Quantum Circuit for Curve25519 with Fewer Qubits

Gyeongju Song

Contents

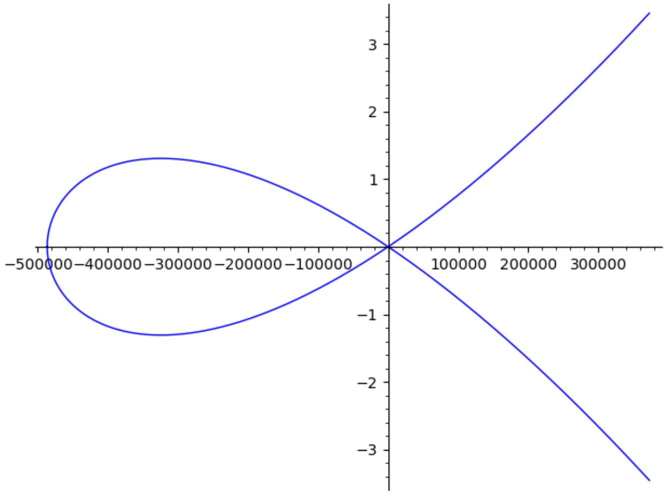
Background - Curve25519, Shor's algorithm, Quantum

Curve25519 quantum circuit

Evaluation

Background - Curve25519

- Elliptic Curve Proposed by D. J. Bernstein in 2005.
- Curve25519 is an elliptic curve designed for efficient and secure ECC.
- It provides 128-bit security (256-bit key size)
- Key exchange algorithm designed based on Curve25519: X25519
- **Curve:** $\boldsymbol{y^2 = x^3 + 488662x^2 + x}$
- **Prime field :** $\boldsymbol{2^{255} - 19}$



Algorithm 1 Curve25519 algorithm

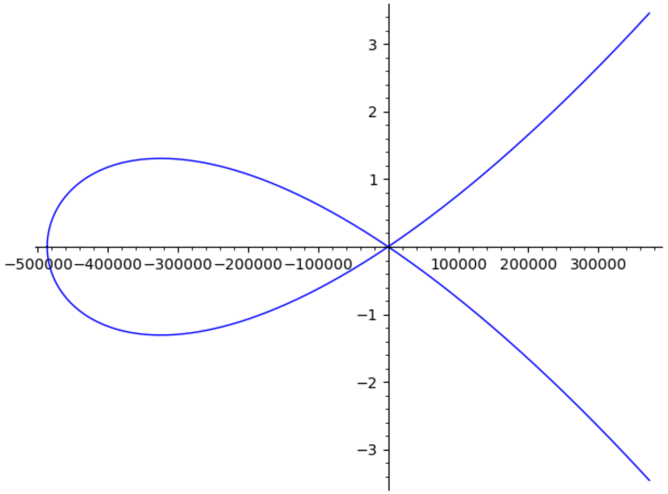
```
Function Curve25519
PointXZMulSecure(&P1, &P2, k, P)
RecoverY(&P1, &P1, &P2, &P2, P, b)
ProToAff(R, &P1)

Function PointXZMulSecure
Input: Scalar k, Point P, R1, R2
xp ← P.x
Initialize points T[0], T[1]
T[0].x, T[0].y, T[0].z[0] ← xp, 0, 1
T[1] ← PointDbl(T[0])
for (i = 253 to −1) :
    ki ← get_bit(k, i)
    T[1-ki] ← PointAdd(T[1-ki], T[ki], xp)
    T[1-ki] ← PointAdd(T[1-ki], T[ki], xp)
    T[ki] ← PointDbl(T[ki])
R1 ← T[0]
R2 ← T[1]
```

Background - Curve25519

- Elliptic Curve Proposed by D. J. Bernstein in 2005.
- Curve25519 is an elliptic curve designed for efficient and secure ECC.
- It provides 128-bit security (256-bit key size)
- Key exchange algorithm designed based on Curve25519: X25519
- **Curve:** $y^2 = x^3 + 488662x^2 + x$
- **Prime field :** $2^{255} - 19$

We implemented a Curve25519 quantum circuit for use in Shor's algorithm.



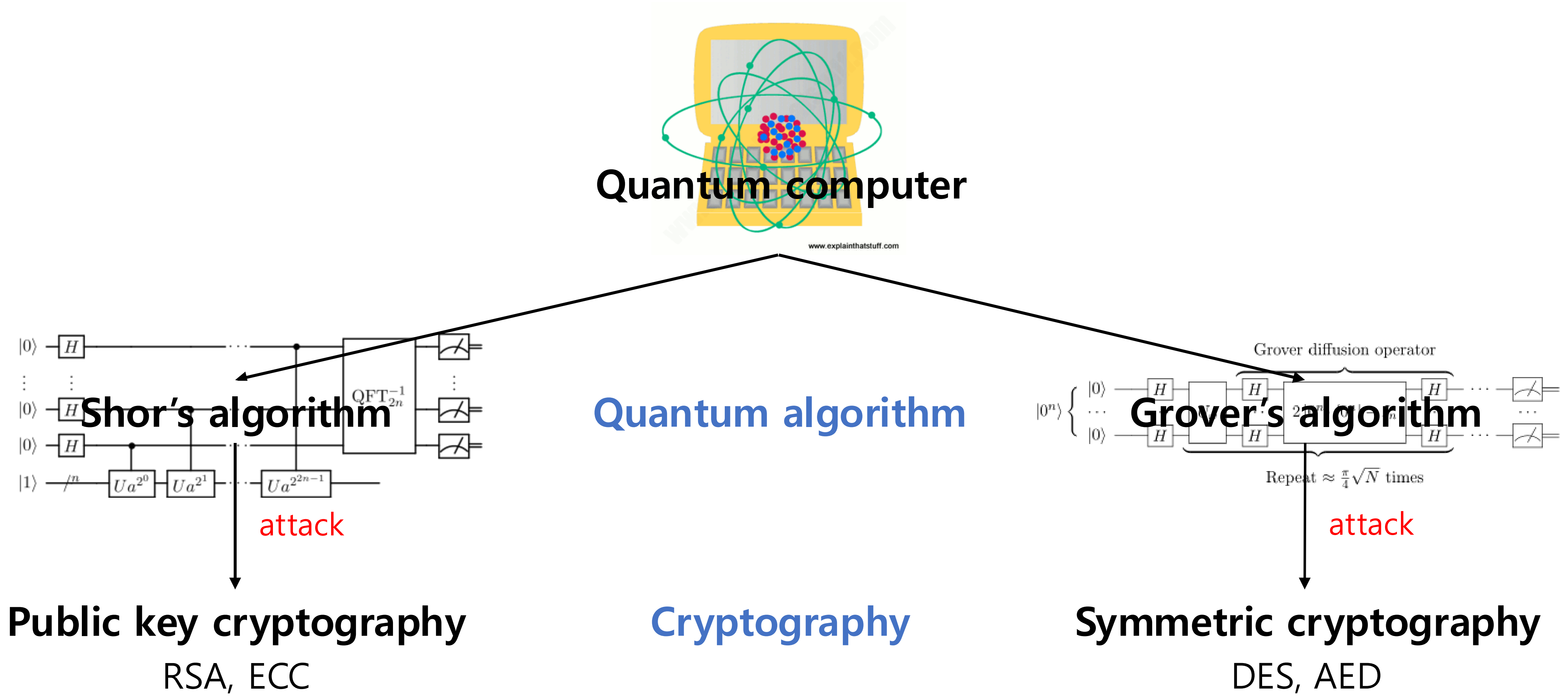
Algorithm 1 Curve25519 algorithm

```
Function Curve25519
PointXZMulSecure(&P1, &P2, k, P)
RecoverY(&P1, &P1, &P2, &P2, P, b)
ProToAff(R, &P1)

Function PointXZMulSecure
Input: Scalar k, Point P, R1, R2
xp ← P.x
Initialize points T[0], T[1]
T[0].x, T[0].y, T[0].z[0] ← xp, 0, 1
T[1] ← PointDbl(T[0])
for (i = 253 to −1) :
    ki ← get_bit(k, i)
    T[1-ki] ← PointAdd(T[1-ki], T[ki], xp)
    T[1-ki] ← PointAdd(T[1-ki], T[ki], xp)
    T[ki] ← PointDbl(T[ki])
R1 ← T[0]
R2 ← T[1]
```

