

# Towards post-quantum symmetric cryptography

*John Gregory Underhill<sup>1</sup> and Stiepan Aurélien Kovac<sup>2</sup>, and Xenia Bogomolec<sup>3</sup>*

- 1) *itk AVtobvS SARL, ch. de Monséjour 2, 1700 Fribourg, Switzerland*  
[john.underhill@protonmail.com](mailto:john.underhill@protonmail.com) · <http://itk.swiss>
- 2) *itk AVtobvS SARL, ch. de Monséjour 2, 1700 Fribourg, Switzerland*  
[stie@itk.swiss](mailto:stie@itk.swiss) · <http://itk.swiss>
- 3) *SOC Hessische Landesbank, 60311 Frankfurt am Main, Germany*  
[indigomind@protonmail.com](mailto:indigomind@protonmail.com) · <http://coder.tjingwan.com>

**Abstract.** With this work, we intend on demonstrating the need for improvements to the currently standardized AES family of cryptosystems, and provide a solution that meets the requirements of long-term security in the rapidly evolving threat landscape. The solution proposed is flexible, dramatically increases the potential security of the cipher, and strongly mitigates many of the most serious attacks on the AES family of cryptosystems. Further, our solution can be easily integrated into existing AES cryptosystem deployments, with only a few small changes required, thus preserving the large investments in this cipher both in hardware and software.

**Keywords:** Symmetric · cryptography · quantum · AES

## 1 State of the art

### 1.1 AES

The Advanced Encryption Standard (AES) is a specification for the encryption of electronic data. The AES selection competition was held in 1998 by the National Institute of Science and Technology (NIST), which after a period of study from the academic community, a subset of the Rijndael family of symmetric ciphers was selected as the AES competition winner.

In 2001 the adoption of AES as a replacement for the DES family of ciphers, was formally proposed with the release of the FIPS document FIPS PUB 197 [1]. Subsequently, the AES family of cryptosystems has been adopted by countries around the world, and has been near universally implemented, and is now the most widely used and critically important symmetric encryption scheme in use today.

AES is in fact, implemented on most of the computers in the world, and is the backbone of the worlds secure communications systems, not only as the primary encryption cipher used by e-commerce and secure communications applications, but also in the many and various diverse use case scenarios such as VPN technology, disk drive encryption, and secure databases.

It is for this reason, that this technology must be kept secure against future threats, for if a serious break in AES were ever discovered, the result would be nothing short of catastrophic, and result in the total and absolute compromise of the worlds secure communications infrastructure.

## 1.2 The need for change

In cryptography we adjust a cipher's strength according to what we consider to be its resistance to attack. For example, if we know that 64 bits of key can be broken, we set the key length to be at least  $2n$ , or  $3n$  that size. If we know that 10 rounds of a rounds-based iterative block cipher can be broken, we set that cipher to twenty or even thirty rounds. This is an assurance we provide that the technology is well beyond the current and expected future technological capabilities required to break that cipher.

Rijndael was not chosen for its security properties, but largely for reasons of its superior performance. Other leading candidates; Serpent and Twofish each scored significantly higher in their security evaluations, with it estimated that Rijndael variants would require being set at between 18 to 24 rounds only to equal the estimated security of the Twofish family of ciphers [2] (rather than the 10 to 14 rounds defined by NIST), and Twofish was in turn considered to be significantly less secure than the Serpent cipher.

The last prolonged and intensive public study made of Rijndael was during the AES standardization competition, and that was nearly 20 years ago. Since that time, many advances in cryptanalysis and computing technology have been made, the rate of technological progress has accelerated, and the immediate future promises to provide changes to our technological capabilities that far exceed what was even considered possible just a few years ago.

Since the AES selection process, many new forms of attack have been discovered and others greatly improved upon; some side-channel attacks that target subtle variations in a cipher's execution timing, have been proven capable of breaking AES under real world conditions [3][4][5]. Other attacks that target the differentially-weak key schedule [6][7], particularly in AES-256, have also raised serious questions about the suitability of AES in a long-term security context.

