ON THE DESIGN AND OPTIMIZATION OF A QUANTUM POLYNOMIAL-TIME ATTACK ON ELLIPTIC CURVE CRYPTOGRAPHY[a]

DMITRI MASLOV
Department of Combinatorics and Optimization, University of Waterloo
Waterloo, Ontario, N2L 3G1, Canada

JIMSON MATHEW
Department of Computer Science, University of Bristol
Bristol, BS8 1UB, UK

DONNY CHEUNG
Department of Computer Science, University of Calgary
Calgary, Alberta, T2N 1N4, Canada

DHIRAJ K. PRADHAN
Department of Computer Science, University of Bristol
Bristol, BS8 1UB, UK

Received (received date)
Revised (revised date)

We consider a quantum polynomial-time algorithm which solves the discrete logarithm problem for points on elliptic curves over $GF(2^m)$. We improve over earlier algorithms by constructing an efficient circuit for multiplying elements of binary finite fields and by representing elliptic curve points using a technique based on projective coordinates. The depth of our proposed implementation, executable in the Linear Nearest Neighbor (LNN) architecture, is $O(m^2)$, which is an improvement over the previous bound of $O(m^3)$ derived assuming no architectural restrictions.

Keywords: Quantum Circuits, Cryptography, Elliptic Curve Discrete Logarithm Problem, Quantum Algorithms

1 Introduction

Quantum computing [14] has the ability to solve problems whose best classical solutions are considered inefficient. Perhaps the best-known example is Shor's polynomial-time integer factorization algorithm [19], where the best known classical technique, the General Number Field Sieve, has superpolynomial complexity $\exp O(\sqrt[3]{n \log^2 n})$ in the number of bits n [22].

[a]An earlier version of this paper has been presented at the 3rd Workshop on Theory of Quantum Computation, Communication, and Cryptography, Tokyo, Japan, January 30 - February 1, 2008.

Table 1. Comparing hardware performance of RSA-3072 and ECC-256 [21].

Mode	RSA-3072	ECC-256
Space-optimized	184 ms 50,000 gates	29 ms 6,660 gates
Speed-optimized	110 ms 189,200 gates	1.3 ms 80,100 gates

Since a hardware implementation of this algorithm on a suitable quantum mechanical system could be used to crack the RSA cryptosystem [22], these results force researchers to rethink the assumptions of classical cryptography and to consider optimized circuits for the two main parts of Shor's factorization algorithm: the quantum Fourier transform [14, 4] and modular exponentiation [13]. Quantum noise and issues of scalability in quantum information processing proposals require circuit designers to consider optimization carefully.

Since the complexity of breaking RSA is subexponential, cryptosystems such as Elliptic Curve Cryptography (ECC) have become increasingly popular. The best known classical attack on ECC requires an exponential search with complexity $O(2^{n/2})$. The difference is substantial: a 256-bit ECC key requires the same effort to break as a 3072-bit RSA key. The largest publicly broken ECC system has a key length of 109 bits [3], while the key lengths of 1024 bits and higher are strongly recommended for RSA. However, the key lengths represent only the communication cost of a cryptographic protocol. It might, in principle, be possible that the hardware implementation of ECC were overwhelmingly less efficient, and then its practical efficiency would be undermined. However, this is not the case; indeed, the situation is rather opposite. Table 1 compares the efficiency of CMOS circuits (with the same clock speed) implementing 3072-bit RSA and 256-bit ECC, which both give equivalent security of 128 bits, in two hardware modes: optimized for space (cost), and speed (runtime). It is clear that the ECC implementation is more efficient. Relative efficiency of ECC as compared to RSA has been widely recognized. For instance, ECC has been acknowledged by National Security Agency as a secure protocol and included in their Suite B [15].

Most ECC implementations are built over $GF(2^m)$, likely, due to the efficiency of relevant hardware and the ease of mapping a key into a binary register. Software implementations, such as ECC over $GF(2^{155})$, are publicly available [1], making ECC ready to use for any interested party.

There exists a quantum polynomial-time algorithm that solves Elliptic Curve Discrete Logarithm Problem (ECDLP) and thus cracks elliptic curve cryptography [17]. As with Shor's factorization algorithm, this algorithm should be studied in detail by anyone interested in studying the threat posed by quantum computing to modern cryptography. The quantum algorithm for solving discrete logarithm problems in cyclic groups such as the one used in ECC requires computing sums and products of finite field elements, such as $GF(2^m)$ [6]. Addition in $GF(2^m)$ requires only a depth-1 circuit consisting of parallel CNOT gates [2]. We present a depth $O(m)$ multiplication circuit for $GF(2^m)$ optimized for the Linear Nearest Neighbor (LNN) architecture. Our circuit is based on the construction by Mastrovito [10]. Previously, a depth $O(m^2)$ circuit in an unrestricted architecture was found in [2]. With the use of our multiplication circuit the depth of the quantum discrete logarithm algorithm over the points on elliptic curves over $GF(2^m)$ drops from $O(m^3)$ to $O(m^2)$. Our implementation is optimized for LNN, unlike previously constructed circuit whose depth, if restricted to LNN, may become as high as $O(m^4)$.

