Hidden Shift Quantum Cryptanalysis and Implications

Xavier Bonnetain[1,2] and María Naya-Plasencia[2]

[1] Sorbonne Université, Collège Doctoral, F-75005 Paris, France
[2] Inria de Paris, France

Abstract. At Eurocrypt 2017 a tweak to counter Simon's quantum attack was proposed: replace the common bitwise addition, with other operations, as a modular addition. The starting point of our paper is a follow up of these previous results:

First, we have developed new algorithms that improve and generalize Kuperberg's algorithm for the hidden shift problem, which is the algorithm that applies instead of Simon when considering modular additions. Thanks to our improved algorithm, we have been able to build a quantum attack in the superposition model on Poly1305, proposed at FSE 2005, largely used and claimed to be quantumly secure. We also answer an open problem by analyzing the effect of the tweak to the FX construction.

We have also generalized the algorithm. We propose for the first time a quantum algorithm for solving the problem with parallel modular additions, with a complexity that matches both Simon and Kuperberg in its extremes. We also propose a generic algorithm to solve the hidden shift problem in non-abelian groups.

In order to verify the theoretical analysis we performed, and to get concrete estimates of the cost of the algorithms, we have simulated them, and were able to validate our estimated complexities.

Finally, we analyze the security of some classical symmetric constructions with concrete parameters, to evaluate the impact and practicality of the proposed tweak, concluding that it does not seem to be efficient.

Keywords: quantum cryptanalysis, hidden shift problem, Simon-meets-Kuperberg, Poly1305, symmetric cryptography, modular additions.

1 Introduction

As years go by, the existence of quantum computers becomes more tangible and the scientific community is already anticipating the consequences of the induced breakthrough in computational power. Cryptology is one of the affected disciplines. Indeed, the current state-of-the-art asymmetric primitives would become insecure, and the NIST has launched a competition for finding new primitives.

Symmetric cryptography, essential for enabling secure communications, seemed much less affected at first sight: for a long time, the greatest known threat was Grover's algorithm, which allows exhaustive key searches in the square root of

the normal complexity. Thus, it was believed that doubling the key lengths suffices to maintain an equivalent security in the post-quantum world.

At the same time, the security proofs in symmetric cryptography often need to make unrealistic assumptions. Therefore, the security of concrete symmetric primitives is mainly based on cryptanalysis: we only gain confidence in their security through extensive and continuous scrutiny. Hence, it is not possible to determine if a symmetric primitive is secure in the quantum world without first understanding how a quantum adversary can attack it. Lately, new results in this direction have appeared, like quantum generic meet-in-the-middle attacks on iterative block ciphers [26], quantum linear and differential attacks [28], or improved algorithms for collisions or multicollisions [16,25].

Using Simon's algorithm. Some other recent attacks are based on the quantum algorithm of Simon [41], like [32,33,39,11] that respectively analyze the security of 3-round Feistel schemes, the Even-Mansour construct, related-key attacks and quantumly break AEZ. For instance, in [33], the authors showed how the popular Even-Mansour construct, classically secure, would be completely broken in the quantum world when considering the superposition scenario. At Crypto 2016, Simon's algorithm was used to break well-known modes of operation for MACs and authenticated encryption as well as for providing quantum slide attacks, with a complexity linear in the block size [27] (see also [40]). An analysis of the FX construct against quantum adversaries was presented at Asiacrypt 2017 [35]. A combination of Grover and Simon showed it was much less secure than expected, and for instance the PRINCE cipher is broken in the quantum setting. These surprising results were the first clearly showing that doubling the key-length of symmetric primitives is not enough – in some cases – to provide an equivalent security against quantum adversaries when considering the superposition scenario, that we discuss next.

The attack model. These last mentioned attacks apply in a scenario of superposition quantum queries. It means that the adversary is not only allowed to perform local computations on a quantum computer[3], but is also allowed to perform superposition queries to a remote quantum cryptographic oracle, to obtain the superposition of the outputs. These attacks have been described as *superposition attacks* [19], *quantum chosen message attacks* [10] or *quantum security* [48].

This is a strong model for the attacker, but there are very good arguments for defending the interest of studying the security of symmetric primitives in this setting (see for instance [24] or [23] for more detailed justifications of the model):

1. This model is simple. Using another model would imply artificial and hard to respect measures with respect to cryptographic oracles in a world with quantum resources, with complex manipulations of yet uncertain outcome [4].
2. Safety in this model implies safety in any other scenario, even advanced ones (*e.g.* obfuscated algorithms).

[3] In [9,13,49,45], it can query a quantum oracle with an arbitrary quantum input.

[4] Implementations of theoretically secure quantum cryptography remain yet not fully understood, as shown by the attacks [50,36,46]

3. Though powerful, this model is not trivial: all primitives are not broken in it. Actually, several resistant constructions have been proposed [4,42,24,10,19].

Countering the attacks [2]. At Eurocrypt 2017, a proposal for countering the attacks from [27] was presented [2]. The authors propose to replace the common $(\mathbb{Z}/(2))^n$ addition, vulnerable to the Simon algorithm, with other operations that imply a harder problem to solve. The most promising of these operations, because of efficiency and implementations issues, already used in several symmetric schemes (*i.e.* [38,47,43]), is addition over $\mathbb{Z}/(2^n)$, i.e. modular addition. The authors claim the quantum hardness of the hidden shift problem proves the security of their proposal against quantum chosen-plaintext attacks.

This approach is a priori an interesting direction to analyze and study. The authors did not provide a more profound analysis of the impact of various parameters on the security. The attacks are no longer $O(n)$ (with n the state size) when using the modular addition, as Simon's algorithm does not apply anymore, but we could describe attacks that are still a lot faster than the generic ones by using Kuperberg's algorithm [30], e.g. $2^{O(\sqrt{n})}$ instead of $O\left(\sqrt{2^n}\right)$.

Indeed, classically, a symmetric primitive is considered secure when no attack better than the generic attack exists. While the complexity of the generic exhaustive search is exponential ($2^{n/2}$), the quantum attacks on primitives with modular additions have a sub-exponential complexity. This implies a need for a redefinition of *security*, when building *secure* primitives with these counter measures, as the best generic attacks that define the security of the cipher (based on Kuperberg now) will be better than the exhaustive search. Also, concrete proposals for the size of the primitives needed in order to provide the typical security needs (*i.e.* 128 bits) are missing.

Describing in detail the new best quantum attacks on the proposed constructions is necessary to provide concrete designs for a given wanted security. To evaluate the interest of such constructions, we should compare these designs with concrete parameters to other (quantum-secure) ones, like AES [18].

On Kuperberg's complexity, improvements, applications. Studying in detail Kuperberg's algorithm, proposing improvements and simulating the complexity for concrete parameters has not been done before and is of algorithmic general interest. These analysis are indispensable to size the primitives. Hidden shift algorithms have an impact beyond the symmetric variants we just mentioned, and can threaten other primitives, such as Poly1305 [6], which uses modular additions. Hidden shift problems also arises in some other cryptographic area, such as isogenies. They are for example relevant to assess the security of CSIDH [15].

1.1 Our contributions

1. Kuperberg's algorithm: improvement, generalization. We studied Kuperberg's quantum algorithm for hidden shifts in the group $\mathbb{Z}/(N)$ [30] and its

applications in symmetric cryptography.[5] We focus on the groups $\mathbb{Z}/(2^n)$, which are vastly used in symmetric cryptography. The original algorithm retrieves one bit of the secret shift at a time and uses a reducibility property to get the next bit. We propose a variant that performs better by getting all the bits in one step, allowing a drastic cost reduction of the attack on Poly1305. We also propose a way of solving the hidden shift problem in non-abelian groups.

2. Simon Meets Kuperberg. We propose a new quantum algorithm that considers a generalization for products of cyclic groups ($\mathbb{Z}/(2^p)^w$ and its subgroups), commonly used in symmetric primitives. The problem is more easily solvable in these groups than in $\mathbb{Z}/(2^{wp})$. Our complexity analysis shows how it meets Simon ($w = 1$) and Kuperberg ($p = 1$) in each extreme.

3. Simulation of the algorithms. We have implemented the classical part of these algorithms (Kuperberg, improved Kuperberg and Simon-meets-Kuperberg) and simulated it in order to estimate the asymptotic query complexity, and to get values for parameters of interest, verifying the expected complexities. Our code is available as *additional material* A and will be made publicly available.

4. Attack on Poly1305 in the superposition model. We propose a quantum attack on Poly1305 [6], a MAC that has been standardized for TLS 1.2 [34] and 1.3 [1], and is notably used by OpenSSH, Firefox and Chrome. In [8] a classical and quantum security of 128 bits is claimed for Poly1305: *"'Information-theoretic' MACs such as GMAC and Poly1305 already protect against quantum computers without any modifications: their security analysis already assumes an attacker with unlimited computing power."* Our attack, that works in the superposition model, has a complexity of 2^{38} and uses our improved Kuperberg's algorithm. It recovers half of the 234-bit key, allowing some forgeries. The attack is not a direct application of the algorithm and requires some additional techniques.

5. Attack on the FX variants. We answer an open question asked in [35], assessing the quantum security of the FX construction with any group law. If the inner key addition is done with a commutative group law, the security gain of the construct is marginal, and the best we can hope to achieve with a non-abelian group is a gain of around $n/3$ bits of security for an n-bit inner key.

[5] Even if some later algorithms have been developed and are more efficient, we focus on the original algorithm for two main reasons. We focus on quantum query and time complexity and the gain from [37] is in memory and [20] needs an exponential time classical post-processing. Moreover, we want concrete values and not asymptotic exponents and the algorithm in [31] is far harder to estimate precisely.

6. Evaluate the proposed countermeasures from [2]. The final aim was to determine how to size the symmetric primitives in order to offer a certain desired security, and to decide whether the proposed countermeasure was sufficient, and efficient enough in practice. Using modular additions in vulnerable constructions instead of xors for key addition increases the complexity of the corresponding quantum key-recovery attack, but we show that the proposal from [2]does not seem practical, and would require an internal state size of a few thousand bits, to be compared with the size of the internal state of AES-256, which is 128 bits.

Organization of the paper. Section 2 introduces some preliminary material. Section 3 presents our study on Kuperberg's algorithm and our improvement, several generalizations, our simulations and the inferred complexities. Section 4 describes our new quantum algorithm for parallel additions. Section 5 presents the first quantum attack on Poly1305 in the superposition model, using Kuperberg's algorithm. Section 6 estimates the strength of the FX construct with new group laws. Section 7 applies our previous results to actual symmetric primitives, deducing the key or internal state size that must be used in those constructs to offer a desired quantum security. The paper ends with a conclusion in Section 8.

2 Preliminaries

In this section, we present the quantum symmetric attacks from [27] and [35], the proposed solution from [2] and our cost model.

