Quantum Algorithms for the k-xor Problem

Lorenzo Grassi[1(✉)], María Naya-Plasencia[2], and André Schrottenloher[2]

[1] IAIK, Graz University of Technology, Graz, Austria
lorenzo.grassi@iaik.tugraz.at
[2] Inria, Paris, France
{maria.naya_plasencia,andre.schrottenloher}@inria.fr

Abstract. The k-xor (or generalized birthday) problem is a widely studied question with many applications in cryptography. It aims at finding k elements of n bits, drawn at random, such that the xor of all of them is 0. The algorithms proposed by Wagner more than fifteen years ago remain the best known classical algorithms for solving them, when disregarding logarithmic factors.

In this paper we study these problems in the quantum setting, when considering that the elements are created by querying a random function (or k random functions) $H : \{0,1\}^n \to \{0,1\}^n$. We consider two scenarios: in one we are able to use a limited amount of quantum memory (i.e. a number $O(n)$ of qubits, the same as the one needed by Grover's search algorithm), and in the other we consider that the algorithm can use an exponential amount of qubits. Our newly proposed algorithms are of general interest. In both settings, they provide the best known quantum time complexities.

In particular, we are able to considerately improve the 3-xor algorithm: with limited qubits, we reach a complexity considerably better than what is currently possible for quantum collision search. Furthermore, when having access to exponential amounts of quantum memory, we can take this complexity below $O(2^{n/3})$, the well-known lower bound of quantum collision search, clearly improving the best known quantum time complexity also in this setting.

We illustrate the importance of these results with some cryptographic applications.

Keywords: Quantum algorithms · Generalized birthday problem
Quantum cryptanalysis · 3-xor · k-xor · List-merging algorithms
Amplitude amplification

1 Introduction

In this paper we consider a generic algorithmic problem with numerous applications in cryptography: the k-xor problem. We study it when considering elements generated by a random function (or k random functions) $H : \{0,1\}^n \to \{0,1\}^n$,

T. Peyrin and S. Galbraith (Eds.): ASIACRYPT 2018, LNCS 11272, pp. 527–559, 2018.
https://doi.org/10.1007/978-3-030-03326-2_18

and we provide the best known quantum algorithms for solving it, taking into account two possible scenarios regarding quantum memory.

In this section we first introduce the studied problem and provide some examples of applications. Second, we recall the scenario of post-quantum cryptography; and finally, we summarize our contributions, that propose the best known quantum time complexities and, in most of the cases, give considerable quantum speedups over the best classical algorithms.

1.1 Generalized Birthday Problem

The *birthday problem* is a widely used cryptanalytical tool.

Birthday Problem. *Given two lists L_1, L_2 of elements drawn at random from $\{0,1\}^n$, find $x_1 \in L_1$ and $x_2 \in L_2$ such that $x_1 \oplus x_2 = 0$ (where $\oplus$ denotes the bitwise exclusive-or, below xor, operation).*

A solution of this problem exists with high probability once $|L_1| \times |L_2| \gg 2^n$ holds, and it can be found in $\mathcal{O}(2^{n/2})$ time by e.g. sorting and then scanning L_1 and L_2.

The birthday problem has many applications, the most used one being perhaps the research of a collision for a hash function $h(\cdot) : \{0,1\}^\star \to \{0,1\}^n$. The application to this case is simple. First of all, one constructs the list L_i by defining the $j-$th element of L_i as $h(i|j)$ (where $i|j$ denotes i concatenated with j). Assuming that h behaves like a random function, the lists contain values distributed uniformly and independently at random, so the premises of the problem statement will be met. Consequently, one may expect to find a solution to the corresponding problem, and a collision for the hash function, with $O(2^{n/2})$ work.

A generalization of this problem – called *generalized birthday problem* (GBP) or *k-list problem* – has been introduced by Wagner [54].

Generalized Birthday Problem. *Given k lists $L_1, L_2, \ldots, L_k$ of elements drawn at random from $\{0,1\}^n$, find $x_1 \in L_1$, $x_2 \in L_2$, $\ldots x_k \in L_k$ such that $x_1 \oplus x_2 \oplus \ldots \oplus x_k = \bigoplus_{i=1}^{k} x_i = 0$.*

Obviously, if $|L_1| \times |L_2| \times \ldots \times |L_k| \geq 2^n$, then with a high probability the solution exists. The real challenge, however, is to find it efficiently. When $k = 2^t$, Wagner's algorithm requires classical time and space $\mathcal{O}(2^{n/(t+1)})$.

Applications. Even if the GBP may not appear very natural at first sight, it has been applied successfully to the cryptanalysis of various systems. In the following, we recall the most relevant applications for symmetric cryptography.

XHASH and the (R)FSB SHA-3 Candidate. XHASH [8] has been introduced as a plausible candidate for an incremental collision-free hash function, defined as

$$H(x) := \bigoplus_{i=1}^{k} h(i|x_i),$$

where each x_i is a b-bit block and $h(\cdot) : \{0,1\}^l \rightarrow \{0,1\}^n$. The size $l = b + \log_2(k)$ is chosen to be large enough to accommodate the block plus an encoding of its index, by dint of making k larger than the number of blocks in any message to be hashed. As showed e.g. in [8,22], it is possible to set up an attack based on GBP that easily finds collisions in XHASH.

Among other designs, this construction appears in the "fast syndrome-based" hash function (R)FSB [4], a candidate of the SHA-3 competition. It uses a compression function in a Merkle-Damgård construction and it, is based on xoring the columns of a random binary matrix and has the advantages to be fast, incremental and parallelizable. In particular, this candidate can be rewritten as

$$FSB(H, m) := \bigoplus_{i=1}^{k} h_i(m_i).$$

As showed in [12,13,22,35,47], the previous GBP attack applies as well also in this case.

AdHash, NASD Incremental Hashing and the SWIFFT SHA-3 Candidate. One proposal for network-attached secure disks (NASD) [27] uses the following hash function for integrity purposes [8]:

$$H(x) := \sum_{i=1}^{k} h(i|x_i) \mod 2^{256},$$

where $x = \langle x_1, ..., x_k \rangle$ denotes a padded k-block message. By simple observation, inverting this hash corresponds to a k-sum problem over the additive group $(\mathbb{Z}/2^{256}\mathbb{Z}, +)$.

This may be viewed as a special case of a general incremental hashing construction proposed by Bellare *et al.* [8], where the sum is computed modulo m and where the modulus m is public and chosen randomly.

Among other designs, such a construction has been exploited in the SWIFFT hash function [42], one candidate of the SHA-3 competition. SWIFFT is a collection of provably secure hash functions, based on the fast Fourier transform (FFT). The SWIFFT function can be described as a simple algebraic expression over some polynomial ring $R = \mathbb{Z}_p[\alpha]/(\alpha^n + 1)$, that is

$$SWIFFT(a, x) = \sum_{i=1}^{m} f(x_i) \mod (\alpha^n + 1) = \sum_{i=1}^{m} (a_i \cdot x_i) \mod (\alpha^n + 1)$$

where the m fixed elements $a_1, ..., a_m \in R$ – called multipliers – specify the hash function, and each x_i is an element of R. Examples of attacks on the SWIFFT hash function based on the k-sum problem over the additive group $(\mathbb{Z}/2^{256}\mathbb{Z}, +)$ are given in [5,35,47].

The PCIHF Hash. Another hash construction that can be attacked using a similar strategy is the PCIHF hash function [28], proposed for incremental hashing and defined as

$$H(x) := \sum_{i=1}^{k} SHA(x_i|x_{i+1}) \mod 2^{160}+1.$$

With respect to the previous case, the main difference is that each x_i affects two terms. To overcome this problem and apply an attack based on the GBP, it is sufficient to choose (and fix) $x_{2j} = 0$ for each j. In this case, the hash computations takes the form

$$H(x) := \sum_{j=1}^{\lfloor (k+1)/2 \rfloor} h(x_{2j-1}) \mod 2^{160}+1 \text{ where } h(x) = SHA(x|0)+SHA(0|x).$$

CAESAR Candidates and the 3-xor Problem. The GBP has been as well applied to the cryptanalysis of authenticated encryption schemes proposed at the ongoing CAESAR competition [18]. To process the final incomplete blocks of messages, some of these schemes use the XLS construction proposed by Ristenpart and Rogaway [49].

Even if XLS was initially proven secure, Nandi [44] pointed out flaws in the security proof and showed a very simple attack that requires three queries to break the construction. Actually, the CAESAR candidates that rely on XLS do not allow this trivial attack as the required decryption queries are not permitted by the schemes. A possible way to overcome this limitation has been proposed by Nandi in [44], whose forgery attack requires only encryption queries. As a result, it is possible the design flaw of XLS can be reduced to the 3-xor problem.

The CAESAR schemes based on XLS are – the COPA modes of – the finalist Deoxys [23], Joltik [32], KIASU [33] and SHELL [55]. As a result, any 3-xor algorithm that goes below the birthday bound results in a slight weakness of some of these candidates. We refer to [45,48] for concrete examples of attacks.

Fast Correlation Attacks. Finally, the k-xor problem (especially for $k \geq 4$) is interesting for searching parity check relations in *fast correlation attacks* [21,51,52], whose main targets are *synchronous stream ciphers*.

A synchronous stream cipher is a stream cipher where the ciphertext is produced by bitwise adding the plaintext bits to a stream of bits called the keystream, which is independent of the plaintext, only produced from the secret key and the initialization vector. A large number of stream ciphers use Linear Feedback Shift Registers (LFSR) as building blocks, the initial state of these LFSRs being related to the secret key and to the initialization vector. In nonlinear combination generators, the keystream bits are then produced by combining the outputs of these LFSRs through a nonlinear boolean function. Examples – among many others – of stream ciphers based on the previous construction are the hardware oriented finalists of the eSTREAM project [24], e.g. Grain-v0 [31].

A fast correlation attack targets nonlinear combination keystream generators. In particular, it requires the existence of linear correlations between LFSR internal stages and the nonlinear function output. GBP can be used to find such correlations – which is the hardest part of the job.

Remark. Though the GBP could be defined with many operations other than XOR, like modular additions (the k-sum problem then), and the algorithms proposed by Wagner would still apply, *in this paper we concentrate for the sake of simplicity, on solving the* k-xor*problem*, i.e. the case of having a XOR operation. In general, our algorithms can be easily adapted to other settings.

1.2 Cryptography in the Quantum World

Post-quantum cryptography (or quantum-resistant cryptography) is a whole new line of research that aims at developing new cryptographic primitives that would (hopefully) withstand attackers equipped with quantum computers. It is now a well-known fact that the existence of sufficiently large quantum computers would severely impact the security of many cryptographic schemes in use today. In particular, the seminal work of Shor [50] showed that such computers would allow to factor numbers and compute discrete logarithms in abelian groups in polynomial time. As almost all public key schemes currently in use are build upon the assumption that those problems are intractable, the advent of quantum computers has motivated the rise of quantum-resistant public-key cryptography.

Post-quantum Symmetric Cryptography. At first sight, the situation seems less critical for symmetric primitives: Grover's algorithm [54] for searching in an unstructured database finds a marked element among 2^n in time $O(2^{n/2})$, providing a quadratic speedup compared to the classical exhaustive search, essentially optimal. Hence doubling the key length of block ciphers seems sufficient to counter that attack, and achieve the same security against quantum attackers.

