Isogeny-based cryptography

Chloe Martindale, Technische Universiteit Eindhoven
Lorenz Panny, Technische Universiteit Eindhoven

chloemartindale@gmail.com
lorenz@yx7.cc


Introduction

Quantum computers threaten to break most of the cryptography we are currently using to secure critical computer systems such as the internet. A quantum computer is a machine which employs quantum-physical phenomena to perform computations in a way that's fundamentally different from a "normal", *classical*, computer. Whereas a classical computer is, at any point in time, in a fixed *state* — such as a bit string representing its memory contents — the state of a quantum computer can be a "mixture", a so-called *superposition*, of several states. Note that the internal state is *hidden*: The only way to get information about the state is to perform a *measurement*, which will return a single non-superimposed *classical* output, such as a bit string, that is *randomly distributed according to the internal state*, and the internal state gets replaced by the measurement outcome. For example, when measuring an equal superposition of the two-qubit states $|00\rangle$ and $|11\rangle$, the result would be one of the bit strings 00 or 11 with probability $1/2$ each. Now a *quantum algorithm* consists of applying a carefully crafted sequence of operations to the internal state of a quantum computer in order to amplify the desired piece of information in the superposition, followed by measurements to extract the result.

The extra computational power thanks to the ability to store and manipulate *superpositions* of states allows for more efficient algorithms to tackle some computational problems. Note that contrary to a common misconception, quantum computers are *not* known to provide massive speedups over classical computers for "many", or even "all", tasks; in fact, there is only a handful of problems where known quantum algorithms outperform the best currently known classical algorithms. Unfortunately, many of these problems are at the heart of today's cryptographic systems; we shall see an example of this in the next section.

To deal with this problem, researchers have come up with *post-quantum cryptography*, a set of proposals for solutions to the looming threat of quantum computers on the cryptography currently in use. It may seem that quantum computers effective enough to break real-world encryption are a long way off: Building a quantum computer with enough qubits, and keeping them stable for long enough to be useful, poses a set of incredibly difficult physics and engineering challenges. However, no matter whether one believes powerful quantum computers are five years off or thirty years off, there is a compelling argument for acting as early as possible: People are now sharing plenty of data via cryptographically secured channels that they intend to stay private forever — or at least for a very long time —, even when stored now and attacked later with a quantum computer. This includes online audio or video telephony, private messages sent through chat services such as WhatsApp, and other sensitive data such as medical or financial records. In the words of a recent report by the United States' National Academy of Sciences:

> Even if a quantum computer that can decrypt current cryptographic ciphers is more than a decade off, the hazard of such a machine is high enough — and the time frame for transitioning to a new security protocol is sufficiently long and uncertain — that prioritization of the development, standardization, and deployment of post-quantum cryptography is critical for minimizing the chance of a potential security and privacy disaster. [13]

Isogeny-based cryptography is a specific type of post-quantum cryptography that uses certain well-behaved maps between abelian varieties over finite fields (typically elliptic curves) as its core building block. Its main advantages are relatively small keys and its rich mathematical structure, which poses some extremely interesting questions to cryptographers and computer algebraists.

The Autumn 2018 issue of this *Rundbrief* contained an article on post-quantum cryptography giving an overview of the main families of proposed constructions. We refer interested readers there for a broader discussion of things happening in the field of post-quantum cryptography.

Classical cryptographic key exchange

An important building block of many cryptographic systems, including the ubiquitous TLS protocol used to secure communication with websites, is a secure *key exchange*. Imagine you and a friend want to be able to send each other messages in a (metaphorical) locked box over an insecure channel, so you both need the same key. Since it's often not practical to exchange keys in person with everyone you want to send messages to, you also need a way of agreeing on a shared secret key over an insecure channel, with nobody else besides you and your friend being able to figure out what that key is. The key can be anything, so long as it's the same for all the parties involved; typically it is encoded as a bit string.

There are several ways to do this, the most common being the *Diffie–Hellman key exchange*. The security of this method is based on the fact that, for a finite group G, if you share an element $g \in G$ and some randomly chosen power $g^a \in G$, it is in general very hard for a corrupt bystander to compute a. The traditional (but now mostly deprecated) instantiation consists of fixing a prime p and computing in the group $\mathbb{F}_p^*$, i.e., the element g is simply an integer and the power g^a is computed modulo p.

To use this operation for a key exchange, our two parties Alice and Bob first agree on a group G and an element $g \in G$. Alice then chooses a secret positive integer a, and Bob chooses a secret positive integer b. Together, they can then compute the value g^{ab} over a public channel as follows:

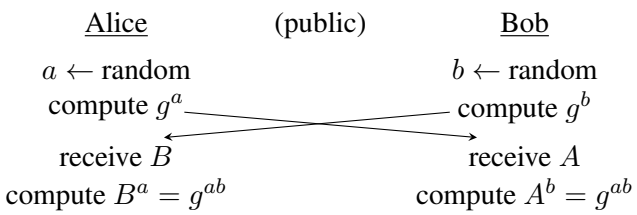

Figure 1: *The Diffie–Hellman key exchange.*

The obvious way to attack this scheme is to recover one of the secrets a and b from the publicly transmitted values g^a and g^b. This is known as the *discrete-logarithm problem*, which appears to be computationally hard for classical computers when the group G is well-chosen.

However, unfortunately, one thing that quantum computers are particularly good at is finding *periods* of computable functions using (variants of) an algorithm by Shor [11], which can be used to attack the discrete-logarithm problem as follows. Note that public values g^x are simply group elements: They can be multiplied together, and this satisfies the rule $g^x \cdot g^y = g^{x+y}$. This is exactly the operation that reduces breaking the scheme to finding a period: Given the element g and a public key $A = g^a$, we can define the group homomorphism

$$f\colon\ \mathbb{Z}^2 \to G,\ (x, y) \mapsto g^x \cdot A^y = g^{x+ay}\,.$$

Hence f is a periodic map whose period lattice is just its *kernel*, i.e., those pairs $(x, y) \in \mathbb{Z}^2$ where $g^{x+ay} = 1$. Shor's algorithm can, in polynomial time, find a basis of this lattice from an efficient description of the map f. We can then look for a vector of the form $(\tau, -1)$ in the lattice, which must equal $(a, -1)$ modulo the order of g, thus we have found a. The bottom line is that Diffie–Hellman is broken by quantum computers in all groups. Is this the end?

Luckily, in the aftermath of the discovery of Shor's algorithm, the traditional Diffie–Hellman framework has been extended to schemes which have similar traits, but do not rely on exponentiation maps in groups being one-way. One of these variants uses *isogeny graphs*.

Key exchange from graphs

Before we talk about isogeny graphs, let's first see how to get a shared value (key) from a more familiar graph. Here "graph" refers to a collection of *nodes* (dots) and *edges* (lines between them). For example, we can create a graph from a map of Manhattan by drawing a node at every junction and drawing an edge for every street. Then if Green and Red want to compute a shared value, they each choose a (secret) path from a common starting point, share the coordinates of their endpoints, then follow the same path from each other's endpoints to end up at the same final coordinates.

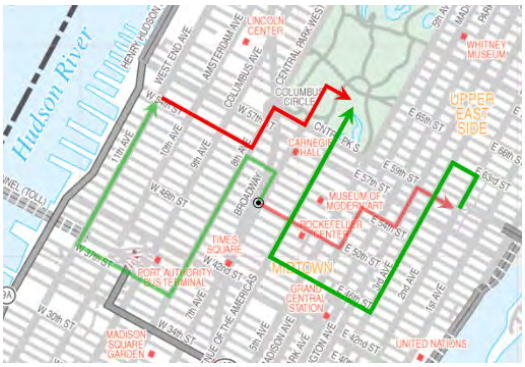

Figure 2: *Diffie–Hellman in Manhattan.*

However, this clearly isn't a secure way of exchanging keys — anyone can find a path from the common starting point to either Green or Red's end-of-path coordinates and thus compute the shared final coordinates, literally by adding and subtracting. To turn this approach into a secure key exchange, we need to replace our graph of Manhattan with a less structured graph, one in which finding a path between two given nodes is infeasible. On the other hand, the graph still needs to have *enough* structure to allow composing paths in a meaningful, commutative way, such that both parties end up at the same spot.