The authors of one of these related subkey attacks, which has to date broken 8 out of 10 rounds of AES-128 and 11 of 14 rounds of AES-256, pose the question of the currently understood AES safety margins [7]:

*"While neither AES-128 nor AES-256 can be directly broken by these attacks, the fact that their hybrid (which combines the smaller number of rounds from AES-128 along with the larger key size from AES-256) can be broken with such a low complexity raises serious concern about the remaining safety margin offered by the AES family of cryptosystems."*

Though these attacks are currently not known to be able to break a full 10, 12, or 14 round version of AES, they do prove two things conclusively: that the weak internal key expansion function is a serious and exploitable flaw in the cipher design, and that the number of mixing rounds is set too low.

Bruce Schneier has made public comments to this effect, repeatedly urging the community to consider increasing the number of rounds, and acknowledging this serious flaw in the ciphers design [8]:

*"Cryptography is all about safety margins. If you can break  $n$  round of a cipher, you design it with  $2n$  or  $3n$  rounds. What we're learning is that the safety margin of AES is much less than previously believed. And while there is no reason to scrap AES in favor of another algorithm, NIST should increase the number of rounds of all three*

*AES variants. At this point, I suggest AES-128 at 16 rounds, AES-192 at 20 rounds, and AES-256 at 28 rounds. Or maybe even more; we don't want to be revising the standard again and again.”*

It must also be considered, that these are only the attacks that have been discovered in the academic community and as such are publicly known. For example, the NSA likely spends more on cryptanalysis than all of the universities in the world combined, they employ many of the best cryptanalysts, cryptologists, mathematicians and engineers in the world, and have been actively engaged in trying to break AES for nearly twenty years.

Quantum computers are now a reality. Engineering problems are being solved, working prototypes are being built, and their capabilities are rapidly improving. Many of the largest companies in the technology sector are now investing billions of dollars into these emerging technologies, along with state agencies from the most powerful countries in the world.

Some estimates are that these machines may be able to break the most widely used asymmetric ciphers in as little as five years [9].

They pose a threat to symmetric ciphers as well, with the potential to halve the key space, making the 128-bit version of AES effectively obsolete.

A new paper proposes a quantum algebraic attack against AES using Boolean equation solving [10] estimates that even greater reductions to the key space may be possible.

A lesson we have learned time and again, is that attacks only improve, and new attacks against symmetric cryptography using quantum computers will almost certainly be discovered once these computers and their properties are better understood.

Lastly, we must consider what we can not predict, the *unknown unknowns*. Some experts believe that we are within twenty years of the technological singularity; the point at which we create an independent machine consciousness more powerful than our own. Beyond this point, it will quickly become impossible to predict the future evolution of our technological capabilities, for by its very definition, we will have created an alien intelligence, which by the end of the century could evolve to become many orders of magnitude more powerful than human consciousness.

The definition of long-term security has changed in the last ten years. Historically, for data to be considered secure it only required that it should remain unreadable for so long as that information was relevant, typically ten to twenty years.

A great deal of evidence has been accumulating that strongly indicates that state sponsored intelligence agencies have become engaged in the collection and long-term storage of much of the worlds secure communications traffic. In parallel, laws that cater for the evolution of genetic medicine, ask for lifetime data protection, such as Germany's patient data protection law (“even after death of the patient”), setting a precedent in this domain to the European Union's GDPR, that now sees global effects.

It is in response to this change in the threat landscape, that we must in turn create and adopt more powerful and flexible encryption technology, to ensure that data is not protected only for the foreseeable future, but beyond what can be estimated or predicted given our current knowledge. We must now answer this present and

evolving threat, and attempt to ensure that sensitive information can be kept secure for an entire lifetime.

### 1.3 The rules of the game

Nearly twenty years after the standardization of AES; created as a replacement for the then popular DES family of ciphers, we can still find DES in widespread use. DES and the strengthened version of the cipher, Triple DES, are still widely used in the electronic payment industry, various software applications, aging TLS configurations and VPNs [11] and integrated into popular browser software.

This failure to update our systems and software to the currently held stronger standards of encryption technology is well understood, and supported by historical trends in the software industry.

The standardization process for a replacement for AES, if judging by past standardization efforts, (including the NIST Post Quantum asymmetric call, which could take three to five years to complete), could take as long as five years.

