itk AVtobvS Sàrl

AHX-Preliminary Security Analysis

Symmetric block cipher proposition for Post-quantum cryptography

KUDELSKI SECURITY

29.04.2019

DOCUMENT PROPERTIES

Version:	01.05
File name:	2019-04-29-Kudelski-AHX-Analysis-v01.05.docx
Publication date:	29.04.2019
Document Owner:	Benoît Gerhard
Document Recipient:	Stiepan Aurélien Kovac
Document Status:	Approved
Client Company Name:	itk AVtobvS Sàrl

Recipients

Stiepan Aurélien Kovac
itk AVtobvS Sàrl
chemin de Monséjour n. 2
CH-1700 Fribourg
Switzerland

Kudelski Contact

In case of questions regarding this document please contact:

Benoît Gerhard
Head of Security Evaluation & Attacks
IoT Security Labs
Route de Genève 22-24
1033 Cheseaux-sur-Lausanne

TABLE OF CONTENTS

TABLE OF FIGURES

TABLE OF TABLES

EXECUTIVE SUMMARY

In a context where the amount of research on quantum computers is increasing, threats for classical cryptography appears.

For asymmetric cryptography and due to the efficiency of Shor's algorithm [15], NIST has launched a challenge to solicit, evaluate, and standardize one or more quantum-resistant public-key cryptographic algorithms.

Due to the quantum Grover's algorithm for hash functions with n bits input, the preimage resistance is reduced to $2^{\frac{n}{2}}$ ([12], [13]). It has the same impact on the key search [9], in case of symmetric block ciphers, thus doubling the key size can effectively enable to maintain security level.

The purpose of the AHX would be to offer a solution for embedded devices in the context of 5G and resistance to attacks possible on quantum computers.

It has been requested by itk AVtobvS Sàrl to Kudelski to provide some feedback regarding the security of the current proposal, that is the main goal of this report and it is not a code review.

INTRODUCTION

The first high level security analysis presented in this report is focused on the block cipher called AHX.

This report analysis (and also rev 1.02) has been conducted using only the source codes available on following links, with the revision 1.0.06g and which corresponds to what was delivered to Kudelski:

- https://github.com/Steppenwolfe65/CEX/tree/master/CEX/AHX.cpp v1.006f
- https://github.com/Steppenwolfe65/CEX/blob/master/CEX/AHX.h v1.006d
- https://github.com/Steppenwolfe65/CEX/blob/master/CEX/HKDF.cpp v1.006f
- https://github.com/Steppenwolfe65/CEX/blob/master/CEX/HMAC.cpp v1.006f

Others files might have been taken into account but due to short time constraint, it has not been possible.

Note that another similar block cipher proposal can be found in the CEX crypto library: SHX which is based on the Serpent block cipher. The difference is the underlying block cipher. AHX could beconsidered as the most interesting of such instances, because optimized hardware and processors are generally available for the AES block cipher.

1. AHX OVERVIEW

1.1. High level proposition

AHX seems to have a flexible design to enable the user to handle the required security level according to the context.

Key lengths from 128 to 1024 bits are handled. The supported key sizes in extended mode are fixed at 256, 512, and an experimental 1024-bit.

The key schedule generator options are limited to HKDF(SHA2-256/512) or cSHAKE-256/512/1024. This is set through the constructors BlockCipherExtension parameter. The cipher can process 128, 192, 256-bit and 512 bits keys in standard mode, and 256, 512, and 1024-bit keys in extended mode. There are no other legal key sizes, and using an unsupported key size will throw an exception.

The selection of the security level is linked to time, so the following question should be answered before the cryptographic primitives' selection: how long shall the data be safe?

One goal of this analysis will be to give first elements regarding the targeted security level.

In Figure 1 we have summarized the proposal.

