WHITEPAPER

LIGHTWEIGHT CRYPTOGRAPHY

JANUARY 2021

ALEXANDRA INSTITUTE

AUTHORS

Kasper Damgård, (previously) Alexandra Institute

Tore Kasper Frederiksen, Alexandra Institute

Published by

THE ALEXANDRA INSTITUTE

January 2021, rev. 2

TABLE OF CONTENTS

1 INTRODUCTION 5

2 GLOSSARY 6

3 CRYPTOGRAPHIC PRIMITIVES 7

3.1 BLOCK CIPHERS 7

3.1.1 AES 8

3.1.2 Speck and Simon 9

3.1.3 KATAN and KTANTAN 10

3.1.4 PRESENT 10

3.2 STREAMING CIPHERS 11

3.2.1 Grain 11

3.2.2 Salsa20 / ChaCha 12

3.2.3 LEX 12

3.3 HASH FUNCTIONS 12

3.3.1 SHA families 13

3.3.2 PHOTON family 13

3.3.3 SPONGENT family 13

3.3.4 QUARK family 14

3.4 MACS 14

3.4.1 Authenticated Encryption Algorithms 14

3.4.2 Chaskey 15

3.5 ASYMMETRIC CIPHERS 15

3.5.1 ECC 16

3.5.2 NTRU 16

3.5.3 McEliece 17

4 SUMMARY 18

5 CONCLUSION 21

6 BIBLIOGRAPHY 22

1 INTRODUCTION

IoT is rapidly spreading, and with the technology becoming smaller and integrated into everything, producers often forget about the security aspects. Examples are plentiful; take for instance credit cards which could be remotely read and copied within minutes as shown by Kristin Paget, or passport information which could just as easily be stolen (Greenberg, u.d.). In essence, there is nothing new in the message in this whitepaper: If you send private data, ensure that the data is encrypted when leaving the device and that only the intended receiver learns the plaintext data. Also, ensure that the data cannot be tampered with without the receiver knowing that something bad happened. We call these properties *Confidentiality* and *Integrity*. In this whitepaper we will walk the reader through what is possible with today's technology and recent advances in the field of lightweight cryptography.

Lightweight cryptography is cryptography designed to run on devices or in environments with very limited resources. This can be both in terms of storage, CPU power, energy consumption, RAM or chip area. The area of lightweight cryptography is focused primarily on microprocessors (8 to 64 bit) and hardware implementations or so-called Application-Specific Integrated Services (ASICs) of crypto primitives. This means it covers anything from programmable Arduinos to RFID tags.

Because of the many variables, it is impossible to say if there is a "best" cipher, especially if we also consider the variables for the security guarantees. In essence, the best selection is made by carefully considering all variables in the individual application. We will still include numbers to give an idea of a comparison, but often those numbers are not entirely comparable as it is difficult to know if certain elements such as the key schedule or decryption part of an algorithm is included in the algorithm size. Also, comparing performance numbers is not going to be fair since implementations are done on different architectures, most likely picked to give the most favourable benchmarks for each algorithm. In other words: take numbers with a grain of salt, and confer with experts if you have a specific application setup that needs to utilize a lightweight primitive.

The rest of the whitepaper is structured as follows: First a glossary of terms which may not be well-known. Then we go through the primitives starting with block ciphers, then streaming ciphers followed by hash functions and MACs and finally ending with asymmetric ciphers. We then sum it all up in a large table containing key numbers, followed by our conclusion.

2 GLOSSARY

This whitepaper uses a few abbreviations which are nice to know in advance.

- **ASIC:** Application-Specific Integrated Service. Hardware circuit hardwired to do a single task.
- **Clock cycle (clk):** Amount defining how quickly a certain operation takes to process.
- **CPU:** Central Processing Unit. Device which processes data at a certain speed.
- **Cipher:** An algorithm which encrypts or decrypts data.
- **FPGA:** Field-Programmable Gate Array. Modern hardware circuit which can be programmed after shipping to the customer.
- **GE:** Gate Equivalent. Notation used for describing the physical size needed for the program to be implemented in hardware. 1 GE equals 1 NAND gate. To get a sense of what is small and what is big GE numbers, early ASICs contained around 5000 GE, while modern ASICs or FPGA's contains up to several millions.
- **RAM:** Random Access Memory. Used in computers for storing temporary data.
- **ROM:** Read-Only Memory. Contains data which cannot be changed without great difficulty, or not at all.
- **Round:** Within this whitepaper, a round is considered to be defined as part of an algorithm which loops a certain code segment multiple times. Each loop is a different round, and often a cipher is proven insecure up to a certain number of rounds (e.g. 10 out of 12 have been broken).
- **RSA:** Standard public key cipher used in many products all over the world.
- **SRAM:** Static RAM. Makes data stored accessible faster.
- **Throughput:** Measurement of how fast an algorithm can produce the intended output. All measurements within this whitepaper are given in kilobit per second (kbps).

3 CRYPTOGRAPHIC PRIMITIVES

A cryptographic primitive is a basic element within the field of cryptography which is combined with other elements to construct a protocol which in short enables either confidentiality or integrity (or both). In this whitepaper we will cover only the primitives themselves and compare them with each other. We will not go into the more advanced protocol level.

There are quite a lot of ciphers within the field of lightweight cryptography, and a large amount of these are surveyed by the university of Luxembourg (Biryukov & Perrin, State of the Art in Lightweight Symmetric Cryptography, 2017) and (University of Luxembourg, 2017), where detailed comparisons also can be found. Those comparisons are done using the open source FELICS (Dinu, et al., 2015) – Fair Evaluation of Lightweight Cryptographic Systems tool developed by the University of Luxembourg.

