nevertheless, a larger state is required to achieve traditional security levels [13].

3.2 Merkle- Damgård Construction

The Merkle–Damgård construction has been adopted in many hash algorithms such as MD5, SHA1/SHA2, STITCH as well as lightweight hash function algorithm, e.g. ARMADILLO [15]. It is used to construct collision-resistant cryptographic hash functions from collision-resistant one-way compression functions. The Merkle–Damgård hash function [23] involves the application of MD-compliant padding function to create an output of size which is twice as that of a fixed number due to the fact that compression function is unable to handle inputs of arbitrary sizes. The hash function decomposes the output into blocks of fixed size and processes them indivi-

dually with the compression function. At each time, an input block is combined with the output obtained from the previous round. In the current work, we have used a compression function H mapping $\{0, 1\}^n \times \{0, 1\}^k$ to $\{0, 1\}^n$, a fixed and public IV of $\{0,1\}^n$, and a message $(m_1, m_2, \ldots, m_t)$, where each m_i is a block of k bits. Then, a hash function H (see Figure 3) was built.

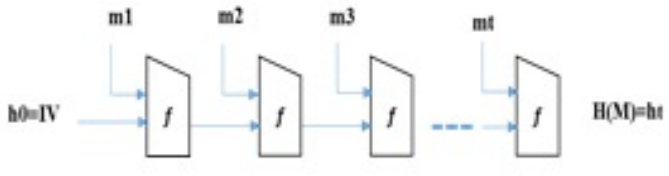

Figure 3. Merkle-Damgård construction

3.3 Davies-Meyer Construction

The modern hash functions were used as compression functions. Their theoretical foundations have been given by Merkle and Damgård [23]. In order to produce a fixed-length output, the compression function H has a fixed-length input that consists of a chaining variable and a message extract [17].

The Davies–Meyer compression function feeds each message block (m_i) with a key to the block cipher. Also, it feeds the previous hash value (H_{i-1}) with an encrypted plaintext. Also, the output of ciphertexts is XORed with the previous hash value (H_{i-1}) in order to yield the next hash value (H_i). Due to the absence of the previous hash value in the first round, the use of a constant pre-specified initial value (H_0) [17] is necessary as shown in Figure 4. It is computed via:

$$Hi = E_{mi}(Hi\text{-}1) \oplus Hi\text{-}1$$

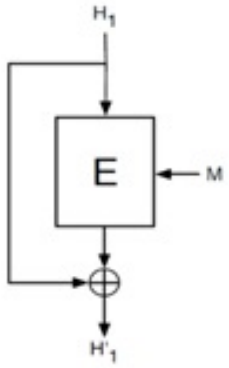

Figure 4. Davies Meyer Compression Function

4 Lightweight Cryptographic Hash Functions

Some lightweight hash functions have been recently reported. Bogdanov et al. [28] described ways of using PRESENT block cipher in hashing modes of operation. The applications of Spongent [8], PHOTON [13] and GLUON [29] in designing a dedicated lightweight hash function based on sponge construction have been highlighted as well. Each algorithm will be described in the next section.

4.1 PHOTON

The PHOTON lightweight hash function was designed by Guo et al. [13]. It uses a sponge-like construction and an AES-like primitive as internal unkeyed permutation. Therefore, it is a compact hash function with 1120 GE for 64-bit collision resistance security [13]. The output size is $64 \leq n \leq 256$, and the input and output bit rates are r and $r`$, respectively. Thus, a PHOTON hash function can be characterized as PHOTON-$n/r/r`$. Its internal state size depends on the hash output size: 100, 144, 196, 256, and 288 bits. The internal permutation P is applied to an internal state of d^2 elements of b bits. Two types of S-boxes are used, i.e. 4-bit PRESENT S-box and 8-bit AES S-box.

4.2 SPONGENT

Bogdanov [8] instantiated the Spongent lightweight hash functions with PRESENT-type permutations. The 4-bit S-box, which is the major block of functional logic in a serial low-area implementation of Spongent, fulfills the PRESENT design criteria in terms of differential and linear properties [9]. Spongent has 13 variants for each collision/(second) preimage resistance level and implementation constraint. Its round function is relatively simple; therefore, its logic size is close to the theoretically smallest size.

In Spongent, the initial value is b-bit 0. In all Spongent variants, the hash size n is equal to either capacity c or $2c$. The message chunks are XORed into the r rightmost bit positions of the state. The same r bit positions form parts of the hash output. Any linear approximation over the S-box (i.e. involves only single bits in the input and output masks) is unbiased. This linear approximation is useful in limiting the linear hull effect discovered in round-reduced PRESENT [8].

