Quantum Procedures for Nested Search Problems

with Applications in Cryptanalysis

André Schrottenloher[a, 1] and Marc Stevens[2]

[1] Univ Rennes, Inria, CNRS, IRISA, Rennes, France
[2] CWI, Amsterdam, The Netherlands

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 of ℓ layers multiplies the complexity by a constant factor $(\pi/2)^\ell$.

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 generic complexity formulas and show that for concrete instances, numerical optimizations enable further improvements, reducing even more the gap to an exact quadratic speedup.

We demonstrate our framework by giving a tighter analysis 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 the foundational results of *quantum cryptanalysis*, Shor's period-finding algorithm [Sho94] 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 [AAC⁺22] 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 [Gro96], generalized into the framework of quantum amplitude amplification (QAA) [BHMT02]. Grover's algorithm generically speeds up any *black-box* search procedure. By *black-box*, we mean that the problem is entirely defined by

E-mail: andre.schrottenloher@inria.fr (André Schrottenloher), marc.stevens@cwi.nl (Marc Stevens)

[a] Part of this work was done while the author was at CWI.

Date of this document: 2024-08-20.

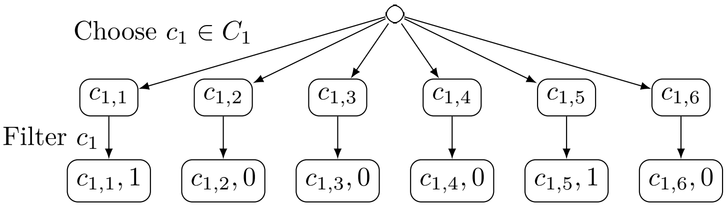

Figure 1: Exhaustive search with a single level.

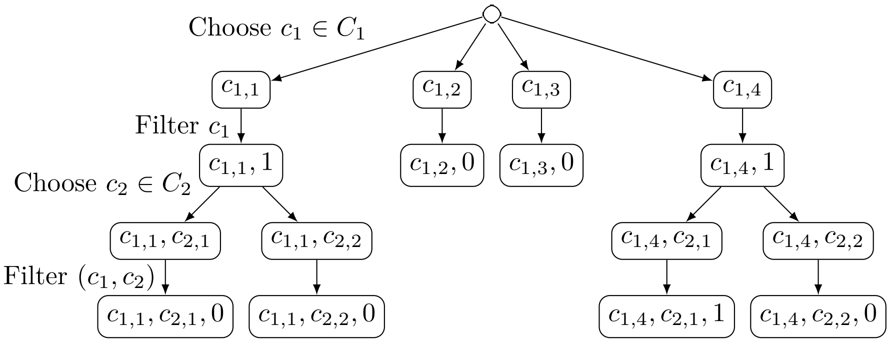

Figure 2: Nested search with two levels.

two algorithms: *sampling* at random from a given search space, and *testing* an element of this search space. Classical black-box search iterates both these algorithms until the test is passed. Quantum search reduces asymptotically the average number of iterations to its square-root. In particular, it accelerates the exhaustive search of a secret key of length $|K|$ from $\mathcal{O}(2^{|K|})$ trial encryptions to $\mathcal{O}(2^{|K|/2})$ 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 algorithms. In asymmetric cryptography, generic decoding algorithms based on information set decoding [Ber10, KT17] 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 [LMvdP15, Laa15, KMPM19].

In symmetric cryptanalysis, most quantum attacks rely to some extent on quantum search, except some dedicated attacks like [KM10, KLLN16a]. For example, the differential and linear attacks in [KLLN16b] and the rebound attacks in [HS20] rephrase a classical cryptanalysis as a nested search problem, and then replace the search steps by Grover's search or QAA.

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 solution. This situation is pictured in Figure 1, where $C_1 := \{c_{1,1}, c_{1,2}, c_{1,3}, c_{1,4}, c_{1,5}, c_{1,6}\}$ is the set of choices. It corresponds to the exploration of a tree with one level.

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, where $C_1 := \{c_{1,1}, c_{1,2}, c_{1,3}, c_{1,4}\}$ and $C_2 := \{c_{2,1}, c_{2,2}\}$). A solution is a path from the root to the last level where all filters evaluate to 1. The complexity of the search depends on the expected number of nodes on each level that need to be explored before finding the solution path.

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 simple 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 such as in Figure 2 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 would like to estimate precisely the quantum complexity of our attacks. However, the analysis of nested quantum searches raises the following problems, which previous works overcome on a case by case basis:

Overhead factor per level: Applying QAA naively results in a multiplicative overhead factor of $\pi/2$ per nesting level, see e.g. [DNPS23]. 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 can be seen in the quantum key-recovery attack on 8-round AES-256 of [BNS19].

Probability of error: Most of the time, QAA does not produce exactly the uniform superposition of solutions. Previous works often try to reduce the probability of error to negligible, leading to more complex procedures and overestimated complexities.

Fixed number of iterates: In a classical nested search like Figure 2, the probability to pass the second filter can depend on the choice made at the first level. 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 over all choices and needs to use a fixed number of iterates for each level.

Contributions and Organization. In this paper, we provide a generic framework to transform a nested search procedure into a quantum procedure. For any attack that fits in our framework, there exists a corresponding quantum algorithm whose complexity and success probability can be generically computed. We provide a formula and explain how numerical optimization can achieve 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, D_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 4, we present a quantum procedure for this problem, which performs a *search with backtracking*. It corresponds to a depth-first exploration of the tree. Our generic complexity analysis handles the three problems identified earlier, and in particular, reduces the constant overhead factor from $(\pi/2)^\ell$ to $\mathcal{O}\left(\sqrt{\ell}\right)$.

In Section 5, we showcase our framework by re-analyzing three attacks on reduced-round AES from [BNS19] and [DNPS23]. We show that the quantum complexity analysis can be entirely outsourced to our framework, and that we can gain up to a factor 2^4 in the time complexity.

In Appendix A, we present a different framework of *search with early aborts*, whose analysis is very similar but which has different applications: it considers the case of a single choice and several filters of different complexities. Such applications are given later on, including a simple algorithm for amplitude amplification of variable-time algorithms.

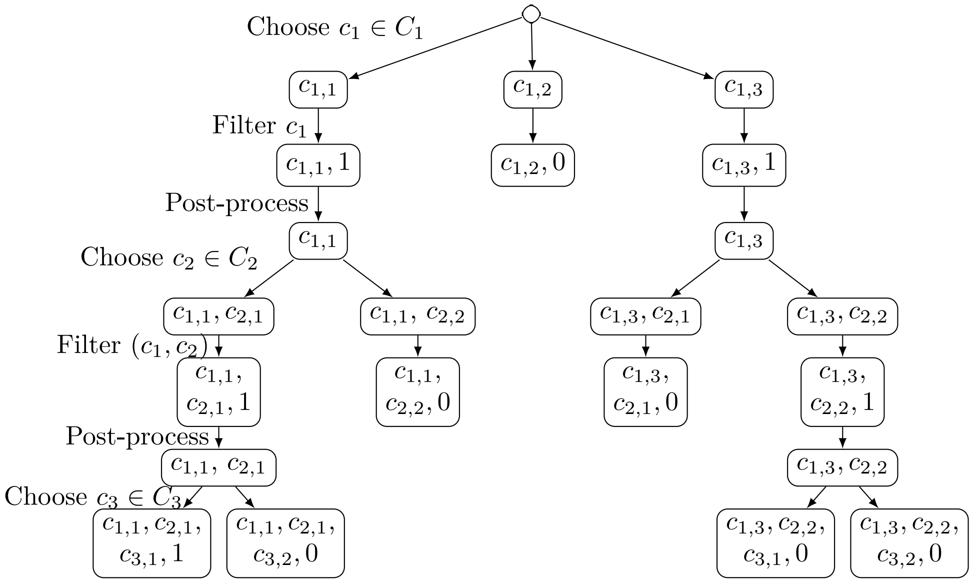

Figure 3: Example of the generic search problem that we want to solve: we look for $(c_1, c_2, c_3) \in C_1 \times C_2 \times C_3$ which evaluates to 1 at the last step.

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 [BNS19]. Our framework contains the *filter* framework, but also provides a generic complexity analysis, contrary to [BNS19]. In [DNPS23], 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 [DP20, KLL15].

Montanaro [Mon18] 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 [ANS18]. 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 [Amb10, Amb12] 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. The framework of Appendix A 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. [AKV23] proposed a new algorithm for variable-time QAA with essentially the same algorithmic structure as the one given in Appendix A. 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 the whole framework.

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:

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

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

2.1 Quantum Algorithms.

We refer to [NC16] 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, X, CX and CCX gates (also known as NOT, CNOT and Toffoli), phase-flip and controlled phase-flip Z and CZ.

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 a rather abstract definition which allows to focus on specific gates or on a specific gate set. For example, if we are interested only in the number of Toffoli gates we can set $G(CCX) = 1$ and $G(\mathcal{U}) = 0$ for any other basic gate $\mathcal{U}$. 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 [BNS19], 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: $|x\rangle|y_1, \ldots, y_M\rangle \xmapsto{\mathsf{Access}_i} |y_i\rangle|y_1, \ldots, y_{i-1}, x, y_{i+1}, y_M\rangle$. 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: $|i\rangle|x\rangle|y_1, \ldots, y_M\rangle \xmapsto{\mathsf{Access}} |i\rangle|y_i\rangle|y_1, \ldots, y_{i-1}, x, y_{i+1}, y_M\rangle$. In the *QRAQM*[1] *model*, we assume that 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. If the memory M contains only classical data, the operation of access can be implemented as: $|i\rangle|x\rangle \xmapsto{\mathsf{CAccess}}$

[1] *Quantum random-access quantum memory*, following the terminology from [Kup13].

