Lightweight Crypto

Alejandro Capella Del Solar and Cristian Fernández Jiménez

Universitat Politècnica de Catalunya, Barcelona, Spain

January 8, 2023

1 Introduction

While the most common cryptosystems require heavy computations and storage to give some security guarantees, there is a growing need to implement some security measures in environments with very limited computation capabilities, like sensor networks, RFID devices or the Internet of Things.

Lightweight cryptography aims at providing a reasonable security level but using the minimal computation and storage resources. There are nowadays many proposals, and the number of applications is increasing fast.

It was introduced in 2018 by the National Institute of Standards and Technology (NIST) as *algorithm standards that can work within the confines of a simple electronic device* (Futurex, 2019).

This encryption technology uses less memory and computing resources, and way less power to provide solutions for this limited devices. For example, AES and SHA are great together, but unable to cope with an IoT environment.

The small and simple nature of the electronic devices of IoT makes them unavailable to process current cryptographic algorithms. Lightweight crypto would function better to secure the sensitive data transmissions ocurring between them.

1.1 Difference Between Conventional and Lightweight Crypto

The conventional cryptography has several algorithms that work pretty well on desktop devices with certain level of computing and memory resources, such as AES, SHA-265, RSA or ECC, but they cannot perform well on embedded systems with constraint resources. Lightweight cryptography comes to rescue when several computational constraints appear. We talk about lightweight crypto when the computational, memory, energy, or circuit resources that are needed to run the algorithm, are way much lower than the conventional crypto ones.

On the other hand, it is not optimal to lose security at the same time that size of the keys or the computational cost are reduced. One of the metrics that is used to consider an algorithm as light is the size of the circuit that will be running the algorithm. Moreover, we can consider the computing and execution times as a measure of lightness of the algorithm.

The throughput of any cryptographic system is calculated as the average of total plain text in a number of k bytes, divided by the average encryption time. In decryption mode, throughput is calculated as the average of total ciphertext, divided by the average decryption time.

To tell wether a crypto algorithm is secure there is an ISO standard that tell us about the security that an algorithm is providing. Lightweight crypto algorithms must follow everything related to Security strength requirements, specified on these standards.

1.2 Why is Lightweight Crypto Required for IoT?

- **Efficiency of end-to-end communication.** We have to remind that encription is already applied into the link layer but sometimes is more efficient applying encryption into the application layer allowing implementing security independently from the type and the structure of the communication system.

- **Applicability to lower resource devices.** The device spectrum for lightweight crypto is described as it follows:
  1. Embedded systems
  2. RFID sensors

  Lightweight crypto is ment to be the answer to those who wanted to deploy algorithmic cryptosystems into resource constrained devices, such as RFID, sensors or any kind of device belonging to the IoT environment.

  Smart and light technologies are new modern trends for the world that use IoT devices. Consequently, researchers have been developing and proposing a range of cryptographic algorithms to suit them. The NIST provides a LWC project that describes the issues and develops a technique for the standardization of lightweight cryptographic algorithms.

  In this project, numerous metrics are identified to evaluate the lightweight properties, in hardware implementation chip size (area or resource) and/or energy consumption (performance), and in software implementation code size and/or RAM size.

1.3 Target Devices

Lightweight cryptography targets a great number of devices. Conventional crypto generally performs well in server and desktop computers, as well as in tablets and smartphones. However, lightweight cryptography is primarily focused on constrained devices such as embedded systems, RFID and Sensors.

Table 1: *Device Spectrum (McKay et al., 2017)*

Servers and desktops	Conventional cryptography
Tablets and smartphones	
Embedded systems	Lightweight cryptography
RFID and sensors networks	

Microcontrollers are available with a wide variety of performance attributes. Even though 8/16/32-bit systems are the most common, 4-bit microcontrollers are on the top of sales, with only a small number of simple instructions. This leads in a large number of cycles to execute crypto algorithms, making them slow and heavy energy-consumers, resulting in a problem. The amount of RAM and ROM memorys can be limited as well.

RFID tags that are not battery-powered, will only have a small amount of energy available from the environment. These devices require crypto algorithms not only to use a limited amount of gate equivalents (GEs), but also meet severe timing and power requirements.

We must not forget that, the environment and applications need to be assessed into the decision of wheter or not to use conventional cryptography. A particular device may not have the complete decision for requiring lightweight standards, but also those devices in the application that are interacting between each other.

2 Requirements and challenges for Lightweight Crypto

There are some factors to be taken into account to achieve lightweight crypto systems. Firstly, we must consider that there are two ways of facing off the design of a lightweight crypto algorithm, as solutions can be implemented on both hardware and software. As it has been said, it is very challenging for resource-limited technologies to implement conventional crypto algorithms, due to their size, power consumption and speed. A trade-off must be done to implement lightweight, using **less memory**, **less computing resources** and **less energy consumption**, providing security solutions over resource-limited technologies (Buchanan, Li, and Asif, 2018).

Hardware implementation is ment to reduce the overall implementation costs, taking as starting point the traditional cryptography, but applying some techniques to make it suitable and low resources devices, pulling the efficiency up as much as possible when performing its tasks. Hence, requires special consideration on chip area covered by the crypto mechanism and its energy consumption, trying to lessen both compared to conventional methods.

To assess the lightness, we require to study metrics as the type of circuit and memory for storing internal and key states, throughput and TP/A ratio. But we cannot forget that a shorter block and key are not better, since it may cause security faults.