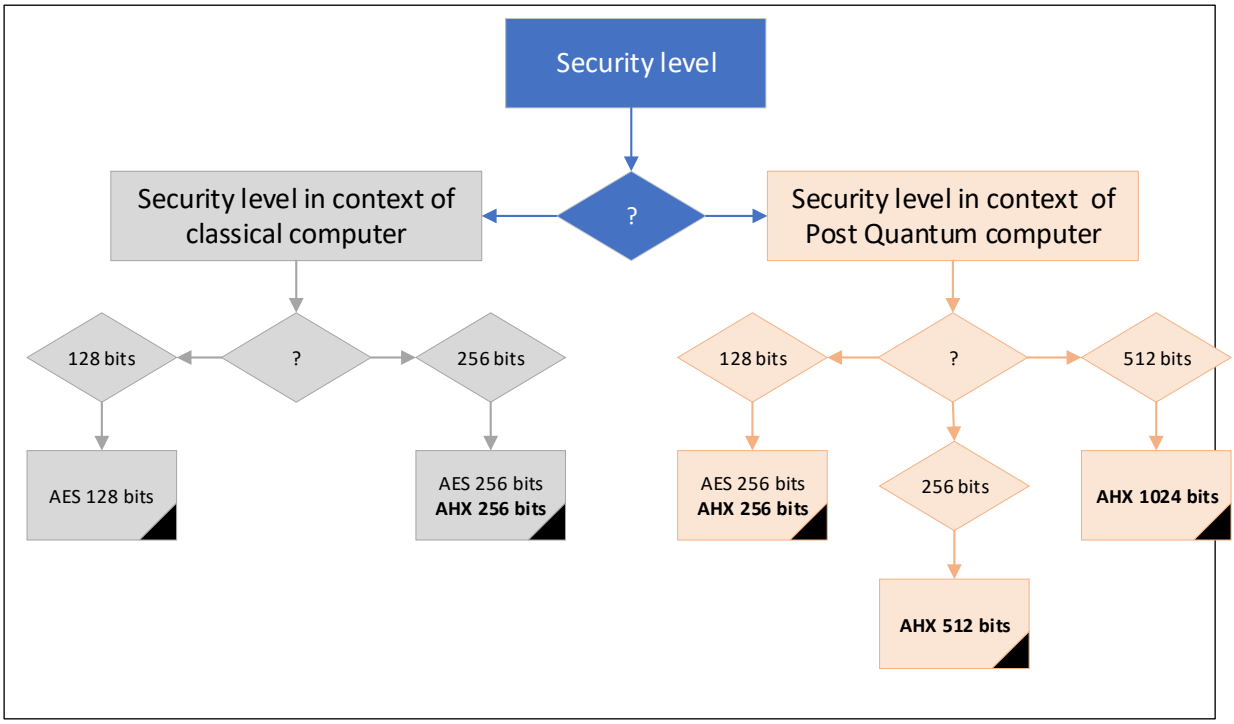

Figure 1: Security levels targeted according to algorithms

1.2. AHX Key schedule

AHX can use two different key schedules.

The first one is the standard AES key schedule, which is originally specified for AES for 128, 192 and 256-bit keys. AHX extends it to be able to process 512-bit keys. The length of the key determines the number of rounds (and round keys); given $k \leq 512$ the number of key bits, the number of rounds is given by $R_{nb} = \frac{k}{32} + 6$.

The second key schedule which is called Secure Expand is based on a key derivation mechanism that can be either HKDF based on HMAC with following hash primitives SHA2-256 or SHA2-512, or cSHAKE-256 or cSHAKE512 [22] which are based on Keccak permutation. In this key schedule context, the round keys are generated by calling iteratively the key derivation until all needed bytes are generated. The number of rounds in this case is given by $R_{nb} = Min\left\{\frac{k}{32} + 14, 38\right\}$.

Key length (in bits)	Hash function in the key schedule
256	SHA256
512	SHA256
1024	SHA512

Table 1: Hash functions used in the HKDF extended key schedule according to key length

Key length (in bits)	Hash function in the key schedule
256	cSHAKE256
512	cSHAKE256
1024	cSHAKE512

Table 2: Spongent functions in the extended key schedule according to key length

1.3. AHX Block cipher

AHX block cipher round is fully equivalent to AES one.

As defined in the key schedule based on HKDF, the number of rounds is defined according to the input key size.
Given $k \leq 1024$ the number of key bits: $R_{nb} = Min\left\{\frac{k}{32} + 14, 38\right\}$.

Key length (in bits)	Rounds numbers
256	22
512	30
1024	38

Table 3: Round numbers according to key length

Figure 2 below shows an overview of the AHX execution when using the HKDF-based key schedule with SHA256 underlying hash function and for key length that should not exceed 512 bits.