The actual changeover to a new symmetric cryptosystem given that AES is near universally implemented, and integrated into every level of our communications systems and encryption software componentry, could take another twenty years to be realized on such a massive scale.

Clearly this is completely unacceptable, particularly given the ample evidence that state agencies have begun harvesting the worlds secure communications traffic and can store that data indefinitely for future analysis.

There is more than sufficient cause to believe that AES is not as secure as we once believed it to be, and that given the massive investments into cryptanalysis by state funded agencies, and that breaking the AES cryptosystem represents the most fundamentally important target of their efforts, it becomes clear that improvements to the AES design that counter known weaknesses and provide greater security is becoming of paramount importance.

A simple formula demonstrates this:

$$x + y \leq z$$

If  $x$ , the number of years we require a cipher to be secure, added to  $y$ , the time it takes to deploy a new cipher, exceeds  $z$ , the amount of time before a cryptanalytic or technological breakthrough might occur, or beyond the time when we may reasonably predict the future state of technology, then we have surpassed the secure lifetime of that cryptographic cipher. We believe that we have already passed that point in time.

The history of our industry is littered with broken ciphers and cryptographic protocols, in fact the history of science is one of theories, once thought concrete and inviolable, swept aside in the advent of new discovery. Newtons law of gravity held for nearly two hundred and fifty years, until a patent clerk in Switzerland proved it to be flawed.

We need to learn from that history, and we need to prepare for what promises to be an uncertain and increasingly unpredictable future, because the security of our communications systems now represents the security of nations, the sustainability of free and democratic states, and the security of all the people of the world.

## 2 A way forward

### 2.1 eAES

What we propose is an intermediate solution. If in fact we have exceeded the reasonable security lifetime of the existing form of the AES cryptosystems, and a new solution can not be widely implemented before such time as these ciphers might be broken, then we believe that an interim solution is the best current option.

This solution involves two steps:

1. The replacement of the differentially-weak internal key expansion function, the ‘key schedule’, with a cryptographically-strong pseudo random generator.
2. Increasing the number of transformation rounds to at least **2n**, or at least twice the best-known cryptanalytic attack against the AES ciphers.

The key schedule is essentially a key expansion function, that expands a small input cipher key, into a much larger internal array of round-keys, used by the ciphers transformation function to create a cipher-text output unique to that key. In the Rijndael cipher, this key expansion function is the weakest part of the construction, it does not produce cryptographic-quality output, and has been the target of several serious related-subkey and timing-based attacks on the cipher. We propose that this function be replaced with a cryptographically-strong pseudo-random generator.

We have produced a model implementation with this change; **RHX** (and the companion AES-NI based **AHX**) [12], which uses either the **HKDF(SHA2) Expand** key derivation function, or a choice of the Keccak based **cSHAKE XOF** function. We have also produced an example C based primitive **RSX** [13] which uses a fixed implementation of the Keccak extended output function **cSHAKE-256**. Both implementations use keyed pseudo-random generators that are widely accepted as producing highly diffused output and are considered to be strong cryptographic-quality generators.

Besides producing a more cryptographically secure output, (used to generate Rijndael’s internal rounds sub-key array), replacing the key schedule with a cryptographically-strong generator also allows for the safe addition of more transformation rounds by securely generating the required longer subkey array. Using an existing and well-regarded strong generator also eliminates the need to modify the existing key expansion function, and replacing it with an *ad hoc* adjustment which could prove to be insecure, and would require substantial study before it could be used safely.

We have chosen both the HKDF(SHA2) Expand function and the Keccak based cSHAKE, because these are both widely regarded within the cryptographic community as being cryptographically-strong pseudo-random generators.

The HKDF version of these ciphers is inter-transitional, that until Keccak and its cSHAKE derivative have seen more widescale usage, and hence more scrutiny from our community, that some implementors may prefer the HKDF generator.

The HKDF variant can use either the SHA2-256 or the SHA2-512 version of the hash function as the primary pseudo-random function, and the cSHAKE variants have the cSHAKE-256 and the [experimental] cSHAKE-512 options in our C++ implementations [12].