The tool compares 20 different ciphers on the parameters Code size, RAM usage and time measured in numbers of CPU cycles per operation. Note that hardware implementations and GE sizes are not included. They then provide a score or *Figure of Merit* (FoM), which gives an impression on the best cipher. The problem for non-experts is that all types of ciphers are gathered in one big table, and comparing a block cipher to a MAC doesn't really make sense as the purpose of each type is not at all the same. We will in the following pick out the most suitable if such exist and possibly runner ups in each category based on surveys done on which lightweight primitives are available. Note that the field moves fast, and is broad, so it is entirely possible that new and better ciphers arrive or that we missed a cipher. We also note that the list of ciphers is far from exhaustive, both in regard to theory and standardization. We direct the interested reader to the NIST lightweight standardization work[1] and the ISO/IEC series 29192 for more info and other possible crypto schemes.

3.1 BLOCK CIPHERS

Block ciphers operate on plaintext in blocks of a certain size. They transform (encrypt or decrypt) a block of data and outputs the transformed result. If the input data is longer than the block size, a mode of operation is needed which tells the block cipher how to go

[1] *https://csrc.nist.gov/Projects/lightweight-cryptography*

about transforming multiple blocks in a secure manner. These modes of operation are outside the scope of this whitepaper.

3.1.1 AES

Probably the most well-known block cipher is the Advanced Encryption Standard (AES). The technology behind the AES is the Rijndael cipher. AES was standardised in 1998 by NIST (Federal Information Processing Standards Publication, 2001) and in 2005 by ISO (International Organization for Standardization, 2005), and to this date it is still considered secure. AES comes in three variants called AES-128, AES-192 and AES-256. They all work on a block size of 128 bits, and the number appended to each variant corresponds to the key size used which also has an impact on the number of rounds it takes to complete an AES operation. AES has reached enormous acceptance, and for this reason and the security of the cipher, it is the NIST recommended cipher to use for lightweight cryptography (McKay, Bassham, Turan, & Mouha, 2017). AES has been studied for many years, which has led to advances in both reducing the code size and the physical size and increased the throughput obtained in both hardware and software implementations.

The most compact hardware implementation known to us was made by Mathew Sanu et al. (Sanu, et al., 2015) and fits within 2090 GE. Mathew Sany et al. also obtain the lowest energy consumption in throughput per Watt compared to all other implementations known to the authors at that time. Saving on energy consumption is important if the device is very constrained in battery size or only has a very limited power source (such as a smart card). We refer the reader to the paper for detailed numbers.

If RAM is limited to 64 bytes or ROM is limited to 512 bytes, it is not yet feasible to implement the full AES (McKay, Bassham, Turan, & Mouha, 2017), and other ciphers should be considered. A paper by Matsui and Murakami (Matsui & Murakami, 2013), shows that the full AES requires 128 bytes of RAM and 970 bytes of ROM. For encryption only, one can make do with 128 bytes of RAM and 486 bytes of ROM. The more space one has for code, the faster the implementation can get (up to a certain point).

AES is susceptible to a range of side-channel attacks. A side-channel attack is e.g. measuring the power consumption spikes or sound of the CPU. They all have in common that instruments for measuring must be physically present during the attack. These are hard to secure against because of the nature of the design of AES. A lot of effort has been put into this (Moradi, Poschmann, Ling, Paar, & Wang, 2011), but we are aware of the fact that no implementations are entirely leakage free. That being said, the effort, time, requirement of external instruments being present and insecurity in the leaks will most likely deter attackers – especially small IoT devices which in themselves might not reveal a critical mass of data if hacked somehow.

We recommend using AES where applicable due to the thorough reviews it has received in the past 20 years. No efficient attacks have been found, and it is expected that AES will survive many years from now.

3.1.2 SPECK AND SIMON

The block ciphers Speck and Simon (Beaulieu R. , et al., 2013/2014) have been developed by NSA to cover the area of general-purpose block ciphers which are efficient in both hardware and software and in any constrained environment. Other ciphers up until the release of Speck and Simon were specialized for a certain systems architecture. Both Simon and Speck are highly efficient in both hardware and software settings. This can be important for some settings where a mix is required.

Simon is hardware-oriented and the best hardware choice to our knowledge if resources are very limited in the hardware setting. If the block size is kept at 64 bits, the GE area can be as low as 1000 GE while still maintaining a 128-bit key. Compared to AES-128, which requires 2090 GE and operates on 128-bit blocks, Simon requires only 1317 GE. The throughput is worse than AES, though. The NSA reports the numbers 22,9 kbps for Simon whereas AES manages 56,6 kbps. These numbers are based on an ASIC implementation.

Speck is software-oriented and the best software choice to our knowledge if resources are limited in terms of code size or RAM usage. Speck, to compare with AES-128, requires only 396 bytes whereas AES requires 943 bytes. The throughput is very good at 768 kbps for Speck where AES manages 445 kbps (Osvik, Bos, Stefan, & Canright, 2010). These numbers are based on a software implementation on an 8-bit microcontroller.

No security analysis was presented along with the protocols themselves, making them stand out in the community. Later, these analyses were made by Farzaneh Abed et al. (Abed, List, Lucks, & Wenzel, 2013) and a pretty large number of others, none of which could prove the ciphers insecure.

Side-channel attacks are known for Speck and Simon but can be mitigated to some extent (Beaulieu R. , et al., 2015). This is work in progress, and if your application should be resistant to those types of attacks, we suggest digging deeper in the research as it is likely that new work has been done.

Even though scores of papers have been written looking into cryptanalysis of Speck and Simon, in large due to their origin of the NSA, no significant attacks have been found.

However, it should be noted that Simon and Speck were rejected from ISO/IEC 29192-2 (International Organization for Standardization, 2019) due to lack of justification of choice of internal random values. Furthermore, neither Speck nor Simon has been accepted as candidates for the NIST standardization for lightweight cryptography[2]. For these reasons, we recommend caution if deciding to use Speck or Simon although we do note that Speck has been ISO standardized for RFID usage (International Organization for Standardization, 2018).

