Quantum Circuit for Curve25519 with Fewer Qubits

No Author Given

No Institute Given

Abstract. In this paper, we propose a quantum circuit for Curve25519, the elliptic curve used in Elliptic-Curve Cryptography (ECC). First, we implemented the addition, subtraction, and multiplication (squaring) operations on prime fields as quantum circuits. These circuits are flexible, allowing for modifications to the prime value and input length, making them applicable not only to Curve25519 but also to other cryptographic systems. We optimally adjusted the algorithmic sequences of the Point Doubling and Point Addition functions used in Curve25519 for quantum circuit implementation. Additionally, we strategically placed inverse operations to eliminate or reduce the use of temporary qubits in the algorithm. As a result of our qubit optimization, we reduced the total number of qubits in the Curve25519 circuit by 2,483.

Keywords: Quantum circuit · Quantum Implementation · Quantum Computing · Curve25519

1 Introduction

With the advent of the quantum computing era, traditional encryption systems, such as public-key cryptography and symmetric-key cryptography, are increasingly at risk. As a result, research on quantum circuit implementations has been continuously conducted to verify whether cryptographic algorithms that are considered secure on classical computers remain secure on quantum computers [7,1,8,16,15,18,14,17,4,9,12]. In theory, Shor's algorithm can efficiently solve the problems of factoring large integers and discrete logarithms [13], while Grover's algorithm can reduce the number of operations needed to find a specific key in cryptographic algorithms to the square root level [6]. This suggests that public-key and symmetric-key cryptography could potentially be broken. However, this is expected to require large-scale quantum computers, and some predict that it may still take decades before quantum computers are capable of breaking cryptography.

Research is underway to transition to post-quantum cryptography (PQC), which is secure against both quantum and classical computers [3]. However, this is expected to be a process that requires significant time and cost. Therefore, it is necessary to assess the quantum security strength of currently used cryptographic algorithms to determine how long they will remain secure and to what extent they can be considered safe. Such efforts not only allow us to accurately predict the appropriate time for transitioning to secure PQC but also help forecast the duration for which each cryptographic algorithm will remain quantum-secure. Motivated by this research, we present a quantum circuit targeting Curve25519 [2], the elliptic curve used in Elliptic-Curve Cryptography (ECC), which is designed for use with Elliptic-Curve Diffie-Hellman (ECDH) key agreement protocols. Our quantum circuit results can be used to apply Shor's algorithm to attack key exchange mechanisms that rely on Curve25519. We propose an optimized quantum circuit for Curve25519 that minimizes qubit usage. To achieve this, we restructure the sequential operation order without affecting the final result and adjust the quantum circuit based on the qubit states. Additionally, we introduce techniques to minimize the use of temporary qubits in the circuit design. As a result, our quantum circuit demonstrates a reduction in the overall qubit count for the Curve25519 quantum circuit.

Section 2 explains basic quantum computing knowledge necessary for understanding the Curve25519 quantum circuit and provides an overview of the Curve25519 algorithm. Section 3 introduces the qubit optimization techniques for the Curve25519 quantum circuit. In Section 4, we present the estimated quantum resources for the quantum circuit after applying the qubit optimization techniques. Finally, the paper concludes with conclusions in Section 5.

2 Background

2.1 Quantum Computing

Quantum computers process data using quantum mechanical phenomena of qubits [11]. These quantum computers can express and process 2^n data at once with n qubits due to the superposition and entanglement properties of qubits, enabling faster calculations than classic computers. Qubits are controlled through quantum gates, and because of the reversible nature of quantum gates, inverse operations are possible. The following shows H, X, CNOT, and Toffoli matrices among representative quantum gates that control qubits:

$$H = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \qquad X = \frac{1}{\sqrt{2}}\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

$$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \qquad Toffoli = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

The quantum gate operation of each gate is shown in Figure 1.

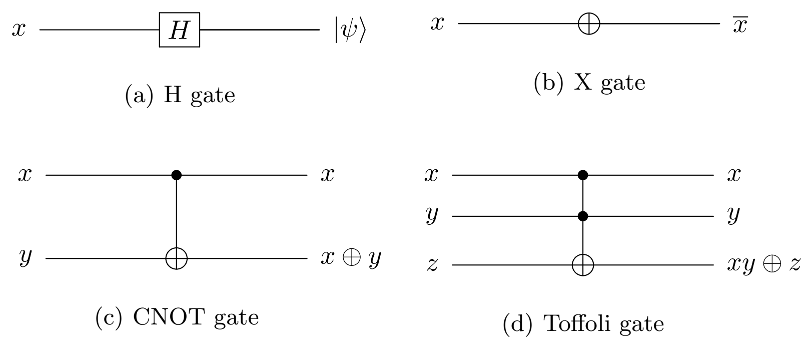

Fig. 1: Quantum gates

(a) **H gate**: The H gate works with a single qubit and makes the input a superposition.
(b) **X gate**: The X gate works with a single qubit and reversed the input.
(c) **CNOT gate**: The CNOT gate works with two qubits: control qubit and target qubit. The state of the target qubit y is reversed when the control qubit x is one.
(d) **CCNOT (Toffoli) gate**: The Toffoli gate works with three qubits: two control qubits and one target qubit. The state of the target qubit z is reversed when the control qubits x and y are both one.