Each variant will produce a completely different output cipher-text, and so must be considered as unique versions of the cipher extension. Additionally, we have added an information string (the distribution code property), that can be set in the cipher implementations as a cipher-tweak, this user definable string can be used to safely produce a unique cipher-text output.

By using these variations, and in fact a flexible model promoting interchangeability throughout our library implementation, we aspire to create a real-time upwardly-flexible security paradigm. One in which within the context of a broader domain-based communications system, the security of data transmissions can be actively modified during a hand-shake negotiation; parameters can be set at run-time, even the ciphers and protocols can be interchanged, guaranteeing the best possible security profile can be achieved and maintained.

The rate of technological progression has accelerated dramatically over the last century, and continues to expand, the advent of strong AI coupled with emerging quantum-based computing systems may serve to vastly accelerate our technological development, and with this in mind, we believe that a security-flexible core cryptographic library is essential to maintaining sustainable long-term security.

We also propose an increase in the number of transformation rounds; the number of mixing cycles applied to the state.

We will not consider the AES-128 or AES-192 members of the AES family of cryptosystems, because currently known attacks using quantum computers [10][14] will one day be able to break these variants, rendering them thus unsuitable for the purposes of long-term security.

The **eAES**, (the formal name of the proposed extension to AES) **256-bit** variant is set to **22 rounds**, or twice the known number of rounds broken by related subkey attacks. This is an increase of 8 rounds, from the 14 rounds used currently by the NIST standardized form of AES-256.

We have produced a **512-bit** key variant, that is set to **30 rounds**. There is a great deal of resistance within our industry to the use of 512-bit keys, many cryptographers consider them as an unnecessary key length, but one must consider what these assumptions are predicated upon; that known quantum attacks that halve the key space will never be improved upon, that no new attacks quantum or cryptanalytic based will ever be discovered, and that we can accurately predict the future of our technological development for at least fifty years.

We believe the fault in these arguments to be clear and self-evident, and that if we are to establish real long-term data protections, we must become better at anticipating the unknown.

The library also contains a 1024-bit variant which is set to 38 rounds, provided for the purposes of future experimentation.

What we are bringing to your consideration, is a working and provably-secure solution to the question of the AES ciphers sustainable long-term security. It would use the existing rounds function, the core component of this cipher, and so preserve the large investments that have been made in this technology, while strongly mitigating many of the most serious known attacks against the AES family of cryptosystems.

It would require only that existing implementations substituted the key schedule's expansion function with a cryptographically-secure alternative, and provide the

increase in rounds necessary to restore the ciphers current security margins to acceptable levels.

The performance penalty incurred by the key schedule when using a cryptographically-strong generator is a negligible one-time penalty for each encrypted stream and so does not significantly impact the transformation of medium to large data sets.

The increase in rounds will have a small effect on the speed of the cipher, but this must be considered as a reasonable cost, required by the necessary increase in the security of the cipher. We have been using AES for almost twenty years, and so transitioning to versions with increased security should not be wholly unexpected, and the relative cost incurred is still negligible, especially when considering the enormous increase of hardware speeds and technologies such as embedded instructions, that have more than compensated for this small loss in performance.

## 2.1 Pseudo code

### The AES rounds function:

```
Transform(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin

    byte state[4,Nb]
    state = in
    AddRoundKey(state, w[0, Nb-1])

    for round = 1 step 1 to Nr-1
        SubBytes(state)
        ShiftRows(state)
        MixColumns(state)
        AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
    end for

    SubBytes(state)
    ShiftRows(state)
    AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])
    out = state

end
```

**Figure A.1:** Note that the rounds function is identical to the AES specification. The number of rounds processed in the main loop is determined by the value of **Nr**.

### The secure key expansion function:

```
SecureExpansion(byte key[4*Nk], word w[Nb*(Nr+1)], word Nr)
begin

    word ks = Nb * (Nr + 1) * 4
    byte tmp[ks]
    tmp = HKDF-Expand(key, dc, ks)
```

```

while (i < ks)
  w[i/4] = word(tmp[i], tmp[i+1], tmp[i+2], tmp[i+3])
  i = i+4
end while
end

```

**Figure A.2:** The octet key size **ks** is calculated as the ciphers block size in 32-bit words **Nb**, times the number of rounds **Nr** + 1, times the number of octets in each 32-bit word.