4.3 Keccak

Kavun and Yalcin [14] reported the lightweight implementations of Keccak- f [200] and Keccak- f [400] permutations. Keccak-f [200] and Keccak-f [400] are variants of the SHA-3 hash function. In fact, Keccak [30] is developed based on the sponge construction. Its basic component is Keccak- f permutation, which contains numerous simple rounds with logical operations and bit permutations. Keccak-f [b] involves permutation chosen from a set of seven permutations, where b denotes the width of the permutation {25, 50, 100, 200, 400, 800, 1600} and the width of the state in the sponge construction.

4.4 Quark

Quark was developed by Aumasson in 2010 based on sponge construction [10]. It is the first lightweight hash function designed based on a single security level in order to minimize the memory requirement. It employs a permutation *P* based on the stream ciphers Grain [11] and block cipher KATAN [12]. There are three types of Quark: U-Quark (64-bit security), D-Quark (80-bit security) and T-Quark (112-bit security). U-Quark gives at least 64-bit security against all attacks [31]. Meanwhile, U-Quark requires 1379 GE and consumes ~2.44 μW at 100 kHz [31]. T-Quark implemented in [31] requires 2296 GE. The internal permutation *P* contains three nonlinear Boolean functions, i.e. *f, g* (similar to that in Grain), and *h*. Also, it consists of one linear Boolean function *p* and involves *P* processes. All the nonlinear Boolean functions are unique for each Quark function. The *P* processes are dependent on three phases, i.e. initialization, state update and computation of output.

4.5 Neiva

In [32] Proposed a lightweight hash function based on sponge construction and PRESENT [33] block cipher. The state b is of 256-bit. The rate and capacity is 32-bit and 224-bit respectively, and 32 rounds. The process of Neiva is as follow, first the Message *M* is padded and then divided into the 32-bit blocks after that the first message block M_1 is XORed to the state. After applying the PRESENT S-box in parallel, the updated register is divided in 16-bit words and apply Feistel structure on every 64-bit. After an 8-bit left rotation, it is added to a round constant. The updated register after modular addition is the output of first round. It keeps feeding to the next round till 32 rounds. In squeezing phase, take the most significant 32-bit of last register of absorbed phase. Then apply f seven times on the updated register and every time take out the most significant 32-bit. In order to get the 224 bit output, the seven 32 bit will be concatenated.

4.6 ARMADILLO

Badel et al. [15] proposed ARMADILLO, which is a multi-application primitive. It was used as MAC and digital signatures such as PRNG and PRF. Its structure is similar to that of the Merkle–Damgård construction. In general, ARMADILLO requires 2923 GE. A total of 176 clock cycles is required to complete one computation (consuming 44 μW power). ARMADILLO2, which is a new variant of ARMADILLO, is more robust than ARMADILLO. It uses a more compact and secure compression function [15].

4.7 GLUON

Berger et al. developed the GLUON hash function [29] based on the sponge construction model [34]. The *f* function was used to call a filtered feedback with carry shift register (FCSR). The filtered FCSR was developed based on the F-FCSR-v3 hardware stream cipher [35] and the X-FCSR-v2 software stream cipher [36]. It is slightly heavier than Quark and PHOTON. The lightest instance of GLUON-64 provides the 64-bit security level and requires 2071 GE [29]. Meanwhile, GLUON-80 provides the 80-bit security level and GLUON-112 provides the 112-bit security level and requires 4724 GE [29]. From a stream cipher with an internal state size of *n*, one can construct a function from $\{0, 1\}^b$ as follows:

1. The b-bit input is filled into an initial state size of n bits.
2. The stream cipher is initialized as usual, where the first b output bits compose the output of the f function.

Assuming that the stream cipher is "perfect," the function will mimic a random function which is used to identify siding in the function (twice of that of stream cipher).

4.8 DM-PRESENT and H-PRESENT

DM-PRESENT80, DM-PRESE128 and H-DM-PRESENT80, DM-PRESE128 and H-PRESENT128 [17] are some of the lightweight hash functions developed based on the block cipher PRESENT [9]. The feed forward feature of the compression function and the reversible components that can be used as a block cipher are discarded.

The DM-PRESENT [17] hash functions rely on a compression function in order to take input from some words of the chaining variable (represented by H_i) and some words of the formatted message extract (represented by M_i). A single 64-bit chaining variable H_i is then updated from the Davies–Meyer operation by using a message extract M_i:

$$H_i = E(H_i, M) \oplus Hi$$

In this case, E denotes encryption with either PRESENT-80 or PRESENT-128, which can provide 64-bit security level.

