Journal of Theoretical and Applied Information Technology
15th October 2017. Vol.95. No 19

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

IMPLEMENTATION OF LIGHTWEIGHT CRYPTOGRAPHIC PRIMITIVES

[1]BARAA TAREQ HAMMAD, [1]NORZIANA JAMIL, [1]MOHD EZANEE RUSLI, [2]MUHAMMAD REZA Z'ABA and ISMAIL T. AHMED

[1] Universiti Tenaga Nasional, Jalan IKRAM-UNITEN, Kajang, Selangor, Malaysia
[2] MIMOS Berhad, Technology Park Malaysia, Kuala Lumpur, Malaysia

ABSTRACT

Lightweight cryptography is not a new branch in cryptography. It is a subject specifically addressing the implementation of security mechanism in pervasive computing that are characterized by smart but resource constrained devices. There are at least two main lightweight symmetric cryptographic primitives namely lightweight block cipher and lightweight hash algorithm. Most of the previous surveys were focusing on implementation of specific cryptographic primitives. In this paper we present a comprehensive survey of all lightweight symmetric cryptographic primitives, from hardware and software perspectives. The survey covers analysis of these algorithms and a comparison between these primitives in terms of throughput, number of cycle, comprehensive area, power, and energy. We also provide a classification of the structure of lightweight block cipher and lightweight hash function. These classifications are very useful because the primitives have different and sometimes contrary characteristics. Finally this comprehensive survey highlights some of the issues related to security aspect of small key length in lightweight cryptographic primitives.

Keywords: *Lightweight Cryptography, Symmetric Cryptography, Block Cipher, Hash Function.*

1- INTRODUCTION

Nowadays, Personal digital assistants (PADs), cellular phones, radio-frequency identification (RFID) tags, low-end smart cards, wireless sensors, custom controllers, smart cards, healthcare devices, and a plethora of small devices have ushered in a new explosion of technology. These devices meet numerous application and consumer demands. However, these devices typically have several limitations in terms of energy/power, computation, memory, storage, and/or resources. These limitations result in challenges to the implementation of cryptographic primitives in these devices. Lightweight cryptography was thus introduced.

Lightweight cryptography is a modern branch of cryptography that resulted from the significant expansion in ubiquitous emerging technologies. In terms of performance, the implementation of traditional cryptography in these devices is impractical because of the complex and heavy mathematical operations of the traditional cryptographic primitives. These operations require high processing power and large memory space. In other words, the implementation of traditional cryptography in constrained environments is expensive. Many researchers have attempted to decrease the execution time of traditional cryptographic primitives [1, 2, 3, 4, 5]. However, the overall implementation costs of these attempts has increased because of the hardware requirements of recommended integrated components.

Lightweight cryptography aims at minimizing the overall implementation costs of cryptographic primitives relative to several aspects, such as key size, cycle rate, throughput rate, power consumption, and areas, which are measured in Gate Equivalence (GE) [66].

Lightweight cryptographic primitives are generally divided into two categories, viz: lightweight symmetric cipher and lightweight asymmetric cipher. The first design of the lightweight symmetric cipher was DESL algorithm [6]. This algorithm was based on the general structure of the Data Encryption Standard (DES) [7], in which different S-boxes were used. The key size is 56 bits, with 1848 GE. Two recent lightweight cryptographic primitives are PRESENT block cipher [8] and PHOTON hash function [9].

Panasenko et al. [10] proposed approach to the design of lightweight cryptographic primitives. They also highlighted some constraints and recommendations for implementing lightweight cryptographic primitives. John [11] conducted a

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

survey of lightweight cryptographic primitives with only two block ciphers and stream ciphers. He analyzed the security features and performances of the hardware implementations of some primitives. Katagi et al. [12] provided an overview of the technology and standardization status of lightweight cryptography primitives. Batina et al. [13] analyzed the requirements of some lightweight block ciphers and compared these requirements with those of the AES algorithm. Juels reported a survey examining the approaches for privacy protection and integrity assurance in RFID systems and then discussed the social and technical contexts of his work [14]. Lata et al. [15] provided an overview of some lightweight primitives and their attributes with a comparison of the possibilities of the applications of such primitives. Arora et al. [16] discussed the lightweight stream cipher and lightweight block cipher primitives and compared the hybrid model of Hummingbird [17] with other lightweight cryptography primitives. Mohd et.al. Provided a taxonomy of lightweight block cipher implementation and showed that the most important metric in low constrained devices is energy metrics [18]. This paper gives a more comprehensive survey of lightweight symmetric cryptographic primitives which include lightweight block ciphers and lightweight hash algorithm. These classifications are very useful because the primitives have different and sometimes contrary characteristics. Also, this survey highlights some of the issues related to security aspect of small key length in lightweight cryptographic primitives.

The remainder of this paper is organized as follows. Section 2 presents the structures of lightweight block ciphers and lightweight hash functions. In Section 3, we provide a classification for both lightweight block cipher and lightweight hash function, and discuss the performance according to their structures. Section 4 discusses the security aspect of small key in lightweight primitives. The conclusion is given in Section 5.

2- LIGHTWEIGHT CRYPTOGRAPHIC PRIMITIVE

We present a holistic view of lightweight cryptographic primitives as shown in Figure 1.

As crucial applications go pervasive, the need for security in RFID and sensor networks is dramatically increasing, which requires secure yet efficiently implementable cryptographic primitives including symmetric and asymmetric cryptography as we shown in Figure 1.

At this time, no promising asymmetric cryptographic primitive has met the desired security and lightweight properties as compared with conventional primitives. Asymmetric cryptography provides more security functionality than the symmetric cipher, but requires more computational power and is slower than symmetric cryptography [59].

Elliptic curve cryptography (ECC) [22], Rivest–Shamir–Adleman (RSA) [23], discrete logarithms [24], LPKI [68] and LEPA [67] are examples of asymmetric cryptographic family. ECC is considered the most effective method for resource-constrained devices because of its small operand lengths and relatively low computational requirements [25]. This interest is normally dictated by the need for good hardware and software requirements. At this time, no promising asymmetric cryptographic primitive has met the desired security and lightweight properties as compared with conventional primitives, such as RSA and ECC. Using Public-key in lightweight cryptography, Public-key constructions need a lot of mathematical operation and computational, such as factoring which require huge resources to complete. Approaches based on public key cryptography are too expensive for the most resource constrained devices.

RSA is the most popular algorithm for asymmetric cryptography and supports key sizes from 1024 to 4096 bits. But require a large hardware footprint and the resource demanding implementations that led researchers to looking for other algorithms for applications in constrained devices. ECC is more attractive for low constrained devices. Its offer the same level of security with shorter keys compared to RSA and lower computational requirements [56].

Asymmetric cryptography provides more security functionality than the symmetric cipher, but requires more computational power and is slower than symmetric cryptography. Batina et al. [26] showed that ECC, which was implemented on a constrained device, requires between 8500 and 14000 gates while in case of symmetric primitives like PRESENT as an example it require only 1570 GE. Obviously, the implementation of asymmetric cryptographic primitives is more costly than symmetric ones. Therefore, it is not the focus of this paper.

