Quantum Procedures for Nested Search Problems with Applications in Cryptanalysis

André Schrottenloher[1⋆] and Marc Stevens[2]

Univ. Rennes, Inria, IRISA, Rennes, France
firstname.lastname@.inria.fr
Cryptology Group, CWI, Amsterdam, The Netherlands
firstname.lastname@cwi.nl

Abstract. In this paper we study search problems that arise very often in cryptanalysis: nested search problems, where each search layer has known degrees of freedom and/or constraints. A generic quantum solution for such problems consists of nesting Grover's quantum search algorithm or amplitude amplification (QAA) by Brassard et al., obtaining up to a square-root speedup on classical algorithms. However, the analysis of nested Grover or QAA is complex and introduces technicalities that in previous works are handled in a case-by-case manner. Moreover, straightforward nesting introduces an overhead factor of $(\pi/2)^\ell$ in the complexity (for ℓ layers).

In this paper, we aim to remedy both these issues and introduce a generic framework and tools to transform a classical nested search into a quantum procedure. It improves the state-of-the-art in three ways: 1) our framework results in quantum procedures that are significantly simpler to describe and analyze; 2) it reduces the overhead factor from $(\pi/2)^\ell$ to $\sqrt{\ell}$; 3) it is simpler to apply and optimize, without needing manual quantum analysis. We give a generic complexity formula that improves the state-of-the-art both for concrete instantiations and in the asymptotic setting. For concrete instances, we show that numerical optimizations enable further improvements over this formula, which results in a further decrease in the gap to an exact quadratic speedup.

We demonstrate our framework with applications in cryptanalysis and improve the complexity of quantum attacks on reduced-round AES.

Keywords: Quantum search · Nested search · Quantum cryptanalysis · Amplitude amplification · Symmetric cryptanalysis.

1 Introduction

The potential advent of large-scale quantum computing devices has prompted the cryptographic community to evaluate the *quantum security* of cryptographic schemes; that is, against an adversary capable of quantum computations. Among

⋆ Part of this work was done while the author was at CWI.

the foundational results of *quantum cryptanalysis*, Shor's period-finding algorithm [37] showed that public-key schemes based on the classical hardness of the factoring and Discrete Logarithm problems would be irremediably broken in the quantum setting. This has led to a massive effort aiming at replacing these schemes by quantum-secure ones, which is embodied by the NIST post-quantum standardization process [2] which selected in 2022 its first set of future standards.

Quantum Search. The second most well-known quantum algorithm impacting cryptanalysis is Grover's quantum search [21], generalized into the framework of quantum amplitude amplification (QAA) [12]. Whereas Shor's algorithm applies only to a handful of problems with a specific algebraic structure, Grover's algorithm generically speeds up any *black-box* search procedure. By *black-box*, we mean that the problem is entirely defined by means of an algorithm that *samples* at random from a given search space, and an algorithm that *tests* an element of this search space, mapping it to $\{0,1\}$. Classical black-box search consists in iterating both these algorithms until a preimage of 1 is found. Quantum search solves the same problem with an asymptotic square-root reduction in the average number of iterations. In particular, it accelerates the exhaustive search of a secret key of length $|K|$ from $\mathcal{O}\left(2^{|K|}\right)$ trial encryptions to $\mathcal{O}\left(2^{|K|/2}\right)$ applications of a quantum encryption circuit.

Because of its generic nature, quantum search is very often used as a building block of more complex quantum cryptanalysis procedures. In asymmetric cryptography, generic decoding algorithms based on information set decoding [7,24] gain most of their quantum speedup from a Grover search. It is also a subroutine of sieving algorithms for the Shortest Vector Problem in lattices [32,31,28]. In most of these cases, the quadratic speedup of quantum search seems so far the best achievable. The design of these algorithms is thus partially guided by the idea to rephrase classical techniques into black-box searches.

In symmetric cryptanalysis, there are examples of dedicated attacks on constructions exploiting specific algebraic structures (see e.g. [30,25]). Nevertheless, largely current research focuses on quadratic speedups obtained with quantum search. The goal of these quantum procedures is to have a better complexity than the generic attack based on Grover search (or quantum collision search in the case of [22]), in the same way that classical attacks are designed to improve over the generic exhaustive search. Many classical cryptanalytic techniques have been shown to admit quantum variants, e.g., differential and linear cryptanalysis [26], rebound attacks on hash functions [22], and so on, often by solving the underlying classical nested search problem by nesting Grover searches.

Nested Search Problems. Informally, an exhaustive search problem explores a space of *choices* C_1; choices are sampled at random and evaluated using a *filtering* function, which maps $c_1 \in C_1$ to $\{0,1\}$, where "0" indicates a bad choice to be discarded, and "1" indicates a valid choice. This situation is pictured in Figure 1, where we use $c_{i,j}$ for $j = 1, 2, \ldots, |C_i|$ to denote the individual members of the set C_i. It corresponds to the exploration of a tree with one level.

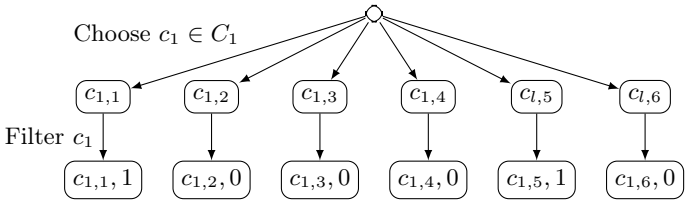

Fig. 1. Exhaustive search with a single level.

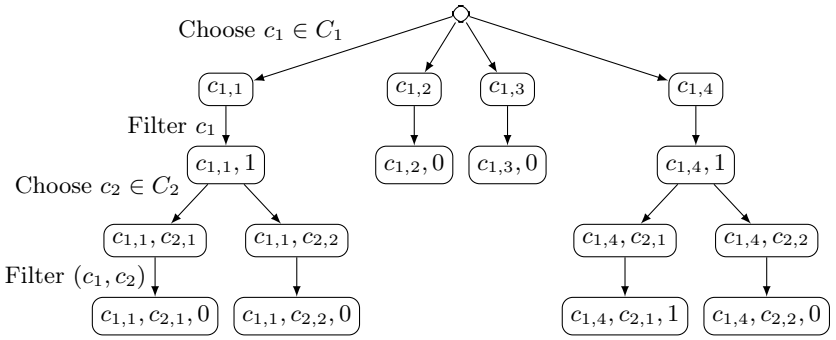

Fig. 2. Nested search with two levels.

Recursively, we can also allow the filtering function to explore another search space C_2 internally. Typically, the first filter will reduce the possibilities for $c_1 \in C_1$, so that we do not have to check all the couples $(c_1, c_2) \in C_1 \times C_2$ for the second filter. This corresponds to the exploration of a tree with two levels (Figure 2). The expected complexity of the search procedure depends on the expected number of nodes on each level that need to be explored before finding a solution $(c_1, c_2, 1)$ at the last level.

If we have quantum algorithms to implement the "choosing" and "filtering" steps, then a search such as in Figure 1 can be turned into a quantum search. While both can be "basic" algorithms such as picking a candidate secret key, they can also embed other quantum searches. This leads to the folklore fact that any classical *nested* search admits a corresponding *nested* quantum search, generally built over the QAA framework, which reaches up to a quadratic speedup.

Computing the Complexities. Both in asymmetric and symmetric cryptanalysis, we need precise complexity estimates, either of generic algorithms applied to certain parameterizations of public-key schemes, or attacks targeting specific symmetric designs. However, the current state of affairs is that the analysis of nested quantum searches raises the following problems:

Overhead factor per level: Applying QAA naively results in a multiplicative overhead factor of $\pi/2$ per nesting level, see e.g. [17]. Thus, the quantum time complexity of a nested search procedure deviates from the square root

of the classical complexity. With 4 or 5 nesting levels, this deviation may become significant with respect to the total complexity: this happens for the quantum key-recovery attack on 8-round AES-256 of [11].

Probability of error: Most of the time, QAA is erroneous: it produces only a noisy approximation of the exact uniform superposition of the solutions. When composing the searches, these errors should not pile up too much.

Fixed number of iterates: In a classical nested search like Figure 2, the filtering probability at the second level (c_1, c_2) can depend on the current choice c_1. This is no concern for a classical time complexity analysis, because we only need to bound the total number of nodes explored at each level. But QAA runs in superposition (e.g., of the choices c_1) and uses a fixed number of iterates, which complicates the analysis.

Previous works overcome these problems on a case by case basis. They often err on the safe side by reducing the error probabilities to negligible amounts. This usually leads to an overestimate of the complexities, and also, to unnecessarily technical analyses.

Contributions and Organization. In this paper, we provide a generic framework and tools to transform nested search procedures into quantum procedures. I.e., we guarantee, by simply rewriting a classical attack in our framework described below, the existence of a corresponding quantum algorithm with given complexity and success probability. We provide a generic formula and, for concrete instances, a numerical-optimization-based tool giving slightly better results.

The search problem that we consider is formally detailed in Section 3, and depicted in Figure 3. It corresponds to the exploration of a search tree with multiple levels. Each node in the tree corresponds to certain choices and includes a certain internal state. Each search layer i comprises three sub-steps: (1) select a random choice $c_i \in C_i$; (2) apply a *filtering* function A_i that decides whether the current choices $(c_1, \ldots, c_i)$ form a valid subsolution or not; (3) post-process the current internal state using a function D_i. At the final layer ℓ, A_ℓ decides if the whole path is good, thus yielding a solution to the search problem. We assume that the tree contains only one solution.

In Section 5, we present a quantum procedure for this problem, which performs a *search with backtracking*. It corresponds to a depth-first exploration of the tree, which eliminates whole subtrees and returns to previous states. Intuitively, this strategy performs better when there are many choices to make, with successive filter and post-processing functions of decreasing complexities. Our generic complexity analysis handles the three problems identified earlier, and in particular, reduces the quantum overhead factor from $(\pi/2)^\ell$ to $\sqrt{\ell}$.

Before this, in Section 4, we first present a quantum procedure for the restricted case of a single choice, i.e., a *search with early aborts* (as depicted in Figure 4). Both our quantum procedures are based on nested QAAs and although their analyses are similar, they correspond to very different strategies.

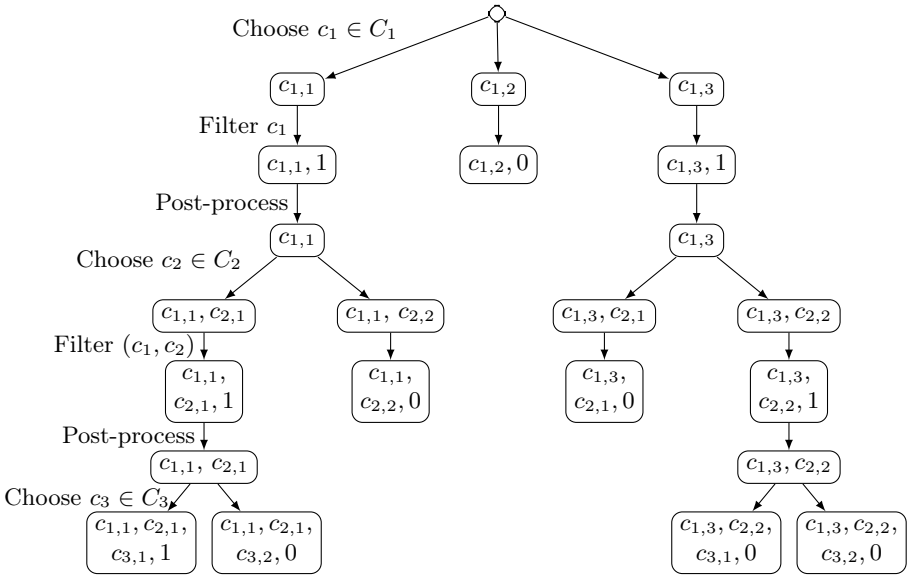

Fig. 3. Example of the generic search problem that we want to solve: we look for $(c_1, c_2, c_3) \in C_1 \times C_2 \times C_3$ which evaluates to 1 at the last step.

In Section 6, we demonstrate our framework on two key-recovery attacks on reduced-round AES, taken from [11]. We externalize entirely the quantum complexity analysis using our framework, which results in a gain up to a factor 2^4 in time complexity.

Further applications are given in Appendix. We study *search with independent tests*, i.e., searching for an x satisfying several independent testing functions $f_i(x)$, both asymptotically and with exact values. Finally, we deduce from our framework a simple algorithm for variable-time amplitude amplification. The code of our optimizations is available at: https://github.com/AndreSchrottenloher/quantum-search.

Related work. Over the time, there has been a few attempts at formalizing the correspondence between classical and quantum nested search algorithms. This is the case of the *filter* framework of [11]. Its goal is only to guide the design of such algorithms. Although they have a very different wording, our backtracking framework contains the *filter* framework. But we also provide a generic complexity analysis, in contrast to [11]. In [17], the authors studied generically the case of impossible differential attacks, which corresponds to the setting of Figure 3 without filtering layers. Contrary to us, they use Exact Amplitude Amplification, which gives a success probability 1, but yields the "naive" multiplicative overhead factor $(\pi/2)^\ell$. Also in the context of cryptanalysis, precise complexity analyses were done for a fixed number of nested searches, e.g., $\ell = 2$ for the *search with two oracles* in [16,27].

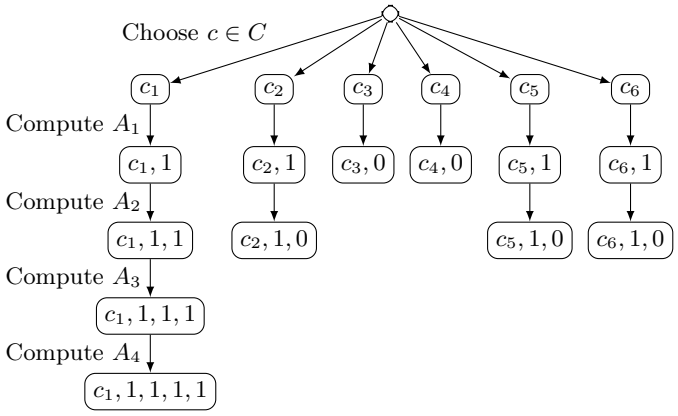

Fig. 4. Search with early aborts, with a single choice $c \in C$, and 4 successive filters A_1, A_2, A_3, A_4.

Montanaro [33] designed a generic quantum tree search algorithm, but its setting is different from ours. Indeed, the main advantage of Montanaro's algorithm is that the tree structure can be discovered on the fly, and does not need to be known entirely in advance. This is the case of the trees explored in lattice enumeration, where this algorithm was applied in [6]. In contrast, our framework is based on nested QAAs and requires knowledge of the filtering and success probabilities of each step. Conversely, the main disadvantage of Montanaro's tree search is that it does not amortize the cost of the different filtering algorithms, while this situation is very common in cryptanalysis: some of them, which cost less, are computed more often (e.g., the last search layer in the tree); others, which cost more, are computed less often (e.g., the first search layer in the tree).

Ambainis [3,4] studied the problem of *variable-time* amplitude amplification, which can be seen as a special case of a search with early aborts. In this problem, one does a search for a good element among N of them, when being "good" is evaluated in time t_i for element number i. Ambainis showed that the solution could be found in time $\widetilde{\mathcal{O}}\left(\sqrt{t_1^2 + \ldots + t_N^2}\right)$ even when the t_i are unknown. However, his solution relied on Quantum Amplitude Estimation. Our framework of search with early aborts yields a solution for this problem, with the same asymptotic complexity as Ambainis'. Our algorithm is much simpler since it contains only nested QAAs. Ambainis also showed in this context that the naive factor $(\pi/2)^\ell$ with ℓ levels of nesting could be reduced to $\mathcal{O}(\sqrt{\ell})$, but his complexity analysis was only asymptotic.