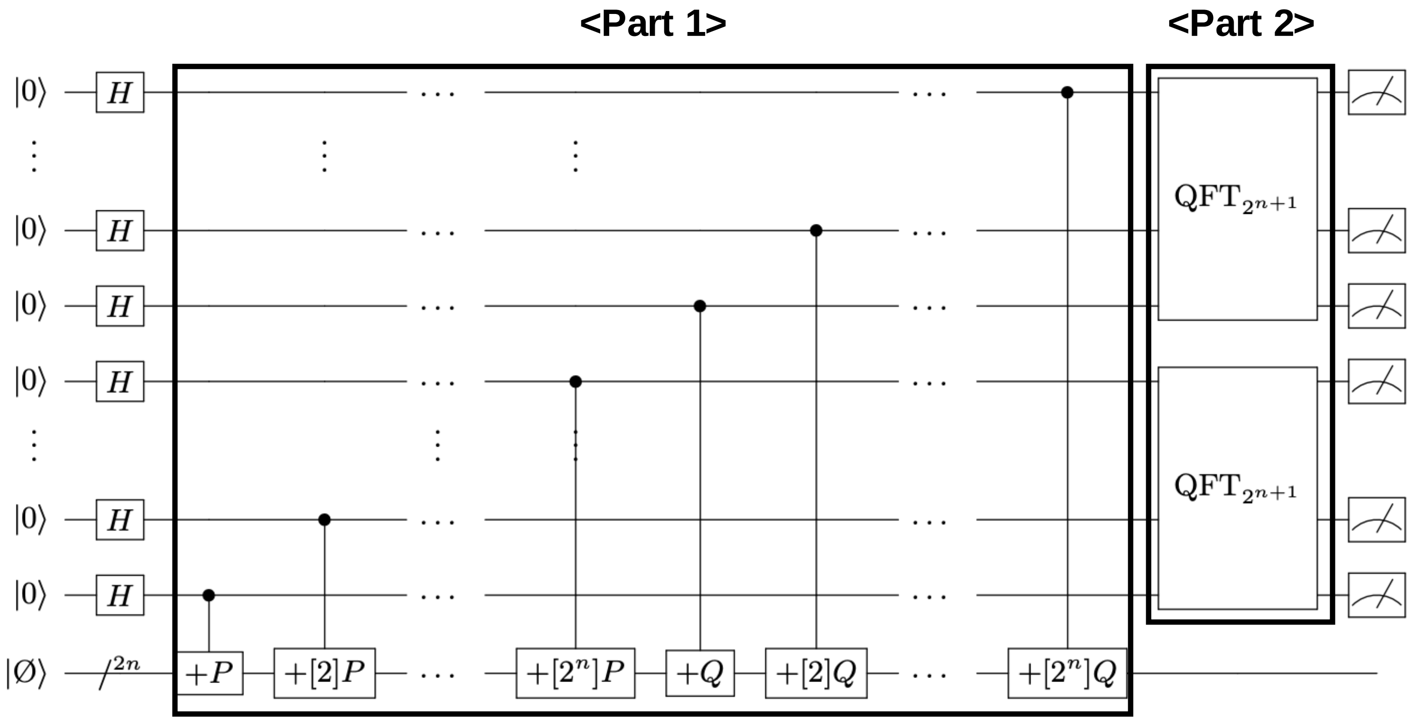
Background – Quantum algorithm

- Quantum Computer with cryptography



Background – Shor's algorithm

- **A Quantum Algorithm for Efficiently Solving the Elliptic Curve Discrete Logarithm Problem (ECDLP).**
- The ECDLP can be solved in polynomial time.
  - **Part 1 : Generate the state $|[k]P + [l]Q\rangle$ using intermediate states k and l through elliptic curve group operations.**
  - **Part 2 : Apply the Quantum Fourier Transform (QFT) to analyze the periodicity in the state $[k]P + [l]Q$, and recover the scalar m from $Q = [m]P$**