We classify the Lightweight Cryptographic Block Cipher according into the structure that build based on it into Feistel structure and SP-network structure. Feistel structure such as DESL [6], KATAN/KTANTAN [35], MIBS [60], CLEFIA

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

[34], HIGHT [29], CURUPIRA [31], LBlock [38], SIMON and SPECK [42], TWINE [40], QTL [39]. SP-network such as AES [2], PRESENT [8], Humming bird-2 [49, 50], LED [41], PRINCE [51], PRINTcipher [52], KLEIN [53].

From this classification, we note that the Feistel structure-based primitives performs faster than SP-network-based primitives. However, in terms of hardware implementation while the SP-network is more hardware friendly where it require less GE.

Also, there was a few attempts to build a Lightweight stream ciphers such as Grain [51], Trivium [36], MICKEY [61], BEAN [62], Wg-8 [63], Fruit [64], and Lizard [65].

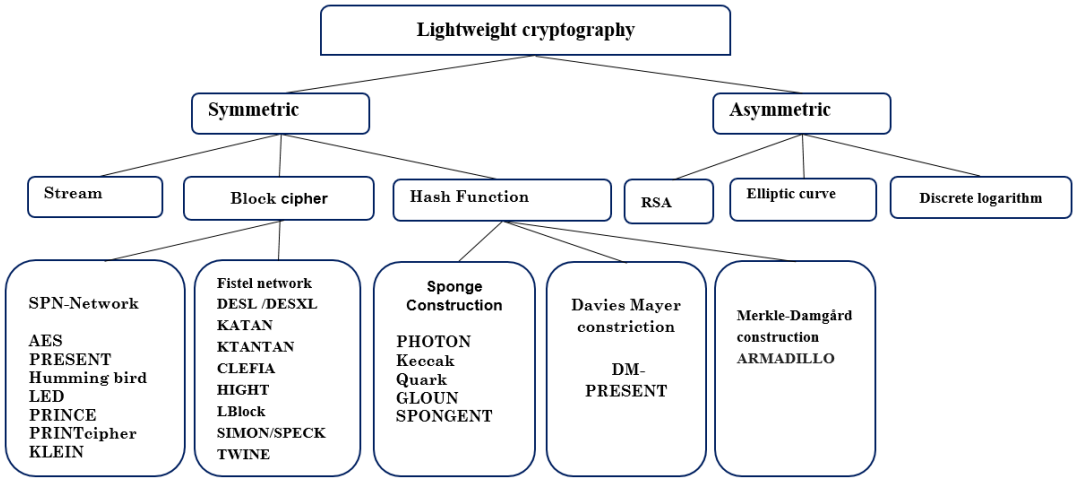

Figure 1. Lightweight Cryptographic Primitives.

2-1 Lightweight Block Ciphers

In this section, we review existing lightweight block ciphers:-

A. AES: - Feldhofer et al. [27] introduced an authentication protocol for RFID tags using AES [28]. The proposed low-power implementation of the AES operates on a fixed input size. The flexibility of AES enables its implementation on different platforms. Efficient implementations are possible on 8-, 32-, 64-, and 128-bit platforms. The hardware implementation of AES to encrypt a 128-bit block of data requires 3595 GE within 996 clock cycles and has a power consumption of 8.15 µA on a 0.35 µm (CMOS) process.

B. HIGHT: - Hong et al. [29] design HIGHT (high security and light weight) was standardized by the Telecommunications Technology Association of Korea. The structure of HIGHT is a generalized Feistel structure GFS, and the round function is light when compared with the SP-like structure. Every operation in HIGHT is 8-bit processor-oriented, making it suitable for low-resource hardware implementation. Therefore, HIGHT is hardware-oriented rather than software-oriented. Hardware implementation of HIGHT requires 3048 GE. The encryption and decryption processes in HIGHT are the same.

C. DESL & DESXL (DES Lightweight) proposed by Poschmann et al. [6]. The design is based on DES. The main idea of DESL and DESXL is to minimize gate complexity by using serial hardware. Furthermore, a single S-box repeated eight times is used in the round function instead of the eight S-boxes employed in the original DES. The single S-box is more resistant to differential and linear cryptanalysis than the original DES S-boxes. The original initial permutation and its inverse are removed because they do not provide additional cryptographic strength, as well as to decrease wiring costs. DESL implementation requires 1848 GE. The small 56-bit key size provides limited protection. Thus, DESL is suitable for applications that require short-term security.

D. Curupira: - Barreto [30] suggested the use of "Curupira-1" for the original key schedule. Marcos [31] proposed the use of "Curupira-2" for the new specification. We simply write "Curupira" when discussing both. The round function structure is used for Curupira [31], with nonlinear layer γ, permutation layer π, linear diffusion layer θ, and key addition layer σ (Kr).

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

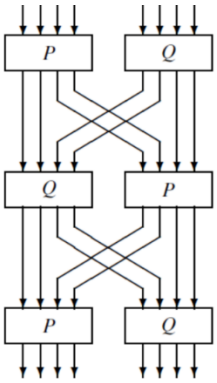

Figure2. Structure of the Curupira-1 S-Box [30].

With a highly nonlinear S-box, the Curupira S-box is the same as that used in Anubis [32] and Khazad [33], in which the implemented 8 × 8-bit Curupira S-box is composed of two 4 × 4-bit S-boxes, namely, *P* and *Q*, to ensure high diffusion speed. The Curupira S-box has the advantage of being cyclical, which indicates that the original key is recovered after a certain number of rounds, thus eliminating the need for storing any intermediary subkey during encryption and decryption. However, the use of the Curupira S-box results in many memory/performance trade-offs.

E. PRESENT: - [8] is an SP network block cipher with 31 rounds. The block size is 64 bits, and the cipher supports two keys sizes, which are 80 and 128 bits. Each round consists of an XOR operation to introduce a secret round subkey, a bitwise permutation, and a nonlinear substitution, which consists of 16 identical S-boxes with 4-bit input and 4-bit output (4 × 4). Hardware implementation of PRESENT-80 requires an area of 1570 GE.

F. CLEFIA: - [34] is a block cipher uses the generalized Feistel structure (GFS). CLEFIA has two different 32-bit F functions per round, with each function containing two distinct 8 × 8 S-boxes and a maximum distance separable (MDS) matrix. This construction is used to maximize cipher resistance against differential and linear cryptanalysis. CLEFIA requires 5979 GE.

G. KATAN and KTANTAN: - De Cannière et al. [35] proposed a new family of block ciphers composed of two variants, namely, KATAN and KTANTAN. The design is based on the Trivium stream cipher [36], which is similar to a nonlinear feedback shift register (NLFSR), with a Feistel structure. KTANTAN is more compact than KATAN and is used in devices in which the key is fixed and can never be changed. The only difference between KATAN and KTANTAN is the key schedule. KATAN and KTANTAN are hardware-efficient block ciphers that require less than 1000 GE. The use of shift registers makes KATAN and KATANTAN suitable for low-resource devices [35]. The KTANTAN32 can be implemented in 462 GE at 100 KHz, whereas KTANTAN48 requires 588 GE and KATAN64, the largest cipher, requires 1054 GE at 100 KHz.

