Quantum impossible differential attacks: Applications to AES and SKINNY

Nicolas David[1], María Naya-Plasencia[1], and André Schrottenloher[2]

[1] Inria, France
firstname.lastname@inria.fr
[2] Cryptology Group, CWI, The Netherlands
firstname.lastname@m4x.org

Abstract. In this paper we propose the first efficient quantum version of key-recovery attacks on block ciphers based on impossible differentials, which was left as an open problem in previous work. These attacks work in two phases. First, a large number of differential pairs are collected, by solving a limited birthday problem with the attacked block cipher considered as a black box. Second, these pairs are filtered with respect to partial key candidates. We show how to translate the pair filtering step into a quantum procedure, and provide a complete analysis of its complexity. If the path of the attack can be properly reoptimized, this procedure can reach a significant speedup with respect to classical attacks. We provide two applications on SKINNY-128-256 and AES-192/256. These results do not threaten the security of these ciphers but allow us to better understand their (post-quantum) security margin.

Keywords: Quantum cryptanalysis · Impossible differential attacks · Block ciphers

1 Introduction

During the last few years, the interest of the community in understanding the resistance of symmetric primitives to quantum adversaries has considerably increased. Some authors have proposed quantized versions of classical attacks like [25,9] as well as new quantum dedicated attacks [7,24,8]. In [10] the authors performed a quantum security analysis of AES. Though none of the proposed quantum attacks reach more rounds than the classical ones, this is because they are compared to an exhaustive search of the key using Grover's algorithm [22], which provides a new generic bound in the quantum setting. The post-quantum security can only be determined with respect to this new bound, and is then defined by such quantum attacks. Therefore, we need to study these attacks with the same care as previous classical attacks.

In [10] the authors showed how to turn classical Square attacks [15,20] and Demirci-Selçuk Meet-in-the-middle attacks [18,19] into quantum attacks. The latter technique gave the best known attack on AES-256 compared to Grover's

algorithm. Although classically, *impossible differential attacks* also provide some trade-offs and comparable complexity, the authors of [10] mention that they did not find a proper way to quantize them, nor a "significant speed-up".

Impossible differential attacks, introduced simultaneously by Knudsen [27] and Biham, Biryukov and Shamir [5], exploit a differential transition that cannot occur to build a distinguisher or to extract information on the secret key of a cipher. Since [10], to the best of our knowledge, there has been no further study of quantum impossible differential attacks, except a proposal [35,34] to use quantum algorithms to efficiently find impossible paths (but no actual speedup of the attack).

This paper. The results presented in this paper improve our knowledge in several directions.

1. We propose the first efficient quantum impossible differential attacks with a competitive speed up regarding classical attacks. An impossible differential key-recovery attack runs in two phases: first, given black-box encryption and decryption access, we build a set of pairs with some truncated input-output difference pattern. Second, partial key candidates are sieved, by removing those which, on some of the given pairs, would make the impossible differential appear. Our main contribution is an efficient quantum algorithm for this *pair filtering step*, with a precise complexity analysis based on Amplitude Amplification [13].
2. We give some results on the applications of these attacks to the popular block ciphers AES and SKINNY, summarized in Table 1 and compared to the best existing post-quantum attacks (by this we imply attacks that are better than Grover's exhaustive key search). We also fill in the gap from [10] by proposing the first quantum impossible differential attacks on AES-192/256.

Organization. We start in Section 2 by introducing (classical) impossible differential attacks. Section 3 introduces some needed quantum preliminaries, including the algorithms used for the pair generation step. Section 4 describes the process of pair filtering in the quantum setting. In Section 5 we apply our technique to SKINNY and in Section 6, to AES. We conclude the paper in Section 7.

2 Classical Impossible Differential Attacks

In this section, we provide a generic depiction of classical impossible attacks, that will be helpful for translating them into quantum algorithms. We give a generic formula for their complexity which is from [12].

2.1 Principle

Impossible differential attacks were independently introduced by Knudsen [27] and Biham, Biryukov and Shamir [5]. The goal of this cryptanalysis technique is to recover some bits of the secret key of a black-box encryption oracle. This is

Table 1. Summary of best quantum attacks on SKINNY-128-256 and AES-192/256 (with lower complexities than Grover's search). Symbol * means we have extrapolated the complexity on 21 rounds from the original attack on 24 rounds for comparison, though it is still too expensive quantumly (the best classical one is the 24-round attack). For SKINNY-128-256 our results clearly provide the best quantum attack and therefore the security margin. For AES-256, we obtain a better memory than [19], and a time complexity comparable to [10]. Our best attack is dominated by the cost of generating the pairs. ** indicates that the memory considered is quantum memory with quantum random-access (QRAQM). Otherwise, this is classical memory.

Algorithm	# rounds	Ref.	Time	Memory	Data	Setting
SKINNY-128-256	21	Section 5	$2^{119.17}$	$(2^{103.17})^{**}$	$2^{119.17}$	Q_2
SKINNY-128-256	21	[33]*	$2^{143.17}$	$2^{103.17}$	2^{128}	Classical
SKINNY-128-256	20	[33]*	$2^{126.46}$	$2^{54.6}$	$2^{126.46}$	Classical
AES-192	7	[10]	$2^{105.6}$	negl.	negl.	Q1
AES-256	7	[10]-Grover	$2^{137.3}$	negl.	negl.	Q1
AES-256	7	[10]-Square	2^{121}	2^{38}	2^{37}	Q1
AES-192	7	[10]-Square	$2^{103.4}$	$2^{38} + (2^{27})^{**}$	2^{37}	Q1
AES-256	7	[10]-Square	2^{107}	$2^{38} + (2^{27})^{**}$	2^{37}	Q1
AES-256	7	[19]	$2^{99} + 2^{98}$	2^{96}	2^{99}	Classical
AES-192/256	7	Section 6	$2^{101.5}$	$(2^{78.5})^{**}$	$2^{99.8}$	Q2
AES-192/256	7	Section 6	$2^{99.8} + 2^{95.2}$	$(2^{78.5})^{**}$	$2^{99.8}$	Q2
AES-256	8	[10]-Grover	2^{138}	negl.	negl.	Q1
AES-256	8	[10]-DS-MITM	2^{136}	2^{88}	2^{88}	Q1

done by discarding all the wrong key guesses, with the help of a pair of plaintexts that leads to an impossible pattern under its partial encryption with the wrong key guesses.

Let E be an n-bit block cipher with key space K: $E : K \times \{0,1\} \to \{0,1\}^n$, which has r rounds in total. We write $E = E_{\text{out}} \circ E_{\text{imp}} \circ E_{\text{in}}$, as on Figure 1, where $E_{\text{out}}, E_{\text{imp}}$ and E_{in} have $r_{\text{out}}, r_{\text{imp}}$ and r_{in} rounds respectively ($r = r_{\text{in}} + r_{\text{imp}} + r_{\text{out}}$).

An impossible differential attack is based on an impossible differential of maximal length, that is, a pair of differentials Δ_X, Δ_Y such that the probability that Δ_X propagates to Δ_Y after r_{imp} rounds is 0. We will then append r_{in} and r_{out} rounds of the cipher respectively before and after the impossible differential. We name *impossible pattern* the tuple of quantities $(\Delta_X, \Delta_Y, r_{\text{imp}}, r_{\text{in}}, r_{\text{out}})$.

Next, we define two sets of differences D_{in} and D_{out} such that Δ_X maps backwards to D_{in} through E_{in}, and Δ_Y maps forwards to D_{out} through E_{out}. Assume that, after querying the black-box $E_k = E(k)(\cdot)$, we found a pair of plaintexts $p = (x, y)$ such that $x \oplus y \in D_{\text{in}}$ and $E_k(x) \oplus E_k(y) \in D_{\text{out}}$. Then, thanks to the

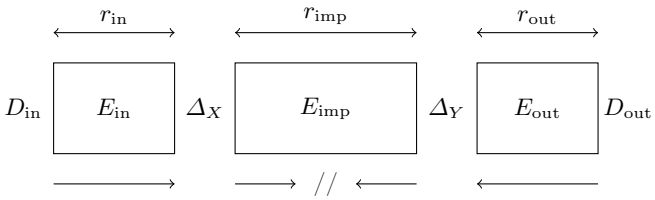

Fig. 1. Impossible differential attack, with the notations used in this paper. The differential $\Delta_X \leftrightarrow \Delta_Y$ through the middle rounds E_{imp} is impossible.

impossible pattern, we can discard any candidate key u that satisfies:

$$\left(E_{\text{in}}(u)(x) \oplus E_{\text{in}}(u)(y) = \Delta_X\right) \wedge \left(E_{\text{out}}^{-1}(u)(E_k(x)) \oplus E_{\text{out}}^{-1}(u)(E_k(y)) = \Delta_Y\right) . \tag{1}$$

The goal of the attack is to discard as many keys as possible using many plaintext-ciphertext pairs. Let K_{in} denote the space of subkeys involved in E_{in}, typically a subset of the bits of the key, and K_{out} be the space of subkeys involved in E_{out}. Finally, let $K_{\text{in}\cup\text{out}}$ be the space of subkeys involved both in E_{in} and E_{out}. In general $K_{\text{in}\cup\text{out}}$ is smaller than $K_{\text{in}} \times K_{\text{out}}$ due to key-schedule relations. Only $K_{\text{in}\cup\text{out}}$ is relevant for our analysis, since the rest of the key is not involved in the impossible differential attack.

In this paper, we adopt a representation inspired from [12]. The attack is a two-step procedure.

Step 1: Pair Generation. In this part, we focus on solving the following problem: given access to the black-box encryption oracle $E : \{0,1\}^n \to \{0,1\}^n$ (and possibly its inverse, the decryption oracle) the definition of $D_{\text{in}} \subset \{0,1\}^n$ and $D_{\text{out}} \subset \{0,1\}^n$, and a parameter N (large), we want N pairs $(x, y) \in \{0,1\}^{2n}$ such that $x \oplus y \in D_{\text{in}}$ and $E(x) \oplus E(y) \in D_{\text{out}}$. This is a *limited-birthday* problem. The most efficient algorithm for solving it is given in [11]. The result is a table of pairs $\mathcal{T}_0$, where $p = (x, y)$ denotes an individual pair.

Step 2: Pair Filtering. In this step, we start from the table $\mathcal{T}_0$ of size N computed above. The goal of this step is to find, among all the subkeys $K_{\text{in}\cup\text{out}}$, those which are invalidated by some pair of $\mathcal{T}_0$ (or alternatively, those which aren't and constitute valid key guesses). Rather than trying all the pairs for all the keys, we optimize this step using the *early abort* technique which was introduced in [30] and describe in detail in [11]. For a given key guess $k \in K_{\text{in}\cup\text{out}}$, this technique filters the current table of pairs only to keep those which will most likely invalidate k.

We formalize the pair filtering step using *test functions*. Let us assume that $K_{\text{in}\cup\text{out}}$ can be decomposed as: $K_{\text{in}\cup\text{out}} = K_1 \times K_2 \times \cdots \times K_\ell$, where $K_1, \ldots, K_\ell$ typically represent choices for some bits of the subkeyspace[3]. Together with

[3] Note that sometimes, not all key schedule relations can be used in the decomposition, and so, the set $K_1 \times K_2 \times \cdots \times K_\ell$ is larger than $K_{\text{in}\cup\text{out}}$ itself.

