mathematics

MDPI

Article

Quantum Automated Tools for Finding Impossible Differentials

Huiqin Xie [1,2,*], Qiqing Xia [3,4,5], Ke Wang [1], Yanjun Li [6] and Li Yang [3,5]

[1] Department of Cryptography Science and Technology, Beijing Electronic Science and Technology Institute, Beijing 100070, China
[2] Key Laboratory of Cryptography of Zhejiang Province, Hangzhou Normal University, Hangzhou 311121, China
[3] Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100085, China
[4] School of Cyber Security, University of Chinese Academy of Sciences, Beijing 100049, China
[5] Key Laboratory of Cyberspace Security Defense, Beijing 100085, China
[6] Information Industry Information Security Evaluation Center, The 15th Research Institute of China Electronics Technology Group Corporation, Beijing 100083, China
* Correspondence: xiehuiqindky@163.com

Abstract: Due to the superiority of quantum computing, traditional cryptography is facing a severe threat. This makes the security evaluation of cryptographic systems in quantum attack models both significant and urgent. For symmetric ciphers, the security analysis heavily relies on cryptanalysis tools. Thus, exploring the use of quantum algorithms in traditional cryptanalysis tools has garnered considerable attention. In this study, we utilize quantum algorithms to improve impossible differential attacks and design two quantum automated tools to search for impossible differentials. The proposed quantum algorithms exploit the idea of miss-in-the-middle and the properties of truncated differentials. We rigorously prove their validity and calculate the quantum resources required for their implementation. Compared to the existing classical automated cryptanalysis, the proposed quantum tools have the advantage of accurately characterizing S-boxes while only requiring polynomial complexity, and can take into consideration the impact of the key schedules in a single-key model.

Keywords: quantum cryptanalysis; symmetric cryptography; impossible differential attack; automated analysis

MSC: 94A60

Citation: Xie, H.; Xia, Q.; Wang, K.; Li, Y.; Yang, L. Quantum Automated Tools for Finding Impossible Differentials. *Mathematics* **2024**, *12*, 2598. https://doi.org/10.3390/math12162598

Academic Editor: Cheng-Chi Lee

Received: 14 July 2024
Revised: 19 August 2024
Accepted: 21 August 2024
Published: 22 August 2024

1. Introduction

The development of quantum computers has progressed steadily. As soon as quantum computers are successfully built, traditional cryptography will be severely threatened. By utilizing Shor's algorithm [1], adversaries possessing quantum computers can break public key cryptosystems built on the integer factorization problem, such as the RSA scheme widely used in secure communication. Apart from public key cryptography, studies on the cryptanalysis of symmetric cryptography against quantum adversaries have also achieved many outstanding results. By utilizing Grover's algorithm, one can achieve a quadratic speed-up when searching unordered databases [2]. Therefore, to restore the same ideal security as that in a classical setting, the key lengths of symmetric ciphers must be doubled in the quantum setting.

The exhaustive attack can only evaluate the ideal security margin of cryptographic schemes. To accurately grasp the quantum security of currently used symmetric schemes, we also need to investigate other possible quantum attacks. In this direction, Simon's algorithm [3] is frequently used. Kuwakado and Morri first applied Simon's algorithm to attack the Feistel structure and proposed a three-round quantum distinguisher [4]. Then they also attacked the Even–Mansour cipher using a similar idea and successfully recovered the key [5]. The authors of [6] forged messages of the CBC-MAC scheme using the method

presented in [4]. Kaplan et al. also further developed the results in [4] and attacked several symmetric systems, including GCM, PMAC, and CLOC [7]. Both [6,7] proved the correctness of the quantum distinguisher even if the Feistel structure has round functions that are not permutations.

Leander and Alexander embedded Simon's algorithm into Grover's algorithm in order to identify the correct key of the FX structure [8]. Following this attack strategy, Dong and Wang broke Feistel schemes and obtained the key using the quantum distinguisher shown in [4]. Afterward, they applied the same strategy to extract the key of the generalized Feistel cipher [9,10]. The above attacks were all implemented under the quantum version of the chosen plaintext model, also known as the Q_2 model [11–13]. In this attack model, the cryptographic oracle can be queried with superposition states. The authors of [14] studied quantum-related-key notions in which quantum adversaries can use superposition states of related keys to query oracles. Hosoyamada et al. then further investigated quantum-related-key notion and recovered the key of the two-round Even–Mansour algorithm [15]. Jaques et al. analyzed the complexity of Grover's algorithm when attacking AES [16].

Apart from specific quantum attacks, studying quantum versions of cryptanalysis tools (such as integral, differential, and linear analyses) is also essential. Zhou et al. utilized Grover's algorithm for differential attacks [17]. Kaplan et al. subsequently used Grover's algorithm to enhance some variants of differential attacks and linear attacks [18]. Xie et al. made use of the Bernstein–Vazirani algorithm to search for high-probability differentials [17]. The authors of [19] implemented quantum collision attacks on Whirlpool and AES-MMO schemes via differential characteristics. Dong et al. enhanced truncated differential analysis through quantum algorithms and broke the Grøstl-512 scheme and the AES-MMO cipher [20].

Our contributions. In this paper, we study the applications of Simon's algorithm to cryptanalysis tools for symmetric ciphers. We bring the superiority of quantum computing into traditional impossible differential analysis, and design quantum automated tools to search for impossible differentials. First, we propose a basic quantum algorithm that can find impossible differentials by imitating the classical impossible differential technique. Subsequently, by allowing the differentials to be truncated, we present another improved quantum algorithm. We provide the correctness proofs for the proposed algorithms and evaluate their quantum complexities. The proposed quantum tools offer several advantages, as follows:

- The quantum algorithms can be implemented in the Q1 attack model, without any query to the quantum encryption or decryption oracle. In contrast, many other quantum attacks [4–7,9,10] require adversaries to perform quantum queries with superposition states. Our quantum tools are much easier to realize and, thus, more practical.
- Classical automated impossible differential cryptanalysis tools include the UID tool [21], U-tool [22], WW-tool [23], MILP tool [24], and SAT tool [25]. When faced with large-scale S-boxes, these classical automated tools either do not describe the construction of S-boxes and simply treat them as bijections or only describe the reduced differential distribution table of S-boxes. So far, there is no classical automated tool that can fully characterize large-scale S-boxes. Even in the case where S-boxes are only partially described, the searching space usually expands rapidly as the number of rounds increases, making it impossible to search many rounds. Our quantum automated tools fully leverage the parallel advantages of quantum computing, allowing for the complete characterization of S-boxes while maintaining complexity within polynomial time. They can fully characterize any nonlinear functions, and the complexity increases linearly with respect to the number of rounds.
- Most classical automated impossible differential cryptanalysis tools cannot take the key schedule into account in a single-key model. However, our tools include the key schedule when implementing the quantum circuit of encryption, allowing the impact of the key schedule on differential propagation types to be fully considered. Specifically, in a related-key model [26], the attacker can introduce a key differential so

that the propagation of this differential in both the key schedule and encryption process is accounted for when searching for impossible differentials. This approach provides a more accurate characterization of differential propagation and helps identify more or longer impossible differentials. However, the single-key model is more practical and more commonly used since the related key model requires too much power from the attacker. In a single-key model, the master key is not allowed to introduce a differential to the key. Therefore, most classical automated tools for searching distinguishers ignore the impact of the key schedule and simply treat the subkeys of different rounds as independent constants. The process of searching for distinguishers lacks the analysis of key schedules. In contrast, although our quantum tools are also in a single-key model, they treat the entire encryption algorithm, including the key schedule, as a black box, and the state of the master key is a part of the input. The encryption of the input superposition state includes the calculation of the key schedule. All subkeys are obtained by running the key schedule on the master key. Thus, the connection between different subkeys is fully considered, which helps to accurately characterize the differential propagation.

Comparison with related works. A periodic function constructed based on a block cipher will yield a quantum distinguisher when combined with Simon's algorithm [4–7]. Owing to this, Xiang et al. proposed a classical algorithm for constructing periodic functions using existing probability-1 truncated differentials and applied this method to two block ciphers [27]. The algorithm they designed to identify periodic functions is a classical algorithm. Their method does not involve searching for truncated differentials, but only uses truncated differentials that already exist to construct periodic functions. In contrast, we study how to utilize quantum algorithms to identify impossible differentials. To achieve this goal, we construct a quantum algorithm for probability-1 truncated differentials based on Simon's algorithm. Our work has different goals from those of [27]. One is to identify impossible differentials, whereas the other is to construct periodic functions. The methods used are also different. One uses classical algorithms, whereas the other uses quantum algorithms.

2. Preliminaries

We present a simple overview of the necessary concepts and their related results.

2.1. Quantum Attack Models

n, m are two arbitrary positive integers. $F : \mathbb{F}_2^n \to \mathbb{F}_2^m$ is a Boolean function. If the unitary operation

$$U_F : \sum_{x,y} |x\rangle|y\rangle \to \sum_{x,y} |x\rangle|y \oplus F(x)\rangle, \tag{1}$$

is realized by a quantum circuit, we say that this circuit evaluates F quantumly. Any vectorial Boolean function can be evaluated by a quantum circuit constructed with gates in a finite but universal set of unitary gates. Such a set is referred to as a universal gate set [28]. For example, the phase gate S, Hadamard gate H, non-Clifford gate T, and controlled-NOT quantum gate $CNOT$ form a universal gate set. Each gate in this set is calculated as a single operation. For any vectorial Boolean function F, let the notation $|U_F|$ denote the number of quantum universal gates required to implement U_F.

Two common attack models for adversaries are considered when analyzing the quantum security of cryptographic primitives [12]. One is the Q1 attack model, where adversaries can utilize quantum computers to perform offline computations but can only make classical online queries. The other is the Q2 attack model, where adversaries can also execute quantum queries. Specifically, a Q2 adversary can also make queries to cryptographic primitives with inputs in superposition states and obtain the quantum states of their outputs. The Q2 attack model is stricter in terms of the adversaries' ability because querying the quantum oracles of cryptographic systems is usually not easy to realize in practice.

2.2. Simon's Algorithm

Given $F : \mathbb{F}_2^n \to \mathbb{F}_2^m$ and a private vector $s \in \mathbb{F}_2^n$ satisfying

$$[F(x_1) = F(x_2)] \Leftrightarrow [x_1 \oplus x_2 \in \{0^n, s\}], \ \ \forall x_1, x_2 \in \mathbb{F}_2^n,$$

Simon's algorithm [3] was originally used to solve the period s. If a function has such a period, we say that it meets Simon's promise. Finding s requires at least $O(2^{n/2})$ classical queries when using classical algorithms, whereas Simon's algorithm only requires $O(n)$ quantum queries. With the quantum circuit of F, Simon's algorithm requires repeating the steps below:

1. Prepare an $(n+m)$-qubit quantum state $|0^n\rangle|0^m\rangle$. We apply the Hadamard transform $H^{\otimes n}$ to the left register, obtaining the following:
$$\frac{1}{\sqrt{2^n}} \sum_{x \in \mathbb{F}_2^n} |x\rangle|0^m\rangle.$$
2. We implement the unitary operator U_F of F and obtain the following state:
$$\frac{1}{\sqrt{2^n}} \sum_{x \in \mathbb{F}_2^n} |x\rangle|F(x)\rangle.$$
3. We measure the last register to obtain a vector $F(z)$; subsequently, the remaining registers will be as follows:
$$\frac{1}{\sqrt{2}}(|z\rangle + |z \oplus s\rangle).$$
4. We perform Hadamard operators $H^{\otimes n}$ on the above state, obtaining the following:
$$\frac{1}{\sqrt{2^{n+1}}} \sum_{\gamma \in \mathbb{F}_2^n} (-1)^{\gamma \cdot z}[1 + (-1)^{\gamma \cdot s}]|\gamma\rangle.$$
5. We measure this state. If a vector γ satisfies $\gamma \cdot s = 1$, its amplitude must be 0. The measurement result γ always satisfies $\gamma \cdot s = 0$.

The process of Simon's algorithm involves repeating the above subroutine $O(n)$ times, yielding $n-1$ vectors that are perpendicular to s and are linearly independent. Using linear algebraic knowledge, we can easily compute s.

A quantum circuit illustration of Simon's subroutine (steps 1–5) is shown in Figure 1. Running steps 1–5 requires $2n$ Hadamard operators and 1 execution of the unitary operator U_F. Therefore, there are $m+n$ qubits and $O(2n^2 + n|U_F|)$ gates in total in Simon's algorithm when run on F.

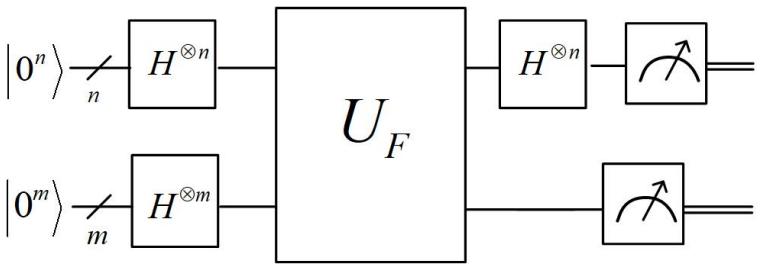

Figure 1. Circuit illustration of Simon's algorithm.