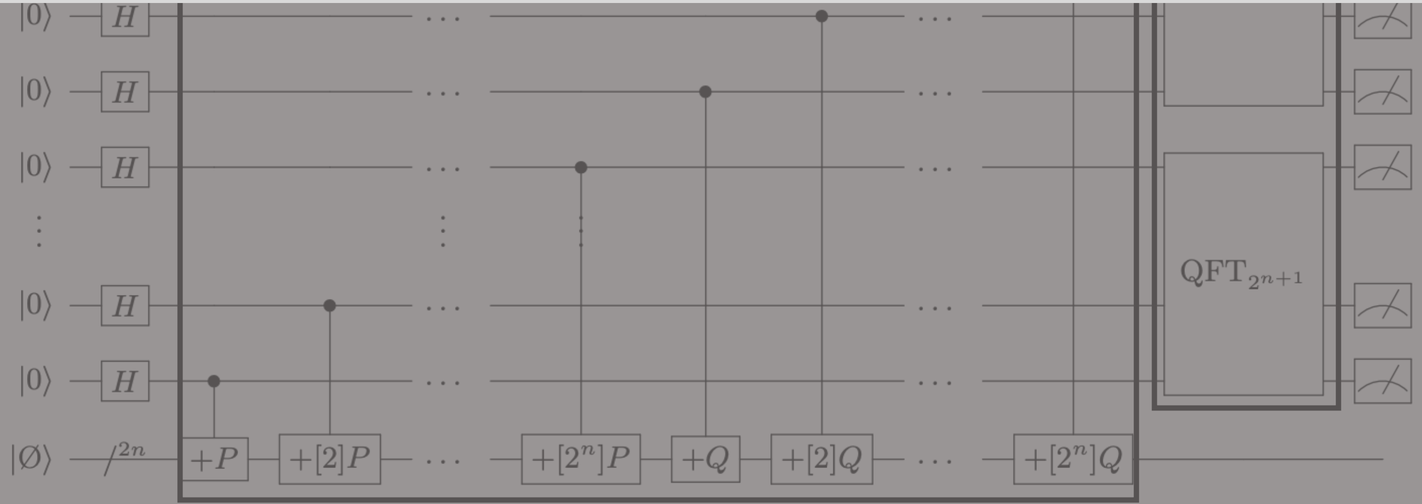


Background – Shor's algorithm

- **A Quantum Algorithm for Efficiently Solving the Elliptic Curve Discrete Logarithm Problem (ECDLP).**
- The ECDLP can be solved in polynomial time.
  - **Part 1 : Generate the state $|[k]P + [l]Q\rangle$ using intermediate states k and l through elliptic curve**

- Currently, the limitations of quantum computers **(number of qubits, errors)** make it challenging to perform real-world attacks using quantum computers.
- The quantum resources available must meet the requirements for the attack to be feasible.

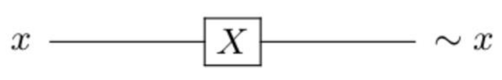
→ **Thus, research focused on optimizing the required qubit for such attacks is essential.**



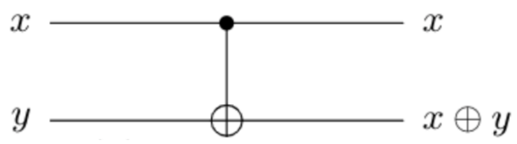
Background – Quantum programing

- Quantum circuit

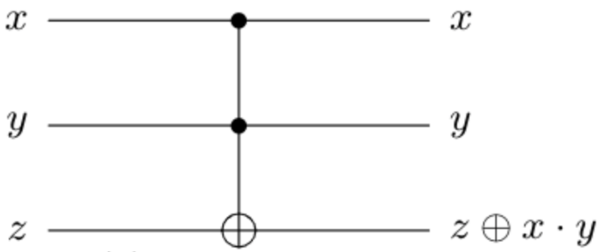
- In quantum computers, the state of qubits is controlled using quantum gates, which function similarly to digital logic gates in classical circuits.
- The input to a quantum gate can only be a qubit.
- Qubits in quantum operations can be divided into Control Qubit and Target Qubit
  - ■ **Control qubit** : Affects the operation but its value remains unchanged ■ **Target qubit** : The operation results are saved
- All operations are **reversible** (except for measurement)
- Common quantum gates include the Hadamard, X, CNOT, Toffoli, and Swap gates.
- A circuit composed of qubits and quantum gates is called a **Quantum Circuit**.



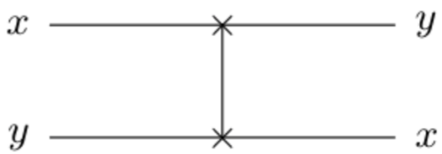
(a) X (NOT) gate



(b) CNOT gate



(c) Toffoli gate



(d) Swap gate

Our Contribution

- We propose a quantum circuit for Curve25519, designed to use fewer qubits.

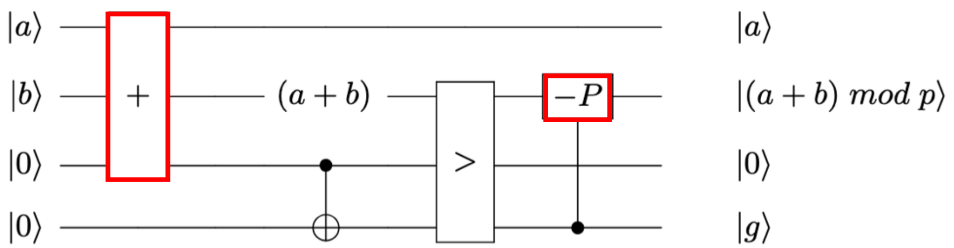
- The modular quantum circuits of addition, subtraction and multiplication are designed for general-purposes.
  - Initially, this is tailored to Curve25519.
  - However, this can also be applied to other prime fields.

- We adjusted the order of the PointAdd function in the sequential structure to reduce the number of qubits.
  - This modification allows us to reduce the number of temporary qubits. (t_1, t_2)

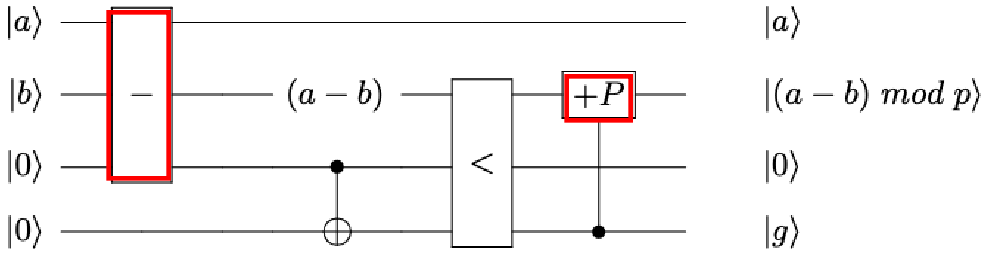
- PointDbl function is designed to maintain the order of all operations and to eliminate only t_1 qubits.
  - We eliminate the use of temporary qubit t_1 by using some techniques.