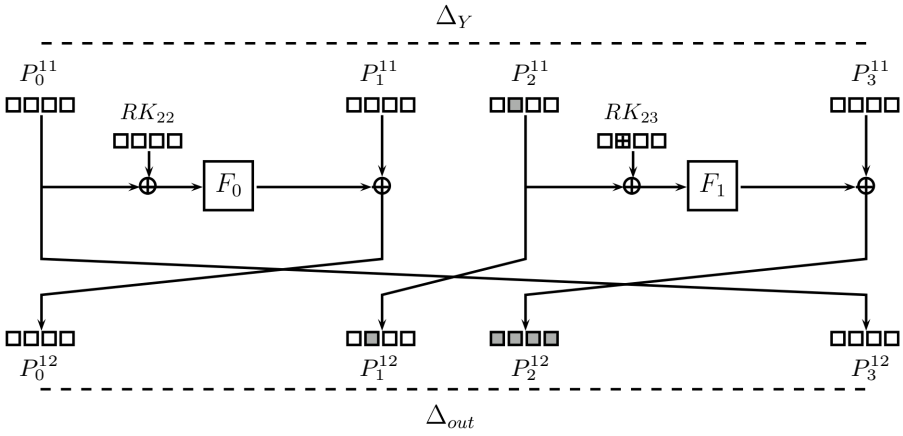

Fig. 2. A round of CLEFIA-128 (figure taken from [12]).

this decomposition, we will have ℓ test functions:

$$T_i \ : \begin{cases} \mathcal{T}_0 \times K_1 \times \ldots \times K_i \to & \{0,1\} \\ (p \ , \ k_1 \ , \ \ldots \ , \ k_i) \mapsto & T_i(p, k_1, \ldots, k_i) \end{cases} . \tag{2}$$

Each test function T_i takes some part of the subkey, some part of the pair, and checks whether they meet some condition. Typically, we start from the differences D_{in} and D_{out} and compute partially the first and last rounds; the successive T_i check that the partial encryption and decryption of the pair satisfies a truncated differential pattern that ultimately leads to the impossible differential (Δ_X, Δ_Y) at rounds r_{in} and r_{out}.

Example 1. Let us give an example of a test function. We study rounds 11 and 12 of the block cipher CLEFIA-128 (see Figure 2). CLEFIA is a 4-branch Feistel scheme and we let $(P_0^{12}, P_1^{12}, P_2^{12}, P_3^{12})$ denote the internal state at round 12. We want no difference at round 11 on $P_0^{11}, P_1^{11}, P_3^{11}$, therefore the following equation needs to hold:

$$F_1(P_1^{12} \oplus RK_{23}) \oplus P_2^{12} = 0 \ . \tag{3}$$

Then, the test function to propagate from round 12 to round 11 is:

$$T \ : (P_1^{12}, P_2^{12}, RK_{23}) \mapsto \left(F_1(P_1^{12} \oplus RK_{23}) = P_2^{12}\right) \ . \tag{4}$$

Next, we define the set of pairs satisfying all the T_i:

$$\mathcal{T}_\ell(k) = \{p \in \mathcal{T}_0 \mid \forall i, T_i(p, k_1, \ldots, k_i) = 1\} \ . \tag{5}$$

Thus, the test functions are defined so that: $\mathcal{T}_\ell(k) \neq \emptyset$ if and only if, there exist a given pair $p \in \mathcal{T}_0$ such that k makes the impossible differential appear for

p. The computation of $\mathcal{T}_\ell(k)$ thus yields a probabilistic procedure that discards a wrong subkey with some probability.

As long as there are less remaining key candidates than the whole of $K_{\text{in}\cup\text{out}}$, we can recover the rest of the key by exhaustive search, and it will be more efficient than an exhaustive search on the whole key space K. Naturally, the pair filtering step must also be more efficient than the exhaustive search of K.

2.2 Classical Complexity

In this section, we provide both the data complexity and the time complexity of the impossible attack previously described in [12].

Number of Pairs. In impossible differential attacks, the data complexity, and the time complexity of the first step, are both determined by the number N of pairs in the table $\mathcal{T}_0$.

Considering a differential pair with input difference in D_{in} and output difference in D_{out}, we denote by c_{in} the number of bit conditions for a pair to propagate to a difference of Δ_X at round r_{in}, and c_{out} the number of bit conditions to reach a difference of Δ_Y at round $r_{\text{in}} + r_{\text{imp}}$. Hence, a pair propagates to the middle part both from the plaintext and the ciphertext with probability $2^{-(c_{\text{in}}+c_{\text{out}})}$. A given trial key $k \in K_{\text{in}\cup\text{out}}$ is kept among the candidate keys with probability

$$P = \left(1 - 2^{-(c_{\text{in}}+c_{\text{out}})}\right)^N \quad . \tag{6}$$

By applying log to both sides, we obtain

$$N = \mathcal{O}\left(\log \frac{1}{P} \cdot 2^{c_{\text{in}}+c_{\text{out}}}\right) \quad . \tag{7}$$

Different trade-offs are therefore possible between P and N. A popular strategy, generally used by default is to choose N such that only the right key is left after the sieving procedure. This means $P < \frac{1}{|K_{\text{in}\cup\text{out}}|}$ therefore $N = \mathcal{O}(2^{c_{\text{in}}+c_{\text{out}}} \cdot |K_{\text{in}\cup\text{out}}|)$. Another way consists in taking $P = \frac{1}{2}$ which removes half of the candidate keys.

Time Complexity. We emphasize that the search of the impossible differential pattern is not involved here. We suppose, as it is common in the literature, that a good impossible differential pattern is already known. We detail here the complexity of both steps of the sieving phase.

Pair Generation. We let Δ_{in} and Δ_{out} be the dimension of the vector spaces D_{in} and D_{out}, so $|D_{\text{in}}| = 2^{\Delta_{\text{in}}}$ and $|D_{\text{out}}| = 2^{\Delta_{\text{out}}}$. The complexity of the Pair Generation problem was studied in [21] for the special case where $N = 1$. The complexity for finding *one* such pair with access to encryption and decryption oracles is given by:

$$C_1 = \max\left(\min_{\Delta\in\{\Delta_{\text{in}},\Delta_{\text{out}}\}} \sqrt{2^{n+1-\Delta}}, 2^{n+1-(\Delta_{\text{in}}+\Delta_{\text{out}})}\right) \quad . \tag{8}$$

This is optimal if E behaves like a random permutation, as shown in [23]. A naive way to build N pairs would be to use this technique N times and get a complexity of $N \cdot C_1$. However, as shown in [12], with access to encryption and decryption oracles, it is possible to reduce this complexity to:

$$C_N = \max\left(\min_{\Delta \in \{\Delta_{\text{in}}, \Delta_{\text{out}}\}} \sqrt{N 2^{n+1-\Delta}}, N 2^{n+1-(\Delta_{\text{in}}+\Delta_{\text{out}})}\right) . \tag{9}$$

Pair Filtering. We describe generically the *early-abort* procedure that finds the good guess of the key, assuming that there is only one. Note that if the subkey has sufficiently many independent parts, and if we are able to perform some precomputations, we can expect the time complexity of this phase to be about $|K_{\text{in}\cup\text{out}}|$. In some cases, especially if there are too many key-schedule relations which reduce the size of $K_{\text{in}\cup\text{out}}$, it will be higher.

We divide $K_{\text{in}\cup\text{out}} = K_1 \times K_2 \times \cdots \times K_\ell$. Recall that there is an ordering of the test functions: T_1 can be computed immediately from the pair p and a guess $k_1 \in K_1$; then T_2 can be computed from p, k_1 and a guess k_2, *etc.* Typically, k_1 will be a key guess for the first round, and k_2 a key guess for the second round. Given this ordering of the test functions, the early-abort strategy relies on the definition of *intermediate tables*: we do not compute directly $\mathcal{T}_\ell(k)$ for all key guesses k, and instead, we compute tables $\mathcal{T}_i$ which depend on some partial key guess, and are partially filtered with respect to a subset of the test functions.

Definition 1 (Intermediate Tables). *Given a partial key guess $(k_1, \ldots, k_i) \in K_1 \times \ldots \times K_i$, given the initial table $\mathcal{T}_0$ of pairs, the* intermediate table of $(k_1, \ldots, k_i)$ *contains all the pairs satisfying the test functions which can be computed from $(k_1, \ldots, k_i)$:*

$$\mathcal{T}_i(k_1, \ldots, k_i) := \{p \in \mathcal{T}_0, T_1(p, k_1) = 1, T_2(p, k_1, k_2) = 1, \ldots, T_i(p, k_1, \ldots, k_i) = 1\} .$$

The early-abort technique (Algorithm 1) relies on the inclusion of intermediate tables: $\forall i \geq 1, \mathcal{T}_i(k_1, \ldots, k_i) \subseteq \mathcal{T}_{i-1}(k_1, \ldots, k_{i-1})$. This allows to enumerate the complete guesses $(k_1, \ldots, k_\ell)$ such that $\mathcal{T}_\ell(k_1, \ldots, k_\ell) = \emptyset$ by looping over the guesses k_i and backtracking to previously computed tables, instead of recomputing them from the start.

For a given sequence of keys $k_1, \ldots, k_i$, we define:

$$\sigma_i = \Pr_{p \xleftarrow{\$} \mathcal{T}_0} \left(T_i(p, k_1, \ldots, k_i) = 1\right) .$$

In general, σ_i can depend on the choice of the key, but in practice, the values are similar, and we can reason with the expectancy of σ_i over all key choices ($\sigma_i \simeq \mathbb{E}_k(\sigma_i(k))$). Then the sizes of the tables $\mathcal{T}_0, \ldots, \mathcal{T}_{\ell-1}$ are $N, \sigma_1 N, \ldots, (\prod_{i=1}^{\ell-1} \sigma_i) N$. Assuming that each new table is built by enumerating the previous table, the

Algorithm 1 Finding the good key guess, by filtering the pairs.

Input: a table $\mathcal{T}_0$ for the pairs
Output: finds the only key k for which there is no invalidating pair

for all $k_1 \in K_1$ **do**
  Build $\mathcal{T}_1(k_1) = \{p \in \mathcal{T}_0, T_1(p, k_1) = 1\}$
  for all $k_2 \in K_2$ **do**
    Build $\mathcal{T}_1(k_1) = \{p \in \mathcal{T}_1, T_2(p, k_1, k_2) = 1\}$
  ...
    for all $k_\ell \in K_\ell$ **do**
      Sieve the table $\mathcal{T}_{\ell-1}(k_1, \ldots, k_{\ell-1})$ according to T_ℓ, obtain $\mathcal{T}_\ell$
      If $\mathcal{T}_\ell = \emptyset$, then return $(k_1, \ldots, k_\ell)$
    end for
  ...
  end for
end for

time complexity of Algorithm 1 can be written as:

$$|K_1| \left(\underbrace{N}_{\text{Build } \mathcal{T}_1} + |K_2| \left(\sigma_1 N + |K_3| \left(\sigma_1\sigma_2 N + \ldots + |K_\ell| \left(\prod_{i=1}^{\ell-1} \sigma_i \right) N \right) \right) \right)$$
$$= N \left(|K_1| + \sigma_1 |K_1||K_2| + \sigma_1\sigma_2 |K_1||K_2||K_3| + \ldots + \prod_{i=1}^{\ell-1} \sigma_i \prod_{i=1}^{\ell} |K_i| \right) \quad (10)$$

Details can be found in [12]. The memory complexity remains N.

There are more technical improvements that allow to reduce this complexity, some of which will be explored later in Section 4.4. But this formula has the advantage of being generic and easy to transpose to the quantum setting, as we will see in Section 4.3. Note that, in the case where there is more than one key left after the sieving, Algorithm 1 will enumerate them all.

3 Preliminaries of Quantum Computing

In this section we first provide some notions of quantum computing required in this paper: we introduce the quantum circuit model, notions of quantum complexity, of quantum-accessible memory, and some of our quantum computing tools. We also define the two quantum attacker scenarios commonly found in the literature, that will be considered in this paper, and give known results about the Pair Generation step of quantum impossible differential attacks.

3.1 Quantum Computing Notions

We refer to [32] for a broad introduction to quantum computing. In this paper, we use the abstract model of *quantum circuits*. A quantum circuit starts from a set

of qubits (basic two-level quantum systems) and applies a sequence of elementary *quantum gates*, analogous to classical logic gates. The state of a quantum system is represented as a normalized vector in a Hilbert space, represented by the "ket" $|\cdot\rangle$ notation, and quantum computations are unitary operators on this space. The width of the circuit (number of qubits) is the memory complexity of the algorithm. The depth of the circuit can be thought of as its wall-clock time, and the number of gates as the time complexity, since it represents the total number of operations applied.

In this paper, we will assume that a set of elementary gates is defined, but refrain from going into more details. As we study the cryptanalysis of block ciphers, when studying a cipher E, our unit of computation will be an evaluation of E or of E^{-1} (or for the AES, of an AES S-Box, which is the most costly component). The cipher can either be evaluated classically or as a quantum circuit. When querying it as a black-box with a secret key, there are two possibilities commonly considered in quantum symmetric cryptanalysis.