[2] *https://csrc.nist.gov/Projects/lightweight-cryptography*

3.1.3 KATAN AND KTANTAN

KATAN and KTANTAN (De Canniere, Dunkelman, & Knežević, KATAN and KTANTAN—a family of small and efficient hardware-oriented block ciphers, 2009) are a family of very hardware-oriented block ciphers. The difference between KATAN and KTANTAN is that for KTANTAN, the key is hardcoded into the circuit. This saves space but could be considered less secure depending on the application. This results in the smallest block cipher with respect to GE area that we are aware of. KTANTAN32 presents numbers of 462 GE but can only handle a block size of 32 bits. For KTANTAN64, which is the largest block size available, the area grows to 688 GE. For KATAN where the key scheduling is not taken out of the equation, KATAN32 requires 802 GE and the larger KATAN64 takes up 1054 GE. Performance-wise, KATAN and KTANTAN are equal as there is only a difference in the key handling. For the 32-, 48- and 64-bit block size variants, the speeds are 12.5, 18.8 and 25.1 kb per second respectively. Thus, if the use-case is right, the KATAN or KTANTAN cipher might be the right one to pick as they are faster and potentially smaller than the Simon cipher (depending on how the key is stored). If larger block sizes are required, or the application should run in software, then neither KATAN nor KTANTAN are correct choices. That being said, there do exist several attacks but none that completely breaks them, assuming reasonable usage (Bogdanov & Rechberger, A 3-Subset Meet-in-the-Middle Attack: Cryptanalysis of the Lightweight Block Cipher KTANTAN, 2010).

Finally, it should be noted that these ciphers are neither standardized by ISO nor part of the NIST standardization process for lightweight cryptography.

3.1.4 PRESENT

PRESENT (Bogdanov, et al., 2007) is an ISO standardized block cipher (International Organization for Standardization, 2019), constructed specifically for the setting of lightweight cryptography. It was developed in 2007 by a group of researchers from Ruhr-University Bochum in Germany, DTU in Denmark and France Telecom. The cipher was designed with security and hardware efficiency in mind so as to complement AES in hardware constrained environments with less stringent security requirements. PRESENT works on 64-bit blocks and has 31 rounds. It supports both 80-bit and 128-bit keys. The throughput is impressive, at 200 Kbps on a 100 KHz CPU, although the GE is larger than the lightweight competitors of Simon and KATAN, at 1570 for the 80-bit variant. If the use-case is applicable with 64-bit blocks and speed is more important than the GE, or that standardization is a priority, then PRESENT is the correct choice.

Finally, we do note that PRESENT is not a candidate for NIST standardization of lightweight cryptography but has been standardized by ISO (International Organization for Standardization, 2019).

3.2 STREAMING CIPHERS

Opposite block ciphers which can only operate on a certain size of bytes at a time, streaming ciphers operate on streams of data. That is, they essentially use a key and what is called an Initialization Vector (IV) to produce a stream of key data which can then be used to encrypt or decrypt data. A well-known cipher is the RC4, but this has long been known to be insecure. Another similarly broken cipher is A5/1 used in the GSM phone network. It is sadly still used even though it has been proven extremely insecure. We mention these to ensure that the reader does not employ these in production.

In general, any block cipher can be turned into a stream cipher by utilizing a mode of operation such as CTR (Counter mode of operation), which makes it possible to work on streams of data. Since AES is the most common symmetric block cipher, we here mention it as a possible stream cipher in the variant of AES-CTR; that is, AES in counter mode of operation. If AES is usable, we recommend using AES-CTR over other mentioned ciphers. However, if the environment is too constrained, consider one of the ciphers mentioned below, or look into using one of the previously mentioned block ciphers in CTR mode.

We note that extensive work has been carried out by ECRYPT through the eStream project (ECRYPT, n.d.) to promote efficient streaming ciphers, which ended with a list of recommendations for both software and hardware stream ciphers.

3.2.1 GRAIN

Grain (Hell, Johansson, & Meier, 2007) comes in two versions. One that uses an 80-bit key and one with a 128-bit key. It is a hardware-oriented stream cipher, and the 80-bit version only uses 1294 GE area. Different versions are suggested, sacrificing the GE's to obtain greater throughput. With a larger area also comes the added advantage of lower power consumption per processed byte. The most secure and fastest Grain version requires 4617 GE, which is larger than the smallest AES counterpart. Fortunately, Grain is much faster and more energy-efficient.

It is also worth noting that Grain has shown to be one of the most efficient eStream ciphers (Good & Benaissa, 2008), with Trivium being the only serious contender. However, we leave Trivium (De Canniere & Preneel, TRIVIUM specifications, 2005) out of our recommendations since its authors themselves state that Trivium should not be used because of possible attacks on the scheme.

Finally, we also note that Grain has made it to round 2 of the NIST lightweight cryptography standardization process[3], although it is not ISO standardized.

[3] *https://csrc.nist.gov/Projects/lightweight-cryptography*

3.2.2 SALSA20 / CHACHA

Salsa20 (Bernstein D. , 2008) and the its improved ChaCha (Bernstein D. , 2008) variant are both software-oriented stream ciphers and should be considered when speed and code size are important. Salsa20 can be implemented in 1452 bytes and has a throughput which is approximately 44% faster than the AES-CTR (Eisenbarth & Kumar, 2007), although other sources state numbers more than 5 times as fast as AES. The data to operate on should be of a considerable size though, if one wants to avoid wasting a lot of bytes. The minimum state size is 512 bits. Salsa20 also requires 280 bytes of SRAM to function.

Although neither Salsa20 nor ChaCha are candidates for the NIST lightweight cryptography standardization, or ISO standardized, we note that Salsa20 is part of the eStream portfolio (ECRYPT, n.d.) and ChaCha has been standardized by the Internet Engineering Task Force (Y. Nir, Dell, A. Langley, Google, 2018).