The paper is organized as follows. In Section 2 we give an overview of quantum computation, $GF(2^m)$ field arithmetic, and elliptic curve arithmetic. Section 3 outlines the quantum algorithm, and presents our improvements: the $GF(2^m)$ multiplication circuit, and projective coordinate representation. The paper concludes with some observations and suggestions for further research.

2 Preliminaries

We will be working in the quantum circuit model, where data is stored in qubits and unitary operations are applied to various qubits at discrete time steps as quantum gates. We assume that any set of non-intersecting gates may be applied within one time step. The total number of time steps required to execute an algorithm as a circuit is the *depth.* Further details on quantum computation in the circuit model can be found in [14].

We will make use of the CNOT, Toffoli and SWAP gates. The CNOT gate is defined as the unitary operator which performs the transformation $|a\rangle |b\rangle \mapsto |a\rangle |a \oplus b\rangle$. The Toffoli gate [20] can be described as a controlled CNOT gate, and performs the transformation over the computational basis given by the formula $|a\rangle |b\rangle |c\rangle \mapsto |a\rangle |b\rangle |c \oplus ab\rangle$. Finally, SWAP interchanges the values of the qubits, i.e., performs operation $|a\rangle |b\rangle \mapsto |b\rangle |a\rangle$.

2.1 Binary Field Arithmetic

The finite field $GF(2^m)$ consists of a set of 2^m elements with addition and multiplication operations, and additive and multiplicative identities 0 and 1, respectively. $GF(2^m)$ forms a commutative ring over these two operations where each non-zero element has a multiplicative inverse. The finite field $GF(2^m)$ is unique up to isomorphism.

We can represent the elements of $GF(2^m)$ where $m \geq 2$ with the help of an irreducible *primitive polynomial* of the form $P(x) = \sum_{i=0}^{m-1} c_i x^i + x^m$, where $c_i \in GF(2)$ [16]. The finite field $GF(2^m)$ is isomorphic to the set of polynomials over $GF(2)$ modulo $P(x)$. In other words, elements of $GF(2^m)$ can be represented as polynomials over $GF(2)$ of degree at most $m-1$, where the product of two elements is the product of their polynomial representations, reduced modulo $P(x)$ [16, 18]. As the sum of two polynomials is simply the bitwise XOR of the coefficients, it is convenient to express these polynomials as bit vectors of length m. Additional properties of finite fields can be found in [16].

Mastrovito has proposed an algorithm along with a classical circuit implementation for polynomial basis (PB) multiplication [10, 11], popularly known as the Mastrovito multiplier. Based on Mastrovito's algorithm, [18] presents a formulation of PB multiplication and a generalized parallel-bit hardware architecture for special types of primitive polynomials, namely trinomials, equally spaced polynomials (ESPs), and two classes of pentanomials.

Consider the inputs $\vec{a}$ and $\vec{b}$, with $\vec{a} = [a_0, a_1, a_2, \ldots, a_{m-1}]^T$ and $\vec{b} = [b_0, b_1, b_2, \ldots, b_{m-1}]^T$, where the coordinates a_i and b_i, $0 \leq i < m$, are the coefficients of two polynomials $A(x)$ and $B(x)$ representing representing two elements of $GF(2^m)$ with respect to a primitive polynomial $P(x)$. We use the following three matrices:

1. an $m \times (m-1)$ reduction matrix M,
2. an $m \times m$ lower triangular matrix L, and

3. an $(m-1) \times m$ upper triangular matrix U.

Vectors $\vec{d}$ and $\vec{e}$ are defined as:

$$\begin{aligned} \vec{d} &= L\vec{b} & (1) \\ \vec{e} &= U\vec{b}, & (2) \end{aligned}$$

where L and U are defined as

$$L = \begin{bmatrix} a_0 & 0 & \dots & 0 & 0 \\ a_1 & a_0 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{m-2} & a_{m-3} & \dots & a_0 & 0 \\ a_{m-1} & a_{m-2} & \dots & a_1 & a_0 \end{bmatrix}, \quad U = \begin{bmatrix} 0 & a_{m-1} & a_{m-2} & \dots & 0 & a_1 \\ 0 & 0 & a_{m-1} & \dots & 0 & a_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & a_{m-1} & a_{m-2} \\ 0 & 0 & 0 & \dots & 0 & a_{m-1} \end{bmatrix}.$$

Note that $\vec{d}$ and $\vec{e}$ correspond to polynomials $D(x)$ and $E(x)$ such that $A(x)B(x) = D(x) + x^m E(x)$. Using $P(x)$, we may construct a matrix M which converts the coefficients of any polynomial $x^m E(x)$ to the coefficients of an equivalent polynomial modulo $P(x)$ with degree less than m. Thus, the vector

$$\vec{c} = \vec{d} + Q\vec{e} \qquad (3)$$

gives the coefficients of the polynomial representing the product of $\vec{a}$ and $\vec{b}$. The construction of the matrix M, which is dependent on the primitive polynomial $P(x)$, is given in [18].

