Quantum Inf Process (2015) 14:813–829
DOI 10.1007/s11128-014-0909-5

Quantum algorithm to find invariant linear structure of *MD* hash functions

WanQing Wu · HuanGuo Zhang · ShaoWu Mao · HouZhen Wang

Received: 9 August 2014 / Accepted: 24 December 2014 / Published online: 7 January 2015

Abstract In this paper, we consider a special problem. "Given a function $f\colon \{0,1\}^n \rightarrow \{0,1\}^m$. Suppose there exists a n-bit string $\alpha \in \{0,1\}^n$ subject to $f(x \oplus \alpha) = f(x)$ for $\forall x \in \{0,1\}^n$. We only know the Hamming weight $W(\alpha) = 1$, and find this α." We present a quantum algorithm with "Oracle" to solve this problem. The successful probability of the quantum algorithm is $(\frac{2^l-1}{2^l})^{n-1}$, and the time complexity of the quantum algorithm is $O(\log(n-1))$ for the given Hamming weight $W(\alpha) = 1$. As an application, we present a quantum algorithm to decide whether there exists such an invariant linear structure of the MD hash function family as a kind of collision. Then, we provide some consumptions of the quantum algorithms using the time–space trade-off.

Keywords *MD* Hash functions · Invariant linear structure · Quantum algorithm · Quantum network

1 Introduction

In this paper, we consider a special collision problem. "Given a function $f\colon \{0,1\}^n \rightarrow \{0,1\}^m$ for any n, m. Suppose there exists a n-bit string $\alpha \in \{0,1\}^n$ such that $f(x \oplus \alpha) = f(x)$ for $\forall x \in \{0,1\}^n$. It only knows the Hamming weight $W(\alpha) = 1$ and find this α." In this problem, the string α is called invariant linear structure of function $f(x)$.

In cryptography, the original work of the collision problem was used to collision resistant of hash functions. Thus, finding the invariant linear structures of *MD* hash function is an important task as a special collision.

W. Wu (✉) · H. Zhang · S. Mao · H. Wang
Computer School of Wuhan University, Wuhan 430072, People's Republic of China
e-mail: wuwanqing8888@126.com

In quantum computation, Simon firstly present an efficiently quantum algorithm only for a special case of this collision problem, where the function f is defined as f: $\{0, 1\}^n \rightarrow \{0, 1\}^m$ with $n \leq m$ in [1]. The Simon algorithm needs $poly(n)$ queries on average, where the $ploy(n)$ is a polynomial in n. Aaronson [2] showed that there needs a lower bound of $\Omega(n^{1/7})$ quantum algorithm with bounded error probability to solve this collision problem. Soon after, Shi [3] pointed the quantum lower bound of $\Omega((n/m)^{1/3})$ for the collision in m-to-1 function, but the size is at least $3n/2$. Kutin [4] extended the results and removes the restriction. Ambainis [5] presented a $\Omega(n^{1/3})$ quantum algorithm for this collision problem with small range.

In classical computation, Wang et al. present the collision differential path to find the collision for more weak messages of $MD4$ and other hash functions in [6–9].

By the above motivation, we present a quantum algorithm to decide whether there exists such an invariant linear structure of the *MD* hash function family as a kind of collision. This paper is structured as follows: In Sect. 2, we present a quantum algorithm and analysis it. In Sect. 3, we present a quantum algorithm to find invariant linear structure of *MD*4 hash function as a kind of collisions. In Sect. 4, we present the consumptions of quantum algorithm for other hash functions. In Sect. 5, we summarize our paper.

2 Quantum algorithm of ILS problem

In this paper, we present a special problem and offer a quantum algorithm to solve it. It obtains the concrete results as follow.

Definition 1 Let n-ary function f: $\{0, 1\}^n \rightarrow \{0, 1\}^m$. The $\alpha \in \{0, 1\}^n$ is called invariant linear structure of function if $f(x \oplus \alpha) \oplus f(x) = 0$ for $\forall x \in \{0, 1\}^n$.

ILS Problem: Given a function f: $\{0, 1\}^n \rightarrow \{0, 1\}^m$. Suppose there exists a n-bit string $\alpha \in \{0, 1\}^n$ such that $f(x \oplus \alpha) = f(x)$ for $\forall x \in \{0, 1\}^n$. We only know the Hamming weight $W(\alpha) = 1$ and find this α.

Theorem 1 *There is an efficiently quantum algorithm to solve the ILS problem. The successful probability is $(\frac{2^l-1}{2^l})^{n-1}$, and time complexity is $O(\log(n-1))$ in the quantum algorithm, where the Hamming weight of the string α is 1 and the l is the times of quantum measurement in quantum algorithm.*

3 The proof of Theorem 1

In this section, we present the proof of the theorem 1 as follows. We firstly describe the quantum algorithm of the problem. Then, we illustrate the correctness of the algorithm. Thirdly, we illustrate the successful probability of the quantum algorithm. Fourthly, we illustrate the time complexity of the quantum algorithm.

3.1 The quantum algorithm of ILS problem

We present the following two unitary transformations in our algorithm. The first one is the standard quantum oracle U_f: $|x>|b>\rightarrow|x>|b\oplus f(x)>$ in [10], where the f is a function from $\{0,1\}^n$ to $\{0,1\}^m$. The second one *Hadamard* transformation

$$H=\frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}. \tag{1}$$

The quantum gate works on a single qubit as follows $H|0\rangle=\frac{1}{\sqrt{2}}|0\rangle+\frac{1}{\sqrt{2}}|1\rangle,\ H|1\rangle=\frac{1}{\sqrt{2}}|0\rangle-\frac{1}{\sqrt{2}}|1\rangle$. In following, we present the quantum algorithm.

Algorithm 1 The quantum algorithm of ILS Problem.