Concurrently and independently, Ambainis et al. [5] proposed a new algorithm for variable-time QAA with essentially the same algorithmic structure as ours. Their complexity analysis improves upon the polynomial factor in the $\widetilde{\mathcal{O}}$ w.r.t. Ambainis' and ours, but the improvement is specific to this special case problem and is unlikely to generalize to our settings.

2 Preliminaries

In this section, we first cover some preliminaries of quantum algorithms (complexities, memory models, QAA). The formal definition of a nested search problem will be given in Section 3. The analysis of quantum search requires some bounds for sin and arcsin. We use the following standard ones.

$$\forall x \geq 0: \quad x\left(1-x^2/6\right) \leq \sin x \leq x, \text{ and } x^2\left(1-x^2/3\right) \leq \sin^2 x \leq x^2 \quad (1)$$

$$\forall 0 \leq x \leq 1: \quad x \leq \arcsin x \leq (\pi/2)x \implies x^2 \leq \arcsin^2 x \leq (\pi^2/4)x^2 \quad (2)$$

Quantum Algorithms. We refer to [34] for an introduction to quantum computing and the quantum circuit model. We assume knowledge of the notion of qubits, the ket notation $|\cdot\rangle$, and basic quantum gates (e.g., the Hadamard gate H). In this paper, we will consider the Clifford+T gate set, which is often used for counting gates in quantum algorithms (see e.g. [23] or [10]).

We use $G(\mathcal{U})$ to denote the gate count of a quantum circuit $\mathcal{U}$, which is the main metric we are interested in. We use $S(\mathcal{U})$ for the width of the circuit, i.e., the number of qubits on which it acts, including ancilla qubits. Any quantum algorithm $\mathcal{U}$ without measurement is reversible. The reverse $\mathcal{U}^\dagger$ is the "uncomputation" of $\mathcal{U}$. Both admit the same gate count.

In symmetric cryptanalysis, time complexity estimates are often expressed relatively to the cost of a cryptographic function. For example, the exhaustive key search of a 128-bit block cipher is estimated as 2^{128} *evaluations of the cipher*. This principle remains true in quantum cryptanalysis. We can consider the *evaluation of a quantum circuit for the cipher* to be the benchmarking operation, or alternatively, as in [11], we can single out some costly component of the cipher (like an S-Box) and consider only the number of evaluations of this component.

Memory Models. Given an array of M qubit registers, any *fixed location* can be queried in polynomial time in the circuit model, as it amounts only to apply quantum gates to pairs of qubits: $|x\rangle|y_1,\ldots,y_M\rangle \xmapsto{\mathsf{Access}_i} |x\oplus y_i\rangle|y_1,\ldots,y_M\rangle$. However, accessing a *variable* memory location (*random access*) can a priori be done only by performing a sequence of $\widetilde{\mathcal{O}}(M)$ such queries, which gives a cost $\widetilde{\mathcal{O}}(M)$ for the operation: $|x\rangle|y_1,\ldots,y_M\rangle|i\rangle \xmapsto{\mathsf{Access}} |x\oplus y_i\rangle|y_1,\ldots,y_M\rangle|i\rangle$. In the *QRAQM*[1] *model*, we assume that the operation $\xmapsto{\mathsf{Access}}$ can be performed in polynomial time. In practice, we will consider its cost to be comparable to a block cipher or an S-Box evaluation.

A quantum algorithm can also access a large *classical memory* with M registers $y_1,\ldots,y_M$. The *sequential access* in time $\widetilde{\mathcal{O}}(M)$ is still allowed by the quantum circuit model, since it amounts to control at runtime the sequence of applied gates: $|x\rangle|i\rangle \xmapsto{\mathsf{CAccess}(y_1,\ldots,y_M)} |x\oplus y_i\rangle|i\rangle$. In the *QRACM*[2] *model*, we assume that the operation $\mathsf{CAccess}(y_1,\ldots,y_M)$ can be performed in polynomial

[1] *Quantum random-access quantum memory*, following the terminology from [29].
[2] *Quantum random-access classical memory.*

time. In practice, we will consider its cost to be comparable to a block cipher call or an S-Box evaluation.

In the following, we will separately consider *classical* and *quantum* memory costs and specify if we use QRAQM / QRACM (otherwise we are in the "plain" quantum circuit model).

Probability of Success. We consider quantum algorithms with varying probabilities of success, and all our results will be statements of the form:

> there exists a quantum algorithm with (exact or average) time complexity T, (classical and/or quantum) memory complexity M, and success probability $\geq p$.

Amplitude Amplification. Quantum amplitude amplification [12], abbreviated QAA in this paper, is a generalization of Grover search which allows to increase the success probability of any measurement-free quantum algorithm by iterating it. Let $\mathcal{U}$ be a quantum circuit such that

$$\mathcal{U}|0\rangle = \sqrt{p}\,|\psi_G\rangle\,|1\rangle + \sqrt{1-p}\,|\psi_B\rangle\,|0\rangle \quad , \tag{3}$$

where p is the success probability of $\mathcal{U}$, $|\psi_G\rangle$ and $|\psi_B\rangle$ are two orthogonal quantum states corresponding to the *good outcomes* and *bad outcomes* of $\mathcal{U}$ respectively. Our goal is to produce a state close to $|\psi_G\rangle$. Let O_0 be the *inversion around zero* operator, which flips the phase of the basis vector $|0\rangle$: $O_0 = I - 2\,|0\rangle\langle 0|$; and O be the operator which flips the phase of all basis vectors $|x, b\rangle$ such that $b = 1$. The QAA computes a sequence of states $|\psi_i\rangle, 0 \leq i \leq t$, defined by the following iterative process:

1. Start from $|\psi_0\rangle = \mathcal{U}|0\rangle$
2. For $i = 1$ to t:
3. $\quad |\psi_{i+1}\rangle = -\mathcal{U}O_0\mathcal{U}^\dagger O\,|\psi_i\rangle$

Let $\theta = \arcsin(\sqrt{p})$. As shown in [12], at each iteration, the amplitude of good outcomes increases as follows:

$$|\psi_i\rangle = \sin((2i+1)\theta)\,|\psi_G\rangle\,|1\rangle + \cos((2i+1)\theta)\,|\psi_B\rangle\,|0\rangle \quad . \tag{4}$$

This implies that after $t = \left\lfloor \pi/(4\arcsin\sqrt{p}) \right\rfloor$ iterations, the value $(2t+1)\theta$ approaches $\frac{\pi}{2}$. The success probability is at least[3] $1-p$.

In quantum search, and in the algorithms presented in this paper, the only computational overhead with respect to the iterations of $\mathcal{U}$ comes from the implementation of O_0 and O. We implement O_0 naively using an n-bit Toffoli gate (a better count for O_0 is also given in [20], but not with a generic formula).

Lemma 1. *The inversion around zero on n qubits can be implemented using $n-1$ ancilla qubits and $44n-39$ Clifford+T gates. The O operator can be implemented using 1 ancilla qubit and 3 Clifford gates.*

[3] Though there exists an *exact* variant of QAA (also given in [12]), which increases this probability to 1 if we know p exactly in advance, it is not particularly helpful for us since we typically prefer to under-amplify the QAAs.

Nesting Many QAAs. Since each iterate contains two calls to $\mathcal{U}$ (one reversed), a QAA needs approximately $\pi/(2\sqrt{p})$ calls to succeed with probability close to 1, thus with overhead factor $\pi/2$ compared to $1/\sqrt{p}$. With ℓ nested QAAs ($\mathcal{U}$ calls a QAA, etc.), this accumulates into an overhead factor $(\pi/2)^{\ell}$.

Recall that $\sin x \simeq x$ when x is small, so the probability of success of the QAA initially grows almost as $(2i+1)^2 p$: it increases quadratically. The $\frac{\pi}{2}$ factor only appears if we want to make it close to 1. Thus, and perhaps counter-intuitively, to avoid piling up these factors we must keep the success probability of the QAAs artificially low. This fact is well-known (see Lemma 9 in [1]) but rarely taken into account in cryptanalysis.

Unknown Success Probability and "Overcooking". In this paper, we will often encounter the situation where we only have a lower bound $p_{\min}$ on p, and we want to find a solution with constant success probability. This can be done in expected time $\mathcal{O}(1/\sqrt{p_{\min}})$, by Theorem 3 in [12]. We settle for a simple method which consists in running a QAA with a random number of iterates.

Lemma 2. *Let $M = \lceil 1.21/\sqrt{p_{\min}} \rceil$. Let $\mathcal{U}$ be defined as in Equation 3, operating on n qubits. There exists a quantum procedure that makes* on average $2M+2$ *calls to $\mathcal{U}$ and $\mathcal{U}^\dagger$ and contains $(M-1) \times (22n-18)$ Clifford+T gates, and returns a good outcome with probability $\geq \frac{1}{2}$.*

Proof. We run the following procedure:

1. Do twice: Execute $\mathcal{U}$ and measure its output flag. If it is "1" then exit and return the measurement result.
2. Do twice: Choose an integer $i \leq M-1$, where $M = \lceil 1.21/\sqrt{p_{\min}} \rceil$, uniformly at random, and execute a QAA with i iterates. If a "1" flag is measured then exit and return the measurement result.

First of all, the average number of calls to $\mathcal{U}$ and $\mathcal{U}^\dagger$ is:

$$\frac{2}{M} \sum_{i=0}^{M-1} (2 + 2 \cdot i) = 4 + 4 \cdot \frac{(M-1)M}{2} \frac{1}{M} = 4 + 2(M-1) = 2M + 2 \ .$$

And the average number of additional Clifford+T gates is: $\frac{M-1}{2} \times (44n - 36)$.

Next, we compute the probability that at least one of the substeps obtains a good outcome. Using standard trigonometric formulas, we obtain:

$$\begin{aligned} \frac{1}{M} \sum_{i=0}^{M-1} \sin^2\left((2i+1)\theta\right) &= \frac{1}{2} - \frac{1}{2M} \frac{\sin(2M\theta)\cos(2M\theta)}{\sin(2\theta)} \\ &\geq \frac{1}{2} - \frac{1}{4M\sin\theta\cos\theta} = \frac{1}{2} - \frac{1}{4M\sqrt{p(1-p)}} \ . \end{aligned}$$

Since we only run step 2 if step 1 fails, which occurs with probability $1-p$ by the structure of $\mathcal{U}$, the whole operation fails (i.e., measures a bad outcome) with probability at most:

$$(1-p)\left(1/2 + 1/(4M\sqrt{p(1-p)})\right) \leq 1/2 + 1/(4M\sqrt{p_{\min}}) \ .$$

Next, we combine two such procedures. By choosing $M = \lceil 1.21/\sqrt{p_{\min}} \rceil$, we can upper bound the failure probability as: $\left(1/2 + 1/(4 \cdot 1.21)\right)^2 \leq 1/2$. □

3 Nested Search Problems

In this section, we define the nested search problem that we want to solve, and introduce all necessary notations for the building blocks of our algorithms. We use 0_r and 1_r to denote bit-strings of zeroes (resp. ones) of length r.

3.1 Preliminaries

Choice Set. A "choice set" is just a set C identified with a set of bit-strings. We consider its size $|C|$ to be a power of 2. If not, we increase its size artificially by adding choices that always lead to a bad outcome. When choosing from C classically, we select $c \in C$ uniformly at random. The corresponding quantum algorithm is a Hadamard transform H that maps $\left|0_{\log_2 |C|}\right\rangle$ to $\frac{1}{\sqrt{|C|}} \sum_{c \in C} |c\rangle$.

Basic Algorithms. We consider a quantum algorithm $\mathcal{A}$, paired with a (classical) function A, acting *reversibly in place* on some *workspace* $W = W_{RO} \times W_V$, where W_{RO} is the *read-only* part of the workspace and W_V is the modifiable *variable* part. To W corresponds a Hilbert space $\mathcal{W} = \mathcal{W}_{RO} \otimes \mathcal{W}_V$, spanned by a basis of vectors of size $2^{|W|}$. Such a pair $(\mathcal{A}, A)$ acts as follows:

$$\begin{cases} A : W \to W \\ \quad x \mapsto A(x) \end{cases} \qquad \begin{cases} \mathcal{A} : \mathcal{W} \to \mathcal{W} \\ \quad |x\rangle \mapsto |A(x)\rangle \end{cases} , \tag{5}$$

where for all $x = (x_{ro}, x_v)$, if we write $A(x) = (x'_{ro}, x'_v)$, we have $x'_{ro} = x_{ro}$. In this definition, A is the function that we are implementing, and $\mathcal{A}$ is a quantum circuit that implements it. For example, a boolean function $f : X \to \{0,1\}$ can be realized by a classical algorithm that acts on $W = X \times \{0,1\}$ and maps (x,b) to $(x, f(x) \oplus b)$; the corresponding quantum algorithm is the standard oracle mapping $|x\rangle |b\rangle$ to $|x\rangle |b \oplus f(x)\rangle$. In that case X is the read-only part of the workspace. It typically contains the results of previous computations, previous choices, and some large memory storage.

Reversibility. The quantum algorithm $\mathcal{A}$ can be derived from A if A is efficiently implementable, though it does not need to be: e.g., $\mathcal{A}$ could be based on Simon's algorithm. If A is implementable as a classical reversible circuit using $G(A)$ gates, then we can implement $\mathcal{A}$ as a quantum circuit using the same number of gates.

Making a classical algorithm reversible is not always easy. A naive way, which we will use by default, is to track all intermediate computations into the workspace (at the expense of the memory complexity only).

Remark 1. By Equation 5, $\mathcal{A}$ implements *exactly* the function $x \mapsto A(x)$, i.e., $\forall x, \mathcal{A}(|x\rangle) = |A(x)\rangle$. This can be relaxed into an implementation up to some uniform error ε: $\forall x, \|\mathcal{A}(|x\rangle) - |A(x)\rangle\| \leq \varepsilon$. Then, assuming that $\mathcal{A}$ is not called more than $\mathcal{O}(1/\varepsilon)$ times, a standard hybrid argument ensures that the final state of the algorithm deviates from the case of a perfect $\mathcal{A}$ by less than a constant error; and so, the probability of success remains constant. This is used e.g. in [9]. The situation is more favorable if the errors raised by $\mathcal{A}$ can be detected, because in that case, whether $\mathcal{A}$ errs or not defines an early-abort layer, and this can go into our search framework.

3.2 Definition of the Problem and Parameters

Let $\ell \geq 1$ be an integer. Let $C_1, \ldots, C_\ell$ be *choice* sets, which are sets of bit-strings of respective lengths $n_1, \ldots, n_\ell$, totaling $n = n_1 + \ldots + n_\ell$. Let $W_1, \ldots, W_\ell$ be sets of bit-strings of respective lengths $w_1, \ldots, w_\ell$, totaling $w = w_1 + \ldots + w_\ell$. Let $F_1 = \ldots = F_\ell = \{0, 1\}$ be *flags*. We define the *workspace* of our algorithms as $F \times C \times W$, where $C = C_1 \times \cdots \times C_\ell$, $W = W_1 \times \cdots \times W_\ell$ and $F = F_1 \times \cdots \times F_\ell$. We use a corresponding Hilbert space $\mathcal{F} \otimes \mathcal{C} \otimes \mathcal{W}$ spanned by all workspace states. Since we represent our algorithms as circuits, we think of a workspace element as a set of *registers*: *flag registers*, *choice registers* and *work registers*. By abuse of notation, we name these registers like the sets in which they take their values: F_i, C_i, W_i.

We consider ℓ algorithms $A_1, \ldots, A_\ell$ (filtering functions) and ℓ algorithms $D_1, \ldots, D_\ell$ (post-processing functions). All of them act on the workspace $F \times C \times W$ with the following restrictions:

- A_i can only modify the work registers $W_i, \ldots, W_\ell$ and the i-th flag F_i; i.e., it does not change the work registers of the previous steps. Its result depends only on the current choices and workspace states;
- D_i can only modify the work registers $W_i, \ldots, W_\ell$;
- For $i \geq 2$, if the $(i-1)$-th flag is 0, both A_i and D_i act as the identity. This means that their respective quantum implementations are *controlled* on the $(i-1)$-th flag.

