Quantum impossible differential attack. Applications to CLEFIA, AES and SKINNY

Nicolas David

Quantum impossible differential attack. Applications to CLEFIA, AES and SKINNY

Nicolas David , Maria Naya-Plasencia, INRIA Paris

September 2, 2019

The general context

Cryptography is a computer discipline that aims to protect messages through encryption systems. In symmetric cryptography, a secret parameter, called a key, is used both to encrypt and to decrypt messages. The security provided by a symmetric encryption system is evaluated using cryptanalysis techniques which aim, for example, to find the secret key.

Quantum computer arrival could impact the cryptographic field. Indeed, in 1994, Shor exhibited that quantum computers could be used to improve assymetric cryptanalysis [17]. With the recent breakthrough in quantum computer, the security of cryptographic primitives against quantum adversary can not be taken as guaranteed. The NIST launched a competition for new primitives that are safe even against adversaries that has access to a quantum computer. To estimate the quantum security of a cryptographic scheme, it is necessary to perform its quantum cryptanalysis. Quantum cryptanalysis techniques sometimes are quantum adaptation of classical cryptanalysis techniques. This transformation is called *quantizing*.

Let's note that an attack is valid if and only if it is more efficient than the naive attack. In the classical setting, the naive attack is the generic exhaustive search, in the quantum setting, it is the Grover search algorithm [14].

The research problem

The report aims to study impossible differential attacks introduced independantly by Knudsen [13] and Biham et al. [4], and then derive a valid quantizing. The authors in [5], that designed a quantum attack on AES, pointed out that impossible differential technique has not been yet quantized and that its supposed efficiency was not clear.

Your contribution

In order to build a valid attack, we will split the classic attack into different steps. Each of these steps will then be modified to obtain the most efficient quantizing. A study of this effectiveness will be given through an accurate analysis of the complexity.

Arguments supporting its validity

Various examples of applications such as CLEFIA-128 [16], AES-128 and ForkSkinny-128-256 [10] will be given to understand how far the validity of quantum impossible differential attacks extends. We will also identify the factors that break the validity of the attack.

Summary and future work

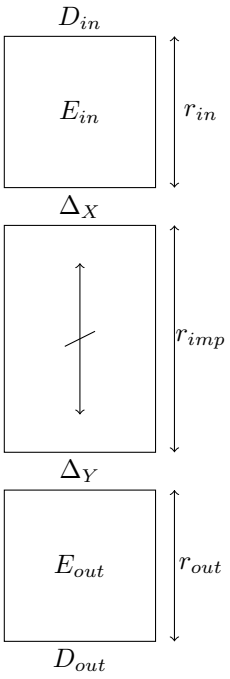
The applications of impossible quantum attacks have shown that some schemes are vulnerable to quantum computer and that security with respect to impossible differential attacks is to be studied before claiming any quantum security. However, an overhead in time complexity induced by the quantizing is not negligible. Indeed, some primitives vulnerable to classical impossible attacks lose this vulnerability when transposition in the quantum world. This overhead seems to be related to the memory. Understanding how to quantify the memory operations of the classical world will reduce this extra cost and therefore provide better attacks.

1 Preliminaries

1.1 Classical impossible differential attacks

Impossible differential attacks were independantly introduced by Knudsen [13] and Biham et al. [4]. The goal of cryptanalysis technique is to recover the secret key k^*, belonging to some key set K, used in the encryption scheme. This is done by discarding all the wrong keys with the help of a pair of plaintexts that leads to an impossible pattern under its encryption with the wrong key.

Building an impossible differential attack of an n-bit block cipher over r rounds mainly consists of two steps. In the first step the goal is to find a maximal length impossible differential, i.e. we want to find a pair of differentials Δ_X, Δ_Y together with an integer r_{imp} (that we look forward to maximize) such that the probability of a difference Δ_X propagates to a difference Δ_Y after r_{imp} rounds is 0. In this step, we also want to find two integers r_{in}, r_{out} such that $r_{in} + r_D + r_{out} = r$. We denote by impossible pattern the quantity $(\Delta_X, \Delta_Y, r_{imp}, r_{in}, r_{out})$. The second step, called the seiving phase, consists on finding two sets $D_{in} \subset \{0,1\}^n$ and $D_{out} \subset \{0,1\}^n$. Let's denote by E the encryption oracle and E_{in} (resp. E_{out}) the partial encryption function between round 0 and r_{in} (resp $r_{in} + r_{imp}$ and r).



If we are given a pair of plaintexts x, y with difference in D_{in} and their corresponding ciphertexts $E(x), E(y)$ with difference belonging to D_{out}, due to the impossible pattern, we can discard any key k that verifies:

$$\big(E_{in}(k)(x) \oplus E_{in}(k)(y) = \Delta_X\big) \wedge \big(E_{out}^{-1}(k)(E(x)) \oplus E_{out}^{-1}(k)(E(y)) = \Delta_Y\big)$$

The seiving phase goal is to discard as many keys as possible using many plaintexts pairs.

Remark. *If we denote by $K_{in} \subset K$ the key bits involved in E_{in} and $K_{out} \subset K$ the key bits involved in E_{out}, we can remark that the key bits involved in the impossible key distinguisher are $K_{in} \cup K_{out}$. Therefore our analysis is done on the subkeyspace $K_{in} \cup K_{out}$.*

The seiving phase that we will describe is inspired from [7]. It decompose iteself in a two steps procedure:

- **Pair Generation**. In this part, we focus in solving the following problem:

Data:	$N \in \mathbb{N}$, $E : \{0,1\}^n \to \{0,1\}^n$
	$D_{in} \subset \{0,1\}^n$, $D_{out} \subset \{0,1\}^n$
Question:	Find N pairs $(x, y) \in \{0,1\}^{2n}$ such that
	$x \oplus y \in D_{in}$ and $E(x) \oplus E(y) \in D_{out}$

  A good analysis of this problem in the classical setting is given in [6] with an efficient algorithm for solving it. At the end of this step, we will dispose of a set of N differential pairs. We will denote by $Pairs$ the set obtained.

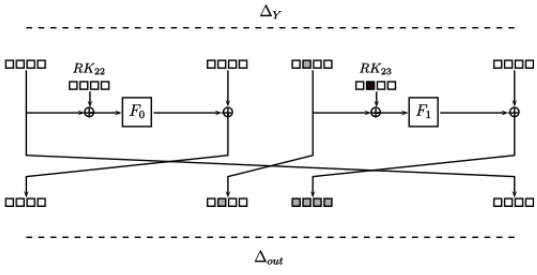
- **Pair Filtering**. This step follows the Pair Generation, hence we dispose of a set of differential pairs. The goal of this step is to split the set of subkey $K_{in} \cup K_{out}$ into two sets : one that countains all the subkeys that have been invalidated by some pair of $Pairs$ and the another that countains the other subkey. A very interresting optimization in this step is called early abort technique. It is described in [6]. For a given key $k \in K_{in} \cup K_{out}$, the goal of this optimization is to filter these pairs only to keep the pairs that will invalidate k.

  To explain how the filtering is done, we will introduced tests functions. Let's assume that

  - There exists sets $K_1, \cdots, K_l$ that decompose $K_{in} \cup K_{out} = K_1 \otimes K_2 \otimes \cdots \otimes K_l$ These sets usually reprensent some subyte of the subkeyspace.
  - There exists sets $P_1, \cdots, P_l$ such that $Pairs \subset P_1 \otimes P_2 \otimes \cdots \otimes P_l$

  Together with this decomposition, we dispose of l test functions $T_i : K_i \otimes P_i \to \{0,1\}$.

Example. Let's give an example of a test function. In this example based on CLEFIA-128 wich specification is given in section 3.1.1. We are looking at round 11 and 12, we note $P_0^{12}, P_1^{12}, P_2^{12}, P_3^{12}$ 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 holds:



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

Then, the test function to propagate from round 12 to round 11 is $T : P_1^{12}, P_2^{12}, RK_{23} \mapsto F_1(P_1^{12} \oplus RK_{23}) = P_2^{12}$

The early abort technique aim to recover the set of pairs that satisfy all the tests functions that we note:

$$Pairs(k) = \{p \in Pairs \mid \forall i \; T_i(k_i, p_i) = 1\}$$

We will assume that the tests function are build such that the following property holds: $Pairs(k) = \emptyset$ if and only if there exists no pairs in $Pairs$ that invalidates k. This assumption is legitimate since it holds in practice. Then a key will be discarded if $Pairs(k) \neq \emptyset$.

