Choosing the Future of Lightweight Encryption Algorithms

(extended abstract of the MSc dissertation)

João Carlos Santos Fernandes
Departamento de Engenharia Informática
Instituto Superior Técnico

Advisor: Professor Ricardo Chaves
Co-Advisor: Professor Tiago Dias

***Abstract*—Lightweight Cryptography is a field that has been growing significantly in the recent years, mainly because of the explosion of the Internet of Things (IoT). It aims to develop algorithms that can operate under low resources (memory, processing power and energy). This thesis aims to complement the state of the art by analyzing and selecting a set of lightweight ciphers and optimize them, targeting a widely used processors in IoT (ARM Cortex-M3). The analysis considered different metrics like code size, execution time and energy consumption. The selected ciphers were AES, CLEFIA, NOEKEON, PRESENT, RECTANGLE, RoadRunneR, SPARX and SPECK. Their performance was improved using techniques like table-based, bit-slicing and code optimizations (e.g.: rearrange of operations, function inlining, unrolling, etc.). The table-based optimizations were able to speedup AES and CLEFIA execution time more than 10$times$. For NOEKEON, an optimization proposed speed up is performance by 3.2$times$. and reduces his code size in 21%. The proposed optimization of RECTANGLE reduces the execution time by 1% and the code size by 10% when compared to the optimized C version of the cipher's authors. SPARX, execution time was speed up by 2.72$times$ and the code size is 1% lower, in comparison to the optimized C version of the cipher's authors. The SPECK execution time was improved 1.4$times$, with only a 5% increase in code size. Finally, an analysis of the energy consumption of the block ciphers was made, using experimentally obtained results, something that has not yet been done in the state of art.**

***Keywords*-Lightweight cryptography, Block Ciphers, Internet of Things (IoT), ARM Cortex-M3 Processor, Performance Optimization, Energy Consumption**

I. Introduction

The Internet of Things (IoT) [1] refers to the networked interconnection of everyday objects, which are often equipped with ubiquitous intelligence. In the IoT World everything is interconnected and exchanging data. That creates opportunities for a more direct integration of the physical world into computer-based systems, resulting in efficiency improvements, economic benefits, and reduced human effort.

Home appliances, surveillance cameras, traffic lights, vehicles, medical devices, watches, fitness bands and different types of sensors (humidity, temperature, light, noise, etc.), are some of the current examples of devices in the IoT World. Environments like Smart Houses, Smart Cities, Industry 4.0, Smart Cars and Roads or Smart Agriculture are some of the examples where the IoT is being applied.

This is a phenomenon that is growing every year. The number of IoT devices is expected to reach 26 billions by 2020 [2] posing a global market value of \$7.1 trillion [3]. Projects like Raspberry Pi, Arduino and NodeMCU are helping the grow of popularity of the IoT, allowing people to easily develop their own cyber-physical devices.

Most IoT devices operate on limited resources because of their tight cost constraints, inherent to their mass deployments, and the fact that they are mainly battery supplied. Because they do not have an unlimited power source, their energy consumption needs to be small. All of that factors lead to devices with limited memory and computing power. So, the involved data processing algorithms, communication protocols and underlying technologies must be carefully chosen to meet the restrict operating requirements of these devices.

Another critical challenge of the IoT World is Security [4]. IoT devices are in permanent connection to the Internet, and they can deal with a lot of sensitive data and information. Such information is quite often private or safety critical, and therefore must be protected from malicious attackers. So, the application of secure cryptographic components becomes imperative.

However, the well known cryptographic primitives used on traditional systems, are not necessarily well suited for such constrained devices. They require too much memory space and processing power (to have a fast execution), which leads to a high energy consumption.

As a consequence, a new field of research has been created in the Cryptographic World: **Lightweight Cryptography** [5], which comprehends the research and development of new cryptographic primitives and algorithms that can be executed quite fast, show diminished memory footprints and consume very little energy, aiming at achieving a good trade-off between security, cost (in terms of memory footprint and energy consumption), and performance.

In the last years, several articles were published addressing Lightweight Cryptography issues (e.g. [5] [6] [7] [8] [9] [10]), such as the proposal of new lightweight ciphers. Nevertheless, most of such works are focused on hardware implementations [11] [12]. However, with the growing of

IoT and its open source projects, the investigation of software oriented lightweight ciphers suitable for implementations on constrained devices has been gaining more and more importance in the recent years.

A. Objectives

The main goal of this research work is to study existing lightweight encryption algorithms, in order to identify the most suitable ones for software implementations in constrained devices. For this, the most recent and commonly used lightweight algorithms must be evaluated based on their characteristics, like complexity, memory requirements, and offered performance levels. Such assessment encompasses also the port of the selected most relevant algorithms to an Arduino Due Board, that is powered by a well known constrained processor, and its evaluation by taking into account other popular algorithms.

In summary, the main objectives of this work are:

- Select a set of the most promising software oriented lightweight algorithms, proposed by the community, based on their characteristics, implementations and optimizations.
- Provide efficient implementations of such algorithms, in terms of performance, memory footprint and energy consumption.
- Evaluate the developed implementations to find the best algorithm and compare it with the state of the art.

II. Related Work

A. Lightweight Algorithms

In the past years, several new algorithms were presented by taking into account the requirements of lightweight encryption. Also, well known algorithms like AES and DES have been optimized to fit in the lightweight world [13]. Besides these well known and mature ciphers, several new lightweight ciphers have been proposed, such as:

SPN Ciphers: PRESENT[11], BORON[14], GIFT[15], KLEIN[16], NOEKEON[17], RECTANGLE[18], SIMON[19][20], LED[21], Fantomas/Robin[22], Mysterion [23], PRIDE [24], PRINCE [25], RobinStar [23]

FN/GFN Ciphers: HIGHT[26], LBlock[27], LiCi[28], XTEA[29], NUX[30], CLEFIA[31], TWINE[32], RoadRunneR[33], Piccolo[34].

ARX Ciphers: RC6[35], SPECK[19][20], SPARX[36], LEA[37], Chaskey[38].

Hybrid Ciphers: Hummingbird[39], Hummingbird 2[40].

NLFSR Ciphers: Halka[41], KATAN & KTANTAN[42].

Some of these ciphers already have known security issues, as discussed in [13]. For example, KLEIN, HIGHT, LBlock, XTEA, TWINE, RC6, Hummingbird, Hummingbird-2 and KATAN & KTANTAN all suffer from significant security vulnerabilities and should be avoided. BORON, GIFT, NOEKEON, RECTANGLE, RoadRunneR, SIMON and SPECK need to be further analyzed for vulnerabilities to evaluate their claimed level of security. AES, PRESENT, CLEFIA are the most studied ciphers and, therefore, the most acceptable solutions.

B. Metrics and Tools

*1) **Metrics**:* With the increase of importance of lightweight cryptography, several new ciphers have been proposed. As a result, comparative analysis of such ciphers have been gaining more and more importance. However, depending on the target platform, different metrics should be considered.

In what concerns software implementations, several different comparative analysis have been presented in the literature. In [43] the goal was to evaluate the energy consumption of the block cipher in memory constrained devices, for which the considered metrics were the number of clock cycles, the RAM footprint, and the code size. The authors collected the amount of clock cycles for the encryption and decryption procedures, as well as for the key expansion, which allowed them to assess the stages that require more processing time. The energy consumption values were obtained with the simulation of power models for the StrongARM SA-1100 processor.