3.2.3 LEX

Leak Extraction (LEX) (Biryukov, A New 128-bit Key Stream Cipher LEX, 2005) builds on AES and utilizes a property of any block cipher, meaning the construction should be transferable to other block ciphers than AES. The authors claim that LEX runs 2.5 times faster in both hardware and software than the AES counterpart, and the survey by Eisenbarth and Kumar (Eisenbarth & Kumar, 2007) states that implementation can be done using 1598 bytes. It does require a significant amount of SRAM, though: 304 bytes. LEX has a throughput of approximately double that of Salsa20 and uses block sizes of 320 bytes. Thus, if no bytes should go to waste, a significant data amount is required.

LEX is not a NIST lightweight cryptography standardization candidate, nor has it been standardized by ISO. It was, however, a candidate in the eStream effort but did not end up as part of its portfolio due to an attack allowing an attacker to recover the secret key after observing 2^{40} bytes of encrypted data (Dunkelman & Keller, 2013).

3.3 HASH FUNCTIONS

A hash function is also called a one-way function. One-way meaning that it is easy to go one way (i.e. from data to the hash value) but hard to obtain the pre-image given a hash value (e.g. the original data from the hash value). A hash value is a small number of bytes which can be considered unique based on the input data. Another important feature of a hash function is collision resistance. A hash function is considered collision resistant if it is hard to find two different data inputs yielding the same hash value.

The size of the hash value is also known as the output size and is measured in bits.

3.3.1 SHA FAMILIES

The SHA-1, SHA-2 and SHA-3 families are likely the best-known hash functions as they are the standard used for just about any use-case. They are standardized by both NIST (Federal Information Processing Standards Publication, 2015) and ISO (International Organization for Standardization, 2018). However, they are not the smallest, neither in terms of code size, nor physical GE size. We do not recommend using the SHA-1 family if it can be avoided, as it is no longer considered secure (McKay, Bassham, Turan, & Mouha, 2017). The remaining hash functions are thoroughly tested and should be used were applicable.

However, in constrained environments, it might not be possible to utilize the SHA families, in which case alternatives can be used. NIST (McKay, Bassham, Turan, & Mouha, 2017) states that it is impossible to implement the SHA-2 and SHA-3 families within 64 bytes of RAM. Still, it might not be very farfetched to use SHA-3 in some constrained environments. SHA-3 is a subset of the KACCAK family (Bertoni, Daemen, Peeters, & Van Assche, 2009) and a construction of the smallest KECCAK hash function primitive takes up only approximately 1300 GE with a throughput of 18.6 Kb per second at 1MHz. However, a newer study of the family (Pessl & Hutter, 2013) presents results of an incredible 4000 Kb per second with an area requirement of just 4900 GE using parallel execution. Matsui and Murakami (Matsui & Murakami, 2013) found that KECCAK can be implemented in software which requires at least 512 bytes of RAM and 453 bytes of ROM.

3.3.2 PHOTON FAMILY

The PHOTON group of hash functions (Gio, Peyrin, & Poschmann, 2011) was developed to allow RFID tags the possibility to use the hash function primitive. In other words, this is designed for hardware. They were ISO standardized in 2016 (International Organization for Standardization, 2016). As for the SHA families, they come in variants: PHOTON-80, PHOTON-128, PHOTON-160, PHOTON-224 and PHOTON-256. The number corresponds to the output size in bits. The GE areas and other technical details are noted in the paper by Jian et al. (Gio, Peyrin, & Poschmann, 2011), but it can be as small as 865 GE for PHOTON-80 and 2177 GE for PHOTON-256. PHOTON has a decent speed as well and should be suitable for cases where the SHA families are not applicable. They claim that power consumption should be quite acceptable but have only run simulations which therefore might not be entirely accurate.

Furthermore, PHOTON has been standardized for lightweight cryptography by ISO (International Organization for Standardization, 2016) and a variant (PHOTON-Beetle) has made it to round 2 of the NIST lightweight cryptography standardization process.

3.3.3 SPONGENT FAMILY

As with PHOTON, SPONGENT (Bogdanov, et al., 2011) is a hardware-oriented hash function family which comes in the variants of output size 88-, 128-, 160-, 224- and 256-bits. SPONGENT is also ISO standardized (International Organization for

Standardization, 2016) and beats PHOTON in GE area by a small margin, ranking from 738 GE for SPONGENT-88 up to 1950 GE for SPONGENT-256. Power consumption numbers are given, but as for PHOTON, they are only simulated numbers. As for throughput, SPONGENT is not doing as well as PHOTON, meaning the GE area requirement should weigh higher than speed. Most often, it seems PHOTON beats SPONGENT.

3.3.4 QUARK FAMILY

The QUARK (Aumasson, Henzen, & Naya-Plasencia, 2010) family comes in three variants: U, D and S where U corresponds to 136 output bits, D to 176 output bits and S to 256 output bits. The area requirements are slightly higher than both PHOTON and SPONGENT at 1379 GE for U-QUARK and 2296 GE for the S-QUARK. The throughput, however, is quite acceptable with QUARK being the fastest but at the sacrifice of a greater GE area. Thus, any of the mentioned hash families are usable, and the right one depends on context and application.

We note that QUARK is not a NIST lightweight cryptography standardization candidate, nor has it been standardized by ISO.

3.4 MACS

A MAC is short for Message Authentication Code and will in itself not secure the confidentiality of the data it operates on. Instead, it is used for ensuring integrity of the data in the sense that the receiver can compute the MAC on the received data, and if it matches the MAC attached to the data, you have a certain guarantee that the data has not been manipulated during transfer. This requires of course that the key used for the MAC is kept confidential from the attacker as the attacker could otherwise just replace both data and MAC to match. Normally, one would use a MAC in conjunction with a scheme that keeps data confidential or use one of the so-called authenticated encryption algorithms which handles both confidentiality and integrity.