In the end, we have described a probabilistic procedure that discards a subkey k (different from the one used in the encryption) with some probability P. We denote by candidate key all the keys that have not been discarded. To recover the remaining key bits, i.e $K \setminus K_{in} \cup K_{out}$, we will perform an exhaustive search on these bits and the candidate keys. Let's remark that if there is less key candidates than $|K_{in} \cup K_{out}|$ then the exhaustive search on the key candidate and the remaining key bits will be more efficient than the exhaustive search on the whole key space. If the procedure discards a certain amount of keys, an attack will be valid if and only if its seiving procedure is more efficient than the naive exhaustive search. In the following section, we will discuss about the complexity analysis of this procedure.

1.2 Complexity analysis

In this section, we will provide both the data complexity and the time complexity of the impossible attack described in the previous section.

1.2.1 Data complexity

In impossible differential attacks, the data complexity is completly determined by the cost of building the set $Pairs$. Hence, it is enough to estimate the size of $Pairs : N$. Considering a differential pair with input difference in D_{in} and output difference in D_{out}, we denote by c_{in} the nuber 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 condition to reach a difference of Δ_Y at round $r_{in} + r_D$. With these notation, the probability P of discarding a bad key can be written

$$P = (1 - 2^{-(c_{in}+c_{out})})^N$$

This equation allows us to write

$$N = O(ln(1/P) \cdot 2^{c_{in}+c_{out}})$$

For instance, if we want to get only one subkey in the end, it is enough to ask for arround $P = \frac{1}{|K_{in} \cup K_{out}}$ therefore $N = O(2^{c_{in}+c_{out}} \cdot \log(\mid K_{in} \cup K_{out}) \mid)$.

1.2.2 Time complexity

To study the time complexity of the attack, we need to emphasize that the search of the impossible differential pattern is not involved here. We suppose as it is common in the

litterature that a good impossible differential pattern is already known. The complexity of the attack is mainly determined by the seiving phase. We detail here the complexity of all its steps.

- **Pair Generation**. To estimate the complexity of this part, it is necessary to introduce some notation for the cardinality of Δ_{in} and Δ_{out}. For the following, we will note $2^{\Delta_{in}} = |D_{in}|$ and $2^{\Delta_{out}} = |D_{out}|$.
  The complexity of the Pair Generation problem was studied in [11] for the special case where $N = 1$. The complexity for finding *one* such pair with acess to encryption oracle is given by:
  $$C_1 = \max\left(\min_{\Delta\in\{\Delta_{in},\Delta_{out}\}} \sqrt{2^{n+1-\Delta}}, 2^{n+1-(\Delta_{in}+\Delta_{out})}\right)$$
  Then to build N pairs, we could use the previous technique N times and get a complexity of $N \cdot C_1$. However, as shown in [7], with acess to encryption oracle, it is possible to reduce this complexity to:
  $$C_N = \max\left(\min_{\Delta\in\{\Delta_{in},\Delta_{out}\}} \sqrt{N2^{n+1-\Delta}}, N2^{n+1-(\Delta_{in}+\Delta_{out})}\right)$$

- **Pair Filtering**.

  Lemma 1. *Given a set of differential pairs, checking if $Pairs(k)$ is empty for all key can be done in time*
  $$|K_1|\ (|P_1| + |K_2|\ (|P_2| + |K_3|(|P_3| + \cdots |K_l||P_l|\ )\ )$$
  The procedure is described in [6], it essentially consists of first guessing the key partial key $k_1 \in K_1$ and then by using an exhaustive search on P_1 to find all the pairs that satisfy T_1. Then we gess $k_2 \in K_2$ and repeat the process until we make the exhaustive search on P_l. At this point, the remaining pairs are the elements of $Pairs(k)$, hence it is enough to check if there is any pair left.

  Remark. *This time complexity can be reduced if an efficient data structure is used. For instance, using lookup table could help to improve the complexity.*

In the end, the total complexity is the sum of the complexity of the two steps:
$$|K_1|\ (|P_1| + |K_2|\ (|P_2| + |K_3|(|P_3| + \cdots |K_l||P_l|\ )\ ) + C_N + P \cdot 2^{|K|}$$
The quantity $P \cdot 2^{|K|}$ corresponds to the exhaustive search that is perform at the end to recover all the bits of the secret key.

1.3 Quantum preliminaries

In this section, we will introduce some quantum tools and describe the most important quantum adversary models that are usually considered. All the normalization terms are omitted for a better readability.

1.3.1 Quantum computing

We will remind here some basis of quantum computing, a better introduction to the subject is given in [9].

Qubits. In a classical computer, the unitary units of computation are bits and their value can either be 0 or 1. In a quantum computer, bits are replaced by qubits that can be a superposition of two states $|0\rangle$ and $|1\rangle$. Therefore, a qubit is represented by a vector of a Hilbert space:

$$\alpha\,|0\rangle + \beta\,|1\rangle$$

where $\alpha, \beta \in \mathbb{C}$ and $|\alpha|^2 + |\beta|^2 = 1$. α and β are called amplitude, the quantity $|\alpha|^2$ and $|\beta|^2$ corresponds to the probability to retrieve $|0\rangle$ or $|1\rangle$ when the qubit is measured. Hence, $|0\rangle$ and $|1\rangle$ are the basis of this Hilbert space.

Entanglement. Combining two qubits, vectors of $\mathcal{H}$, creates a system belonging to $\mathcal{H}^{\otimes^2}$. Indeed, any system of two qubits can be describe by a superposition over the following states $|00\rangle$,$|10\rangle$,$|01\rangle$ and $|11\rangle$. It is important to remark that there is a strict difference between $\mathcal{H}^2$ and $\mathcal{H}^{\otimes^2}$. Therefore a system of two qubits can't always be described by a pair of qubits.

Quantum algorithm. In this standard model of quantum computing, a quantum algorithm is a sequence of elementary operations: quantum gates. A good example of quantum gate is the Hadamard gate that performs the following transformation:

$$H^{\otimes^n} : |x\rangle \mapsto \frac{1}{\sqrt{2^n}} \sum_{y\in\{0,1\}^n} (-1)^{x\cdot y}\,|y\rangle$$

Each of the gate can be represented by a unitary and reversible operation, therefore each algorithm is a unitary and reversible operation. A consequence of this property is that there exists no algorithm that copies a qubit, since the copy operation is not unitary. Therefore, to duplicate a qubit, it is necessary to repeat the process that creates it.

The quantum computation ends with a measurement that will make the quantum state collapse to an element of the basis.

1.3.2 Models

There exist several models that aim at modelizing quantum adversary. With the same terminology used in [12], we will introduce two of them that are the most often considerd.

- $Q1$ **Model**. The adversary is allowed to make quantum computations but she can only make classical queries to the encryption and decryption oracles.
- $Q2$ **Model**. The adversary can make quantum superposition queries to the encryption oracle. This model refers to IND-qCPA ("quantum chosen-plaintext") in [8, **?**]

Our quantum version of impossible differential attack will be placed in $Q2$. This is necessary if we want to give an efficient quantum optimization of the Pair Generation problem.

For the data model, we will allow the use of a quantum RAM (qRAM). The qRAM model used in the following is called classical RAM with quantum acess. It consists of storing classical

information that we can query in superposition. The following quantum gate represents the qRAM

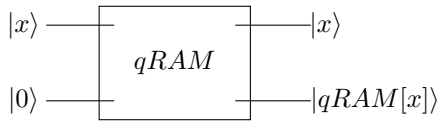


Figure 1: qRAM

1.3.3 Quantum Tools

In the following, we will introduce some usefull tools from quantum information theory.

1. **Grover search**. Grover search algorithm introduced in [14], is an algorithm that solves the followig problem:

Data:	N elements including t marked elements
Question:	Find a marked element

   A precise description of this algorithm is given in [9]. The complexity associated to this problem is

   $$O\left(\sqrt{\frac{N}{t}}\left(S+C\right)\right)$$

   where S is the complexity of building the quantum superposition on the N elements and C is the cost of checking if an element is marked. In the classical case the complexity is $\frac{N}{t} \cdot C$.
   In the following, for any boolean function $T : X \to \{0,1\}$, we denote by Grover on T the application of Grover algorithm on the set X. The marked elements corresponds to all $x \in X$ such that $T(x) = 1$.