Other evaluations based on software platforms were also reported in [13] [9] [44] [10]. Although such studies addressed different target platforms, they considered the same set of metrics mentioned before.

When considering evaluations based on hardware platforms [13] [45] [46] [47], the metrics usually employed are Area, which is measured in Gate Equivalents (GE) and Throughput, measured in bytes per second.

*2) **Tools**:* With such a vast number of different metrics, the need for a consistent and simplified evaluation methodology started to grow within the research community. As a result, several tools have been designed for this purpose, have been developed and have become widely used. Two good examples of these tools are BLOC[48] and FELICS[49].

The BLOC project [48] aims at studding the design of block ciphers in constrained environments. The underlying target device is the 16-bit MSP430F1611 microcontroller, which is commonly used in sensor nodes. Three metrics are considered: the execution time, the RAM requirements and the code size. The metric extraction is done automatically through Bash scripts and the results are exported into LaTeX tables. Unfortunately, as mentioned in [49], a bug was found on the source code causing the RAM footprint to be wrongly computed. Overall, the project has the merit of being one of the first attempts to perform automated evaluation a set of lightweight block ciphers on an embedded device.

The FELICS (Fair Evaluation of Lightweight Cryptographic Systems) tool was first introduced in [50] but it was formally presented in [49]. It is a free, open source and flexible framework that can be used to assess the performance of C and assembly software implementations of lightweight primitives (block and stream ciphers) on a set of embedded devices. In fact, this framework has been widely used in the most recent papers presenting analyzes of ciphers, like in [44], [9] and [51].

C. Algorithm Optimizations

The optimization of the ciphers has also been a major concern of lightweight encryption (e.g. [51]). Several different approaches can be used to optimize a cipher, each one targeting a specific goal. Still, most optimizations aim at reducing the execution time (increase on performance), the code size (ROM footprint), or the volatile memory requirements (RAM footprint).

The Algorithm Optimizations involve modifications in the structure of the cipher, which consist in replacing some of the operations by other less complex operations providing the same result. The most commonly adopted techniques are:

- Table based implementations;
- Bit-slice implementations;

The goal of these optimizations is to increase the performance of the cipher.

*1) **Table Based Implementations**:* Tabulating operations for efficiency purposes is quite an old technique that is very well known by programmers. When applied to block ciphers, the goal is to tabulate as much as possible the different operations composing one round. So, with this optimization method, multiple operations will be exchanged for a table lookup, leading to a much higher performance. The result of this implementation will be a round that is composed of:

- the key addition layer (can be performed before or after the table lookups, depending on the cipher structure);
- selection of slices from the cipher state using shift and mask operations;
- perform the round transformation through several table lookups;
- combine the result of the table lookup to obtain the updated cipher state.

This approach is trivial to implement in ciphers that follow an SPN structure but more specifically in ciphers like the AES, where rounds are based in substitution layers (S-Box) and a permutation layer with a multiplier boxes (M-Box). This technique can also be applied to other SPN structures, but it will be harder and might not achieve a much better performance that justifies the trade-off. In [52] the authors of the AES cipher already proposed this table-based implementation to efficiently perform AES operations in 32-bits processors. Also, in [12] a general approach for a table-based implementation is proposed and table implementations for LED[21], PRESENT [11] and Piccolo[34] are presented.

Table Based Reduction. The main problem of this table-based implementations is that they involve a trade-off between memory and performance. The tables are usually big, and they need to be stored in memory or calculated on the fly. If they are calculated on the fly, this will lead to a not so good performance improvement, because of the overhead. Conversely, storing them in memory will require a large amount of memory, which can be critical in constrained devices. For example, the proposed AES table-based implementation requires 8 tables (4 for encryption and 4 for decryption) where each table has an input of 8-bits and an output of 32-bits. This leads to tables of 1 KB, which would result in an increase of 8 KB in the cipher size. Luckily, when the tables are extracted from a SPN structure with a multiplier box (M-Box), a relation between the T-Boxes is noticed, because each row of the M-Box corresponds to a table and usually the rows are rotations of each other. So, this makes possible to obtain the T-Boxes from only one T-Box by using shift operations. These shift operations will add a little overhead but will enable a big reduction in the size of the cipher, which makes it more suitable to be used on constrained devices. With this approach the increase on a cipher like AES is reduced from 8 KB to only 2 KB, since only 1 T-Box is required for encryption and another for decryption. This property of the T-Box implementation is also presented in [52].

Security Issues. A main security issue of the table-based implementations is that they are susceptible to cache timing attacks, which makes this table-based implementations not reliable on processors with cache.

*2) **Bit-slice Implementations**:* The bit-slicing technique was first introduced by Biham in 1997 [53]. It was used to speed up the software performance of DES. The optimization was used for brute force key search of DES in the late-1990s. More recently, it has been used to improve the performance other ciphers. The fastest known software implementation of AES, uses the bit-slicing technique and was implemented by Käsper and Schwabe [54] on an Intel Core 2 utilizing its enhanced SIMD architecture. For PRESENT, a bit-slice implementation on an Intel Core 2 and other x86 processors have been also presented [55] [12].

The basic concept of bit-slicing is to simulate hardware performance in software. To achieve such goal the entire algorithm is represented as a sequence of logical operations. Also, the state of the cipher changes from n-bit words to one-bit words, which allows to compute the operations in parallel to n-bits. So, in a bit-slice implementation one software logical instruction corresponds to the simultaneous execution of n hardware logical gates, where n is the size of a register. In the bit-slice approach, S-boxes are computed using bit-logical instructions rather than table lookups. Since the execution time of these instructions is independent of the input and key values, the bit-slice implementations require less code size and are generally resistant to timing attacks. Hence bit-slicing can be efficient when the entire hardware complexity of a target cipher is small, and the target processor has many long registers. Despite all this, a conversion of the cipher state is required for compatibility with the bit-slicing implementation. This conversion can lead to an overhead in the cipher performance.

Bitslice in the Lightweight World. To enable the deployment of bit-slicing ciphers, in processors with small registers, several ciphers have been proposed in the last years with designs considering bit-slice approaches. Such ciphers do not involve a conversion of the cipher state and focuses on processors with registers of smaller sizes. The application of this technique to lightweight ciphers aims to produce ciphers with small code size, because it does not require any S-Boxes, and that have better performance. Some examples of ciphers that have been designed using the bit-slice implementation are Fantomas/Robin [22], RECTANGLE [18], NOEKEON [17], Mysterion [23] and RoadRunneR[33] (both inspired by Fantomas/Robin).

D. Code Optimizations

Usually when talking about optimizations the ones presented before are the most commons. But other, more simplest, optimizations can be made to ciphers. Despite this optimizations being more simple they still be relevant and could lead to good improvements in performance. For the CLEFIA cipher the authors changed the S-Box size from 4-bits to 8-bits on the software implementation, to achieve better results when performing on software [56]. In [43] ciphers finalist to the AES contest were optimized, with memory footprint in focus, by reducing the code size using functions to replace macros and other code repetitions. The authors of RECTANGLE [18] and SPARX [36] did the opposite of [43]. In order to optimized their ciphers they inlined the functions, or replaced them by macros, to achieve a faster performance by removing the overhead of the calls, these optimized versions are available in the FELICS project [57]. The author of SPARX [36] also used an optimization that tries to reduce the number of memory accesses by keeping the cipher state into different variables and not on a single pointer variable. With that he achieved a faster performance since it reduced the number of writes and reads on the memory, this version is also available in the FELICS project [57]. In the FELICS project, SPECK and SIMON have been optimized by loop unrolling [58]. This means that, some rounds were unrolled to better explore data path of instructions and achieve a better performance.