Each iteration of compression function involves the compressions of 64 bits of chaining variable and 80 bits of message-related input. Therefore, DM-PRESENT-80 and DM-PRESENT-128 are able to give a compromise between space and throughput. Replacing PRESENT with a different block cipher will definitely augment the space required during the implementation stage.

H-PRESENT-128 compression function treats two 64-bit chaining variables and one 64-bit message extract as inputs (denoted by the triple (H_1, H_2, M)). A pair of updated chaining variables (H_1', H_2') is then produced (output) based on the following computation:

$$H_1' = E(H_1, H_2, \|M) \oplus H_1 \text{ and}$$
$$H_2' = E(H1 \oplus c, H_2, \|M) \oplus H_1$$

Where *E* denotes PRESENT-128 and *c* is a nonzero constant (fixed). Thus, the chaining variable $H_1 \,||\, H_2$ is 128 bits long, and 64 bits of message-related input are hashed per iteration. Hirose proved that an adversary of at least 2^n queries is required in an ideal cipher model in order to obtain a collision with non-negligible advantage. Here, *n* is the block size of the cipher. Similar analysis can be done for preimage resistance to

show that adversary of at least 2^{2n} queries is required in order to identify a preimage.

4.9 Lesamnta-LW

Hirose et al. [37] proposed a lightweight 256-bit hash function called Lesamnta-LW. They claimed that its security level was at least 2^{120} with respect to collision, preimage, and second preimage attacks. Lesamnta-LW was designed based on the Merkle-Damgård as domain extension. It employs AES as the compression function. The weight of Lesamnta-LW hardware is 8.24 KG on 90 nm technology. Lesamnta-LW offers 50 bytes of RAM and deals with short messages on 8-bit CPUs.

4.10 Tav-128

Peris-Lopez et.al. [38] developed the Tav-128 lightweight hash function based on Merkle- Damgård construction. The output was 128 bit and the input message was split into 32-bit blocks. The compression function uses two filter functions (A and B) and two expansion functions (C and D). The internal state consists of five 32-bit words and the final output consists of four 32-bit state registers. The finalization function *g* truncates the state and the outputs to 128 least significant bits. The authors analyzed the statistical properties of its output and estimated that the required hardware footprint was approximately 2.6K GEs.

5 Applications of Lightweight Cryptographic Hash Function

Lightweight cryptographic hash functions are widely applied in cryptography and programming practice. We will describe several scenarios, which can be used as models for evaluating the security levels of practical applications.

5.1 Digital signatures

Hash functions are applied in digital signature schemes [39]. Digital signature is a generic mechanism that transforms any signature scheme for signing messages of a fixed length and a collision-resistant hash function into a signature scheme that can handle messages of arbitrary length. The mechanism is called the hash-and-sign paradigm and it is the basis for all modern practical signature schemes.

Adida et al. [40] described several ways to use digital signature in Email application. One of the techniques is using lightweight cryptographic hash function as the Single Server Key Algorithm. Lightweight cryptographic hash function can be used as a digital signature. According to Ari Juels [41], many American companies such as Texas Instruments and VeriSign Inc. have proposed a "chain-of-custody" approach to apply RFID in their industries. The model involves digital signing of tag data to provide integrity assurance. Hanaoka et al. [42] proposed the Light-Weight Secure Electronic Transaction Protocol (LITESET) and applied the signcryption scheme for verification purpose.

5.2 MAC and HMAC

MAC and HMAC Message-Authentication Code (MAC) are keyed hash functions satisfying certain cryptographic properties. MAC generates a tag from a message and a secret key in order to verify the authenticity and the integrity of the message.

Typically, MAC can be generated based on a collision-resistant keyless hash function. In fact, the most commonly used MAC construction is HMAC. For certain applications, occasionally accepting an inauthentic message may delimit the security level of an application; therefore, shorter tags are recommended [43]. Many researchers have worked on lightweight HMAC such as Shen [44] Chaskey [45, 46], TuLP [47], SipHash [48] and LightMAC [49].