However, recent works have shown that Grover's algorithm might *not* be the only threat for symmetric cryptography. One of the most relevant works is the one by Kuwakado and Morii [36,37], who first showed that the Even-Mansour construction [25] could be broken in polynomial time in the quantum CPA setting. Briefly, the Even-Mansour construction consists of a public permutation P on n bits and of two secret keys k_1 and k_2 that are used as pre- (resp. post-) whitening keys for the encryption $Enc_{EM}(m) := k_2 \oplus P(m \oplus k_2)$ of some message m. The main idea of [36,37] was to consider the function

$$f(x) := Enc_{EM}(x) \oplus P(x) = P(x \oplus k_1) \oplus k_2 \oplus P(x).$$

Since such a function has period k_1, it is possible to exploit Simon's quantum algorithm [15,53] to compute the (unknown) period in polynomial time.

Many other works have since appeared in the literature – such as attacks on symmetric cryptosystems based on quantum period finding [34], a quantum attack of the FX-construction [38],... – showing that the post-quantum security

of some symmetric primitives, depending on the quantum adversary model, could fall largely below the limit provided by the Grover's algorithm.

As we are trying to build quantum-safe primitives, understanding and improving quantum algorithms, as well as designing new quantum attacks is of main importance: only this way can we know what are the needs in order to resist to the mentioned attacks.

1.3 Our Contributions

How can we solve the k-xor *problem in the quantum setting?* We answer this question by proposing new quantum algorithms. We consider two different settings, of separate interest: *(1st)* the case in which the adversary has access to a big amount of quantum memory and *(2nd)* the case in which she has access to small quantum memory, say $O(n)$. How one should treat classical vs. quantum memory is an open problem (e.g. can quantum memory become as cheap as classical memory?) that we do not attempt to fix here. Instead, we consider separately the two cases and take both *classical* and *quantum memory* into account in the cost of our algorithms.

About the 2-xor problem, Brassard *et al.* [17] provide a quantum algorithm that requires $O(2^{n/3})$ time and $O(2^{n/3})$. When "only" $O(n)$ qubits of memory are allowed, Grover's algorithm provides a solution in time $O(2^{n/2})$. Chailloux *et al.* [19] showed that the problem can be solved in quantum time $O(2^{2n/5})$ and using $O(2^{n/5})$ classical memory.

While the quantum *query* complexity of the k-xor problem is well-known, and can be attained by a modification of the algorithm in [3], there has been - to the best of our knowledge - no previous attempt at systematic time-efficient quantum algorithms (apart from the 2-xor case above).

Parallelized Algorithms. While the k-xor algorithms using $O(n)$ quantum memory that we develop are first intended to be used by "small" quantum computers, we further remark that they can be efficiently parallelized. Even with the most restrictive (and debatable) benchmark on "total cost" (which counts together the number of processors and the memory consumption, and multiplies this "hardware cost" by the time complexity), we show that our parallelized 3-xor algorithm reaches below the classical product $O(2^{n/2})$. We conclude that it attains a range of effectiveness unreached by all collision search algorithm previously known.

Our Results. In this paper, we present the *first* analysis of the k-xor problem in the quantum world for generic $k \geq 3$ *with competitive quantum time* with respect to both algorithms in the classical and in the quantum setting present in the literature. Our results – compared to others in the literature – are provided in Table 1.

Linear – Quantum Memory. For the case in which the adversary can use only $O(n)$ quantum memory, we propose solutions with better time complexity than

Table 1. Complexity of k-xor quantum algorithms (without logarithmic factors). Our results are in bold. When referring to Ambainis' work [3], we hint at our own quantum time complexity analysis from Sect. 3.

k (collision)	Quantum time	Superposition queries	Quantum memory	Classical memory	Reference
2	$2^{n/2}$	$2^{n/2}$	$O(n)$	-	[29]
2	$2^{n/3}$	$2^{n/3}$	$2^{n/3}$	-	[17]
2	$2^{2n/5}$	$2^{2n/5}$	$O(n)$	$2^{n/5}$	[19]
3	$\mathbf{2^{5n/14}}$	$\mathbf{2^{5n/14}}$	$\mathbf{O(n)}$	$\mathbf{2^{n/7}}$	**Theorem 1**
3	$2^{n/2}$	$2^{n/4}$	$2^{n/4}$	-	[3]
3	$\mathbf{2^{3n/10}}$	$\mathbf{2^{3n/10}}$	$\mathbf{2^{n/5}}$	**-**	**Theorem 2**
4	$\mathbf{2^{n/3}}$	$\mathbf{2^{n/3}}$	$\mathbf{O(n)}$	$\mathbf{2^{n/9}}$	**Theorem 1**
4	$2^{n/2}$	$2^{n/5}$	$2^{3n/10}$	-	[3]
4	$\mathbf{2^{n/4}}$	$\mathbf{2^{n/4}}$	$\mathbf{2^{n/4}}$	**-**	**Theorem 4**
5	$\mathbf{2^{7n/22}}$	$\mathbf{2^{7n/22}}$	$\mathbf{O(n)}$	$\mathbf{2^{n/11}}$	**Theorem 1**
5	$2^{n/2}$	$2^{n/6}$	$2^{1/3}$	-	[3]
5	$\mathbf{2^{n/4}}$	$\mathbf{2^{n/4}}$	$\mathbf{2^{n/4}}$	**-**	**Theorem 4**
6	$\mathbf{2^{4n/13}}$	$\mathbf{2^{4n/13}}$	$\mathbf{O(n)}$	$\mathbf{2^{n/13}}$	**Theorem 1**
6	$2^{n/2}$	$2^{n/7}$	$2^{5n/14}$	-	[3]
6	$\mathbf{2^{n/4}}$	$\mathbf{2^{n/4}}$	$\mathbf{2^{n/4}}$	**-**	**Theorem 4**
7	$\mathbf{2^{3n/10}}$	$\mathbf{2^{3n/10}}$	$\mathbf{O(n)}$	$\mathbf{2^{n/15}}$	**Theorem 1**
7	$2^{n/2}$	$2^{n/8}$	$2^{3n/8}$	-	[3]
7	$\mathbf{2^{n/4}}$	$\mathbf{2^{n/4}}$	$\mathbf{2^{n/4}}$	**-**	**Theorem 4**
$k \geq 8$	$2^{n/2}$	$2^{n/(k+1)}$	$2^{\frac{n(k+1)}{2(k+1)}}$	-	[3]
$\mathbf{k \geq 8}$	$\mathbf{2^{n/(2+\lfloor \log_2(k) \rfloor)}}$	$\mathbf{2^{n/(2+\lfloor \log_2(k) \rfloor)}}$	$\mathbf{2^{n/(2+\lfloor \log_2(k) \rfloor)}}$	**-**	**Theorem 4**

classical algorithms up to $k < 8$. We use building blocks from [19] (initially used for collision search) and ideas from [46], inspired for instance from the *parallel matching techniques*.

Exponential – Quantum Memory. When the adversary might use big amounts quantum memory, we propose a strategy that improves k-xor problems for all $k \geq 3$. For $k \geq 4$, we use the well-known quantum walk framework. Our attack requires time $O(2^{n/(2+\lfloor \log_2(k) \rfloor)})$ and memory $O(2^{n/(2+\lfloor \log_2(k) \rfloor)})$, giving an exponential quantum speedup over Wagner's algorithm. For the 3-xor problem, we specially design an algorithm with time $O(2^{3n/10})$ and $O(2^{n/5})$ quantum memory.

We highlight that, in the two cases above, the 3-xor algorithm has an exponential acceleration over collision search, which was not the case classically.

Organization. In the next section, we detail some basic notions of quantum computing and building blocks for our new algorithms. In Sect. 3, we recall the algorithms present in the literature to solve the k-xor problem both in the classical and in the quantum setting. New quantum algorithms for the 3-xor problem – both for the linear and the exponential quantum memory – are proposed in Sect. 4, while in Sect. 5 we describe algorithms for the k-xor problem for $k \geq 4$. We emphasize again that our goal is to set up algorithms with optimal time

and memory complexities (rather than query complexity). We give insights on *parallelization* in Sect. 6. We conclude in Sect. 7 with implications of our results and some open problems for future research.

2 Preliminaries

In this section, we recall some definitions and simple quantum algorithmic techniques that will be used throughout the paper. We stress that most of our algorithms, and the design principles thereof, can be understood with only some basic notions of quantum computing, which we provide below.

2.1 Quantum Algorithms

For a comprehensive introduction into quantum algorithms, we suggest the textbooks of Mermin [41] and Lipton, Regan [39].

Quantum Circuit Model. We only work in the standard quantum circuit model. A quantum circuit is an abstract representation of a quantum algorithm running on a universal quantum computer. Given a number of qubits, put in an arbitrary initial state (say $|0\rangle$), we apply a succession of quantum gates, analog to classical boolean gates. After, the state of the qubits is measured. The final measurement should contain the result of the algorithm. The quantum computing literature also often considers that a quantum algorithm can run in a number of successive steps, which we will do below. The sequence of gates of a step can depend on the results of the previous measurements.

Superposition Oracles. When solving k-xor instances, if the elements in the lists are produced by a random function H (or multiple random functions) – which we safely assume below, then instead of mere *classical query access* to H, we require access to a superposition oracle:

$$O_H \ : |x\rangle\,|0\rangle \to |x\rangle\,|H(x)\rangle$$

which, as a linear operator, acts on superposition of states:

$$O_H \ : \left(\sum \alpha_i\,|x_i\rangle\right)|0\rangle \to \sum \alpha_i\,|x_i\rangle\,|H(x_i)\rangle \ .$$

This implies that H has been implemented as a quantum circuit.

Quantum Complexities. We adopt the following usual definitions of complexities:

- The quantum *query complexity* is the number of superposition oracle calls performed.
- The *time complexity* is the gate count of the quantum circuit. In all algorithms in this paper, it will turn out to be equal to the circuit depth, up to a logarithmic factor.
- The *memory complexity* is the number of qubits (including ancillas) on which it runs. Our memory complexities hide the constant overhead induced by running an oracle O_H.

Conventions. Hereafter, we count oracle queries and n-qubit register operations such as comparisons between n-bit numbers as a single time unit $O(1)$, in order to make the complexities more readable. We use the notation $\widetilde{O}$ when the time or memory complexity contains additional factors due to the management of quantum data structures, since the details of such implementations remain out of the scope of our work.

2.2 Grover's Algorithm and Amplitude Amplification

Alongside Shor's, Grover's algorithm [29] is one of the most widely known quantum algorithms. While a complete description - for which we refer to the quantum computing literature - would be outside the scope of this work, we recall that this algorithm speeds up quadratically exhaustive search.

More precisely, given a search space, e.g. $\{0,1\}^n$, and a function $f : \{0,1\}^n \to \{0,1\}$ for which there are 2^t preimages of 1, such a preimage can be found in quantum time $O(2^{(n-t)/2})$, assuming that a superposition oracle O_f can be efficiently implemented. Grover's algorithm first constructs the uniform superposition over the whole search space, then repeatedly applies an operator ($O(2^{(n-t)/2})$ times) which moves the current state towards the superposition of all preimages of 1. There are some errors, which can in turn be corrected if the exact number of preimages is known. Such errors will not impact our algorithms below.

