Bachelor thesis
Computing Science

Radboud University

Comparison of the second round candidates of the NIST lightweight cryptography competition

Author:
Eline Bovy
s1007567

First supervisor/assessor:
Prof.dr. Joan Daemen
joan@cs.ru.nl

Second supervisor/assessor:
Dr.ir. Bart Mennink
b.mennink@cs.ru.nl

January 15, 2020

Abstract

This thesis gives an overview of the second round candidates of the lightweight cryptography competition organized by the National Institute of Standards and Technology. It contains three comparisons. First, a global comparison of the schemes, based on the underlying primitives and the modes of operation. Second, a comparison of the number of primitive calls for different encryption situations. Third, a comparison of the number of bitwise operations for single encryption with a new key and nonce. The first two comparisons are based on all second round candidates, the third comparison focuses on six schemes: Ascon-128, Ascon-128a, Gimli-24-Cipher, Xoodyak, Subterranean-SAE and SKINNY-AEAD M1.
The global comparison gives an overview of important basic information, which can be used by the cryptographic community for their research of the schemes. The combination of the last two comparisons gives an efficiency estimation.

Contents

Chapter 1

Introduction

1.1 The NIST lightweight cryptography competition

On August 27, 2018, the U.S. National Institute of Standards and Technology (NIST) published the document: *Submission Requirements and Evaluation Criteria for the Lightweight Cryptography Standardization Process* [35]. This document contains the requirements and the evaluation criteria of submissions to the NIST competition for a lightweight cryptography standard. NIST is interested in a new standard, because NIST's current standards do not seem suited for the resource-constrained devices of emerging areas like healthcare and the Internet of Things.
NIST received 57 submissions, of which 56 were accepted as first round candidates. Of these 56 candidates, NIST selected 32 for the second round, as announced on August 30, 2019. The selection of these schemes is for a part based on the efforts of the cryptographic community. Cryptographers analyze the schemes and might attempt to break the security claims. A successful attempt can eliminate a scheme from the competition. An unsuccessful attempt increases the confidence in a scheme's security.

1.2 Our contribution

In this thesis, we analyze the second round candidates of the NIST competition. We compare the submissions in three ways: on a global level, on the number of primitive calls and on the number of bitwise operations. The first two comparisons look at all schemes, the third looks only at Ascon-128, Ascon-128a, Gimli-24-Cipher, Xoodyak, Subterranean-SAE and SKINNY-AEAD M1. The combination of the number of primitive calls and the number of bitwise operations results in an estimation of the efficiency of the schemes.

1.3 Related work

In October 2019 the *Status Report on the First Round of the NIST Lightweight Cryptography Standardization Process* [38] was published. This report contains a categorization of the 56 first-round candidates of the NIST competition, based on the primitive type and whether the submission contained a hashing functionality. Aside from this, there are no global comparisons of the candidates of the NIST lightweight cryptography competition.
Sasaki has given a presentation [37] on the transition from the first to the second round of the

competition. This presentation contains a survival rate of the different structure choices and of the submissions with hash functionality. It also analyzes the frequency of used primitives and summarizes the comments given during the first round.
In the article *The SKINNY Family of Block Ciphers and its Low-Latency Variant MANTIS* [12] an efficiency analysis strategy, similar to the one in this thesis, is used. Three versions of SKINNY are compared to nine other primitives, including AES-128/128 and AES-128/256, which are used in multiple second round candidate schemes, and SIMON-128/128, which is used in the primitives of the primary member of the Oribatida submission. This comparison is based, among others, on the number of bitwise operations per bit per round. The analysis in the article differs from the analysis in this thesis, because we look at the number of operations in total instead of per bit per round, and we look at the entire AE scheme and not only the primitive. There are also a few differences in the way of counting the number of operations. For example, [12] does not count the addition of constants, whereas we do.
Although [12] analyzes a few primitives that are used in the NIST competition, there is no article yet that purposely applies this type of efficiency analysis for the AE schemes of the NIST competition.
The submissions Ascon [28] and Gimli [15] both contain an efficiency analysis based on cycles per byte. Subterranean 2.0 [26] contains an efficiency analysis based on memory consumption.

1.4 Chapter overview

Chapter 2 contains the necessary information to understand this thesis. In Chapter 3 the second round candidates of the NIST competition are compared on a global level, in Chapter 4 on the number of primitive calls and in Chapter 5 on the number of bitwise operations. Finally, Chapter 6 concludes this thesis.

Chapter 2

Preliminaries

As mentioned in the Introduction, this research compares the second round candidate schemes of the NIST competition for a lightweight cryptography standard. The global comparison is based on the underlying primitive and the mode of operation. Additionally, the schemes are compared on the number of primitive calls and bitwise operations.
This chapter gives more information about the main type of cryptographic functionality of the NIST competition (Section 2.1) and the NIST competition itself (Section 2.2). The underlying primitive and the mode of operation are explained in Section 2.3 and Section 2.4, respectively.

2.1 Authenticated encryption with associated data

Authenticated encryption with associated data (AEAD) is an cryptographic functionality which assures both the confidentiality and the authenticity of the data. Present-day authenticated encryption essentially always uses associated data, therefore AEAD is often abbreviated to AE.
AE takes as input four byte-strings: a key, a nonce, associated data and a plaintext. It produces as output two byte-strings: a ciphertext and a tag. The ciphertext provides the confidentiality, the tag provides the authenticity. The reverse of AE, authenticated decryption, takes as input a key, a nonce, associated data, a ciphertext and a tag. The output only reveals the decrypted plaintext, if the calculated tag is the same as the tag from the input. If the tags do not match, an error is thrown. See Figure 2.1.

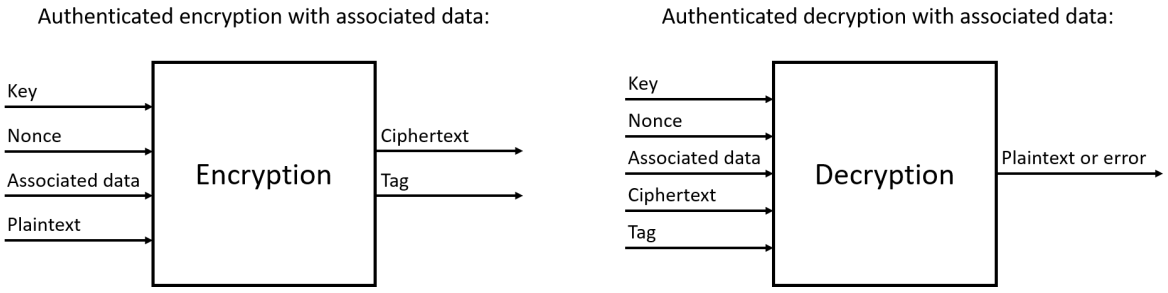

Figure 2.1: High-level overview of AE

2.1.1 Single encryption versus session encryption

Within authenticated encryption, we make distinction between two forms of encryption: single encryption and session encryption. Single encryption is the encryption of one message per key-nonce-combination. Session encryption is the encryption of multiple messages with the same key-nonce-combination. The tags produced in session encryption, authenticate all message received under that combination so far.

2.2 The NIST competition

Each submission to the NIST competition for a lightweight cryptographic standard has to contain a family of AE schemes. Such a family is allowed to have between 1 and 10 members. The submitters must indicate one primary member. Additionally, a family of hash functions can be submitted, with at most 10 members. In this research we limit ourselves to the AE families.
NIST published requirements to which the submissions must adhere in order to compete [35]. For the primary member of the AE family some additional requirements are defined. Below, the requirements for the AE schemes are specified.

NIST has defined the following requirements for the sizes of the input and output byte-strings of the AE schemes:

- The key is of fixed length ≥ 128 bits.
- The nonce is of fixed length.
- The plaintext is of variable length.
- The associated data is of variable length.
- The ciphertext is of variable length.

The AE schemes must accept all inputs that satisfy these length requirements. For the primary member of the AE family, some additional length requirements are defined:

- The nonce is of length ≥ 96 bits.
- The tag is of length ≥ 64 bits.

NIST also defined requirements concerning the security of the AE schemes. The AE schemes must ensure confidentiality of the plaintexts and integrity of the ciphertexts. The offline attack complexity must be at least 2^{112} computations, i.e. primitive calls, on a classical computer in a single-key setting. For the primary member, the online attack complexity must be at least $2^{50} - 1$ bytes. Additionally, the AE schemes must maintain their security as long as the nonce is unique. In case of nonce re-use, no security level is specified.

2.3 Underlying primitive

A cryptographic primitive is a well-established low-level function, that acts as a, typically fixed length, building block for larger schemes. Generally there are no security proofs for

cryptographic primitives. The expected security comes from thorough examination by the cryptographic community: if many people attempt to break a primitive, but no one succeeds, then the confidence that the primitive is secure increases.
There are four main types of primitives in the second round schemes of the NIST competition: permutations, stream ciphers, block ciphers and tweakable block ciphers. We consider the forkcipher [6] as part of the tweakable block cipher type. Primitives can be compared on multiple characteristics. In this research we only look at the block size, key size and tweak size. Not all characteristics are defined for all primitive types. Permutations do not use a key and only tweakable block ciphers use a tweak. In the next four subsections, the main primitive types used in the NIST competition are explained. Additionally, we explain the different sizes of the four primitive types in subsection 2.3.5.

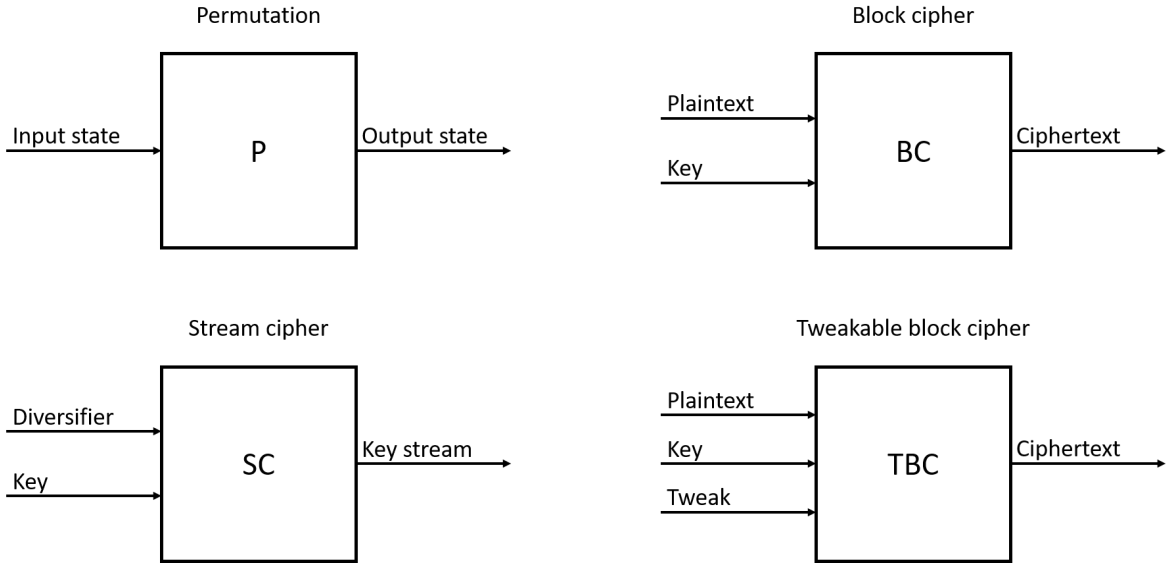

Figure 2.2: High-level overview of four primitive types

2.3.1 Permutation

A permutation is a bijective function from a set S to itself, where typically $S = \{0,1\}^b$. In other words, a permutation transforms its input in an invertible way. A permutation has one b-bit string input, the input state, and one b-bit string output, the output state. See Figure 2.2.

2.3.2 Block cipher

A block cipher is a family of permutations, indexed by a key. A block cipher has two input strings, the n-bit plaintext and the k-bit key. It has one output string, the n-bit ciphertext. See Figure 2.2.

2.3.3 Tweakable block cipher

A tweakable block cipher is a family of permutations, indexed by a key and a tweak. In other words, a tweakable block cipher is a family of block ciphers, indexed by a tweak. A tweakable

block cipher has three input strings, the n-bit plaintext, the k-bit key and the w-bit tweak. It has one output string, the n-bit ciphertext. See Figure 2.2.

2.3.4 Stream cipher

A stream cipher has two input strings, the d-bit diversifier and the k-bit key, and one variable length output string, the key stream. The ciphertext is generated by bitwise addition of the plaintext and the output stream. See Figure 2.2.

2.3.5 Sizes of the underlying primitive

As can be read in the previous four subsections, the different primitive types get different types of input. In Chapter 3 we use the sizes of these inputs to compare the schemes. Therefore, we summarize the input sizes per primitive type in Table 2.1. The letters used in the table correspond to the letters used in the pervious four subsections.
Besides the input sizes, we also look at the state sizes. The state size represents the amount of data the primitive operates on. For permutations, block ciphers and tweakable block ciphers, the state size is the sum of the sizes of their inputs. For the stream cipher, the state size is the sum of its registers.

Primitive type	block size	key size	tweak size	width	state size
Permutation	-	-	-	b	b
Block cipher	n	k	-	-	$n+k$
Tweakable block cipher	n	k	w	-	$n+k+w$
Stream cipher	-	k	-	-	$\lvert$ registers $\rvert$

Table 2.1: The sizes of the primitive types.

2.4 Mode of operation

A mode of operation is an algorithm that uses its underlying primitive. Such a mode can occur at different levels in a scheme. Modes on a high level can use other modes on lower levels. For example, the Encrypt-then-MAC mode is a high level mode. As the name implies, this mode first encrypts the input and then produces a Message Authentication Code (MAC). How the input is encrypted and how the MAC is produced, is determined by lower level modes.
In this research, we look at the type and the capacity of the modes, whether the modes are inverse free and parallelizable and whether the mode uses rekeying. As with the primitives, not all characteristics are defined for all modes.
An important construction for this thesis is the duplex construction. This construction is used by five of the schemes in the comparison of the number of bitwise operations in Chapter 5. These schemes use the duplex construction, or a variant, as part of a higher-level mode of operation. Therefore, we treat this construction in detail below.