2.1 Quantum attacks using Simon's algorithm from [27]

In [27] Simon's quantum algorithm was applied to cryptanalyse several widely used modes of operation and CAESAR candidates. This was possible due to the exponential speedup of Simon's algorithm, that solves the following problem:

Let $f : \{0,1\}^n \to \{0,1\}^n$. Given the promise that there exists $s \in \{0,1\}^n$ such that for any $(x,y) \in \{0,1\}^n$, $[f(x) = f(y)] \Leftrightarrow [x \oplus y \in \{0^n, s\}]$, find s.

The authors applied Simon's algorithm to find a secret information in time linear in the block size ($O(n)$ instead of $O(2^{n/2})$ classically). One implication of the problem was not verified in the attacks: with a small probability, we might have $f(x) = f(y)$ and $x \oplus y \notin \{0^n, s\}$. However, they showed that the algorithm is still efficient with a random function in place of a random permutation.

2.2 Solution proposed in [2]

In [2], the authors propose to change, in the primitives broken by [27], the group law, to prevent the use of Simon's algorithm. They also propose a security reduction from the primitives to the corresponding hidden shift problem, and claim that they are safe, as no polynomial algorithm for these problems is known. They notably propose $(\mathbb{Z}/(2^n), +)$ (for which Kuperberg's algorithm is, in a sense, not a threat, as it is superpolynomial), or the symmetric group $\mathcal{S}_n$.

2.3 Cryptanalysis of the FX construct [35]

The FX construct [29] uses a block cipher E_k and two additional keys k_1, k_2, and is defined as $\mathrm{FX}_{k_0,k_1,k_2}(x) = E_{k_0}(x \oplus k_1) \oplus k_2$. It can be broken by combining Simon's and Grover's algorithms : one can perform an exhaustive search on k_0 and then see the FX construct as an Even-Mansour with the public permutation E_{k_0}, which can be broken with Simon's algorithm. The authors left as an open problem the case where the whitening keys were added with modular addition.

2.4 Cost Model

We're interested in the explicit costs of the algorithms we study. These algorithm have all a similar shape: they use a generation circuit that produces some relevant qubits, a combination circuit that uses the produced qubits, and a control circuit have chooses which qubits are to be combined. The generation circuit is a Quantum Fourier Transform applied to an oracle, whose total cost in time and memory is the number of queries. The combination circuit has a fixed cost, and can only be used once per query. The control circuit can be more complex, but only have to reason about classical values, and hence can be implemented purely classically, and its cost in time and memory will be the cost in query, with a polynomial overhead. As we expect that a classical computer will be far more efficient than a quantum computer to apply the same number of gates, we estimated that the bottleneck of our algorithm will be the quantum part of it, and that the relevant cost unit here is the number of queries.

3 New Results on Kuperberg's Algorithm

In this section, we study Kuperberg's quantum algorithm for solving the hidden shift problem. While the final aim is to be able to accurately estimate the complexities of the cryptanalysis on primitives whose security rely on the hidden shift problem, we have also performed a more profound work that verifies and helps better understanding Kuperberg's algorithm and its performance. We propose a new variant of the algorithm that reduces its cost, and that will allow to build the performant attack from section 5. We've implemented the classical part of these algorithms and made some simulations in order to get concrete estimates of the asymptotic complexity and values for parameters of interest, that match and refine the theoretical expectations. We finally propose a generalization of the algorithm for the case of non-abelian groups.

3.1 Hidden Shift Problem and Quantum Algorithms

The hidden shift problem (HSP) is defined as follows:

Let f, g two injective functions, $(\mathbb{G}, \cdot)$ a group. Given the promise that there exists $s \in \mathbb{G}$ such that, for all x, $f(x) = g(x \cdot s)$, retrieve s.

We say that f is a shifted version of g, the shift being s. To estimate the complexity, we consider $n = \log_2 |\mathbb{G}|$. The hardness of the problem depends

vastly on the group law. If it is a bitwise xor, Simon's algorithm [41] solves it in polynomial time. If the group law is a modular addition, it can be solved with a linear number of queries [20], but this method requires an exponential-time classical post-processing, and as such, won't be interesting for us. The firsts sub-exponential (in quantum query and quantum and classical time) algorithms are presented in [30]. They have a time and space complexity in $2^{O(\sqrt{n})}$ for a group of size 2^n. Other variants were developed later, with an algorithm with quantum polynomial space, but slightly worse time complexity, in $2^{O(\sqrt{n \log(n)})}$ [37], and some algorithms in [31], that generalize the previous one, allowing some trade-offs between classical and quantum memory and time.

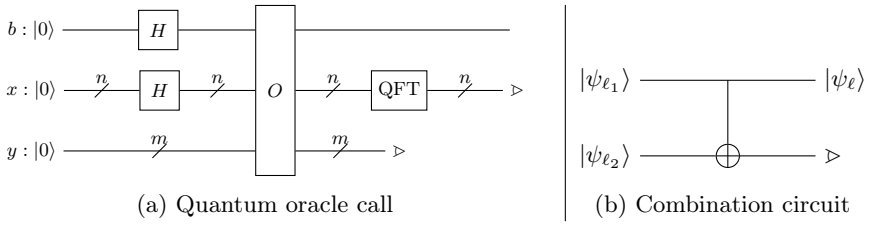
From this point, we focus on additions modulo a power of 2, as they are very common in symmetric cryptography, due to implementation reasons.

Single modular addition. All these algorithms are in two parts: an oracle that calls f and g to produce some qubits and a combination circuit that transforms them into more interesting ones. The combination part uses the quantum oracle

$$O : |b\rangle |x\rangle |y\rangle \mapsto \begin{cases} |0\rangle |x\rangle |y \oplus f(x)\rangle \text{ if } b = 0 \\ |1\rangle |x\rangle |y \oplus g(x)\rangle \text{ if } b = 1 \end{cases}.$$

Generation. The oracle circuit (Figure 1a) produces the uniform superposition in the registers b and x with Hadamard gates (H), feeds them to the oracle (O), and then measures register y. This measurement gives a result y_0 and collapses the b and x registers in the state $\sum_{f(x)=y_0} |0\rangle |x\rangle + \sum_{g(x)=y_0} |1\rangle |x\rangle$, which is, thanks to the promise, the state $|0\rangle |x_0\rangle + |1\rangle |x_0 + s\rangle$ for a given (unknown) x_0. We then apply a quantum Fourier transform (QFT) on the x register and measure the result. This gives us an ℓ with a uniform probability, and collapses the remaining qubit in the state $|\psi_\ell\rangle = |0\rangle + \exp\left(2\iota\pi\frac{s\ell}{2^n}\right)|1\rangle$.

This qubit depends on s, but is not directly exploitable. The qubit $|\psi_{2^{n-1}}\rangle = |0\rangle + \exp(\iota\pi s)|1\rangle$ is very interesting, as it is $|+\rangle$ if the lowest bit of s is 0, and else is $|-\rangle$. Hence, if me measure it in the $\{|-\rangle, |+\rangle\}$ basis, we get one bit of s.



(a) Quantum oracle call (b) Combination circuit

Fig. 1: Quantum circuits for Kuperberg's algorithm

Combination. We have then a combination part, that uses the produced qubits to generate some more interesting ones. The combination is done with the circuit in Figure 1b, that consists of one controlled-not and a measurement of the

second register. By doing so, we destroy two elements in order to produce one. Before the measurement, the system is in the state CNOT $|\psi_{\ell_1}\rangle |\psi_{\ell_2}\rangle =$

$$|00\rangle + \exp\left(2\iota\pi\frac{s(\ell_1+\ell_2)}{2^n}\right)|10\rangle + \exp\left(2\iota\pi\frac{s\ell_2}{2^n}\right)\left(|01\rangle + \exp\left(2\iota\pi\frac{s(\ell_1-\ell_2)}{2^n}\right)|11\rangle\right)$$

If we measure a 0 we'll get the qubit $|\psi_{\ell_1+\ell_2}\rangle$, and if we measure a 1 we'll get $|\psi_{\ell_1-\ell_2}\rangle$. Both outcomes are equiprobable. If we only see the ℓs and not the qubits, we can produce random numbers and combine them 2 by 2 using an operation, $+$ or $-$, that we discover afterwards. All the numbers can be used once, and we want to obtain 2^{n-1}. This abstract problem would be a problem of subset-sum modulo 2^n if the operation at each step was fixed, and not picked randomly in $\{+,-\}$, as we would want to find a tuple satisfying

$$\sum_{i\in I}\ell_i = 2^{n-1} \mod 2^n.$$

However, in our situation, the problem is closer to finding a tuple satisfying

$$\sum_{i\in I}\delta_i\ell_i = 2^{n-1} \mod 2^n,$$

with $\delta_i \in \{-1,1\}$ unknown before the actual destructive computation.

With these quantum tools, we can produce random elements and combine them, but we need an algorithm to choose which elements to combine.

Choosing the elements to combine. As a combination produces either $a+b$ or $a-b$, we need to find a property preserved in both cases, to not lose everything if the wrong outcome occurs. It turns out divisibility by 2 is such a property: if both a and b are multiples of 2^k, $a+b$ and $a-b$ will also be multiples of 2^k. Hence comes naturally the idea of the combination algorithm: from the elements we have, generate elements with a higher divisibility by 2, until we get 2^{n-1}. To achieve this, we can combine elements such that $a+b$ or $a-b$ has a high divisibility by 2 (e.g. have a long trail of 0 in their binary representation).

Hence, an algorithm to find 2^{n-1} is then, beginning with the odd numbers, to combine the two elements that can produce a number with the highest possible divisibility by 2. As this property corresponds to the longest partial collision in the binary representation of the elements, they can be efficiently found with a radix tree. There is however one caveat: we don't want the useless 0 element, so we try to not combine two identical elements, or one element and its opposite.

As the interesting a and b collide on their lowest bits, they have the same divisibility by 2, that is, $a = 2^k(2a'+1)$ and $b = 2^k(2b'+1)$. Then, $a+b = 2^{k+1}(a+b+1)$ and $a-b = 2^{k+1}(a-b)$. This means that even in the bad case (with a small divisibility by 2), we still get a slightly better divisibility by 2. Then, the algorithm consists in using this heuristic until we get 2^{n-1}.

This is Algorithm 1, from [30]. The paper also presents a sketch of proof that its complexity is in $\widetilde{O}\left(2^{\sqrt{2\log_2(3)n}}\right)$. As the paper only focuses on the asymptotic exponent complexity, the polynomial part is not well known. We can however deduce from the sketch of proof a complexity in $O\left(n^2 2^{\sqrt{2\log_2(3)n}}\right)$ to retrieve the whole hidden shift, which, due to the way the sketch of proof works, may not be a tight bound (both for the polynomial and the exponent).