The temporary array of bytes **tmp** is generated by HKDF Expand (or alternatively cSHAKE) using the input cipher key and the optional distribution code **dc** arrays as input. This array of octet sized integers is then converted to 32-bit words and added to the rounds subkey array **w**.

## 2.2 Horizon 2040 and beyond

We believe the future of symmetric ciphers is in authenticated stream ciphers. They are fast, trivial to parallelize, easy to use and thus less likely to incur implementation mistakes, and as we have learned, authentication has become an essential aspect to ensuring data integrity.

We have implemented several authenticated stream ciphers based on the Threefish and ChaCha ciphers, which can use either the Keccak based **KMAC**, or **HMAC(SHA2)** message authentication code generators, the option selected through a constructor setting. These ciphers have been strengthened in keeping with our mission to provide real and sustainable long-term security, and are implemented in C++ using CPU intrinsics and optional multi-threaded parallelization [12].

These stream cipher primitives have also been implemented with 512-bit keys, (and in the case of Threefish, a 1024-bit key), and an increase to the number of permutation rounds in the 512-bit and 1024-bit key versions, in accordance to our currently projected security levels that we feel will be required for periods exceeding the 2040 horizon.

## 2.3 Cryptanalysis

In annex, a first cryptanalysis of the suggested engineering changes to AES, done by researchers at Kudelski AG in Chéseaux, Switzerland, is provided.

## 2.4 Future work

A performance analysis of the suggested changes to AES will be published in a later phase on: [git.fsfe.org/Stie/Pqsym](https://git.fsfe.org/Stie/Pqsym). Likewise and as has been suggested by Kudelski in their report, physical attacks on hardware implementations of eAES will be studied.

## References

- 1) FIPS 197: ADVANCED ENCRYPTION STANDARD (AES)  
NIST, November 2001  
<https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.197.pdf>
- 2) Improved Cryptanalysis of Rijndael



- Niels Ferguson, John Kelsey, Stefan Lucks, Bruce Schneier, Mike Stay, David Wagner, and Doug Whiting, 2000  
<https://www.schneier.com/academic/paperfiles/paper-rijndael.pdf>
- 3) A Diagonal Fault Attack on the Advanced Encryption Standard  
Dhiman Saha, Debdeep Mukhopadhyay, and Dipanwita RoyChowdhury, 2009  
<https://eprint.iacr.org/2009/581.pdf>
- 4) Cache Games – Bringing Access-Based Cache Attacks on AES to Practice  
Endre Bangerter, David Gullasch, Stephan Krenn, 2010  
<https://eprint.iacr.org/2010/594.pdf>
- 5) Design, Implementation and Performance Analysis of Highly Efficient Algorithms for AES Key Retrieval in Cache Access Attacks  
Ashokkumar C, Ravi Prakash Giri, Bernard Menezes, 2017  
<https://ieeexplore.ieee.org/document/7467359>
- 6) Super-Sbox Cryptanalysis: Improved Attacks for AES-like permutations  
Henri Gilbert and Thomas Peyrin, 2009  
<https://eprint.iacr.org/2009/531.pdf>
- 7) Key Recovery Attacks of Practical Complexity on AES Variants With Up To 10 Rounds  
Alex Biryukov, Orr Dunkelman, Nathan Keller, Dmitry Khovratovich, and Adi Shamir, 2009  
<https://eprint.iacr.org/2009/374.pdf>
- 8) Another New AES Attack  
Schneier on Security, July 30, 2009  
[https://www.schneier.com/blog/archives/2009/07/another\\_new\\_aes.html](https://www.schneier.com/blog/archives/2009/07/another_new_aes.html)
- 9) IBM warns of instant breaking of encryption by quantum computers: 'Move your data today'  
<https://www.zdnet.com/article/ibm-warns-of-instant-breaking-of-encryption-by-quantum-computers-move-your-data-today/>  
Arvind Krishna, director of IBM Research, May 18, 2018
- 10) Quantum Algorithms for Boolean Equation Solving and Quantum Algebraic Attack on Cryptosystems  
Yu-Ao Chen and Xiao-Shan Gao, 2018  
<https://arxiv.org/pdf/1712.06239.pdf>
- 11) RFC 8221: ESP and AH Algorithm Requirements, p. 7  
IETF, October 2017  
<https://tools.ietf.org/html/rfc8221#page-7>
- 12) The CEX Cryptographic library in C++  
John G. Underhill, last retrieved: November 23, 2018  
<https://github.com/Steppenwolfe65/CEX>
- 13) RSX: Rijndael-cSHAKE hybrid  
John G. Underhill, last retrieved: November 23, 2018  
<https://github.com/Steppenwolfe65/RSX>
- 14) A fast quantum mechanical algorithm for database search  
Lov K. Grover, 1996  
<https://arxiv.org/pdf/quant-ph/9605043.pdf>