5.3 Authenticated Encryption

Recently, permutation-based constructions [22] are applied in a wide range of platforms such as lightweight devices. Lightweight permutation-based hash functions include Gluon [29], Photon [13], Quark [10, 31], and Spongent [8]. Lightweight cryptographic hash functions are used in hash function application. It is employed to provide Authenticated Encryption (AE) as well. AE is a cryptographic primitive that guarantees privacy and integrity [50]. Usually, it is built from block cipher [51, 52, 53]. Many permutation-based AE schemes have been proposed recently, e.g. deterministic key-wrap scheme [54] and SpongeWrap [55, 56]. Many researchers designed their primitives based on hash functions such as ALE [57], LAMP [58], JHAE [59] and Humenberidg [60].

6 Comparative Analysis of Existing Lightweight Cryptographic Hash Functions

Comparison of lightweight cryptographic primitives is challenging because many characteristics should be considered and these primitives are technology-dependent. In general, a fair comparison can be made if the utilized tools and libraries are similar.

In order to assess the implementation of a lightweight hash function, the following metrics should be considered:

- Area: Measured in GE.
- Cycles: The number of clock cycles used to compute and read out the ciphertexts.
- Time: The ratio of the number of cycles to the operating frequency in seconds.
- Throughput: The rate at which new output is produced with respect to time.
- Power: The estimated power consumption on the gate level by using the Power Compiler.

- Efficiency: Hardware efficiency is measured by dividing the throughput to area ratio.

A basic RFID tag may have a total gate count of between 1000 and 10000 gates [13]. No more than 2,000 GE are available for security in low-cost RFID tags [61, 28, 35, 62]. A common metric to measure the efficiency of the proposed algorithm is the number of GE. Basically, the GE can be calculated by dividing the silicon area that is used for a cipher with a given standard cell library by the area of a two-input NAND gate [63]]. In addition, the power required for 100 KHz RFIDs must be less than 27 μW power [36, 64]. Therefore, we can conclude that the area metric is more important than the power, especially when we want to measure the efficiency of the primitives. Table 2 shows the comparison of the performances of lightweight hash functions of different hash functions based on the direct application of sponge-based construction. In principle, comparing the performances of designs implemented in different platforms is not easy. Nevertheless, the results shown in the table are calculated based on measurements as reported in the references.

First, the DM-PRESENT-80 consumes 6.28μW at a clock frequency of 100 KHz in the round-based implementation with a total area of 2213 GE, whereas the serialized implementation consumes 1.83μW and requires 1600 GE. We observed that the parallel design needs more area than the serialized design. However, the parallel design is faster because it needs fewer clock cycles and consumes less power than the serialized design. A similar finding can also be observed when the Parallel Keccak-*f* and the Serial Keccak- f were compared. In terms of power consumption of the serialized implementation of PHOTON-80/20/16, PHOTON-128/16/16, PHOTON-160/36/36, PHOTON-224/32/32, and PHOTON-256/32/32, each of these variants of PHOTON consumes 1.59μW, 2.29μW, 2.74μW, 4.01μW, and 4.55μW of power, respectively.

By contrast, the parallel implementations of the same hash functions require 2.7μW, 3.45μW, 4.35μW, 6.5μW, and 8.38μW of power, respectively. These values are higher than the serialized implementation. We also observed that parallel implementation of PHOTON generally requires a higher number of GE than serialized implementations. A similar observation was also observed when Spongent and Quark were compared.

The most recently published hash families like Keccak, SPONGENT, PHOTON and Quark is based on a sponge construction. The sponge construction can be seen as an alternative to the classical Merkle-Damgård construction. It rather relies on a single permutation, and message blocks are integrated with a simple XOR with the internal state. There is No feed-forward necessary for the sponge construction as in Davies- Meyer constructions, however they need a larger state to achieve traditional security levels that compromises memory savings. Using sponge functions as operating mode is another step towards compactness. Avoiding any feed-forward such as that in sponge construction saves a lot of memory registers at the cost of an invertible iterative process that induces a lower (second)-preimage security for the same internal state size. The sponge construction keeps the internal memory size as low as possible. This can be seen when we compare the result of all sponge construction functions with Merkle-Damgård construction (ARMADELO) and Davies-Meyer mode (DM-PRESENTS).

Table 1 Comparison of Performances of Lightweight Hash Functions.

