IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2025, No. 2, pp. 781–804. DOI:10.46586/tches.v2025.i2.781-804

New Quantum Cryptanalysis of Binary Elliptic Curves

Kyungbae Jang[1⋆], Vikas Srivastava[2], Anubhab Baksi[3∗], Santanu Sarkar[2] and Hwajeong Seo[1⋆‡]

[1] Division of IT Convergence Engineering, Hansung University, Seoul, South Korea
[2] Department of Mathematics, Indian Institute of Technology Madras, Chennai, India
[3] School of Physical & Mathematical Sciences, Nanyang Technological University, Singapore, Singapore

starj1023@gmail.com, vikas.math123@gmail.com, anubhab.baksi@ntu.edu.sg, santanu@iitm.ac.in, hwajeong84@gmail.com

Abstract. This paper improves upon the quantum circuits required for the Shor's attack on binary elliptic curves. We present two types of quantum point addition, taking both qubit count and circuit depth into consideration.

In summary, we propose an in-place point addition that improves upon the work of Banegas et al. from CHES'21, reducing the qubit count – depth product by more than 73% – 81% depending on the variant. Furthermore, we develop an out-of-place point addition by using additional qubits. This method achieves the lowest circuit depth and offers an improvement of over 92% in the qubit count – quantum depth product (for a single step).

To the best of our knowledge, our work improves from all previous works (including the CHES'21 paper by Banegas et al., the IEEE Access'22 paper by Putranto et al., and the CT-RSA'23 paper by Taguchi and Takayasu) in terms of circuit depth and qubit count – depth product.

Equipped with the implementations, we discuss the post-quantum security of the binary elliptic curve cryptography. Under the MAXDEPTH metric (proposed by the US government's NIST), the quantum circuit with the highest depth in our work is 2^{24}, which is significantly lower than the MAXDEPTH limit of 2^{40}. For the gate count – full depth product, a metric for estimating quantum attack cost (proposed by NIST), the highest complexity in our work is 2^{60} for the curve having degree 571 (which is comparable to AES-256 in terms of classical security), considerably below the post-quantum security level 1 threshold (of the order of 2^{156}).

Keywords: Binary Elliptic Curves · Shor's Algorithm · Quantum Cryptanalysis

⋆: This work is partially supported by the Institute for Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korean government (MSIT) (⟨Q|Crypton⟩, No. 2019-0-00033, "Study on Quantum Security Evaluation of Cryptography based on Computational Quantum Complexity", contribution to this project is about 25%) and the National Research Foundation of Korea(NRF) grant funded by the Korean government (MSIT) (No. RS-2023-00277994, "Quantum Circuit Depth Optimization for ARIA, SEED, LEA, HIGHT, and LSH of KCMVP Domestic Cryptographic Algorithms", contribution to this project is about 25%) and the Institute for Information & communications Technology Promotion(IITP) grant funded by the Korean government (MSIT) (No. 2018-0-00264, "Research on Blockchain Security Technology for IoT Services", contribution to this project is about 25%).

∗: This project is partially supported by the Wallenberg – NTU Presidential Post-doctoral Fellowship.

‡: Corresponding author.

Received: 2024-10-15 Accepted: 2024-12-15 Published: 2025-03-04

1 Introduction

The *Elliptic Curve Cryptography* (ECC) [Kob94] makes use of the algebraic structure of the elliptic curves over finite fields for public-key cryptography, and are crucial in modern cryptography. The potential use of elliptic curves in cryptography was suggested over three decades ago, independently by Miller [Mil85] and Koblitz [Kob87]. In the present time, ECC has become a staple component in modern electronic communication, such as, key exchange [DH22], or digital signatures [ElG85, JMV01].

The security of ECC is based on the difficulty of solving the discrete logarithms in elliptic curve groups (known as the *Elliptic Curve Discrete Logarithm Problem*, or ECDLP for short). The efficiency of ECC comes from the fact that the best-known algorithms [GG16] for solving ECDLP have an exponential time complexity relative to the input size. ECC is particularly appealing due to its efficiency. It offers strong security bound while requiring smaller key sizes than RSA (and other public key cryptographic systems) [RSA78]. For instance, Barker's recommendations [Bar20], on behalf of the National Institute of Standards and Technology (NIST) by the United States' government, indicate that a 224-bit elliptic curve offers comparable classical security as a 2048-bit RSA modulus. Table 1 provides estimated classical security comparison for the ECC and RSA (here, the order of generator point of the prime order subgroup of the elliptic curve group in number of bits are considered for ECC). Note that classical security (in bits) is less than the length of the key, unlike the (typical) symmetric key ciphers. We refer to [JSB+25, Appendix A] for a non-exhaustive collection of use-cases of ECC in modern electronic communication.

Table 1: Comparison of classical security between ECC and RSA.

Classical security (bits)	RSA✶	ECC✷
80	1024	160 – 223
112	2048	224 – 255
128	3072	256 – 383
192	7680	384 – 511
256	15360	≥ 512

✶: Product of two primes (in number of bits).

✷: Order of generator point (in number of bits).

We have seen the rapid progress of the quantum computers in the past few years. It is already well-known that ECC based systems cannot withstand the threat posed by the quantum computing paradigm. This realization, in turn, has motivated the researchers in the community to look for quantum-secure alternatives to the public-key ciphers, ultimately what is now known as the post-quantum cryptography (see [BL17] for reference). Specially, one may notice from [RNSL17] that; when compared with cryptographically relevant sizes (i.e., similar classical security), prime elliptic curves can be solved more easily on a quantum computer than factoring an RSA modulus based on the currently-known best results at that time. Recently, Gidney and Ekerå have reduced the cost of attacking RSA in quantum in [GE21]. Additionally, Banegas, Bernstein, van Hoof and Lange have presented a concrete quantum cryptanalysis of binary field ECCs [BBvHL20], demonstrating that it is easier to attack than prime field ECCs with comparable classical security [RNSL17].

Given the accelerating improvements in achieving a functional quantum computer relatively soon, one may wonder exactly how hard/easy it is to break ECC. In this work, we humbly strive to answer that question by presenting new results on the quantum cryptanalysis of binary field ECC, which has only become possible by standing on the shoulder of giants (including but not limited to [BBvHL20, PWLK22, TT23]).

Contribution

This paper improves quantum point addition on binary elliptic curves, with a primary focus on optimizing quantum circuit depth, while the number of qubits is considered as a secondary factor. Compared to previous works, we achieve the lowest Toffoli depth and circuit depth. For the product of depth and qubit count, we achieve improvements of the order of 73% – 81% in our in-place point addition and more than 92% in our out-of-place point addition. These improvements (see Table 9) are realized through optimizations at the following three logical levels. We begin by optimizing at the component level (Sections 3.2 and 3.3), where we use depth-efficient quantum circuits for binary field operations. This includes an out-of-place squaring technique (with a proposed/used optimization, as detailed in Section 3.2.1) and the depth-optimized multiplication method proposed by Jang et al. in [JKL+23]. Moving on to the combination level (Section 3.4), the division algorithm benefits from an inversion approach based on the *Fermat's Little Theorem* (FLT). Our inversion leverages a shallow technique that reuses qubits through reverse operations, keeping the circuit depth unchanged. Moreover, we achieve further improvements when multiple sequential multiplications are required, such as in the inversion process, as the multiplication method by Jang et al. [JKL+23] is particularly effective due to its capability to reuse the ancilla qubits. Finally, at the architecture level (Section 3.5), we present two implementations: *FLT-in* and *FLT-out*. We modify the in-place point addition method from [BBvHL20], FLT-in. We add a copy process for the control qubit in the Shor's quantum circuit and compress the conditional operations in the middle steps of [BBvHL20] by using a pre-computed result. We also develop the out-of-place point addition, FLT-out, which computes the result of point addition independently while preserving the input. This approach significantly reduces both the circuit depth and gate count by allowing the use of additional qubits.

We construct the quantum circuit required for running the Shor's algorithm (Section 4) using our point addition techniques and discuss its efficiency while evaluating the post-quantum security of binary ECC (Section 5).

The paper concludes in Section 6 with a note on future directions. Our source codes are written in ProjectQ [SHT18][1] and can be accessed online as an open-source project[2]. The extended version of this paper, which is available as [JSB+25], contains further discussion along with examples.

For clarity, the following major results are produced in this paper: Tables 3, 5, 6 (although this is a collection of algorithms), 7 and 8; Figures 3(b) and 3(c); and Algorithms 1 and 2; on top of our results being highlighted in Tables 5 and 9. Except for multiplication (Section 3.3), the rest of the algorithms are newly proposed in this paper; thus the algorithms used for squaring, inversion, and in-place & out-of-place point additions are our innovation.

2 Background

2.1 Binary Elliptic Curves

Binary elliptic curves, as the name suggests, are defined over binary fields $\mathbb{F}_{2^n}$. An ordinary binary elliptic curve of degree n (i.e., it is defined over a binary field of order 2^n) is given by

$$B_{a,b} : y^2 + xy = x^3 + ax^2 + b; \text{ where constants } a, b \in \mathbb{F}_{2^n} \text{ and } b \neq 0. \quad (1)$$

The above representation of binary elliptic curve is also called short/simplified Weierstrass form. The curve is well-defined for any value of a and b, as long as b is a non-zero element.

[1] See also https://projectq.ch/ and https://github.com/ProjectQ-Framework/ProjectQ.
[2] Accessible at https://github.com/starj1023/Binary_ECC.

The elliptic curve includes all points (x, y) which satisfy the elliptic curve equation (Equation 1) over $\mathbb{F}_{2^n}$ (where x and $y \in \mathbb{F}_{2^n}$). An elliptic curve group consists of the points on the elliptic curve, together with a *point at infinity* (denoted by ∞).

The point at infinity serves as the identity element in the elliptic curve group. In other words, given an arbitrary point $P \in B_{a,b}$, $P + \infty = \infty + P = P$. For any point $P = (u, v)$ on $B_{a,b}$, the inverse point, denoted as $-P$, is $(u, u + v)$, and it satisfies $P + (-P) = \infty$.

Let $P_1 = (u_1, v_1)$ and $P_2 = (u_2, v_2)$ be distinct points on $B_{a,b}$ such that $P_1 \neq \pm P_2$. Then, the point addition $P_1 + P_2$ yields the point $Q = (u, v)$ where

$$u = \delta^2 + \delta + a - u_1 - u_2 \text{ and } v = \delta(u_1 + u) - u - v_1, \text{ with } \delta = (v_2 + v_1)/(u_2 + u_1). \quad (2)$$

For the case where $P = (u_1, v_1)$ is a point on $B_{a,b}$ such that $P \neq -P$. Then, the point doubling operation, $P + P = 2P$ is represented by $Q = (u, v)$ where

$$u = \delta^2 + \delta + a = u_1^2 + b/u_1^2 \text{ and } v = \delta(u_1 + u) - u - v_1, \text{ with } \delta = u_1 + v_1/u_1. \quad (3)$$

In point addition, arithmetic operations such as addition, squaring, multiplication and division within the binary field are involved[3]. For faster point addition, efficient implementations of multiplication and division are particularly crucial in the classical computing.