Amplitude Amplification [16] is a generalization of Grover's algorithm where the search space has some structure. If *(1st)* there are 2^t solutions among a search space of size 2^n, *(2nd)* this search space is constructed using a quantum algorithm $\mathcal{A}$ and *(3rd)* the test uses the oracle O_f, then Amplitude Amplification returns (up to some error) the superposition of all preimages of 1 in time:

$$c \cdot 2^{(n-t)/2} \left(|\mathcal{A}| + |O_f|\right)$$

where c is a constant, and $|\mathcal{A}|$ and $|O_f|$ are the respective quantum time complexities of $\mathcal{A}$ and O_f.

More precisely, the procedure starts in an initial state $|s\rangle$, the uniform superposition over the whole search space, and applies $c2^{(n-t)/2}$ iterations. Each iteration contains a reflection through the search space (applying the operator $2|s\rangle\langle s| - I$) and another through the "good" subspace (the uniform superposition over all wanted solutions). The first reflection requires to recompute $|s\rangle$, the second to apply O_f and flip the phase of the good elements. After $c2^{(n-t)/2}$ iterations, the state is the uniform superposition over the good subspace.

2.3 Quantum Algorithms with Small Quantum Space

While constant progress has been made towards quantum fault-tolerant computation, the number of qubits seems to be, to date, a more challenging limitation on the realizability of universal quantum computers. Indeed, a quantum computer acting on S qubits needs to maintain a coherent superposition over

this whole system during the computation. In light of this potential caveat, some time-efficient quantum algorithms may reveal themselves costly. This was already argued by Grover and Rudolph regarding collision search in [30].

This is why we are interested in reducing at most the quantum time complexity *while working with a limited number of qubits*. "Limited", in the rest of this paper, means $O(n)$ (the same number as Grover's algorithm).

A technique helping to turn quantum memory requirements into classical ones is used in [19] for collision search: if one is interested in collision search *with few qubits*, the best time complexity manageable is $O(2^{2n/5})$ (instead of the lower bound $O(2^{n/3})$), using distinguished points.

Given a random function $H : \{0,1\}^n \to \{0,1\}^n$, the query-optimal BHT algorithm for collision search [17] works in two steps:

- Query $2^{n/3}$ arbitrary inputs;
- With Grover, search for a collision on one of these inputs: there are $2^{n/3}$ solutions among 2^n, hence $2^{n/3}$ Grover iterations.

In order to perform each iteration in time $O(1)$, this algorithm needs superposition query access to the memory that holds the $2^{n/3}$ results of the first step. In other terms, this algorithm requires $O(n2^{n/3})$ qubits.

Sequential Membership Testing. To overcome this cost, Chailloux *et al.* first remark that testing membership in a set of size 2^t, without quantum memory, can be done in time $O(2^t)$ (even in superposition). Indeed, given an input x, it suffices to compare sequentially x against all 2^t elements. This replaces the initial need for *quantum* memory by *classical storage*, as performing this test amounts to go through the whole set in a sequential manner.

Now, since this would bring the time complexity of BHT's algorithm to the heights of $2^{2n/3}$, we reduce the size of the list, and we now replace the arbitrary inputs by distinguished points: the list now contains only inputs x such that $H(x)$ is distinguished (say, by a zero-prefix of some size).

Since the collision we are looking for happens on a distinguished point, the search space is more structured: we use amplitude amplification instead of a simple Grover search. Assume that the list has size v and the distinguished points have all the same prefix of length u. The first step costs $2^{v+u/2}$, as each element now needs to be constructed using Grover search. The second step has $2^{(n-u-v)/2}$ iterations, as there are 2^v solutions among all distinguished points (2^{n-u}). Inside each iteration, the set of distinguished points needs to be constructed (time $2^{u/2}$) and membership to the intermediate list needs to be tested (time 2^v). This gives:

$$2^{v+u/2} + 2^{(n-u-v)/2}(2^{u/2} + 2^v)$$

optimized to $O(2^{2n/5})$ by taking $u = 2n/5$ and $v = n/5$.

3 State-of-the-Art: Known Results for the k-xor Problem

3.1 Classical Algorithms for the k-xor Problem

In [54], Wagner analyses the k-xor problem between k lists $L_1, \dots L_k$ of elements drawn uniformly at random from $\{0,1\}^n$. The goal is to find a k-tuple of elements $x_1 \in L_1, \dots, x_k \in L_k$ which xor to 0. Alternatively, one may consider a random function $H : \{0,1\}^n \to \{0,1\}^n$; the elements of the lists are created by querying H and the goal is to find $x_1, \dots x_k$ such that $H(x_1) \oplus \dots \oplus H(x_k) = 0$.

Problem 1 (k-xor with a random function). Given query access to a random function $H : \{0,1\}^n \to \{0,1\}^n$, find $x_1, \dots x_k$ such that $H(x_1) \oplus \dots \oplus H(x_k) = 0$.

Problem 2 (k-xor, with k random functions). Given query access to k random functions $H_1, \dots, H_k : \{0,1\}^n \to \{0,1\}^n$, find $x_1, \dots x_k$ such that $H_1(x_1) \oplus \dots \oplus H_k(x_k) = 0$.

Both problems will remain equivalent throughout this paper. All algorithms studied and developed below have the same time and memory complexities in either formulation.

Wagner gives an algorithm that requires $O\left(k \cdot 2^{n/(\lfloor \log_2(k) \rfloor + 1)}\right)$ time and space. The design principle is to construct a binary tree whose leafs are the k initial lists. We number these levels from 1 (leafs) to $\lfloor \log_2(k) \rfloor + 1$ (root). Level i, for $i \leq \lfloor \log_2(k) \rfloor$, contains lists of 2^i-tuples which xor to 0 on the $\frac{n}{\lfloor \log_2(k) \rfloor + 1} \times i$ first bits, of size $2^{\frac{n}{\lfloor \log_2(k) \rfloor + 1}}$ each. The root of the tree contains the expected k-xor instance. We omit constant factors in the analysis.

The base operation of Wagner's k-tree algorithm is *merging* two lists in order to obtain their parent in the tree. Merging two lists at level $i-1$ costs time $O(2^{\frac{n}{\lfloor \log_2(k) \rfloor + 1}})$ (the size of the lists). The resulting list at level i contains all pairs of 2^{i-1}-tuples (hence 2^i-tuples) which collide on $\frac{n}{\lfloor \log_2(k) \rfloor + 1}$ more bits. This explains why the parent list has (up to a constant) the same size as its children.

Remark 1. The information-theoretic query lower bound for the k-xor (alternatively, the k-sum) problem is $O(2^{n/k})$. Using a simple time-memory tradeoff, a trivial algorithm for this problem runs in time and memory $O(2^{n/2})$ if $k \geq 3$. (When $k = 2$, we fall back on collision search).

Wagner's algorithm offers *classically* the best time complexity exponent. In particular, by taking $k = 2^{\sqrt{n}}$, finding a k-xor can be done in time $O(2^{2\sqrt{n}})$.

Various improvements have proposed [12,14,43,48] but, as they target the logarithmic factors in the k-tree algorithm, study specific instances or concern time-memory tradeoffs, they remain out of scope of this paper.

ℓ-xor is easier than k-xor for $\ell \geq k$. Classically and quantumly, an algorithm for the k-xor problem can also be applied to the ℓ-xor problem for $\ell \geq k$, with the same time complexity. This reduction was outlined by Wagner [54]. Using a formulation such as Problem 2, one can remark that given an instance $H_1, \dots, H_\ell$ it suffices to call the k-xor algorithm with functions $G_1 = H_1, \dots, G_{k-1} = H_{k-1}$, $G_k = H_k \oplus \dots \oplus H_\ell$.

3.2 Quantum Algorithms for the k-xor Problem

In this section, we review known quantum algorithms that can be applied to the k-xor problem or some of its instances. As we turn ourselves towards quantum algorithms, instead of considering lists of elements drawn at random (as Wagner does in his work), we consider these lists to be produced by random functions that we can query in superposition (Problem 1 or 2).

Ambainis [3] presented a quantum algorithm for element distinctness and extended it to k-distinctness. With this algorithm, deciding k-distinctness among 2^n elements can be done with $O(2^{nk/(k+1)})$ queries and $\widetilde{O}(2^{nk/(k+1)})$ quantum time, using the same amount of *quantum memory* (i.e., qubits), by a quantum random walk on the Johnson graph. It was later noticed [20] that this algorithm works as well for the k-sum problem, or any k-relation, giving a good *query* complexity. In [9], Belovs and Spalek proved this upper bound to be optimal, using an adversary method.

Lemma 1 ([9,20]). *The quantum query complexity of k-xor for a random function is $O(2^{n/(k+1)})$, the bound is tight.*

A Problem with Time. While no best method is currently known for general k when limited to $O(2^{n/(k+1)})$ superposition queries, the algorithm derived from Ambainis' is highly uncompetitive with respect to time. We estimate that it needs at least $\widetilde{O}(2^{n/2})$ operations, for any k. In a sense, this method can be seen as the quantum equivalent of classically taking the cross-product of k lists of size $2^{n/k}$.

Grover Search. Using Grover's algorithm [29] in a "raw" manner seems also a poor idea. If the search space spans all k-tuples in input to H, looking for one whose images xors to 0, the probability that this happens is 2^{-n}. Hence a k-xor will be found in time $O(2^{n/2})$. This complexity is trivially beaten by classical algorithms for $k > 2$ and does not perform better than classical collision search when $k = 2$. Grover is known to be parallelized on 2^s quantum processors with a $2^{s/2}$ time speedup. Improvements of this speedup have been obtained for some search problems (see [6] for preimage search) but we choose to focus primarily on single-processor algorithms.

Collision Search. As mentioned above, any k-xor instance can be reduced to a ℓ-xor instance for $\ell \leq k$. From the point of view of time complexity alone, the best quantum algorithm for collision search (2-xor) runs in time and queries $O(2^{n/3})$ for an n-bit to n-bit random function [17]. This has been proven to be optimal [1,2,56]. In return, this means that there exists a quantum algorithm for the k-xor problem, for any k, running with the same time complexity. As Wagner's algorithm already obtains time $O(2^{n/3})$ for $k = 4$, this seems only relevant for collision and 3-xor search.

Subset-sum Problem. In the subset-sum problem, one is given a set of elements and looks for a subset which xors (or sums) to zero. The k-xor problem can be seen as a simpler case where the size of the sum is fixed. This problem has been widely studied classically and the quantum walk framework has been successfully applied to it [11], but these works remain, to our knowledge, unrelated to ours.

4 Quantum Algorithms for the 3-xor Problem

We now present our new quantum algorithms for the 3-xor problem. Further results for the k-xor problem for $k \geq 4$ are left to the next section. In Sect. 3, we saw that 3-xor was at least easier than collision search; while there is no exponential gap in the classical setting, we find better quantum algorithms for 3-xor than the current best known algorithms for collision search:

- First with $O(n)$ memory, improving on the time complexity of $O(2^{2n/5})$ [19];
- Second, with an exponential number of qubits, improving on $O(2^{n/3})$ [17], which is optimal for quantum collision search.

4.1 First Approach

We consider a first approach to the 3-xor problem, formulated as Problem 1, with a single random function $H : \{0,1\}^n \to \{0,1\}^n$, to which we have superposition query access via a quantum oracle O_H. The algorithm obtained below gives an overview of the techniques that enable us to overcome the complexity of collision search. In the rest of this paper, when storing the results of queries of H, we will often omit that we keep track of the antecedents of these queries. We put the focus on outputting a k-xor of *images* $H(x_1), \ldots, H(x_k)$ while disregarding the $x_1, \ldots x_k$.

