Lightweight Cryptography Methods

Prof William J. Buchanan (1), Dr Shancang Li (2), Dr Rameez Asif (1)
(1) The Cyber Academy, Edinburgh Napier University, UK
(2) University of the West of England, UK

***Abstract*—While our conventional cryptography methods, such for AES (encryption), SHA-256 (hashing) and RSA/Elliptic Curve (signing), work well on systems which have reasonable processing power and memory capabilities, these do not scale well into a world with embedded systems and sensor networks. Thus lightweight cryptography methods are proposed to overcome many of the problems of conventional cryptography. This includes constraints related to physical size, processing requirements, memory limitation and energy drain. This paper outlines many of the techniques that are defined as replacements for conventional cryptography within an Internet of Things (IoT) space and discuss some trends in the design of lightweight algorithms.**

Index Terms— Lightweight cryptography, Resource limited devices, encryption, Hashing functions. PHOTON, SPONGENT, Lesamanta-LW, Enocoro, Trivium, PRESENT, CLEFIA

I. INTRODUCTION

While AES and SHA work well together within computer systems, they struggle in an IoT/embedded world as they take up: too much processing power; too much physical space; and consume too much battery power. In the last decade, a large number of lightweight cryptography primitives have been proposed and used over resource-limited devices. Both the national (NIST) and international (ISO/IEC) organizations outline a number of methods which can be used for lightweight cryptography, and which could be useful in IoT and RFID devices [1]. They define the device spectrum as:

- **Conventional cryptography**. Servers and Desktops. Tablets and smart phones.
- **lightweight cryptography.** Embedded Systems. RFID and Sensor Networks.

With embedded systems, we commonly see 8-bit, 16-bit and 32-bit microcontrollers, and which would struggle to cope with real-time demands for conventional cryptography methods. And in the 40+ years since the first 4-bit processor, there is even a strong market for 4-bit processors. RFID and sensor network devices, especially, have limited numbers of gates available for security, and are often highly constrained with the power drain on the device.

So AES is often a non-starter for many embedded devices. In lightweight cryptography, we often see smaller block size (typically 64 bits or 80 bits), smaller keys (often less than 90 bits) and less complex rounds (and where the S-boxes often just have 4-bits). Along with this it has been identified as having weaknesses our side channel attacks. In [2], the researchers attack a 128-bit AES key on an Arduino device using differential power analysis (DPA) and correlation power analysis (CPA), and crack it within 30 minutes.

For lightweight cryptography, the main constraints that we have are typically related to power requirements, gate equivalents (GEs), and timing. With passive RFID devices, we do not have an associated battery for the power supply, and where the chip must power itself from energy coupled from the radio wave. An RFID device is thus likely to be severely constrained in the power drain associated with any cryptography functions, along with being constrained for the timing requirements and for the number of gates used. Even if an RFID device has an associated battery (active RFID), it may be difficult to recharge the battery, so the drain on power must often be minimised.

On the other hand, the IoT are unleashing the next wave of innovations due to its inherent capability of connecting intelligent ‘things’ in a physical world into cloud-based information technology architecture. The data and privacy protection in IoT is fundamental to the success of IoT and it will present new security challenges in cryptographic security, credentialing, and identity management [3].

There is thus often a compromise between the cryptography method used and the overall security of the method. Thus often lightweight cryptography methods balance performance (throughput) against power drain and GE, and do not perform as well as main-stream cryptography standards (such as AES and SHA-256). Along with this the method must also have a low requirement for RAM (where the method requires the usage of running memory to perform its operation) and ROM (where the method is stored on the device). In order to assess the strengths of various methods we often define the area that the cryptography function will use on the device – and which is defined in μm^2.

In the Internet of Things (IoT), many interconnected resource constrained devices are not designed to carry out expensive conventional cryptographic computation, which makes it difficult to implement sufficient cryptographic functions. To guarantee security and privacy protection in the IoT becomes a serious concern when integrating resource constrained devices into the IoT securely since they are incapable of carrying out sufficient cryptographic algorithms [3].

In recent, the lightweight symmetric cryptography has been developed for IoT, including hash functions and MACs like Quark, Marvin, and block/streaming ciphers such as PRESENT, SPONGENT, and so on. Asymmetric cryptography that can be used for IoT includes number-theoretic cryptography, such as ECC, PBC, etc., post-quantum cryptography lattices and codes [4]. Since most IoT devices are

working on a multi-task mode, so the software performance is crucial for the lightweight cryptography and existing lightweight solutions such as Chaskey, FLY, LEA, SPARX, etc. show good evaluation results [5]. In the IoT case, the cipher types, block size, key size, relevant attacks, etc. should be taken into considerations.

In the lightweight cryptanalysis, typical attacks include single-key/related-key, distinguisher/key-recovery, weak-keys, meet-in-the-middle-attack, etc. In [4], a lightweight crypto-analysis model is addressed to against generic attacks. For lightweight cryptography, the size of key, block and tag are usually considered in cryptanalysis, specifically, for multi-key attacks, power of precomputation, brute-force attacks, etc. In many cases, the applicability over resourced-limited devices are crucial.