This is where isogenies comes in: They give rise to two families of graphs which are believed to have all the required properties for a (post-quantum) key exchange. Typical examples of these graphs look like this:

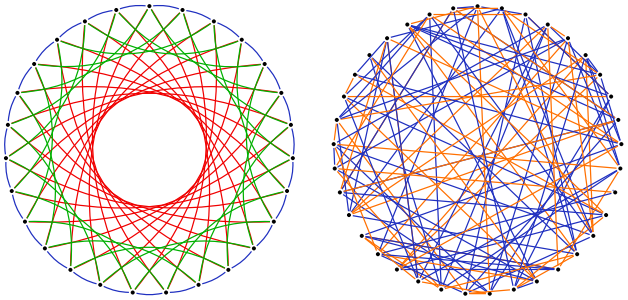

Figure 3: *Special isogeny graphs over a finite field.*

In both graphs, each node represents an *elliptic curve*, which can be represented as a certain kind of polynomial, and the edges represent maps between the elliptic curves called *isogenies*.

Elliptic curves and isogenies

We shall now explain some of the necessary background for understanding isogeny-based cryptography. Let $p \geq 5$ be a prime. An *elliptic curve over* $\mathbb{F}_{p^k}$ can then be defined as a smooth curve with equation $E\colon\ y^2 = f(x)$, where $f(x)$ is a degree-3 polynomial with coefficients in $\mathbb{F}_{p^k}$.

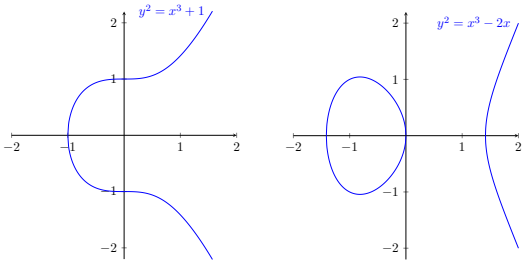

Figure 4: *Two elliptic curves over* $\mathbb{R}$.

"Smooth" means that the graph does not intersect itself or have any sharp points ("cusps"). These equations are especially interesting because the solutions that are defined over $\mathbb{F}_{p^k}$, together with one extra element, form an abelian group denoted $E(\mathbb{F}_{p^k})$. The extra element is referred to as the *point at infinity* ∞ and is defined to be lying on every vertical line that intersects the curve. The point ∞ is also the identity element of the group. The inverse of a point (a solution $P = (x_0, y_0)$ to the defining polynomial of E) is defined to be $-P = (x_0, -y_0)$. Notice that on a vertical line that intersects the elliptic curve in a point $P = (x_0, y_0)$, there are a total of three points in the group of solutions to the polynomial of E: the point P, its inverse $-P$, and the point at infinity ∞.

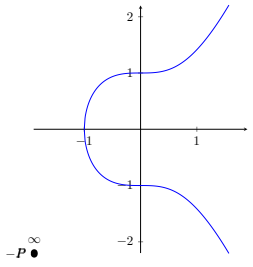

Figure 5: *Negating a point.*

In fact, this is no coincidence: any straight line that intersects the elliptic curve will intersect it exactly three times (when counting tangents as intersecting twice and also taking ∞ into account). We use this fact to define the group operation on $E(\mathbb{F}_{p^k})$: Given any two solutions $P = (x_P, y_P)$ and $Q = (x_Q, y_Q)$, draw the straight line passing through P and Q. This line will intersect the elliptic curve in exactly one more point R, and it turns out that the coordinates of this point will also be defined over $\mathbb{F}_{p^k}$. We then define a *group law*, written as $+$, by requiring that $P + Q + R = \infty$, the neutral element.

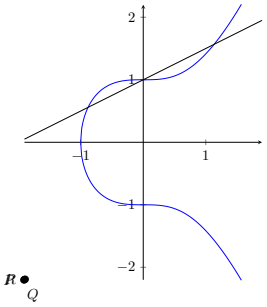

Figure 6: *Adding points.*

This group structure has led number theorists and geometers to study interesting properties of elliptic curves for centuries, and more recently elliptic curves have also enticed cryptographers, most importantly because one can base a very compact and efficient Diffie–Hellman key exchange on it. We need a little more though for a post-quantum scheme, since Shor's quantum algorithm to compute discrete logarithms of course also applies to this group.

Especially important in isogeny-based cryptography is a specific subclass of elliptic curves: *Supersingular* elliptic curves. An elliptic curve E defined over $\mathbb{F}_{p^k}$ is supersingular if $p \mid \left(p^k + 1 - \#E(\mathbb{F}_{p^k})\right)$. The most important special cases that come up are E defined over $\mathbb{F}_p$ with $\#E(\mathbb{F}_p) = p + 1$, and E defined over $\mathbb{F}_{p^2}$ with $\#E(\mathbb{F}_{p^2}) = (p+1)^2$. In a nutshell, this is useful because it allows to easily enforce a special group order: Given any prime p and $k \in \{1, 2\}$, it is known how to generate a supersingular elliptic curve with $(p+1)^k$ points over $\mathbb{F}_{p^k}$, hence we can control the group structure by choosing p in a special way. By contrast, it is not generally known how to efficiently find an elliptic curve with a given number of points, except in particularly nice special cases.

Recall that each node in the graphs we want to use for our post-quantum key exchange represents an elliptic curve. Since these graphs also have edges we need a way of passing from one node to another, which naturally will be a rational map that maps one curve to the other, and we will also want these maps to preserve the group structure of the elliptic curves. An *isogeny* is a non-zero map between elliptic curves that satisfies these things. More precisely, it is a surjective morphism of abelian varieties with finite kernel. The kernel subgroup is of utmost importance: In fact, one can prove that a (separable) isogeny is essentially uniquely defined by its

kernel subgroup, and one can compute an isogeny from its kernel in time linear in the size. Every isogeny has a *degree*, and typically (for separable isogenies) the degree is equal to the size of the kernel. Thus in a sense, the degree quantifies the algebraic and algorithmic complexity of an isogeny. However, since the secret keys in our cryptosystems are isogenies, we will use isogenies with "crypto-sized" (big) degrees! So how do we compute these isogenies quickly?

The solution is to use an isogeny of very *smooth* degree, say $\deg \varphi = \ell^k$ for a small prime ℓ (less than a few hundred or so) that is coprime to the (usually big) characteristic, and factor it into a composition of much smaller prime-degree maps:

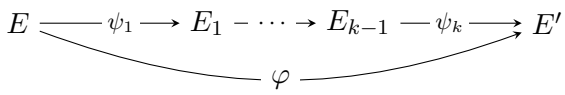

Figure 7: *Decomposing a smooth-degree isogeny.*

Note that the sequence $(\psi_1, ..., \psi_k)$ can be computed in $O(k \cdot \ell^2)$ field operations, whereas naïvely computing the entire map φ all at once would take time $\Theta(\deg \varphi) = \Theta(\ell^k)$: *exponentially* more.

The mathematics behind all of this is much richer than we can show in this short article. Interested readers are kindly referred to Luca De Feo's lecture notes [4].

Key exchange on isogeny graphs

In Figure 3 we saw two very different isogeny graphs that are used in cryptographic protocols. The two protocols built on these two types of graphs are called *CSIDH* [2] (pronounced "seaside", stands for "Commutative Supersingular-Isogeny Diffie–Hellman") and *SIDH* [8, 7] (pronounced as individual letters, stands for "Supersingular-Isogeny Diffie–Hellman").