2.2 Curve25519

Curve25519 is an elliptic curve designed for efficient and secure elliptic-curve cryptography (ECC), offering 128 bits of security(256-bit key size). The curve used is a Montgomery curve of the form $y^2 = x^3 + 486662x^2 + x$, operating on a finite field defined by the prime numbers $2^{255} - 192$. It is widely used for key exchange (via X25519) in protocols like ECDH and is optimized for speed and resistance to side-channel attacks, such as timing attacks. The curve operates over a prime field using a base point with the x-coordinate of 9 and is birationally equivalent to the Ed25519 curve, which is used for digital signatures. Due to security concerns, it has gained significant adoption, especially as an alternative to the P-256 curve. Algorithm 1 shows the seudo code to the main algorithm and subfunctions for Curve25519. All additions, subtractions, squaring, and multiplications used operate on the prime field $\mathbb{Z}/(2^{255}-19)$. In Function Curve25519, $P, P1, P2$, and R are Point structures, and k is a constant array. Line 10: perform a point doubling operation on T[0] to obtain T[1]. Line 11–15: The get_bit() function extracts the i-th bit of the scalar. Depending on the bit value (ki), point addition and point doubling are performed. The Functions RecoverY and ProToAff consist of addition, subtraction, squaring, multiplication, and inverse operations on the prime field $\mathbb{Z}/(2^{255} - 19)$.

Algorithm 1 Curve25519 algorithm

```
 1: Function Curve25519
 2: PointXZMulSecure(&P1, &P2, k, P)
 3: RecoverY(&P1, &P1, &P2, &P2, P, b)
 4: ProToAff(R, &P1)

 5: Function PointXZMulSecure
 6: Input: Scalar k, Point P, R1, R2
 7: xp ← P.x
 8: Initialize points T[0], T[1]
 9: T[0].x, T[0].y, T[0].z[0] ← xp, 0, 1
10: T[1] ← PointDbl(T[0])
11: for ($i = 253$ to $-1$) :
12:     ki ← get_bit(k, i)
13:     T[1-ki] ← PointAdd(T[1-ki], T[ki], xp)
14:     T[1-ki] ← PointAdd(T[1-ki], T[ki], xp)
15:     T[ki] ← PointDbl(T[ki])
16: R1 ← T[0]
17: R2 ← T[1]


18: Function RecoverY
19: Input: Points P1, P2 (in projective coordinates), Point P (in affine coordinates),
    Scalar b
20: x1, z1, x2, z2, x, y ← P1.x, P1.z, P2.x, P2.z, P.x, P.y
21: xr, yr, zr ← R.x, R.y, R.z
22: t1 ← x × x1
23: t1 ← t1 - z1
24: t2 ← z1 × x
25: t2 ← x1 - t2
26: t3 ← z2 × t1
27: t4 ← x2 × t2
28: t2 ← 4 × b
29: t2 ← t2 × y
30: t2 ← t2 × z2
31: t2 ← t2 × x2
32: t2 ← t2 × z1
33: zr ← t2 × z1
34: xr ← t2 × x1
35: t2 ← t3 + t4
36: t3 ← t3 - t4
37: yr ← t2 × t3


38: Function ProToAff
39: xp, yp, zp ← P.x, P.y, P.z
40: xr, yr, zr ← R.x, R.y, R.z
41: t1 ← 1/zp
42: yr ← yp×t1
43: xr ← xp×t1
```

3 Proposed method

In this section, we present a qubit-optimized quantum circuit for Curve25519. To reduce qubit usage in the functions used within Curve25519, we modify the original operation sequence and reuse qubits through inverse operations by adding minor quantum circuits. We analyze the operations of the existing algorithm and propose a new order and method to reduce qubit usage without affecting the overall algorithm. Additionally, some inverse operations further reduce the number of qubits required. The main operations used in Curve25519 consist of addition, subtraction, squaring, and multiplication in the prime field $\mathbb{Z}/p = \mathbb{Z}/(2^{255} - 19)$. Squaring($F_p\_sqr$) and multiplication($F_p\_mul$) use the same algorithm, the difference being that for squaring, the two target qubits are the same. We denote these basic operations as F_p_add, F_p_sub, F_p_sqr, and F_p_mul, respectively. Each quantum circuit is structured as follows:

1. F_p_add: This is an in-place operation, designed to perform modular arithmetic by determining the carry qubits for internal modular reduction. The basic addition used internally utilizes Cuccaro's adder [5]. Ultimately, it outputs the addition over the prime field $\mathbb{Z}/(2^{255} - 19)$. (Figure 2)

2. F_p_sub: This is an in-place operation, designed to perform modular arithmetic by determining the bottom qubits for internal modular reduction. The subtraction is performed by inverting the adder. Ultimately, it outputs the subtraction over the prime field $\mathbb{Z}/(2^{255} - 19)$. (Figure 3)

3. F_p_sqr, F_p_mul: These are out-of-place operations, with an additional reduction function (red256_func) implemented as a quantum circuit to perform internal modular reduction. The basic multiplication used internally utilizes Muñoz-Coreas's multiplier [10]. Ultimately, they output the squaring/multiplication over the prime field $\mathbb{Z}/(2^{255} - 19)$. (Figure 4)

The addition, subtraction, and multiplication (squaring) we propose for prime fields are universally applicable to various input lengths and primes. Moreover, since the internal adder and multiplier can be replaced, the design is highly versatile. In Figure 4, the F_p_mul and F_p_sqr perations use a reduction function called red256, which performs reduction operations for prime field arithmetic. The red256 quantum circuit we implemented is described in Algorithm 2. This quantum circuit is implemented with 5120 CCCNOT gates, 11848 Toffoli gates, 2445 CNOT gates, and 3 X gates. The mod value is defined as mod=[19 $\times$ 2,0,0,0,0,0,0,0,0], and mod2 is defined as mod2 $= (2^{255} - 19)$. The addition, subtraction, and multiplication used within red256 utilize the same Cuccaro's adder [5] and Coreas's multiplier [10] as those employed in prime field operations. Lines 11 to 13 represent a loop that only operates when the state of $carry_2$ is 1, making the value of N insignificant.

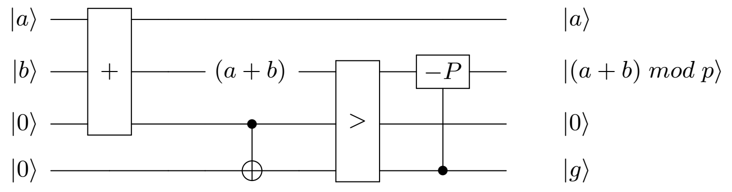

Fig. 2: Quantum circuit for modular addition in prime field $\mathbb{Z}/(2^{255}-19)$. $|a\rangle$ and $|b\rangle$ are qubit registers of length n, with two single clean ancilla qubits. One of these ancilla qubits is used as the carry qubit for addition, while the other is used for comparison. If the value of $|(a+b)\rangle$ exceeds p, then $-p$ is applied (where p is allocated to qubits and the inverse version of addition is used).

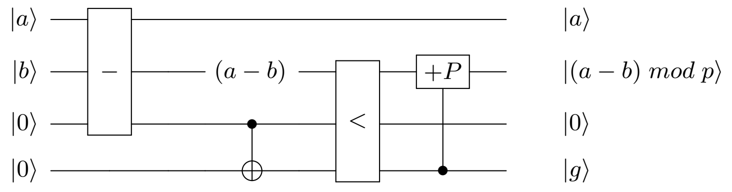

Fig. 3: Quantum circuit for modular subtraction in prime field $\mathbb{Z}/(2^{255}-19)$. $|a\rangle$ and $|b\rangle$ are qubit registers of length n, with two single clean ancilla qubits. One of these ancilla qubits is used as the borrow qubit for subtraction, while the other is used for comparison operations. If the value of $|(a-b)\rangle$ is less than p, then $+p$ is applied (similarly, p is allocated).

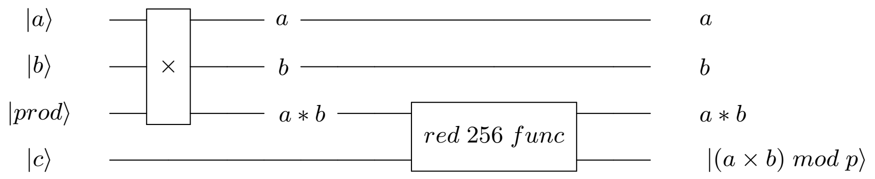

Fig. 4: Quantum circuit for modular multiplication in prime field $\mathbb{Z}/(2^{255}-19)$. $|a\rangle$ and $|b\rangle$ are qubit registers of length n, and in the case of squaring, $|a\rangle$ and $|b\rangle$ are identical. $prod$ is a qubit register of length $2n+1$ that stores the multiplication result $(a*b)$, which undergoes modular reduction through the $red256function$, as described in Algorithm 2. c is an n-length qubit register where the final modular multiplication (or squaring) result is stored.