Area can be stated in terms of logic blocks for field-programmable gate arrays (FPGAs), where a logic block is the basic reconfigurable unit, containing a number of look-up tables (LUTs).

The Gate Equivalents (GEs) depend on the logic gates of the CMOS technology. When the time comes to manufacture the technology, GEs are measured to count off the complexity of the circuit. In the case of a CMOS circuit, we would have NAND2 gates with a given silicon area. Knowing the number of GE on a circuit, allows us to figure out how many gates are available. For meeting the security standards on a low-cost RFID tag, this could have a total gate count of 1,000 to 10,000 gates, with only 200 to 2,000 available for security purposes. (McKay et al., 2017).

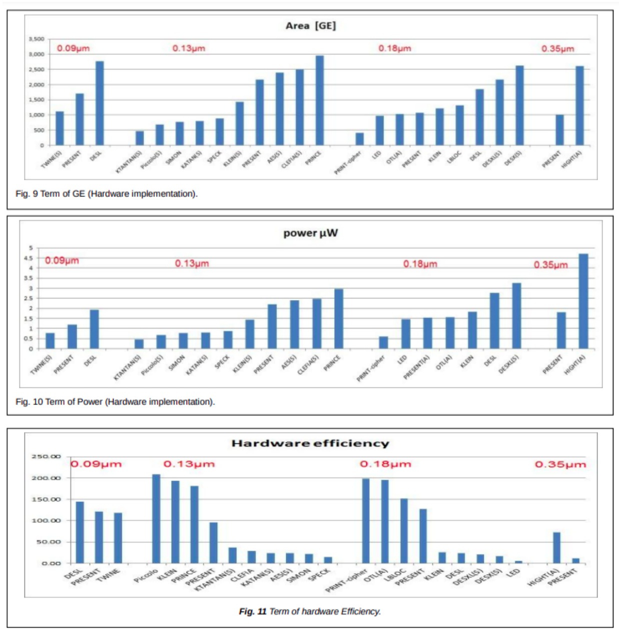


Figure 1: *Comparision of Lightweight crypto (Abdulrazzaq H. A. Al-ahdal, 2020)*

Throughput is another metric to consider, being necessary to use simple cycles and minimize **power consumption**. The latter is heavily important, not only for passive devices such as RFID tags or smart cards, but for active devices like wireless sensors. Energy use is directly related to the chip area: the lower the area, the lower the power consumption.

Table 2: *Comparison of Hardware Implementations (Diehl et al., 2017)*

Name	Rounds	Cycles/Block	Freq	Area (LUTs)	Throughput (Mbps)	TP/A (Mbps/LUT)
AES-180	10	306	287	318	119	0.38
SIMON 96/96	52	52	564	435	1041	2.39
SPECK 96/96	28	28	473	452	1622	3.59
PRESENT-80	21	32	542	311	1084	3.49
LED-80	48	244	423	358	111	0.31
TWINE-80	36	64	658	306	1170	3.82

This comparison has been done over six secret-key block ciphers using hardware implementations over Register Transfer Level designs. Five of them are lightweight, and AES-128 was included for purposes of comparison. According to the results, TWINE has the highest throughput-area ratio.

Software implementation cares about the code size and the required RAM, trying to reduce them for existing methods for a same platform. It considers metrics as implementation size, number of bytes of RAM and ROM that are required and bytes per cycle. ROM is used to store the program code and fixed data, while RAM is used to store intermediate values necessary for computation. In software cases, it is preferable as small as possible, and solutions are measured against the Fair Evaluation of Lightweight Cryptographic Systems (FELICS) framework, see table 3.

Table 3: *Comparison of Software Implementations (Diehl et al., 2017)*

Name	Program Bytes	Data bytes	Table bytes	Cycle blocks	Cycle bytes
AES	396	56	256	14,360	898
SIMON	274	43	0	58,234	4853
PRESENT	245	32	0	29,046	2421
SPECK	156	48	0	20,030	2504
LED	313	54	80	30,015	3752
TWINE	253	64	80	19.892	2487

On the other hand, this comparison has been using a lightweight 8-bit microprocessor. The results in terms of memory and cycles are shown in the table, AES shows the lowest cycle-per-byte number.

Based on these metrics, most algorithms are going to require small internal states, short blocks and key sizes. Generally, lightweight implementations use only 64-bit blocks. However, this will cause problems. Short blocks can bring **CBC erosion** faster when the number of blocks encrypted approaches $2^{n/2}$, for n-bit size blocks. In addition, short keys can increase **key-related attacks** as well.

2.1 Metrics of LWC

Generally, we will consider the following metrics when talking about lightweight cryptography:

- **Security strength**: any selected algorithm must be providing high security.
- **Hardware technology**: The area is a measure of hardware size, expressed in μm, and based on CMOS technology used. The number of Gate Equivalent (GE) refers to area and complexity in the implementation, which depends on the technology used. This number is a common metric to measure efficiency, and can be calculated by dividing the silicon area used by the area of a twoinput NAND gate.
- **Throughput**: Frequencies resulting from encryption and decryption processes are measured in (Kb/s).

- **Latency**: It refers to the number of clock-cycles necessary to compute a single block's encryption / decryption.
- **Power and energy consumption**: The estimate of power relies on the hardware technology and GEs. In addition, the energy consumption per bit can be computed by the formula:

$$Energy = Latency \cdot \frac{Power}{Block\ size} \tag{1}$$

Figure 2: *Energy consumption formula (Diehl et al., 2017)*

Where *energy* is measured by μJ, *latency* by $Cycles/Block$, *power* using μW and the *block size* in bits.