CSIDH

We will show the CSIDH key exchange on a small (definitely not cryptographically-sized) example. We have seen how to perform an (insecure) key exchange on the graph on Manhattan — CSIDH is an implementation of the same idea on an *isogeny graph* with:

- Nodes given by *supersingular elliptic curves* E_A with equation $y^2 = x^3 + Ax^2 + x$ for $A \in \mathbb{F}_{419}$.
- Edges given by 3-, 5-, and 7-isogenies.

Almost as though by magic, this graph turns out to be very structured: Every node has exactly two outgoing edges of each colour, and the resulting cycles are compatible in the sense that a red step is always equivalent to the same number of blue steps, etc., independent of the position:

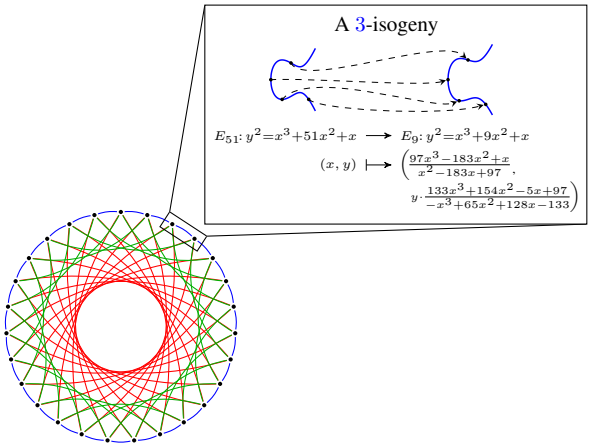

Figure 8: *An isogeny graph with a zoomed-in edge.*

Now if Alice and Bob want to compute a shared value in this graph, they each choose a (secret) path from a common starting point, share their endpoints, then follow the same path from the respective other party's endpoint to end up at the same final node — just like in the Manhattan example. We describe a path by a list of directions: one step clockwise (+) or anticlockwise (−) on the subgraph of a given colour.

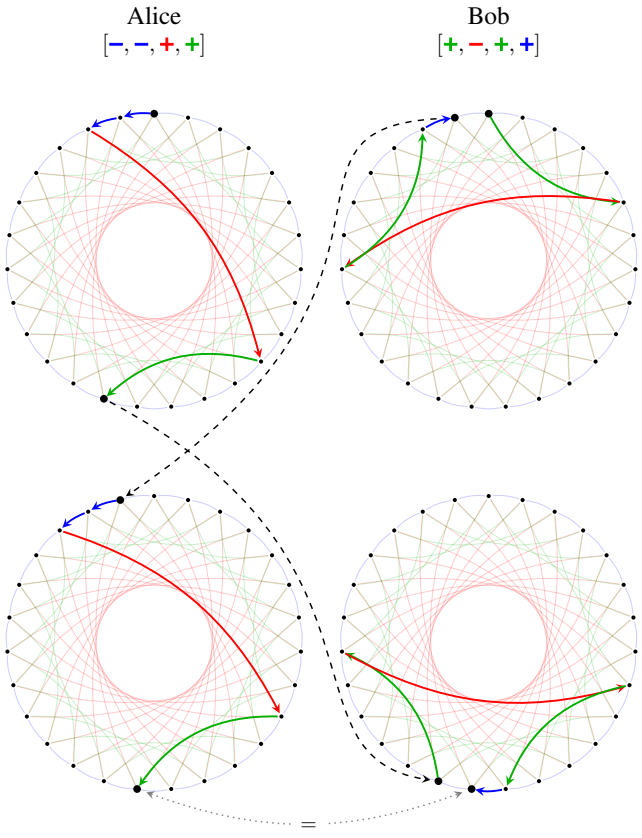

Figure 9: *Key exchange on a regular isogeny graph.*

On an unstructured graph, there is a priori no reason why Alice and Bob would end up at the same node after following this "graph key exchange" method. So what is it about the structure of *this* graph that makes this work?

Consider the set of clockwise and anticlockwise ℓ-isogenies as ℓ ranges over the non-negative integers, taken up to isomorphism. This set forms a *commuta-*

tive group G that *acts* on the set of supersingular elliptic curves $E_A\colon y^2 = x^3 + Ax^2 + x$ with $A \in \mathbb{F}_p$.[1] Write f_ℓ^+ and f_ℓ^- for clockwise and anticlockwise ℓ-isogenies respectively. In our key-exchange example above, Alice computes her curve E_A by computing the *action* of

$$f_A = f_7{}^+ \cdot f_5{}^+ \cdot f_3{}^- \cdot f_3{}^-$$

on E_0, written as $E_A := f_A * E_0$, and Bob computes his elliptic curve E_B by computing the action of

$$f_B = f_7{}^+ \cdot f_5{}^- \cdot f_7{}^+ \cdot f_3{}^+$$

on E_0, written as $E_B := f_B * E_0$. Then Alice and Bob send each other E_A and E_B and compute the actions $f_A * E_B$ and $f_B * E_A$ respectively. Now, as f_A and f_B are both elements in a *commutative* group,

$$\begin{aligned} f_A * E_B &= (f_A \cdot f_B) * E_0 \\ &= (f_B \cdot f_A) * E_0 = f_B * E_A. \end{aligned}$$

So Alice's endpoint $f_A * E_B$ and Bob's endpoint $f_B * E_A$ are the same!

Observe that, during this exchange, Alice never needs to communicate the isogeny group element f_A; this is her *private key*. With a much larger example, i.e., replacing 419 by a prime of several hundred (or even thousand) bits, and using a much longer list of ℓs, recovering this secret isogeny group element given just the start and end curve becomes infeasible for an attacker.

You may be thinking: doesn't Shor's algorithm apply to groups? Can I attack this commutative "isogeny group" with a quantum computer? The fundamental difference is that in traditional Diffie–Hellman, the public keys themselves are elements of the group, whereas in CSIDH, only the private keys are elements of a group — the public keys are elements of a set on which the group acts, and the operation $g^x \cdot g^y = g^{x+y}$ we've seen exploited earlier to apply Shor's algorithm to the discrete-logarithm problem simply does not exist. However, there is a quantum algorithm due to Kuperberg which attacks the action of a commutative group on the set of public keys to get a *subexponential* (but still super-polynomial) quantum attack [9, 10, 3]. In practice, this means that in order to be post-quantum secure, the parameters (like the prime p) have to be chosen larger than if the best attack was exponential; exactly how much larger is an ongoing research question.

SIDH

On the second graph in Figure 3, there is no evident group action — it looks just random. This rightfully suggests that Kuperberg's algorithm may not apply to finding a path on this graph, so the security level might scale better. However, it is not obvious how to even make a key-exchange protocol *work* on this graph: The extremely regular structure of the CSIDH graph aided in getting Alice and Bob's operations to commute, whereas in this case everything looks rather messy.

Happily, the graph does still carry enough structure, due to the fact that an isogeny is uniquely defined by its kernel. Alice and Bob (publicly) agree on a common starting curve $E/\mathbb{F}_{p^2}$ and choose secret subgroups A and B of $E(\mathbb{F}_{p^2})$. Writing φ_A and φ_B for the isogenies with those subgroups as kernel, the following diagram commutes (up to isomorphism) when $A' = \varphi_B(A)$ and $B' = \varphi_A(B)$, and therefore an isomorphism invariant of the curve $E/\langle A, B\rangle$ can be used as a shared secret:

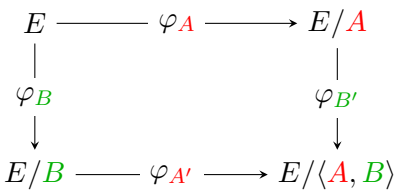

Figure 10: *High-level view of SIDH.*

Here, the horizontal arrows are computed by Alice and the vertical arrows are computed by Bob. Focussing on Alice, the only problem left is that she needs to somehow obtain $A' = \varphi_B(A)$, but Bob cannot give out φ_B since that's his secret.