Step1 Put the first register in the uniform superposition states of n-qubits. This leaves our machine in state

$$\sum_{x=0}^{2^n-1}1/\sqrt{2^n}|x\rangle^n|0\rangle. \tag{2}$$

Step2 Compute $f(x)$ in the second register. Since we keep x in the first register this can be done reversibly. This leaves our machine in state

$$\sum_{x=0}^{2^n-1}1/\sqrt{2^n}|x\rangle|f(x)\rangle. \tag{3}$$

Step3 Perform *Hadamard* transformation on the first register. It has

$$\sum_{x=0}^{2^n-1}1/\sqrt{2^n}(H^{\otimes n}|x\rangle)|f(x)\rangle. \tag{4}$$

Step4 Measure the first quantum register and obtain the state $|c'\rangle$.
a) If the $W(\overline{c'})=1$, we obtain the result $\alpha=\overline{c'}$ and output it, where the $\overline{c'}$ denotes the compensation value of string c'.
b) If the $W(\overline{c'})\neq 1$, we repeat the above algorithm again and obtain $|c''\rangle$. Then, we calculate the $|c\rangle=|c'\rangle\vee|c''\rangle$, where " $\vee$ " denote the bitwise OR operation. Repeat the above process until the

$$W(c)=W(|c'\rangle\vee|c''\rangle\vee\cdots)=n-1.$$

Step5 Output the α.

3.2 Correctness and an example

Without loss of generality, let the string $\alpha=|0\rangle^{n-1}|1\rangle$ and $|x\rangle=|x_1\parallel x_2\rangle$, where $x_1\in\{0,1\}^{n-1}$ and $x_2\in\{0,1\}$. Thus, the $|x\oplus\alpha\rangle=|x_1\parallel\overline{x_2}\rangle$, where $|\overline{x_2}\rangle$ denote the compensation value of single qubit $|x_2\rangle$.

In steps 1 and 2 of algorithm, it obtains the quantum states $\sum_{x=0}^{2^n-1}1/\sqrt{2^n}|x\rangle|f(x)\rangle$. By the property $f(x\oplus\alpha)=f(x)$, it obtains

$$\sum_{x=0}^{2^n-1} 1/\sqrt{2^n}|x\rangle|f(x)\rangle = \sum_{x=0}^{2^n-1} 1/\sqrt{2^n}(|x\rangle + |x \oplus \alpha\rangle)|f(x)\rangle$$
$$= 1/\sqrt{2^n} \sum_{x_1=0}^{2^{n-1}-1} (|x_1 \parallel x_2\rangle + |x_1 \parallel \overline{x_2}\rangle)|f(x_1 \parallel x_2)\rangle.$$

In step 3, it obtains the quantum states

$$H^{\otimes n}\Big(1/\sqrt{2^n} \sum_{x_1=0}^{2^{n-1}-1} (|x_1 \parallel \overline{x_2}\rangle + |x_1 \parallel x_2\rangle)|f(x_1 \parallel x_2)\rangle\Big)$$
$$= 1/\sqrt{2^{n-1}} \sum_{x_1=0}^{2^{n-1}-1} (H^{\otimes n-1}|x_1\rangle)|0\rangle|f(x_1 \parallel x_2)\rangle,$$

since $\frac{1}{\sqrt{2}}H(|0\rangle + |1\rangle) = |0\rangle$. Note that the H transforms the behind single qubits of the first quantum register to $|0\rangle$. It reflects the characteristics of α.

In step 4, it measures the first quantum register and obtains the state $|c^{(1)}\rangle$. If the $W(c^{(1)}) = n - 1$, it obtains the result $\alpha = \overline{c^{(1)}}$ and output it. Otherwise, if the $W(c^{(1)}) < n - 1$, it repeats the above algorithm again and obtains $|c^{(2)}\rangle$. Furthermore, it can obtain a series of strings $c^{(i)} \in \{0, 1\}^n$ to implement the bitwise OR as $c = c^{(1)} \vee c^{(2)} \ldots$. Finally, the $W(c) = n - 1$ (or $W(\overline{c}) = 1$) then it obtains the success string $\alpha = \overline{c}$.

In following, we present a concrete example.

Example 1 Suppose $n = 3$ and $W(\alpha) = 1$. Given a Boolean function f: $\{0, 1\}^3 \rightarrow \{0, 1\}^2$ satisfies $f(000) = f(001)$, $f(010) = f(011)$, $f(100) = f(101)$, $f(110) = f(111)$. Find α.

Let the initial state be $|000\rangle|00\rangle$ and applying $H^{\otimes 3}$ for the first quantum register. Compute $f(x)$ in the second register, we obtain

$$\left(\frac{1}{\sqrt{8}}|000\rangle + \frac{1}{\sqrt{8}}|001\rangle\right) f(000) + \left(\frac{1}{\sqrt{8}}|010\rangle + \frac{1}{\sqrt{8}}|011\rangle\right) f(010)$$
$$+ \left(\frac{1}{\sqrt{8}}|100\rangle + \frac{1}{\sqrt{8}}|101\rangle\right) f(100) + \left(\frac{1}{\sqrt{8}}|110\rangle + \frac{1}{\sqrt{8}}|111\rangle\right) f(110)$$

Thus, we perform *Hadamard* transformation on the first register again and obtain the quantum states

$$\frac{1}{4}\Big((|000\rangle + |010\rangle + |100\rangle + |110\rangle)f(000) + (|000\rangle - |010\rangle + |100\rangle - |110\rangle)f(010)$$
$$+ (|000\rangle + |010\rangle - |100\rangle - |110\rangle)f(100) + (|000\rangle - |010\rangle - |100\rangle + |110\rangle)f(110)\Big).$$

It measures the first quantum register and obtains the state $|110\rangle$ with probability $\frac{1}{4}$. If it obtains the string $c' = |100\rangle$, then implements the quantum algorithm again. Let us suppose that it obtains a string $c'' = |010\rangle$. Then, it calculates $|c\rangle = |c'\rangle \vee |c''\rangle = |110\rangle$. So, output $\alpha = |\bar{c}\rangle = |001\rangle$.