H. Hummingbird: - Engels et al. design Hummingbird [17], and Hummingbird-2 [58], Hummingbird is not classified under the block cipher or stream cipher category, but has the properties of both. The 16-bit block cipher is a typical SP-network consisting of four rounds and a final round that only includes the key mixing and the S-box substitution steps. The round comprises three stages, namely, a key mixing step, a substitution layer, and a permutation layer. The block size is 16 bits, which is suitable for low constrained devices because it deals only with small messages. The implementation of Hummingbird requires 3220 GE. Authenticated Encryption with Associated Data is a method in Hummingbird that authenticates any associated data that travels with cipher text. Processing of associated data occurs only after an entire encrypted payload has been processed.

L. PRINTcipher: - Knudsen et al. [37] proposed the PRINTcipher block cipher for integrated circuit printing or IC-printing as one of the low constrained devices. The structure of PRINTcipher is an SP network. The cipher state is combined with a round key using bitwise XOR. Then, the cipher state is shuffled using a fixed linear diffusion layer. Thereafter, the cipher state is combined with a round constant by using bitwise XOR. The 3-bit entry to each S-box is permuted in a key-dependent permutation layer. Finally, the cipher state is mixed using a layer of b3 nonlinear S-box substitutions.

J. LBlock: - Wu and Zhang [38] proposed the LBlock lightweight block cipher with Feistel structure. The LBlock consists of eight 4 × 4 S-boxes in parallel and requires 1320 GE. The LBlock consists of three parts, namely, encryption algorithm, decryption algorithm, and key scheduling. Each round consists of round function, confusion function, and diffusion function (permutation of eight 4-bit words). The number of S-boxes is decreased, and the size of each S-box is minimized. More rounds are needed to achieve adequate security margins. In each round of LBlock, only half of the data is selected to undergo round function, whereas the other half undergoes a simple rotation operation.

K- LED: - Guo et al. [39] presented an LED block cipher with SP-network structure. The cipher state is conceptually arranged in a 4-bit matrix, with each

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

nibble representing an element from GF (24) with a polynomial expressed as X4 + X + 1. The S-box in the LED cipher is the same as the PRESENT S-box, and its implementation requires 1265 GE.

L. TWINE: - Suzaki et al. [40] proposed TWINE. The structure of TWINE depends on Type-2 generalized Feistel network structure (GFS), with 16 4-bit sub blocks. TWINE uses only one 4-bit S-box and 4-bit XOR. A round function of TWINE consists of a nonlinear layer using 4-bit S-boxes and a diffusion layer, which permutes the 16 blocks. The diffusion layer is not a circular shift and is designed to provide better diffusion than the circular shift. The decryption of TWINE is the same as the encryption in that it uses the same S-box and key schedule with the inverse block shuffle. TWINE requires 2285 GE.

M. PRINCE: - Borghoff et al. [41] proposed the PRINCE. A ciphertext is computed within a single clock cycle and requires 8679 GE. The cipher uses a 4-bit S-box. The same S-box is used 16 times. PRINCE is the first lightweight block cipher that takes latency as the main priority. The cipher is optimized with respect to latency when implemented in the hardware. The internal block cipher is based on the SP-network structure. The cipher has an interesting feature in that one can perform decryption by reusing the encryption process with a slightly different key. This feature provides an advantage in implementations requiring encryption and decryption, but at the same time induces some structure.

N. SIMON and SPECK: - are two families of block ciphers publicly released by the National Security Agency in 2013 and proposed by Beaulieu et al. [42]. SPECK is tuned for optimal performance in software implementations, whereas SIMON is tuned for optimal performance in hardware implementations. The structure for SIMON and SPECK is the Feistel network. SIMON requires 1234 GE, whereas SPECK requires 1280 GE.

O.KLEIN

Gong [43] proposed the KLEIN cipher. The structure of KLEIN is a typical SP-network, the same as AES, and has a 4 bit S-box. KLEIN requires 2,213 GE.

P. QTL: - [44] Propose a Feistel network structures block cipher, QTL. Supports 64 bits block with 64 or 128 bits keys. In traditional Feistel structure process only half the block message, but in QTL it changed the whole message. They don't use the key schedule to reduce the energy consumption.

Q. LiCi:- Patil, et.al. [69] proposed LiCi: a lightweight block cipher. Its Feistel based network the input is 64 bits and the key size is 128 to generate 64 bits cipher text. It requires 1153 GE and consumes 30mW.

R. Oppel-1: presented by Ali, Arshad. [70]. non-Fiestel, substitution-permutation network. input lengths 128 bits and 128 bit key length. The key divides into subkeys by using specially designed subkey generation mechanism.

2-2 Lightweight Cryptographic Hash Functions

Hash function takes an arbitrary input size of messages and produces output messages with a fixed size. Although no secret is involved in the computation, one would like to preserve collisions (two distinct messages hashing to the same value) or (second) preimage (a message input that hashes to a given challenge output value) to be computationally difficult for the attacker. For an n-bit ideal hash function, an attacker performs $2^{n/2}$ and 2^n computations to obtain a collision and a (second) preimage, respectively [9].

Some significant work on lightweight hash functions was recently reported. Bogdanov et al. [45] describes ways of using PRESENT block cipher in hashing modes of operation, whereas Spongent [20], PHOTON [9], and GLUON [46] took the approach of designing a dedicated lightweight hash function based on sponge construction. In the following section, the three different types of construction will be discussed.

2-2-1 permutation-based Sponge Construction

The challenge with the design of lightweight hash functions is dealing with the balance between the security requirement and memory requirements. Most designers of lightweight hash functions focus on the security requirement by producing an output size of at least 256 bits to prevent any collision. This requirement caused the computation of hash function to consume more memory registers. Therefore, most of the proposed hash functions are software-oriented. Sponge construction was recommended to solve this issue. This process can save on memory registers, which reduces (second) preimage security for the same internal state size [20].

Sponge construction is based on b-bit permutation P, with capacity c bits and bit rate r. mi is the r-bit message block and Zi is part of the hash value and has an output length n. The width of a sponge construction corresponds to the size of its internal state $b = r + c \geq n$. In the first step, the bits of the

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

state are all initialized to zero. Then, the input message is padded and divided into blocks of *r*-bit. The construction consists of two phases: the 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 interleaving with the function *P*. After processing all of the message blocks, the squeezing phase begins. The first *r*-bit of the state is returned as an output block, which then interleaves with the function *P* as shown in Figure3. The number of output blocks is chosen by the user [19], [47].

The sponge construction [27] seems to be the only alternative to the classical Merkle–Damgård construction in terms of lightweight design. This process relies on a single permutation, and message blocks are integrated with a simple XOR with the internal state. Sponge functions do not require storage of message blocks nor "feed forward" intermediate values, as in Davies–Meyer constructions. However, sponge functions require a larger state to achieve traditional security levels, which compensates those memory savings [9].