Algorithm Description. Let S be the set of all $x \in \{0,1\}^n$ such that $H(x)$ has a prefix of u zeroes.

Our algorithm runs in two mains steps:

1. Build two lists L_1 and L_2 of size 2^v, where v is a parameter to be set later, which have the form in Fig. 1. That is, they contain images $H(x) \in \{0,1\}^n$ such that $H(x)$ has a prefix of u zeroes (for example, in the first u bits).[1]
2. Using *Amplitude Amplification* [16], look for an element $x \in S$ (the search space of this subprocedure) such that $H(x) \oplus z_1 \oplus z_2 = 0$ for some $z_1, z_2 \in L_1 \times L_2$.

[1] At first sight, the parameters already seem over-restricted: nothing prevents us to use lists L_1 and L_2 of different sizes. We considered this situation and did not find any advantage.

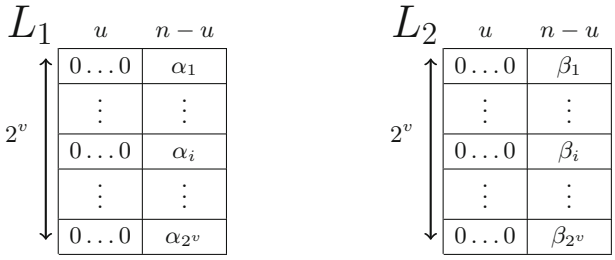

Fig. 1. Structure of the lists L_1 and L_2, of size 2^v. In the rest of this paper, the elements of the lists and their structures refer only to $H(x)$, while we may keep x alongside in order to output the antecedents of our final k-xor tuple.

Algorithm Analysis: First Step. Finding an element of S can be done using Grover's algorithm in $O(2^{u/2})$ iterations, as there is a proportion $\frac{1}{2^u}$ of "good elements" to find, the prefix condition being a u-bit condition. This gives in total $2^v \times 2^{u/2}$ calls to O_H (for simplicity, we dismiss constant factors in the complexity analyses).

Algorithm Analysis: Second Step. The second step is an Amplitude Amplification instance. It starts from the *initial state* $|s\rangle$, which is a uniform superposition over the whole search space S, and applies a sequence of iterations. Inside each iteration, we must recompute the initial state and check (in superposition) whether elements are good or not (see Sect. 2 for more details).

Checking Step. Given $x \in \{0,1\}^n$, checking if there exists $z_1 \in L_1$ and $z_2 \in L_2$ such that $H(x) \oplus z_1 \oplus z_2 = 0$ can be done in time 2^{2v} via sequential testing. More precisely, given a precomputed list L of 2^t elements in $\{0,1\}^n$, it is easy to build a quantum oracle which tests if an input x appears in L. On input $|x\rangle|0\rangle$, the oracle returns $|x\rangle|1\rangle$ if $x \in L$ and $|x\rangle|0\rangle$ otherwise. It runs in quantum time $O(n \cdot 2^t)$, without any quantum memory requirement: this amounts to control a sequence of n-bit comparisons against x (see Sect. 2 for a reminder of [19]).

The fact that L is known beforehand introduces a cost in *classical storage*. This storage is read sequentially (it only instructs which operations to perform) and does not need random-access. In our case, the list L contains the sums of all pairs $z_1 \in L_1, z_2 \in L_2$. It is produced on the fly and does not need to be stored itself. As it has 2^{2v} elements, the checking step costs 2^{2v} comparisons.

Initial State. The initial state of this Amplitude Amplification is the uniform superposition over the search space S (elements whose image has a prefix of u zeroes). It can be produced in $2^{u/2}$ time and queries using Grover's algorithm.

Number of Iterations. The search space S is of size 2^{n-u}. A "good element" in this search space gives a solution to the 3-xor problem. As it must collide with some sum $z_1 \oplus z_2$ in $L_1 \times L_2$ and there are 2^{2v} such sums, the number of good

elements is 2^{2v}. Hence the number of iterations is $O\left(\sqrt{2^{n-u}/2^{2v}}\right)$. In each of these iterations, the initial state is computed and uncomputed, and the current superposition goes through the checking step.

All in all, the second step costs a quantum time:

$$2^{(n-u-2v)/2}\left(2^{u/2}+2^{2v}\right)$$

Optimizing both parameters u and v gives $v=\frac{n}{8}$ and $u=4v$, which yields a time complexity $O(2^{3n/8})$. The classical memory complexity is $O(2^{n/8})$. All of this analysis is average-case. With a random function H, the fluctuations (e.g., in the size of S_u) cannot, with overwhelming probability, yield more than constant variations in the total time complexity.

Remark 2. Although the obtained time complexity is higher than the collision query lower bound $O(2^{n/3})$, it improves on $O(2^{2n/5})$, the current best known collision query and time complexity with $O(n)$ quantum memory.

Another Consequence of this Approach. If we disregard quantum memory consumption, the lists L_1 and L_2 may be stored using qubits. More precisely, to perform the checking step more efficiently, one may store L_1 in a quantum memory and then, given x, try every element y in L_2 sequentially and test whether $x \oplus y \in L_1$ efficiently (even when x is given in superposition). This decreases the cost of this test from 2^{2v} to 2^v. As a consequence, the time complexity becomes:

$$2^{u/2+v}+2^{(n-u-2v)/2}\left(2^{u/2}+2^{v}\right)$$

which implies $2u=v$ as best parameters and $v=\frac{n}{6}$. We obtain a time complexity of $O(2^{n/3})$ using $O(2^{n/6})$ quantum memory, improving on [17] w.r.t quantum memory.

4.2 Second Approach

In order to improve over the previous algorithm, we modify the structure of the lists (Fig. 2). This new algorithm is inspired from the list-merging ones from [46]. We introduce not two, but three parameters v, u, t such that:

- Both lists[2] have size 2^v;
- The "completely free" part has size t;
- Elements of L_1 take 0 on u bit positions;
- Elements of L_2 take 0 on u *different* bit positions;
- Elements of both L_1 and L_2 take 0 on the $n-2u-t$ remaining bits (a common prefix, as before).

We first consider the case $v \geq u$.

[2] As before, we seem to over-restrict the parameters, since both lists could have different sizes. We found that it gave no improvement: intuitively, we wish to maximize a certain number of "good elements" given by a cross-product of L_1 and L_2, whose size is maximized w.r.t the cost of producing L_1 and L_2 when both have equal size.

L_1

$n-2u-t$	u	u	t
$0\dots0$	$0\dots0$	x_1	β_1
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$0\dots0$	$0\dots0$	x_i	β_i
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$0\dots0$	$0\dots0$	x_{2^v}	β_{2^v}

(2^v rows)

L_2

$n-2u-t$	u	u	t
$0\dots0$	y_1	$0\dots0$	α_1
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$0\dots0$	y_i	$0\dots0$	α_i
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$0\dots0$	y_{2^v}	$0\dots0$	α_{2^v}

(2^v rows)

Fig. 2. Structure of the lists L_1 and L_2 of size 2^v.

Algorithm Design. As above, we consider the set S of "distinguished" elements x such that $H(x)$ has zeroes in the first $n-2u-t$ bits. We build the lists L_1 and L_2 and look for a 3-xor instance $H(x) \oplus z_1 \oplus z_2 = 0$ with $z_1 \in L_1, z_2 \in L_2$. But the new structure of the lists L_1 and L_2 makes the checking step more efficient: there will be no need to go through the whole product $L_1 \times L_2$.

Our algorithm runs in two main steps:

1. Build the lists L_1 and L_2;
2. Using Amplitude Amplification, look for $x \in S$ (the search space) such that $H(x) \oplus z_1 \oplus z_2 = 0$.

Analysis: First Step. The first step builds two lists of 2^v elements with zeroes in $u+n-2u-t$ positions. Each of these elements is produced separately using Grover search. The total time complexity, without constant factors, is:

$$2^v \times 2^{\frac{n-u-t}{2}} \quad .$$

Analysis: Second Step. In the second step, the search space is S, and it contains 2^{2u+t} elements among 2^n. The initial state $\sum_{x \in S} |x\rangle$ can be constructed using Grover's algorithm in $2^{\frac{n-2u-t}{2}}$ time and queries. To estimate the number of iterations, we have to find the number of good elements, that is, the number of $x \in S$ such that there exists $z_1 \in L_1, z_2 \in L_2$, $H(x) \oplus z_1 \oplus z_2 = 0$. For $x \in S$, there are on average 2^{v-u} elements z_1 in L_1 that collide with $H(x)$ on the third column and 2^{v-u} elements z_2 in L_2 that collide with $H(x)$ on the second column. Each of these $2^{2(v-u)}$ pairs z_1, z_2 yields a 3-xor to 0 on the three first columns. For each of these pairs, there are t remaining bits to cancel (the last column). Hence the probability that x yields a solution is $\frac{2^{2(v-u)}}{2^t}$, which gives $2^{\frac{t-2(v-u)}{2}}$ iterations.

Checking Step. We now detail how to check quantumly whether $x \in S$ yields a solution or not, in 2^{2v-u} comparisons only (with minor constants). Since $v \geq u$, we can cut the list L_1 in sublists of size 2^u and expect each of these sublists to contain an element z_1 which collides partially with $H(x)$ on the third column. We will simply assume that there is exactly one. If there are more, these additional

solutions will be dismissed. If there is none, we will skip this sublist and go directly to the next one.

We can build a unitary that, given x in input, does for each sublist L'_1:

1. Go through L'_1 and retrieve z_1 which yields the partial collision. This requires 2^u comparisons, since this is the size of L'_1, and no additional quantum memory, since these comparisons are performed sequentially as above;
2. After retrieving z_1 and storing it, go through L_2 and find z_2 which yields a 3-xor, if it exists. This requires 2^v comparisons.
3. If a solution is found, return it, if not, return None.

As there are 2^{v-u} sublists to analyze, there are in total $2^{v-u}(2^u + 2^v) = 2^{2v-u}$ comparisons performed (since $v \geq u$). The output gives whether x is a good element or not and if so, the corresponding 3-xor instance.

Reduction of the Solution Space. Keeping only *one* partially colliding z_1 where there could be more has the consequence of reducing the actual set of good elements of the Amplitude Amplification procedure (the test function drops some good elements). We show that this has no asymptotic consequence.

Let x be a fixed element of the search space. There are 2^{v-u} sublists L'_1, from which $(1 - e^{-1})2^{v-u}$ contain at least one solution z_1 (the others yield no solution). We bound probabilistically the total number of z_1 that will be dropped. Let $Z(x)$ be the total number of z_1 over all these sublists, then $Z(x)$ is the sum of $(1 - e^{-1})2^{v-u}$ independent random variables of expectation 1. An additive Chernoff bound applies. For any $0 < \delta \leq 1$:

$$Pr\left(Z(x) \geq (1+\delta)(1-e^{-1})2^{v-u}\right) \leq e^{-\delta^2(1-e^{-1})2^{v-u}/3} \ .$$

Where $\left(Z(x) - (1 - e^{-1})2^{v-u}\right)$ represents the total number of z_1 lost for x.

We can do a union bound with x spanning the whole search space (of size 2^{2u+t}):

$$Pr\left(\exists x, Z(x) \geq (1+\delta)(1-e^{-1})2^{v-u}\right) \leq 2^{2u+t}e^{-\delta^2(1-e^{-1})2^{v-u}/3} \ .$$

By taking an appropriate δ, we find that with high and constant probability, for all x simultaneously, the number of z_1 dropped is negligible w.r.t the total amount. Assume now that x should have been a solution. Some z_1 yields a 3-xor instance: with our test, it may or not be dropped. We see that the probability for it to be dropped is negligible. With high probability, x remains a solution; the same goes for any x. Hence the final solution space is only negligibly smaller than the previous one, with no consequence on the time complexity.