3.3 Correctness probability

In this paper, l denotes the times of measurement. For general l and n, we consider the equation $c = x_1 \vee x_2 \vee \ldots \vee x_l$, where $x_i \in \{0, 1\}^n$, $i = 1, \ldots, l$. We determine firstly the failure case, i.e., $c = 000\ldots000, 010\ldots000, 100\ldots000, \ldots,$ and $111\ldots100$ after bitwise OR. When the $|c\rangle = |\underbrace{0XX\ldots X}_{n-1}0\rangle$, we obtain the $(2^l)^{n-2}$ cases, where X is an undecided element. If we observe the $|c\rangle = |\underbrace{10XX\ldots X}_{n-1}0\rangle$, we obtain $(2^l - 1) \times (2^l)^{n-3}$ cases. For general state $|\underbrace{1\ldots1}_{y}0\underbrace{XX\ldots X}_{n-2-y}0\rangle$, we obtain $(2^l - 1)^y \times (2^l)^{n-2-y}$ cases. Let the symbol $P_{(l,n)}$ denote the successful probability of the quantum algorithm after l measurements and "$\vee$" operate. So we obtain the successful probability

$$\begin{aligned}
P_{(l,n)} &= 1 - \frac{(2^l)^{n-2} + (2^l - 1) \times (2^l)^{n-3} + \cdots + (2^l - 1)^{n-2} \times 1}{(2^l)^{n-1}} \\
&= \frac{(2^l)^{n-1} - (2^l)^{n-2} - (2^l - 1) \times (2^l)^{n-3} - \cdots - (2^l - 1)^{n-2} \times 1}{(2^l)^{n-1}} \\
&= \frac{(2^l)^{n-2} \times (2^l - 1) - (2^l - 1) \times (2^l)^{n-3} - \cdots - (2^l - 1)^{n-2} \times 1}{(2^l)^{n-1}} \\
&= \frac{(2^l - 1)^{n-1}}{(2^l)^{n-1}} = \left(\frac{2^l - 1}{2^l}\right)^{n-1}.
\end{aligned}$$

Thus, the more measurements, the higher probability of the quantum algorithm. In Table 1, we present some examples.

3.4 The time complexity on average

In this section, we consider the time complexity of the quantum algorithm on average. Let the time complexity as $T(n)$ for general n.

Table 1 The successful probability of the quantum algorithm, when the $l > 2$

n	$W(\alpha)$	Times (l)	Pr
100	1	7	0.4600
100	1	8	0.6788
100	1	9	0.8240

We firstly evaluate a special case, i.e., $n = 2$. Assume $\alpha = |01\rangle$. After the measure, we only have two states $|00\rangle$, $|10\rangle$ and each probability is $\frac{1}{2}$. If we obtain the state $|10\rangle$, the algorithm finishes the task. Otherwise, it performs the quantum algorithm again. Thus, it has

$$T = 1 \times \frac{1}{2} + 2 \times \left(\frac{1}{2}\right)^2 + 3 \times \left(\frac{1}{2}\right)^3 + \cdots = \sum_{j=1}^{\infty} j \left(\frac{1}{2}\right)^j$$

From the equation $\sum_{a=1}^{\infty} ax^a = \frac{x}{(1-x)^2}$ for $|x| < 1$, it has $T = 2$.

Next, we roughly estimate the solution of $T(n)$. Let the $W(\alpha) = 1$, then the result of measurement is the string $\overline{\alpha} = |\underbrace{1 \ldots 1}_{n-1} 0\rangle$. After the measure of the quantum algorithm, it can obtain the half of $n - 1$ bits hold one on average. If it repeats the processing one time, the half of the rest about $\lfloor \frac{n-1}{2} \rfloor$ bits holds one on average. If it repeats $\log(n-1)$ times, only the rest one bit of $\overline{\alpha}$ is unknown. By the $T = 2$, it obtains the approximate value, i.e.,

$$T(n) \approx 2 + \log(n - 1). \tag{5}$$

4 The quantum algorithm to find invariable linear structure of *MD*4

In this section, we present a quantum algorithm to decide whether there exists a invariant linear structure as a kind of collision of the *MD*4, while we provide a time–space trade-off to save the consumption.

4.1 The *MD*4 algorithm

1. The padding message. Given any arbitrary bit length message M, the *MD*4 algorithm firstly pads M into a multiple of 512-bit length message. We can skip the padding technology, since it has no impact on the quantum attack.
2. The blocks of message. The *MD*4 algorithm divides the padding message into a sequence of 512-bit length message blocks, i.e., $M_1, \ldots, M_n$.
3. Set the initial value. The *MD*4 algorithm provides a 128-bit length register to store the intermediate values of Hash function. Let the 128-bit length register be (a, b, c, d), where a, b, c, d, respectively, are 32-bit. Let the initial value $H_0 = (a, b, c, d) = (0x67452301,\ 0xefcdab89,\ 0x98badcfe,\ 0x10325476)$.
4. The iterative compression. For each 512-bit length message M_i, we compute the 128-bit length hash value H_{i+1} using the compression function f, i.e., $H_i = f(H_{i-1}, M_i)$.
5. Output the message digest. The H_{n+1} is the output as the message digest of *MD*4 algorithm.

4.2 The MD4 compression function

1. The function f compresses the 128-bit intermediate variable and 512-bit message block into 128-bit length hash value. Let the intermediate variable $H_i = (aa, bb, cc, dd)$. If the first message block is M_1, the $H_0 = (aa, bb, cc, dd)$ are set to be the initial value. Otherwise, the input of algorithm is the compressing variable from the previous operation.
2. Each round of the compression function performs 48 steps operations as follows: For $j = 0, 1, 2$ and $i = 0, 1, 2, 3$,

$$\begin{aligned}a &= f_j(a, b, c, d, w_{j,4i}, s_{j,4i})\\ d &= f_j(d, a, b, c, w_{j,4i+1}, s_{j,4i+1})\\ c &= f_j(c, d, a, b, w_{j,4i+2}, s_{j,4i+2})\\ b &= f_j(b, c, d, a, w_{j,4i+3}, s_{j,4i+3})\end{aligned}$$

   Each step updates the register (a, b, c, d). The $w_{j,4i+k}$ is a 32-bit string from message extension, where $k = 0, 1, 2, 3$. Here, $s_{j,4i+k}$ is a constant, and $\lll s_{j,4i+k}$ is circularly left-shift by $s_{j,4i+k}$-bit positions. Each round of the compression function implements 16 step operations, and these operations f_j, $j = 0, 1, 2$ are defined as:

$$\begin{aligned}f_0(a, b, c, d, w_k, s) &= ((a + F(b, c, d) + w_k) \mod 2^{32}) \lll s\\ f_1(a, b, c, d, w_k, s) &= ((a + G(b, c, d) + w_k + 0x5a827999) \mod 2^{32}) \lll s\\ f_2(a, b, c, d, w_k, s) &= ((a + H(b, c, d) + w_k + 0x6ed9eba1) \mod 2^{32}) \lll s\end{aligned}$$

   The compression function has three rounds. Each round contains a different nonlinear Boolean function as follows:

$$\begin{aligned}F(X, Y, Z) &= (X \wedge Y) \vee (\neg X \wedge Z)\\ G(X, Y, Z) &= (X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z)\\ H(X, Y, Z) &= X \oplus Y \oplus Z\end{aligned}$$