- **Efficiency**: Refers to the reduction of resource usage in hardware or software. It is trade-off between implementation size and performance.

3 Catalog of Primitives

Conventional crypto methods such as Advanced Encryption Standard (AES), Secure Hash Algorithms (SHA-265), Rivest–Shamir–Adleman (RSA) and Elliptic curve cryptography (ECC) work well on systems with high computational power and memory, but they cannot deliver on embedded systems.

In many cases, encryption is more effective in the application layer, please see Figure 3, giving an end to end protection from device to server. The encryption is aplied at the processor, thus, it must be as lightweight as possible.

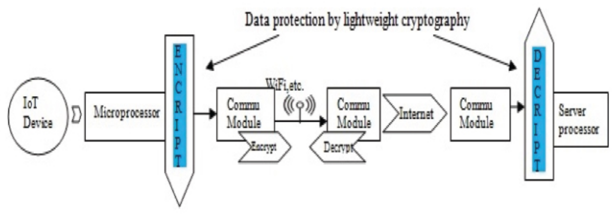


Figure 3: *Application Layer Implementation of Lightweight Crypto (Jadhav, 2019)*

Elliptic Curve Crypto has been considered as one of the best decisions for resource-limited devices. For example, security increases as the key size much faster in asymmetric algorithms such as RSA as compared to ECC. Hyper Elliptic Curve Crypto, its counterpart, is considered to have even better performance (Jadhav, 2019).

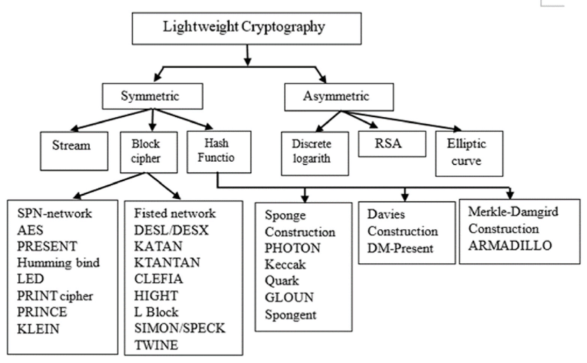

Lightweight Cryptography
Symmetric
Asymmetric
Stream
Block cipher
Hash Functio
Discrete logarith
RSA
Elliptic curve
SPN-network AES PRESENT Humming bind LED PRINT cipher PRINCE KLEIN
Fisted network DESL/DESX KATAN KTANTAN CLEFIA HIGHT L Block SIMON/SPECK TWINE
Sponge Construction PHOTON Keccak Quark GLOUN Spongent
Davies Construction DM-Present
Merkle-Damgird Construction ARMADILLO

Figure 4: *Classification of Lightweight crypto (Badr, Zhang, and Umar, 2019)*

3.1 Symmetric Key Crypto

The catalog of lightweight cryptographic primitives is formed by the following categories (Mileva et al., 2021):

Block Ciphers encrypt blocks of plaintext one at a time, to blocks of ciphertext, using a multiple-round system an a secret key. Moreover, in each round, a derived key is used, providing confusion and diffusion. We use a key in each round which is derived from a key schedule algorithm using a secret key.

The standard block cipher can be taken to a lightweight approach by using smaller key sizes (80 to 96 bits for instance), smaller block sizes, smaller key schedules, lighter hardware implementations, smaller building blocks, etc:

- **Smaller key sizes**: Small key sizes are used by lightweight block ciphers (less than 96 bits) for efficiency purposes. At the time of this writing, the minimum key size required by NIST is 112 bits (Barker and Roginsky, 2019), but a transition to 128 is planned near the year 2030.
- **Smaller block sizes**: Lightweight approaches may use smaller block sizes than AES (e.g., 64/80 bits, rather than 128 bits). However, it must be said that using smaller blocks reduces the limit on the maximum number of plaintext blocks to be encrypted, thus, leading to attacks such as plaintext recovery or key recovery.
- **Simpler key schedules**: It is preferable to generate sub-keys without increasing memory, latency or power consumption, thus, lightweight block ciphers tend to use simple key schedules that do this process on the fly. Attacks may occur as well, using related or weak keys.
- **Lighter implementations**: We must ensure to implement only the necessary functionality of a cipher to require the minimum of resources, rather than implementing the full cipher.

If we take the example of AES-128, we see that this algorithm meets these requirements with its area of 2400 GE and 0.18 μm technology, despite not being able to be applied on every scenario. Please see table 4 for a more complete comparison.

According to the structure, block ciphers can be subclassified into several types:

- Substitution Permutation Network (SPN). Each round consists of substitution (non-linear) and permutation (linear), providing confusion and diffusion respectively.
- Feistel Network (Feistel). Input is separated into two halves, L_i and R_i and in each round, the output block is $(L_{i+1}, R_{i+1}) = (R_i, L_i \oplus F(R_i, K_{+1}))$, where F is the round-function introduced by H. Feistel.
- Add-Rotate-XOR (ARX). Modular addition, rotation and XOR.
- Generalized Feister Network (GFN). Divides the input block into *n* parts. Each rount consists of a round-function and a block permutation.
- LFSR-based. The round function uses one or more Linear Feedback Shift Registers (LFSRs).
- LS-design. Each round combines linear diffusion L-boxes with non-linear bitslice S-boxes.
- XLS-design. Variation of LS-design, using additional Shift Columns operation and Super S-boxes.