Table 2: List of binary fields considered within context.

Degree	Modulus	Source/Reference
$n = 8$	$x^8 + x^4 + x^3 + x + 1$	CFADLNV [CFA+05]
$n = 16$	$x^{16} + x^5 + x^3 + x + 1$	
$n = 127$	$x^{127} + x + 1$	
$n = 163$	$x^{163} + x^7 + x^6 + x^3 + 1$	CMRRR [CMR+23]
$n = 233$	$x^{233} + x^{74} + 1$	
$n = 283$	$x^{283} + x^{12} + x^7 + x^5 + 1$	
$n = 571$	$x^{571} + x^{10} + x^5 + x^2 + 1$	

2.2 Key Establishment using ECC

The *Elliptic Curve Diffie-Hellman* (ECDH) key establishment protocol is an ECC-based anonymous key agreement protocol that allows two parties (say Alice and Bob), each possessing an elliptic curve public-private key pair, to establish a shared secret over an insecure communication channel. Private keys are randomly chosen integer scalars, while public keys are points on the curve. The security of ECDH depends on the variant of discrete logarithm problem known as Elliptic Curve Discrete Logarithm Problem (ECDLP). The ECDLP problem is states as follows: Let E be an elliptic curve defined over a finite field $\mathbb{F}_q$; and let $G, H \in E(\mathbb{F}_q)$ be points on the elliptic curve group such that $H \in \langle G \rangle$. The ECDLP asks to find the integer m, such that $H = [m]G$. The ECDLP is a special case of the discrete logarithm problem in which the cyclic group is represented by the group $\langle G \rangle$ of points on an elliptic curve.

At the first step, all parties must agree on all the elements defining the elliptic curve (also called, *domain parameters*) for the protocol. The binary field is determined by the pair $(n, m(x))$ (i.e., $\mathbb{F}_{2^n} = \mathbb{F}_2[x]/m(x)$). The binary elliptic curve $B_{a,b}$ is determined by the constants a, b used in the defining equation 1. The cyclic subgroup is defined by its

[3]Arithmetic operations on binary elliptic curves are well-suited for hardware implementation due to the structure of the binary field. Notably, in quantum circuit implementations, these operations are highly optimized. Indeed, the work in [BBvHL20] implement quantum circuits for binary curves and achieve greater efficiency compared to those for prime curves [RNSL17, HJN+20].

generator $G = (G_x, G_y)$. The *order of* G is defined as the smallest positive number p such that $pG = \infty$. For practical cryptography purposes, p is usually a prime number. The cofactor h is set as $|B_{a,b}(\mathbb{F}_{2^n})|/p$[4]. Note that order of elliptic curve group is given by hp. The domain parameters for binary ECC is given by $(n, m(x), a, b, G, p, h)$. Once the domain parameter has been decided, Alice and Bob proceed as follows. Alice randomly selects an integer sk_A from $\{2, \ldots, p-1\}$, and computes $PK_A = [sk_A]G$, and sends PK_A to Bob. Similarly, Bob selects an integer sk_B from $\{2, \ldots, p-1\}$, computes $PK_B = [sk_B]G$, and sends PK_B to Alice. Upon receiving PK_B and PK_A, Alice and Bob can compute the shared secret key SK independently as,

$$S = [sk_A]PK_B = [sk_B]PK_A = [sk_A][sk_B]G = [sk_A \cdot sk_B \pmod p]G$$

Both Alice and Bob arrive at the same S, thereby establishing the shared key.

2.3 Elliptic Curve Cryptography vs. Shor's Algorithm

The elliptic curve cryptography (ECC) is renowned for its security, largely due to the difficulty of solving the elliptic curve discrete logarithm problem (ECDLP) on a classical computer. Given a binary elliptic curve E over a finite field $\mathbb{F}_{2^n}$ and two points P and Q on E, the objective of ECDLP is to find an integer m such that

$$Q = [m]P,$$

where $[m]P$ denotes the scalar multiplication of P by m. This problem is computationally hard for classical algorithms.

However, the Shor's algorithm [Sho94] poses a significant threat to ECC by efficiently solving the discrete logarithm problem (on a powerful-enough quantum computer). The algorithm leverages quantum parallelism (which differs from classical parallelism) and a *Quantum Fourier Transform* (QFT) to achieve exponential speedup. The process involves the following steps:

Initialization: Allocate three quantum registers: the first two registers, k and ℓ, each of size $n+1$ qubits, are initialized to the $|0\rangle$ state, and the third register is used for point addition. After that, apply a Hadamard gate to each qubit in the first two registers (i.e., k and ℓ), resulting in a uniform superposition state,

$$|\psi\rangle = H^{\otimes n+1} |k, \ell\rangle^{\otimes n+1} = \frac{1}{2^{n+1}} \sum_{k,\ell=0}^{2^{n+1}-1} |k, \ell\rangle.$$

Conditional Addition: Based on the qubits in the first two registers, add the corresponding multiple of P and Q to the third register, implementing the map,

$$\frac{1}{2^{n+1}} \sum_{k,\ell=0}^{2^{n+1}-1} |k, \ell\rangle |[k]P + [\ell]Q\rangle.$$

In the Shor's circuit, only the first and second registers are measured, while the third register containing the state $|[k]P + [\ell]Q\rangle$ is discarded in the final stage (see Figure 1).

[4]Since p is the size of a subgroup of $B_{a,b}(\mathbb{F}_{2^n})$, it follows from the *Lagrange's theorem* that the number $h = \frac{1}{p}|B_{a,b}(\mathbb{F}_{2^n})|$ is an integer.

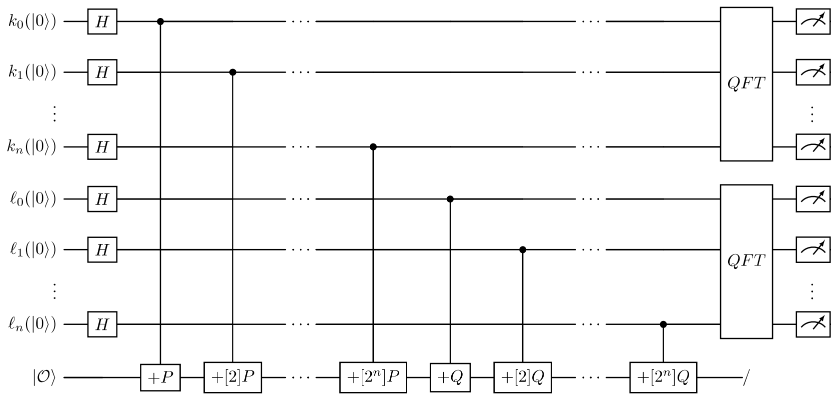


Figure 1: Circuit of Shor's algorithm for solving ECDLP.

Quantum Fourier Transform (QFT): Apply the QFT to the first two registers, each holding $n + 1$ qubits. The QFT involves phase shift gates and Hadamard gates, enabling the algorithm to determine the period r of the function. Using the period r, the discrete logarithm m can be recovered through classical post-processing, as described in [Sho94].

The quantum circuit for Shor's algorithm is shown in Figure 1. It illustrates the initialization of quantum registers, the conditional addition of elliptic curve points, the application of the QFT, and the measurement of the registers.

Shor's quantum circuit can be implemented using only a single control qubit by employing a semi-classical Fourier transform [GN96].

3 Quantum Circuit Construction for Binary Elliptic Curves

In this section, we present our quantum circuits for applying Shor's algorithm to binary ECC. We first introduce quantum circuits for binary field arithmetic. Following this, we design a depth-optimized quantum circuit for point addition, which is essential for Shor's algorithm in ECC, using the previously introduced quantum circuits for binary field arithmetic.

3.1 Addition & Binary Shift

In the integer domain ($\mathbb{Z}$), efforts to design efficient adders have been made in both classical and quantum computing, as demonstrated in [CDKM08, DKRS04, Dra00, TTK09]. However, in a binary field ($\mathbb{F}_{2^n}$), addition is equivalent to the XOR operation, and hence, it can be implemented using only n CNOT gates (incurring the depth of 1).

Shift and rotation operations can be implemented using a logical swap method, which rearranges the indices of qubits without the use of quantum swap gates. Even when swap gates are used for convenience in implementing shift and rotation operations in quantum circuits, they are often ignored in resource estimation. In this work, we implement binary shift operations by rearranging the indices of qubits, i.e., using logical swaps.

3.2 Squaring (Binary Non-Singular Matrix Multiplication)

Squaring over binary fields can be implemented using binary shift operations (e.g., $1111^2 = 1010101$). As described in Section 3.1, binary shift operations are generally implemented without additional cost, only the modular reduction of the shifted result is implemented using CNOT gates.

In previous works [BBvHL20, PWLK22, TT23], most squaring operations for point addition are implemented in-place using PLU factorization; except only two are implemented out-of-place. In-place quantum circuits compute the result directly in the input qubits by replacing the input with the output[5]. Thus, no ancilla qubit is required. However, this often leads to the higher circuit depth due to the constrained space.

In contrast, we consider the out-of-place approach, computing the result on newly allocated output qubits. The result of squaring can be represented as a matrix multiplication due to the linear nature of the squaring operation. For an element in the binary field $a \in \mathbb{F}_2[x]/m(x)$, the results of single squaring $a^2 \pmod{m(x)}$ as well as multiple squaring $a^{2^p} \pmod{m(x)}$ can be represented as binary non-singular matrices. For example, the results of a^2 and a^{2^2} in the binary field $\mathbb{F}_2[x]/(x^8+x^4+x^3+x+1)$ are represented as follows:

$$a = \begin{pmatrix} 0&0&0&0&0&0&0&1 \\ 0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&1&0&0 \\ 0&0&0&0&1&0&0&0 \\ 0&0&0&1&0&0&0&0 \\ 0&0&1&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0 \\ 1&0&0&0&0&0&0&0 \end{pmatrix}, \; a^2 = \begin{pmatrix} 0&1&0&1&0&0&0&1 \\ 1&1&0&1&0&0&0&0 \\ 0&0&1&0&0&0&1&0 \\ 1&1&1&1&0&0&0&0 \\ 1&0&0&1&0&1&0&0 \\ 0&1&1&0&0&0&0&0 \\ 0&0&1&0&1&0&0&0 \\ 1&1&0&0&0&0&0&0 \end{pmatrix}, \; a^{2^2} = \begin{pmatrix} 1&1&1&0&1&1&0&1 \\ 0&1&1&1&1&1&0&0 \\ 1&0&1&1&0&0&0&0 \\ 0&0&0&1&1&1&0&0 \\ 0&1&1&1&0&1&1&0 \\ 0&1&0&0&1&0&0&0 \\ 1&0&0&1&0&0&0&0 \\ 1&1&1&0&1&0&0&0 \end{pmatrix}.$$

CNOT gates operate on positions where there is 1. The input qubits (a) act as control qubits, while the newly allocated output qubits (out) serve as target qubits (CNOT (control, target)). For example, for the first row (corresponding to out_0) in the squaring matrix a^2, CNOT(a_0, out_0), CNOT(a_4, out_0), and CNOT(a_6, out_0) are applied. By constructing the matrix, we avoid redundant CNOT gate operations compared to the schoolbook squaring method. The total number of CNOT gates required is equal to the Hamming weight of the matrix[6]. Table 3 shows the quantum resources required for squaring using our proposed out-of-place approach. The quantum resources in terms of CNOT gates and circuit depth vary depending on the matrix, which is determined by the number of squaring operations (i.e., p). For $n = 8$ and 16, we set $p = 2$; and for the rest ($n = 127, \ldots, 571$), we set $p = 8$.