2.4.1 The duplex construction

The duplex construction uses a fixed-length permutation as underlying primitive. A keyed block cipher can also function as primitive for this construction, because a block cipher is a family of permutations, indexed by a key. The same goes for a tweaked and keyed tweakable block cipher.
The duplex construction has a b-bit state, which consists of an r-bit outer part and a c-bit inner part, with $b = r + c$, where r stands for rate and c for capacity. The input of the construction, σ, is split in blocks of r bits. The following three steps are repeated for each σ block:

1. The σ block is padded (if necessary) and XOR-ed with the outer part of the state.
2. The permutation is applied to the entire state.
3. The outer part of the state is copied and truncated to the length of the σ block to form the corresponding output block Z.

See Figure 2.3. The inner part, the capacity, is not involved in the processing of the input and output blocks. Due to this, a user cannot directly modify this part of the state. The security of the duplex construction depends on the capacity.

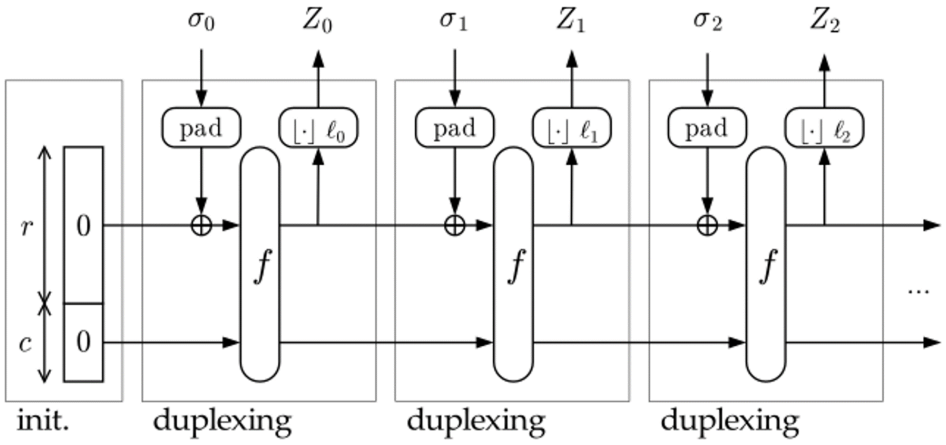

Figure 2.3: The duplex construction [16]

The relations between the in- and outputs of this construction (σ and Z) and the in- and outputs of the AE algorithms (key, nonce, associated data and plaintext, and ciphertext and tag) depend on the mode of operation used on top of this construction.

Chapter 3

Global comparison

This chapter provides a global comparison of the basic information of the second round candidate schemes of the NIST competition for a lightweight cryptography standard. In Section 3.1 we explain the attributes used for the comparison. The results are presented in Section 3.2 and interpreted in Section 3.3.

3.1 Attributes

We compare the 32 second round candidates on ten attributes divided into two main topics. First, we look at the underlying primitives of the schemes. Within this topic, we look at the primitive type, the block size, the key size and the tweak size. Second, we look at the modes of operation. Within this topic, we look at the mode of operation, the parallelizability and the capacity. Additionally, we look whether the inverse primitive is required and rekeying is used. The values of the ten attributes are not necessarily the same for different members of the same submission, therefore we look at each member separately.
We represent the comparison in four tables, split by the underlying primitive type: one table for permutation schemes, one for block cipher schemes, one for tweakable block cipher schemes and one for other schemes. The table for other schemes consists of the stream cipher scheme of Grain-128AEAD [31], the single encryption scheme of Saturnin [19] and the schemes of Spook [14], which use two underlying primitives.
The first two columns of the tables are solely used for identification of a scheme. The other nine columns are used for the remaining attributes mentioned above, so excluding the primitive type.
Below, the columns of the tables are briefly explained.

Identification:

- **Submission**: The name of the submission.
- **Scheme**: The name of the scheme.

Underlying primitive:

- **Name**: The name of the underlying primitive. It is possible for a scheme to use multiple underlying primitives.

- **bs**: The block size of the underlying primitive. Permutations have no block size, so we use the width. Stream ciphers have neither a block size nor a width, so we leave this open.
- **ks**: The key size of the underlying primitive.
- **ts**: The tweak size of the underlying primitive.

Mode of operation:

- **Name**: The name of the mode of operation per member. It is possible for a scheme to use multiple modes of operation.
- **if**: Whether the mode of operation is inverse free or not. A mode is inverse free, if it does not require the inverse version of the underlying primitive. It is possible for a scheme to use multiple modes of operation and hence to be partially inverse free. This is denoted by hybrid. This attribute is relevant for all primitive types, i.e. permutations, block ciphers, tweakable block ciphers and stream ciphers.
- **p**: Whether the mode of operation is parallelizable or not. A mode is parallelizable, if it is possible to execute multiple computations at the same time, i.e. in parallel. If a computation requires the result of previous computations, the mode is not parallelizable, but sequential. It is possible for a scheme to use multiple modes of operation and hence to be partially parallelizable. This is denoted by hybrid. This attribute is relevant for all primitive types, i.e. permutations, block ciphers, tweakable block ciphers and stream ciphers.
- **rk**: Whether the mode of operation uses rekeying or not. A mode uses rekeying, if different primitive calls are made with different keys. It is possible for a scheme to use multiple modes of operation and hence to partially use rekeying. This is denoted by hybrid. This attribute is only relevant for keyed primitive types, i.e. block ciphers, tweakable block ciphers and stream ciphers.
- **c**: The size of the inner part of the state. This attribute is only relevant for sponge-based modes of operation.

3.2 Results

On the next pages four tables are displayed. These tables only contain the primary member of each submission. For the full comparison, see Appendix A. The tables are first sorted on parallelizability and then on whether the inverse primitive is required. The rest is done alphabetically.

Please keep in mind the following remarks:

- KNOT [10], TinyJAMBU [39] and Grain128-AEAD [31] do not clearly define the name of their underlying primitive, hence we left this open.
- We consider the underlying primitive of Grain-128AEAD to be the NSFR and the LFSR together. All other attributes are based upon this.

- We altered the names of underlying primitives used by multiple submission, to enable better detection of similarities among schemes.

Overall, we aimed to summarize all schemes as accurately as possible. Minor differences between this comparison and the original specifications may have occurred due to differences in convention and presentation.

Table 3.1: Primary members with permutation primitive

Identification		**Underlying Primitive**				**Mode of operation**				
Submission	**Scheme**	**Name**	**bs**	**ks**	**ts**	**Name**	**if**	**p**	**rk**	**c**
Parallelizable										
Elephant [17]	Dumbo	Spongent-π[160]	160	-	-	Encrypt-then-MAC Even-Mansour	Yes	Yes	-	-
Sequential										
ACE [1]	ACE-AE-128	ACE	320	-	-	Duplex	Yes	No	-	256
Ascon [28]	Ascon-128	Ascon-p	320	-	-	Monkeyduplex	Yes	No	-	256
DryGASCON [36]	DryGASCON128	GASCON$_{C5}$	320	-	-	DrySponge	Yes	No	-	448
Gimli [15]	Gimli-24-Cipher	Gimli permutation	384	-	-	Duplex	Yes	No	-	256
ISAP [27]	ISAP-K-128A	Keccak-p[400]	400	-	-	Encrypt-then-MAC Sponge	Yes	No	-	256
KNOT [10]	KNOT-AEAD(128,256,64)		256	-	-	Monkeyduplex	Yes	No	-	192
ORANGE [24]	ORANGE-Zest	PHOTON256	256	-	-	ORANGE-Zest	Yes	No	-	128
Oribatida [18]	Oribatida-256-64	SimP-256-4 SimP-256-2	256	-	-	Duplex	Yes	No	-	128
PHOTON-Beetle [9]	PHOTON-Beetle-AEAD[128]	PHOTON256	256	-	-	Beetle	Yes	No	-	128
SPRAKLE (SCHWAEMM and ESCH) [11]	SCHWAEMM-256-128	SPARKLE384	384	-	-	SCHWAEMM-256-128	Yes	No	-	128
SPIX [4]	SPIX	sLiSCP-light-[256]	256	-	-	Monkeyduplex	Yes	No	-	192
SpoC [3]	SpoC-64_sLiSCP-light-[192]	sLiSCP-light-[192]	192	-	-	SpoC-64	Yes	No	-	128
Subterranean 2.0 [26]	Subterranean-SAE	Subterranean round function	257	-	-	Duplex	Yes	No	-	225
WAGE [2]	WAGE-AE-128	WAGE	259	-	-	Duplex	Yes	No	-	195
Xoodyak [25]	Xoodyak	Xoodoo[12]	384	-	-	Cyclist	Yes	No	-	192

Table 3.2: Primary members with block cipher primitive

Identification			**Underlying Primitive**				**Mode of operation**				
Submission		**Scheme**	**Name**	**bs**	**ks**	**ts**	**Name**	**if**	**p**	**rk**	**c**
Parallelizable											
Pyjamask	[29]	Pyjamask-128-AEAD	Pyjamask-128	128	128	-	OCB	No	Yes	No	-
Sequential											
COMET	[30]	COMET-128_AES-128/128	AES-128/128	128	128	-	COMET	Yes	No	Yes	-
GIFT-COFB	[8]	GIFT-COFB	GIFT-128	128	128	-	COFB	Yes	No	No	-
HYENA	[22]	HYENA	GIFT-128	128	128	-	HYFB	Yes	No	No	-
mixFeed	[23]	mixFeed	AES-128/128	128	128	-	Minimally Xored Feedback Mode	Yes	No	Yes	-
SAEAES	[34]	SAEAES128_64_128	AES-128/128	128	128	-	SAEB	Yes	No	No	192
SUNDAE-GIFT	[7]	SUNDAE-GIFT-96	GIFT-128	128	128	-	SUNDAE	Yes	No	No	-
TinyJAMBU	[39]	TinyJAMBU-128		128	128	-	TinyJAMBU	Yes	No	No	96
Hybrid											
Saturnin	[19]	Saturnin-CTR-Cascade	Saturnin	256	256	-	Encrypt-then-MAC Counter mode Cascade construction	Yes	Hybrid: Yes No	Hybrid: No Yes	-

Table 3.3: Primary members with tweakable block cipher primitive

Identification		**Underlying Primitive**				**Mode of operation**				
Submission	**Scheme**	**Name**	**bs**	**ks**	**ts**	**Name**	**if**	**p**	**rk**	**c**
Parallelizable - inverse free										
Lotus-AEAD and LOCUS-AEAD [21]	TweGIFT-64_LOTUS-AEAD	TweGIFT-64	64	128	4	LOTUS-AEAD	Yes	Yes	Yes	-
Parallelizable - not inverse free										
ForkAE [5]	PAEF-ForkSkinny-128-288	ForkSkinny-128-288	128	128	160	PAEF	No	Yes	No	-
SKINNY-AEAD/ SKINNY-HASH [13]	SKINNY-AEAD M1	SKINNY-128-384	128	128	256	ΘCB3	No	Yes	No	-
Sequential										
ESTATE [20]	ESTATE_TweAES-128	TweAES-128	128	128	4	MAC-then-Encrypt FCBC* OFB	Yes	No	No	-
Romulus [33]	Romulus-N1	SKINNY-128-384	128	128	256	Romulus-N	Yes	No	No	-

Table 3.4: Other primary members

Identification		**Underlying Primitive**				**Mode of operation**				
Submission	**Scheme**	**Name**	**bs**	**ks**	**ts**	**Name**	**if**	**p**	**rk**	**c**
Grain-128AEAD [31]	Grain-128AEAD		-	128	-	Grain-128AEAD	Yes	No	No	-
Spook [14]	Spook[128,512,su]	Shadow-512 Clyde-128	512 128	- 128	- 127	S1P	No	No	No	256

3.3 Interpretation

The interpretation of the results is based only on the primary members, to ensure that each submission has an equal weight. In the next three subsections we look at the primitive type, at the block size and width, and whether the mode of operation is inverse free and parallelizable. In subsection 3.3.4 we discuss what stands out in the remaining attributes.

3.3.1 Primitive type

Half of the second round candidate schemes are permutation based. Block cipher schemes take up a bit more than a quarter. There is only one scheme that uses a stream cipher primitive. See Figure 3.1.

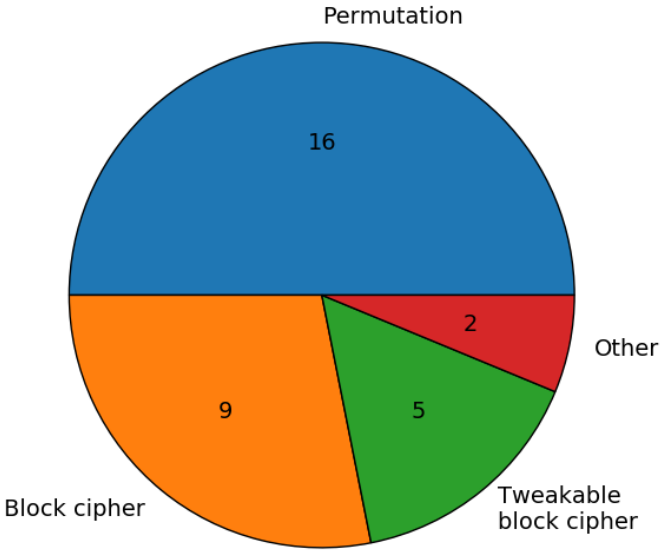

Figure 3.1: Number of primary members per primitive type

3.3.2 Block size, width and state size

This subsection focuses on the block size, width and state size. All three are characteristics of the underlying primitive. Grain-128AEAD has neither a block size nor a width and Spook uses two different underlying primitives. These schemes are not considered in this subsection. Both the block cipher schemes and the tweakable block cipher schemes have a block size of 128 bits, for all except one. The width of permutation schemes is generally (all but two) larger than the block size or width of schemes with other primitives. See Figure 3.2.