Algorithm 1 Kuperberg's original algorithm [30], without qubits, in base 2

Generate N random numbers in $\mathbb{Z}/(2^n)$ ▷ Queries
Separate them in pools P_i of elements divisible by 2^i and not 2^{i+1}
for $i := 0$ to $n-2$ **do**
  while $|P_i| \geq 2$ **do**
    Pop two elements (a, b) of P_i where $a+b$ or $a-b$ has the highest possible divisibility by 2 (and is not 0)
    c is chosen randomly in $\{a+b, a-b\}$ ▷ Combination
    Insert c in the corresponding P_j
    if $P_{n-1} \neq \emptyset$ **then** ▷ Found $|\psi_{2^{n-1}}\rangle$?
   return Found
    end if
  end while
end for
return Failure

If this succeeds, we get the value of the lowest significant bit of the hidden shift, s_0. We have then to retrieve the other bits of s. This can be done using a recursive procedure: with the knowledge of $s \mod 2 = s_0$, we can construct the functions $f'(x) = f(2x)$ and $g'(x) = g(2x + s_0)$, that have the hidden shift $s' = (s - s_0)/2$ in $\mathbb{Z}/(2^{n-1})$. The 2nd bit of s is the lowest bit of s', and we can reapply the routine, and so on until we get all the bits.

3.2 New variant with improved the time complexity

In this section we propose an optimization of the previous algorithm that allows to perform the attack in Section 5. Previously each bit of the shift were retrieved independently. We have noticed that the remaining qubits of each step can be reused. The phase of the element ℓ is $2\pi\frac{\ell s}{2^n} = 2\pi\frac{\ell(s_0+2s')}{2^n} = 2\pi\frac{\ell s'}{2^{n-1}} + 2\pi\frac{\ell s_0}{2^n}$. If $s_0 = 0$, we can reuse them directly as elements of $\mathbb{Z}/(2^{n-1})$ for the next step (we just have to see the them modulo 2^{n-1}, that is, drop the most significant bit).

If it is 1, we have an additional phase of $2\pi\frac{\ell}{2^n}$ that prevents us to do so. We can get rid of it by applying a phase shift gate of angle $-2\pi\frac{\ell}{2^n}$ before reusing it. We can use that to reuse all our remaining qubits. Moreover, in the 2nd phase, the interesting elements are `010...0` and `110...0`, that is, any element of the penultimate pool. Likewise, we can use an element in a pool to retrieve one bit of the shift if we know all the preceding bits. We may get one less combination by pool, but as it will be the last one, it would have been the least interesting one. This strategy leads to the improved Algorithm 2, where we ensure that each pool stays non-empty. If we miss one qubit, won't have the value of the corresponding bit of s, and, as we won't know which rotations to do, on the following bits of s.

This variant is still subexponential, but due to the fact that we do only one pass instead of n, we have a polynomial gain, and we can estimate its complexity, using the same arguments as for Algorithm 1, as $O\left(n2^{\sqrt{2\log_2(3)n}}\right)$. A more precise complexity estimation is done in Section 3.4.

Algorithm 2 Variant to get all the qubits in one pass

Generate N random numbers in $\mathbb{Z}/(2^n)$
Separate them in pools P_i of elements divisible by 2^i and not 2^{i+1}
for $i := 0$ to $n-2$ **do**
  while $|P_i| \geq 3$ **do** ▷ Ensures P_i stays non-empty
    Pop two elements (a, b) of P_i where $a+b$ or $a-b$ has the highest possible divisibility by 2 (and is not 0)
    c is chosen randomly in $\{a+b, a-b\}$
    Insert c in the corresponding P_j
    if $\forall i \in [0, n-1], P_i \neq \emptyset$ **then return** Found
    end if
  end while
end for
return Failure

3.3 Approximated promise

In concrete attacks, we may want to use this algorithm on functions that respect partially the promise. We study in this section various cases.

Lemma 1 (Unwanted collisions). *Let $f : \mathbb{Z}/(2^n) \to \mathbb{Z}/(2^n)$ a random function , $s \in \mathbb{Z}/(2^n)$. We can retrieve s in Q quantum queries if we can solve the hidden shift problem in $\mathbb{Z}/(2^n)$ with a permutation using Q/e quantum queries*

Proof. This case was studied in section 2.2 of [27] in the context of Simon's algorithm. It corresponds to the hidden subgroup problem with a non-injective function. It then still respect for all x, $f(x) = g(x+s)$ for a secret s.

Let's decompose each step. The measurement after the oracle produces

$$|0\rangle \sum_{i=1}^{c} |x_i\rangle + |1\rangle \sum_{i=1}^{c} |x_i + s\rangle$$

and measured $f(x_i)$ with probability $c/2^n$. After the QFT, the measurement will give us a label ℓ and a qubit

$$\left(\sum_{i=1}^{c} \exp\left(2\iota\pi \frac{x_i \ell}{2^n}\right)\right)\left(|0\rangle + \exp\left(2\iota\pi \frac{s\ell}{2^n}\right)|1\rangle\right)$$

As a qubit is invariant by a global phase shift, we still get a valid element. However, it is not uniformly sampled, and the probability of getting a given ℓ is

$$p = \frac{1}{c2^n}\left|\sum_{i=1}^{c} \exp\left(2\iota\pi \frac{x_i \ell}{2^n}\right)\right|^2.$$

Notably, the case $\ell = 0$, which is useless for us, is the most probable.

It is known [22] that for a random function, the expected number of images with r preimages is $2^n/(er!)$. The first measurement samples on the images, uniformly if it is a bijection, and proportionally to the number of preimages in

the general case. That means we'll have a probability of $r/(er!) = 1/(e(r-1)!)$ of getting an image with r preimages. We'll get a unique preimage with probability $1/e$, so that means with e times the number of samples, we'll get enough elements with only one preimage. This is a very rough approximation, as the multiple preimages induces only a bias on the generated elements.

Remark 1. Alternatively, we can consider the function $F(x) = (f(x), f(x+1), \dots)$, that has the same shifts as f, but has a smaller probability of unwanted collisions, at the cost of having to query f multiple time for one query of F.

Lemma 2 (Multiple shifts). *Let $(s_i)_{i\leq t} \in \mathbb{Z}/(2^n)^t$, let f, g two permutations of $\mathbb{Z}/(2^n)$ such that, for all x, i, $f(x) = g(x + s_i)$. The first bits of the s_i can be retrieved if and only if they are all equal. They can be retrieved by solving the HSP in $\mathbb{Z}/(2^k)$ with the same functions, with $2^k = \gcd_{i\neq j}(2^n, s_i - s_j)$.*

Proof. We can study what happens with two shifts: x, $f(x) = g(x+s) = g(x+t)$.

From these equalities, we can deduce that for all x and λ, $f(x) = f(x+\lambda(s-t)) = g(x+s+\lambda(s-t))$. The functions have in fact plenty of shifts: $s+\lambda(s-t)$, the exact number depending on the divisibility by 2 of $s-t$. The bits of x that are above this level have in fact no impact on the value of f, so this problem is degenerate: if $s - t = 2^k\mu$, we have an instance of the problem in $\mathbb{Z}/(2^k)$, with a hidden shift $s' = s \mod 2^k = t \mod 2^k$, and we have $2^k = \gcd(2^n, s-t)$. We cannot get the other bits of s or t, as all the $s + \lambda(s-t)$ are also valid shifts.

For more shifts, we need to consider the difference that have the smallest divisibility by two, that is, the gcd of all the differences with 2^n.

As the divisibility by two of the difference corresponds to an equality in the first bits, the lemma holds.

Remark 2. If we don't know that the functions have multiple shifts, or if the gcd is not known in advance, this is still detectable, as the labels we measure will always divide 2^{n-k}.

Proof. The formula of the probability of measuring ℓ is $p(\ell) = \frac{1}{c2^n}\left|\sum_{i=1}^{c} \exp\left(2\iota\pi\frac{x_i\ell}{2^n}\right)\right|^2$ with c shifts. This reduces to

$$\frac{1}{c2^n}\left|\exp\left(2\iota\pi\frac{x\ell}{2^n}\right)\sum_{\lambda}\exp\left(2\iota\pi\frac{\lambda 2^k\ell}{2^n}\right)\right|^2.$$

This is 0 if $\exp\left(2\iota\pi\frac{2^k\ell}{2^n}\right) \neq 1$, that is, if $2^{n-k} \nmid \ell$. This means we'll only get some ℓs with at least $n-k$ trailing zeros.

Lemma 3 (Partial shift). *Let f, g two permutations of $\mathbb{Z}/(N)$, $s \in \mathbb{Z}/(N)$, $X \subset \mathbb{Z}/(N)$ such that, for all $x \in X$, $f(x) = g(x + s)$. Then if the hidden subgroup problem in $\mathbb{Z}/(N)$ costs Q queries, we can retrieve s given quantum oracle access to f and g in Q queries, with probability $(|X|/N)^Q$.*

Proof. If we measure an $f(x)$ whose x is in X, then we have a valid element. This happens with probability $|X|/N$. If this is not the case, we get a malformed qubit. We can expect the algorithm to succeed only if all the Q queried elements are valid, which happens with probability $(|X|/N)^Q$.

Remark 3. It would also be possible for the algorithm to succeed if we have a way to identify the bad x from the value $f(x)/g(x)$, which is measured, as it would allow us to drop the corrupted qubit when we create it. The problem would then only concern the unidentified bad x.

Lemma 4 (Input restriction). *Let f, g be two permutations of $\mathbb{Z}/(N)$, $s \in \mathbb{Z}/(N)$ such that, for all x, $f(x) = g(x+s)$. Given a quantum oracle access to f and g restricted to the inputs $0 \leq x < 2^n$, if $0 \leq s < 2^{n-1}$ and the hidden subgroup problem in $\mathbb{Z}/(2^{n-1})$ can be solved in Q queries, s can be retrieved in eQ^2 queries.*

Proof. We are only given access to the interval $[0; 2^n)$. We cannot see the hidden shift in $\mathbb{Z}/(N)$ as a hidden shift in $\mathbb{Z}/(2^n)$. However, if s is small enough, we have an instance of a partial hidden shift, the valid elements being the ones such that $0 \leq x < 2^n$ and $0 \leq x + s < 2^n$. The probability to get a bad element is less than $s/2^n$ in this case. If we need Q queries, and $s/2^n \simeq 1/Q$, then the success probability will be greater than $(1 - 1/Q)^Q \simeq 1/e$. This fails for greater s.

However, we can query a subinterval of $[0; 2^n)$ for f and g. For $A \in [0; 2^{n-1})$, if we query $[0; 2^{n-1})$ to $f(x + A)$ and $g(x)$, we will retrieve s with probability $1/e$ if $0 \leq s - A < 2^{n-1}/Q'$, if we need Q' queries to solve the hidden subgroup problem in $\mathbb{Z}/(2^{n-1})$.

To retrieve s, we can sequentially test for all A multiples of $2^{n-1}/Q'$, until we reach 2^{n-1}. We then have Q' intervals to test, and each test costs Q' queries. Moreover, the algorithm will succeed if the test with the right guess of A succeeds, and can be verified with a few classical queries. As the right guess has a success probability greater than $1/e$, we expect to find the shift in eQ'^2 queries.

Remark 4. Here, we do a sequential test of the intervals. We could do a Grover search on it instead, but we would need to choose a slightly higher number of queries, in order to have a success probability very close to one. Moreover, it would force us to implement all the control system that chooses which qubit to collide quantumly and not classically.