Total Time. The time complexity rewrites:

$$2^{v+\frac{u}{2}+\frac{n-2u-t}{2}} + 2^{\frac{t-2(v-u)}{2}}\left(2^{\frac{n-2u-t}{2}} + 2^{2v-u}\right) = 2^{\frac{n}{2}+v-\frac{u+t}{2}} + 2^{\frac{n}{2}-v} + 2^{\frac{t}{2}+v} \ .$$

To find the right point of optimization, let us write the partial derivative in v and nullify it:

$$2^{\frac{n}{2}+v-\frac{u+t}{2}} - 2^{\frac{n}{2}-v} + 2^{\frac{t}{2}+v} = 0 \ .$$

This gives an equality between the exponents: $\frac{n}{2} - v = \frac{t}{2} + v$ i.e. $t = n - 4v$ and $\frac{n}{2} + v - \frac{u+t}{2} = \frac{n}{2} - v$ i.e. $u = 8v - n$.

Optimization. The final complexity is $2^{\frac{n}{2}-v}$ with an (apparently) free parameter v. Let us have a look at the conditions on the range of v: first, we have considered the case $v \geq u$, i.e. $v \geq 8v - n \Rightarrow v \leq \frac{n}{7}$. Second, we must have $t - 2(v - u) \geq 0$ (the Amplification Amplitude procedure needs a positive number of iterations, 1 means that all elements of the initial space are solutions), i.e. $v \geq \frac{n}{10}$. Finally, we must have $u \geq 0$, i.e. $8v - n \geq 0$ i.e. $v \geq \frac{n}{8}$. This means that this technique works only in the range $v \in \left[\frac{n}{8}; \frac{n}{7}\right]$ where it gives a quantum time complexity $2^{\frac{n}{2}-v}$ and a classical memory complexity 2^v.

Case $u > v$. When $u > v$, the probability that an element $x \in S$ in the search space yields a partial collision with $z \in L_1$ is $2^{v-u} < 1$. The checking procedure needs to be reconsidered with this point of view: we now go through the whole list L_1 sequentially (and computationally, by performing comparisons) and find the element z_1, if it exists, which collides with $H(x)$ on the u bits of the third column. If it does not exist, we return 0 immediately (not a good element). Otherwise, we go through L_2 and find the element z_2, if it exists, which collides with $H(x)$ on the u bits of the second column. The number of comparisons performed by this checking step is now 2^v. The other terms in the total time complexity are unchanged. It rewrites:

$$2^{v+\frac{u}{2}+\frac{n-2u-t}{2}} + 2^{\frac{t-2(v-u)}{2}}\left(2^{\frac{n-2u-t}{2}} + 2^{2v-u}\right) = 2^{\frac{n}{2}+v-\frac{u+t}{2}} + 2^{\frac{n}{2}-v} + 2^{\frac{t}{2}+u} \ .$$

Optimizing gives $t = 10v - n$ and $u = n - 6v$, but $u \geq v$ enforces the condition $n - 6v \geq v$ i.e. $v \leq \frac{n}{7}$, which means that we fall back in the complexity range of above.

This leads to a few remarks:

- When v is minimal in this range, $v = \frac{n}{8}$ leads to $u = 0$: we obtain the first approach as above.
- When v is maximal, $v = \frac{n}{7}$ leads to $u = v$, and the best time complexity, in $O(2^{5n/14})$. This is very close to $2^{n/3}$, but does not yet reach the quantum collision bound (optimal number of queries).

Details of the Best Method. The case $v = \frac{n}{7}$ and $u = v$ - represented in Fig. 3 - gives the best quantum time complexity. Given an element $x \in \{0,1\}^n$ such that $H(x)$ has the according zero-prefix, we expect it to collide on average with one element of L_1 on the third column and with one element of L_2 on the second column. Finding these elements takes time 2×2^v to go through both lists. It remains to verify if $H(x) \oplus z_1 \oplus z_2$ sums to zero in the last t bits.

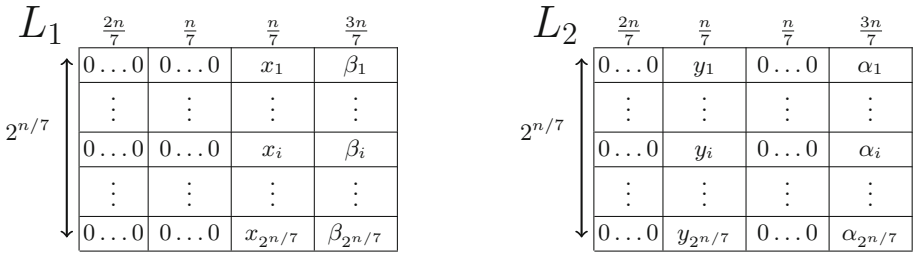

Fig. 3. Structure of the lists L_1 and L_2 of size $2^{n/7}$.

To build the lists L_1 and L_2, one needs time $2^{n/7+n/7+n/14} = 2^{5n/14}$. To find a 3-xor, one needs time:

$$2^{3n/14}\left(2^{n/7} + 2^{n/7}\right)$$

since, given an element of the $2n/7$-zero prefix space, there exists a match on the intermediate $n/7$ bits of L_1 and L_2 with high probability; then the probability that it is the good one only depends on the $3n/7$ remaining degrees of freedom (hence $2^{3n/14}$ iterations are necessary).

Theorem 1 (Quantum 3-xor Algorithm with Small Number of Qubits). *There exists a quantum algorithm for the 3-xor problem running in quantum time $O(2^{5n/14})$, using $O(n)$ qubits and $O(2^{n/7})$* classical *memory.*

It is worth to notice that this algorithm, as the others in this paper, is inherently quantum: although we can write a classical counterparts (by replacing Grover search steps with classical exhaustive searches), trying to optimize the classical time complexity gives a time $O(2^{n/2})$ and $O(2^{n/4})$ classical memory (we get $u = v = t = \frac{n}{4}$).

4.3 Using Exponential Quantum Memory

If we allow an exponential amount of qubits to be used, we can also take the time complexity of the 3-xor problem below the best quantum time for collision search. This time, it is more surprising, since we go below the *optimal* query complexity for collision search.

Theorem 2 (Quantum 3-xor Algorithm). *There exists a quantum algorithm for the 3-xor problem running in time $\widetilde{O}(2^{3n/10})$ and using $O(2^{n/5})$ qubits.*

Proof. This procedure is inspired from the low-memory one. Since we authorize quantum memory, the $2n/7$ common prefix of zeroes is not necessary anymore (it has been used to amortize the cost of the membership oracle in the amplitude amplification procedure).

As before, building two lists of different sizes does not give better results, nor does building lists of size 2^v with $v \leq u$, where u is the number of inner zeroes in the intermediate columns. So we take $v \geq u$ and write the time complexity:

$$2^{v+\frac{u}{2}} + 2^{\frac{n-2v}{2}} \left(2^{v-u}\right) = 2^{v+\frac{u}{2}} + 2^{\frac{n-2u}{2}} \quad .$$

Since we store the lists in quantum memory, testing membership now costs a logarithmic overhead which we dismiss. The 2^{v-u} factor stems from the fact that there are approximately 2^{v-u} partial collisions on L_1, each of which yields a membership test to L_2.

Optimization now yields $v + \frac{u}{2} = \frac{n}{2} - u$ i.e. $u = \frac{n}{3} - \frac{2v}{3}$. The complexity is $2^{\frac{2v}{3}+\frac{n}{6}}$. We also need $v \geq u$, hence $v \geq \frac{n}{5}$. □

We cannot reduce v below $\frac{n}{5}$, but there is also no interest in increasing it, since this would increase both the time and memory complexity. Taking $v = \frac{n}{5}$ also implies $u = v = \frac{n}{5}$.

5 Quantum Algorithms for the k-xor Problem, $k \geq 4$

In this section, we present new algorithms for the k-xor problem, with $k \geq 4$. Again, we propose algorithms in two different models. When using exponential quantum memory, we propose a general quantum algorithm for the k-xor problem which gives a speedup over Wagner's k-tree method for any k. With $O(n)$ qubits, we find quantum speedups for specific values of k.

Table 2 gives a summary of our k-xor quantum algorithms with exponential quantum memory, while Table 3 gives a summary of k-xor quantum algorithms with $O(n)$ quantum memory. In both cases, complexities given are those of the best algorithms available, with respect to the time (in particular, there can be memory-efficient algorithms with higher time complexity).

Table 2. k-xor quantum algorithms with exponential quantum memory. Complexities C are given as $\log_2(C)/n$. The complexities of the classical algorithms are given by Pollard's rho algorithm for collisions, and by [54].

k	Classical time	Classical memory	Quantum time	Quantum memory	Reference
2	1/2	0	1/3	1/3	[17]
3	1/2	0	3/10	1/5	Theorem 2
4	1/3	1/3	1/4	1/4	Theorem 4
5	1/3	1/3	1/4	1/4	Theorem 4
6	1/3	1/3	1/4	1/4	Theorem 4
7	1/3	1/3	1/4	1/4	Theorem 4
8	1/4	1/4	1/5	1/5	Theorem 4
...	...	...	...	...	...
k	$(1 + \lfloor \log_2(k) \rfloor)^{-1}$	$(1 + \lfloor \log_2(k) \rfloor)^{-1}$	$(2 + \lfloor \log_2(k) \rfloor)^{-1}$	$(2 + \lfloor \log_2(k) \rfloor)^{-1}$	Theorem 4
...	...	...	...	...	...

Table 3. k-xor quantum algorithms with *polynomial* quantum memory. Complexities C are given as $\log_2(C)/n$. The complexities of the classical algorithms are given by Pollard's rho algorithm for collisions, and by [54].

k	Classical time	Classical memory	Quantum time	Classical memory	Reference
2	1/2	0	2/5	1/5	[19]
3	1/2	0	5/14	1/7	Theorem 1
4	1/3	1/3	1/3	1/9	Theorem 3
5	1/3	1/3	7/22	1/11	Theorem 3
6	1/3	1/3	4/13	1/13	Theorem 3
7	1/3	1/3	3/10	1/15	Theorem 3

A graphical comparison between these cases is provided in Fig. 4.