2. **Amplitude amplification**.In [1], the following problem is introduced:

Data:	$A : \emptyset \to Y$ a random quantum algorithm performing no measurement
	$\chi : Y \to \{0,1\}$ a Boolean function
	p the probability that $\chi(A()) = 1$
Question:	Find a element $y \in Y$ such that $\chi(y) = 1$

   In the classical version of this problem, $1/p$ computation of $A()$ are recquired in average. Quantummly, it is possible to solve this problem with only $\sqrt{1/p}$ computations of $A()$ as explaind in [9].

3. **Collision finding : Ambainis algorithm**. The Collision finding problem was introduced by Andris Ambainis in [1]. It consists of

Data:	$H : \{0,1\}^n \to \{0,1\}^n$ a random function
Question:	Find *two* elements $x, y \in \{0,1\}^n$ such that $H(x) = H(y)$.

The time complexity obtain by the quantum algorithm in [1] is $O(2^{n/3})$. Due to the birthday paradox, the complexity for solving this problem in the classical setting is $O(2^{n/2})$.

4. **Quantum walk**. The last problem that we will need is the quantum walk, introduced in [1]. The problem that we're looking to solve is

Data:	A graph G of N vertices including an ϵ proportion of marked vertices
	An oracle deciding if a vertex is marked in time C
Question:	Find *one* marked vertice

The idea of the algorithm is to quantify the classical random walk algorithm:

Start at some specific vertex y on the graph.
Repeat the following process a given number of times: Query the oracle to check if y is marked, if not, choose one of its neighbors at random and set y to this value.

With the approach given in [9], the classical time complexity of the procedure is

$$S + \frac{1}{\epsilon}\,(C + \frac{1}{\delta}U\,)$$

and the quantum time complexity is

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

where S is the complexity of building the quantum superposition on all the vertices of G, ϵ is the fraction of marked states, C is the cost of checking if a vertex is marked, δ is the spectral gap[1], G and U is the cost of updating a vertex to its neighbor.

[1] if we note A the normalized adjacency matrix of G and $\lambda_1 = 1 > \lambda_2 > \cdots > \lambda_N$ its eigenvalue, the spectral gap of G is the quantity $1 - \max_{i\geq 2} \lambda_i$

2 Quantum Impossible Differential

In this section, I will present the quantized version of impossible differential attack that I developped during my internship. For this, we will provide a quantized version of the Pair Generation, the Pair filtering phases.

2.1 Pair Generation : Quantum Limited Birthday Problem

In this section we come back to the Pair Generation problem.

$$\begin{array}{rl} \text{Data:} & N \in \mathbb{N},\ E : \{0,1\}^n \to \{0,1\}^n \\ & D_{in} \subset \{0,1\}^n\ ,\ D_{out} \subset \{0,1\}^n \\ \text{Question:} & \text{Find } N \text{ pairs } (x,y) \in \{0,1\}^{2n} \text{ such that} \\ & x \oplus y \in D_{in} \text{ and } E(x) \oplus E(y) \in D_{out} \end{array}$$

Let's first note we consider to have acces to both the encryption and the decryption oracles. Therefore it is possible to swap the roles of D_{in} and D_{out} to optimize the final complexity.

In [12] an algorithm is given to solve this problem for $N = 1$. The complexity reached is

$$Q_1 = \max\left(2^{(n-\Delta_{out})/3}, 2^{(n-\Delta_{out})/2-\Delta_{in}/3}\right)$$

Like in the classical setting, we could pay $N \cdot Q1$ to recover N pairs but there exits procedures that reduce this time complexity. In the following, such a procedure will be described.

Let's first remark that solving the Collision Finding problem provides a solution to Pair Generation. Let's pick any $y \in \{0,1\}^n$, we can define H_y:

$$\begin{array}{rccc} \mathrm{H}_y : & \mathrm{D}_{in} & \to & \{\ 0{,}1\ \}^{n-\Delta_{out}} \\ & \mathrm{x} & \to & \mathrm{E(y)} \oplus E(y \oplus x) \mod D_{out} \end{array}$$

here $\mod D_{out}$ stands for the relation $x = y \mod D_{out} \Leftrightarrow x \oplus y \in D_{out}$. In practice D_{in} and D_{out} are correspond to subits of the plaintext and ciphertext, then $\mod D_{out}$ can be checked by a projection on the complementary set of D_{out} in $\{0,1\}^n$.

We remark that finding a collision for H_y corresponds exactly to finding a good differential pair for E. We will denote by structure any function H_y that follows the pattern previously described.

Lemma 2. *Let $H : \{0,1\}^n \to \{0,1\}^m$ be a function such that H has 2^{2c} collision pairs. Let $c' < 2c$, there exists a quantum algorithm which outputs $2^{c'}$ such collisions in time and memory $2^{\frac{2}{3}(n-c+c')}$.*

Proof. We will give a quantum algorithm that achieves the complexity goal. It consists of a quantum walk.
First, let $r \in \mathbb{N}$ be some integer such that $2^r << 2^n$. We will walk on the graph $G = <V, E>$ with

1. $V = \{1, \cdots, 2^n\}^{2^r}$, i.e 2^r-tuples of integers of $\{1, \cdots, 2^n\}$.

2. $E = \{(x, y) \in E^2 \mid \exists! i \; x_i \neq y_i\}$

We define the set of marked state as follows $M = \{x \in V \mid \exists i, j \; x_i \neq x_j \wedge H(x_i) = H(x_j)\}$, hence a vertex is marked whenever it countains at least *one* non trivial collision. Our procedure splits in three parts:

1. **Setup**. First, we construct the initial uniform superposition of all vertices. We will build a superposition in each of the 2^r registers (this represents the quantum memory used in this procedure).

2. **Quantum walk**. Then, we will perform a walk on the graph[2] until we reach a superposition of marked states. Because there is 2^{2c} collision pairs, the fraction of marked vertices is given by
$$\epsilon = \frac{2^{c+r}2^{c+r-1}}{2^n 2^{n-1}} \approx (2^{c+r-n})^2$$
Therefore the quantum walk needs to run 2^{n-c-r} steps. Furthermore, the spectral gap is estimate by $\delta = \frac{2^n}{2^r(2^n-2^r)} \approx 2^{-r}$.

3. **Measurement**. Once we got the quantum state $|\psi\rangle$ corresponding to the superposition over the marked states, we compute the value of each element through H.
$$\begin{aligned} |\psi\rangle &= \sum_{(x_1,x_2,\cdots,x_{2^r})\in M} |(x_1, x_2, \cdots, x_{2^r})\rangle \\ &\to \sum_{(x_1,x_2,\cdots,x_{2^r})\in M} |(x_1, x_2, \cdots, x_{2^r})\rangle \, |(H(x_1), H(x_2), \cdots, H(x_{2^r}))\rangle \end{aligned}$$
At this point, we perform a sort on the H side to put the collision in the two first register. Then, we will measure these two registers to get the first collision. Each of the $2^r - 2$ registers left collapse to the uniform superposition over the 2^n elements. Therefore by adding two registers countaining the uniform superposition, we produce the same state that we had at the beginning of step2. This way, we can repeat this process $\widetilde{O}(2^{c'})$ times to collect the $2^{c'}$ collisions.

This technique allow us to avoid the setup cost when we loop back to step2 $2^{c'}$ times. In the end the complexity of this algorithm is
$$2^r + 2^{c'} \cdot 2^{n-c-(r/2)}$$
By taking $r = \frac{2}{3}(n - c + c')$, we obtain a complexity of $2^{\frac{2}{3}(n-c+c')}$. □

Remark. *The exact complexity of the procedure involves a polynomial factor in n (due to the sort) that we will neglect.*

Remark. *Under the assumption that $H : \{0,1\}^n \to \{0,1\}^m$ follows the random oracle model, there is arround 2^{2n-m} collisions.*

For every structure H_y, we have described a way to collect collisions from which we can derive pairs. Depending on the value of N, Δ_{in} and Δ_{out}, the number of structures that we will need to consider in order to recover N pairs will be different. Indeed, for each structure we expect $2^{2\Delta_{in}-(n-\Delta_{out})}$ collisions.