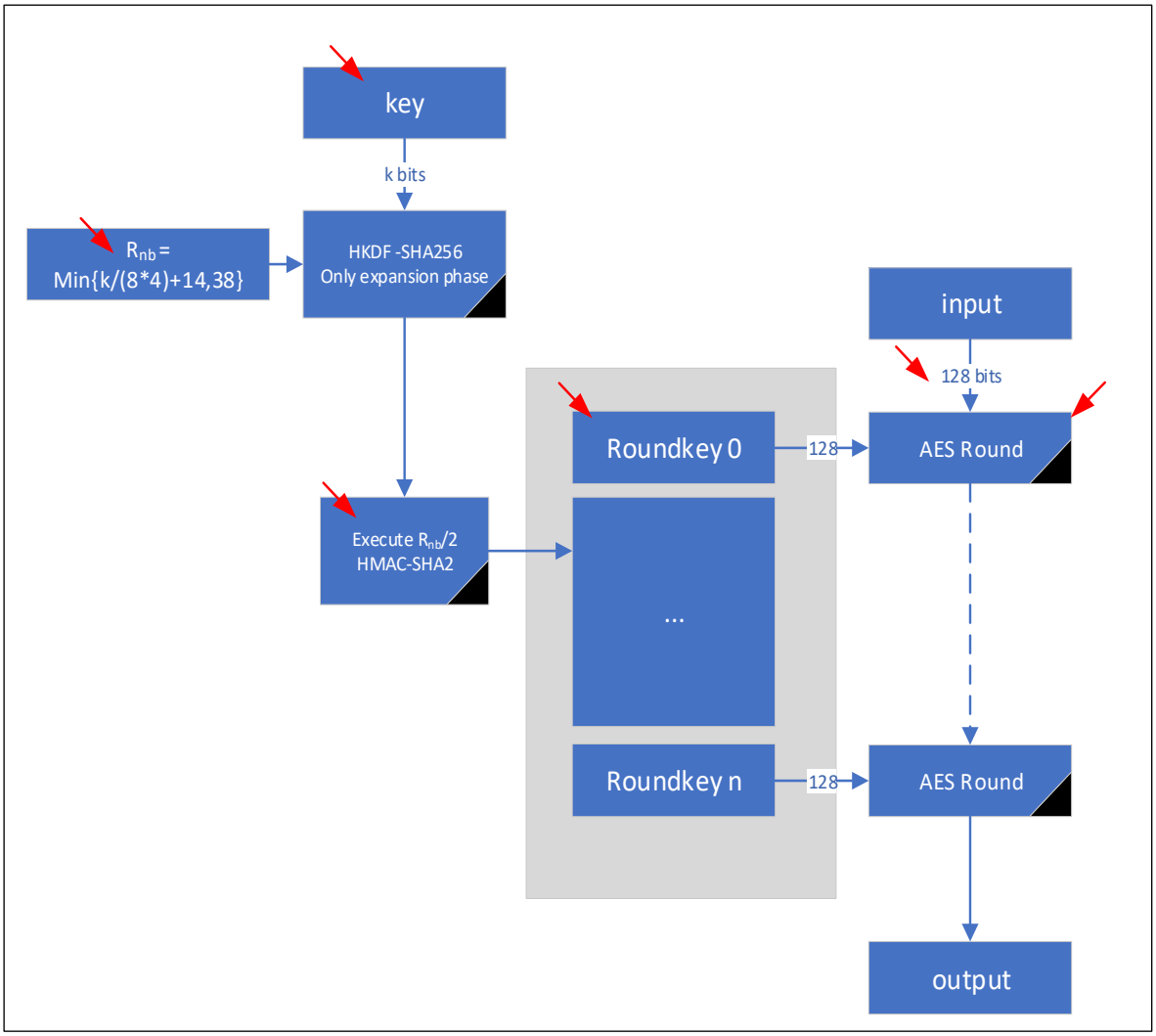

Figure 2: AHX global scheme

2. SECURITY ANALYSIS

In Figure 2 we have pointed in red the elements that must be handled carefully to guarantee the security level of the whole algorithm.

2.1. Consistency of the primitives

In the analyzed library CEX, it is proposed to manage the security consistency of the cryptographic primitives by the LegalKeySize() function in the code. The rounds are fixed at 22, 30, and 38, corresponding to key sizes 256, 512, and 1024-bit and the rounds count is not user-definable.

The construction proposes HKDF(SHA2) and cSHAKE hash functions for the key schedule. Any other choice will throw an exception.

Regarding the block cipher round the proposal should include other options, indeed, in the unlikely event that a weakness is found on the AES structure in the future, it might be

possible to have it whatever the key size and the key schedule used. The developer has also proposed the SHX block cipher, based on Serpent, to address these concerns.