If the use-case enables the use of standard MAC schemes such as HMAC (Krawczyk, Canetti, & Bellare, 1997) or AES-CMAC (Song, Poovendran, Lee, & Iwata, 2006), those schemes should be used due to the security of the schemes and thoroughly tested implementations.

3.4.1 AUTHENTICATED ENCRYPTION ALGORITHMS

GCM (McGrew & Viega, 2004) and CCM (Dworkin, 2007) are modes of operation used on any block cipher to simultaneously provide confidentiality and integrity of the data. They are not the only modes capable of doing so but the only ones recommended by NIST (McKay, Bassham, Turan, & Mouha, 2017). More lightweight versions also exist such as JAMBU (Wu & Huang, 2016) or Hummingbird-2 (Engels, Saarinen, Schweitzer, & Smith, 2011). However, the reason none of these are recommended is that they have

not yet been analysed to full extent, and the security claims of the authors are not always correct (as proven by Payrin et al. (Payrin, Sim, Wang, & Zhang, 2015) in the JAMBU case and by Kai et al. (Zhang, Ding, & Guan, 2012) in the Hummingbird-2 case). There may still be reason to consider these, as the cryptanalysis has not entirely broken the schemes but instead reduces the security margins significantly. Implementations of Hummingbird-2 can be done with approximately 2000 GE, making it worth considering if space is very critical.

GCM or CCM are commonly available options in most implementations of AES but could be applied to any of the lightweight block ciphers previously mentioned in this whitepaper. If you think you need to implement it yourself, consult with an expert before doing so, as implementation details are of immense importance within security.

3.4.2 CHASKEY

Chaskey (Mouha N. , 2017) is a simple clean MAC variant with a very efficient implementation and code size. It is intended to perform exceptionally well on software for a wide range of 8-bit to 32-bit microcontrollers which are considered constrained environments.

The code is open source and available free of charge. Several cryptanalysis attacks have been made but none succeed in completely breaking Chaskey. We suggest using Chaskey when a MAC is needed in constrained environments.

Even though Chaskey is optimized for software, extremely efficient hardware implementations have been made by e.g. (Lan, Zhou, & Liu, 2016), requiring approximately 3334 GEs and, using a clock frequency of 1MHz, 33 clock cycles to complete a single operation on 128-bit input. This corresponds to a throughput of 3787 Kb per second.

The downside of Chaskey is a worse security guarantee. An attacker can break Chaskey with time complexity of approximately $2^{128}/D$ if he obtains D plaintext/ciphertext pairs. If one is worried about the small security margin (7 out of 8 rounds have been broken, and even though this sounds bad, it actually doesn't mean that it's just about to fall apart), a proposal has been made to increase the number of rounds to 12 or 16 with a relatively small cost to the implementation size and efficiency (Mouha N. , 2015) which would increase the security margin.

When it comes to standardization, Chaskey has been ISO standardized as a lightweight MAC (International Organization for Standardization, 2019).

3.5 ASYMMETRIC CIPHERS

Asymmetric ciphers are also known as public key cryptography. They differ from the previously mentioned ciphers in that the key used to encrypt and decrypt data is not the same, hence the asymmetry. Often, asymmetric ciphers release a public key to the public (thus, not a secret value) which can be used for encrypting data. Only the owner of the

private key can then decrypt the data again. This can be helpful in certain scenarios where you don't want several devices to share a key, since a successful attack on a single key will compromise the entire system. Giving each device their own key can also result in problems since keeping track of which device uses which symmetric key is a hassle. It could also be that you simply don't know who will send you data in advance, making it impossible to share a key in advance.

The problem is that asymmetric ciphers are orders of magnitude slower than symmetric ciphers, making them less appealing in a lightweight cryptography scenario. Still, there have been made advances in recent years, proving that it is indeed possible to utilize asymmetric ciphers in constrained environments.

3.5.1 ECC

Elliptic curve cryptography has been the target of many researchers in search of an asymmetric cipher which could be used in constrained environments. It was actually proven already 10 years ago that ECC could fit onto an RFID chip (Hein, Wolkerstorfer, & Felber, 2008), where the implementation took up 15,000 GE. More recent work has reduced this number and increased the speed. Among these, Kurahatti (Kurahatti, 2018) managed to reduce the size to 8,580 GE and needs only 381 μW at 35.8 kHz to run. This means that chips such as the RFID tag can make use of asymmetric encryption, which in some applications is critical to security and where symmetric cryptography is not applicable.

3.5.2 NTRU

NTRU (NTRUOpenSourceProject, u.d.) is a lattice-based public key approach which has been accepted by the IEEE standardization as IEEE Standard 1363.1-2008. NTRU exists in two main variants: NTRUEncrypt which handles encryption and decryption and NTRUSign which handles digital signatures. The code can be compressed to around 8 kb, making it a contender for software-based asymmetric cryptography. It also has the added advantage of being based on security assumptions which do not break if the attacker has a quantum computer, as would be the case for e.g. RSA or ECC. NTRUSign for commercial use requires a license, but NTRUEncrypt can be used free of charge. NTRU claims to be the fastest public key crypto, beating both RSA and ECC.

NTRU has also been investigated for use in somewhat lightweight cryptography, namely the FPGAs. A paper by Kamal and Youssef (Kamal & Youssef, 2009) shows an implementation on an FPGA and achieves speeds of more than 130 Mb/s for both encryption and decryption. Unfortunately, the hardware area used is expressed in slices, not GEs, making comparisons difficult since the definition of a slice varies with the hardware.

A great downside to NTRUEncrypt is that it has been shown to have several vulnerabilities for certain, although unusual, parameter choices (Albrecht, Bai, & Ducas,

2016; Kirchner & Fouque, 2016). Furthermore, the standard signature scheme, NTRUSign, has been all but completely broken (Ducas & Nguyen, 2012).