In the past decade, a number of lightweight cryptography protocols, algorithms, and primitives have been standardized as the ISO/ICE 29121. For security communication, the lightweight primitives have been embedded into existing protocols, such as IPSec, TLS, and a number of embedded cryptography libraries have been released such as wolfSSL, CyaSSL, sharkSSL, RELIC, etc.

II. Challenges in Lightweight Cryptography

Lightweight cryptography targets a very wide variety of resource constrained devices such as IoT end nodes and RFID tags [6] that can be implemented on both hardware and software with different communication technologies. It is very difficult for resource-limited environment to implement the standard cryptographic algorithms due to the implementation size, speed or throughput, and energy-consumption. The lightweight cryptography trade-offs implementation cost, speed, security, performance, and energy consumption on resource-limited devices. The motivation of lightweight cryptography is to use less memory, less computing resource, less power supply to provide security solution that can work over resource-limited devices. The lightweight cryptography is expected simpler and faster compare to conventional cryptography. The disadvantage of lightweight cryptography is less secured [6].

A. Hardware Implementation

In hardware implementation of the lightweight cryptography primitives, the code size, the memory consumption (RAM), and energy consumption are the important metrics. To well evaluate the lightweight cryptography, the exact type of circuit (such as the clock), memory, storing of the internal states and key states should be taken into consideration. However, it does not mean shorter block and key size are better since it may cause insecure against related-key attacks [6]. In some case, the read-only 'Mask' technology is used to burn keys into devices (chips) to reduce the key space. In recent, in [7] an energy efficiency of hardware implementation metric is proposed, in which the latency is used to evaluate the time taken to perform a given operation [8].

B. Software Implementation

In software implementation case, the implementation size, and RAM consumption, and the throughput (bytes per cycle) are preferable metrics for the lightweight applications [5]. The smaller the better. In software cases, the unified FELICS (Fair Evaluation of lightweight Cryptographic Systems) framework is proposed to evaluate the performances of lightweight block or stream ciphers' performances in implementation size, RAM consumption, and time taken to perform a given operation [9]. Table 1 shows the FELICS results of popular lightweight cipher algorithms for three different implementations: 8-bit AVR, 6-bit MSP, and 32-bit ARM.

Due to the circuit implementation, the implementation size, RAM consumption and the throughput are not independent and reduce the number of operations can decrease both memory and time consumption [6][9].

Table 1. FELICS results for lightweight ciphers [6]

General info			AVR (8-bit)			MSP (16-bit)			ARM (32-bit)		
Name	block	key	Code	RAM	Time	Code	RAM	Time	Code	RAM	Time
Chaskey	128	128	770	84	1597	490	86	1351	178	80	614
Speck	64	96	448	53	2829	328	48	1959	256	56	1003
Speck	64	128	452	53	2917	332	48	2013	276	60	972
Chaskey-LTS	128	128	770	84	2413	492	86	2064	178	80	790
Simon	64	96	600	57	4269	460	56	2905	416	64	1335
Simon	64	128	608	57	4445	468	56	3015	388	64	1453
Lea	128	128	906	80	4023	722	78	2814	520	112	1171
Rectangle	64	128	602	56	4381	480	54	2651	452	76	2432
Rectangle	64	80	606	56	4433	480	54	2651	452	76	2432
Sparx	64	128	662	51	4397	580	52	2261	654	72	2338
Sparx	128	128	1184	74	5478	1036	72	3057	1468	104	2935
RC5-20	64	128	1068	63	8812	532	60	15925	372	64	1919
Aes	128	128	1246	81	3408	1170	80	4497	1348	124	4044
Hight	64	128	636	56	6231	636	52	7117	670	100	5532
Fantomas	128	128	1712	76	9689	1920	78	3602	2184	184	4550
Robin	128	128	2530	108	7813	1942	80	4913	2188	184	6250

C. Lightweight Cryptography Design Trends

Based on metrics discussed both in hardware case and software case, most lightweight algorithms are designed to use smaller internal states, short block and key sizes. Indeed, most lightweight block ciphers use only 64 bit blocks (AES is demanded a 128-bit block and a 128-bit key). The lightweight implementation usually leads smaller RAM consumption and it is good at processing smaller messages as well. In designing lightweight cryptography solutions, following trends are noticed: (1) Short block and key size will bring problems: short block can cause problems such as CBC erodes faster than other part when the number of n-bit blocks encrypted approaches $2^{n/2}$ [9], meanwhile the short key size can increase the risks of key-related attacks [3]; (2) The number of operations in symmetric lightweight cryptography roughly doubles when the input size of a symmetric-key primitive double [5]. In PHOTON family, the number of rounds is always 12, the number of S-box doubles if the size doubles. Similarly, in AES 256, the number of rounds is 14, the number of s-box doubles if the block size doubles; (3) The lightweight cryptography always is driven by the applications; as a result, lightweight primitives should be designed to apply new academic insights as well as to best match existing protocols.