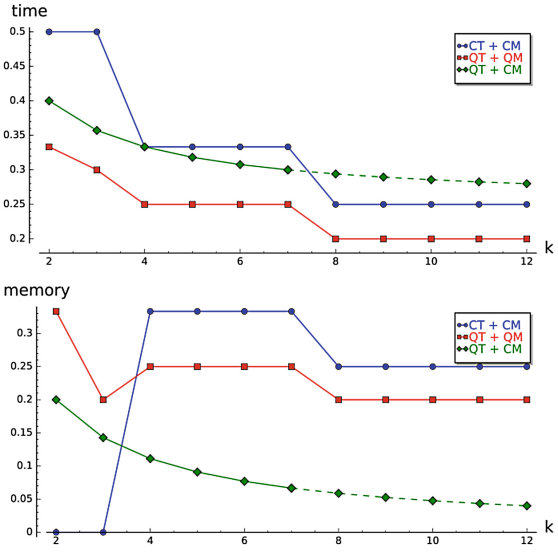

Fig. 4. Time and memory complexities of some k-xor algorithms. *Blue*: algorithm with classical time and classical memory (CT+CM), as provided in Pollard's rho algorithm for collisions, and by [54]. *Red*: algorithm with quantum time and exponential quantum memory (QT+QM), as provided in Theorems 2 and 4. *Green*: algorithm with quantum time and $O(n)$ quantum memory (QT+CM), as provided in Theorems 1 and 3. (Color figure online)

5.1 Quantum k-xor Algorithms With Low Quantum Memory

We propose an algorithm that enables us to find better-than-classical quantum time complexities, when using $O(n)$ qubits only. The result can be applied successfully for $k = 5, 6, 7$. Its complexity is given by the following theorem:

Theorem 3. *For each k, there exists a quantum algorithm for solving the k-xor problem running in time $O\left(2^{\frac{(k+2)n}{2(2k+1)}}\right)$ and using $O\left(2^{n/(2k+1)}\right)$ classical storage.*

L_1

t	u	u	$\dots$	u	$n-(k-1)u-t$
$0\dots0$	x_1	$0\dots0$	$0\dots0$	$0\dots0$	α_1
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$0\dots0$	x_i	$0\dots0$	$0\dots0$	$0\dots0$	α_i
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$0\dots0$	x_{2^u}	$0\dots0$	$0\dots0$	$0\dots0$	α_{2^u}

(2^u rows)

Fig. 5. Structure of the list L_1 of size 2^u.

L_j

t	u	$\dots$	u (column $j+1$)	$\dots$	u	$n-(k-1)u-t$
$0\dots0$	$0\dots0$	$0\dots0$	y_1	$0\dots0$	$0\dots0$	β_1
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$0\dots0$	$0\dots0$	$0\dots0$	y_i	$0\dots0$	$0\dots0$	β_i
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$0\dots0$	$0\dots0$	$0\dots0$	y_{2^u}	$0\dots0$	$0\dots0$	β_{2^u}

(2^u rows)

Fig. 6. Structure of the list L_j of size 2^u.

Proof. We take $k-1$ lists $L_1, \dots, L_{k-1}$, each of size 2^u and containing elements with $t+(k-2)u$ zeroes, a prefix of size t and $k-2$ ranges of size u (Figs. 5 and 6). Of these ranges in list L_j, they all contain zeroes, except column $j+1$ (see Figs. 5 and 6).

Given $x \in \{0,1\}^n$, it collides on average with one element in L_1 on its corresponding non-zero column of size u, the same for L_2, etc. It remains to obtain 0 in $n-(k-2)u-t$ bits. The time is:

$$2^{u+(k-2)u/2+t/2} + 2^{\frac{n-(k-1)u-t}{2}}\left(2^{t/2}+2^u\right) \tag{1}$$

which gives $u=\frac{n}{2k+1}$, $t=2u$ and a complexity exponent $\frac{(k+2)n}{2(2k+1)}$. □

When $k=4$, we fall back on complexity $2^{n/3}$, but the memory complexity is better than Wagner's $2^{n/3}$: it drops to $2^{n/9}$.

When $4 < k < 8$, this gives better-than-classical time complexities in the bounded quantum memory setting[3]. In particular, setting $k = 7$ gives $O(2^{3n/10})$ time.

For $k \geq 8$, the cost of the Grover search step fails to decrease enough to be competitive against Wagner's algorithm, which is why Table 3 stops at $k = 7$. With $k \geq 8$, no quantum speedup for k-xor, with polynomial quantum memory, is known.

5.2 A Quantum Walk 4-xor Algorithm

In this subsection and the next one, we present a quantum walk algorithm for the general k-xor problem. First, we focus on the 4-xor case; a generalization of this method to any power of 2 will be presented later. Our algorithm uses the framework of quantum walks as described in [40]. It is inspired by [3] in that it also walks on the Johnson graph.

We formulate the 4-xor problem as following: given superposition query access to a random function $H : \{0,1\}^n \to \{0,1\}^n$, finding 4 or less distinct elements x_1, x_2, x_3, x_4 such that $H(x_1) \oplus H(x_2) \oplus H(x_3) \oplus H(x_4) = 0$. This algorithm adapts when we consider 4 random functions instead of one, when we consider quantum random memory accesses instead of oracle calls and when we enforce exactly 4 outputs (and refuse "smaller" collisions).

We define the Johnson graph $J(2^n, 2^r)$, where vertices are subsets of 2^r elements in $\{0,1\}^n$. This is the same graph as used by Ambainis' element-distinctness algorithm in [3]; its spectral gap is approximately $\delta = 2^{-r}$.

We add additional information to a vertex in the graph: we maintain the list of all r-collisions (collisions on the first r bits) within this set. There are approximately 2^r such collisions. A vertex in the graph is *marked* if two of these r-collisions collide. Hence, a random vertex in the graph has probability $2^{2r-(n-r)}$ of being marked, as there are approximately 2^{2r} pairs of r-collisions to construct and $n - r$ bit conditions to check.

Remark 3. In the way we mark vertices, we are losing information: indeed, there are vertices which contain a 4-sum to zero, but are not marked. Any method to efficiently mark more vertices would improve the complexity of our quantum walk, but it would also have consequences on the classical complexity of k-xor.

There is actually no need to perform this check on the fly; in the data structure representing a vertex, we store the list of r-collisions in a sorted manner (using e.g. a skip list, as in [3], which adds logarithmic overhead) and keep updated a "flag" bit which indicates whether the vertex is marked.

[3] Note that $2^{\frac{(k+2)n}{2(2k+1)}} \geq 2^{\frac{n}{1+\lfloor \log_2 k \rfloor}}$ for each $k \geq 8$, since

$$\frac{k+2}{4k+2} \geq \frac{1}{1+\lfloor \log_2 k \rfloor} \Leftrightarrow (1+\lfloor \log_2 k \rfloor)(k+2) \geq 4(k+2) \geq 4k+2.$$

Using this data structure, the cost of an update (constructing the superposition of all neighbors) is constant. Indeed, we only need to perform one query and update the list of r-collisions: on average, a given element appears in 1 of these collisions; we remove this one and add the potential new collision (which is found in constant time, since the list of elements is also sorted).

The time complexity of this quantum walk is, by [40]:

$$S + \frac{1}{\sqrt{\epsilon}}\left(C + \frac{1}{\sqrt{\delta}}U\right)$$

where U is the update cost (constant), C the checking cost (constant), S the setup cost (equal to 2^r, as there are 2^r initial queries to perform), ϵ the proportion of marked vertices and δ the spectral gap. In our case, this gives:

$$2^r + 2^{\frac{n-3r}{2}}\left(2^{r/2}\right) = 2^r + 2^{\frac{n}{2}-r} \ .$$

As it appears, the optimal time complexity is $\widetilde{O}(2^{n/4})$ (there are logarithmic factors to take into account while updating the quantum data structures) with a quantum memory complexity of $\widetilde{O}(2^{n/4})$. Besides, this gives a quantum time-quantum memory trade-off curve $T \times S = 2^{n/2}$ for the 4-xor problem.

Lemma 2. *For any quantum memory size $S \leq 2^{n/4}$, there exists a quantum algorithm for the 4-xor problem running in time $T = 2^{n/2}/S$.*

Remark 4. There are some possible caveats to this method, which we address in the next subsection. First, it is possible that the final 4-xor is a sum of the form $H(x_1) \oplus H(x_2) \oplus H(x_3) \oplus H(x_4) = 0$ where $H(x_1) = H(x_3)$ and $H(x_2) = H(x_4)$, giving a trivial result. This kind of failure happens infrequently. Furthermore, we did not bound precisely the cost of a vertex update: such an update is performed in superposition on all vertices. When we remove and add an element of a vertex, although there would be on average only one r-collision to remove and add, the "average case" is not enough, since we are considering all vertices at the same time. This may incur another quantum time overhead.

5.3 A Quantum Walk 2^k-xor Algorithm

The technique used successfully for the 4-xor problem can be applied for the 2^k-xor, in a similar manner as Wagner's classical k-tree technique. We perform a quantum walk on the Johnson graph $J(2^n, 2^r)$. Marked ("good") vertices contain a 2^k-xor, but most of the vertices with a 2^k-xor are actually not marked: this is where we lose the most against the quantum query lower bound.

Our quantum data structure for vertices stores much additional information, multiplying the quantum memory by a factor k. There are k sorted lists on k levels:

- On the first level, we store the 2^r elements of the vertex (L_1);

- On the second level, we store the 2^r r-collisions expected from these elements (and store, for each collision, a pair of pointers towards the elements which produced it) (L_2);
- On the third level, we store the 2^r $2r$-4-xor (4-xor on $2r$ bits) expected from the collisions above (collisions of collisions) (L_3);
- ...
- On the k-th level, we store the (approximately) 2^r $(k-1)r$-2^{k-1}-xor expected (L_k).

What remains is, from pairs of these 2^r $(k-1)r$-2^{k-1}-xor, to obtain a collision on the $n-(k-1)r$ remaining bits. Such a collision is easy to find, since the list at the k-th level is sorted.

Updating the vertex data structure (i.e., removing an element and adding another) should be done in time $O(k)$, as we have k levels to go through and to update. The time (and memory) complexity of this walk then amounts to:

$$k \times 2^r + 2^{\frac{n-(k-1)r-2r}{2}}(2^{r/2} \times k)$$

which we optimize using $r = \frac{n}{k+2}$ to $O\left(k2^{\frac{n}{k+2}}\right)$. A number of technicalities remain to be handled.

The Diminution of the Number of Collisions. Suppose that at level i, the 2^r bit-strings corresponding to the non-colliding part of the $(i-1)r$-2^{i-1}-xor of this level take exactly 2^r different values (i.e., $|L_i| = 2^r$). At level $i+1$, they make take strictly *less* values (i.e., $|L_{i+1}| < |L_i|$). This could make the number of distinct elements stored at level $i+2$ decrease further, and so on. If we authorize non-distinct elements in the 2^k-xor instance in output, this is not an issue. Indeed, $|L_{i+1}| < |L_i|$ happens only if a pair of elements at level i already *completely collides*. We can check this while computing or updating L_i and immediately mark the vertex: a full collision at level $i+1$ corresponds to a 2^i-xor instance, which in turn corresponds trivially to a 2^k-xor instance with non-distinct elements. If this is not authorized, then the average size of the lists indeed decreases. We will handle this below.

Increasing the Number of Collisions. On the contrary, there may be too much collisions at level i and L_{i+1} may increase in size. This is an issue for all vertices, since the quantum memory used stores the data structures in superposition for all. We must ensure that L_i is bounded for all i and for *all vertices*. But this is trivially done: if there are too much collisions, we dismiss them. Recall that the goal of our marking procedure is to mark *efficiently some* vertices containing a 2^k-xor, not all of them.

Outputting the 2^k-xor Result. At the end of the quantum walk, we perform a measurement and get the whole data structure of a vertex. We make sure to store, for each partial collision or xor at level i, the two elements of level $i-1$ which produced it (this introduces only a constant overhead). Hence, outputting the whole 2^k-xor instance amounts to the traversal of a tree of pointers, performed in time $O(2^k)$. This adds to the complexity of our algorithm.

The Real Cost of an Update. Updating the data structure is done when we remove, and then add an element to the vertex (going to one of its neighbors). We then have to go through the k levels. We expect "on average" $O(1)$ computations to be performed at each level, as each element on level i intervenes "on average" in one collision at level $i+1$ (collision which we have to remove, or to add, depending on the situation). Classically, we may consider such an average case, but quantumly, we are studying all elements in superposition.