Quantum attacker scenarios. Following a standard terminology [25], we define two scenarios. In the *Q1* setting, the attacker has access to a quantum machine for accelerating his computations, but has only access to a classical cryptographic oracle: the black-box E can only be queried classically. In the *Q2* setting, the attacker can encrypt arbitrary quantum states using a *quantum embedding* O_E of E which performs the operation: $|x\rangle|0\rangle \mapsto |x\rangle|E(x)\rangle$.

Quantum memory. We will consider three types of memory: (1) large classical memories with classical random access; (2) quantum circuits with large amounts of qubits; (3) large amounts of qubits with *quantum random access*. The latter is named QRAQM in [28]. It can be modelled as the addition, to the quantum circuit model, of a "qRAM gate" (see *e.g.* [1]) which performs the equivalent of a classical memory access, but in superposition. Implementing a qRAM gate with only standard quantum gates would require a time proportional to the number of qubits in the circuit, the QRAQM model assumes that the qRAM gate is given and costs time 1. This powerful memory model is used to obtain optimal time complexities in many advanced quantum algorithms, e.g. collision finding for the limited-birthday problem [1,6].

3.2 Quantum Search

In this paper, we will use Grover's algorithm [22] and its generalization from [13], *amplitude amplification*, that we will regroup under the name *quantum search*. In fact, we mostly use the simple setting of Grover search, where we are given a function $f : \{0,1\}^n \to \{0,1\}$ and we want to find x_g such that $f(x_g) = 1$. If we have an implementation of the unitary O_f: $|x\rangle \mapsto (-1)^{f(x)}|f(x)\rangle$, and assuming that there is a single solution x_0, we can find it in $\mathcal{O}\left(2^{n/2}\right)$ calls to O_f instead of $\mathcal{O}(2^n)$ calls to f classically.

Let us make two important remarks on this oracle O_f. First, O_f is equivalent, up to a few basic gates, to an oracle writing $|x\rangle|0\rangle \mapsto |x\rangle|f(x)\rangle$. This allows us to focus on the implementation of this second version. Second, O_f can be

Algorithm 2 Quantum search (Grover's algorithm).

Input: access to O_f
Output: an n-qubit state such that when measured, the state collapses to x_g with high probability

Initialize n qubits in the state $|0\rangle$
Apply a Hadamard transform $H^{\otimes n}$
Repeat t **times**
    Apply O_f
    Apply $H^{\otimes n}$
    Apply the inversion around zero operator: $O_0 : |x\rangle \mapsto (-1)^{(x=0)} |x\rangle$
        ▷ This requires $\mathcal{O}(n)$ basic gates and is usually neglected
    Apply $H^{\otimes n}$
EndRepeat

implemented with an *auxiliary state* $\mathcal{T}$: $O_f |x\rangle |0\rangle |\mathcal{T}\rangle \mapsto |x\rangle |f(x)\rangle |\mathcal{T}\rangle$. This auxiliary state can be understood as a memory (e.g. QRAQM) which O_f reads from, without modifying it. As long as it does not get entangled with the register $|x\rangle$, it can be omitted from the analysis of quantum search.

In Grover's algorithm, and more generally amplitude amplification [13] we start from a state of the form:

$$\sin\theta |G\rangle + \cos\theta |B\rangle \quad ,$$

where $|G\rangle$ is the uniform superposition over the solutions ($G = f^{-1}(1)$), $|B\rangle$ is the uniform superposition of the non-solutions ($B = f^{-1}(0)$) and θ is such that $\sin^2\theta$ is the initial probability of success. In our case, $|G\rangle = |x_g\rangle$, $|B\rangle = \frac{1}{\sqrt{2^n-1}} \sum_{x \neq x_g} |x\rangle$ and $\theta = \arcsin 2^{n/2}$. After t iterates in Algorithm 2, the current state becomes:

$$\sin((2t+1)\theta) |G\rangle + \cos((2t+1)\theta) |B\rangle \quad .$$

By choosing t such that $(2t+1)\theta$ approaches $\pi/2$, the state becomes close to $|G\rangle$, and we will measure a solution with high probability (in our case, the single solution x_g). In particular $t = \left\lfloor \frac{\pi}{4\theta} \right\rfloor$ allows to succeed with probability at least $\max\left(\sin^2\theta, 1-\sin^2\theta\right)$.

However, in our case, we will need to increase this probability of success to 1 exactly. This will allow to use Algorithm 2 as an oracle testing the existence of solutions without making any errors. To do so, we would like to apply a non-integer number of iterations $t = \frac{\pi}{4\theta} - \frac{1}{2}$, which would make the final state equal to $|G\rangle$. While the original Grover search performs only an integer number of iterates, it is possible, as shown in [13] (Theorem 4) to correct the state with a final *partial* iterate. Of course, this correction is only possible because we know the success probability exactly. It requires an arbitrary rotation gate, which can be approximated to a high precision using basic gates, thanks to the Solovay-Kitaev theorem [17,26]. We will consider this implementation to be exact.

Theorem 1 (From Theorem 4 in [13]). *Let $f : \{0,1\}^n \to \{0,1\}$ with an oracle O_f (possibly using an auxiliary state), such that one of the two cases is*

true: there exists a single $x_g \in \{0,1\}^n$ such that $f(x_g) = 1$, or $f^{-1}(1) = \emptyset$. There is a quantum algorithm that does not use any measurement, acts on n qubits initially in the state $|0\rangle$ and outputs $|x_g\rangle$ in the first case. It uses $\left\lceil \frac{\pi}{4} \frac{1}{\arcsin 2^{-n/2}} - \frac{1}{2} \right\rceil \leq \frac{\pi}{4} 2^{n/2} + \frac{1}{2}$ calls to O_f and about $\mathcal{O}(n)$ times more additional gates.

Proof. We run an exact amplitude amplification assuming that a single solution exists, i.e., we run Algorithm 2 with a final, partial iterate to bring the probability of success exactly to 1. □

Thus, we can use Theorem 1 to check if f admits a solution with probability 1: in the first case, by applying f to the output, we obtain 1 with probability 1 (there are no false negatives). In the second, by applying f, we obtain 0 with probability 1 (there are no false positives).

Corollary 1. *Let $f_{\mathcal{T}} : \{0,1\}^n \mapsto \{0,1\}$ be a family of boolean functions such that $f_{\mathcal{T}}^{-1}(1)$ contains 0 or 1 element. Let O_f be a unitary that applies $f_{\mathcal{T}}$ using an auxiliary state $|\mathcal{T}\rangle$: $O_f : |x\rangle |0\rangle |\mathcal{T}\rangle \mapsto |x\rangle |f(x)\rangle |\mathcal{T}\rangle$. Let $g(\mathcal{T}) = 1$ iff $|f_{\mathcal{T}}^{-1}(1)| = 1$. Then there exists a quantum circuit that implements the unitary $O_g : |\mathcal{T}\rangle |0\rangle \mapsto |\mathcal{T}\rangle |g(\mathcal{T})\rangle$. This algorithm uses at most $\frac{\pi}{2} 2^{n/2} + 2$ calls to O_f.*

Proof. For a fixed auxiliary state $\mathcal{T}$, we run the algorithm $\mathcal{A}$ of Theorem 1. It outputs a state which is either $\left|f^{-1}(1)\right\rangle$ (the single preimage of 1 by f) or a superposition of inputs which map to 0. After applying O_f on this output, the result depends only on $\mathcal{T}$. We can uncompute $\mathcal{A}$ and keep only this bit, which specifies whether $f_{\mathcal{T}}$ has a solution or not. Since all the operations applied are unitary, this applies as well to a superposition of auxiliary states$|\mathcal{T}\rangle$. □

3.3 Quantum Collision Search and Pair Generation

The first step of impossible differential attacks is to solve a *limited birthday* problem as defined in Section 2. For this step, we use existing quantum collision search algorithms [25,6]. Indeed, both quantum and classical algorithms for the limited birthday problem reduce it first to a *multiple collision search* problem.

Problem 1 (Multiple collision search). Let $f : \{0,1\}^n \to \{0,1\}^m$ be a random function,with $n \leq m \leq 2n$ and k such that $k \leq 2n - m$. Given query access to f, find 2^k *collision pairs* of f, i.e., 2^k pairs of entries (x, y) such that $x \neq y$ and $f(x) = f(y)$.

The constraints on k, n, m above ensure that enough collisions actually exist. For the best known classical and quantum algorithms, the query complexity matches the time complexity, although the memory consumption can be quite large depending on the respective values of k, n, m. In the classical setting, the problem is solved in time $\mathcal{O}\left(2^{(m+k)/2}\right)$. In the quantum setting, the best known complexities are summarized as follows in [6]:

Theorem 2 (From [6]). *Let $f : \{0,1\}^n \to \{0,1\}^m$, $n \leq m \leq 2n$ be a random function. Let $k \leq 2n - m$. There exists an algorithm finding 2^k collisions in quantum time $\widetilde{\mathcal{O}}\left(2^{C(k,m,n)}\right)$ (incl. qRAM gates), and using $\widetilde{\mathcal{O}}\left(2^{C(k,m,n)}\right)$ quantum queries to f, where:*

$$C(k,m,n) = \max\left(\frac{2k}{3} + \frac{m}{3}, k + \min\left(m-n, \frac{m}{4}\right)\right) \quad . \tag{11}$$

The corresponding query lower bound, valid for any acceptable k and m, was given by Liu and Zhandry in [29]: $\Omega(2^{2k/3+m/3})$. It is conjectured to be tight everywhere. Note that Theorem 2 includes as special cases the BHT collision search algorithm ($k = 0, m = n$) and Ambainis' element distinctness algorithm [1] ($k = 1, m = 2n$). For the latter, a precise analysis was done in by Childs and Eisenberg [14]. They showed that the polynomial factor in the complexity was essentially a constant. By running $\left(\left\lfloor \frac{\pi}{2} 2^{m/3} \right\rceil\right)^2$ evaluations of f, and additional memory operations, a solution is obtained with probability exponentially close to 1. The algorithm of [6], similarly to Ambainis', relies on a quantum walk, so we can also expect its polynomial factor to be small.

In this paper, we assume the evaluation of the function to be the most costly step, so we take these complexities (including the one of Theorem 2) without $\mathcal{O}$ and with a multiplicative factor $\frac{\pi^2}{4}$ only. We focus hereafter only on the exponent.

Formulas for the Quantum Complexity. Like the classical ones, the quantum algorithms for Pair Generation consist in finding multiple collisions in *structures*, which are affine subspaces of $\{0,1\}^n$ of the form $T_x = \{x \oplus v, v \in D_{\text{in}}\}$ (among the inputs) or $T'_x = \{x \oplus v, v \in D_{\text{out}}\}$ (among the outputs). For any input (resp. output) structure we can define a function $H_x : \{0,1\}^{\Delta_{\text{in}}} \to \{0,1\}^{n-\Delta_{\text{out}}}$ (resp. $H'_x : \{0,1\}^{\Delta_{\text{out}}} \to \{0,1\}^{n-\Delta_{\text{in}}}$ which is the restriction of E to the structure, composed with a linear mapping that projects to a space orthogonal to D_{out} (resp. D_{in}). Thus any collision of H_x yields a pair (x,y) such that $x \oplus y \in D_{\text{in}}$ (by definition of the structure) and $E(x) \oplus E(y) \in D_{\text{out}}$ (by definition of H_x). Similarly for collisions of H'_x.

Following [6], we have three choices for input structures, depending on the number of required pairs N:

- If $N < \frac{2^{2\Delta_{\text{in}}}}{2^{n-\Delta_{\text{out}}}} \iff \log_2 N < 2\Delta_{\text{in}} - n + \Delta_{\text{out}}$, then we need only one structure. To recover all the pairs, we need a time exponent :

$$\max\left(\frac{2\log_2 N}{3} + \frac{n-\Delta_{\text{out}}}{3}, \log_2 N + \min\left(n - \Delta_{\text{out}} - \Delta_{\text{in}}, \frac{n-\Delta_{\text{out}}}{4}\right)\right)$$

- If $\frac{2^{2\Delta_{\text{in}}}}{2^{n-\Delta_{\text{out}}}} < 1$, then we fall back on the approach of [25], which is to repeat N times a Grover search among structures, to find one that contains a pair. Checking if a structure of size $2^{\Delta_{\text{in}}}$ contains a pair is an element distinctness problem, which is solved with Ambainis' algorithm in time $\mathcal{O}\left(2^{2\Delta_{\text{in}}/3}\right)$. The time exponent for this case is $\log_2 N + \frac{n-\Delta_{\text{out}}}{2} - \frac{\Delta_{\text{in}}}{3}$.