**itk AVtobvS Sàrl**

## **AHX-Preliminary Security Analysis**

Symmetric block cipher proposition for Post-quantum cryptography

KUDELSKI SECURITY  
01.12.2018  
Confidential

## 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 post-quantum computer.

It has been requested by itk AVtobvS Sàrl to Kudelski to provide some feedbacks regarding the security of the current proposal, that is the main goal of this report.

## INTRODUCTION

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

The analysis has been conducted using only the source codes available on following links:

- <https://github.com/Steppenwolf65/CEX/tree/master/CEX/AHX.cpp>
- <https://github.com/Steppenwolf65/CEX/blob/master/CEX/AHX.h>
- <https://github.com/Steppenwolf65/CEX/blob/master/CEX/HKDF.cpp>
- <https://github.com/Steppenwolf65/CEX/blob/master/CEX/HMAC.cpp>

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

Note that other similar block cipher proposals can be found in the CEX crypto library: the comments in the analyzed code mention THX which is based on the Twofish block cipher, and SHX which is based on the Serpent block cipher. The difference is the underlying block cipher. According to the developer, Twofish is no longer present in the latest revision of the code and has been replaced by authenticated stream cipher implementations of Threefish 256/512/1024. AHX is indeed 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 is handled, but there no boundary check on key length. According to the developer, the supported key sizes in extended mode are fixed at 256, 512, and an experimental 1024-bit.

For a 2048-bit use-case the hash function to use in the key schedule is not consistent to the security level expected or is missing. According to the developer, there is no 2048-bit key option and the key schedule generator options are limited to HKDF(SHA2-256/512) or cSHAKE-256/512/1024. This is set through the constructors CipherExtension parameter. The cipher can process 128, 192, and 256-bit 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 also linked to time and before primitive selection following question should be answered: 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.

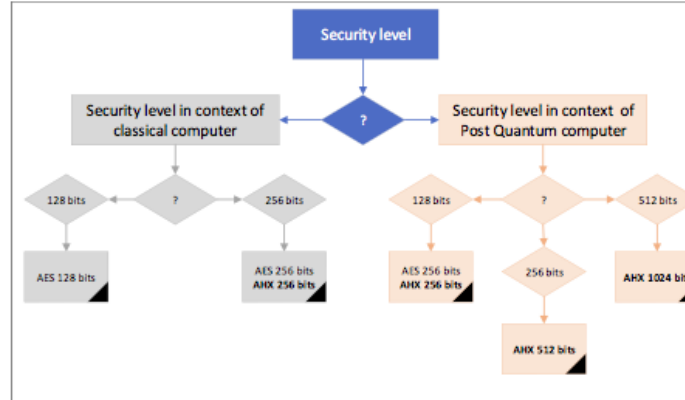


Figure 1: Security levels targeted according to algorithms

## 1.2. AHX Key schedule

AHX can use two different key schedules.