   These functions are all bitwise operation, and the X, Y, Z are 32-bit string. The $\wedge, \vee, \oplus$ are, respectively, the bitwise AND, OR, XOR and the $\neg X$ is a string obtained by reversing all bits of X as $0 \leftrightarrow 1$.
3. In *MD*4 algorithm, it computes the $(H_i \oplus H_{i+1}) \mod 2^{32}$ as the intermediate variables H_{i+1} after updating. The process is as follows.

$$\begin{aligned}aa &= (a + aa) \mod 2^{32}\\ bb &= (b + bb) \mod 2^{32}\\ cc &= (c + cc) \mod 2^{32}\\ dd &= (d + dd) \mod 2^{32}\end{aligned}$$

4.3 The elementary quantum arithmetic operations

The quantum computation is constituted by quantum logic gates and measurement. It can accept superposition states input and output corresponding superposition states. This process is as follow

$$U_f : \ |x, 0\rangle \rightarrow |x, f(x) \oplus 0\rangle, \tag{6}$$

where f is any function and U_f is a unitary operator.

In order to realize quantum algorithm, we use the three quantum gates *NOT*, *control-NOT*, *Toffoli* to construct the operation in this paper.

***AND* and *OR* operation**. We describe the two n-bit length strings $a, b \in \{0, 1\}^n$. We have $a = a_1a_2 \ldots a_n$ and $b = b_1b_2 \ldots b_n$, where $a_i, b_i \in \{0, 1\}$ for $i = 1, \ldots, n$. We define the bitwise AND operation as $c = a \wedge b \in \{0, 1\}^n$, where $c_i = a_ib_i$ for $i = 1, \ldots, n$. We define the bitwise OR operation as $c = a \vee b \in \{0, 1\}^n$, where $c_i = \max(a_i, b_i)$ for $i = 1, \ldots, n$. In Fig. 1, we present the quantum network of AND and OR operation, where the control qubits are represented by a dot, the target qubits by a plus.

***Rotate* and *Shift* operation**. Let $ROTL_n$ and SHL_n(or $ROTR_n$ and SHR_n) are, respectively, the left-rotate and left-shift (or right-rotate and right-shift) by place n-bits.

The $ROTL_n$(or $ROTR_n$) is an important arithmetic operation in block cipher. We can achieve the quantum network of rotate operation through some $control - NOT$ gates, i.e., see Fig. 2(1) for $n = 1$.

Let $|x\rangle$ be a n-qubit state, and $ROTL_m$ denotes left-shift m-bits. In quantum network, we initialize the state $|x\rangle^{\otimes n}|0\rangle^{\otimes m}$ and implement $ROTL_m(|x\rangle^{\otimes n}|0\rangle^{\otimes m})$ operation. We only consider the first n-qubit of $ROTL_m(|x\rangle^{\otimes n}|0\rangle^{\otimes m})$ as the result of SHL_m and omit the last m-bits, i.e., see Fig. 2(2) for $n = 1$.

In Fig. 2, we present two examples, i.e., $ROTL_1$ and SHL_1 operation.

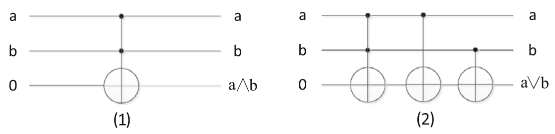

Fig. 1 *1* The AND operation. *2* The OR operation

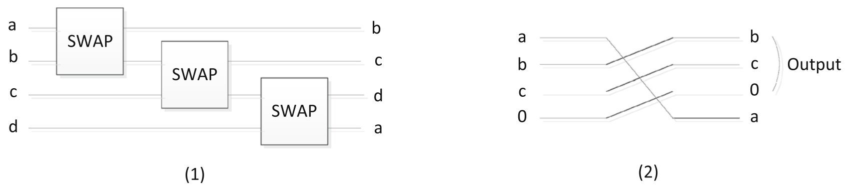

Fig. 2 *1* The $ROTL_1$ operation; *2* The SHL_1 operation

4.4 The quantum network for nonlinear Boolean function

These are three different nonlinear Boolean functions in *MD*4 algorithm. They are constituted by the four elementary arithmetic operations AND, OR, XOR, and $\neg$. By combining the circuit diagram of the Fig. 1, the first nonlinear Boolean function $F(X, Y, Z)$ can be written as

$$U_F|X, Y, Z, 0, 0, 0\rangle \rightarrow |X, Y, Z, X \wedge Y, \neg X \wedge Z, F(X, Y, Z)\rangle. \tag{7}$$

The nonlinear Boolean function $F(X, Y, Z)$ is stored in the last temporary register. Thus, the operation of the function $F(X, Y, Z)$ can be completed. In order to optimize this operation, we will consider a slight changes and rewrites the function computation. The function $F(X, Y, Z)$ can be rewritten as

$$U_F|X, Y, Z, 0\rangle \rightarrow |X, Y, Z, F(X, Y, Z)\rangle. \tag{8}$$

This way is the most optimal, that is, the number of qubits is minimal. The reason is as follow. Assume the function $G(X, Y, Z)$ can be written as

$$U_F|X, Y, Z\rangle \rightarrow |X, Y, F(X, Y, Z)\rangle, \tag{9}$$

Without loss of generality, we suppose that each string $X, Y, Z, F(X, Y, Z)$ has 1-bit size and the output is $|X, Y, F(X, Y, Z)\rangle$. Thus, we obtain the two same states $|100\rangle$ as output. This is a contradiction with the uniqueness of the element representation.

So the operation of the functions $F(X, Y, Z), G(X, Y, Z), and H(X, Y, Z)$ are illustrated in Fig. 3. In Fig. 3(1), the $\times$ denote the component of element X.

In addition, the inverse operation of the U_f can be implemented in the reverse path. The inverse operation can be denoted by ${}_fU$. Thus, the ${}_FU, {}_GU, {}_HU$, respectively, are the inverse operation of U_F, U_G, U_H.