Remark 2. Ancilla qubits used in the quantum implementation of the A_i and D_i are counted in the workspace.

Remark 3. While the separation of D_i from A_i might seem anecdotal, it will help to save computation time if D_i is noticeably more expensive than A_i.

Remark 4. If all A_i and D_i have the same complexity, there is no need for nesting, and the analysis of this paper will not be of particular help (one would do a single search in the space $C_1 \times \ldots \times C_\ell$). However, they typically have different complexities, and the more costly ones must be performed less times.

Since our problem can be viewed as a tree search, a tuple of choices $(c_1, \ldots, c_i)$ that passes the first $i-1$ filtering steps is called a *path*. The i-th flag indicates if the path $(c_1, \ldots, c_i)$ passes the i-th filter, and the ℓ-th flag indicates if it is a solution. We will use the notation $|_{\text{flag } i}$ for extracting the value of the i-th flag. We assume that a single solution $(c_1^g, \ldots, c_\ell^g)$ exists. Our search problem can thus be formulated as:

$$\begin{aligned}&\text{Find the path } (c_1^g, \ldots, c_\ell^g) \in C_1 \times \cdots \times C_\ell \text{ such that:}\\ &(D_\ell \circ A_\ell) \circ \cdots \circ (D_1 \circ A_1)(0_\ell, c_1^g, \ldots, c_\ell^g, 0_w)|_{\text{flag } \ell} = 1.\end{aligned}$$

For all i, let $\mathcal{A}_i$ and $\mathcal{D}_i$ be quantum implementations of A_i and D_i, of gate complexities $G(\mathcal{A}_i)$ and $G(\mathcal{D}_i)$.

Filtering and Success Probabilities. We define the *filtering probability* $\alpha_i'^2$ as the probability that, starting from the good subpath $(c_1^g, \ldots, c_{i-1}^g)$, a uniformly random c_i passes the i-th filter:

$$\alpha_i'^2 = \Pr_{c_i \xleftarrow{\$} C_i}\left((D_i \circ A_i) \circ \cdots \circ (D_1 \circ A_1)(0_\ell, c_1^g, \ldots, c_{i-1}^g, c_i, 0_w)|_{\text{flag } i} = 1\right) \ . \quad (6)$$

We define the *success probability* α_i^2 as the probability that, after passing the filtering at step i, a subpath $(c_1^g, \ldots, c_{i-1}^g, c_i)$ is actually the good subpath $(c_1^g, \ldots, c_i^g)$. By our assumption that there is a single good path:

$$\forall i, \alpha_i^2 \alpha_i'^2 = \frac{1}{|C_i|} \implies \forall i, \alpha_i^2 = \frac{2^{-n_i}}{\alpha_i'^2} \ . \quad (7)$$

At the last step, the filtering decides already if the path is a solution. Thus, we have $\alpha_\ell' = 1$, and the last post-processing step D_ℓ is the identity.

Remark 5. The assumption of having exactly one solution path, which is quite common in cryptanalytic algorithms, allows us to reason only with respect to this path: in particular, we only need to bound correctly the filtering $(\alpha_i'^2)$ and success (α_i^2) probabilities *along this path*. The behavior on the other paths turns out to be irrelevant.

If there is more than one solution, our analysis may still apply, but only if the solutions have locally the same parameters α_i, α_i'. This happens for example if the tree is regular and if there is no filtering ($\alpha_i' = 1$). Otherwise we cannot guarantee that the solution sub-paths are correctly amplified at each level, and our analysis breaks down.

3.3 Simplified Case of Search with Early Aborts

Search with early aborts (Figure 4) can be seen as a special case of our generic tree search, in which a single choice is made at the beginning of the search. This simplifies considerably the setting and parameterization of the problem.

Let $\ell \geq 1$ be an integer. Let C be a choice set of bit-length n. Let W be a set of bit-strings of length w. Let $F = F_1 \times \cdots \times F_\ell$ be the set of flags. The *workspace*

is defined as $F \times C \times W$ with a corresponding Hilbert space $\mathcal{F} \otimes \mathcal{C} \otimes \mathcal{W}$. We use the same naming conventions as in the more general case.

We consider ℓ algorithms $A_1, \ldots, A_\ell$ acting on this workspace with the following restrictions: • A_i only modifies W and the i-th flag; • if $i \geq 2$ and the $(i-1)$-th flag is 0, A_i is the identity. Thus the quantum implementation of A_i is controlled on the $(i-1)$-th flag.

In this case, we do not need to assume that a single solution exists. Our search problem is formulated as:

$$\text{Find } c \in C \text{ such that: } A_\ell \circ \cdots \circ A_1(0_\ell, c, 0_w)|_{\text{flag } \ell} = 1.$$

For all i, we let $\mathcal{A}_i$ be a quantum implementation of A_i, of gate complexity $G(\mathcal{A}_i)$.

Filtering Probabilities. We define *filtering probabilities* which contain all necessary information to analyze the algorithms. The definition is different than in Section 3.2. We define them as:

$$\beta_1^2 = \Pr_{c \xleftarrow{\$} C}(A_1(0_\ell, c, 0_w)|_{\text{flag } 1} = 1) \tag{8}$$

$$\forall 2 \leq i \leq \ell, \beta_i^2 = \frac{\Pr_{c \xleftarrow{\$} C}(A_i \circ \cdots \circ A_1(0_\ell, c, 0_w)|_{\text{flag } i} = 1)}{\Pr_{c \xleftarrow{\$} C}(A_{i-1} \circ \cdots \circ A_1(0_\ell, c, 0_w)|_{\text{flag } i-1} = 1)}\ , \tag{9}$$

so that β_i^2 is the probability that it also passes the i-th step, conditioned on $c \in C$ having passed the $(i-1)$-th step. We may assume for now they are known.

Difference between Backtracking and Early-Abort. While the "backtracking" case splits the search space into several choice sets, the "early abort" case splits the testing function instead. Both approaches could be somewhat unified by considering a backtracking algorithm in which each testing step is separated into multiple sub-steps; it will then embed "early abort" algorithms at each level. However, most applications seem to fall in one of the two cases.

4 Quantum Search with Early Aborts

Since it is a simpler case, we start with the problem defined in Section 3.3. From a single choice set C, we want to find an element that passes a sequence of filters; these filters have typically increasing complexities. Only the final filter reveals the actual solution.

In this section, we detail our quantum algorithm for this problem, compute its success probability and time complexity, and explain how to optimize this complexity numerically. We prove in these computations that with ℓ levels of nesting, the "naive" overhead factor $(\pi/2)^\ell$ can be optimized into $\sqrt{\ell}$.

Although this case has some applications in cryptanalysis, we stress that it is fundamentally different from our main framework (search with *backtracking*) which will be detailed in Section 5. Nevertheless, despite their differences, most of the technical analysis from this section will be reused for the next one.

Algorithm 1 B_1: finds an element that passes the first filter A_1.

Workspace: F, C, W
Modifies: F_1, C, W
repeat
  Increment the value c stored in register C
  Compute A_1 in place ▷ Will update the flag F_1 and the workspace
until F_1 contains 1 or register C overflows

Algorithm 2 B_i: samples an element passing the filters A_1 to A_i.

Workspace: F, C, W
Modifies: F_i, C, W
repeat
  Call B_{i-1}
  Compute A_i in place ▷ Will update the flag F_i and the workspace
until F_i contains 1 or the register C overflows

4.1 Classical Early-Abort Algorithm

We start with a classical algorithm which will serve as inspiration for the quantum one. We define ℓ algorithms $B_1, \ldots, B_\ell$ where each B_i finds elements that pass the filters A_1 to A_i. Note that we use a counter to explore the choice set C, that makes the search deterministic (the register C is initialized to 0). This is one of the key differences between classical and quantum search, where quantum search instead may be seen as sampling choices at random.

In Algorithm 1, we start by sampling new values for c until we pass the first filter. In Algorithm 2, we call B_{i-1} to setup the workspace according to a new element that passes filters A_1 to A_{i-1}, then we evaluate A_i, and we continue until it is passed. When B_ℓ stops, either C has overflown (i.e., the whole choice set was explored without finding a solution), or the workspace contains a solution.

4.2 Description of the Quantum Algorithm

Our quantum algorithm is analogous to Algorithm 1 and Algorithm 2, except that it replaces the ℓ **Repeat-Until** loops by ℓ nested QAAs. The QAA at level i produces a superposition of choices which, with high probability, pass the i first filters. It is not necessary to succeed with probability 1 for these intermediate levels, and instead, the best strategy is to maintain the current probability of success *below* a certain level, which is inverse-linear in ℓ. The number of iterates of each QAA is then selected depending on our knowledge of the filtering probabilities β_i defined in Section 3.3.

We define a sequence of algorithms $\mathcal{B}_i$ $(1 \leq i \leq \ell)$ where $\mathcal{B}_i$ calls $\mathcal{A}_i$ and $\mathcal{B}_{i-1}$ as a subroutine. Each $\mathcal{B}_i$ acts on the same Hilbert space $\mathcal{F} \otimes \mathcal{C} \otimes \mathcal{W}$ as the building blocks $\mathcal{A}_i$. The complete algorithm is $\mathcal{B}_\ell$, which is parameterized by the number of iterations $(k_1, \ldots, k_\ell)$ chosen for each QAA.

In the definition of the algorithms (Algorithm 3 and Algorithm 4) we denote by O_i the test oracle that simply flips the phase if the i-th flag is 1. Each O_i

Algorithm 3 $\mathcal{B}_1$: performs k_1 iterates of QAA to filter the elements passing the first test.

Workspace: F, C, W
Modifies: F_1, C, W
Apply a Hadamard transform on the register C
Compute $\mathcal{A}_1$
Repeat k_1 **times**
    Flip the phase if the first flag qubit is 1
    Uncompute $\mathcal{A}_1$
    Apply a Hadamard transform on the register C
    Apply $-O_0$ on all registers
    Apply a Hadamard transform on the register C
    Compute $\mathcal{A}_1$
EndRepeat

Algorithm 4 $\mathcal{B}_i$: performs k_i iterates of QAA on top of $\mathcal{B}_{i-1}$, to filter the elements passing the i-th test.

Workspace: F, C, W
Modifies: $F_1, \ldots, F_i, C, W$
Compute $\mathcal{A}_i \circ \mathcal{B}_{i-1}$
Repeat k_i **times**
    Flip the phase if the i-th flag qubit is 1
    Uncompute $\mathcal{A}_i \circ \mathcal{B}_{i-1}$
    Apply $-O_0$ on all registers
    Compute $\mathcal{A}_i \circ \mathcal{B}_{i-1}$
EndRepeat

costs 3 Clifford gates. Recall that O_0 is the inversion around zero, which has a gate cost linear in the number of workspace qubits.

Example 1. If we take two levels of QAA and one iterate at each level, we obtain at the first level:

$$\mathcal{B}_1 = \mathcal{A}_1 H O_0 H \mathcal{A}_1^\dagger O_1 \mathcal{A}_1 H \ ,$$

and by writing $\mathcal{A}'_1 = \mathcal{A}_1 H$, at the second level:

$$\begin{aligned}\mathcal{B}_2 &= \mathcal{A}_2\mathcal{B}_1 O_0 (\mathcal{A}_2\mathcal{B}_1)^\dagger O_2 \mathcal{A}_2 \mathcal{B}_1 \\ &= \mathcal{A}_2\mathcal{A}'_1 O_0 \mathcal{A}_1'^\dagger O_1 \mathcal{A}'_1 O_0 (\mathcal{A}_2\mathcal{A}'_1 O_0 \mathcal{A}_1'^\dagger O_1 \mathcal{A}'_1)^\dagger O_2 \mathcal{A}_2 \mathcal{A}'_1 O_0 \mathcal{A}_1'^\dagger O_1 \mathcal{A}'_1 \\ &= (\mathcal{A}_2\mathcal{A}'_1) O_0 (\mathcal{A}'_1)^\dagger O_1 (\mathcal{A}'_1) O_0 (\mathcal{A}'_1)^\dagger O_1 (\mathcal{A}'_1) O_0 (\mathcal{A}_2\mathcal{A}'_1)^\dagger O_2 (\mathcal{A}_2\mathcal{A}'_1) O_0 (\mathcal{A}'_1)^\dagger O_1 (\mathcal{A}'_1) \ .\end{aligned}$$

So the algorithm looks like a QAA applied to the algorithm $\mathcal{A}_2\mathcal{A}_1 H$, except that we stop some of its iterations earlier.

4.3 Analysis

We do the analysis of $\mathcal{B}_\ell$ in three steps. First, we express its success probability as a function of $(k_1, \ldots, k_\ell)$ and $(\beta_1, \ldots, \beta_\ell)$. Second, we express its time complexity

as a function of $(k_1, \ldots, k_\ell)$ and the gate complexities of $\mathcal{A}_i$. Finally, we express the constraints which allow to choose optimal values for the k_i.

Success Probability. For all i, let ν_i^2 be the probability that after measuring $\mathcal{B}_i |0_\ell 0_n 0_w\rangle$, the i-th flag qubit is projected on 1; i.e., after running $\mathcal{B}_i$ on input a zero state, we obtain a choice that passes all the first i filters. By definition, ν_ℓ^2 is the probability of success of the whole procedure. We show that ν_i^2 depends only on the parameters β_i (from the search problem) and k_i (from the algorithm) as follows:

Lemma 3. *Let $\nu_0 = 1$ by convention. For all $i \geq 1$ we have:*

$$\nu_i = \sin\left[(2k_i + 1) \arcsin \beta_i \nu_{i-1}\right] \quad . \tag{10}$$

Proof. To facilitate the notation, we introduce the projectors P_i^0, P_i^1 which project a quantum state in $\mathcal{F} \otimes \mathcal{C} \otimes \mathcal{W}$ on the i-th flag 0 or 1, i.e.:

$$\forall 1 \leq i \leq \ell, P_i^b = I_{i-1} \otimes |b\rangle \langle b| \otimes I_{\ell-i-1+c+w} \quad ,$$

where I_r is the identity operator applied to r qubits.

Let $|\psi_i\rangle$ be the uniform superposition of choices c and corresponding state of the work register at step i, such that c passes all the filters from 1 to i included. We prove by induction that:

$$\mathcal{B}_i |0_{\ell+n+w}\rangle = \nu_i |1_i 0_{\ell-i}\rangle |\psi_i\rangle + |\chi_i\rangle \quad , \tag{11}$$

where $|\chi_i\rangle = P_i^0 \mathcal{B}_i |0_{\ell+n+w}\rangle$ is the part of the superposition where the i-th flag is zero. In general, it will be a superposition of failed states from levels $1, \ldots, i$, with different flags of the form $|1_j 0_{\ell-j}\rangle$ depending on the level which failed. In other words, $\mathcal{B}_i$ outputs $|\psi_i\rangle$ with probability ν_i^2.