2.2. Key

Regarding the key element and according to security level expected, a specific attention must be paid to the quality of the random number generator used to guarantee the expected entropy of the key.

The storage of the keys must manage them in way that it will guarantee confidentiality and integrity. The confidentiality is partially addressed by the fact that the cipher key can be stored in either a SymmetricKey or SymmetricSecureKey structure. The SymmetricSecureKey stores all data in encrypted arrays, either type of key structure can be used with the ISymmetricKey interface used by the Initialize function.
However, what is the source of the key used to do this operation and where it is stored is not known from the files reviewed.
In the analyzed files, the enforcement of the key size in bits is covered and using an unsupported key size will throw an exception.

2.3. Input Block size

A block cipher input block size should be such that no complete dictionary can be built for a given key. Nowadays it is quite common to set the block size to 128 bits, which is considered to completely protect from such brute force attacks.

Moreover, if we consider the input/output size of the block cipher, there are $(2^{128})!$ possible permutations. Whatever the key lengths considered 256, 512 or 1024 bits this is much larger than the key spaces, so collisions are not expected.

However, some academic results have alerted that it might not be enough to just double the key length of the symmetric primitives to block the attackers from the post-quantum world [8]. And given that the AHX is proposed to handle 512-bit security level, considering quantum resistance, the block size bit length should also be taken into account and solution to handle larger block size should be investigated. The developer mentions that the proposed solution is meant to be an intermediate drop-in replacement for AES. Larger block sizes are available in the CEX library but they implementation is not in the scope of this report. They are the Threefish and ChaCha20 algorithms. However, they do not fit one of the requirements of the proposal that is to use already existing hardware or software implementation.

The original Rijndael proposal already specifies the possible choice of 256-bit block size, possibility that has not been retained for the AES standard.

In the AHX proposal, the 256-bit block was removed as it would have required to generate twice as many round subkeys, and because of the weak diffusion characteristics of the expansion function. The 256-bit rounds function with its wider block and altered row-column shuffle, was also thought to be possibly introducing algebraic differences which might be exploited in some future attacks.

Nevertheless, we argue that the choice to switch to 256 bits block size should be available, although a thorough analysis of attack scenarios should be produced to justify it.

2.4. Key schedule

It appears important here to recall security breach published on AES 256 bits key. Indeed, it has been proved by researchers that AES-192 and AES-256 are weaker than AES-128 against some classes of attacks [6].
This can be explained at very high level by the fact that AES-256 tries to squeeze two times the key information into a construction that was essentially built for 128-bit keys, and this has serious side effects.
The attacks are related-key and related subkey attacks [19] which requires the cryptanalyst to have access to plaintexts encrypted with multiple keys that are related in a specific way -a scenario which is debatable. They are also very far to be practical.
However, this work casts serious doubts on the possibility to extend the AES to bigger key sizes, at least on theoretical grounds without changing its key schedule to match bigger key sizes.
This is precisely the motivation behind the proposal of the second AHX schedule, namely the one based on either cSHAKE or HKDF; the related key attack scenario is certainly not applicable to the second proposed AHX key schedule. In this case round keys are derived with a one-way cryptographic function, which has higher computational cost, but certainly renders the proposal more attractive.

HKDF is well known and studied. If the hash function is correctly chosen and implemented, it enables to produce many more round-keys keeping maximal entropy for each round key. The choice of the KDF used to generate the round keys essentially determines the upper bound on the security of the scheme. The HKDF is built on HMAC primitive, which in turn is based on a hash function.

Regarding cSHAKE based variants they rely on the pseudorandom property provided by Spongent construction that has also been deeply analyzed.

We remind here that the options are:

- HKDF(SHA2-256)
- HKDF(SHA2-512)
- cSHAKE-256
- cSHAKE-512
- and an experimental cSHAKE-1024

The choice poses constraint on the actual key space of AHX.

SHA2-256 and cSHAKE256 have claimed security of 256 bits against pre-image attacks (collision attacks are not meaningful in the KDF scenario) [Note that cSHAKE256 should be used in this case with at least 512-bit output].

This is coherent with the fact that the main key can be up to 512 bits long; in fact, Grover's algorithm [9] run on a quantum computer, would break such keys with a 2^{256} effort, which is

a value coherent with the choice of either SHA2-256 or cSHAKE256. Additionally, the Boolean Equation Solving break [20] proposes that AES-256 could be broken in 2^{74}.
Therefore, such choice is possible if the key of AHX is no longer than 512 bits.