Among all the block ciphers, you can find also tweakable block ciphers where, in addition to the key and the message, there is a third input, which is the tweak. A tweak is a family of permutations in which each pair selects one permutation.

The following table gives a summary of the known lightweight block ciphers and their characteristics just mentioned:

Table 4: *Lightweight block ciphers.*

Name	Type	Key Size (bits)	Block Size (bits)	No. of rounds	Area (μm)
SIMON	Feistel	64/72/96/128	32/48/64/96	32/36/42/52	0.13
SPECK	ARX+Feistel	64/72/96/128	32/48/64/96	22/26/28/32	0.13
TWINE	GFN	80/128	64	36	0.09
LED	SPN	64/128	64	32/48	0.18
AES-128	SPN	128	128	10	0.18
PRESENT	SPN	80/128	64	31	0.18
KATANn/	LFSR-based	80	32/48/64	254	0.13

Stream Ciphers encrypt small pieces of data at a time using a secret key. It generates a pseudorandom keystrem which, combined with the plaintext, produces the ciphertext bits. Stream ciphers often use a large keystream period, and a different key and/or IV should be used afterwards. Two phases occur, initialization an encryption. Usually a fast initialization comes with many short messages, while when large are received, fast encryption is preferable.

Stream ciphers can be made lightweight similarly to block ones, using smaller key sizes, smaller IV/nonces, smaller internal states, simpler key schedules and easier hardware implementations.

In addition, there is a golden rule for stream ciphers; we must do not encrypt two different messsages with the same pair key/iv. For that reason, stream ciphers usually have long keystreams using different IVs.

The following table gives a summary of the known lightweight stream ciphers and their characteristics just mentioned.

Table 5: *Lightweight stream ciphers.*

Name	Key Size (bits)	IV (bits)	Internal State (bits)	No. of rounds	Area (\mu m)
A2U2	61	64	95	var.	-
ChaCha	256	64	512	8/12/20	0.18
Salsa20	256	64	512	20	0.18
Trivium	80	80	288	1152	0.35
Quavium	80	80	288	1152	-
WG-8	80	80	160	40	0.065
ZUC	128	128	560	32	0.065

For a fair evaluation and comparison of lightweight block and stream ciphers implementations, a free and open-source benchmarking framework FELICS (Fair Evaluation of Lightweight Cryptographic Systems) can be used (Dinu et al., 2015).

Hash functions are methods to map variable size inputs into fixed length outputs, called as hash or digest. These functions must be one-way and collision resistant. The process usually starts with a padding generation and then dividing the message into fixed size blocks, afterwards, a compression function iterates over each block and produces a chaining value (Merkle-Damgård construction).

Other approaches, as sponge constructions, are based on fixed-length unkeyed permutation (P-Sponge) or randomn function (T-Sponge).

Their main issue of conventional hash functions is their large internal state. SHA-3 uses a 1600-bit IS and SHA-256 uses 256. Lightweight approaches can have smaller internal states and digests, small hardware implementations and better performance on short messages:

- **Smaller internal states**: For applications that do not require collision resistance, smaller states and outputs can be used.
- **Smaller message size**: Most lightweight targets require much smaller input sizes (at most 256 bits). Therefore, optimized hash functions for short messages may be more suitable.

The table that follows lists the cryptographic properties of some of the known lightweight hash functions.

Table 6: *Lightweight hash functions.*

Name	Type	Digest (bits)	Internal State (bits)	Rate (bits)
SPONGENT	P-Sponge	80/128/160	88/136/176	8/16
Lesamnta-LW	MD	256	256	128
PHOTON	P-Sponge	80/128/60/224	100/144/196/256	16/20/32/36
DM-PRESENT	MD	64	64	80/128
GLUON	T-Sponge	128/160/224	136/176/256	8/16/32
QUARK	P-Sponge	136/176/256	136/176/256	8/16/32

Message Authentication Codes consist of a tag generation from the message and a secret key. MAC constructions are based on block-ciphers (CBC-MAC, OCB-MAC), or one hash functions (HMAC), to name some. Lightweight implementations include shorter tags, smaller hardware/software constructions, skipping the use of nonces and use of lightweight block ciphers and hashes.

Table 7: *Lightweight MACs.*

Name	Key size (bits)	Block Size (bits)	Tag size (bits)	No. of rounds
Chaskey	128	128	64	8
LightMAC	2 X 80/128	64/128	64/128	var.
SipHash	128	256	64	2 + 4
TuLP	80/160	64/128	64	14

Authenticated Encryption Schemes combine ciphers and MACs in one primitive, providing all confidentiality, integrity and authentication. Moreover, both input and keys are of variable length, accepting as well Associated Data, a public nonce and an optional secret one. AD must be authenticated, but not encrypted.