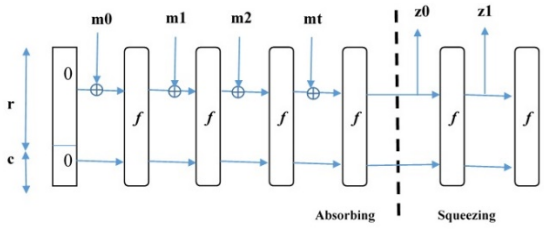

Figure 3. Sponge Construction [19]

2-2-2 Merkle- Damgård construction

The Merkle–Damgård construction is used in many hash algorithms, such as MD5, SHA1 and SHA2, as well as lightweight Hash function algorithm ARMADILLO [48]. The Merkle–Damgård construction is a method used to build collision-resistant cryptographic hash functions from collision-resistant one-way compression functions. The Merkle–Damgård [49] hash function first applies an MD-compliant padding function to create an output with size that is double that of a fixed number because compression functions cannot handle inputs of arbitrary size. The hash function then breaks the output into blocks of fixed size and processes them one at a time with the compression function. At each time, a block of the input is combined with the output of the previous round. We consider 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 (*m1*, *m2*, …, *mt*), where each *mi* is a block of *k* bits. Then, we can build a hash function *H*, as shown in Figure 4.

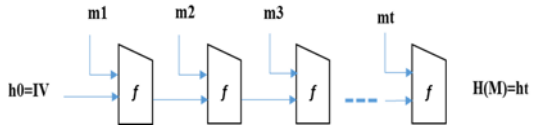

Figure 4. Merkle-Damgård construction [49].

2-2-3 Davies Meyer construction

Hash functions in use today are built around the use of a compression function and appeal to the theoretical foundations laid down by Merkle and Damgård [49]. The compression function *H* has a fixed-length input that consists of a chaining variable and a message extract and yields a fixed-length output [21].

The Davies–Meyer compression function feeds each block of the message (*mi*) as the key to a block cipher. This function also feeds the previous hash value ($Hi-1$) as the plaintext to be encrypted. The output ciphertext is then also XORed with the previous hash value ($Hi-1$) to produce the next hash value (*Hi).* In the first round, the lack of a previous hash value requires the use of a constant pre specified initial value (H_0) [21], as shown in Figure 5 and is based on the following computation:

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

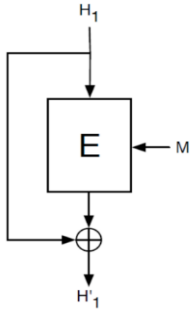

Figure 5. Davies Meyer Compression Function [8]

2-2-4 lightweight cryptographic hash function

The lightweight cryptographic hash functions are given below:

A. DM-PRESENT and H-PRESENT

DM-PRESENT80, DM-PRESE128 and H-DM-PRESENT80, DM-PRESE128, and H-PRESENT128 [21] are lightweight designs of the hash function based on the block cipher PRESENT [8]. The reversible components that can be used as a block cipher are discarded, and the feed forward from the compression function was removed.

The DM-PRESENT [21] hash functions use a compression function that takes input from some words of the chaining variable, represented by *Hi*, and some words of the (formatted) message extract, represented by *Mi*. A single 64-bit chaining variable

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Hi is updated in a Davies–Meyer mode of operation by using a message extract *Mi* based on the following computation:

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

In this case, *E* denotes encryption with either PRESENT-80 or PRESENT-128, which can provide 64-bit security level. At each iteration of the compression function, 64 bits of chaining variable and 80 bits of message-related input are compressed. Therefore, DM-PRESENT-80 and DM-PRESENT-128 provide a simple trade-off between space and throughput. Replacing PRESENT with a different block cipher will almost certainly cause an increase in the space required for an implementation.

The scheme of the H-PRESENT-128 compression function takes as input two 64-bit chaining variables and a 64-bit message extract, denoted by the triple (*H1*,*H2*, *M*), and outputs the pair of updated chaining variables (H_1', H_2') based on the following computation:

$H_1' = E\,(H1, H2, \|M) \oplus H1$ and

$H_2' = E\,(H1 \oplus c, H2, \|M) \oplus H1$

Where *E* denotes PRESENT-128, and *c* is a nonzero constant that should be fixed. Thus, the chaining variable $H1\|H2$ is 128 bits long, and 64 bits of message-related input are hashed per iteration.

Hirose showed that in the ideal cipher model, an adversary has to make at least 2^n queries to the cipher to obtain a collision with non-negligible advantage, where n is the block size of the cipher. The same kind of analysis can be made for preimage resistance to show that any adversary has to make at least 2^{2n} queries to the cipher to identify a preimage.

B. Keccak

Kavun and Yalcin [47] presented a lightweight implementation of the Keccak-*f* [200] and Keccak-f [400] permutations. Keccak-*f* [200] and Keccak-*f* [400] are variants of a SHA-3 hash function, Keccak [50] Keccak is a family of hash functions based on the sponge construction. The basic component is the Keccak-f permutation, which consists of a number of simple rounds with logical operations and bit permutations. The fundamental function of Keccak is a permutation chosen from a set of seven permutations denoted by Keccak-*f*[*b*], 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.

C. Quark

Quark was designed by Aumasson in 2010 [19]. Quark is the first lightweight hash function based on a single security level using sponge construction to minimize memory requirements. Quark uses a permutation P based on the stream ciphers Grain [51] and block cipher KATAN [35]. The hash function family Quark is composed of three different flavors: U-Quark (64-bit security), D-Quark (80-bit security), and T-Quark (112-bit security).

The lightest instance of U-Quark provides at least 64-bit security against collision attacks and all other types of attacks [19]. U-Quark fits in 1379 GE and consumes an average of 2.44 µW at 100 kHz in 0.18 µm [19]. T-Quark in [19] was implemented with 2296 GE. The internal permutation *P* contains three nonlinear Boolean functions *f*, *g* (similar to that in Grain), and *h*, one linear Boolean function *p*, and the *P* processes. All the nonlinear Boolean functions are distinctive in each flavor of Quark. The *P* processes rely on three phases: initialization, state update, and computation of the output, as shown in Figure 6.

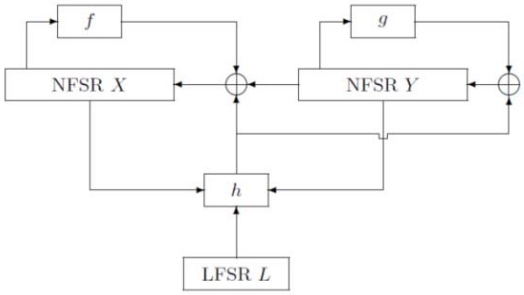

Figure 6. Diagram of the Permutation of Quark [19].

D. ARMADILLO

Badel et al. [48] proposed the ARMADILLO, a multi-application primitive used as a MAC and digital signatures used as a PRNG and PRF. The structure of ARMADILLO depends on the Merkle–Damgård construction. ARMADILLO requires 2,923 GE. One computation could be performed within 176 clock cycles, consuming 44 µW power. A better trade-off would use 4,030 GE, 77 µW power, and 44 cycles.