4.5 The quantum network for compression function f_j

The compression function $f_j (j = 0, 1, 2)$ contains two operations, i.e., adder modulo and left-shift. Vedral presented the quantum network of adder module in [11]. The adder module of two registers $|a\rangle$ and $|b\rangle$ can be written as

$$|a, b\rangle \rightarrow |a, (a + b) \mod 2^N\rangle, \tag{10}$$

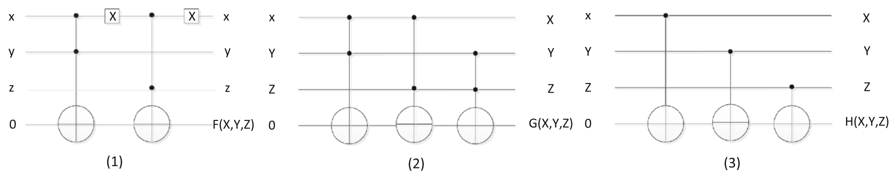

Fig. 3 *1 $F(X, Y, Z)$; 2 $G(X, Y, Z)$; 3 $H(X, Y, Z)$*

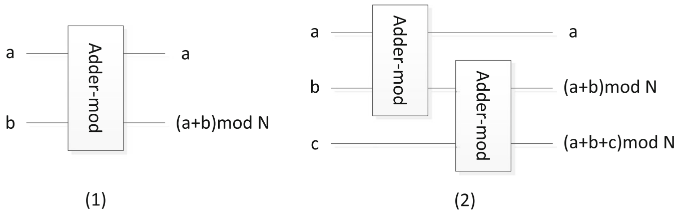

Fig. 4 *1* The adder module of two registers. *2* The adder module of three registers

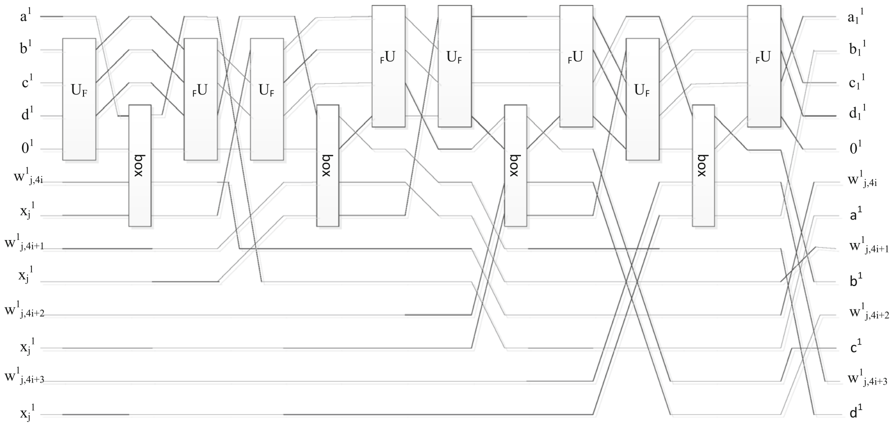

Fig. 5 The compression function f_j of *MD*4 algorithm

where the $N = 2^{32}$ in *MD*4 algorithm. Thus, we can obtain adder module of three registers $|a\rangle$, $|b\rangle$, and $|c\rangle$ through combining these adder modules. The adder module is illustrated in Fig. 4. So we can obtain the compression function f_j for $j = 0, 1, 2$ through combining the adder module and shift operation as above discussion.

4.6 The construction of compression function of *MD*4 algorithm

In *MD*4 algorithm, it has three rounds; each round of compression function implements 16 step operations and every four steps update the register (a, b, c, d).

In Fig. 5, we present the quantum network of register update. In order to complete quantum computation, we made a slight changes, i.e., convert each states $a, b, c, d, 0, w, x_0, x_1, x_2$ into $a', b', c', d', 0', w', x_0', x_1', x_2'$ satisfying $x' = x \parallel \underbrace{0\dots0}_{s}$, where s is the largest number of displacement in *MD*4.

The first U_F implements a nonlinear Boolean function $F(X, Y, Z)$ as follow,

$$U_F : |b', c', d', \underbrace{0\dots0}_{32+s}\rangle = |b \parallel \underbrace{0\dots0}_{s}, c \parallel \underbrace{0\dots0}_{s}, d \parallel \underbrace{0\dots0}_{s}, \underbrace{0\dots0}_{32+s}\rangle$$
$$\rightarrow |b \parallel \underbrace{0\dots0}_{s}, c \parallel \underbrace{0\dots0}_{s}, d \parallel \underbrace{0\dots0}_{s}, F(b, c, d) \parallel \underbrace{0\dots0}_{s}\rangle.$$

Then, the state $|a'\rangle$ is exchanged with the states $|b', c', d'\rangle$.

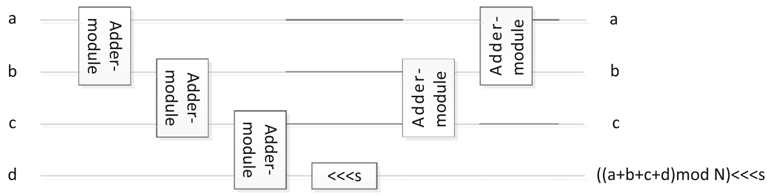

Fig. 6 The "*box*" operation

Furthermore, it applies the "box" operation for computation in Fig. 6. The compression function f_j can be defined as following

$$\begin{aligned} U_{f_i} &: |a', F(b,c,d)||\underbrace{0\cdots0}_{s}, w'_{j,4i}, x'_j\rangle \\ &= |a \parallel \underbrace{0\cdots0}_{s}, F(b,c,d)||\underbrace{0\cdots0}_{s}, w'_{j,4i}, x_j||\underbrace{0\cdots0}_{s}\rangle \\ &\rightarrow |a||\underbrace{0\cdots0}_{s}, F(b,c,d)||\underbrace{0\cdots0}_{s}, w'_{j,4i}, f_i(a',b',c',d',w'_{j,4i},s)\rangle, \end{aligned}$$

where the $f_i(a',b',c',d',w'_{j,4i},s)$ are (32+s) qubits. In "box" operation, there are only first 32-qubits to participate the adder modulo N operation.

Next, the states $|b',c',d'\rangle$ are exchanged with the state $|a'\rangle$ and performs the reserve of U_F. After $_FU$, it obtains the initial states $|a',b',c',d',0\rangle$. The state $w'_{j,4i}$ as a output has no change after the "*box*" operation. Furthermore, we obtain the state $|f_i(a',b',c',d',w'_{j,4i},s),b',c',d',0\rangle$ as the input of the next operation. So the *MD*4 algorithm completes 1 step operation.

In this way, the *MD*4 algorithm can complete the all round computations. The "box" operation includes three adder modules and two subtractors and one shift operation. In Fig. 6, we present the quantum network of "box" operation.