3.2.1 Out-of-Place Implementation

For the naïve out-of-place implementation of an $n \times n$ binary matrix, we first need to introduce n ancilla qubits (effectively doubling the qubit count) initialized at $|0\rangle$. Then the ancilla qubits are updated with the initial values of the first n qubits. This helps to retain the original matrix on the initial n qubits, while the n qubits are overwritten.

The out-of-place squaring method typically results in a low quantum depth than the in-place method, but requires newly allocated qubits for the output[7]. However, in our case, we do not allocate new qubits each time squaring is performed in division and point addition. Instead, we initialize the output qubits and reuse them in subsequent squaring operations (i.e., manage it at the combination of the components level). This is described in detail in Section 3.4.

In this work, we additionally propose/apply a *compiler-friendly* optimization to the out-of-place squaring operations, through shuffling the sequence. As a result, our approach

[5] This corresponds to the so-called s_1-*XOR* from [BDK+21].

[6] Note that, we only need 'Hamming weight − number of rows' CNOT gates to implement the naïve classical circuit. However, for the naïve quantum circuit, we first need to copy to the ancilla qubits, that would require additional 'number of rows' CNOT gates, totalling in 'Hamming weight' CNOT gates. See [RBC23, Example 1] for a toy example.

[7] Note that, the naïve out-of-place implementation is used for the linear layer (which is effectively a 320×320 binary non-singular matrix) in [OJBS24].

Table 3: Comparison of the quantum resources needed for squaring (matrix multiplication).

n	Method		#CNOT	#Qubit (Reuse)	Quantum depth
8	Out-of-place	Naïve	30	16 (8)	13
		Compiler-friendly※			8
16	Out-of-place	Naïve	80	32 (16)	22
		Compiler-friendly※			14
127	Out-of-place	Naïve	2529	254 (127)	175
		Compiler-friendly※			125
163	Out-of-place	Naïve	11094	326 (163)	270
		Compiler-friendly※			149
233	Out-of-place	Naïve	6743	466 (233)	158
		Compiler-friendly※			97
283	Out-of-place	Naïve	32762	566 (283)	459
		Compiler-friendly※			256
571	Out-of-place	Naïve	88183	1142 (571)	772
		Compiler-friendly※			468

※: Proposed and used in this work.

reduces quantum depth by more than 38% on average compared to naïve implementation. An overview of the compiler-friendly implementation is presented subsequently for the sake of completeness, though a thorough description can be found in [JSB+25, Appendix C].

Compiler-Friendly Implementation (Optimization for Quantum Depth) We explore the optimization of the out-of-place implementation of the squaring matrices in terms of quantum depth. In process, we introduce and use a deterministic algorithm for compiler-friendly implementation. Our optimization is motivated by the observation that quantum programming tools often fail to find an optimal circuit depth for CNOT gates when the same qubits are continuously involved (even when they can be parallelized with other CNOT gates). By reordering CNOT gate operations to avoid iterative calls to the same input and output qubits, the proposed method reduces the quantum depth.

3.2.2 In-Place Implementation

Our analysis on the in-place implementations of the matrices is summarized here[8]. We test with the two legacy algorithms that are known to produce in-place implementations, namely the Gauss-Jordan elimination and the PLU factorization (refer to [RBC23, Examples 4 and 5] for toy examples). In this work, we experiment with these two algorithms while incorporating the following changes/adjustments:

- We consider random row and column permutations, then the obtained implementation is adjusted accordingly.
- We run the inverse of the given matrix, and then reverse the sequence to get back the given matrix.
- To reduce the quantum depth, we adopt a randomized shuffling of once an implementation is obtained (then choose that one with the least quantum depth).

Despite this, we observe that the quantum depth and the CNOT count × quantum depth are high compared to what we get from the out-of-place method (the benchmarks are

[8]We are aware of the tool by [XZL+20] that finds in-place implementations for a given binary non-singular matrix. However, its source-code uses a data structure (hard-coded) that works for matrices up to dimension of 64×64 only (the authors clearly state that their tool is not expected to scale-up beyond that). Thus, in our context, this tool exclusively works for the trivial cases (viz., $n = 8$ and 16), and hence is not considered here. The same goes for the follow-up work by [YWS+24]. The SMT/MILP model proposed in [BKD21] does not scale-up either.

omitted here for brevity) for the cases $n \geq 127$. Consequently, we choose not to use the in-place implementations for this work. Whether or not it is possible to find more efficient in-place implementations is left as a future work.

3.3 Multiplication

Multiplication is used in the implementation of inversion and point addition, making its efficiency crucial. In [BBvHL20], van Hoof's space-efficient multiplication [vH19] is utilized. In [PWLK22], a modified version of van Hoof's multiplication is presented. Recently, in [TT23], Kim et al.'s Toffoli gate count-optimized and space-efficient multiplication [KKKH22] is adopted. These multiplications share the common feature of not using any ancilla qubits except for the input and output qubits. Stated in other words, for multiplying $h = f \cdot g$ of size n, only $3n$ qubits are used. Table 4 shows a comparison of the required quantum resources for multiplications from [vH19, PWLK22, TT23, JKL+23].

Table 4: Comparison of the quantum resources required for multiplication.

n	Source	#CNOT	#Toffoli	#Qubit	Toffoli depth	Depth	Full depth
8	vH [vH19]	200	27	24	N/A	124	N/A
	P+ [PWLK22]	102	27	24	N/A	82	N/A
	J+[JKL+23]※	237	27	81	1	22	34
16	vH [vH19]	678	81	48	N/A	365	N/A
	P+ [PWLK22]	655	81	48	N/A	286	N/A
	K+ [KKKH22]	974	64	48	N/A	405	N/A
	J+ [JKL+23]※	828	81	243	1	29	43
127	vH [vH19]	20632	2185	381	N/A	8769	N/A
	P+ [PWLK22]	20300	2183	381	N/A	7000	N/A
	K+ [KKKH22]	49040	737	381	N/A	6953	N/A
	J+ [JKL+23]※	24660	2185	6555	1	36	50
163	vH [vH19]	37168	4387	489	N/A	17906	N/A
	P+ [PWLK22]	36439	4355	489	N/A	13814	N/A
	K+ [KKKH22]	76262	992	489	N/A	10210	N/A
	J+ [JKL+23]※	46329	4387	13161	1	52	66
233	vH [vH19]	63655	6323	699	N/A	29530	N/A
	P+ [PWLK22]	60453	6307	699	N/A	19294	N/A
	K+ [KKKH22]	154892	1441	699	N/A	16383	N/A
	J+ [JKL+23]※	71197	6323	18969	1	42	56
283	vH [vH19]	89620	10273	849	N/A	41548	N/A
	P+ [PWLK22]	87929	10241	849	N/A	31894	N/A
	K+ [KKKH22]	224246	1784	849	N/A	22050	N/A
	J+ [JKL+23]※	110571	10273	30819	1	56	70
571	vH [vH19]	270940	31171	1713	N/A	121821	N/A
	P+ [PWLK22]	267771	31139	1713	N/A	95863	N/A
	K+ [KKKH22]	862604	3813	1713	N/A	61771	N/A
	J+ [JKL+23]※	337968	31171	93513	1	60	74

※: Used in this work.

In this work, we use a depth-efficient Karatsuba algorithm proposed by Jang et al. [JKL+23], which reduces the Toffoli depth to one by allocating an additional ancilla qubits. The Karatsuba algorithm can recursively reduce the size of the multiplication. In [JKL+23], the authors copied the operands of the divided (reduced-size) multiplications and performed them simultaneously, optimizing both the Toffoli depth and the full depth (for more details, the inquisitive readers are directed to [JKL+23, Figure 2]). We note that this multiplication method is particularly effective for implementing inversion and point addition, which require multiple multiplications, since the ancilla qubits used can be reused. This is described in Section 3.4.2.

3.4 Division using Fermat's Little Theorem (FLT)

For our quantum circuit implementation, we focus on the Fermat's Little Theorem (FLT)-based inversion algorithm to optimize circuit depth. We briefly review the method for computing the multiplicative inverse in a binary field $\mathbb{F}_{2^n}$ based on FLT. This theorem states that for any integer a and a prime number p, if a is not divisible by p, then $a^{p-1} \equiv 1 \pmod p$. From this, it follows that the multiplicative inverse of a modulo p is given by $a^{-1} \equiv a^{p-2} \pmod p$ since $a^{p-1} = a \cdot a^{p-2}$.

In binary fields $\mathbb{F}_{2^n}$, we can apply a similar concept. The elements of the field can be expressed as polynomials of degree $n-1$ with coefficients in $\mathbb{F}_2$. Given an element $a \in \mathbb{F}_{2^n}$, the multiplicative inverse a^{-1} can be computed as: $a^{-1} = a^{2^n-2}$.

3.4.1 Itoh-Tsujii Algorithm for Inversion

The Itoh-Tsujii algorithm [IT88] computes the inverse more efficiently instead of computing a^{2^n-2} in the naïve manner. The algorithm leverages on the following two mathematical observations:

Recursive Reduction: The problem of computing a^{2^n-2} can be recursively reduced by expressing it as: $a^{2^n-2} = \left(a^{2^{n-1}-1}\right)^2$.

Multi-Level Exponentiation: The exponentiation required at each level can itself be decomposed further using previously computed results: $a^{2^{2^t}-1} = \left(a^{2^{2^{t-1}}-1}\right)^{2^{2^{t-1}}} \cdot a^{2^{2^{t-1}}-1}$.

The Itoh-Tsujii algorithm can be outlined as follows:

1. Begin by expressing $n-1$ as a sum of powers of 2, i.e., $n-1 = \sum_{i=1}^{t} 2^{k_i}$, where $k_1 > k_2 > \cdots > k_t \geq 0$. This step is equivalent to writing $n-1$ in its binary form, where each k_i corresponds to the position of a binary 1. For example, $n = 12$ corresponds to $[k_1, k_2, k_3] = [3, 1, 0]$.

2. Compute the intermediate values $a^{2^{k_1}-1}, a^{2^{k_2}-1}, \ldots, a^{2^{k_t}-1}$ recursively. The key advantage here is that each successive value can be computed using previously calculated results, thus reducing the number of required multiplications.

3. Combine the intermediate results to compute the full exponentiation:

$$a^{2^n-2} = \left(\left(\left(\left(a^{2^{2^{k_1}}-1}\right)^{2^{2^{k_2}}} \cdot a^{2^{2^{k_2}}-1}\right)^{2^{2^{k_3}}} \cdot a^{2^{2^{k_3}}-1} \ldots\right)^{2^{2^{k_t}}} \cdot a^{2^{2^{k_t}}-1}\right)^2.$$

3.4.2 Depth-Optimized Quantum Circuit for Inversion

We implement the depth-efficient quantum circuit of Itoh-Tsujii-based inversion using the multiplication method from [JKL+23] and out-of-place squaring.