Remark 5. We can see this method as trying to solve the HSP in $\mathbb{Z}$. It also shows that considering only the cyclic groups $\mathbb{Z}/(2^n)$ allows to solve the problem in *any* cyclic group in subexponential time, despite a different group structure.

3.4 Simulations

We have simulated the classical part of the algorithm by replacing the quantum measurements by random outcomes. We used this to get an estimate of the query complexity: We generate a certain amount of random numbers, and then combine them in order to get the values we want. We hence get an estimate of the success probability for a given amount of samples (Figure 2), and deduce from it an asymptotic complexity for a constant success probability. Table 1 shows some results of these simulations for different values of n, for 90% success probability. The code of this implementation is available as *additional material*.

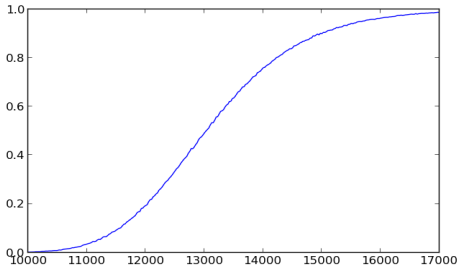


Fig. 2: Estimated success probability in the number of samples, for 64 bits

Figure 2 shows the estimated probability of retrieving the whole secret in function of the number of initial queries for a 64-bit secret. We've considered this parameter instead of some finer ones, such as the numbers of bits we retrieved because of the dependency between the bits we can retrieve: we have to retrieve them in order, and the first ones are the hardest to get. We can try to guess the missing bits, but as we destroy our qubits when we measure them, we can't recover from a wrong guess. It shows a transition from a negligible probability of success to a negligible probability of failure in less than a factor 2. As the algorithm is collision-based, it performs significantly better if it is run once with a bigger initial pool than many times with smaller pools. It also shows that the gap to get an arbitrarily small failure probability is small, which is useful if we want to combine it with another quantum algorithm, like a Grover search.

n	queries	$\log_2(\text{queries})$	$1.8 * \sqrt{n} - 0.5$	number of tests
16	118	6.9	6.7	10^6
32	826	9.7	9.7	10^6
64	14975	13.9	13.9	$5 * 10^5$
80	49200	15.6	15.6	10^5
128	$9.8 * 10^5$	19.9	19.9	$5 * 10^4$

Table 1: Some results of the simulation of Algorithm 2 for 90% success probability

We can then deduce an approximate complexity in query of $0.7 \times 2^{1.8\sqrt{n}}$ for a 90% success probability for Algorithm 2, which matches the exponent complexity of $\widetilde{O}\left(2^{\sqrt{2\log_2(3)n}}\right)$ of the less efficient Algorithm 1, as $\sqrt{2\log_2(3)} \simeq 1.8$. We see that the polynomial part is in fact a constant next to 1 for Algorithm 2, which hints that the bound in [30] for Algorithm 1, $O\left(n2^{\sqrt{2\log_2(3)n}}\right)$ to retrieve the last bit, is probably tight for the exponent part, but not for the polynomial part.

3.5 Hidden Shift in Non-Abelian Groups

In this section, we study the hardness of the HSP in other groups that can replace the xor (if we ignore the cardinality problem, as their size is not a power of 2), that is, $GL_2(q)$ (the group of invertible matrix of size 2×2 in $\mathbb{F}_q$ and $\mathcal{S}_n$, the symmetric group on n elements (which was proposed as an alternative to

$\mathbb{Z}/(2^n)$ in [2]), and propose a generic algorithm that can be used to assess the security provided by any group. Kuperberg's Algorithm solves the Hidden Shift Problem for cyclic groups, and a natural extension can be applied to products of cyclic groups (We've only considered powers of 2 here, but variants exists for all moduli). With the fundamental theorem of finite abelian groups, this means that variants of Kuperberg's algorithm can be used for all abelian groups. Moreover, as the exponent of the complexity for $\mathbb{Z}/(N)$ seems to be in all cases at most $\sqrt{2\log_2(3)\log_2(N)}$, and as a product of cyclic groups tends to reduce the complexity (see Section 4), we can estimate that the algorithm would cost at most $2^{\sqrt{2\log_2(3)n}}$ for an abelian group of size around 2^n (this would need a deeper investigation, but it gives a comparison point for a first estimate).

Generic non-abelian hidden shift. The hidden shift problem is a special case of a collision-finding problem. As such, it can be solved classically in $2^{n/2}$ queries. This classical cost matches the cost of an exhaustive shift search with Grover's algorithm. This is however not optimal, as collisions can be found more efficiently with a quantum computer, in $2^{n/3}$ queries [14]. If we also consider time, the gain is smaller, but the $2^{n/2}$ cost of Grover's algorithm can still be beaten [16]. This has two important implications for non-abelian groups. First, it suggests that for a fixed size, a cipher based on a hidden shift problem cannot match the best we can expect from a symmetric cipher. Second, it means we need to beat the cost of $2^{n/3}$ in order to have an interesting hidden shift algorithm.

Non-abelian hidden shift algorithm This algorithm can't be used with non-abelian groups. However, these groups contains some abelian subgroups (as the iterated powers of an element). We can apply the algorithm on such a subgroup, and it will succeed if the hidden shift lies in this subgroup. The idea is then to look for this situation. It can be done by considering the right cosets of the abelian subgroup $\mathbb{A}$. Indeed, all group elements can be uniquely written as ar, with $a \in \mathbb{A}$ and r a fixed representative of a right coset. The hidden shift can be decomposed as $s = s_a s_r$, and with $f(xs) = g(x)$, the relation $f(xs_a s_r) = g(x)$ can be seen as $f'(xs_a) = g(x)$, which is an instance of the hidden subgroup problem in $\mathbb{A}$. The complete algorithm is then to do a Grover search on the right coset, and then try to solve the problem in $\mathbb{A}$, as presented in Algorithm 3.

As the hidden shift is a joint property of the two functions, we cannot do a collision search as in the generic case, and need an exhaustive search. This procedure can also be used to solve the hidden period problem, as this is the case $f = g$. Hence, we can upper bound the hardness of the generic hidden shift problem, in function of the size of the group (around 2^n) and the size of its maximal abelian subgroup (around 2^a), which would be around $2^{(n-a)/2+\sqrt{2\log_2(3)a}}$.

$GL_2(q)$ contains $(q^2-q)(q^2-1)$ elements, some of which of order q^2-1. If we consider the group generated by such an element, we'll have $\mathbb{A}$ of size around $q^2 \simeq 2^{n/2}$. This means the complexity of the HSP in $GL_2(q)$ is at most around $q2^{2.6\sqrt{\log_2(q)}} = 2^{n/4+1.3\sqrt{n}}$. For $\mathcal{S}_n$, we can consider the subgroup generated by

Algorithm 3 Generic resolution of a hidden shift problem in a non-abelian group

$\mathbb{A} \leftarrow$ A commutative subgroup of $\mathbb{G}$
$\mathcal{C} \leftarrow$ {A representative of each right coset}
procedure GroverSearch($c \in \mathcal{C}$) ▷ c is assigned in quantum superposition
  $s \leftarrow$ Kuperberg's algorithm result in $\mathbb{A}$ for $f(xc)$ and $g(x)$
  if $f(xsc) = g(x)$ for a few random x **then**
    Mark c
  end if
end procedure ▷ c is now the representative of the coset of the shift
$s \leftarrow$ Kuperberg's algorithm result in $\mathbb{A}$ for $f(xc)$ and $g(x)$
return sc

the 2-cycles $\{(2i-1, 2i) | 1 \leq i \leq n/2\}$. As all the cycles are disjoint, this is an abelian group, of size $2^{n/2}$, isomorphic to $(\mathbb{Z}/(2))^{n/2}$. This allows the use of Simon's algorithm to find the hidden shift, with a total cost of around $n2^{-n/4}\sqrt{n!}$, for a group of size $n!$. This is however asymptotically worse than $\sqrt[3]{n!}$.

Group	$(\mathbb{Z}/(2))^{128}$	$\mathbb{Z}/(2^{128})$	$GL_2(2^{32})$	$\mathcal{S}_{34}$	$(\mathbb{Z}/(2))^{2^{30}}$	$\mathbb{Z}/(2^{5000})$	$GL_2(2^{100})$	$\mathcal{S}_{78}$
Size (bits)	128	128	$\sim$128	$\sim$128	2^{30}	5000	$\sim$400	$\sim$382
Cost	2^7	2^{20}	2^{37}	2^{42}	2^{30}	2^{127}	2^{126}	2^{127}

Table 2: Estimate of the cost of solving the hidden shift problem for some groups.

This suggests that even for non-abelian groups, the structure can lower the security of a scheme based on the hidden shift problem. To illustrate this we present in Table 2 the security of different group laws and their respective size.

4 New algorithm: Simon Meets Kuperberg

We describe in this section a new quantum algorithm, that, for the first time, solves efficiently the HSP problem when considering a product of cyclic groups, which often appears in symmetric constructions [5,44,21,7]. We also provide a simulation of the algorithm in section 4.3, showing that our complexity estimations are correct.

4.1 Solving the Problem for Parallel Modular Additions

An interesting generalization for, inter alia, symmetric cryptography is to consider p termwise additions modulo 2^w, that is, a modular addition in $\mathbb{Z}/(2^w)^p$. The hidden shift in this case is a vector $s = (s_1, \ldots, s_p)$ of p words of w bits each. The aim of this section is to propose a new algorithm that deals efficiently with that group. The first logic approach was to apply an adapted variant of Kuperberg, but its complexity significantly differs from optimal. We propose a new

algorithm which complexity is close to optimal. It exploits three facts in particular that allow us to consequently improve the complexity. In order to describe our algorithm, we need to previously adapt the first part of Kuperberg's algorithm by considering a quantum Fourier transform compatible with the group law, so the original one is changed to into a termwise variant. The oracle circuit produces the qubits $\left|\psi_{\ell_1,\ldots,\ell_p}\right\rangle = |0\rangle + \exp\left(2\iota\pi\frac{\sum s_i\ell_i}{2^w}\right)|1\rangle$, the product is replaced by an inner product. The combination circuit also works the same way, and produces a termwise sum or difference.

Better worst-case gain. The first fact that allows to improve the complexity over a basic algorithm is realizing that, though the combination strategy can be quite similar with a research of partial collisions on the lowest significant bits of each term, there is however a difference in the behavior in the disadvantageous case: while we gained only one 0 in the former situation, here, we'll get a 0 in each term in which we have a collision in the lowest 1 (p zeros) while the size of the corresponding list is big enough. We also have more choices in the combinations, and we can have various equivalent and incompatible possibilities, with collisions on different parts of the vector.

With $p+1$ equations we can always gain p zeros. As before, we can separate the elements in pools, depending on the divisibility by 2 of each term. Instead of looking at the position of the first one, we look at the position of the first one in any component of the vector to separate in pools. In each w pool, we can restrict ourselves to the bit slice corresponding to the corresponding level. This slice corresponds to a vector in $(\mathbb{Z}/(2))^p$. Hence, we can produce a vector that will fit in the next pool if we manage to find some linearly dependent vectors, that is, whose sum (or difference, as it is the same in $\mathbb{Z}/(2)$) is 0.