This type of optimizations involve the implementation of small changes to the algorithm, in order to improve its performance but without changing the structure of the cipher. The most known ones can be divided in different groups:

- Code Cleanup;
- Changing architecture orientation (e.g.: 8-bits to 32-bits);
- Changing the size of the S-Box;
- Constants Calculation vs Constants Tables;
- Function Calls vs Function Inlining;
- Store the Cipher State in Registers;
- Loop unrolling;
- Reordering of the of operations;

Although the main objective of these optimizations is to improve the performance, some of them also allow to achieve a smaller code size or a smaller usage of RAM.

III. Proposed Improvements to Lightweight Encryption

A. Target Platform

To fairly assess the advantages offered by the proposed cipher optimizations for practical IoT applications and products, it is mandatory to conduct a thorough experimental evaluation procedure involving a hardware platform containing a constrained processor widely used in the IoT world. ARM is one of the most popular manufacturers of constrained processors and the dominant player in the IoT world. Furthermore, its 32-bits Cortex-M3 processors have been central to the development of the most recent and cutting-edge IoT products across several different market segments.

The main features that make this a revolutionary processor for the IoT and for the development of embedded applications are its: High Performance, because of its Harvard architecture and Thumb-2 instruction set. It support of bit banding, or multiplications in hardware, which are performed in a single clock cycle. It has a barrel shifter placed before the ALU, which allows to shift register values before an arithmetic or logical operation, with no overhead. And it Low Power Consumption, because of its low number of gate counts and of its sleep modes that enable it to have a power consumption similar to 8-bits and 16-bits processors. Given all these very important characteristics, and also to the fact that the Cortex-M3 processors are supported by the FELICS Framework [49] (which facilitates the evaluation process), the presented research work is focused on the Cortex-M3 processor.

Regarding the hardware platform that was used to conduct the experimental procedures, it consists of the Arduino Due board [59], which is one of the most popular boards of the Arduino Project [60]. This board includes an Atmel SAM3X8E microcontroller that is powered by an ARM Cortex-M3 revision 2.0.

The FELICS tool [49] was used to deploy the considered set of ciphers to this board, as well as to perform its evaluation in terms of execution time (in clock cycles) and code size. Both the standard and the proposed optimized versions of such ciphers were implemented in the C programming language and compiled using the FELICS scripts for the ARM Cortex-M3 processor.

B. Considered Ciphers

Despite the vast number of ciphers that have been proposed in cryptographic literature and also exist in implementations and product worlwide, the study herein presented focus only eight distinct lightweight ciphers. This set of algorithms was selected based on the following criteria. Since, a lot of work has already been done and published in this field, our study mostly addresses ciphers that have not been widely studied, not only to investigate their potential for optimizations but also to provide new contributions to the state of the art. Due to the enormous diversity of applications and constrained devices that exists in the IoT world, we chose to include in our study algorithms with different designs, small block sizes (typically 64 bits, max 128 bits), key sizes between 80 and 128 bits, and a reduced number of rounds, which are all very important features to reduce the power consumption and obtain efficient implementations on software.

Based on these criteria, the chosen ciphers were the following:

AES, PRESENT and CLEFIA: These 3 ciphers were mandatory to be chosen. AES is the most widely used cipher due to having been established as the standard for encryption in 2002, which is why it is also used by many IoT devices, despite the fact that it is not a lightweight cipher. PRESENT and CLEFIA are two ciphers that have also been standardized, but as lightweight ciphers. Therefore, it was mandatory to include these three ciphers in this study so that they can be used as anchors when evaluating the other ciphers.

NOEKEON, RECTANGLE and RoadRunneR: These 3 ciphers share a characteristic: their design meakes use of the bit-slice technique. Bit-slice is showing promising results also in the lightweight world for the following reasons: it leads to a reduced code size, since it does not stores the tables in memory; most operations are performed in place using simple logical operators and rotations for permutations, which makes them very efficient when performing on software; they have a low number of rounds, which can lead to better performances and lower energy consumption.

SPARX: This is one of the few ciphers with an ARX design, that has not yet been broken. It uses ARX-Boxes as S-Boxes, taking the advantages of the ARX design, i.e. light operations, fast speed and small code size. Because of these characteristics, SPARX is a very promising cipher that can achieve a good performance in software implementations. Moreover, it has been poorly considered in the state of the art.

SPECK: This cipher is discussed in almost all comparative analysis that have been published in the literature due to its ultra-lightweight characteristics. It is a very simple and small ARX cipher that can achieve one of the best performances on software, but that has not yet managed to find a place in the lightweight world because of its origin (NASA). Thus, it was chosen because it may be helpful to see how well this cipher performs when optimized.

Table I presents an overview of the cryptographic properties of these algorithms.

C. Optimized Implementations

In order to obtain better implementations of the considered ciphers for the ARM Cortex-M3 processor, both in terms of execution time and memory requirements, the optimization techniques previously described were exploited.

1) AES: Since the version available on the FELICS framework was poorly designed, a new reference implementation, was designed. Such implementation is based on the data presented in [52] and is oriented towards a 32-bits platform. Based on this implementation, a new version was developed by implementing some modifications to the mix columns algorithm of the encryption procedure. That version reuses the value of the GF multiplication to reduce the number of performed multiplications. The other implementations that were developed exploit the use of T-Boxes. First, a standard version was developed, then an improved version was devised using reduced T-Boxes. In this implementation, a cycle was used to pass the four rows of the state through the T-Box algorithm. Two improved versions of this implementation that focus the minimization of the execution time were also developed, by performing the partial and the full unrolling of this loop. Finally, the same modification was applied to these two implementations, in order to reduce the involved memory accesses. This modification consisted in maximizing the use of the processor registers to compute the state operations. In total were developed 8 versions for AES.

2) CLEFIA: The reference implementation was obtained from the CLEFIA website [56] and adapted to the FELICS framewok. Such implementation is 8-bits oriented and performs several memory copies that are totally unnecessary. Therefore, the first implemented optimizations consisted in removing this code and, subsequently, in its adaptation towards a 32-bits architecture. The reference algorithm computes the constants to be used in the key scheduler, which leads to an unnecessarily higher execution time. Therefore, an alternative version of the 32-bits implementation that has the pre-computed values of all the constants stored in a table was devised. The remaining seven implementations that were developed exploit the use of T-Boxes. While the implementations apply standard T-Boxes to the 8-bits oriented reference algorithm and to its optimized 32-bits oriented version, respectively, all the other implementations exploit the use of reduced T-Boxes. In four of these reduced T-Boxes implementations, the F0 and F1 functions were also converted to inline-functions, in order to reduce the execution time resulting from the overhead imposed by the calls to these functions that were performed in each round of the algorithm. For such implementations, two different optimization techniques were further applied: full loop unrolling, to remove all the dependencies and push the cipher to its best execution time; and the maximization of the use of the processors registers, in order to reduce the memory accesses. In total were developed 11 versions for CLEFIA.