Algorithm 2 Quantum circuit for red256 function

```
Input: $a, c$ ($a$: qubit in clean state, $c$: qubit to be reduction target)

 1: $a_{msb} = a[255 : 511]$                    //The upper 256 bits of $a$ (255 511)
 2: $a_{lsb} = a[0 : 255]$                      //The lower 256 bits of $a$ (0 255)
 3: $temp_1 \leftarrow$ multiplication(mod, $a_{msb}, temp_1$)
 4: $c \leftarrow$ addition($a_{lsb}$, $c$, $carry_1$)       //$carry_1$ stores the final carry of $a_{lsb}+c$

 5: $temp_1\{8\} = temp_1[233 : 255]$
 6: $carr_2 \leftarrow$ CNOT($carry_2, temp_1\{8\}$)
 7: $temp_1\{8\} \leftarrow$ addition($carry_2$, $temp_1\{8\}$)
 8: $temp_{msb} = temp_1\{8\}||temp_1[255 : 511]$
 9: prod $\leftarrow$ multiplication(mod, $temp_{msb}$, prod)

10: $c \leftarrow$ addition(prod[0:256], c)
11: for ($i = 0$ to $N$) :
12:     if Control qubit $carry_2 = 1$:
13:         $c \leftarrow$ subtraction(mod2, $c$)
```

Algorithm 3 shows the algorithmic sequence for the traditional Curve25519 PointAdd function. Each line represents operations over the prime field $\mathbb{Z}/(2^{255} - 19)$. While this algorithm performs efficiently on classical computers, it leads to inefficiencies on quantum computers, where qubit resources are constrained. In quantum circuits, results are stored in qubits; however, unlike classical bits, qubits cannot be easily overwritten with new values due to the nature of quantum superposition. As a result, if the original operations from Algorithm 3 are used, clean temp storage (in the form of qubits) is required for every assignment. This leads to wasted qubits, as clean qubits $t_{1,2}$ must be continuously reallocated during the execution of the PointAdd function. For instance, after using t_1 in line 1, if t_1 is needed again in line 7, its value cannot be overwritten, necessitating the allocation of a new qubit. Similarly, after using t_2 in line 2, clean qubits must be reallocated in lines 8 and 10 to continue the operation. This is a key difference between classical bits and quantum qubits, and optimizing this aspect is crucial for quantum circuit efficiency. To reduce the number of wasted qubits, we identified portions of the sequential operations that can be reordered, and we restructured them to reduce qubit usage in the flow of the quantum algorithm, resulting in Algorithm 4.

The $\clubsuit_1$ in line 2 is moved to line 5, and the $\clubsuit_2$ in line 7 is placed in line 9. This process was challenging, as it required placing operations in positions suitable for quantum circuit design without interfering with subsequent operations in the sequential algorithm. This process posed challenges, as it not only had to ensure that the algorithm's flow remained intact on a classical computer, but also required checking the qubit storage states on a quantum computer to ensure that no values overlapped. Ultimately, although this modified the sequential operations, it did not affect the overall result while optimizing qubit usage in the quantum circuit. In the revised algorithm, line 2 and line 7 from the original algorithm are placed in lines 5 and 9, respectively. The detailed reasoning for these changes is included in the explanation of Algorithm 5.

Algorithm 3 The operation sequence of the original PointAdd function

```
Input: $t_{1,2}$

 1: $t_1 = xp + zp$
 2: $\clubsuit_1$ $t_2 = xp - zp$
 3: $xr = xq - zq$
 4: $zr = t_1 \times xr$
 5: $t_1 = xq + zq$
 6: $xr = t_1 \times t_2$
 7: $\clubsuit_2$ $t_1 = xr - zr$
 8: $t_2 = xr + zr$
 9: $xr = t_2 \times t_2$
10: $t2 = t_1 \times t_1$
11: $zr = xd \times t_2$
```

Algorithm 4 The operation sequence of the modified PointAdd function

```
Input: $t_{1,2}$

 1: $t_1 = xp + zp$
 2: $xr = xq - zq$
 3: $zr = t_1 \times xr$
 4: $t_1 = xq + zq$
 5: $\clubsuit_1$ $t_2 = xp - zp$
 6: $xr = t_1 \times t_2$
 7: $t_2 = xr + zr$
 8: $xr = t_2 \times t_2$
 9: $\clubsuit_2$ $t_1 = xr - zr$
10: $t2 = t_1 \times t_1$
11: $zr = xd \times t_2$
```