$|i\rangle |x \oplus y_i\rangle$. Assuming that it can be done in polynomial time is the (weaker) *QRACM*[2] *model*.

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) gate count complexity G, (classical and/or quantum) memory complexity S, and success probability $\geq p$.

2.2 Amplitude Amplification.

Quantum amplitude amplification [BHMT02], abbreviated QAA in this paper, increases 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 [BHMT02], 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 $t = \lfloor \pi/(4 \arcsin \sqrt{p}) \rfloor$ iterations, the value $(2t+1)\theta$ approaches $\frac{\pi}{2}$. The success probability is at least $1 - p$. Though there exists an *exact* variant of QAA (also given in [BHMT02]), 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.

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 (the latter is a CZ gate). We implement O_0 with a single ancilla qubit by a method of Gidney [Gid15], which requires slightly less than $6n$ Toffoli gates when the input has n bits.

Lemma 1. *The O_0 operator on n qubits can be implemented with:*

$$S(O_{0,n}) = n + 1, \qquad G(O_{0,n}) = 6nG(CCX) + 2nG(X) + 2G(H) \tag{5}$$

The O operator can be implemented with 1 ancilla qubit and $G(O) = 2G(H) + G(CX)$.

We define $G_0(n) := G(O_{0,n}) + G(O)$.

[2] *Quantum random-access classical memory.*

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 [AA05]) 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 [BHMT02]. 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 returns a good outcome with probability $\geq \frac{1}{2}$ and average gate count:*

$$(2M+2)G(\mathcal{U}) + (M-1)G_0(n) \ . \tag{6}$$

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 gates is: $(M-1)G_0(n)$.

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 $|0_{\log_2 |C|}\rangle$ 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:

$$\begin{cases} A \;:\; & W & \to & W \\ & x & \mapsto & A(x) \end{cases} \quad \begin{cases} \mathcal{A} \;:\; & \mathcal{W} & \to & \mathcal{W} \\ & |x\rangle & \mapsto & |A(x)\rangle \end{cases}, \tag{7}$$

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. 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 7, $\mathcal{A}$ 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 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 [BHN+19]. 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, which can be added to 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 be a set of bit-strings of length w. Let $F_1 = \ldots = F_{\ell+1} = \{0,1\}$ be *flags*. We define the *workspace* of our algorithms as $F \times C \times W$, where $C = C_1 \times \cdots \times C_\ell$, and $F = F_1 \times \cdots \times F_{\ell+1}$. 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 the *work register*. By abuse of notation, we name these registers like the sets in which they take their values: F_i, C_i, W.

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 register W and the i-th flag F_i. Its result depends only on the current choices and state of the work register;
- D_i can only modify the work register W, and in addition D_ℓ can modify the flag $F_{\ell+1}$;
- 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 work register.

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. Also, D_ℓ acts as the final test which determines if the choices $c_1, \ldots, c_\ell$ are good.

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:

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

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:

$$\alpha_i'^2 = \Pr_{c_i \xleftarrow{\$} C_i} \left((D_i \circ A_i) \circ \cdots \circ (D_1 \circ A_1)(0_\ell, c_1^g, \ldots, c_{i-1}^g, c_i, 0_w)|_{\text{flag } i} = 1 \right) \quad . \tag{8}$$

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:

$$\forall i, \alpha_i^2 \alpha_i'^2 = \frac{1}{|C_i|} \implies \forall i, \alpha_i^2 = \frac{2^{-n_i}}{\alpha_i'^2} \quad . \tag{9}$$

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 Applications to Cryptanalysis

While its applications are more general, the definition of our framework is motivated by the occurrences of such search problems in symmetric cryptanalysis, specifically key-recovery attacks on block ciphers. While specific examples will be covered in Section 5, we give here a high-level overview.

Let E_K be a block cipher with key K. Most key-recovery attacks start from a *distinguisher*, i.e., an algorithm that distinguishes r rounds of the cipher from a random permutation. These r rounds are extended by several key-recovery rounds.

Given access to the black-box cipher E_K, one can guess the key material k involved in these additional rounds, reduce the cipher to r rounds, and run the distinguisher: if k is guessed correctly, it returns 1, otherwise it should return 0. Thus, the naive attack is an instance of a single-level exhaustive search.

Most of the time, this single-level search is improved by separating k into subkey guesses $k = k_1| \dots |k_\ell$. In our framework, these k_i correspond to *choice sets*. At each new guess, one performs intermediate computations which amortize the cost of computing the entire distinguisher. Let us take a few examples.

- Differential cryptanalysis: a *differential distinguisher* tests if a given difference of plaintexts maps to a given difference of ciphertexts with larger probability than for a random permutation. A differential key-recovery attack starts from a large set of pairs $(P, C = E_K(P)), (P', C' = E_K(P'))$ and tries to find, for each choice of k, how many pairs $(P, P'), (C, C')$ satisfy the internal differential property, by partially encrypting or decrypting them. However, at each new guess of k_i, the number of pairs which can still satisfy the differential is reduced; computing this new set corresponds to our algorithm D_i.

- Impossible differential cryptanalysis (see Section 5): the situation is similar, except that the right key is the one for which no pair has remained valid after all the sieving steps.

- Partial sums technique and integral attacks (see Section 5): an integral distinguisher takes a sum (of one or several output bits) over a structured set of inputs of the cipher, which should be 0. To compute this sum for all choices of k, we separate it into several parts, and count how many times these parts take a given value. Each new guess of key material leads to a recomputation of the table of counters.

4 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.

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.

This section is organized as follows. In Subsection 4.1, we start by giving a classical backtracking algorithm with several nested loops. The nesting structure is the same in the quantum algorithm, which is given in detail in Subsection 4.2. We give here a general result on the success probability (Lemma 3). Afterwards in Subsection 4.3 and later we analyze the algorithm, and show how to choose the number of iterates and how to compute its complexity. We prove in these computations that with ℓ levels of nesting, the "naive" overhead factor $(\pi/2)^\ell$ can be optimized into $\sqrt{\ell}$.

4.1 Classical Backtracking Algorithm

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

Recall that we have a sequence of filtering functions $A_1, \dots, A_\ell$ and of post-processing functions $D_1, \dots, D_\ell$ acting in place on the workspace $F \times C \times W$. From them, we define a sequence of algorithms A'_i (Algorithm 1) and B_i (Algorithm 2) 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 $(c_1, \dots, c_{i-1}) = (c_1^g, \dots, c_{i-1}^g)$ (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 1 A'_i: finds the next choice that passes the filter at step i.

```
Workspace: $F_1, \dots, F_\ell$, $C_1, \dots, C_\ell$, $W$
Modifies: $C_i, F_i, W$
repeat
    Increment the value $c_i$ stored in register $C_i$
    Compute $A_i$ in place                              ▷ Overwrites $w, f_i$
until $f_i = 1$ or the choice register overflows
```

Algorithm 2 B_i: given a partial path of choices $(c_1, \dots, c_{i-1})$ (i.e., a node in the tree), finds whether $(c_1, \dots, c_{i-1}) = (c_1^g, \dots, c_{i-1}^g)$, and in that case, completes the solution path.

```
Workspace: $F_1, \dots, F_\ell$, $C_1, \dots, C_\ell$, $W$
Modifies: all registers
Initialize a counter in register $C_i$ to value 0
repeat
    Compute $A'_i$ in place                       ▷ Overwrites $c_i, w, f_i$
    if $f_i = 1$ then                             ▷ A new path to explore
        Compute $D_i$ in place                    ▷ Overwrites $w$ and possibly $f_{\ell+1}$
        Compute $B_{i+1}$ in place                ▷ Overwrites all registers from $i+1$ to $\ell$
until $f_i = 0$ or $f_{\ell+1} = 1$
  ▷ If $f_{\ell+1} = 1$, we found the solution path $(c_1^g, \dots, c_\ell^g)$, so we must stop here and the
  choice registers $(c_1, \dots, c_\ell)$ will contain this path. If $f_i = 0$, we stopped because there
  wasn't any path left to explore.
```

Note that we use a counter to explore the choice sets C_i, that makes the search deterministic (the register C_i 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.

4.2 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 1 and Algorithm 2 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 $(k'_1, \ldots, k'_\ell)$ 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 3 and Algorithm 4.

Algorithm 3 $\mathcal{A}'_i$: filters the choices at step i.

```
Workspace: $F$, $C_1, \ldots, C_\ell$, $W$
Modifies: $C_i, F_i, W$
Apply a Hadamard transform $H$ on $C_i$
Repeat $k'_i$ times
    Compute $\mathcal{A}_i$
    Flip the phase if the $i$-th flag qubit $F_i$ is 1
    Uncompute $\mathcal{A}_i$
    Apply $H$, $-O_0$, and another $H$, all on $C_i$
EndRepeat
Compute $\mathcal{A}_i$
```

Algorithm 4 $\mathcal{B}_i$: performs a QAA on the algorithm $\mathcal{B}_{i+1} \circ \mathcal{D}_i \circ \mathcal{A}'_i$.

```
   Workspace: $F$, $C_1, \ldots, C_\ell$, $W$
   Modifies: all registers ($F_i$, $C_i$, $W$) numbered from $i$ to $\ell$
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 $\ell + 1$-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 $C_i, C_{i+1}, \ldots, C_\ell$
6:     Compute $\mathcal{B}_{i+1} \circ \mathcal{D}_i \circ \mathcal{A}'_i$
7: EndRepeat
```

Remark 6. The reason why we need to apply O_0 only to the *choice* registers at Step 5 in Algorithm 4 is not obvious. It follows from the fact that among the registers numbered from i to $\ell + 1$, on which $\mathcal{B}_i$ acts, only the choice registers are non-zero at this point. Indeed, the uncomputation of $\mathcal{B}_{i+1} \circ \mathcal{D}_i \circ \mathcal{A}'_i$ brought back W to 0 and erased $F_i, \ldots, F_{\ell+1}$. By definition, a qubit that is always 0 can be removed from O_0 without changing its effect on the circuit.