ARMADILLO2 was developed based on the design of ARMADILLO. However, ARMADILLO2 is more robust than ARMADILLO. ARMADILLO2 uses the new compression function, which is more compact in hardware than the one used in ARMADILLO and more secure [48].

E. PHOTON

The PHOTON lightweight hash function family was designed by Guo, Peyrin, and Poschmann [9]. PHOTON uses a sponge-like construction and an AES-like primitive as internal unkeyed permutation, which enables it to be a compact hash function with 1120 GE for 64-bit collision resistance security [9]. The PHOTON family of

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

hash functions varies the output size n to be between $64 \leq n \leq 256$, and the input and output bit rates are r and r', respectively. Thus, each of the PHOTON family hash functions is denoted as PHOTON-$n/r/r'$. The internal state size depends on the hash output size and takes five different values: 100, 144, 196, 256, and 288 bits. The internal permutation P applied to an internal state of d^2 elements of b bits. The PHOTON hash functions also use two types of S-boxes: the 4-bit PRESENT S-box SBOXPRE and the 8-bit AES S-box SBOXAES.

F. GLUON

Berger et al. proposed the GLUON family hash function [46] based on a sponge construction model [52], where the f function calls a filtered feedback with carry shift register (FCSR). The filtered FCSR is directly based on the F-FCSR-v3 hardware stream cipher [53] and the X-FCSR-v2 software stream cipher [54].The hardware size of such a primitive is slightly heavier than the basic building blocks used in Quark and PHOTON. From the generic definition of lightweight hash function, three different instances were derived on the basis of the desired security level that must be achieved. The lightest instance, GLUON-64, provides a 64-bit security level and fits in 2071 GE [46], GLUON-80 provides an 80-bit security level, GLUON-112 provides a 112-bit security level and 4724 GE [46]. The design choice of f comes from this simple idea: 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.

Under the assumption that the stream cipher is "perfect," the function will look like a random function and will be used to identify siding in the function and the equivalent of twice the stream cipher.

G. SPONGENT

Bogdanov [20] designed Spongent lightweight hash functions based on the sponge construction instantiated with PRESENT-type permutations. The 4-bit S-box is the major block of functional logic in a serial low-area implementation of Spongent. The 4-bit S-box fulfills the PRESENT design criteria in terms of differential and linear properties [8]. Spongent has 13 variants for different levels of collision and (second) preimage resistance, as well as for various implementation constraints. Spongent keeps the round function simple, which reduces the logic size close to the smallest theoretically possible size.

In Spongent, the b-bit 0 is taken as the initial value before the absorbing phase. 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 involving only single bits in the input and output masks is unbiased. This linear approximation aims to restrict the linear hull effect discovered in round-reduced PRESENT [20].

H. Neeva

[55] Proposed a lightweight hash function based on sponge construction and PRESENT 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 Neeva is as follow, first the Message M is padded and then divided into the 32-bit blocks after that the first message block $M1$ 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.

I. Tav-128

Peris-Lopez, et.al. [71] propose the Tav-128 lightweight hash function based on Merkle-Damgård construction. The output is 128 bit and the input message is split into 32-bit blocks. The internal state is composed of five 32-bit words and the final output consists of the four 32-bit state registers. The finalization function g truncates the state and outputs its 128 least significant bits. The authors analyzed the statistical properties of its output and provided an estimation of the hardware footprint required, stating that around 2.6K GEs would be needed.

3- RESULT AND DISCUSSION: CLASSIFICATION AND PERFORMANCE ANALYSIS OF LIGHTWEIGHT SYMMETRIC PRIMITIVES

A fair comparison between different designs of lightweight cryptographic primitives is difficult because many characteristics should be considered.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

In addition, these primitives depend on the technology being used in the design. Generally, a fair comparison is only achieved if the same tools and the same library are used.

To assess the efficiency of the implementation of lightweight cryptographic primitives, the following metrics should be considered:

- ***Area***: Measured in gate equivalent (GE) stands for a unit of measure which allows to specify manufacturing technology independence complexity of digital electronic circuits. It's the silicon area of a two NAND gate usually constitutes the technology-dependent unit area.
- ***Cycles***: The number of clock cycles used to compute and read out the ciphertext. It is one aspect of a processor's performance, and measured in Hz, megahertz MHz or gigahertz GHz.
- ***Time:*** The ratio of the number of cycles to the operating frequency in seconds, and also it's the time during which a program is executing, in contrast to other program life cycle phases such as compile time, link time and load time.
- ***Throughput:*** In general terms, throughput is the maximum rate of production or the maximum rate at which something can be processed. We define it as the rate at which new output is produced with respect to time, measured in bits per second (bps), megabits per second (Mbps) or gigabits per second (Gbps)
- ***Power:*** The estimated power consumption on the gate level by using the Power Compiler.it performs simultaneous optimization for timing, power and area, and measured by Watt W.
- ***Efficiency:*** It is the ability to minimize resource usage. Optimizing the speed and memory requirements of a computer program. 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 [9]. No more than 2,000 GE are available for security in low-cost RFID tags [12], [45], [9]. 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 [13]. In addition, the power required for 100 KHz RFIDs must be less than 27 µW power [56]. 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.

That makes the traditional cryptographic are not suitable to work on such devices. The traditional cryptographic algorisms focuses only on providing high level of security which requiring a complicated mathematical without focusing on the hardware requirement. The evaluation of the strength and effectiveness of existing primitives raises a serious challenge to researchers. Overcoming this challenges, it is necessary for realizing these limitations.

Tables 1 and 2 show a classification of different block cipher primitives and a comparison between them in terms of comprehensive area, power, and energy. In Table 1, we observe that KTANTAN family block cipher, SIMON and SPECK require much less GE as compared to other lightweight block ciphers. The similarity between these block ciphers is they use hardware-friendly operations such as shift registers in KTANTAN and AND, XOR and rotation in SIMON and SPECK. To increase diffusion property, all of these primitives spend many rounds. In the other hand, TWINE block cipher requires the highest GE. Looking at its structure, TWINE does not use circular shift but using 4-bit S-box and 4-bit permutation. The cost probably lies on its permutation layer which is not a simple circular shift although good diffusion is achieved through fewer round than that in KTANTAN, SIMON and SPECK. However, TWINE is superior than KTANTAN, SIMON and SPECK in terms of the number of cycle per block. Again, this is due to fewer number of rounds implemented in TWINE where only half as much rounds as a circular shift for one sub-block difference is needed to diffuse all the other sub-blocks. Among all, PRINCE requires only 1 cycle per block and this is due to its involutive structure that allows the encryption and decryption to be implemented in the same circuit.

From the classification, we can easily see that the Feistel structure-based primitives such as HIGHT, DESL, DESXL, KATAN and CLEFIA performs faster than SP-network-based primitives. However, in terms of hardware implementation we can see that SP-network is more hardware friendly where it require less GE.

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,

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

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-Damg°ard 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-Damg°ard construction (ARMADELO) and Davies-Meyer mode (DM-PRESENTS).