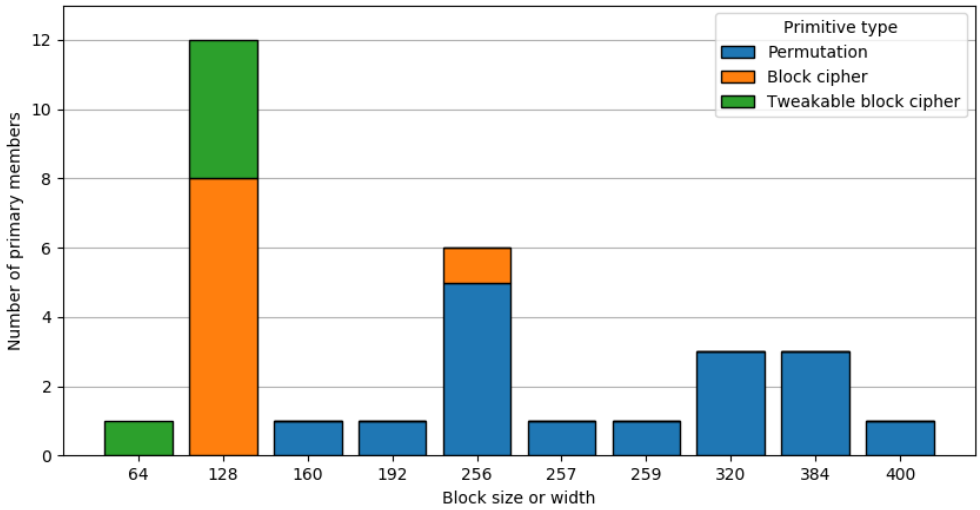

Figure 3.2: Number of primary members per primitive type per block size (block ciphers and tweakable block ciphers) or width (permutations and stream ciphers). Grain-128AEAD and Spook are excluded.

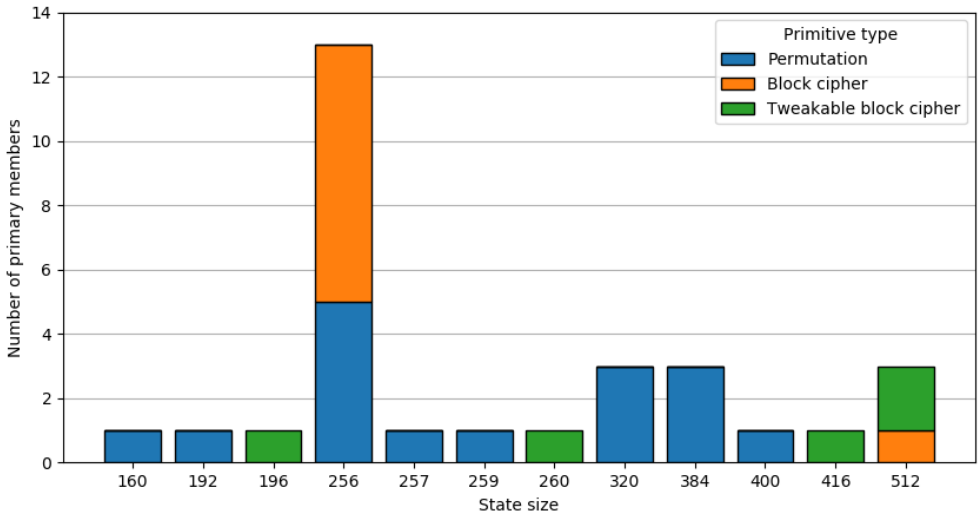

Figure 3.3: Number of primary members per primitive type per state size. Grain-128AEAD and Spook are excluded.

Looking at the state size gives quite a different picture (see Figure 3.3). Although the block cipher schemes are still centered at one value, all but one have a width of 256 bits, the tweakable block cipher schemes are now much more distributed. Furthermore the state size of permutation schemes is no longer generally larger than the state size of schemes with other primitives. In fact the lowest state sizes belong to permutation schemes.

3.3.3 Inverse free and parallelizable

This subsection focusses on two attributes of the mode of operation, namely whether it is inverse free and parallelizable. Saturnin is hybrid in terms of parallelizability and is therefore not considered in this subsection.
Most second round candidate schemes are inverse free and sequential. All permutation schemes are inverse free. The inverse free, non-inverse free ratio of the block cipher schemes (7:1 = 28:4) is very similar to that of all schemes together (27:4). See Figure 3.4.

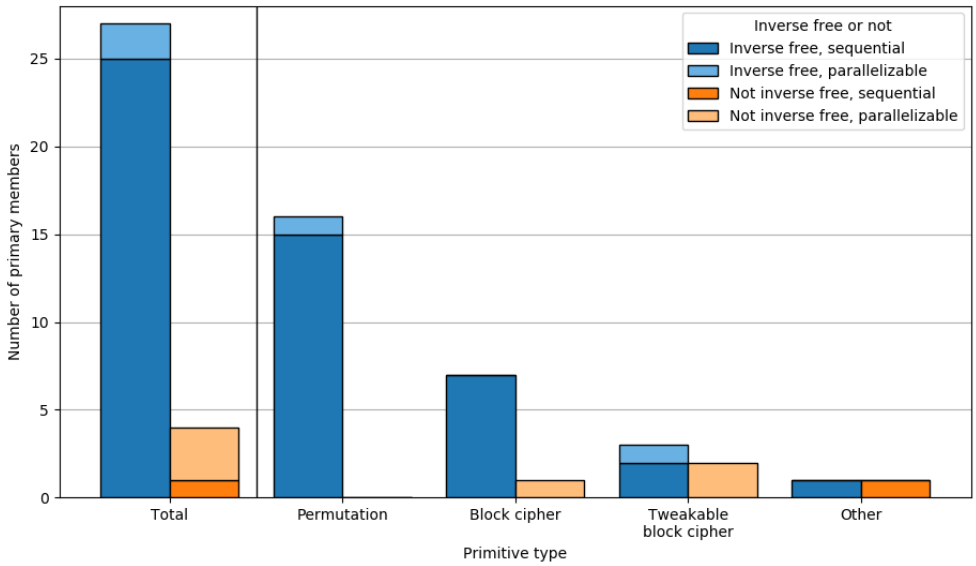

Figure 3.4: Number of (non-)inverse free, (non-)parallelizable primary members per primitive type. Saturnin is excluded.

3.3.4 Other observations

In the remaining attributes there are two things that stand out. First, all schemes with a keyed primitive, except for Saturnin, use a key size of 128 bits. Second, none of the non-inverse free schemes uses rekeying.

Chapter 4

Comparison of the number of primitive calls

In this chapter, we provide a comparison of the number of primitive calls used during different forms of encryption. We explain the method in Section 4.1. The results are presented in Section 4.2 and interpreted in Section 4.3.

4.1 Method

The calculation of the number of primitive calls is dependent on the number of bytes of plaintext and of associated data. Some schemes handle messages with empty plaintext or associated data differently, therefore we assume that the number of bytes of plaintext and of associated data is always greater than zero.
There are four encryption situations to consider:

(A) Single encryption with a new key and a new nonce.
(B) Single encryption with a used key and a new nonce.
(C) Session encryption with a new key and a new nonce.
(D) Session encryption with a used key and a new nonce.

To cover all these situations, we split the calculation of the number of primitive calls in three. Calculation 1: the number of primitive calls used to process a new key and nonce. Calculation 2: the number of primitive calls used to process a used key and a new nonce. These two calculations do not depend on the number of bytes of plaintext and of associated data. Calculation 3: the number of primitive calls used to encrypt a message, after the key and nonce are processed. This calculation does depend on the number of bytes of plaintext and of associated data. Some schemes do not explicitly process the nonce and key, but only use them during the encryption. For these schemes only the third calculation is relevant.
The four above mentioned situations can be calculated as follows:

(A) Calculation 1 + Calculation 3
(B) Calculation 2 + Calculation 3
(C) Calculation 1 + Calculation 3 separately for each message in the session.
(D) Calculation 2 + Calculation 3 separately for each message in the session.

4.2 Results

We present the results in table format. Similar to the tables of the global comparison in Chapter 3, the primitive type is used to split the comparison in four smaller tables: one for permutation schemes, one for block cipher schemes, one for tweakable block cipher schemes and one for other schemes. The tables are first sorted on parallelizability and then on whether the inverse primitive is required. The rest is done alphabetically.
Below, the columns of the tables are briefly explained:

- **Submission**: The name of the submission.
- **Scheme**: The name of the scheme.
- **nk, nn**: The number of primitive calls used to process a new key and nonce.
- **sk, nn**: The number of primitive calls used to process a used key and a new nonce.
- **Message**: The number of primitive calls used to encrypt a message, after the key and nonce are processed.

The first two columns are solely used for identification of the scheme. The other three columns are the three calculations explained in section 4.1.

Please keep in mind the following remarks:

- Some schemes use their underlying primitive with different numbers of rounds. This is denoted by (r) where r is the number of rounds. E.g. ISAP-K-128A which, to process a new key and a new nonce, makes 127 calls to its primitive with 1 round, 3 calls to its primitive with 8 rounds and 1 call to its primitive with 16 rounds.
- The two schemes of the Oribatida submission [18] use their underlying primitive with different numbers of steps per round. This is denoted by $(\theta = \text{s})$ where s is the number of steps.
- Some schemes behave differently depending on the length of the plaintext or associated data. This is denoted in parentheses and/or with an asterisk and remark.
- The four schemes of the Spook [14] submission use both a permutation and a tweakable block cipher. This is denoted by (TCB) for the tweakable block cipher and (P) for the permutation.
- The specifications of DryGASCON128 and DryGASCON256 [36] do not specify the round numbers of their *CoreRound* functions GASCON_{C5} and GASCON_{C9}, respectively. We therefore assume that these primitive calls consist of one round.
- The number of primitive calls in *KSneq32* (DryGASCON submission) depends on the value of the key and can hence not be specified precisely. At least one primitive call is made, so we use ≥ 1 as the number of primitive calls for processing a key.
- If no nonce length is specified, we use 96 bits. This is the minimum nonce length, specified by NIST in [35].

The four tables in this section only contain the primary member of each submission. For the full comparison, see Appendix B.

Table 4.1: Number of primitve calls for permutation primary members. x = bytes plaintext, y = bytes associated data, z = static data assumed > 0 (DryGASCON only).

Submission	Scheme	nk, nn	sk, nn	Message
Parallelizable				
Elephant [17]	Dumbo	1	0	$\lceil \frac{8x}{160} \rceil + \lceil \frac{8x+1}{160} \rceil + \lceil \frac{8y+97}{160} \rceil$
Sequential				
ACE [1]	ACE-AE-128	3	3	$\lceil \frac{8x+1}{64} \rceil + \lceil \frac{8y+1}{64} \rceil + 2$
Ascon [28]	Ascon-128	1 (12)	1 (12)	$\lceil \frac{8x+1}{64} \rceil + \lceil \frac{8y+1}{64} \rceil - 1$ (6) $+1$ (12)
DryGASCON [36]	DryGASCON128	≥ 1	0	$20 \cdot (\lceil \frac{8x}{128} \rceil + \lceil \frac{8y}{128} \rceil + \lceil \frac{8z}{128} \rceil) + 38$
Gimli [15]	Gimli-24-Cipher	1	1	$\lceil \frac{x+1}{16} \rceil + \lceil \frac{y+1}{16} \rceil$
ISAP [27]	ISAP-K-128A	127 (1) $+ 3$ (8) $+ 1$ (16)	127 (1) $+ 1$ (8) $+ 1$ (16)	127 (1) $+ \lceil \frac{8x}{144} \rceil + 1$ (8) $+ \lceil \frac{8x}{144} \rceil + \lceil \frac{8y}{144} \rceil + 1$ (16)
KNOT [10]	KNOT-AEAD(128,256,64)	1 (52)	1 (52)	$\lceil \frac{8x+1}{64} \rceil + \lceil \frac{8y+1}{64} \rceil - 1$ (28) $+ 1$ (32)
ORANGE [24]	ORANGE-Zest	1	1	$\lceil \frac{8x}{256} \rceil + \lceil \frac{8y}{256} \rceil$
Oribatida [18]	Oribatida-256-64	0	0	$\lceil \frac{8x}{128} \rceil + 2$ ($\theta = 4$) $+ \lceil \frac{8y}{128} \rceil - 1$ ($\theta = 2$)
PHOTON-Beetle [9]	PHOTON-Beetle-AEAD[128]	1	1	$\lceil \frac{8x}{128} \rceil + \lceil \frac{8y}{128} \rceil$
SPRAKLE (SCHWAEMM and ESCH) [11]	SCHWAEMM-256-128	1 (11)	1 (11)	$\lceil \frac{8x}{256} \rceil + \lceil \frac{8y}{256} \rceil - 2$ (7) $+ 2$ (11)
SPIX [4]	SPIX	3 (18)	3 (18)	$\lceil \frac{8x+1}{64} \rceil + \lceil \frac{8y+1}{64} \rceil$ (9) $+ 2$ (18)
SpoC [3]	SpoC-64_sLiSCP-light-[192]	2	2	$\lceil \frac{8x}{64} \rceil \lceil \frac{8y}{64} \rceil$
Subterranean 2.0 [26]	Subterranean-SAE	17	12	$\lceil \frac{8x+1}{32} \rceil + \lceil \frac{8y+1}{32} \rceil + 12$
WAGE [2]	WAGE-AE-128	3	3	$\lceil \frac{8x+1}{64} \rceil + \lceil \frac{8y+1}{64} \rceil + 2$
Xoodyak [25]	Xoodyak	1	1	$\lceil \frac{x}{24} \rceil + \lceil \frac{y}{44} \rceil + 1$

Table 4.2: Number of primitve calls for block cipher primary members. x = bytes plaintext, y = bytes associated data.