2.2 Elliptic Curve Groups

In the most general case, we define an *elliptic curve* over a field F as the set of points $(x, y) \in F \times F$ which satisfy the equation

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_5.$$

By extending this curve to the projective plane, we may include the point at infinity $\mathcal{O}$ as an additional solution. By defining a suitable addition operation, we may interpret the points of an elliptic curve as an Abelian group, with $\mathcal{O}$ as the identity element.

In the specific case of the finite field $GF(2^m)$, it is possible to reduce the degrees of freedom in the coefficients defining the elliptic curve by the use of linear transformations on the variables x and y. In addition, it was shown in [12] that for a class of elliptic curves called *supersingular* curves, it is possible to reduce the discrete logarithm problem for the elliptic curve group to a discrete logarithm problem over a finite field in such a way that makes such curves unsuitable for cryptography. For $GF(2^m)$, these correspond to elliptic curves with parameter $a_1 = 0$. We will restrict our attention to non-supersingular curves over $GF(2^m)$, which are of the form $y^2 + xy = x^3 + ax^2 + b$, where $b \neq 0$.

The set of points over an elliptic curve also forms an Abelian group with $\mathcal{O}$ as the identity element. For a non-supersingular curve over $GF(2^m)$, the group operation is defined in the following manner. Given a point $P = (x_1, y_1)$ on the curve, we define $(-P)$ as $(x_1, x_1 + y_1)$. Given a second point $Q = (x_2, y_2)$, where $P \neq \pm Q$, we define the sum $P + Q$ as the point (x_3, y_3) where $x_3 = \lambda^2 + \lambda + x_1 + x_2 + a$ and $y_3 = (x_1 + x_3)\lambda + x_3 + y_1$, with $\lambda = \frac{y_1+y_2}{x_1+x_2}$. When $P = Q$, we define $2P$ as the point (x_3, y_3) where $x_3 = \lambda^2 + \lambda + a$ and $y_3 = x_1^2 + \lambda x_3 + x_3$,

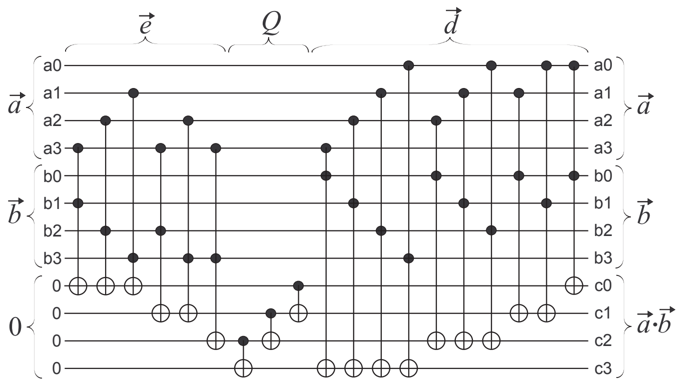

Fig. 1. Circuit for $GF(2^4)$ multiplier with $P(x) = x^4 + x + 1$.

with $\lambda = x_1 + \frac{y_1}{x_1}$. Also, any group operation involving $\mathcal{O}$ simply conforms to the properties of a group identity element. Finally, scalar multiplication by an integer can be easily defined in terms of repeated addition or subtraction.

The Elliptic Curve Discrete Logarithm problem (ECDLP) is defined as the problem of retrieving a constant scalar d given that $Q = dP$ for known points P and Q. With this definition, we may define cryptographic protocols such as Diffie-Hellman or digital signature, using the ECDLP by modifying analogous protocols using the discrete logarithm problem over finite fields.

3 Quantum Polynomial-Time Attack

With a reversible implementation for the basic elliptic curve group operations, it is possible to solve the ECDLP with a polynomial-depth quantum circuit. Given a base point P and some scalar multiple $Q = dP$ on an elliptic curve over $GF(2^m)$, Shor's algorithm for discrete logarithms [19] constructs the state

$$\frac{1}{2^m}\sum_{x=0}^{2^m-1}\sum_{y=0}^{2^m-1}|x\rangle\,|y\rangle\,|xP + yQ\rangle\,,$$

then performs a two-dimensional quantum Fourier transform over the first two registers. It was shown in [17] that the creation of the above state can be reduced to adding a classically known point to a superposition of points, by using a "double and add" method analogous to the square and multiply method of modular exponentiation. Points of the form 2^kP and 2^kQ can be classically precomputed, and then, starting with the additive identity, group addition operations can be performed, controlled by the appropriate bits from $|x\rangle$ or $|y\rangle$. Note that all of the intermediate sums must be preserved until the computation is completed before they can be uncomputed.

3.1 Linear Depth Circuit for $GF(2^m)$ Multiplication in the LNN Architecture

We now discuss how to implement multiplication over $GF(2^m)$ as a quantum circuit. Firstly, using equations (1–3), derive expressions for $\vec{d}$, $\vec{e}$ and $\vec{c}$. We next perform the following steps breaking the entire computation into three distinct stages/circuits:

1. Compute $\vec{e}$ in an ancillary register of m qubits.
2. Transform $\vec{e}$ into $M\vec{e}$, using a linear reversible implementation.
3. Compute and add $\vec{d}$ to the register occupied by $M\vec{e}$.