3.5.3 MCELIECE

The McEliece family of schemes had its inception in 1978 (McEliece, 1978) and is based on the hardness of general decoding of a certain type of error correction code. The scheme, and a follow-up work by Neiderreiter (Niederreiter, 1986), didn't really gain much traction in the cryptographic community until quantum attacks on more popular schemes, such as RSA and ECC, were introduced (Shor, 1994). McEliece is not vulnerable to these types of attack and thus regained some research popularity resulting in a digital signature variant (Courtois, Finiasz, & Sendrier, 2001) along with further optimizations (Misoczki, Tillich, Sendrier, & Barreto, 2013). That, along with the fact that encryption and decryption can be computed using simple bit manipulations on short words, has made the McEliece family a serious contender for public key cryptography on constrained devices.

The main negative aspect of this family is that it requires quite large public keys (around 600 bytes using the most recent optimizations (Misoczki, Tillich, Sendrier, & Barreto, 2013)) and that, without careful implementation, is vulnerable to timing-based side-channel attacks.

An implementation of the encryption version of the scheme (Misoczki, Tillich, Sendrier, & Barreto, 2013) has been investigated in a somewhat lightweight setting, that is using FPGAs (Maurich & Güneysu, 2014). The authors report speeds of about 2.2 Mb/s for encryption and 0.36 Mb/s for decryption, however the number of slices needed are over an order of magnitude less than that required by NTRU.

Finally, we note that a version of McEliece has made it to round 3 of the NIST post-quantum cryptography standardization process[4].

[4] *https://classic.mceliece.org/nist.html*

4 SUMMARY

In the table below, we present an overview of our findings. An entry written as "-" denotes that the information was either not applicable or not obtainable. Note that throughput should not be taken as a fair number to compare with the other primitives. Most software throughput numbers are taken from FELICS (University of Luxembourg, 2017) and more precisely using the AVR architecture in scenario 2 – III: best running time for encrypting 128 bits using the CTR mode of operation. Throughput is measured in kilobit per second and always assumes a CPU of 1 MHz (numbers found using a different CPU speed have been scaled, which may not be entirely accurate as throughput is not exactly linear in CPU speed). Power usage numbers are not easy to find and depends heavily on CPU and hardware used. Those listed are based on the papers investigated and should be used indicatively only. Power numbers without a (*) are for CPU's running at 100 KHz. For (*) numbers, consult the section describing the primitive.

Not all variants of the primitives are mentioned. Most have multiple different settings, which yields different numbers. Where more than one variant is listed, we always list the minimum size as the first one and the maximum throughput as the other.

Note also that the hardware technologies used may differ from primitive to primitive (some use 0,13 μm technology and others 0,18 μm). We do not mention this in the table. This also makes comparisons slightly skewed.

Cryptographic primitive	*Ware orientation*	*Block size /Key size*	*Gate equiv. (GE)*	*Code size (bytes)*	*RAM (bytes)*	*Throughput (kbps)*	*Power usage*
Block ciphers							
AES	Software	128/128	2090	943	79	S: 37,3 (H: 566)	3,7 μA
SIMON	Hardware	48/96	763	196	55	S: 36,8 H: 150	-
SIMON	Hardware	128/128	1317	732	55	S: 21,37 H: 229	-
SPECK	Software	48/96	884	134	52	S: 59	-

Cryptographic primitive	*Ware orientation*	*Block size /Key size*	*Gate equiv. (GE)*	*Code size (bytes)*	*RAM (bytes)*	*Throughput (kbps)*	*Power usage*
						H: 120	
SPECK	Software	128/128	1396	396	52	S: 48 H: 121	-
KATAN64	Hardware	64/80	1054	272	18	25,1	555 nW
KTANTAN64	Hardware	64/80	688	-	18	25,1	555 nW
PRESENT	Hardware	64/80	1570	-	-	2000	5 µW
Grain80	Hardware	Stream/80	1294	-	-	10	1,09 µW
Grain80x16	Hardware	Stream/80	3239	-	-	160	2 µW
Grain128	Hardware	Stream/128	1857	-	-	10	1,6 µW
Grain128x32	Hardware	Stream/128	4617	-	-	320	3,4 µW
Salsa20	Software	Stream(min. 512)/128	-	1452	280	214	-
LEX	Software	Stream(min. 320)/128	-	1598	304	93	-
Hash functions							
KECCAK (SHA-3)	Software/ Hardware	64	1300	453	512	18,6	-
KECCAK (SHA-3) - parallel	Software/ Hardware	64	4900	-	-	400	27,6 µW
PHOTON-80	Hardware	80	865	-	-	28,2	2 µW
PHOTON-256	Hardware	256	4362	-	-	205	8,3 µW
SPONGENT-88	Hardware	88	738	-	-	8,1	1,9 µW
SPONGENT-256	Hardware	256	3281	-	-	114,3	7,5 µW

Cryptographic primitive	*Ware orientation*	*Block size /Key size*	*Gate equiv. (GE)*	*Code size (bytes)*	*RAM (bytes)*	*Throughput (kbps)*	*Power usage*
U-QUARK	Hardware	64	1379	-	-	14,7	2,44 μW
S-QUARKx16	Hardware	256	4640	-	-	500	8,39 μW
MACs							
Chaskey	Hardware	128/128	3334	624	80	S: 91,1 H: 3787	
Asymmetric ciphers							
ECC	Software	N/A	8580	-	-	N/A	381* μW
NTRU	Software	N/A	-	8000	-	130.000 (FPGA hardware)	
McEliece	Software	N/A	-	-	-	2180/358 (FPGA hardware)	

Table 1: Overview of all primitives and the key numbers found.

5 CONCLUSION

We investigated a large number of lightweight ciphers for use in constrained environments. Good progress has been made towards integrating cryptographic primitives into even the smallest hardware, making it possible to secure your data no matter how small the devices your application uses are. There is a large discrepancy between hardware and software ciphers in terms of performance and size, so choose carefully and consult with an expert if you're the least bit in doubt. Cryptography should be used with care, and even though a certain cipher is suggested in this whitepaper, it might be virtually useless if it is used in a wrong way or parameters are set incorrectly.