Components for Curve25519 quantum circuit

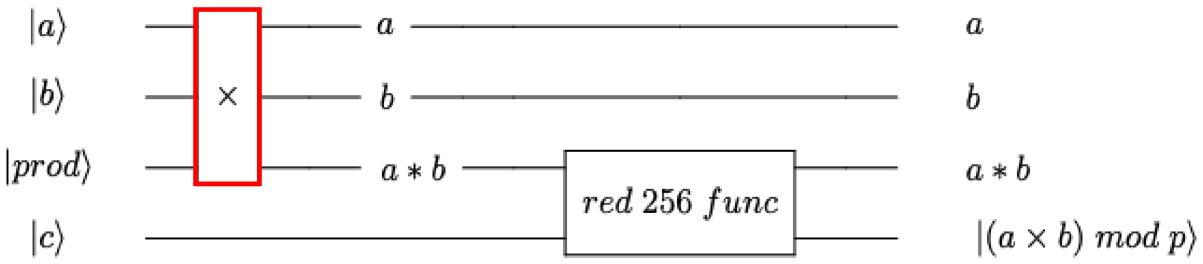
	Circuit	Ancilla	Size	Toffoli	Depth	
$+,-$	Cuccaro[1]	1	$9n-8$	$2n-1$	$2n+4$	In place
$\times$	Muñoz-Coreas[2]	$2n+1$	$7n^2-9n$	$5n^2-4n$	$3n^2-2$	Out-of-place



<Modular addition in $Z/(2^{255}-19)$>



<Modular subtraction in $Z/(2^{255}-19)$>

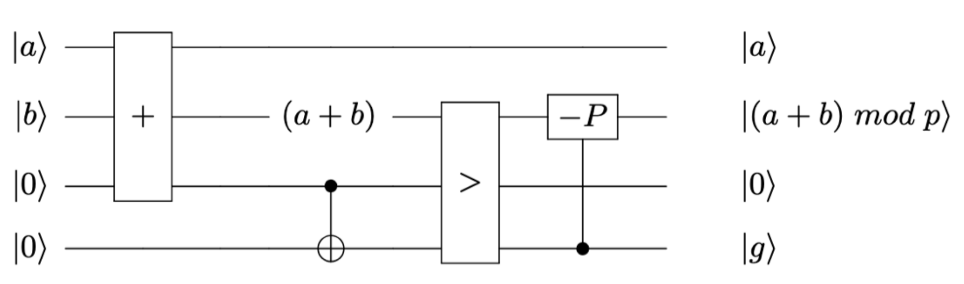


<Modular multiplication in $Z/(2^{255}-19)$>

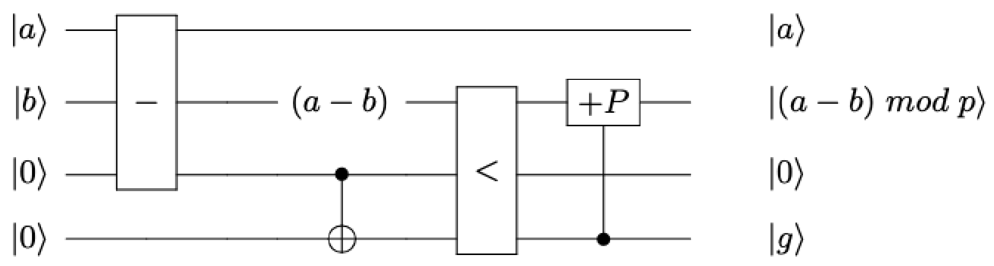
[1] S. A. Cuccaro, T. G. Draper, S. A. Kutin, and D. P. Moulton (2005), A new quantum ripplecarry addition circuit, The Eighth Workshop on Quantum Information Processing. Also on quant-ph/0410184.
[2] Muñoz-Coreas, E., & Thapliyal, H. (2017). T-count optimized design of quantum integer multiplication. arXiv preprint arXiv:1706.05113.

Components for Curve25519 quantum circuit

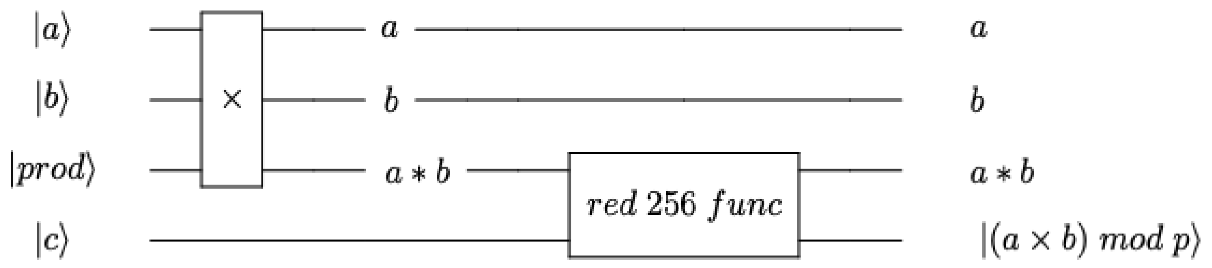
	Circuit	Ancilla	Size	Toffoli	Depth	
$+, -$	Cuccaro[1]	1	$9n-8$	$2n-1$	$2n+4$	In place
$\times$	Muñoz-Coreas[2]	$2n+1$	$7n^2-9n$	$5n^2-4n$	$3n^2-2$	Out-of-place