Suppose that $x \in \{0,1\}^n$ intervenes in multiple collisions of a random function $H : \{0,1\}^n \to \{0,1\}^n$. This means that $f(x) \oplus f(y_i) = 0$ for multiple values: $y_1, \dots y_{\ell-1}$. This means, in turn, that x is actually involved in a ℓ-collision for some $\ell > 2$.

The average number of ℓ-collisions in a random n-bit to n-bit function is $\frac{e^{-1}2^n}{\ell!}$ ([26], Theorem 4). At each level, computing the next collisions is done by sorting a list of 2^r values; this enables us to select *only* 2-collisions and avoid any element to appear twice. That way, only *one* update has to be propagated towards the next levels. Besides, some of the branches die out, as the number of collisions is below 2^r: we expect $2^r e^{-1}/2 \simeq 0.184 \times 2^r$.

This seems to be a caveat of our technique: as the number of levels grows higher, the number of collisions at the k-th level is of the order $\frac{2^r}{(2e)^k}$. The number of marked vertices, in turn, is smaller than our first estimation.

Adapted Time Complexity. The remarks above make us rewrite the time complexity by taking k into account more precisely:

$$k \times 2^r + 2^{\frac{n-(k-1)r-2r}{2}}(2e)^k(2^{r/2} \times k) + 2^k = k2^r + k2^{\frac{n-kr}{2}+k(1+1/\ln 2)} + 2^k$$

which optimizes to $(k+2)r = n + 2k\left(1 + \frac{1}{\ln 2}\right)$. Notice that $2^{2k\left(1+\frac{1}{\ln 2}\right)/(k+2)} \leq 4e^2$, a constant.

General Case. We studied the case of a power of 2, but generally, the K-xor problem may be solved using a similar quantum walk as the $2^{\lfloor \log_2 K \rfloor}$-xor.

Theorem 4 (Quantum Walk K-xor Algorithm). *For any integer K, there exists a quantum algorithm that, given superposition oracle access to a random function $H : \{0,1\}^n \to \{0,1\}^n$, solves the K-xor problem in quantum time and memory (asymptotic in K and with additional factors in r due to data structure management):*

$$\widetilde{O}\left(K 2^{\frac{n}{\lfloor \log_2 K \rfloor + 2}}\right) .$$

At first, the time complexity could have been with a factor K^c with $c = 1 + \frac{1}{\ln 2}$ slightly greater than 1, coming from the way we propagate the update in the $\log_2 K$-leveled data structure. We correct this time complexity by slightly increasing the size of the lists stored in the structure, which in turn increases the number of partial collisions we get, and makes the time and memory complexities equal.

When querying K independent oracles instead of one (alternatively, when we have K independent lists), the complexities are the same. Since the lists are

independent, we consider K levels instead of $\log_2 K$; level i contains increasing partial collisions between elements from $L_1 \dots L_i$.

Multiple K-xor. Let $\lfloor \log_2 K \rfloor = k$. Using Wagner's algorithm, [54], one can output 2^c K-xor instances in time $O\left(K 2^{\frac{n}{k+1}} \times 2^{\frac{c}{k+1}}\right)$, provided that $c \leq \frac{n}{k}$. Using our quantum walk algorithm, it is possible to obtain a similar result. We use more quantum memory. The initialization step is performed as above. The march is then divided in 2^c steps, each of which corresponds to a new K-xor instance, that we may store until the end. First, we march on the Johnson graph and marked vertices are those which contain at least a K-xor; second, we march on *vertices that contain a* K-*xor* and marked vertices are those which contain at least *two* K-xors, and so on. The end of each subwalk is the beginning of the next one. The quantum time complexity becomes approximately:

$$K \times 2^r + 2^c \times 2^{\frac{n-(k-1)r-2r}{2}} (2^{r/2} \times K)$$

with a final $r = \frac{2c+n}{k+2}$ and a time complexity: $\widetilde{O}\left(K 2^{\frac{2c+n}{k+2}}\right)$. This is legitimate as long as $n \geq (k+1)r$ (positive number of iterations), i.e. $n \geq \frac{2c+n}{k+2}(k+1)$, i.e. $c \leq \frac{n}{2(k+1)}$.

6 Quantum k-xor Parallel Algorithms

While an involved discussion on all respective parallelization strategies for classical and quantum algorithms for the k-xor problem would be outside the scope of our work, we can make a few remarks on the cost of parallelized versions of our algorithms.

First of all, we consider as "quantum parallel algorithm" a quantum algorithm written in the circuit model, with an adapted definition of *time complexity*: instead of merely counting the number of quantum gates, we authorize up to 2^s of them, if there are 2^s "quantum processors" available, to be applied in a single time step. Such a definition, which puts away possible communication overheads between quantum processors, has been extensively justified in [7].

Parallelization of quantum algorithms seems sometimes to give more negative than positive results: as an example, although Grover's algorithm can be parallelized, it cannot gain more than a factor $2^{s/2}$ in time when using 2^s processors, while classical exhaustive search does not suffer from this issue.

Quantum Parallel 3*-xor Algorithm.* Consider our 3-xor algorithm of Sect. 4, running on one processor in time $O(2^{5n/14})$ with classical storage of size $O(2^{n/7})$. We distribute it over 2^s quantum processors, in the following way: both the computation of intermediate lists and the final step (amplitude amplification) are shared among these processors; the former more efficiently (a factor 2^{-s}) than the latter ($2^{-s/2}$). Rewriting the time complexity and performing a similar optimization process, we obtain a time $O\left(2^{\frac{n}{2}-v-\frac{s}{2}}\right)$ for $v \leq \frac{n+s}{7}$.

By taking $v = \frac{n+s}{7}$, i.e. increasing the classical resources[4], we get a time $\widetilde{O}\left(2^{\frac{5n}{14}-\frac{9s}{14}}\right)$, with a more significant parallelization speedup compared to Grover's algorithm.

Although trading quantum memory for classical storage seems advantageous, many alternative and debatable benchmarks can be used to quantify the overall cost of running algorithms. One of these benchmarks (used in [10] against the quantum collision algorithm of [19]) binds together the memory and the number of processors used as *hardware resources*. Using this particular benchmark, single-processor Pollard rho costs $O(2^{n/2})$, while 2^s-processor parallelized rho costs $O(2^{n/2-s} \times 2^s) = O(2^{n/2})$, and no quantum collision search (because of their memory usage) outperforms the classical ones.

Even with this benchmark, our new 3-xor algorithm performs efficiently. Suppose that we equalize the number of processors and the memory used (which remains single-access): $s = \frac{n+s}{7}$ gives $s = \frac{n}{6}$. This gives a time $\widetilde{O}\left(2^{n/4}\right)$. The product time-resources is $\widetilde{O}\left(2^{5n/12}\right)$ and improves over the best classical complexity.

7 Conclusion

In this work, we have studied quantum k-xor algorithms, proposing new ones with the lowest known *time complexity*.

The previous best known quantum attacks could only rely on collision search for $k = 3$ and did not outperform Wagner's algorithm for $k \geq 4$. Even though the optimal quantum query complexity could be attained, there was virtually no quantum time-efficient method for the k-xor problem.

We filled this gap in two settings, depending on the status of *quantum memory* (see Fig. 4 for more details).

- If quantum memory is considered as cheap as classical memory, we authorize the adversary to use exponential amounts of it. We obtain an improvement over quantum collision search for the 3-xor problem ($\widetilde{O}(2^{3n/10})$ instead of $\widetilde{O}(2^{n/3})$, with significantly reduced quantum memory). For general k, we also improve Wagner's time and memory complexities of $O(2^{n/(1+\lfloor \log_2(k) \rfloor)})$ towards $O(2^{n/(2+\lfloor \log_2(k) \rfloor)})$ quantum time and memory.
- If quantum memory is reduced to $O(n)$, we obtain quantum speedups for k-xor up to $k = 7$. In particular, 3-xor search can be performed in time $O(2^{5n/14})$, which is better than the current state of the art for low-qubits quantum collision search.

[4] Notice also that, although all 2^s processors seem to require concurrent accesses to the classical intermediate storage, we can do better if communication overheads between processors are negligible, as advocated by [7]. All processors perform the sequential membership oracle at the same time and in exactly the same manner. Hence, lists elements need only to be sent to one of them and broadcasted to the others with logarithmic overhead.

In particular, contrary to classical algorithms, we have clearly shown that the quantum 3-xor problem is exponentially easier to solve than the quantum collision finding problem, which was not an intuitive conclusion. In contrast, classical time improvements of the 3-xor problem have concerned only logarithmic factors.

Parallelization. Our algorithms for k-xor running with $O(n)$ qubits give rise to efficient parallel versions. In particular, our quantum parallelized 3-xor algorithm attains, using $2^{n/6}$ processors and the same amount of classical storage, a time-hardware product of $\widetilde{O}(2^{5n/12})$, effectively below classical algorithms.

7.1 Implications of Our Results

Our results, in particular for small k, can be used to improve quantumly cryptanalysis results of particular hash constructions or authenticated encryption schemes.

In the following, we give some practical examples:

XHASH and the (R)FSB SHA-3 Candidate: we are able to improve the best GBP attack on the SHA-3 candidate (R)FSB. Referring to [4], the parameters for $\text{FSB}_{\text{length}}$ (the SHA-3 proposal contains five versions of FSB, that is FSB_{160}, FSB_{224}, FSB_{256}, FSB_{384} and FSB_{512}) are given by r – i.e. the output size in bits – and n – i.e. the size of the message to be hashed, where the message is split in ω blocks of size $u = 2^a$. Given the FSB hash function

$$FSB(H, m) := \bigoplus_{i=1}^{k} h_i(m_i),$$

to set up the GBP the idea is to construct $l = 2\log_2(u) - 1$ lists (see [22] for details), where each list is given by the xor-sum of ω/l values $h_i(m_i)$. The complexity of a classical GBP is well estimated $2^{n/(1+\lfloor \log_2(l) \rfloor)}$. A Wagner-type attack (see [4, Table 7]) against FSB_{160} finds a 16-xor between 16 lists which contain elements of size 632. Time and memory, up to smaller constant factors, are given by Wagner's 16×2^{127}. If we are able to query the elements of these lists in superposition, in other words, to produce them quantumly on-the-fly, the quantum time and memory complexities of this operation decrease to 2^{105}. Similar results can be obtained also for the SWIFFT hash function previously recalled.

Authenticated Encryption Schemes - CAESAR: we are able to improve the best forgery attacks on the CAESAR schemes based on XLS. Let us focus on the 128-bit CAESAR candidates Deoxys and KIASU (64 bits of – claimed – security level). The 3-xor problem for XLS in these candidates has the parameter $n = 128$. According to Table 1, the 3-xor can be produced in quantum time $2^{45.7}$ and $2^{18.3}$ classical memory (w.r.t. quantum time $2^{51.2}$ and $2^{25.6}$ classical memory of [19]) or in quantum time $2^{38.4}$ and $2^{25.6}$ quantum memory (w.r.t. quantum time $2^{42.7}$ and $2^{42.7}$ quantum memory of [17]).

Similar results can be obtained for the other applications previously discussed.