We illustrate the above steps with an example using $P(x) = x^4 + x + 1$. Expressions for $\vec{d}$ and $\vec{e}$ derived from equations (1–2) are shown below.

$$\vec{d} = \begin{bmatrix} a_0b_0 \\ a_1b_0 + a_0b_1 \\ a_2b_0 + a_1b_1 + a_0b_2 \\ a_3b_0 + a_2b_1 + a_1b_2 + a_0b_3 \end{bmatrix}, \quad \vec{e} = \begin{bmatrix} a_3b_1 + a_2b_2 + a_1b_3 \\ a_3b_2 + a_2b_3 \\ a_3b_3 \end{bmatrix}.$$

We also construct the matrix $M = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$.

From (3), we compute the multiplier output $\vec{c} = \vec{d} + M\vec{e} = \begin{bmatrix} d_0 + e_0 \\ d_1 + e_1 + e_0 \\ d_2 + e_1 + e_2 \\ d_3 + e_2 \end{bmatrix}$.

1. We first compute e_0, e_1, and e_2 in the ancilla, as shown in Figure 1 (gates 1–6).
2. We next implement the matrix transformation $M\vec{e}$ (gates 7–9).
3. Finally, we compute the coefficients d_i, $0 \le i < m$, and add them to the ancilla to compute $\vec{c}$ (gates 10–19).

At this point, we have a classical reversible circuit which implements the transformation $|a\rangle|b\rangle|0\rangle \mapsto |a\rangle|b\rangle|a \cdot b\rangle$. However, if we input a superposition of field elements, then the output register will be entangled with the input. If one of the inputs, such as $|b\rangle$ is classically known, then we may also obtain $|b^{-1}\rangle$ classically. Since we may construct a circuit which maps $|a \cdot b\rangle|b^{-1}\rangle|0\rangle \mapsto |a \cdot b\rangle|b^{-1}\rangle|a\rangle$, we may apply the inverse of this circuit to the output of the first circuit to obtain $|a\rangle|b\rangle|a \cdot b\rangle \mapsto |0\rangle|b\rangle|a \cdot b\rangle$ using an ancilla set to $|b^{-1}\rangle$. This gives us a quantum circuit which takes a quantum input $|a\rangle$ and classical input $|b\rangle$, and outputs $|a \cdot b\rangle|b\rangle$. When $|b\rangle$ is not a classical input, the output of the circuit may remain entangled with the input, and other techniques may be required to remove this entanglement. However, we emphasize that this is not required for a polynomial-time quantum algorithm for the ECDLP [17].

In some circumstances, we may derive exact expressions for the number of gates required in the GF multiplication circuit.

Lemma 1 *A binary field multiplier for primitive polynomial $P(x)$ can be designed using at most $2m^2 - 1$ gates. If $P(x)$ is a trinomial or the all-one polynomial, where each coefficient is 1, we require only $m^2 + m - 1$ gates.*

Proof. There are three phases to the computation: computing $\vec{e}$, computing $M\vec{e}$, and adding $\vec{d}$ to the result. For $\vec{e}$ and $\vec{d}$, each pair of coefficients which are multiplied and then added to another qubit requires one Toffoli gate. This requires

$$\sum_{i=0}^{m-1} i = \frac{m(m-1)}{2}, \text{ and } \sum_{i=0}^{m} i = \frac{m(m+1)}{2}$$

gates respectively, for a total of m^2 gates. Next, consider the implementation of the transformation M.

In general, $m^2 - 1$ CNOT gates suffice for any linear reversible computation defined by the matrix M in equation (3) [23]. This gives a general upper bound of $2m^2 - 1$ gates. In the specific case of the All-One-Polynomial, the operation M consists of adding e_1 to each of the other qubits, requiring $m - 1$ CNOT operations. This gives a total of $m^2 + m - 1$ operations.

For a trinomial, we have a primitive polynomial $P(x) = x^m + x^k + 1$ for some constant k such that $1 \leq k < m$. To upper bound the number of gates required to implement M, we may consider the inverse operation, in which we begin with a polynomial of degree at most $m - 1$, and we wish to find an equivalent polynomial where each term has degree between $m - 1$ and $2m - 2$. Increasing the minimum degree of a polynomial requires one CNOT operation, and this must be done $m - 1$ times. Again, this gives a total of $m^2 + m - 1$ operations. *QED*

3.1.1 Parallelization

We construct a parallelized version of this network by considering the three parts of the computation: computation of $\vec{e}$, multiplication by M and in-place computation of $\vec{d}$. For $\vec{e}$ and $\vec{d}$, note that given coefficients a_i and b_j where the value of $i - j$ is fixed, the target qubit of each separate term $a_i b_j$ is different. This means that they may be performed in parallel. In the case of $\vec{e}$, we evaluate $a_i b_j$ whenever $i + j \geq m$. This means that the values of $i - j$ may range from $-(m-2)$ to $m - 2$, giving a depth $2m - 3$ circuit for finding e. Similarly, for $\vec{d}$, we evaluate $a_i b_j$ whenever $i + j < m$. The values of $i - j$ range from $-(m-1)$ to $m - 1$, giving a depth $2m - 1$ circuit. Evaluation of $\vec{d}$ in the case of $GF(2^8)$ is illustrated in Figure 2.