We start from the output of $\mathcal{A}_1 H$, which by definition, is:

$$\mathcal{A}_1 H |0_{\ell+n+w}\rangle = \beta_1 |10_{\ell-1}\rangle |\psi_1\rangle + \left(P_1^0 \mathcal{A}_1 H |0_{\ell+n+w}\rangle\right) \quad .$$

Then, $\mathcal{B}_1$ applies k_1 iterates of QAA on top of $\mathcal{A}_1 H$, so by Equation 4, we have:

$$\mathcal{B}_1 |0_{\ell+n+w}\rangle = \sin\left[(2k_i + 1) \arcsin \beta_1\right] |10_{\ell-1}\rangle |\psi_1\rangle + \left(P_1^0 \mathcal{B}_1 |0_{\ell+c+w}\rangle\right) \quad ,$$

where we get the definition of ν_1. Next, we assume that Equation 11 is true for some $i \geq 1$. By definition, $\mathcal{B}_{i+1}$ applies k_{i+1} QAA iterates to the algorithm $\mathcal{A}_{i+1}\mathcal{B}_i$. We first look at the output of $\mathcal{A}_{i+1}\mathcal{B}_i$:

$$\begin{aligned}
\mathcal{A}_{i+1}\mathcal{B}_i |0_{\ell+n+w}\rangle &= \mathcal{A}_{i+1}\left(\nu_i |1_i 0_{\ell-i}\rangle |\psi_i\rangle + \sqrt{1-\nu_i^2}\,|\chi_i\rangle\right) \\
&= \nu_i \left(\beta_{i+1} |1_{i+1} 0_{\ell-i-1}\rangle |\psi_{i+1}\rangle + \left(P_{i+1}^0 \mathcal{A}_{i+1}(|1_i 0_{\ell-i}\rangle |\psi_i\rangle)\right)\right) + \sqrt{1-\nu_i^2}\,|\chi_i\rangle \\
&= \nu_i \beta_{i+1} |1_{i+1} 0_{\ell-i-1}\rangle |\psi_{i+1}\rangle + P_{i+1}^0 \mathcal{A}_{i+1}\mathcal{B}_i |0_{\ell+c+w}\rangle \quad .
\end{aligned}$$

Indeed, since $|\chi_i\rangle$ has always 0 in the flag bit i, $\mathcal{A}_{i+1}$ leaves it unchanged. By Equation 4 we have:

$$\mathcal{B}_{i+1}\,|0_{\ell+n+w}\rangle = \sin\left[(2k_i+1)\arcsin(\nu_i\beta_{i+1})\right]|1_{i+1}0_{\ell-i-1}\rangle\,|\psi_{i+1}\rangle \\ + P^0_{i+1}\mathcal{B}_{i+1}\,|0_{\ell+c+w}\rangle \quad ,$$

which completes the recurrence. □

Time Complexity. The time complexity of $\mathcal{B}_\ell$ is a function of the k_i and the gate counts of $\mathcal{A}_i$. Recall that we note $G(\mathcal{A}_i)$ the gate count of $\mathcal{A}_i$, that includes the controls on the flag qubit number $i-1$.

Lemma 4. *The gate complexity of $\mathcal{B}_\ell(k_1,\ldots,k_\ell)$ is given by:*

$$G(\mathcal{B}_\ell) = \left(\sum_{i=1}^{\ell}\left(\prod_{j=i}^{\ell}(2k_j+1)\right)G(\mathcal{A}_i)\right) + (44(w+\ell+n)-36+n)\prod_{j=1}^{\ell}(2k_j+1) \quad . \tag{12}$$

It uses $2(\ell+n+w)-1$ qubits.

Proof. This is a simple induction on the number of applications of each $\mathcal{A}_i$. For all i, $\mathcal{B}_\ell$ contains $\prod_{j=i}^{\ell}(2k_j+1)$ calls to $\mathcal{A}_i$. Each time O_0 is called, it is applied on $w+n+\ell$ qubits (the workspace, choices and flags), so it contains less than $44(w+n+\ell)-39$ Clifford+T gates, and uses $w+n+\ell-1$ ancillas with the simple implementation of Lemma 1. By assumption, ancilla qubits used by the $\mathcal{A}_i$ are counted as workspace qubits. Also each O_i contains 3 Clifford gates, and each H contains n Hadamard gates. Finally, the total number of calls to O_0 (resp. H and the O_i) is $\prod_{j=1}^{\ell}(2k_j+1)$. □

Choice of the k_i. Using approximations of the sin and arcsin functions, we transform Equation 10 into exploitable bounds on the ν_i^2.

Lemma 5. *Let $\nu_0 = 1$ by convention, and $s_i = (2k_i+1)^2\beta_i^2$. Then for all $i \geq 1$ we have:*

$$s_i\nu_{i-1}^2\left(1 - s_i\nu_{i-1}^2\right) \leq \nu_i^2 \leq s_i\nu_{i-1}^2 \quad . \tag{13}$$

Proof. Recall Equation 10: $\forall i \geq 1, \nu_i = \sin\left[(2k_i+1)\arcsin(\beta_i\nu_{i-1})\right]$ and $\nu_0 = 1$. For the upper bound, we use the inequality:

$$|\sin\left[(2k_i+1)x\right]| \leq (2k_i+1)|\sin x| \quad .$$

For the lower bound, we use the bounds on sin, then on arcsin:

$$\begin{aligned}\nu_i^2 &\geq (2k_i+1)^2\arcsin^2(\beta_i\nu_{i-1})\left[1 - \frac{(2k_i+1)^2\arcsin^2(\beta_i\nu_{i-1})}{3}\right] \\ &\geq (2k_i+1)^2(\beta_i\nu_{i-1})^2\left[1 - \frac{1}{3}\frac{\pi^2}{4}(2k_i+1)^2(\beta_i\nu_{i-1})^2\right] \quad ,\end{aligned}$$

and the bound follows from $1/3 \cdot \pi^2/4 \leq 1$. □

Theorem 1. *Let* $s_i = (2k_i+1)^2\beta_i^2$. *Assume that* $\forall i \leq \ell, \prod_{j=1}^{i} s_j \leq \frac{1}{2\ell}$. *Then we have:* $\nu_\ell^2 \geq \frac{1}{2}\prod_{j=1}^{\ell} s_j$.

Proof. We start by unfolding recursively the upper bound in Equation 13:

$$\forall i, \nu_i^2 \leq \prod_{j=1}^{i} s_j \ .$$

Then we can use this in the lower bound:

$$\nu_\ell^2 \geq \left(\prod_{i=1}^{\ell} s_i\right) \prod_{i=1}^{\ell} \left(1 - \left(\prod_{j=1}^{i} s_j\right)\right) \ .$$

Assuming that for all j, we have: $\prod_{j=1}^{i} s_j \leq 1$, then we can use the generalized Bernoulli's inequality[4] to simplify the lower bound:

$$\nu_\ell^2 \geq \left(\prod_{i=1}^{\ell} s_i\right) \left(1 - \sum_{i=1}^{\ell} \left(\prod_{j=1}^{i} s_j\right)\right) \ . \tag{14}$$

The bound $\prod_{j=1}^{i} s_j \leq \frac{1}{2\ell}$ allows to obtain: $\nu_\ell^2 \geq \frac{1}{2}\left(\prod_{i=1}^{\ell} s_i\right)$, which proves the theorem. □

Thus, by keeping the success probability of $\mathcal{B}_i$ inverse-linear in ℓ, we ensure that the nested QAAs amplify without any multiplicative factor in the complexity. However, the final probability of success remains inverse-linear in ℓ. We need to combine the procedure with an "overcooked" QAA (Lemma 2) with $\sqrt{\ell}$ iterates to obtain a constant probability. Overall, we have replaced the naive overhead factor $(\pi/2)^\ell$ by $\sqrt{\ell}$.

One can remark that in order to set properly the iteration numbers, it is required to know only the *cumulative* success probabilities $\prod_{j=1}^{i} \beta_j^2$, i.e., the amount of choices passing the first i filters. This fact is important for the analysis of variable-time amplitude amplification that we do in Appendix C.

4.4 Optimizing the Complexity Numerically

The complexity is easy to optimize if we know intervals on the β_i^2 of the form: $\beta_i^2 \in [l_i^2; u_i^2]$. Indeed, we observe that as long as we keep the iteration numbers sufficiently low, we can bound the final success probability using these known upper and lower bounds.

Lemma 6. *Assume that:* $\forall i, k_i \leq \left\lfloor \frac{\pi}{4}\frac{1}{\arcsin u_i} - \frac{1}{2} \right\rfloor$. *Let* ν_i^l *(lower bound) and* ν_i^u *(upper bound) be obtained by replacing the* β_i *by* l_i *and* u_i, *respectively, in the formulas of Lemma 3. Then we have:* $\nu_\ell^u \geq \nu_\ell \geq \nu_\ell^l$.

[4] This technique is inspired by Appendix A in [5].

Proof. These upper bounds on k_i ensure that, regardless of the exact value of β_i, all inputs to sin stay in the interval $[0; \pi/2]$ where the function is increasing. The bounds on ν_ℓ then follow by a simple induction. □

Therefore, we can bypass Theorem 1 and directly express the lower bound on the success probability ν_ℓ^2 of $\mathcal{B}_\ell$ as a function of the k_i:

$$\begin{cases} \nu_0^l = 1 \\ \forall i, \nu_i^l = \sin\left[(2k_i+1)\arcsin\left[l_i \nu_{i-1}^l\right]\right] \end{cases} \tag{15}$$

Then, given the formula for the gate complexity of $\mathcal{B}_\ell$ as a function of the iteration numbers, our goal is to:

$$\text{minimize } G(\mathcal{B}_\ell)/{\nu_\ell^l}^2 \text{ under the constraints: } \forall i, k_i \leq \left\lfloor \frac{\pi}{4} \frac{1}{\arcsin u_i} - \frac{1}{2} \right\rfloor.$$

We first optimize this numerically, then take the floor of the values k_i obtained (in order to avoid going above the bounds). This last rounding is, in practice, insignificant for the complexity.

In Appendix B, we study the problem of a *search with independent tests*, in which we first use Theorem 1 for the asymptotic case, then the optimization method for practical cases.

5 Search with Backtracking

In this section, we present a quantum algorithm to solve the search problem defined in Section 3, which uses a *backtracking* approach. We stress that this method differs strongly from Section 4: it targets a search problem with several choice levels, and would perform badly with a mere sequential combination of filters. However, the analysis will look similar, even though the ℓ QAAs are actually nested in reverse order compared to Section 4.

In general, a *backtracking* algorithm explores a search space by making partial choices for partial values of the solution and being able to check whether a partial solution may lead a full solution. Classically this can be seen as a depth-first tree search where it recognizes whether the current node can lead to a solution, and if it doesn't, returns to the parent node.

5.1 Classical Backtracking Algorithm

We start by explaining how to solve the problem with a *classical* algorithm.

Recall that we have a sequence of filtering functions $A_1, \ldots, A_\ell$ and of post-processing functions $D_1, \ldots, D_\ell$ acting in place on the workspace $F \times C \times W$. From them, we define a sequence of algorithms A'_i (Algorithm 5) and B_i (Algorithm 6) that also work in place. From a given node in the tree (i.e., a path of choices $c_1, ...c_{i-1}$), A'_i will sample new choices c_i passing the filter A_i, and B_i will write 1 in the flag F_ℓ if $(c_1, \ldots, c_{i-1}) = (c_1^g, \ldots, c_{i-1}^g)$ (i.e., if the path extends to the solution) and 0 otherwise. Furthermore, when it writes 1, the choice registers contain the good path. Each B_i calls B_{i+1} recursively, $B_{\ell+1}$ does nothing, and B_1 solves the problem.

Algorithm 5 A'_i: finds the next choice that passes the filter at step i. At level ℓ we define $A'_\ell = A_\ell$ (no loop).

Workspace: $F_1, \ldots, F_\ell$, $C_1, \ldots, C_\ell$, $W_1, \ldots, W_\ell$
Modifies: $C_i, F_i, W_i, \ldots, W_\ell$
1: **repeat**
2: Increment the value c_i stored in register C_i
3: Compute A_i in place ▷ Overwrites $w_i, \ldots, w_\ell, f_i$
4: **until** $f_i = 1$ or the choice register overflows

Algorithm 6 B_i: given a partial path of choices $(c_1, \ldots, c_{i-1})$ (i.e., a node in the tree), finds whether $(c_1, \ldots, c_{i-1}) = (c^g_1, \ldots, c^g_{i-1})$, and in that case, completes the solution path.

Workspace: $F_1, \ldots, F_\ell$, $C_1, \ldots, C_\ell$, $W_1, \ldots, W_\ell$
Modifies: all registers (F_i, C_i, W_i) numbered from i to ℓ
1: Initialize a counter in register C_i to value 0
2: **repeat**
3: Compute A'_i in place ▷ Overwrites $c_i, w_i, \ldots, w_\ell, f_i$
4: **if** $f_i = 1$ **then** ▷ A new path to explore
5: Compute D_i in place ▷ Overwrites $w_i, \ldots, w_\ell$
6: Compute B_{i+1} in place ▷ Overwrites all registers from $i+1$ to ℓ
7: **until** $f_i = 0$ or $f_\ell = 1$
▷ If $f_\ell = 1$, we found the solution path $(c^g_1, \ldots, c^g_\ell)$, so we must stop here and the choice registers $(c_1, \ldots, c_\ell)$ will contain this path. If $f_i = 0$, we stopped because there wasn't any path left to explore.

5.2 Description of the Algorithm

Similarly to the classical backtracking, we define a sequence of quantum algorithms $\mathcal{B}_i$ $(1 \leq i \leq \ell)$ such that each $\mathcal{B}_i$ calls $\mathcal{A}_i$ and $\mathcal{B}_{i+1}$ as a subroutine. Intuitively, each **Repeat-Until** loop that we wrote in Algorithm 5 and Algorithm 6 will now become a QAA.

These algorithms act on the same Hilbert space $\mathcal{F} \otimes \mathcal{C} \otimes \mathcal{W}$ as the $\mathcal{A}_i$ and $\mathcal{D}_i$. Each of them only modifies a subset of the registers. They are parameterized by two sequences of integers $(k_1, \ldots, k_\ell)$ and $(k'_1, \ldots, k'_\ell = 0)$ that we will choose afterwards, which are the numbers of QAA iterates performed at each level, for $\mathcal{B}_i$ and $\mathcal{A}'_i$ respectively. The complete algorithm, which is intended to solve our search problem, is $\mathcal{B}_1$. It starts from a workspace initialized to zeroes.

Intuitively, $\mathcal{B}_i$ is an amplified version of the sequence of steps from i to ℓ; if it starts from the good subpath $(c^g_1, \ldots, c^g_{i-1})$, it returns the complete solution; otherwise it fails. We start from $\mathcal{B}_{\ell+1} = I$, and then the definition of the algorithms is given in Algorithm 7 and Algorithm 8.

Success Probability. As before, the analysis of $\mathcal{B}_1$ starts with an expression of its success probability as a function of its parameters k_i, k'_i and the parameters of the problem $\alpha_i^2, \alpha_i'^2$. In the following, we omit to write the work registers, and

Algorithm 7 $\mathcal{A}'_i$: filters the choices at step i. At level ℓ we have $k'_\ell = 0$ and so $\mathcal{A}'_\ell = \mathcal{A}_\ell \circ H$.

Workspace: $F_1, \ldots, F_\ell$, $C_1, \ldots, C_\ell$, $W_1, \ldots, W_\ell$
Modifies: $C_i, F_i, W_i, \ldots, W_\ell$
1: Apply a Hadamard transform H on C_i
2: **Repeat** k'_i **times**
3: Compute $\mathcal{A}_i$
4: Flip the phase if the i-th flag qubit F_i is 1
5: Uncompute $\mathcal{A}_i$
6: Apply H, $-O_0$, and another H, all on C_i
7: **EndRepeat**
8: Compute $\mathcal{A}_i$

Algorithm 8 $\mathcal{B}_i$: performs a QAA on the algorithm $\mathcal{B}_{i+1} \circ \mathcal{D}_i \circ \mathcal{A}'_i$.