Success Probability. 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 register, and 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 register 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 register contain a valid output of

$\mathcal{D}_{i-1}$, and $\mathcal{D}_i$ when it contains a valid output of $\mathcal{A}_i$. In order to simplify the notation, we define 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* $c_1^g, \dots, c_{i-1}^g$, $\mathcal{B}_i$ will output the complete good path. By definition ν_1^2 is the success probability of our complete algorithm. We have:

$$\begin{cases} \forall i \geq 2, & \nu_i = \| P_\ell^1 \mathcal{B}_i \left| 1_{i-1} 0_{\ell-i+2} \right\rangle \left| c_1^g, \dots, c_{i-1}^g \right\rangle \| \\ & \nu_1 = \| P_{\ell+1}^1 \mathcal{B}_1 \left| 0_\ell 0_c \right\rangle \| \end{cases} \tag{10}$$

Lemma 3. *Let $\nu_{\ell+1} = 1$ by convention, then for all $i \leq \ell$:*

$$\nu_i = \sin\left[(2k_i + 1) \arcsin\left[\nu_{i+1} \alpha_i \sin((2k_i' + 1)\arcsin(\alpha_i'))\right]\right] \quad . \tag{11}$$

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.:

$$\forall i, \forall (c_1, \dots, c_{i-1}) \neq (c_1^g, \dots, c_{i-1}^g), \| P_{\ell+1}^1 \mathcal{B}_i \left| 1_{i-1} 0_{\ell-i+2} \right\rangle \left| c_1, \dots, c_{i-1} \right\rangle \| = 0 \quad .$$

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

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

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

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

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 $\frac{1}{\sqrt{2^{n_i}} \alpha_i'} = \alpha_i$, we see that the output of $\mathcal{A}_i'$ is a superposition of flags and paths where $(c_1^g, \dots, c_{i-1}^g, c_i^g)$ has amplitude: $\alpha_i \sin((2k_i' + 1)\arcsin \alpha_i')$.

Then, we apply $\mathcal{B}_{i+1} \circ \mathcal{D}_i$. For $i = \ell$, $\mathcal{B}_{\ell+1} = I$ does nothing, but $\mathcal{D}_\ell$ flags the good subpath. Otherwise $\mathcal{D}_i$ does not modify any flag, but we use the recurrence hypothesis on $\mathcal{B}_i$. In both cases the good subpath $(c_1^g, \dots, c_{i-1}^g, c_i^g)$ is flagged with probability ν_{i+1}^2, so we have:

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

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. □

4.3 Choosing the Iteration Numbers

We have to determine the iteration numbers k_i and k'_i to maximize the success probability given by Lemma 3. For this, we do not need to know α_i and α'_i exactly, but only two intervals of the form:

$$\begin{cases} l_i'^2 \leq \alpha_i'^2 \leq u_i'^2 \\ l_i^2 := \frac{1}{u_i'^2 |C_i|} \leq \alpha_i^2 \leq \frac{1}{l_i'^2 |C_i|} := u_i^2 \end{cases} \tag{12}$$

where the second is deduced from the first by $\alpha_i^2 \alpha_i'^2 = \frac{1}{|C_i|}$. 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 ($\alpha'_i = 1$).

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

Lemma 4. *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 3. Then we have: $\nu_1^u \geq \nu_1 \geq \nu_1^l$.*

Proof. These upper bounds on k_i ensure that, regardless of the exact value of α_i, all inputs to sin stay in the interval $[0; \pi/2]$ where the function is increasing. The bounds on ν_1 then follow by a simple induction. □

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 3 into lower and upper bounds on the ν_i. The proof is almost the same as Lemma 8, except that the order of indices has changed, and new factors intervene due to the intermediate QAAs $\mathcal{A}'_i$.

Lemma 5. *Let $\nu_{\ell+1} = 1$. 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:*

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

Proof. We start from Equation 11, where we have replace α_i, α'_i by l_i, l'_i. We have $\nu_{\ell+1} = 1$ and for all $i \leq \ell$:

$$\nu_i = \sin\left[(2k_i + 1)\arcsin\left[\nu_{i+1} l_i \underbrace{\sin((2k'_i + 1)\arcsin(l'_i))}_{:=\sqrt{s'_i}}\right]\right] \ .$$

For the upper bound, we use the inequality:

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

which leads to $|\nu_i| \leq (2k_i + 1) l_i |\nu_{i+1}| \sqrt{s'_i}$. For the lower bound, we use the bounds on sin, then on arcsin:

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

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

We now state our main theorem, which follows entirely from Equation 13.

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

Proof. We start by unfolding recursively the upper bound in Equation 13: $\forall i, \nu_i^2 \leq \prod_{j=i}^{\ell} s_j s_j'$. Then we replace this in the lower bound:

$$\forall i, \nu_i^2 \geq s_i s_i' \nu_{i+1}^2 \left(1 - \prod_{j=i}^{\ell} s_j s_j'\right) \quad , \tag{14}$$

and we unfold it recursively:

$$\nu_1^2 \geq \left(\prod_{i=1}^{\ell} s_i s_i'\right) \prod_{i=1}^{\ell} \left(1 - \left(\prod_{j=i}^{\ell} s_j s_j'\right)\right) \quad . \tag{15}$$

Since by assumption: $\prod_{j=i}^{\ell}(s_j s_j') \leq 1$, we can use the generalized Bernoulli's inequality[3] to simplify the lower bound:

$$\nu_1^2 \geq \left(\prod_{i=1}^{\ell} s_i s_i'\right) \left(1 - \sum_{i=1}^{\ell} \left(\prod_{j=i}^{\ell} s_j s_j'\right)\right) \quad . \tag{16}$$

The bound $\prod_{j=i}^{\ell} s_j s_j' \leq \frac{1}{2\ell}$ allows to obtain: $\nu_1^2 \geq \frac{1}{2}\left(\prod_{i=1}^{\ell} s_i s_i'\right)$, 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 $\sqrt{\ell}$.

4.4 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 count of $\mathcal{A}$ and $S(\mathcal{A})$ the number of qubits on which it acts.

For the space complexity, we count all the registers and add the two ancilla qubits required by O_0 and O:

$$\forall i, S(\mathcal{B}_i) = \ell + 3 + w + \sum_{1 \leq i \leq \ell} n_i \quad . \tag{17}$$

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

$$(2k_i+1)\Big(G(\mathcal{B}_{i+1})+G(\mathcal{D}_i)+(2k_i'+1)\left(G(\mathcal{A}_i) + n_i G(H)\right)+k_i' G_0(n_i)\Big)+k_i G_0(n_i+\ldots+n_\ell) \quad ,$$

where $G(\mathcal{B}_{\ell+1}) = 0$ since $\mathcal{B}_{\ell+1}$ does nothing. We simplify it into:

$$G(\mathcal{B}_i) \leq (2k_i+1)\left(G(\mathcal{B}_{i+1})+G(\mathcal{D}_i)+(2k_i'+1)\left(G(\mathcal{A}_i) + n_i G(H) + \frac{1}{2}G_0(n_i)\right)+\frac{1}{2}G_0(n)\right) \quad . \tag{18}$$

Finally, we choose the maximal iteration numbers such that the condition of under-amplification (Theorem 1 and Lemma 5) is satisfied. We deduce the success probability of $\mathcal{B}_1$ and its gate count. This completes the "analytic" study. Note that this is a guaranteed strategy, but not the optimal one.

[3] This technique is inspired by Appendix A in [AKV23].

Theorem 2. *Choose:*

$$\begin{cases} k_\ell = \max\left(\left\lfloor \frac{1}{2\sqrt{2\ell}}\frac{1}{u_\ell} - \frac{1}{2}\right\rfloor, 0\right) \\ \forall i < \ell, k_i = \max\left(\left\lfloor \frac{1}{2}\frac{1}{u_i} - \frac{1}{2}\right\rfloor, 0\right) \\ \forall i \leq \ell, k'_i = \max\left(\left\lfloor \frac{\pi}{4\arcsin(u'_i)} - \frac{1}{2}\right\rfloor, 0\right) \end{cases} \tag{19}$$

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

$$\nu_1^2 \geq \frac{1}{2}\prod_i (2k_i+1)^2 l_i^2 \prod_i \sin^2\left((2k'_i+1)\arcsin(l'_i)\right) \quad , \tag{20}$$

and its gate count is upper bounded by:

$$\sum_{i=1}^{\ell}\left(\prod_{j=i}^{\ell}(2k_j+1)\right)\left(G(\mathcal{D}_i) + (2k'_i+1)\left(G(\mathcal{A}_i) + n_i G(H) + \frac{1}{2}G_0(n_i)\right) + \frac{1}{2}G_0(n)\right) \ . \tag{21}$$

Proof. We check that the iteration numbers satisfy the conditions of Lemma 4, 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 $\sin^2\left((2k'_i+1)\arcsin(l'_i)\right)$ is 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: $\forall i, (2k_i+1)^2 l_i^2 \leq (2k_i+1)^2 u_i^2 \leq 1$ and $(2k_\ell+1)^2 l_\ell^2 \leq (2k_\ell+1)^2 u_\ell^2 \leq \frac{1}{2\ell}$ so we can use Theorem 1 to conclude. The gate count is obtained by recursively unfolding Equation 18. □

As in Theorem 3, the constraint on k_ℓ (which is necessary for Theorem 1 to hold) creates an inverse-linear lower bound in ℓ. Consequently, in order to bring the probability of success of $\mathcal{B}_1$ to a constant, we need to use a QAA with $\mathcal{O}(\sqrt{\ell})$ iterates. This is why the $(\pi/2)^\ell$ overhead factor is replaced by $\sqrt{\ell}$. In our applications, we compute this exactly using an "overcooked" QAA (Lemma 2).