– If $1 < \frac{2^{2\Delta_{\text{in}}}}{2^{n-\Delta_{\text{out}}}} < N$, we need to consider several structures and to extract all of their collision pairs. This gives a time exponent:

$$\log_2 N + \max\left(\frac{2}{3}(n-\Delta_{\text{in}}-\Delta_{\text{out}}), \min\left(n-\Delta_{\text{out}}-\Delta_{\text{in}}, \frac{n-\Delta_{\text{out}}}{4}\right)\right) \ .$$

We can also swap the roles of Δ_{in} and Δ_{out}.

Conjecture. If we conjecture that the complexity $\mathcal{O}(2^{\frac{2k}{3}+\frac{m}{3}})$ for the multiple collision search problem can actually be reached for all values of k, n and m (and therefore, that the query lower bound is tight everywhere), all the six cases above can be merged into a single formula reminiscent from the classical one.

Conjecture 1. Let $E : \{0,1\}^n \to \{0,1\}^n$ be a block cipher. Given quantum oracles for E and its inverse (O_E and $O_{E^{-1}}$), the limited birthday problem can be solved in quantum time

$$Q_N = \max\left(\min_{\Delta\in\{\Delta_{\text{in}},\Delta_{\text{out}}\}} N^{\frac{2}{3}} 2^{\frac{n-\Delta}{3}}, \min\left(N 2^{\frac{2}{3}(n-\Delta_{\text{out}}-\Delta_{\text{in}})}, \min_{\Delta\in\{\Delta_{\text{in}},\Delta_{\text{out}}\}} N 2^{\frac{n}{2}-\frac{\Delta}{6}-\frac{\Delta_{\text{in}}+\Delta_{\text{out}}}{3}}\right)\right) \ . \quad (12)$$

4 Quantum Pair Filtering

In this section, we focus on the pair filtering step. We design a quantum version of the early-abort algorithm (Algorithm 1) and we study its time complexity.

4.1 Assumptions on the Attack

Recall that we have divided $K_{\text{in}\cup\text{out}} = K_1 \times K_2 \times \cdots \times K_\ell$, and that there exists separate *test functions* $T_1(p,k_1)$, $T_2(p,k_1,k_2), \ldots, T_\ell(p,k_1,\ldots,k_\ell)$ that decide whether a pair p invalidates the key guess $k_1,\ldots,k_\ell$. Our goal is to solve the following problem.

Problem 2. Given the table $\mathcal{T}_0$ that contains the precomputed pairs, find the keys $(k_1,\ldots,k_\ell)$ such that:

$$\exists! p, (T_1(p,k_1) = 1 \wedge \ldots \wedge T_\ell(p,k_1,\ldots,k_\ell) = 1) \iff \mathcal{T}_\ell(k_1,\ldots,k_\ell) = \emptyset \ .$$

Algorithm 1 enumerates all the key guesses for which there exists no invalidating pair, using ℓ nested loops. The quantum version of this algorithm relies on nested amplitude amplification subroutines, using Corollary 1. In order to ensure its correctness, we first need to make some *classical* assumptions on the attack. If these assumptions are satisfied, then we will show that our algorithm solves Problem 2 *with probability 1*. So if the assumptions are satisfied with probability a, then it will succeed with the same probability a.

First of all, we want a single solution.

Assumption 1 *Given the initial table $\mathcal{T}_0$, there exists a single key $(k_1, \ldots, k_\ell)$ such that $\mathcal{T}_\ell(k_1, \ldots, k_\ell) = \emptyset$.*

Second, we need global bounds on the size of intermediate tables.

Assumption 2 *No intermediate table exceeds twice its expected size:*

$$\forall k_1, \ldots, k_i, N(k_1, \ldots, k_i) \leq \left(\prod_{j=1}^{i} \sigma_j \right) N \ .$$

Assumption 2 can be reduced to the assumption that, though it can variate between key guesses, there is an upper bound σ_i on the probability to be filtered.

Assumption 3 *Let $\mathcal{T}_i(k_1, \ldots, k_i)$ be any intermediate table (incl. the case $\mathcal{T}_0$). Then:*

$$\forall k_{i+1}, \Pr_{P \leftarrow \mathcal{T}_i(k_1, \ldots, k_i)} (T_{i+1}(P, k_1, \ldots, k_{i+1})) \leq \sigma_i \ . \tag{13}$$

In other words, the probability to pass the condition T_i can variate, but it has some global upper bound σ_i.

We can then use a Chernoff inequality to upper bound the maximal size of intermediate tables, over all key guesses.

Lemma 1. *Let $N_i(k_1, \ldots, k_i) = |\mathcal{T}_i(k_1, \ldots, k_i)|$. Then under Assumption 3:*

$$\mathop{\mathbb{E}}_{\mathcal{T}_0} (N_i(k_1, \ldots, k_i)) \leq \left(\prod_{j=1}^{i} \sigma_j \right) N_0 \ , \quad \textit{and:} \tag{14}$$

$$\Pr_{\mathcal{T}_0} \left(\exists i, \exists k_1, \ldots, k_i, N_i(k_1, \ldots, k_i) \geq 2 \mathop{\mathbb{E}}_{\mathcal{T}_0} (N_i(k_1, \ldots, k_i)) \right) \leq \sum_{i=1}^{\ell} \left(\prod_{j=1}^{i} |K_j| \right) 2^{-0.48 (\prod_{j=1}^{i} \sigma_j) N_0} \ . \tag{15}$$

Proof. Let us fix i and a choice of $k_1, \ldots, k_i$. For all pairs P in $\mathcal{T}_{i-1}(k_1, \ldots, k_{i-1})$, the events that P fall in $\mathcal{T}_i(k_1, \ldots, k_i)$ are independent and of probability less than σ_i (by Assumption 3). This also allows to deduce:

$$\forall k_1, \ldots, k_i, \Pr_{P \leftarrow \mathcal{T}_0} (P \in \mathcal{T}_i(k_1, \ldots, k_i))$$

$$\leq \Pr_{P \leftarrow \mathcal{T}_0} (T_1(P, k_1) = 1) \times \ldots \times \Pr_{P \leftarrow \mathcal{T}_{i-1}(k_1, \ldots, k_{i-1})} (T_i(P, k_1, \ldots, k_i) = 1) \leq \prod_{j=1}^{i} \sigma_j \ .$$

By independence, we deduce Equation 14. Next, we apply a Chernoff bound for this given choice of $k_1, \ldots, k_i$:

$$\Pr_{\mathcal{T}_0} (N_i(k_1, \ldots, k_i) \geq 2 \mathbb{E} (N_i(k_1, \ldots, k_i))) \leq e^{-\mathbb{E}(N_i(k_1, \ldots, k_i))/3} \ .$$

By summing this over all choices of key sequences, we obtain the bound of Equation 15. □

Thanks to Lemma 1, we can check that Assumption 2 is satisfied with high probability. In practice, we can just stop the filtering process when the intermediate tables become expectedly too small, e.g. of size below 2^{16}.

4.2 Filtering of a Table

From now on, we assume that Assumption 1 and Assumption 2 are satisfied. Let $M_i = 2\,\mathbb{E}\,(N_i) = 2\left(\prod_{j\leq i}\sigma_j\right)N_0$ be the maximal size of all intermediate tables $\mathcal{T}_i$. We manipulate quantum states that contain either:

- Partial key guesses k_i: in that case, we just use a register with as many qubits as there are bits in k_i
- Intermediate tables $\mathcal{T}_i$: in that case, we allocate $M_i = 2\,\mathbb{E}\,(N_i)$ qubit registers, where each register holds either a pair or a dummy element. By Assumption 2, this is enough space to represent all intermediate tables; thus, although this data can be in superposition (as it depends on the key guess), there is no case in which the table would exceed the space that we allocated to write it.

We introduce additional notations to make precise time and memory complexity estimates. Since we want to count the time complexity relatively to a cipher evaluation, we introduce: • t_i the time to evaluate the condition T_i; • t the time to perform an operation such as a copy or swap on a register that contains a pair. We count the memory complexity in number of pairs (a pair can be stored on a $4n$-bit register).

First of all, we compute the time to filter an intermediate table.

Lemma 2. *For all i, there exists a quantum circuit F_i that maps:*

$$|\mathcal{T}_i(k_1,\ldots,k_i)\rangle\,|k_{i+1}\rangle\,|0\rangle \mapsto |\mathcal{T}_i(k_1,\ldots,k_i)\rangle\,|k_{i+1}\rangle\,|\mathcal{T}_{i+1}(k_1,\ldots,k_i,k_{i+1})\rangle\,|*\rangle \quad ,$$

where $*$ are computation qubits that depend only on $k_1,\ldots,k_i,k_{i+1}$. The time complexity, relative to a cipher evaluation, is bounded by:

$$\begin{cases} M_i(t_{i+1}+t) \text{ in the QRAQM model} \\ M_i(t_{i+1}+\frac{(\log_2 M_i)^2}{4}t) \text{ otherwise} \end{cases} \tag{16}$$

and the memory complexity by M_i or $M_i\frac{(\log_2 M_i)^2}{4}$ (mainly due to the $*$ state).

Proof. We start by computing the condition T_i for all pairs in the table $\mathcal{T}_i(k_1,\ldots,k_i)$. We store the result (0 or 1) in additional qubits. We initialize the output table $\mathcal{T}_{i+1}$ with M_{i+1} "dummy" pairs. Then, we take all the good pairs in $\mathcal{T}_i$ and move them to $\mathcal{T}_{i+1}$. There are two ways to do so.

- If we can use qRAM gates, this operation is efficient. We create two index registers, one to iterate over the elements of $\mathcal{T}_i$ and one to keep track of the next position to write in $\mathcal{T}_{i+1}$. Each time we increment the second index, we move to the next good pair in $\mathcal{T}_i$ and copy it. This requires only $|\mathcal{T}_i| = M_i$ increment and copy operations.

– If we don't want to use qRAM gates, the above cannot be applied, as it requires to read from $\mathcal{T}_i$ in superposition. The alternative is to apply a reversible sorting network, which will move all the elements of $\mathcal{T}_i$ flagged "1" to, say, the M_i first indices in the memory. We then copy these M_i cells (conditioned on whether the copied pair is actually good).

In the second case, the amount of computations increases by a factor $\frac{(\log_2 M_i)^2}{4}$ due to the sorting network. However, the sorting network contains only comparators and swaps, and does not require to recompute the filtering conditions.

It should be noted that the $|*\rangle$ is here to ensure reversibility of these operations. We could perform uncomputations to erase it immediately, but we prefer to wait until the table $\mathcal{T}_{i+1}(k_1, \ldots, k_i, k_{i+1})$ is not needed anymore. Then we will erase not only the table, but also all of the computations that led to it.

Remark 1 (Reversible sorting networks). In order to sort a table of 2^ℓ values, we use the *odd-even mergesort* of Batcher [2]. It uses $S(2^\ell) = 2^{\ell-1}\frac{\ell(\ell-1)}{2} + 2^\ell - 1$ comparators and swaps of registers, that we can approximate to $2^{\ell-2}(\ell^2)$ n-qubit register operations (in practice, these operations remain less costly than our other cost metrics, *e.g.*, S-Boxes). In order to make it reversible, we need $S(2^\ell)$ qubits that contain the results of the comparators.

4.3 Exact Pair Filtering

We can build our Pair Filtering algorithm by interleaving calls to the filtering circuit of Lemma 2 with instances of exact amplitude amplification. For a current set of partial key guesses $(k_1, \ldots, k_i)$, with a corresponding intermediate table, we use Corollary 1 to check whether this set of key guesses leads to the single solution or not. We use $k_1^g, \ldots, k_\ell^g$ to denote this solution.