III. Methods

For lightweight cryptography PHOTON [10], SPONGENT [4] and Lesamanta-LW [11] are defined as standards for hashing methods within ISO/IEC 29192-5:2016, PRESENT and CLEFIA for block methods within ISO/IEC 29192-2:2012, and Enocoro and Trivium for stream methods within ISO/IEC 29192-3:2012.

A. Hashing

While we will all have 32-bit or 64-bit processors in our mobile phones and desktops, and have much more the 1GB of memory, in an IoT world we often measure memory capacity in just a few KiloBytes (KB), and where 8-bit processors rule the roost. The cost of a simple 8-bit processor can be defined in 10s of cents, compared with hundreds of dollars for our complex processors. And so our crypto hash functions for MD5 and SHA-1, and most of our other modern hash methods, are just not efficient for IoT devices. NIST have thus recommended new hashing methods such SPONGENT, PHOTON, Quark and Lesamnta-LW. These methods produce a much smaller memory footprint, and have a target an input of just 256 characters (whereas typically hash functions support up to 2^{64} bits).

SPONGENT uses the sponge function (Figure 1) [4]. With the sponge construction, we use a fixed-length permutation (or transformation) and a padding rule. This construction thus takes a variable length input and map it to a variable-length output. The input is $(Z2)^*$ of any length and then converts it into $(Z2)^n$, where *n* is defined as part of the process. Overall the method uses a finite-state machine process, and iterates through the states with the addition of the input data. The concept of sponge function was created Bertoni, who created Keccak [12]. They can use either use a publicly known unkeyed permutation (P-Sponge) or with a random function (T-Sponge). Along with their usage in hashing, they can also be used in creating stream ciphers.

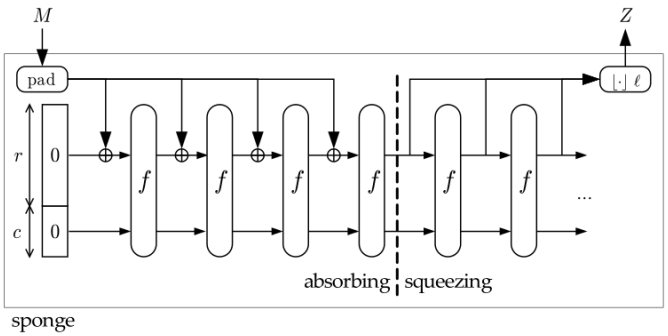

Figure 1 Sponge function [4]

The sponge construction uses a function *f* which has a variable-length input and a defined output length. It operates on a fixed number of bits (*b*) - the width. The sponge construction then operates on a state of *b=r+c* bits. *r* is defined as the bitrate and c as the capacity (Figure 1). Initially an input string is padded using a reversible padding rule (such as adding NULL characters), and then segmented into blocks of *r* bits. Next the b bits of the state are set to zero, and the sponge construction next defines:

- **Absorbing phase.** This is where the r-bit input blocks are X-ORed into the first *r* bits of the state, interleaved with applications of the function *f*. After all the input blocks have been processed, we then move to a squeezing phase.
- **Squeezing phase**. This is where the first *r* bits of the state are outputted as blocks and, interleaved with the function f. The number of bits of the output are defined as part of the process.

Overall the last *c* bits of a state are never changed by the input blocks and never output within the squeezing phase. For an 88-bit hash we have (SPONGENT-088-080-00 - Spongent-88/80/8: n=88 bits, b=88 bits, c=80 bits, r=8 bits, R=45) and for 128-bit (SPONGENT-128-128-008 - Spongent-128/128/8: n=128 bits, b=136 bits, c=128 bits, r=8 bits, R=70) [13].

Lesamnta-LW which uses AES methods as its core [11]. One thing to notice about Lesamnta-LW is that the S-box structure is the same as you would find in AES. The authors think that it only requires 8.24 kGates, and has a throughput of 125Mbit/sec (which is five times faster than SHA-256, which also gives a 256-bit hash): For the RAM requirements on an 8-bit processor, the authors estimate that Lesamnta-LW only requires 50 bytes of RAM [14].

Quark is defined in three main methods: u-Quark, d-Quark, and s-Quark, and uses a sponge function [15]. It can be used for hashing and in stream encryption. u-Quark has the lowest footprint and provides 64-bit security on 1379 digital gate, whereas s-Quark provides 112-bit security [16].

Keccak is a family of cryptographic sponge functions that has become the FIPS 202 (SHA-3) standard in 2015 [17]. The Keccak is based on the sponge construction, in which the underlying function is a permutation chosen in a set of seven Keccak-f permutations, denoted as Keccak-*f* (25, 50, 100, 200, 400, 800, 1600) with seven different width of the permutation of {1, 2, 4, 8, 16, 32, 64}. The Keccak can provide nice flexibility and good performance both in hardware and software with moderate implementation size and RAM consumption and suitable for lightweight applications.

PHOTON is lightweight cryptography method for hashing and is based on an AES-type approach [10]. It can create 80-bit, 128-bit, 160-bit, 224-bit and 256-bit hashes [10]. It takes an arbitrary-length input and produces a variable-length output.