4.5 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 2 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:

$$\begin{cases} \nu_\ell^l = \sin\left[(2k_\ell+1)\arcsin(l_\ell)\right] \\ \forall i, \nu_i^l = \sin\left[(2k_i+1)\arcsin\left[\nu_{i+1}^l l_i \sin((2k'_i+1)\arcsin l'_i)\right]\right] \quad . \end{cases} \tag{22}$$

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 [JNRV20], 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.

4.6 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 Subsection 2.1 for an overview of quantum memory models). We can summarize it with the following rules:

- All registers F_i, C_i, W, 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 read/write 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 read-only 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).

5 Applications to AES cryptanalysis

In this section, we demonstrate our framework by improving the algorithms and complexity of several quantum attacks on the AES block cipher given in [BNS19] and [DNPS23]: a Square attack, a Demirci-Selçuk Meet-in-the-middle attack (DS-MITM) and an Impossible Differential attack (ID).

Preliminaries on AES. The AES [DR02] 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. Since the AES S-Box S is the only nonlinear component, it dominates the cost of quantum circuits, and the complexity can be estimated by counting S-Box circuits[4].

5.1 Workflow and Results

Our strategy to turn a classical attack into a quantum attack consists in the following steps.

1. We write the attack algorithm as a search problem in multiple levels as defined in Subsection 3.2. We define the filtering functions A_i, the post-processing functions D_i and the choice sets C_i at each level, and determine their parameters:

[4] 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.

   - Size of the choice sets
   - Lower and upper bounds on the success probability of each A_i that can be expected from the solution path (if there is no filtering, then the probability of success is simply 1)

2. We define the corresponding quantum algorithms $\mathcal{A}_i$ and $\mathcal{D}_i$ and determine their space and gate complexities. Often, this simply consists in checking that A_i and D_i can be implemented reversibly and without erasing from the workspace.

3. The nested quantum search algorithm is then generically defined by our framework. It remains to find the number of iterates of each QAA (k_i, k'_i appearing in Algorithm 3 and Algorithm 4), which determine its gate count. We use one of the two methods:
   - Analytic complexity formula (Theorem 2)
   - Numerical optimization of the total gate count (Subsection 4.5)

4. We return the corresponding gate count and lower bound on the probability of success. If we used the analytic complexity formula, we use the "overcooked" QAA to bring the probability to 0.5 (Lemma 2).

Results. Our results are summarized in Table 1. The memory usage remains identical to the previous works: approximately 2^{25} qubits with QRAQM access, and 2^{36} classical memory without QRACM access for the Square attack; 2^{88} classical memory without QRAQM / QRACM for the DS-MITM attack (with a small number of qubits); $2^{78.5}$ QRAQM for the ID attack.

We can observe that for the 6-round quantum 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.

The ID attack is an intermediate case in which the analytic formula performs worse, but the numerical optimization allows to gain a small factor.

5.2 Square Attack on AES

The Square attack on 6-round AES using the partial sums technique [FKL⁺00] is a good example of a simple backtracking algorithm. Its quantum version is given in Appendix A.2 of [BNS19]. The path of the attack is reproduced in Figure 4. It is based on the following 3-round integral distinguisher: by encrypting a set of 2^8 plaintexts which vary only in byte 0 (a δ-set), after 3 full rounds of AES, each byte of the output 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 4) takes all 2^{32} possible values, while the rest is constant. A structure maps to 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 $k_6[0, 1, 2, 3]$ (● 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

Table 1: Estimation of attack complexities, in number of S-Box circuits. "Analytic": using Theorem 2. "Optimized": using numerical optimization as per Subsection 4.5. "Normalized Time": time complexity divided by the success probability.

Attack		[BNS19]	Analytic	Optimized
AES Square (6 rounds)	Time Success prob. Normalized Time	$2^{44.73}$ 1 $2^{44.73}$	$2^{48.24}$ 0.5 $2^{49.24}$	$2^{44.81}$ 0.98 $2^{44.84}$
AES-256 DS-MITM (8 rounds)	Time Success prob. Normalized Time	$2^{136.17}$ 0.73 $2^{136.62}$	$2^{133.71}$ 0.5×0.73 $2^{135.16}$	$2^{132.05}$ 0.92×0.73 $2^{132.62}$
Attack		[DNPS23]	Analytic	Optimized
AES ID (7 rounds)	Time Success prob. Normalized Time	$2^{101.5}$ 1 $2^{101.5}$	$2^{103.41}$ 0.5 $2^{104.41}$	$2^{100.78}$ 0.97 $2^{100.82}$

structure. This sum is expressed as:

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

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 5 amortizes this cost thanks to intermediate tables: this is a common technique in integral cryptanalysis.

Our formalization in the framework of Section 3 is the following: for each key byte $k_6[0,1,2,3], u_5[0]$ we have a choice set of size 2^8. There is no filtering: we must make a choice for all key bytes before computing the final sum, and checking whether our choice was good or not. Therefore, each algorithm $\mathcal{A}_i$ has success probability 1, and no post-processing layer is needed: $\mathcal{D}_i = I$ at all layers. In $\mathcal{A}_i$, given the new key guess, we must construct the table for the next step. We notice that the classical computation of a table, i.e., computing T_3 from T_2 (resp. T_2 from T_1) is a reversible process, as long as we keep the previous table in memory: we just need to perform a series of increments on the counters, and in the reverse operation, we decrement instead.

Using the generic formula of Theorem 2, we obtain a worse time complexity than the one given in [BNS19]. 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 [BNS19] 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 Subsection 4.5) "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.

5.3 Impossible Differential Attack on 7-round AES

We consider the quantum impossible differential (ID) attack given in [DNPS23] (Section 6.2, first attack), which targets AES-192 and AES-256. It runs in total quantum time