Submission	Scheme	nk, nn	sk, nn	Message
Parallelizable				
Pyjamask [29]	Pyjamask-128-AEAD	2	1	$\lceil \frac{8x}{128} \rceil + \lceil \frac{8y}{128} \rceil + 1$
Sequential				
COMET [30]	COMET-128_AES-128/128	1	1	$\lceil \frac{8x}{128} \rceil + \lceil \frac{8y}{128} \rceil + 1$
GIFT-COFB [8]	GIFT-COFB	1	1	$\lceil \frac{8x}{128} \rceil + \lceil \frac{8y}{128} \rceil$
HYENA [22]	HYENA	1	1	$\lceil \frac{8x}{128} \rceil + \lceil \frac{8y}{128} \rceil$
mixFeed [23]	mixFeed	2	2	$\lceil \frac{8x}{128} \rceil + \lceil \frac{8y}{128} \rceil + 2$
SAEAES [34]	SAEAES128_64_128	0	0	$\lceil \frac{8x}{64} \rceil + \lceil \frac{8y}{64} \rceil + 1$
SUNDAE-GIFT [7]	SUNDAE-GIFT-96	1	0	$2 \cdot \lceil \frac{8x}{128} \rceil + \lceil \frac{8y+96}{128} \rceil$
TinyJAMBU [39]	TinyJAMBU-128	1 (1024) + 4 (384)	3 (384)	$\lceil \frac{8x}{32} \rceil$ (1024) $+ \lceil \frac{8y}{32} \rceil + 1$ (384)
Hybrid				
Saturnin [19]	Saturnin-CTR-Cascade	1	1	$2 \cdot \lceil \frac{8x}{256} \rceil + \lceil \frac{8y}{256} \rceil + 1$

Table 4.3: Number of primitve calls for tweakable block cipher primary members. x = bytes plaintext, y = bytes associated data.

Submission	Scheme	nk, nn	sk, nn	Message
Parallelizable - inverse free				
Lotus-AEAD and LOCUS-AEAD [21]	TweGIFT-64_LOTUS-AEAD	2	1	$2 \cdot \lceil \frac{8x}{64} \rceil + \lceil \frac{8y}{64} \rceil + 1$
Parallelizable - not inverse free				
ForkAE [5]	PAEF-ForkSkinny-128-288	0	0	$\lceil \frac{8x}{128} \rceil$ (87) $+ \lceil \frac{8y}{128} \rceil$ (56)
SKINNY-AEAD/ SKINNY-HASH [13]	SKINNY-AEAD M1	0	0	$\lceil \frac{8x}{128} \rceil + \lceil \frac{8y}{128} \rceil + 1$
Sequential				
ESTATE [20]	ESTATE_TweAES-128	1	1	$2 \cdot \lceil \frac{8x}{128} \rceil + \lceil \frac{8y}{128} \rceil$
Romulus [33]	Romulus-N1	0	0	$\lceil \frac{8x}{128} \rceil + \lceil \frac{8y}{256} \rceil$ (+1)* *: if not $1 \leq (8y \bmod 256) \leq 128$

Table 4.4: Number of primitve calls for other primary members. x = bytes plaintext, y = bytes associated data.

Submission	Scheme	nk, nn	sk, nn	Message
Grain-128AEAD [31]	Grain-128AEAD	384	384	$16x + 16y + 18$ (if $y \leq 127$) $16x + 16y + 6 \cdot \lfloor \log_{256}(y) \rfloor + 50$ (if $y > 127$)
Spook [14]	Spook[128,512,su]	1 (TCB) $+ 1$ (P)	1 (TCB) $+ 1$ (P)	1 (TCB) $+ \lceil \frac{8x}{256} \rceil + \lceil \frac{8y}{256} \rceil$ (P)

4.3 Interpretation

Different primitives have different costs, therefore two schemes with a different primitive cannot be compared fairly by the number of primitive calls. Nevertheless, these results can still be used to make an estimation of the efficiency of the schemes, by combining them with the results of the comparison of the number of bitwise operations in Chapter 5. This comparison of the number of bitwise operations calculates the costs in the underlying primitive, which makes it possible to make a fair comparison between schemes with different primitives.
Alternatively, one can use the number of primitive calls to compare schemes that use the same primitive. Below, we compare different schemes that use the same primitives. One comparison is among COMET-128_AES-128/128, mixFeed and SAEAES128_64_128, all of which use AES-128/128 as their underlying primitive. The other comparison is among GIFT-COFB, HYENA and SUNDAE-GIFT-96, all of which use GIFT-128 as their underlying primitive. All six schemes are primary members, inverse free and sequential. We consider a single encryption with a new key and nonce. See Figure 4.1

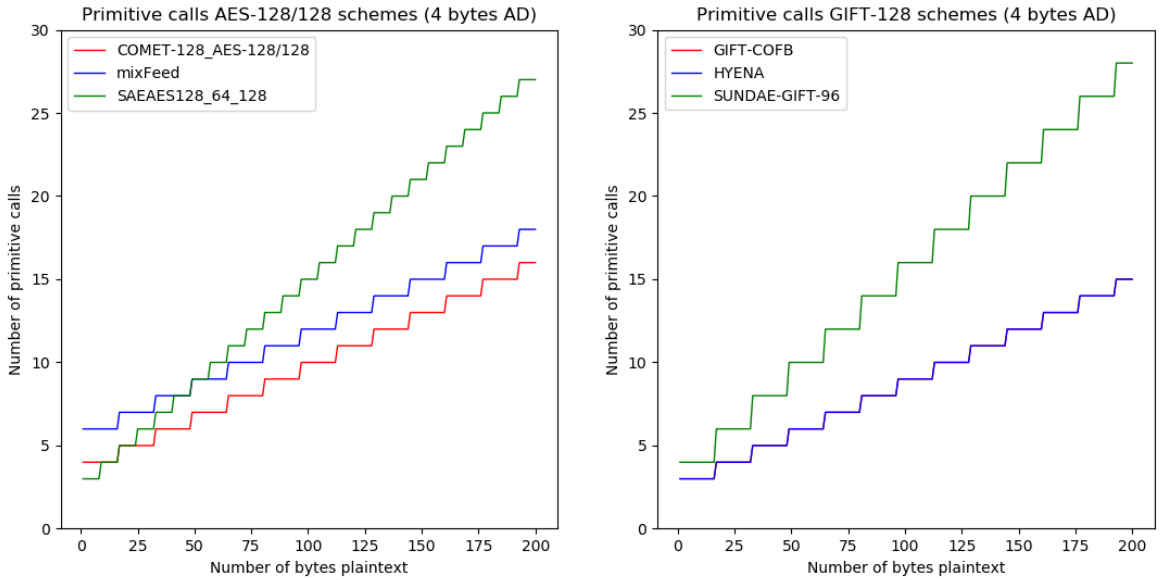

Figure 4.1: Number of primitive calls of single encryption with a new key and new nonce of blockcipher schemes that use AES-128/128 (left) or GIFT-128 (right).

Looking at the AES-128/128 schemes, SAEAES128_64_128 performs best for messages up to 8 bytes of plaintext, but COMET-128_AES-128/128 performs better for 24 or more bytes. In the interval of 9 to 23 bytes the schemes use the same number of primitive calls.
Looking at the GIFT-128 schemes, GIFT-COFB and HYENA both perform best, because they use the exact same number of primitive calls per bytes of plaintext.

Chapter 5

Comparison of the number of bitwise operations

One aspect of the NIST competition for a new lightweight cryptography standard is the efficiency of the schemes. To get some insight on this, we calculated the number of bitwise operations of six candidate schemes: Ascon-128, Ascon-128a, Gimli-24-Cipher, Xoodyak, Subterranean-SAE and SKINNY-AEAD M1.
In Section 5.1 we explain the choice of schemes and in Section 5.2 the method used to calculate the number of bitwise operations of these schemes. The results are presented in Section 5.3 and interpreted in Section 5.4.

5.1 Schemes

As mentioned above, the six schemes that we analyze in this chapter are: Ascon-128, Ascon-128a, Gimli-24-Cipher, Xoodyak, Subterranean-SAE and SKINNY-AEAD M1. These schemes come from five submissions: Ascon [28], Gimli [15], Xoodyak [25], Subterranean 2.0 [26] and SKINNY-AEAD/SKINNY-HASH [13]. The first five schemes all use a permutation as underlying primitive and a duplex-based mode of operation. Due to this, all these schemes are inverse free and sequential. The sixth scheme, SKINNY-AEAD M1, was selected due to the similar efficiency estimation of the SKINNY primitive in [12]. SKINNY-AEAD M1 uses a tweakable block cipher as underlying primitive and the ΘCB3 mode of operation. This schemes is not inverse free, but it is parallelizable.

5.2 Method

When calculating the number of bitwise operations, we look at a single encryption with a new key and a new nonce. We calculate the operations used in the primitive and in the entire AE algorithm separately. We also use the results from the previous chapter.
We multiply the number of operations in the primitive by the number of primitive calls (for single encryption with a new key and nonce) and add the number of operations in the AE algorithm. This gives the total number of bitwise operations in a scheme. Assigning a cost to the different bitwise operations will result in an estimation of the encryption costs, and hence of the efficiency, of the schemes.
For the calculation of the number of bitwise operations we ignore shifts and rotations. We

do not count XOR operations on known zero-bits and we count XOR operations on known one-bits as NOT operations. We ignore the generating of constants in the primitive, because not all submissions provide information on how this is done.

5.3 Results

Below, the results of the comparison of the number of bitwise operations are presented. We first repeat the number of primitive calls, as calculated in chapter 4, for the six schemes of this comparison. Next, we give the number of bitwise operations within the primitives of the schemes. Finally, we give the number of bitwise operations in the AE algorithm.

Number of primitive calls of a single encryption with a new key and new nonce, where $x =$ bytes plaintext, $y =$ bytes associated data:

- Ascon-128: $\lceil \frac{8x+1}{64} \rceil + \left\lceil \frac{8y+1}{64} \right\rceil - 1$ (6 rounds) +2 (12 rounds).
- Ascon-128a: $\lceil \frac{8x+1}{128} \rceil + \left\lceil \frac{8y+1}{128} \right\rceil - 1$ (8 rounds) +2 (12 rounds).
- Gimli-24-Cipher: $\lceil \frac{x+1}{16} \rceil + \left\lceil \frac{y+1}{16} \right\rceil + 1$.
- Xoodyak: $\lceil \frac{x}{24} \rceil + \lceil \frac{y}{44} \rceil + 2$.
- Subterranean-SAE: $\lceil \frac{8x+1}{32} \rceil + \left\lceil \frac{8y+1}{32} \right\rceil + 29$.
- SKINNY-AEAD M1: $\lceil \frac{8x}{128} \rceil + \left\lceil \frac{8y}{128} \right\rceil + 1$.

Number of bitwise operations in the primitive, where $r =$ number of rounds (for Ascon schemes):

- Ascon-128: (1352·XOR + 320·AND + 384·NOT)·r.
- Ascon-128a: (1352·XOR + 320·AND + 384·NOT)·r.
- Gimli-24-Cipher: 17778·XOR + 6144·AND + 3072·OR + 90·NOT.
- Xoodyak: 13944·XOR + 4608·AND + 4608·NOT.
- Subterranean-SAE: 771·XOR + 257·AND + 258·NOT.
- SKINNY-AEAD M1: 24528·XOR + 7168·OR + 7224·NOT.

Number of bitwise operations in the AE algorithm, where $x =$ bytes plaintext, $y =$ bytes associated data:

- Ascon-128: $(8x + 8y + 386)$·XOR + 1·NOT.
- Ascon-128a: $(8x + 8y + 386)$·XOR + 1·NOT.
- Gimli-24-Cipher: $(16x + 8y + 3)$·XOR + 2·NOT.
- Xoodyak: $(16x + 8y + 256)$·XOR + $(2 \cdot \lceil \frac{x}{24} \rceil + \lceil \frac{y}{44} \rceil + 8)$·NOT.
- Subterranean-SAE: $(16x + 8y + 352)$·XOR + $(\lceil \frac{8x+1}{32} \rceil + \left\lceil \frac{8y+1}{32} \right\rceil + 29)$·NOT.
- SKINNY-AEAD M1: $(8x+3\cdot\lceil \frac{8x}{128} \rceil+128\cdot\left\lceil \frac{8y}{128} \right\rceil+3\cdot\left\lfloor \frac{8y}{128} \right\rfloor-128+(8x \bmod 128))$·XOR (+ 1·NOT if $8x \bmod 128 \neq 0$).

5.4 Interpretation

To cover different environments, we look at three weight distributions of the bitwise operations. We always assign a weight of 0 to the NOT operation, because this is negligible. The

three weight distributions are:

Weight distribution 1: XOR, AND and OR all of weight 1.
Weight distribution 2: XOR of weight 1, AND and OR of weight 0.
Weight distribution 3: XOR of weight 0, AND and OR of weight 1.

Weight distribution 1 is relevant for software bitslice implementations on platforms where shifts and rotations are cheap. The XOR, AND and OR operations have equal costs. In this research we count each operation separately, but in reality the costs are cycles per 32 or 64 bits, depending on the length of words in the CPUs. If there is a mismatch between the length of words in the CPU and the length of words in the AE scheme, then the rotations have significant overhead. However, if the word lengths match, it can be quite cheap. Subterranean uses operations on 33 bits, so the results in this research give a distorted image of the costs, in favor of subterranean.
Weight distribution 2 is relevant for hardware, where an XOR operation costs more than twice as much as an AND or OR operation. The exact factor depends on the used technology. In the extreme case, the XOR operation costs way more than the AND and OR operations, hence we only count the XOR operation. The reality lies somewhere between the first two weight distributions.
Weight distribution 3 is relevant for hardware and software bitslice implementations, protected by masking. When masking with d shares, every XOR operation requires d XOR operations and every AND operation expands into d^2 AND operations [32]. OR operations behave in the same way as AND operations. With increasing sharing order d the AND and OR operations dominate the computational cost more and more. For very high values of d, the contribution of XORs becomes negligible and this weight distribution applies. For moderate values of d, the reality lies somewhere between the first and the third weight distribution.