In [9], it is shown that every linear computation, such as that of the product $M\vec{e}$, can be done in a linear number of stages, with a depth of at most $5m$. Thus, a total depth of $(2m - 3) + 5m + (2m - 1) = 9m + O(1)$ suffices to implement the multiplication circuit. As such, an implementation which replaces the Toffoli gate with 2-qubit gates ([14], page 182) can be done by a circuit with the depth upper bounded by the expression $25m + O(1)$.

3.1.2 Execution of the multiplication circuit in LNN

In this subsection we explain how to execute the GF multiplication circuit in linear depth $O(m)$ in the LNN architecture. Our circuit consists of three distinct stages: creation of $\vec{e}$, followed by a linear reversible transformation, and the in-place calculation of $\vec{d}$. As shown in [9], the middle part of this calculation can be executed as a depth $5m$ computation in the LNN architecture. We next show that first and third parts in our construction can also be modified to become a linear depth computation in the LNN. At this point, we note that both subcircuits share identical structure, and as such we only need to consider either one. We choose the circuit for computing $\vec{d}$. In the following, we will use its parallelized version described in the previous subsection and separate every two computational stages by a depth-

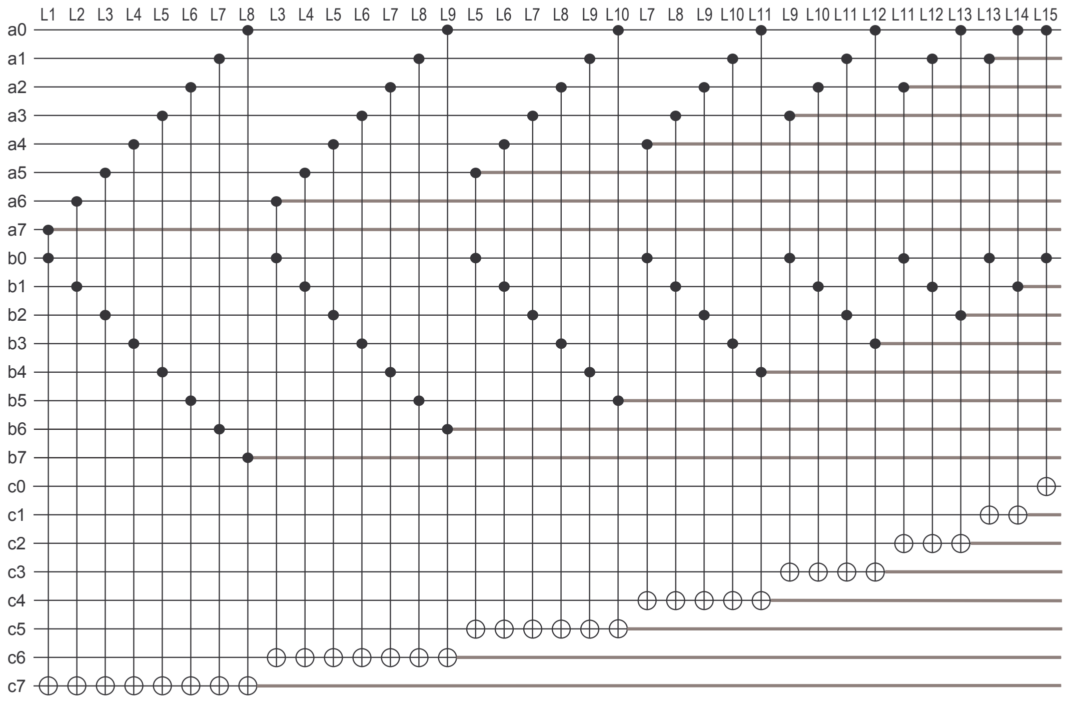
L1 L2 L3 L4 L5 L6 L7 L8 L3 L4 L5 L6 L7 L8 L9 L5 L6 L7 L8 L9 L10 L7 L8 L9 L10 L11 L9 L10 L11 L12 L11 L12 L13 L13 L14 L15
a0
a1
a2
a3
a4
a5
a6
a7
b0
b1
b2
b3
b4
b5
b6
b7
c0
c1
c2
c3
c4
c5
c6
c7

Fig. 2. Subcircuit computing $\vec{d}$ illustrated in the case of multiplication in $GF(2^8)$. $L*$ indicate which stage does the given gate gets executed at. The qubit line turns gray after a given qubit was used last during the computation.

2 qubit swapping stage such as to make it possible to execute the entire computation in the LNN in linear depth.