Let $n = 8$, which corresponds to $[k_1, k_2, k_3] = [2, 1, 0]$. Following the Itoh-Tsujii algorithm, we can compute the inverse of a as:

$$a^{-1} = \left(\left(\left(a^{2^{2^{k_1}}-1}\right)^{2^{2^{k_2}}} \cdot a^{2^{2^{k_2}}-1}\right)^{2^{2^{k_3}}} \cdot a^{2^{k_3}-1}\right)^2$$

Following the second observation in the Itoh-Tsujii algorithm (*Multi-Level Exponentiation*), we can represent the exponentiations $a^{2^{2^{k_1}}}$, $a^{2^{2^{k_2}}}$ and $a^{2^{2^{k_3}}}$ as follows:

$$a^{2^{2^{k_3}}-1} = a$$

$$A \rightarrow a^{2^{2^{k_2}}-1} = \left(a^{2^{2^{k_3}}-1}\right)^{2^{2^{k_3}}} \cdot a^{2^{2^{k_3}}-1} = a^2 \cdot a$$

$$B \rightarrow a^{2^{2^{k_1}}-1} = \left(a^{2^{2^{k_2}}-1}\right)^{2^{2^{k_2}}} \cdot a^{2^{2^{k_2}}-1} = A^{2^2} \cdot A$$

Figure 2 illustrates the proposed quantum circuit for inversion using the Itoh-Tsujii algorithm for $n = 8$. Here, M and S represent multiplication and squaring, respectively. In $M(result)$ and $S(result)$, the $result$ is the output derived from the operation. Here, $S^\dagger(result)$ denotes the reverse operation of $S(result)$ to initialize the output qubits (i.e., $result \rightarrow |0\rangle$). Additionally, the quantum circuit for $n = 16$ is given in [JSB+25, Appendix E].

Low-Depth Multiplication One may recall from Section 3.3 that the effectiveness of the multiplication method from [JKL+23] for implementing division and point addition. In [JKL+23], the authors mentioned a reuse technique that allows the initialization of allocated ancilla qubits with trivial overhead (see Section 3.3 in [JKL+23]). We note that this technique is particularly effective for implementing division and point addition, which require multiple multiplications. This is because the ancilla qubits used in the initial multiplication can be reused in subsequent operations without the need for additional qubits.

For $n = 8$, a total of four multiplications are performed to compute $A = a \cdot a^2$, $B = A \cdot A^2$, $a \cdot A^2$, and $(a \cdot A^2) \cdot B^{2^2}$. In our quantum circuit, only one ancilla set ($|M_{anc}\rangle$) is allocated for the initial multiplication ($A = a \cdot a^2$) and reused for the rest ($B = A \cdot A^2$, $a \cdot A^2$, $(a \cdot A^2) \cdot B^{2^2}$)[9]. We can achieve low-depth multiplications with only the initial qubit overhead. Note that this ancilla set will also be reused in the point addition (Section 3.5).

Low-Depth Squaring We use the out-of-place squaring method described in Section 3.2. In inversion, the result of squaring is used as an operand for multiplication but is no longer needed afterward (i.e., it is an intermediate value). Thus, we initialize the output qubits after their use and reuse them for subsequent squaring operations. For example, after the multiplication $A = a \cdot a^2$, the output qubits containing the result of squaring a^2 are initialized using a reverse operation.

However, while the reverse operation is often used in quantum implementations to reduce the number of qubits, it increases the circuit depth. For instance, in Figure 2, assume that the initialized output qubits from the reverse operation $S^\dagger(a^2)$ are reused in $S(A^{2^2})$. To reuse initialized output qubits from the reverse operation $S^\dagger(a^2)$, the squaring $S(A^{2^2})$ is delayed.

To address this delay, we allocate two sets of output qubits, $|S_{anc_0}\rangle$ and $|S_{anc_1}\rangle$, and use them alternately[10] In Figure 2, when the reverse operation of the squaring $S^\dagger(a^2)$ initializes

[9]Even in inversion and point addition, the multiplications are performed sequentially, making the reuse technique more effective. If the multiplications are not sequential (i.e., can be performed in parallel), each multiplication would require its own set of ancilla qubits (for parallelization). In that case, implementing space-efficient multiplications (e.g., [vH19, PWLK22, KKKH22]) in parallel might be more efficient, depending on the degree of parallelization.

[10]This concept is first introduced in [JBK+22] (referred to as the *shallow architecture*) to eliminate the depth overhead caused by the reverse operations of SubBytes in AES, and later adopted in [LPZW23, SF24]. In addition, it also serves as the inspiration behind the *interval architecture* of [JLO+24].

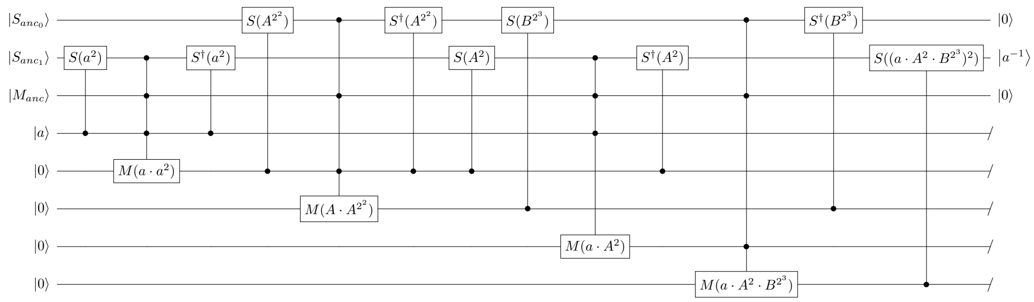


Figure 2: Proposed quantum circuit for inversion using FLT for $n = 8$.

the output qubits $|S_{anc_1}\rangle$, the current squaring $S(A^{2^2})$ is performed simultaneously using the other output qubits $|S_{anc_0}\rangle$. Thanks to this approach, low-depth squarings are achieved with only two sets of output qubits (a total of $2n$ qubits).

Comparison with Previous Division Algorithms In Table 5, we compare the quantum resources required for division, $h = h + f \cdot g^{-1}$ (involving one inversion, one multiplication, and one addition), with those in previous works[11]. Note that the division in Table 5 includes reverse operations[12] (resulting in twice the cost) to initialize ancilla qubits.

For [BBvHL20] and [PWLK22] in Table 5 (corresponding to $n = 163, 233, 283$ and 571), we use the re-estimated quantum resources from [TT23]. In this re-estimation, the multiplication methods in [BBvHL20] and [PWLK22] were replaced by the method proposed by Kim et al. [KKKH22]. The reasons for using the results from [TT23] are as follows. First, [BBvHL20] reported the upper bounds for gate count and circuit depth. For a fair comparison, we rely on the re-estimated results from [TT23], which, in fact, show improved performance.

Second, it is difficult to compare the results of [PWLK22] due to inconsistencies between [PWLK22, Tables 3 and 4 (multiplication), 5 (inversion), 6 (point addition)]. However, the authors in [TT23] provide corrected quantum resource estimates for [PWLK22].

As shown in Table 5, the Toffoli gate count is lower in previous works, but their CNOT gate count is higher compared to ours. This difference arises from the use of the Toffoli gate-optimized multiplication from [KKKH22] in their re-estimation. While our qubit count (M) is higher, we achieve the lowest circuit depth (D) and Toffoli depth. In terms of the product of circuit depth and qubit count (D-M), we observe a 31% improvement for $n = 8$ and exceeding $72\% - 78\%$ in other cases.

Although we are unable to compare the full depth and the full depth-qubit count product (as those are not reported in [BBvHL20, PWLK22, TT23, KKKH22]), we achieve further improvements in these metrics. This is because our low Toffoli depth results in a slight increase from the depth (D) to the full depth (after the decomposition of Toffoli gates), which is not the case for the previous works.

3.5 Point Addition

In Shor's quantum circuit, conditional point additions $|[k]P + [\ell]Q\rangle$ are performed according to the control qubits in the first register (i.e., k's and ℓ's in Section 2.3 and Figure 1).

[11] In [KH23], the authors present a quantum GCD-based inversion algorithm. However, we do not include their algorithm in Table 5, as they do not provide estimates for the number of CNOT gates and the circuit depth.

[12] These reverse operations are performed in the division for the in-place point addition (FLT-in, Algorithm 1), but not for the out-of-place point addition (FLT-out, Algorithm 2).

Table 5: Comparison of the quantum resources required for division.

n	Source		#Toffoli	#CNOT	#Qubit (M)	Toffoli depth	Depth (D)	Full depth	D-M
8	B+ [BBvHL20]	(FLT)	243	2212	56	N/A	1314	N/A	73584
		(GCD)	3641	1516	67	N/A	4113	N/A	275571
	This paper	(FLT)	270	2762	213	10	238	358	50694
16	B+ [BBvHL20]	(FLT)	1053	10814	144	N/A	5968	N/A	859392
		(GCD)	10403	5072	124	N/A	12145	N/A	1505980
	This paper	(FLT)	1134	13510	777	14	458	654	182595
127	B+ [BBvHL20]	(FLT)	50255	502870	1778	N/A	203500	N/A	361823000
		(GCD)	277195	227902	903	N/A	378843	N/A	342095229
	This paper	(FLT)	52440	681717	30971	24	2423	2645	75042733
163	B+ [BBvHL20]	(FLT)	18848	1601716	1956	N/A	342516	N/A	669961296
		(GCD)	438766	414586	1156	N/A	510628	N/A	590285968
	P+ [PWLK22]	(FLT)	18848	1558180	3097	N/A	300924	N/A	931961628
	T+ [TT23]	(FLT-basic)	18848	1557528	2771	N/A	300920	N/A	833849320
		(FLT-extended)	18848	1579944	1956	N/A	310830	N/A	607983480
	This paper	(FLT)	87740	1176499	53133	20	2801	3046	148825533
233	B+ [BBvHL20]	(FLT)	30261	3374430	3029	N/A	459709	N/A	1392458561
		(GCD)	823095	834256	1646	N/A	992766	N/A	1634092836
	P+ [PWLK22]	(FLT)	30261	3346938	4660	N/A	435001	N/A	2027104660
	T+ [TT23]	(FLT-basic)	30261	3345540	3961	N/A	434995	N/A	1723015195
		(FLT-extended)	30261	3353750	3029	N/A	437747	N/A	1325935663
	This paper	(FLT)	139106	2114587	82898	22	3476	3716	288153448
283	B+ [BBvHL20]	(FLT)	41032	5644678	3962	N/A	985710	N/A	3905383020
		(GCD)	1194498	1222600	1997	N/A	1449098	N/A	2893848706
	P+ [PWLK22]	(FLT)	41032	5492126	6226	N/A	837106	N/A	5211821956
	T+ [TT23]	(FLT-basic)	41032	5489296	4811	N/A	837096	N/A	4027268856
		(FLT-extended)	41032	5502090	3962	N/A	840612	N/A	3330504744
	This paper	(FLT)	246552	3705491	144671	24	5412	5685	782959452
571	B+ [BBvHL20]	(FLT)	102951	26043772	9136	N/A	4401901	N/A	40215767536
		(GCD)	4434315	4857244	4014	N/A	5602181	N/A	22487154534
	P+ [PWLK22]	(FLT)	102951	25189566	14275	N/A	3556815	N/A	50773534125
	T+ [TT23]	(FLT-basic)	95325	23458648	10849	N/A	3433263	N/A	37247470287
		(FLT-extended)	95325	23514068	8565	N/A	3456469	N/A	29604656985
	This paper	(FLT)	872788	14649243	500450	28	11723	12087	5866775350

In conditional point addition, if the control qubit q is 1, the point addition $P_3(x_3, y_3) = P_1(x_1, y_1) + P_2(x_2, y_2)$ is computed; otherwise, the input $P_1(x_1, y_1)$ remains unchanged.