Lemma 3. *Let $1 \leq i \leq \ell$. Let t_i be the time (in quantum operations) to compute the condition T_i. There exists a quantum circuit (unitary) U_i that, on an input state of the form:*

$$|k_1, \ldots, k_i\rangle\, |\mathcal{T}_{i-1}(k_1, \ldots, k_{i-1})\rangle \quad ,$$

writes 1 in an output qubit iff there exists a completion $k_{i+1}, \ldots, k_\ell$ such that $k_1, \ldots, k_\ell$ is the good key, and 0 otherwise. It runs in time:

$$2\sum_{j=i}^{\ell} M_{j-1}(t_j + t)\left(\frac{\pi}{2}\right)^{j-i}\sqrt{\prod_{m=i}^{j}|K_m|} \quad , \text{ using a memory } M_{i-1}.$$

Proof. We start by describing U_ℓ, the innermost step. It starts from the table $\mathcal{T}_{\ell-1}(k_1, \ldots, k_{\ell-1})$, and $k_1, \ldots, k_\ell$. It simply needs to compute T_ℓ for all the elements of the table, and check if one of them passes the test (if it's the case, then this is not the good key guess). This is done in time $2M_{\ell-1}(t_\ell + t)$ (including an uncomputation).

Next, we assume the existence of U_{i+1} for some $i \geq 1$, and we show how to build U_i from U_{i+1}.

Consider an input state of the form: $|k_1, \ldots, k_i\rangle\,|\mathcal{T}_{i-1}(k_1, \ldots, k_{i-1})\rangle$. We first use the circuit F_{i-1} of Lemma 2 to create the table $|\mathcal{T}_i(k_1, \ldots, k_i)\rangle$ (as well as the additional $|*\rangle$ state). Next, we try to find k_{i+1} such that $k_1, \ldots, k_i, k_{i+1}$ can be completed into the good key. This is the situation that we considered in Corollary 1. We have:

- an auxiliary state $|\mathcal{T}\rangle$ containing $|k_1, \ldots, k_i\rangle$, the tables $\mathcal{T}_{i-1}$ and $\mathcal{T}_i$;
- a search space K_{i+1} of fixed size, containing exactly one or zero solution;
- a test function that, using the auxiliary state, can find whether k_{i+1} can be completed into the good key: this is the unitary U_{i+1} given by our recurrence hypothesis.

By Corollary 1, which uses a single instance of amplitude amplification, we obtain a unitary that tests whether the current auxiliary state leads to a solution or not, which uses less than $\frac{\pi}{2}\sqrt{|K_{i+1}|} + 2$ calls to U_{i+1}. We apply this unitary. Finally, we uncompute the circuit F_{i-1} to delete the table $\mathcal{T}_i$ and the associated computation qubits.

We have by recursion, and by Lemma 2 (assuming the qRAM case):

$$\begin{cases} \mathsf{time}(U_i) = \left(\frac{\pi}{2}\sqrt{|K_{i+1}|} + 2\right)(t + \mathsf{time}(U_{i+1})) + 2M_{i-1}(t_i + t) \\ \mathsf{time}(U_\ell) = 2M_{\ell-1}(t_\ell + t) \end{cases}$$

so we approximate $\mathsf{time}(U_i)$ to:

$$2\sum_{j=i}^{\ell} M_{j-1}(t_j + t)\left(\frac{\pi}{2}\right)^{j-i}\sqrt{\prod_{m=i+1}^{j}|K_m|}\ .$$

The memory complexity is equal to the sum of all $M_{j-1}(t_j+t)$ for the involved filtering circuits. In practice, the size of the first table dominates all the others significantly, and we can approximate it by $M_{i-1}(t_i + t)$. □

Our pair filtering algorithm is then obtained by using U_1 in a quantum search for k_1, where we start from the table $\mathcal{T}_0$ (this one is also an exact amplitude amplification, since we assumed that there was exactly one solution). The memory complexity is dominated by $|\mathcal{T}_0|$.

Corollary 2. *Under our assumptions, there exists a quantum pair filtering algorithm that finds the single good $k_1, \ldots, k_\ell$ in time:*

$$2N\sum_{j=1}^{\ell}(t_j + t)\left(\prod_{m=1}^{j-1}\sigma_m\right)\left(\frac{\pi}{2}\right)^{j}\sqrt{\prod_{m=1}^{j}|K_m|}\ ,$$

and using little more than N memory.

Compared with the classical formula (Equation 10), we have put a square root on each $|K_i|$. Indeed, the algorithm that we obtain is no more than Algorithm 1 in which each exhaustive search loop on a partial key subspace is replaced by a quantum search.

Example. We take an example with 3 conditions (Algorithm 3). Here, there are three nested amplitude amplification routines.

- the innermost one searches a good $k_3 \in K_3$ if it exists, by filtering $\mathcal{T}_2$: it runs in time $\frac{\pi}{2}\sqrt{|K_3|} \times (\sigma_1\sigma_2 N)(t_3 + t)$
- the middle one searches a good $k_2 \in K_2$ if it exists, by filtering $\mathcal{T}_1$ and building $\mathcal{T}_2$: it runs in time:

$$\frac{\pi}{2}\sqrt{|K_2|}\left((\sigma_1 N)(t_2 + t) + \frac{\pi}{2}\sqrt{|K_3|}(\sigma_1\sigma_2 N)(t_3 + t)\right)$$

- the outermost one searches a good $k_1 \in K_1$:

$$\frac{\pi}{2}\sqrt{|K_1|}\left(N(t_1 + t) + \frac{\pi}{2}\sqrt{|K_2|}\left((\sigma_1 N)(t_2 + t) + \frac{\pi}{2}\sqrt{|K_3|}(\sigma_1\sigma_2 N)(t_3 + t)\right)\right) ,$$

  which we can simplify into:

$$\frac{\pi}{2}\sqrt{|K_1|}\left(N + \frac{\pi}{2}\sqrt{|K_2|}\left((\sigma_1 N) + \frac{\pi}{2}\sqrt{|K_3|}(\sigma_1\sigma_2 N)\right)\right)\max_i(t_i + t) \ .$$

If we do not want to use qRAM, we can replace the term $\max_i(t_i + t)$ by $(\max_i t_i) + \frac{(\log_2 N)^2}{4}t$, according to Lemma 2 ($N$ is the size of the largest table that we will have to sort during the algorithm).

4.4 Improved Pair Filtering in Practice

In some cases, we can improve the computation of intermediate tables given in Lemma 2. Since usually, the treatment of the first table ($\mathcal{T}_0$, of size N) dominates the other steps, we will focus on this one. Note that these improvements do not incur any change in the rest of the algorithm.

Precomputation of the Conditions. Let us assume that the first filtering condition $T_1(p, k_1)$ depends actually only on a small part of the pair p, say p_1. Instead of having to compute T_1 for all p in $\mathcal{T}_0$, we can recompute the entire lookup table of T_1. Then, given k_1, we can obtain the next pairs in two steps: (1) we read the values of the p_1 such that $T(p_1, k_1) = 1$; (2) for each p_1, we fetch the pairs that have this value.

Previously, we needed a time $N(t_1+t)$ (with QRAQM) to compute $\mathcal{T}_1(k_1)$ for a given k_1. Now, we can reduce this time to $2\sigma_1 N t$, assuming the precomputation of T_1. Indeed, we can sort the pairs in $\mathcal{T}_0$ by their p_1 value, in order to read them efficiently.

We can also use this precomputation at a later step. For example, if $T_2(p, k_1, k_2)$ depends actually only on p_2, then we can also:

- precompute the lookup table of $T_2(\cdot, k_1, \cdot)$ for a given k_1, before running the nested search on k_2;
- for a given k_2, compute $\mathcal{T}_2(k_1, k_2)$ efficiently in time $2\sigma 2\sigma_1 N t$ instead of time $2\sigma_1 N(t_2 + t)$

However, this now requires to sort $\mathcal{T}_1(k_1)$ with respect to p_2 when we compute it, and this additional term must be taken into account.

Algorithm 3 Quantum pair filtering with 3 conditions (simplified).

Input: a table $\mathcal{T}_0$ for the pairs
Output: the key k_1, k_2, k_3 for which there is no invalidating pair

Initialize registers for k_1, k_2, k_3 to 0
Apply a Hadamard transform on the k_1 register
Repeat $\frac{\pi}{4 \arcsin |K_1|^{-1/2}} - \frac{1}{2}$ **times**
▷ Actually an amplitude amplification with partial final iterate
  Test k_1**:** ▷ At this point, $|k_2, k_3\rangle$ contain $|0, 0\rangle$
  Compute $\mathcal{T}_1(k_1) = \{p \in \mathcal{T}_0, T_1(p, k_1) = 1\}$
  Apply the following unitary U_1:
  Apply a Hadamard transform on the k_2 register
  Repeat $\frac{\pi}{4 \arcsin |K_2|^{-1/2}} - \frac{1}{2}$ **times**
    Test k_2**:** ▷ At this point, $|k_3\rangle$ contains $|0\rangle$
    Compute $\mathcal{T}_2(k_1, k_2) = \{p \in \mathcal{T}_1(k_1), T_2(p, k_2) = 1\}$
    Apply the following unitary U_2 on $|k_3\rangle$:
    Apply a Hadamard transform on the k_3 register
    Repeat $\frac{\pi}{4 \arcsin |K_3|^{-1/2}} - \frac{1}{2}$ **times**
      Flip the phase if $\mathcal{T}_3$ is empty
      Apply the Grover diffusion transform on $|k_3\rangle$
    EndRepeat
    If a solution was found, flip the phase of $|k_2\rangle$
    Uncompute U_1
    Erase $\mathcal{T}_1(k_1)$
    End of test for k_2
    Apply the Grover diffusion transform on $|k_2\rangle$
  EndRepeat
  If a solution was found, flip the phase of $|k_1\rangle$
  Uncompute U_1
  Erase $\mathcal{T}_1(k_1)$
  End of test for k_1
  Apply the Grover diffusion transform on $|3\rangle$
EndRepeat

Quantum Filtering. It is possible to speed up the filtering using quantum search. Indeed, when computing $\mathcal{T}_1(k_1)$ from $\mathcal{T}_0$, we expect to find $\sigma_1 N$ filtered pairs among the N initial ones. In the QRAQM model, we can find filtered pairs using a Grover search among all the pairs. We need about $\frac{1}{\sqrt{\sigma_1}}$ search iterates, thus all of them are found in time $\sqrt{\sigma_1 N}$ instead of N. However, we must fix a number of quantum search iterates for a computation that is done in superposition over k_1. Thus, there will be some probability of error.

Assumption 2 tells us that the actual number of filtered pairs, at any level, does not exceed twice its expectation. By strengthening this assumption a little, we can suppose that the actual probability of success of a search for good pairs will fall in $[a/2; 3a/2]$ where a is the expected success probability. Then, by Lemma 5 in [10], the search succeeds with probability larger than $3/4$ (if we

measured its result immediately). If the table being constructed is large enough, this is sufficient.

Lemma 4. *There exists a quantum circuit QF_i that performs the same operation as in Lemma 2, up to an error ε bounded by $\varepsilon \leq e^{\frac{-1}{12}M_{i+1}}$. The time complexity, relative to a cipher evaluation, is bounded by: $2M_i\frac{\pi}{2}\sqrt{\sigma_{i+1}}(t_{i+1}+t)$, with QRAQM, and the memory complexity by M_i.*

Proof. Since we are building the table $\mathcal{T}_{i+1}$ from $\mathcal{T}_i$ and a new key guess k_{i+1}, we know that there is a maximum of $2\,\mathbb{E}\,(N_{i+1})$ pairs to look for in all cases. We run in parallel $4\,\mathbb{E}\,N_{i+1} = 2M_{i+1}$ quantum searches with $\frac{\pi}{4}\frac{1}{\sqrt{\sigma}}$ search iterates each. The search space is the current table of all pairs. The condition of success is the test function T_i. By our assumption, such an individual search succeeds with probability at least $3/4$ for all values of k.

We move all the obtained (possibly) good pairs into the next table. At this point, errors occur if the individual searches fail to obtain all the N_i independent pairs. We perform the following test on the search results: we test if at least $2\,\mathbb{E}\,(N_{i+1})$ contain good pairs. If this is the case, then we can expect to have met all the N_{i+1} pairs independently (up to some random collisions). We can write the result of this test in a qubit, but we do not measure it: we only use it to bound away the cases of failure.