Primitives		Hash output size	Data path size	Cycles per block	Through-put at 100 KHz	Power μW	Logic process μm	GE
DM-PRESENT-80 [21].	Davies-Meyer mode	64	4	4547	14.63	6.28	0.18	1600
		64	64	45	242.42	1.83	0.18	2213
DM-PRESENT-128 [21]		64	4	559	22.9	7.49	0.18	1886
		64	128	74	387.88	2.94	0.18	2530
PHOTON-80/20/16 [9]	sponge-like construction	80	4	708	2.82	1.59	0.18	865
		80	20	132	2.82	2.7	0.18	1168
PHOTON-128/16/16 [9]		128	4	996	1.61	2.29	0.18	1122
		128	24	156	15.15	3.45	0.18	1708
PHOTON-160/36/36 [9]		160	4	1332	2.70	2.74	0.18	1396
		160	28	180	10.26	4.35	0.18	2117
PHOTON-224/32/32 [9]		224	4	1716	1.86	4.01	0.18	1735
		224	32	204	15.69	6.5	0.18	2786
PHOTON-256/32/32 [9]		256	4	996	3.21	4.55	0.18	2177

		256	48	156	20.51	8.38	0.18	4362
Parallel Keccak-f[1600] [47]		256	64	24	4533	315.1	0.18	4763
Serial Keccak-f[1600] [47]		256	64	1200	90.66	44.9	0.18	2079
Parallel Keccak-f[400] [47]		128	16	20	720	78.1	0.18	1056
Serial Keccak-f[400] [47]		128	16	1000	14.4	11.5	0.18	509
Parallel Keccak-f[200] [47]		64	8	18	400	27.6	0.18	409
Serial Keccak-f[200] [47]		64	8	900	8	5.6	0.18	252
U-Quark [19]		128	1	544	1.47	2.44	0.18	1379
		128	8	68	11.76	4.07	0.18	2392
D-Quark [19]		160	1	704	2.27	3.10	0.18	1702
		160	8	88	18.18	4.67	0.18	2819
T-Quark [19]		224	1	1024	3.13	4.35	0.18	2296
		224	16	64	50	8.39	0.18	4640
GLUON-64 [46]		128	8	66	12.12	N/A	0.13	2071
GLUON-80 [46]		160	16	50	32	N/A	0.13	2799.3
GLUON-112 [46]		224	32	55	58.18	N/A	0.13	4724
spongent-88/80/8 [20]		88	4	990	0.81	1.57	0.13	738
		88	88	45	17.78	2.31	0.13	1127
spongent-128/128/8 [20]		128	8	2380	0.34	2.20	0.13	1060
		128	136	70	11.43	3.58	0.13	1687
spongent-160/160/16 [20]		160	4	3960	0.40	2.85	0.13	1329
		160	176	90	17.78	4.74	0.13	2190
spongent-224/224/16 [20]		224	4	7200	0.22	3.74	0.13	1728
		224	240	120	13.33	5.97	0.13	2903
spongent-256/256/16 [20]		256	4	9520	0.17	4.21	0.13	1950
		256	272	140	11.43	6.62	0.13	3281
Neiva [55]		224	32	12067	4.99	-	-	
ARMADILLO **[48]**	Merkle–Damgård	48	80	176	272	44	0.18	2,923
Lesamnta-LW [37]		256	20	188.3	125.55	-	90 nm	8,24

7 Conclusion

In this paper, we provide a comprehensive survey of lightweight cryptographic hash algorithms. These classifications are very useful because these primitives have various characteristics. Also, this paper highlights the security levels of lightweight cryptographic hash functions. The structures show that primitive designs with hardware-friendly operations give a smaller GE. Also, parallel design gives better performance; however, its GE is higher than that of the serial design. The classification would assist researchers to optimize the design of lightweight cryptographic primitive in order to achieve the compromise between the security level and the resource constraint.

References

[1] I. Vanda, L. Buttyán et al., "Lightweight authentication protocols for low-cost RFID tags," in Second Workshop on Security in Ubiquitous Computing–Ubicomp, vol. 2003, 2003.

[2] J. H.-F. Constantin, A. P. Burg, and F. K. Gurkaynak, "Investigating the potential of custom instruction set extensions for sha-3 candidates on a 16-bit microcontroller architecture," 2012.

[3] A. J. Elbirt, "Fast and efficient implementation of aes via instruction set extensions," in Advanced Information Networking and Applications Workshops, 2007, AINAW'07. 21st International Conference on, vol. 1. IEEE, 2007, pp. 396–403.

[4] P. Grabher, J. Großschädl, and D. Page, "Light-weight instruction set extensions for bit-sliced cryptography," in International Workshop on Cryptographic Hardware and Embedded Systems. Springer, 2008, pp. 331–345.

[5] A. Hodjat and I. Verbauwhede, "Interfacing a high speed crypto accelerator to an embedded cpu," in Signals, Systems and Computers, 2004. Conference Record of the Thirty-Eighth Asilomar Conference on, vol. 1. IEEE, 2004, pp.