Table 8: *Lightweight AEs.*

Name	Type	Key (bits)	Internal State (bits)	Nonce (bits)	Tag (bits)
ALE	SC	128	256	128	128
APE	MD	160	196	36	160
C-QUARK	Sponge	256	384	64	64
FIDES	Sponge	80/96	160/192	80/96	80/96
Hummingbird-2	Hybrid	128	128	64	128
LAC	SC	64	144	80	64

Let's analyze how some of the most common lightweight algorithms work:

- **Present algorithm** : PRESENT Cipher model combines both 80-bit and 128-bit encryption and decoding steps for 64-bit input data security at hardware level. It is a SP-Network structure of 31 rounds. It operates with a block length of 64-bits and two keys of 80 and 128 bits. Each of the 31 rounds includes XOR to add K_i for $1 \leq I < 32$, where K_{32} is used for post-whitening, bit by bit. The pipeline of the algorithm 6, includes operations of blocks with plain text and applying a key. Then, it uses a number of rounds of S-boxes5 and P-boxes, through a bitwise XOR operation, while parts of the key are included through the rounds. When decrypting, the S-boxes and P-boxes are reversed in their operations.

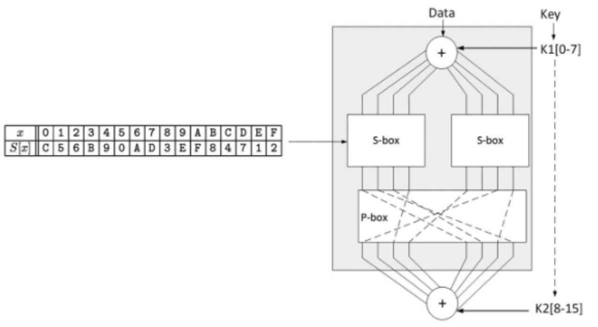
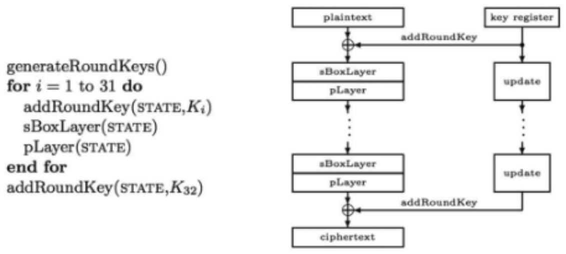

Data
Key
K1[0-7]
x 0 1 2 3 4 5 6 7 8 9 A B C D E F
S[x] C 5 6 B 9 0 A D 3 E F 8 4 7 1 2
S-box
S-box
P-box
K2[8-15]

Figure 5: *Substitution box*



```
generateRoundKeys()
for $i = 1$ to 31 do
    addRoundKey(STATE, $K_i$)
    sBoxLayer(STATE)
    pLayer(STATE)
end for
addRoundKey(STATE, $K_{32}$)
```

Figure 6: *Present algorithm*

- **HIGHT algorithm:** The hight algorithm (high security and lightweight) is ment to be deployed on a FPGA, just like PRESENT. It was adopted as the international cryptographic algorithm specification for ISO/IEC 18033-3. HIGHT block cipher is implemented in low-cost hardware, addressing low-end devices and implementations, with an ARX architecture that may process 64-bit blocks and 128-bit key sizes. The HIGHT algorithm is made out from an initial transformation, followed by round functions, and a final transformation.
- **SIMON algorithm:** It is based on Feistel Architecture. The authors suggested a cipher with variable parameters of 32/48/64/96/128-bit block size and 64/72/96/128/144/192/256-bit key size, with a number of rounds from 32 to 72. Each of the repeated rounds consist of several bit-wise and logic operators, such as XOR or Rotate, offering a clear image of potential examination, required to protect the IoT environment.
- **Tiny Encryption Algorithm (TEA):** Block cipher created by the founders of the Cambridge Computer Security Laboratory; and first implemented in the 1994 Fast Machine Encryption Workshop. It consists of a Feistel structure of 64 rounds (or 32 cycles with two rounds each). The plaintext block has a length of 64 bits and the key size proposed was 128 bits. In addition, the key is split into four 32-bit blocks, `k[0]` to `k[3]`.

3.2 Public Key Crypto

Since 2012, there are some public key algorithms designed to be deployed in resource limited devices, just like the one published by Hermans, 2012, which consists in a new way to identify and track these kind of sensors, based on public key infraestructures.

The lightweight criptography based on symmetric key was the most common technique, until Lee et al., 2008 and Hein, Wolkerstorfer, and Felber, 2009 proved that algorithms based on elliptic curves, allowed also the use of public key criptography on RFID, due to their relative small footprint. Previously, public key cryptography was considered as too demanding, in terms of resources, power, and memory consumption.

Subsequently, Hermans, 2012, began to focus into trying to make impossible the forgery of an RFID tag. At the same time, he tried to keep privacy meeting the required standards. In this case, privacy means that no adversary would be able to trace and/or identify tag appearance, taking some concepts introduced by Vaudenay, 2007, like his privacy model. In this model, the adversary has the ability to influence all communications between tag and reader, being able to perform man-in-the-middle attacks within its range.

Other types of interactions would be drawing and freeing random tags, placing them in-and-out their range, using a virtual identifier. This would refer those tags that are inside the range, besides corrupting them and learning their entire state. Here is when public key algorithms come into the picture, to prevent information leaks from the tags. Batina et al., 2012 designed a grouping proof protocol based on public cryptography. An attack, as we understand it, it the abbility of generating a valid grouping proof, where T_a and T_b are scanned together.