In general, we suggest using AES when applicable for keeping data confidential if your application is able to utilize symmetric ciphers. If integrity is also required, we recommend using AES in either GCM or CCM mode. Even though it may be a slower alternative, AES implementations have a larger probability of not being flawed since they have been under scrutiny many times over. Due to similar reasoning for choice of hash function, we recommend using SHA-3 if possible.

6 BIBLIOGRAPHY

Greenberg, A. (n.d.). (Forbes) Retrieved January 2018, from https://www.forbes.com/sites/andygreenberg/2012/01/30/hackers-demo-shows-how-easily-credit-cards-can-be-read-through-clothes-and-wallets/#2062d04f78a6

Biryukov, A., & Perrin, L. P. (2017). State of the Art in Lightweight Symmetric Cryptography.

University of Luxembourg. (2017, March 15). *FELICS*. Retrieved from https://www.cryptolux.org/index.php/FELICS

Dinu, D., Biryukov, A., Grossschädl, J., Khovratovich, D., Le Corre, Y., & Perrin, L. (2015). FELICS – Fair Evaluation of Lightweight Cryptographic Systems . *NIST Workshop on Lightweight Cryptography*, *128.*

Federal Information Processing Standards Publication. (2001). FIPS PUB 197, Specification for the Advanced Encryption Standard (AES). Gaithersburg, MD, USA.: National Institute of Standards and Technology.

International Organization for Standardization. (2005). ISO/IEC 18033-3:2005. Geneva, Switzerland.

McKay, K. A., Bassham, L., Turan, M. S., & Mouha, N. (2017, March). *Report on Lightweight Cryptography.* Retrieved from https://doi.org/10.6028/NIST.IR.8114

Sanu, M., Satpathy, S., Suresh, V., Anders, M., Kaul, H., Agarwal, A., . . . Krishnamurthy, R. (2015). 340 mV–1.1 V, 289 Gbps/W, 2090-gate nanoAES hardware accelerator with area-optimized encrypt/decrypt GF (2 4) 2 polynomials in 22 nm tri-gate CMOS. *IEEE Journal of Solid-State Circuits*, *50(4)*, pp. 1048-1058.

Matsui, M., & Murakami, Y. (2013). Minimalism of software implementation. *International Workshop on Fast Software Encryption*, (pp. 393-409). Berlin.

Moradi, A., Poschmann, A., Ling, S., Paar, C., & Wang, H. (2011). Pushing the Limits: A Very Compact and a Threshold Implementation of AES. *Advances in Cryptology – EUROCRYPT 2011.* Springer.

Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., & Wingers, L. (2013/2014). The SIMON and SPECK families of lightweight block ciphers. *Cryptology ePrint Archive.*

Osvik, D. A., Bos, J. W., Stefan, D., & Canright, D. (2010). Fast Software AES Encryption. *FSE*, *10*, pp. 75-93.

Abed, F., List, E., Lucks, S., & Wenzel, J. (2013). Cryptanalysis of the Speck Family of Block Ciphers. *IACR Cryptology ePrint Archive.*

Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., & Wingers, L. (2015). SIMON and SPECK: Block Ciphers for the Internet of Things. *IACR Cryptology ePrint Archive 2015.*

International Organization for Standardization. (2019). ISO/IEC 29192-2:2019. Geneva, Switzerland.

International Organization for Standardization. (2018). ISO/IEC 29167-21:2018. Geneva, Switzerland.

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

Bogdanov, A., & Rechberger, C. (2010). A 3-Subset Meet-in-the-Middle Attack: Cryptanalysis of the Lightweight Block Cipher KTANTAN. *Selected Areas in Cryptography.* Waterloo, Canada.

Bogdanov, A., Knudsen, L., Leander, G., Paar, C., Poschmann, A., Robshaw, M., . . . Vikkelsoe, C. (2007). PRESENT: An Ultra-Lightweight Block Cipher. *CHES.* Vienna, Austria.

ECRYPT. (n.d.). *eSTREAM: the ECRYPT Stream Cipher Project*. (ECRYPT) Retrieved 2020, from https://www.ecrypt.eu.org/stream/

Hell, M., Johansson, T., & Meier, W. (2007). Grain: a stream cipher for constrained environments. *International Journal of Wireless and Mobile Computing 2*, *1*, pp. 86-93.

Good, T., & Benaissa, M. (2008). Hardware performance of eStream phase-III stream cipher candidates. *State of the Art of Stream Ciphers Workshop*, (pp. 163-173).

De Canniere, C., & Preneel, B. (2005). TRIVIUM specifications. *eSTREAM, ECRYPT stream Cipher Project.*

Bernstein, D. (2008). The Salsa20 family of stream ciphers. *Lecture Notes in Computer Science*, (pp. 84-97).

Bernstein, D. (2008). ChaCha, a variant of Salsa20. *Workshop Record of SASC*, *8*, pp. 3-5.

Eisenbarth, T., & Kumar, S. (2007). A survey of lightweight-cryptography implementations. *IEEE Design & Test of Computers 24.6.*

Y. Nir, Dell, A. Langley, Google. (2018). RFC 8439 - ChaCha20 and Poly1305 for IETF Protocols. Internet Research Task Force (IRTF).

Biryukov, A. (2005). A New 128-bit Key Stream Cipher LEX. *eSTREAM, ECRYPT Stream Cipher Project.*

Dunkelman, O., & Keller, N. (2013). Cryptanalysis of the Stream Cipher LEX. *Designs, Codes and Cryptography*.

Federal Information Processing Standards Publication. (2015). FIPS PUB 180-4, Secure Hash Standard (SHS). Gaithersburg, MD, USA.: National Institute of Standards and Technology.

International Organization for Standardization. (2018). ISO/IEC 10118-3:2018. Geneva, Switzerland.

Bertoni, G., Daemen, J., Peeters, M., & Van Assche, G. (2009). Keccak sponge function family main document. *Submission to NIST*.