The solution is that isogenies are also group homomorphisms on the corresponding groups of elliptic curve points: While Bob cannot give out φ_B, he *can* evaluate this map on publicly known points $P, Q \in E(\mathbb{F}_{p^2})$ and reveal $P' = \varphi_B(P)$ and $Q' = \varphi_B(Q)$. If Alice then chooses her subgroup A of the form $\langle P + [a]Q\rangle$ (that is, a cyclic group generated by $P + [a]Q$), she can simply compute A' as $\langle P' + [a]Q'\rangle$.

Similarly, Alice publishes images of known points under her own secret φ_A, allowing Bob to find B'.

This is the high-level mathematical overview of the protocol — of course there are many more interesting details in practice, for example one still has to ensure that the points P', Q' do not leak computationally useful[2] information about φ_B (similarly for φ_A), and choose the other parameters of the system in such a way that everything Alice and Bob need to compute is efficient in practice.

More advanced protocols

In this article we've only shown two simple key-exchange protocols using different kinds of isogeny graphs. However, the underlying mathematical ideas give rise to many other interesting cryptographic constructions, some of which seem impossible or harder to build without the use of isogenies.

[1] For number theory experts: a (separable) isogeny is (up to isomorphism) uniquely defined by its kernel, which corresponds to an ideal in the common $\mathbb{F}_p$-rational endomorphism ring $\mathbb{Z}[\sqrt{-p}]$ of every such elliptic curve. The *codomains* of such isogenies then only depend on the *class* of the corresponding ideal, hence the action of G can be considered an action of (a subgroup of) the class group $\mathrm{cl}(\mathbb{Z}[\sqrt{-p}])$.

[2] We emphasize that while the action on a set of points often uniquely identifies an isogeny, it is not generally known how to *compute* the isogeny from that information.

For instance, one can build a *verifiable delay function* from isogenies [6]: A VDF is a random-looking function which is inherently slow to compute — independently of the algorithms used, the amount of hardware available, and parallelism — but on the other hand it is efficient for anyone to verify afterwards that the result is correct. (In the isogeny-based construction, the slow part consists of a long isogeny evaluation, and the verification part is a single elliptic-curve pairing.) This functionality may seem like not more than an odd curiosity, but in fact it has enough relevance in the distributed-systems world that blockchain projects are currently investing one hundred thousand US dollars [14] in the development of VDF technology (albeit founded on different mathematical ideas).

Another interesting example is the construction of *digital signatures* from isogenies: A recent paper [1] proposes a practical signature scheme CSI-FiSh (pronounced "seafish", stands for "Commutative Supersingular-Isogeny Fiat–Shamir") based on CSIDH's 512-bit parameter set. The main contribution is a massive precomputation effort in the form of a record-breaking *class-group computation*, which allows uniform sampling of isogeny walks — an important ingredient of the signature scheme. It is not known how to adapt this scheme to bigger (read: more secure) parameters: the effort for the class-group computation quickly grows too big for currently known techniques.

References

[1] Ward Beullens, Thorsten Kleinjung, and Frederik Vercauteren. CSI-FiSh: Efficient isogeny based signatures through class group computations. In *Cryptology ePrint Archive*, Report 2019/498. 2019. https://ia.cr/2019/498.

[2] Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. CSIDH: an efficient post-quantum commutative group action. In *ASIACRYPT (3)*, volume 11274 of *LNCS*, pages 395–427. Springer, 2018. https://ia.cr/2018/383.

[3] Andrew M. Childs, David Jao, and Vladimir Soukharev. Constructing elliptic curve isogenies in quantum subexponential time. In *J. Mathematical Cryptology*, volume 8, pages 1–29. 2014. https://arxiv.org/abs/1012.4019v1.

[4] Luca De Feo. Mathematics of isogeny based cryptography. 2017. https://arxiv.org/abs/1711.04062.

[5] Luca De Feo and Steven D. Galbraith. SeaSign: Compact isogeny signatures from class group actions. In *EUROCRYPT (3)*, volume 11478 of *LNCS*, pages 759–789. Springer, 2019. https://ia.cr/2018/824.

[6] Luca De Feo, Simon Masson, Christophe Petit, and Antonio Sanso. Verifiable Delay Functions from supersingular isogenies and pairings. In *Cryptology ePrint Archive*, Report 2019/166. 2019. https://ia.cr/2019/166.

[7] David Jao, Reza Azarderakhsh, Matthew Campagna, Craig Costello, Luca De Feo, Basil Hess, Amir Jalali, Brian Koziel, Brian LaMacchia, Patrick Longa, Michael Naehrig, Joost Renes, Vladimir Soukharev, and David Urbanik. SIKE. Submission to [12]. http://sike.org.

[8] David Jao and Luca De Feo. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In *PQCrypto*, volume 7071 of *LNCS*, pages 19–34. Springer, 2011. https://ia.cr/2011/506.

[9] Greg Kuperberg. A subexponential-time quantum algorithm for the dihedral hidden subgroup problem. *SIAM J. Comput.*, 35(1):170–188, 2005. https://arxiv.org/abs/quant-ph/0302112.

[10] Greg Kuperberg. Another subexponential-time quantum algorithm for the dihedral hidden subgroup problem. In *TQC*, volume 22 of *LIPIcs*, pages 20–34. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2013. https://arxiv.org/abs/1112.3333.

[11] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. In *SIAM Journal on Computing*, volume 26(5), pages 1484–1509. 1997.

[12] National Institute of Standards and Technology. Post-quantum cryptography standardization, December 2016. https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Post-Quantum-Cryptography-Standardization.

[13] National Academy of Sciences, Engineering, and Medicine. Quantum computing: Progress and prospects. ISBN: 978-0-309-47969-1. 2019. https://nap.edu/catalog/25196.

[14] VDF Alliance. FPGA design competition. See https://vdfalliance.org/contest.

Towards Post-Quantum Secure Symmetric Cryptography: A Mathematical Perspective

Xenia Bogomolec, Quant-X Security & Coding, Hanover
John Gregory Underhill, itk AVtobvS SARL, Fribourg
Stiepan Aurélien Kovac, QRCrypto SA, Fribourg

xb@quant-x-sec.com
john.underhill@protonmail.com
contact@qrcrypto.ch

Introduction

We introduce an independent research project on symmetric cryptography with a focus on foreseeable industrial needs. It was initiated by the independent IT-Security experts Kovac and Underhill. The result is the new symmetric cryptographic algorithm EAES, which is intended to be a stronger brother of the widely used AES algorithm (Advanced Encryption Standard), the standardized version of the RIJNDAEL algorithm.

The algorithm EAES is designed by Kovac and Underhill. They published the e-print [1]. Underhill had previously started the implementation in his CEX-NET project, within which he created the cryptographic C++ library CEX [2]. The library is open source and published under the GPL-license. Bogomolec is responsible for the mathematical analysis of the algorithm. All of us are independent IT-Security experts.

EAES mitigates threats by formerly published attacks on AES from binary devices [3, 4, 5] and additionally offers enhanced security against attacks performed by moderate quantum computers [6, 7].

We also outline the necessary considerations and the steps which have to be taken in order to place a new cryptographic algorithm in global industry.

The importance of standardization

A successful standardization of an algorithm ensures compatibility with other global standards. Furthermore it is a seal of quality for users who don't understand the mechanisms and properties of the algorithm in depth.

There are two big players in international standardization for symmetric cryptography, ISO (International Organization for Standardization) and NIST (National Institute of Standards and Technologies, USA). The NIST launched a standardization process on asymmetric post-quantum cryptography, i.e. cryptography resisting quantum computing attacks. The reason for the anticipation of standardization to the availability of strong quantum computing machines is very well explained in the article *Isogeny-based cryptography*.