488–492.
[6] S. O'Melia and A. J. Elbirt, "Instruction set extensions for enhancing the performance of symmetric-key cryptography," in Computer Security Applications Conference, 2008. ACSAC 2008. Annual. IEEE, 2008, pp. 465–474.
[7] H. Yoshida, D. Watanabe, K. Okeya, J. Kitahara, H. Wu, Ö. Küçük, and B. Preneel, "Mame: A compression function with reduced hardware requirements," in CHES, vol. 4727. Springer, 2007, pp. 148–165.
[8] A. Bogdanov, M. Knezevic, G. Leander, D. Toz, K. Varici, and I. Verbauwhede, "Spongent: The design space of lightweight cryptographic hashing," IEEE Transactions on Computers, vol. 62, no. 10, pp. 2041–2053, 2013.
[9] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. Robshaw, Y. Seurin, and C. Vikkelsoe, "Present: An ultra-lightweight block cipher," in International Workshop on Cryptographic Hardware and Embedded Systems. Springer, 2007, pp. 450–466.
[10] J.-P. Aumasson, L. Henzen, W. Meier, and M. Naya-Plasencia, "Quark: A lightweight hash," in International Workshop on Cryptographic Hardware and Embedded Systems. Springer, 2010, pp. 1–15.
[11] M. Hell, T. Johansson, and W. Meier, "Grain: a stream cipher for constrained environments," International Journal of Wireless and Mobile Computing, vol. 2, no. 1, pp. 86–93, 2007.
[12] C. De Canniere, O. Dunkelman, and M. Kneževi´c, "Katan and ktantan—a family of small and efficient hardware-oriented block ciphers," in International Workshop on Cryptographic Hardware and Embedded Systems. Springer, 2009, pp. 272–288.
[13] J. Guo, T. Peyrin, and A. Poschmann, "The photon family of lightweight hash functions," in Annual Cryptology Conference. Springer, 2011, pp. 222–239.
[14] E. B. Kavun and T. Yalcin, "A lightweight implementation of keccak hash function for radio-frequency identification applications," in International Workshop on Radio Frequency Identification: Security and Privacy Issues. Springer, 2010, pp. 258–269.
[15] S. Badel, N. Da˘gtekin, J. Nakahara Jr, K. Ouafi, N. Reffé, P. Sepehrdad, P. Sušil, and S. Vaudenay, "Armadillo: a multi-purpose cryptographic primitive dedicated to hardware," in International Workshop on Cryptographic Hardware and Embedded Systems. Springer, 2010, pp. 398–412.
[16] D. Hankerson, A. J. Menezes, and S. Vanstone, "Guide to elliptic curve cryptography," 2006.
[17] A. Poschmann, "Lightweight cryptography," Ph.D. dissertation, 2009.
[18] I. Damgård, "A design principle for hash functions," in Advances in Cryptology, CRYPTO'89 Proceedings. Springer, 1990, pp. 416–427.
[19] A. Kahate, "Cryptography and network security," 2013.
[20] M. R. S. Abyaneh, "Security analysis of lightweight schemes for rfid systems," 2012.
[21] E. Andreeva, B. Mennink, and B. Preneel, "Security properties of domain extenders for cryptographic hash functions." JIPS, vol. 6, no. 4, pp. 453–480, 2010.
[22] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "Sponge functions," in ECRYPT hash workshop, vol. 2007, no. 9, 2007.
[23] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone, "Handbook of applied cryptography," 1996.
[24] S. Bakhtiari, R. Safavi-Naini, J. Pieprzyk et al., "Cryptographic hash functions: A survey," Centre for Computer Security Research, Department of Computer Science, University of Wollongong, Australie, 1995.
[25] O. Dunkelman and E. Biham, "A framework for iterative hash functions: Haifa," in 2nd NIST cryptographich hash workshop, vol. 22, 2006.
[26] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "Cryptographic sponges," online] http://sponge. noekeon. org, 2011.
[27] M. Feldhofer, S. Dominikus, and J.Wolkerstorfer, "Strong authentication for RFID systems using the AES algorithm," in International Workshop on Cryptographic Hardware and Embedded Systems. Springer, 2004, pp. 357–370.
[28] A. Bogdanov, G. Leander, C. Paar, A. Poschmann, M. J. Robshaw, and Y. Seurin, "Hash functions and rfid tags: Mind the gap," in International Workshop on Cryptographic Hardware and Embedded Systems. Springer, 2008, pp. 283– 299.
[29] T. Berger, J. D'Hayer, K. Marquet, M. Minier, and G. Thomas, "The gluon family: a lightweight hash function family based on fcsrs," Progress in Cryptology- AFRICACRYPT 2012, pp. 306–323, 2012.
[30] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "Keccak sponge function family main document," Submission to NIST (Round 2), vol. 3, p. 30, 2009.
[31] J.-P. Aumasson, L. Henzen, W. Meier, and M. Naya-Plasencia, "Quark: A lightweight hash," Journal of cryptology, pp. 1–27, 2013. [32] K. Bussi, D. Dey, M. K. Biswas, and B. Dass, "Neiva: A lightweight hash function." IACR Cryptology ePrint Archive, vol. 2016, p. 42, 2016.