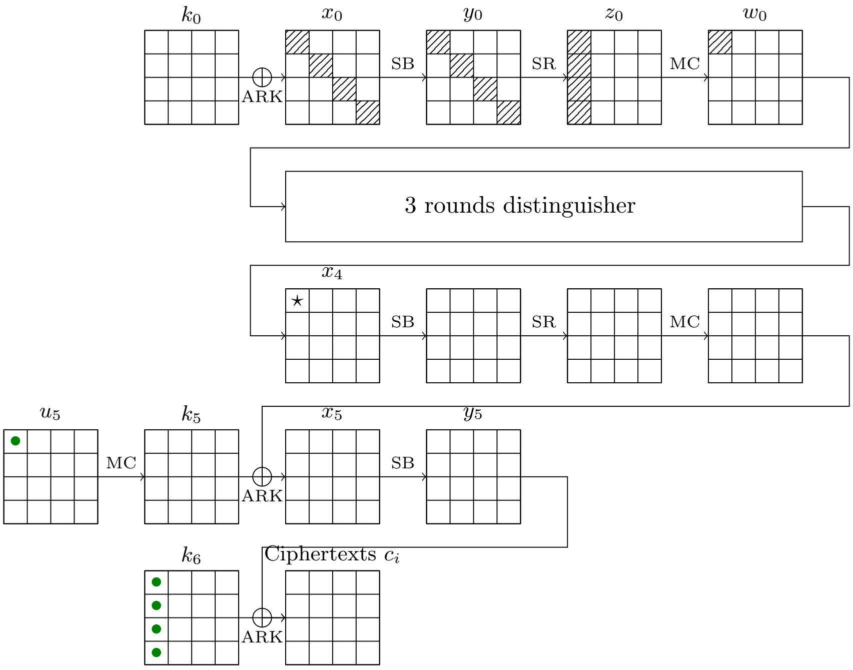

Figure 4: Path of the AES Square attack

Algorithm 5 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: $w_1 = 8 \times 2^{24} \times 32 = 2^{32}$
                  { 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: $t_1 \oplus a_2 S^{-1}(t_2 \oplus k_6[2]), t_3$
                  { by accessing $T_1$. Build a table $T_2$ of $2^{16}$ entries that stores, for each
$\mathcal{A}_2$   { 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: $w_2 = 8 \times 2^{16} \times 32 = 2^{24}$
                  { 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 $8 \times 2^{16} = 2^{19}$ S-Boxes and $w_3 = 8 \times 2^8 \times 32 = 2^{16}$ qubits.
8 bits            Choose $u_5[0]$
                  { Using the table, compute the sum.
$\mathcal{A}_4$ and $\mathcal{D}_4$ { This costs: $8 \times 2^8 \times 1$ S-Boxes and additional small computations,
                  { and approx. $8 \times 32 = 2^8$ qubits.
```

$2^{101.5}$ (counted relatively to an S-Box) and using $2^{78.5}$ QRAQM (counted relatively to the block size).

ID attacks were introduced independently by Knudsen [Knu98] and Biham, Biryukov and Shamir [BBS05]. They rely on an impossible differential distinguisher $\Delta_i \not\to \Delta_o$, defined by an input difference Δ_i which *cannot* propagate to the output difference Δ_o through some rounds of the cipher. Furthermore, truncated differential patterns are used, so Δ_i and Δ_o should be thought of as sets of differences.

The ID attack runs in two phases:

1. Pair generation: the attacker calls the cipher as a black-box and produces many input-output pairs satisfying a given truncated differential pattern.

2. Pair filtering: the attacker finds a subkey for which no pair propagates internally to the impossible differential pattern. Starting from a sufficient number of pairs, it is expected that a single guess of the subkey will be in this situation.

The attack of [DNPS23] reuses a pattern from [MDRM10], which is represented in Figure 5. The propagation of the differential from the input and outputs towards the internal ID distinguisher depends on 16 bytes of key, so the set of subkeys to sieve is of size 2^{128}.

The analysis of the pair generation step in [DNPS23] shows that $N = 2^{78.5}$ pairs are sufficient to ensure that a single solution exists in the pair filtering step. Furthermore, these pairs can be obtained in quantum time $2^{99.8}$ using a quantum collision algorithm. They are stored in a table $\mathcal{T}_0$. In the following, we focus on the pair filtering step.

Pair Filtering Algorithm. The pair filtering algorithm follows a backtracking approach which is very similar to the partial sums technique, and translates effortlessly into our framework. It guesses the subkey in several groups, and each time, reduces the set of pairs by discarding those which do not follow the truncated differential path of Figure 5. We define the following *intermediate tables*:

- $\mathcal{T}_1(k_0[0,5,10,15])$: the set of pairs (P, P') in $\mathcal{T}_0$ such that $w_0[1] = w'_0[1]$ and $w_0[3] = w'_0[3]$, of expected size $N2^{-16}$

- $\mathcal{T}_2(k_0[0,5,10,15], k_0[2,7,8,13])$: the set of pairs in $\mathcal{T}_1$ such that $w_0[9] = w'_0[9]$ and $w_0[11] = w'_0[11]$, of expected size $N2^{-32}$

- $\mathcal{T}_3(k_0, k_7)$: the set of pairs in $\mathcal{T}_2$ such that $MC^{-1}(w_5[0,1,2,3])$ is active only on byte 0, of expected size $N2^{-32-24}$

- $\mathcal{T}_4(k_0, k_1, k_7)$: the set of pairs in $\mathcal{T}_3$ such that $w_1[0] = w'_1[0]$ and $w_1[10] = w'_1[10]$, i.e., the truncated differential path at round 1 is satisfied.

By assumption, $N = 2^{78.5}$ ensures that $\mathcal{T}_4(k_0, k_1, k_7)$ is empty only when (k_0, k_1, k_7) is guessed right. While the size of the intermediate tables may vary depending on the current subkey guess, under a mild heuristic assumption, [DNPS23] shows that no intermediate table should exceed twice its expected size. The computation of each table is done by going through the previous one, and for each pair, computing a new *test function* which checks whether the new condition is satisfied. Each test function requires 8 S-Boxes.

Our backtracking algorithm has a similar structure as the attack in [DNPS23], but it has a tighter complexity analysis.

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

We consider the Demirci-Selçuk Meet-in-the-Middle key-recovery attack on 8-round AES-256 given in [BNS19]. We refer to [BNS19] for more details on this attack.

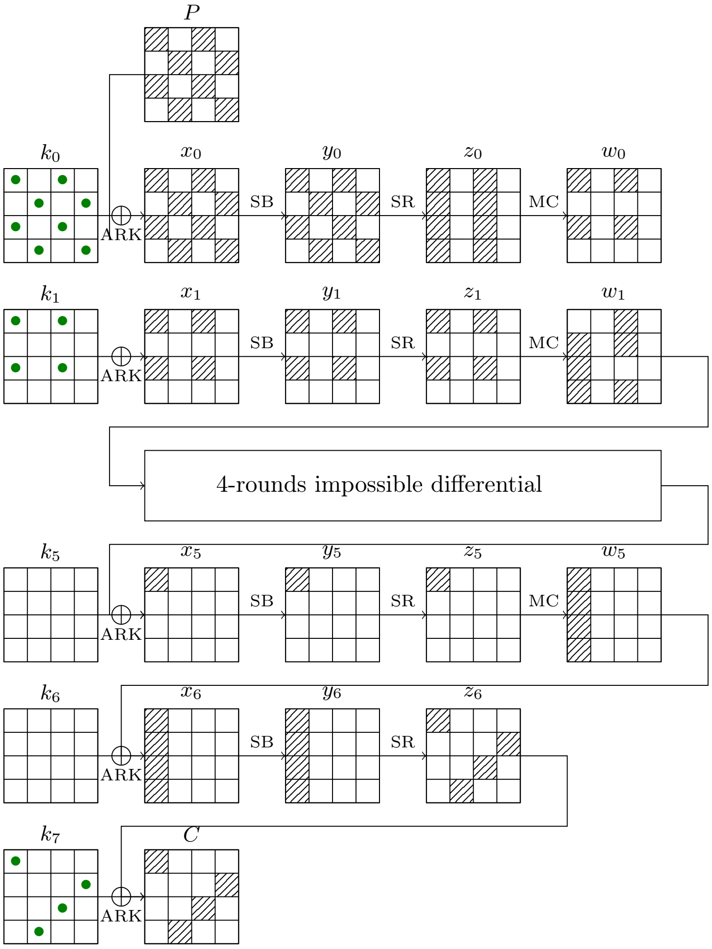

Figure 5: Path of the ID attack of [MDRM10, DNPS23].

Algorithm 6 ID attack on 7-round AES.