3) NOEKEON: The reference implementation was also obtained from the NOEKEON web site [61] and was adapted to the FELICS framework. Such implementation was already 32-bits oriented but was targetted for systems with a big-endian memory organization scheme. That is not the default case if the ARM Cortex-M3 architecture. Therefore, a little-endian version was implemented in order to reduce the overhead and improve the performance in the target platform. Since the only difference between the Direct and the Indirect modes was the key scheduler, most of the implemented versions use the Direct mode. In the reference implementation, NOEKEON 2 different round constants are calculated on the fly. In order to reduce the overhead due to these computations, alternative implementations using tables to store the pre-computed values of these constants were devised. Due to the way the reference code was organized, more than four function calls are performed in each cycle

	Target	Structure	Block Size	Key Size	Rounds
AES	Hardware/Software	SPN	128	128 192 256	10 12 14
CLEFIA	Hardware/Software	GFN	128	128 192 256	18 22 26
NOEKEON	Hardware/Software	SPN	128	128	16
PRESENT	Hardware	SPN	64	80 128	31
RECTANGLE	Hardware/Software	SPN	64	80 128	25
RoadRunneR	Software	FN	64	80 128	10 12
SPARX	Hardware/Software	ARX	64 128	128 256	24 32 40
SPECK	Software	ARX	32 48 64 96 128	64 72 96 128 144 192 256	22 23 26 27 28 29 32 33 34

Table I
BLOCK CIPHERS CHARACTERISTICS

implementing a round. So, to reduce the execution time resulting from the overhead imposed by the calls to these functions, they were converted to inline-functions. To further improve the execution time, another two optimization techniques were implemented: loop unrolling and keeping the cipher state within the processor's registers most of the time. In total were developed 12 versions for NOEKEON.

4) PRESENT: Although FELICS has a an implementation of PRESENT, it has been poorly designed. Therefore, a 32-bits oriented reference implementaion was designed base on the information provided in [11]. This implementation was subsequently optimized using three different techniques. First, the 4-bits S-Boxes were replaced by 8-bits S-Boxes to improve the software performance. Then, all the permutations were unrolled, due to their excessive cost for software implementations. Moreover, the loops were also fully unrolled to remove all the dependencies and push the cipher code to the fastest performance level. Finally, memory accesses were minimized by keeping the cipher state within the processor's registers for most of the cipher's execution time. In total were developed 7 versions for PRESENT.

5) RECTANGLE: The considered reference implementation for RECTANGLE was also available on FELICS. Since RECTANGLE is a 16-bits oriented bit-slice cipher, the first implemented optimization consisted in the adaptation of the code towards a 32-bits architecture. Nevertheless, not all the code could be optimized for a 32-bits processor due to the intrinsic 16-bits nature of this cipher. So, in the devised 32-bits partially oriented implementations, only the code sections involving 32-bits variables were optimized. The RECTANGLE reference implementation has a lot of functions strictly for code organization purposes, none of such functions are reused in the cipher computation. As a result, these functions were converted to inline-functions, in order to reduce the execution time resulting from the overhead imposed by the calls to such functions. RECTANGLE is a bit-slice cipher, thus several operations are performed involving the cipher state. Consequently, a couple of modifications to the code were also implemented, so that the cipher state can be kept in the processor's register for most of the computation time. In addition the loops were also fully unrolled to remove all the dependencies and push the cipher code to the fastest performance level. In total were developed 5 versions for RECTANGLE.

6) RoadRunneR: The reference implementation of the RoadRunneR cipher was available on FELICS has two versions: one with two distinct key schedulers, for the encryption and the decryption, procedures, and another one without a key scheduler that presents only a small overhead due to the key being managed inside the cipher encryption and decryption tasks. So, the first optimization that was implemented consisted in the design of a single key scheduler for both the encryption and the decryption procedures. From that version, an implementation without a key scheduler was also devised. Then, the reference implementation was optimized for the target platform. However, since RoadRunneR is an 8-bits oriented bit-slice cipher, it was only possible to develop a 32-bits partially oriented implementation, likewise for the RECTANGLE implementations. To reduce the execution time, three different optimizations were further implemented. Firstly, all the functions that are called in the loop implementing the rounds were converted to inline-functions. Secondly, the code was modified to keep the cipher state in the processor"s registers, and thus reduce the number of memory accesses. Lastly, the loops were fully unrolled not only to remove all the dependencies and the branches but also to minimize the overhead resulting from the computation of the iteration count.In total were developed 12 versions for RoadRunneR.

7) SPARX: The SPARX cipher was proposed by the same author of FELICS, so its reference implementation was already available in the platform. However, such implementation is not fully oriented to 32-bits. Consequently, the first optimization made to the SPARX source code consisted in adapting it for the 32-bits, architecture of the target processor. SPARX resorts on the Speckey-Box to perform the substitution layer of the cipher, which is implemented as a function, in the reference code for code reuse purposes. Therefore, a version that receives the arguments by value and outputs the return value, was created. Also, another version of this function was implemented, in which pointers are used to pass the arguments to the function and return the result of the Speckey transformation. The Speckey function is also called a lot of times. So, this function was converted to an inline-function to reduce the overhead of its function calls. Likewise the other ciphers, the full unrolling of the loops corresponding to the cipher rounds was another optimization that was implemented to reduce the execution time. The SPARX reference implementation already tries to maximize the use of the processor registers, so there was not much work to do in this field. Still, the source code was modified to include the *register* keyword wherever it was found to be necessary, in order to guarantee that the compiler kept the variables in the registers, as much as possible. The last optimization technique that was implemented aimed at reducing the code size and consisted in reorganizing the steps of the cipher into cycles, since in the considered reference implementation of SPARX such steps have been unrolled.

8) SPECK: Although some implementations of the SPECK cipher are already available in the FELICS platform, a reference implementation was designed based on the algorithm presented in [19][20]. SPECK is a very simple cipher that can not be greatly optimized, because this cipher has no functions, nor tables or constants, therefore, the only possible optimizations were unrolling the rounds and changing the code to maximize the use of the registers reducing the memory accesses. The other modification to the source code that was implemented focused a more efficient implementation of the decryption rounds. Such change consisted in the rearrangement of the instructions to make a better use of the positioning of the barrel shifter in the processor's datapath (i.e. before the ALU). It should be noted that this type of optimization is not required for the SPECK encryption rounds, since the algorithm already makes the best use of the barrel shifter. In total were developed 5 versions for SPECK.

IV. Evaluation

For the evaluation process the the FELICS framework was used. The ciphers were executed in the scenario 0, a scenario that performs the encryption key scheduling, data encryption, and performs the decryption key schedule (if needed), followed by the decryption of the test vectors. The considered results are focused on the Execution Time (Clock Cycles) and Code Size (ROM). For the Energy Consumption evaluation the Arduino IDE [62] was the tool used for deployment.

This section presents comparative analysis between the best obtained results, for different trade-offs, namely the smallest, the more balanced and the fastest optimized versions of each cipher. The ciphers reference implementations and state of art implementations [57] are also compared with that results.