In subsection 5.4.1 we look at the number of primitive calls for the six schemes of this comparison and in subsections 5.4.2 to 5.4.4 we look at their number of bitwise operations for each of the three weight distributions.

5.4.1 Primitive calls

Subterranean-SAE uses the smallest steps, per 4 bytes, and has the highest number of starting calls, namely 29. Because of this Subterranean uses by far the most primitive calls, for any number of bytes plaintext and associate data.
After Subterranean-SAE, Ascon-128 uses the smallest steps, per 8 bytes.
Ascon-128a, Gimli-24-cipher and SKINNY-AEAD M1 all use the same steps, per 16 bytes. The number of primitive calls per bytes plaintext differ a little because Ascon-128a and Gimli-24-Cipher use padding, but SKINNY-AEAD M1 does not, and because Ascon-128a has a higher starting number because of the primitive calls with 12 rounds.
Xoodyak uses 24-byte steps, the biggest steps of the six schemes. Depending on the number of associated data bytes, Gimli-24-Cipher and SKINNY-AEAD M1 use less or the same number of primitive calls, for small plaintexts. If the number of associated data bytes ≤ 15 Gimli-24-Cipher uses less or the same number of calls for plaintexts up to 32 bytes. The same holds for SKINNY-AEAD M1 for 16 or less associated data bytes. As soon as there are more than 16 associated data bytes, Xoodyak always uses less or the same number of primitive calls as those two schemes. See Figure 5.1.

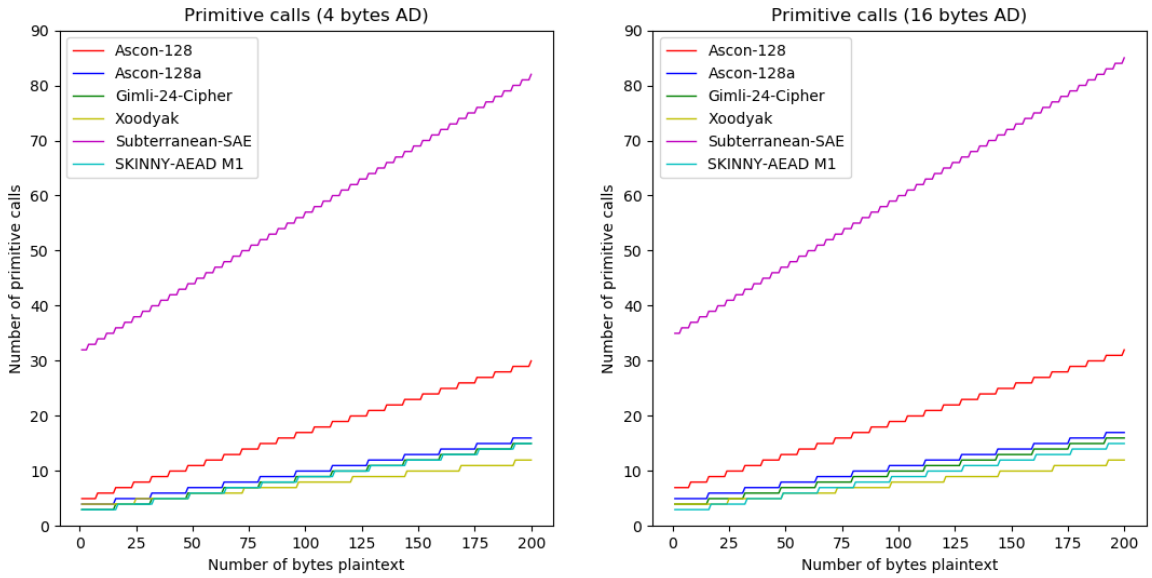

Figure 5.1: Primitive calls

5.4.2 Weight distribution 1 (equal weight)

The impact of the different primitives becomes clear when assigning a weight to the bitwise operations. Subterranean-SAE, which performs the worst when looking at the number of primitive calls, performs best when looking at the number of bitwise operations, because it uses a very lightweight primitive compared to the other schemes.
The opposite is true for Gimli-24-Cipher and SKINNY-AEAD M1, which perform reasonably well when looking at the number of primitive calls, but the worst when looking at the number of bitwise operations.
If the number of associated data bytes is a multiple of 16, then SKINNY-AEAD M1 uses less bitwise operations than Gimli-24-Cipher for small plaintexts. This is caused by the padding rule of Gimli-24-Cipher, which causes an extra primitive call at the multiples of 16 bytes. SKINNY-AEAD M1 makes this step one byte later. After 144 bytes, Gimli-24-Cipher is always cheaper than SKINNY-AEAD M1, because by then the weights of the primitives dominate. The number of plaintext bytes where the two schemes intersect, becomes smaller as the number of associated data goes to 144.
Additionally, SKINNY-AEAD M1 uses less bitwise operations than Gimli-24-Cipher at specific plaintext lengths for small numbers of associated data bytes. With 48 or less associated data bytes, SKINNY-AEAD M1 performs better for 16 bytes plaintext. With 32 or less, also for 32 bytes plaintext, and with 16 or less, also for 48 bytes plaintext.
For small numbers of associated data bytes, Ascon-128a and Xoodyak partially intertwine concerning the number of bitwise operations. For example, with 4 bytes associated data, the two schemes keep alternating between being the cheapest, up until 463 bytes of plaintext, after which Xoodyak wins. If the number of associated data bytes ≥ 64 or in the interval 32 to 44, then Xoodyak always uses less bitwise operations than Ascon-128a.

Barely anything notable happens with Ascon-128. The scheme is not close to any of the other schemes, and instead of the fifth, it is now the fourth best scheme. See Figure 5.2.

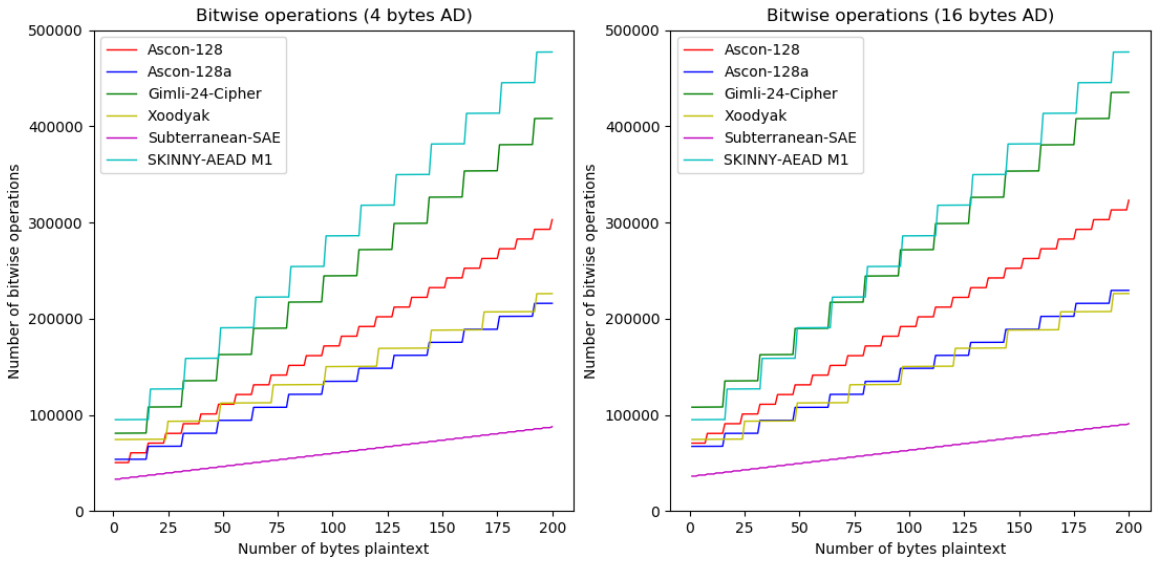

Figure 5.2: Bitwise operations with XOR of weight 1, AND of weight 1, OR of weight 1 and NOT of weight 0

5.4.3 Weight distribution 2 (only XOR)

In the case that only XOR has a weight of 1 and all other bitwise operations have a weight of 0, the difference between both Gimli-24-Cipher and SKINNY-AEAD M1, and Ascon-128a and Xoodyak increases.
SKINNY-AEAD M1 now only performs better than Gimli-24-Cipher in six specific cases: 16 bytes associated data with 16, 32 and 48 bytes plaintext, 32 bytes associated data with 16 and 32 bytes plaintext, and 48 bytes associated data with 16 bytes plaintext. Apart from these cases, Gimli-24-Cipher is always cheaper.
Xoodyak now uses less bitwise operations than Ascon-128a if the number of associated data bytes ≥ 48 (instead of 64) or in the interval 32 to 44.
Subterranean-SAE is still the cheapest of the six schemes, and, although the distances between Ascon-128 and Gimli-24-Cipher, and between Ascon-128 and Ascon-128a have decreased, Ascon-128 is still the fourth best scheme for the majority of the message lengths.
Gimli-24-Cipher is only cheaper than Ascon-128 in a few intervals: in the interval 40 to 47 bytes associated data Gimli-24-Cipher performs better in the interval 8 to 15 bytes plaintext. In the interval 24 to 31 bytes associated data, it also performs better in the interval 24 to 31 bytes plaintext. In the interval 8 to 15 bytes associated data, also in the interval 40 to 47 bytes plaintext. The fact that there are only small modification can be explained by the domination of the XOR operation in the formulas. Therefore, the other operations have a relatively small impact on the outcome and removing their weight has little influence. See Figure 5.3.

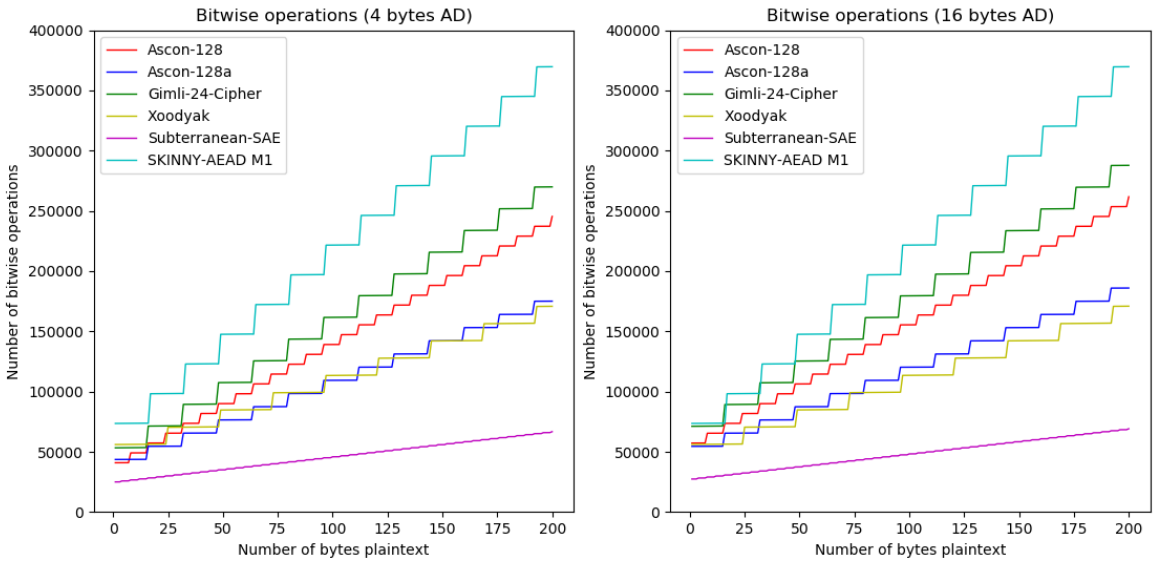

Figure 5.3: Bitwise operations with XOR of weight 1, AND of weight 0, OR of weight 0 and NOT of weight 0

5.4.4 Weight distribution 3 (only AND and OR)

In the case that XOR has a weight of 0 and AND and OR have a weight of 1, a lot changes compared to the other weight distributions: SKINNY-AEAD M1 now always performs better than Gimli-24-Cipher, and Ascon-128a and Xoodyak are no longer intertwined; Ascon-128a is always cheaper than Xoodyak with this weight distribution.
Depending on the number of associated data bytes, Ascon-128 now intersects with Xoodyak and Ascon-128a and is therefore cheaper for some plaintext lengths. Nevertheless, for large plaintexts, Ascon-128 remains the fourth best scheme.
Subterranean-SAE is still the cheapest of the six schemes, but the distance to the other schemes has decreased a lot. See Figure 5.4.