<Modular addition in $Z/(2^{255}-19)$>



<Modular subtraction in $Z/(2^{255}-19)$>

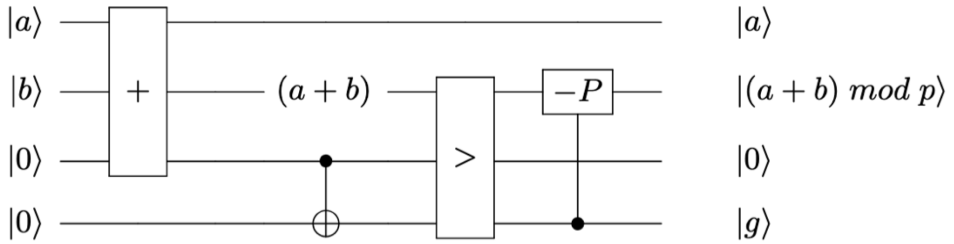


<Modular multiplication in $Z/(2^{255}-19)$>

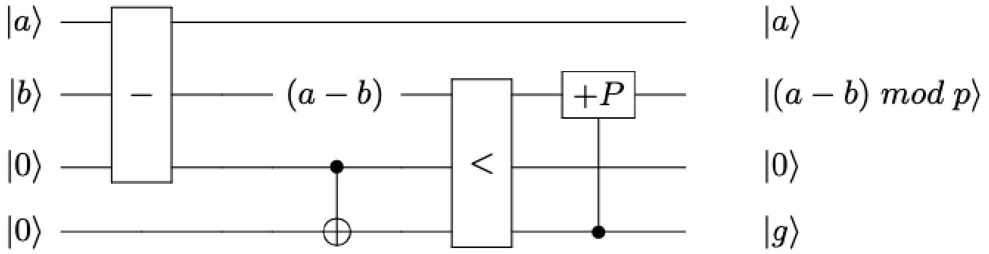
[1] S. A. Cuccaro, T. G. Draper, S. A. Kutin, and D. P. Moulton (2005), A new quantum ripplecarry addition circuit, The Eighth Workshop on Quantum Information Processing. Also on quant-ph/0410184.
[2] Muñoz-Coreas, E., & Thapliyal, H. (2017). T-count optimized design of quantum integer multiplication. arXiv preprint arXiv:1706.05113.

Components for Curve25519 quantum circuit

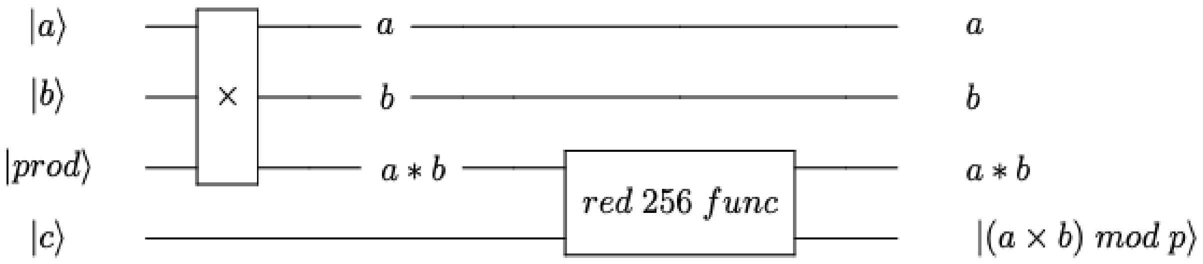
	Circuit	Ancilla	Size	Toffoli	Depth	
$+, -$	Cuccaro[1]	1	$9n - 8$	$2n - 1$	$2n + 4$	In place
$\times$	Muñoz-Coreas[2]	$2n + 1$	$7n^2 - 9n$	$5n^2 - 4n$	$3n^2 - 2$	Out-of-place



<Modular addition in $Z/(2^{255} - 19)$>



<Modular subtraction in $Z/(2^{255} - 19)$>

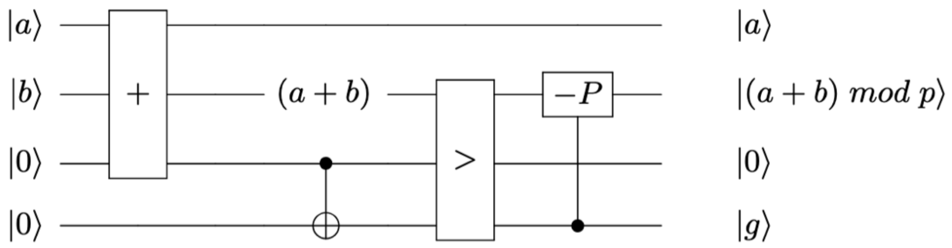


<Modular multiplication in $Z/(2^{255} - 19)$>

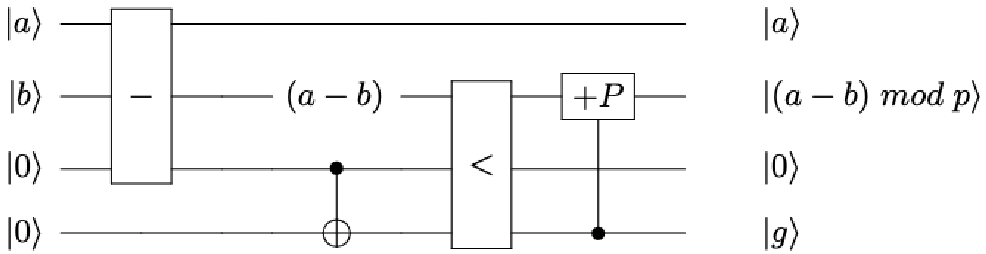
[1] S. A. Cuccaro, T. G. Draper, S. A. Kutin, and D. P. Moulton (2005), A new quantum ripplecarry addition circuit, The Eighth Workshop on Quantum Information Processing. Also on quant-ph/0410184.
[2] Muñoz-Coreas, E., & Thapliyal, H. (2017). T-count optimized design of quantum integer multiplication. arXiv preprint arXiv:1706.05113.