In the cryptanalysis scenario, it is not always easy to construct a Boolean function that satisfies Simon's promise. Even if a periodic function is constructed, unwanted collisions not caused by this period may occur. Kaplan et al. relaxed Simon's promise and proved the following theorem [7].

Theorem 1 ([7]). *If $F : \mathbb{F}_2^n \to \mathbb{F}_2^n$ satisfies $\epsilon(F, s) \leq e_0 < 1$ for a period $s \in \mathbb{F}_2^n$ and some constant e_0, where we have the following:*

$$\epsilon(F, s) = \max_{t \in \mathbb{F}_2^n \setminus \{0^n, s\}} \Pr_x[F(x \oplus t) = F(x)],$$

then by repeating the subroutine cn times, the probability that Simon's algorithm returns s is not less than $1 - (2(\frac{1+e_0}{2})^c)^n$.

2.3. Linear Structure

We will transform the problem of seeking impossible differentials into a problem of seeking linear structures.

Definition 1 ([29]). *Given a function $F : \mathbb{F}_2^n \to \mathbb{F}_2^m$, $a \in \mathbb{F}_2^n$ is named a linear structure if we have the following:*

$$F(x \oplus a) \oplus F(x) = b, \ \ \forall x \in \mathbb{F}_2^n \tag{2}$$

for some vector $b \in \mathbb{F}_2^m$. In other words, $F(x \oplus a) \oplus F(x)$ is constant.

For any $a \in \mathbb{F}_2^n$, $b \in \mathbb{F}_2^m$ satisfying Equation (2), we refer to the pair (a, b) as F's linear structure duad. If b is a zero vector 0^m, then a is called F's period. If (a_1, b_1), (a_2, b_2) are two linear structure duads of F, then we have the following:

$$F(x \oplus a_1 \oplus a_2) \oplus F(x) = F(x \oplus a_1) \oplus b_2 \oplus F(x) = b_1 \oplus b_2.$$

Thus, $(a_1, b_1) \oplus (a_2, b_2)$ remains as one of F's linear structure duad. All of F's linear structure duads form a subspace within the vector space $\mathbb{F}_2^{n+m}$. This subspace is referred to as the linear structure space and is denoted by L_F.

For any vectors $v \in \mathbb{F}_2^m$, $u \in \mathbb{F}_2^n$, if there is $x \in \mathbb{F}_2^n$ satisfying $F(x) \oplus F(u \oplus x) = v$, then (u, v) is said to make a "*match*" of F at x. Being a linear structure duad is equivalent to causing *matches* of F at all points $x \in \mathbb{F}_2^n$.

3. A Basic Quantum Tool for Finding Impossible Differentials

We present a universal quantum algorithm that finds impossible differentials of an arbitrary block cipher. The main idea is to use probability-1 differentials to construct impossible differentials. Since probability-1 differentials of an encryption function are also its linear structure duads, we can find them by constructing a quantum algorithm that finds linear structure duads. We first show a quantum algorithm that finds linear structure duads; based on this algorithm, we propose a basic quantum tool for impossible differentials.

3.1. Finding Linear Structure Duads via Simon's Algorithm

L_F is the linear structure space of function $F : \mathbb{F}_2^n \to \mathbb{F}_2^m$ as defined in Section 2.3. Namely,

$$L_F = \{(a, b) \in \mathbb{F}_2^n \times \mathbb{F}_2^m | F(x) \oplus F(x \oplus a) = b, \forall x \in \mathbb{F}_2^n\}.$$

We aim to obtain L_F. The value of m does not need to be equal to n. We define a new function as follows:

$$\begin{aligned} G : \mathbb{F}_2^n \times \mathbb{F}_2^m &\to \mathbb{F}_2^m \\ (x, y) &\to F(x) \oplus y. \end{aligned} \tag{3}$$

For any duad $(a, b) \in \mathbb{F}_2^n \times \mathbb{F}_2^m$, if (a, b) is G's period, it will also be F's linear structure duad. Therefore, F's linear structure duads can be found using Simon's algorithm. Based on this analysis, we propose Algorithm 1, referred to as FindStruct, which is used to identify linear structure duads, as follows:

Algorithm 1 Algorithm FindStruct

Input: a parameter c and the access to the quantum unitary operator U_F of a function $F : \mathbb{F}_2^n \to \mathbb{F}_2^m$.

Output: the linear structure space L_F.

1: **for** $i = 1$ to $c(n+m)$ **do**

2: Prepare an $(n+2m)$-qubit state $|0^n\rangle|0^m\rangle|0^m\rangle$ and implement the Hadamard gate $H^{\otimes(n+m)}$, obtaining

$$|\Psi_1\rangle = \frac{1}{\sqrt{2^{n+m}}} \sum_{x\in\mathbb{F}_2^n, y\in\mathbb{F}_2^m} |x\rangle|y\rangle|0^m\rangle.$$

3: Use the unitary operator U_F to obtain the state

$$|\Psi_2\rangle = \frac{1}{\sqrt{2^{n+m}}} \sum_{x\in\mathbb{F}_2^n, y\in\mathbb{F}_2^m} |x\rangle|y\rangle|F(x)\rangle.$$

4: Apply CNOT operators to the last two registers to obtain the state

$$|\Psi_3\rangle = \frac{1}{\sqrt{2^{n+m}}} \sum_{x\in\mathbb{F}_2^n, y\in\mathbb{F}_2^m} |x\rangle|y\rangle|F(x)\oplus y\rangle.$$

5: Measure the rightmost register to obtain a value $z \in \mathbb{F}_2^m$, then there exist vectors $x_0 \in \mathbb{F}_2^n$, $y_0 \in \mathbb{F}_2^m$ such that $F(x_0) \oplus y_0 = z$. Thus, the two leftmost registers are collapsed into the following:

$$\frac{1}{\sqrt{|S_z|}} \sum_{(x,y)\in S_z} |x\rangle|y\rangle, \tag{4}$$

where $S_z = \{(x,y) \in \mathbb{F}_2^{n+m} | F(x) \oplus y = z\}$.

6: Implement the Hadamard gate $H^{\otimes(n+m)}$ on the above state to obtain

$$\frac{1}{\sqrt{|S_z|2^{n+m}}} \sum_{\substack{\gamma_1\in\mathbb{F}_2^n \\ \gamma_2\in\mathbb{F}_2^m}} \sum_{(x,y)\in S_z} (-1)^{x\cdot\gamma_1\oplus y\cdot\gamma_2} |\gamma_1\rangle|\gamma_2\rangle,$$

then measure this state to obtain a vector $\gamma^{(i)} \in \mathbb{F}_2^{n+m}$.

7: **end for**

8: After obtaining the vectors $\gamma^{(1)}, \gamma^{(2)}, \cdots, \gamma^{(c(m+n))} \in \mathbb{F}_2^{n+m}$ by steps 1–7, solve the following linear equation

$$\begin{cases} \gamma^{(1)} \cdot (x,y) = 0 \\ \gamma^{(2)} \cdot (x,y) = 0 \\ \vdots \\ \gamma^{(c(m+n))} \cdot (x,y) = 0, \end{cases} \tag{5}$$

where $(x,y) \in \mathbb{F}_2^n \times \mathbb{F}_2^m$ are unknowns, and output its solution space.

The quantum computing part of the FindStruct algorithm involves repeating steps 2–6 independently for $c(m+n)$ times. We refer to steps 2–6 as the FindStruct subroutine. Its quantum circuit is shown in Figure 2. Steps 1–7 involve executing Simon's subroutine $c(m+n)$ times on $G(x,y) = F(x) \oplus y$ to independently obtain $c(m+n)$ vectors $\gamma^{(1)}, \cdots, \gamma^{(c(m+n))}$.

We expect that, as with the original Simon's algorithm, the periods of G are orthogonal to $\gamma^{(1)}, \cdots, \gamma^{(c(m+n))}$; thus, we can obtain the periods of G by solving Equation (5), which are also linear structure duads of F. Theorem 1 provides the conditions for Simon's algorithm to output the periods. Therefore, for the FindStruct algorithm to successfully output L_F, the G function must satisfy this condition. However, G may have more than one period since function F may have more than one linear structure duad, or the length of G's output may not be equal to that of the input. Therefore, simply applying Theorem 1 is insufficient to justify the soundness of the FindStruct algorithm. To address this, we define a new parameter, $\theta(\cdot)$. Function $F : \mathbb{F}_2^n \to \mathbb{F}_2^m$ is defined as follows:

$$\begin{aligned}\theta(F) &= \max_{\substack{a\in\mathbb{F}_2^n b\in\mathbb{F}_2^m \\ (a,b)\notin L_F}} \Pr_x\left[F(x)\oplus F(x\oplus a)=b\right] \\ &= \max_{\substack{a\in\mathbb{F}_2^n b\in\mathbb{F}_2^m \\ (a,b)\notin L_F}} \frac{1}{2^n}\left|\left\{x\in\mathbb{F}_2^n | F(x)\oplus F(x\oplus a)=b\right\}\right|. \end{aligned} \tag{6}$$

It is obvious that $0 \le \theta(F) < 1$. If (a, b) is in L_F; that is, if it is F's linear structure duad, then it will cause a *match* of F at each point $x \in \mathbb{F}_2^n$. If $(a, b) \notin L_F$, then the number of *matches* caused by (a, b) will be less than 2^n. The closer the value of $\theta(F)$ is to zero, the fewer *matches* that the vector (a, b) not in L_F can cause. Theorem 2 shows the validity of Algorithm 1 (FindStruct).

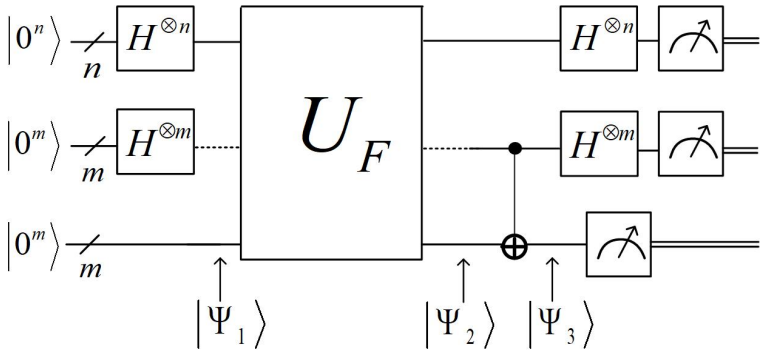

Figure 2. Quantum circuit of the FindStruct subroutine.

Theorem 2. *Let L be the solution set output by the FindStruct algorithm run on $F : \mathbb{F}_2^n \to \mathbb{F}_2^m$ with a parameter, c, then $L_F \subseteq L$. Moreover, if there is a constant, e_0, such that $\theta(F) \le e_0 < 1$, then the probability of L_F being equal to L is no less than $1 - \left(2(\frac{1+e_0}{2})^c\right)^{n+m}$.*

The idea of proving Theorem 2 is almost the same as that of Theorem 1 in [7], with the exception of cases where the function has multiple periods or linear structures, as well as cases where the lengths of the output and input are unequal, need to be considered. The proof is presented in Appendix A.

According to Theorem 2, setting c greater than $3/(1 - e_0)$ ensures that the probability of the FindStruct algorithm outputting vectors not in L_F decreases exponentially with n. The condition $\theta(F) \le e_0 < 1$ implies that the vectors that are not linear structure duads of F should not cause too many *matches*, or in other words, vectors that are not periods of G defined in Equation (3) should not cause too many collisions.

3.2. Quantum Tool for Impossible Differentials

The method of finding impossible differentials involves finding probability-1 differential characteristics that propagate, respectively, from the input end and the output end of the cipher but cannot match when they meet [30].