The method is defined as PHOTON-n-r-r′, where *n* is the hash size, *r* is the input bit rate, and r′ is the output bit rate. Sample hashed values for "abc" are [18]:

```
Photon 80 signature (PHOTON-80/20/16)
("abc") = 3151cb8f09f5a4908531
------------
Photon 128 signature (PHOTON-128/16/16)
("abc") = e1bb314c7c9ace3ea0ed6fd1d762d216
------------
Photon 160 signature (PHOTON-160/36/36)
("abc") = c11d4cd3da84bc245430ba7cf696d0092941ba58
------------
Photon 224 signature (PHOTON-224/32/32)
("abc")                                                 =
7798abbae697af77eaa56f358ec9845ee947c6d3c7daca9e7ae4
76ec
------------
Photon 256 signature (PHOTON-256/32/32)
("abc")                                                 =
c412435e329f6f4837a5e55eda83d66d8a8eae5d9744931f9c7c
bb7e55584df6
------------
```

The internal state is defined as *t* (bits), and is calculated as *t=c+r* (where *c* is the capacity). With PHOTON we use a sponge function and where we take input bits and XOR with bits taken from the current state. Overall there are three main phases:

- **Initialisation**. This phase takes the input bit stream and breaks into *r* bits (and pads if required).
- **Absorbing**. In this phase, for all the message blocks, we take *r*-input bits and XOR with *r* bits of the state, and interleave with a *t*-bit permutation function.
- **Squeezing**. In this phase, we extract r bits from the current internal state, and apply a permutation function (P) to it. This will continue until the number of output bits is equal to the required hash size.

The internal permutation function (P) is similar to AES with 12 rounds, and which each round has the functions of (Figure 2):

- **AddConstants**. In this function, the first column with the internal state is XOR-ed with round (r) and internal constants.
- **SubCells**. In this function, the internal state is fed through the PRESENT S-box (Figure 5).
- **ShiftRows**. In this function, the internal state cell row [*i*] is cyclically shifted by *i* positions to the left.
- **MixColumnsSerial**. In this function, the internal state cell column is multiplied by the MDS (Maximum Distance Separable) matrix.

Table. 2 shows a summary of lightweight hashing functions commonly used in both academic and industry [6].

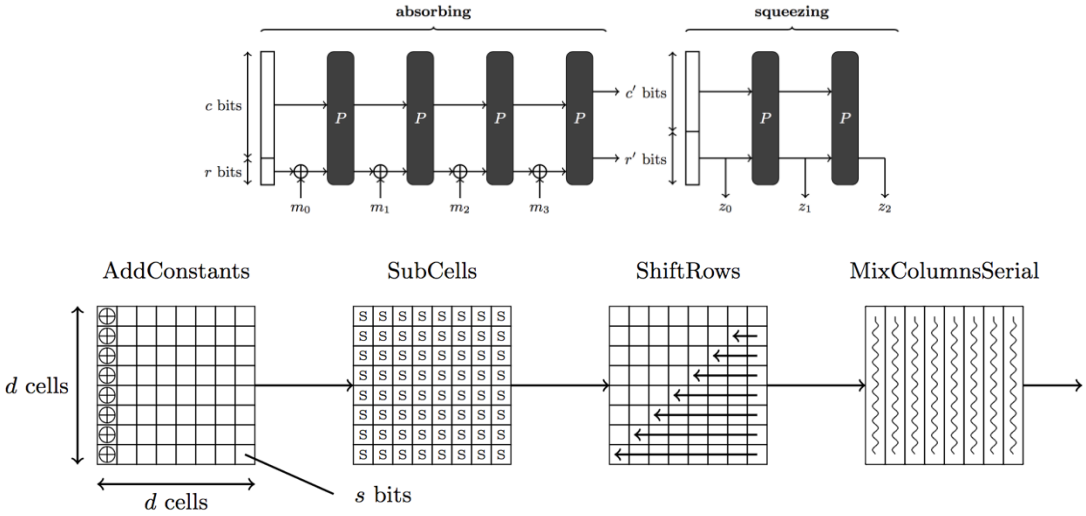

Figure 2 PHOTON functions [10]

Table 2. lightweight hash functions [6]

lightweight Hash Function	Digest [bits]	Code [bytes]	RAM [bytes]	RAM [bytes]	RAM stack	Cycle (8-byte msg)	Cycle (50-byte msg)	Cycle (100-byte msg)	Cycle (500-byte msg)
SPONGENT-256/256/128	256	364	16	96	5	1 542 923	3 856 916	6 170 900	25 454 100
SPONGENT-160/160/80	160	598	10	60	6	795 294	2 783 241	4 771 186	20 674 746
S-Quark	256	1106	4	60	5	708 783	1 417 611	2 339 023	9 4270 23
D-Quark	176	974	2	42	5	631 871	1 516 685	2 570 035	10 996 835
Keccak[r=40,c=160]	160	752	5	45	3	58 063	162 347	278 269	1 205 627
Keccak[r=144,c=256]	256	608	18	92	4	90 824	181 466	317 221	1 313 291
PHOTON-160/36/36	160	764	9	39	11	620 921	1 655 364	2 793 265	11 999 914
PHOTON-256/32/32	256	1244	4	68	10	254 871	486 629	787 896	3 105 396