The first one is the standard AES key schedule, which is fully 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 described the comments of the analyzed code is a HKDF based on HMAC with selectable hash primitive. According to the developer there are actually 5 generator options: HKDF(SHA2-256), HKDF(SHA2-512), cSHAKE-256, cSHAKE-512, and an experimental cSHAKE-1024. The chosen hash function can be any of those implemented by the CEX library (e.g. Blake2, Skein, SHA3). According to the developer the only options for hash function with either HKDF, HMAC, or these ciphers is SHA2-256 or SHA2-512, and cSHAKE uses the Keccak permutation. Using a non-SHA2 hash function with HMAC or HKDF will throw an exception. By looking at the code we can see that possible choices are SHA-2 (256 & 512) and the more recent SHAKE-256 and SHAKE-512 hash functions. The round keys are generated by calling iteratively the HKDF 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 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:  $R_{nb} = \text{Min}\left\{\frac{k}{32} + 14, 38\right\}$ .

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

Table 2: 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.

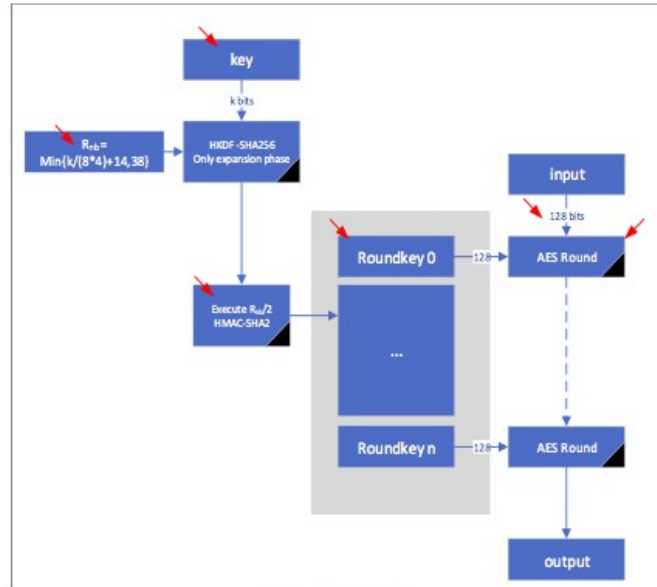


Figure 2: AHX global scheme

## 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 `LegalRounds()` and `LegalKeySize()` functions in the code, it might be interesting to enforce that. According to the developer the `LegalRounds` property accessor is no longer a part of these ciphers. 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 others hash functions for the key schedule and should include the alternatives for the block cipher rounds construction. According to the developer only HKDF(SHA2) and cSHAKE are implemented. Any other choice will throw an exception.

Indeed, if a weakness is found on the AES structure in the future, it might be possible to have it whatever the key size.

## 2.2. Key

Regarding the key element and according to security level expected, a specific attention must be paid to following elements and which are partially or not covered by the proposal:

- The quality of the random number generator used to guarantee the expected entropy of the key.
- The secure storage of the key including confidentiality and integrity.

In the analyzed files, the enforcement of the key size in bits is partially or not covered. According to the developer the key sizes are now enforced. 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 (even 2048 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 (and maybe more) - 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, hence Rijndael with a 256 block size is problematic. Larger block sizes are available with the Threefish and ChaCha standard and extended stream ciphers.

We argue that maybe the choice could be to switch to 256 bits block size, although a thorough analysis of attack scenarios should be produced to justify the choice.

We note that the original Rijndael proposal already specifies the possible choice of 256-bit block size, possibility that has not been retained (or judged valuable) for the AES standard.

## 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<sup>1</sup> 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.

This can have serious implications for the first key schedule of AHX, namely the one fully based on AES but extended to accept 512-bit keys. According to the developer, the 512bit key option using the Rijndael key schedule was an artifact, and not recommended in the literature. It has been removed from the cipher. In our opinion the author of AHX would have to show or at least mention that such attacks are not feasible against its proposal of extended AES key schedule.

It is not possible for us to validate the hypothesis, given the limited time available for this preliminary analysis.

The related key attack scenario is certainly not applicable to the second proposed AHX key schedule, namely the one based on HKDF. 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.