Let us consider a fixed key k. We already know that $N_{i+1} \leq 2\,\mathbb{E}\,(N_{i+1})$. The total number of successful searches among $4\,\mathbb{E}\,(N_{i+1})$ is the sum X of $4\,\mathbb{E}\,(N_{i+1})$ independent random variables X_j. By our assumption, each X_j has expectation at least $3/4$ and by a Chernoff-Hoeffding bound:

$$\Pr_{\mathcal{T}_i}(X \leq 2\,\mathbb{E}\,(N_{i+1})) \leq \Pr_{\mathcal{T}_0}\left(X \leq \frac{2}{3}\,\mathbb{E}\,(X)\right) \leq e^{-(1/3)^2\frac{\mathbb{E}(X)}{2}} \leq e^{\frac{-\,\mathbb{E}\left(N_{i+1}\right)}{6}} = e^{\frac{-1}{12}M_{i+1}} \quad .$$

By using this erroneous QF_i in a quantum search with t iterates, we will obtain a total error amplitude of about $t\sqrt{\varepsilon}$ (i.e. $t^2\epsilon$ failure probability). This is the result of a standard hybrid argument [4]. Therefore, if M_{i+1} is sufficiently small (typically 2^{16}), it remains negligible.

5 Application to SKINNY

In this section, we describe an impossible differential attack on SKINNY-128-256 and its quantization.

5.1 Description

SKINNY-128-256 [3] is a SPN tweakable block cipher inspired by the AES. It relies on the *tweakey* framework, so the 128-bit tweak and the 128-bit key are glued together in a 256-bit tweakey. The goal of our attack is to recover this 256-bit string. The 128-bit internal state is represented by a 4×4 matrix of s-bit cells ($s = 8$).

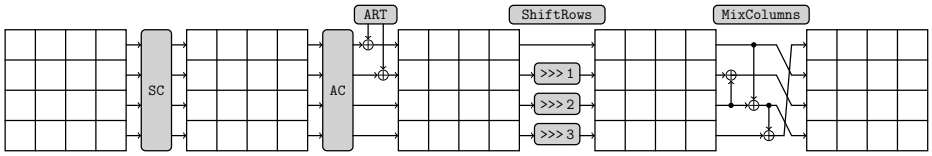

Fig. 3. Skinny round function (image from [3])

Round function. The round function of SKINNY is represented in Figure 3. The operations are:

- *SubCells* (*SC*): an 8-bit S-Box is applied to each cell of the matrix. Its description in given in [3].
- *AddConstants*(*AC*): A constant depending on the round number is XORed to the internal state. This operation is irrelevant for differential attacks.
- *AddRoundTweakey* (*ART*): The tweakey material, derived from the tweakey, is XORed to the two first lines of the internal state. A tweakey schedule is used to compute the tweakey material.
- *ShiftRows* (*SR*): This operation leaves the first line intact, rotates the second line by 1 cell, the third line by 2 cells and the fourth line by 3 cells.
- *MixColumns* (*MC*): This operation multiplies each column by a (non-MDS) matrix M

Tweakey schedule. The tweakey schedule (Figure 4) takes a crucial part in the attack. The internal state is composed of *two* 4×4 matrices of bytes. The tweakey schedule follows this pattern:

1. The two first lines of both matrices are extracted. Then the lines with the same position are XORed to build the tweakey material used in the round function.
2. A cell permutation P_T is applied to both matrices. We can remark that the permutation globally swaps the two first lines and the two last lines.

$$\begin{pmatrix} 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 \\ 12 & 13 & 14 & 15 \end{pmatrix} \overset{P_T}{\longmapsto} \begin{pmatrix} 9 & 15 & 8 & 13 \\ 10 & 14 & 12 & 11 \\ 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \end{pmatrix}$$

3. An LFSR is applied to the cells of the first two collumns of the second matrix. It is important to remark that the LFSR used here (described in [3]) is linear and that there exists non zero elements of order 15, i.e., $\exists x$ $\text{LFSR}^{15}(x) = x$.

5.2 Classical Impossible Attack

In this section, we give an impossible differential attack on SKINNY that achieves 21 rounds based on work from [33]. Since the goal is to recover the full 256-bit tweakey, the generic bound is 2^{256}.

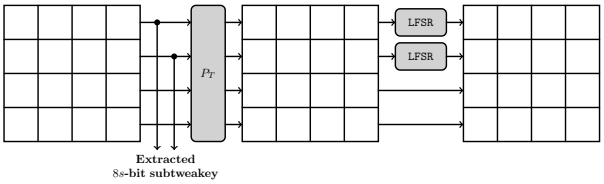

Fig. 4. Tweakey schedule (from [3]).

Impossible Pattern. We give the pattern in Figure 5.

Pair Generation. Here we give some parameters of the attack described in [33]. With our notation we get $\Delta_{\text{in}} = 32$, $\Delta_{\text{out}} = 72$, $c_{\text{in}} = 24$, $c_{\text{out}} = 72, N = 2^{103.17}$, where N ensures that only one subkey will remain in the end.

Pair Filtering. In this section we will describe the tweakey recovery.

1. Round 21
   (a) From the knowledge of the ciphertext, compute $\Delta X_{21}[8]$ and $\Delta X_{21}[12]$. Based on the properties of MC operation on col(1) of W_{20}, we have $\Delta X_{21}[8] = \Delta X_{21}[12]$. This results in an s-bit filter.
   (b) For this step we guess the value of $TK_{21}[0]$. From the knowledge of $Z_{21}[12]$ and $\Delta Z_{21}[12]$, we can compute $\Delta X_{21}[12]$. Based on the properties of MC operation on col(1) of W_{20}, we have $\Delta X_{21}[0] = \Delta X_{21}[12]$. Since $Y_{21}[0] = Z_{21}[0] \oplus TK_{21}[0]$, we perform a single-cell (s-bit) filter on the pairs by verifying the following equations.

$$S^{-1}(Y_{21}[0]) \oplus S^{-1}(Y_{21}[0] \oplus \Delta Y_{21}[0]) = \Delta X_{21}[0] \ .$$

   (c) For this step we guess the value of $TK_{21}[5]$. From the knowledge of $Z_{21}[13]$ and $\Delta Z_{21}[13]$, we can compute $\Delta X_{21}[13]$. Based on the properties of MC operation on col(2) of W_{20}, we have $\Delta X_{21}[5] = \Delta X_{21}[13]$. Since $Y_{21}[5] = Z_{21}[5] \oplus TK_{21}[5]$, we perform a single-cell (s-bit) filter on the pairs by verifying the following equations.

$$S^{-1}(Y_{21}[5]) \oplus S^{-1}(Y_{21}[5] \oplus \Delta Y_{21}[5]) = \Delta X_{21}[5] \ .$$

   (d) For this step we guess the value of $TK_{21}[1]$. Therefore with the knowledge of the ciphertext, we can compute ΔZ_{20} and check $\Delta Z_{20}[1] = \Delta TK_{20}[1]$. This leads to an s-bit filter.

$$S^{-1}(Y_{21}[1]) \oplus S^{-1}(Y_{21}[1] \oplus \Delta Y_{21}[1]) = \Delta X_{21}[1] \ .$$

   (e) For this step we guess the value of $TK_{21}[2]$. Based on the properties of MC operation on col(3) of W_{20} , we have $\Delta X_{21}[2] = \Delta X_{21}[14]$. Since $Y_{21}[2] = Z_{21}[2] \oplus TK_{21}[2]$ we perform an s-bit filter on the pairs by verifying the following equation.

$$S^{-1}(Y_{21}[2]) \oplus S^{-1}(Y_{21}[2] \oplus \Delta Y_{21}[2]) = \Delta X_{21}[2] \ .$$

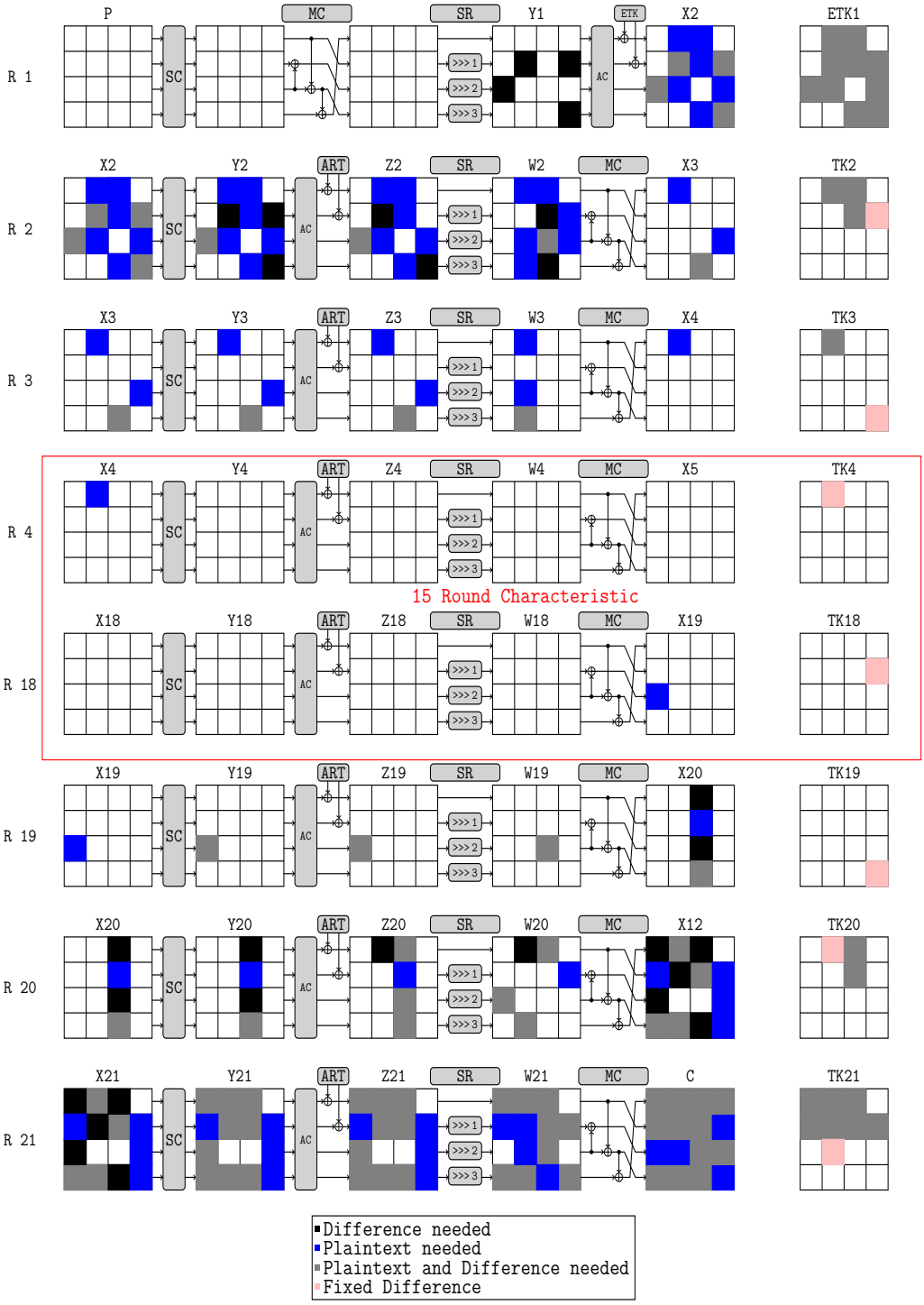

Fig. 5. Impossible pattern (from [33]).

(f) For this step we guess the value of $TK_{21}[6]$. Based on the properties of MC operation on col(3) of W_{20} , we have $\Delta X_{21}[6] = \Delta X_{21}[14]$. Since $Y_{21}[6] = Z_{21}[6] \oplus TK_{21}[6]$, we perform an s-bit filter on the pairs by verifying the following equation.

$$S^{-1}(Y_{21}[6]) \oplus S^{-1}(Y_{21}[6] \oplus \Delta Y_{21}[6]) = \Delta X_{21}[6] \ ,$$

To continue the key recovery, we will perform two additional key guesses $TK_{21}[4,7]$ so that the value of $X_{21}[4,7]$ is known.