First, prepare the qubits in the LNN connectivity pattern $c_0 - c_1 - ... - c_{m-1} - a_{m-1} - b_0 - a_{m-2} - b_1 - ... - a_0 - b_{m-1}$. For that, at most linear depth qubit swapping stage is required, no matter what is the starting connectivity pattern. Next, execute a computational stage. For every Toffoli gate $TOF(c_i; a_j, b_k)$ applied and a qubit x on the left from c_i in the present LNN connectivity pattern use the depth-2 swapping stage SWAP(c_i, a_j) SWAP(x, a_j) SWAP(c_i, b_k) to prepare the qubits for the next computational stage (LNN connectivity pattern $x - c_i - a_j - b_k$ gets transformed to $a_j - x - b_k - c_i$). The workings of such adaptation to the LNN architecture are illustrated in Figures 2 and 3. Note that the number of swapping stages is no more than twice the number of the computational stages. Therefore, the entire computation can be executed in linear depth, not exceeding $34m + O(1)$ (counting 1- and 2-qubit operations), in the LNN architecture.

3.2 Projective Representation

When points on an elliptic curve are represented as *affine coordinates* (x, y), performing group operations on such points requires finding the multiplicative inverse of elements of $GF(2^m)$. This operation takes much longer to perform than the other field operations required, and it is desirable to minimize the number of division operations. For example, [7] gives a quantum

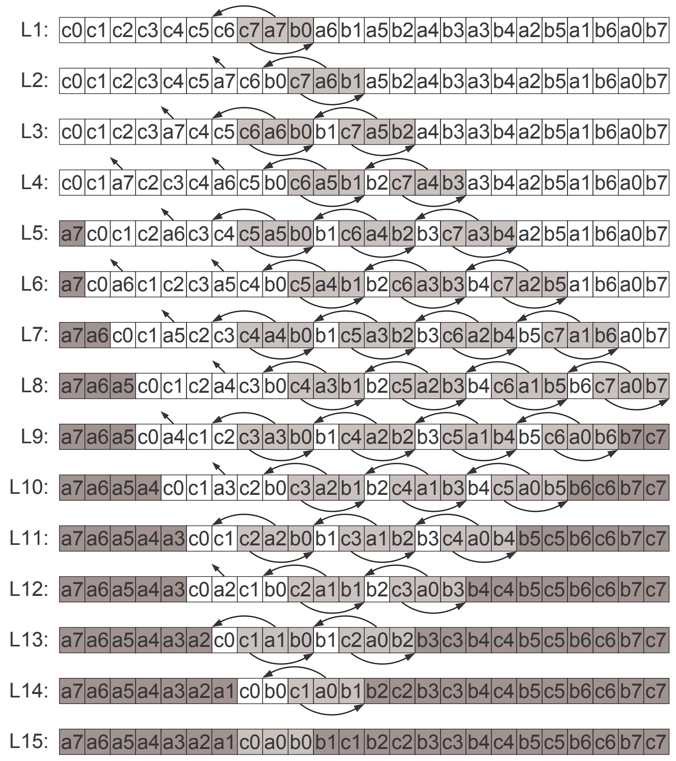
L1: c0 c1 c2 c3 c4 c5 c6 c7 a7 b0 a6 b1 a5 b2 a4 b3 a3 b4 a2 b5 a1 b6 a0 b7
L2: c0 c1 c2 c3 c4 c5 a7 c6 b0 c7 a6 b1 a5 b2 a4 b3 a3 b4 a2 b5 a1 b6 a0 b7
L3: c0 c1 c2 c3 a7 c4 c5 c6 a6 b0 b1 c7 a5 b2 a4 b3 a3 b4 a2 b5 a1 b6 a0 b7
L4: c0 c1 a7 c2 c3 c4 a6 c5 b0 c6 a5 b1 b2 c7 a4 b3 a3 b4 a2 b5 a1 b6 a0 b7
L5: a7 c0 c1 c2 a6 c3 c4 c5 a5 b0 b1 c6 a4 b2 b3 c7 a3 b4 a2 b5 a1 b6 a0 b7
L6: a7 c0 a6 c1 c2 c3 a5 c4 b0 c5 a4 b1 b2 c6 a3 b3 b4 c7 a2 b5 a1 b6 a0 b7
L7: a7 a6 c0 c1 a5 c2 c3 c4 a4 b0 b1 c5 a3 b2 b3 c6 a2 b4 b5 c7 a1 b6 a0 b7
L8: a7 a6 a5 c0 c1 c2 a4 c3 b0 c4 a3 b1 b2 c5 a2 b3 b4 c6 a1 b5 b6 c7 a0 b7
L9: a7 a6 a5 c0 a4 c1 c2 c3 a3 b0 b1 c4 a2 b2 b3 c5 a1 b4 b5 c6 a0 b6 b7 c7
L10: a7 a6 a5 a4 c0 c1 a3 c2 b0 c3 a2 b1 b2 c4 a1 b3 b4 c5 a0 b5 b6 c6 b7 c7
L11: a7 a6 a5 a4 a3 c0 c1 c2 a2 b0 b1 c3 a1 b2 b3 c4 a0 b4 b5 c5 b6 c6 b7 c7
L12: a7 a6 a5 a4 a3 c0 a2 c1 b0 c2 a1 b1 b2 c3 a0 b3 b4 c4 b5 c5 b6 c6 b7 c7
L13: a7 a6 a5 a4 a3 a2 c0 c1 a1 b0 b1 c2 a0 b2 b3 c3 b4 c4 b5 c5 b6 c6 b7 c7
L14: a7 a6 a5 a4 a3 a2 a1 c0 b0 c1 a0 b1 b2 c2 b3 c3 b4 c4 b5 c5 b6 c6 b7 c7
L15: a7 a6 a5 a4 a3 a2 a1 c0 a0 b0 b1 c1 b2 c2 b3 c3 b4 c4 b5 c5 b6 c6 b7 c7