$E^{(r)}$ is an arbitrary block cipher that has r rounds. E denotes the round function. The block size is n and the key space is $\mathcal{K} = \mathbb{F}_2^m$. For each $k \in \mathcal{K}$, the output of $E^{(r)}$ on

plaintext x is $E_k^{(r)}(x)$. Our goal is to obtain impossible differentials of $E^{(r)}$. Namely, we find $(\alpha, \beta) \in \mathbb{F}_2^n \times \mathbb{F}_2^n$, such that we have the following:

$$E_k^{(r)}(x) \oplus E_k^{(r)}(\alpha \oplus x) \neq \beta, \forall x \in \mathbb{F}_2^n, \forall k \in \mathbb{F}_2^m.$$

We divide $E^{(r)}$ into two functions, $E^{(r)} = E^{(r_2)} \circ E^{(r_1)}$. Here, $1 \leq r_1, r_2 \leq r-1$ and $r_1 + r_2 = r$. Let ${E^{(r_2)}}^{-1}$ be the inverse function of $E^{(r_2)}$. As shown in Figure 3, if $(\Delta x_1, \Delta y_1)$ is a differential of $E^{(r_1)}$, $(\Delta x_2, \Delta y_2)$ is a differential of ${E^{(r_2)}}^{-1}$, satisfying the following:

$$\begin{aligned} E_k^{(r_1)}(x \oplus \Delta x_1) \oplus E_k^{(r_1)}(x) &= \Delta y_1, \forall x \in \mathbb{F}_2^n, \forall k \in \mathbb{F}_2^m \\ {E_k^{(r_2)}}^{-1}(x \oplus \Delta x_2) \oplus {E_k^{(r_2)}}^{-1}(x) &= \Delta y_2, \forall x \in \mathbb{F}_2^n, \forall k \in \mathbb{F}_2^m, \end{aligned}$$

and $\Delta y_1 \neq \Delta y_2$, then $(\Delta x_1, \Delta x_2)$ will be an impossible differential of $E^{(r)}$. Therefore, to identify impossible differentials of $E^{(r)}$, we only need to obtain differentials of $E^{(r_1)}$ and ${E^{(r_2)}}^{-1}$ with a probability of 1.

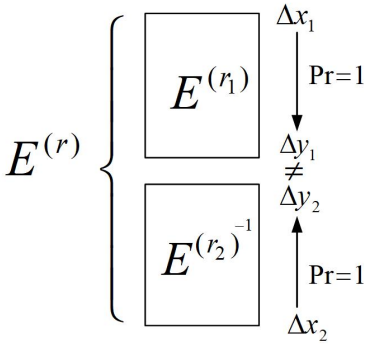

Figure 3. Construction idea of impossible differentials.

For any t-round block cipher $E^{(t)}$ that has a key length of m and block size n, we treat both the plaintext and key as inputs of $E^{(t)}$, then the function, i.e.,

$$\begin{aligned} E^{(t)} : \mathbb{F}_2^m \times \mathbb{F}_2^n &\to \mathbb{F}_2^n \\ (k, x) &\to E_k^{(t)}(x) \end{aligned}$$

is public and completely determined. The Q1 adversaries can construct the unitary operator

$$U_{E^{(t)}} : \sum_{\substack{(k,x) \in \mathbb{F}^{m+n} \\ y \in \mathbb{F}_2^n}} |k, x\rangle |y\rangle \longrightarrow \sum_{\substack{(k,x) \in \mathbb{F}^{m+n} \\ y \in \mathbb{F}_2^n}} |k, x\rangle |y \oplus E_k^{(t)}(x)\rangle$$

themselves. As

$$\begin{aligned} & E^{(t)}(k \oplus 0^m, x \oplus \Delta x) \oplus E^{(t)}(k, x) = \Delta y, \forall (k, x) \in \mathbb{F}_2^{m+n} \\ \iff & E_k^{(t)}(x \oplus \Delta x) \oplus E_k^{(t)}(x) = \Delta y, \forall k \in \mathbb{F}_2^m, \forall x \in \mathbb{F}_2^n, \end{aligned}$$

if $((0^m, \Delta x), \Delta y)$ is $E^{(t)}$'s linear structure duad, then $(\Delta x, \Delta y)$ will be identified as $E^{(t)}$'s differential of probability-1. Thus, we can use the FindStruct algorithm to obtain $E^{(t)}$'s differentials of probability-1, with the additional requirement that the first m bits of the linear structures be 0. This can be achieved by adding additional equations to Equation (5) when Algorithm 1 (FindStruct) is run on $E^{(t)}$.

The following Algorithm 2 is designed to search impossible differentials.

Algorithm 2 FindImpoDiff algorithm

Input: a parameter c and a cipher $E^{(r)}: \mathcal{K} \times \mathbb{F}_2^n \to \mathbb{F}_2^n$. ($\mathcal{K} = \mathbb{F}_2^m$ is the key space.)
Output: Impossible differentials of $E^{(r)}$.

1: **for** $r_1 = 1$ to $r-1$ **do**
2: Let $r_2 = r - r_1$, divide $E^{(r)}$ into two functions $E^{(r)} = E^{(r_2)} \circ E^{(r_1)}$.
3: Run steps 1–7 of Algorithm 1 (FindStruct) on $E^{(r_1)}: \mathbb{F}_2^m \times \mathbb{F}_2^n \to \mathbb{F}_2^n$ with parameter c obtaining $c(m+2n)$ vectors $\gamma^{(1)}, \gamma^{(2)}, \cdots, \gamma^{c(m+2n)} \in \mathbb{F}_2^{m+2n}$.
4: Solve the following equation:

$$\begin{cases} \gamma^{(1)} \cdot (k,x,y) = 0 \\ \gamma^{(2)} \cdot (k,x,y) = 0 \\ \vdots \\ \gamma^{(c(m+2n))} \cdot (k,x,y) = 0 \\ k = 0^m, \end{cases} \tag{7}$$

where $(k,x,y) \in \mathbb{F}_2^{m+n+n}$ are unknowns, to obtain the solution set A_{r_1}.
5: Run steps 1–7 of Algorithm 1 (FindStruct) on ${E^{(r_2)}}^{-1}: \mathbb{F}_2^m \times \mathbb{F}_2^n \to \mathbb{F}_2^n$ with parameter c obtaining $c(m+2n)$ vectors $\tilde{\gamma}^{(1)}, \tilde{\gamma}^{(2)}, \cdots, \tilde{\gamma}^{c(m+2n)} \in \mathbb{F}_2^{m+2n}$.
6: Solve the following equation:

$$\begin{cases} \tilde{\gamma}^{(1)} \cdot (k,x,y) = 0 \\ \tilde{\gamma}^{(2)} \cdot (k,x,y) = 0 \\ \vdots \\ \tilde{\gamma}^{(c(m+2n))} \cdot (k,x,y) = 0 \\ k = 0^m, \end{cases} \tag{8}$$

where $(k,x,y) \in \mathbb{F}_2^{m+n+n}$ are unknowns, to obtain the solution set B_{r_2}.
7: **for** $(0^m, \Delta x_1, \Delta y_1) \in A_{r_1}$ **do**
8: **for** $(0^m, \Delta x_2, \Delta y_2) \in B_{r_2}$ **do**
9: **if** $\Delta x_1 \neq 0^n \wedge \Delta x_2 \neq 0^n \wedge \Delta y_1 \neq \Delta y_2$ **then**
10: Output $(\Delta x_1, \Delta x_2)$.
11: **end if**
12: **end for**
13: **end for**
14: **end for**

Figure 4 shows the flowchart of Algorithm 2 (FindImpoDiff). Steps 3–4 are used to identify the differentials of $E^{(r_1)}$. Steps 5–6 are used to identify the differentials of ${E^{(r_2)}}^{-1}$. Since $E^{(r_1)}$ and ${E^{(r_2)}}^{-1}$ are public and determinate functions, the adversary can execute the unitary operators $U_{E^{(r_1)}}$ and $U_{{E^{(r_2)}}^{-1}}$ when invoking the FindStruct subroutine.

Given a block cipher $E^{(r)}$, we define

$$\Theta(E^{(r)}) = \max\{\theta(E^{(t)}) | 1 \leq t \leq r-1\},$$

where $E^{(t)}$ is a t-round reduced version of $E^{(r)}$ and $\theta(E^{(t)})$ is defined by Equation (6); as follows:

$$\theta(E^{(t)}) = \max_{\substack{(a_1,a_2) \in \mathbb{F}_2^m \times \mathbb{F}_2^n \\ b \in \mathbb{F}_2^n \\ ((a_1,a_2),b) \notin L_{E^{(t)}}}} \frac{1}{2^{m+n}} \left| \left\{ (k,x) \in \mathbb{F}_2^m \times \mathbb{F}_2^n | E^{(t)}(k,x) \oplus E^{(t)}(k \oplus a_1, x \oplus a_2) = b \right\} \right|,$$

where $L_{E^{(t)}}$ denotes the linear structure space of $E^{(t)}$. $\theta(E^{(t)})$ denotes the maximum number of *matches* that vectors not in the linear structure duads of $E^{(t)}$ can cause. Therefore, the smaller the parameter $\Theta(E^{(r)})$ is, the fewer *matches* the vectors not in the linear structure duads of the reduced version of $E^{(r)}$ can cause. According to Theorem 2, the following theorem holds:

Theorem 3. *Block cipher $E^{(r)}$ satisfies $\Theta(E^{(r)}) \leq e_0 < 1$ for a constant e_0. If executing the FindImpoDiff algorithm on $E^{(r)}$ with parameter c outputs $(\Delta x_1, \Delta x_2)$, the probability of $(\Delta x_1, \Delta x_2)$ being $E^{(r)}$'s impossible differential is no less than $1 - 2\left(2(\frac{1+e_0}{2})^c\right)^{2n+m}$, where m is the key length and n is the block size.*

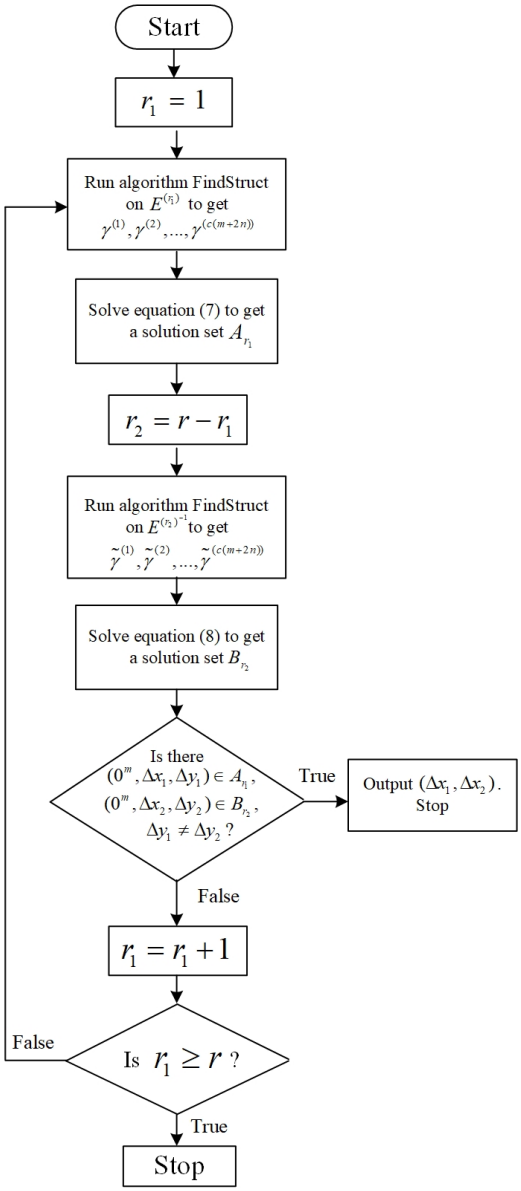

Figure 4. Flowchart of Algorithm 2 (FindImpoDiff).

According to Theorem 3, setting c greater than $3/(1 - e_0)$ guarantees that the probability of the FindImpoDiff algorithm outputting vectors that are not impossible differentials decreases exponentially with n.

Notably, according to Theorem 2, any linear structure duad of $E^{(r_1)}$ whose first m bits are zero must belong to the solution set A_{r_1}. On the other hand, $((0^m, \Delta x_1), \Delta y_1)$ being a linear structure duad of $E^{(r_1)}$ is equivalent to $(\Delta x_1, \Delta y_1)$ being a probability-1 differential

of $E^{(r_1)}$. Thus, all probability-1 differentials of $E^{(r_1)}$ must be in the set A_{r_1}. Similarly, all probability-1 differentials of ${E^{(r_2)}}^{-1}$ must be in the set B_{r_2}. Therefore, all impossible differentials linked by two differentials of probability-1 as in Figure 3 must be output by the FindImpoDiff algorithm. This holds even if the condition $\Theta(E^{(r)}) \leq e_0 < 1$ is not satisfied. The condition $\Theta(E^{(r)}) \leq e_0 < 1$ is used only to ensure that the probability of incorrectly outputting a vector that is not an impossible differential is exponentially small.

3.3. Complexity of Algorithm 2 (FindImpoDiff)

Since quantum computers have not yet been built, we cannot obtain actual execution results of the proposed quantum tools on specific block ciphers. The corresponding simulation would require, at a minimum, a polynomial quantum execution of a block cipher, which is currently too large for simulation. For quantum attacks, as actual running or simulation of the attacks is not yet possible, validation is often demonstrated by rigorously deriving the success probability and complexity [2,3,7,8]. Theorem 2 establishes the lower bound of the success probability for the FindImpoDiff algorithm. The process of Algorithm 2 (FindImpoDiff) does not involve quantum queries and, thus, can be executed by Q1 adversaries. We evaluate the complexity by calculating the number of qubits and quantum gates needed.