B. Streaming

One of the first to show promise for a replacement for AES for lightweight cryptography is PRESENT [19]. It uses smaller block sizes and the potential for smaller keys (such as for an 80-bit key). PRESENT users either an 80-bit (10 hex characters) or a 128-bit encryption key (16 hex characters). It operates on 64-bit blocks and uses an SPN (substitution-permutation network) method. With SPN, as with AES (Rijndael), we operate on blocks of plaintext and apply a key and then use a number of rounds which we use substitution boxes (S-boxes) and permutation boxes (P-boxes). The operations used are typically achieved through XOR/bitwise rotation, and parts of the key are introduced though the rounds of operation. The decryption process is then the reverse of the encryption rounds, and the S-boxes/P-boxes are reversed in their operation.

Within Figure 3 we see an example of a single round, and where 8-bits of data is entered, and then EX-OR with the first eight bits of the key. Next the output from this operation is fed into an S-box which maps in the inputs to the output (for example 0x0 will be mapped to 0xC. After this we feed the output into a P-box which will scramble the bits in a defined way. The output of this is then fed into the next round, and

which will follow the same process, but this time our input is from the previous round, and from the next eight bits of the key.

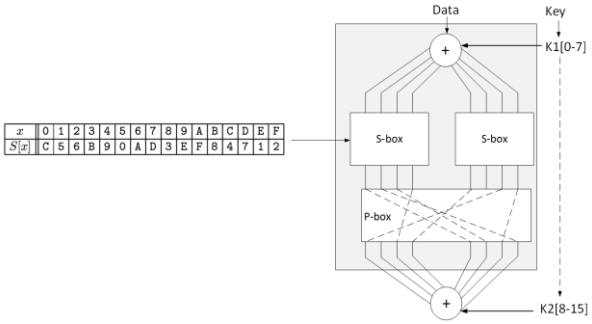

Figure 3 SPN method

An S-box substitutes a small block of bits (the input of the S-box) by another block of bits (the output of the S-box). This substitution should be one-to-one, to ensure invertability (hence decryption). In particular, the length of the output should be the same as the length of the input (the picture on the right has S-boxes with 4 input and 4 output bits), which is different from S-boxes in general that could also change the length, as in DES (Data Encryption Standard), for example. An S-box is usually not simply a permutation of the bits. Rather, a good S-box will have the property that changing one input bit will change about half of the output bits (or an avalanche effect). It will also have the property that each output bit will depend on every input bit [20].

Within PRESENT, we take a block of 64 bits and apply an 80-bit or a 128-bit key. Overall it has 32 rounds (Figure 4), which is made up of: a round key operation; an S-box layer; and a P-box layer. The key round operation takes part of the key and EX-ORs it with the data input into the round. It then operates on 4x4 bit S-boxes, and which considerably cuts down on processing power (Figure 5). In AES we map for 16 bit inputs to 16-bit outputs (0x00 to 0xFF) but for PRESENT we have 4-bit values and which map onto 16 output values (0x0 to 0xF). For example, an input of 0x0 would output a value of 0xC. In Figure 6 we see the permutation of the bits for inputs of 32 bits, so that Bit 1 is mapped to Bit 16. The output from the layer provides the output from the round.

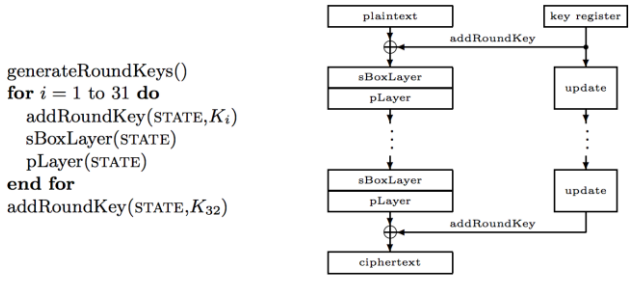

Figure 4 PRESENT method [19]

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S[x]$	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

Figure 5 sboxlayer mapping [19]

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$P(i)$	0	16	32	48	1	17	33	49	2	18	34	50	3	19	35	51
i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$P(i)$	4	20	36	52	5	21	37	53	6	22	38	54	7	23	39	55
i	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
$P(i)$	8	24	40	56	9	25	41	57	10	26	42	58	11	27	43	59
i	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
$P(i)$	12	28	44	60	13	29	45	61	14	30	46	62	15	31	47	63

Figure 6 pLayer mapping [19]