In [BBvHL20], the authors presented an in-place point addition algorithm ([BBvHL20, Algorithm 3]) by modifying Algorithm 1 from [RNSL17] to suit the binary case, and subsequent works [PWLK22, TT23] have followed this algorithm.

In this work, we introduce the in-place point addition described in Algorithm 1, modified from [BBvHL20, Algorithm 3] by us; with the presented quantum circuits for addition, squaring, multiplication, and division. This approach has the advantage of reducing the number of qubits while maintaining a reasonably low depth. For better clarity, Figure 3(a) shows the unmodified version, whereas Figure 3(b) portrays the modifications proposed by us.

Additionally, we develop the out-of-place point addition of Algorithm 2 (Figure 3(c)), which computes $P_3(x_3, y_3)$ independently while preserving the input $P_1(x_1, y_1)$. This significantly reduces both the circuit depth and gate count.

In Figure 3, D is the division (which includes inversion and multiplication), and C is the copy operation for the control qubit q.

3.5.1 In-Place Implementation

The in-place point addition of Algorithm 1 computes the result on the input $P_1(x_1, y_1)$, and this result changes based on the control qubit q. As a result, the point either becomes $P(x, y) = P_3(x_3, y_3)$ or remains as $P_1(x_1, y_1)$. Compared to the point addition in [BBvHL20], our modified point addition differs in two key aspects.

First, we copy the control qubit q to ancilla qubits used in multiplication. In [BBvHL20, PWLK22, TT23], only a single control qubit q is used for controlled constant additions and controlled additions, meaning all CNOT and Toffoli gates are applied sequentially. Using a single control qubit for operations on arrays reduces the number of qubits but significantly increases the circuit depth[13]. In contrast, we copy the control qubit q to the ancilla qubits for multiplication (Algorithm 1; Steps 2, 8, 15) to optimize the circuit depth. Since we already have a sufficient number of ancilla qubits (M_{anc}) for the copy, we can perform controlled constant addition (CTRL_CONST_ADD) and controlled addition (CTRL_ADD) with depth 1 using these copies without additional allocation. As a side note, an efficient tree-based copy is implemented, where previous copies are used in subsequent copying steps (reducing the depth), and the copies are initialized to a clean state after use.

Second, we optimize the middle steps of [BBvHL20, Algorithm 3] to compute $x_2 + x_3$ if $q = 1$, or $x_1 + x_2$ if $q = 0$. In the previous implementation, one controlled constant addition for $q(a + x_2)$ and two controlled additions for $q \cdot \lambda^2$ and $q \cdot \lambda$ are performed (see Figure 3(a)). In Algorithm 1, Step 6, we compute $\lambda^2 + \lambda$ in a single squaring operation by constructing the matrix for $\lambda^2 + \lambda$ through the addition of the matrices for λ^2 and λ (similar to the addition of the matrices for a and a^2 in Section 3.2). Additionally, in Step 7, the constant $a + x_2$ is added to $\lambda^2 + \lambda$, resulting in $\lambda^2 + \lambda + a + x_2$. Finally, in Step 9, only one controlled addition is performed using this precomputed result $\lambda^2 + \lambda + a + x_2$.

Table 6**??** presents the step-by-step procedure of Algorithm 1, and the quantum circuit is shown in Figure 3(b).

3.5.2 Out-of-Place Implementation

The out-of-place point addition in Algorithm 2 preserves $P_1(x_1, y_1)$ and computes $P_3(x_3, y_3)$ independently of the control qubit q. Recall that the in-place method (Algorithm 1) requires reverse operations to revert the value to its intermediate state, which is necessary to compute the conditional result of either $P_1(x_1, y_1)$ or $P_3(x_3, y_3)$. On the other hand, the out-of-place approach avoids these additional operations by allocating output qubits during the process and computing the intermediate values directly on them (see Figure 3(c)). As a result, Algorithm 1 (in-place) requires 2 divisions, 2 squarings, and 2 multiplications; whereas Algorithm 2 (out-of-place) consists of 1 division, 1 squaring, and 1 multiplication.

Additionally, the out-of-place point addition reduces the number of controlled operations that use the control qubit q. In Algorithm 1 (in-place), 2 controlled constant additions and 2 controlled additions are performed (3 controlled constant additions and 3 controlled additions are required in [BBvHL20, PWLK22, TT23]).

In Algorithm 2, we swap the results $P_1(x_1, y_1)$ and $P_3(x_3, y_3)$ based on the control qubit q in the final stage (Steps 13 and 14). A controlled-swap (CTRL_SWAP) is performed twice: once for (x_1, y_1) and once for (x_3, y_3). These two controlled-swap operations are performed in parallel, and similar to the in-place method, the same copy technique is applied.

Table 6(b) presents the step-by-step walk-through of Algorithm 2, and the quantum circuit is illustrated in Figure 3(c).

[13] In [BBvHL20], the authors consider this trade-off but retain the approach, as their focus is on minimizing the qubit count. In our case, since we have a sufficient number of ancilla qubits (thanks to the design of the components, which reuses idle qubits), it is more efficient to copy.

Algorithm 1: Proposed in-place point addition on binary elliptic curves.

Classical input: A constant a from the elliptic curve, a fixed point $P_2(x_2, y_2)$.
Quantum input: A control qubit q, a point $P_1(x_1, y_1)$ on the elliptic curve, ancilla qubits M_{anc} for multiplication, ancilla qubits for inversion, ancilla qubits for λ.
Output: $P_1 + P_2 = P_3(x_3, y_3)$ if $q = 1$, $P_1(x_1, y_1)$ if $q = 0$, all ancilla qubits in a clean state.

$x \leftarrow \text{CONST\_ADD}(x_2,\, x_1)$ ▷ $x = x_1 + x_2$
$M_{anc} \leftarrow \text{COPY}(q,\, M_{anc})$ ▷ Copy q to M_{anc}
$y \leftarrow \text{CTRL\_CONST\_ADD}(M_{anc},\, y_2,\, y_1)$ ▷ $y = y_1 + q \cdot y_2$
$\lambda \leftarrow \text{DIV}(x_1, y_1, 0)$ ▷ $\lambda = y/x$
$y \leftarrow \text{MUL}(x_1,\, \lambda,\, y_1)$ ▷ $y = y + x \cdot (y/x) = 0$
$y \leftarrow \text{SQR}(\lambda^2 + \lambda,\, y_1)$ ▷ $y = \lambda^2 + \lambda$
$y \leftarrow \text{CONST\_ADD}(a + x_2,\, y_1)$ ▷ $y = \lambda^2 + \lambda + a + x_2$
$M_{anc} \leftarrow \text{COPY}(q,\, M_{anc})$ ▷ Copy q to M_{anc}
$y \leftarrow \text{CTRL\_ADD}(M_{anc},\, y_1,\, x_1)$ ▷ $x = x_1 + x_2 + q(\lambda^2 + \lambda + a + x_2)$
$y \leftarrow \text{SQR}(\lambda^2 + \lambda,\, y_1)$ ▷ $y = \lambda^2 + \lambda + a + x_2 + \lambda^2 + \lambda = a + x_2$
$y \leftarrow \text{CONST\_ADD}(a + x_2,\, y_1)$ ▷ $y = a + x_2 + a + x_2 = 0$
$y \leftarrow \text{MUL}(x_1,\, \lambda,\, y_1)$ ▷ $y = x \cdot \lambda$
$\lambda \leftarrow \text{DIV}(x_1, y_1, \lambda)$ ▷ $\lambda = \lambda + (x \cdot \lambda)/x = 0$
$x \leftarrow \text{CONST\_ADD}(x_2,\, x_1)$ ▷ $x = x_1 + q(\lambda^2 + \lambda + a + x_2)$
$M_{anc} \leftarrow \text{COPY}(q,\, M_{anc})$ ▷ Copy q to M_{anc}
$y \leftarrow \text{CTRL\_CONST\_ADD}(y_2,\, y_1)$ ▷ $y = y + q \cdot y_2$
$y_1 \leftarrow \text{CTRL\_ADD}(x_1,\, y_1)$ ▷ $y = y + q \cdot x_3$
return $(x,\, y)$

3.6 Windowing Technique

The windowing technique is an effective method to optimize conditional point additions by adding a superposition of a single point P_2. Windowing utilizes quantum random access memory (qRAM) and represents qubits in superposition over the indices $i = 0, 1, 2, \ldots, 2^\ell - 1$. The addition of points can then be described as adding $[i]P_2$. To achieve this, the lookup table consists of precomputed points: $P_2, [1]P_2, [2]P_2, \ldots, [2^\ell - 1]P_2$ (fixed point T is used to avoid infinity). This table enables the addition of $[i]P_2$ through a look-up operation in superposition, significantly reducing the number of point additions required. The efficiency of the windowing method depends on the chosen window size ℓ, which determines how many points are precomputed and stored. A larger value of ℓ reduces the number of point additions but increases the cost of constructing the lookup using qRAM.

In [BBvHL20, PWLK22, TT23], the reduction in the number of Toffoli gates through windowing is estimated for each field size n, by using the optimal window size ℓ. For windowing, the addition of precomputed points via look-ups must be considered. In [BBvHL20], the additions of $P_2(x_2, y_2)$ in Figure 3(a) are replaced with lookup additions, and the controlled additions are changed to regular additions. The same modification applies to Algorithm 1. Similarly, for Algorithm 2 (out-of-place), additions of $P_2(x_2, y_2)$ are replaced with lookup additions, and the controlled-swap operations in the final step can be removed.

As in previous works [BBvHL20, PWLK22, TT23], we too estimate the results after applying windowing (in Section 4). These works have reported a reduction in Toffoli gate count due to windowing. Similarly, we report the reduced Toffoli gate count after applying windowing and provide the total reduced quantum gate count after decomposing the Toffoli gates.

Algorithm 2: Proposed out-of-place point addition on binary elliptic curves.