[2]This graph is in fact in our case really close to the Johnson graph $J(2^r, 2^n)$

1. First, we will consider $N < \frac{2^{2\Delta_{in}}}{2^{n-\Delta_{out}}}$.

   Property 1 (One Structure)**.** *If $N < \frac{2^{2\Delta_{in}}}{2^{n-\Delta_{out}}}$ then one can recover N differential pairs in time $N^{2/3}2^{\frac{n-\Delta_{out}}{3}}$.*

   Proof. If E is IND-CPA, we can suppose that H_y is random for any $y \in \{0,1\}^n$. Then we can assume that it has $\frac{2^{2\Delta_{in}}}{2^{n-\Delta_{out}}}$ collisions (hence $c = \Delta_{in} - \frac{n-\Delta_{out}}{2}$). Then, by applying Lemma 1, we can find N collisions for H in time:

$$\begin{aligned} 2^{\frac{2}{3}(\Delta_{in}-(\Delta_{in}-\frac{n-\Delta_{out}}{2})+\log_2(N))} &= 2^{\frac{n-\Delta_{out}}{3}+\frac{2}{3}\log_2(N)} \\ &= N^{\frac{2}{3}}2^{\frac{n-\Delta_{out}}{3}} \end{aligned}$$

   □

2. Now, let's consider another scenario : $\frac{2^{2\Delta_{in}}}{2^{n-\Delta_{out}}} < 1$. This part follows the ideas of [12]. This corresponds to the case in which there is less than one pair per structure on average.

   Lemma 3. *If $\frac{2^{2\Delta_{in}}}{2^{n-\Delta_{out}}} < 1$, one can retrieve a differential pair in quantum time:*

$$2^{\frac{n-\Delta_{out}}{2}-\frac{\Delta_{in}}{3}}$$

   and quantum memory:

$$2^{\frac{2}{3}\Delta_{in}}$$

   Proof. Let's consider $y \in \{0,1\}^n$. If we apply Ambainis' algorithm to H_y, we can find a Collision (if exists) in time and memory:

$$2^{\frac{2}{3}\Delta_{in}}$$

   Then because the probability of getting a structure in which there is a differential pair is $\frac{2^{2\Delta_{in}}}{2^{n-\Delta_{out}}}$, by applying an amplitude amplification algorithm, one can get a differential pair in time:

$$\sqrt{2^{n-\Delta_{out}-2\Delta_{in}}} \cdot 2^{\frac{2}{3}\Delta_{in}} = 2^{\frac{n-\Delta_{out}}{2}-\frac{\Delta_{in}}{3}}$$

   □

   Property 2 (Less than one pair)**.** *If $\frac{2^{2\Delta_{in}}}{2^{n-\Delta_{out}}} < 1$, one can retrieve N differential pairs in quantum time:*

$$N \cdot 2^{\frac{n-\Delta_{out}}{2}-\frac{\Delta_{in}}{3}}$$

   and quantum memory:

$$2^{\frac{2}{3}\Delta_{in}}$$

   Remark. *In this case, the final complexity corresponds to $N \cdot Q_1$.*

3. There is one last scenario to consider: $1 < \frac{2^{2\Delta_{in}}}{2^{n-\Delta_{out}}} < N$. Here, we will need to consider multiple structures that all have severals differential pairs (we expect $\frac{2^{2\Delta_{in}}}{2^{n-\Delta_{out}}}$ pairs per structure). For the following, we will need to define n_s:

$$n_s = \log_2(N) + n - 2\Delta_{in} - \Delta_{out}$$

This quantity represents the number of structures that we will need to consider to recover N differential pairs.

Property 3 (multiple structures)**.** *If $1 < \frac{2^{2\Delta_{in}}}{2^{n-\Delta_{out}}} < N$, one can retrieve N differential pairs in quantum time*

$$N2^{\frac{2}{3}(n-\Delta_{out}-\Delta_{in})}$$

Proof. We will apply the procedure describe in Property 1 2^{n_s} times. Each times, we are looking to collect $N' = \frac{2^{2\Delta_{in}}}{2^{n-\Delta_{out}}}$ collisions. The quantum time cost for the full procedure is

$$\begin{aligned} 2^{n_s} \cdot N'^{\frac{2}{3}} 2^{\frac{n-\Delta_{out}}{3}} &= 2^{n_s} \cdot 2^{\frac{4}{3}\Delta_{in} - \frac{n-\Delta_{out}}{3}} \\ &= N2^{\frac{2}{3}(n-\Delta_{in}-\Delta_{out})} \end{aligned}$$

□

Now that we have studied all the cases, we can consider the knowledge of both oracles and conclude:

Theorem 1. *Let $E : \{0,1\}^n \to \{0,1\}^n$ be a block cipher such that both E and E^{-1} can efficiently handle superposition queries. The Limited Birthday Problem can be solved in quantum time*

$$Q_N = \max\{\min_{\Delta \in \{\Delta_{in}, \Delta_{out}\}} N^{2/3} 2^{(n-\Delta)/3}, \min(N2^{\frac{2}{3}(n-\Delta_{out}-\Delta_{in})}, \min_{\Delta \in \{\Delta_{in}, \Delta_{out}\}} N2^{\frac{n}{2} - \frac{\Delta}{6} - \frac{\Delta_{in}+\Delta_{out}}{3}})\}$$

This theorem concludes the quantum version of Pair Generation. The set $Pairs$ will be stored in a classical RAM with quantum access (qRAM). The $qRAM$ will be set as a boolean function

$$qRAM[p] = (p \in Pairs) \in \{0,1\}$$

2.2 Quantum Pair Filtering

The notations of section 1.1 are used here, hence $Pairs = P_1 \otimes \cdots \otimes P_l$ and $K_{in} \cup K_{out} = K_1 \otimes \cdots \otimes K_l$. If classically, the goal would be to recover the set

$$Pairs(k) = \{(p_1, p_2, p_3 \cdots p_l) \in Pairs \mid \forall i \; T_i(k_i, p_i) = 1\}$$

Quantumly, we aim to recover the superposition of the elements of this set:

$$\sum_k |k\rangle \sum_{p \in Pairs(k)} |p\rangle$$

To lighten the notation, let's define

- $S_i(k_i) = \{p_i \in P_i \mid T_i(k_i, p_i) = 1\}$. In practice, the quantity $|S_i(k_i)|$ does not depends too much on the subkey k. In our analysis, we will assume that it is constant. We will denote by $\sigma_i = |S_i(k_i)|$.
- Likewise, we will suppose that the cardinality of the set $Pairs(k)$ (when nonempty) does not depend too much on k. We will denote by ϵ the quantity such that $2^\epsilon = |Pairs(k)|$.

Lemma 4. *One can retrieve the vector* $\sum_k |k\rangle \sum_{p \in Pairs(k)} |p\rangle$ *in quantum time*

$$O\left(\left(\sum_i \sqrt{\frac{|P_i|}{\sigma_i}}\right)\sqrt{\frac{\prod_i \sigma_i}{2^\epsilon}}\right)$$

Proof. We will now describe our original approach for quantum Pair Filtering. Computing $\sum_k |k\rangle \sum_{p \in Pairs(k)} |p\rangle$ can be done in a 2-steps procedure. For more clarity, we start by describing the procedure for one fixed subkey. The final procedure will consider the uniform superposition on these subkeys.

- **Collecting the conditions**. For each $i \in [1, l]$, let's pick $k_i \in K_i$. We can perform a Grover on the predicates $T_i(k_i) : P_i \mapsto \{0, 1\}$ to recover the superposition of the elements of S_i. The complexity of one Grover is $c_i(setup_i + check_i)$ where $c_i = O\left(\sqrt{\frac{|P_i|}{\sigma_i}}\right)$, $setup_i = O(1)$ and $check_i = O(1)$. Because all the tests T_i are independant, the complexity for l tests is

  $$O\left(\sum_{i=1}^{l} \sqrt{\frac{|P_i|}{\sigma_i}}\right)$$

  At this point we have computed for each $i \in [1, l]$ the quantum vector $|k_i\rangle \sum_{s \in S_i(k_i)} |s\rangle$ for some $k_i \in K_i$. By concatenating these l vectors, we obtain

  $$\psi_k = |k\rangle \sum_{s_1 \in S_1(k_1), \cdots, s_l \in S_l(k_l)} |s_1\rangle |s_2\rangle \cdots |s_l\rangle$$

- **Retrieving the pairs**. We have built ψ_k, a vector that carries the bit conditions for a pair to belong to $Pairs(k)$. To recover the superposition over $Pairs(k)$, it is enough to perform a Grover on the $qRAM$ (seen as a boolean function) to recover the pairs that belong to the solution space $\bigotimes_i S_i(k_i)$. This Grover search will set ψ_k as initial state. Hence, the total complexity is

  $$O\left(\left(\sum_i \sqrt{\frac{|P_i|}{\sigma_i}}\right)\sqrt{\frac{\prod_i \sigma_i}{2^\epsilon}}\right)$$