Hybrid encryption systems are used in all major crypto protocols: TLS, SSH and PGP. They combine the advantages of both cryptography classes. Symmetric protocols allow securely sharing a key via digital connections, and symmetric protocols are about $10^5\times$ faster than asymmetric ones. The secret session key SK, which is limited in time, is shared via asymmetric cryptography, and the message itself is symmetrically encrypted with the securely shared session key.

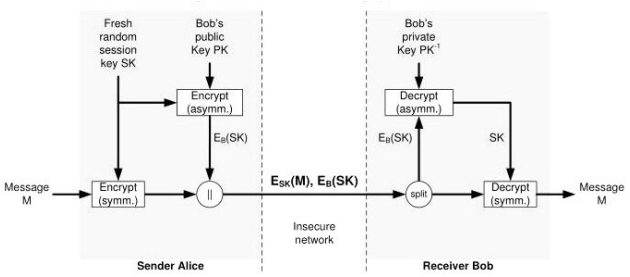

Figure 1: *Hybrid encryption.*

No asymmetric encryption within hybrid encryption systems can outbalance weaknesses of the symmetric part. ISO currently makes it possible to standardize new symmetric cryptography algorithms and to amend existing ones. Therefore, we strive to establish EAES as an ISO-Standard. Its predecessor AES has been standardized by both organizations.

Published attacks on AES

There are various types of attacks on cryptographic algorithms, amongst which side-channel attacks, implementation attacks, brute-force attacks and cryptanalysis attacks on AES are of importance in our context. In some cases, those attack types can be combined to

achieve a more efficient decryption of a ciphertext.

Side-channel attacks use unintended side effects of cryptographic operations to glean information about the plaintext and/or secret key being processed. **Implementation attacks** use weaknesses in implementation of an encryption scheme, e.g. weak key generation. **Brute-force attacks** attempt every possible combination for a key. **Cryptanalysis attacks** rely on alternative algorithms for finding a secret key of an encrypted text. The latter can be applied to stationary data. Therefore it is interesting for data collectors who are patient enough to wait for the availability of potent-enough machines to perform such an attack.

Mathematical structure of AES
All RIJNDAEL functions are linear. Only the basic encryption function *SubBytes* (see section Basic RIJNDAEL encryption functions) is often referred to as the non-linear part of AES, but in fact it is linear as well. Well-chosen linear layers with very strong diffusion properties protect against conventional attacks using statistical properties of a cryptosystem. In the case of the quantum algebraic attack [7], this effect is limited.

Attacks performed on binary devices
Various published attacks on AES [3, 4, 5] take advantage of the invertibility of the original RIJNDAEL key schedule besides other properties such as the relatively low number of rounds, implementation weaknesses, and the same block size for standardized all key sizes.

Grover's search
AES was generally still considered post-quantum secure for key sizes larger than 192. Even Grover's search algorithm [8] is not regarded a threat for AES-256 in the near future. It can be used to extract the key from a small number of AES plaintext–ciphertext pairs (5 for AES-256). Grover's search is a alternative algorithm for an exhaustive key search (brute-force attack).

Quantum algebraic attack
A harder impact on the security of AES-256 is posed by the quantum algebraic attack on cryptosystems which can be reduced to Boolean equation solving [7]. This attack reduces security level of AES-256 from 256 to 78.53. The quantum algebraic attack is a classical cryptanalysis attack.

Consequences
Both Grover's search algorithm and the quantum algebraic attack can be applied to collected data without the corresponding key exchanges as soon as potent-enough quantum computers are available.

Therefore we propose EAES as a symmetric alternative for AES, with higher security against all previously mentioned attacks [3, 4, 5, 6, 7].

The following sections are written for readers with deeper technical knowledge in cryptography and block ciphers. In the last section we summarize the advantages of EAES.

RIJNDAEL, the base of EAES

Here we only give a high-level overview of RIJNDAEL in order to classify our modifications. For a detailed description of its standardized version AES, we refer to the Federal Information Processing Standards Publication 197 [9].

Computation grounds
RIJNDAEL is an iterative rounds-based block cipher. It relies on a substitution–permutation network. In AES, the network is operating on a 4×4 column-major order array of the 128 bits (block size), called the "state." Each input data block is intially transformed into a "state." The term "state" refers to the digital representation as well as to the fact of the continuous transformation during the encryption. So each array entry consists of 8 bits. Columns of the matrix are also called "words." All operations are performed in the Galois field $GF(2^8) \cong \mathbb{F}_2[x]/(x^8 + x^4 + x^3 + x + 1)$.

RIJNDAEL algorithm architecture
RIJNDAEL is a block cipher. That means that the basic encryption functions are iterated a certain number of rounds with dedicated round keys over the state. With $r =$ number of rounds, the encryption process is:

(1) **Derive round keys**
Call *ExpandKey* (key) to derive $r + 1$ round keys from the secret key. The $+ 1$ stands for the fact that *AddRoundKey* is called additionally before each block encryption.

(2) **Encryption**
Perform on each block of data represented by the "state":

a) Initial round:
Add the plaintext to the state
AddRoundKey (state, 1st round key)

b) For ($i = 2$; $i \leq r - 1$, i++):
SubBytes (state)
ShiftRows (state)
MixColumns (state)
AddRoundKey (state, i-th round key)
(4 basic RIJNDAEL encryption functions)

c) The final round:
SubBytes (state)
ShiftRows (state)
AddRoundKey (state, last round key)

The decryption process is composed by the inversed chain of the encryption functions. A symmetric encryption algorithm needs to be composed by invertible functions, but this property wouldn't be necessary for the key expansion. In fact, the invertibility of *ExpandKey* opened the door for various side-channel attacks.

Basic RIJNDAEL encryption functions
The 4 basic functions of the RIJNDAEL encryption are:

1) *AddRoundKey* – addition in $(\mathbb{F}_2)^{128}$:
   Bitwise addition of the state and the correspondent round key.
2) *SubBytes* – non-linear substitution:
   Each byte is replaced by another according to the specified substitution table (S-Box). A more resource friendly option is to treat a state byte as an element $\alpha \in \mathbb{F}_2[x]/(x^8+x^4+x^3+x+1)$, where the multiplicative inverse of α needs to be found.
3) *ShiftRows* – transposition for diffusion:
   The second, third and fourth row of the state are shifted to the left, by 1, 2 and 3 steps.
4) *MixColumns* – mixing for diffusion:
   Multiplication of each column of the state with the following matrix M:

$$M = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix}$$

As a chain of invertible algebraic functions, RIJNDAEL and AES can be represented as a polynomial system [10]. This polynomial system can re reduced to a Boolean equation system. We will look at this property in the context of the quantum algebraic attack [7].

Modifications for EAES

Our modifications affect the key schedule *ExpandKey* and the number of rounds per version. Furthermore we implemented a version for a 512-bit key, which is no outlier amongst post-quantum secure algorithm key sizes.

RIJNDAEL inheritance
The original RIJNDAEL algorithm includes a block size option of 256 bits, which was not admitted for the AES standard. We decided to keep the 128 bits for EAES in order to ensure compatibility with existing hardware implementations [1]. Several mentioned attacks take advantage of the fact that AES-256 runs with the same block size as AES-128. Those vulnerabilities are at least mitigated in our algorithm by the higher number of rounds.

We also kept the 4 basic RIJNDAEL encryption functions and the RIJNDAEL algorithm architecture.

More rounds and a 512-bit key version
We increased the number of rounds taking in account recommendations of renowned cryptographers and cryptanalysts such as Bruce Schneier. The 256-bit key variant EAES-256 runs 22 rounds of the original RIJNDAEL transformation function, which is 8 more rounds than AES-256, and twice the best known attack which breaks 11 rounds [4]. Based on the same considerations, we fixed the number of 30 rounds for EAES-512.