Classical input: A constant a from the elliptic curve, a fixed point $P_2(x_2, y_2)$.
Quantum input: A control qubit q, a point $P_1(x_1, y_1)$ on the elliptic curve, ancilla qubits M_{anc} for multiplication, qubits for (x, y), ancilla qubits for inversion, ancilla qubits for λ.
Output: $P_1 + P_2 = P_3(x_3, y_3)$ if $q = 1$, $P_1(x_1, y_1)$ if $q = 0$; ancilla qubits M_{anc} in a clean state.

```
1:  x ← CNOT (x1, x)                          ▷ x = x1
2:  y1 ← CONST_ADD (y2, y1)                   ▷ y1 = y1 + y2
3:  x ← CONST_ADD (x2, x)                     ▷ x = x1 + x2
4:  λ ← DIV (x, y1, 0)                        ▷ λ = (y1 + y2)/(x1 + x2)
5:  y1 ← CONST_ADD (y2, y1)                   ▷ y1 = y1 + y2 + y2 = y1
6:  x ← CONST_ADD (a + x2, x)                 ▷ x = x1 + x2 + a + x2 = x1 + a
7:  x ← SQR (λ^2 + λ, x)                      ▷ x = x1 + a + λ^2 + λ = x2 + x3
8:  y ← MUL (x, λ, 0)                         ▷ y = (x2 + x3)λ
9:  x ← CONST_ADD (x2, x)                     ▷ x = x1 + a + λ^2 + λ + x2 = x3
10: y ← CONST_ADD (y2, y)                     ▷ y = (x2 + x3)λ + y2
11: y ← CNOT (x, y)                           ▷ y = (x2 + x3)λ + y2 + x3 = y3
12: M_anc ← COPY (q, M_anc)                   ▷ Copy q to M_anc
13: CTRL_SWAP (M_anc, x1, x)                  ▷ x = x3 (if q = 1) or x1 (if q = 0)
14: CTRL_SWAP (M_anc, y1, y)                  ▷ y = y3 (if q = 1) or y1 (if q = 0)
15: return (x, y)
```

Table 6: Steps of point addition.

<table>
<tr><th>Step</th><th>$q = 1$</th><th>$q = 0$</th></tr>
<tr><td>1</td><td colspan="2">$x = x_1 + x_2$</td></tr>
<tr><td>2, 3</td><td>$y = y_1 + y_2$</td><td>$y = y_1$</td></tr>
<tr><td>4</td><td>$\lambda = \frac{y_1+y_2}{x_1+x_2}$</td><td>$\lambda = \frac{y_1}{x_1+x_2}$</td></tr>
<tr><td>5</td><td colspan="2">$y = 0$</td></tr>
<tr><td>6, 7</td><td colspan="2">$y = \lambda^2 + \lambda + a + x_2$</td></tr>
<tr><td>8, 9</td><td>$x = x_2 + x_3$</td><td>$x = x_1 + x_2$</td></tr>
<tr><td>10, 11</td><td colspan="2">$y = 0$</td></tr>
<tr><td>12</td><td>$y = (x_2 + x_3)\lambda$</td><td>$y = y_1$</td></tr>
<tr><td>13</td><td colspan="2">$\lambda = 0$</td></tr>
<tr><td>14</td><td>$x = x_3$</td><td>$x = x_1$</td></tr>
<tr><td>15, 16, 17</td><td>$y = y_3$</td><td>$y = y_1$</td></tr>
</table>

(b) Algorithm 2 (out-of-place, by us).

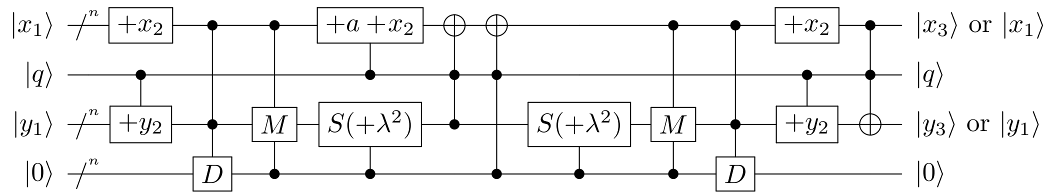
<table>
<tr><th>Step</th><th>$q = 1$</th><th>$q = 0$</th></tr>
<tr><td>1, 2, 3</td><td colspan="2">$x = x_1 + x_2$
$y_1 = y_1 + y_2$</td></tr>
<tr><td>4</td><td colspan="2">$\lambda = \frac{y_1+y_2}{x_1+x_2}$</td></tr>
<tr><td>5</td><td colspan="2">$y_1 = y_1$</td></tr>
<tr><td>6, 7</td><td colspan="2">$x = x_2 + x_3$</td></tr>
<tr><td>8</td><td colspan="2">$y = (x_2 + x_3)\lambda$</td></tr>
<tr><td>9</td><td colspan="2">$x = x_3$</td></tr>
<tr><td>10</td><td colspan="2">$y = (x_2 + x_3)\lambda + y_2$</td></tr>
<tr><td>11</td><td colspan="2">$y = y_3$</td></tr>
<tr><td>12, 13, 14</td><td>$(x, y) = (x_3, y_3)$</td><td>$(x, y) = (x_1, y_1)$</td></tr>
</table>

4 Results

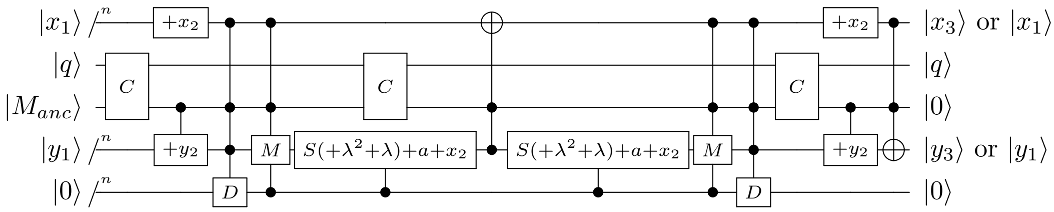
Table 9 shows the quantum resources required for a single point addition on binary elliptic curves. Similar to Table 5, for [BBvHL20, PWLK22] in Table 9, we use the re-estimated results from [TT23].

Since [BBvHL20], two research works [PWLK22, TT23] have been reported. However, it can be argued that the improvement (in terms of performance) over [BBvHL20] in these subsequent works is not that significant or noteworthy. Indeed, in Table 9, the results by [BBvHL20] achieve the best performance in terms of the product of depth and qubit count (D-M) for $n = 163, 283$ and 571 in comparison to [PWLK22, TT23].

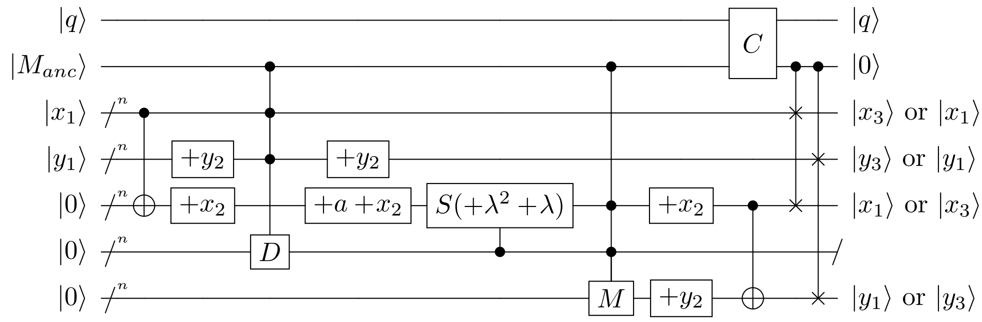
The point additions in this work demonstrate superior performance, surpassing previous works [BBvHL20, PWLK22, TT23] with significantly lower Toffoli depth and circuit depth by utilizing additional ancilla qubits. Our Toffoli gate count is higher than the re-estimated result in [TT23] because they replace the Toffoli gate-optimized multiplication from [KKKH22]. However, this multiplication method requires more CNOT gates. For the product of depth and qubit count (D-M), we achieve improvements of 73% − 81% and



(a) Point addition in [BBvHL20, Algorithm 3] (in-place, used in [PWLK22, TT23]).



(b) Algorithm 1 (in-place, modified from [BBvHL20, Algorithm 3] by us).



(c) Algorithm 2 (out-of-place, by us).

Figure 3: Quantum circuits for point addition.

more than 92% for in-place and out-of-place point additions (FLT-in and FLT-out) in each binary field $\mathbb{F}_{2^n}$, respectively.

In Table 9, note that there is a slight increase from depth (D) to full depth in our implementation (compared to [BBvHL20, PWLK22, TT23]), because of the low Toffoli depth. Although the product of full depth and qubit count could not be reported in Table 9 due to insufficient data by the authors, we achieve further improvements in this metric.

Relation to Shor's Algorithm

As mentioned in Section 2.3, $2n+2$ point additions over the binary field $\mathbb{F}_2^n$ are required to construct Shor's quantum circuit for solving the ECDLP. Table 7 reports the required quantum resources for Shor's algorithm on binary elliptic curves. For the estimation, we decompose the Toffoli gates and estimate the total number of quantum gates, consisting of Clifford and T gates, as well as the full depth. We adopt one of the decomposition methods from [AMM+13]; where a Toffoli gate is decomposed into 8 Clifford gates plus 7 T gates, incurring the T-depth of 4 and the full depth of 8[14].

As mentioned in Section 3.6, the number of point additions (steps) can be reduced to $2 \cdot \lceil \frac{n+1}{\ell} \rceil$. Each point addition requires 6 lookups, and each lookup involves $2 \cdot (2^\ell - 1)$ Toffoli gates. Given these, we determine the optimal size for each of the binary fields.

[14] It is worth noting that further improvements can be achieved by replacing Toffoli gates with quantum AND gates (as in [JNRV19, LPZW23, SF24, JBK+22]). Further details are given in [JSB+25, Appendix B]

Table 8 presents the reduced Toffoli gate count and the total gate count for each binary field based on the choice of window size ℓ.

Table 7: Quantum resource requirement by Shor's algorithm on binary elliptic curves.