[33] M. R. Z'aba, N. Jamil, M. E. Rusli, M. Z. Jamaludin, and A. A. M. Yasir, "I-present ™: An involutive lightweight block cipher," Journal of Information Security, vol. 2014, 2014.
[34] F. Arnault, T. Berger, C. Lauradoux, M. Minier, and B. Pousse, "A new approach for fcsrs," in International Workshop on Selected Areas in Cryptography. Springer, 2009, pp. 433–448.
[35] A. Juels and S. A. Weis, "Authenticating pervasive devices with human protocols," in Annual International Cryptology Conference. Springer, 2005, pp. 293–308.
[36] X. Guo and P. Schaumont, "The technology dependence of lightweight hash implementation cost," in ECRYPT Workshop on Lightweight Cryptography, 2011.
[37] S. Hirose, K. Ideguchi, H. Kuwakado, T. Owada, B. Preneel, and H. Yoshida, "A lightweight 256-bit hash function for hardware and low-end devices: lesamntalw," in International Conference on Information Security and Cryptology. Springer, 2010, pp. 151–168.
[38] H. Martin, P. P. Lopez, E. San Millan, and J. E. Tapiador, "A lightweight implementation of the tav-128 hash function," IEICE Electronics Express, vol. 14, no. 11, pp. 20 161 255–20 161 255, 2017.
[39] R. Gennaro and P. Rohatgi, "How to sign digital streams," Advances in Cryptology— CRYPTO'97, pp. 180–197, 1997.
[40] B. Adida, S. Hohenberger, and R. L. Rivest, "Lightweight encryption for email." in SRUTI, 2005.
[41] A. Juels, "Rfid security and privacy: A research survey," IEEE journal on selected areas in communications, vol. 24, no. 2, pp. 381–394, 2006.
[42] G. Hanaoka, Y. Zheng, and H. Imai, "Liteset: A light-weight secure electronic transaction protocol," in Information Security and Privacy. Springer, 1998, pp. 215–226.
[43] K. A. McKay, L. Bassham, M. S. Turan, and N. Mouha, "Report on lightweight cryptography," NIST DRAFT NISTIR, vol. 8114, 2016.
[44] M. M. Fouda, Z. M. Fadlullah, N. Kato, R. Lu, and X. S. Sheen, "A lightweight message authentication scheme for smart grid communications," IEEE Transactions on Smart Grid, vol. 2, no. 4, pp. 675–685, 2011.
[45] N. Mouha, B. Mennink, A. Van Herrewege, D. Watanabe, B. Preneel, and I. Verbauwhede, "Chas key: an efficient mac algorithm for 32-bit microcontrollers," in International Workshop on Selected Areas in Cryptography. Springer, 2014, pp. 306–323.
[46] K. Mahmood, S. A. Chaudhry, H. Naqvi, T. Shon, and H. F. Ahmad, "A lightweight message authentication scheme for smart grid communications in power sector," Computers & Electrical Engineering, vol. 52, pp. 114–124, 2016.
[47] Z. Gong, P. Hartel, S. Nikova, S.-H. Tang, and B. Zhu, "Tulp: A family of lightweight message authentication codes for body sensor networks," Journal of computer science and technology, vol. 29, no. 1, p. 53, 2014.
[48] J.-P. Aumasson and D. J. Bernstein, "Siphash: A fast short-input prf." in INDOCRYPT, vol. 7668. Springer, 2012, pp. 489–508.
[49] A. Luykx, B. Preneel, E. Tischhauser, and K. Yasuda, "A mac mode for lightweight block ciphers," in International Conference on Fast Software Encryption. Springer, 2016, pp. 43–59.
[50] E. Andreeva, B. Bilgin, A. Bogdanov, A. Luykx, B. Mennink, N. Mouha, and K. Yasuda, "Ape: authenticated permutation-based encryption for lightweight cryptography," in International Workshop on Fast Software Encryption. Springer, 2014, pp. 168–186.
[51] D. A. McGrew and J. Viega, "The security and performance of the galois/counter mode (gcm) of operation." in Indocrypt, vol. 3348. Springer, 2004, pp. 343–355. [52] D. Whiting, N. Ferguson, and R. Housley, "Counter with cbc-mac (ccm)," 2003.
[53] P. Rogaway, M. Bellare, and J. Black, "Ocb: A block-cipher mode of operation for efficient authenticated encryption," ACM Transactions on Information and System Security (TISSEC), vol. 6, no. 3, pp. 365–403, 2003.
[54] D. Khovratovich, "Key wrapping with a fixed permutation." in CT-RSA, 2014, pp. 481–499.
[55] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "Duplexing the sponge: single-pass authenticated encryption and other applications," in International Workshop on Selected Areas in Cryptography. Springer, 2011, pp. 320–337.
[56] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "Permutation-based encryption, authentication and authenticated encryption," Directions in Authenticated Ciphers, 2012.
[57] A. Bogdanov, F. Mendel, F. Regazzoni, V. Rijmen, and E. Tischhauser, "ALE: AES-based lightweight authenticated encryption," in International Workshop on Fast Software Encryption. Springer, 2013, pp. 447–466.
[58] P. Peris-Lopez, J. C. Hernandez-Castro, J. M. Estévez-Tapiador, and A.