2. Round 20. For this step we guess the value of $TK_{20}[2]$. From the knowledge of $Z_{20}[14]$ and $\Delta Z_{20}[14]$, we can compute $\Delta X_{20}[14]$ and $\Delta X_{20}[10]$. Based on the properties of MC operation on col(3) of W_{19} , we have $\Delta X_{20}[14] = \Delta X_{20}[2] = \Delta X_{20}[10]$. Since $Y_{20}[2] = Z_{20}[2] \oplus TK_{20}[2]$, we perform a two-cell ($2s$-bit) filter on the pairs by verifying the following equations:

$$S^{-1}(Y_{20}[2]) \oplus S^{-1}(Y_{20}[2] \oplus \Delta Y_{20}[2]) = \Delta X_{20}[14].$$
$$\Delta X_{20}[14] = \Delta X_{20}[10]$$

3. Round 19. For this step we guess the value of $TK_{20}[6]$. Hence we can compute the value of $\Delta X19[8]$ and check if it matches our impossible pattern.
4. Round 1.
   (a) Guess $ETK[7]$. Hence by using the knowledge of the plaintext, we can compute $\Delta Y_2[7]$ and check $\Delta Y_2[7] = \Delta TK_2[7]$.
   (b) For this step we guess the value of $ETK[8]$. From the knowledge of the plaintext, we can compute $\Delta Y_2[8]$ and $\Delta Y_2[15]$. Based on the properties of MC operation on col(3) of W_2 , we have $\Delta Y_2[8] = \Delta Y_2[15]$. We thus perform a single-cell (s-bit) filter on the pairs.
   (c) For this step we guess the value of $ETK[1]$. From the knowledge of the plaintext, we can compute $\Delta Y_2[5]$ and $\Delta Y_2[8]$. Based on the properties of MC operation on col(3) of W_2 , we have $\Delta Y_2[5] = \Delta Y_2[8]$. We thus perform a single-cell (s-bit) filter on the pairs.
   (d) To continue the key recovery, we will perform two additional key guesses $ETK[2,9,11]$ so that the value of $Y_2[2,9,11]$ is known.
5. Round 2 and 3. At this step we have performed enough guesses on the tweakey material to know the value of $TK_3[1]$. We guess $TK_2[1,2,6]$ and compute Y_3 and ΔY_3. Therefore, from the knowledge of $TK_3[1]$, $\Delta Y_4[1]$ can be simply determined. Checking if $\Delta Y_4[1] = TK_4[2]$, will lead to an s-bit filter.

In Table 2 we give the cost of each step. In the end, the time complexity for the classical pair filtering is $2^{143.17}$.

5.3 Quantum impossible attack

Since 2 plaintext-ciphertext pairs are required to discriminate the right tweakey, the exhaustive search with Grover's algorithm has a complexity of $2 \cdot \frac{\pi}{2} \cdot 2^{129.65}$ encryptions. Our quantum attack will be valid if we manage to outperform this. From Section 3.3 we compute the complexity of the quantum pair generation as $Q_N = 2^{119.17}$ encryptions (in the classical setting $C_N = 2^{128}$ encryptions).

Table 2. Successive steps of the attack on SKINNY. Each step reduces the table size but increases the key space.

Index	σ_i	$\lvert K_i\rvert$	Quantum partial complexity: $\prod_{j=1}^{i-1}\sigma_j\sqrt{\prod_{j=1}^{i-1}\lvert K_j\rvert}$	Classical partial complexity: $\prod_{j=1}^{i-1}\sigma_j\prod_{j=1}^{i-1}\lvert K_j\rvert$
1.a	2^{-s}	2^0	2^{-s}	2^{-s}
1.b	2^{-s}	2^s	$2^{-3s/2}$	2^{-s}
1.c	2^{-s}	2^s	2^{-2s}	2^{-s}
1.d	2^{-s}	2^s	$2^{-5s/2}$	2^{-s}
1.e	2^{-s}	2^s	2^{-3s}	2^{-s}
1.f	2^{-s}	2^s	$2^{-7s/2}$	2^{-s}
	2^0	2^{2s}	$2^{-5s/2}$	2^s
2	2^{-2s}	2^s	2^{-4s}	2^0
3	2^{-s}	2^s	$2^{-9s/2}$	2^0
4.a	2^{-s}	2^s	2^{-5s}	2^0
4.b	2^{-s}	2^s	$2^{-11s/2}$	2^0
4.c	2^{-s}	2^s	2^{-6s}	2^0
4.d	2^0	2^{3s}	$2^{-9s/2}$	2^{3s}
5	2^{-s}	2^{3s}	2^{-4s}	2^{5s}

Filtering. Following the idea of Section 4, we build the quantum key recovery based on the classical key recovery described in Section 5.2. We use the formula of Corollary 2 to compute the quantum complexity of this step. This formula features a $(t+t_i)$ term that carries the complexity of $4n$ bit operations and the complexity of performing the test T_i. We assume that $(t+t_i)$ is smaller than the complexity of an encryption, so our formula with the values in Table 2, we get a quantum time of about $2^{96.83}$ encryptions. Putting together the complexities of pair generation and pair filtering, we give the total complexity for 21 rounds of SKINNY-128-256 in Table 3.

6 Application to 7-round AES

In this section, we use our framework to improve the key-recovery attacks against 7-round AES-192 and AES-256 in the quantum setting (the best previous attacks were Square attacks in [10]). The path of the attack that we consider is the one from [31] (Figure 6).

Like SKINNY, the block cipher standard AES [16] is an SPN represented as a 4×4 matrix of 8-bit cells. There exists three versions (AES-128, -192, -256) which change the number of rounds and the key schedule algorithm, but our attack will be independent of the key schedule and we will not consider any relation between subkey bytes.

Table 3. Time complexity of the different steps for 21-round SKINNY-128-256 (in encryptions).

Complexity	Quantum time	Quantum mem.	Classical time	Classical mem.
Pair Generation	$2^{119.17}$	2^{24}	2^{128}	negl.
Pair Filtering	$2^{96.834}$	$2^{103.17}$	$2^{143.17}$	$2^{103.17}$
Total	$2^{119.17}$	$2^{103.17}$	$2^{143.17}$	$2^{103.17}$
Generic	$2^{129.65}$	negl.	2^{256}	1

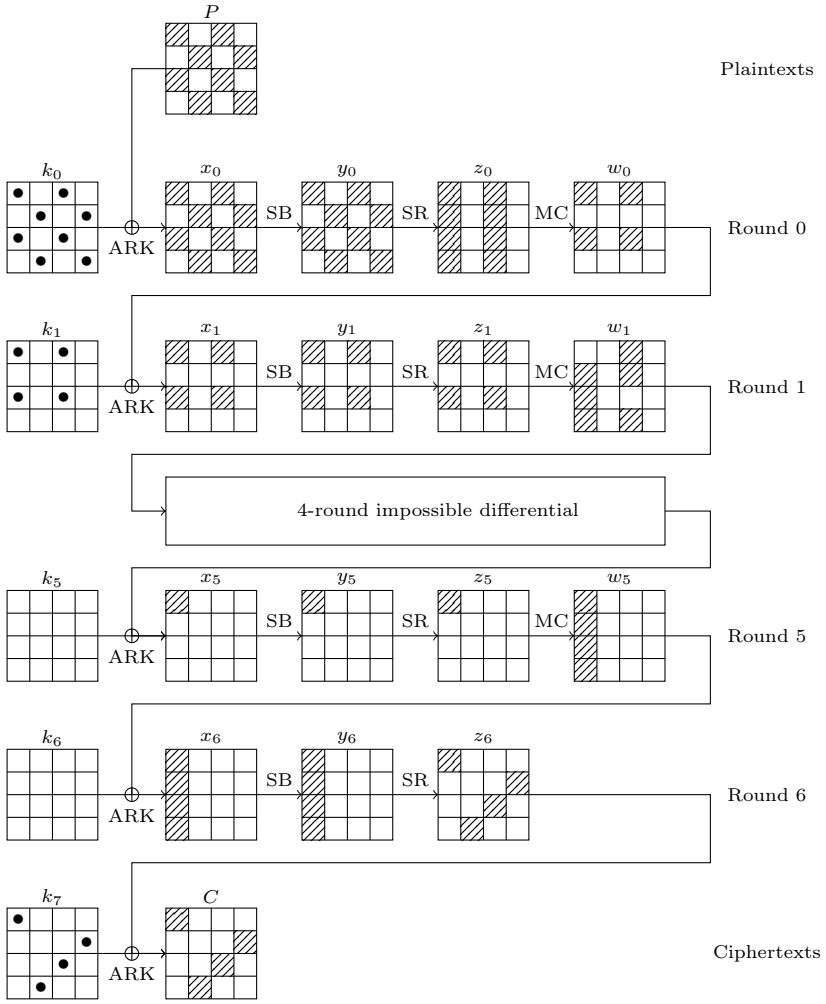

Fig. 6. Path of the attack on 7-round AES of [31]. Key guesses are indicated by •.

A round of AES applies the following operations, as represented in Figure 6: **AddRoundKey** (ARK) XORs the current round key to the state; **SubBytes** (SB) applies the AES S-Box to the cells in parallel, **ShiftRows** (SR) rotates row i by i cells left, **MixColumns** (MC) multiplies the columns by an MDS matrix. We number the rounds starting from 0 and the successive states within round i as x_i, y_i, z_i, w_i, respectively after ARK, SB, SR and MC.

Attack. Let $(P, P'), (C, C')$ be a pair satisfying the limited-birthday conditions on Figure 6. It will invalidate any guess of these 16 bytes of key such that:

- $w_0[1] = w'_0[1]$ and $w_0[3] = w'_0[3]$: this is a 2-byte condition on $k_0[0, 5, 10, 15]$, $p_0[0, 5, 10, 15]$ and $p'_0[0, 5, 10, 15]$. For a given pair, there exists on average 2^{16} matching guesses of $k_0[0, 5, 10, 15]$.
- $w_0[9] = w'_0[9]$ and $w_0[11] = w'_0[11]$: this is an independent 2-byte condition on $k_0[2, 7, 8, 13], p_0[2, 7, 8, 13]$ and $p'_0[2, 7, 8, 13]$. For a given pair, there are on average 2^{16} matching guesses of $k_0[2, 7, 8, 13]$.
- $w_1[0] = w'_1[0]$: this is a single-byte condition on $k_1[0, 10]$, and all the previous plaintext bytes. For a given pair, there are on average 2^8 matching guesses of $k_1[0, 10]$.
- $w_1[10] = w'_1[10]$: this is another single-byte condition with independent key bytes $k_1[2, 8]$, but the same plaintext bytes as before.
- $MC^{-1}(w_5[0, 1, 2, 3])$ is a single byte: this is an independent condition on $k_7[0, 7, 10, 14]$ and the corresponding ciphertext bytes. For a given pair, there are on average 2^8 guesses of k_7 that satisfy this condition.

Thus each pair removes 2^{56} key guesses from a space 2^{128}. In order to have a single remaining key guess, we take $N = 2^{78.5}$. These pairs can be obtained in about $2^{99.8}$ Q2 queries. There are 4 subkey spaces: $K_1 = k_0[0, 5, 10, 15]$, $K_2 = k_0[2, 7, 8, 13]$, $K_3 = k_7[0, 7, 10, 14]$ and $K_4 = k_1[0, 10, 2, 8]$, each of size 2^{32}. The filtering probabilities are $\sigma_1 = 2^{-16}$, $\sigma_2 = 2^{-16}$, $\sigma_3 = 2^{-24}$.

Using the simple early-abort strategy, we obtain a classical time complexity:

$$2^{32}\Bigg(\underbrace{2^{78.5}}_{\text{Build } \mathcal{T}_1} + 2^{32}\Bigg(\underbrace{2^{62.5}}_{\text{Build } \mathcal{T}_2} + 2^{32}\Bigg(\underbrace{2^{46.5}}_{\text{Build } \mathcal{T}_3} + 2^{32}\Bigg(\underbrace{2^{22.5}}_{\text{Sieve } \mathcal{T}_3}\Bigg)\Bigg)\Bigg)\Bigg) = 2^{150.5} \ .$$

And using Corollary 2, we obtain a quantum time complexity:

$$2N\Bigg(\left(\frac{\pi}{2}\right)\sqrt{|K_1|} + \left(\frac{\pi}{2}\right)^2 \sigma_1\sqrt{|K_1||K_2|} + \left(\frac{\pi}{2}\right)^3 \sigma_1\sigma_2\sqrt{|K_1||K_2||K_3|} + \left(\frac{\pi}{2}\right)^4 \sigma_1\sigma_2\sigma_3\sqrt{|K_1||K_2||K_3||K_4|}\Bigg) \max_i(t + t_i) \ , \quad (17)$$

where $t + t_i$ is the time to evaluate the test functions relatively to an encryption. In the case of AES, to facilitate comparison with [10], we count the number of S-Boxes, the S-Box being the most costly component in quantum circuits for the AES. We remark that computing each test function requires 8 S-Boxes, so we approximate the factor $t + t_i$ by 8. This gives:

$$2^{79.5} \cdot 8 \cdot \left(\left(\frac{\pi}{2}\right)2^{16} + \left(\frac{\pi}{2}\right)^2 2^{16} + \left(\frac{\pi}{2}\right)^3 2^{16} + \left(\frac{\pi}{2}\right)^4 2^8\right) \leq 2^{79.5} \cdot 2^3 \cdot 2^{19} \leq 2^{101.5} \ .$$

We use $2^{78.5}$ qubits with QRAQM access to store the pairs.