4. DISCUSSION ON THE SECURITY OF SMALL KEY LIGHTWEIGHT PRIMITIVES

Since the world is moving towards smart, small and light technologies, information security always becomes the main concern. As a consequence, several cryptographic primitives have been actively proposed and analyzed recently. The security of small keys deployed in these primitives is well defined and discussed in a standardization project of lightweight cryptography namely, ISO/IEC 29192 in ISO/IEC JTC 1/SC 27. In the project, several criteria were identified to evaluate the lightweight properties, viz: chip size and/or energy consumption for hardware implementation and code size and/or RAM size for software implementation. To provide adequate security, a minimum of 80-bit key is recommended [54]. We observe that these factors determine the significance of small key in securing a communication via resource constrained devices;

- The value of the information.

Usually a communication via resource constrained devices involve data with low level of confidentiality. It means that, with various type of attacks and attacker's processing capabilities, the time taken to deduce the key or retrieve the data exceeds the worthiness of the data at times when the data is retrieved.

- Cost of serious attack.

A report in [57] and rough estimation from Moore's law indicate that the cost to attack 75-bit security level would take more than 30 days with hundreds of million, i.e. the lifetime of time/success ratio would be larger than the life-time of the protected data. However, with the advance of storage and computational power, one should take into account the security level that is really desired for data protection. It means that small bits would be breakable with the correct technology and sophisticated attack are in place.

- Applications with special properties.

There are some applications with special requirements and properties that allow low level security requirement, i.e. to balance out the needs of data security and performance-based business applicability. Practically, small organizations might prefer to have medium term data protection as the security safeguards involve high cost and big organizations might prefer medium and long term data protection for different categories of their data.

- The security of small key.

The security of small key becomes the topic of interest by many cryptographers recently. In principle, we can move into having a more secure communication provided that we are flexible to transition to longer key lengths to sustain data confidentiality over a period of time.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Table 1. Classification of Block Lightweight Primitives

Primitives	Structure	Block [bits]	Key [bits]	cycle per block	Throughput at 100 KHz	Power µA	Logic process µm	GE
DESL [6].	Feistel structure	64	56	144	44.4	O.89	0.18	1848
DESX [6].		64	128	144	44.4	1.42	0.18	2629
DESXL [6].		64	184	144	44.4	1.17	0.18	2168
KATAN32 [35]		32	80	245	12.5	3.81	0.13	802
KATAN48 [35]		48	80	245	18.8	4.39	0.13	927
KATAN64 [35]		64	80	245	25.1	5.55	0.13	1054
KTANTAN32 [35]		32	80	245	12.5	1.46.	0.13	462
KTANTAN48 [35]		48	80	245	18.8	2.34.	0.13	588
KTANTAN64 [35]		64	80	245	25.1	2.92.	0.13	688
CLEFIA [34]		128	128	18	268.63	N/A	0.9	5979
		128	196	22	140.81	N/A	0.9	8536
		128	256	26	119.89	N/A	0.9	8482
HIGHT [29]		64	128	34	188.20	1.65	0.25	3048
LBlock [38]		64	80	3955	200	N/A	0.18	1320
SIMON 48/96 [42]		48	96	187	5.0	N/A	0.13	739
SPECK 48/96 [42]		48	96	104	4.0	N/A	0.13	794
SIMON 64/96 [42]		64	96	205	4.4	N/A	0.13	809
SPECK 64/96 [42]		64	96	114	3.6	N/A	0.13	860
SIMON 64/128 [42]		64	128	217	4.2	N/A	0.13	958
SPECK 64/128 [42]		64	128	118	3.4	N/A	0.13	996
SIMON 96/96 [42]		96	96	249	3.7	N/A	0.13	955
SPECK 96/96 [42]		96	96	123	3.4	N/A	0.13	1012
SIMON 128/128 [42]		128	128	333	2.9	N/A	0.13	1234
SPECK 128/128 [42]		128	128	139	3.0	N/A	0.13	1280
TWINE [40]		64	80	36	178	N/A	0.09	1799
TWINE [40]		64	128	36	178	N/A	0.09	2285
QTL [44]		64	64	16	200	N/A	0 .18	1026
AES-128 [2]	SP-network	128	128	992	12.4	8.15	0.35	3,628
PRESENT-80 [8].		64	80	32	200	5	0.18	1570
PRESENT-128 [8].		64	128	32	200	3.3	0.18	1886
Humming bird-2 [58]		16	128	4	400.00	4.17	0.13	3220
LED-64 [39]		64	64	1,248	5.1	1.67	0.18	966
LED-80 [39]		64	80	1,872	3.4	2.2	0.18	1040
LED-96 [39]		64	96	1,872	3.4	2.2	0.18	1116
LED-128 [39]		64	128	1,872	3.4	2.2	0.18	1265
PRINCE [41]		64	128	1	N/A	4.1	0.13	8679
PRINTcipher-48 [37]		48	80	768	6.25	2.6	0.18	402
PRINTcipher-48 [37]		48	80	48	100	2.6	0.18	503
PRINTcipher-96 [37]		96	160	3072	3.13	2.6	0.18	726
PRINTcipher-96 [37]		96	160	96	100	2.6	0.18	967

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

KLEIN-64 [43]		64	64	64	0.44	N/A	0.18	1981
KLEIN-80 [43]		64	80	64	0.57	N/A	0.18	2097
KLEIN-96 [43]		64	96	64	0.71	N/A	0.18	2213

Table 2. Comparison of Performances of Lightweight Hash Functions.

Primitives		**Hash output size**	**Data path size**	**Cycles per block**	**Throughput 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 Keccakf[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

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

		256	272	140	11.43	6.62	0.13	3281
Neeva [55]		224	32	12067	4.99	-	-	
ARMADILLO **[48]**	Merkle–Damgård	48	80	176	272	44	0.18	2,923

5- CONCLUSION

In this paper, we gives comprehensive survey of lightweight symmetric cryptographic primitives which include lightweight block ciphers and lightweight hash algorithm. These classifications are very useful because the primitives have different and sometimes contrary characteristics. Also, this survey highlights some of the issues related to security aspect of small key length in lightweight cryptographic primitives. The structures show that primitive designs with hardware-friendly operations give a smaller GE. Also, parallel design gives faster performance but higher GE than serialized design. The observation from the classification would help the researcher to further investigate an optimized design of lightweight cryptographic primitives that balance the security aspect and resource constraints.

REFERENCES

[1] Constantin, J., Burg, A., & Gürkaynak, F. K. (2012). Investigating the Potential of Custom Instruction Set Extensions for SHA-3 Candidates on a 16-bit Microcontroller Architecture. IACR Cryptology ePrint Archive, 2012, 50.

[2] Elbirt, A. J. (2007, May). 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, pp. 396-403). IEEE.

[3] Grabher, P., Großschädl, J., & Page, D. (2008). Light-weight instruction set extensions for bit-sliced cryptography. In Cryptographic Hardware and Embedded Systems–CHES 2008 (pp. 331-345). Springer Berlin Heidelberg.