```
            Compute the table $\mathcal{T}_0$ of $2^{78.5}$ pairs
32 bits     Choose $k_0[0,5,10,15]$
$\mathcal{A}_1$      { Compute $\mathcal{T}_1(k_0[0,5,10,15])$
            { This costs $\leq 8 \times 2^{78.5}$ S-Boxes and the new table is of size $\leq 2^{63.5}$
32 bits     Choose $k_0[2,7,8,13]$
$\mathcal{A}_2$      { Compute $\mathcal{T}_2(k_0[0,5,10,15], k_0[2,7,8,13])$
            { This costs $\leq 8 \times 2^{63.5}$ S-Boxes and the new table is of size $\leq 2^{47.5}$
32 bits     Choose $k_7[0,7,10,13]$
$\mathcal{A}_3$      { Compute $\mathcal{T}_3(k_0[0,5,10,15], k_0[2,7,8,13], k_7[0,7,10,13])$
            { This costs $\leq 8 \times 2^{47.5}$ S-Boxes and the new table is of size $\leq 2^{23.5}$
32 bits     Choose $k_1[0,2,8,10]$
            { Compute $\mathcal{T}_4(k_0, k_1, k_7)$
$\mathcal{A}_4$ and $\mathcal{D}_4$ { This costs $\leq 8 \times 2^{23.5}$ S-Boxes
            { If $\mathcal{T}_4$ is empty then this is the solution
```

The attack uses a 5-round distinguisher which is extended forwards and backwards by several rounds using key guesses. In Figure 6 the distinguisher is placed between x_1 and x_6. It has the following property: given a pair of states satisfying the full differential path of Figure 6, i.e., with a nonzero difference only in $x_1[3]$ and $x_6[5]$, if we encrypt 2^8 plaintexts which make the byte $x_1[3]$ take 2^5 different values, then the sequence of corresponding differences in $w_6[5]$ takes only one out of 2^{192} possibilities (instead of 2^{256} for a random sequence). During the attack, these possibilities are enumerated on the fly. We can easily check on Figure 6 that when the green key bytes are guessed, and when a good plaintext-ciphertext pair is chosen, we can make $x_1[3]$ vary, compute backwards to determine a set of 2^5 plaintexts, encrypt them, compute backwards from the ciphertexts and obtain the differences in $w_6[5]$.

The attack algorithm has then the following structure. One first guesses the required key bytes $k_6[0, 5, 10, 15], k_1[3], u_7[1], u_8[0, 7, 10, 13]$, 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. First, several state differences for the initial pair of plaintexts are guessed. From these differences, one deduces several internal state values, by solving differential equations for the AES S-Box (find a byte x such that $S(x \oplus \Delta) \oplus S(x) = \Delta'$). These differential equations can have 0 ("no match" in Algorithm 7), 2 or 4 solutions. Finally, four "state-key" equations are given in [BNS19] to relate the internal state values with the key guess, which reduces the number of required guesses. Therefore, for each guess of 10 key bytes, there are on average only $2^{192-4\times 8} = 2^{160}$ possible δ-sequences to enumerate.

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 [BNS19]

- some steps ($\mathcal{A}_3$ to $\mathcal{A}_6$ in Algorithm 7) have varying success probabilities, depending on the current key and state guesses. It is estimated in [BNS19] that this probability varies less than by 2^{-8}

- the computation of the state-key equations in $\mathcal{D}_6$ costs less than 2^{10} S-Boxes (most of these computations are only linear)

- at most 4 different values are found after solving the state-key equations in $\mathcal{D}_6$

Using Theorem 2, 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 [BNS19].

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

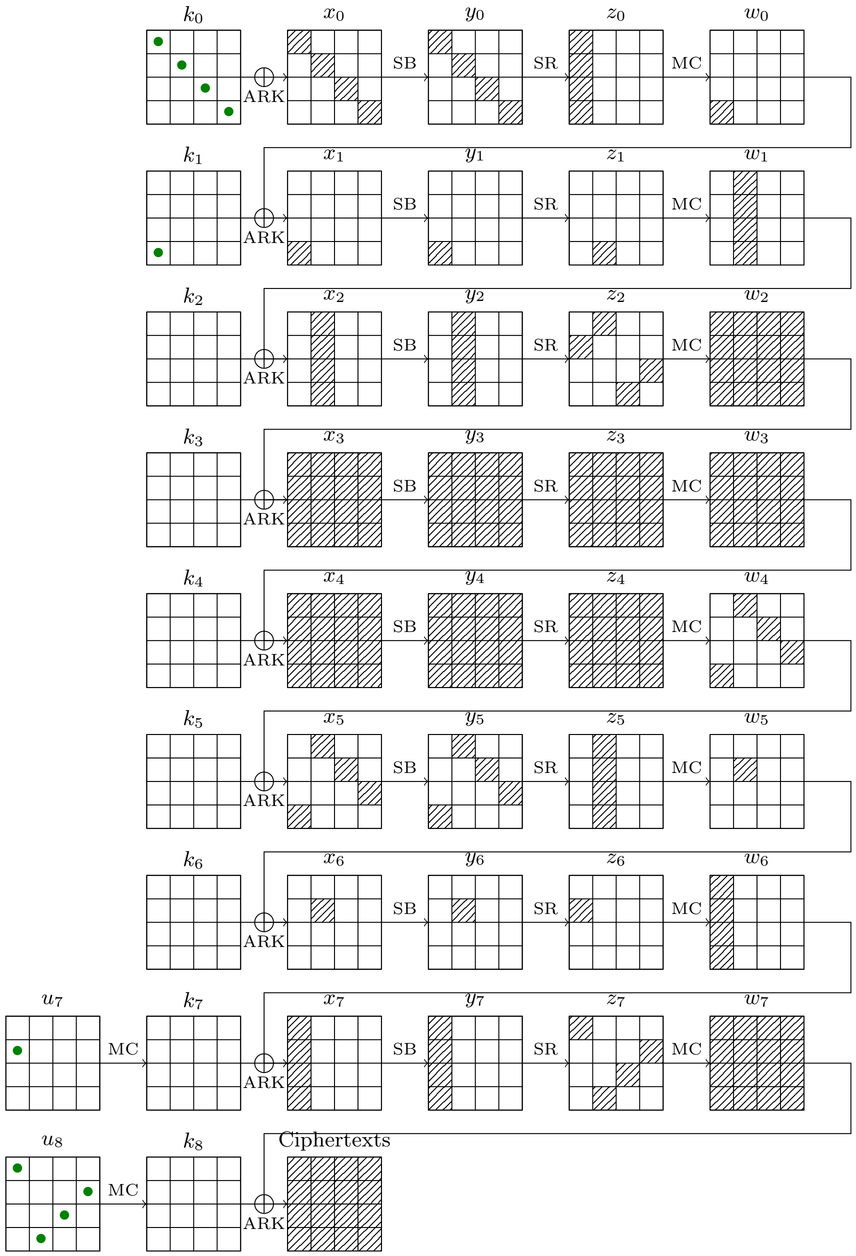

Figure 6: Path of the 8-round DS-MITM attack of [BNS19].

Algorithm 7 DS-MITM attack on 8-round AES-256.

80 bits $\mathcal{A}_1$ $\mathcal{D}_1$	**Choose** $k_0[0,5,10,15]$, $k_1[3]$, $u_7[1]$, $u_8[0,7,10,13]$ (Empty) $\{$ Find a pair which satisfies the differential path (2^{53} S-Boxes) Compute the δ-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 $\mathcal{A}_2$ Success: 2^{-8} $\mathcal{D}_2$	**Choose** $\Delta y_2[4-7], \Delta x_5[3,4,9,14]$ $\{$ Match $\Delta y_2[4{-}7], \Delta x_5[3,4,9,14]$ with $\Delta x_2[4{-}7]$, $\Delta y_5[3,4,9,14]$ **Stop** if no match $\{$ 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 $\mathcal{A}_3$ Succ.: $2^{-8}(1 \pm 2^{-8})$	**Choose** $\Delta x_4[0-3]$ $\{$ Match $\Delta x_4[0-3]$ with Δx_3 and Δy_4 **Stop** if no match
32 bits $\mathcal{A}_4$ Succ.: $2^{-8}(1 \pm 2^{-8})$	**Choose** $\Delta x_4[4-7]$ $\{$ Match $\Delta x_4[4-7]$ with Δx_3 and Δy_4 **Stop** if no match
32 bits $\mathcal{A}_5$ Succ.: $2^{-8}(1 \pm 2^{-8})$	**Choose** $\Delta x_4[8-11]$ $\{$ Match $\Delta x_4[8-11]$ with Δx_3 and Δy_4 **Stop** if no match
32 bits $\mathcal{A}_6$ Succ.: $2^{-8}(1 \pm 2^{-8})$ $\mathcal{D}_6$	**Choose** $\Delta x_4[12-15]$ $\{$ Match $\Delta x_4[12-15]$ with Δx_3 and Δy_4 **Stop** if no match $\{$ 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 $\mathcal{A}_7$ and $\mathcal{D}_7$ Success: 2^{-9}	**Choose** one of 2^9 possibilities for $x_4[8,9,12,13,14]$, $x_5[4,9,14]$ $\{$ Compute the expected δ-sequence ($2^5 \times 40$ S-Boxes) **Check** if it equals the expected sequence

project under Contract ANR-18-CE39-0007, and through the France 2030 program under grant agreement No. ANR-22-PETQ-0008 PQ-TLS.

References

[AA05] Scott Aaronson and Andris Ambainis. Quantum search of spatial regions. *Theory Comput.*, 1(1):47–79, 2005. URL: https://doi.org/10.4086/toc.2005.v001a004, doi:10.4086/TOC.2005.V001A004.

[AAC$^+$22] Gorjan Alagic, Daniel Apon, David Cooper, Quynh Dang, Thinh Dang, John Kelsey, Jacob Lichtinger, Carl Miller, Dustin Moody, Rene Peralta, et al. Status report on the third round of the nist post-quantum cryptography standardization process. *US Department of Commerce, NIST*, 2022.

[AKV23] Andris Ambainis, Martins Kokainis, and Jevgenijs Vihrovs. Improved algorithm and lower bound for variable time quantum search, 2023. doi:10.4230/LIPICS.TQC.2023.7.

[Amb10] Andris Ambainis. Quantum search with variable times. *Theory Comput. Syst.*, 47(3):786–807, 2010. URL: https://doi.org/10.1007/s00224-009-9219-1, doi:10.1007/S00224-009-9219-1.

[Amb12] Andris Ambainis. Variable time amplitude amplification and quantum algorithms for linear algebra problems. In *STACS*, volume 14 of *LIPIcs*, pages 636–647. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2012. doi:10.4230/LIPICS.STACS.2012.636.

[ANS18] Yoshinori Aono, Phong Q. Nguyen, and Yixin Shen. Quantum lattice enumeration and tweaking discrete pruning. In *ASIACRYPT (1)*, volume 11272 of *LNCS*, pages 405–434. Springer, 2018. doi:10.1007/978-3-030-03326-2_14.

[BBS05] Eli Biham, Alex Biryukov, and Adi Shamir. Cryptanalysis of skipjack reduced to 31 rounds using impossible differentials. *J. Cryptol.*, 18(4):291–311, 2005. URL: https://doi.org/10.1007/s00145-005-0129-3, doi:10.1007/S00145-005-0129-3.

[Ber10] Daniel J. Bernstein. Grover vs. mceliece. In *PQCrypto*, volume 6061 of *Lecture Notes in Computer Science*, pages 73–80. Springer, 2010. doi:10.1007/978-3-642-12929-2_6.

[BHMT02] Gilles Brassard, Peter Hoyer, Michele Mosca, and Alain Tapp. Quantum amplitude amplification and estimation. *Contemporary Mathematics*, 305:53–74, 2002.

[BHN$^+$19] Xavier Bonnetain, Akinori Hosoyamada, María Naya-Plasencia, Yu Sasaki, and André Schrottenloher. Quantum attacks without superposition queries: The offline simon's algorithm. In *ASIACRYPT (1)*, volume 11921 of *LNCS*, pages 552–583. Springer, 2019. doi:10.1007/978-3-030-34578-5_20.

[BNS19] Xavier Bonnetain, María Naya-Plasencia, and André Schrottenloher. Quantum security analysis of AES. *IACR Trans. Symmetric Cryptol.*, 2019(2):55–93, 2019. URL: https://doi.org/10.13154/tosc.v2019.i2.55-93, doi:10.13154/TOSC.V2019.I2.55-93.

[CGJ19] Shantanav Chakraborty, András Gilyén, and Stacey Jeffery. The power of block-encoded matrix powers: Improved regression techniques via faster hamiltonian simulation. In *ICALP*, volume 132 of *LIPIcs*, pages 33:1–33:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICS.ICALP.2019.33.