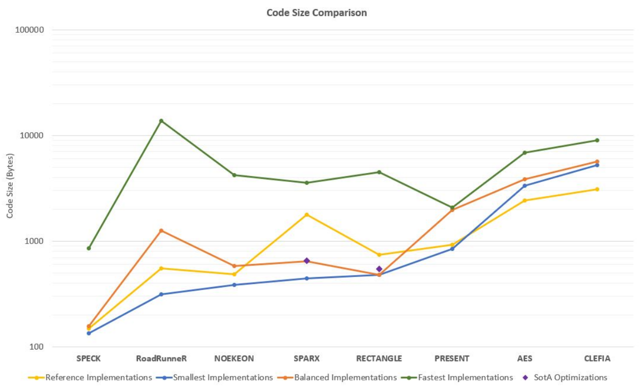

Figure 1. Code Size Results of the different implementations

Figure 1 depicts the code size results for all the ciphers. It is clear from the results that AES and CLEFIA are consistently the largest ciphers, for all the different implementations. The proposed implementations are even bigger that the reference implementations, given the use of T-Box, which significantly increases the code size. Nevertheless, the proposed reduced T-Box solutions allow to reduce that code size cost. The fastest implementations are clearly the ones with bigger code size. This was expected since they mostly use the full unroll or funtion inlining to achieve better execution times, which are optimizations that increase the code size. As expected the smallest implementations have the lowest code size among all the implementations. Also, the majority of the ciphers has a size lower than 500 bytes, which is very good. Another positive point is that some of the balanced implementations are close to the code size of the small implementations, which is very promising since their execution time shows very good results. SPECK is the smallest cipher in all implementations, as expected, followed by RoadRunneR, SPARX and NOEKEON.

Figure 2 depicts the execution time results for all the proposed ciphers implementations, when operating on a 128-bits of data. As expected the reference implementations show

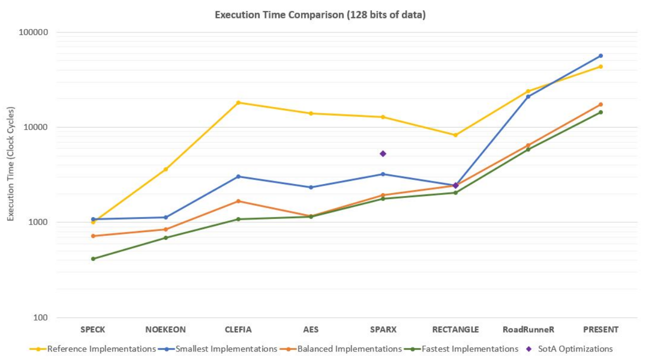

Figure 2. Execution Time Results of the different implementations

the slowest results, except for PRESENT where the slowest execution time is for the small code size implementation. This shows that most of the optimzations used allowed for improvements in terms of execution time, even when the focus is reducing the code size. While the fastest implementations show the best execution time results, they are followed very closely by the balanced implementations, which have significant smaller code sizes. An interesting thing that can be noticed is that for SPARX all the proposed versions show not only better execution time, than the reference implementation, but are also better that the optimized version available in the state of art. The ciphers with bigger improvements on the execution time are clearly the AES and CLEFIA, given the T-Box Reduced optimization. Another good result of the proposed implementations is that for the balanced ones, the execution time is close to 1000 clock cycles for almost every cipher, just RoadRunneR and PRESENT which are far from that value. PRESENT has the bit-oriented permutations that is heavy to compute in software. RoadRunneR is an 8-bits oriented bit-slice cipher which makes it not so interesting for 32-bits devices. On the other hand, SPECK is clearly the fastest cipher, which was expected for the simple and small cipher that it is. It is followed by NOEKEON which had a small code size. That good results are achieved because these ciphers are mainly performing arithmetic and logical operations, not performing other type of more costly operations, like memory accesses, since SPECK and NOEKEON is a bit-slice cipher. AES and CLEFIA have smaller code size that SPARX and RECTANGLE in the balanced implementations, because while AES and CLEFIA support blocks of 128 bits, SPARX and RECTANGLE only support 64-bits so they need to perform the encryption 2 times to achieve the same amount of data.

Table II summarizes the results obtained for the balanced implementations proposed with this work. It shows the improvements in terms of performance for each cipher and the cost that it had in code size. For some ciphers the code size increased in order to reduce the execution time (AES, CLEFIA, PRESENT, RoadRunneR and SPECK), but for others the performance improved and the code size also reduced (NOEKEON, RECTANGLE, SPARX).

Figure 3 presents an overview of the the proposed balanced implementations, and several ciphers from the state of art. These results were obtained using the best implementations available from the FELICS Project [57], on the ARM Cortex-M3. Some of them are the highest scored implementations in the Triathlon Competition [50]. The results for this competition can be seen in the Triathlon Webpage [63]. The ciphers are ordered by their throughput (MB/s). From this results it is easy to conclude, which are the fastest ciphers when encrypting/decrypting data. Among the proposed optimizations SPECK is clearly the fastest one, followed by NOEKEON. AES and CLEFIA T-Box implementations are the ones that follow, followed by SPARX and RECTANGLE. The slowest ones are RoadRunneR and PRESENT. RoadRunneR big disadvantage is the fact that is an 8-bit oriented bit-slice cipher.

It this figure is possible to recognize that the throughput of the proposed optimized implementations are among the best ones, only Chaskey, LEA and SIMON can match or challenge the performance of the proposed implementations. That is because these are ARX based ciphers, thus, like SPECK, they have very simple operations and are very small, which may compromise there security. In the other hand, the code size of the proposed implementations is very similar with the other ciphers, depending on the algorithm design.

A. Energy Consumption

In the state of art no experimental measurements of energy consumption was found for software implementation of lightweight ciphers. Therefore, in this work an experimental approach to evaluate the energy consumption is presented. For that was used an oscilloscope (Picoscope [64]) connected to a computer to measure the tension variations, caused by the processor, when running the implemented ciphers.

The goal was to detect the small fluctuations of voltage (equation 1) and thus in the consumed power energy. Since the resistor is fixed the current can be obtained by equation 2 and consequently allowing to compute the power consumption using equation 3. The consumed energy by the processor, in a given time interval, can be obtained by integrating over time the power value (equation 4). Finally, the average energy consumed by the system, can be obtained by dividing the times the cipher takes to compute a given data set. To obtain the energy consumption per data block equation 5 can be used.

$$\Delta V(t) = V_B(t) - V_D(t); \tag{1}$$

$$I(t) = \frac{\Delta V(t)}{R} \tag{2}$$

$$P(t) = V_D(t) \times I(t) \tag{3}$$

$$E_{total} = \int P(t)dt \tag{4}$$

Ciphers	Code Size Difference	Code Size	Speedup	Optimizations
AES	+68%	3852 bytes	12× Faster	• T-Box Reduced • Permutations Unrolling • State in Registers
CLEFIA	+189%	5684 bytes	11× Faster	• T-Box Reduced • Constants Stored • Functions Inlined • State in Registers
NOEKEON	+19%	580 bytes	4.3× Faster	• Constants Stored • Functions Inlined • State in Registers
NOEKEON (Small)	-21%	385 bytes	3.2× Faster	• Single Function • State in Registers
PRESENT	+112%	1968 bytes	2.5× Faster	• 8-bits S-Box • Unrolling Permutations
RECTANGLE	-36%[1]	476 bytes	3.2×[2] Faster	• 32-bits orientation • Functions Inlined • State in Registers
RoadRunneR	+129%	1264 bytes	3.7× Faster	• No Key Schedule • 32-bits orientation • Functions Inlined • State in Registers
SPARX	-64%[3]	644 bytes	6.6×[4] Faster	• 32-bits orientation • Speckey Function Inlined • State in Registers
SPECK	+5%	156 bytes	1.4× Faster	• State in Registers • Decryption Optimization[5]