By replacing $|k\rangle$ by $\sum_k |k\rangle$ in the previous procedure, we will recover $\sum_k |k\rangle \sum_{p \in Pairs(k)} |p\rangle$. □

Remark. *Another way to proceed is to start with the superposition over the pairs and then filter with respect to the T_i. The Grover tests in this case won't be independant and the final complexity will be $O(\sqrt{\frac{\prod_i |P_i|}{2^\epsilon}}) = O(\sqrt{\frac{N}{2^\epsilon}})$ which is too much.*

Theorem 2. *One can retrieve a candidate key in quantum time*

$$O\left(\sqrt{|K_{in} \cup K_{out}|}\left(\sum_i \sqrt{\frac{|P_i|}{\sigma_i}}\right)\sqrt{\frac{\prod_i \sigma_i}{2^\epsilon}} + Q_N\right)$$

Proof. Let's describe the full procedure and its quantum complexity:

- **Pair Generation**. The complexity to store N pairs in the qRAM is Q_N. Its precise value is given in section 2.1.
- **Pair Filtering**. It is possible to generate $\sum_k |k\rangle \sum_{p \in Pairs(k)} |p\rangle$ in time $O\left(\left(\sum_i \sqrt{\frac{|P_i|}{\sigma_i}}\right)\sqrt{\frac{\prod_i \sigma_i}{2^\epsilon}}\right)$ as explained before. Now, for a given k, if $Pairs(k) = \emptyset$, the second register of $\sum_k |k\rangle \sum_{p \in Pairs(k)} |p\rangle$ will correspond to a pair that does not invalidate k. Therefore it is enought to check if the quantum vector $\sum_{p \in Pairs(k)} |p\rangle$ invalidates k to decide if k is a key candidate. Here is the final test:

  $$T : k, p \mapsto \text{p invalidates k}$$

  A Grover on T will retrun the superposition over the candidate keys. The complexity of this test is

  $$O\left(\sqrt{|K_{in} \cup K_{out}|}\left(\sum_i \sqrt{\frac{|P_i|}{\sigma_i}}\right)\sqrt{\frac{\prod_i \sigma_i}{2^\epsilon}}\right)$$

In the end, the total complexity is

$$O\left(\sqrt{|K_{in} \cup K_{out}|}\left(\sum_i \sqrt{\frac{|P_i|}{\sigma_i}}\right)\sqrt{\frac{\prod_i \sigma_i}{2^\epsilon}} + Q_N\right)$$

□

If N is build such that the only candidate key corresponds to the key used during encryption, by measuring the superposition over the candidate keys, we will recover k^*.

To retrieve the secret key, we perform a Grover search on the missing key bits. In the end, the complexity to recover the secret key is

$$O\left(\sqrt{|K_{in} \cup K_{out}|}\left(\sum_i \sqrt{\frac{|P_i|}{\sigma_i}}\right)\sqrt{\frac{\prod_i \sigma_i}{2^\epsilon}} + Q_N + \sqrt{|K \setminus K_{in} \cup K_{out}|}\right)$$

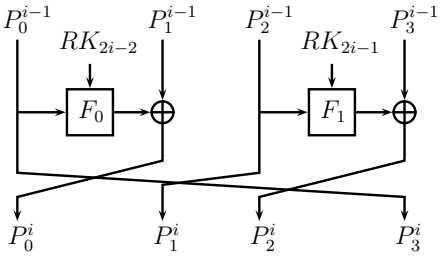
3 Applications

3.1 Application : 12 rounds CLEFIA-128

CLEFIA is a popular lightweight 128-block cipher designed by Shirai et al. in 2007 [16]. It is based on a 4-branch Feistel network. We will provide a short descrition of the encryption function. The best classical impossible differential attack reaches 13 rounds and it is described in [7]. The quantum attack that follows attempts to reach 12 rounds of CLEFIA-128. The last quantum impossible differential attack that could build works on 11 rounds.

3.1.1 Description

We will now describe a R rounds CLEFIA-128 encryption scheme. Let $P = P_0 \mid P_1 \mid P_2 \mid P_3$ be a 128-bit plaintext with each P_i a 32-bits vectors and C be the corresponding cyphertext. A key schedule algorithm, whose description is given in [16], is used to generate $2R$ round keys $(Rk_i)_{0 \leq i \leq 2R-1}$ and 4 whitening keys $Wk_0 \cdots Wk_3$ that all are 32-bits vectors. F_0 and F_1 are the two round functions considered here, they are composed of an S-box followed by a linear layer M_0 for F_0 and M_1 for F_1.



The encryption function is described in the following algorithm:

Algorithm 1 CLEFIA-128 encryption scheme

Require: P
  $P_0^0 \mid P_1^0 \mid P_2^0 \mid P_3^0 \leftarrow P_0 \mid P_1 \oplus Wk_0 \mid P_2 \mid P_3 \oplus Wk_1$
  for $i = 1$ to R **do**
    $P_0^i \leftarrow F_0(P_0^{i-1}, Rk_{2i-2}) \oplus P_1^{i-1}$
    $P_1^i \leftarrow P_2^{i-1}$
    $P_2^i \leftarrow F_1(P_2^{i-1}, Rk_{2i-1}) \oplus P_3^{i-1}$
    $P_3^i \leftarrow P_0^{i-1}$
  end for
  $C \leftarrow P_0^R \mid P_1^R \oplus Wk_2 \mid P_2^R \mid P_3^R \oplus Wk_3$
  return C

3.1.2 Properties of CLEFIA-128

CLEFIA-128 is a 4-branch feistel network. Therefore the scheme does not diffues very well, there is two consequences.

- It is possible to build test function that have a small domain. The filtering will then be very effective

- The diffusion of the impossible pattern to D_{in} and D_{out} will be small too. Hence, D_{in} and D_{out} tend to be small sets. The Pair Generation part might be significantly too ineffective.

3.1.3 Impossible Differential attack

The complexity that follows the analysis given in appendix A are the following:

- Pair Generation: 2^{88}
- Pair Filtering: 2^{32}

Since the size of the key space is 2^{128}, the Grover search will find the secret key in time 2^{64}. Therefore the Pair Generation part is not effective enought.

3.2 Application to AES-128

In [7], a classical impossible differential attack on 7 rounds of AES-128 is described which arguably proposes the best attack known on reduced round AES. Directly quantizying this attack is not going to produce a valid attack since the number of pairs considered is $N = 2^{68} > 2^{64}$. One could try to consider a version of AES-128 on 6 rounds but the complexity to generate the pairs does not decrease enough. In the end, it seems that collecting the differential pair is not possible for this scheme. However, we could generate the superposition over all the diffential pairs on the fly. This time, the complexity of filtering will be drastically increased since it will not be possible to decompose the Grover search on the test function.

In the end, AES-128 seems relatively strong to quantum impossible differential attack at first glance. In the analysis above, the impossible pattern used is borrowed from [7]. A sharper analysis of AES-128 security against quantum impossible should definitly use other impossible patterns.

3.3 FORKAE: SKINNY based Fork scheme

During this internship, I improved the best known classical cryptanalysis of a candidate of the NIST lightweight competition: ForkSkinny and I proposed a quntized version of this attack. In this section, we will describe the improved impossible diffential attack on ForkSkinny-128-256 and its quantizing. The attack described in the following is a relative tweakey attack: we will consider difference in the tweakey to build the attack. To the best of our knoledge, this is the best classical attack so far on this scheme, and the first quantum cryptanalysis on it.

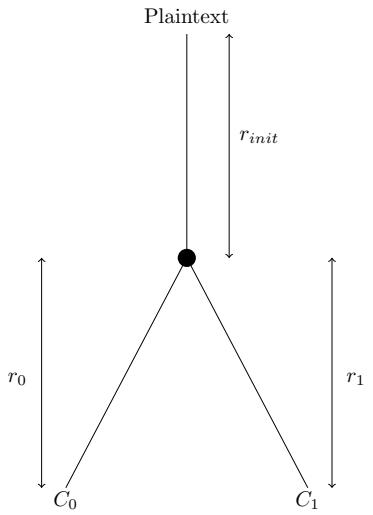
3.3.1 Description

ForkSkinny-128-256 is a NIST candidate introduced in [10]. It is based on the block cipher Skinny-128-256 that was introduced in [3]. The Skinny ciphers follow the tweakey[3] framework, so each instance of plaintext P and tweakey TK produce a ciphertext C. This block cipher has an internal state of 128 bits and a tweakey of 256 bits. ForkSkinny-128-256 is a forkcipher [2], therefore the encryption function will follow the pattern of forkciphers as represented in the following figure.