Say goodbye to 128- and 192-bit keys
Organizations such as the NIST and the EU quantum flagship [11] recommend to use the maximal key lengths of currently used cryptographic algorithms whenever possible. But for operational staff in IT, the options very often simply come down to the question about what choices they have for establishing a confidential communication with a business or cooperation partner. EAES doesn't offer the 128- and 192-bit key options anymore.

Replacing *ExpandKey*
The invertibility of the original RIJNDAEL key schedule *ExpandKey* opened the door for various published attack schemes. Furthermore the output of the original RIJNDAEL key schedule does not offer cryptographic quality. In the simple cases, round keys could be extracted within encryption or decryption processes, and the original key computed by applying the inverted key schedule function. For cryptanalysis attacks, this is just another convenient property for finding a replacement of the chain of RIJNDAEL functions.
Therefore we replace *ExpandKey* by cryptographically secure Pseudo Random Number Generators (PRNGs) with strong diffusion properties in EAES.

Pseudo Random Number Generators
PRNGs are built for deriving keys of a fixed size for further cryptographic operations by using an underlying pseudo random function. In our case those pseudo random functions are hashing algorithms. They are not even injective, but produce a well defined and collision resistant output.

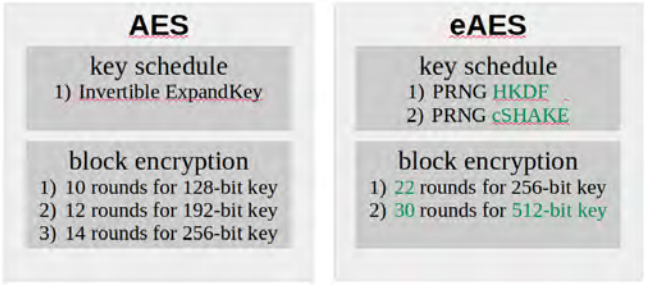

Figure 2: *High level comparison of AES and eAES.*

Furthermore, it is very cost-intensive under foreseeable technical developments to perform brute-force attacks on the chosen PRNGs. So they are also hardly reversible apart from the non-existant mathematical invertibility.

[1] We have also implemented an authenticated stream cipher (RCS) using the 256-bit block transform along with a cryptographically strong key-schedule, and an increase in transformation rounds that parallels eAES.

Key schedule variants

Our first choice is HKDF, a HMAC-based extract-and-expand key derivation function, with SHA-2 as the underlying hash function. As a second option, we implemented the KECCAK-derived and customizable CSHAKE-function. KECCAK, the SHA-3 competition finalist, was chosen by the NIST for its algorithmic unrelatedness from SHA-2, while offering flexibility and a comparable speed in computation.

Common features

Both HKDF and CSHAKE produce arrays of bytes which are converted to big-endian-ordered 32-bit integers for the round subkeys. Each round key has the same size as the cipher's block size, namely 128 bits. So the output of our key derivation functions needs to be of the size $(r + 1) \cdot 128$ bit, where r represents the number of rounds. We renounce on the usage of a salt due to the fact that within our algorithm, the input is cryptographically strong already.

HKDF

We consider the HMAC-based HKDF as a sensible intermediary solution for the derivation of the round keys, because it is already widely available. We use HKDF(SHA-256) for EAES-256 and HKDF(SHA-512) for EAES-512 to align with expected security strengths.

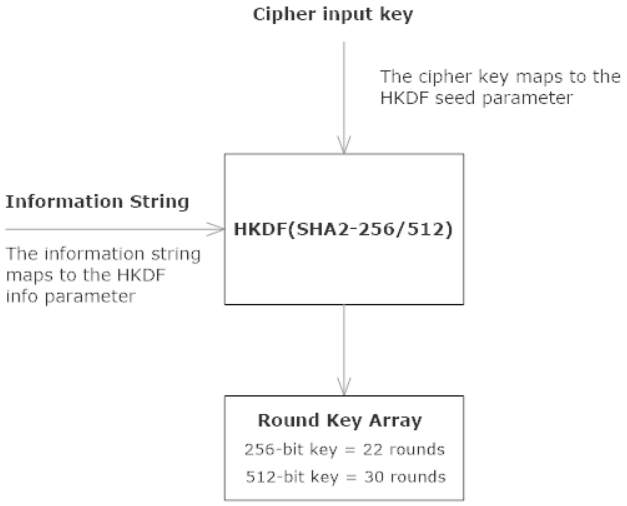

Figure 3: *Rijndael HKDF eXtension.*

SHA-256 and SHA-512 are members of the SHA-2 function family. They use the Merkle–Damgard construction, a method of building collision-resistant cryptographic hash functions from collision-resistant one-way compression functions. The compression function for SHA-2 uses the Davies–Meyer structure from a (classified) specialized block cipher.

HKDF generates cryptographically strong output of any desired length by repeatedly generating hash-blocks, concatenating them, and finally truncating the result to the desired length. Each call to HMAC involves two calls to the SHA-2 hash function to generate a pseudo-random 256-bit or 512-bit output block.

So if the number of rounds is r and the SHA-2 output length is n, SHA-2 has to be called $\lceil \frac{(r+1)\cdot 128}{n} \rceil$ times. This results in 12 times with SHA-256 for EAES-256 and 8 times with SHA-512 for EAES-512.

CSHAKE

The SHA-3 competition finalist KECCAK is a so called *sponge function*. Those are functions with finite internal state, taking an input bit stream of any length and producing an output bit stream of any desired length.

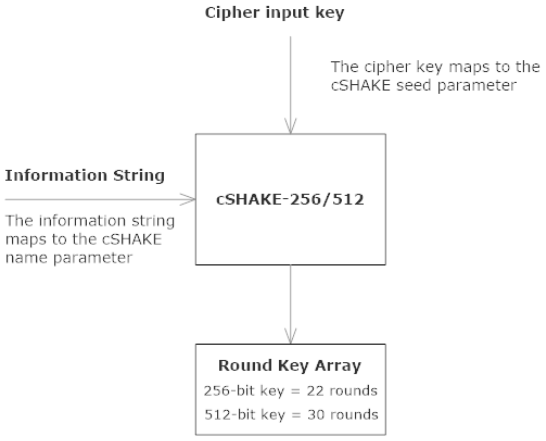

Figure 4: *Rijndael SHAKE eXtension.*

We implemented the SHA-3-derived SHAKE-function as a second option for the key schedule. CSHAKE is the customizable version of SHAKE. It is originally designed for 128- and 256-bit security strength [12]. Underhill has additionally created a 512-bit security implementation, which mirrors the internal block-size, squeeze and permutation settings of SHA3-512. CSHAKE-256 is used for EAES-256, and CSHAKE-512 is used for EAES-512.

The initial state of SHAKE is a 5×5 array of 64-bit unsigned integers, 1600 zero bits it total. The first call is to initialize the custom state. Then the original EAES-key is absorbed into the previously intialized state, and with each call to the inner permutation function of KECCAK, rates of $(1600-2\cdot n,\ n \in \{256, 512\})$ bits are returned. So CSHAKE-256 returns 136, and CSHAKE-512 returns 72 pseudo-random bytes per call.

So if the number of EAES-rounds is r and the number of returned bytes per rate are n, we have $\lceil \frac{(r+1)\cdot 128}{n\cdot 8} \rceil$ calls to the inner permutation functions of CSHAKE.For EAES-256 we have 3 and for EAES-512 we have 7 calls to the inner permutation function, +1 call for the initial state.

Note that here we are talking about calls to the inner permutation function of KECCAK, while we are talking about calls to SHA-2 itself in the previous section. Fewer calls of the permutation function lead to higher

efficiency of cSHAKE compared to the one of HKDF. On the other hand, KECCAK is prone to quantum algebraic attacks [7].