Algorithm 5 Modified PointAdd function $\Rightarrow$ quantum circuit

```
 1: $t_1 = xp + zp$          $\Rightarrow$  $xp \leftarrow F_p\_add(xp, zp)$
 2: $xr = xq - zq$          $\Rightarrow$  $xq \leftarrow F_p\_sub(zq, xq)$
 3: $zr = t_1 \times xr$          $\Rightarrow$  $zr_{anc1} \leftarrow F_p\_mul(xq, xp)$
 4: $t_1 = xq + zq$          $\Rightarrow$  $xq \leftarrow F_p\_add(zp, xq)$
 5: $\clubsuit_1$ $t_2 = xp - zp$          $\Rightarrow$  $xp \leftarrow F_p\_sub(zp, xp)$ $F_p\_sub(zp, xp)$
 6: $xr = t_1 \times t_2$          $\Rightarrow$  $xr_{anc1} \leftarrow F_p\_mul(xq, xp)$
 7: $t_2 = xr + zr$          $\Rightarrow$  $xr_{anc1} \leftarrow F_p\_add(zr_{anc1}, xr_{anc1})$
 8: $xr = t_2 \times t_2$          $\Rightarrow$  $xr_{anc2} \leftarrow F_p\_sqr(xr_{anc1}, xr_{anc2})$
 9: $\clubsuit_2$ $t_1 = xr - zr$          $\Rightarrow$  $xr_{anc1} \leftarrow F_p\_sub(zr_{anc}, xr_{anc1})$ $F_p\_sub(zr_{anc}, xr_{anc1})$

10: $t2 = t_1 \times t_1$          $\Rightarrow$  Combine lines 10 and 11:
11: $zr = xd \times t_2$                 $zr_{temp} \leftarrow F_p\_mul(xr_{anc1}, xd)$,
                                         $zr_{anc2} \leftarrow F_p\_mul(xr_{anc1}, zr_{temp})$
    // **Inverse operations**
12: $xq \leftarrow F_p\_add(zq, xq)$
13: $xp \leftarrow F_p\_add(zp, xp)$
```

Algorithm 5 shows the quantum circuit implementation of PointAdd based on the restructured Algorithm 4. By converting the original Algorithm 3 into Algorithm 4, we can eliminate the need for temporary qubits t_1 and t_2. Although Algorithm 4 forms the basis for the quantum circuit, the detailed quantum circuit design is more complex. To reduce computational complexity, the target qubits and control qubits must be properly set for the qubits involved in the operations. This is later connected to the number of inverse operations at the end of Algorithm 5. We appropriately arranged the qubits to minimize the inverse operations and achieved this with two addition operations (line 12, line 13).

The change from line 2 in Algorithm 3 to line 5 in Algorithm 4 is a strategy to store the results of line 2 and line 7 in xp instead of t_1. t_1 is a qubit that requires reallocation, while xp is an already allocated qubit. In the quantum circuit design based on the original Algorithm 3, the result of line 1 would be stored in xp instead of t_1, but since xp must also be used in line 2's $t2$, this would cause a conflict. In other words, if the result of line 1 is stored in xp (or zp), xp cannot be used in line 2, requiring additional inverse operations or the allocation of temporary qubits. However, by using Algorithm 4, since $xp + zp$ (i.e., t_1) is already used in line 4, modifying the qubit values is possible. Therefore, as shown in Algorithm 5, we arranged for xp (i.e., t_1) to be reused in line 5 after it is no longer needed in line 3.

In line 1, after storing the result of $xp + zp$ in xp, we can simply perform $xp \leftarrow F_p\_sub(zp, xp)$ twice to obtain $xp - zp$. Since line 6 is an out-of-place operation, the result is stored in the clean state xr_{anc2} In line 7, the subtraction result $xr_{anc1} - zr_{anc1}$ is stored in xr_{anc1}. We moved to line 7 from the original algorithm to line 9, allowing xr_{anc1}, used in line 8, to be reused in line 9. This is the reason we restructured the algorithm. Finally, lines 10 and 11 were merged to reduce the qubits required for out-of-place operations (in purple text). In line 11, $zr = xd \times t_2$, and when combined with line 10, this becomes $zr = xd \times t_1 \times t_1$ Thus, we first compute $z = t_1 \times xd$ and then compute $z = z \times t_1$, resulting in $zr = xd \times t1 \times t1$. Lines 12 and 13 are inverse operations that restore xq and xp to their original states after the computations. Table 2 shows the qubit state transitions for each line of Algorithm 5, helping to illustrate the process. The final qubit states in line 13 represent the results stored in each qubit, which match the expected final output. This method reduces the total number of qubits by stacking computations on the qubits without using temporary qubits for intermediate value storage.