In the code comments, we can clearly see the following choices mentioned:

- SHAKE256
- SHAKE512
- HKDF-SHA2-256
- HKDF-SHA2-512

According to the developer, 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 SHAKE256 have claimed security of 256 bits against pre-image attacks (collision attacks are not meaningful in the KDF scenario) [Note that SHAKE256 should be used in this case with at least 512-bit output. According to the developer the cSHAKE

<sup>1</sup><https://eprint.iacr.org/2009/374.pdf> Section 4.2

function will undergo the same number of permutation cycles regardless if the output is less than the rate of 1088 bits per cSHAKE-256].

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 SHAKE256. Additionally, the Boolean Equation Solving break 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 post quantum computer and considering an attacker could exploit one round key leakage to get the original key using pre-image attack, we would recommend the following:

- SHA256-SHAKE256 for key up to 512 bits to guarantee a security level up to 256 bits
- SHA512-SHAKE512 for key up to 1024 bits to guarantee a security level up to 512 bits

It is at this moment unclear how one would attain a key space for AHX longer than 1024 bits. We think it would be desirable to limit AHX key length to 1024 bits. According to the developer key length is fixed at 1024-bit maximum.

The info and salt elements that are provided to the HKDF are less critical but could bring additional entropy if needed. According to the developer the HKDF Expand function is used in these ciphers, only HKDF Extract uses the salt input parameter, which would reduce efficiency with an extra permutation call, and the halved output of that call would mean IPAD/OPAD arrays in HMAC are only half populated with p-rand.

The one-way property of the second proposed key-schedule is an asset to protect the implementation regarding 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 execution).

## 2.5. Rounds number

The formula for choosing round number should be justified. It is in general quite dangerous for a block cipher to let the user decide the number of rounds, because that would lead to trivial attacks (differential analysis of  $n$  and  $n+1$  rounds). According to the developer It is justified per the original Rijndael design:

$Kw = \text{key byte-size} / 4$

$Nr = Kw + 6$

which is 22 rounds for a 512-bit key. However, because 11 rounds have been broken by a related subkey attack, he believes that 22 rounds or  $2n$  the best attack should be the

<sup>2</sup> <https://arxiv.org/pdf/1712.06239.pdf>

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. If more than 38 rounds are ever required, the cipher should be abandoned at that point.

Examining AHX code, it seems that round number is implicitly determined by the key length, so it seems correct (not freely selectable by user), and via the KDF a different key length would in any case lead to different KDF output. We only mention that in source code comments it is reported that number of rounds is selectable by user, which seems to contradict what has been said.

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. According to the developer the rounds counts are fixed.

Regarding the number of rounds, we would like to state the following: 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. According to the developer using at least  $2n$  the best-known attack, AES with 256-bit keys should feature 22 rounds.

The AES standard dictates 14 rounds for 256-bit keys, and in fact we have seen that its security is far from ideal under certain scenario (related key attacks).

If we conservatively assume that 10 rounds would be needed for every 128 bits of key material, this would mean 40 rounds for 512-bit keys. The developer mentions Bruce Schneier who recommended 20-28 for a 256-bit key<sup>3</sup>. 40 rounds seems excessive to him and might impact performance, particularly on mobile devices,

The formulas chosen by the AHX author are less conservative, as they prescribe:

- 30 rounds for 512-bit keys using the KDF key schedule
- 22 rounds for 512-bit keys using the extended AES key schedule. Note that according to the developer, that option has been removed from the cipher design, it was only there for historical purposes, and not recommended in the literature.

The choice seems particularly non-conservative for the extended AES key schedule. As we already discussed, the KDF key schedule seems a more robust proposal overall.

Deeper analysis should be conducted for 1024-bit and 2048-bit keys use cases.

## 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.

<sup>3</sup> [https://www.schneier.com/blog/archives/2009/07/another\\_new\\_aes.html](https://www.schneier.com/blog/archives/2009/07/another_new_aes.html)

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 if available. The HMAC and HKDF implementations should be protected from side-channel attacks. Every permutation in the library has both a compact permutation used for memory constrained devices, and a version of the function that has been completely unrolled and written for timing-neutrality, this includes each hash function (Blake2, Keccak, SHA2 and Skein), as well as symmetric cipher permutations (ChaCha and Threefish), with the choice of permutation function defined in CexConfig.h. The SHA2-256 implementation also defaults to the SHA-NI implementation if those instructions are available on the 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) 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 block cipher.