Table II
SUMMARY OF BEST PROPOSED IMPLEMENTATIONS RESULTS

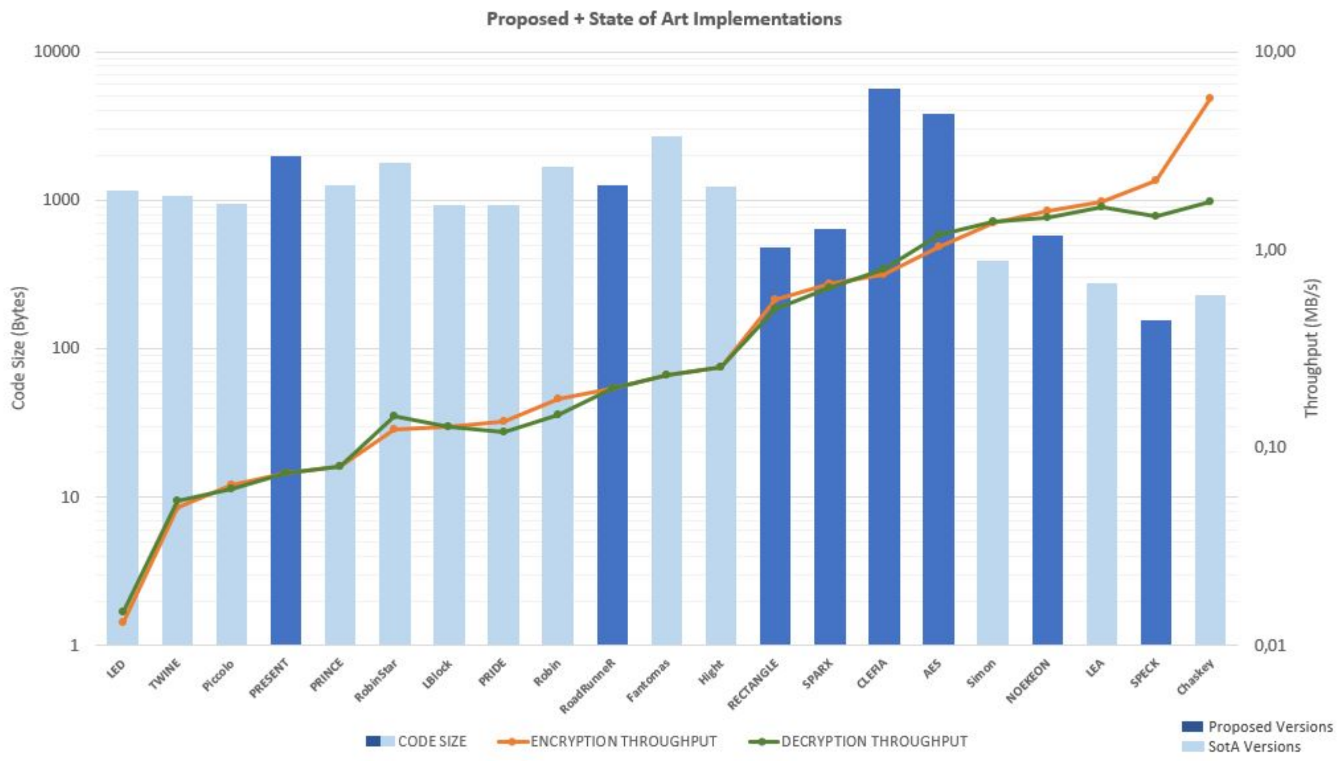

Figure 3. Proposed Implementations vs State Of Art Results

$$E_{per\ block} = \frac{E_{total}}{Number\ of\ blocks} \tag{5}$$

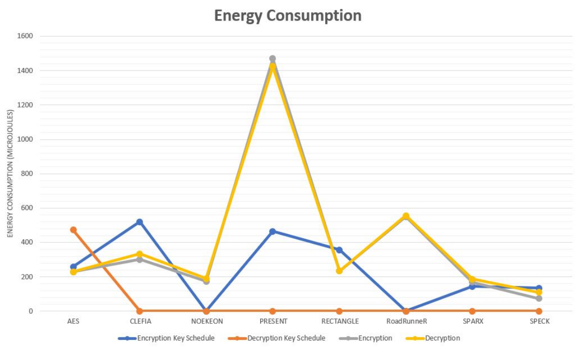

Figure 4. Energy Consumption per Block/Key Schedule (µJ)

Figure 4 shows the obtained results for the balanced implementations of the ciphers, presented before. The measures obtained allowed to condlude that for the same time interval, the energy spent by the processor is very similar, so, the execution time will be the factor that mostly impacts the energy consumption. This means that ciphers that process more in the same interval of time, will be the ones that use less energy per computation (block encryption/decryption, and calculation of round keys). As expected the results shows that the fastest ciphers are the ones with smaller energy consumption.

V. Conclusions

With the IoT paradigm growing every year and the usage of constrained devices increasing the need for lightweight encryption and efficient implementation becomes clear being a hot topic of research with a lot of work done in the past years. This work tales into account the fact that the current standard for lightweight encryption clearly does not fit well on software, demanding for a new standard on lightweight encryption, that must consider software implementations.

Given this, the work here presented focuses on analyzing, improving the implementations and in evaluating 7 different lightweight block ciphers plus AES, to find good alternatives for lightweight encryption. These ciphers were carefully selected given their characteristics, The proposed optimizations were applied based on what the ciphers needed and in their structure. Each cipher had different levels of optimizations, considering which optimizations fitted well in the cipher structure and code. With that, different levels of optimization were introduced to achieve different implementations of each cipher, focusing on different requirements. Some implementations focus on reducing the code size, while others focus on reducing the execution time, with the respective trade-offs. AES and CLEFIA are the two clear examples of ciphers that spend a lot of resources on code size in order to achieve a faster performance. But they also take advantage from the ARM Cortex-M3 characteristic, which allows to shift a register before an arithmetic or logical operation, with no overhead, allowing for a reduction on the code size. SPECK also takes advantage of that feature to increase performance of the encryption and decryption when the rounds are properly rolled. Bit-slice ciphers became a clear good alternatives with NOEKEON being one of the fastest lightweight ciphers yet presented with a small code size. RECTANGLE also achieved good results on a 32-bits platform. RoadRunneR was a little below expectations, given its 8-bits orientation. Despite all this, all bit-slice ciphers had very small code sizes. SPARX also showed very good and interesting results for an ARX-based SPN cipher, that combines the security of SPN ciphers with the light performance of ARX ciphers. PRESENT, even with several optimizations, is still a very heavy cipher on software. The main conclusions from this analyses are:

- The current standard lightweight encryption algorithms do not show very good results on software when compared with new lightweight algorithms.
- PRESENT is an unfriendly lightweight cipher when targeting software implementations.
- AES and CLEFIA with the proposed T-Box Reduced optimization showed very good performances, but with a big trade-off on code size, being only viable for devices that have some memory to spend.
- The feature of shifting without generating overhead of the ARM architecture processors is very useful for some ciphers, particularly, AES and CLEFIA T-Box, and SPECK.
- Bit-slice ciphers show very good results, in both performance and code size, and could be the best way to go, on future standards, for the lightweight world.
- Among the ciphers studied, NOEKEON is the cipher with best memory/execution time performance apart from SPECK (created by NASA, not being very well seen by the community).