Recovering the shift. We realized that the elements with $\ell_i \in \left\{0, 2^{w-1}\right\}$ are of the form
$\left|\psi_{\ell_1,\ldots,\ell_p}\right\rangle = |0\rangle + \exp\left(\iota\pi\sum s_i\ell_i\right)|1\rangle$, so measuring them in the $\{|-\rangle, |+\rangle\}$ basis will give us the parity of $\sum s_i\ell_i$, that is, a linear equation in the parity bits of the s_i. In the case $w = 1$, we get a variant of Simon's algorithm for hidden shifts.

We describe next how to apply each approach separately, and then describe how our algorithm combines them to obtain an optimized complexity, that will be discussed and analyzed in section 4.2.

First Idea: Kuperberg's variant with a better worst-case gain. A simple strategy represented in Algorithm 4 is to mimic the former one: we apply directly the strategy with the first term to zero all its bits except the most significant one, and then process the second term, and so on. We can also apply it the other way around: we can see the vector $(s_1^{w-1}\cdots s_1^0, \ldots, s_p^{w-1}\cdots s_p^0)$ as the number $s_p^{w-1}s_{p-1}^{w-1}\cdots s_1^{w-1}s_p^{w-2}\cdots s_p^0\cdots s_1^0$, and apply directly the former strategy, until we get enough elements of the form $s_p^{w-1}s_{p-1}^{w-1}\cdots s_1^{w-1}0\cdots 0$ that

we can measure. Another approach is to weight all the possible combinations with the expected gain in the total number of trailing zeros, and choose the most favorable one. The first two have the advantage of being classically easy to implement, with a radix tree.

Algorithm 4 Variant 1 for termwise additions

Generate N random numbers in $\mathbb{Z}/(2^w)^p$
Separate them in pools P_i of elements with all p terms divisible by 2^i and at least one term not divisible by 2^{i+1}
for $i := 0$ to $w - 1$ **do**
  while $|P_i| \geq 2$ **do**
    Pop two elements (a, b) of P_i where $a + b$ or $a - b$ has the highest possible divisibility by 2 on each term
    c is chosen randomly in $\{a + b, a - b\}$
    Insert c in the corresponding P_j
  end while
end for
if $P_{n-1} \neq \emptyset$ **then return** Found
else return Failure
end if

Second Idea: $p + 1$ dependent equations always gain p zeros. There is however another way to use the parallel structure of the hidden subgroup: given $p + 1$ random elements, we can find a subset whose sum (or difference) will always be even on all the components: if we look at the parity vector of the elements, this corresponds to a linearly dependent subset of the vectors. This approach can be useful if p is big with respect to the size of the pools: with on average $p/2 + 1$ vectors, we can zero p bits. We can then iterate the technique to set to zero the next row of bits, and so on. This is described in Algorithm 5.

Moreover, seeing the elements in a pool as equations allows us to perform the same optimisation we have proposed for the case $p = 1$, to get all the secret in one pass. Instead of storing one element per pool, we have to store p elements that are linearly independent, that is, a full system of equations. As this optimisation does not depend on what we do to each pool, we can also apply it to improve Algorithm 4.

As, on average, we combine $(p/2 + 1)$ elements, we divide at each w step the pool by $(p/2+1)$. This algorithm has a complexity in $O((p/2+1)^w)$. If $w = 1$, it matches Simon's complexity (and is, indeed, Simon's algorithm). It is interesting for big p, as it is polynomial in p, but it quickly becomes costly if w rises, as it is exponential in it.

Our new algorithm: combining both ideas. As the two variants merge the elements to progressively create new elements with a greater number of zeros, we

Algorithm 5 Variant 2 for termwise additions

Generate N random numbers in $\mathbb{Z}/(2^w)^p$
Separate them in pools P_i of elements with each terms divisible by 2^i and at least one term not divisible by 2^{i+1}
`System` $= \emptyset$
for $i := 0$ to $w-1$ **do**
  Pop p elements from P_i linearly independent at the level i, put them in `System`
  `Basis` $= \emptyset$
  for $e \in P_i$ **do**
    if $\left\{x \mod 2^{i+1} \middle| x \in \{e\} \cup \texttt{Basis}\right\}$ is linearly independent **then**
      Add e to `Basis`
    else
      Find a linearly dependent subset J
      Compute $c = \underset{x \in J}{\pm} x$
      Insert c in the corresponding P_j
    end if
  end for
end for
if `System` is full **then return** Found
else return Failure
end if

can, to be more efficient, combine both methods. The algorithm uses a threshold to choose between the two approaches. It determines when we have to change the method of sieving. The value of this threshold is calculated and studied in the next section. Our new algorithm is described in Algorithm 6, where all the bits are also recovered in one pass thanks to our adapted improvement.

4.2 Complexity analysis

In this section we provide a complexity analysis of the previously described algorithm, that will depend on the relation between the parameters w and p. A summary can be found in Table 5.

We first estimate the complexity of Algorithms 4 and 5, and then combine these costs to compute the best thresholds, and derive the final complexity.

Complexity using partial collisions. To estimate the complexity of partial collisions in $(\mathbb{Z}/(2))^p$, we had the same approach as for the original algorithm: we performed simulations. As we do not have bad outcomes in this case, we expected a more efficient algorithm. An optimistic approach could estimate that the complexity is $2^{\sqrt{2p}}$, which would mean that a pool of 2^a elements produces a pool of 2^{a-1} elements that all have a more zeroes. In practice, this is not what we observed, and we found a complexity of around $2^{\sqrt{2.3p}}$, as presented in Table 3. This algorithm is far from the best method to solve this problem, but it can become relevant if we need a huge number of elements that are zeroed on p bits.

Algorithm 6 Combined algorithm for termwise additions

Generate N random numbers in $\mathbb{Z}/(2^w)^p$
Separate them in pools P_i of elements with each terms divisible by 2^i and at least one term not divisible by 2^{i+1}
`System` $= \emptyset$
for $i := 0$ to threshold **do** ▷ Partial collisions
  Pop p elements from P_i linearly independent at the level i, put them in `System`
  while $|P_i| \geq 2$ **do**
    Pop two elements (a, b) of P_i where $a + b$ or $a - b$ has the highest possible divisibility by 2 on each term
    c is chosen randomly in $\{a + b, a - b\}$
    Insert c in the corresponding P_j
  end while
end for
for $i :=$ threshold $+ 1$ to $w - 1$ **do** ▷ Zero-sum
  Pop p elements from P_i linearly independent at the level i, put them in `System`
  `Basis` $= \emptyset$
  for $e \in P_i$ **do**
    if $\left\{x \mod 2^{i+1} \middle| x \in \{e\} \cup \texttt{Basis}\right\}$ is linearly independent **then**
      Add e to `Basis`
    else
      Find a linearly dependent subset J
      Compute $c = \pm\limits_{x \in J} x$
      Insert c in the corresponding P_j
    end if
  end for
end for
if `System` is full **then return** Found
else return Failure
end if

Collision cost. We have two heuristics for collision cost. When we don't have enough elements to have some collisions, we estimate from our simulations the cost to produce N elements to be around $2^{\sqrt{2.3p+\log_2(N)^2}}$ in this situation.

The minimal cost is around $2N$, as we need to combine every element with another one. Doing simulations, we found that it is $(2N + 2^{p-2})/(1 + 1/2^p)$, asymptotically in N. The $2N$ comes from the fact that we do for almost all non-zero elements one combination, the 2^{p-2} corresponds to the small number of elements that don't have a total collision with another element, and the $(1+1/2^p)$ corresponds to the zero element that can naturally occur, with probability $1/2^p$.

Alternative approach. Instead of considering only one row of bits and try to zero it, we can consider a bigger chunk. That is, we want to produce N elements that have a certain amount (greater than p) of zeroes in their firsts bits. In order to estimate this cost, we refer to Kuperberg's original algorithm. In practice, the algorithm will be more efficient, but we can approximate it with the original

p	queries	$\log_2$(queries)	$\sqrt{2.3p} - 0.2$	number of tests
40	642	9.3	9.4	10^6
80	10770	13.4	13.4	10^6
100	33100	15.0	15.0	10^6
128	132600	17.0	17.0	10^5
140	228500	17.8	17.8	10^5
170	808000	19.6	19.6	10^4

Table 3: Some results of the simulations, for 90% success.

complexity. We can then estimate the cost of the algorithm to zero q bits to be $2^{\sqrt{2\log_2(3)q+\log_2(N)^2}}$. As before, this will not hold if we have to many elements to produce, as the minimal cost is $3N$, but we should never be in this regime.

Complexity using equations.

Lemma 5 (Equation cost). *A step of Algorithm 5 produces N elements with p zeroed bits using $N(p/2+1)$ elements on average, and needs p qubits.*

Proof. A step of Algorithm 5 uses random equations to produce a zeroed element. If we have p elements that form a basis of $\mathbb{Z}/(2)^p$, any other element is a linear combination of $p/2$ elements, on average, in this basis. If we have a basis, we can hence get an equation that has, on average $p/2+1$ elements, and that sums to zero on the p bits. We can then construct such a basis by choosing p random elements : if they form a free family, we have a basis, if not, we then have some elements that sums to zero. This allows to perform the algorithm on-the-fly : each time a new element arrives, we can try to form a basis with the previous one. If we new element is linearly independent, we add it in our memory. If it is not, we combine all the elements that sums to zero.

Theorem 1. *Algorithm 5 has a complexity in quantum queries and time of around $2(p/2+1)^w$. It needs $2p(w-1)$ quantum memory, plus the oracle cost.*

Proof. At each step, we store p independent elements that will allow us to retrieve p bits, and divide the remaining number of elements by $p/2+1$. At the end, we want p elements (with only p elements, as they would be random, the success probability is only of $1/e$, but we can get arbitrarily close to 1 with a fixed overhead). The total cost is then of

$$p(p/2+1)^{w-1} + p(p/2+1)^{w-2} + \cdots + p,$$

which reduces to $2(p/2+1)^w$. The total cost in quantum memory is then $p(w-1)$ qubits for the $w-1$ steps, and $p(w-1)$ qubits that will yield an equation in the bits of the shift, but that we cannot measure immediately. This cost in memory is optional, as we could do the algorithm w times, but we would then have to pay the constant overhead at each step, not only at the last one.

Remark 6. We found that the marginal cost of $(p/2+1)$ elements to produce one can be beaten if the total number of elements is huge by sorting them before searching for a zero-sum set. As extracting values from a radix tree naturally produces a sorted list, this was observed in our simulations.

Determining the total complexity. To determine the complexity, we will run the algorithm backwards : we estimate how many elements we need at a point of the algorithm, and then deduce how many elements we need before to obtain this number of elements. More precisely, we consider a fixed p, and estimate what we have to do to get the elements we want as w grows.

The final steps. The final steps uses Algorithm 5. The complexity to process w rows is then $C_0(p, w) = 2(p/2 + 1)^w$.