The schema of the colluding tag prevention protocol is shown on figure 7. Jens Hermans provided another example of public key crypto, taking his own privacy model as a start point. He created new yoking proofs, based on public key with two models of grouping, one using an external trusted party and another without it. We understand yoking as the act of joining proofs. His proposal consisted in a game defined as follows:

1. First, the challenger picks a random challenge bit b and then sets up the system S with a security parameter k.
2. Next, the adversary A can use a subset (depending on the privacy notion) of the following oracles to interact with the system: `CreateTag(ID)`, `DrawTag($T_{i}$, $T_{j}$)`, `Free(vtag)`, `SendTag(vtag, $m$)`, `SendReader($\pi$, $m$)`, `Result($\pi$)` and `Corrupt($T_{i}$)`.

`DrawTag` defines which tags to interact with.

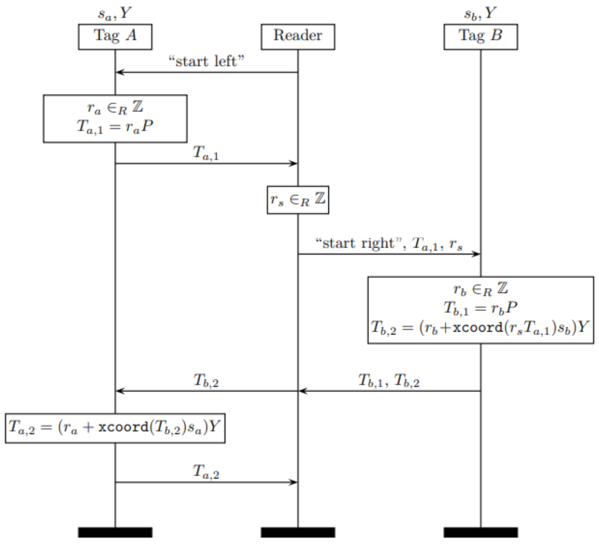


Figure 7: *Colluding tag prevention schema (Batina et al., 2012)*

This grouping proof has mainly two properties: correctness and soundness. **Correcteness** is definded by the following statement (Hermans, 2012): *A scheme is correct if a legitimate grouping proof is rejected with negligible probability and all tags involved are identified correctly with overwhelming probability*. The second one is **soundness**, note there is a distinction between timed grouping proof soundness and non-timed grouping proof soundness, considering wether the time at wich the proof is generated and recorded can be verified afterwards or not. Furthermore, for a timed grouping proof we will need a third party to provide us the timestamps.

- Timed grouping soundness starts with the interaction phase, where the adversary may reach all tags. On the second phase the adversary will also be able to interact with all tags, except for $T_c \subset S$, where $S \subset T$ is the set of tags for which a grouping proof is produced by the adversary. The tag T_c will also remain uncorrupted along the entire game. At the end of the second phase, the adversary outputs a candidate grouping proof σ. Having this in mind, we may say that *a grouping proof scheme is sound if no polynomially bounded strong adversary, with non-negligible probability, is able to produce a valid grouping proof for a set of tags S with time $t_1 > t_2$.*

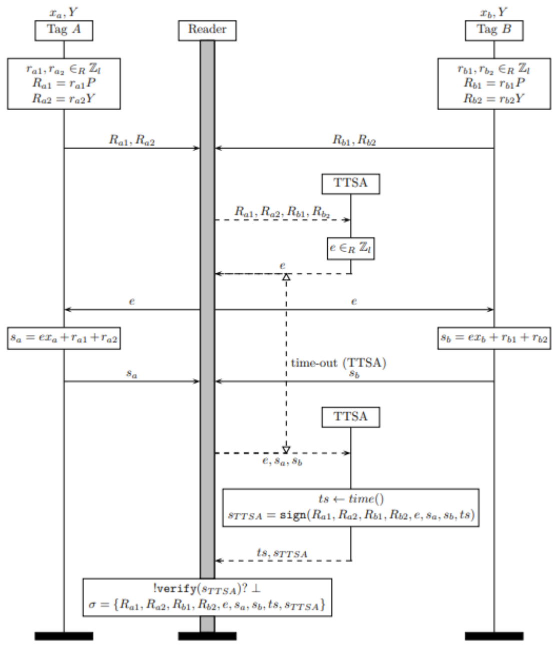
  Hence, we need the collaboration of all tags to produce a valid grouping proof. A non-timed grouping proof will be necessary, we need to ensure that all tags were together and completed the protocol.

These grouping proofs allow to be reused as much as needed.

- Non-timed grouping soundness. In this case, the first step is similar to the timed grouping soundness, where the adversary may interact with all tags, excluding T_a. This means that T_a can not be corrupted during this first phase. Secondly, the adversary can only interact with T_a, producing a candidate grouping proof σ at the end of the phase, from where we finally derive the following statement: *A grouping proof scheme is sound if no polynomially bounded strong adversary, with non-negligible probability, is able to produce a valid grouping proof for the group of tags $S = T_a, T_b, ...$ even when allowed to corrupt T_a in the second phase.*

  In this way, we ensure that two of the tags in the grouping proof cannot perform a yokin protocol together, that T_a cannot be used in the first nor second phase, and only T_b can be used.

A schema of the Two-party grouping-proof protocol with timestamp is shown below, see figure 8.