Algorithm 6 shows the quantum circuit implementation for PointDbl. As with previous operations, all calculations are performed over the prime field $\mathbb{Z}/(2^{255} - 19)$. Since all code lines in PointDbl are sequential, structural changes were not possible, so the original order of operations was maintained, but the allocation of the t1 qubit was removed. As a result, while the reduction of just one 256-bit t_1 on a classical computer may seem minor, on a quantum computer, where a clean qubit must be allocated every time t_1 is used, this translates to a reduction of 3×256 qubits for lines 1, 2, and 6. This was achieved by appropriately setting target and control qubits and adding F_p_add and F_p_sub operations at specific locations. In line 1, the result of the $xp + zp$ operation is stored in xp instead of t_1, and the next multiplication operation in line 2 is performed out-of-place, with the result $t_1 * t_1$ stored in t_2. Then, the result of line 3 is stored in xp, but since $xp + zp$ from line 1 was already used in line 2, two applications of F_p_sub are applied to produce the result of $xp - zp$. In lines 4

Table 1: Qubit state for Algorithm 5 (Each operation represents a prime field operation)

Line	Qubit State					
	xr_{anc1}	xr_{anc2}	zr_{anc1}	zr_{anc2}	xp	xq
1	-	-	-	-	$xp + zp$	xq
2	-	-	-	-	$xp + zp$	$xq - zq$
3	-	-	$xq * xp$	-	$xp + zp$	$xq - zq$
4	-	-	$xq * xp$	-	$xp + zp$	$xq + zq$
5	-	-	$xq * xp$	-	$xp - zp$	$xq + zq$
6	$xq * xp$	-	$xq * xp$	-	$xp - zp$	$xq + zq$
7	$xr_{anc1} + zr$	-	$xq * xp$	-	$xp - zp$	$xq + zq$
8	$xr_{anc1} + zr$	$(xr_{anc1})^2$	$xq * xp$	-	$xp - zp$	$xq + zq$
9	$xr_{anc1} - zr$	$(xr_{anc1})^2$	$xq * xp$	-	$xp - zp$	$xq + zq$
10 11	$xr_{anc1} - zr$	$(xr_{anc1})^2$	$xq * xp$	$xr_{anc1} * xr_{anc1} * xd$	$xp - zp$	$xq + zq$
12	$xr_{anc1} - zr$	$(xr_{anc1})^2$	$xq * xp$	$xr_{anc1} * xr_{anc1} * xd$	$xp - zp$	xq
13	$xr_{anc1} - zr$	$(xr_{anc1})^2$	$xq * xp$	$xr_{anc1} * xr_{anc1} * xd$	xp	xq

and 5, the results of each out-of-place multiplication are stored in zr_{anc1} and xr, respectively. Additionally, in line 6, the result of subtracting zr from t_2 is stored in t_2, and in line 7, the multiplication of t_2 and the constant c is stored in t_{2anc1}. In line 8, the result of multiplying t_1 and t_2 is stored in zr_{anc2}, and finally, in line 9, an F_p_add operation is added to restore xp (which was modified to $xp - zp$) to its original state. Figure 5 illustrates the quantum circuit for PointDbl, showing the intermediate results of the operations.

Algorithm 6 PointDbl function ⇒ quantum circuit

```
1: $t_1 = xp + zp$     ⇒   $xp \leftarrow F_p\_add(xp, zp)$
2: $t_2 = t_1 \times t_1$     ⇒   $t_2 \leftarrow F_p\_sqr(t_1)$
3: $t_1 = xp - zp$     ⇒   $xp \leftarrow F_p\_sub(zp, xp)$ $F_p\_sub(zp, xp)$
4: $zr = t_1 \times t1$     ⇒   $zr_{anc1} \leftarrow F_p\_mul(xp, zr_{anc1})$
5: $xr = t_2 \times zr$     ⇒   $xr \leftarrow F_p\_mul(t_2, zr)$
6: $t_1 = t_2 - zr$     ⇒   $t_2 \leftarrow F_p\_sub(zr, t_2)$
7: $t_2 = t_1 \times c$     ⇒   $t_{2anc1} \leftarrow F_p\_mul(t_2, c)$
8: $zr = t_1 \times t_2$     ⇒   $zr_{anc2} \leftarrow F_p\_mul(t_1, t_2)$

   // Inverse operations
9: $xp \leftarrow F_p\_add(zp, xp)$
```