In Algorithm 2 (FindImpoDiff), finding probability-1 differentials of $E^{(r_1)}$ requires executing the FindStruct subroutine on $E^{(r_1)}$ for $c(m+2n)$ times. Each execution includes $2m+4n$ Hadamard gates, n CNOT gates, and one unitary operator $U_{E^{(r_1)}}$. Finding probability-1 differentials of ${E^{(r_2)}}^{-1}$ requires executing the FindStruct subroutine on ${E^{(r_2)}}^{-1}$ for $c(m+2n)$ times. Each execution includes $2m+4n$ Hadamard gates, n CNOT gates, and one unitary operator $U_{{E^{(r_2)}}^{-1}}$. Thus, the number of Hadamard gates in Algorithm 2 (FindImpoDiff) is as follows:

$$\begin{aligned}
&\sum_{\substack{r_1=1,\cdots,r-1\\ r_2=r-r_1}} [c(m+2n)(2m+4n) + c(2n+m)(4n+2m)] \\
&= \sum_{r_1=1}^{r-1} (4m+8n)c(2n+m) \\
&= 4c(r-1)(m^2+4n^2+4nm) \in O(n^2).
\end{aligned}$$

The number of CNOT gates in Algorithm 2 (FindImpoDiff) is as follows:

$$\begin{aligned}
&\sum_{\substack{r_1=1,\cdots,r-1\\ r_2=r-r_1}} [(2n+m)cn + (2n+m)cn] \\
&= \sum_{r_1=1}^{r-1} (4cn^2+2cmn) \\
&= 2(r-1)c(2n^2+mn) \in O(n^2).
\end{aligned}$$

Algorithm 2 (FindImpoDiff) also requires executing the unitary operators $U_{E^{(r_1)}}$ and $U_{{E^{(r_2)}}^{-1}}$ $c(m+2n)$ times for each $1 \leq r_1 \leq r-1, r_2 = r-r_1$. As explained in Section 2.1, $|U_{E^{(r_1)}}|$ and $|U_{{E^{(r_2)}}^{-1}}|$ denote the numbers of universal gates required to implement $U_{E^{(r_1)}}$ and $U_{{E^{(r_2)}}^{-1}}$, respectively. We have the following:

$$\begin{aligned}
&\sum_{\substack{r_1=1,\cdots,r-1\\ r_2=r-r_1}} c(m+2n)|U_{E^{(r_1)}}| + c(m+2n)|U_{E^{(r_2)^{-1}}}| \\
&= c(m+2n) \sum_{\substack{r_1=1,\cdots,r-1\\ r_2=r-r_1}} (|U_{E^{(r_1)}}| + |U_{E^{(r_2)}}|) \\
&= c(m+2n) \sum_{r_1=1}^{r-1} |U_{E^{(r)}}| \\
&= (r-1)c(m+2n)|U_{E^{(r)}}| \in O(poly(n)),
\end{aligned}$$

Algorithm 2 (FindImpoDiff) additionally requires executing the quantum circuit of $E^{(r)}$ for $(r-1)c(m+2n)$ times. The quantum resources used to implement Algorithm 2 (FindImpoDiff) are listed in Table 1.

Table 1. The quantum resources of Algorithm 2 (FindImpoDiff) and Algorithm 3 (FindImpoDiff2) [1].

Algorithm	#CNOT	#Hadamard	$U_{E^{(r)}}$
Algorithm 2 FindImpoDiff	$2\tau(2n^2+nm)$	$4\tau(m^2+4n^2+4mn)$	$\tau(m+2n)$
Algorithm 3 FindImpoDiff2	$2\tau(n^2+nm+n)$	$4\tau n(n+m+1)^2$	$\tau(1+n+m)$

[1] Here, n, m, and r are the block size, length of the key, and the number of rounds, respectively. c is the parameter chosen by the attacker, $\tau = c \cdot (r-1)$.

We then calculate the number of qubits needed for Algorithm 2 (FindImpoDiff). Running the FindStruct subroutine on $E^{(r_1)}$ requires $(m+n)+n+n = m+3n$ qubits. Running the FindStruct subroutine on ${E^{(r_2)}}^{-1}$ also requires $m+3n$ qubits. Due to the reusability of qubits, $m+3n$ qubits are sufficient to execute the FindImpoDiff algorithm.

In addition to the quantum computing part, Algorithm 2 (FindImpoDiff) also involves solving linear equations. Solving Equation (7) is equivalent to solving the following equation:

$$\begin{cases}
(\gamma_{m+1}^{(1)}, \gamma_{m+2}^{(1)}, \cdots, \gamma_{m+2n}^{(1)}) \cdot (x,y) = 0 \\
(\gamma_{m+1}^{(2)}, \gamma_{m+2}^{(2)}, \cdots, \gamma_{m+2n}^{(2)}) \cdot (x,y) = 0 \\
\vdots \\
(\gamma_{m+1}^{(c(m+2n))}, \gamma_{m+2}^{(c(m+2n))}, \cdots, \gamma_{m+2n}^{(c(m+2n))}) \cdot (x,y) = 0.
\end{cases}$$

Here, $\gamma_i^{(j)}$ denotes the i-th bit of $\gamma^{(j)}$ $(1 \le j \le c(m+2n))$. This linear system contains $2n$ unknowns and $c(m+2n)$ equations. The complexity of solving this linear system using Gaussian elimination is $O(cn^2(m+2n))$. Similarly, the complexity of solving Equation (8) is also $O(cn^2(m+2n))$. Thus, by omitting a constant coefficient, the complexity of the classical computing part of Algorithm 2 (FindImpoDiff) is as follows:

$$\begin{aligned}
&\sum_{\substack{r_1=1,\cdots,r-1\\ r_2=r-r_1}} [cn^2(m+2n) + cn^2(m+2n)] \\
=&2c(r-1)n^2(m+2n) \in O(n^3).
\end{aligned}$$

Therefore, the classical computing part has a complexity of $O(n^3)$.

Algorithm 3 Algorithm 2 (FindImpoDiff)

Input: a parameter c and a block cipher $E^{(r)} : \mathcal{K} \times \mathbb{F}_2^n \to \mathbb{F}_2^n$ where $\mathcal{K} = \mathbb{F}_2^m$.
Output: Impossible differentials of $E^{(r)}$.

1: **for** $r_1 = 1$ to $r - 1$ **do**
2: Let $r_2 = r - r_1$, divide $E^{(r)}$ into two parts $E^{(r)} = E^{(r_2)} \circ E^{(r_1)}$.
3: **for** $i = 1$ to n **do**
4: Run steps 1–7 of the FindStruct algorithm on $E^{(r_1)}[i] : \mathbb{F}_2^m \times \mathbb{F}_2^n \to \mathbb{F}_2$ with parameter c, obtaining $c(1+n+m)$ vectors $\gamma^{(1)}, \gamma^{(2)}, \cdots, \gamma^{(c(1+n+m))} \in \mathbb{F}_2^m \times F_2^n \times F_2^1$.
5: Solve the following equation:

$$\begin{cases} \gamma^{(1)} \cdot (k, x, y) = 0 \\ \gamma^{(2)} \cdot (k, x, y) = 0 \\ \vdots \\ \gamma^{(c(1+n+m))} \cdot (k, x, y) = 0 \\ k = 0^m, \end{cases} \tag{9}$$

where $(k, x, y) \in \mathbb{F}_2^{m+n+1}$ are unknowns, to obtain the solution set $A_{r_1}^i$.
6: Run steps 1–7 of the FindStruct algorithm on ${E^{(r_2)}}^{-1}[i] : \mathbb{F}_2^m \times \mathbb{F}_2^n \to \mathbb{F}_2$ with parameter c, obtaining $c(1+n+m)$ vectors $\tilde{\gamma}^{(1)}, \tilde{\gamma}^{(2)}, \cdots, \tilde{\gamma}^{(c(1+n+m))} \in \mathbb{F}_2^{m+n+1}$.
7: Solve the following equation:

$$\begin{cases} \tilde{\gamma}^{(1)} \cdot (k, x, y) = 0 \\ \tilde{\gamma}^{(2)} \cdot (k, x, y) = 0 \\ \vdots \\ \tilde{\gamma}^{(c(1+n+m))} \cdot (k, x, y) = 0 \\ k = 0^m, \end{cases} \tag{10}$$

where $(k, x, y) \in \mathbb{F}_2^{m+n+1}$ are unknowns, to obtain the solution set $B_{r_2}^i$.
8: **for** $(0^m, \Delta x_1, \Delta y_1) \in A_{r_1}^i$ **do**
9: **for** $(0^m, \Delta x_2, \Delta y_2) \in B_{r_2}^i$ **do**
10: **if** $\Delta x_1 \neq 0^n \wedge \Delta x_2 \neq 0^n \wedge \Delta y_1 \neq \Delta y_2$ **then**
11: Output $(\Delta x_1, \Delta x_2)$.
12: **end if**
13: **end for**
14: **end for**
15: **end for**
16: **end for**

4. Quantum Attacks Based on Truncated Differentials

For some block ciphers, it is almost impossible to identify a differential of which the probability is strictly 1. Thus, many attacks that have been proposed consider the truncated probability-1 differentials. For truncated differentials [31], only partial bits instead of the full differentials are certain. For many block ciphers, such as SAFERK64 [32] and Camellia [33], truncated differential analysis can attack more rounds than traditional differential analysis, or the attack complexity is greatly reduced when attacking the same number of rounds. In this section, we improve the FindImpoDiff algorithm by allowing the differentials with a probability of 1 to be truncated differentials.

4.1. Improved Algorithm for Impossible Differentials

To improve Algorithm 2 (FindImpoDiff), we allow the unmatched probability-1 differentials to be truncated differentials when applying the miss-in-the-middle method. That is, only partial bits of the probability-1 differentials are predicted.

Let (α, β) denote a truncated differential of $E^{(r)} : \mathcal{K} \times \mathbb{F}_2^n \to \mathbb{F}_2^n$, where $\alpha, \beta \in \{0, 1, *\}^n$. Suppose $\beta = (\beta_1, \beta_2, \cdots, \beta_n)$, $\alpha = (\alpha_1, \alpha_2, \cdots, \alpha_n)$, then $\beta_i, \alpha_i, \in \{0, 1, *\}$ for $i = 1, \cdots, n$. The notation "$*$" indicates that the corresponding bits of the input or output differences are undetermined. If $\alpha_i/\beta_i \in \{0, 1\}$, then we refer to the i-th bit (the determined bit of α/β), otherwise, we refer to it as the undetermined bit of α/β. A truncated difference can actually be regarded as a set of full differences. Let

$$\Omega_\alpha = \left\{\Delta x = (\Delta x_1, \Delta x_2, \cdots, \Delta x_n) \in \mathbb{F}_2^n \middle| \Delta x_i = \alpha_i \text{ if } \alpha_i \neq *, i = 1, 2, \cdots, n\right\},$$

$$\Omega_\beta = \left\{\Delta y = (\Delta y_1, \Delta y_2, \cdots, \Delta y_n) \in \mathbb{F}_2^n \middle| \Delta y_i = \beta_i \text{ if } \beta_i \neq *, i = 1, 2, \cdots, n\right\}.$$

The truncated input difference α is equivalent to the set Ω_α, and the truncated output difference β is equivalent to the set Ω_β. If a full input difference $\Delta x = (\Delta x_1, \cdots, \Delta x_n) \in \mathbb{F}_2^n$ is included in the set Ω_α; that is, $\Delta x_i = \alpha_i$ for all $i \in \{1, \cdots, n\}$ where $\alpha_i \neq *$, then Δx is said to coincide with the truncated input difference α, and denoted as $\Delta x \sim \alpha$. Similarly, if a full output difference Δy is in the set Ω_β, then Δy coincides with the truncated output difference β, and is denoted as $\Delta y \sim \beta$. Two truncated differentials, $\alpha, \alpha' \in \{0, 1, *\}^n$, are said to contradict each other if there exists an $i \in \{1, \ldots, n\}$ satisfying $\alpha_i \neq *$, $\alpha_i' \neq *$ and $\alpha_i \neq \alpha_i'$.

The probability of the truncated differential (α, β) is the conditional probability, expressed as follows:

$$\begin{aligned}\Pr_{\substack{k \leftarrow \mathcal{K} \\ x \leftarrow \mathbb{F}_2^n}}[\alpha \xrightarrow{E^{(r)}} \beta] &= \Pr_{\substack{k \leftarrow \mathcal{K} \\ x \leftarrow \mathbb{F}_2^n}}[E_k^{(r)}(x \oplus \Delta x) \oplus E_k^{(r)}(x) \sim \beta | \Delta x \sim \alpha] \\ &= \Pr_{\substack{k \leftarrow \mathcal{K} \\ x \leftarrow \mathbb{F}_2^n}}[E_k^{(r)}(x \oplus \Delta x) \oplus E_k^{(r)}(x) \in \Omega_\beta | \Delta x \in \Omega_\alpha].\end{aligned}$$

If this differential probability is equal to one, we refer to (α, β) as a probability-1 truncated differential of $E^{(r)}$.