Changing to collision finding. With collision finding, we can erase one row and produce N elements at a cost of $2^{\sqrt{2.3p+\log_2(N)^2}}$. We will prefer to use this algorithm instead of the previous one if $C_0(p, w) > 2^{\sqrt{2.3p+\log_2(C_0(p,w-1))^2}}$. This means that $2(p/2 + 1)^w \geq 2^{\sqrt{2.3p+(1+(w-1)\log_2(p/2+1))^2}}$, which implies

$$w \geq w_0 = \lceil 1.15p/\log_2(p/2 + 1)^2 + 1/\log_2(p/2 + 1) - 1/2 \rceil.$$

This threshold is the number of steps in which we should use Algorithm 5, and the previous steps are solved using 4.

The total cost is then around $C_1(p, w) = 2^{\sqrt{(1+w_0\log_2(p/2+1))^2+2.3p(w-w_0)}}$. However, this approximation regime is only valid while the total number of elements is small enough.

Saturated regime of collisions. We saw before that the cost of zeroing one row is asymptotically around $2N$, and cannot outperform this bound. We can now estimate when our previous estimate violates this bound. This occurs when $2^{\sqrt{2.3p+\log_2(N)^2}} \leq 2N$, which implies $N \geq 2^{\frac{2.3p-1}{2}} \simeq 2^{1.15p}$. Using this constraint to the previous complexity, we get that w must be lower than

$$w_1 = \lfloor 2.3p/4 + w_0 - (1 + w_0 \log_2(p/2 + 1))^2/2.3p \rfloor.$$

We can still use the algorithm in this saturated regime, and estimate that one row can be erased if we divide by 2 the number of elements. Then, the complexity is $C_2(p, w) = 2^{w-w_1+\sqrt{(1+w_0\log_2(p/2+1))^2+2.3p(w_1-w_0)}}$.

Multiple steps at once. The complexity we got at the previous step does not have any constraint. It can however become irrelevant, as we have a better approximation if w is big enough, as it is exponential in w. Indeed, we can estimate that we can erase pw' zeros and get N elements at a cost of $2^{\sqrt{2\log_2(3)pw'+\log_2(N)^2}}$. This approximation will become relevant when $2^{\sqrt{2\log_2(3)p+\log_2(C_2(p,w-1))^2}} \leq C_2(p, w)$, which implies

$$w \geq w_2 = \lfloor \log_2(3)p - 1/2 + w_1 - \sqrt{(1 + w_0 \log_2(p/2 + 1))^2 + 2.3p(w_1 - w_0)} \rfloor.$$

The total complexity is then $C_3(p, w) = 2^{\sqrt{2\log_2(3)p(w-w_2)+(\log_2(C_2(p,w_2))^2}}$.

$w_0 =$	$\lceil 1.15p/\log_2(p/2+1)^2 + 1/\log_2(p/2+1) - 1/2 \rceil$
$w_1 =$	$\lfloor 2.3p/4 + w_0 - (1+w_0\log_2(p/2+1))^2/2.3p \rfloor$
$w_2 =$	$\lfloor \log_2(3)p - 1/2 + w_1 - \sqrt{(1+w_0\log_2(p/2+1))^2 + 2.3p(w_1-w_0)} \rfloor$

Table 4: Threshold points for Algorithm 6.

Constraint	Cost
$(w \leq w_0)$	$C_0(p,w) = 2(p/2+1)^w$
$w_0 \leq w \leq w_1$	$C_1(p,w) = 2^{\sqrt{(\log_2(C_0(p,w_0))^2 + 2.3p(w-w_0)}}$
$w_1 \leq w\ (\leq w_2)$	$C_2(p,w) = 2^{w-w_1} C_1(p,w_1)$
$w_2 \leq w$	$C_3(p,w) = 2^{\sqrt{2\log_2(3)p(w-w_2) + \log_2(C_2(p,w_2))^2}}$

Table 5: Complexity of Algorithm 6.

Simon Meets Kuperberg. From Table 5 we can see how Simon's complexity is met in the extreme case where $w = 1$ and Kuperberg's complexity is obtained when $p = 1$, as expected. It also shows that even if asymptotically in w, the complexity becomes closer to the complexity of Kuperberg's algorithm in $\mathbb{Z}/(2^{pw})$, the last w_2 rows of bits of the state do not provide as much security.

4.3 Simulations of the algorithm

We have performed various simulations of the algorithm, in order to confirm our models and theoretical complexities. For $w = 1$, the obtained complexity corresponds to solving an equation system, hence it needs around p queries, and our model holds. For $p = 1$, the complexity is reduced to $2^{\sqrt{2\log_2(3)p}}$, which corresponds to our previous simulations. We've considered two types of simulations in order to confirm our algorithm. First, as before, we simulated the success probability of the algorithm for a given input size. Second, we simulated the number of elements at each step of the algorithm, in order to see more precisely the accuracy of each model.

p/w	2/50	4/25	5/20	10/10	20/5	25/4	50/2	2/64	4/32	8/16	16/8	32/4	64/2
Model	17.7	17.5	17.3	15.3	14.2	13.7	10.4	20.1	19.9	18.8	16.6	15.2	11.1
Sim	17.9	17.5	16.9	15.3	14.4	13.9	10.6	20.3	19.7	18.2	16.7	15.4	11.2

Table 6: Simulations compared with our model, with a success probability of 90%, 1000 tests per estimation, in log scale, for $pw = 100$ and 128.

From Table 6, we see that our estimates correspond to the simulations in the ranges we were able to simulate, with a slightly pessimistic estimation when p is not too small and w is bigger than p. In order to estimate the accuracy of our different models, we also simulated the number of elements in each pool at the beginning of each step, as for example in Figure 3.

The computed thresholds for Figure 3 are (2,3,7). As they are in reverse, they correspond to (13,12,8) on the graph. The two curves are converging at around step 9, which suggests that our models 3 and 4 are slightly pessimistic. This is explained by the fact that model 3 neglects the gains of good combinations and model 4 the gains of bad combinations.

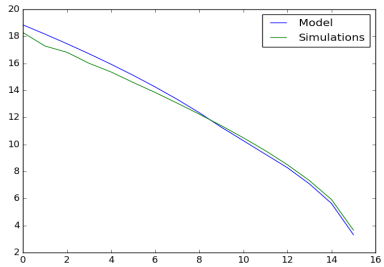


Fig. 3: Number of elements in each pool at each step for $p, w = 8, 16$, in log scale.

5 Cryptanalysis of Poly1305 in the Superposition Model

We propose in this section the first quantum superposition attack on the Poly1305 primitive, with a complexity of about 2^{38} in time and queries, that shows that it is not secure in the superposition model.

5.1 Poly1305 description

Poly1305 is a MAC designed by Bernstein [6]. It has been standardized for TLS 1.2 [34], is currently a part of a recommended cipher suite in the TLS 1.3 draft [1], and is notably supported by OpenSSH, Firefox and Chrome. The designer announced in [8] a classical and quantum security of Poly1305 of 128 bits. We'll describe Poly1305-AES, but our analysis works with any internal block cipher used.

Poly1305-AES uses two 128-bit keys (r, k) and a 128-bit nonce n, takes as input a variable-length message m considered as an array of 128-bit blocks, and outputs a 128-bit tag. For efficiency purposes, some bits of r are fixed to 0, which means it can only take 2^{106} different values. The function is

$$\text{Poly1305-AES}_{(r,k,n)}((m_i)_{i\leq q}) = \left(\sum_{i=1}^{q}(m_{q-i+1} + 2^{128})r^i \mod 2^{130} - 5\right) + \text{AES}_k(n).$$

5.2 Quantum attack in the superposition setting

For our quantum attack, we consider having access to the oracle

$$Poly_n : |m_1\rangle\,|m_2\rangle\,|0\rangle \mapsto |m_1\rangle\,|m_2\rangle\,|\text{Poly1305-AES}_{(r,k,n)}(m_1, m_2)\ \rangle\,.$$

The nonce is classical, and changes at each query. As we consider the superposition scenario, we consider that the function can be called in superposition. We aim at retrieving r (and not k), as r is sufficient to retrieve $\text{AES}_k(n)$ for any tag, which allows some forgeries. If one also wants k, one can perform a Grover search on it, with an additional cost of 2^{64}. In the additional material B we describe a distinguisher on Poly1305 and a simple key-recovery attack, but in this section we propose a more evolved attack that uses Kuperberg's algorithm.

Poly1305 uses a polynomial structure for hashing, and the commutative algebra $\mathbb{Z}/(2^{130}-5)[X]$ contains many possible shift structures, both in $\mathbb{Z}/(2^{130}-5)$ (with addition) and in $\mathbb{Z}/(2^{130}-6)$ (with multiplication). For example, one can consider the two functions $f(x) = xr + r^2 + 2^{128}(r + r^2)$ and $g(x) = xr + 2^{128}(r + r^2)$, which satisfies $f(x) = g(x + r)$. We cannot call them directly, but we can call $F(x) = \text{Poly1305-AES}_{(r,k,n)}(1, x)$ and $G(x) = \text{Poly1305-AES}_{(r,k,n)}(0, x)$, which, if the nonce is the same, also satisfy $F(x) = G(x + r)$.

There are two issues that do not allow the direct application of Kuperberg's algorithm: first, the nonce changes at each query, which means that in order to have $F(x) = G(x+r)$, we must compute F and G in only one query to Poly1305. This is feasible, as both are of the form $\text{Poly1305}(a(x))$, with $a(x)$ a function of x: one can compute $a_F(x) = (1, x)$ and $a_G(x) = (0, x)$ in superposition in an auxiliary register, and then call $Poly_n$ on it. Second, and more annoyingly, the inputs of Poly1305 are restrained to be between 0 and $2^{128} - 1$, which means we cannot sample all group elements.

This can still be solved by using Lemma 4, as we can query $[0; 2^{128})$. Solving the hidden shift in $\mathbb{Z}/(2^{127})$ costs around 2^{20}. We can thus set the interval size at 2^{106}. r can be retrieved if it is below 2^{127}. This is the case, as the bit constraints on r implies $r < 2^{124}$, which means we need only to test 2^{18} intervals. The total cost is then $2^{20} \times 2^{18} = 2^{38}$, for a success probability better than one half. We can check if the found r is the right one by trying to forge some valid messages, or we can use the distinguisher presented in Appendix B.

Grover acceleration. As the previous attack involves an exhaustive search on the correct interval among the 2^{18}, one might want to use Grover's algorithm, in order to gain up to 2^9 on the attack. We automatically lose a factor 2 because of the uncomputation of the algorithm. Moreover, we would need to compute all the qubit choices quantumly, and we must have a success probability of the inner function very close to one. All these factors make the attack more efficient in query (around 2^{31}), with a small time gain.

5.3 Impact of our improvements

The total cost of the attack depends vastly on the precise cost of Kuperberg's algorithm. The original algorithm, with an estimated complexity of around $n^2 2^{\sqrt{2\log_3(2^n)}}$, has here a cost of around 2^{34} queries. The total attack is then vastly more costly, around 2^{65}, which is very close to the cost of a simple exhaustive search on the key if AES-128 is used, and exceeds the cost of the simple quantum attack described in Appendix B. We could also use a Grover search, which would lead to a cost estimate of around 2^{50}, which is far higher than both the non-Grover variant of our attack (at 2^{38}) and the Grover variant (at 2^{31}).