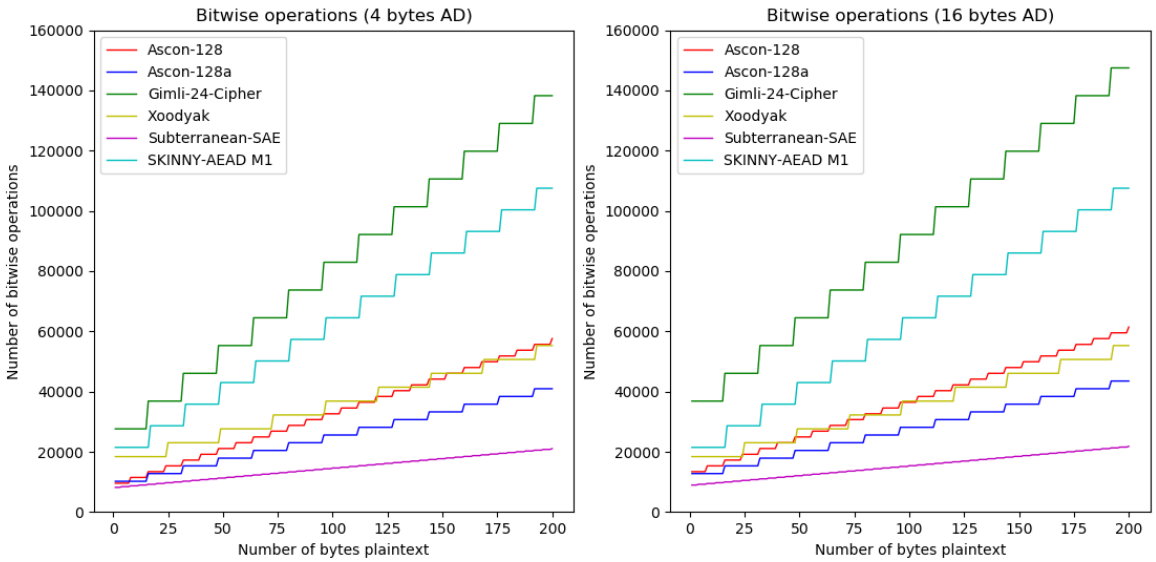

Figure 5.4: Bitwise operations with XOR of weight 0, AND of weight 1, OR of weight 1 and NOT of weight 0

Chapter 6

Conclusions

In this thesis we have compared the 32 second round candidates of the NIST lightweight cryptography competition. We have compared the schemes on a global level, on the number of primitive calls and on the number of bitwise operations.
From the global comparison we can conclude that permutation schemes dominate the competition, taking up half of the submissions. Furthermore, the majority of the submissions uses inverse free and sequential schemes. Lastly, the permutation schemes generally have a larger block size than the tweakable block cipher and block cipher schemes, but looking at the width scrambles things: the two smallest values belong to permutation schemes and the three biggest values belong to two tweakable block cipher schemes and one block cipher scheme.
Comparing the number of primitive calls for schemes with the same primitives shows that of the compared schemes with AES-128/128 as primitive, SAEAES128_64_128 performs best for messages of 8 or less bytes plaintext, and COMET-128_AES-128/128 performs best for plaintexts of 24 or more bytes plaintext. Of the compared schemes with GIFT-128 as primitive, GIFT-COFB and HYENA perform the best. This comparison does not take into account any costs outside the primitive and is only fair if the same underlying primitive is used.
Combining the comparison of the number of primtive calls with the comparison of the number of bitwise operations gives an efficiency estimation. This estimation, favors Subterranean-SAE in all tested weight distributions. Either Gimli-24-Cipher or SKINNY-AEAD M1 performs the worst, depending on the weight distribution.
For future research it would be interesting to extend the comparison of the number of bitwise operations to all candidates, which also increases the number of efficiency estimations. Additionally it would be interesting to look at the security levels of the schemes, both on its own and in combination with the comparison of the number of bitwise operations. This combination would give the opportunity to compare the efficiencies of schemes with equal security levels. Another option is to look at the security margin of the underlying primitives of the schemes: the used round number compared to the maximum round number for which an attack exists. Finally, there are many questions regarding the modes of operation, that are interesting for future research, such as: How many times does the mode pass the data? How much state must be kept track of in the mode? How does the mode handle and combine the different parts of authenticated encryption (authenticating the associated data, encrypting and authenticating the plaintext)?
The global comparison in this thesis gives an overview of the basic information of all schemes in the second round of the competition. Such an overview had not been made yet and can be

used for further research of the schemes. The efficiency estimation of the last two comparisons can be used in the evaluation of the schemes in the competition. Although a similar efficiency estimation has been done, that contains a few primitives that are used by second round candidates, this research purposefully focuses on the schemes of the NIST competition and looks at the entire AE algorithm, making it a new and useful addition to the NIST lightweight cryptography competition.

Bibliography

[1] Mark Aagaard, Riham AlTawy, Guang Gong, Kalikinkar Mandal, and Raghvendra Rohit. ACE. Submission to NIST Lightweight Cryptography competition, 2019.

[2] Mark Aagaard, Riham AlTawy, Guang Gong, Kalikinkar Mandal, Raghvendra Rohit, and Nusa Zidaric. WAGE. Submission to NIST Lightweight Cryptography competition, 2019.

[3] Riham AlTawy, Guang Gong, Morgan He, Ashwin Jha, Kalikinkar Mandal, Mridul Nandi, and Raghvendra Rohit. SpoC. Submission to NIST Lightweight Cryptography competition, 2019.

[4] Riham AlTawy, Guang Gong, Morgan He, Kalikinkar Mandal, and Raghvendra Rohit. Spix. Submission to NIST Lightweight Cryptography competition, 2019.

[5] Elena Andreeva, Virginie Lallemand, Antoon Purnal, Reza Reyhanitabar, Arnab Roy, and Damian Vizár. ForkAE v.1. Submission to NIST Lightweight Cryptography competition, 2019.

[6] Elena Andreeva, Virginie Lallemand, Antoon Purnal, Reza Reyhanitabar, Arnab Roy, and Damian Vizár. Forkcipher: A new primitive for authenticated encryption of very short messages. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology – ASIACRYPT 2019*, pages 153–182, Cham, 2019. Springer International Publishing.

[7] Subhadeep Banik, Andrey Bogdanov, Thomas Peyrin, Yu Sasaki, Siang Meng Sim, Elmar Tischhauser, and Yusoke Todo. SUNDAE-GIFT v1.0. Submission to NIST Lightweight Cryptography competition, 2019.

[8] Subhadeep Banik, Avik Chakraborti, Tetsu Iwata, Kazuhiko Minematsu, Mridul Nandi, Thomas Peyrin, Yu Sasaki, Siang Meng Sim, and Yosuke Todo. GIFT-COFB v1.0. Submission to NIST Lightweight Cryptography competition, 2019.

[9] Zhenzhen Bao, Avik Chakraborti, Nilanjan Datta, Jian Guo, Mridul Nandi, Thomas Peyrin, and Kan Yasuda. PHOTON-Beetle. Submission to NIST Lightweight Cryptography competition, 2019.

[10] Zhenzhen Bao, Tianyou Ding, Fulei Ji, Zejun Xiang, Bohan Yang, Wentao Zhang, and Xuefeng Zhao. KNOT. Submission to NIST Lightweight Cryptography competition, 2019.

[11] Christof Beierle, Alex Biryukov, Luan Cardoso dos Santos, Johann Großschädl, Léo Perrin, Aleksei Udovenko, Vesselin Velichkov, and Qingju Wang. SPARKLE (SCHWAEMM and ESCH). Submission to NIST Lightweight Cryptography competition, 2019.

[12] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The SKINNY family of block ciphers and its low-latency variant MANTIS. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II*, volume 9815 of *Lecture Notes in Computer Science*, pages 123–153. Springer, 2016.

[13] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. SKINNY-AEAD and SKINNY-Hash v1.1. Submission to NIST Lightweight Cryptography competition, 2019.

[14] Davide Bellizia, Francesco Berti, Olivier Bronchain, Gaëtan Cassiers, Sébastien Duval, Chun Guo, Gregor Leander, Gaëtan Leurent, Itamar Levi, Charles Momin, Olivier Pereira, Thomas Peters, Franćois-Xavier Standaert, and Friedrich Wiemer. Spook. Submission to NIST Lightweight Cryptography competition, 2019.

[15] Daniel J. Bernstein, Stefan Kölbl, Stefan Lucks, Pedro Maat Costa Massolino, Florian Mendel, Kashif Nawaz, Tobias Schneider, Peter Schwabe, Franćois-Xavier Standaert, Yosuke Todo, and Benoît Viguier. Gimli. Submission to NIST Lightweight Cryptography competition, 2019.

[16] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Duplexing the sponge: Single-pass authenticated encryption and other applications. In Ali Miri and Serge Vaudenay, editors, *Selected Areas in Cryptography*, pages 320–337, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[17] Tim Beyne, Yu Long Chen, Christoph Dobraunig, and Bart Mennink. Elephant v1.1. Submission to NIST Lightweight Cryptography competition, 2019.

[18] Arghya Bhattacharjee, Eik List, Cuauhtemoc Mancillas López, and Mridul Nandi. Oribatida v1.2. Submission to NIST Lightweight Cryptography competition, 2019.

[19] Anne Canteaut, Sébastien Duval, Gaëtan Leurent, María Naya-Plasencia, Léo Perrin, Thomas Pornin, and André Schrottenloher. Saturnin v1.1. Submission to NIST Lightweight Cryptography competition, 2019.

[20] Avik Chakraborti, Nilanjan Datta, Ashwin Jha, Cuauhtemoc Mancillas Lopez, Mridul Nandi, and Yu Sasaki. ESTATE. Submission to NIST Lightweight Cryptography competition, 2019.

[21] Avik Chakraborti, Nilanjan Datta, Ashwin Jha, Cuauhtemoc Mancillas Lopez, Mridul Nandi, and Yu Sasaki. LOTUS-AEAD and LOCUS-AEAD. Submission to NIST Lightweight Cryptography competition, 2019.

[22] Avik Chakraborti, Nilanjan Datta, Ashwin Jha, and Mridul Nandi. HYENA. Submission to NIST Lightweight Cryptography competition, 2019.

[23] Bishwajit Chakraborty and Mridul Nandi. mixFeed. Submission to NIST Lightweight Cryptography competition, 2019.

[24] Bishwajit Chakraborty and Mridul Nandi. ORANGE. Submission to NIST Lightweight Cryptography competition, 2019.

[25] Joan Daemen, Seth Hoffert, Michaël Peeters, Gilles van Assche, and Ronny van Keer. Xoodyak v1.1. Submission to NIST Lightweight Cryptography competition, 2019.

[26] Joan Daemen, Pedro Maat Costa Massolino, and Yann Rotella. Subterranean 2.0 v1.1. Submission to NIST Lightweight Cryptography competition, 2019.

[27] Christoph Dobraunig, Maria Eichlseder, Stefan Mangard, Florian Mendel, Bart Mennink, Robert Primas, and Thomas Unterluggauer. ISAP v2.0. Submission to NIST Lightweight Cryptography competition, 2019.

[28] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1.2. Submission to NIST Lightweight Cryptography competition, 2019.

[29] Dahmun Goudarzi, Jérémy Jean, Stefan Kölbl, Thomas Peyrin, Matthieu Rivain, Yu Sasaki, and Siang Meng Sim. Pyjamask v1.0. Submission to NIST Lightweight Cryptography competition, 2019.

[30] Shay Gueron, Ashwin Jha, and Mridul Nandi. COMET. Submission to NIST Lightweight Cryptography competition, 2019.

[31] Martin Hell, Thomas Johansson, Willi Meier, Jonathan Sönnerup, and Hirotaka Yoshida. Grain-128AEAD. Submission to NIST Lightweight Cryptography competition, 2019.

[32] Yuval Ishai, Amit Sahai, and David A. Wagner. Private circuits: Securing hardware against probing attacks. In *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481. Springer, 2003.

[33] Tetsu Iwata, Mustafa Khairallah, Kazuhiko Minematsu, and Thomas Peyrin. Romulus v1.2. Submission to NIST Lightweight Cryptography competition, 2019.

[34] Yusuke Naito, Mitsuru Matsui, Yasuyuki Sakai, Daisuke Suzuki, Kazuo Sakiyama, and Takeshi Sugawara. SAEAES. Submission to NIST Lightweight Cryptography competition, 2019.

[35] National Institute of Standards and Technology. Submission requirements and evaluation criteria for the lightweight cryptography standardization process, August 2018. `https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/final-lwc-submission-requirements-august2018.pdf`.

[36] Sébastien Riou. DryGASCON. Submission to NIST Lightweight Cryptography competition, 2019.

[37] Yu Sasaki. NIST Candidates. Presentation at FrisiaCrypt 2019, September 2019.

[38] Meltem Sönmez Turan, Kerry A. McKay, Ćağdaś Ćalık, Donghoon Chang, and Larry Bassham. Status report on the first round of the nist lightweight cryptography standardization process, October 2019. https://nvlpubs.nist.gov/nistpubs/ir/2019/NIST.IR.8268.pdf.

[39] Hongjun Wu and Tao Huang. TinyJAMBU. Submission to NIST Lightweight Cryptography competition, 2019.

Appendix A

Global comparison

In this appendix, we present the complete version of the tables of Section 3.2. In these tables, not only the primary members, but all members of the second round candidates of the NIST lightweight cryptography competition are included. Notation and terminology are inherited from Chapter 3.

Table A.1: Schemes with permutation primitive

Identification			**Underlying Primitive**			**Mode of operation**				
Submission	**Scheme**	**Name**	**bs**	**ks**	**ts**	**Name**	**if**	**p**	**rk**	**c**
Parallelizable										
Elephant [17]	Dumbo	Spongent-π[160]	160	-	-	Encrypt-then-MAC Even-Mansour	Yes	Yes	-	-
	Jumbo	Spongent-π[176]	176	-	-	Encrypt-then-MAC Even-Mansour	Yes	Yes	-	-
	Delirium	Keccak-f[200]	200	-	-	Encrypt-then-MAC Even-Mansour	Yes	Yes	-	-
Sequential										
ACE [1]	ACE-AE-128	ACE	320	-	-	Duplex	Yes	No	-	256
Ascon [28]	Ascon-128	Ascon-p	320	-	-	Monkeyduplex	Yes	No	-	256
	Ascon-128a	Ascon-p	320	-	-	Monkeyduplex	Yes	No	-	192
DryGASCON [36]	DryGASCON128	G GASCON$_{C5}$	320	-	-	DrySponge	Yes	No	-	448
	DryGASCON256	GASCON$_{C9}$	576	-	-	DrySponge	Yes	No	-	704
Gimli [15]	Gimli-24-Cipher	Gimli permutation	384	-	-	Duplex	Yes	No	-	256
ISAP [27]	ISAP-K-128A	Keccak-p[400]	400	-	-	Encrypt-then-MAC Sponge	Yes	No	-	256
	ISAP-A-128A	Ascon-p	320	-	-	Encrypt-then-MAC Sponge	Yes	No	-	256
	ISAP-K-128	Keccak-p[400]	400	-	-	Encrypt-then-MAC Sponge	Yes	No	-	256
	ISAP-A-128	Ascon-p	320	-	-	Encrypt-then-MAC Sponge	Yes	No	-	256
KNOT [10]	KNOT-AEAD(128,256,64)		256	-	-	Monkeyduplex	Yes	No	-	192
	KNOT-AEAD(128,384,192)		384	-	-	Monkeyduplex	Yes	No	-	192
	KNOT-AEAD(192,384,96)		384	-	-	Monkeyduplex	Yes	No	-	288
	KNOT-AEAD(256,512,128)		512	-	-	Monkeyduplex	Yes	No	-	384
ORANGE [24]	ORANGE-Zest	PHOTON256	256	-	-	ORANGE-Zest	Yes	No	-	128