Another contender is the super-fast XTEA method. XTEA (eXtended TEA) is a block cipher which uses a 64-bit block size and a 64-bit key. It was designed by David Wheeler and Roger Needham at the Cambridge Computer Laboratory, and part of an unpublished technical report in 1997. The amazing thing about XTEA is that it does its operations with just a few lines of code [21]:

```
#include <stdint.h>

/* take 64 bits of data in v[0] and v[1] and 128 bits of key[0] - key[3]
*/

void encipher(unsigned int num_rounds, uint32_t v[2], uint32_t const
key[4]) {
    unsigned int i;
    uint32_t v0=v[0], v1=v[1], sum=0, delta=0x9E3779B9;
    for (i=0; i < num_rounds; i++) {
        v0 += (((v1 << 4) ^ (v1 >> 5)) + v1) ^ (sum + key[sum & 3]);
        sum += delta;
        v1 += (((v0 << 4) ^ (v0 >> 5)) + v0) ^ (sum + key[(sum>>11) &
3]);
    }
    v[0]=v0; v[1]=v1;
}
void decipher(unsigned int num_rounds, uint32_t v[2], uint32_t const
key[4]) {
    unsigned int i;
    uint32_t v0=v[0], v1=v[1], delta=0x9E3779B9, sum=delta*num_rounds;
    for (i=0; i < num_rounds; i++) {
        v1 -= (((v0 << 4) ^ (v0 >> 5)) + v0) ^ (sum + key[(sum>>11) &
3]);
        sum -= delta;
        v0 -= (((v1 << 4) ^ (v1 >> 5)) + v1) ^ (sum + key[sum & 3]);
    }
    v[0]=v0; v[1]=v1;
}
```

Other block ciphers, too, are now being called back from retirement, including RC5, as they have proven to be fairly simple in their operation, but relatively secure. The great thing about RC5 is that it has a variable block size (32, 64 or 128 bits), and has key sizes from 0 to 2,040 bits. Along with this, it can have from 0 to 255 rounds. When it was first created, the recommended implementation was a block size of 64 bits, a 128-bit key and 12 rounds, but, in an IoT world, this can be optimised to the device.

For lightweight crypto, the NSA released SIMON in 2013, and which was optimized for hardware implementations. It has key sizes of 64, 72, 96, 128, 144, 192 or 256 bits, and block sizes of 32, 48, 64, 96 or 128 bits [22] and SPECK (which is optimized for software implementations) [23].

Mickey V2 is a lightweight stream cipher and was written by Steve Babbage and Matthew Dodd. It creates a key stream from an 80-bit key and a variable length initialization vector (of up to 80 bits). The keystream has a maximum length of 2^{40} bits [24].

Trivium is a lightweight stream cipher and It was created Christophe De Cannière and Bart Preneel, and has a low footprint for hardware. It uses an 80-bit key, and generates up to 2^{64} bits of output, with an 80-bit IV [25].

Grain is a Light Weight Stream Cipher and was written by Martin Hell, Thomas Johansson and Willi Meier. It has a relatively low gate count, power consumption and memory. It

has an 80-bit key, and has two shift registers and a non-linear output function [26].

Enocoro is a lightweight stream cipher and was defined by Hitachi. It has a 128-bit key and a 64-bit IV value. Along with this it is included in ISO/IEC 29192 International Standard for a lightweight stream cipher method (ISO/IEC 29192-3:2012) [27].

C. Block

CLEFIA is a well-studied lightweight block cipher and was written by Taizo Shirai, Kyoji Shibutani, Toru Akishita, Shiho Moriai, and Tetsu Iwata, and can be implemented with 6K gates. It was defined by Sony, and has 128, 192 and 256 bit keys, and 128-bit block sizes. Along with this it is included in ISO/IEC 29192 International Standard for a lightweight block cipher method (ISO/IEC 29192-2:2012) [28].

RC5 ("Ron's Cipher 5"), created in 1994 by Ron L. Rivest, also shows great potential for a lightweight cryptography method. It is a block cipher which has a variable block size (32, 64 or 128 bits), a variable number of rounds, and a variable key size (0 to 2,048 bits). It can thus be used to match the encryption to the capabilities of the device. If it is a low-powered device with a limited memory and a relatively small physical footprint, we could use a 32-bit block size and an 80-bit key, with just a few rounds. But we can ramp up the security if the device can cope with it, and use 128-bit block sizes and a 128-bit key. It can also be flexible, where a single change on either side can improve or reduce the requirements.

The flexibility around the key size, block size and rounds, supports a range of design choices, in a way that AES struggles with. AES, for example, uses relatively large key sizes of 128 bits, 192 bits and 256-bits, with 128-bit block sizes. It also a fixed number of rounds depending on the key size, such as 10 rounds for 128-bit encryption, 12 rounds for 192-bit encryption, and 14 rounds for 256-bit encryption. These requirements, for an IoT device, often consume considerable amounts of memory and processing resource, and will often have a significant effect on the power consumption, draining the battery resource. The following uses RC5/32/12/16 (32-bit blocks, 12 rounds and 16-byte key - 128 bits): [29].