The previous best quantum attacks on 7-round AES-192 and AES-256 are the quantum Square attacks of [10], with respectively $2^{102.73}$ and $2^{106.73}$ reversible S-Boxes. So far our attack is comparable.

Quantum Pair Filtering. Following the discussion in Section 4.4, we can use quantum searches to speed up the construction of successive tables. The table $\mathcal{T}_1$, which is of expected size $2^{62.5}$, is constructed in time $4\left(\frac{\pi}{2}\right)\sqrt{\sigma_1}|\mathcal{T}_1|$ instead of $|\mathcal{T}_1|$. We do likewise for $\mathcal{T}_3$. Finally, when sieving $\mathcal{T}_3$ with respect to a guess of $k_1[0,10,2,8]$, we can use a quantum search instead of going through the table. This modifies the time complexity of Equation 17 into:

$$\left(\frac{\pi}{2}\right)\sqrt{|K_1|}\Bigg(4N\left(\frac{\pi}{2}\right)\sqrt{\sigma_1}+\left(\frac{\pi}{2}\right)\sqrt{|K_2|}\Bigg(4N\sigma_1\left(\frac{\pi}{2}\right)\sqrt{\sigma_2}+$$
$$\left(\frac{\pi}{2}\right)\sqrt{|K_3|}\Bigg(4N\sigma_1\sigma_2\left(\frac{\pi}{2}\right)\sqrt{\sigma_3}+\left(\frac{\pi}{2}\right)\sqrt{|K_4|}\left(\left(\frac{\pi}{2}\right)\sqrt{2N\sigma_1\sigma_2\sigma_3}\right)\Bigg)\Bigg)\Bigg)\max_i(t+t_i)$$
$$=\max_i(t+t_i)\Bigg(\left(\frac{\pi}{2}\right)^5\sqrt{|K_1||K_2||K_3||K_4|}\sqrt{2N\sigma_1\sigma_2\sigma_3}+$$
$$4N\Bigg(\left(\frac{\pi}{2}\right)^2\sqrt{|K_1|\sigma_1}+\left(\frac{\pi}{2}\right)^3\sqrt{|K_1||K_2|}\sigma_1\sqrt{\sigma_2}+\left(\frac{\pi}{2}\right)^4\sqrt{|K_1||K_2||K_3|}\sigma_1\sigma_2\sqrt{\sigma_3}\Bigg)\Bigg)$$

We can now replace with our numerical values, obtaining:

$$8\cdot\left(\left(\frac{\pi}{2}\right)^5 2^{128/2}2^{23.5/2}+2^{80.5}\left(\left(\frac{\pi}{2}\right)^2 2^8+\left(\frac{\pi}{2}\right)^3 2^8+\left(\frac{\pi}{2}\right)^4 2^8\right)\right)\leq 2^{95.2}\ , \tag{18}$$

where the dominant term is the computation of the table $\mathcal{T}_3$. The memory complexity stays the same as before.

7 Conclusion

In this paper we provided a quantized version of impossible differential attacks. Our framework, which includes a generic complexity analysis, could be a good tool for analyzing the post-quantum security margin of symmetric primitives, as we have shown with SKINNY-128-256. Regarding AES, we have only been able to provide limited improvements of the 7-round attacks of [10]. Currently, our best attack is limited by the generation of pairs. While the filtering step benefits significantly from quantum search, potentially up to a complete quadratic speedup on its complexity, the first step seems a relatively more difficult problem in the quantum setting, as the limited birthday problem reaches a less than quadratic speedup. Consequently, as observed in [10] with other attack families, quantum impossible differential attacks need to be optimized differently from the classical ones. One could for example try to reduce the data used as much as possible, and in particular, to make the pair generation step completely classical (thereby falling in the Q1 setting).

Acknowledgments. A.S. is supported by ERC-ADG-ALGSTRONGCRYPTO (project 740972).

References

1. Ambainis, A.: Quantum walk algorithm for element distinctness. SIAM J. Comput. **37**(1), 210–239 (2007). https://doi.org/10.1137/S0097539705447311
2. Batcher, K.E.: Sorting networks and their applications. In: AFIPS Spring Joint Computing Conference. AFIPS Conference Proceedings, vol. 32, pp. 307–314. Thomson Book Company, Washington D.C. (1968)
3. Beierle, C., Jean, J., Kölbl, S., Leander, G., Moradi, A., Peyrin, T., Sasaki, Y., Sasdrich, P., Sim, S.M.: The SKINNY family of block ciphers and its low-latency variant MANTIS. In: CRYPTO (2). Lecture Notes in Computer Science, vol. 9815, pp. 123–153. Springer (2016)
4. Bennett, C.H., Bernstein, E., Brassard, G., Vazirani, U.V.: Strengths and weaknesses of quantum computing. SIAM J. Comput. **26**(5), 1510–1523 (1997). https://doi.org/10.1137/S0097539796300933
5. Biham, E., Biryukov, A., Shamir, A.: Cryptanalysis of skipjack reduced to 31 rounds using impossible differentials. J. Cryptol. **18**(4), 291–311 (2005). https://doi.org/10.1007/s00145-005-0129-3
6. Bonnetain, X., Chailloux, A., Schrottenloher, A., Shen, Y.: Finding many collisions via reusable quantum walks. Cryptology ePrint Archive, Paper 2022/676 (2022), https://eprint.iacr.org/2022/676
7. Bonnetain, X., Hosoyamada, A., Naya-Plasencia, M., Sasaki, Y., Schrottenloher, A.: Quantum attacks without superposition queries: The offline simon's algorithm. In: ASIACRYPT (1). Lecture Notes in Computer Science, vol. 11921, pp. 552–583. Springer (2019)
8. Bonnetain, X., Jaques, S.: Quantum period finding against symmetric primitives in practice. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2022**(1), 1–27 (2022)
9. Bonnetain, X., Naya-Plasencia, M., Schrottenloher, A.: On quantum slide attacks. In: SAC. Lecture Notes in Computer Science, vol. 11959, pp. 492–519. Springer (2019)
10. Bonnetain, X., Naya-Plasencia, M., Schrottenloher, A.: Quantum security analysis of AES. IACR Trans. Symmetric Cryptol. **2019**(2), 55–93 (2019). https://doi.org/10.13154/tosc.v2019.i2.55-93
11. Boura, C., Lallemand, V., Naya-Plasencia, M., Suder, V.: Making the impossible possible. J. Cryptol. **31**(1), 101–133 (2018). https://doi.org/10.1007/s00145-016-9251-7
12. Boura, C., Naya-Plasencia, M., Suder, V.: Scrutinizing and improving impossible differential attacks: Applications to clefia, camellia, lblock and simon. In: ASIACRYPT (1). Lecture Notes in Computer Science, vol. 8873, pp. 179–199. Springer (2014)
13. Brassard, G., Hoyer, P., Mosca, M., Tapp, A.: Quantum amplitude amplification and estimation. Contemporary Mathematics **305**, 53–74 (2002)
14. Childs, A.M., Eisenberg, J.M.: Quantum algorithms for subset finding. Quantum Inf. Comput. **5**(7), 593–604 (2005)
15. Daemen, J., Knudsen, L.R., Rijmen, V.: The block cipher square. In: FSE. Lecture Notes in Computer Science, vol. 1267, pp. 149–165. Springer (1997)

16. Daemen, J., Rijmen, V.: The Design of Rijndael: AES - The Advanced Encryption Standard. Information Security and Cryptography, Springer (2002). https://doi.org/10.1007/978-3-662-04722-4
17. Dawson, C.M., Nielsen, M.A.: The solovay-kitaev algorithm. Quantum Inf. Comput. **6**(1), 81–95 (2006)
18. Demirci, H., Selçuk, A.A.: A meet-in-the-middle attack on 8-round AES. In: FSE. Lecture Notes in Computer Science, vol. 5086, pp. 116–126. Springer (2008)
19. Derbez, P., Fouque, P., Jean, J.: Improved key recovery attacks on reduced-round AES in the single-key setting. In: EUROCRYPT. Lecture Notes in Computer Science, vol. 7881, pp. 371–387. Springer (2013)
20. Ferguson, N., Kelsey, J., Lucks, S., Schneier, B., Stay, M., Wagner, D.A., Whiting, D.: Improved cryptanalysis of rijndael. In: FSE. Lecture Notes in Computer Science, vol. 1978, pp. 213–230. Springer (2000)
21. Gilbert, H., Peyrin, T.: Super-sbox cryptanalysis: Improved attacks for aes-like permutations. In: FSE. Lecture Notes in Computer Science, vol. 6147, pp. 365–383. Springer (2010)
22. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: STOC. pp. 212–219. ACM (1996)
23. Hosoyamada, A., Naya-Plasencia, M., Sasaki, Y.: Improved attacks on sliscp permutation and tight bound of limited birthday distinguishers. IACR Trans. Symmetric Cryptol. **2020**(4), 147–172 (2020)
24. Kaplan, M., Leurent, G., Leverrier, A., Naya-Plasencia, M.: Breaking symmetric cryptosystems using quantum period finding. In: CRYPTO (2). Lecture Notes in Computer Science, vol. 9815, pp. 207–237. Springer (2016)
25. Kaplan, M., Leurent, G., Leverrier, A., Naya-Plasencia, M.: Quantum differential and linear cryptanalysis. IACR Trans. Symmetric Cryptol. **2016**(1), 71–94 (2016). https://doi.org/10.13154/tosc.v2016.i1.71-94
26. Kliuchnikov, V., Maslov, D., Mosca, M.: Fast and efficient exact synthesis of single-qubit unitaries generated by clifford and T gates. Quantum Inf. Comput. **13**(7-8), 607–630 (2013)
27. Knudsen, L.: Deal-a 128-bit block cipher. complexity **258**(2), 216 (1998)
28. Kuperberg, G.: Another subexponential-time quantum algorithm for the dihedral hidden subgroup problem. In: TQC. LIPIcs, vol. 22, pp. 20–34. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2013)
29. Liu, Q., Zhandry, M.: On finding quantum multi-collisions. In: EUROCRYPT (3). Lecture Notes in Computer Science, vol. 11478, pp. 189–218. Springer (2019)
30. Lu, J., Kim, J., Keller, N., Dunkelman, O.: Improving the efficiency of impossible differential cryptanalysis of reduced camellia and MISTY1. In: CT-RSA. Lecture Notes in Computer Science, vol. 4964, pp. 370–386. Springer (2008)
31. Mala, H., Dakhilalian, M., Rijmen, V., Modarres-Hashemi, M.: Improved impossible differential cryptanalysis of 7-round AES-128. In: INDOCRYPT. Lecture Notes in Computer Science, vol. 6498, pp. 282–291. Springer (2010)
32. Nielsen, M.A., Chuang, I.: Quantum computation and quantum information (2002)
33. Sadeghi, S., Mohammadi, T., Bagheri, N.: Cryptanalysis of reduced round SKINNY block cipher. IACR Trans. Symmetric Cryptol. **2018**(3), 124–162 (2018). https://doi.org/10.13154/tosc.v2018.i3.124-162
34. Xie, H., Yang, L.: Quantum impossible differential and truncated differential cryptanalysis. CoRR **abs/1712.06997** (2017)
35. Xie, H., Yang, L.: Using bernstein-vazirani algorithm to attack block ciphers. Des. Codes Cryptogr. **87**(5), 1161–1182 (2019). https://doi.org/10.1007/s10623-018-0510-5