Workspace: $F_1, \ldots, F_\ell$, $C_1, \ldots, C_\ell$, $W_1, \ldots, W_\ell$
Modifies: all registers (F_i, C_i, W_i) numbered from i to ℓ
1: Compute $\mathcal{B}_{i+1} \circ \mathcal{D}_i \circ \mathcal{A}'_i$
2: **Repeat** k_i **times**
3: Flip the phase if the ℓ-th flag qubit is 1
4: Uncompute $\mathcal{B}_{i+1} \circ \mathcal{D}_i \circ \mathcal{A}'_i$
5: Apply $-O_0$ on the registers numbered from i to ℓ
6: Compute $\mathcal{B}_{i+1} \circ \mathcal{D}_i \circ \mathcal{A}'_i$
7: **EndRepeat**

focus only on the flag and choice registers; we also omit the trailing zeroes of an incomplete path of choices. One should note that the state of the work registers is a deterministic function of the choice registers. Besides, the definition of our algorithms ensures that we always apply the algorithms in the right order, i.e., we apply $\mathcal{A}_i$ when the current work registers contain a valid output of $\mathcal{D}_{i-1}$, and $\mathcal{D}_i$ when they contain a valid output of $\mathcal{A}_i$. In order to simplify the notation, we reuse the projectors P_i^0, P_i^1 which project a quantum state in $\mathcal{F} \otimes \mathcal{C} \otimes \mathcal{W}$ on the i-th flag 0 or 1. We define the success probability ν_i^2 of $\mathcal{B}_i$ as the probability that, *starting from the good subpath* $c_1^g, \ldots, c_{i-1}^g$, $\mathcal{B}_i$ will output the complete good path. By definition ν_1^2 is the success probability of our complete algorithm.

$$\begin{cases} \forall i \geq 2, & \nu_i = \| P_\ell^1 \mathcal{B}_i \left| 1_{i-1} 0_{\ell-i+1} \right\rangle \left| c_1^g, \ldots, c_{i-1}^g \right\rangle \| \\ & \nu_1 = \| P_\ell^1 \mathcal{B}_1 \left| 0_\ell 0_c \right\rangle \| \end{cases} \tag{16}$$

Lemma 7. *We have:* $\nu_\ell = \sin\left[(2k_\ell + 1) \arcsin \alpha_\ell\right]$ *and for all* $i < \ell$*:*

$$\nu_i = \sin\left[(2k_i + 1) \arcsin\left[\nu_{i+1} \alpha_i \sin((2k'_i + 1) \arcsin(\alpha'_i))\right]\right] \ . \tag{17}$$

Proof. We do a descending recursion on the value of i, starting from $i = \ell$. We will also verify during this recursion that for all i, if we start $\mathcal{B}_i$ on a wrong subpath, we obtain only zero flags, i.e.:

$$\forall i, \forall (c_1, \ldots, c_{i-1}) \neq (c_1^g, \ldots, c_{i-1}^g), \| P_\ell^1 \mathcal{B}_i \left| 1_{i-1} 0_{\ell-i+1} \right\rangle \left| c_1, \ldots, c_{i-1} \right\rangle \| = 0 \ .$$

We start by analyzing the behavior of $\mathcal{A}'_i$, assuming that we start from the good path $(c_1^g, \ldots, c_{i-1}^g)$. After a Hadamard layer and a call to $\mathcal{A}_i$ (i.e., before amplification), we obtain the state:

$$\begin{aligned}
&\mathcal{A}_i H \left|1_{i-1}0_{\ell-i+1}\right\rangle \left|c_1^g, \ldots, c_{i-1}^g\right\rangle |0\rangle \\
&\qquad = \alpha'_i \left|1_i 0_{\ell-i}\right\rangle \underbrace{\left|c_1^g, \ldots, c_{i-1}^g\right\rangle \frac{1}{\sqrt{2^{n_i}\alpha'^2_i}} \sum_{\substack{c_i \in C_i \\ \text{passes the filter}}} |c_i\rangle}_{:=|\psi_i\rangle} \\
&+\sqrt{1-\alpha'^2_i}\left|1_{i-1}0_{\ell-i+1}\right\rangle \underbrace{\left|c_1^g, \ldots, c_{i-1}^g\right\rangle \frac{1}{\sqrt{2^{n_i}}\sqrt{1-\alpha'^2_i}} \sum_{\substack{c_i \in C_i \\ \text{does not pass the filter}}} |c_i\rangle}_{:=|\chi_i\rangle} ,
\end{aligned}$$

where $|\psi_i\rangle$ and $|\chi_i\rangle$ are two normalized quantum states. Therefore, after amplification (using Equation 4), $\mathcal{A}'_i$ produces:

$$\begin{aligned}
&\mathcal{A}'_i \left|1_{i-1}0_{\ell-i+1}\right\rangle \left|c_1^g, \ldots, c_{i-1}^g\right\rangle |0\rangle \\
&= \sin((2k'_i+1)\arcsin\alpha'_i)\left|1_i 0_{\ell-i}\right\rangle |\psi_i\rangle + \cos((2k'_i+1)\arcsin\alpha'_i)\left|1_{i-1}0_{\ell-i+1}\right\rangle |\chi_i\rangle \ .
\end{aligned}$$

In the state $|\psi_i\rangle$, which is a uniform superposition over all c_i passing the filter, we single out the good choice c_i^g. Since we have $\frac{1}{\sqrt{2^{n_i}\alpha'^2_i}} = \alpha_i$, we see that the output of $\mathcal{A}'_i$ is a superposition of flags and paths where $(c_1^g, \ldots, c_{i-1}^g, c_i^g)$ has amplitude: $\alpha_i \sin((2k'_i+1)\arcsin\alpha'_i)$.

Applying $\mathcal{D}_i$ leaves the superposition unchanged (it only modifies the work registers). Then, we apply $\mathcal{B}_{i+1}$. On wrong subpaths, $\mathcal{B}_{i+1}$ will always write the ℓ-th flag to zero, by the recurrence hypothesis. The good subpath $(c_1^g, \ldots, c_{i-1}^g, c_i^g)$ is flagged with probability ν_{i+1}^2, so we have:

$$\left\|P_\ell^1 \mathcal{B}_{i+1}\mathcal{D}_i\mathcal{A}'_i \left|1_{i-1}0_{\ell-i+1}\right\rangle \left|c_1^g, \ldots, c_{i-1}^g\right\rangle |0\rangle\right\| = \nu_{i+1}\alpha_i \sin((2k'_i+1)\arcsin\alpha'_i) \ .$$

Using Equation 4 again, we obtain the wanted formula for ν_i^2. If we start from a wrong path, $\mathcal{A}'_i$ produces a superposition of wrong paths. Going through $\mathcal{B}_{i+1}$, the last flag written is always zero, so even after amplification this remains the case. □

5.3 Choosing the Iteration Numbers

We have to determine the iteration numbers k_i and k'_i to maximize the success probability given by Lemma 7. For this, we do not need to know α_i and α'_i exactly, but only two intervals of the form:

$$\begin{cases} l'^2_i \leq \alpha'^2_i \leq u'^2_i \\ l_i^2 := \frac{1}{u'^2_i |C_i|} \leq \alpha_i^2 \leq \frac{1}{l'^2_i |C_i|} := u_i^2 \end{cases} \tag{18}$$

where the second is deduced from the first by $\alpha_i^2 \alpha_i'^2 = \frac{1}{|C_i|}$. Note that $u_\ell = l_\ell = \frac{1}{\sqrt{|C_\ell|}}$ by our assumptions. More generally, the case $l_i = u_i$ happens when α_i and α_i' are known exactly, for example where there is no filtering.

First, we remark that such intervals are enough as long as we keep the iteration numbers sufficiently low.

Lemma 8. *Assume that:* $\forall i, k_i \leq \frac{\pi}{4} \frac{1}{\arcsin u_i} - \frac{1}{2}$ *and* $\forall i < \ell, k_i' \leq \frac{\pi}{4} \frac{1}{\arcsin u_i'} - \frac{1}{2}$. *Let* ν_i^l *(lower bound) and* ν_i^u *(upper bound) be obtained by replacing the* α_i *and* α_i' *by* l_i *and* l_i'*, and by* u_i *and* u_i'*, respectively, in the formulas of Lemma 7. Then we have:* $\nu_1^u \geq \nu_1 \geq \nu_1^l$.

Proof. Same as Lemma 6: all inputs to sin stay in the interval $[0; \pi/2]$ where the function is increasing. □

Our strategy is now as follows: instead of bounding the true ν_1, which depends on parameters that we do not know, we will bound ν_1^l depending on parameters that we know (l_i, l_i', u_i, u_i'). To simplify the notation, we write ν instead of ν^l in what follows.

In order to bound ν_1, we first unfold the recursive formula of Lemma 7 into lower and upper bounds on the ν_i. The proof is almost the same as Lemma 5, except that the order of indices has changed, and new factors intervene due to the intermediate QAAs $\mathcal{A}_i'$.

Lemma 9. *Let* $\nu_{\ell+1} = 1$, $k_\ell' = 0$ *and* $l_\ell' = 1$. *Let* $s_i = (2k_i + 1)^2 l_i^2$ *and* $s_i' = \sin^2\left[(2k_i' + 1)\arcsin(l_i')\right]$. *Then for all* $i \leq \ell$ *we have:*

$$s_i s_i' {\nu_{i+1}}^2 \left(1 - s_i s_i' {\nu_{i+1}}^2\right) \leq {\nu_i}^2 \leq s_i s_i' {\nu_{i+1}}^2 \ . \tag{19}$$

We now state our main theorem, which follows entirely from Equation 19. The proof is identical to the one of Theorem 1, except for the replacement of s_i by $s_i s_i'$ and the order of indices.

Theorem 2. *Assume that* $\forall i, \prod_{j=i}^{\ell}(s_j s_j') \leq \frac{1}{2\ell}$. *Then:* $\nu_1^2 \geq \frac{1}{2}\prod_{j=1}^{\ell}(s_j s_j')$.

5.4 Analytic Complexity Formula

We give the gate and space complexities of the algorithm depending on k_i and k_i'. Recall that we use $G(\mathcal{A})$ to denote the gate cost of $\mathcal{A}$ and $S(\mathcal{A})$ the number of qubits on which it acts. For the cost of an inversion around zero on r qubits, we use the costs of Lemma 1, i.e., $44r - 39$ Clifford+T gates (and 3 gates for the bit-flipping). We write $I(r) = 44r - 36$.

For the space complexity, we count all the registers: $\ell + \sum_{1 \leq i \leq \ell}(w_i + n_i)$, and we add to this the number of ancilla qubits required by our implementation of O_0:

$$\forall i, S(\mathcal{B}_i) = 2 \sum_{j \leq i \leq \ell} (1 + w_j + n_j) - 1 = \ell - j + 2 \sum_{j \leq i \leq \ell} (w_j + n_j) \ . \tag{20}$$

The gate complexity of $\mathcal{B}_i$ is given by the recursive formula:

$$(2k_i+1)\Big(G(\mathcal{B}_{i+1})+G(\mathcal{D}_i)+(2k'_i+1)\left(G(\mathcal{A}_i)+n_i\right)+k'_i I(n_i)\Big)+k_i I(S(\mathcal{B}_i)) \ ,$$

where $G(\mathcal{D}_\ell)=0$, $k'_\ell=0$, $G(\mathcal{B}_{\ell+1})=0$ (since $\mathcal{B}_{\ell+1}$ and $\mathcal{D}_\ell$ do nothing). We simplify it into:

$$G(\mathcal{B}_i)\leq(2k_i+1)\Big(G(\mathcal{B}_{i+1})+G(\mathcal{D}_i)+(2k'_i+1)\left(G(\mathcal{A}_i)+n_i+\frac{1}{2}I(n_i)\right)+\frac{1}{2}I(S(\mathcal{B}_i))\Big) \ . \tag{21}$$

Finally, we choose appropriate iteration numbers, deduce the success probability of $\mathcal{B}_1$ by Theorem 2 and its gate count by the formula above. This completes the "analytic" study.

Theorem 3. *Choose:*

$$\begin{cases} k_\ell=\max\left(\left\lfloor\frac{1}{2\sqrt{2\ell}}\frac{1}{u_\ell}-\frac{1}{2}\right\rfloor,0\right) \\ \forall i<\ell, k_i=\max\left(\left\lfloor\frac{1}{2}\frac{1}{u_i}-\frac{1}{2}\right\rfloor,0\right) \\ \forall i<\ell, k'_i=\max\left(\left\lfloor\frac{\pi}{4\arcsin(u'_i)}-\frac{1}{2}\right\rfloor,0\right) \end{cases} \tag{22}$$

Then the probability of success of $\mathcal{B}_1$ is lower bounded by:

$$\nu_1^2\geq\frac{1}{2}\prod_i(2k_i+1)^2l_i^2\prod_i\sin^2\left((2k'_i+1)\arcsin(l'_i)\right) \ , \tag{23}$$

and its gate count is upper bounded by:

$$\sum_{i=1}^{\ell}\left(\prod_{j=i}^{\ell}(2k_j+1)\right)\left(G(\mathcal{D}_i)+(2k'_i+1)\left(G(\mathcal{A}_i)+n_i+\frac{1}{2}I(n_i)\right)+\frac{1}{2}I(S(\mathcal{B}_i))\right) \ . \tag{24}$$

Proof. We check that the iteration numbers satisfy the conditions of Lemma 8, especially the choice of k_i. If a step has no filtering, then $l'_i=u'_i=1$, in which case we take $k'_i=0$ and the term $\sin^2\left((2k'_i+1)\arcsin(l'_i)\right)$ equals 1.

Indeed we have for all x, $\arcsin x\leq\frac{\pi}{2}x$, so:

$$k_i\leq\frac{\pi}{4}\left(\arcsin\left(\frac{1}{l'_i\sqrt{|C_i|}}\right)\right)^{-1}-\frac{1}{2}\leq\frac{\pi}{4\arcsin u_i}-\frac{1}{2} \ .$$

Next, we have: $\forall i$, $(2k_i+1)^2l_i^2\leq(2k_i+1)^2u_i^2\leq1$ and $(2k_\ell+1)^2l_\ell^2\leq(2k_\ell+1)^2u_\ell^2\leq\frac{1}{2\ell}$ so we can use Theorem 2 to conclude. The gate count is obtained by recursively unfolding Equation 21. □

We could simplify further the formula by using a bound on the size of the workspace. However, in some examples (Section 6.1) the factor $S(\mathcal{B}_i)$ is not negligible and should not be bounded away for all i simultaneously.

5.5 Optimizing the Complexity Numerically

For practical applications, especially when ℓ is not too large, we will obtain better results with a direct optimization. We bypass Theorem 3 and directly focus on the value of ν_1^l, the lower bound of the success probability, which as we recall, is obtained by the following recursion:

$$\begin{cases} \nu_\ell^l = \sin\left[(2k_\ell + 1)\arcsin\left(l_\ell\right)\right] \\ \forall i, \nu_i^l = \sin\left[(2k_i + 1)\arcsin\left[\nu_{i+1}^l l_i \sin((2k_i' + 1)\arcsin l_i')\right]\right] \end{cases} . \tag{25}$$

We simplify our optimization problem by setting $k_i' = \left\lfloor \frac{\pi}{4}\frac{1}{\arcsin(u_i')} - \frac{1}{2} \right\rfloor$ (intuitively we do not gain anything by postponing the early-abort step). Then, by simply running $\mathcal{B}_1$ repeatedly, a solution is found with average time complexity $\frac{1}{{\nu_1^l}^2} G(\mathcal{B}_1)$. Thus, given the formula for the gate complexity of $\mathcal{B}_1$, as a function of the iteration numbers, our goal is to:

$$\text{minimize } \left(G(\mathcal{B}_1)/{\nu_1^l}^2\right) \text{ under the constraints: } \forall i, k_i \leq \frac{\pi}{4}\frac{1}{\arcsin u_i} - \frac{1}{2} .$$

We can observe, as it was done in [23], that this direct optimization actually leads to lower probabilities of success, e.g. between 70% and 80%. In our code, we increased the exponent on ν_1^l, in order to naturally bring the success probability closer to 1.

5.6 Analysis of the Memory

When turning a classical nested search into a quantum one, using our framework, some conversion of the memory model is needed (see Section 2 for an overview of quantum memory models). We can summarize it with the following rules:

- All registers F_i, C_i, W_i, since they will be overwritten by one of the $\mathcal{A}_i$ or $\mathcal{D}_i$, are qubits.
- If one of the $\mathcal{A}_i$ needs fast random access to one of the previous work registers (e.g., reading from a table in memory), then the QRAQM model is required. Otherwise, performing *sequential* access, or accessing cells of fixed (global constant) position can be done with the standard quantum circuit model.
- If one of the $\mathcal{A}_i$ needs fast random access to a table that was initialized *before the first step*, then the QRACM model is required. The data in this table will remain classical, but the indices accessed will be put in superposition. Otherwise, performing *sequential* access, or accessing cells of fixed (global constant) position can be done with the standard quantum circuit model, and no QRACM would be required (only a classical memory).