Impact of Grover's search algorithm

Grover's algorithm offers a $\sqrt{k}$ speed-up [8] over a classical exhaustive search (brute-force attack) on the set of keys of size k. It seems to be one of the most relevant quantum cryptanalytic impact for the study of block ciphers. The authors of [6] present quantum circuits to implement the key search for AES and analyze the quantum resources required to carry out such an attack for key sizes of 128, 192, and 256 bits.

The number of required logical qubits is relatively low, 6,681 for AES-256. On the other hand, the large circuit depth of enrolling the entire Grover iteration poses a challenge to an implementation on an actual physical quantum computer, even if the gates (basic quantum circuit operating on qubits) are not error-corrected. The key schedule *ExpandKey* causes much of the circuit cost within each Grover iteration. Replacing it by our chosen PRNGs will even be considerably more cost-intensive.

Here we only summarize the basic prerequisites of the involved procedures described in [6] for AES-256 and compare them to EAES for $k \in \{256, 512\}$.

Algorithm architecture
A quantum circuit implementing a Boolean function f is the input of Grover's search algorithm:

$$f : \{0,1\}^k \to \{0,1\}$$

The basic algorithm finds an element x_0 such that $f(x_0) = 1$. This is realized by repeatedly applying the operation G to measure an element x_0 such that $f(x_0) = 1$ with constant probability.

$$G = U_f \left((H^{\otimes k}(2|0\rangle\langle 0| - 1_{2^k})H^{\otimes k}) \otimes 1_2 \right)$$

H denotes the 2×2 Hadamard transform and U_f involves the computation of the cipher functions. To ensure uniqueness of the solution (the found key), a small number of plaintext–ciphertext pairs are needed. This number is 5 for a 256-bit key and 9 for 512-bit key.

Quantum resources and graph theory
Quantum resources are represented by logical qubits, gates, and circuit depth. In the model of a Boolean circuit as a directed acyclic graph, the circuit depth is the maximal length of a path from an input gate to the output gate. The sum of required gates represent the complexity of an algorithm.

Reversible circuits are needed for the application of Grover on AES to provide invertibilty of the operations. Therefore the proposed solution in [6] relies on a set of fault-tolerant reversible logical gates. It consists of called Toffoli gates, controlled NOT gates and NOT gates. A Toffoli gate is a universal reversible logic gate, i.e. any reversible circuit can be constructed from Toffoli gates.

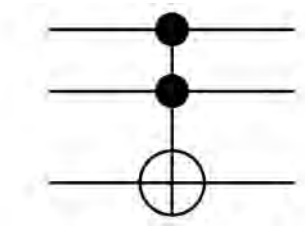

Figure 5: *Toffoli gate (controlled-controlled-not gate).*

For our key schedule replacements, we will base our comparisons on the results of "Estimating the cost of generic quantum pre-image attacks on SHA-2 and SHA-3s" [13]. The authors present an implementation including Grover's algorithm on reversible circuits on on a surface-code-based fault-tolerant quantum computer.

Quantum resources for the search
H represents the Hadamard operation on a single qubit. A single solution to the equality function f can be found by applying $H^{\otimes k+1}$ to the initial state, where k is the size of the key. The next step is applying

$$G^{\lceil \frac{\pi}{4}\sqrt{2^k} \rceil},$$

followed by a measurement of the entire quantum register which will yield a solution x_0 with high probability.

The exact decomposition of the search is outlined in [6], Sect. 2. For our comparison we only look at the cost for the operation $(2|0\rangle\langle 0| - 1_{2^k}))$, which is determined by the reduction to the implementation of a k-fold NOT gate, where $k \in \{256, 512\}$ is the key size. In terms of Toffoli gates we have the formula $8k - 24$ [14] and if we count only T-gates, we have the formula $32k - 84$ as an upper bound for a k-fold controlled NOT gate [15]. This results in:

Cost for the operation $(2|0\rangle\langle 0| - 1_{2^k}))$:

key size	Toffoli gates	T-gates
256	2,024	8,108
512	4,072	16,300

Grover's search will have to be performed on the small number n of plaintext–ciphertext pairs. The equality function f can be implemented by a multiple controlled NOT gate that has $128 \cdot n$ controls. With above formulas, the needed number of Toffoli counts and T-counts to compute the equality function f come down to:

Cost for the equality function f:

key size	Toffoli counts	T-counts
256	5,096	20,396
512	9,192	36,780

The search resources only depend on the key sizes and not on the ciphers and their options. In this regard we don't have to include further differing considerations.

Quantum resources for the ciphers

The number of rounds depends on the specific key length k, while the four basic RIJNDAEL functions are independent of the key length k for both AES and EAES. The realization of all RIJNDAEL functions is analyzed in dedicated sections in [6]. Here we only present the results and comparisons per function.

Cost for SHA-2 and -3 functions (256 bit):

function	gates	depth	qubits
SHA-256	$1.42 \cdot 2^{146}$	$1.69 \cdot 2^{144}$	$2^{12.6}$
SHA3-256	$1.52 \cdot 2^{147}$	$1.33 \cdot 2^{137}$	2^{20}

The values for SHA-256 and SHA3-256 are results from the resource estimates done in [13], converted to a representation in powers of 2 instead of 10. For both functions, the total cost comes down to approximately 2^{166} basic operations. The estimation of the resources for the according 512-bit versions will be done in a further step of our research.

For our EAES-options we have to multiply the gate values by the number of calls to the hash and inner permutation functions.

Cost for AES and EAES key expansion (256 bit):

function	gates	depth	qubits
ExpandKey	54,331	23,896	512
HKDF	$1.07 \cdot 2^{150}$	$1.26 \cdot 2^{148}$	$2^{12.6} + 3072$
CSHAKE	$1.52 \cdot 2^{149}$	$1.33 \cdot 2^{139}$	$2^{20} + 1024$

The numbers of gates and depths take the number of calls to the according hash and permutation functions into account. Additional qubits are needed to store the output of all calls to the hash functions.

As mentioned before, we cannot refer to values for 512-bit keys at this time. But we can see that the 256-bit versions already offer considerable advantages over the original RIJNDAEL key schedule *ExpandKey*.

Cost for holding the state and initial round:

function	gates	depth	qubits
state	0	0	128
initial round v_1	64 NOT	0	128
initial round v_2	128 CNOT	1	256

Initial round v_1 represents the realization with flipping bits. These values are the same for all ciphers and options.

Cost for basic functions

function	gates	depth	qubits
SubBytes	22,326	1	9
ShiftRows	0	0	0
MixColumns	277 CNOT	39	0
AddRoundKey	128 CNOT	1	0

The current round key is available on 128 dedicated wires. *SubBytes* is realized by finding the inverse of the byte in $GF(2^8)$, seen as permutation in $\mathbb{F}_{256}$. No extra gates are necessary to implement *ShiftRows*, as it corresponds to a permutation of the qubits. The position of the subsequent gates is simply adjusted to the correct input wire.

Cost for RIJNDAEL rounds

To compute all rounds of RIJNDAEL, 536 qubits are needed for 10 rounds in the 128-bit key version and 664 qubits are required for any round number $r \geq 12$. In this regard, the higher number of rounds in EAES does not add complexity to the algorithm. It does add complexity to the number of gates and to the depth, though.

Cipher output for 256-bit keys

The number of gates and depth for EAES is derived as follows:

Values from the rounds row in table 4, AES-256 in [6] :

g_{14} = sum of number of gates in [6]
d_{14} = sum of depths in [6]

Values from the key expansion cost table in this article:

g_{hkdf} = number of gates for HKDF-256
d_{hkdf} = depths for HKDF-256
q_{hkdf} = number of qubits for HKDF-256
g_{shake} = number of gates for CSHAKE-256
d_{shake} = depths for CSHAKE-256
q_{shake} = number of qubits for CSHAKE-256

Then we compute the values for EAES(HKDF) from the formulas:

$$gates = g_{hkdf} + \frac{22}{14} g_{14}, \quad depth = d_{hkdf} + \frac{22}{14} d_{14}$$

The values for EAES(CSHAKE) from the formulas:

$$gates = g_{shake} + \frac{22}{14} g_{14}, \quad depth = d_{shake} + \frac{22}{14} d_{14}$$

And the number of qubits for is the simple addition $664 + q_{sha}$, $664 + q_{sha3}$ respectively.