Fig. 3. Qubit permutation stages for adaptation of the GF multiplication circuit the the LNN architecture, illustrated in the case $GF(2^8)$ and the computation of $\vec{d}$. The arrows indicate which position do the individual qubits get swapped to. In particular, arrow $\nwarrow$ indicates that the given qubit will not be used in the remainder of the computation, and it gets moved to the rightmost position. Triples of qubits highlighted light gray experience application of the Toffoli gates. Dark grey color is used to highlight qubits that are not used in the remainder of computation.

circuit of depth $O(m^2)$ which uses the extended Euclidean algorithm.

By using *projective* coordinate representation, we can perform group operations without division. Instead of using two elements of $GF(2^m)$ to represent a point, we use three elements, (X, Y, Z) to represent the point $(\frac{X}{Z}, \frac{Y}{Z})$ in affine coordinates. Dividing X and Y by a certain quantity is now equivalent to multiplying the third coordinate (Z) by this quantity. Extensions to this concept have also been explored, where different information about an elliptic curve point is stored in several coordinates. Another advantage to projective coordinates is that the point at infinity $\mathcal{O}$ can simply be represented by setting Z to zero. However, in order to retrieve the elliptic curve point in the affine representation, we still need to perform one multiplicative inversion at the end.

To represent the point (X, Y), we simply begin with the representation

$$|P(X,Y)\rangle = |X\rangle\,|Y\rangle\,|1\rangle\,.$$

As we perform elliptic curve group operations, the third coordinate will not remain constant.

Exact formulas for point addition in projective coordinates can be easily derived by taking the formulas for the affine coordinates under a common denominator and multiplying the Z coordinate by this denominator. These are detailed in [5]. Since the ECDLP can be solved by implementing elliptic curve point addition where one point is “classically known” [17], we may implement these formulas using the multiplication algorithm presented in Section 3.1 and by being careful to uncompute any temporary registers used. Since the number of multiplication operations used in these formulas is fixed, we may implement elliptic curve point addition with a known classical point with a linear depth circuit.

Finally, to construct the state required for solving the ECDLP, we use the standard “double and add” technique, which requires implementing the point addition circuit for each value 2^iP and 2^iQ, where $0 \leq i < m$. Note that these points are classically known, so that at each step, we are performing point addition with one classically known point. When the final state

$$\frac{1}{2^m}\sum_{x=0}^{2^m-1}\sum_{y=0}^{2^m-1}|x\rangle\,|y\rangle\,|xP+yQ\rangle$$

is constructed, each $|xP+yQ\rangle$ will consist of three coordinates $|X\rangle\,|Y\rangle\,|Z\rangle$. Since the presence of the third coordinate Z will interfere with the discrete logarithm algorithm, we must revert to an affine coordinate representation. An algorithm to compute the multiplicative inverse of an element of $GF(2^m)$ using an $O(m^2)$-depth circuit is given in [7]. Using $|Z^{-1}\rangle$, we may compute $|XZ^{-1}\rangle\,|YZ^{-1}\rangle$, as required, before uncomputing $|Z^{-1}\rangle$. Since $|X\rangle\,|Y\rangle\,|Z\rangle$ must now be uncomputed, this step must occur before any of the temporary registers used in computing them are themselves uncomputed. The result is the desired state

$$\frac{1}{2^m}\sum_{x=0}^{2^m-1}\sum_{y=0}^{2^m-1}|x\rangle\,|y\rangle\,|xP+yQ\rangle$$

in affine coordinates. As a final detail, we also need to address the *point at infinity*, $\mathcal{O}$, which is the identity element of the elliptic curve group. In projective coordinates, $\mathcal{O}$ is represented by any (X,Y,Z) where $Z=0$. In this case, we will not be able to perform multiplicative inversion on Z. However, since the ensuing quantum Fourier transform only requires that each point have a consistent representation, we may simply select the coordinates of a point which is known not to lie on the elliptic curve to represent $\mathcal{O}$. The final registers can simply be set to these coordinates in the case that $Z=0$.

This represents an improvement on the algorithm of [7], as multiplicative inversion is used only once, at the end of this algorithm, rather than at each elliptic curve point operation. In total, we perform $O(m)$ instances of the linear depth multiplication circuit, one instance of the $O(m^2)$-depth multiplicative inversion circuit, and finally, a quantum Fourier transform. This gives a final depth complexity of $O(m^2)$ for the circuit which solves the ECDLP over $GF(2^m)$ in the LNN architecture. This improves the previously known upper bound of $O(m^3)$ [17].