D. Signing

Chaskey Cipher is a permutation-based lightweight cryptography method for signing messages (MAC) using a 128-bit key. The Chaskey takes a 128-bit block using a 128-bit Addition-Rotation-XOR based permutation. The hardware implementation only requires 3,334.33 gates equivalent with an operating clock frequency of 1 MHz. With SHA-256 we need around 15,000 gates, while Keccak (SHA-3) requires 4,658 gates [30].

```
Message:                hello
Key (128 bits - 32 hex):    BD63710BAF4753D0367DBF6A875ACAAB
Signature:              db6a554716651bc3a818e0c1d01d582d
Encrypt (CBC):          18c381d3811319c24af6cd71af70f97f
```

E. Asymmetric Encryption

Our normal public key methods do not quite work on RFID devices, so let's look at the proposed method for proving that a RFID device is real. The method proposed by the ISO/IEC is ISO/IEC 29192-4:2013 includes ELLI (Elliptic Light), cryptoGPS [31] and ALIKE [32]. ELLI uses Elliptic Curves along with a Diffie-Hellman related handshake between the RFID tag and the RFID reader [33]. Within Elliptic Curve we start with a point on a curve (P) which is known. Then we multiply this point with a large number (ε) to produce another point (A) on the curve:

$$A = \varepsilon P \tag{1}$$

and where A will be the public key, and ε is the private key. If ε is large enough it is then difficult to compute ε even though we have A and P. Now let's look at the basics of ELLI. For this RFID tag contains a random value of ε (the private key), and the RFID reader generates a random value of λ. On creating the RFID tag, we calculate (Figure 7):

$$B = \varepsilon P \tag{2}$$

along with the signature of B which has been signed by a key that the RFID reader can validate. Thus the tag contains: [ε, B, PublicKeySign(B)]. Each time the RFID reader wants to validate the tag it takes its random value (λ) and computes:

$$A = \lambda P \tag{3}$$

Next the RFID reader sends A to the RFID tag. The RFID tag then multiplies the value of A by its private key (ε) to get C:

$$C = \varepsilon A \tag{4}$$

It then sends back its public key (B), the value of C and the signature of the public key which the reader can verify. The reader then computes D:

$$D = \lambda B \tag{5}$$

and compares C and D. If they are the same we have verified the private key. This is true as:

$$C = \varepsilon (A) = \varepsilon (\lambda P) \tag{6}$$
$$D = \lambda (B) = \lambda (\varepsilon P) \tag{7}$$

It is secure as it uses the Elliptic Curve Diffie Hellman Problem (ECDHP). If Eve wants to produce a fake RFID tag she receives the challenge of:

$$A = \lambda P$$

and now must return a valid response (C), along with a public key which has been signed by an authority. Since Eve only has A and B, she cannot compute a valid response for C as she does not know λ and ε, in order to compute [33]:

$$\lambda.\varepsilon.P$$

IV. Conclusion

Lightweight cryptography has received increasing attentions from both academic and industry in the past two decades. A large number of lightweight algorithms have been proposed such as PRESENT, CLEFIA, LED, KANTAN, etc. This paper

reviews the most popular lightweight cryptography solutions over resource limited devices and analyzed the strengths and disadvantages. This paper also gives an overview of the state-of-the-art ultra-lightweight and IoT cryptography that could be used over resource-limited smart devices such as intelligent sensor, RFID, and so on.

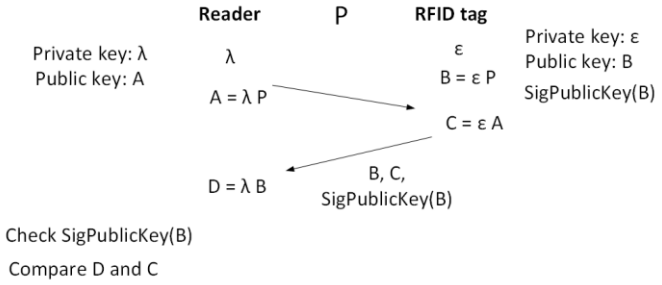

Figure 7 Abstraction of the ELLI method

V. References

[1] K. A. McKay, L. Bassham, M. S. Turan, and N. Mouha, "Report on lightweight cryptography," 2017.

[2] O. Lo, W. J. Buchanan, and D. Carson, "Power analysis attacks on the AES-128 S-box using differential power analysis (DPA) and correlation power analysis (CPA)," *J. Cyber Secur. Technol.*, vol. 1, no. 2, pp. 1–20, 2016.

[3] S. Li, L. Da Xu, and S. Zhao, "The internet of things: a survey," *Inf. Syst. Front.*, vol. 17, no. 2, pp. 243–259, Apr. 2015.

[4] A. Bogdanov, M. Knežević, G. Leander, D. Toz, K. Varici, and I. Verbauwhede, "{SPONGENT}: The Design Space of Lightweight Cryptographic Hashing," 2011.

[5] A. Biryukov and L. Perrin, "State of the Art in Lightweight Symmetric Cryptography."

[6] N. Mouha, "The Design Space of Lightweight Cryptography," *NIST Light. Cryptogr. Work. 2015*, pp. 1–19, 2015.