Oribatida [18]	Oribatida-256-64	SimP-256-4 SimP-256-2	256	-	-	Duplex	Yes	No	-	128
	Oribatida-192-96	SimP-192-4 SimP-192-2	192	-	-	Duplex	Yes	No	-	96
PHOTON-Beetle [9]	PHOTON-Beetle-AEAD[128]	PHOTON256	256	-	-	Beetle	Yes	No	-	128
	PHOTON-Beetle-AEAD[32]	PHOTON256	256	-	-	Beetle	Yes	No	-	224
SPRAKLE (SCHWAEMM and ESCH) [11]	SCHWAEMM-256-128	SPARKLE384	384	-	-	SCHWAEMM-256-128	Yes	No	-	128
	SCHWAEMM-128-128	SPARKLE256	256	-	-	SCHWAEMM-128-128	Yes	No	-	128
	SCHWAEMM-192-192	SPARKLE384	384	-	-	SCHWAEMM-192-192	Yes	No	-	192
	SCHWAEMM-256-256	SPARKLE512	512	-	-	SCHWAEMM-256-256	Yes	No	-	256
SPIX [4]	SPIX	sLiSCP-light-[256]	256	-	-	Monkeyduplex	Yes	No	-	192
SpoC [3]	SpoC-64_sLiSCP-light-[192]	sLiSCP-light-[192]	192	-	-	SpoC-64	Yes	No	-	128
	SpoC-128_sLiSCP-light-[256]	sLiSCP-light-[256]	256	-	-	SpoC-128	Yes	No	-	128
Subterranean 2.0 [26]	Subterranean-SAE	Subterranean round function	257	-	-	Duplex	Yes	No	-	225
WAGE [2]	WAGE-*AE*-128	WAGE	259	-	-	Duplex	Yes	No	-	195
Xoodyak [2]	Xoodyak	Xoodoo[12]	384	-	-	Cyclist	Yes	No	-	192

Table A.2: Schemes with block cipher primitive

Identification		**Underlying Primitive**				**Mode of operation**				
Submission	**Scheme**	**Name**	**bs**	**ks**	**ts**	**Name**	**if**	**p**	**rk**	**c**
Parallelizable										
Pyjamask [29]	Pyjamask-128-AEAD	Pyjamask-128	128	128	-	OCB	No	Yes	No	-
	Pyjamask-96-AEAD	Pyjamask-96	96	128	-	OCB	No	Yes	No	-

Sequential										
COMET [30]	COMET-128_AES-128/128	AES-128/128	128	128	-	COMET	Yes	No	Yes	-
	COMET-128_CHAM-128/128	CHAM-128/128	128	128	-	COMET	Yes	No	Yes	-
	COMET-64_Speck-64/128	Speck-64/128	64	128	-	COMET	Yes	No	Yes	-
	COMET-64_CHAM-64/128	CHAM-64/128	64	128	-	COMET	Yes	No	Yes	-
GIFT-COFB [8]	GIFT-COFB	GIFT-128	128	128	-	COFB	Yes	No	No	-
HYENA [22]	HYENA	GIFT-128	128	128	-	HYFB	Yes	No	No	-
mixFeed [23]	mixFeed	AES-128/128	128	128	-	Minimally Xored Feedback Mode	Yes	No	Yes	-
SAEAES [34]	SAEAES128_64_128	AES-128/128	128	128	-	SAEB	Yes	No	No	192
	SAEAES128_64_64	AES-128/128	128	128	-	SAEB	Yes	No	No	192
	SAEAES128_120_64	AES-128/128	128	128	-	SAEB	Yes	No	No	192
	SAEAES128_120_128	AES-128/128	128	128	-	SAEB	Yes	No	No	192
	SAEAES192_64_64	AES-128/192	128	192	-	SAEB	Yes	No	No	256
	SAEAES192_64_128	AES-128/192	128	192	-	SAEB	Yes	No	No	256
	SAEAES192_120_128	AES-128/192	128	192	-	SAEB	Yes	No	No	256
	SAEAES256_64_64	AES-128/256	128	256	-	SAEB	Yes	No	No	320
	SAEAES256_64_128	AES-128/256	128	256	-	SAEB	Yes	No	No	320
	SAEAES256_120_128	AES-128/256	128	256	-	SAEB	Yes	No	No	320
SUNDAE-GIFT [7]	SUNDAE-GIFT-96	GIFT-128	128	128	-	SUNDAE	Yes	No	No	-
	SUNDAE-GIFT-0	GIFT-128	128	128	-	SUNDAE	Yes	No	No	-
	SUNDAE-GIFT-128	GIFT-128	128	128	-	SUNDAE	Yes	No	No	-
	SUNDAE-GIFT-64	GIFT-128	128	128	-	SUNDAE	Yes	No	No	-
TinyJAMBU [39]	TinyJAMBU-128		128	128	-	TinyJAMBU	Yes	No	No	96
	TinyJAMBU-192		128	192	-	TinyJAMBU	Yes	No	No	96
	TinyJAMBU-256		128	256	-	TinyJAMBU	Yes	No	No	96
Hybrid										
Saturnin [19]	Saturnin-CTR-Cascade	Saturnin	256	256	-	Encrypt-then-MAC Counter mode Cascade construction	Yes	Hybrid: Yes No	Hybrid: No Yes	-

Table A.3: Schemes with tweakable block cipher primitive

Identification		**Underlying Primitive**				**Mode of operation**				
Submission	**Scheme**	**Name**	**bs**	**ks**	**ts**	**Name**	**if**	**p**	**rk**	**c**
Parallelizable - inverse free										
Lotus-AEAD and LOCUS-AEAD [21]	TweGIFT-64_LOTUS-AEAD	TweGIFT-64	64	128	4	LOTUS-AEAD	Yes	Yes	Yes	-
Parallelizable - not inverse free										
ForkAE [5]	PAEF-ForkSkinny-128-288	ForkSkinny-128-288	128	128	160	PAEF	No	Yes	No	-
	PAEF-ForkSkinny-64-192	ForkSkinny-64-192	64	128	64	PAEF	No	Yes	No	-
	PAEF-ForkSkinny-128-192	ForkSkinny-128-192	128	128	64	PAEF	No	Yes	No	-
	PAEF-ForkSkinny-128-256	ForkSkinny-128-256	128	128	128	PAEF	No	Yes	No	-
Lotus-AEAD and LOCUS-AEAD [21]	TweGIFT-64_LOCUS-AEAD	TweGIFT-64	64	128	4	LOCUS-AEAD	No	Yes	Yes	-
SKINNY-AEAD/ SKINNY-HASH [13]	SKINNY-AEAD M1	SKINNY-128-384	128	128	256	ΘCB3	No	Yes	No	-
	SKINNY-AEAD M2	SKINNY-128-384	128	128	256	ΘCB3	No	Yes	No	-
	SKINNY-AEAD M3	SKINNY-128-384	128	128	256	ΘCB3	No	Yes	No	-
	SKINNY-AEAD M4	SKINNY-128-384	128	128	256	ΘCB3	No	Yes	No	-
	SKINNY-AEAD M5	SKINNY-128-256	128	128	128	ΘCB3	No	Yes	No	-
	SKINNY-AEAD M6	SKINNY-128-256	128	128	128	ΘCB3	No	Yes	No	-
Sequential - inverse free										
ESTATE [20]	ESTATE_TweAES-128	TweAES-128	128	128	4	MAC-then-Encrypt FCBC* OFB	Yes	No	No	-
	sESTATE_TweAES-128-6	TweAES-128-6	128	128	4	MAC-then-Encrypt FCBC* OFB	Yes	No	No	-
	ESTATE_TweGIFT-128	TweGIFT-128	128	128	4	MAC-then-Encrypt FCBC* OFB	Yes	No	No	-

Romulus [33]	Romulus-N1	SKINNY-128-384	128	128	256	Romulus-N	Yes	No	No	-
	Romulus-N2	SKINNY-128-384	128	128	256	Romulus-N	Yes	No	No	-
	Romulus-N3	SKINNY-128-256	128	128	128	Romulus-N	Yes	No	No	-
	Romulus-M1	SKINNY-128-384	128	128	256	Romulus-M	Yes	No	No	-
	Romulus-M2	SKINNY-128-384	128	128	256	Romulus-M	Yes	No	No	-
	Romulus-M3	SKINNY-128-256	128	128	128	Romulus-M	Yes	No	No	-
Sequential - not inverse free										
ForkAE [5]	SAEF-ForkSkinny-128-192	ForkSkinny-128-192	128	128	64	SAEF	No	No	No	-
	SAEF-ForkSkinny-128-256	ForkSkinny-128-256	128	128	128	SAEF	No	No	No	-

Table A.4: Other schemes

Identification		**Underlying Primitive**				**Mode of operation**				
Submission	**Scheme**	**Name**	**bs**	**ks**	**ts**	**Name**	**if**	**p**	**rk**	**c**
Grain-128AEAD [31]	Grain-128AEAD		-	128	-	Grain-128AEAD	Yes	No	No	-
Saturnin [19]	Saturnin-Short	Saturnin	256	256	-	Single encryption	-	-	-	-
Spook [14]	Spook[128,512,su]	Shadow-512 Clyde-128	512 128	- 128	- 127	S1P	No	No	No	256
	Spook[128,512,mu]	Shadow-512 Clyde-128	512 128	- 128	- 127	S1P	No	No	No	256
	Spook[128,384,su]	Shadow-384 Clyde-128	384 128	- 128	- 127	S1P	No	No	No	256
	Spook[128,384,mu]	Shadow-384 Clyde-128	384 128	- 128	- 127	S1P	No	No	No	256

Appendix B

Primitive calls

In this appendix, we present the complete version of the tables of Section 4.2. In these tables, not only the primary members, but all members of the second round candidates of the NIST lightweight cryptography competition are included. Notation and terminology are inherited from Chapter 4.

Table B.1: Number of primitve calls for permutation schemes. x = bytes plaintext, y = bytes associated data, z = static data assumed > 0 (DryGASCON only).

Submission	Scheme	nk, nn	sk, nn	Message
Parallelizable				
Elephant [17]	Dumbo	1	0	$\lceil \frac{8x}{160} \rceil + \lceil \frac{8x+1}{160} \rceil + \lceil \frac{8y+97}{160} \rceil$
	Jumbo	1	0	$\lceil \frac{8x}{176} \rceil + \lceil \frac{8x+1}{176} \rceil + \lceil \frac{8y+97}{176} \rceil$
	Delirium	1	0	$\lceil \frac{8x}{200} \rceil + \lceil \frac{8x+1}{200} \rceil + \lceil \frac{8y+97}{200} \rceil$
Sequential				
ACE [1]	ACE-*AE*-128	3	3	$\lceil \frac{8x+1}{64} \rceil + \lceil \frac{8y+1}{64} \rceil + 2$
Ascon [28]	Ascon-128	1 (12)	1 (12)	$\lceil \frac{8x+1}{64} \rceil + \lceil \frac{8y+1}{64} \rceil - 1$ (6) $+1$ (12)
	Ascon-128a	1 (12)	1 (12)	$\lceil \frac{8x+1}{128} \rceil + \lceil \frac{8y+1}{128} \rceil - 1$ (8) $+1$ (12)
DryGASCON [36]	DryGASCON128	≥ 1	0	$20 \cdot (\lceil \frac{8x}{128} \rceil + \lceil \frac{8y}{128} \rceil + \lceil \frac{8z}{128} \rceil) + 38$
	DryGASCON256	≥ 1	0	$15 \cdot (\lceil \frac{8x}{128} \rceil + \lceil \frac{8y}{128} \rceil + \lceil \frac{8z}{128} \rceil) + 35$
Gimli [15]	Gimli-24-Cipher	1	1	$\lceil \frac{x+1}{16} \rceil + \lceil \frac{y+1}{16} \rceil$
ISAP [27]	ISAP-K-128A	127 (1) $+ 3$ (8) $+ 1$ (16)	127 (1) $+ 1$ (8) $+ 1$ (16)	127 (1) $+ \lceil \frac{8x}{144} \rceil + 1$ (8) $+ \lceil \frac{8x}{144} \rceil + \lceil \frac{8y}{144} \rceil + 1$ (16)
	ISAP-A-128A	127 (1) $+ 4$ (12)	127 (1) $+ 2$ (12)	127 (1) $+ \lceil \frac{8x}{64} \rceil$ (6) $+ \lceil \frac{8x}{64} \rceil + \lceil \frac{8y}{64} \rceil + 2$ (12)
	ISAP-K-128	130 (12) $+ 1$ (20)	129 (12) $+ 1$ (20)	$\lceil \frac{8x}{144} \rceil + 128$ (12) $+ \lceil \frac{8x}{144} \rceil + \lceil \frac{8y}{144} \rceil + 1$ (20)