Components for Curve25519 quantum circuit

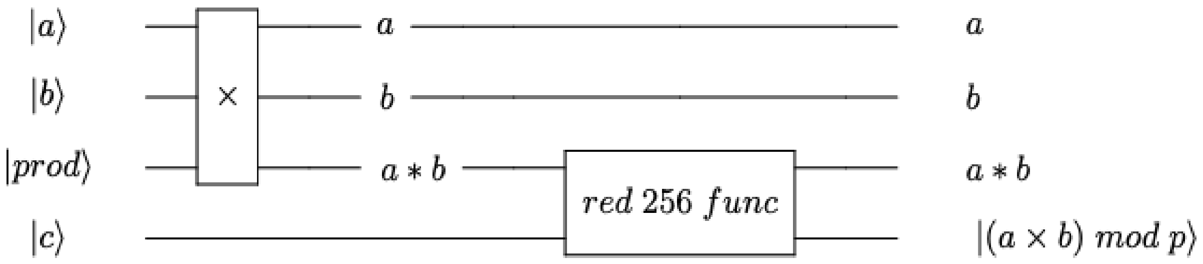
	Circuit	Ancilla	Size	Toffoli	Depth	
$+, -$	Cuccaro[1]	1	$9n - 8$	$2n - 1$	$2n + 4$	In place
$\times$	Muñoz-Coreas[2]	$2n + 1$	$7n^2 - 9n$	$5n^2 - 4n$	$3n^2 - 2$	Out-of-place



<Modular addition in $Z/(2^{255} - 19)$>



<Modular subtraction in $Z/(2^{255} - 19)$>



<Modular multiplication in $Z/(2^{255} - 19)$>

[1] S. A. Cuccaro, T. G. Draper, S. A. Kutin, and D. P. Moulton (2005), A new quantum ripplecarry addition circuit, The Eighth Workshop on Quantum Information Processing. Also on quant-ph/0410184.
[2] Muñoz-Coreas, E., & Thapliyal, H. (2017). T-count optimized design of quantum integer multiplication. arXiv preprint arXiv:1706.05113.

Curve25519 Quantum Circuit – PointAdd function

- We reordered the steps in the PointAdd algorithm to eliminate the use of t_1 and t_2 variables from the original algorithm.
- Moved Line 2 to Line 5 and Line 7 to Line 9.
  - **Hard work: 1) The sequential algorithm must be rearranged to be compatible with quantum circuits without affecting the computation.**
    2) The omission of t_1 and t_2 must not change the valid qubit.

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: $zr = xd \times t_2$

Original PointAdd function

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: $zr = xd \times t_2$

Modified PointAdd function

Curve25519 Quantum Circuit – PointAdd function

- Point 1. Reordered the algorithm steps to eliminate the use of temporary qubits (t_1 and t_2).
- Point 2. Set the target and control qubits appropriately for the qubits involved in the computation.

  → **This directly impacts the final number of inverse operations.**

- Point 3. Combined the order of Line 10 and Line 11 to reduce the qubits required for out-of-place operations (shown in purple text). By combining Line 10 and Line 11, the operation becomes $zr = xd * t_1 * t_1$. Therefore, $z = t_1 * xd$ is calculated first, followed by $zr = zr * t_1$

Algorithm 5 Modified PointAdd function ⇒ quantum circuit

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

10: $t2 = t_1 \times t_1$ ⇒ 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: $xp \leftarrow F_p\_add(zp, xp)$

Curve25519 Quantum Circuit – PointAdd function

- Point 1. Reordered the algorithm steps to eliminate the use of temporary (t_1 and t_2) qubits.
- Point 2. Set the target and control qubits appropriately for the qubits involved in the computation.

  → **This directly impacts the final number of inverse operations.**

- Point 3. Combined the order of Line 10 and Line 11 to reduce the qubits required for out-of-place operations (shown in purple text). By combining Line 10 and Line 11, the operation becomes $zr = xd * t_1 * t_1$. Therefore, $z = t_1 * xd$ is calculated first, followed by $zr = zr * t_1$

Algorithm 5 Modified PointAdd function ⇒ quantum circuit

1: $t_1 = xp + zp$ ⇒ $xp \leftarrow F_p\_add(xp, zp)$
2: $xr = xq - zq$ ⇒ $xq \leftarrow F_p\_sub(zq, xq)$
3: $zr = t_1 \times xr$ ⇒ $zr_{ancl} \leftarrow F_p\_mul(xq, xp)$
4: $t_1 = xq + zq$ ⇒ $zq \leftarrow F_p\_add(zp, xq)$
5: ♣$_1$ $t_2 = xp - zp$ ⇒ $xp \leftarrow F_p\_sub(zp, xp)$ $F_p\_sub(zp, xp)$
6: $xr = t_1 \times t_2$ ⇒ $xr_{ancl} \leftarrow F_p\_mul(xq, xp)$
7: $t_2 = xr + zr$ ⇒ $zr_{ancl} \leftarrow F_p\_add(zr_{ancl}, xr_{ancl})$
8: $xr = t_2 \times t_2$ ⇒ $xr_{anc2} \leftarrow F_p\_sqr(xr_{ancl}, zr_{anc2})$
9: ♣$_2$ $t_1 = xr - zr$ ⇒ $xr_{ancl} \leftarrow F_p\text{-}sub(zr_{anc}, xr_{ancl})$ $F_p\text{-}sub(zr_{anc}, xr_{ancl})$

10: $t2 = t_1 \times t_1$ ⇒ Combine lines 10 and 11:
11: $zr = xd \times t_2$
$zr_{temp} \leftarrow F_p\_mul(xr_{ancl}, xd)$,
$zr_{anc2} \leftarrow F_p\_mul(xr_{ancl}, zr_{temp})$

// Inverse operations
12: $xq \leftarrow F_p\_add(zq, xq)$
13: $xp \leftarrow F_p\_add(zp, xp)$

Perform inverse operations to return the qubits used in place of t_1 and t_2 to their required states.

Curve25519 Quantum Circuit – PointAdd function

- Point 1. Reordered the algorithm steps to eliminate the use of temporary (t_1 and t_2) qubits.
- Point 2. Set the target and control qubits appropriately for the qubits involved in the computation.
  - → **This directly impacts the final number of inverse operations.**
- Point 3. Combined the order of Line 10 and Line 11 to reduce the qubits required for out-of-place operations (shown in purple text). By combining Line 10 and Line 11, the operation becomes $zr = xd * t_1 * t_1$. Therefore, $z = t_1 * xd$ is calculated first, followed by $zr = zr * t_1$