[4] Hodjat, Alireza, and Ingrid Verbauwhede. (2004, November). 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, pp. 488-492). IEEE.

[5] O'Melia, Sean, and Adam J. Elbirt (2008, December). Instruction Set Extensions for Enhancing the Performance of Symmetric-Key Cryptography. In Computer Security Applications Conference, 2008. ACSAC 2008. Annual (pp. 465-474). IEEE.

[6] Poschmann, A., Leander, G., Schramm, K., & Paar, C. (2007). New Lightweight DES Variants Suited for RFID Applications. In FSE (Vol. 4593, pp. 196-210).

[7] Standard, N. F. (1999). Data Encryption Standard (DES). Federal Information Processing Standards Publication.

[8] Bogdanov, A., Knudsen, L. R., Leander, G., Paar, C., Poschmann, A., Robshaw, M. J. ... & Vikkelsoe, C. (2007). PRESENT: An ultra-lightweight block cipher. In Cryptographic Hardware and Embedded Systems-CHES 2007 (pp. 450-466). Springer Berlin Heidelberg.

[9] Guo, J., Peyrin, T., & Poschmann, A. (2011a). The PHOTON family of lightweight hash functions. In Advances in Cryptology–CRYPTO 2011 (pp. 222-239). Springer Berlin Heidelberg.

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

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

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

[13] Batina, L., Das, A., Ege, B., Kavun, E. B., Mentens, N., Paar, C. ... & Yalçın, T. (2013). Dietary recommendations for lightweight block ciphers: Power, energy and area analysis of recently developed architectures. In Radio Frequency Identification (pp. 103-112). Springer Berlin Heidelberg.

[14] Juels, A. (2006). RFID security and privacy: A research survey. Selected Areas in Communications, IEEE Journal on, 24(2), 381-394.

[15] Lata, Manju, and Adarsh Kumar. (2014). Survey on Lightweight Primitives and Protocols for RFID in Wireless Sensor Networks. International Journal of Communication

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Networks and Information Security (IJCNIS), 6(1).

[16] 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.

[17] 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.

[18] 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.

[19] Aumasson, J. P., Henzen, L., Meier, W., & Naya-Plasencia, M. (2013). Quark: A lightweight hash. Journal of cryptology, 26(2), 313-339.

[20] Bogdanov, A., Knezevic, M., Leander, G., Toz, D., Varici, K., & Verbauwhede, I. (2013). Spongent: The design space of lightweight cryptographic hashing. Computers, IEEE Transactions on, 62(10), 2041-2053.

[21] Poschmann, A. Y. (2009). Lightweight cryptography: cryptographic engineering for a pervasive world. In PH. D. THESIS.

[22] Hankerson, D., Vanstone, S., & Menezes, A. J. (2004). Guide to elliptic curve cryptography. Springer.

[23] Rivest, R. L., Shamir, A., & Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM, 21(2), 120-126.

[24] Odlyzko, A. M. (1985, January). Discrete logarithms in finite fields and their cryptographic significance. In Advances in cryptology (pp. 224-314). Springer Berlin Heidelberg.

[25] Chien, Hung-Yu, and Chi-Sung Laih. (2009). ECC-based lightweight authentication protocol with untraceability for low-cost RFID. Journal of parallel and distributed computing, 69(10), 848-853.

[26] Batina, L., Guajardo, J., Kerins, T., Mentens, N., Tuyls, P., & Verbauwhede, I. (2006). An Elliptic Curve Processor Suitable For RFID-Tags. IACR Cryptology ePrint Archive, 2006, 22.

[27] Feldhofer, M., Dominikus, S., & Wolkerstorfer, J. (2004). Strong authentication for RFID systems using the AES algorithm. In Cryptographic Hardware and Embedded Systems-CHES 2004 (pp. 357-370). Springer Berlin Heidelberg.

[28] Daemen, Joan, and Vincent Rijmen (1999). AES Proposal: Rijndael. AES Algorithm Submission, September 3, 1999. URL http://www. nist. gov/CryptoToolKit.

[29] Hong, D., Sung, J., Hong, S., Lim, J., Lee, S., Koo, B. S. ... & Chee, S. (2006). HIGHT: A new block cipher suitable for low-resource device. In Cryptographic Hardware and Embedded Systems-CHES 2006 (pp. 46-59). Springer Berlin Heidelberg.

[30] Barreto, P. S. L. M., and M. Simplicio. (2007). CURUPIRA, a block cipher for constrained platforms. In 5th Brazilian Symposium on Computer Networks and Distributed Systems (pp. 61-74).

[31] Simplicio Jr, M., Barreto, P. S., Carvalho, T. C., Margi, C. B., & Näslund, M. (2008). The CURUPIRA-2 block cipher for constrained platforms: Specification and benchmarking. Barreto, P. S. L. M., & Rijmen, V. (2000). The Anubis block cipher. Submission to the NESSIE Project.

[32] Barreto, P. S. L. M., and Vincent Rijmen. (2000b). The Anubis block cipher, NESSIE Algorithm Submission.

[33] Barreto, P. S. L. M., and Vincent Rijmen (2000). The Khazad legacy-level block cipher. Primitive submitted to NESSIE, 97.

[34] Shirai, T., Shibutani, K., Akishita, T., Moriai, S., & Iwata, T. (2007, January). The 128-bit blockcipher CLEFIA. In Fast software encryption (pp. 181-195). Springer Berlin Heidelberg.

[35] De Canniere, C., Dunkelman, O., & Knežević, M. (2009). KATAN and KTANTAN—a family of small and efficient hardware-oriented block ciphers. In Cryptographic Hardware and Embedded Systems-CHES 2009 (pp. 272-288). Springer Berlin Heidelberg.

[36] Canniere, C. De, and B. Preneel. (2005). Trivium specifications. ESTREAM. ECRYPT Stream Cipher Project, Report, 30, 2005.

[37] Knudsen, L., Leander, G., Poschmann, A., & Robshaw, M. J. (2010). PRINTcipher: a block cipher for IC-printing. In Cryptographic Hardware and Embedded Systems, CHES 2010 (pp. 16-32). Springer Berlin Heidelberg.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

[38] Wu, Wenling, and Lei Zhang. (2011, January). LBlock: a lightweight block cipher. In Applied Cryptography and Network Security (pp. 327-344). Springer Berlin Heidelberg.

[39] Guo, J., Peyrin, T., Poschmann, A., & Robshaw, M. (2011). The LED block cipher. In Cryptographic Hardware and Embedded Systems–CHES 2011 (pp. 326-341). Springer Berlin Heidelberg.

[40] Suzaki, T., Minematsu, K., Morioka, S., & Kobayashi, E. (2011, November). Twine: A lightweight, versatile block cipher. In ECRYPT Workshop on Lightweight Cryptography (pp. 146-169).

[41] Borghoff, J., Canteaut, A., Güneysu, T., Kavun, E. B., Knezevic, M., Knudsen, L. R. ... & Yalçın, T. (2012). PRINCE–A low-latency block cipher for pervasive computing applications. In Advances in Cryptology–ASIACRYPT 2012 (pp. 208-225). Springer Berlin Heidelberg.