[DNPS23] Nicolas David, María Naya-Plasencia, and André Schrottenloher. Quantum impossible differential attacks: Applications to aes and skinny, 2023. doi:10.1007/s10623-023-01280-y.

[DP20] James H. Davenport and Benjamin Pring. Improvements to quantum search techniques for block-ciphers, with applications to AES. In *SAC*, volume 12804 of *LNCS*, pages 360–384. Springer, 2020. doi:10.1007/978-3-030-81652-0_14.

[DR02] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Information Security and Cryptography. Springer, 2002.

[FHK+17] Jean-Charles Faugère, Kelsey Horan, Delaram Kahrobaei, Marc Kaplan, Elham Kashefi, and Ludovic Perret. Fast quantum algorithm for solving multivariate quadratic equations. *IACR Cryptol. ePrint Arch.*, page 1236, 2017.

[FKL+00] Niels Ferguson, John Kelsey, Stefan Lucks, Bruce Schneier, Michael Stay, David A. Wagner, and Doug Whiting. Improved cryptanalysis of Rijndael. In *FSE*, volume 1978 of *LNCS*, pages 213–230. Springer, 2000. doi:10.1007/3-540-44706-7_15.

[Gid15] Craig Gidney. Constructing large controlled nots. https://algassert.com/circuits/2015/06/05/Constructing-Large-Controlled-Nots.html, 2015.

[Gro96] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *STOC*, pages 212–219. ACM, 1996. doi:10.1145/237814.237866.

[HS20] Akinori Hosoyamada and Yu Sasaki. Finding hash collisions with quantum computers by using differential trails with smaller probability than birthday bound. In *EUROCRYPT (2)*, volume 12106 of *Lecture Notes in Computer Science*, pages 249–279. Springer, 2020. doi:10.1007/978-3-030-45724-2_9.

[JNRV20] Samuel Jaques, Michael Naehrig, Martin Roetteler, and Fernando Virdia. Implementing grover oracles for quantum key search on AES and lowmc. In *EUROCRYPT (2)*, volume 12106 of *Lecture Notes in Computer Science*, pages 280–310. Springer, 2020. doi:10.1007/978-3-030-45724-2_10.

[KLL15] Shelby Kimmel, Cedric Yen-Yu Lin, and Han-Hsuan Lin. Oracles with costs. In *TQC*, volume 44 of *LIPIcs*, pages 1–26. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPICS.TQC.2015.1.

[KLLN16a] Marc Kaplan, Gaëtan Leurent, Anthony Leverrier, and María Naya-Plasencia. Breaking symmetric cryptosystems using quantum period finding. In *CRYPTO (2)*, volume 9815 of *Lecture Notes in Computer Science*, pages 207–237. Springer, 2016. doi:10.1007/978-3-662-53008-5_8.

[KLLN16b] Marc Kaplan, Gaëtan Leurent, Anthony Leverrier, and María Naya-Plasencia. Quantum differential and linear cryptanalysis. *IACR Trans. Symmetric Cryptol.*, 2016(1):71–94, 2016. URL: https://doi.org/10.13154/tosc.v2016.i1.71-94, doi:10.13154/TOSC.V2016.I1.71-94.

[KM10] Hidenori Kuwakado and Masakatu Morii. Quantum distinguisher between the 3-round feistel cipher and the random permutation. In *ISIT*, pages 2682–2685. IEEE, 2010.

[KMPM19] Elena Kirshanova, Erik Mårtensson, Eamonn W. Postlethwaite, and Subhayan Roy Moulik. Quantum algorithms for the approximate k-list problem and their application to lattice sieving. In *ASIACRYPT*, volume 11921 of *LNCS*, pages 521–551. Springer, 2019. doi:10.1007/978-3-030-34578-5_19.

[Knu98] Lars Knudsen. Deal-a 128-bit block cipher. *complexity*, 258(2):216, 1998.

[KT17] Ghazal Kachigar and Jean-Pierre Tillich. Quantum information set decoding algorithms. In *PQCrypto*, volume 10346 of *Lecture Notes in Computer Science*, pages 69–89. Springer, 2017. doi:10.1007/978-3-319-59879-6_5.

[Kup13] Greg Kuperberg. Another subexponential-time quantum algorithm for the dihedral hidden subgroup problem. In *TQC*, volume 22 of *LIPIcs*, pages 20–34. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2013. doi:10.4230/LIPICS.TQC.2013.20.

[Laa15] Thijs Laarhoven. *Search problems in cryptography*. PhD thesis, PhD thesis, Eindhoven University of Technology, 2015.

[LMvdP15] Thijs Laarhoven, Michele Mosca, and Joop van de Pol. Finding shortest lattice vectors faster using quantum search. *Des. Codes Cryptogr.*, 77(2-3):375–400, 2015. doi:10.1007/s10623-015-0067-5.

[MDRM10] Hamid Mala, Mohammad Dakhilalian, Vincent Rijmen, and Mahmoud Modarres-Hashemi. Improved impossible differential cryptanalysis of 7-round AES-128. In *INDOCRYPT*, volume 6498 of *Lecture Notes in Computer Science*, pages 282–291. Springer, 2010. doi:10.1007/978-3-642-17401-8_20.

[Mon18] Ashley Montanaro. Quantum-walk speedup of backtracking algorithms. *Theory Comput.*, 14(1):1–24, 2018. URL: https://doi.org/10.4086/toc.2018.v014a015, doi:10.4086/TOC.2018.V014A015.

[NC16] Michael A. Nielsen and Isaac L. Chuang. Quantum computation and quantum information (10th anniversary edition), 2016.

[Pri18] Benjamin Pring. Exploiting preprocessing for quantum search to break parameters for *MQ* cryptosystems. In *WAIFI*, volume 11321 of *LNCS*, pages 291–307. Springer, 2018. doi:10.1007/978-3-030-05153-2_17.

[Sho94] Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *FOCS*, pages 124–134. IEEE Computer Society, 1994. doi:10.1109/SFCS.1994.365700.

[SW16] Peter Schwabe and Bas Westerbaan. Solving binary *MQ* with grover's algorithm. In *SPACE*, volume 10076 of *LNCS*, pages 303–322. Springer, 2016. doi:10.1007/978-3-319-49445-6_17.

Appendix

A Quantum Search with Early Aborts

Search with early aborts (Figure 7) is a different case of tree search in which a *single choice* is made at the beginning of the search, but *multiple layers* of filtering are applied. These filters have typically increasing complexities. This situation is represented in Figure 7.

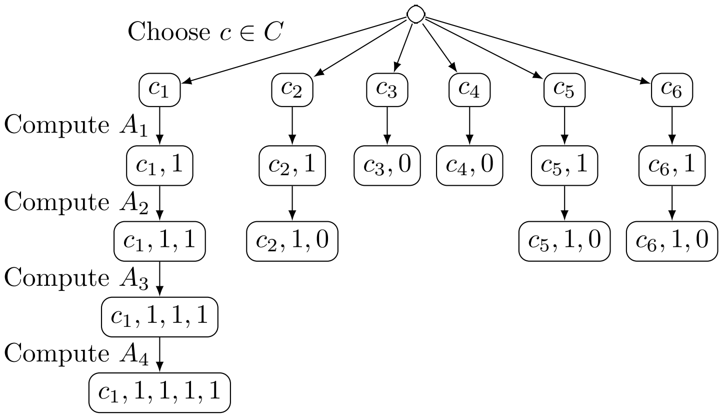

Figure 7: Search with early aborts, with a single choice $c \in C$, and 4 successive filters A_1, A_2, A_3, A_4.

We show here a generic framework for this problem. It is actually very different from Section 4, and the applications in cryptanalysis are also different. However, the analysis will look similar, as it also involves ℓ nested QAAs.

The structure of this section is similar to Section 4: after a definition of the problem, we give a classical algorithm based on nested search. Then, we give a quantum algorithm, compute its success probability and time complexity, and explain how to optimize this complexity numerically.

Comparison with [Amb12, AKV23]. As mentioned in the introduction, the setting considered in this section is a generalization of variable-time quantum search, which has been studied in other works [Amb12, AKV23]. In Appendix C we show that our framework yields an algorithm for variable-time search simpler than the one of Ambainis [Amb12]. Concurrent work of Ambainis, Kokainis and Vihrovs [AKV23] obtained a similar algorithm with a slightly better complexity. However, their analysis is specific to variable-time search and does not support the full early-abort framework.

A.1 Description 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 before.

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. Contrary to Subsection 3.2, we only need to define filtering probabilities, and their definition is different:

$$\beta_1^2 = \Pr_{c \xleftarrow{\$} C} (A_1(0_\ell, c, 0_w)|_{\text{flag } 1} = 1) \tag{23}$$

$$\forall 2 \le i \le \ell, \beta_i^2 = \frac{\Pr_{c \xleftarrow{\$} C} (A_i \circ \cdots \circ A_1(0_\ell, c, 0_w)|_{\text{flag } i} = 1)}{\Pr_{c \xleftarrow{\$} C} (A_{i-1} \circ \cdots \circ A_1(0_\ell, c, 0_w)|_{\text{flag } i-1} = 1)} \ , \tag{24}$$

so that β_i^2 is the probability that $c \in C$ passes the i-th step, conditioned on having passed the $(i-1)$-th step.

Difference between Backtracking and Early-Abort. The search with early aborts, which makes a single choice and splits the testing function in several layer, is fundamentally different from our main "backtracking" framework (Subsection 3.2), which splits the search space into several choice sets. 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, we have not found any interest for such an algorithm.

A.2 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.

Algorithm 8 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 9 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
```