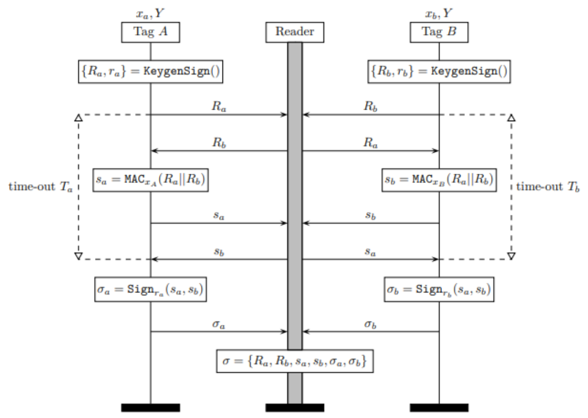


x_a, Y
Tag A
Reader
x_b, Y
Tag B
$r_{a1}, r_{a2} \in_R \mathbb{Z}_l$
$R_{a1} = r_{a1}P$
$R_{a2} = r_{a2}Y$
$r_{b1}, r_{b2} \in_R \mathbb{Z}_l$
$R_{b1} = r_{b1}P$
$R_{b2} = r_{b2}Y$
R_{a1}, R_{a2}
R_{b1}, R_{b2}
TTSA
$R_{a1}, R_{a2}, R_{b1}, R_{b2}$
$e \in_R \mathbb{Z}_l$
e
e
e
$s_a = ex_a + r_{a1} + r_{a2}$
$s_b = ex_b + r_{b1} + r_{b2}$
time-out (TTSA)
s_a
s_b
TTSA
e, s_a, s_b
$ts \leftarrow time()$
$s_{TTSA} = \mathtt{sign}(R_{a1}, R_{a2}, R_{b1}, R_{b2}, e, s_a, s_b, ts)$
ts, s_{TTSA}
$!\mathtt{verify}(s_{TTSA})? \perp$
$\sigma = \{R_{a1}, R_{a2}, R_{b1}, R_{b2}, e, s_a, s_b, ts, s_{TTSA}\}$

Figure 8: *Two-party grouping-proof protocol with timestamp (Hermans, 2012)*

The public keys X_a, X_b can be checked in the database of the verifier. This ensures that tag T_a and tag T_b were scanned together, at time ts. Look how the schema keeps the privacy, since the reader nor the TTSA (Trusted Time stamping Authority) are able to learn the identity of the tags. Watch carefully how the TTSA receives the commitments from the tags $R_{a1}, R_{a2}, R_{b1}, R_{b2}$, generating a exam e and ensuring the proper order of the authentication protocol. Eventually, the TTSA will sign with the timestamp, along with s_a, s_b, before the session times out (8).

In the case where no trusted party is available a sort of MAC will be needed, for validation. Both tags, T_a and T_b, will then generate a one-time key pair for signing, with public key R_a and private key r_a. In the second round, figure 9, both tags have a key pair x_a. Notice that a signature scheme would be also suitable instead of using a MAC. Both MACS will be signed in the final round, using the one time signing key.



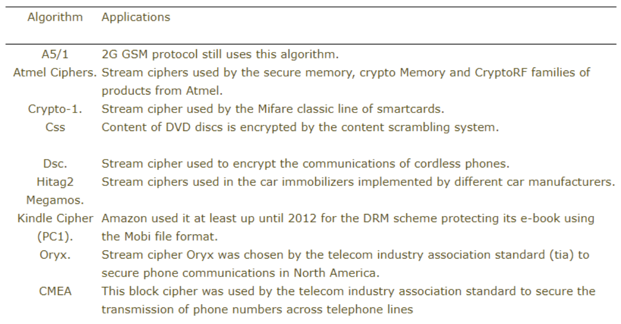
x_a, Y
Tag A
Reader
x_b, Y
Tag B
$\{R_a, r_a\} = \mathtt{KeygenSign}()$
$\{R_b, r_b\} = \mathtt{KeygenSign}()$
R_a
R_b
R_b
R_a
time-out T_a
time-out T_b
$s_a = \mathtt{MAC}_{x_A}(R_a \| R_b)$
$s_b = \mathtt{MAC}_{x_B}(R_a \| R_b)$
s_a
s_b
s_b
s_a
$\sigma_a = \mathtt{Sign}_{r_a}(s_a, s_b)$
$\sigma_b = \mathtt{Sign}_{r_b}(s_a, s_b)$
σ_a
σ_b
$\sigma = \{R_a, R_b, s_a, s_b, \sigma_a, \sigma_b\}$

Figure 9: *Two-party grouping-proof protocol without trusted party (Hermans, 2012)*

This protocol can be used by multiple tags, with the drawback of additional communication, that despite the overhead is relatively small. Nevertheless, if wanted to sign or deploy the MAC algortihm on bigger messages, the computational effort will raise.

These would be some examples of current yoking proofs, deployed on a public key infraestructure over RFID tags, to provide privacy and security.

The first was based on Batina et al., 2012 researchs, and the rest were new proposals using a trusted third party. These methods are scalable, and meet the security and privacy standards. At this point, nowadays we have plenty of asimetric key algorithms prepared to be deployed on IoT devices. The initial thoughts which stated that asimetric key algorithms were too heavy to be carryed out in small devices are not valid any more. On table 10, we can appreciate some of the lightweight crypto algorithms based on public key.