[7] S. Banik, A. Bogdanov, T. Isobe, K. Shibutani, H. Hiwatari, T. Akishita, and F. Regazzoni, "Midori: A Block Cipher for Low Energy," in *Proceedings, Part II, of the 21st International Conference on Advances in Cryptology --- ASIACRYPT 2015 - Volume 9453*, Springer-Verlag New York, Inc., 2015, pp. 411–436.

[8] R. Avanzi, "The QARMA Block Cipher Family. Almost MDS Matrices Over Rings With Zero Divisors, Nearly Symmetric Even-Mansour Constructions With Non-Involutory Central Rounds, and Search Heuristics for Low-Latency S-Boxes," *IACR Trans. Symmetric Cryptol.*, vol. 2017, no. 1, pp. 4–44, Jan. 2017.

[9] D. Dinu, A. Biryukov, J. Großschädl, D. Khovratovich, Y. Le Corre, and L. Perrin, "FELICS – Fair Evaluation of Lightweight Cryptographic Systems," 2015.

[10] J. Guo, T. Peyrin, and A. Poschmann, "The PHOTON Lightweight Hash Functions Family," *Crypto*, pp. 222–239, 2000.

[11] 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: Lesamnta-LW," Springer, Berlin, Heidelberg, 2011, pp. 151–168.

[12] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "Keccak," Springer, Berlin, Heidelberg, 2013, pp. 313–314.

[13] William J Buchanan, "SPONGENT." [Online]. Available: http://asecuritysite.com/encryption/spongent. [Accessed: 30-Jul-2017].

[14] William J Buchanan, "Lesamnta-LW." [Online]. Available: http://asecuritysite.com/encryption/lw. [Accessed: 30-Jul-2017].

[15] J. P. Aumasson, L. Henzen, W. Meier, and M. Naya-Plasencia, "Quark: A lightweight hash," *J. Cryptol.*, vol. 26, no. 2, pp. 313–339, 2013.

[16] William J Buchanan, "QUARK." [Online]. Available: http://asecuritysite.com/encryption/quark. [Accessed: 30-Jul-2017].

[17] J. Kelsey, S. Change, and R. Perlner, "SHA-3 derived functions: cSHAKE, KMAC, TupleHash and ParallelHash," Gaithersburg, MD, Dec. 2016.

[18] William J Buchanan, "PHOTON." [Online]. Available: http://asecuritysite.com/encryption/photon. [Accessed: 30-Jul-2017].

[19] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe, "PRESENT: An Ultra-Lightweight Block Cipher," *Cryptogr. Hardw. Embed. Syst. - CHES 2007*, pp. 450–466.

[20] William J Buchanan, "PRESENT." [Online]. Available: http://asecuritysite.com/encryption/present. [Accessed: 30-Jul-2017].

[21] William J Buchanan, "XTEA (eXtended TEA)." [Online]. Available: http://asecuritysite.com/encryption/xtea. [Accessed: 30-Jul-2017].

[22] William J Buchanan, "SIMON." [Online]. Available: http://asecuritysite.com/encryption/simon. [Accessed: 30-Jul-2017].

[23] William J Buchanan, "SPECK." [Online]. Available: http://asecuritysite.com/encryption/speck. [Accessed: 30-Jul-2017].

[24] William J Buchanan, "Mickey V2 Light Weight Stream Cipher." [Online]. Available: http://asecuritysite.com/encryption/mickey. [Accessed: 30-Jul-2017].

[25] William J Buchanan, "Trivium Light Weight Stream Cipher." [Online]. Available: http://asecuritysite.com/encryption/trivium. [Accessed: 30-Jul-2017].

[26] William J Buchanan, "Grain Light Weight Stream Cipher." [Online]. Available: http://asecuritysite.com/encryption/grain. [Accessed: 30-Jul-2017].

[27] "ISO/IEC 29192-3:2012 - Information technology -- Security techniques -- Lightweight cryptography -- Part 3: Stream ciphers." [Online]. Available: https://www.iso.org/standard/56426.html. [Accessed: 22-Aug-2017].

[28] William J Buchanan, "CLEFIA." [Online]. Available: https://asecuritysite.com/encryption/clefia. [Accessed: 30-Jul-2017].

[29] William J Buchanan, "RC5." [Online]. Available: https://asecuritysite.com/encryption/rc5. [Accessed: 30-Jul-2017].

[30] William J Buchanan, "Chaskey Cipher." [Online]. Available: http://asecuritysite.com/encryption/chas. [Accessed: 30-Jul-2017].

[31] Q. Dong, W. Ding, and L. Wei, "Improvement and optimized implementation of cryptoGPS protocol for low-cost radio-frequency identification authentication," *Secur. Commun. Networks*, vol. 8, no. 8, pp. 1474–1484, May 2015.

[32] C. Fan, T. Chiang, and R. Hsu, "Light-Weight Authentication and Key Exchange Protocols with Forward Secrecy for Digital Home," *J. Comput.*, vol. 18, no. 2, pp. 61–74, 2007.

[33] M. Braun, E. Hess, and B. Meyer, "Using Elliptic Curves on

RFID Tags," *IJCSNS Int. J. Comput. Sci. Netw. Secur.*, vol. 8, no. 2, 2008.