[3]for tweak and key, we aim to recover the 256 bits of tweakey

In a forkcipher, the plaintext is first encrypted during r_{init} rounds. At this point, the forking point is reached and *three* options are available.

1. If option 1 is chosen, the encryption will continue on branch 1 for r_1 rounds ending up with C_1.
2. If option 0 is chosen, some branch constant (known to the attacker) will be xored to the current state and then the encryption will continue on branch 0 for r_0 rounds ending up with C_0.
3. The last option available is b (both), in this case the encryption scheme will produce both C_0 and C_1.



Similar options are available when decrypting, from C_b we can recover P, $C_{1\oplus b}$ or $(P, C_{1\oplus b})$.

In the case of ForkSkinny-128-256, $r_{init} = 21$ and $r_0 = r_1 = 27$. As for the AES, the internal state is represented by a 4×4 matrix and each cell of the matrix is a byte.

Round function. Let's give a description of the round function of ForkSkinny. The operations are:

- $SubCells$ (SC): Each of the 16 cells of the matrix is modified by a 8×8 Sbox. The precise description of the Sbox in given in [3].
- $AddConstants(AC)$: A constant depending on the round number is xored to the internal state. Since we are building a diffential attack, these constants will cancel out. Hence, we don't need to focus on this precise piece of the round function.
- $AddRoundTweakey$ (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 acts linearly on the columns. Its action is described on the circuit of Figure 2.

Tweakey Schedule. We will describe now the tweakey schedule since it 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:

- First, 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 rond function.

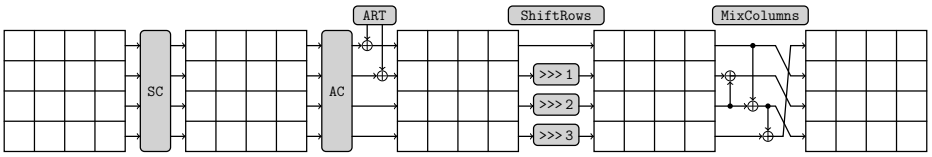


Figure 2: ForkSkinny round function (image from [3])

- Then apply a cell permutation P_T to both matrices. We can remark that the permutation globally swaps the two first lines and the two last lines.

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

- Finally, a LFSR is applied only 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 $\exists x$ $\text{LFSR}^{15}(x) = x$.

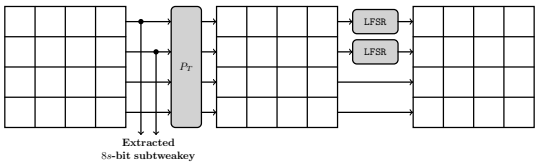
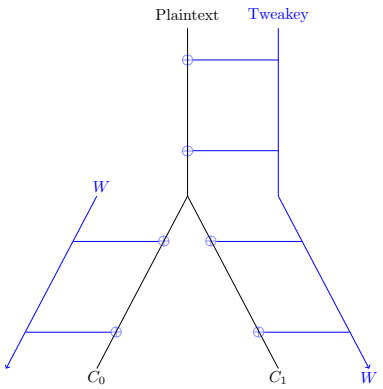


Figure 3: Tweakey schedule (image from [3])

The round function used in ForkSkinny differs slightly from the block cipher SKINNY: *AddConstants* will now take the number of the round into account. This number of round is determined in ForkSkinny following this pattern:

From the plaintext to the forking point, the round number goes from 1 to r_{init} then from the forking point to C_1 the rounds are numbered from $r_{init} + 1$ to $r_{init} + r_1$ and finally, from the forking point to C_0, it goes from $r_{init} + r_1 + 1$ to $r_{init} + r_1 + r_0$. The number of the round minus 1 corresponds to the number of iteration of the tweakey schedule applied so far. The graph shows how the tweakey schedule and the round function align one and other.



In [3], the autors remarked a property of the tweakey schedule that we will use in the attack. It is described in the following lemma:

Lemma 5. *Let δ be a byte such that $LFSR^{15}(\delta) = \delta$. For every position i on the first two lines of a 4×4 matrix, if we set a difference of δ in position i for both of the tweakey matrices, there will be a cancelation of difference in the tweakey material every 30 rounds.*

In the following, this cancelation property is used to optimize the impossible pattern of the impossible differential attack on ForkSkinny.

3.3.2 Classical impossible attack

In this section, we present an impossible differential attack on reduced round ForkSkinny with $r_{init} = 7$, $r_1 = 27$ and $r_0 = 19$. Let's note that we aim to recover the tweakey that is 256 bits. Hence, our attack will be valid as long as its time complexity is smaller than 2^{256}. This attack is inspired by the attack on reduced round SKINNY in [15] that reached up to 23 rounds. First, a description of the impossible pattern is given.

Impossible Pattern. Similarly to the attack in [15], a precise difference will be injected in the tweakey material so that it cancels itself at round 6 and at round 36. Lemma 6 justifies that a difference satisfying these conditions can be found.

The fork structure of ForkSkinny enables an important optimization of the impossible pattern. Indeed, this new impossible pattern is *three* rounds bigger than the impossible pattern presented in [15]. This new pattern is placed between round 4 and round 48 over the initial branch and the left branch (the 0 branch). A precise image of the impossible pattern is given in appendix B.

If in [15] *one* cancelation was used in the impossible pattern, *two* cancelations in a row are used in the impossible pattern for ForkSkinny. By remarking that $r_1 = 27$, the double cancelation is a consequence of Lemma 6. Indeed, if we place a cancelation *two* rounds before the forking point, another cancelation will appear *two* rounds after the forking point on branch 0, since 30 rounds of tweak schedule separate the two cancelations. In appendix B, a precise description of the impossible pattern is given.

Seiving. The seiving phase is mostly the same as in [15]. The differential path $D_{out} \to \Delta_Y$ has no changes with respect to the one in [15] and the differential path $D_{in} \to \Delta_X$ undergoes minor modifications. This similarity in the construction between the two attacks has as a consequence that the complexities both in memory and time of both attacks are equal. Since the attack in [15] has a better complexity than the exhaustive search, this new attack provides a valid attack on reduced round ForkSkinny. The precise differential path is given in appendix B.

In my intership, I designed other efficient classical impossible attacks against some other variants of ForkSkinny: Forkskinny-128-288 and Forkskinny-64-192. In this setting, the impossible pattern is placed between the two ciphertexts.

3.3.3 Quantum impossible attack

The quantum attack will consider a reduced ForkSkinny encryption scheme with $r_{init} = 7$, $r_1 = 27$ and $r_0 = 17$. The classical attack in [15] can not be quantized as memory needs wouls be to hight, therefore we need to perform on this attack to build a classical attack and then quantize it. Both of these transformations induce an overhead in the time complexity. A reduction in the number of rounds seems necessary to balance this overhead. In our case, the number of rounds is reduced by 2. We will use the same impossible pattern as previously. With the 2 rounds reduction, the following update will be made $r_{in} = 3$, $r_{imp} = 18$ and $r_{out} = 3$.

Pair Generation Let's remark that $\Delta_{in} = 32$ and $\Delta_{out} = 72$, $c_{in} = 24$ and $c_{out} = 72$, $|K_{in} \cup K_{out}| = 2^{144}$. By using the formulae of section 3.1, we can state that $N = 2^{104}$ and $Q_N = 2^{120}$.

Filtering First, we aim to collect the conditions in the plaintext and in the ciphertext to propagate to the impossible pattern. By following the ideas of section 3.2, we introduce tests functions. First, we can introduce $T_{\Delta_{in} \to \Delta_X}$ (resp. $T_{\Delta_{out} \to \Delta_y}$) that returns *true* whenever a pair of plaintexts propagates to Δ_X after r_{in} rounds (resp. a pair of ciphertext propagates to Δ_Y after r_{out} rounds). Even if it is not clear how to compute the tests, we can state that since the scheme is IND-CPA[4], they are independant. The conditions in the input and the output can be searched independantly.

Let's now give a descrition of $T_{\Delta_{in} \to \Delta_X}$. We will split the test in subtests $T_{\Delta_{in} \to \Delta_X} = T_{R_1 \to R_2} \wedge T_{R_2 \to R_3}$ where $T_{R_i \to R_{i+1}}$ denotes the test that checks if the differential path is followed from R_i ro R_{i+1}. These *two* tests are not independant therefore the complexity of the Grover searches will multiply one and other. Finally, each round test is a product of tests that verify byte conditions induced by $MixColumns$.