Algorithm	Applications
A5/1	2G GSM protocol still uses this algorithm.
Atmel Ciphers.	Stream ciphers used by the secure memory, crypto Memory and CryptoRF families of products from Atmel.
Crypto-1.	Stream cipher used by the Mifare classic line of smartcards.
Css	Content of DVD discs is encrypted by the content scrambling system.
Dsc.	Stream cipher used to encrypt the communications of cordless phones.
Hitag2 Megamos.	Stream ciphers used in the car immobilizers implemented by different car manufacturers.
Kindle Cipher (PC1).	Amazon used it at least up until 2012 for the DRM scheme protecting its e-book using the Mobi file format.
Oryx.	Stream cipher Oryx was chosen by the telecom industry association standard (tia) to secure phone communications in North America.
CMEA	This block cipher was used by the telecom industry association standard to secure the transmission of phone numbers across telephone lines

Figure 10: *Current Lightweight crypto algorithms based on public key (Jadhav, 2020)*

References

Futurex (June 2019). "How Will Lightweight Cryptography Impact You?" In: *Futurex*. URL: https://www.futurex.com/blog/how-will-lightweight-cryptography-impact-you/.

McKay, Kerry A. et al. (Mar. 2017). "Report on Lightweight Cryptography". In: *NISTIR 8114*. URL: https://nvlpubs.nist.gov/nistpubs/ir/2017/NIST.IR.8114.pdf.

Buchanan, William J., Shancang Li, and Rameez Asif (Mar. 2018). "Lightweight cryptography methods". In: *Taylor and Francis Online*. URL: https://www.tandfonline.com/doi/full/10.1080/23742917.2017.1384917.

Abdulrazzaq H. A. Al-ahdal, Nilesh K.Deshmukh (Mar. 2020). "A Systematic Technical Survey Of Lightweight Cryptography On Iot Environment". In: *INTERNATIONAL JOURNAL OF SCIENTIFIC AND TECHNOLOGY RESEARCH*. URL: http://www.ijstr.org/final-print/mar2020/A-Systematic-Technical-Survey-Of-Lightweight-Cryptography-On-Iot-Environment.pdf.

Diehl, William et al. (2017). "Comparison of hardware and software implementations of selected lightweight block ciphers". In: *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, pp. 1–4. URL: https://people-ece.vse.gmu.edu/~kgaj/publications/conferences/GMU_FPL_2017_LWBC.pdf.

Jadhav, Santosh Pandurang (Jan. 2019). "Towards Light Weight Cryptography Schemes for Resource Constraint Devices in IoT". In: *Journal of Mobile Multimedia*. URL: https://www.riverpublishers.com/journal_read_html_article.php?j=JMM/15/1/5.

Badr, Aymen Mudheher, Yi Zhang, and Hafiz Gulfam Ahmad Umar (Feb. 2019). "Dual Authentication-Based Encryption with a Delegation System to Protect Medical Data in Cloud Computing". In: *MDPI*. URL: https://www.researchgate.net/publication/330819474_Dual_Authentication-Based_Encryption_with_a_Delegation_System_to_Protect_Medical_Data_in_Cloud_Computing.

Mileva, Aleksandra et al. (Jan. 2021). "Catalog and Illustrative Examples of Lightweight Cryptographic Primitives". In: *Security of Ubiquitous Computing Systems*. URL: https://link.springer.com/chapter/10.1007/978-3-030-10591-4_2.

Barker, Elaine and Allen Roginsky (Mar. 2019). "Transitioning the Use of Cryptographic Algorithms and Key Lengths". In: *NIST Special Publication 800-131A*. URL: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-131Ar2.pdf.

Dinu, Daniel et al. (July 2015). "FELICS – Fair Evaluation of Lightweight Cryptographic Systems". In: *University of Luxembourg*. URL: https://csrc.nist.gov/csrc/media/events/lightweight-cryptography-workshop-2015/documents/presentations/session7-dinu.pdf.

Hermans, Jens (Aug. 2012). "Lightweight Public Key Cryptography". PhD thesis. Arenberg Doctoral School of Science, Engineering and Technology. URL: https://www.esat.kuleuven.be/cosic/publications/thesis-212.pdf.

Lee, Yong et al. (Nov. 2008). "Elliptic-Curve-Based Security Processor for RFID". In: *IEEE Trans. Computers* 57, pp. 1514–1527. DOI: 10.1109/TC.2008.148.

Hein, Daniel, Johannes Wolkerstorfer, and Norbert Felber (2009). "ECC Is Ready for RFID — A Proof in Silicon". In: *Selected Areas in Cryptography: 15th International Workshop, SAC 2008, Sackville, New Brunswick, Canada, August 14-15, Revised Selected Papers*, pp. 401–413. URL: https://doi.org/10.1007/978-3-642-04159-4_26.

Vaudenay, Serge (2007). "On Privacy Models for RFID". In: *Advances in Cryptology – ASIACRYPT 2007*. Ed. by Kaoru Kurosawa. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 68–87.

Batina, Lejla et al. (Mar. 2012). "Extending ECC-Based RFID Authentication Protocols to Privacy-Preserving Multi-Party Grouping Proofs". In: *Personal Ubiquitous Comput.* 16.3, pp. 323–335. DOI: 10.1007/s00779-011-0392-2. URL: https://doi.org/10.1007/s00779-011-0392-2.

Jadhav, Santosh Pandurang (Mar. 2020). "Towards Light Weight Cryptography Schemes for Resource Constraint Devices in IoT". In: *Journal of mobile multimedia* 16.3, pp. 91–110. DOI: https://doi.org/10.13052/jmm1550-4646.15125. URL: https://www.riverpublishers.com/journal_read_html_article.php?j=JMM/15/1/5.

Katagi, Masanobu and Shiho Moriai (May 2012). "Lightweight Cryptography for the Internet of Things". In: *Sony Corporation*. URL: https://iab.org/wp-content/IAB-uploads/2011/03/Kaftan.pdf.