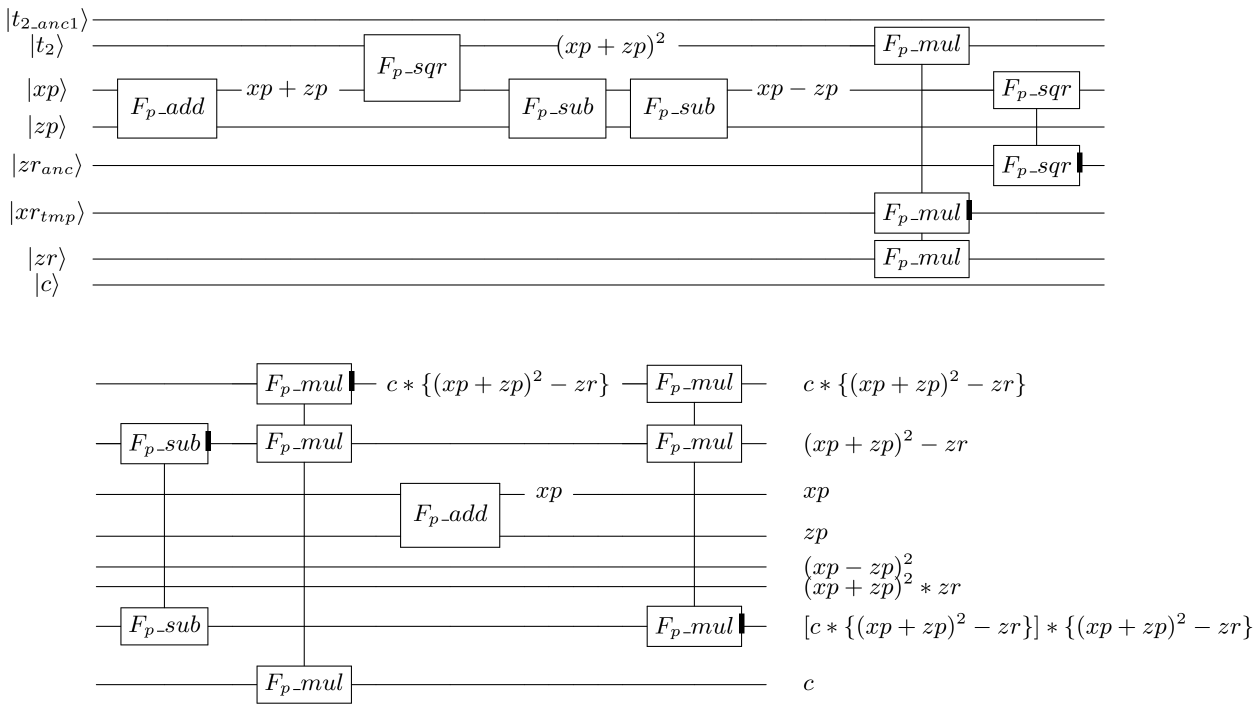

Fig. 5: Quantum circuit diagram of PointDbl. (The circuit is divided into two sections (connected from top to bottom))

4 Evaluation

This paper presents a qubit-optimized quantum circuit for Curve25519. We implemented the quantum circuit for the entire Curve25519, and Table 2 shows the estimated quantum resources for the optimized PointDbl and PointAdd functions. The qubits highlighted in red indicate the number of qubits reduced through optimization. The results are compared with the implementation of Curve25519 using the same arithmetic quantum circuits on the unoptimized original Curve25519 code. To our knowledge, our Curve25519 quantum circuit is the first implementation, making it difficult to compare with previous studies. When comparing the number of qubits, the PointDbl function reduces 746 qubits per execution. Since the PointDbl function is used 255 times in the overall Curve25519 operation, it reduces 190,230 qubits in total throughout the algorithm. In the case of PointAdd, 1,737 qubits are reduced per execution, and since it is used 254 times in the entire algorithm, it results in a total reduction of 441,198 qubits. Ultimately, our quantum circuit optimization demonstrates a significant reduction in the use of qubits. We propose quantum circuits for addition, subtraction, and squaring (multiplication) over prime fields, which are universally applicable to various prime field operations. Table 3 shows the estimated quantum resources for these prime field operations. These results are based on the prime field $\mathbb{Z}/(2^{255}-19)$ with 256 bit inputs, but the circuits can also be used for other n bit inputs and specific primes. Additionally, the adders and multipliers used internally can be replaced as needed.