The experimental energy consumption evaluation is other point novel of this work. This is something that has not been made in the state of art, adding value to this work. The results were not ideal, given the overall processor power consumption. These results suggests that the energy consumption is mostly imposed by the time spent computing.

When comparing the results with the objectives outlined in the beginning of this thesis, it becomes clear that most of them were achieved. Several different ciphers were studied and different optimized versions proposed for each one. Most of them with better performances and code sizes than the optimizations proposed by the authors. The implementations showed very good results when compared with other ciphers, on the state of art for the target processor. Good alternatives for lightweight encryption were also found. The only drawback of this work is the RAM consumption measurement that was not possible to be performed, because the J-Link Debugger needed for that was to expensive to buy.

Overall, a though evaluation of existing lightweight cipher as been presented while several improved implementations have also been proposed.

A. Future Work

Future work can focus on the improvement of the energy consumption evaluation of the ciphers, by considering the conclusions obtained in this work and complement them, and understanding how the different instructions affect the energy consume of the processor. This is particularly relevant given energy is one of the main constrains in the IoT world. Also, an evaluation on other platforms and devices would be very interesting, considering processors with cache, architectures of 8-bits and 16-bits, in order to find a cipher that could perform well in different devices.

References

[1] F. Xia, L. T. Yang, L. Wang, and A. Vinel, "Internet of things," *International Journal of Communication Systems*, vol. 25, no. 9, pp. 1101–1102, 2012.

[2] "Gartner says the internet of things installed base will grow to 26 billion units by 2020," 2018, https://www.gartner.com/newsroom/id/2636073, *Accessed: 19/07/2018*.

[3] L. Spencer, "Internet of things market to hit $7.1 trillion by 2020: Idc," 2018, https://www.zdnet.com/article/internet-of-things-market-to-hit-7-1-trillion-by-2020-idc/, *Accessed: 19/07/2018*.

[4] T. Xu, J. B. Wendt, and M. Potkonjak, "Security of iot systems: Design challenges and opportunities," in *Proceedings of the 2014 IEEE/ACM International Conference on Computer-Aided Design*. IEEE Press, 2014, pp. 417–423.

[5] S. Panasenko and S. Smagin, "Lightweight cryptography: Underlying principles and approaches," *International Journal of Computer Theory and Engineering*, vol. 3, no. 4, p. 516, 2011.

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

[7] A. Moradi and A. Poschmann, "Lightweight cryptography and dpa countermeasures: A survey." in *Financial Cryptography Workshops*. Springer, 2010, pp. 68–79.

[8] C. Manifavas, G. Hatzivasilis, K. Fysarakis, and K. Rantos, "Lightweight cryptography for embedded systems–a comparative analysis," in *Data Privacy Management and Autonomous Spontaneous Security*. Springer, 2014, pp. 333–349.

[9] W. J. Okello, Q. Liu, F. A. Siddiqui, and C. Zhang, "A survey of the current state of lightweight cryptography for the internet of things," in *Computer, Information and Telecommunication Systems (CITS), 2017 International Conference on*. IEEE, 2017, pp. 292–296.

[10] J. Hosseinzadeh and A. G. Bafghi, "Software implementation and evaluation of lightweight symmetric block ciphers of the energy perspectives and memory," *arXiv preprint arXiv:1706.03909*, 2017.

[11] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. Robshaw, Y. Seurin, and C. Vikkelsoe, "Present: An ultra-lightweight block cipher," in *CHES*, vol. 4727. Springer, 2007, pp. 450–466.

[12] R. Benadjila, J. Guo, V. Lomné, and T. Peyrin, "Implementing lightweight block ciphers on x86 architectures," in *International Conference on Selected Areas in Cryptography*. Springer, 2013, pp. 324–351.

[13] G. Hatzivasilis, K. Fysarakis, I. Papaefstathiou, and C. Manifavas, "A review of lightweight block ciphers," *Journal of Cryptographic Engineering*, pp. 1–44, 2017.

[14] G. Bansod, N. Pisharoty, and A. Patil, "Boron: an ultra-lightweight and low power encryption design for pervasive computing," *Frontiers of Information Technology & Electronic Engineering*, vol. 18, no. 3, pp. 317–331, 2017.

[15] S. Banik, S. K. Pandey, T. Peyrin, Y. Sasaki, S. M. Sim, and Y. Todo, "Gift: a small present," in *International Conference on Cryptographic Hardware and Embedded Systems*. Springer, 2017, pp. 321–345.

[16] Z. Gong, S. Nikova, and Y. W. Law, "Klein: A new family of lightweight block ciphers." *RFIDSec*, vol. 7055, pp. 1–18, 2011.

[17] J. Daemen, M. Peeters, G. Van Assche, and V. Rijmen, "Nessie proposal: Noekeon," in *First Open NESSIE Workshop*, 2000, pp. 213–230.

[18] W. Zhang, Z. Bao, D. Lin, V. Rijmen, B. Yang, and I. Verbauwhede, "Rectangle: a bit-slice lightweight block cipher suitable for multiple platforms," *Science China Information Sciences*, 2015.

[19] R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers, "The simon and speck families of lightweight block ciphers," Cryptology ePrint Archive, Report 2013/404, 2013, https://eprint.iacr.org/2013/404.

[20] R. Beaulieu, S. Treatman-Clark, D. Shors, B. Weeks, J. Smith, and L. Wingers, "The simon and speck lightweight block ciphers," in *Design Automation Conference (DAC), 2015 52nd ACM/EDAC/IEEE*. IEEE, 2015, pp. 1–6.

[21] J. Guo, T. Peyrin, A. Poschmann, and M. Robshaw, "The led block cipher," in *Cryptographic Hardware and Embedded Systems – CHES 2011*, B. Preneel and T. Takagi, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 326–341.

[22] V. Grosso, G. Leurent, F.-X. Standaert, and K. Varıcı, "Ls-designs: Bitslice encryption for efficient masked software implementations," in *International Workshop on Fast Software Encryption*. Springer, 2014, pp. 18–37.

[23] A. Journault, F.-X. Standaert, and K. Varici, "Improving the security and efficiency of block ciphers based on ls-designs," *Designs, Codes and Cryptography*, vol. 82, no. 1-2, pp. 495–509, 2017.

[24] M. R. Albrecht, B. Driessen, E. B. Kavun, G. Leander, C. Paar, and T. Yalçın, "Block ciphers–focus on the linear layer (feat. pride)," in *International Cryptology Conference*. Springer, 2014, pp. 57–76.

[25] J. Borghoff, A. Canteaut, T. Güneysu, E. B. Kavun, M. Knezevic, L. R. Knudsen, G. Leander, V. Nikov, C. Paar, C. Rechberger *et al.*, "Prince–a low-latency block cipher for pervasive computing applications," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2012, pp. 208–225.

[26] D. Hong, J. Sung, S. Hong, J. Lim, S. Lee, B. Koo, C. Lee, D. Chang, J. Lee, K. Jeong *et al.*, "Hight: A new block cipher suitable for low-resource device," in *CHES*, vol. 4249. Springer, 2006, pp. 46–59.

[27] W. Wu and L. Zhang, "Lblock: a lightweight block cipher," in *Applied Cryptography and Network Security*. Springer, 2011, pp. 327–344.