7.2 Open Questions

There are still some open questions and further lines of research that would be interesting to investigate. Most of them concern k-xor algorithms with $O(n)$ qubits:

1. Does there exist such a 3-xor algorithm reaching below the quantum collision bound of $O(2^{n/3})$?
2. Is it possible to find a 4-xor quantum algorithm with $O(n)$ quantum memory giving a quantum time speedup over Wagner's method?
3. Still with $O(n)$ quantum memory, can we give a quantum speedup over Wagner's method for a general k?
4. How to adapt our algorithms to the k-sum case? The evolved ones will have a certain overhead that should be computed.

Another question that we believe to be of interest is the fact that classical algorithms for solving the 3-xor problem had a comparable complexity to collision-finding algorithms. With our new quantum algorithm, the 3-xor problem is clearly easier to solve, and might therefore imply that new applications of this problem can appear, as for instance for building bricks of attacks.

Acknowledgements. 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). The results presented here were started during the Flexible Symmetric Cryptography workshop held at the Lorentz Center in Leiden, Netherlands. The authors would like to thank Yunwen Liu and Arnab Roy for preliminary discussions, as well as Elena Kirshanova and Steven Galbraith for all the helpful comments and remarks.

References

1. Aaronson, S.: Quantum lower bound for the collision problem. In: STOC, pp. 635–642. ACM (2002)
2. Aaronson, S., Shi, Y.: Quantum lower bounds for the collision and the element distinctness problems. J. ACM **51**(4), 595–605 (2004)
3. Ambainis, A.: Quantum walk algorithm for element distinctness. SIAM J. Comput. **37**(1), 210–239 (2007)
4. Augot, D., Finiasz, M., Gaborit, P., Manuel, S., Sendrier, N.: SHA-3 proposal: FSB. https://www.rocq.inria.fr/secret/CBCrypto/fsbdoc.pdf
5. Bai, S., Galbraith, S.D., Li, L., Sheffield, D.: Improved exponential-time algorithms for inhomogeneous-sis. Cryptology ePrint Archive, Report 2014/593 (2014). https://eprint.iacr.org/2014/593
6. Banegas, G., Bernstein, D.J.: Low-communication parallel quantum multi-target preimage search. In: Adams, C., Camenisch, J. (eds.) SAC 2017. LNCS, vol. 10719, pp. 325–335. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-72565-9_16
7. Beals, R., et al.: Efficient distributed quantum computing. Proc. R. Soc. A **469**(2153), 20120686 (2013)

8. Bellare, M., Micciancio, D.: A new paradigm for collision-free hashing: incrementality at reduced cost. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 163–192. Springer, Heidelberg (1997). https://doi.org/10.1007/3-540-69053-0_13
9. Belovs, A., Spalek, R.: Adversary lower bound for the k-sum problem. In: Innovations in Theoretical Computer Science, ITCS 2013, pp. 323–328. ACM (2013)
10. Bernstein, D.J.: 2017.10.17: Quantum algorithms to find collisions. The cr.yp.to blog (2017). https://blog.cr.yp.to/20171017-collisions.html
11. Bernstein, D.J., Jeffery, S., Lange, T., Meurer, A.: Quantum algorithms for the subset-sum problem. In: Gaborit, P. (ed.) PQCrypto 2013. LNCS, vol. 7932, pp. 16–33. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38616-9_2
12. Bernstein, D.J., Lange, T., Niederhagen, R., Peters, C., Schwabe, P.: FSBday. In: Roy, B., Sendrier, N. (eds.) INDOCRYPT 2009. LNCS, vol. 5922, pp. 18–38. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-10628-6_2
13. Bernstein, D.J., Lange, T., Niederhagen, R., Peters, C., Schwabe, P.: Implementing wagner's generalized birthday attack against the SHA-3 round-1 candidate FSB. Cryptology ePrint Archive, Report 2009/292 (2009). https://eprint.iacr.org/2009/299
14. Bouillaguet, C., Delaplace, C., Fouque, P.: Revisiting and improving algorithms for the 3XOR problem. IACR Trans. Symmetric Cryptol. **2018**(1), 254–276 (2018). https://doi.org/10.13154/tosc.v2018.i1.254-276
15. Brassard, G., Høyer, P.: An exact quantum polynomial-time algorithm for Simon's problem. In: Fifth Israel Symposium on Theory of Computing and Systems, ISTCS 1997, pp. 12–23. IEEE Computer Society (1997). https://doi.org/10.1109/ISTCS.1997.595153
16. Brassard, G., Hoyer, P., Mosca, M., Tapp, A.: Quantum amplitude amplification and estimation. Contemp. Math. **305**, 53–74 (2002)
17. Brassard, G., HØyer, P., Tapp, A.: Quantum cryptanalysis of hash and claw-free functions. In: Lucchesi, C.L., Moura, A.V. (eds.) LATIN 1998. LNCS, vol. 1380, pp. 163–169. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0054319
18. CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness. http://competitions.cr.yp.to/caesar.html
19. Chailloux, A., Naya-Plasencia, M., Schrottenloher, A.: An efficient quantum collision search algorithm and implications on symmetric cryptography. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10625, pp. 211–240. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70697-9_8
20. Childs, A.M., Eisenberg, J.M.: Quantum algorithms for subset finding. Quantum Inf. Comput. **5**(7), 593–604 (2005)
21. Chose, P., Joux, A., Mitton, M.: Fast correlation attacks: an algorithmic point of view. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 209–221. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-46035-7_14
22. Coron, J.S., Joux, A.: Cryptanalysis of a provably secure cryptographic hash function. Cryptology ePrint Archive, Report 2004/013 (2004). https://eprint.iacr.org/2004/013
23. Datta, N., Nandi, M.: ELmD. https://competitions.cr.yp.to/round1/elmdv10.pdf
24. eSTREAM: the ECRYPT Stream Cipher Project. http://www.ecrypt.eu.org/stream/
25. Even, S., Mansour, Y.: A construction of a cipher from a single pseudorandom permutation. J. Cryptol. **10**(3), 151–161 (1997)
26. Flajolet, P., Odlyzko, A.M.: Random mapping statistics. In: Quisquater, J.-J., Vandewalle, J. (eds.) EUROCRYPT 1989. LNCS, vol. 434, pp. 329–354. Springer, Heidelberg (1990). https://doi.org/10.1007/3-540-46885-4_34

27. Gobioff, H., Nagle, D., Gibson, G.: Integrity and performance in network attached storage. In: Polychronopoulos, C., Fukuda, K.J.A., Tomita, S. (eds.) ISHPC 1999. LNCS, vol. 1615, pp. 244–256. Springer, Heidelberg (1999). https://doi.org/10.1007/BFb0094926
28. Bok-Min, G., Siddiqi, M.U., Hean-Teik, C.: Incremental hash function based on pair chaining & modular arithmetic combining. In: Rangan, C.P., Ding, C. (eds.) INDOCRYPT 2001. LNCS, vol. 2247. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45311-3_5
29. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing 1996, pp. 212–219. ACM (1996). https://doi.org/10.1145/237814.237866
30. Grover, L.K., Rudolph, T.: How significant are the known collision and element distinctness quantum algorithms? Quantum Inf. Comput. **4**(3), 201–206 (2004). http://portal.acm.org/citation.cfm?id=2011622
31. Hell, M., Johansson, T., Meier, W.: Grain - a stream cipher for constrained environments. http://www.ecrypt.eu.org/stream/p3ciphers/grain/Grain_p3.pdf
32. Jean, J., Nikolić, I., Peyrin, T.: ELmD. https://competitions.cr.yp.to/round2/joltikv13.pdf
33. Jean, J., Nikolić, I., Peyrin, T.: KIASU. https://competitions.cr.yp.to/round1/kiasuv1.pdf
34. Kaplan, M., Leurent, G., Leverrier, A., Naya-Plasencia, M.: Breaking symmetric cryptosystems using quantum period finding. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016. LNCS, vol. 9815, pp. 207–237. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53008-5_8
35. Kirchner, P.: Improved generalized birthday attack. Cryptology ePrint Archive, Report 2011/377 (2011). https://eprint.iacr.org/2011/377
36. Kuwakado, H., Morii, M.: Quantum distinguisher between the 3-round Feistel cipher and the random permutation. In: IEEE International Symposium on Information Theory, ISIT 2010, pp. 2682–2685. IEEE (2010)
37. Kuwakado, H., Morii, M.: Security on the quantum-type Even-Mansour cipher. In: Proceedings of the International Symposium on Information Theory and its Applications, ISITA 2012. IEEE (2012)
38. Leander, G., May, A.: Grover meets simon – quantumly attacking the FX-construction. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10625, pp. 161–178. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70697-9_6
39. Lipton, R.J., Regan, K.W.: Quantum Algorithms via Linear Algebra: A Primer. The MIT Press, Cambridge (2014)
40. Magniez, F., Nayak, A., Roland, J., Santha, M.: Search via quantum walk. SIAM J. Comput. **40**(1), 142–164 (2011)
41. Mermin, N.D.: Quantum Computer Science: An Introduction. Cambridge University Press, New York (2007)
42. Micciancio, D., Arbitman, Y., Dogon, G., Lyubashevsky, V., Peikert, C., Rosen, A.: SWIFFT. https://www.eecs.harvard.edu/alon/PAPERS/lattices/swifftx.pdf
43. Minder, L., Sinclair, A.: The extended k-tree algorithm. J. Cryptol. **25**(2), 349–382 (2012)
44. Nandi, M.: XLS is not a strong pseudorandom permutation. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014. LNCS, vol. 8873, pp. 478–490. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45611-8_25
45. Nandi, M.: Revisiting security claims of XLS and COPA. Cryptology ePrint Archive, Report 2015/444 (2015). https://eprint.iacr.org/2015/444

46. Naya-Plasencia, M.: How to improve rebound attacks. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 188–205. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22792-9_11
47. Niebuhr, R., Cayrel, P.L., Buchmann, J.: Improving the efficiency of Generalized Birthday Attacks against certain structured cryptosystems. In: Workshop on Coding and Cryptography, WCC 2011, pp. 163–172 (2011)
48. Nikolić, I., Sasaki, Y.: Refinements of the k-tree algorithm for the generalized birthday problem. In: Iwata, T., Cheon, J.H. (eds.) ASIACRYPT 2015. LNCS, vol. 9453, pp. 683–703. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48800-3_28
49. Ristenpart, T., Rogaway, P.: How to enrich the message space of a cipher. In: Biryukov, A. (ed.) FSE 2007. LNCS, vol. 4593, pp. 101–118. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74619-5_7
50. Shor, P.W.: Algorithms for quantum computation: discrete logarithms and factoring. In: Proceedings of the 35th Annual Symposium on Foundations of Computer Science, SFCS 1994, pp. 124–134. IEEE Computer Society (1994)
51. Siegenthaler, T.: Correlation-immunity of nonlinear combining functions for cryptographic applications. IEEE Trans. Inf. Theory **30**(5), 776–780 (1984)
52. Siegenthaler, T.: Decrypting a class of stream ciphers using ciphertext only. IEEE Trans. Comput. **34**(1), 81–85 (1985)
53. Simon, D.R.: On the power of quantum computation. SIAM J. Comput. **26**(5), 1474–1483 (1997)
54. Wagner, D.: A generalized birthday problem. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 288–304. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45708-9_19
55. Wang, L.: SHELL. https://competitions.cr.yp.to/round2/shellv20.pdf
56. Zhandry, M.: A note on the quantum collision and set equality problems. Quantum Info. Comput. **15**(7–8), 557–567 (2015). http://dl.acm.org/citation.cfm?id=2871411.2871413