We still divide $E^{(r)} = E^{(r_2)} \circ E^{(r_1)}$, where $r_1 + r_2 = r$. Let $E^{(r_1)}[i]$ be the i-th component function of $E^{(r_1)}$, as follows:

$$E_k^{(r_1)}(x) = (E_k^{(r_1)}[1](x), E_k^{(r_1)}[2](x), \cdots, E_k^{(r_1)}[n](x)).$$

Similarly, ${E^{(r_2)}}^{-1}[i]$ denotes the i-th component function of ${E^{(r_2)}}^{-1}$. If the truncated differential (α, β) of $E^{(r_1)}$ has probability-1, the truncated differential (α', β') of ${E^{(r_2)}}^{-1}$ also has probability-1, and β contradicts with β', then (α, α') will be $E^{(r)}$'s impossible differential. These conditions imply that there exists $i \in \{1, \cdots, n\}$ such that (α, β_i) and (α', β_i') are probability-1 differentials of $E^{(r_1)}[i]$ and ${E^{(r_2)}}^{-1}[i]$, respectively, and $\beta_i \neq \beta_i'$ ($\beta_i, \beta_i' \neq *$). In this case, (α, α') will be $E^{(r)}$'s impossible differential, as shown in Figure 5. Thus, we only need to traverse i to identify the differentials of $E^{(r_1)}[i]$ and ${E^{(r_2)}}^{-1}[i]$ with probability-1. This can be performed using the FindStruct algorithm to identify their linear structures. According to these analyses, we designed an improved algorithm that finds impossible differentials, which is named Algorithm 3 (FindImpoDiff2).

Figure 6 shows the flowchart of Algorithm 3 (FindImpoDiff2). Algorithm 3 (FindImpoDiff2) traverses r_1 from 1 to $r - 1$, dividing $E^{(r)}$ into $E^{(r)} = E^{(r_2)} \circ E^{(r_1)}$, and then traverses i from 1 to n to obtain differentials of $E^{(r_1)}[i]$ and ${E^{(r_2)}}^{-1}[i]$, which have probability-1 but lead to a contradiction in the middle. We define the following:

$$\bar{\Theta}(E^{(r)}) = \max\{\theta(E^{(t)}[i]) | 1 \leq t \leq r - 1, 1 \leq i \leq n\},$$

where $E^{(t)}$ is the t-round reduced cipher of $E^{(r)}$, $E^{(t)}[i]$ is the i-th component function of $E^{(t)}$, and $\theta(E^{(t)}[i])$ is defined as Equation (6). That is, we have the following:

$$\theta(E^{(t)}[i]) = \max_{\substack{(a_1,a_2)\in\mathbb{F}_2^m\times\mathbb{F}_2^n \\ b=0,1 \\ ((a_1,a_2),b)\notin L_{E^{(t)}[i]}}} \frac{1}{2^{m+n}}\Big|\{(k,x)\in\mathbb{F}_2^m\times\mathbb{F}_2^n | E^{(t)}[i](k,x)\oplus E^{(t)}[i](k\oplus a_1, x\oplus a_2) = b\}\Big|,$$

where $L_{E^{(t)}[i]}$ is the linear structure space of $E^{(t)}[i]$. According to Theorem 2, the following theorem holds.

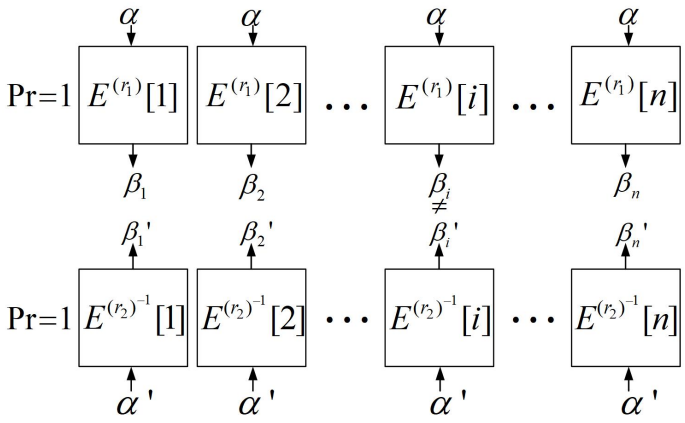

Figure 5. Illustration of how truncated differentials constitute an impossible differential.

Theorem 4. *$E^{(r)}$ is the block cipher whose round number is r, the block size is n, and the key length is m. $\bar{\Theta}(E^{(r)}) \le e_0 < 1$ denotes the constant e_0. If $(\Delta x_1, \Delta x_2)$ is output by Algorithm 3 (FindImpoDiff2) when running on $E^{(r)}$, then the probability of $(\Delta x_1, \Delta x_2)$ being an impossible differential of $E^{(r)}$ is no less than $1 - 2\left(2(\frac{1+e_0}{2})^c\right)^{m+n+1}$.*

According to Theorem 4, setting c greater than $3/(1-e_0)$ guarantees that Algorithm 3 (FindImpoDiff2) outputs impossible differentials of $E^{(r)}$.

4.2. Complexity of Algorithm 3 (FindImpoDiff2)

The process of the FindImpoDiff algorithm does not involve quantum queries and, thus, can be executed by Q1 adversaries. We still evaluate the complexity by calculating the amounts of qubits and quantum gates.

In Algorithm 3 (FindImpoDiff2), finding probability-1 differentials of $E^{(r_1)}[i]$ requires executing steps 1–7 of Algorithm 1 (FindStruct) on $E^{(r_1)}[i]$ with parameter c. This requires $2c(1+n+m)^2$ Hadamard gates, $c(1+n+m)$ CNOT gates, and $c(1+n+m)$ executions of the unitary operator $U_{E^{(r_1)}[i]}$. Similarly, finding probability-1 differentials of $E^{(r_2)^{-1}}$ requires $2c(1+n+m)^2$ Hadamard gates, $c(1+n+m)$ CNOT gates, and $c(1+n+m)$ executions of the unitary operator $U_{E^{(r_2)^{-1}}[i]}$. Thus, the number of Hadamard gates in Algorithm 3 (FindImpoDiff2) is as follows:

$$\begin{aligned}
&\sum_{\substack{r_1=1,\cdots,r-1 \\ r_2=r-r_1}} \sum_{i=1}^{n} [2c(1+n+m)^2 + 2c(1+n+m)^2] \\
&= \sum_{r_1=1}^{r-1} \sum_{i=1}^{n} [4c(1+n+m)^2] \\
&= 4(r-1)cn(1+n+m)^2 \in O(n^3).
\end{aligned}$$

The number of CNOT gates in Algorithm 3 (FindImpoDiff2) is as follows:

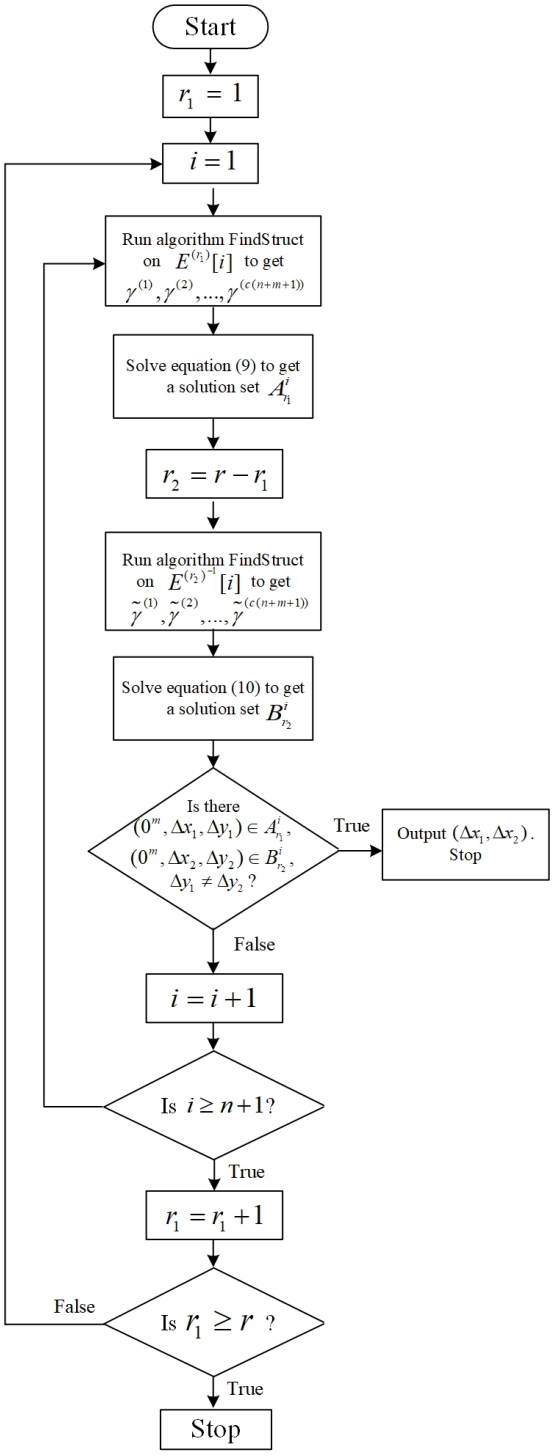

Figure 6. Flowchart of Algorithm 2 (FindImpoDiff).

$$\sum_{\substack{r_1=1,\cdots,r-1 \\ r_2=r-r_1}} \sum_{i=1}^{n} [c(1+n+m) + c(1+n+m)]$$
$$= 2(r-1)cn(1+n+m) \in O(n^2).$$

Algorithm 3 (FindImpoDiff2) also requires the execution of the unitary operators $U_{E^{(r_1)[i]}}$ and $U_{E^{(r_2)^{-1}}[i]}$ $c(m+n+1)$ times for each $1 \leq r_1 \leq r-1$, $r_2 = r - r_1$ and $1 \leq i \leq n$. We have the following:

$$\begin{aligned}
&\sum_{\substack{r_1=1,\cdots,r-1\\ r_2=r-r_1}} \sum_{i=1}^{n} \left[c(m+n+1)|U_{E^{(r_1)}[i]}| + c(1+n+m)|U_{E^{(r_2)^{-1}}[i]}|\right] \\
&= c(1+n+m) \sum_{\substack{r_1=1,\cdots,r-1\\ r_2=r-r_1}} \sum_{i=1}^{n} \left(|U_{E^{(r_1)}[i]}| + \sum_{i=1}^{n} |U_{E^{(r_2)^{-1}}[i]}|\right) \\
&= c(1+n+m) \sum_{\substack{r_1=1,\cdots,r-1\\ r_2=r-r_1}} \left(|U_{E^{(r_1)}}| + |U_{E^{(r_2)}}|\right) \\
&= (r-1)c(1+n+m)|U_{E^{(r)}}| \in O(poly(n)),
\end{aligned}$$

Algorithm 3 (FindImpoDiff2) additionally requires executing the quantum circuit of $E^{(r)}$ for $(r-1)c(1+n+m)$ times. The quantum resources used to implement Algorithm 2 (FindImpoDiff) are listed in Table 1.

Running the FindStruct subroutine on $E^{(r_1)}[i]$ requires $(m+n)+1+1 = m+n+2$ qubits. Running the FindStruct subroutine on $E^{(r_2)^{-1}}[i]$ also requires $m+n+2$ qubits. Because of the reusability of qubits, $m+n+2$ qubits are enough for Algorithm 3 (FindImpoDiff2).

We compare classical automated tools to search for impossible differentials with the proposed quantum tools in Table 2, where the SAT problem is the Boolean satisfiability problem and MILP is the abbreviation for mixed-integer linear programming. As shown in Table 2, the UID tool, U-tool, and WW-tool cannot characterize the impact of S-boxes. The MILP tool can only describe small S-boxes. The SAT tool can describe big S-boxes but the impact of S-boxes can only be partially characterized. In contrast, the quantum algorithms we propose can fully characterize nonlinear functions, including S-boxes of any size. The MILP tool and SAT tool need to solve the MILP problem and SAT problem, respectively, both of which are NP-complete. In contrast, the proposed quantum algorithms only need to solve linear equations, which cost polynomial complexity using Gaussian elimination.

It is difficult to directly compare the complexity of classical tools for searching impossible differentials with quantum tools on specific block ciphers. The search for impossible differentials does not constitute a complete attack in itself. The identified impossible differentials are used to eliminate incorrect keys during the subsequent key recovery phase. The complexity of a classical attack is generally measured by the number of calculations and the number of chosen plaintexts required in the online key recovery phase, rather than by the calculations or time needed to identify the distinguishers in the early phase. In fact, the practicality of classical tools for searching impossible differentials on specific ciphers is determined by whether the search can be completed within a reasonable timeframe, rather than by adhering to a strict complexity formula. For example, the authors of [34] show that searching for impossible differentials of 14-round PRESENT-80 can be accomplished, but the search for 15 rounds could not be completed even after 20 days, leading the authors to finally terminate the search for 15 rounds. Classical automated tools for searching various distinguishers are usually compared using the number of rounds of specific ciphers that can be searched. However, in quantum attacks, since quantum computers have not been built yet, we cannot obtain the actual execution results of the quantum tools. Thus in quantum attacks, the success probability and complexity (the number of qubits and gates) are rigorously computed to measure the efficiency. Although the cost of a quantum operation is different from that of a classical operation, it is generally believed that polynomial-level quantum complexity is far superior to exponential classical complexity. This is why Shor's algorithm has attracted widespread attention and even led to research on post-quantum cryptography.