4 Conclusion

We considered the optimization of the quantum attack on the elliptic curve discrete logarithm problem, on which elliptic curve cryptography is based. Our constructions include a linear depth circuit for binary field multiplication and efficient data representation using projective coordinates. Our main result is the depth $O(m^2)$ circuit executable in the LNN architecture for computing the discrete logarithm over elliptic curves over $GF(2^m)$. Further research may be devoted toward a better optimization, further study of architectural implications, and the fault tolerance issues.

Interestingly, our circuit is slightly (by a logarithmic factor) more efficient than the best known circuit for integer factoring optimized for the LNN architecture, allowing linear ancilla and assuming gates with diminishingly small parameters cannot be used [8]. (We believe this is related to necessity of performing carry over during the integer addition, while it is not required for the addition over $GF(2^m)$.) However, our circuit reduces an exponential classical search to a polynomial time quantum, whereas integer factoring can be done classically with a subexponential time algorithm. Considering relative efficiency of ECC as compared to RSA, we suggest referring to the ability to solve ECDLP as a stronger practical argument for quantum computing.

References

1. G. B. Agnew, R. C. Mullin, and S. A. Vanstone. An implementation of elliptic curve cryptosystems over $GF(2^{155})$. *IEEE Journal on Selected Areas in Communications*, 11(5):804–813, 1993.
2. S. Beauregard, G. Brassard, and J. M. Fernandez. Quantum arithmetic on Galois fields. `arXiv:quant-ph/0301163`, 2003.
3. Certicom. Certicom announces elliptic curve cryptography challenge winner. Certicom press release, 2004.
4. R. Cleve and J. Watrous. Fast parallel circuits for the quantum Fourier transform. *IEEE Symposium on Foundations of Computer Science*, 41:526–536, 2000.
5. D. Hankerson, J. López Hernandez, and A. Menezes. Software implementation of elliptic curve cryptography over binary fields. *Proceedings of the Second International Workshop on Cryptographic Hardware and Embedded Systems*, 2:1–24, 2000.
6. R. Jozsa. Quantum algorithms and the Fourier transform. *Proc. R. Soc. Lond. A*, 454:323–337, 1998.
7. P. Kaye. Optimized quantum implementation of elliptic curve arithmetic over binary fields. *Quantum Information and Computation*, 5(6):474-491, 2005.
8. S. A. Kutin. *Shor's algorithm on a nearest-neighbor machine.* Asian conference on Quantum Information Science, pp. 12–13, September 2007, `arXiv:quant-ph/0609001`.
9. S. A. Kutin, D. P. Moulton, and L. M. Smithline. Computation at a distance, January 2007, `arXiv:quant-ph/0701194`.
10. E. D. Mastrovito. VLSI designs for multiplication over finite fields $GF(2^m)$. *Proceedings of the Sixth Symposium on Applied Algebra, Algebraic Algorithms, and Error Correcting Codes*, 6:297–309, 1988.
11. E. D. Mastrovito. *VLSI Architectures for Computation in Galois Fields.* PhD Thesis, Linkoping University, Linkoping, Sweden, 1991.
12. A. J. Menezes, T. Okamoto, and S. A. Vanstone. Reducing elliptic curve logarithms to logarithms in a finite field. *IEEE Transactions on Information Theory*, 39:1639–1646, 1993.
13. R. Van Meter and K. M. Itoh. Fast quantum modular exponentiation. *Physical Review A*, 71:052320, 2005.
14. M. Nielsen and I. Chuang. *Quantum Computation and Quantum Information.* Cambridge University Press, 2000.
15. NSA Suite B Factsheet. http://www.nsa.gov/ia/industry/crypto_suite_b.cfm.
16. D. K. Pradhan. A theory of Galois switching functions. *IEEE Transactions on Computers*, 27:239–248, 1978.
17. J. Proos and C. Zalka. Shor's discrete logarithm quantum algorithm for elliptic curves. *Quantum*

Information and Computation, 3:317–344, 2003.

18. A. Reyhani-Masoleh and M. A. Hasan. Low complexity bit parallel architectures for polynomial basis multiplication over $GF(2^m)$. *IEEE Transactions on Computers*, 53:945–959, 2004.
19. P. W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal of Computing*, 26:1484–1509, 1997.
20. T. Toffoli. Reversible computing. Tech memo MIT/LCS/TM-151, MIT Lab for Computer Science, 1980.
21. S. A. Vanstone. Elliptic Curve Cryptography: The next generation of wireless security. *Markt and Technik*, 17:20–23, April 23, 2004.
22. J. Von Zur Gathen and J. Gerhard. *Modern Computer Algebra.* Cambridge University Press, 1999.
23. To see that this is true, write a reversible linear function in the $(m \times m)$ matrix form, and apply Gauss-Jordan elimination algorithm to synthesize a circuit using CNOT gates. This requires application of at most m^2 gates, one per each entry of the matrix; and it may be observed that at least one gate needs not be applied. The resulting gate count is thus at most $m^2 - 1$.