A similar decomposition can be done to compute $T_{\Delta_{out} \to \Delta_y}$. By putting all together, the filtering phase time complexity is:

$$(\sqrt{2^8} + \sqrt{2^8} + \sqrt{2^{16}}) \times (\sqrt{2^8} + \sqrt{2^8}) \times \sqrt{2^{16}} + \sqrt{2^{16}}\sqrt{2^8} = O\left(2^{20}\right)$$

Recover the superposition of the pairs that satifies the conditions can be done in time

$$\sqrt{2^{48}/2^7} = \sqrt{2^{41}}$$

Complexity. The final complexity to the attack is

$$2^{120} + \sqrt{2^{144}}2^{20}\sqrt{2^{41}} + 2^{\frac{256-144}{2}} = O(2^{120}) < 2^{128}$$

In the end, we provided a quantum impossible differential attack on 24 rounds of ForkSkinny.

4 Conclusion

In this report we were able to provide a quantized version of impossible differential attacks. This quantum version of impossible differential attacks is crutial for analyzing in the future the security margin of symetric primitives. We have provided several applications that was left in [5] as an open problem. Also during the internship, we improved the best known attack on a NIST candidate for the lightweight competition. Both results will be submitted to well renowed cryptography conferences.

Since the memory request is a major concern in quantum impossible attack, it is crutial to handle efficiently the pair generation problem to describe an efficient attack. Some scheme that are weak to classical impossible diffential attacks are loosing this weakness while transposing to the quantum world whenever it seems not possible to collect efficently the diffential pairs. A better understanding on the Pair Generation problem seems to be the best improvement towards better quantum impossible differential attacks.

[4] computationally indistinguishable from a random oracle

References

[1] Andris Ambainis. Quantum walk algorithm for element distinctness. *SIAM J. Comput.*, 37(1):210–239, 2007.

[2] Elena Andreeva, Reza Reyhanitabar, Kerem Varici, and Damian Vizár. Forking a blockcipher for authenticated encryption of very short messages. *IACR Cryptology ePrint Archive*, 2018:916, 2018.

[3] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The SKINNY family of block ciphers and its low-latency variant mantis. In *Annual International Cryptology Conference*, pages 123–153. Springer, 2016.

[4] Eli Biham, Alex Biryukov, and Adi Shamir. Cryptanalysis of skipjack reduced to 31 rounds using impossible differentials. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 12–23. Springer, 1999.

[5] Xavier Bonnetain, María Naya-Plasencia, and André Schrottenloher. Quantum security analysis of AES. Cryptology ePrint Archive, Report 2019/272, 2019. `https://eprint.iacr.org/2019/272`.

[6] Christina Boura, Virginie Lallemand, María Naya-Plasencia, and Valentin Suder. Making the impossible possible. *Journal of Cryptology*, 31(1):101–133, Jan 2018.

[7] Christina Boura, María Naya-Plasencia, and Valentin Suder. Scrutinizing and improving impossible differential attacks: Applications to clefia, camellia, lblock and simon. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology – ASIACRYPT 2014*, pages 179–199, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[8] Ran Canetti and Juan A. Garay, editors. *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*, volume 8043 of *Lecture Notes in Computer Science*. Springer, 2013.

[9] Ronald De Wolf. Quantum computing: Lecture notes. *University of Amsterdam*, 2011.

[10] Antoon PURNAL Reza REYHANITABAR Arnab ROY Damian VIZAR Elena ANDREEVA, Virginie LALLEMAND. Forkae v.1. *NIST*, 2019.

[11] Henri Gilbert and Thomas Peyrin. Super-sbox cryptanalysis: Improved attacks for AES-like permutations. In Seokhie Hong and Tetsu Iwata, editors, *Fast Software Encryption*, pages 365–383, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[12] Marc Kaplan, Gaëtan Leurent, Anthony Leverrier, and María Naya-Plasencia. Quantum differential and linear cryptanalysis. *IACR Transactions on Symmetric Cryptology*, 2016(1):71–94, Dec. 2016.

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

[14] Gary L. Miller, editor. *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*. ACM, 1996.

[15] Sadegh Sadeghi, Tahereh Mohammadi, and Nasour Bagheri. Cryptanalysis of reduced round SKINNY block cipher. *IACR Transactions on Symmetric Cryptology*, pages 124–162, 2018.

[16] Taizo Shirai, Kyoji Shibutani, Toru Akishita, Shiho Moriai, and Tetsu Iwata. The 128-bit blockcipher CLEFIA (extended abstract). In Alex Biryukov, editor, *Fast Software Encryption*, pages 181–195, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[17] Peter W Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *Proceedings 35th annual symposium on foundations of computer science*, pages 124–134. Ieee, 1994.

A Detail CLEFIA-128

In [7], a classical impossible attack on CLEFIA is given. We will use the same impossible pattern, i.e $\Delta_X = 0_{32} \mid 0_{32} \mid 0_{32} \mid (\alpha, 0_8, 0_8, 0_8)$, $\Delta_Y = 0_{32} \mid 0_{32} \mid (0_8, \beta, 0_8, 0_8) \mid 0_{32}$ and $r_{imp} = 9$. However this time, we will consider $r_{in} = 2$ and $r_{out} = 1$. Classically, we had $r_{in} = 2$ and $r_{out} = 2$.

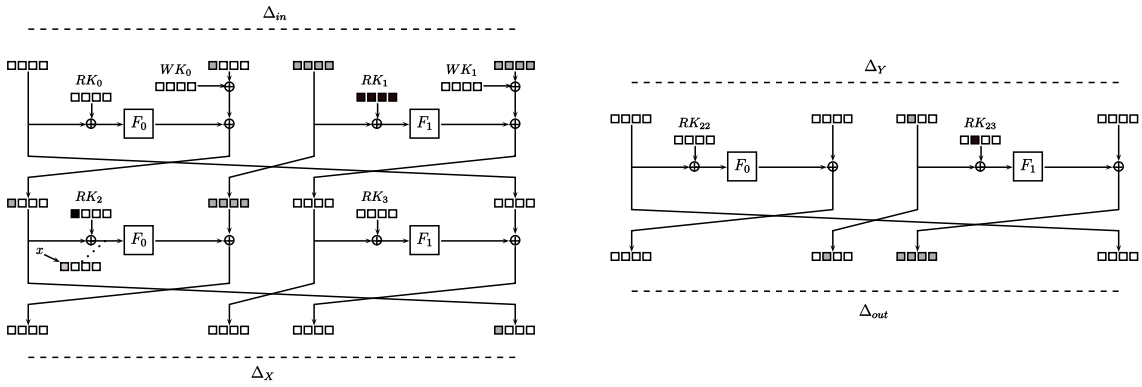


Figure 4: Impossible attack on CLEFIA-128

In our case, the parameters of the attack are:

- $\Delta_{in} = 48$. We are considering the difference on the first byte of ΔP_1. On ΔP_2, the shape of the differences allowed is $M_0(\delta)$ where $\delta = (* \mid 0_{24})$. Finally on ΔP_3, all the 32 bit difference are allowed.
- $\Delta_{out} = 16$. For ΔC_2, differences on the first byte are considered. On ΔC_3, the shape of the differences considered is $M_1(\delta)$ where $\delta = (* \mid 0_{24})$.
- $c_{in} = 40$ and $c_{out} = 8$

Our goal is to recover the keybits of RK_1 and $RK_{23}[1]$, hence $|K_{in} \cup K_{out}| = 2^{40}$. By taking a look at the formulae of section 2, we can remark that the Pair Generation considered here is the "less than one pair" scenario. The number of structures considered is arround 2^{61} to recover $N = 2^{56}$ pairs. This 2^{61} sieve will be performed on the input corresponding to P_0 and P_3 (it is legitimate since $\mid P_0 \otimes P_3 \mid = 2^{64}$). Therefore there exists some 32 bit vector a such

that for all pairs the P_2 part can be written $(a, a + \delta)$ for some vector δ.

qRAM. In this attack, we will have a qRAM function with domain $\Delta P_1 \otimes \Delta P_2 \otimes \Delta P_3 \otimes \widetilde{C_1} \otimes \Delta C_1 \otimes \Delta C_2$ where $\widetilde{C_1}$ corresponds to the second byte of C_1. For every pair, we will write a 1 in the cell of the qRAM that has the corresponding difference and ciphertext.