Block cipher Camellia has 24 rounds [35]. The longest impossible differential of Camellia found so far has eight rounds [36]. It is found through theoretical derivation and does not involve any construction of S-boxes. Since the scale of the S-boxes in Camellia is 8-bit, no classical automated tool has yet been able to find longer impossible differentials by characterizing the S-boxes. AES-128 cipher has 10 rounds [37]. By the partial description of S-boxes, the SAT tool can accomplish a five-round search for impossible differentials of

AES-128 [38]. However, the full description of S-boxes or the search for more rounds cannot be realized in an affordable time. Block cipher Midori-128 has 20 rounds [39]. The S-boxes applied by Midori-128 have eight bits. But each eight-bit S-box is actually constructed by two four-bit S-boxes. By the full description of four-bit S-boxes, the MILP tool can accomplish an eight-round search for impossible differentials of Midori-128 [40]. The search for more rounds cannot be realized in an affordable time. From the above examples, it can be seen that classical automated tools have difficulty increasing the number of rounds since they need to solve problems like MILP or SAT. However, our quantum tools only require polynomial complexity when applied to all the block ciphers mentioned above, even if the S-boxes are fully characterized. Moreover, as shown in Table 1, the number of qubits and gates required increases linearly with the number of rounds. This shows the superiority of quantum tools.

Some prior works have also investigated the use of quantum algorithms to enhance classical analytic tools. We list some representative examples in Table 3 for comparison. The goals of the second and third quantum tools in Table 3 are not to identify differentials or linear approximations, but to use Grover's algorithm to accelerate the search for subkeys in the key recovery stage of traditional differential analysis and linear analysis.

Table 2. Comparisons with classical automated tools.

Tools	Description of Nonlinear Components	Problems Need to Solve	Ref.
UID tool	No	/	[21]
U-tool	No	/	[22]
WW-tool	No	Linear equations system	[23]
MILP-tool	Full description of small S-boxes ($\leq$ 6-bit)	MILP problem	[40,41]
SAT-tool	Partial description of big S-boxes (8-bit)	SAT problem	[38]
Quantum tools	Any nonlinear functions	Linear equations system	This paper

Table 3. Quantum cryptanalysis tools.

Cryptanalysis Tool to Be Enhanced	Goal of Quantum Algorithms	Underlying Quantum Algorithm Used	Ref.
Differential cryptanalysis	Find high-probability differentials	Bernstein–Vazirani algorithm	[42,43]
Differential cryptanalysis	Accelerate the key search	Grover's algorithm	[18]
Linear cryptanalysis	Accelerate the key search	Grover's algorithm	[18]
Zero correlation linear cryptanalysis	Find zero correlation linear approximations	Bernstein–Vazirani algorithm	[44]
Impossible differential cryptanalysis	Find impossible differentials	Simon's algorithm	This paper

4.3. Simulation

We use a simple Boolean function as an example to simulate the process of using the FindStruct algorithm to identify linear structure duads (i.e., probability-1 differentials). This demonstrates the validity of the FindStruct algorithm. The main process of Algorithm 2 (FindImpoDiff) involves repeating the FindStruct algorithm multiple times. According to the principle of miss-in-the-middle, as long as the outputs of the FindStruct algorithm are indeed probability-1 differentials, then the outputs of Algorithm 2 (FindImpoDiff) must be impossible differentials. Therefore, the validity of Algorithm 2 (FindImpoDiff) is also justified.

Specifically, we choose function $F : \mathbb{F}_2^4 \to \mathbb{F}_2$ as an example, whose truth table is shown in Table 4. We use Qiskit to simulate the FindStruct algorithm on F. This requires repeating the FindStruct subroutine to obtain measurement results $\gamma_i's$. The corresponding quantum circuit diagram generated by Qiskit is shown in Figure 7. The middle part of the circuit realizes the unitary operator of F.

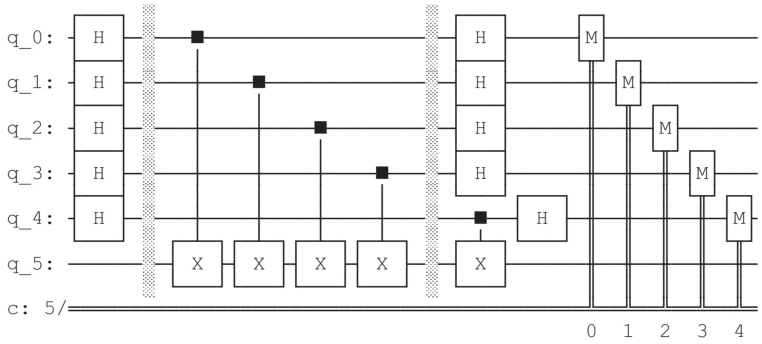

Figure 7. Quantum circuit diagram generated by Qiskit.

Table 4. Truth table of F.

x_1	x_2	x_3	x_4	F	x_1	x_2	x_3	x_4	F
0	0	0	0	0	1	0	0	0	1
0	0	0	1	1	1	0	0	1	0
0	0	1	0	1	1	0	1	0	0
0	0	1	1	0	1	0	1	1	1
0	1	0	0	1	1	1	0	0	0
0	1	0	1	0	1	1	0	0	0
0	1	1	0	0	1	1	0	0	0
0	1	1	1	1	1	1	0	0	0

Because the number of simulations for quantum algorithms is often set at 1024 or 2048, we chose to simulate 1024 times. As shown in Figure 8, the measurement results only yield two values: 00000 or 11111. Computing the following equation, i.e.,

$$\begin{cases} (00000) \cdot (x, y) = 0 \\ (11111) \cdot (x, y) = 0 \end{cases}$$

yields a fundamental solution system, as follows:

$$\begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix},$$

where $(x, y) \in \mathbb{F}_2^4 \times \mathbb{F}_2$ are the unknowns. Therefore, {(0000,0), (1100,0), (1010,0), (1001,0) (0110,0), (0011,0), (0101,0), (1111,0), (1000,1), (0100,1), (0010,1), (0001,1) (1110,1), (1011,1), (1101,1), (0111,1)} should all be probability-1 differentials (linear structure duals) of F. It is easy to verify that these pairs are indeed probability-1 differentials of F.

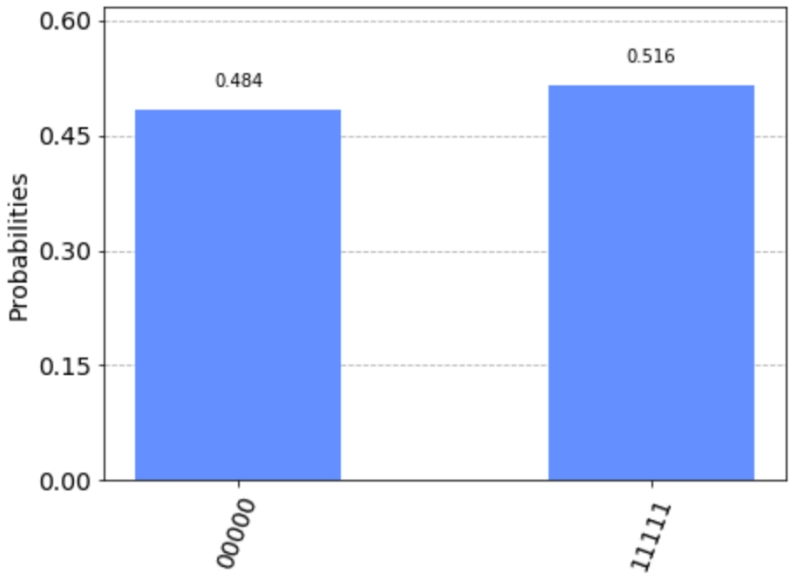

Figure 8. Measurement results simulated by Qiskit.

5. Results and Discussion

In this work, we developed quantum automated cryptanalysis tools to search for impossible differentials. Our tools combine the impossible differential attack with Simon's algorithm. We rigorously prove that, if an impossible differential of the block cipher is linked by two truncated differentials with probability-1, the proposed quantum algorithms must be able to output these impossible differentials in polynomial time. To analyze the complexity, we calculate the number of various quantum operators required in the quantum algorithms and summarized them in Table 1. Since the round number, r, as shown in each dataset in Table 1, has a degree of 1, the complexity of the proposed algorithms increases linearly with the number of rounds.

Our quantum cryptanalysis tools do not require any query to the encryption or decryption oracle and can be implemented in the Q1 model. Thus, they can be easily realized using quantum computers. Compared with the classical automated impossible differential cryptanalysis, our quantum automated tools fully utilize the superiority of quantum computing, allowing for complete characterization of any nonlinear functions while maintaining complexity within polynomial time. Moreover, the proposed quantum cryptanalysis tools take the key schedule into account in a single-key model, thereby compensating for the shortcomings of traditional tools.

A natural direction for further research is to enhance the cryptanalysis tools proposed in this study to reduce quantum resource consumption, or to enhance other cryptanalysis methods with quantum algorithms, such as integral and linear attacks. Combining cryptanalysis tools with Grover's algorithm may also be a direction worth exploring.

Author Contributions: Conceptualization, H.X. and Q.X.; methodology, H.X., Q.X., K.W. and L.Y.; validation, H.X. and Y.L.; formal analysis, H.X., K.W. and L.Y.; writing—original draft preparation, H.X. and Q.X.; writing—review and editing, H.X. and Y.L.; visualization, H.X. and K.W.; funding acquisition, H.X. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the Beijing Natural Science Foundation (no. 4234084) and the Open Research Fund of Key Laboratory of Cryptography of Zhejiang Province (no. ZCL21012).

Data Availability Statement: The original contributions presented in this study are included in the article; further inquiries can be directed to the corresponding author.

Conflicts of Interest: Author L.Y. was employed by the company Information Industry Information Security Evaluation Center, The 15th Research Institute of China Electronics Technology Group Corporation. The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest. The authors declare that this study received funding from Beijing Natural Science Foundation (no. 4234084). The funder had the following involvement with the study: Conceptualization, methodology, validation, formal analysis, writing—original draft preparation, writing—review and editing, visualization.

Appendix A

Theorem A1. *Let L be the solution set output by the FindStruct algorithm run on $F : \mathbb{F}_2^n \to \mathbb{F}_2^m$ with a parameter, c, then $L_F \subseteq L$. Moreover, if there is a constant, e_0, such that $\theta(F) \leq e_0 < 1$, then the probability of L_F being equal to L is no less than $1 - \left(2(\frac{1+e_0}{2})^c\right)^{n+m}$.*

Proof. We first prove that $L_F \subseteq L$. To do this, we only need to prove that each linear structure $(a, b) \in L_F$ must be a solution to Equation (5). In step 5 of the FindStruct algorithm, the state

$$|\Psi_3\rangle = \frac{1}{\sqrt{2^{n+m}}} \sum_{x \in \mathbb{F}_2^n, y \in \mathbb{F}_2^m} |x\rangle|y\rangle|F(x) \oplus y\rangle$$

is measured and the measurement result is denoted as z. Then there are $x_0 \in \mathbb{F}_2^n$, $y_0 \in \mathbb{F}_2^m$ such that $F(x_0) \oplus y_0 = z$. We define the set, i.e.,

$$S_z = \left\{(x, y) \in \mathbb{F}_2^n \times \mathbb{F}_2^m | F(x) \oplus y = z\right\},$$

then the first two registers of $|\Psi_3\rangle$ are collapsed to the state $\frac{1}{\sqrt{|S_z|}} \sum_{(x,y) \in S_z} |x\rangle|y\rangle$. Obviously, $(x_0, y_0) \in S_z$. For any $(x, y) \in \mathbb{F}_2^{n+m}$, let $a = x_0 \oplus x, b = y_0 \oplus y$, then $y = b \oplus y_0$, $x = a \oplus x_0$ and

$$\begin{aligned}
&F(x) \oplus y = z \\
\Leftrightarrow &(b \oplus y_0) \oplus F(a \oplus x_0) = z \\
\Leftrightarrow &F(a \oplus x_0) \oplus b \oplus y_0 = y_0 \oplus F(x_0) \\
\Leftrightarrow &F(x_0) \oplus F(x_0 \oplus a) = b \\
\Leftrightarrow &(a, b) \text{ causes a } \textit{match} \text{ of } F \text{ at point } x_0.
\end{aligned}$$

Thus,

$$S_z = \left\{(x_0 \oplus a, y_0 \oplus b) \middle| (a, b) \in \mathbb{F}_2^{n+m} \text{ and } (a, b) \text{ causes a } \textit{match} \text{ of } F \text{ at } x_0\right\}.$$