Algorithm 5 Modified PointAdd function ⇒ quantum circuit

1: $t_1 = xp + zp$	⇒	$xp \leftarrow F_p\_add(xp, zp)$
2: $zr = xq - zq$	⇒	$xq \leftarrow F_p\_sub(zq, xq)$
3: $zr = t_1 \times zr$	⇒	$zr_{ancl} \leftarrow F_p\_mul(xq, xp)$
4: $t_1 = xq + zq$	⇒	$zq \leftarrow F_p\_add(zp, xq)$
5: ♣$_1$ $t_2 = xp - zp$	⇒	$xp \leftarrow F_p\_sub(zp, xp)$ $F_p\_sub(zp, xp)$
6: $xr = t_1 \times t_2$	⇒	$xr_{ancl} \leftarrow F_p\_mul(xq, xp)$
7: $t_2 = xr + zr$	⇒	$zr_{ancl} \leftarrow F_p\_add(zr_{ancl}, xr_{ancl})$
8: $xr = t_2 \times t_2$	⇒	$xr_{anc2} \leftarrow F_p\_sqr(xr_{ancl}, xr_{anc2})$
9: ♣$_2$ $t_1 = xr - zr$	⇒	$zr_{ancl} \leftarrow F_p\text{-}sub(zr_{anc}, xr_{ancl})$ $F_p\text{-}sub(zr_{anc}, xr_{ancl})$
10: $t2 = t_1 \times t_1$	⇒	Combine lines 10 and 11:
11: $zr = xd \times t_2$		$zr_{temp} \leftarrow F_p\_mul(xr_{ancl}, xd)$,
		$zr_{anc2} \leftarrow F_p\_mul(xr_{ancl}, zr_{temp})$
// **Inverse operations**		
12: $xq \leftarrow F_p\_add(zq, xq)$		Perform inverse operations to return the qubits used in place of t_1 and t_2 to their required states.
13: $xp \leftarrow F_p\_add(zp, xp)$		

Curve25519 Quantum Circuit – PointAdd function

- Point 1. Reordered the algorithm steps to eliminate the use of temporary (t_1 and t_2) qubits.
- Point 2. Set the target and control qubits appropriately for the qubits involved in the computation.

  → **This directly impacts the final number of inverse operations.**

- Point 3. Combined the order of Line 10 and Line 11 to reduce the qubits required for out-of-place operations (shown in purple text). By combining Line 10 and Line 11, the operation becomes $zr = xd * t_1 * t_1$. Therefore, $z = t_1 * xd$ is calculated first, followed by $zr = zr * t_1$

Algorithm 5 Modified PointAdd function ⇒ quantum circuit

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

10: $t2 = t_1 \times t_1$ ⇒
11: $zr = xd \times t_2$

Combine lines 10 and 11:
$zr_{temp} \leftarrow F_p\_mul(xr_{anc1}, xd)$,
$zr_{anc2} \leftarrow F_p\_mul(xr_{anc1}, zr_{temp})$

Original: (1) $xr\_sqr_{temp} = xr * xr_{temp}$ (2) $zr_{temp} = xr_{temp} * xd$
→ Require 3 temporary qubit (xr_sqr_{temp}, xr_{temp}, zr_{temp})
Combined: (1) $zr_{temp} = xr * xd$ (2) $zr = xr * zr_{temp}$
→ Require 1 temporary qubit (zr_{temp})

// Inverse operations
12: $xq \leftarrow F_p\_add(zq, xq)$
13: $xp \leftarrow F_p\_add(zp, xp)$

Perform inverse operations to return the qubits used in place of t_1 and t_2 to their required states.

Curve25519 Quantum Circuit – PointAdd function

- **line 1, line 5**: Since the result of xp+zp is stored in xp in line 1, simply applying $F_p\_sub(zp,\, xp)$ twice in line 5 will produce xp−zp.
- **Line 6**: Use xq and xp instead of t_1 and t_2.
- **line 9**: Store the result of $xr_{anc1} - zr_{anc1}$ in xr_{anc1} instead of t_1
  * Move line 7 of the original algorithm to line 9, allowing xr_{anc1} used in line 8 to be reused in line 9.

Algorithm 5 Modified PointAdd function ⇒ quantum circuit

1: $t_1 = xp + zp \quad \Rightarrow \quad xp \leftarrow F_p\_add(xp, zp)$
2: $xr = xq - zq \quad \Rightarrow \quad xq \leftarrow F_p\_sub(zq, xq)$
3: $zr = t_1 \times xr \quad \Rightarrow \quad zr_{anc1} \leftarrow F_p\_mul(xq, xp)$
4: $t_1 = xq + zq \quad \Rightarrow \quad xq \leftarrow F_p\_add(zp, xq)$
5: $♣_1\ t_2 = xp - zp \quad \Rightarrow \quad xp \leftarrow F_p\_sub(zp, xp)\ F_p\_sub(zp, xp)$
6: $xr = t_1 \times t_2 \quad \Rightarrow \quad xr_{anc1} \leftarrow F_p\_mul(xq, xp)$
7: $t_2 = xr + zr \quad \Rightarrow \quad xr_{anc1} \leftarrow F_p\_add(zr_{anc1}, xr_{anc1})$
8: $xr = t_2 \times t_2 \quad \Rightarrow \quad xr_{anc2} \leftarrow F_p\_sqr(xr_{anc1}, xr_{anc2})$
9: $♣_2\ t_1 = xr - zr \quad \Rightarrow \quad xr_{anc1} \leftarrow F_p\_sub(zr_{anc}, xr_{anc1})\ F_p\_sub(zr_{anc}, xr_{anc1})$

10: $t2 = t_1 \times t_1 \quad \Rightarrow$ 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: $xp \leftarrow F_p\_add(zp, xp)$

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

Curve25519 Quantum Circuit – PointAdd function

- **line 1, line 5**: Since the result of $xp+zp$ is stored in xp in line 1, simply applying $F_p\_sub(zp, xp)$ twice in line 5 will produce $xp-zp$.
- **Line 6**: Use xq and xp instead of t_1 and t_2.
- **line 9**: Store the result of $xr_{anc1} - zr_{anc1}$ in xr_{anc1} instead of t_1
  - * Move line 7 of the original algorithm to line 9, allowing xr_{anc1} used in line 8 to be reused in line 9.