Let's consider a fixed subkey k, i.e a 40-bits vector.
Collecting conditions. $\Delta_{in} \to \Delta_X$. We will use the state test technique as explained in [7]. Let's call x the 8 bits vector in the leftmost nibble before the F_0 function of round 2 and δ_x the associated difference. One can remark that $\delta_x = \delta P_1$ and that $x = y \oplus RK_2[0]$. Therefore x can take every possible value if we choose the appropiate key $RK_2[0]$. If we now look at the difference at position P_0^2 (that needs to be 0_{32}), we remark that x and δ_x define δp_2, i.e. there exists some function $F_k(x, \delta_x) = \delta p_2$. Similarly, by remarking that the difference in P_3^1 is 0_{32}, there exists some function G_k such that $G_k(\delta p_2)$ (it does not depends of p_2 since it is constant equal to a).

$\Delta_{out} \to \Delta_Y$. With the same idea, one can find H_k such that $\delta c_2 = H_k(c_1, \delta c_1)$.

Retrieving the pairs. Now, we want to find the superposition on the pairs of $Pairs$ that satisfy:

- There exists some 8-bit vector $x \in X$ such that $\delta p_2 = F_k(\delta p_1, x)$
- $\delta p_3 = G_k(F_k(\delta p_1))$
- $\delta c_2 = H_k(c_1, \delta c_1)$

Since $N \cdot 2^{-(c_{in}+c_{out})} = 2^8$, we expect 2^8 pairs satifying the conditions (hence with our notation $\epsilon = 8$) . Therefore retrieving the quantum superposition cost

$$\sqrt{\frac{|X| \cdot |\Delta_x| \cdot |\Delta C_1| \cdot |C_1|}{2^8}} = \sqrt{\frac{2^{16} \cdot 2^{16}}{2^8}} = 2^{12}$$

Now we just need to follow the procedure described in Section 2 to finish the attack. In the end, the complexity for filtering and deciding is

$$2^{12} \cdot 2^{\frac{8+32}{2}} = 2^{32} < 2^{64}$$

However the complexity of gernerating the pairs is $Q_N = 2^{88}$ which invalidate this attack since the final complexity exceeds 2^{64}. To obtain a complexity that does not exceed 2^{64}, we need to consider a reduced version of CLEFIA-128 on 11 rounds. In the end, an improvement on the quantum pair geneation part is needed. The complexity reached for the filtering and deciding is still encouraging.

B ForkSkinny: Impossible pattern and Differential path

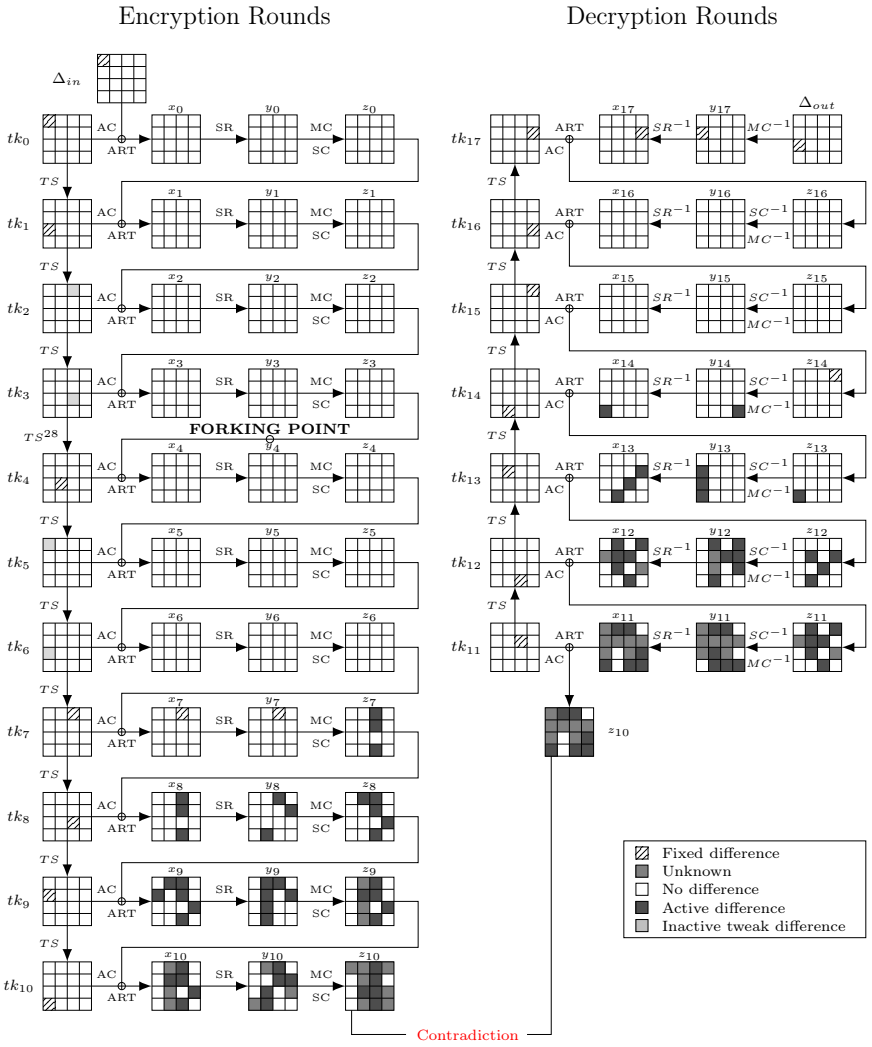


Figure 5: Impossible Pattern

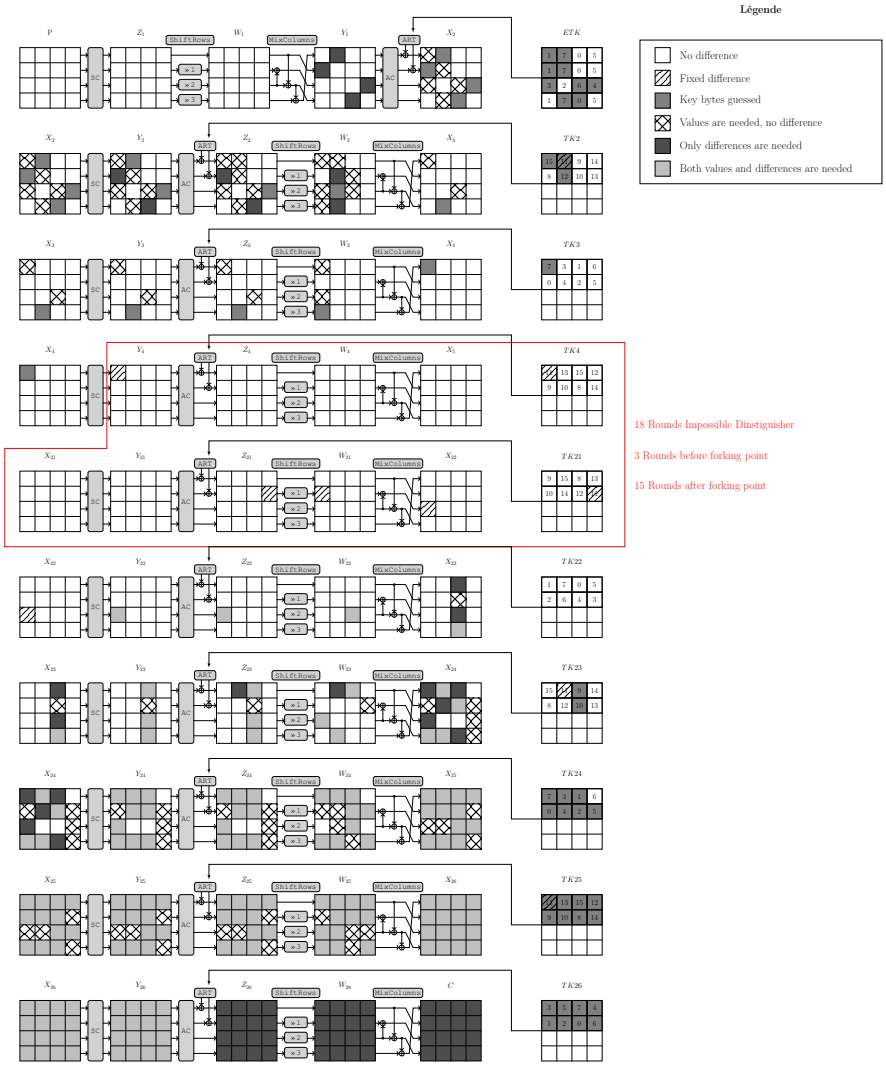


Figure 6: Differential path