Since the linear structure duads of F cause *matches* of F at all points $x \in \mathbb{F}_2^n$, for each $(a, b) \in L_F$, $(x_0 \oplus a, y_0 \oplus b)$ is in S_z. Suppose that $\{(a_1, b_1), (a_2, b_2), \cdots, (a_t, b_t)\}$ is the basis of space L_F, then for any $k_1, k_2, \cdots, k_t \in \{0, 1\}$, we have the following:

$$(k_1 a_1 \oplus k_2 a_2 \oplus ... \oplus k_t a_t, k_1 b_1 \oplus k_2 b_2 \oplus ... \oplus k_t b_t) \in L_F.$$

Therefore,

$$(x_0 \oplus \bigoplus_{j=1}^{t} k_j a_j, y_0 \oplus \bigoplus_{j=1}^{t} k_j b_j) \in S_z.$$

In addition to linear structures, there may be other vectors causing a *match* at x_0. Let $(\hat{a}, \hat{b})$ denote such vectors. Namely, $F(x_0 \oplus \hat{a}) \oplus (y_0 \oplus \hat{b}) = z$ but $(\hat{a}, \hat{b}) \notin L_F$. Thus, for any $k_1, \cdots, k_t \in \{0,1\}$,

$$F(x_0 \oplus \hat{a} \oplus \bigoplus_{j=1}^{t} k_j a_j) \oplus (y_0 \oplus \hat{b} \oplus \bigoplus_{j=1}^{t} k_j b_j) = F(x_0 \oplus \hat{a}) \oplus (y_0 \oplus \hat{b}) = z$$

So, $(x_0 \oplus \hat{a} \oplus \bigoplus_{j=1}^{t} k_j a_j, y_0 \oplus \hat{b} \oplus \bigoplus_{j=1}^{t} k_j b_j)$ is also in the set S_z. Suppose $(\hat{a}_1, \hat{b}_1), (\hat{a}_2, \hat{b}_2)$ are two of these vectors, i.e., they both cause a *match* at x_0 but are not in L_F. Since L_F is a linear space, the following two sets, i.e.,

$$\Big\{(x_0 \oplus \hat{a}_1 \oplus \bigoplus_{j=1}^{t} k_j a_j, y_0 \oplus \hat{b}_1 \oplus \bigoplus_{j=1}^{t} k_j b_j) \Big| k_1 \cdots, k_t \in \{0,1\}\Big\}$$

$$\Big\{(x_0 \oplus \hat{a}_2 \oplus \bigoplus_{j=1}^{t} k_j a_j, y_0 \oplus \hat{b}_2 \oplus \bigoplus_{j=1}^{t} k_j b_j) \Big| k_1, \cdots, k_t \in \{0,1\}\Big\}$$

are either equal or have no intersection at all. Therefore, the measurement in step 5 of the FindStruct algorithm causes the first two registers of $|\Psi_3\rangle$ to collapse into a state with the following form:

$$\begin{aligned}
&\frac{1}{\sqrt{2^t(l+1)}}\Big(\sum_{k_1,\cdots,k_t\in\{0,1\}} |x_0 \oplus \bigoplus_{j=1}^{t} k_j a_j\rangle |y_0 \oplus \bigoplus_{j=1}^{t} k_j b_j\rangle \\
&+ \sum_{k_1,\cdots,k_t\in\{0,1\}} |x_0 \oplus \hat{a}_1 \oplus \bigoplus_{j=1}^{t} k_j a_j\rangle |y_0 \oplus \hat{b}_1 \oplus \bigoplus_{j=1}^{t} k_j b_j\rangle \\
&+ \cdots \\
&+ \sum_{k_1,\cdots,k_t\in\{0,1\}} |x_0 \oplus \hat{a}_l \oplus \bigoplus_{j=1}^{t} k_j a_j\rangle |y_0 \oplus \hat{b}_l \oplus \bigoplus_{j=1}^{t} k_j b_j\rangle\Big),
\end{aligned}$$

where $(\hat{a}_1, \hat{b}_1), \cdots, (\hat{a}_l, \hat{b}_l)$ are the vectors that cause *matches* at x_0 but not in L_F. Denoting $\hat{a}_0 = 0^n$, $\hat{b}_0 = 0^m$, this state can be written as follows:

$$\frac{1}{\sqrt{2^t(l+1)}} \sum_{\substack{i\in\{0,1,\cdots,l\}\\ k_1,\cdots,k_t\in\{0,1\}}} |x_0 \oplus \hat{a}_i \oplus \bigoplus_{j=1}^{t} k_j a_j\rangle |y_0 \oplus \hat{b}_i \oplus \bigoplus_{j=1}^{t} k_j b_j\rangle.$$

In step 6, after performing a Hadamard gate $H^{\otimes(n+m)}$, the first two registers are as follows:

$$\begin{aligned}
&\sum_{\substack{i\in\{0,\cdots,l\}\\ k_1,\cdots,k_t\in\{0,1\}\\ (\gamma_1,\gamma_2)\in\mathbb{F}_2^{n+m}}} (-1)^{\gamma_1\cdot(x_0\oplus\hat{a}_i\oplus\bigoplus_{j=1}^{t} k_j a_j)+\gamma_2\cdot(y_0\oplus\hat{b}_i\oplus\bigoplus_{j=1}^{t} k_j b_j)} |\gamma_1\rangle|\gamma_2\rangle \\
=&\sum_{\substack{\gamma_1\in\mathbb{F}_2^n\\ \gamma_2\in\mathbb{F}_2^m}} \Big(\sum_{i=0}^{l} (-1)^{\gamma_1\cdot(x_0\oplus\hat{a}_i)\oplus\gamma_2\cdot(y_0\oplus\hat{b}_i)}\Big)\big[(-1)^{(\gamma_1,\gamma_2)\cdot(a_1,b_1)}+1\big]\times \\
&\big[(-1)^{(\gamma_1,\gamma_2)\cdot(a_2,b_2)}+1\big]\times\cdots\times\big[(-1)^{(\gamma_1,\gamma_2)\cdot(a_t,b_t)}+1\big]|\gamma_1,\gamma_2\rangle,
\end{aligned}$$

where we omit the global coefficient $1/\sqrt{2^{n+m+t}(l+1)}$. For any $(\gamma_1, \gamma_2) \in \mathbb{F}_2^{n+m}$, if there exists $j \in \{1, 2, \cdots, t\}$ such that $(\gamma_1, \gamma_2) \cdot (a_j, b_j) \neq 0$, then the amplitude of $|\gamma_1, \gamma_2\rangle$ in the above quantum state is zero. Therefore, the FindStruct algorithm always outputs a

random value $\gamma \in \mathbb{F}_2^{n+m}$ such that $\gamma \cdot (a_j, b_j) = 0$ for all $j = 1, 2, \cdots, t$, which means $(a_1, b_1), \cdots, (a_t, b_t)$ must be in the solution space L of Equation (5). Thus, we have $L_F \subseteq L$.

Then, we prove the probability that $L_F = L$ is no less than $1 - 2((\frac{1+e_0}{2})^c)^{n+m}$ as long as $\theta(F) \leq e_0 < 1$. $L_F \neq L$ means that there is a vector (a, b), which is a solution to Equation (5) but not in L_F. Thus, we have the following:

$$\begin{aligned}
&\Pr\big[L_F \neq L\big] \\
=&\Pr\big[\exists (a,b) \notin L_F, s.t.,\ \gamma^{(1)} \cdot (a,b) = \gamma^{(2)} \cdot (a,b) = \cdots = \gamma^{(c(n+m))} \cdot (a,b) = 0\big] \\
\leq& \sum_{(a,b) \in \mathbb{F}_2^{n+m} \setminus L_F} \Pr\big[\gamma^{(1)} \cdot (a,b) = \cdots = \gamma^{(c(n+m))} \cdot (a,b) = 0\big] \\
=& \sum_{(a,b) \in \mathbb{F}_2^{n+m} \setminus L_F} \big(\Pr[\gamma^{(1)} \cdot (a,b) = 0]\big)^{c(n+m)} \\
\leq& (2^{n+m} - |L_F|) \max_{(a,b) \in \mathbb{F}_2^{n+m} \setminus L_F} \big(\Pr[\gamma^{(1)} \cdot (a,b) = 0]\big)^{c(n+m)} \\
\leq& \max_{(a,b) \in \mathbb{F}_2^{n+m} \setminus L_F} \Big(2\Pr[\gamma^{(1)} \cdot (a,b) = 0]^c\Big)^{n+m},
\end{aligned} \tag{A1}$$

where $\gamma^{(1)}, \cdots, \gamma^{(c(n+m))}$ are $c(m+n)$ outputs of the FindStruct subroutine and are independent and identically distributed random variables. To calculate the probability $\Pr[\gamma^{(1)} \cdot (a,b) = 0]$, all measurements of the FindStruct subroutine are moved to the end. According to the principle of deferred measurement, this does not change the outputs. Therefore, the state without being measured is as follows:

$$\begin{aligned}
&\frac{1}{2^{n+m}} \sum_{\substack{x \in \mathbb{F}_2^n \\ y \in \mathbb{F}_2^m}} \sum_{\substack{\gamma_1 \in \mathbb{F}_2^n \\ \gamma_2 \in \mathbb{F}_2^m}} (-1)^{x \cdot \gamma_1 \oplus y \cdot \gamma_2} |\gamma_1\rangle |\gamma_2\rangle |F(x) \oplus y\rangle \\
=&\frac{1}{2^{n+m}} \sum_{\substack{(\gamma_1,\gamma_2) \in \mathbb{F}_2^{n+m} \\ (\gamma_1,\gamma_2) \cdot (a,b) = 0}} \sum_{\substack{x \in \mathbb{F}_2^n \\ y \in \mathbb{F}_2^m}} (-1)^{x \cdot \gamma_1 \oplus y \cdot \gamma_2} |\gamma_1\rangle |\gamma_2\rangle |F(x) \oplus y\rangle \\
&+ \frac{1}{2^{n+m}} \sum_{\substack{(\gamma_1,\gamma_2) \in \mathbb{F}_2^{n+m} \\ (\gamma_1,\gamma_2) \cdot (a,b) = 1}} \sum_{\substack{x \in \mathbb{F}_2^n \\ y \in \mathbb{F}_2^m}} (-1)^{x \cdot \gamma_1 \oplus y \cdot \gamma_2} |\gamma_1\rangle |\gamma_2\rangle |F(x) \oplus y\rangle.
\end{aligned}$$

The probability of $\gamma^{(1)}$ satisfying $(a,b) \cdot \gamma^{(1)} = 0$ is as follows:

$$\begin{aligned}
&\Pr[(a,b) \cdot \gamma^{(1)} = 0] \\
=&\Big\| \frac{1}{2^{n+m}} \sum_{\substack{(\gamma_1,\gamma_2) \in \mathbb{F}_2^{n+m} \\ (\gamma_1,\gamma_2) \cdot (a,b) = 0}} \sum_{\substack{x \in \mathbb{F}_2^n \\ y \in \mathbb{F}_2^m}} (-1)^{x \cdot \gamma_1 \oplus y \cdot \gamma_2} |\gamma_1\rangle |\gamma_2\rangle |F(x) \oplus y\rangle \Big\|^2 \\
=&\frac{1}{2^{2(n+m)}} \sum_{\substack{(\gamma_1,\gamma_2) \cdot (a,b) = 0 \\ x,x' \in \mathbb{F}_2^n \\ y,y' \in \mathbb{F}_2^m}} (-1)^{\gamma_1 \cdot (x \oplus x') \oplus \gamma_2 \cdot (y \oplus y')} \langle F(x') \oplus y' | F(x) \oplus y\rangle \\
=& \sum_{\substack{x,x' \in \mathbb{F}_2^n \\ y,y' \in \mathbb{F}_2^m}} \Big(\frac{\langle y \oplus F(x) | y' \oplus F(x')\rangle}{2^{2(n+m)}} \sum_{(\gamma_1,\gamma_2) \cdot (a,b) = 0} (-1)^{\gamma_1 \cdot (x' \oplus x) \oplus \gamma_2 \cdot (y' \oplus y)} \Big).
\end{aligned}$$

According to Lemma 1 in [7], the following holds:

$$\frac{1}{2^{n+m}} \sum_{\substack{(\gamma_1,\gamma_2) \in \mathbb{F}_2^{n+m} \\ (\gamma_1,\gamma_2) \cdot (a,b) = 0}} (-1)^{\gamma_1 \cdot x + \gamma_2 \cdot y} = \frac{1}{2} (\delta_{(0^n,0^m),(x,y)} + \delta_{(a,b),(x,y)}).$$