[28] J. Patil, G. Bansod, and K. S. Kant, "Lici: A new ultra-lightweight block cipher," in *Emerging Trends & Innovation in ICT (ICEI), 2017 International Conference on*. IEEE, 2017, pp. 40–45.

[29] S. Kotel, M. Zeghid, M. Machhout, and R. Tourki, "Lightweight encryption algorithm based on modified xtea for low-resource embedded devices," in *Proceedings of the 21st International Database Engineering & Applications Symposium*. ACM, 2017, pp. 192–199.

[30] G. BANSOD, S. SUTAR, A. PATIL, and J. PATIL, "Nux: A new lightweight block cipher for security at wireless sensor node level."

[31] T. Shirai, K. Shibutani, T. Akishita, S. Moriai, and T. Iwata, "The 128-bit blockcipher clefia," in *FSE*, vol. 4593. Springer, 2007, pp. 181–195.

[32] T. Suzaki, K. Minematsu, S. Morioka, and E. Kobayashi, "Twine: A lightweight block cipher for multiple platforms." in *Selected Areas in Cryptography*, vol. 7707. Springer, 2012, pp. 339–354.

[33] A. Baysal and S. Şahin, "Roadrunner: A small and fast bitslice block cipher for low cost 8-bit processors," in *International Workshop on Lightweight Cryptography for Security and Privacy*. Springer, 2015, pp. 58–76.

[34] K. Shibutani, T. Isobe, H. Hiwatari, A. Mitsuda, T. Akishita, and T. Shirai, "Piccolo: an ultra-lightweight blockcipher," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2011, pp. 342–357.

[35] R. L. Rivest, M. Robshaw, R. Sidney, and Y. L. Yin, "The rc6tm block cipher," in *First Advanced Encryption Standard (AES) Conference*, 1998, p. 16.

[36] D. Dinu, L. Perrin, A. Udovenko, V. Velichkov, J. Großschädl, and A. Biryukov, "Sparx: A family of arx-based lightweight block ciphers provably secure against linear and differential attacks."

[37] D. Hong, J.-K. Lee, D.-C. Kim, D. Kwon, K. H. Ryu, and D.-G. Lee, "Lea: A 128-bit block cipher for fast encryption on common processors," in *International Workshop on Information Security Applications*. Springer, 2013, pp. 3–27.

[38] N. Mouha, B. Mennink, A. Van Herrewege, D. Watanabe, B. Preneel, and I. Verbauwhede, "Chaskey: an efficient mac algorithm for 32-bit microcontrollers," in *International Workshop on Selected Areas in Cryptography*. Springer, 2014, pp. 306–323.

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

[40] D. W. Engels, M.-J. O. Saarinen, P. Schweitzer, and E. M. Smith, "The hummingbird-2 lightweight authenticated encryption algorithm." *RFIDSec*, vol. 11, pp. 19–31, 2011.

[41] S. Das, "Halka: A lightweight, software friendly block cipher using ultra-lightweight 8-bit s-box." *IACR Cryptology ePrint Archive*, vol. 2014, p. 110, 2014.

[42] C. De Canniere, O. Dunkelman, and M. Knežević, "Katan and ktantan—a family of small and efficient hardware-oriented block ciphers," in *Cryptographic Hardware and Embedded Systems-CHES 2009*. Springer, 2009, pp. 272–288.

[43] J. Großschädl, S. Tillich, C. Rechberger, M. Hofmann, and M. Medwed, "Energy evaluation of software implementations of block ciphers under memory constraints," in *Proceedings of the conference on Design, automation and test in Europe*. EDA Consortium, 2007, pp. 1110–1115.

[44] S. KOTEL, F. SBIAA, M. ZEGHID, M. MACHHOUT, A. BAGANNE, and R. TOURKI, "Performance evaluation and design considerations of lightweight block cipher for low-cost embedded devices."

[45] T. Eisenbarth and S. Kumar, "A survey of lightweight-cryptography implementations," *IEEE Design & Test of Computers*, vol. 24, no. 6, 2007.

[46] M. Appel, A. Bossert, S. Cooper, T. Kußmaul, J. Löffler, C. Pauer, and A. Wiesmaier, "Block ciphers for the iot–simon, speck, katan, led, tea, present, and sea compared."

[47] S. Kerckhof, F. Durvaux, C. Hocquet, D. Bol, and F.-X. Standaert, "Towards green cryptography: a comparison of lightweight ciphers from the energy viewpoint," *Cryptographic Hardware and Embedded Systems–CHES 2012*, pp. 390–407, 2012.

[48] M. Cazorla, S. Gourgeon, K. Marquet, and M. Minier, "Implementations of lightweight block ciphers on a wsn430 sensor," 2015, http://bloc.project.citi-lab.fr/library.html.

[49] D. Dinu, A. Biryukov, J. Großschädl, D. Khovratovich, Y. Corre, and L. Perrin, "Felics–fair evaluation of lightweight cryptographic systems," in *NIST Workshop on Lightweight Cryptography*, vol. 128, 2015.

[50] D. Dinu, Y. L. Corre, D. Khovratovich, L. Perrin, J. Großschädl, and A. Biryukov, "Triathlon of lightweight block ciphers for the internet of things," IACR ePrint archive, Tech. Rep., 2015.

[51] R. J. Cruz, T. B. Reis, D. F. Aranha, J. López, and H. K. Patil, "Lightweight cryptography on arm."

[52] J. Daemen and V. Rijmen, *The design of Rijndael: AES-the advanced encryption standard*. Springer Science & Business Media, 2013.

[53] E. Biham, "A fast new des implementation in software," in *International Workshop on Fast Software Encryption*. Springer, 1997, pp. 260–272.

[54] E. Käsper and P. Schwabe, "Faster and timing-attack resistant aes-gcm," in *Cryptographic Hardware and Embedded Systems-CHES 2009*. Springer, 2009, pp. 1–17.

[55] S. Matsuda and S. Moriai, "Lightweight cryptography for the cloud: exploit the power of bitslice implementation," *Cryptographic Hardware and Embedded Systems–CHES 2012*, pp. 408–425, 2012.

[56] S. Corporation, "Clefia website," https://www.sony.net/Products/cryptography/clefia/, Accessed: 25/07/2018.

[57] CryptoLUX, "Cryptolux ¿ felics," 2017, https://www.cryptolux.org/index.php/FELICS.

[58] P. M. Knijnenburg, T. Kisuki, and M. F. O'Boyle, "Combined selection of tile sizes and unroll factors using iterative compilation," *The Journal of Supercomputing*, vol. 24, no. 1, pp. 43–67, 2003.

[59] "Arduino due board," 2018, https://store.arduino.cc/arduino-due, *Accessed: 01/08/2018*.

[60] "Arduino," 2018, https://www.arduino.cc/, *Accessed: 19/07/2018*.

[61] "Noekeon website," 2018, https://http://gro.noekeon.org/, Accessed: 20/08/2018.

[62] "Arduino ide," 2018, https://www.arduino.cc/en/Main/Software, *Accessed: 10/10/2018*.

[63] CryptoLUX, "Cryptolux ¿ felics triathlon," 2017, https://www.cryptolux.org/index.php/FELICS_Triathlon.

[64] "Picoscope," 2018, https://www.picotech.com/oscilloscope/9300/picoscope-9300-sampling-oscilloscopes, *Accessed: 10/10/2018*.