4.7 The quantum algorithm for a kind of collision of *MD*4

In Fig. 5, the compression function f_j of algorithm *MD*4 transforms the state $|x'_j, x'_j, x'_j, x'_j >$ into the state $|a'_1, b'_1, c'_1, d'_1 >$. Thus, the algorithm needs 16 registers to store the intermediate variables $|a'_1, b'_1, c'_1, d'_1, \ldots, a'_4, b'_4, c'_4, d'_4 >$ in first round. For the all three rounds, the quantum algorithm needs 48 registers to store the intermediate variables $|a'_1, b'_1, c'_1, d'_1, \ldots, a'_{12}, b'_{12}, c'_{12}, d'_{12} >$. While it needs a 32-bits temporary register used to store the module $N = 2^{32}$ and a one bit to complement the adder module in [11]. The all adder module operations only need a temporary 33-bits in our quantum algorithm. For convenience, we ignore its existence in quantum network and it is included in the "box" operation. By the [12], the largest number of displacement is $s = 19$ in *MD*4. So the quantum algorithm of *MD*4 needs 3,552-qubits length registers to complete the computation in total. The specific conditions are as follows.

As a hash function, it can be written as

$$f: |w_1, \ldots, w_{16} > \rightarrow |f(M)>, \tag{11}$$

where the $M = w_1 \parallel \ldots \parallel w_{16}$ is 512-bits length message block and $f(M)$ is 128-bits length message digest. This computational procession can be written as

$$\begin{aligned} &U_f|a', b', c', d', 0, w'_1, \ldots, w'_{16}, \underbrace{x'_0, \ldots, x'_0}_{16}, \underbrace{x'_1, \ldots, x'_1}_{16}, \underbrace{x'_2, \ldots, x'_2}_{16} > \\ &\quad \rightarrow |a', b', c', d', 0, w'_1, \ldots, w'_{16}, a'_1, b'_1, c'_1, d'_1, \ldots, a'_{11}, b'_{11}, c'_{11}, d'_{11}, \\ &\qquad f(x) \parallel s \parallel s \parallel s \parallel s >, \end{aligned} \tag{12}$$

where the function f is the compression function of algorithm $MD4$ and each register is 51-qubits. By above discussion, we present the quantum algorithm to find the invariant linear structure as a kind of collision of $MD4$ as follows.

4.8 Time–space trade-off

The time–space trade-off is a trade-off strategy between the time complexity and space complexity. In general, the more bits are utilized the faster it implement the algorithm. But, if the number of bits are too much, it is not easy to achieve. In the quantum algorithm, we present a trade-off method to find a kind of the collision of *MD* hash function family.

We increase some measure steps and control steps in the quantum circuit of compression function of $MD4$ algorithm for saving the qubits. During the process of the quantum algorithm, we need some qubits to store the intermediate variables $|a'_i, b'_i, c'_i, d'_i >, i = 1, \ldots, 12$. In order to save the cost of qubits, we omit the storage of intermediate variables. In Fig. 5, we obtain the $|a'_1, b'_1, c'_1, d'_1, 0', w'_{j,4i}, a', w'_{j,4i+1}, b', w'_{j,4i+2}, c', w'_{j,4i+3}, d' >$ as the output in the first step. Thus, we implement the measure for the specific qubits states $|a', b', c', d' >$ in the output. Furthermore, quantum states can be converted into classical state including only 0 or 1. Then, we implement $\neg$ operation for classical state $|a', b', c', d' >$, if the qubit is 1. Otherwise do nothing. Now, the classical states $|a', b', c', d' >$ are written as $|0', 0', 0', 0' >$. For the next step computation, we setup the $|0', 0', 0', 0' > \rightarrow |x'_j, x'_j, x'_j, x'_j >$ through $\neg$ operation, since the known state $|x'_j >$ is also a classical state. So we obtain the state $|a'_1, b'_1, c'_1, d'_1, 0', w'_{j,4i}, x'_0, w'_{j,4i+1}, x'_0, w'_{j,4i+2}, x'_0, w'_{j,4i+3}, x'_0 >$ as the input in the second step. Similarly, we implement the same operations for other intermediate variables. Thus, the input of quantum algorithm can be written as