Thus,

$$\begin{aligned}
&\Pr[(a,b)\cdot\gamma^{(1)}=0]\\
=&\sum_{\substack{(x,y)\in\mathbb{F}_2^{n+m}\\(x',y')\in\mathbb{F}_2^{n+m}}}\frac{\langle y\oplus F(x)|y'\oplus F(x')\rangle}{2^{n+m+1}}\Big(\delta_{(x,y),(x',y')}+\delta_{(x,y),(a\oplus x',b\oplus y')}\Big).\\
=&\frac{1}{2^{n+m+1}}\sum_{(x,y)\in\mathbb{F}_2^{n+m}}\Big(1+\langle F(x)\oplus y|F(x\oplus a)\oplus y\oplus b\rangle\Big).\\
=&\frac{1}{2}\big(1+\frac{1}{2^{n+m}}|\{(x,y)|F(x\oplus a)\oplus F(x)=b\}|\big)\\
=&\frac{1}{2}\big(1+\Pr_x[F(x\oplus a)\oplus F(x)=b]\big)\\
\leq&\frac{1}{2}\Big(1+\theta(F))\Big).
\end{aligned}$$

Combining Equation (A1), we have the following:

$$\begin{aligned}
&\Pr[L_F\neq L]\\
\leq&\max_{(a,b)\in\mathbb{F}_2^{m+n}\setminus L_F}\Big(2\Pr[(a,b)\cdot\gamma^{(1)}=0]^c\Big)^{n+n}\\
\leq&\Big(2(\frac{1+\theta(F)}{2})^c\Big)^{m+n}\\
\leq&\Big(2(\frac{1+e_0}{2})^c\Big)^{m+n}.
\end{aligned}$$

□

References

1. Shor, P.W. Algorithms for quantum computation: Discrete logarithms and factoring. In Proceedings of the 35th Annual Symposium on Foundations of Computer Science, Santa Fe, NM, USA, 20–22 November 1994; pp. 124–134.
2. Grover, L.K. A fast quantum mechanical algorithm for database search. In Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing, Philadelphia, PA, USA, 22–24 May 1996; pp. 212–219.
3. Simon, D.R. On the power of quantum computation. *SIAM J. Comput.* **1997**, *10*, 1474–1483. [CrossRef]
4. Kuwakado, H.; Morii, M. Quantum distinguisher between the 3-round Feistel cipher and the random permutation. In Proceedings of the IEEE International Symposium on Information Theory, Austin, TX, USA, 13–18 June 2010; pp. 2682–2685.
5. Kuwakado, H.; Morii, M. Security on the quantum-type Even-Mansour cipher. In Proceedings of the Information Theory and Its Applications, Honolulu, HI, USA, 28–31 October 2012; pp. 312–316.
6. Santoli, T.; Schaffner, C. Using Simon's algorithm to attack symmetric-key cryptographic primitives. *Quantum Inf. Comput.* **2017**, *17*, 65–78. [CrossRef]
7. Kaplan, M.; Leurent, G.; Leverrier, A.; Naya-Plasencia, M. Breaking symmetric cryptosystems using quantum period finding. In Proceedings of the Advances in Cryptology—CRYPTO 2016, Santa Barbara, CA, USA, 14–18 August 2016; Springer: Berlin/Heidelberg, Germany, 2016; pp. 207–237.
8. Leander, G.; May, A. Grover Meets Simon–Quantumly Attacking the FX-construction. In Proceedings of the Advances in Cryptology—ASIACRYPT 2017, Hong Kong, China, 3–7 December 2017; Springer: Cham, Switzerland, 2017; pp. 161–178.
9. Dong, X.; Wang, X. Quantum key-recovery attack on Feistel structures. *Sci. China Inf. Sci.* **2018**, *10*, 240–246. [CrossRef]
10. Dong, X.; Wang, X. Quantum cryptanalysis on some generalized Feistel schemes. *Sci. China Inf. Sci.* **2019**, *62*, 22501:1–22501:12. [CrossRef]
11. Damgård, I.; Funder, J.; Nielsen, J.B.; Salvail, L. Superposition attacks on cryptographic protocols. In Proceedings of the International Conference on Information Theoretic Security, Cham, Switzerland, 28–30 November 2013; pp. 142–161.
12. Boneh, D.; Zhandry, M. Secure signatures and chosen ciphertext security in a quantum computing world. In Proceedings of the Advances in Cryptology—CRYPTO 2013, Santa Barbara, CA, USA, 18–22 August 2013; Springer: Berlin/Heidelberg, Germany, 2013; pp. 361–379.
13. Gagliardoni, T.; Hlsing, A.; Schaffner, C. Semantic security and indistinguishability in the quantum world. In Proceedings of the Advances in Cryptology—CRYPTO 2016, Santa Barbara, CA, USA, 14–18 August 2016; Springer: Berlin/Heidelberg, Germany, 2016; pp. 60–89.
14. Roetteler, M.; Steinwandt, R. A note on quantum related-key attacks. *Inf. Process. Lett.* **2015**, *115*, 40–44. [CrossRef]

15. Hosoyamada, A.; Aoki, K. On quantum related-key attacks on iterated Even-Mansour ciphers. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.* **2019**, *102*, 27–34. [CrossRef]
16. Jaques, S.; Naehrig, M.; Roetteler, M.; Virdia, F. Implementing Grover Oracles for Quantum Key Search on AES and LowMC. In Proceedings of the Advances in Cryptology—EUROCRYPT 2020, Zagreb, Croatia, 10–14 May 2020; Springer: Cham, Switzerland, 2020; pp. 280–310.
17. Zhou, Q.; Lu, S.; Zhang, Z.; Sun, J. Quantum differential cryptanalysis. *Quantum Inf. Process.* **2015**, *14*, 2101–2109. [CrossRef]
18. Kaplan, M.; Leurent, G.; Leverrier, A.; Naya-Plasencia, M. Quantum differential and linear cryptanalysis. *IACR Trans. Symmetric Cryptol.* **2016**, *2016*, 71–94. [CrossRef]
19. Hosoyamada, A.; Sasaki, Y. Finding Hash Collisions with Quantum Computers by Using Differential Trails with Smaller Probability than Birthday Bound. In Proceedings of the Advances in Cryptology—EUROCRYPT 2020, Zagreb, Croatia, 10–14 May 2020; Springer: Cham, Switzerland, 2020; pp. 249–279.
20. Dong, X.; Sun, S.; Shi, D.; Gao, F.; Wang, X.; Hu, L. Quantum Collision Attacks on AES-Like Hashing with Low Quantum Random Access Memories. In Proceedings of the Advances in Cryptology—ASIACRYPT 2020, Daejeon, Republic of Korea, 7–11 November 2020; Springer: Cham, Switzerland, 2020; pp. 727–757.
21. Luo, Y.; Lai, X.; Wu, Z.; Gong, G. A unified method for finding impossible differentials of block cipher structures. *Inf. Sci.* **2014**, *263*, 211–220. [CrossRef]
22. Kim, J.; Hong, S.; Sung, J.; Lee, S.; Lim, J.; Sung, S. Impossible differential cryptanalysis for block cipher structures. In Proceedings of the 4th International Conference on Cryptology, New Delhi, India, 8–10 December 2003.
23. Wu, S.; Wang, M. Automatic search of truncated impossible differentials for word oriented block ciphers. In Proceedings of the Progress in Cryptology—INDOCRYPT 2012, Kolkata, India, 9–12 December 2012; Springer: Berlin/Heidelberg, Germany, 2012; pp. 283–302.
24. Liu, Y.; Xiang, Z.; Chen, S.; Zhang, S.; Zeng, X. A Novel Automatic Technique Based on MILP to Search for Impossible Differentials. In Proceedings of the Applied Cryptography and Network Security (ACNS 2023), Kyoto, Japan, 19–22 June 2023; Springer: Cham, Switzerland, 2023; pp. 119–148.
25. Sun, L.; Wang, M. SoK: Modeling for large s-boxes oriented to differential probabilities and linear correlations. *IACR Trans. Symmetric Cryptol.* **2023**, *2023*, 111–151. [CrossRef]
26. Winternitz, R.; Hellman, M. Chosen-key attacks on a block cipher. *Cryptologia* **1987** *11*, 16–20. [CrossRef]
27. Xiang, Z.; Wang, X.; Yu, B.; Sun, B.; Zhang, S.; Zeng, X.; Shen, X.; Li, N. Links between Quantum Distinguishers Based on Simon's Algorithm and Truncated Differentials. *IACR Trans. Symmetric Cryptol.* **2024**, *2024*, 296–321. [CrossRef]
28. Nielsen, M.; Chuang, I. *Quantum Computation and Quantum Information*, 1st ed.; Cambridge University Press: Cambridge, MA, USA, 2000.
29. Li, H.; Yang, L. A quantum algorithm to approximate the linear structures of Boolean functions. *Math. Struct. Comput. Sci.* **2016**, *28* 1–13. [CrossRef]
30. Biham, E.; Biryukov, A.; Shamir, A. Cryptanalysis of Skipjack reduced to 31 rounds using impossible differentials. In Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques, Prague, Czech Republic, 2–6 May 1999; Springer: Berlin/Heidelberg, Germany, 1999; pp. 12–23.
31. Knudsen, L.R. Truncated and higher order differentials. In Proceedings of the Fast Software Encryption: Second International Workshop, Leuven, Belgium, 1–16 December 1994; Springer: Berlin/Heidelberg, Germany, 1994; pp. 196–211.
32. Wu, H.; Bao, F.; Deng, R.H.; Ye, Q. Improved truncated differential attacks on SAFER. In Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, 18–22 October 1998; Springer: Berlin/Heidelberg, Germany, 1998; pp. 133–147.
33. Kanda, M.; Matsumoto, T. Security of Camellia against truncated differential cryptanalysis. In Proceedings of the Fast Software Encryption: 8th International Workshop, Yokohama, Japan, 2–4 April 2001; Springer: Berlin/Heidelberg, Germany, 2001; pp. 286–299.
34. Sun, S.; Hu, L.; Wang, P.; Qiao, K.; Ma, X.; Song, L. Automatic security evaluation and (related-key) differential characteristic search: Application to SIMON, PRESENT, LBlock, DES (L) and other bit-oriented block ciphers. In Proceedings of the 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, 7–11 December 2014; Springer: Berlin/Heidelberg, Germany, 2014; pp. 158–178.
35. Aoki, K.; Ichikawa, T.; Kanda, M.; Matsui, M.; Moriai, S.; Nakajim, J.; Tokita, T. Camellia: A 128-bit block cipher suitable for multiple platforms—Design and analysis. In Proceedings of the 7th SAC, Selected Areas in Cryptography (SAC 2000), Waterloo, ON, Canada, 14–15 August 2000; Springer: Berlin/Heidelberg, Germany, 2012; pp. 39–56.
36. Jia, K.; Wang, N. Impossible differential cryptanalysis of 14-round camellia-192. In Proceedings of the 21st Australasian Conference on Information Security and Privacy, Melbourne, VIC, Australia, 4–6 July 2016; Springer: Cham, Switzerland, 2016; pp. 363–378.
37. Sanchez-Avila, C.; Sanchez-Reillol, R. The Rijndael block cipher (AES proposal): A comparison with DES. In Proceedings of the IEEE 35th Annual International Carnahan Conference on Security Technology, London, UK, 16–19 October 2001; pp. 229–234.
38. Hu, X.; Li, Y.; Jiao, L.; Tian, S.; Wang, M. Mind the propagation of states: New automatic search tool for impossible differentials and impossible polytopic transitions. In Proceedings of the 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, Republic of Korea, 7–11 December 2020; Springer: Cham, Switzerland, 2020; pp. 415–445.

39. Banik, S.; Bogdanov, A.; Isobe, T.; Shibutani, K.; Hiwatari, H.; Akishita, T.; Regazzoni, F. Midori: A block cipher for low energy. In Proceedings of the 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, 29 November–3 December 2015; Springer: Berlin/Heidelberg, Germany, 2015; pp. 411–436.
40. Sasaki, Y.; Todo, Y. New impossible differential search tool from design and cryptanalysis aspects: Revealing structural properties of several ciphers. In Proceedings of the 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, 30 April–4 May 2017; Springer: Cham, Switzerland, 2017; pp. 185–215.
41. Cui, T.; Chen, S.; Jia, K.; Fu, K.; Wang, M. New Automatic Search Tool for Impossible Differentials and Zero-Correlation Linear Approximations. Cryptology ePrint Archive, 2016. Available online: https://eprint.iacr.org/2016/689 (accessed on 1 May 2022).
42. Li, H.; Yang, L. Quantum differential cryptanalysis to the block ciphers. In Proceedings of the 6th International Conference on Applications and Techniques in Information Security, Beijing, China, 4–6 November 2015; Springer: Berlin/Heidelberg, Germany, 2015; pp. 44–51.
43. Xie, H.; Yang, L. Using Bernstein-Vazirani algorithm to attack block ciphers. *Des. Codes Cryptogr.* **2019**, *87*, 1161–1182. [CrossRef]
44. Zhang, K.; Shang, T.; Tang, Y.; Liu, J. Zero-correlation linear analysis for block ciphers based on the Bernstein-Vazirani and Grover algorithms. *Quantum. Inf. Process* **2024**, *23*, 289. [CrossRef]