Ribagorda, "Lmap: A real lightweight mutual authentication protocol for low-cost RFID tags," in Proc. of 2nd Workshop on RFID Security, 2006, p. 06.

[59] J. Alizadeh, M. R. Aref, N. Bagheri, and A. Rahimi, "Jhae: A novel permutationbased authenticated encryption mode based on the hash mode jh," Journal of Computing and Security, vol. 2, no. 1, 2016.

[60] D. Engels, X. Fan, G. Gong, H. Hu, and E. M. Smith, "Hummingbird: ultra-lightweight cryptography for resource-constrained devices," in International Conference on Financial Cryptography and Data Security. Springer, 2010, pp. 3–18.

[61] M. Katagi and S. Moriai, "Lightweight cryptography for the internet of things," Sony Corporation, pp. 7–10, 2008.

[62] Z. Shi, S. Ren, F. Wu, and C. Wang, "The vulnerability analysis of some typical hash-based rfid authentication protocols," Journal of Computer and Communications, vol. 4, no. 08, p. 1, 2016.

[63] L. Batina, A. Das, B. Ege, E. B. Kavun, N. Mentens, C. Paar, I. Verbauwhede, and T. Yalçın, "Dietary recommendations for lightweight block ciphers: power, energy and area analysis of recently developed architectures," in International Workshop on Radio Frequency Identification: Security and Privacy Issues. Springer, 2013, pp. 103–112.

[64] M.-J. O. Saarinen and D. W. Engels, "A do-it-all-cipher for RFID: Design requirements," IACR Cryptology EPrint Archive, vol. 2012, p. 317, 2012.

[65] Panasenko, Sergey, and Sergey Smagin. (2011). Lightweight Cryptography: Underlying Principles and Approaches. International Journal of Computer Theory and Engineering, 3(4).

[66] John, J. (2012). Cryptography for Resource Constrained Devices: A Survey. International Journal on Computer Science & Engineering, 4(11).

[67] Katagi, Masanobu, and Shiho Moriai. (2008). Lightweight cryptography for the Internet of Things. Sony Corporation, 7-10.

[68] Lata, Manju, and Adarsh Kumar. (2014). Survey on Lightweight Primitives and Protocols for RFID in Wireless Sensor Networks. International Journal of Communication Networks and Information Security (IJCNIS), 6(1).

[69] Arora, Nikita, and Yogita Gigras (2013). LIGHT WEIGHT CRYPTOGRAPHIC ALGORITHMS: A SURVEY, IJRDTM – Kailash | ISBN No. 978-1-63041-994-3| Vol.20 | Issue 08.

[70] Fan, X., Hu, H., Gong, G., Smith, E. M., & Engels, D. (2009, November). Lightweight implementation of Hummingbird cryptographic algorithm on 4-bit microcontrollers. In Internet Technology and Secured Transactions, 2009. ICITST 2009. International Conference for (pp. 1-7). IEEE.

[71] Mohd, B.J., Hayajneh, T. and Vasilakos, A.V., 2015. A survey on lightweight block ciphers for low-resource devices: Comparative study and open issues. Journal of Network and Computer Applications, 58, pp.73-93.