6 Attack on the FX Construction

The FX construction, proposed by Killian and Rogaway [29], is a simple way to extend the key-length of a block cipher. It uses a block cipher E_{k_0} and two

additional keys k_1, k_2 whose length is the block size of the block cipher, and the new cipher is

$$\mathrm{FX}_{k_0,k_1,k_2}(x) = E_{k_0}(x \oplus k_1) \oplus k_2.$$

We can see it as an Even-Mansour construction, with a block cipher taking the role of the random permutation. The quantum security of this scheme has been studied by Leander and May in [35] in the superposition model. Their conclusion is that this construction is essentially as secure as the inner cipher E_{k_0}.

Their approach is close to the quantum attack against the Even-Mansour construction, with the addition that the key of the inner cipher has to be seeked. They consider the function $f(k, x) = \mathrm{FX}_{k_0,k_1,k_2}(x) \oplus E_k(x)$, which fulfils the promise $f(k_0, x) = f(k_0, x \oplus k_1)$. They then recover k_0 and k_1 by performing a Grover search on k_0, with a test function that is the application of Simon's algorithm to the partial function $x \mapsto f(k_0, x)$. If this function is periodic, then k_0 has a very high probability of being correct, and the period of the function is k_1. It can moreover be efficiently checked, by testing the periodicity for a few values. Once k_0 and k_1 are known, k_2 can be retrieved with one known plaintext/ciphertext pair. The total cost is around $2|k_1|2^{|k_0|/2}$.

This leads to some efficient attacks against the FX-based primitives DESX, PRINCE [12] and PRIDE [3]. For PRINCE and PRIDE, $|k_0| = |k_1| = 64$, the attack costs around 2^{39} queries and time, whereas for DESX, $|k_0| = 56$ and $|k_1| = 64$, the attack costs around 2^{35} queries and time [6].

The authors only considered the original construction, that uses some xors, and left as an open problem the evaluation of the security using another group law. We can here give an answer for the most natural variant, which is to use modular additions instead of xors, with the cipher

$$\mathrm{FX}+_{k_0,k_1,k_2}(x) = E_{k_0}(x + k_1) + k_2.$$

The function is no longer periodic in this situation, but we can find a hidden shift problem with the two functions $f(k, x) = \mathrm{FX}+_{k_0,k_1,k_2}(x)+E_k(-x)$ and $g(k, x) = \mathrm{FX}+_{k_0,k_1,k_2}(-x)+E_k(x)$, which fulfils the promise $f(k_0, x) = g(k_0, x+k_1)$. These two function can efficiently be computed in superposition, for a total cost of one query and one encryption. The attack consists then in a Grover search that uses Kuperberg's algorithm as a test function. The Grover search needs the same number of iterations ($2^{|k_0|/2}$), but Kuperberg's algorithm needs around $2^{1.8\sqrt{|k_1|}}$ samples. The total cost is around $2^{|k_0|/2+1.8\sqrt{|k_1|}} \times 2$ queries (we can factor the query to f and g to only one query to FX+, and we double to uncompute Kuperberg's algorithm).

Other group laws. If the group is abelian, the attack can be straightforwardly applied. If the group law is not abelian, we need a slightly different approach. With $FX \cdot_{k_0,k_1,k_2}(x) = E_{k_0}(x \cdot k_1) \cdot k_2$ and a a fixed non-zero value, we can consider $f(x) = FX \cdot_{k_0,k_1,k_2}(x) \cdot (FX \cdot_{k_0,k_1,k_2}(ax))^{-1}$ and $g(k, x) = E_k(x) \cdot$

[6] In [35], they considered the time of a parallelized Simon's algorithm, which can be neglected, leaving a complexity of 2^{32}.

$E_k(ax)$, which satisfies $f(x) = g(k_0, x \cdot k_1)$. We can then have the same approach, using the corresponding non-abelian hidden shift algorithm in a Grover search. The concrete cost of the attack depends of the group structure, and can be estimated from the values of Table 2.

Quantum attack on PRINCE+ and PRIDE+. We can directly attack a variant of PRINCE and PRIDE where the key whitening is done through a modular addition. Concretely, we can attack them in around $2^{47.4}$ queries and time, which is smaller than the ideal 64-bits of quantum security. We also attack DESX+ in $2^{43.4}$ queries and time.

7 Concrete Proposals

The most interesting idea from [2] for preventing Simon-based attacks is using modular additions, which is already common in symmetric primitives (see for instance [38,47,43]). Based on the complexities of the new algorithms and attacks from the previous sections, we can now correctly size some of the primitives that were broken using Simon-based algorithms, now patched to use modular additions, in order to provide a certain desired post-quantum security.

Let us point out that we used a slightly unconventional definition of the *security*: we consider a cipher to provide a security of Q bits when no attack of complexity lower than 2^Q exists (the more conventional definition being when no attack better than the generic exhaustive search is known, whose complexity usually is $2^Q = 2^{k/2}$).

7.1 Concrete Parameters and Security of Some Generic Constructions

If we consider the generic Even-Mansour constructions, with a xor, it will provide a security of 8 or 9 bits for an state size of 128 or 256 bits respectively. When using one or several modular additions, this security is augmented, becoming 20 or 28.5 bits for states of 128 or 256 bits respectively, but all the constructions are far from the ideal 2^{64} security offered by an ideal cipher with a 128-bit key, and even more from the 2^{128} offered quantumly by a classical primitive with a 256-bit key. To the best of our knowledge, the quantum security offered by the Advanced Encryption Standard (AES) [18] meets these ideal claims.

In Table 7 we show the needed security parameters of some popular constructions in order to resist their corresponding attacks when using Kuperberg's algorithm. As expected, p modular additions of words of size w provide less security than one modular addition of the state size. We can see that, in all the cases, the size of the state needed to achieve a certain security becomes much bigger than for common symmetric primitives (128 bits for instance), needing to be bigger than 5200 bits in some cases. The problem of a bigger state is not limited to implementation issues: designing a secure permutation for such a big state would be a very challenging task. From table 2 we can also estimate the needed size for some constructions that uses non-abelian groups.

(p/w)	E-M$(1/n)$	O-M/LRW$(1/n)$	E-M(2048/13)	E-M(1024/14)	E-M(4/1304)
State	5168	5168	26624	14336	5216
Key	5168	$k \geq 256$	26624	14336	5216

Table 7: Examples of parameters for 128-bit security when using modular additions instead of $\oplus$. E-M stands for Even-Mansour and O-M for operation modes.

8 Conclusion

Modular additions are not enough. We have shown that the proposal from [2] does not seem practical. Using modular additions instead of xors would increase the security, but it would need a large internal state to provide reasonable security, far beyond the size of classical symmetric constructions. For instance, the key-alternating cipher with modular addition would need a few thousand bits of internal state and key size, to be compared with the 128 bits of internal state and 256 key-bits of AES-256. Beyond the obvious efficiency drawback, the design of a correspondingly large secure permutation would be a very challenging task.

Kuperberg's algorithm simulation and verification. We have been able to study, improve and simulate Kuperberg's algorithm: the concrete complexity of our tweaked version is $2^{1.8\sqrt{n}}$, which is small enough for a practical use on typical parameters of n (we have therefore implemented it). We also have presented a way to solve the hidden shift problem in various situations (including non-abelian groups), and provided an estimate of the complexity.

New algorithm representing Simon-meets-Kuperberg. We provided a new efficient algorithm that solves the problem when considering parallel modular additions. We have simulated the algorithm and verified that our estimated complexity is met in practice.

Cryptanalysis of FX variants and Poly1305. This paper proposes some new quantum attacks, mainly using our generalized and improved Kuperberg's algorithm, that provide an important speed-up with respect to Grover's quantum generic exhaustive search attack.

Further applications. Hidden shift algorithms can be applied in other cryptographic fields. They have in particular been successfully applied to ordinary isogenies [17] and are relevant to assess the security of some proposed post-quantum asymetric schemes, such as CSIDH [15]. We leave as an open problem the evaluation of the security offered by this construction against quantum hidden shift algorithms, which was not addressed in the CSIDH paper.

9 Acknowledgements

The authors would like to thank André Chailloux, Anthony Leverrier and André Schrottenloher for their helpful comments and discussions.

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement no. 714294 - acronym QUASYModo).

References

1. See https://tools.ietf.org/html/draft-ietf-tls-tls13-23#section-9.1
2. Alagic, G., Russell, A.: Quantum-Secure Symmetric-Key Cryptography Based on Hidden Shifts. In: Coron, J.S., Nielsen, J.B. (eds.) EUROCRYPT (3). LNCS, vol. 10212, pp. 65–93 (2017)
3. Albrecht, M.R., Driessen, B., Kavun, E.B., Leander, G., Paar, C., Yalçin, T.: Block ciphers - focus on the linear layer (feat. PRIDE). In: CRYPTO 2014. LNCS, vol. 8616, pp. 57–76. Springer (2014)
4. Anand, M.V., Targhi, E.E., Tabia, G.N., Unruh, D.: Post-Quantum Security of the CBC, CFB, OFB, CTR, and XTS Modes of Operation. In: Takagi, T. (ed.) Post-Quantum Cryptography - 7th International Workshop, PQCrypto 2016, Fukuoka, Japan, February 24-26, 2016, Proceedings. Lecture Notes in Computer Science, vol. 9606, pp. 44–63. Springer (2016)
5. Berger, T.P., Francq, J., Minier, M., Thomas, G.: Extended generalized feistel networks using matrix representation to propose a new lightweight block cipher: Lilliput. IEEE Trans. Computers 65(7), 2074–2089 (2016)
6. Bernstein, D.J.: The Poly1305-AES Message-Authentication Code. In: FSE. LNCS, vol. 3557, pp. 32–49. Springer (2005)
7. Bernstein, D.J.: The salsa20 family of stream ciphers. In: New Stream Cipher Designs - The eSTREAM Finalists, Lecture Notes in Computer Science, vol. 4986, pp. 84–97. Springer (2008)
8. Bernstein Daniel J., Lange Tanja: Post-quantum cryptography. Nature 549(7671), 188–194 (sep 2017)
9. Boneh, D., Dagdelen, Ö., Fischlin, M., Lehmann, A., Schaffner, C., Zhandry, M.: Random Oracles in a Quantum World. In: Lee, D., Wang, X. (eds.) Advances in Cryptology – ASIACRYPT 2011, LNCS, vol. 7073, pp. 41–69. Springer Berlin Heidelberg (2011)
10. Boneh, D., Zhandry, M.: Secure Signatures and Chosen Ciphertext Security in a Quantum Computing World. In: Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II. pp. 361–379 (2013)
11. Bonnetain, X.: Quantum key-recovery on full AEZ. In: Selected Areas in Cryptography - SAC 2017 - 24th International Conference, Ottawa, ON, Canada, August 16-18, 2017, Revised Selected Papers. Lecture Notes in Computer Science, vol. 10719, pp. 394–406. Springer (2018)
12. Borghoff, J., Canteaut, A., Güneysu, T., Kavun, E.B., Knezevic, M., Knudsen, L.R., Leander, G., Nikov, V., Paar, C., Rechberger, C., Rombouts, P., Thomsen, S.S., Yalçin, T.: PRINCE - A low-latency block cipher for pervasive computing applications - extended abstract. In: Asiacrypt 2012. vol. 7658, pp. 208–225. Springer (2012)
13. Brassard, G., Høyer, P., Kalach, K., Kaplan, M., Laplante, S., Salvail, L.: Merkle puzzles in a quantum world. In: Advances in Cryptology–CRYPTO 2011, pp. 391–410. Springer (2011)
14. Brassard, G., Høyer, P., Tapp, A.: Quantum Algorithm for the Collision Problem. Springer New York, New York, NY (2016)
15. Castryck, W., Lange, T., Martindale, C., Panny, L., Renes, J.: Csidh: An efficient post-quantum commutative group action. Cryptology ePrint Archive, Report 2018/383 (2018), https://eprint.iacr.org/2018/383