Method	n	Qubits	Total gates (G)	Full depth (FD)	T-depth	Cost (G-FD)	MAXDEPTH	NIST security
FLT-in (Algorithm 1)	8	$1.67 \cdot 2^{7}$	$1.14 \cdot 2^{18}$	$1.83 \cdot 2^{13}$	$1.83 \cdot 2^{10}$	$1.04 \cdot 2^{32}$	✓ ($\leq 2^{40}$)	✗*
	16	$1.52 \cdot 2^{9}$	$1.13 \cdot 2^{21}$	$1.47 \cdot 2^{15}$	$1.13 \cdot 2^{12}$	$1.66 \cdot 2^{36}$		
	127	$1.89 \cdot 2^{14}$	$1.51 \cdot 2^{29}$	$1.34 \cdot 2^{20}$	$1.69 \cdot 2^{15}$	$1.02 \cdot 2^{50}$		
	163	$1.62 \cdot 2^{15}$	$1.66 \cdot 2^{30}$	$1.95 \cdot 2^{20}$	$1.84 \cdot 2^{15}$	$1.62 \cdot 2^{51}$		
	233	$1.26 \cdot 2^{16}$	$1.98 \cdot 2^{31}$	$1.69 \cdot 2^{21}$	$1.43 \cdot 2^{16}$	$1.67 \cdot 2^{53}$		
	283*	$1.10 \cdot 2^{17}$	$1.05 \cdot 2^{33}$	$1.56 \cdot 2^{22}$	$1.87 \cdot 2^{16}$	$1.64 \cdot 2^{55}$		
	571*	$1.91 \cdot 2^{18}$	$1.99 \cdot 2^{35}$	$1.70 \cdot 2^{24}$	$1.06 \cdot 2^{18}$	$1.70 \cdot 2^{60}$		
FLT-out (Algorithm 2)	8	$1.60 \cdot 2^{11}$	$1.22 \cdot 2^{16}$	$1.04 \cdot 2^{12}$	$1.97 \cdot 2^{8}$	$1.27 \cdot 2^{28}$		
	16	$1.42 \cdot 2^{14}$	$1.18 \cdot 2^{19}$	$1.67 \cdot 2^{13}$	$1.20 \cdot 2^{10}$	$1.97 \cdot 2^{32}$		
	127	$1.75 \cdot 2^{22}$	$1.53 \cdot 2^{27}$	$1.36 \cdot 2^{18}$	$1.75 \cdot 2^{13}$	$1.04 \cdot 2^{46}$		
	163	$1.90 \cdot 2^{23}$	$1.69 \cdot 2^{28}$	$1.02 \cdot 2^{19}$	$1.92 \cdot 2^{13}$	$1.72 \cdot 2^{47}$		
	233	$1.06 \cdot 2^{25}$	$1.00 \cdot 2^{30}$	$1.73 \cdot 2^{19}$	$1.49 \cdot 2^{14}$	$1.74 \cdot 2^{49}$		
	283*	$1.14 \cdot 2^{26}$	$1.06 \cdot 2^{31}$	$1.64 \cdot 2^{20}$	$1.94 \cdot 2^{14}$	$1.74 \cdot 2^{51}$		
	571*	$1.00 \cdot 2^{29}$	$1.98 \cdot 2^{33}$	$1.76 \cdot 2^{22}$	$1.12 \cdot 2^{16}$	$1.74 \cdot 2^{56}$		

*: Corresponds to 128-bit classical security (i.e., comparable to AES-128).
*: Corresponds to 256-bit classical security (i.e., comparable to AES-256).
*: Level 1 security is achieved if G-FD cost $\geq 2^{156}$ (based on [JBK+22]).

Table 8: Approximate quantum gate requirement after applying windowing technique.

Method	n	Window size (ℓ)	Steps	Look-ups	Toffoli gates	Total gates (G)
FLT-in (Algorithm 1)	8	5	4	24	$1.01 \cdot 2^{12}$	$1.35 \cdot 2^{16}$
	16	6	6	36	$1.24 \cdot 2^{14}$	$1.86 \cdot 2^{18}$
	127	10	26	156	$1.56 \cdot 2^{21}$	$1.30 \cdot 2^{26}$
	163	11	30	180	$1.56 \cdot 2^{22}$	$1.30 \cdot 2^{27}$
	233	12	40	240	$1.68 \cdot 2^{23}$	$1.46 \cdot 2^{28}$
	283	13	44	264	$1.66 \cdot 2^{24}$	$1.42 \cdot 2^{29}$
	571	14	82	492	$1.26 \cdot 2^{27}$	$1.20 \cdot 2^{32}$
FLT-out (Algorithm 2)	8	3	6	36	$1.54 \cdot 2^{10}$	$1.04 \cdot 2^{15}$
	16	5	8	48	$1.03 \cdot 2^{13}$	$1.45 \cdot 2^{17}$
	127	8	32	192	$1.94 \cdot 2^{19}$	$1.61 \cdot 2^{24}$
	163	10	34	204	$1.97 \cdot 2^{20}$	$1.58 \cdot 2^{25}$
	233	10	48	288	$1.01 \cdot 2^{22}$	$1.78 \cdot 2^{26}$
	283	11	52	312	$1.97 \cdot 2^{22}$	$1.70 \cdot 2^{27}$
	571	12	96	576	$1.48 \cdot 2^{25}$	$1.39 \cdot 2^{30}$

5 Applicability & Impact of Shor's Algorithm

It is important to note that in-place point addition has the advantage that the qubit count does not increase during the $2n + 2$ point additions in Shor's quantum circuit. Simply put, the required number of qubits for Shor's algorithm (using the semi-classical Fourier transform, see[JSB+25, Appendix F]) remains the same as shown in Table 9.

In contrast, in the case of out-of-place point additions, new output qubits must be allocated for each execution in Shor's quantum circuit. The continuous production of garbage qubits during the process is a clear disadvantage in quantum computations [VP98]. However, their gate and depth complexity are much lower compared to in-place point additions. Additionally, it should be noted that we do not manage all the qubits listed in Table 7 throughout the entire computation.

In these considerations, a careful choice between the in-place and out-of-place point additions should be made, and our work provides the best options for both approaches.

Table 9: Quantum resources required for a single point addition on binary elliptic curves.

n	Source		#Toffoli	#CNOT	#Qubit (M)	Toffoli depth	Depth (D)	Full depth	D-M
8	B+ [BBvHL20]	(GCD)	7360	3522	68	N/A	8562	N/A	582216
	This paper	(FLT-in)	664	6580	214	26	517	831	110638
		(FLT-out)	178	1761	241	7	159	236	38319
16	B+ [BBvHL20]	(GCD)	21016	11686	125	N/A	25205	N/A	3150625
	This paper	(FLT-in)	2624	30560	778	34	959	1412	746102
		(FLT-out)	680	7953	859	9	283	402	243097
127	B+ [BBvHL20]	(GCD)	559141	497957	904	N/A	776234	N/A	701715536
	This paper	(FLT-in)	113874	1464162	30972	54	4994	5507	154674168
		(FLT-out)	28659	370120	33157	14	1267	1394	42009919
163	B+ [BBvHL20]	(FLT)	40169	3357029	1957	N/A	706512	N/A	1382643984
		(GCD)	880005	982769	1157	N/A	1042736	N/A	1206445552
	P+ [PWLK22]	(FLT)	40169	3269957	3098	N/A	623328	N/A	1931070144
	TT [TT23]	(FLT-basic)	40169	3268653	2772	N/A	623320	N/A	1727843040
		(FLT-extended)	40169	3313485	1957	N/A	643140	N/A	1258624980
	This paper	(FLT-in)	193354	2540458	53134	46	5685	6227	302066790
		(FLT-out)	48583	650277	57521	12	1495	1631	85993895
233	B+ [BBvHL20]	(FLT)	64103	7059764	3030	N/A	953699	N/A	2889707970
		(GCD)	1649771	1979416	1647	N/A	2019813	N/A	3326632011
	P+ [PWLK22]	(FLT)	64103	7004780	4661	N/A	904283	N/A	4214863063
	TT [TT23]	(FLT-basic)	64103	7001984	3962	N/A	904271	N/A	3582721702
		(FLT-extended)	64103	7018404	3030	N/A	909775	N/A	2756618250
	This paper	(FLT-in)	303970	4516616	82899	50	7059	7589	585184041
		(FLT-out)	76342	1158949	89222	13	1800	1942	160599600
283	B+ [BBvHL20]	(FLT)	86481	11739723	3963	N/A	2017360	N/A	7994797680
		(GCD)	2393413	2895567	1998	N/A	2944136	N/A	5882383728
	P+ [PWLK22]	(FLT)	86481	11434619	6227	N/A	1720152	N/A	10711386504
	TT [TT23]	(FLT-basic)	86481	11428959	4812	N/A	1720132	N/A	8277275184
		(FLT-extended)	86481	11454547	3963	N/A	1727164	N/A	6844750932
	This paper	(FLT-in)	534762	7856986	144672	54	10957	11556	1585171104
		(FLT-out)	134115	2007399	154945	14	2902	3025	449650390
571	B+ [BBvHL20]	(FLT)	215241	53816483	9137	N/A	8931056	N/A	81603058672
		(GCD)	8877969	11443427	4015	N/A	11331616	N/A	45496438240
	P+ [PWLK22]	(FLT)	215241	52108071	14276	N/A	7240884	N/A	103370859984
	TT [TT23]	(FLT-basic)	199989	48646235	10850	N/A	6993780	N/A	75882513000
		(FLT-extended)	199989	48757075	8566	N/A	7040192	N/A	60306284672
	This paper	(FLT-in)	1871402	30657812	500450	62	23596	24399	11808618200
		(FLT-out)	468707	7833731	531621	16	6313	6449	3356123373

NIST Post-Quantum Security

NIST has introduced criteria for quantum attacks. In particular, the MAXDEPTH constraint[15] involves limiting quantum attacks by setting a maximum quantum circuit depth (corresponding to runtime). The lower limit of MAXDEPTH is $\leq 2^{40}$ (though the maximum allowable limit for MAXDEPTH is $\leq 2^{96}$), and we can observe that none of the full depths from Table 7 exceed this limit. Additionally, NIST employs post-quantum security measures against quantum attacks to evaluate the robustness of cryptographic algorithms[16]. For the post-quantum security level 1, the cost of a quantum key search (using Grover's algorithm) on AES-128 is used for evaluation. For the calculation of the cost, the product of the total quantum gates and full depth is used (i.e., G-FD in Table 7), and the cost for AES-128 is 2^{156}, based on the results of [JBK+22] at the time of writing

[15] Refer to page 16 of the NIST documentation: https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/call-for-proposals-dig-sig-sept-2022.pdf.

[16] Refer to pages 15 – 17 of the NIST documentation: https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/call-for-proposals-dig-sig-sept-2022.pdf.

this paper. Note that, the bound was formerly estimated as 2^{157} based on the results in [JNRV19]; however, it was later found that this was overestimated due to a bug Q# (see [JBK+22, Section 5] for more details)[17] As anticipated, the costs in Table 7 are orders of magnitude lower than this, and thus cannot achieve the post-quantum security.

6 Conclusion

It is expected that the common public key systems currently employed, including those are based on the binary elliptic curves, will be broken with a powerful enough quantum computer sometime in the near future. Few research works have been carried out in this direction, still, arguably there has not been any remarkable advancement since the work of Banegas et al. [BBvHL20], from where our work picks up.

We significantly reduce the quantum resources required to break binary field ECC. We focus on FLT-based division and depth-efficient point addition on binary elliptic curves using both in-place and out-of-place approaches. Compared to the previous best results, our point addition achieve the lowest circuit depth and improvements of more than 73% – 81% (in-place, Algorithm 1) and 92% (out-of-place, Algorithm 2) in trade-off performance (the product of depth and qubit count) for all binary fields, as shown in Table 7. As far as we can tell, this work shows the most advanced results in quantum cryptanalysis of binary ECC.