For longer keys, one could choose either SHA2-512 or SHAKE512, but knowing that for the same motivations outlined above, in this case the key space of AHX cannot exceed 1024 bits.

Thus, in the context of quantum computers and considering an attacker could exploit one round key leakage to get the original key using pre-image attack, we consider that the proposal is consistent as it recommends the following:

- SHA256-cSHAKE256 for key up to 512 bits to guarantee a security level up to 256 bits
- SHA512-cSHAKE512 for key up to 1024 bits to guarantee a security level up to 512 bits

We think it would be desirable to limit AHX key length to 1024 bits and which is for the moment the case in the proposal.

It is important to mention that the one-way property of the second proposed key-schedule is an asset to protect the implementation against attacks; an attacker would need to get all the round-keys to break the full encryption function (if the original key is well protected and does not leak during first HMAC or Spongent function execution. This could be verified by side-channel leakage analysis of a physical implementation).

2.5. Rounds number

Examining AHX code, the round count number is implicitly determined by the key length, and via the KDF a different key length would in any case lead to different KDF output. We think that a more desirable method would be to let the number of rounds enter the KDF function as ancillary data. This would enforce the security in context of attack.

The formula for choosing rounds number is justified by the authors as based on the original Rijndael design:

$K_w = \frac{key\ byte\ size}{4}$ and $N_R = K_w + 6$

which is 22 rounds for a 512-bit key.

However, because 11 rounds have been broken by a related subkey attack, the authors [23] believe that 22 rounds or 2n the best attack should be the minimum and is applied to the 256-bit key instead. The 512-bit key uses an intermediate 30 rounds, and the 1024-bit key uses 38, which matches the original formula.

The amount of (round) key bits that can be introduced into a round of AES is 128. If the AES standard proposed 10 rounds for 128-bit keys, we would conservatively assume that AES with 256-bit keys should feature 20 rounds which catches up with applying the principle of 2n the best attack: 22 rounds for AES-256. That is aligned with the proposal.

If we conservatively keep the same principle for 512-bit key, assuming that 10 rounds would be needed for every 128 bits of key material, this would mean 40 rounds for 512-bit keys. It is important to mention here that in context of hardware implementation one AES round costs few clock cycles and in case of software implementation, the key schedule will be the most impacting part.

The formulas chosen by the AHX author are less conservative, as they prescribe 30 rounds for 512-bit keys using the KDF key schedule.

Deeper analysis should be conducted for 1024-bit key proposal.

2.6. Implementations

The AHX cipher can be implemented in software and dedicated hardware (ASIC and/or FPGA). It is based on well-known cryptographic primitives, and from a functional point of view should not pose problems for a skilled coder/designer.
According to the context, we want to mention that implementations should be secure against active (fault) and passive (side-channel) physical attacks (exploiting for example cache and/or timing differences).

The developer commented this as follows: “the permutation functions in both the Keccak and SHA2 implementations are by default unrolled (as is every permutation function in the library including Blake2, Skein, Threefish, ChaCha etc.). Both an unrolled and a compact form of the permutations are available and can be set per the definition of the CEX_DIGEST_COMPACT and CEX_CIPHER_COMPACT contained in CexConfig.h. The SHA2.h class also contains the SHA2-NI instructions which are selected automatically at run-time if available on the host CPU”.

While techniques for doing that are quite known in the literature, especially for software implementations, the addition of more cryptographic primitives (KDF, HMAC, hash functions and spongent function) that manipulate key material means that every one of these primitives should be secured against such attacks, making the complexity and therefore the cost of the implementation higher.
In addition, for pure software implementation specific attention must be paid to keys, round keys storage and entropy generation. According to the developer, the source of entropy is also mentioned, the default is ACP (Auto entropy Collection Provider) which uses a collection provided by the collection and concentration of every available entropy provider (RDRAND, RDSEED, CPU Jitter, and the system provider) along with hundreds of operating system timers and system unique values, all concentrated through cSHAKE (the strongest entropy provider found was selected, and improved to make it a lot stronger).

It is certainly possible to implement the AHX block cipher with KDF key schedule in a dedicated chip (ASIC) and programmable logic (FPGA). It would require deeper analysis regarding memory constraints, performances requirements and physical attacks of the whole cryptographic construction.