In Algorithm 8, we start by sampling new values for c until we pass the first filter. In Algorithm 9, 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.

Algorithm 10 $\mathcal{B}_1$: 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 the register $C$
    Apply a Hadamard transform on the register $C$
    Compute $\mathcal{A}_1$
EndRepeat
```

Algorithm 11 $\mathcal{B}_i$: 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: $F_1, \dots, F_i, C, W$
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 the register $C$
    Compute $\mathcal{A}_i \circ \mathcal{B}_{i-1}$
EndRepeat
```

A.3 Description of the Quantum Algorithm

Our quantum algorithm is analogous to Algorithm 8 and Algorithm 9, 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 Subsection A.1.

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 $(k_1, \dots, k_\ell)$ chosen for each QAA.

In the definition of the algorithms (Algorithm 10 and Algorithm 11) we denote by O_i the test oracle that simply flips the phase if the i-th flag is 1. Each O_i 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:

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

$$= (\mathcal{A}_2\mathcal{A}'_1)O_0(\mathcal{A}'_1)^\dagger O_1(\mathcal{A}'_1)O_0(\mathcal{A}'_1)^\dagger O_1(\mathcal{A}'_1)O_0(\mathcal{A}_2\mathcal{A}'_1)^\dagger O_2(\mathcal{A}_2\mathcal{A}'_1)O_0(\mathcal{A}'_1)^\dagger O_1(\mathcal{A}'_1) \ .$$

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. One can notice that each time O_0 is applied, the work register W and flags contain 0, which is why it is only necessary to apply it on the choice register C.

A.4 Analysis

We do the analysis of $\mathcal{B}_\ell$ in three steps. First, we express its success probability as a function of $(k_1, \dots, k_\ell)$ and $(\beta_1, \dots, \beta_\ell)$. Second, we express its time complexity as a function of $(k_1, \dots, k_\ell)$ 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 $\mathcal{B}_i \left|0_\ell 0_n 0_w\right\rangle$, 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 6. *Let $\nu_0 = 1$ by convention. For all $i \geq 1$ we have:*

$$\nu_i = \sin\left[(2k_i + 1)\arcsin \beta_i \nu_{i-1}\right] \ . \tag{25}$$

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.:

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

where I_r is the identity operator applied to r qubits.

Let $\left|\psi_i\right\rangle$ 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:

$$\mathcal{B}_i \left|0_{\ell+n+w}\right\rangle = \nu_i \left|1_i 0_{\ell-i}\right\rangle \left|\psi_i\right\rangle + \left|\chi_i\right\rangle \ , \tag{26}$$

where $\left|\chi_i\right\rangle = P_i^0 \mathcal{B}_i \left|0_{\ell+n+w}\right\rangle$ 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 $1, \dots, i$, with different flags of the form $\left|1_j 0_{\ell-j}\right\rangle$ depending on the level which failed. In other words, $\mathcal{B}_i$ outputs $\left|\psi_i\right\rangle$ with probability ν_i^2.

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

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

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

$$\mathcal{B}_1 \left|0_{\ell+n+w}\right\rangle = \sin\left[(2k_1+1)\arcsin \beta_1\right] \left|1 0_{\ell-1}\right\rangle \left|\psi_1\right\rangle + \left(P_1^0 \mathcal{B}_1 \left|0_{\ell+c+w}\right\rangle\right) \ ,$$

where we get the definition of ν_1. Next, we assume that Equation 26 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 $\mathcal{A}_{i+1}\mathcal{B}_i$:

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

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. □

A.4.1 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 7. *The gate complexity of $\mathcal{B}_\ell(k_1, \dots, k_\ell)$ is given by:*

$$G(\mathcal{B}_\ell) = \left(\sum_{i=1}^{\ell}\left(\prod_{j=i}^{\ell}(2k_j+1)\right)G(\mathcal{A}_i)\right) + n\prod_{j=1}^{\ell}(2k_j+1)G(H) + k_1\prod_{j=2}^{\ell}(2k_j+1)G_0(n) \quad . \tag{27}$$

It uses $\ell+n+w+2$ 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 n qubits. By assumption, ancilla qubits used by the $\mathcal{A}_i$ are counted as workspace qubits.

The total number of H gates is n times the calls to $\mathcal{A}_1$, i.e., $n\prod_{j=1}^{\ell}(2k_j+1)$. Finally, the total number of calls to O_0 (resp. the O_i) is $k_1\prod_{j=2}^{\ell}(2k_j+1)$. □

A.4.2 Choice of the k_i.

Using approximations of the sin and arcsin functions, we transform Equation 25 into exploitable bounds on the ν_i^2. This leads to conditions on the number of iterates k_i to ensure the success of the algorithm (Theorem 3).

Lemma 8. *Let $\nu_0 = 1$ by convention, and $s_i = (2k_i+1)^2\beta_i^2$. Then for all $i \geq 1$ we have:*

$$s_i\nu_{i-1}^2\left(1 - s_i\nu_{i-1}^2\right) \leq \nu_i^2 \leq s_i\nu_{i-1}^2 \quad . \tag{28}$$

The proof is similar to Lemma 5 by replacing $s_is'_i$ by s_i and changing the indices.

Theorem 3. *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: $\nu_\ell^2 \geq \frac{1}{2}\prod_{j=1}^{\ell} s_j$.*

The proof is similar to Theorem 1 by replacing $s_is'_i$ by s_i and changing the indices.

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.

A.5 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 9. *Assume that:* $\forall i, k_i \leq \frac{\pi}{4} \frac{1}{\arcsin u_i} - \frac{1}{2}$. *Let* ν_i^l *(lower bound) and* ν_i^u *(upper bound) be obtained by replacing the* β_i *by* l_i *and* u_i*, respectively, in the formulas of Lemma 6. Then we have:* $\nu_\ell^u \geq \nu_\ell \geq \nu_\ell^l$.

Proof. The proof is similar to Lemma 4. □

Therefore, we can bypass Theorem 3 and directly express the lower bound on the success probability ν_ℓ^2 of $\mathcal{B}_\ell$ as a function of the k_i:

$$\begin{cases} \nu_0^l = 1 \\ \forall i, \nu_i^l = \sin\left[(2k_i + 1) \arcsin\left[l_i \nu_{i-1}^l\right]\right] \end{cases} \tag{29}$$

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 3 for the asymptotic case, then the optimization method for practical cases.

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 Appendix A.

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 3:

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

By Equation 27 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 $\mathcal{O}\left((\log^*_{\sqrt{2}} n)^{3/2} 2^{n/2}\right)$.

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:

$$\begin{cases} \Pr\left(||X_{m_1}| - 2^{n-m_1}| \geq \varepsilon_1 2^{n-m_1}\right) \leq 2\exp\left(\frac{-\varepsilon_1^2 2^{n-m_1}}{3}\right) \\ \Pr\left(||X_{m_1} \cap X_{m_1+m_2}| - 2^{n-m_1-m_2}| \geq \varepsilon_2 2^{n-m_1-m_2}\right) \leq 2\exp\left(\frac{-\varepsilon_2^2 2^{n-m_1-m_2}}{3}\right) \end{cases} \tag{30}$$

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:

$$\begin{cases} \beta_1^2 \in \left[2^{-m_1}(1-\varepsilon_1); u_1^2 := 2^{-m_1}(1+\varepsilon_1)\right] \\ \beta_2^2 \in \left[2^{-m_2}\frac{1-\varepsilon_2}{1+\varepsilon_1}; u_2^2 := 2^{-m_2}\frac{1+\varepsilon_2}{1-\varepsilon_1}\right] \\ \beta_3^2 \in \left[2^{-n+m_1+m_2}\frac{1}{1+\varepsilon_2}; u_3^2 := 2^{-n+m_1+m_2}\frac{1}{1-\varepsilon_2}\right] \end{cases} \tag{31}$$

Thus, after choosing values for m_1 and m_2, we can follow the approach of Subsection A.5: 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:

$$k_1 \leq \left\lfloor \frac{\pi}{4}\frac{1}{\arcsin u_1} - \frac{1}{2} \right\rfloor, k_2 \leq \left\lfloor \frac{\pi}{4}\frac{1}{\arcsin u_2} - \frac{1}{2} \right\rfloor, k_3 \leq \left\lfloor \frac{\pi}{4}\frac{1}{\arcsin u_3} - \frac{1}{2} \right\rfloor \ . \tag{32}$$

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.

C Variable-time Amplitude Amplification without Amplitude Estimation

In this section, we show how the framework of Appendix A allows to solve the problem of amplitude amplification with a variable-time algorithm [Amb10, Amb12, CGJ19]. 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 [AKV23]. 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:

$$T_2 := \sqrt{\frac{1}{|C|}\sum_{c \in C} t(c)^2} \ . \tag{33}$$

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 [Amb10] that even if the running times

$t(c)$ are not known, there exists a quantum algorithm that finds a good c in time $\mathcal{O}\Big(t_{\max}\sqrt{t_{\max}} + \frac{T_2}{\sqrt{p}}(\log t_{\max})^{3/2}\Big)$. 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 3.

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:

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

Besides, it can be noticed that by definition of p: $\prod_{i=1}^{\ell} \beta_i^2 = p$. So, in order to use Theorem 3, 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:

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

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:

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

Complexity. The gate complexity is given by Equation 27. If w is the number of qubits in the workspace of $\mathcal{A}$, we can bound it simply as follows:

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

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 $\mathcal{O}(\ell 3^{\ell})$.

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:

$$\mathcal{O}\left(\frac{\ell 3^{\ell}}{\sqrt{P}}\right) = \mathcal{O}\left(\frac{1}{\sqrt{p}}\ell^{3/2}T_2\right) = \mathcal{O}\left((\log t_{\max})^{3/2}\frac{T_2}{\sqrt{p}}\right) \quad . \tag{34}$$

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.