[42] Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., & Wingers, L. (2013). The Simon and speck families of lightweight block ciphers. Cryptology ePrint Archive, Report 2013/404, 2013. http://eprint. iacr. Org.

[43] Gong, Z., Nikova, S., & Law, Y. W. (2012). KLEIN: a new family of lightweight block ciphers. In RFID. Security and Privacy (pp. 1-18). Springer Berlin Heidelberg.

[44] Li, L., Liu, B. and Wang, H., 2016. QTL: A new ultra-lightweight block cipher. Microprocessors and Microsystems.

[45] Bogdanov, A., Leander, G., Paar, C., Poschmann, A., Robshaw, M. J., & Seurin, Y. (2008). Hash functions and RFID tags: Mind the gap. In Cryptographic Hardware and Embedded Systems–CHES 2008 (pp. 283-299). Springer Berlin Heidelberg.

[46] Berger, T. P., D'Hayer, J., Marquet, K., Minier, M., & Thomas, G. (2012). The GLUON family: a lightweight Hash function family based on FCSRs. In Progress in Cryptology-AFRICACRYPT 2012 (pp. 306-323). Springer Berlin Heidelberg.

[47] Kavun, Elif Bilge, and Tolga Yalcin. (2010). A lightweight implementation of Keccak hash function for radio-frequency identification applications. In Radio frequency identification: security and privacy issues (pp. 258-269). Springer Berlin Heidelberg.

[48] Badel, S., Dağtekin, N., Nakahara Jr, J., Ouafi, K., Reffé, N., Sepehrdad, P. ... & Vaudenay, S. (2010). ARMADILLO: a multi-purpose cryptographic primitive dedicated to hardware. In Cryptographic Hardware and Embedded Systems, CHES 2010 (pp. 398-412). Springer Berlin Heidelberg.

[49] Menezes, Alfred J., Paul C. Van Oorschot, and Scott A. Vanstone (1996). Handbook of applied cryptography. CRC press.

[50] Macé, F., Standaert, F. X., & Quisquater, J. J. (2008). FPGA implementation (s) of a scalable encryption algorithm. Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, 16(2), 212-216. Menezes, A. J., Van Oorschot, P. C., & Vanstone, S. A. (1996). Handbook of applied cryptography. CRC press.

[51] Daemen, Joan, and Vincent Rijmen (2002). The design of Rijndael: AES-the advanced encryption standard. Springer.

[52] Arnault, F., Berger, T., Lauradoux, C., Minier, M., & Pousse, B. (2009). A New Approach for FCSRs, Selected Areas in Cryptography: 16th Annual International Workshop, SAC 2009, Calgary, Alberta, Canada, August 13-14, 2009, Revised Selected Papers.

[53] Juels, Ari, and Stephen A. Weis. (2005, January). Authenticating pervasive devices with human protocols. In Advances in Cryptology–CRYPTO 2005 (pp. 293-308). Springer Berlin Heidelberg.

[54] Guo, Xu, and Patrick Schaumont (2011, November). The Technology Dependence of Lightweight Hash Implementation Cost. In ECRYPT Workshop on Lightweight Cryptography 2011.

[55] Bussi, K., Dey, D., Kumar, M. and Dass, B.K., 2016. Neeva: A Lightweight Hash Function.

[56] Manifavas, C., Hatzivasilis, G., Fysarakis, K., & Rantos, K. (2014). Lightweight Cryptography for Embedded Systems–A Comparative Analysis. In Data Privacy Management and Autonomous Spontaneous Security (pp. 333-349). Springer Berlin Heidelberg.

[57] Dutertre, B., Cheung, S., & Levy, J. (2004). Lightweight key management in wireless sensor networks by leveraging initial trust. Technical Report SRI-SDL-04-02, SRI International.

[58] Engels, D., Saarinen, M. J. O., Schweitzer, P., & Smith, E. M. (2012). The Hummingbird-2 lightweight authenticated encryption algorithm. In RFID. Security and Privacy (pp. 19-31). Springer Berlin Heidelberg.

[59] Ågren, M. (2012). On some symmetric lightweight cryptographic designs.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

[60] Izadi, M., Sadeghiyan, B., Sadeghian, S. S., & Khanooki, H. A. (2009, December). MIBS: a new lightweight block cipher. In International Conference on Cryptology and Network Security (pp. 334-348). Springer, Berlin, Heidelberg.

[61] Babbage, Steve, and Matthew Dodd. (2008). "The MICKEY stream ciphers." In New Stream Cipher Designs, pp. 191-209. Springer Berlin Heidelberg, 2008.

[62] Kumar, Naveen, Shrikant Ojha, Kritika Jain, and Sangeeta Lal. (2009) "BEAN: a lightweight stream cipher." In Proceedings of the 2nd international conference on Security of information and networks, pp. 168-171. ACM, 2009.

[63] Fan, Xinxin, Kalikinkar Mandal, and Guang Gong. (2013). "Wg-8: A lightweight stream cipher for resource-constrained smart devices." In International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness, pp. 617-632. Springer, Berlin, Heidelberg, 2013.

[64] Ghafari, Vahid Amin, Honggang Hu, and Chengxin Xie. (2016). "Fruit: ultra-lightweight stream cipher with shorter internal state." eSTREAM, ECRYPT Stream Cipher Project, https://eprint. iacr. org/2016/355. pdf (2016).

[65] Hamann, Matthias, Matthias Krause, and Willi Meier. (2017). "LIZARD–A lightweight stream cipher for power-constrained devices." IACR Transactions on Symmetric Cryptology 2017, no. 1 (2017): 45-79.

[66] McKay, Kerry A., Larry Feldman, and Gregory A. Witte. (2017) "Toward Standardizing Lightweight Cryptography." ITL Bulletin- (2017).

[67] Li, Song, Jie Cui, Hong Zhong, Yiwen Zhang, and Qiang He. (2017) "LEPA: A Lightweight and Efficient Public Auditing Scheme for Cloud-Assisted Wireless Body Sensor Networks." Security and Communication Networks 2017 (2017).

[68] Toorani, Mohsen, and A. Beheshti. (2008). "LPKI-a lightweight public key infrastructure for the mobile environments." In Communication Systems, 2008. ICCS 2008. 11th IEEE Singapore International Conference on, pp. 162-166. IEEE, 2008.

[69] Patil, Jagdish, Gaurav Bansod, and Kumar Shashi Kant. (2017). "LiCi: A new ultra-lightweight block cipher." In Emerging Trends & Innovation in ICT (ICEI), 2017 International Conference on, pp. 40-45. IEEE, 2017.

[70] Ali, Arshad. (2017). "Oppel-1: A new block cipher." In Applied Sciences and Technology (IBCAST), 2017 14th International Bhurban Conference on, pp. 441-447. IEEE, 2017.

[71] Martin, Honorio, Pedro Peris Lopez, Enrique San Millan, and Juan E. Tapiador. (2017). "A lightweight implementation of the Tav-128 hash function." IEICE Electronics Express 14, no. 11 (2017): 20161255-20161255.