Pessl, P., & Hutter, M. (2013). Pushing the limits of SHA-3 hardware implementations to fit on RFID. *International Workshop on Cryptographic Hardware and Embedded Systems.* Berlin.

Gio, J., Peyrin, T., & Poschmann, A. (2011). The PHOTON Family of Lightweight Hash Functions. *Advances in Cryptology–CRYPTO 2011.*

International Organization for Standardization. (2016). ISO/IEC 29192-5:2016 . Geneva, Switzherland.

Bogdanov, A., Knežević, M., Leander, G., Toz, D., Varıcı, K., & Verbauwhede, I. (2011). SPONGENT: A lightweight hash function. *Cryptographic Hardware and Embedded Systems–CHES 2011.*

Aumasson, J.-P., Henzen, L., & Naya-Plasencia, M. (2010). Quark: A lightweight hash. *CHES*, *6225.*

Krawczyk, H., Canetti, R., & Bellare, M. (1997). HMAC: Keyed-hashing for message authentication. *RFC 2104.*

Song, J., Poovendran, R., Lee, J., & Iwata, T. (2006). The aes-cmac algorithm. *RFC 4493.*

McGrew, D., & Viega, J. (2004). The Galois/counter mode of operation (GCM). *Submission to NIST Modes of Operation Process 20.*

Dworkin, M. J. (2007). Recommendation for block cipher modes of operation: Galois/Counter Mode (GCM) and GMAC. *Special Publication (NIST SP)-800-38D.*

Wu, H., & Huang, T. (2016). The JAMBU Lightweight Authentication Encryption Mode (v2. 1).

Engels, D. W., Saarinen, M.-J., Schweitzer, P., & Smith, E. M. (2011). The Hummingbird-2 Lightweight Authenticated Encryption Algorithm. *RFIDSec 11*, (pp. 19-31).

Payrin, T., Sim, S. M., Wang, L., & Zhang, G. (2015). Cryptanalysis of JAMBU. *International Workshop on Fast Software Encryption.* Berlin.

Zhang, K., Ding, L., & Guan, J. (2012). Cryptanalysis of Hummingbird-2. *IACR Cryptology ePrint Archive 2012.*

Mouha, N. (2017, May). *Chaskey*. Retrieved January 2018, from https://mouha.be/chaskey/

Lan, J., Zhou, J., & Liu, X. (2016). An area-efficient implementation of a Message Authentication Code (MAC) algorithm for cryptographic systems. *Region 10 Conference (TENCON)*, (pp. 1977-1979).

Mouha, N. (2015). a MAC Algorithm for Microcontrollers -- Status Update and Proposal of Chaskey-12. *IACR Cryptology ePrint Archive.*

International Organization for Standardization. (2019). *ISO/IEC 29192-6:2019.* Geneva, Switzerland.

Hein, D. M., Wolkerstorfer, J., & Felber, N. (2008). ECC Is Ready for RFID-A Proof in Silicon. *Selected Areas in Cryptography*, *5381*, pp. 401-413.

Kurahatti, N. G. (2018). Design and Implementation of ECC-Based RFID Tag for Wireless Communications on FPGAs. *International Proceedings on Advances in Soft Computing, Intelligent Systems and Applications*, (pp. 415-430). Singapore.

NTRUOpenSourceProject. (n.d.). Retrieved January 19, 2018, from https://github.com/NTRUOpenSourceProject/ntru-crypto

Kamal, A. A., & Youssef, A. M. (2009). An FPGA implementation of the NTRUEncrypt cryptosystem. *Microelectronics (ICM)*, (pp. 209-212).

Albrecht, M. R., Bai, S., & Ducas, L. (2016). A subfield lattice attack on overstretched NTRU assumptions - cryptanalysis of some FHE and graded encoding schemes. *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference.* Santa Barbara, California.

Kirchner, P., & Fouque, P.-A. (2016). omparison between subfield and straightforward attacks on NTRU. IACR Cryptology ePrint Archive, 2016:717.

Ducas, L., & Nguyen, P. Q. (2012). Learning a Zonotope and More: Cryptanalysis of NTRUSign Countermeasures. Advances in Cryptology - {ASIACRYPT} 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security. Beijing, China.

McEliece, R. J. (1978). A Public-Key Cryptosystem Based On Algebraic Coding Theory. *Deep Space Network Progress Report*(44), 114-116.

Niederreiter, H. (1986). Knapsack type cryptosystems and algebraic coding theory. *Problems of Control and Information Theory. Problemy Upravlenija i Teorii Informacii*(15), 159-166.

Shor, P. W. (1994). Algorithms for quantum computation: Discrete logarithms and factoring. *35th Annual Symposium on Foundations of Computer Science.* Santa Fe, New Mexico.

Misoczki, R., Tillich, J.-P., Sendrier, N., & Barreto, P. S. (2013). MDPC-McEliece: New McEliece variants from moderate density parity-check codes. *IEEE International Symposium on Information Theory.* Istanbul, Turkey.

Maurich, I. v., & Güneysu, T. (2014). Lightweight code-based cryptography: QC-MDPC McEliece encryption on reconfigurable devices. *Design, Automation & Test in Europe Conference & Exhibition (DATE).* Dresden, Germany.

Courtois, N., Finiasz, M., & Sendrier, N. (2001). How to achieve a McEliece-based digital signature scheme. Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security. Gold Coast, Australia.

THE ALEXANDRA INSTITUTE
IT CITY OF KATRINEBJERG
Aabogade 34 ▪ DK-8200 Aarhus N
+45 70 27 70 12

UNIVATE
Njalsgade 76, 3rd floor ▪ DK-2300 Copenhagen S
+45 70 27 70 91

The Alexandra Institute helps private and public organisations develop innovative solutions, products and services based on state-of-the art IT research. Our mission is to enable Danish companies realise the potential of new technology.