6 Applications to AES cryptanalysis

In this section, we demonstrate our framework by improving the algorithms and complexity of the quantum attacks on AES given in [11]. Since we are interested merely in the shape of these algorithms and their resulting complexities, we refer to [11] for the details of the attacks and their correctness.

Preliminaries on AES. The AES [15] is the standardized version of the block cipher Rijndael [14], which is a substitution-permutation network (SPN) operating on a 4×4 matrix of bytes. An AES round applies the operations **AddRoundKey** (ARK, XORs the current round key to the state), **SubBytes** (SB, applies the S-Box S to each byte individually), **ShiftRows** (SR, permutes the bytes), **MixColumns** (MC, applies a linear function to each column).

The states of round i after ARK, SB, SR and MC are denoted respectively as x_i, y_i, z_i, w_i, and k_i is the round key of round i. The bytes of these states are numbered from $x_i[0]$ to $x_i[15]$ (top to bottom, left to right in the byte matrix).

In these attacks, we are given *classical chosen-plaintext* access to a black-box E_K implementing a reduced-round AES with a secret key K. The attacks recover some key material, i.e., bytes of some round keys k_i. A valid quantum attack must outperform the exhaustive search with Grover's algorithm, which always applies. Since the AES S-Box S is the only nonlinear component, it dominates the cost of quantum circuits, and we estimate the complexity not by counting individual gates, but by counting S-Box circuits (see e.g. [11])[5].

Results. Our numbers are rounded to the second decimal. The results are summarized in Table 1. The memory usage is the same for all approaches, and equal to [11]: approximately 2^{25} qubits with QRAQM access, and 2^{36} classical memory without QRACM access for the Square attack; and 2^{88} classical memory without QRAQM / QRACM for the DS-MITM attack (with a small number of qubits).

We can observe that for the Square attack, there is no benefit with respect to the previous approach. This is because the nested search terms (and piling-up constants) do not dominate here. The only advantage of our approach was to externalize the quantum algorithm design and complexity analysis.

For the DS-MITM attack, already the analytic formula performs better than the previous analysis, despite the loss in success probability (as can be seen on the ratio of cost over probability of success). The optimized variant wins a factor 2^4 at practically no loss in success probability. We are very close (up to a factor 2) of what we would obtain if we simply took the square-root of iteration numbers in the complexity estimate.

6.1 Square Attack on AES

The Square attack on 6-round AES using the partial sums technique [19] is a good example of a simple backtracking algorithm, without early-abort. Its quantum version is given in Appendix A.2 of [11]. The path of the attack is reproduced in Figure 5. It is based on the following 3-round distinguisher: by encrypting a set of 2^8 plaintexts which vary only in byte 0 (a δ-set), after 3 rounds of AES, each byte is balanced: the sum of the 2^8 values that it takes is equal to zero.

We consider a few *structures* defined as follows: each structure contains 2^{32} plaintext-ciphertext pairs $(p_i, c_i)_{0 \leq i \leq 2^{32}-1}$, such that the first diagonal of p_i (diagonal in the state x_0 in Figure 5) takes all 2^{32} possible values. It then maps to

[5] We keep this metric for simplicity despite the cases when we consider QRAQM / QRACM, where the S-Box could be also implemented by a (quantum) table lookup.

Table 1. Estimation of attack complexities, in number of S-Box circuits. "Analytic": using Theorem 3. "Optimized": using numerical optimization as per Section 5.5. "Square root": numbers obtained by taking exact square roots of the number of iterates in the classical search. "Normalized Time": time complexity divided by the success probability.

		[11]	Analytic	Optimized	Square root
6-round AES Square	Time	$2^{44.73}$	$2^{48.24}$	$2^{44.81}$	$2^{44.05}$
	Success prob.	1	0.5	0.98	
	Normalized Time	$2^{44.73}$	$2^{49.24}$	$2^{44.84}$	
8-round AES-256 DS-MITM	Time	$2^{136.17}$	$2^{133.71}$	$2^{132.05}$	$2^{131.07}$
	Success prob.	0.73	0.5×0.73	0.92×0.73	
	Normalized Time	$2^{136.62}$	$2^{135.16}$	$2^{132.62}$	

2^{24} δ-sets through the first round, and after the distinguisher, byte $x_4[0]$ should be balanced. For a given guess of key bytes $u_5[0]$ and $k_6[0,1,2,3]$ ($\bullet$ on the picture), we can compute backwards one round on each ciphertext c_i to reach the value of the byte $x_4[0]$. By taking the sum of all these values, we check whether $x_4[0]$ is balanced over the whole structure. This sum is expressed as:

$$\sum_{0 \le i \le 2^{32}-1} S^{-1}\big(u_5[0] \oplus a_0 S^{-1}(c_i[0] \oplus k_6[0]) \oplus a_1 S^{-1}(c_i[1] \oplus k_6[1]) \\ \oplus a_2 S^{-1}(c_i[2] \oplus k_6[2]) \oplus a_3 S^{-1}(c_i[3] \oplus k_6[3])\big) \ ,$$

for known constants a_0, a_1, a_2, a_3 coming from **MixColumns**. We want to find the choice of $k_6[0,1,2,3]$ and $u_5[0]$ for which this sum is equal to 0 for all structures. Instead of having to do 2^{32} computations per key guess, Algorithm 9 amortizes this cost thanks to intermediate tables. Here we take $\mathcal{D}_i = I$ for all i. The successive algorithms $\mathcal{A}_i$ correspond to constructing these tables.

Using the generic formula of Theorem 3, we obtain a worse time complexity than the one given in [11]. Indeed, we must take the following number of iterates for the successive steps: 127, 7, 7, 2, with the last one quite small to allow for a reduced probability of success. We obtain a count of $2^{44.24}$ S-Boxes for a probability of success $2^{-4.74}$. But in the attack, the calls to $\mathcal{A}_1$ dominate the time complexity. In [11] there are roughly $\frac{\pi}{2}2^8$ such calls, but in our case, we need to re-amplify the last layer with 16 calls, so we will call $\mathcal{A}_1$ roughly 16×2^8 times and this is not competitive.

However, the numerical optimization (see Section 5.5) "sees" that $\mathcal{A}_1$ dominates, and amplifies the non-dominating steps to a success probability closer to 1. This results in the numbers of iterates 186, 11, 11, 11 for a success probability 0.98 and a time complexity of $2^{44.81}$ S-Boxes.

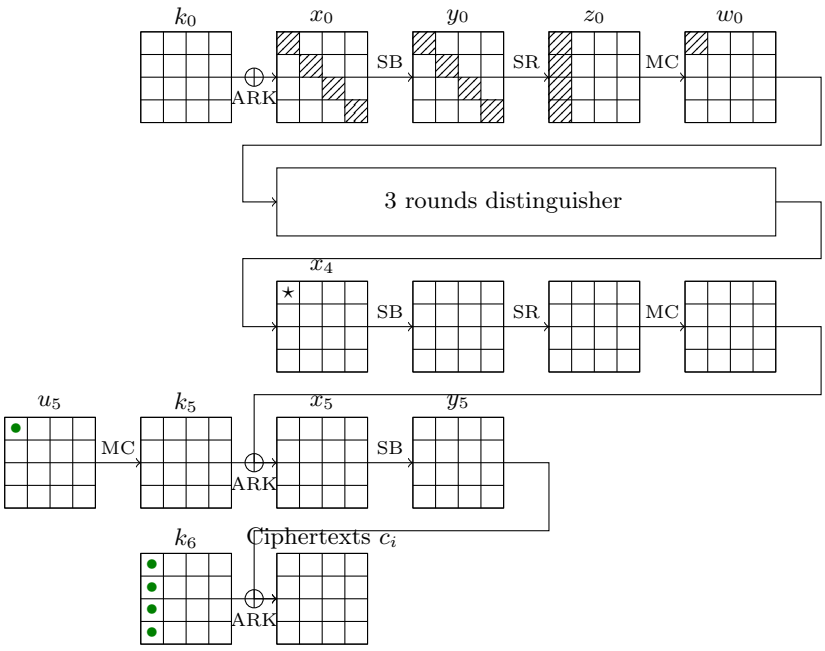

Fig. 5. AES Square attack

6.2 DS-MITM Attack on 8-round AES-256

We consider the DS-MITM key-recovery attack on reduced AES-256 given in [11]. We refer to [11] for the full attack algorithm, and instead focus on its translation in our framework.

The attack uses a 5-round distinguisher with the following property. Given a pair of states with only one active byte in input and output of these 5 rounds, by making the input active byte vary and take all 2^8 possible values, the sequence of values (δ-sequence) obtained in the output byte has a relatively small number of possibilities. In the quantum attack, all possible sequences are actually enumerated. These 5 middle rounds are extended forwards and backwards by several rounds using key guesses.

The attack algorithm has then the following structure. One first guesses the required key bytes, then identifies a pair of plaintexts which satisfies the differential condition for the distinguisher. The plaintexts corresponding to the δ-sequence are encrypted and the δ-sequence is computed. Then, all the possible δ-sequences are enumerated, and if the found δ-sequence appeared among them, the key guess is deemed correct with overwhelming probability.

The enumeration of δ-sequences is the most technical step of the algorithm, as it requires to guess several state differences for the chosen pair of plaintexts; from these differences, one deduces several internal state values, by solving differential

Algorithm 9 Square attack on 6-round AES.

	Compute 8 structures of 2^{32} classical chosen-plaintext queries
16 bits	**Choose** $k_6[0], k_6[1]$
	For each structure, for each ciphertext c_i, compute:
	$(t_1, t_2, t_3) = a_0 S^{-1}(c_i[0] \oplus k_6[0]) \oplus a_1 S^{-1}(c_i[1] \oplus k_6[1]), c_i[2], c_i[3]$.
$\mathcal{A}_1$	Build a table T_1 of 2^{24} entries that stores, for each three-byte value (t_1, t_2, t_3), how many times it appears.
	This costs $2^{32} \times 8 \times 2 = 2^{36}$ S-Boxes and space: $w_1 = 8 \times 2^{24} \times 32 = 2^{32}$ qubits (we keep large 32-bit counters)
8 bits	**Choose** $k_6[2]$
	For each 3-byte value (t_1, t_2, t_3), compute: $t_1 \oplus a_2 S^{-1}(t_2 \oplus k_6[2]), t_3$
$\mathcal{A}_2$	by accessing T_1. Build a table T_2 of 2^{16} entries that stores, for each 2-byte value (s_1, s_2), how many times it appears.
	This costs $8 \times 2^{24} \times 1 = 2^{27}$ S-Boxes and space: $w_2 = 8 \times 2^{16} \times 32 = 2^{24}$ qubits (we keep large counters).
8 bits	**Choose** $k_6[3]$
	For each 2-byte value (s_1, s_2), compute $s_1 \oplus a_3 S^{-1}(s_2 \oplus k_6[3])$.
$\mathcal{A}_3$	Build a table T_3 that stores how many times each byte appears.
	This costs $8 \times 2^{16} \times 1 = 2^{19}$ S-Boxes and space: $w_3 = 8 \times 2^8 \times 32 = 2^{16}$ qubits.
8 bits	**Choose** $u_5[0]$
	Using the table, compute the sum.
$\mathcal{A}_4$	This costs: $8 \times 2^8 \times 1$ S-Boxes and additional small computations, and approx. $8 \times 32 = 2^8$ qubits.

equations for the AES S-Box. These differential equations can have 0 ("no match" in Algorithm 10), 2 or 4 solutions. Finally, several "state-key" equations relate the internal state values with the key guess.

The complexity analysis of the algorithm relies on several estimations:

- There are 40 S-Box differential equations of the form $S(x \oplus \Delta) = S(x) \oplus \Delta'$ for known Δ, Δ' in the differential path. We suppose that for the good subkey guess, all of them have 2 solutions exactly, and not 4. This happens for the whole differential path with probability $2^{-0.45}$ as estimated in [11]
- some steps ($\mathcal{A}_3$ to $\mathcal{A}_6$ in Algorithm 10) have varying success probabilities, depending on the current key and state guesses. It is estimated in [11] that this probability varies less than by 2^{-8}
- the computation of the state-key equations in $\mathcal{D}_6$ cost less than 2^{10} S-Boxes (most of these computations are only linear)
- at most 4 different values are found after $\mathcal{D}_6$

Using Theorem 3, we obtain an algorithm of complexity: $2^{129.53}$ (S-Boxes) with success probability $\geq 2^{-5.40}$. Most of this uncertainty comes from the constant factor and the reduction of the success probability that ensures the correctness of our algorithm. Using Lemma 2, we find that with *on average* 18 calls to this procedure and its inverse, we can bring the success probability to 1/2. This already improves over [11].

Algorithm 10 DS-MITM attack on 8-round AES-256.

80 bits	**Choose** $k_0[0,5,10,15]$, $k_1[3]$, $u_7[1]$, $u_8[0,7,10,13]$
	Find a pair which satisfies the differential path (2^{53} S-Boxes)
$\mathcal{D}_1$	Compute the 2^5-sequence of differences $\delta w_5[5]$ ($< 2^{88}$ S-Boxes)
	Compute $\Delta x_2[4-7]$ and $\Delta y_5[3,4,9,14]$
64 bits	**Choose** $\Delta y_2[4-7], \Delta x_5[3,4,9,14]$
$\mathcal{A}_2$	Match $\Delta y_2[4-7], \Delta x_5[3,4,9,14]$ with $\Delta x_2[4-7]$, $\Delta y_5[3,4,9,14]$
Success: 2^{-8}	**Stop** if no match
$\mathcal{D}_2$	Compute the possible states
	Compute Δx_3 and Δy_4
	(Here we have assumed that the S-Box differential equations yield only two solutions)
32 bits	**Choose** $\Delta x_4[0-3]$
$\mathcal{A}_3$	Match $\Delta x_4[0-3]$ with Δx_3 and Δy_4
Succ.: $2^{-8}(1 \pm 2^{-8})$	**Stop** if no match
32 bits	**Choose** $\Delta x_4[4-7]$
$\mathcal{A}_4$	Match $\Delta x_4[4-7]$ with Δx_3 and Δy_4
Succ.: $2^{-8}(1 \pm 2^{-8})$	**Stop** if no match
32 bits	**Choose** $\Delta x_4[8-11]$
$\mathcal{A}_5$	Match $\Delta x_4[8-11]$ with Δx_3 and Δy_4
Succ.: $2^{-8}(1 \pm 2^{-8})$	**Stop** if no match
32 bits	**Choose** $\Delta x_4[12-15]$
$\mathcal{A}_6$	Match $\Delta x_4[12-15]$ with Δx_3 and Δy_4
Succ.: $2^{-8}(1 \pm 2^{-8})$	**Stop** if no match
$\mathcal{D}_6$	Using all the known states, write the state-key equations
	Determine the values of $x_3, x_2[4-7], x_4[0-7,10,11,15]$
	(Here we assume that at most 4 different values are found, which leaves 2^9 choices in total at the next step.)
9 bits	**Choose** one of 2^9 possibilities for $x_4[8,9,12,13,14]$, $x_5[4,9,14]$
$\mathcal{A}_7$	Compute the expected δ-sequence ($2^5 \times 40$ S-Boxes)
Success: 2^{-9}	**Check** if it equals the expected sequence

By running a numerical optimization instead, we obtain a complexity of $2^{132.05}$ S-Boxes with a success probability ≥ 0.92, which gives an average complexity $2^{132.17}$ to reach a success. The attack has then a 27% chance of failure, which is the same for all procedures.