16. Chailloux, A., Naya-Plasencia, M., Schrottenloher, A.: An efficient quantum collision search algorithm and implications on symmetric cryptography 10625, 211–240 (2017)
17. Childs, A.M., Jao, D., Soukharev, V.: Constructing elliptic curve isogenies in quantum subexponential time. J. Mathematical Cryptology 8(1), 1–29 (2014)
18. Daemen, J., Rijmen, V.: The Design of Rijndael: AES - The Advanced Encryption Standard. Information Security and Cryptography, Springer (2002)
19. Damgård, I., Funder, J., Nielsen, J.B., Salvail, L.: Superposition Attacks on Cryptographic Protocols. In: Padró, C. (ed.) Information Theoretic Security - 7th International Conference, ICITS 2013, Singapore, November 28-30, 2013, Proceedings. LNCS, vol. 8317, pp. 142–161. Springer (2013)
20. Ettinger, M., Høyer, P.: On Quantum Algorithms for Noncommutative Hidden Subgroups. In: STACS 99, 16th Annual Symposium on Theoretical Aspects of Computer Science, Trier, Germany, March 4-6, 1999, Proceedings. LNCS, vol. 1563, pp. 478–487. Springer (1999)
21. Ferguson, N., Lucks, S., Schneier, B., Whiting, D., Bellare, M., Kohno, T., Callas, J., Walker, J.: The skein hask function family (2010)
22. Flajolet, P., Odlyzko, A.M.: Random Mapping Statistics. In: Quisquater, J., Vandewalle, J. (eds.) Advances in Cryptology - EUROCRYPT '89, Workshop on the Theory and Application of of Cryptographic Techniques, Houthalen, Belgium, April 10-13, 1989, Proceedings. LNCS, vol. 434, pp. 329–354. Springer (1989)
23. Gagliardoni, T.: Quantum Security of Cryptographic Primitives. Ph.D. thesis, Darmstadt University of Technology, Germany (2017)
24. Gagliardoni, T., Hülsing, A., Schaffner, C.: Semantic Security and Indistinguishability in the Quantum World. In: Robshaw, M., Katz, J. (eds.) Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part III. LNCS, vol. 9816, pp. 60–89. Springer (2016)
25. Hosoyamada, A., Sasaki, Y., Xagawa, K.: Quantum multicollision-finding algorithm. In: Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part II. Lecture Notes in Computer Science, vol. 10625, pp. 179–210. Springer (2017)
26. Kaplan, M.: Quantum attacks against iterated block ciphers. CoRR abs/1410.1434 (2014)
27. Kaplan, M., Leurent, G., Leverrier, A., Naya-Plasencia, M.: Breaking Symmetric Cryptosystems Using Quantum Period Finding. In: Robshaw, M., Katz, J. (eds.) Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II. LNCS, vol. 9815, pp. 207–237. Springer (2016)
28. Kaplan, M., Leurent, G., Leverrier, A., Naya-Plasencia, M.: Quantum Differential and Linear Cryptanalysis. IACR Trans. Symmetric Cryptol. 2016(1), 71–94 (2016)
29. Kilian, J., Rogaway, P.: How to Protect DES Against Exhaustive Key Search. In: Koblitz, N. (ed.) CRYPTO. LNCS, vol. 1109, pp. 252–267. Springer (1996)
30. Kuperberg, G.: A Subexponential-Time Quantum Algorithm for the Dihedral Hidden Subgroup Problem. SIAM J. Comput. 35(1), 170–188 (2005)
31. Kuperberg, G.: Another Subexponential-time Quantum Algorithm for the Dihedral Hidden Subgroup Problem. In: Severini, S., Brandão, F.G.S.L. (eds.) 8th Conference on the Theory of Quantum Computation, Communication and Cryptography, TQC 2013, May 21-23, 2013, Guelph, Canada. LIPIcs, vol. 22, pp. 20–34. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2013)

32. Kuwakado, H., Morii, M.: Quantum distinguisher between the 3-round Feistel cipher and the random permutation. In: Information Theory Proceedings (ISIT), 2010 IEEE International Symposium on. pp. 2682–2685 (June 2010)
33. Kuwakado, H., Morii, M.: Security on the quantum-type Even-Mansour cipher. In: Information Theory and its Applications (ISITA), 2012 International Symposium on. pp. 312–316 (Oct 2012)
34. Langley, A., Chang, W., Mavrogiannopoulos, N., Strombergson, J., Josefsson, S.: "chacha20-poly1305 cipher suites for transport layer security (tls)". In: RFC 7905, DOI 10.17487/RFC7905 (June 2016)
35. Leander, G., May, A.: Grover Meets Simon - Quantumly Attacking the FX-construction. In: Takagi, T., Peyrin, T. (eds.) Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part II. Lecture Notes in Computer Science, vol. 10625, pp. 161–178. Springer (2017)
36. Lydersen, L., Wiechers, C., Wittmann, C., Elser, D., Skaar, J., Makarov, V.: Hacking commercial quantum cryptography systems by tailored bright illumination. Nature photonics 4(10), 686–689 (2010)
37. Regev, O.: A Subexponential Time Algorithm for the Dihedral Hidden Subgroup Problem with Polynomial Space. CoRR (2004)
38. Rivest, R.L., Robshaw, M.J.B., Yin, Y.L.: RC6 as the AES. In: AES Candidate Conference. pp. 337–342 (2000)
39. Roetteler, M., Steinwandt, R.: A note on quantum related-key attacks. Information Processing Letters 115(1), 40–44 (2015)
40. Santoli, T., Schaffner, C.: Using Simon's Algorithm to Attack Symmetric-Key Cryptographic Primitives. arXiv preprint arXiv:1603.07856 (2016)
41. Simon, D.R.: On the Power of Quantum Cryptography. In: 35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994. pp. 116–123. IEEE Computer Society (1994)
42. Song, F., Yun, A.: Quantum security of NMAC and related constructions - PRF domain extension against quantum attacks. In: Katz, J., Shacham, H. (eds.) Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part II. Lecture Notes in Computer Science, vol. 10402, pp. 283–309. Springer (2017)
43. of Standards, N.B.: GOST 28147-89. In: Federal Information Processing Standard-Cryptographic Protection - Cryptographic Algorithm (1989)
44. Suzaki, T., Minematsu, K., Morioka, S., Kobayashi, E.: $\textnormal{\textsc{TWINE}}$: A lightweight block cipher for multiple platforms. In: Selected Areas in Cryptography, 19th International Conference, SAC 2012, Windsor, ON, Canada, August 15-16, 2012, Revised Selected Papers. Lecture Notes in Computer Science, vol. 7707, pp. 339–354. Springer (2012)
45. Unruh, D.: Non-interactive zero-knowledge proofs in the quantum random oracle model. In: Eurocrypt 2015. vol. 9057, pp. 755–784. Springer (2015), preprint on IACR ePrint 2014/587
46. Xu, F., Qi, B., Lo, H.K.: Experimental demonstration of phase-remapping attack in a practical quantum key distribution system. New Journal of Physics 12(11), 113026 (2010)
47. Yuval, G.: Treyfer (1997)
48. Zhandry, M.: How to Construct Quantum Random Functions. In: 53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012. pp. 679–687 (2012)

49. Zhandry, M.: Secure identity-based encryption in the quantum random oracle model. International Journal of Quantum Information 13(04), 1550014 (2015)
50. Zhao, Y., Fung, C.H.F., Qi, B., Chen, C., Lo, H.K.: Quantum hacking: Experimental demonstration of time-shift attack against practical quantum-key-distribution systems. Physical Review A 78(4), 042333 (2008)

Appendix

A Kuperberg and SMK simulation code

Available at `https://who.paris.inria.fr/Xavier.Bonnetain/extra/code.tar.gz`.

B A distinguisher attack on Poly1305

We can use a simple distinguisher, that allows to discriminate between a $\sum_x |x\rangle |A(s)\rangle$ and $\sum_x |x\rangle |A(x,s)\rangle$. If one apply a Hadamard gate to the first register, the first state would become $|0\rangle |A\rangle$, while the second become $\sum_{x,y}(-1)^{x\cdot y} |y\rangle |A(x)\rangle$. If $x \mapsto A(x,s)$ is a permutation, then there will be no interference, and the probability of measuring a 0 in the first register is $1/2^n$ in the second case, whereas it is 1 in the first case.

We can apply this to Poly1305, with

$$A(x,s) = \text{Poly1305}(x) - \left((x+2^{128})s \mod 2^{130}-5\right) \mod 2^{128},$$

where Poly1305(x) is the tag of the one-block message x, with an unspecified nonce. If $s = r$, then $A(x,r) = \text{AES}_k(n)$ (which is a fixed value for each query), and if not, it will be $A(x,s) = \left((x+2^{128})r \mod 2^{130}-5\right) - \left((x+2^{128})s \mod 2^{130}-5\right) + \text{AES}_k(n) \mod 2^{128}$. As the function $x \mapsto xs$ in $\mathbb{Z}/(2^{130}-5)$ is, with an overwhelming probability, a permutation, we will measure a 0 with a wrong guess with a probability of around 2^{-126}.

This distinguisher can then be used in a Grover search on r that calls Poly1305 : at each step, we compute $\sum_x |x\rangle A(x,r)$, apply a Hadamard gate on the first register, and mark r if the first register is 0. As the test function is not perfect, the success probability of the algorithm will be smaller than 1, but the error of the test function is small enough to have a negligible final error.

There is still one problem : to perform a Grover search, we need to uncompute our computations. And as Poly1305 generates a fresh nonce at each query, we cannot uncompute the function. This is however not a problem, as a nonce difference in the uncomputation will produce a fixed difference in the output $(\text{AES}_k(n) - \text{AES}_k(n'))$. This means that the uncomputation would leave the register in a non-zero but non-entangled state, which allows to safely erase and reset it.

We can hence apply a Grover search on r. As it has 106 variable bits, it would cost around 2^{53} queries and time to retrieve r.