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: $xp \leftarrow F_p\_add(zp, xp)$

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

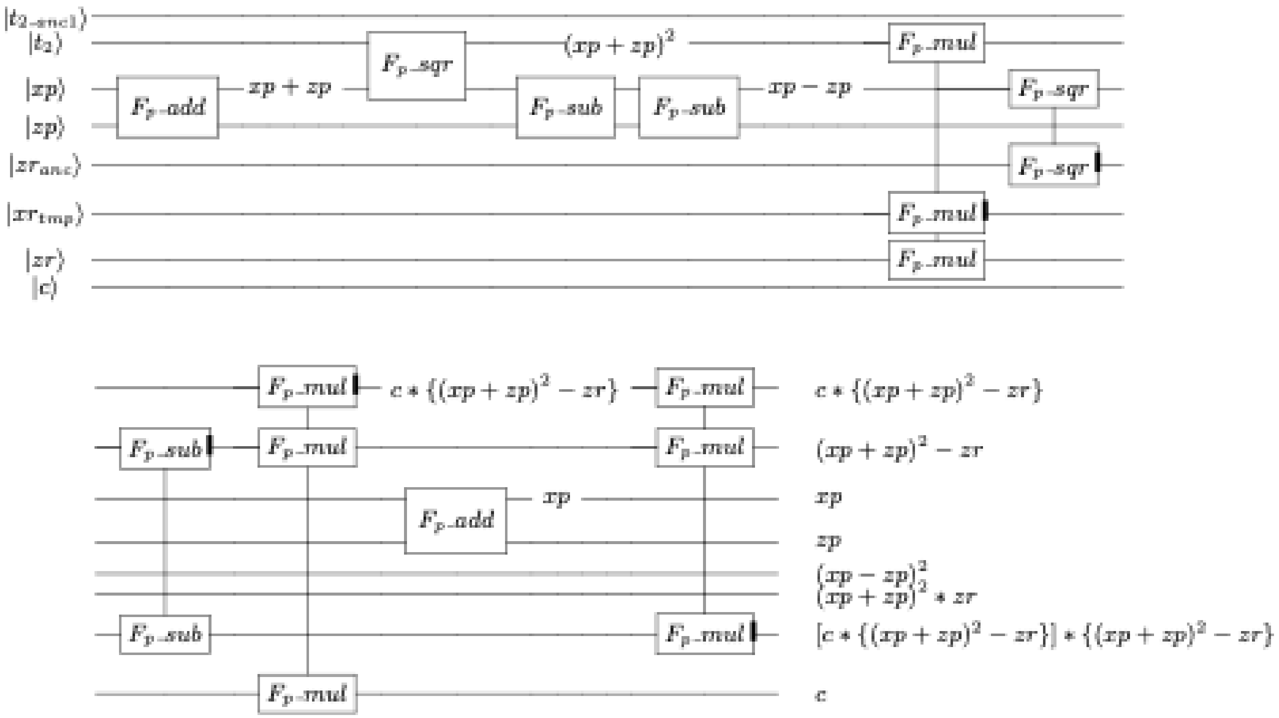
Line	Qubit State					
	xr_{anc1}	xr_{anc2}	zr_{anc1}	zr_{anc2}	xp	xq
1	-	-	-	-	$xp+zp$	xq
2	-	-	-	-	$xp+zp$	$xq-zq$
3	-	-	$xq*xp$	-	$xp+zp$	$xq-zq$
4	-	-	$xq*xp$	-	$xp+zp$	$xq+zq$
5	-	-	$xq*xp$	-	$xp-zp$	$xq+zq$
6	$xq*xp$	-	$xq*xp$	-	$xp-zp$	$xq+zq$
7	$xr_{anc1}+zr$	-	$xq*xp$	-	$xp-zp$	$xq+zq$
8	$xr_{anc1}+zr$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

In the final result of the algorithm, each qubit stores a valid value.

Curve25519 Quantum Circuit – PointDbl function

- The PointDbl function is challenging to reorder across all lines.
- Thus, we maintained the original sequential order of operations **but removed the use of the t_1 qubit.**
  - How? (1) Adjusting the target and control qubits for each operation.

    (2) Accordingly, F_p_add and F_p_sub operations are added at specific points to adjust the changing qubit.



Curve25519 Quantum Circuit – PointDbl function

- **Line 1**: The result of xp+zp is stored in $\boldsymbol{xp}$ instead of t_1.
- **Line 2**: Since multiplication is an out-of-place operation, the result of $t_1 \times t_1$ (i.e., $xp \times xp$) is stored in $\boldsymbol{t_2}$.
- **Line 3**: As xp+zp calculated in Line 1 was used in Line 2, applying F_p_sub twice generates the result of xp−zp (target qubit: $\boldsymbol{xp}$).
- **Lines 4** and 5: Store the results of the out-of-place multiplication operations in $\boldsymbol{zr_{anc1}}$ and xr, respectively.
- **Line 6**: Store the result of $t_2 - zr$ in $\boldsymbol{t_2}$, and in Line 7, store the result of the multiplication $t_2 \times c$ in $\boldsymbol{t_{2anc1}}$.
- **Line 8**: Store the result of the multiplication $t_1 \times t_2$ in $\boldsymbol{zr_{anc2}}$.
- **Line 9**: Add an F_p_add operation to restore $\boldsymbol{xp}$, which was modified to xp−zp, back to its previous value.

Algorithm 6 PointDbl function $\Rightarrow$ quantum circuit

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

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

Evaluation

- **This paper presents a qubit-optimized quantum circuit for Curve25519.**
- We implemented a quantum circuit for the entire Curve25519, and this table shows quantum resource estimates for the optimized PointDbl and PointAdd functions.
- To compare our idea, we compared it with the quantum circuits of PointDbl and PointAdd functions without applying our optimization technique. (Assuming the quantum circuits for the prime field operations used internally are identical.)
  - **PointDbl: Reduced 746 qubits (746 qubits × 255 iterations = a total reduction of 190,230 qubits)**
  - **PointAdd: Reduced 1,737 qubits (1,737 qubits × 254 iterations = a total reduction of 441,198 qubits).**

Function	Qubit	Quantum gates				
		CCCNOT	CCNOT	CNOT	X	Depth
PointDbl	12,080 (-746)	56,320	125,092	18,947	15	162,816
PointAdd	12,604 (-1737)	81,920	179,902	23,866	854	276,549

Future work

- We plan to compare Curve25519 with NIST standard curves (P-192, P-224, P-256, P-384, and P-521).

- For this, we will use the same adder and multiplier used in the quantum circuit implementation of the NIST standard curves.

- We expect that the quantum resources required for Curve25519 will be smaller.

Thank you ☺

E-mail: thdrudwn98@gmail.com