As a potential direction for future research, point addition on *projective coordinates* (see [JSB+25, Section 2.3]). This method may reduce circuit depth at the expense of increasing the qubit count, similar to our out-of-place point addition. It would be worthwhile to adapt our implementations to projective coordinates and benchmark its efficiency. Another interesting direction is to extend our approaches to other elliptic curves (such as the Curve-25519 by Bernstein in [Ber06]), or RSA (e.g., following up on [YYT+23]). One might also be curious by the work of [GE21], i.e., estimating the time required to break by a quantum computer. At the circuit component level, it could be useful to find more efficient in-place implementations than that of the Gauss-Jordan elimination or PLU factorization for large ($> 64 \times 64$) binary matrices.

References

[AMM+13] Matthew Amy, Dmitri Maslov, Michele Mosca, Martin Roetteler, and Martin Roetteler. A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 32(6):818–830, Jun 2013. 797

[Bar20] Elaine Barker. NIST special publication 800-57 part 1, revision 5, recommendation for key management: Part 1 – general. *NIST, Tech. Rep*, page 171, 2020. 782

[BBvHL20] Gustavo Banegas, Daniel J. Bernstein, Iggy van Hoof, and Tanja Lange. Concrete quantum cryptanalysis of binary elliptic curves. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021:451–472, 12 2020. 782, 783, 784, 787, 789, 792, 793, 794, 795, 796, 797, 799, 800

[BDK+21] Anubhab Baksi, Vishnu Asutosh Dasu, Banashri Karmakar, Anupam Chattopadhyay, and Takanori Isobe. Three input exclusive-or gate support for boyar-peralta's algorithm. In Avishek Adhikari, Ralf Küsters, and Bart Preneel, editors, *Progress in Cryptology - INDOCRYPT 2021, Jaipur, India, December*

[17] The authors of [JNRV19] subsequently corrected the bug and updated their paper.

12-15, 2021, Proceedings, volume 13143 of *Lecture Notes in Computer Science*, pages 141–158. Springer, 2021. 787

[Ber06] Daniel J. Bernstein. 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*, pages 207–228. Springer, 2006. 800

[BKD21] Anubhab Baksi, Banashri Karmakar, and Vishnu Asutosh Dasu. POSTER: optimizing device implementation of linear layers with automated tools. In *Applied Cryptography and Network Security Workshops - ACNS 2021 Satellite Workshops, AIBlock, AIHWS, AIoTS, CIMSS, Cloud S&P, SCI, SecMT, and SiMLA, Kamakura, Japan, June 21-24, 2021, Proceedings*, volume 12809 of *Lecture Notes in Computer Science*, pages 500–504. Springer, 2021. 788

[BL17] Daniel J Bernstein and Tanja Lange. Post-quantum cryptography. *Nature*, 549(7671):188–194, 2017. 782

[CDKM08] Steven Cuccaro, Thomas Draper, Samuel Kutin, and David Moulton. A new quantum ripple-carry addition circuit. arXiv, 2008. https://arxiv.org/pdf/quant-ph/0410184.pdf. 786

[CFA+05] Henri Cohen, Gerhard Frey, Roberto Avanzi, Christophe Doche, Tanja Lange, Kim Nguyen, and Frederik Vercauteren. *Handbook of elliptic and hyperelliptic curve cryptography.* CRC press, 2005. 784

[CMR+23] Lily Chen, Dustin Moody, Karen Randall, Andrew Regenscheid, and Angela Robinson. Recommendations for discrete logarithm-based cryptography: Elliptic curve domain parameters nist special publication nist sp 800-186, February 2023. 784

[DH22] Whitfield Diffie and Martin E. Hellman. *New Directions in Cryptography*, page 365–390. Association for Computing Machinery, New York, NY, USA, 1 edition, 2022. 782

[DKRS04] Thomas G Draper, Samuel A Kutin, Eric M Rains, and Krysta M Svore. A logarithmic-depth quantum carry-lookahead adder. *arXiv preprint quant-ph/0406142*, 2004. 786

[Dra00] Thomas G Draper. Addition on a quantum computer. *arXiv preprint quant-ph/0008033*, 2000. 786

[ElG85] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Advances in Cryptology*, pages 10–18. Springer, 1985. 782

[GE21] Craig Gidney and Martin Ekerå. How to factor 2048 bit rsa integers in 8 hours using 20 million noisy qubits. *Quantum*, 5:433, 2021. 782, 800

[GG16] Steven D Galbraith and Pierrick Gaudry. Recent progress on the elliptic curve discrete logarithm problem. *Designs, Codes and Cryptography*, 78:51–72, 2016. 782

[GN96] Robert B Griffiths and Chi-Sheng Niu. Semiclassical fourier transform for quantum computation. *Physical Review Letters*, 76(17):3228, 1996. 786

[HJN+20] Thomas Häner, Samuel Jaques, Michael Naehrig, Martin Roetteler, and Mathias Soeken. 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*, pages 425–444. Springer, 2020. 784

[IT88] Toshiya Itoh and Shigeo Tsujii. A fast algorithm for computing multiplicative inverses in gf (2m) using normal bases. *Information and computation*, 78(3):171–177, 1988. 790

[JBK+22] Kyungbae Jang, Anubhab Baksi, Hyunji Kim, Gyeongju Song, Hwajeong Seo, and Anupam Chattopadhyay. Quantum analysis of aes. Cryptology ePrint Archive, Paper 2022/683, 2022. https://eprint.iacr.org/2022/683. 791, 797, 798, 799, 800

[JKL+23] Kyungbae Jang, Wonwoong Kim, Sejin Lim, Yeajun Kang, Yujin Yang, and Hwajeong Seo. Optimized implementation of quantum binary field multiplication with Toffoli depth one. In *Information Security Applications: 23rd International Conference, WISA 2022, Jeju Island, South Korea, August 24–26, 2022, Revised Selected Papers*, pages 251–264. Springer, 2023. 783, 789, 790, 791

[JLO+24] Kyungbae Jang, Sejin Lim, Yujin Oh, Hyunjun Kim, Anubhab Baksi, Sumanta Chakraborty, and Hwajeong Seo. Quantum implementation and analysis of SHA-2 and SHA-3. Cryptology ePrint Archive, Paper 2024/513, 2024. 791

[JMV01] Don Johnson, Alfred Menezes, and Scott Vanstone. The elliptic curve digital signature algorithm (ecdsa). *International journal of information security*, 1:36–63, 2001. 782

[JNRV19] Samuel Jaques, Michael Naehrig, Martin Roetteler, and Fernando Virdia. Implementing grover oracles for quantum key search on AES and LowMC. Cryptology ePrint Archive, Report 2019/1146, 2019. https://eprint.iacr.org/2019/1146. 797, 800

[JSB+25] Kyungbae Jang, Vikas Srivastava, Anubhab Baksi, Santanu Sarkar, and Hwajeong Seo. New quantum cryptanalysis of binary elliptic curves (extended version). Cryptology ePrint Archive, Paper 2025/017, 2025. 782, 783, 788, 791, 797, 798, 800

[KH23] Hyeonhak Kim and Seokhie Hong. New space-efficient quantum algorithm for binary elliptic curves using the optimized division algorithm. *Quantum Information Processing*, 22(6):237, 2023. 792

[KKKH22] Sunyeop Kim, Insung Kim, Seonggyeom Kim, and Seokhie Hong. Toffoli gate count optimized space-efficient quantum circuit for binary field multiplication. *Cryptology ePrint Archive*, 2022. 789, 791, 792, 796

[Kob87] Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of computation*, 48(177):203–209, 1987. 782

[Kob94] Neal Koblitz. *A course in number theory and cryptography*, volume 114. Springer Science & Business Media, 1994. 782

[LPZW23] Qun Liu, Bart Preneel, Zheng Zhao, and Meiqin Wang. 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*, pages 67–98. Springer, 2023. 791, 797

[Mil85] Victor S Miller. Use of elliptic curves in cryptography. In *Conference on the theory and application of cryptographic techniques*, pages 417–426. Springer, 1985. 782

[OJBS24] Yujin Oh, Kyungbae Jang, Anubhab Baksi, and Hwajeong Seo. Depth-optimized quantum circuits for ascon: Aead and hash. *Mathematics*, 12(9), 2024. 787

[PWLK22] Dedy Septono Catur Putranto, Rini Wisnu Wardhani, Harashta Tatimma Larasati, and Howon Kim. Another concrete quantum cryptanalysis of binary elliptic curves. *Cryptology ePrint Archive*, 2022. 782, 787, 789, 791, 792, 793, 794, 795, 796, 797, 799

[RBC23] Soham Roy, Anubhab Baksi, and Anupam Chattopadhyay. Quantum implementation of ASCON linear layer. NIST Lightweight Cryptography Workshop, 2023. `https://csrc.nist.gov/csrc/media/Events/2023/lightweight-cryptography-workshop-2023/documents/accepted-papers/06-quantum-implementation-ascon-linear-layer.pdf`. 787, 788

[RNSL17] Martin Roetteler, Michael Naehrig, Krysta M Svore, and Kristin Lauter. Quantum resource estimates for computing elliptic curve discrete logarithms. In *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part II 23*, pages 241–270. Springer, 2017. 782, 784, 793

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, February 1978. 782

[SF24] Haotian Shi and Xiutao Feng. Quantum circuits of aes with a low-depth linear layer and a new structure. *Cryptology ePrint Archive*, 2024. 791, 797

[Sho94] Peter W Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th annual symposium on foundations of computer science*, pages 124–134. Ieee, 1994. 785, 786

[SHT18] Damian S Steiger, Thomas Häner, and Matthias Troyer. Projectq: an open source software framework for quantum computing. *Quantum*, 2:49, 2018. 783

[TT23] Ren Taguchi and Atsushi Takayasu. Concrete quantum cryptanalysis of binary elliptic curves via addition chain. In *Cryptographers' Track at the RSA Conference*, pages 57–83. Springer, 2023. 782, 787, 789, 792, 793, 794, 795, 796, 797, 799

[TTK09] Yasuhiro Takahashi, Seiichiro Tani, and Noboru Kunihiro. Quantum addition circuits and unbounded fan-out. *arXiv preprint arXiv:0910.2530*, 2009. 786

[vH19] Iggy van Hoof. Space-efficient quantum multiplication of polynomials for binary finite fields with sub-quadratic Toffoli gate count. *arXiv preprint arXiv:1910.02849*, 2019. 789, 791

[VP98] Vlatko Vedral and Martin B Plenio. Basics of quantum computation. *Progress in quantum electronics*, 22(1):1–39, 1998. 798

[XZL+20] Zejun Xiang, Xiangyong Zeng, Da Lin, Zhenzhen Bao, and Shasha Zhang. Optimizing implementations of linear layers. *IACR Trans. Symmetric Cryptol.*, 2020(2):120–145, 2020. 788

[YWS+24] Yufei Yuan, Wenling Wu, Tairong Shi, Lei Zhang, and Yu Zhang. A framework to improve the implementations of linear layers. *IACR Transactions on Symmetric Cryptology*, 2024(2):322–347, Jun. 2024. 788

[YYT+23] Junpei Yamaguchi, Masafumi Yamazaki, Akihiro Tabuchi, Takumi Honda, Tetsuya Izu, and Noboru Kunihiro. Estimation of shor's circuit for 2048-bit integers based on quantum simulator. *Cryptology ePrint Archive*, 2023. 800