	ISAP-A-128	131	(12)	129	(12)	$2 \cdot \lceil \frac{8x}{64} \rceil + \lceil \frac{8y}{64} \rceil + 129$	(12)
KNOT [10]	KNOT-AEAD(128,256,64)	1	(52)	1	(52)	$\lceil \frac{8x+1}{64} \rceil + \lceil \frac{8y+1}{64} \rceil - 1$ $+ 1$	(28) (32)
	KNOT-AEAD(128,384,192)	1	(76)	1	(76)	$\lceil \frac{8x+1}{192} \rceil + \lceil \frac{8y+1}{192} \rceil - 1$ $+ 1$	(28) (32)
	KNOT-AEAD(192,384,96)	1	(76)	1	(76)	$\lceil \frac{8x+1}{96} \rceil + \lceil \frac{8y+1}{96} \rceil - 1$ $+ 1$	(40) (44)
	KNOT-AEAD(256,512,128)	1	(100)	1	(100)	$\lceil \frac{8x+1}{128} \rceil + \lceil \frac{8y+1}{128} \rceil - 1$ $+ 1$	(52) (56)
ORANGE [24]	ORANGE-Zest	1		1		$\lceil \frac{8x}{256} \rceil + \lceil \frac{8y}{256} \rceil$	
Oribatida [18]	Oribatida-256-64	0		0		$\lceil \frac{8x}{128} \rceil + 2$ $+ \lceil \frac{8y}{128} \rceil - 1$	$(\theta = 4)$ $(\theta = 2)$
	Oribatida-192-96	0		0		$\lceil \frac{8x}{96} \rceil + 2$ $+ \lceil \frac{8y}{96} \rceil - 1$	$(\theta = 4)$ $(\theta = 2)$
PHOTON-Beetle [9]	PHOTON-Beetle-AEAD[128]	1		1		$\lceil \frac{8x}{128} \rceil + \lceil \frac{8y}{128} \rceil$	
	PHOTON-Beetle-AEAD[32]	1		1		$\lceil \frac{8x}{32} \rceil + \lceil \frac{8y}{32} \rceil$	
SPRAKLE (SCHWAEMM and ESCH) [11]	SCHWAEMM-256-128	1	(11)	1	(11)	$\lceil \frac{8x}{256} \rceil + \lceil \frac{8y}{256} \rceil - 2$ $+ 2$	(7) (11)
	SCHWAEMM-128-128	1	(10)	1	(10)	$\lceil \frac{8x}{128} \rceil + \lceil \frac{8y}{128} \rceil - 2$ $+ 2$	(7) (10)
	SCHWAEMM-192-192	1	(11)	1	(11)	$\lceil \frac{8x}{192} \rceil + \lceil \frac{8y}{192} \rceil - 2$ $+ 2$	(7) (11)
	SCHWAEMM-256-256	1	(12)	1	(12)	$\lceil \frac{8x}{256} \rceil + \lceil \frac{8y}{256} \rceil - 2$ $+ 2$	(8) (12)
SPIX [4]	SPIX	3	(18)	3	(18)	$\lceil \frac{8x+1}{64} \rceil + \lceil \frac{8y+1}{64} \rceil$ $+ 2$	(9) (18)
SpoC [3]	SpoC-64_sLiSCP-light-[192]	2		2		$\lceil \frac{8x}{64} \rceil \lceil \frac{8y}{64} \rceil$	
	SpoC-128_sLiSCP-light-[256]	1		1		$\lceil \frac{8x}{128} \rceil \lceil \frac{8y}{128} \rceil$	
Subterranean 2.0 [26]	Subterranean-SAE	17		12		$\lceil \frac{8x+1}{32} \rceil + \lceil \frac{8y+1}{32} \rceil + 12$	
WAGE [2]	WAGE-AE-128	3		3		$\lceil \frac{8x+1}{64} \rceil + \lceil \frac{8y+1}{64} \rceil + 2$	
Xoodyak [25]	Xoodyak	1		1		$\lceil \frac{x}{24} \rceil + \lceil \frac{y}{44} \rceil + 1$	

Table B.2: Number of primitve calls for block cipher schemes. x = bytes plaintext, y = bytes associated data.

Submission	Scheme	nk, nn	sk, nn	Message
Parallelizable				
Pyjamask [29]	Pyjamask-128-AEAD	2	1	$\lceil \frac{8x}{128} \rceil + \lceil \frac{8y}{128} \rceil + 1$
	Pyjamask-96-AEAD	2	1	$\lceil \frac{8x}{96} \rceil + \lceil \frac{8y}{96} \rceil + 1$
Sequential				
COMET [30]	COMET-128_AES-128/128	1	1	$\lceil \frac{8x}{128} \rceil + \lceil \frac{8y}{128} \rceil + 1$
	COMET-128_CHAM-128/128	1	1	$\lceil \frac{8x}{128} \rceil + \lceil \frac{8y}{128} \rceil + 1$
	COMET-64_Speck-64/128	1	0	$\lceil \frac{8x}{64} \rceil + \lceil \frac{8y}{64} \rceil + 1$
	COMET-64_CHAM-64/128	1	0	$\lceil \frac{8x}{64} \rceil + \lceil \frac{8y}{64} \rceil + 1$
GIFT-COFB [8]	GIFT-COFB	1	1	$\lceil \frac{8x}{128} \rceil + \lceil \frac{8y}{128} \rceil$
HYENA [22]	HYENA	1	1	$\lceil \frac{8x}{128} \rceil + \lceil \frac{8y}{128} \rceil$
mixFeed [23]	mixFeed	2	2	$\lceil \frac{8x}{128} \rceil + \lceil \frac{8y}{128} \rceil + 2$
SAEAES [34]	SAEAES128_64_128	0	0	$\lceil \frac{8x}{64} \rceil + \lceil \frac{8y}{64} \rceil + 1$
	SAEAES128_64_64	0	0	$\lceil \frac{8x}{64} \rceil + \lceil \frac{8y}{64} \rceil + 1$
	SAEAES128_120_64	0	0	$\lceil \frac{8x}{64} \rceil + \lceil \frac{8y}{120} \rceil + 1$
	SAEAES128_120_128	0	0	$\lceil \frac{8x}{64} \rceil + \lceil \frac{8y}{120} \rceil + 1$
	SAEAES192_64_64	0	0	$\lceil \frac{8x}{64} \rceil + \lceil \frac{8y}{64} \rceil + 1$
	SAEAES192_64_128	0	0	$\lceil \frac{8x}{64} \rceil + \lceil \frac{8y}{64} \rceil + 1$
	SAEAES192_120_128	0	0	$\lceil \frac{8x}{64} \rceil + \lceil \frac{8y}{120} \rceil + 1$
	SAEAES256_64_64	0	0	$\lceil \frac{8x}{64} \rceil + \lceil \frac{8y}{64} \rceil + 1$
	SAEAES256_64_128	0	0	$\lceil \frac{8x}{64} \rceil + \lceil \frac{8y}{64} \rceil + 1$
	SAEAES256_120_128	0	0	$\lceil \frac{8x}{64} \rceil + \lceil \frac{8y}{120} \rceil + 1$
SUNDAE-GIFT [7]	SUNDAE-GIFT-96	1	0	$2 \cdot \lceil \frac{8x}{128} \rceil + \lceil \frac{8y+96}{128} \rceil$
	SUNDAE-GIFT-0	1	0	$2 \cdot \lceil \frac{8x}{128} \rceil + \lceil \frac{8y}{128} \rceil$
	SUNDAE-GIFT-128	1	0	$2 \cdot \lceil \frac{8x}{128} \rceil + \lceil \frac{8y+128}{128} \rceil$
	SUNDAE-GIFT-64	1	0	$2 \cdot \lceil \frac{8x}{128} \rceil + \lceil \frac{8y+64}{128} \rceil$
TinyJAMBU [39]	TinyJAMBU-128	1 (1024) + 4 (384)	3 (384)	$\lceil \frac{8x}{32} \rceil$ (1024) + $\lceil \frac{8y}{32} \rceil + 1$ (384)
	TinyJAMBU-192	1 (1152) + 4 (384)	3 (384)	$\lceil \frac{8x}{32} \rceil$ (1152) + $\lceil \frac{8y}{32} \rceil + 1$ (384)
	TinyJAMBU-256	1 (1280) + 4 (384)	3 (384)	$\lceil \frac{8x}{32} \rceil$ (1280) + $\lceil \frac{8y}{32} \rceil + 1$ (384)
Hybrid				
Saturnin [19]	Saturnin-CTR-Cascade	1	1	$2 \cdot \lceil \frac{8x}{256} \rceil + \lceil \frac{8y}{256} \rceil + 1$

Table B.3: Number of primitve calls for tweakable block cipher schemes. x = bytes plaintext, y = bytes associated data.

Submission	Scheme	nk, nn	sk, nn	Message
Parallelizable - inverse free				
Lotus-AEAD and LOCUS-AEAD [21]	TweGIFT-64_LOTUS-AEAD	2	1	$2 \cdot \lceil \frac{8x}{64} \rceil + \lceil \frac{8y}{64} \rceil + 1$
Parallelizable - not inverse free				
ForkAE [5]	PAEF-ForkSkinny-128-288	0	0	$\lceil \frac{8x}{128} \rceil$ (87) $+ \lceil \frac{8y}{128} \rceil$ (56)
	PAEF-ForkSkinny-64-192	0	0	$\lceil \frac{8x}{128} \rceil$ (63) $+ \lceil \frac{8y}{128} \rceil$ (40)
	PAEF-ForkSkinny-128-192	0	0	$\lceil \frac{8x}{128} \rceil$ (75) $+ \lceil \frac{8y}{128} \rceil$ (48)
	PAEF-ForkSkinny-128-256	0	0	$\lceil \frac{8x}{128} \rceil$ (75) $+ \lceil \frac{8y}{128} \rceil$ (48)
Lotus-AEAD and LOCUS-AEAD [21]	TweGIFT-64_LOCUS-AEAD	2	1	$2 \cdot \lceil \frac{8x}{64} \rceil + \lceil \frac{8y}{64} \rceil + 1$
SKINNY-AEAD/ SKINNY-HASH [13]	SKINNY-AEAD M1	0	0	$\lceil \frac{8x}{128} \rceil + \lceil \frac{8y}{128} \rceil + 1$
	SKINNY-AEAD M2	0	0	$\lceil \frac{8x}{128} \rceil + \lceil \frac{8y}{128} \rceil + 1$
	SKINNY-AEAD M3	0	0	$\lceil \frac{8x}{128} \rceil + \lceil \frac{8y}{128} \rceil + 1$
	SKINNY-AEAD M4	0	0	$\lceil \frac{8x}{128} \rceil + \lceil \frac{8y}{128} \rceil + 1$
	SKINNY-AEAD M5	0	0	$\lceil \frac{8x}{128} \rceil + \lceil \frac{8y}{128} \rceil + 1$
	SKINNY-AEAD M6	0	0	$\lceil \frac{8x}{128} \rceil + \lceil \frac{8y}{128} \rceil + 1$
Sequential - inverse free				
ESTATE [20]	ESTATE_TweAES-128	1	1	$2 \cdot \lceil \frac{8x}{128} \rceil + \lceil \frac{8y}{128} \rceil$
	sESTATE_TweAES-128-6	1 (6)	1 (6)	$\lceil \frac{8x}{128} \rceil + \lceil \frac{8y}{128} \rceil - 2$ (6) $+ \lceil \frac{8x}{128} \rceil + 2$ (10)
	ESTATE_TweGIFT-128	1	1	$2 \cdot \lceil \frac{8x}{128} \rceil + \lceil \frac{8y}{128} \rceil$
Romulus [33]	Romulus-N1	0	0	$\lceil \frac{8x}{128} \rceil + \lceil \frac{8y}{256} \rceil$ (+1)* *: if not $1 \leq (8y \bmod 256) \leq 128$
	Romulus-N2	0	0	$\lceil \frac{8x}{128} \rceil + \lceil \frac{8y}{224} \rceil$ (+1)* *: if not $1 \leq (8y \bmod 224) \leq 128$
	Romulus-N3	0	0	$\lceil \frac{8x}{128} \rceil + \lceil \frac{8y}{224} \rceil$ (+1)* *: if not $1 \leq (8y \bmod 224) \leq 128$
	Romulus-M1	0	0	$\lceil \frac{8x}{128} \rceil \lceil \frac{8x}{256} \rceil + \lceil \frac{8y}{256} \rceil$ (+1)* *: if not $1 \leq (8x \bmod 256) \leq 128$ and not $1 \leq (8y \bmod 256) \leq 128$
	Romulus-M2	0	0	$\lceil \frac{8x}{128} \rceil \lceil \frac{8x}{224} \rceil + \lceil \frac{8y}{224} \rceil$ (+1)* *: if not $1 \leq (8x \bmod 224) \leq 128$ and not $1 \leq (8y \bmod 224) \leq 128$

	Romulus-M3	0	0	$\lceil \frac{8x}{128} \rceil \lceil \frac{8x}{224} \rceil + \lceil \frac{8y}{224} \rceil (+1)$* *: if not $1 \leq (8x \bmod 224) \leq 128$ and not $1 \leq (8y \bmod 224) \leq 128$
Sequential - not inverse free				
ForkAE [5]	SAEF-ForkSkinny-128-192	0	0	$\lceil \frac{8x}{128} \rceil$ (75) $+ \lceil \frac{8y}{128} \rceil$ (48)
	SAEF-ForkSkinny-128-256	0	0	$\lceil \frac{8x}{128} \rceil$ (75) $+ \lceil \frac{8y}{128} \rceil$ (48)

Table B.4: Number of primitve calls for other schemes. x = bytes plaintext, y = bytes associated data.

Submission	Scheme	nk, nn	sk, nn	Message
Grain-128AEAD [31]	Grain-128AEAD	384	384	$16x + 16y + 18$ (if $y \leq 127$) $16x + 16y + 6 \cdot \lfloor \log_{256}(y) \rfloor + 50$ (if $y > 127$)
Saturnin [19]	Saturnin-Short	0	0	1
Spook [14]	Spook[128,512,su]	1 (TCB) $+ 1$ (P)	1 (TCB) $+ 1$ (P)	1 (TCB) $+ \lceil \frac{8x}{256} \rceil + \lceil \frac{8y}{256} \rceil$ (P)
	Spook[128,512,mu]	1 (TCB) $+ 1$ (P)	1 (TCB) $+ 1$ (P)	1 (TCB) $+ \lceil \frac{8x}{256} \rceil + \lceil \frac{8y}{256} \rceil$ (P)
	Spook[128,384,su]	1 (TCB) $+ 1$ (P)	1 (TCB) $+ 1$ (P)	1 (TCB) $+ \lceil \frac{8x}{128} \rceil + \lceil \frac{8y}{128} \rceil$ (P)
	Spook[128,384,mu]	1 (TCB) $+ 1$ (P)	1 (TCB) $+ 1$ (P)	1 (TCB) $+ \lceil \frac{8x}{128} \rceil + \lceil \frac{8y}{128} \rceil$ (P)