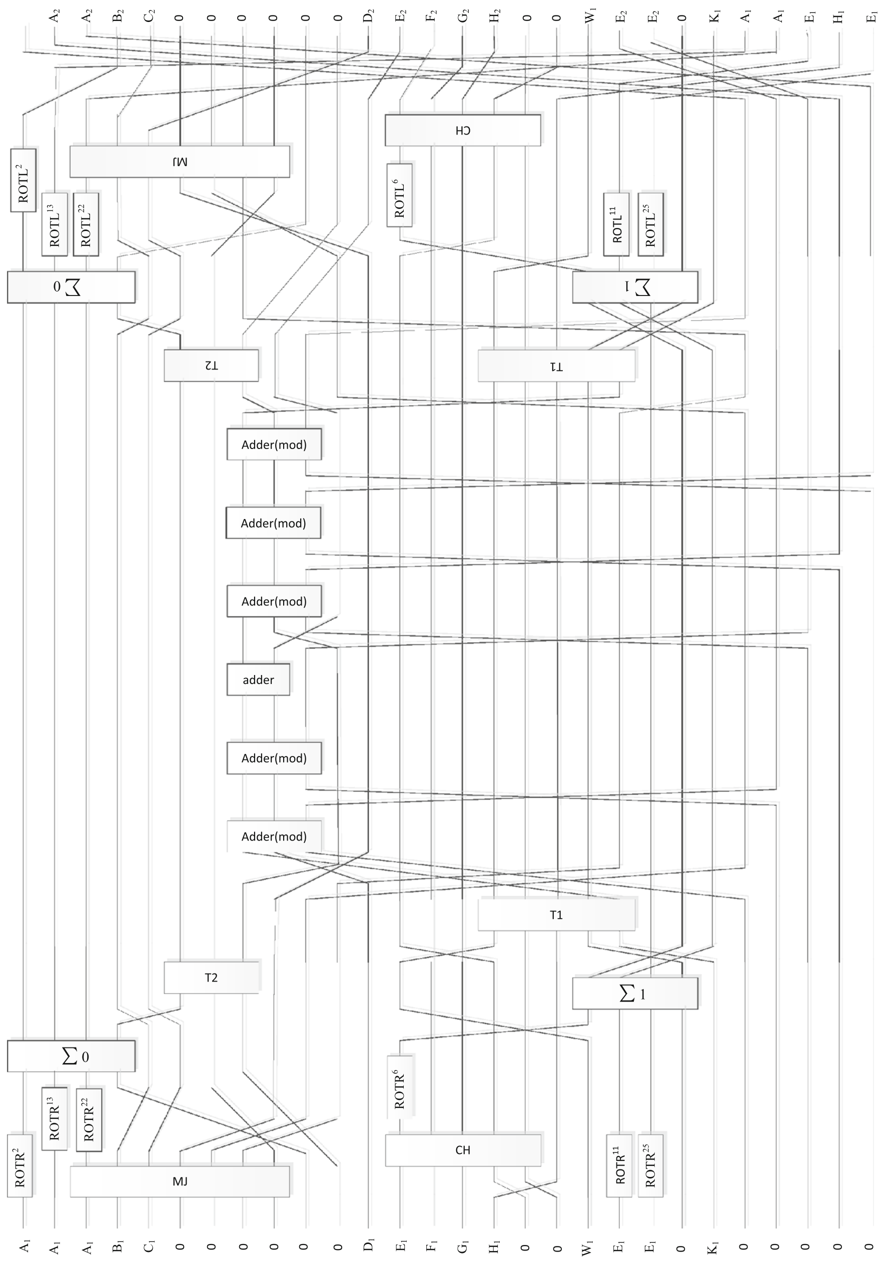

Fig. 7 The quantum network of SHA-256

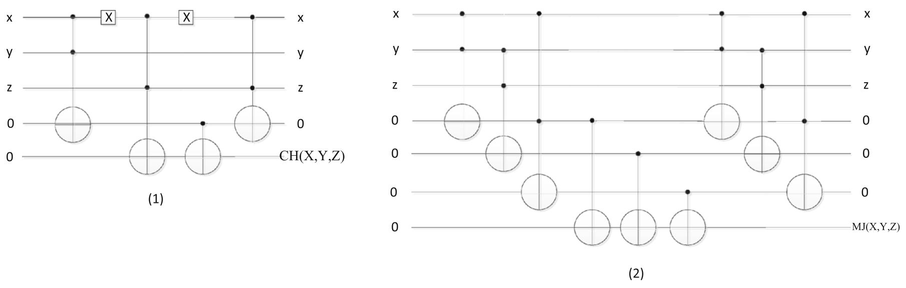

Fig. 8 *1* The quantum network of CH; *2* The quantum network of MJ

Algorithm 2 The quantum algorithm for finding a kind of collision of *MD*4.

Step1 Setup the initial state. Let the classical states $a', b', c', d', 0', x'_0, x'_1, x'_2$ satisfy $x' = x \parallel \underbrace{0\cdots0}_{19}$. The initial state can be defined as

$$|x>_{in}= |\underbrace{0', \cdots, 0'}_{16}> |a', b', c', d', 0', \underbrace{x'_0, \cdots, x'_0}_{16}, \underbrace{x'_1, \cdots, x'_1}_{16}, \underbrace{x'_2, \cdots, x'_2}_{12}> |x'_2, x'_2, x'_2, x'_2>, \quad (13)$$

where the $a = 0x67452301, b = 0xefcdab89, c = 0x98badcfe, d = 0x10325476, x_1 = 0x5a827999, x_2 = 0x6ed9eba1$ and put the other qubit state into $|0>$.

Step2 Apply $U_H^{\otimes 16}$ operations on the first 16 quantum registers. Thus, it has

$$U_H^{\otimes 16}|x>_{in}= \sum_{x=0}^{2^{16\times 51}-1} \lambda_x|x> |a', b', c', d', 0, \underbrace{x'_0, \cdots, x'_0}_{16}, \underbrace{x'_1, \cdots, x'_1}_{16}, \underbrace{x'_2, \cdots, x'_2}_{12}> |x'_2, x'_2, x'_2, x'_2>, \quad (14)$$

where λ_x denotes probability amplitude.

Step3 Apply the quantum oracle U_f and store the functions $f(x)$ in the last four quantum registers. Thus, it has

$$\sum_{x=0}^{2^{16\times 51}-1} \lambda_x|x> |a', b', c', d', 0, a'_1, b'_1, c'_1, d'_1, \cdots, a'_{11}, b'_{11}, c'_{11}, d'_{11}> |f(x) \parallel s \parallel s \parallel s \parallel s>, \quad (15)$$

where the function f is the quantum algorithm of *MD*4 and $f(x) = (a_{12}, b_{12}, c_{12}, d_{12})$ is 128-qubits.

Step4 Apply $U_H^{\otimes n}$ operations for the first 16 quantum registers again. It has

$$\sum_{x=0}^{2^{16\times 51}-1} \lambda_x(U_H^{\otimes 16}|x>)|a', b', c', d', 0, a'_1, b'_1, c'_1, d'_1, \cdots, a'_{11}, b'_{11}, c'_{11}, d'_{11}> |f(x) \parallel s \parallel s \parallel s \parallel s>, \quad (16)$$

Step5 We perform $\{0, 1\}$ measurements for the first 16 quantum register and obtain the state $|c'>$. If the $W(\overline{c'}) = w$, we obtain $\alpha = \overline{c'}$ and output it. If the $W(\overline{c'}) \neq w$, we rerun the above algorithm again and obtain $|c''>$. Then, we calculate $|c>=|c'> \vee |c''>$, where "$\vee''$ denote the bitwise OR operate.

Step6 Compute $f(x)$ and $f(x \oplus \alpha)$. If the $f(x \oplus \alpha)$, we obtain the result $\alpha = \overline{c}$ and output it. Otherwise, it returns fails.

Table 2 The consumption of hash functions

Hash functions	$MD4$	$MD5$	SHA-0	SHA-1	SHA-224/256	SHA-384/512
Word(w)	32	32	32	32	32	64
Block	512	512	512	512	512	1,024
Digest	128	128	160	160	224/256	384/512
The number of qubits	3,552	5,473	7,925	7,925	15,041	37,249
Quantum time complexity	$<2^4$	$<2^4$	$<2^4$	$<2^4$	$<2^4$	$<2^5$
The number of qubits (tradeoff)	1,308	1,408	1,459	1,459	1,611	3,000
Quantum time complexity (tradeoff)	$<2^8$	$<2^8$	$<2^8$	$< 2^8$	$< 2^8$	$< 2^9$

$$|x >_{in} = |\underbrace{0', \ldots, 0'}_{16} > |a', b', c', d', 0', x_0', x_0', x_0', x_0' >, \tag{17}$$

In summary, we need $((32 + s) \times 25 + 33)$-qubits to complete the quantum algorithm, where the $s = 19$. At the same time, it needs to increase 16 steps measurement and control, that is, the time complexity is expanded 16 times.