Cost for each cipher output (256 bit)

cipher	gates	depth	qubits
AES-256	$1.65 \cdot 2^{21}$	$1.46 \cdot 2^{17}$	1,336
EAES(HKDF)	$1.07 \cdot 2^{150}$	$1.26 \cdot 2^{148}$	$1.21 \cdot 2^{13}$
EAES(cSHAKE)	$1.52 \cdot 2^{149}$	$1.33 \cdot 2^{139}$	$1.002 \cdot 2^{20}$

Note that the gates and depth values for EAES equal the values in the key expansion cost table. This comes from the fact that the contribution of the gates and depth from the rounds is in much lower power of 2 than the cost for the key expansions HKDF and cSHAKE. So we see that this replacements are the essential change and contribution to a higher algorithm complexity.

Grover algorithm

For the overall T-count for Grover on RIJNDAEL we have an estimate of $\lfloor \frac{\pi}{4} 2^{k/2} \rfloor \cdot (6t_k + f_k)$, where k is the key size, t_k is the number of T-gates and f_k is the cost for the equality function. For the overall circuit depth we obtain with $r =$ number of rounds: $2 \cdot r \cdot$ total T-depth. Here we are ignoring some of the gates which do not contribute significantly to the bottom line. Both formulas are from [6], Sect. 3.4. So by a moderate estimation we get the following complexities for Grover's search on the compared ciphers:

Grover's search on ciphers

cipher	gates	depth	qubits
AES-256	$3.24 \cdot 2^{151}$	$1.71 \cdot 2^{145}$	6,681
EAES(HKDF)	$> 2^{278}$	$> 2^{274}$	$> 2^{14}$
EAES(cSHAKE)	$> 2^{277}$	$> 2^{257}$	$> 2^{21}$

Both HKDF and cSHAKE have a considerable impact on the increased cost of Grover's search. The higher number of rounds for EAES-256 and the 512-bit key version additionally contibute to the complexity of the rounds by factors of at least $r/14$. The authors of [6] recommended in 2015 to move away from AES-128 when expecting the availability of at least a moderate-size quantum computer. Our implementation excludes that option and offers an increased complexity compared to AES-256.

Impact of quantum algebraic attacks

The authors of [7] present an algorithm which leads to new considerations of the security of systems which can be reduced to solving Boolean equations. A solution a for the equation $\mathcal{F} \cdot a = 0$ with a set of polynomials $\mathcal{F} \subset \mathbb{C}[X]$ is called *Boolean* if each coordinate of a is either 0 or 1.

The resulting quantum algebraic attack algorithm includes quantum-monomial solving of polynomial systems over $\mathbb{C}$ by applying a Macaulay linear system. Like this they constructed a Boolean equation solving algorithm, with the following properties:

1) It decides if there exists a Boolean solution.
2) It returns a Boolean solution with a given probability if there are such solutions to the system.
3) It returns $\emptyset$ if no Boolean solution exists.

The runtime complexity of the resulting quantum algebraic attack is considerably lower than the one of Grover's Search, but it depends on two factors: a constant c and a condition number κ. The complexity of $2^{78.53} c\kappa^2$ for AES-256 is not much higher than the complexity of $2^{73.30} c\kappa^2$ for AES-128 due to the same block size of 128 bit. Therefore we can assume that the complexity won't be much higher for 512-bit key sizes, and it will also not considerably increased by the higher number of rounds in EAES.

The conclusion of [7] is that systems which can be solved by Boolean equation solving are only secure under quantum algebraic attack if the condition number κ is large. The construction of such systems is a topic for further research. Besides AES and KECCAK, stream ciphers such as TRIVIUM and the multivariate public key cryptosystem MPKC are affected by the attack.

Conclusion

Considering the lower exponent of the highest power of 2 in the total algorithm complexity, EAES offers a higher complexity of a factor $\geq 2^{277-151} = 2^{126}$ regarding Grover's search algorithm compared to AES, even if only the 256-bit version is used. Regarding the quantum algebraic attacks, we can say that there is no attack outlined yet for the version with HKDF. Regarding the version with cSHAKE, the quantum algebraic attack has to be adapted to two phases: KECCAK and then on the rounds functions. The complexity of such a solution remains to be investigated.

We consider EAES a sensible candidate for a first-generation post-quantum secure symmetric encryption. It runs efficiently on currently used devices and is compatible with existing hardware implementations. We hope it is able to contribute to a smooth transition into the new post-quantum cryptographical era.

References

[1] S. Kovac and J. Underhill, Towards post-quantum symmetric cryptography, *https://eprint.iacr.org/2019/553*.

[2] J. Underhill, The CEX Cryptographic library in C++, *https://github.com/Steppenwolfe65/CEX*.

[3] A. Biryukov and D. Khovratovich, Related-key Cryptanalysis of the Full AES-192 and AES-256, *https://eprint.iacr.org/2009/317.pdf*.

[4] A. Biryukov, O. Dunkelman, N. Keller, D. Khovratovich and A. Shamir Key Recovery Attacks of Practical Complexityon AES-256 Variants With Up To 10 Rounds *http://www.wisdom.weizmann.ac.il/∼orrd/crypt/PracticalAES256.pdf*.

[5] S. Lucks, Attacking Seven Rounds of RIJNDAEL under 192-bit an 256-bit Keys, *https://madoc.bib.uni-mannheim.de/10615*, 2000.

[6] M. Grassl, B. Langenberg, M. Roetteler, R. Steinwandt, Applying Grover's algorithm to AES: quantum resource estimates, *https://arxiv.org/abs/1712.06239*, 2018.

[7] Y.-A. Chen, X.-S. Gao, Quantum Algorithms for Boolean Equation Solving and Quantum Algebraic Attack on Cryptosystems, *https://arxiv.org/abs/1712.06239*, 2018.

[8] L. K. Grover, A fast quantum mechanical algorithm for database search, *Gary L. Miller, editor, Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing (STOC 1996), pages 212–219* ACM, 1996.

[9] Federal Information Processing Standards Publication 197, NIST, Announcing the ADVANCED ENCRYPTION STANDARD (AES), *https://csrc.nist.gov/csrc/media/publications/fips/197/final/documents/fips-197.pdf*, 2000.

[10] C. Cid, Information Security Group, University of London, Algebraic Analysis of AES, *https://www.cosic.esat.kuleuven.be/ecrypt/AESday/slides/AES-Day-CarlosCid.pdf*, October 2012.

[11] J. Baloo, KPN, Everything is Quantum – The EU Quantum Flagship, *https://2017.pqcrypto.org/conference/slides/baloo.pdf*, 2017.

[12] J. Kelsey, S. -J. Chang, R. Perlner, SHA-3 Derived Functions: cSHAKE, KMAC, TupleHash and ParallelHash, *https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-185.pdf*, December 2016.

[13] M. Amy, O. Di Matteo, V. Gheorghiu, M. Mosca, A. Parent, J. Schanck, Estimating the cost of eneric quantum pre-image attacks on SHA-2 and SHA-3s, *https://eprint.iacr.org/2016/992.pdf* QCrypt, 2016.

[14] A. Barenco, C.H. Bennett, R. Cleve, D.P. DiVincenzo, N. Margolus, P.W. Shor, T. Sleator, J. Smolin, H. Weinfurter, Elementary gates for quantum computation, *Phys. Rev. A, 52(5):3457–3467*, 1995.

[15] N. Wiebe, M. Roetteler, Quantum arithmetic and numerical analysis using Repeat-Until-Success circuits, *https://arxiv.org/abs/1406.2040*, 2014.