Acknowledgments. A.S. would like to thank Xavier Bonnetain and Ronald de Wolf for helpful discussions on quantum search and variable-time amplitude amplification. A.S. has been supported by ERC-ADG-ALGSTRONGCRYPTO (project 740972). This work has been partially supported by the French National Research Agency through the DeCrypt project under Contract ANR-18-CE39-0007, and through the France 2030 program under grant agreement No. ANR-22-PETQ-0008 PQ-TLS.

References

1. Aaronson, S., Ambainis, A.: Quantum search of spatial regions. Theory Comput. **1**(1), 47–79 (2005)
2. Alagic, G., Apon, D., Cooper, D., Dang, Q., Dang, T., Kelsey, J., Lichtinger, J., Miller, C., Moody, D., Peralta, R., et al.: Status report on the third round of the nist post-quantum cryptography standardization process. US Department of Commerce, NIST (2022)
3. Ambainis, A.: Quantum search with variable times. Theory Comput. Syst. **47**(3), 786–807 (2010)
4. Ambainis, A.: Variable time amplitude amplification and quantum algorithms for linear algebra problems. In: STACS. LIPIcs, vol. 14, pp. 636–647. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2012)
5. Ambainis, A., Kokainis, M., Vihrovs, J.: Improved algorithm and lower bound for variable time quantum search (2023), to appear in TQC 2023
6. Aono, Y., Nguyen, P.Q., Shen, Y.: Quantum lattice enumeration and tweaking discrete pruning. In: ASIACRYPT (1). LNCS, vol. 11272, pp. 405–434. Springer (2018)
7. Bernstein, D.J.: Grover vs. mceliece. In: PQCrypto. Lecture Notes in Computer Science, vol. 6061, pp. 73–80. Springer (2010)
8. Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y., Vikkelsoe, C.: PRESENT: an ultra-lightweight block cipher. In: CHES. LNCS, vol. 4727, pp. 450–466. Springer (2007)
9. Bonnetain, X., Hosoyamada, A., Naya-Plasencia, M., Sasaki, Y., Schrottenloher, A.: Quantum attacks without superposition queries: The offline simon's algorithm. In: ASIACRYPT (1). LNCS, vol. 11921, pp. 552–583. Springer (2019)
10. Bonnetain, X., Jaques, S.: Quantum period finding against symmetric primitives in practice. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2022**(1), 1–27 (2022)
11. Bonnetain, X., Naya-Plasencia, M., Schrottenloher, A.: Quantum security analysis of AES. IACR Trans. Symmetric Cryptol. **2019**(2), 55–93 (2019)
12. Brassard, G., Hoyer, P., Mosca, M., Tapp, A.: Quantum amplitude amplification and estimation. Contemporary Mathematics **305**, 53–74 (2002)
13. Chakraborty, S., Gilyén, A., Jeffery, S.: The power of block-encoded matrix powers: Improved regression techniques via faster hamiltonian simulation. In: ICALP. LIPIcs, vol. 132, pp. 33:1–33:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2019)
14. Daemen, J., Rijmen, V.: AES Proposal: Rijndael. Submission to the NIST AES competition (1999)
15. Daemen, J., Rijmen, V.: The Design of Rijndael: AES - The Advanced Encryption Standard. Information Security and Cryptography, Springer (2002)
16. Davenport, J.H., Pring, B.: Improvements to quantum search techniques for block-ciphers, with applications to AES. In: SAC. LNCS, vol. 12804, pp. 360–384. Springer (2020)
17. David, N., Naya-Plasencia, M., Schrottenloher, A.: Quantum impossible differential attacks: Applications to aes and skinny. Cryptology ePrint Archive, Paper 2022/754 (2022), https://eprint.iacr.org/2022/754, https://eprint.iacr.org/2022/754
18. Faugère, J., Horan, K., Kahrobaei, D., Kaplan, M., Kashefi, E., Perret, L.: Fast quantum algorithm for solving multivariate quadratic equations. IACR Cryptol. ePrint Arch. p. 1236 (2017)

19. Ferguson, N., Kelsey, J., Lucks, S., Schneier, B., Stay, M., Wagner, D.A., Whiting, D.: Improved cryptanalysis of Rijndael. In: FSE. LNCS, vol. 1978, pp. 213–230. Springer (2000)
20. Grassl, M., Langenberg, B., Roetteler, M., Steinwandt, R.: Applying grover's algorithm to AES: quantum resource estimates. In: PQCrypto. Lecture Notes in Computer Science, vol. 9606, pp. 29–43. Springer (2016)
21. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: STOC. pp. 212–219. ACM (1996)
22. Hosoyamada, A., Sasaki, Y.: Finding hash collisions with quantum computers by using differential trails with smaller probability than birthday bound. In: EUROCRYPT (2). Lecture Notes in Computer Science, vol. 12106, pp. 249–279. Springer (2020)
23. Jaques, S., Naehrig, M., Roetteler, M., Virdia, F.: Implementing grover oracles for quantum key search on AES and lowmc. In: EUROCRYPT (2). Lecture Notes in Computer Science, vol. 12106, pp. 280–310. Springer (2020)
24. Kachigar, G., Tillich, J.: Quantum information set decoding algorithms. In: PQCrypto. Lecture Notes in Computer Science, vol. 10346, pp. 69–89. Springer (2017)
25. Kaplan, M., Leurent, G., Leverrier, A., Naya-Plasencia, M.: Breaking symmetric cryptosystems using quantum period finding. In: CRYPTO (2). Lecture Notes in Computer Science, vol. 9815, pp. 207–237. Springer (2016)
26. Kaplan, M., Leurent, G., Leverrier, A., Naya-Plasencia, M.: Quantum differential and linear cryptanalysis. IACR Trans. Symmetric Cryptol. **2016**(1), 71–94 (2016)
27. Kimmel, S., Lin, C.Y., Lin, H.: Oracles with costs. In: TQC. LIPIcs, vol. 44, pp. 1–26. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2015)
28. Kirshanova, E., Mårtensson, E., Postlethwaite, E.W., Moulik, S.R.: Quantum algorithms for the approximate k-list problem and their application to lattice sieving. In: ASIACRYPT. LNCS, vol. 11921, pp. 521–551. Springer (2019). https://doi.org/10.1007/978-3-030-34578-5_19
29. Kuperberg, G.: Another subexponential-time quantum algorithm for the dihedral hidden subgroup problem. In: TQC. LIPIcs, vol. 22, pp. 20–34. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2013)
30. Kuwakado, H., Morii, M.: Quantum distinguisher between the 3-round feistel cipher and the random permutation. In: ISIT. pp. 2682–2685. IEEE (2010)
31. Laarhoven, T.: Search problems in cryptography. Ph.D. thesis, PhD thesis, Eindhoven University of Technology (2015)
32. Laarhoven, T., Mosca, M., van de Pol, J.: Finding shortest lattice vectors faster using quantum search. Des. Codes Cryptogr. **77**(2-3), 375–400 (2015). https://doi.org/10.1007/s10623-015-0067-5
33. Montanaro, A.: Quantum-walk speedup of backtracking algorithms. Theory Comput. **14**(1), 1–24 (2018)
34. Nielsen, M.A., Chuang, I.: Quantum computation and quantum information (2002)
35. Pring, B.: Exploiting preprocessing for quantum search to break parameters for *MQ* cryptosystems. In: WAIFI. LNCS, vol. 11321, pp. 291–307. Springer (2018)
36. Schwabe, P., Westerbaan, B.: Solving binary *MQ* with grover's algorithm. In: SPACE. LNCS, vol. 10076, pp. 303–322. Springer (2016)
37. Shor, P.W.: Algorithms for quantum computation: Discrete logarithms and factoring. In: FOCS. pp. 124–134. IEEE Computer Society (1994)

Appendix

A Search with Two Oracles and Exhaustive Key Search

A simple, but common, example of search with early aborts, as studied in Section 4, is the *search with two oracles* which was considered e.g. in [16,27]. This algorithm reduces the gate count of a Grover search where the oracle is the AND of two functions. One should note that if the metric considered is only sequential *depth* and / or if we consider a parallelized algorithm, this technique might not be as useful.

Consider a quantum search on $\{0,1\}^n$ with two test functions f_1 and f_2, implemented as two oracles O_{f_1} and O_{f_2}, with $X_1 := f_1^{-1}(1)$, $X_2 := f_2^{-1}(1)$, $|X_1| = \alpha_1^2 2^n$, $|X_2 \cap X_1| = \alpha_2^2 \alpha_1^2 2^n$ where the combined success probability $p = \alpha_1^2 \alpha_2^2$ is known. Here we use a layered QAA with two levels and two differing tests O_{f_1} and O_{f_2} (which write in two different flag results). We compute the success probability as:

$$\sin^2\left[(2k_2+1)\arcsin\left[\alpha_2 \sin\left((2k_1+1)\arcsin\alpha_1\right)\right]\right] \quad .$$

As we have seen before, α_1 does not need to be known exactly; we only need to ensure that the internal QAA amplifies only to a small success probability, and does not overamplify. Indeed, if an upper bound $\alpha_1 \leq \alpha$ is given, we have:

$$\begin{aligned}\alpha_2 \sin\left((2k_1+1)\arcsin\alpha_1\right) &\geq \alpha_2\alpha_1(2k_1+1)\left(1-\frac{(2k_1+1)^2\alpha_1^2\pi^2}{36}\right)\\ &\geq \sqrt{p}(2k_1+1)\left(1-\frac{(2k_1+1)^2\alpha^2\pi^2}{24}\right) \quad .\end{aligned}$$

We can then choose k_1 and k_2 to maximize the success probability:

$$\begin{aligned}&\sin^2\left[(2k_2+1)\arcsin\left[\sqrt{p}(2k_1+1)\left(1-\frac{(2k_1+1)^2\alpha^2\pi^2}{24}\right)\right]\right]\\ &\qquad\geq \sin^2\left[\sqrt{p}(2k_2+1)(2k_1+1)\left(1-\frac{(2k_1+1)^2\alpha^2\pi^2}{24}\right)\right] \quad .\end{aligned}$$

To do so we choose: $k_2 = \left\lfloor \pi / \left(4\sqrt{p}(2k_1+1)\left(1-\frac{(2k_1+1)^2\alpha^2\pi^2}{24}\right)\right)\right\rfloor$. This ensures that the success probability is bigger than:

$$\begin{aligned}&1-\sin^2\left[\sqrt{p}(2k_1+1)\left(1-\frac{(2k_1+1)^2\alpha^2\pi^2}{24}\right)\right]\\ &\qquad\geq 1-\sqrt{p}(2k_1+1)\left(1-\frac{(2k_1+1)^2\alpha^2\pi^2}{24}\right) \quad .\end{aligned}$$

Let T_1 and T_2 be the gate counts of O_{f_1} and O_{f_2}. The time complexity is:

$$T = k_2(2k_1+1)T_1 + k_2 T_2 \leq \frac{\pi}{4\sqrt{p}} T_1 \frac{1+\frac{T_2}{T_1(2k_1+1)}}{\left(1-\frac{(2k_1+1)^2\alpha^2\pi^2}{24}\right)} \quad . \tag{26}$$

Block Cipher Exhaustive Key Search. Consider a block cipher E_K and several given plaintext-ciphertext pairs $(P_i, C_i), i \leq \ell$, such that with overwhelming probability there is a single key K such that $E_K(P_i) = C_i$ for all i. We are looking for this K. For example, with AES-256, $\ell = 3$ would be enough. For all intents and purposes, we can consider $\ell = 5$, which works for most practical block ciphers, e.g. AES [15], PRESENT [8], etc. Furthermore, we can consider a key size $\kappa := |K| \geq 64$ and a block size $n \geq 32$.

Let trunc be a truncation of n bits to 32 positions. Let O_{f_1} and O_{f_2} be two quantum oracles evaluating the test functions:

$$\begin{cases} f_1(K) = 1 \iff \mathrm{trunc}(E_K(P_1)) = \mathrm{trunc}(C_1) \\ f_2(K) = 1 \iff \forall i, E_K(P_i) = C_i \end{cases} \tag{27}$$

Since $\ell = 5$ plaintext-ciphertexts suffice, we can consider a ratio of gate counts of $T_2/T_1 = 10$ (this is assuming that computing the truncation is twice as efficient as computing the whole ciphertext). The success probability of f_1 can be bounded using a Chernoff-Hoeffding bound (for this we have to assume that E is selected at random from all possible block ciphers): $\Pr\left(\alpha_1^2 \geq 2^{-31}\right) \leq e^{-2^{\kappa-32}/3}$. Since $\kappa \geq 64$, this event has negligible probability. So we can assume that $\alpha_1^2 \leq 2^{-31}$. The optimal choice of k_1 is $k_1 = 1480$ for which the multiplying factor is smaller than $1 + 2^{-7.625} \leq 2^{0.0073}$. This means that the cost of f_2 is completely amortized. The corresponding success probability is bigger than:

$$1 - 2^{-64}(2k_1 + 1)\left(1 - \frac{(2k_1+1)^2 2^{-31}\pi^2}{24}\right) \geq 1 - 2^{-52.47} \ . \tag{28}$$

Fact 1. *For any reasonable block cipher with block size $n \geq 32$ and key size $\kappa \geq 64$, if O_{f_1} is the oracle defined above (testing only 32 bits of a single plaintext), then the complexity of a quantum exhaustive search of the key (to reach an overwhelming success) is smaller than: $\frac{\pi}{4} 2^{\kappa/2+0.01} T_1$.*

B Search with Independent Tests

In this section, we tackle the problem of *search with many independent tests*, which arises in several cryptographic applications. It corresponds to a search with early aborts as given in Section 4.

Problem 1 (Sequence of independent tests). Let $f_1, \ldots, f_m$ be m functions: $f_i : \{0,1\}^n \to \{0,1\}$. Let $X_i = \{x \in \{0,1\}^n, \forall j \leq i, f_j(x) = 1\}$, with $X_0 = \{0,1\}^n$. Clearly we have $\forall i, X_{i+1} \subseteq X_i$. Assume that the f_i are all independent random boolean functions, which can be evaluated in time t. Sample from X_m.

B.1 Search with Independent Tests (Asymptotic)

Typically we will have $m \leq n + 3$ since with the assumption of independence, this ensures that $|X_m| = 1$ with a large probability. For intermediate values

of i, $|X_i|$ does not deviate too much from its average due to the multiplicative Chernoff-Hoeffding bound:

$$\forall \delta \geq 0, \Pr(|X_i| \geq 2^{n-i}(1+\delta)) \leq e^{-2^{n-i}\delta^2/(2+\delta)} \leq e^{-2^{n-i}\delta^2/2} \ ,$$

where we take $\delta = 2^{-(n-i)/3}$ to obtain:

$$\forall i, \Pr(|X_i| \geq 2^{n-i} + 2^{2(n-i)/3}) \leq e^{-2^{(n-i)/3-1}} \ .$$

Using a union bound, we can ensure that most of the X_i are close to their average size:

$$\begin{aligned}\Pr(\exists i \leq n-9, |X_i| \geq 2^{n-i} + 2^{2(n-i)/3}) &\leq \sum_{i=0}^{n-9} e^{-2^{(n-i)/3-1}} \leq \sum_{i=9}^{\infty} \left(e^{-1}\right)^{2^{i/3-1}} \\ &\leq e^{-4} \sum_{i=0}^{\infty} e^{-i} \leq 0.029 \ ,\end{aligned}$$

where we noticed that $2^{i/3-1} > i-5$ for $i \geq 9$.