2.7. Additional Comments

The source code has been updated during the redaction of this report and is now available at the following links, with the revision 1.0.0.7c commited on December 13th 2018:

- https://github.com/Steppenwolfe65/CEX/tree/master/CEX/AHX.cpp v1.007c
- https://github.com/Steppenwolfe65/CEX/blob/master/CEX/AHX.h v1.007
- https://github.com/Steppenwolfe65/CEX/blob/master/CEX/HKDF.cpp v1.006f
- https://github.com/Steppenwolfe65/CEX/blob/master/CEX/HMAC.cpp v1.006f

DOCUMENT HISTORY

Version	Status	Date	Authors	Comments
01.05	Final	**29.04.2019**	**M. Macchetti, K. Villegas**	Approved version

Reviewer	Position	Date	Document Version
Benoit Gerhard	Head of Security Evaluation & Attacks	**29.04.2019**	01.05

Approver	Position	Date	Document Version
Benoit Gerhard	Head of Security Evaluation & Attacks	**29.04.2019**	01.05

DOCUMENT RECIPIENTS

Name	Position	Contact Information
Stiepan A. Kovac		**stie@itk.swiss**

KUDELSKI SECURITY CONTACTS

Name	Position	Contact Information
Benoît Gerhard		**benoit.gerhard@nagra.com**

REFERENCES

#	References Descriptions
[1]	https://github.com/Steppenwolfe65/CEX/tree/master/CEX
[2]	https://tools.ietf.org/pdf/rfc5869.pdf HMAC-based Extract-and-Expand Key Derivation Function (HKDF)
[3]	https://csrc.nist.gov/projects/post-quantum-cryptography
[4]	D. J. Bernstein (2009). "Introduction to post-quantum cryptography". (Introductory chapter to book "Post-quantum cryptography").
[5]	D.J. Bernstein, Cost analysis of hash collisions: Will quantum computers make SHARCS obsolete?, 2009
[6]	A. Biryukov and D. Khovratovich, Related-key Cryptanalysis of the Full AES-192 and AES-256, 2009
[7]	Joan Daemen and Vincent Rijmen, The Design of Rijndael, AES – The Advanced Encryption Standard, Springer-Verlag 2002 (238 pp.)
[8]	Xiaoyang Dong, Bingyou Dong, Xiaoyun Wang,Quantum Attacks on Some Feistel Block Ciphers, Journal of latex class files, VOL. 14, NO. 8, 2015
[9]	Grover L.K.: A fast quantum mechanical algorithm for database search, Proceedings, 28th Annual ACM Symposium on the Theory of Computing, (May 1996) p. 212
[10]	Grover L.K.: From Schrödinger's equation to quantum search algorithm, American Journal of Physics, 69(7): 769-777, 2001. Pedagogical review of the algorithm and its history.
[11]	Grover L.K.: QUANTUM COMPUTING: How the weird logic of the subatomic world could make it possible for machines to calculate millions of times faster than they do today The Sciences, July/August 1999, pp. 24–30.
[12]	Grover L.K, Quantum mechanics helps in searching for a needle in a haystack, Physical Review Letters 79 (1997), 325–328.
[13]	Grover L.K, Terry Rudolph, How significant are the known collision and element distinctness quantum algorithms ? Quantum Information & Computation 4 (2003), 201–206. MR 2005c:81037. URL: http://arxiv.org/abs/quant-ph/0309123
[14]	Peter W. Shor, Algorithms for quantum computation: discrete logarithms and factoring., in [7] (1994), 124–134.
[15]	Peter W. Shor, Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer, SIAM Journal on Computing 26, 1997
[16]	Paul C. van Oorschot, Michael Wiener, Parallel collision search with application to hash functions and discrete logarithms, in [2] (1994), 210–218
[17]	Paul C. van Oorschot, Michael Wiener, Parallel collision search with cryptanalytic applications, Journal of Cryptology 12 (1999), 1–28
[18]	John Underhill, CEX++ 1.0- An Introduction to the CEX Cryptographic Library, john.underhill@protonmail.com, July 03, 2017
[19]	https://eprint.iacr.org/2009/374.pdf Section 4.2
[20]	https://arxiv.org/pdf/1712.06239.pdf
[21]	https://www.schneier.com/blog/archives/2009/07/another_new_aes.html
[22]	https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-185.pdf
[23]	John Gregory Underhill1 and Stiepan Aurélien Kovac2, and Xenia Bogomolec, Towards post-quantum symmetric cryptography 2018

Table 4: References