5 Conclusions

At present, there are many hash functions based on the $MD4$ [12]. The variant versions of $MD4$ include $MD5$ [13], SHA-0 [14], SHA-1 [15], SHA-224/256, SHA-384/512 [16]. They can be called MD hash function family.

The $MD4$, $MD5$ hash functions have same structure; then, we can obtain similar quantum networks from $MD4$ as above discussion. The structure of SHA hash functions is different with $MD4$. In Fig. 7, we provide the quantum network of SHA-256 to explain the quantum algorithm for finding invariable linear structure, where the adder can be defined as $|a, b, 0\rangle \rightarrow |a, b, (a + b) \mod N\rangle$. In Fig. 8, we provide the quantum network of CH and MJ operation used in algorithm SHA-256.

These hash functions have similar structures called by Merkle-Damgard method. These different round functions include only bitwise AND, OR, XOR, and $\neg$. These elementary arithmetic operations can be implemented by the quantum network in [11,17]. Thus, we can find a kind of collisions of other hash functions using the above quantum algorithm. In Table 2, we provide the results of consumption for some hash functions.

6 Summary

The quantum algorithm can solve some intractable problems in classical computers. For example, the Shor algorithm can solve the discrete logarithm and factorization problem in polynomial time in [18]. The Shor algorithm is a serious threat for the classical public key cryptosystems such as *RSA*, *ELGamal*, *ECC*. In 2003, Proos and Zalka pointed out that there exists an effective quantum algorithm to solve the K-bits elliptic curves on N-qubits quantum computer in [19], where the $N = 5k + 8\sqrt{k} + 5log_2^k$. In 2003, Beauregard pointed that there exists an effective quantum algorithm to solve the K-bits factorization problem on N-qubits quantum computer in [19], where the $N = 2k$. Thus, we believe that 1,448-qubits quantum computer can solve the 256-bits elliptic curves, and 2,048-qubits quantum computer can compute the 1,024-bits RSA in [20]. Thus, our quantum algorithm is also effective to find the invariant linear structure as a kind of collision of the *MD* hash function family using the quantum "Oracle". Since the number of qubits are from 1,000 to 3,000.

In this paper, we consider a special version of this collision. We present a quantum algorithm with "Oracle" to solve this problem. As an application, we present a quantum algorithm to decide whether there exists such an invariant linear structure of the *MD* hash functions family. Finally, we provide some consumptions of these quantum algorithms through the time–space trade-off.

Acknowledgments WanQing Wu: Supported by the Fundamental Research Funds for the Central Universities (No. 2012211020213). HuanGuo Zhang: Supported by the Major Research Plan of the National Natural Science Foundation of China (No. 91018008), the National Natural Science Foundation of China (No. 61303212, 61202386), and Major State Basic Research Development Program of china (No. 2014CB340600). E-mail: liss@whu.edu.cn

References

1. Simon, D.R.: On the power of quantum computation. SIAM J. Comput. **26**(5), 1474–1483 (1997)
2. Aaronson, S.: Quantum lower bound for the collision problem. In: Proceedings of the Thiry-Fourth Annual ACM Symposium on Theory of Computing, pp. 635–642. ACM, New York (2002)
3. Shi, Y.: Quantum lower bounds for the collision and the element distinctness problems. In: Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002, pp. 513–519. IEEE (2002)
4. Kutin, S.: Quantum lower bound for the collision problem with small range. Theory Comput. **1**(1), 29–36 (2005)
5. Ambainis, A.: Polynomial degree and lower bounds in quantum complexity: collision and element distinctness with small range. Theory Comput. **1**(1), 37–46 (2005)
6. Wang, X., Yu, H.: How to Break MD5 and Other Hash Functions. Advances in Cryptology-EUROCRYPT. Springer, Berlin (2005)
7. Wang, X., Yin, Y.L., Yu, H.: Finding Collisions in the Full SHA-1. Advances in Cryptology-CRYPTO. Springer, Berlin (2005)
8. Wang, X., Lai, X., Feng, D., et al.: Cryptanalysis of the Hash Functions MD4 and RIPEMD. Advances in Cryptology-EUROCRYPT. Springer, Berlin (2005)
9. Wang, X., Yu, H., Yin, Y.L.: Efficient Collision Search Attacks on SHA-0. Advances in Cryptology-CRYPTO, 1st edn. Springer, Berlin (2005)
10. Kashefi, E., Kent, A., Vedral, V., et al.: Comparison of quantum oracles. Phys. Rev. A **65**(5), 050304 (2002)
11. Vedral, V., Barenco, A., Ekert, A.: Quantum networks for elementary arithmetic operations. Phys. Rev. A **54**(1), 147 (1996)
12. Rivest, R.L.: The MD4 Message-Digest Algorithm. Advances in Cryptology, Crypto'90. Springer, Berlin (1991)
13. Rivest, R.L.: The MD5 Message-Digest Algorithm, Request for Comments (RFC 1320), Internet Activities Board, Internet Privacy Task Force (1992)

14. Secure Hash Standard. Federal Information Processing Standard Publication 180, U.S. Department of Commerce, National Institute of Standards and Technology (1993)
15. National Institute of Standards and Technology (NIST) FIPS Publication 180-1: secure Hash Standard (1994)
16. National Institute of Standards and Technology (NIST), FIPS 180–2(2002). http://csrc.nist.gov/encryption/tkhash.html
17. Cleve, R.: An introduction to quantum complexity theory. In: Collected Papers on Quantum Computation and Quantum Information Theory, pp. 103–127 (2000)
18. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM Rev. **41**(2), 303–332 (1999)
19. Proos, J., Zalka, C.: Shor's discrete logarithm quantum algorithm for elliptic curves. Quantum Inf. Comput. **3**, 317–344 (2003)
20. Darrel, H., Alfrend, M., Scott, V.: Guide to Elliptic Curve Cryptography. Springer, Berlin (2004)