In particular $|X_i| \leq 2^{n-i}\frac{9}{8}$, which simplifies the asymptotic computations. We will now cut the sequence $f_1, \ldots, f_n$ into a variable-time algorithm $\mathcal{A}_\ell \circ \cdots \circ \mathcal{A}_1$ (note that ℓ is a different parameter than the number of tests m; in particular $\ell < m$). Each $\mathcal{A}_i$ will run m_i successive functions f_i, so that in total $m_1 + \ldots + m_\ell = n-9$. A final search will be performed for the remaining conditions, but it will only add a constant overhead. Due to the Chernoff bounds, we know that the cumulative probabilities of success of the $\mathcal{A}_i$ are upper bounded by: $\prod_{j=1}^{i} \beta_j^2 \leq 2^{-\sum_{j=1}^{i} m_j + 1}$. So we can take for the number of iterates k_i:

$$k_1 = \left\lfloor \frac{1}{2\sqrt{9/8 \times 2\ell}} \sqrt{2^{m_1}} - \frac{1}{2} \right\rfloor, \forall i \geq 1, k_i = \left\lfloor \frac{1}{2}\sqrt{2^{m_i}} - \frac{1}{2} \right\rfloor \ ,$$

which are sufficient to ensure the conditions of Theorem 1:

$$\forall i \geq 1, \prod_{j=1}^{i} (2k_j+1)^2 \leq \frac{1}{2\ell} 2^{m_1+\ldots+m_i} \frac{8}{9} \implies \prod_{j=1}^{i} (2k_j+1)^2 \beta_j^2 = \prod_{j=1}^{i} (2k_j+1)^2 \frac{|X_i|}{2^n} \leq \frac{1}{2\ell} \ .$$

By Equation 12 the complexity of the full procedure is upper bounded by:

$$\begin{aligned}\sum_{i=1}^{\ell} \left(\prod_{j=i}^{\ell} (2k_j+1) \right) m_i &\leq \left(\sum_{i=2}^{\ell} \sqrt{2^{m_i+\ldots+m_\ell}} m_i \right) + \sqrt{2^{m_1+\ldots+m_\ell}} \sqrt{\frac{4}{9\ell}} m_1 \\ &\leq \sqrt{2^{n-9}} \left(\sqrt{\frac{4}{9\ell}} m_1 + \frac{m_2}{2^{m_1/2}} + \frac{m_3}{2^{(m_1+m_2)/2}} + \ldots + \frac{m_\ell}{2^{(m_1+\ldots+m_{\ell-1})/2}} \right) \ .\end{aligned}$$

It appears clearly that when n is very large, we can minimize the complexity by choosing $m_2, m_3, \ldots, m_\ell$ as follows:

$$m_\ell = n - 9 - \left\lfloor \log_{\sqrt{2}} n \right\rfloor, \forall i, m_{\ell-i} = \left\lfloor \log_{\sqrt{2}}^{(i)} n \right\rfloor - \left\lfloor \log_{\sqrt{2}}^{(i+1)} n \right\rfloor, m_1 = \left\lfloor \log_{\sqrt{2}}^{(\ell-1)} n \right\rfloor$$

where we have used an iterated logarithm in base $\sqrt{2}$. As long as all these numbers are strictly positive, we have for all $i \geq 1$:

$$\frac{m_i}{2^{(m_1+\ldots+m_{i-1})/2}} \leq 1$$

and so the time complexity is upper bounded by $2^{(n-9)/2}\left(\sqrt{\frac{4}{9\ell}}\log_{\sqrt{2}}^{(\ell-1)} n + \ell\right)$. We are still free to choose ℓ under the condition $k_1 \geq 1$ i.e. $\sqrt{\frac{4}{9\ell}}\sqrt{2^{m_1}} \geq \frac{3}{2}$. It can be remarked that $\ell = \mathcal{O}(\log_{\sqrt{2}}^{*} n)$, and the complexity (after the final amplification) to have a success probability $\frac{1}{2}$ is $\mathcal{O}\Big((\log_{\sqrt{2}}^{*} n)^{3/2} 2^{n/2}\Big)$.

B.2 Search with Independent Tests (Exact)

For cryptographically relevant parameters, we estimate that cutting the independent tests in three groups should be enough. We select two parameters m_1, m_2; we first perform m_1 tests, then m_2 tests, then the remaining $m - m_1 - m_2$ where $m = n + 3$ to ensure a single solution. We count the complexity in number of tests:

$$m_1(2k_1+1)(2k_2+1)(2k_3+1) + m_2(2k_2+1)(2k_3+1) + (m-m_1-m_2)(2k_3+1) \ .$$

Since the tests are independent, the probabilities of success of the three steps, $\beta_1^2 = \frac{|X_{m_1}|}{2^n}$, $\beta_2^2 = \frac{|X_{m_1} \cap X_{m_1+m_2}|}{|X_{m_1}|}$, $\beta_3^2 = \frac{1}{2^n \beta_1^2 \beta_2^2}$, can be bounded using Chernoff-Hoeffding bounds. There are on average 2^{n-m_1} elements passing the first step and $2^{n-m_2-m_1}$ elements passing the second. We have for all ε_1 and ε_2:

$$\begin{cases} \Pr\left(\left||X_{m_1}| - 2^{n-m_1}\right| \geq \varepsilon_1 2^{n-m_1}\right) \leq 2\exp\left(\frac{-\varepsilon_1^2 2^{n-m_1}}{3}\right) \\ \Pr\left(\left||X_{m_1} \cap X_{m_1+m_2}| - 2^{n-m_1-m_2}\right| \geq \varepsilon_2 2^{n-m_1-m_2}\right) \leq 2\exp\left(\frac{-\varepsilon_2^2 2^{n-m_1-m_2}}{3}\right) \end{cases} \tag{29}$$

Therefore, assuming that $n - m_1 - m_2 \geq 12$, we can take both $\varepsilon_1 = \varepsilon_2 = 2^{-4}$ and these two events occur with overwhelming probability. This gives bounds:

$$\begin{cases} \beta_1^2 \in \left[2^{-m_1}(1-\varepsilon_1); u_1^2 := 2^{-m_1}(1+\varepsilon_1)\right] \\ \beta_2^2 \in \left[2^{-m_2}\frac{1-\varepsilon_2}{1+\varepsilon_1}; u_2^2 := 2^{-m_2}\frac{1+\varepsilon_2}{1-\varepsilon_1}\right] \\ \beta_3^2 \in \left[2^{-n+m_1+m_2}\frac{1}{1+\varepsilon_2}; u_3^2 := 2^{-n+m_1+m_2}\frac{1}{1-\varepsilon_2}\right] \end{cases} \tag{30}$$

Thus, after choosing values for m_1 and m_2, we can follow the approach of Section 4.4: we optimize the complexity as a function of k_1, k_2, k_3 (counting the total number of individual tests performed) under the constraints:

$$k_1 \leq \left\lfloor \frac{\pi}{4}\frac{1}{\arcsin u_1} - \frac{1}{2} \right\rfloor, k_2 \leq \left\lfloor \frac{\pi}{4}\frac{1}{\arcsin u_2} - \frac{1}{2} \right\rfloor, k_3 \leq \left\lfloor \frac{\pi}{4}\frac{1}{\arcsin u_3} - \frac{1}{2} \right\rfloor \ . \tag{31}$$

We can then try with different m_1 and m_2 and see which ones perform best. For cryptographic parameters, n usually does not exceed 2^{10}, and we expect m_1 to be quite small, so there are not many parameters to try.

B.3 Application: Solving Binary Multivariate Quadratic Systems

As an example, we can consider the problem of solving binary quadratic equation systems using exhaustive search, which was considered in [36] and improved in [35] (note that an asymptotically better algorithm was given in [18], but its exact complexity has not been analyzed to date). Assume that we have a system of n quadratic equations in n boolean variables, with a single solution: $\forall i, f_i(\mathbf{x}) = 0$. The algorithm in [36] essentially performs a Grover search on the whole search space $\{0,1\}^n$ for $\mathbf{x}$, which tests each equation separately and checks if $f_i(\mathbf{x}) = 0$ for all x. If an equation can be tested in time t (returning a single bit), this gives a time complexity: $\left\lfloor \frac{\pi}{4} 2^{n/2} \right\rfloor nt$.

By applying our framework, for $n = 128$ we obtain an average complexity of $2^{67.93}t$ with $m_1, m_2 = 5, 10$ (instead of $2^{70.65}t$); for $n = 256$ we obtain $2^{132.02}t$ with $m_1, m_2 = 5, 12$ (instead of $2^{135.65}t$). The improvement over the naive exhaustive search is comparable to the *preprocessing* method used in [35]; however both methods are different, since the preprocessing uses the structure of the quadratic equations. It might be possible to combine them both to further reduce the cost, and we leave this as an interesting open question.

C Variable-time Amplitude Amplification without Amplitude Estimation

In this section, we show how the framework of Section 4 allows to solve the problem of amplitude amplification with a variable-time algorithm [3,4,13]. In order to fit in our framework, we consider the amplified algorithm to be classical; however, our method would still apply for any quantum algorithm, with proper definitions of the average time complexity and probability of success of each layer.

A similar algorithm was proposed in a concurrent and independent work by Ambainis, Kokainis and Vihrovs [5]. Their algorithm also performs a layered QAA with one iteration per level. With a dedicated complexity analysis (rather than our generic bound), they obtained a slightly better complexity that reduces the logarithmic factor from a power 3/2 to a power 1.

Setting. A variable-time algorithm A operates on a workspace $\{0,1\} \times C \times W$, where C is an initial choice set and W a work register. We assume that together W and C have a size of w bits. The algorithm runs for a variable number of steps, and either: • stops with a flag bit 1, which indicates a good choice; • stops with a flag bit 0, which indicates a bad choice. We let $t_{\max}$ be the maximal runnning time of A on inputs from the choice set C, and assume that a flag bit 1 can only be returned at the very last step.

For each input $c \in C$, we let $0 < t(c) \leq t_{\max}$ be the running time of A on input c. We define the average time complexity of A (in L^2) as:

$$T_2 := \sqrt{\frac{1}{|C|} \sum_{c \in C} t(c)^2} \ . \tag{32}$$

Furthermore, we let p be the *success probability* of A, i.e., the proportion of $c \in C$ which return a flag 1. Ambainis showed [3] that even if the running times $t(c)$ are not known, there exists a quantum algorithm that finds a good c in time $\mathcal{O}\left(t_{\max}\sqrt{t_{\max}} + \frac{T_2}{\sqrt{p}}(\log t_{\max})^{3/2}\right)$. In other words, this algorithm is capable of averaging the evaluation times of the individual elements (but in L^2 norm). His method is a nested QAA which stops A at certain times. Amplitude Estimation is used to estimate the times $t(c)$ on the fly, and find out how many iterations are necessary at each layer.

New Method. Our method relies on a simple quantum search with early aborts. For now, assume that $T_2, t_{\max}$ and p are known. We introduce a sequence of algorithms $A_1, \ldots, A_\ell$ which simply run some substeps of A, and at specific points, copy the current flag register to a new flag. We choose the following times: $t_i = 3^i$ where $1 \leq i \leq \ell = \lceil \log_3 t_{\max} \rceil$. So, A_1 runs the first t_1 steps of A, then A_2 runs the next $t_2 - t_1$ steps, and so on. When A has finished, A_ℓ continues to run dummy computing steps until it reaches t_ℓ.

Obviously, the corresponding search problem is equivalent to finding a good output of A. Now it remains to analyze the resulting search with early aborts. The key to our approach is to estimate, for all $i \leq \ell - 1$, the quantity $\prod_{j=1}^{i} \beta_i^2$ only depending on T_2, thanks to Markov's inequality. Then, we will be able to set iteration numbers in order to use Theorem 1.

By definition, $\prod_{j=1}^{i} \beta_i^2$ is the probability that a uniform random $c \in C$ passes the tests 1 to i included. We can relate this to $t(c)$, as follows: if all these tests are passed, it means that A runs in strictly more time steps than t_{i-1}. Thus:

$$\prod_{j=1}^{i} \beta_i^2 \leq \Pr_{c \xleftarrow{\$} C}\left(t(c) > t_{i-1}\right) \leq \Pr_{c \xleftarrow{\$} C}\left(t(c)^2 \geq t_{i-1}^2\right) \leq \frac{T_2^2}{t_{i-1}^2} \ .$$

Besides, it can be noticed that by definition of p: $\prod_{i=1}^{\ell} \beta_i^2 = p$. So, in order to use Theorem 1, we will try to satisfy the condition:

$$\forall i \leq \ell, \left(\prod_{j=1}^{i} (2k_j + 1)^2\right) \frac{T_2^2}{t_{i-1}^2} \leq \frac{1}{2\ell}$$

and in that case, we will obtain a resulting success probability greater than $\frac{p}{2}\left(\prod_{j=1}^{\ell}(2k_j+1)^2\right)$ for the nested QAA procedure.

Success Probability. We will now set the iteration numbers for the ℓ steps. We start with $k_i = 0$ for $i = 1, \ldots, i_m$ for some well chosen i_m, then $k_i = 1$ for $i = i_m + 1, \ldots \ell - 1$, then $k_\ell = 0$. The value of i_m is determined using the known value of T_2. Indeed:

$$i_m \geq \log_3\left[3\sqrt{2\ell}T_2\right] \implies 3^{i_m} \geq 3\sqrt{2\ell}T_2 \implies \frac{1}{2\ell} \leq \frac{T_2^2}{t_{i_m-1}^2} \ ,$$

and the inequalities for bigger i follow immediately. Thus we use

$$i_m = \left\lceil \log_3 \left[3\sqrt{2\ell} T_2 \right] \right\rceil$$

and lower bound the success probability of the nested QAA as:

$$P := \frac{p}{2} \prod_{j=1}^{\ell} (2k_j+1)^2 = \frac{p}{2} 3^{2(\ell - i_m - 1)} \geq 3^{2\ell-6} \frac{p}{4\ell T_2^2} \ .$$

Complexity. The gate complexity is given by Equation 12. If w is the number of qubits in the workspace of $\mathcal{A}$, we can bound it simply as follows:

$$\begin{aligned}
&\sum_{i=1}^{\ell} \left(\prod_{j=i}^{\ell} (2k_j+1) \right) (t_i - t_{i-1}) + (44(w+\ell)) \prod_{j=1}^{\ell} (2k_j+1) \\
&\quad \leq \sum_{i=1}^{i_m} 3^{\ell - i_m - 1} (3^i - 3^{i-1}) + \sum_{i=i_m+1}^{\ell} 3^{\ell-i} (3^i - 3^{i-1}) + 44(w+\ell) 3^{\ell - i_m - 1} \\
&\quad \leq (\ell - i_m) 3^{\ell} + 3^{i_m} 3^{\ell - i_m - 1} + 44(w+\ell) 3^{\ell - i_m - 1} \ .
\end{aligned}$$

The term w (the number of qubits used in the workspace of $\mathcal{A}$) is problematic here, as a priori, $\mathcal{A}$ can use up to 3^ℓ qubits. However we can remark that each O_0 operator only needs to act on the qubits that are currently non-zero. Even if $\mathcal{A}$ uses 3^ℓ qubits, when calling O_0 at level i, it needs to act at most on 3^i qubits (the number of time steps performed by $\mathcal{A}_i \cdots \mathcal{A}_1$). Thus, the second term is also dominated by $\mathcal{O}\big(\ell 3^\ell\big)$.

At this point, we can directly use Lemma 2, since we have a lower bound on the success probability: a solution is found, with probability $\frac{1}{2}$, after a procedure that applies $\mathcal{O}\left(1/\sqrt{P}\right)$ QAA iterates. This gives a final time complexity of order:

$$\mathcal{O}\left(\frac{\ell 3^\ell}{\sqrt{P}} \right) = \mathcal{O}\left(\frac{1}{\sqrt{p}} \ell^{3/2} T_2 \right) = \mathcal{O}\left((\log t_{\max})^{3/2} \frac{T_2}{\sqrt{p}} \right) \ . \tag{33}$$

A priori, we need to know T_2, $t_{\max}$ and p to define this algorithm. However an upper bound on $t_{\max}$ can be obtained from T_2 and p by: $t_{\max} \leq T_2/\sqrt{p}$, so we only need T_2 and p. As for T_2, we only need to know an upper bound, because in that case the computed values of i_m, P and $t_{\max}$ which determine the algorithm remain good. So we take as bounds increasing powers of 3 until we find a solution. The asymptotic complexity remains unchanged. Note that Lemma 2 needs only a lower bound on P, and so, only a lower bound estimate of p is necessary.