Table 2: Quantum resources for PointDbl and PointAdd function

Function	Qubit	Quantum gates				
		CCCNOT	Toffoli	CNOT	X	Depth
PointDbl	12,080 (-746)	56,320	125,092	18,947	15	162,816
PointAdd	12,604 (-1737)	81,920	178,902	23,866	854	276,549

Table 3: Quantum resources for prime field $\mathbb{Z}/(2^{255}-19)$ operations

Function	Qubit	Quantum gates				
		CCCNOT	Toffoli	CNOT	X	Depth
F_p_add	780	5,120	10,762	1,035	0	16,907
F_p_sub	781	5,120	10,762	1,035	0	16,905
F_p_sqr, mul	3,121	5,120	12,104	2,702	3	19,653

5 Conclusion

In this paper, we proposed a quantum circuit for Curve25519, the elliptic curve used in Elliptic-Curve Cryptography (ECC). To achieve this, we implemented addition, subtraction, and multiplication over the prime field and applied them to the respective functions. To optimize qubit usage, we adapted the Point Doubling and Point Addition functions used in Curve25519 to be more suitable for quantum circuits. Additionally, we strategically placed inverse operations to eliminate or reduce the use of temporary qubits in the algorithm. As a result, our Curve25519 quantum circuit significantly reduces the number of qubits compared to existing quantum circuits for the same algorithm.

References

1. Banegas, G., Bernstein, D.J., Van Hoof, I., Lange, T.: Concrete quantum cryptanalysis of binary elliptic curves. Cryptology ePrint Archive (2020)

2. Bernstein, D.J.: Curve25519: new diffie-hellman speed records. In: Public Key Cryptography-PKC 2006: 9th International Conference on Theory and Practice in Public-Key Cryptography, New York, NY, USA, April 24-26, 2006. Proceedings 9. pp. 207–228. Springer (2006)
3. Bernstein, D.J., Lange, T.: Post-quantum cryptography. Nature **549**(7671), 188–194 (2017)
4. Chauhan, A.K., Sanadhya, S.K.: Quantum resource estimates of grover's key search on aria. In: Security, Privacy, and Applied Cryptography Engineering: 10th International Conference, SPACE 2020, Kolkata, India, December 17–21, 2020, Proceedings 10. pp. 238–258. Springer (2020)
5. Cuccaro, S.A.: A new quantum ripple-carry addition circuit. arXiv preprint quant-ph/0410184 (2004)
6. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: Proceedings of the twenty-eighth annual ACM symposium on Theory of computing. pp. 212–219 (1996)
7. Häner, T., Jaques, S., Naehrig, M., Roetteler, M., Soeken, M.: Improved quantum circuits for elliptic curve discrete logarithms. In: Post-Quantum Cryptography: 11th International Conference, PQCrypto 2020, Paris, France, April 15–17, 2020, Proceedings 11. pp. 425–444. Springer (2020)
8. Larasati, H.T., Kim, H.: Quantum circuit designs of point doubling for binary elliptic curves. arXiv preprint arXiv:2306.07530 (2023)
9. Liu, Q., Preneel, B., Zhao, Z., Wang, M.: Improved quantum circuits for aes: Reducing the depth and the number of qubits. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 67–98. Springer (2023)
10. Muñoz-Coreas, E., Thapliyal, H.: T-count optimized design of quantum integer multiplication. arXiv preprint arXiv:1706.05113 (2017)
11. Nielsen, M.A., Chuang, I.L.: Quantum computation and quantum information. Cambridge university press (2010)
12. Shi, H., Feng, X.: Quantum circuits of aes with a low-depth linear layer and a new structure. Cryptology ePrint Archive (2024)
13. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM review **41**(2), 303–332 (1999)
14. Song, G., Jang, K., Kim, H., Eum, S., Sim, M., Kim, H., Lee, W., Seo, H.: Speedy quantum circuit for grover's algorithm. Applied Sciences **12**(14), 6870 (2022)
15. Song, G., Jang, K., Kim, H., Lee, W.K., Hu, Z., Seo, H.: Grover on SM3. In: Information Security and Cryptology–ICISC 2021: 24th International Conference, Seoul, South Korea, December 1–3, 2021, Revised Selected Papers. pp. 421–433. Springer (2022)
16. Song, G., Jang, K., Kim, H., Seo, H.: A parallel quantum circuit implementations of LSH hash function for use with Grover's algorithm. Applied Sciences **12**(21), 10891 (2022)
17. Song, G., Jang, K., Seo, H.: Improved low-depth sha3 quantum circuit for fault-tolerant quantum computers. Applied Sciences **13**(6), 3558 (2023)
18. Song, G., Seo, H.: Grover on scrypt. Electronics **13**(16), 3167 (2024)